Microcontroller Based Data Acquisition and Control of a Solar Thermal Energy System

University of KwaZulu-Natal

By

Gonçalves Justino Doho

Microcontroller Based Data Acquisition and Control

Of a Solar Thermal Energy System

Gonçalves Justino Doho

December 2009

Submitted to the School of Physics, Faculty of Science and Agriculture,

University of KwaZulu – Natal, Westville,

in partial fulfilment of the requirements for the degree of

Master of Science

in Renewable Energy Systems

Supervisor: Dr. Kessie Govender (UKZN)

As the candidate's supervisor I have/have not approved this thesis/dissertation for submission.

Signed: _____ Name: _____date: _____

Declaration - Plagiarisms

I, Gonçalves Justino Doho, declare that

1. The work presented in this thesis, except where otherwise indicated, is my original work.
2. This thesis has not been submitted to any degree or examination at any other university.
3. This thesis does not contain other persons' data, pictures, graphs or other information, unless explicitly acknowledged as being sourced from other persons.
4. This thesis does not contain other persons' writings, unless explicitly acknowledged as being sourced from other persons. Where other written sources have been quoted, then:
    a. Their words have been re-written but the general information attributed to them has been referenced.
    b. Where their exact words have been used, their writing has been placed inside quotation marks, and referenced.
5. This thesis does not contain text, graphics or tables copied and pasted from the Internet, unless explicitly acknowledged, and the source being indicated and included in the References section.

Signed:

_____

Dedication

To my family, Eros, Othis, Eca and Suzy, for your tireless support and endless love. You are my inspiration!

Acknowledgements

This work, developed mainly at the University of KwaZulu-Natal (UKZN), could not have reached its end without the valuable contribution of many institutions and individuals:

I am especially grateful to Dr. Kessie Govender, my academic supervisor at UKZN, for his support, guidance, patience, time, and valuable criticisms throughout the course of this work and for his input in the thesis report in particular.

I express my acknowledgement and gratefulness to the University of KwaZulu-Natal, for the opportunity of doing this work and also for the prompt and friendly assistance I received from many people of the university staff which left me with a lifelong impression. This acknowledgement and gratefulness goes especially to the School of Physics and its staff, for their support whenever I needed.

I also express my acknowledgement and gratefulness to the University Eduardo Mondlane (UEM) and its staff for the opportunity of doing this work. I am particularly grateful to those members of the Faculty of Engineering staff who did in no moment deny their continuous support, confidence and/or constructive attitude whenever I needed collaboration and assistance.

I especially acknowledge and thank Prof. Carlos Lucas, for his tireless support and guidance, as TecPro programme Coordinator at the Faculty of Engineering, UEM.

I especially acknowledge and thank Prof. Boaventura Cuamba (UEM), for his continued support and guidance which led to the realization of this work at UKZN.

I express my acknowledgement and gratefulness to the Swedish SIDA/SAREC, for their continued financial support. I specially thank Prof. Bjorn Karlsson (EBD-LTH Lund University, Sweden), for his support and positive criticisms.

I express my acknowledgement and gratefulness to the Norwegian NUFU programme, for their financial support to the renewable energy projects at both UKZN and UEM. I am especially grateful to Prof. Jorgen Løvseth (NUFU/NTNU) for his support and valuable criticisms.

I express my gratefulness to my colleagues and friends at the UKZN, namely Zhandire, Robert, Zucule, Macome, Celia, Mendes, Elisa, Mukharo and Stefania, for their assistance and criticisms and friendship, which helped me in the realization of this work in different moments.

I express my especial gratefulness to all not namely mentioned, family, friends, colleagues, for their assistance, friendly attitude or moral support, without which the realization of this work could be difficult or impossible. Thank you all!

My very special thanks to my mother. She was and is all for me. I could not have grown and reached this point without her love, support and inspiration. I love you!

I lastly, but endlessly, thank God!, for His protection and all what He has giving!

Microcontroller Based Data Acquisition and Control of a Solar Thermal Energy System

Abstract

A solar thermal energy system is being rebuilt at University of KwaZulu-Natal School of Physics. A similar system is also being built in the University Eduardo Mondlane – Maputo Mozambique, in a team development work. The system is composed mainly of the following subsystems:

(i) An Energy capture subsystem: paraboloidal dish concentrator with a heat receiver, mounted on a dual axis polar mount sun tracking assembly;
(ii) An Energy storage subsystem: rock-bed thermal energy storage (TES) system;
(iii) An Energy utilization subsystem: any user heat utilization (like a cooking or water boiling appliance); and
(iv) A monitoring and control subsystem.

The subsystem (iv) for performing a controlled charging of the Thermal Energy Storage from a hot plate simulated solar heat, was formerly developed and it was based on 2 conventional data loggers (HP/Agilent) and programs running on 2 PCs. The present work is aimed at performing the same plus additional monitoring and control tasks, based on a low cost microcontroller design. The monitoring and control subsystem based on the Atmel ATmega 32 MCU has been designed and built, capable of performing data acquisition, data logging and control of relevant system variables such as, hour and declination angles of the tracking concentrator; to cite some of the main variables.

Besides a huge work of designing, building, programming and testing the microcontroller system itself, a special focus was given to the monitoring and control of the solar heat concentrator, to perform a dual axis sun tracking, so as to get as much as possible of the available solar radiation. Measurements of various system parameters such as, the sun tracking actual hour and declination angles, the inlet and outlet temperatures of both the heat receiver and the rock bed heat storage, etc., for the system under consideration have been carried out.

Preface

This thesis report is presented in 9 chapters and 5 appendices. A brief description of what is addressed in each chapter and appendix follows below.

Chapter I presents an introduction to this work. It discusses the context and motivations of the work and thereby the objectives. It ends up defining the building of a "Microcontroller Based Data Acquisition and Control of a Solar Thermal Energy System" as the specific objective of this work.

Chapter II addresses a theoretical background for the present work. Solar radiation, earth-sun geometry, sun tracking and thermal energy collection, as well as basic and applied control systems concepts, are discussed.

In Chapter III the pre-existing solar Thermal energy system at UKZN is studied and described, where we build up a general understanding of system: The main components that the system is composed of, as well as its basic working principles including safety considerations.

Chapter IV discusses a conceptual model of the Data Acquisition and Control subsystem for the Solar Thermal Energy System studied in previous chapter. This started by defining the list of required inputs/outputs and thereby establishing the system layout. At the final stage, the designing of specific controllers for the sun tracker and for the pumps, along with their software Implementation, are discussed.

Chapter V addresses the electronic design of the data acquisition system according to the layout established in the previous chapter. A detailed treatment of the inputs and outputs, led to the MCU system architecture and thereby the electronic design of the pc boards, with the help of CADD/CAE/CAM tools. As a result and follow-up, pc boards were manufactured. Then the prototype system was assembled.

Chapter VI discusses the design and implementation of the real time monitoring and control program, which is the operating system for the data acquisition and control system. The generic architecture of the program and the specific software components that make up the program, are designed. On the other hand and closing the chapter, the design and implementation of the Windows PC side data logging program is addressed.

Chapter VII presents the schedule and description of the experiments and at the same time discusses the objectives of each experiment. According to the discussion, the main objective of the experiments was to test the functionality of the newly built system in data acquisition, data logging and control and, on the other hand, evaluate how it can positively contribute to the heat collection process.

Chapter VII presents and discusses the experimental results, which led to the conclusion that the newly system is capable of performing the monitoring, data

logging and control of important process variables and thereby positively influence the heat collection process.

The conclusions and recommendations are presented in the Chapter 9, where the roadmap, the difficulties, the achievements and failures are discussed and, finally, recommendations for the follow-up work are presented.

Appendix A presents the source code listing of the real time operating program (ST-RTOP), the operating system embedded into the microcontroller. This program is addressed in chapter VI.

Appendix B presents the source code listing of the PC side data logging Windows program. This program is in charge of receiving and saving the data collected by the microcontroller and sent via RS232 serial interface. This program is addressed in the last section of chapter VI.

Appendix C contains sample tables of the data collected and logged during the experiments. These experiments are in turn described and discussed in chapters VII and VIII.

The appendix D contains datasheets of the main or uncommon electronic components or devices. Many of the presented datasheets are only sample pages (the most relevant ones) since the presentation of entire datasheets could be impractical because some are hundreds of pages long.

The appendix E contains photographs of the most relevant ST-KZN system components. The first illustrations in the appendix are photographs of sample boards, taken during different moments of the data acquisition system development. At the end of the appendix, photographs of the thermal energy system components, are shown.

Contents

List of Figures

List of symbols

G_{sc} – Solar constant
G_{on} – Solar constant that accounts for variation of extraterrestrial radiation
I_b – Beam (direct) radiation
I_d – Diffuse radiation
SE - Sun-Earth mean distance
SD - Sun diameter
ε - Sun apparent diameter
ED - Earth diameter
δ_0 - Earth polar axis declination
δ – Time (day of the year) dependent declination angle derived from δ_0
ω - Hour angle
L_{Loc} – Longitude of the observer at certain geographic location
L_{st} – Standard meridian
L_{time} - Local or standard time (the local clock time)
d - Day number
B – Fractional year
E- Equation of time
Θ – Incidence angle
Θ_z – Zenith angle
γ - Surface azimuth angle
γ_s - Solar azimuth angle
α_s - Solar altitude or elevation angle
β - Slope or surface tilt angle
$\boldsymbol{\phi}$ - Latitude angle: the latitude of the observer
λ - Longitude angle (λ = L_{loc}). The longitude of the observer.
Kp – proportional gain of a PID type controller
Ki – integral gain of a PID type controller
Kd – derivative gain of a PID type controller
R_e - Reynolds number
$\dot{Q}_c$ - Energy flux at the sun dish receiver
$\dot{Q}_{Rout}$ - Receiver's output effective energy flux.
$\dot{Q}_{Rloss}$ – Receiver's losses flux
η_C – Collector's efficiency
A_{eff} Effective surface of the collector aperture
$\dot{Q}_R$ – The energy transferred to the rock bed storage
$\dot{m}_{ch}$ - The heat transfer oil's charging circuit mass flow rate
$C_f(T)$ - The heat transfer oil's temperature dependant specific heat capacity
T_{Rin} - The heat transfer oil's inlet temperature to the receiver.
T_{Rout} - The heat transfer oil's outlet temperature from the receiver
$\dot{Q}_{TESin}$ - The rock bed storage inlet energy flux
$\dot{Q}_{RSloss}$ - Energy losses flux between the receiver's outlet and the storage inlet

$Q_{TESloss}$ - The rock bed storage energy losses
Q_{TESeff} - Rock bed storage effective energy stored in a specific period of time
η_{TES} - The rock bed storage efficiency
Q_{TESmax} -The theoretical maximum energy that can be absorbed by the heat storage
T_{TESin} – The rock bed heat storage's inlet temperature
T_{TES0} – The rock bed storage's initial temperature (just before the beginning of the heat transfer process)
c_p - The specific heat capacity of rock bed material
m_p - The total effective mass of the rock bed material
ρ_p - The density of the rock bed material
V_{TES} - The total rock bed storage volume
ε_p - The storage pebbles void fraction
$\dot{Q}_f$ - The Total energy of the heat transfer oil that traverses the rock bed storage
$c_f(T)$ – The Temperature dependant heat transfer oil's heat capacity
$\dot{V}_{ch}$ – The charging circuit volumetric flow rate
$\rho_f(T)$ - The density of the heat transfer oil as function of temperature
ω_{ch} - The rotating speed of the charging pump in revolution per unity time
HourSeekError - Seeking accuracy for the hour axis
DeclSeekError - Seeking accuracy for the declination axis
HourStayError - Step Tracking step size for the hour axis
DeclStayError- Step Tracking step size for the declination axis
HourError - The error: StPtHourAngle - ActualHourAng
DeclError - The error: StPtDeclAngle - ActualDeclAng
Fwd - Variable to specify tracker forward motion (along with Rev)
Rev- Variable to specify tracker backwards motion (along with Fwd)
MSel- Variable to specify which is the tracker axis to rotate
ActualDeclAng - The actual declination angle
ActualHourAng - The actual hour angle
StPtDeclAngle - The set point declination angle
StPtHourAngle - The set point hour angle

List of acronyms and abbreviations

3D - tri-dimensional
A/D (converter) - Analogue to Digital (converter)
AC – Alternating Current
ADC – Analogue to Digital Converter
ARM – Advanced RISC Machine (a MCU's architecture)
ATmega – A brand name of an AVR subfamily of MCUs from Atmel Corporation
AVC – Angle to voltage converter
AVR - A name of a family of microcontrollers from the Atmel Corporation
CAD – Computer aided design
CADD - Computer aided drafting and design
CAE - Computer aided engineering
Calflo AF – the brand name of a heat transfer fluid
CAM - Computer aided manufacturing
CCD – Charge Coupled Device (a photo detector)
CNC – Computer numerical control
CPC – Compound Parabolic concentrator
CPU – Central Processing Unit
CSV – Comma Separated Values
D/A (converter) - Digital to Analogue (converter)
DAC - Digital to Analogue Converter
DC – Direct Current
DIP – Dual Inline Package
DPDT – Dual pole dual through (switch).
DSP – Digital Signal Processor (MCU's architecture)
EEPROM – Electrically erasable programmable read-only memory
FFSM – Fuzzy Finite State Machine
FFSM-PID – Combined FFSM and PID
FLC – Fuzzy logic controller
FLR - Fresnel linear reflector
FPGA – Field programmable gate array
FSM – Finite State Machine
FSMC – Finite State Machine Controller
FSM-On/Off – Combined FSM and On/Off
FSM-PID– Combined FSM and PID
GDT – Gas discharge tube (gas arrestor)
GPS – Global positioning system
GSM – Global System for Mobile communications
HTF – heat transfer fluid
I/O – input/output
ICSP – In-Circuit Serial Programming
JTAG – Joint Test Action Group (a set of electronic design and testing standards)
LCD – Liquid Crystal Display

LDR – Light dependent resistor
M2M – Machine-to-machine (wireless interface)
MCU – Microcontroller unit
MIMD – multiple instruction, multiple data (stream)
MIMO - Multiple input, multiple output (system)
MOV – Metal oxide varistor
Mux – Abbreviated name for multiplexer
PC – Personal Computer
PCB – Printed circuit board
PD – Proportional and Derivative (controller)
PI – Proportional and Integral (controller)
PIC - A brand name of a family of MCUs from Microchip Corporation
PID – Proportional, Integral and Derivative (controller)
PLC – Programmable logic controller
PSU - Power supply unit
PTC - Parabolic trough concentrator
PV – Photovoltaic
PWM – Pulse Width Modulation
RAM – Random access memory (a read/write memory).
RISC – Reduced Instruction Set Computer
RMS – root mean square
RS232 – The name of a serial data transfer protocol
RTC - real time clock
RTOS – Real time operating system
SD (memory card) – Secure Digital (memory card)
SISD – Single instruction, single data (stream)
SISO – Single input, single output (system)
SPI – Serial peripheral interface
TCO – Thermal cut-out (fuse).
TD (conversion) – Thermocouple-to-Digital (conversion)
TDC – Thermocouple to Digital Converter
TES – Thermal Energy Storage
TVC – Temperature to voltage converter.
Xmega – A brand name of a AVR subfamily of MCUs from the Atmel Corporation

Acronyms or abbreviations specific to this work

These abbreviations were created and used solely for the convenience of simplifying naming, avoiding the frequent repetition of long sentences such as "The real time operating program", etc.

LB1 – Local bus 1

LB2 – Local bus 2

RTOP – Real time operating program

ST - Solar Tech

Solar Tech (controller) – Abbreviated name for the Data Acquisition and Control System developed in this work, deriving from "Solar Energy Technology Controller".

ST-KZN – Abbreviated name for the entire UKZN's Solar Thermal Energy System, deriving from "Solar Thermal – KZN"

ST-RTOP – Solar Tech's real time operating program

Glossary of some technical or non-common terms used

Fuzzy Logic – a logic in which a variable is not treated as fixed (crisp) value and rather as a zone of values (fuzzy value), along with the manner in which these values are evaluated in relation to each other (fuzzy inferencing).

Crisp (value) – is an exact (fixed) value as treated by a Boolean or arithmetic evaluation, as opposed to a fuzzy value. "Crisp" is a frequent term of the fuzzy logic arena.

Fuzzy inferencing – Is an evaluation of propositions where the value of a variable and/or expression (as well as the result), is not treated as an exact (crisp) value.

Timer tic – Expression commonly used to name a timer interrupt in a microcomputer system, used for timing purposes. The term "tic" comes probably from the analogy with the clock's timing "tic-toc".

Parsing – A common word in computing, used to name the action of dividing a string into individual elements. This applies for instance to a text data file where each line of text is a record containing various fields. Then, parsing in this case, is the action of recovering one or more individual fields from a data line in the text file.

Sustainable – sustainable use of a resource is an efficient use of the resource to meet the required needs in a manner that does not compromise future generations. According to the World Commission on Environment and Development (http://www.un-documents.net/a42r187.htm), sustainability is "meeting the needs of the present without compromising the ability of future generations to meet their own needs". See also http://www.gdrc.org/sustdev/definitions.html for more details.

Energy-centric – Is used frequently although it is not explicitly defined. Being energy-centric, is the ability of a power management system to give privilege to energy saving and energy efficiency. This means spending as less as possible energy, minimizing losses, while constantly meeting all the strict consumer's energy needs. This eventually results in a longer lasting energy supply, among other advantages. Meeting energy needs depends also on the energy input, which is an external factor to the energy management system. See Gruian (Gruian, 2002) and/or Al-Karaki (Al-Karaki, et al., 2007), for samples and discussions of energy-centric systems.

The Earth's sun belt – it is the equatorial region, characterized by having the highest solar radiation availability and intensity, ranging from about 1900 to 3000 kWh/m^2 per year (Geyer, et al., 2008). Various sources agree in that the sun belt is the region from the equator to about ±35 or ±40 degrees of latitude: According to (Geyer, et al., 2008), the sun belt is 35 degrees latitude either side of the equator. This delimitation is shared by other sources like (Ungeheuer, 2005), while Cuamba, et al. (2006) as also Solar Millennium (Sollar_Millenium_AG, 2009), consider that the sun belt is located roughly between the 40th parallels north and south.

Chapter I - Introduction

1.1 Background

Some global facts and needs

The fossil fuels, which are non-renewable energy resources (oil, gas and coal) are becoming scarce, towards a predictable exhaustion. According to Shafiee and Topal (Shafiee, et al., 2009) the reserves of oil, coal and gas will be completely depleted in approximately 35, 107 and 37 years, respectively. The emissions or by-products from the use of fossil fuels are environmentally harmful causing severe and irreversible damage to the climate, like global warming[1]. While fossil fuels are becoming scarce, energy demand is increasing.
Thus, the search for new types of energy (especially renewable, clean, and environmentally friendly) sources, has become today a global concern, in view of insuring energy sustainability and climate stability to our big home: the Earth. The generalization of solar energy use (which is clean, environmentally friendly) not only will aid development, but also reduce the use of fossil fuels, reduce CO_2 emissions, fight desertification, and help globally control climate change.

Mozambique (and generally Southern Africa) energy context

Mozambique is a huge and long country, with a largely dispersed population characterized by a low literacy level, very low income, having a lack of water and energy and living below the absolute line of poverty. Mozambique was, until the last few years, the poorest country of the world.

Mozambique has 801,590km^2 of total area (UN_Statistics_Division, 2009); a population of about 22.4 million (2008), of which 56% is illiterate, 42% have access to improved water sources, the gross national income per capita is about $380 (which equates to about $1.04 per day) and population living below line of poverty is 54% (World_Bank, 2009). According to Hankins(2009) only about 12% of the population have access to on-grid electricity. Hankins(2009) states also that

[1]There are undeniable evidences of anthropogenic climate change. According to IPCC-Intergovernmental Panel on Climate Change (IPCC, 2007), from observed widespread change in temperature of the earth's surface, free atmosphere and ocean , together with other parts of the climate system, there is strong evidence of warming caused by greenhouse gases during the past several decades. Some of the scaring evidences of global warming contribution are the widespread melting of snow and ice, and rising of the global mean sea level [See also (B. Ekwurzel , 2007)]. Among other effects, the sea level and temperature changes are, in turn, contributing to the slowing of the termohaline circulation (THC) (Schiermeier, 2006), what may lead to many other perturbations to the ecosystem, like the collapse of ocean plankton.
See also (Ehrhart, et al., 2006) for registered climate change impacts in Mozambique.

the main energy source for more than 80% of the population is wood and charcoal biomass, while Cumbe, et.al (2008) had placed this percentage at 89%.

In Mozambique, like in many other Southern Africa regions, most of the countryside families are scattered throughout the land, rather than being confined together inside villages or village like locations. This makes it very costly and difficulty to install and maintain grid based electricity distribution (as also happens for gas and water) to each family of such rural areas. This is confirmed by different sources: According to Hankins (2009), "Power transmission in Mozambique is an especially critical issue for the country for two reasons. First, the large size of the country and its dispersed settlement patterns make dispatching power to the entire population extremely expensive". In turn, Cumbe et al, (2008) confirm that about 80% of the Mozambique population live in geographically dispersed rural areas.

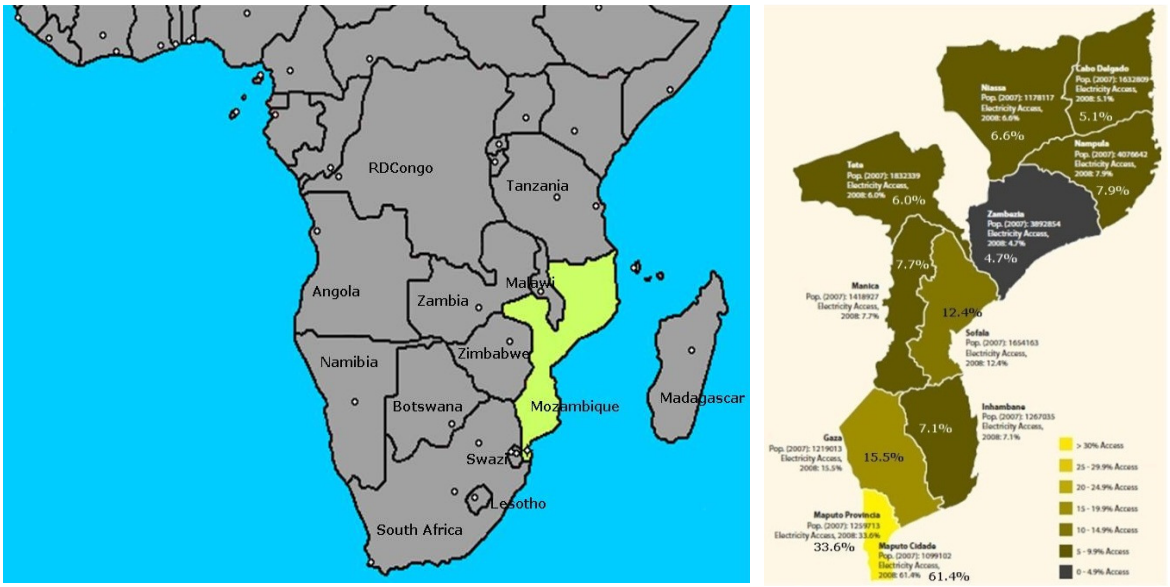


Figure 1 - At left: Mozambique and Southern Africa (Source: adapted from www.luventicus.org). At right: Mozambique On-grid Mean Electricity Access by Province. Adapted from: (Hankins, 2009).

Country development depends mainly on some major strategic factors: energy, water, transportation and communication infrastructures. In Mozambique, renewable hydroelectric energy (Cahora Bassa dam) is being distributed to all provincial cities and some district villages. As discussed above, installing and maintaining long power lines, aiming at distributing on-grid electricity to all the population, is not a cost effective solution for this big and long country. It is obvious that this type of energy distribution will scarcely ever cover the extent of the whole country, particularly the remote rural communities or dispersed families.

So, Mozambique (and generally Southern Africa) rural energy development should rely on renewable energies which can be generated and administered locally at off-grid small scale level. This point of view is shared by (Hankins, 2009) and other sources.

Mozambique (and Southern Africa) is located under the so called sun belt[2], which is characterized with high solar radiation, including its high availability during the day and throughout the year, not largely affected by seasonal variations. Thus, there is a special focus on solar sources, as a viable energy source. According to Cuamba, et al.(2006), Mozambique has an average daily solar radiation of about 5.7kWh/m^2. This results in about 2080 kWh/m^2 per year, which complies with the range of available radiation into the earth's sun belt.

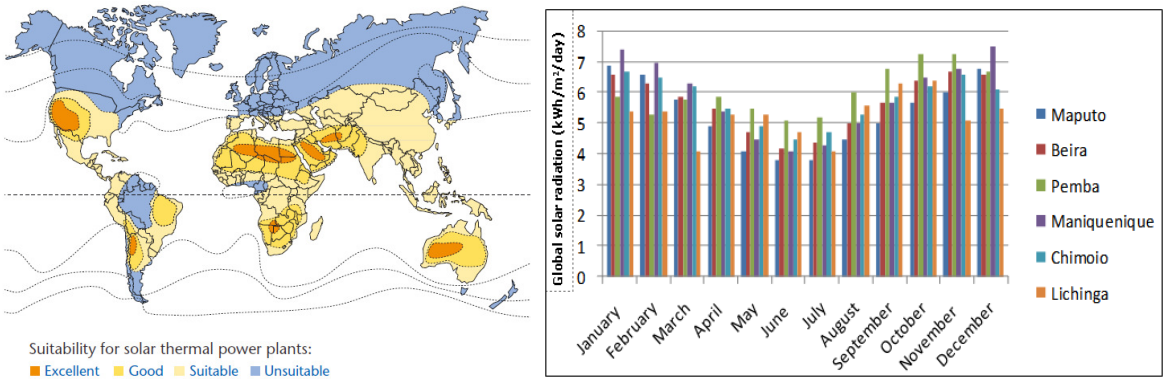


Figure 2 - At left: The world insolation distribution. Source: Schott AG (Ungeheuer, 2005). At right: Mozambique insolation in 6 distributed stations. Chart computed from data in (Cuamba, 2006).

1.2 Obstacles on the way towards the sustainable use of solar energy systems in current Mozambique context (similar to many developing countries)

There are not readily available low cost and appropriate systems for sustainable energy use in the Mozambique (Southern Africa) context. Commercially available photovoltaic systems must be imported and are very expensive for the very low country per capita income. Adding to the above, currently available systems are in many cases not energy efficient due to many possible reasons: (1) the intermittent[3] nature of solar source availability; (2) the embodied electronic management systems not being energy-centric[4]; (3) systems not being hybrid (dual or multi-source); etc.

All the above suggest the need to create cheaper and energy-centric / energy-efficient solar energy systems, appropriate and sustainable for a low per capita income country. This has to be accompanied by local capacity building in both human resources and infrastructures.

---

[2] Sun belt is the region of the earth inside roughly ±40 degrees of latitude from the equator. See glossary for further details.
[3] Intermittent availability of solar radiation: it is available only during the day and subject also to cloudy conditions.
[4] It is the qualification given to a system that privileges energy saving and energy efficiency, aiming at constantly meeting consumer's energy needs. See glossary for details.

1.3 Objectives

1.3.1 General Objectives:

- To create and contribute towards creating appropriate renewable energy technologies (focus on solar based sources) for sustainable distribution of energy to the rural or dispersed communities in Southern Africa;
- To promote the use of clean energies (and also the cleaner use of non renewable energies) in view of increasing the local energetic development, particularly the poor rural communities, thereby positively contributing to the global energetic and climatic future.

1.3.2 Technical and Specific Objectives:

- Study the model of an existing solar thermal energy system with heat storage, in the stand point of monitoring and control;
- Design, build and test a microcontroller based real-time monitoring, data-logging and control system for the aforementioned solar thermal energy system, focusing on the sun tracking subsystem. Study the behaviour and fitness of relevant system variables, focusing namely on the tracking related variables and some basic thermal collecting and storage related variables.

1.4 Support for collaborative work

The collaboration between UKZN and UEM (University Eduardo Mondlane, Maputo, Mozambique) made possible this work. This work from the UKZN side had a great support of the Norwegian Program for Development, Research and Education (NUFU) and from UEM side the work was jointly supported by NUFU as the umbrella of the "Research, Environment and Climate Change" programme at UEM/Physics and SIDA/SAREC (Swedish International Development Agency / Department of Research and Cooperation) as the umbrella of the "Technology Processing of Natural Resources for Sustainable Development" programme at UEM/Engineering. The fact that both universities are building identical solar thermal energy systems, gave the opportunity for this specific work in control subjects, as well as other works integrated in this solar thermal energy system development team.

1.5. Outline of the thesis report

Chapter I gives the introduction to the thesis giving a brief background and objectives of the research.

A summary of literature review and theoretical background in the fields of solar

energy and control techniques, is described in chapter II.

Chapter III describes a solar energy system pre-existing at UKZN School of Physics. It described the study that was carried out on the system in order to come up with a new model.

Chapter IV describes the new proposed model for microcontroller based monitoring and control.

In Chapters V and VI , the design of the experimental prototypes of the microcontroller based system, along with its real time operating program as well as the PC side data logging program, are explained.

In Chapters VII and VIII, the experiments are described and the collected results are discussed.

Chapter IX closes the report with conclusions and outlook of further work.

There are also five appendices (A to E) which contain: (A) Listing of the real time operating program, (B) Listing of the PC side data logging program, (C) Sample tables of collected/logged data, (D) Datasheets of some relevant components and (E) Photographs of some relevant system components.

Chapter II – Summary of literature review in the fields of solar energy and related applied control

In this chapter, we firstly approach the most relevant concepts of solar radiation and earth-sun geometry. Secondly, we discuss sun tracking and thermal energy collection concepts and techniques. Finally, we address the basic control systems concepts like: process identification, behaviour and model, the choice of the appropriate control technique and the most used control techniques.

2.1 Solar energy and its availability

2.1.1 Solar radiation

The Sun is an "inexhaustible" source of energy that is produced in the process of nuclear fusion of gases (mainly hydrogen into helium), similarly to what occurs in other stars life cycles. The radiation from the sun can be approximated to that of a blackbody with an effective temperature of about 5777 K (Duffie, et al., 2006), which irradiates energy with a given spectral distribution, according to Planck's law. The major portion of this irradiated energy lies between 250 and 3000 nm. Of this, a significant portion is the visible spectrum (380 nm to 780 nm) with a share of about 48% of the total irradiated energy. The other big share belongs to the infrared region (>780 nm) with 45.6% of the sun irradiated energy (Duffie, et al., 2006). Figure 3, shows the solar energy spectral distribution.

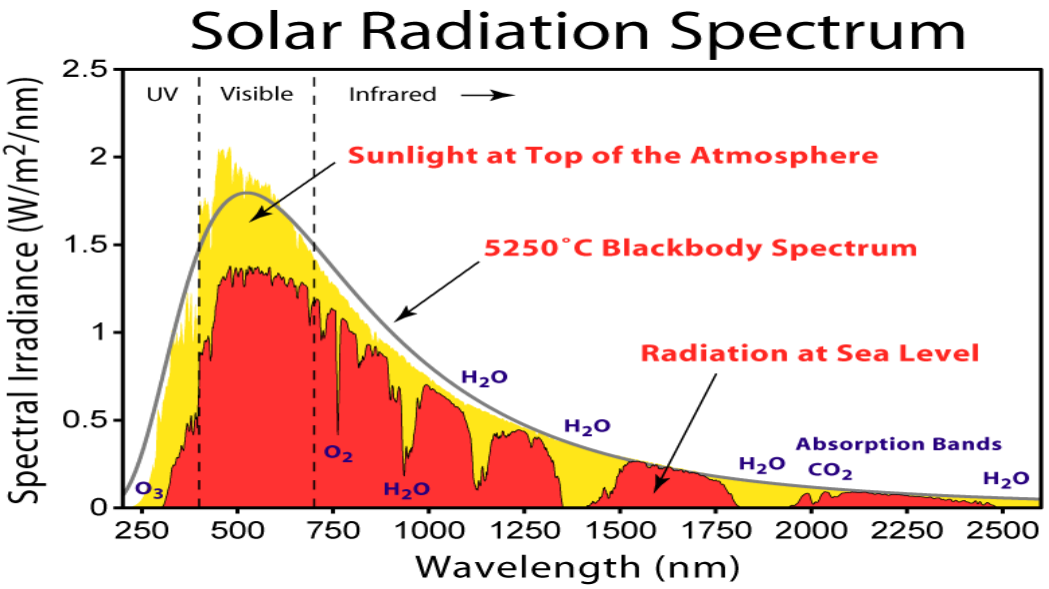


Figure 3 – Spectral energy distribution of solar radiation (for sun effective blackbody temperature of 5525K). Source: R.Rohde (http://www.globalwarmingart.com)

The solar radiation that hits the earth's upper atmosphere is called extraterrestrial radiation and fits nearly the Planck's blackbody law. The extraterrestrial solar radiation may vary due to many reasons, including sun spot activity (by ±1.5%), the earth's orbital eccentricity that varies the earth-sun distance (by ±3.3%) (Duffie, et al., 2006).

The solar constant (G_{sc}) expresses the extraterrestrial energy received from the sun per unit time outside the atmosphere, per unit area of a surface normal to the radiation, at mean sun-earth distance ($1.49x10^{11}$m).

$G_{sc} = 1367\ W/m^2$ (2.1.1.1)

To account for variations of extraterrestrial radiation due to variations of the earth-sun distance a variable magnitude of the energy expressed in terms of the solar constant is represented by G_{on}:

$$G_{on} = G_{sc}\left(1 + 0.033\cos\frac{360\,d}{365}\right) \quad (2.1.1.2)$$

or more accurately:

$$G_{on}=G_{sc}(1.000110 + 0.034221\cos B + 0.001280\sin B + 0.000719\cos 2B + 0.000077\sin 2B) \quad (2.1.1.2.1)$$

Where **d** is the day of the year (from 1 to 365), and B the fractional year, expressed as:

$$B = (d-1)\frac{360}{365} \quad (2.1.1.2.2)$$

When the radiation from the sun passes through the earth's atmosphere, it is partly scattered and/or selectively absorbed by atmospheric gases (mainly by O_3, O_2, H_2O, CO_2), dust and other substances present in the atmosphere (Brogren, 2004)[5]. The solar radiation that hits the earth's surface is reduced by about 30% (Brogren, 2004). This means also that the amount of solar radiation that reaches the earth's surface is affected by the zenith angle (see Figure 4 and Figure 6) since the greater the zenith angle the larger the atmospheric gap that it has to cross. This factor is expressed in terms of the air mass (AM), the ratio of the path length of the radiation through the atmosphere, given a zenith angle θ_z:

$$m = \frac{1}{\cos(\theta_z)} \quad (2.1.1.3)$$

where *m* is the air mass number. The minimum *m* (=1) is when θ_z is null, when the beam radiation is incident on the zenith line at equatorial latitudes, and it grows progressively with latitude.

The path length L (see Figure 4) corresponds to m=1 (θ_z=0) while L_z corresponds to $m=1/\cos(\theta_z)$ for any $\theta_z > 0$. The given formula (2.1.1.3) is a good approximation only for zenith angles between 0 and 70^0 at sea level . For angles near 90^0, and to account for effects of the earth curvature and the altitude (***h***), the expression below by Kasten and Young (1989) (as cited by Duffie, et al. 2006), is more accurate:

$$m = \frac{e^{-0.0001184*h}}{\cos\theta_z + 0.5057(96.080 - \theta_z)^{-1.634}} \quad (2.1.1.3.1)$$

where ***h*** is the altitude of the observer and θ_z the zenith angle.

[5] As also reported by JPL (Jet Propulsion Laboratory, 1998)

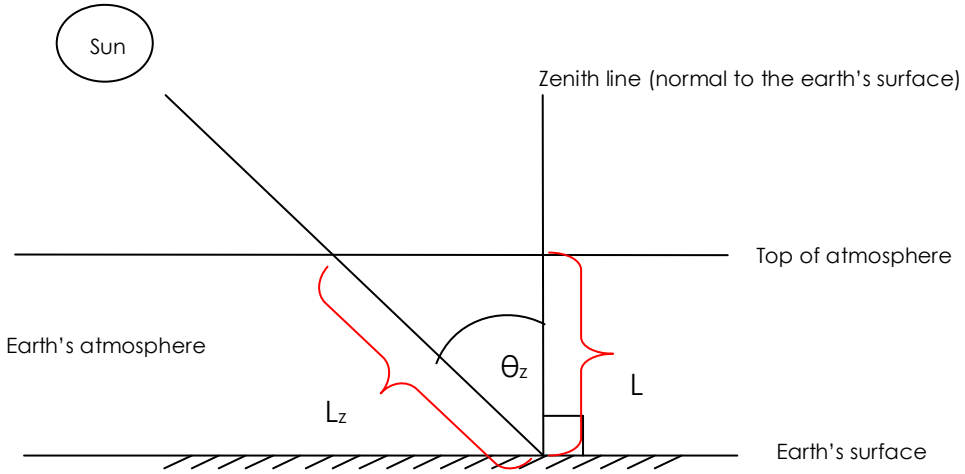


Figure 4 - Zenith angle and Air Mass

The amount of solar radiation that passes the top atmosphere layers may be further influenced by atmospheric conditions such as clouds, fog, rain, snow, etc.

In the end, besides all the obstacles described above, the solar radiation that is available in a certain geographic location, is intermittent[6] due to the earth's rotational movement. This is not quite a disadvantage, since the rotation guarantees a periodic distribution of the available sun energy to all the earth's regions affected only by the latitude and the day of the year. The selective absorption of some wave lengths (mainly the ultraviolet region) preventing them from reaching the biosphere[7] is also beneficial because they could otherwise cause serious health and long term irreversible ecosystem problems. Example: the ozone's depletion (a global climate change problem) allows high energy radiation (which is dangerous to live species) to enter the biosphere.

2.1.1.1 Types of solar radiation

The total solar radiation that hits the earth's surface classified in two types: beam and diffuse radiations:

- **Beam or direct radiation**: Is the one that hits the earth's surface without having been formerly deviated due to scattering, denoted as I_b;
- **Diffuse radiation**: the one that reaches the earth's surface after being scattered, denoted by I_d;
- **The total or global radiation**: is the sum of the beam and diffuse radiations, denoted as I_t.

[6] Intermittent: it is available during the day and also affected by cloudy conditions.
[7] Biosphere is the Earth's region comprising the living organisms and their habitats. This is a shared space which includes: a part of the atmosphere (the air zone of the lower layer), a part of the hydrosphere (waters) and a part of the lithosphere (solid earth).

2.1.1.2 Solar radiation detection and measurement

There are various types of photodetectors for detecting and measuring light, with different spectral responsivities. These include photodiodes, photoresistors, thermopiles, CCDs, solar cells, etc. For measuring the components of solar radiation (beam, diffuse or global as well as other related quantities), there are special instruments (built from elementary photodetectors) classified as radiometers or sun photometers. The key instruments are the pyrheliometer (sun photometer) for measuring beam, and the pyranometer (radiometer) for measuring either global or diffuse radiations (Duffie, et al., 2006) (Brooks, 2006).

2.1.1.3 Available clear sky solar radiation

The direct radiation (I_b) incident on a surface (say, a two axis tracking collector) with known latitude and time of day can be estimated through the equation of Hu and White as cited by Alata (Alata, et al., 2005):

$$I_b = G_{on} \times 0.7^{m^{0.678}} \qquad (2.1.1.4)$$

where G_{on} is the solar constant (as in eq.2.1.1.2 and 2.1.1.2.1) and m (as in eq.2.1.1.3 and 2.1.1.3.1) is the air mass number. This magnitude is important when evaluating the efficiency of the tracking and concentrating systems for clear sky conditions.

2.1.2 Solar Radiation Geometry with Respect to Earth and Collector Surface

2.1.2.1 Solar Radiation and The Generic Collector Surface

The position of the sun relative to earth and collector surface, varies continuously during the day with the earth's rotation and during the year with earth's translation. The orientation of collector surface should allow capture of maximum possible energy. To meet this requirement the collector surface may need sun tracking. This in turn, normally requires a coordinating and control system. The purpose of tracking the sun for maximizing the power output of a solar concentrator system or photovoltaic panel or thermal collector, needs a basic understanding of solar position with respect to the desired geographic location and with respect to the surface receiving the solar radiation.

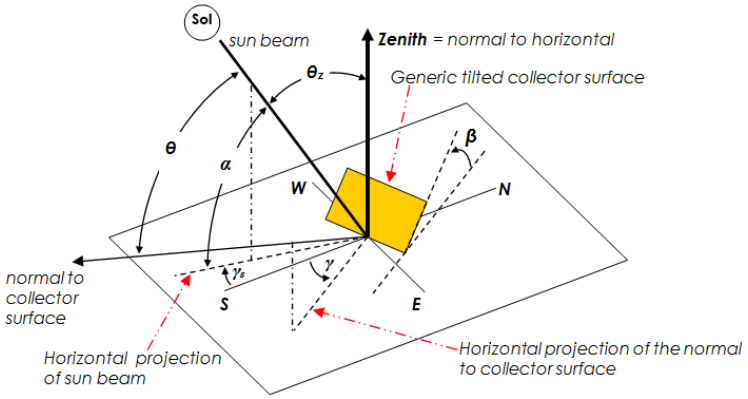


Figure 5 – Sun-earth-collector geometry. Adapted from Duffie, et al., (2006)

Figure 5, shows the main geometric relations that are used to describe the way that the sun rays, the collector surface and earth surface positions, relate to each other. Additional relations are depicted in Figure 6. All these and additional variables and constants are listed and discussed in detail in next section, while details about collectors and tracking techniques are discussed in section 2.1.3.

2.1.2.2 Sun-Earth-Collector Geometry and Other Related Variables and Constants

There are many variables and constants that describe different geometric and time magnitudes. These magnitudes can be found in different cited sources in a dispersed fashion. So, to address a concise and clear approach for the present work, we collect together the ones that are immediately relevant. Other quantities are derived, calculated or otherwise well known. Some basic quantities and definitions (many of them as found in Duffie, et. al., 2006) are:

(1) **Sun-earth mean distance** (*SE*) =1.495x10^{11} m (= 1 astronomical unit);
(2) **Sun diameter** (*SD*) =1.39x10^{9} m;
(3) **Earth diameter** (*ED*) =1.27x10^{7} m;
(4) **Apparent diameter of the sun:** maximum angle between 2 sun rays that hit the earth's surface is ε=32'. This can be obtained from the above, as:
$\varepsilon = 2arcsin\left(\frac{SD}{2SE-ED}\right) \cong 0.53^0 \cong 32' \cong 9.3\, miliradians$ (2.1.1.4)
This becomes larger in December solstice and smaller in June solstice.
This value should be taken into account when determining the acceptance angle of a solar collector and hence in determining the necessary sun tracking resolution and accuracy. This quantity represents the nominal (minimal) acceptance of any collecting surface.
(5) **Earth's polar axis declination**: $\delta_0 = 23.45^0$;
(6) **Solar apparent angular speed** (due to earth's rotation):
ω_s = 360^0/24h = 15^0/h = 0.25^0/min = 15''of arc/s (2.1.1.5)
$\omega_s = 2\pi\, rad/24h = \frac{\pi}{12} rad/h = \frac{\pi}{720} rad/\,min = 72.72\mu rad/s$ (2.1.1.5.1)
(7) **Local meridian**(L_{loc}): is the longitude of the observer (ex: at Durban/UKZN plant: L_{loc} = 30^0 56'40.0"E = 30.94^{0}E);
(8) **Standard meridian** (L_{st}): is the longitude of the standard meridian for the local time zone. Example: Durban/UKZN plant is in GMT+2 time zone, which corresponds to 30^{0}E;
$L_{st} = 15 * (t_{loc} - t_{GMT})$ (2.1.1.6)
For UKZN (and entire RSA): $L_{st} = 15 * 2 = 30^0$ (2.1.1.6.1)
(9) **Solar noon**: is the time when the sun is over the local meridian. Then, the projection of zenith line is collinear with the N-S axis; hour and azimuth angles are then null.
(10) **Local or standard time** (L_{time}): the local clock time (politically/economically determined);
(11) **Day number** (***d***): is the day of the year, being *d=1* at 1st January. In 31th December *d*=365 (= 366 for leap years);

(12) **Equation of time** (*E*): is an equation that accounts for perturbations on the earth's rotation speed and hence on the apparent sun angular speed (this also makes solar time a varying axis). E in minutes is:

$E = 229.2(0.000075 + 0.001868\,cos\,B - 0.032077 sin\,B - 0.014615 cos2B - 0.040849\,sin2B)$ (2.1.1.7)

where *B* is the fractional year. See equation 2.1.1.2.2

(13) **Solar time** (S_{time}): time based on the above sun's apparent angular motion:

$S_{time} = L_{time} + 4(L_{st} - L_{loc}) + E$ (in minutes) (2.1.1.8)

For Durban/UKZN solar plant S_{time} becomes:

$S_{time} = L_{time} + 4 * (30 - 30.94) + E = -3.76\,min + E(\text{min}) + L_{time}$ (2.1.1.9)

(14) **Zenith line**: the normal to the horizontal surface which is also the line from the centre of the earth, that crosses (the point of) the desired geographic location (see also Figure 4 and Figure 6);

(15) **Zenith angle** (θ_z): the angle of incidence of beam radiation with respect to the zenith line on the desired geographic location (see also Figure 4 and Figure 6);

(16) **Incidence Angle**(θ): the angle of beam of radiation with respect to the normal to a desired surface - examples: the normal to the receiving face of a flat plate collector or (the axis of) a paraboloidal concentrator;

(17) **Surface azimuth angle**(γ): If a desired surface is denoted by $\boldsymbol{\sigma}$ and the normal to that surface by $N_{\boldsymbol{\sigma}}$: Then, the surface azimuth angle is the angle of the horizontal plane projection of $N_{\boldsymbol{\sigma}}$ with respect to local meridian;

(18) **Solar azimuth angle**(γ_s): the beam radiation's horizontal projection angle with respect to south, where angular displacements eastwards are negative;

It is worth mentioning that, there are other used conventions in the definition of the azimuth angle, for example the one whose zero degrees reference is north (See Figure 6);

(19) **Solar altitude or elevation angle** (α): It is the complement to the zenith angle (θ_z);

(20) **Slope or surface tilt angle** (β): It is the angle of the desired surface with respect to the horizontal;

(21) **Declination angle**(δ): It is the angle between the sun beam radiation and the equatorial plan. This magnitude varies from - δ_0 to + δ_0 (-23.45^0 to +23.45^0) governed by the following equations [by Cooper(1969) and Spencer (1971) respectively, as cited by Duffie, et al., 2006]:

$$\delta = \delta_0 sen\left(360\frac{284+d}{365}\right) \quad (2.1.1.10)$$

Where d is the current day of the year. For more accuracy:

$\delta = 0.006918 - 0.399912\,cos\,B + 0.070257\,sin\,B - 0.006758\,cos\,2B + 0.000907\,sin\,2B - 0.002679\,cos\,3B + 0.00148\,sin\,3B$ (2.1.1.10.1)

where *B* is the fractional year. See equation 2.1.1.2.2.

(22) **Latitude angle**(ϕ): It is the angle between the zenith line and the equatorial

plane; it is the latitude of the observer (At UKZN solar plant: $\phi = 29^0\,49'2.0''S$).

(23) **Tracking angle** (ρ): It is the angle that is produced when a tracking axis rotates about itself. This angle (or angles) is determined by the tracking type adopted. For example, in one axis tracking (see section 2.1.3.3.1.2) the tracking angle ρ corresponds to the hour angle.

(24) **Longitude angle** (λ_c): It is the longitude of the observer($\lambda = L_{loc}$).

(25) **Hour angle**(ω): It is the angular displacement of sun beam from the local meridian, being negative eastward and positive westward. ω (in degrees) relates to S_{time} (in minutes) as:

$$\omega = S_{time} * 0.25 - 180^o \qquad (2.1.1.11)$$

(26) **Sunrise angle**: It is the hour angle at sunrise;

$\omega_{sunrise} = -\cos^{-1}(-\tan(\phi) * \tan(\delta))$

(27) **Sunset angle**: It is the hour angle at sunset. It is numerically the symmetric of the sunrise angle.

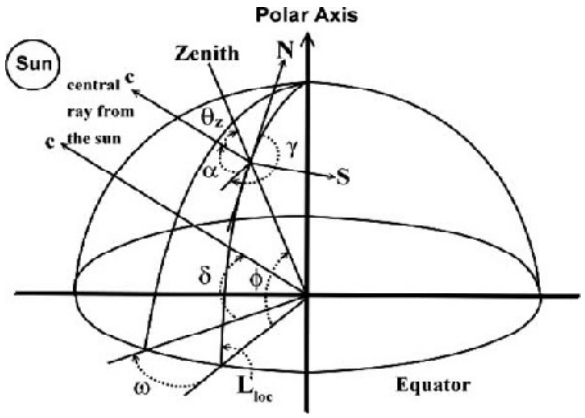


Figure 6 - Representation of solar angles (Stine, et al., 2008).

Note that the azimuth angle (γ) depicted in the Figure 6, is the one whose zero degrees reference is pointing due north, while the same angle defined formerly, references to south; however both the angle defined formerly and the one depicted assume the clockwise angular motion as positive.

Of all these magnitudes, the most important ones in the point of view of sun tracking for maximum power collection, namely α, γ, δ, ω (and other), will be further discussed in the following chapters/sections.

2.1.3 Solar energy collection and tracking configuration

Solar energy collecting may be performed by using different types of solar collectors in fixed or tracking configurations. During the solar energy collection technology long roadmap, a number of different types of collectors and tracking approaches were used, some of which are highlighted here.

2.1.3.1 Solar collectors

There are various types of solar collectors of different shapes, in either single unit or multiple unit arrangements, of which, the most widely used are briefly considered below.

1) **Photovoltaic (PV) panels**. A PV panel is based on (it is an association of) PV cells, that convert sun light directly into electrical power. In turn, PV cells (or even PV panels) can be found as receivers for concentrating (or reflecting) collectors. PV panels are found normally as flat surfaces. Figure 7 shows an example of PV panel;

Figure 7 - A PV panel array (Stine, et al., 2008)

2) **Solar-thermal collectors**: Are the ones where the collected sun light is converted into heat (thermal energy), which can be stored for later use, including the conversion to other forms of energy. There are different types and shapes (and collective arrangements) of thermal collectors, namely:
   a) Flat plate collector (shown in Figure 8): converts sun light directly into thermal energy, which is transferred to a fluid (commonly water, as in widely used water heating applications).

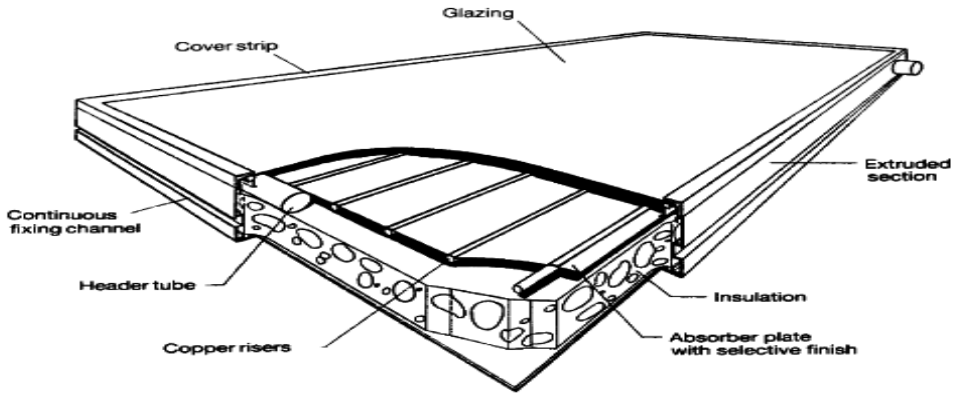


Figure 8 - A flat plate collector (Kalogirou, 2004)

b) Parabolic trough concentrator (PTC): concentrates sun light onto a (heat, PV or hybrid) linear receiver. In Figure 9 below, a diagram and a photograph of a PTC are shown.

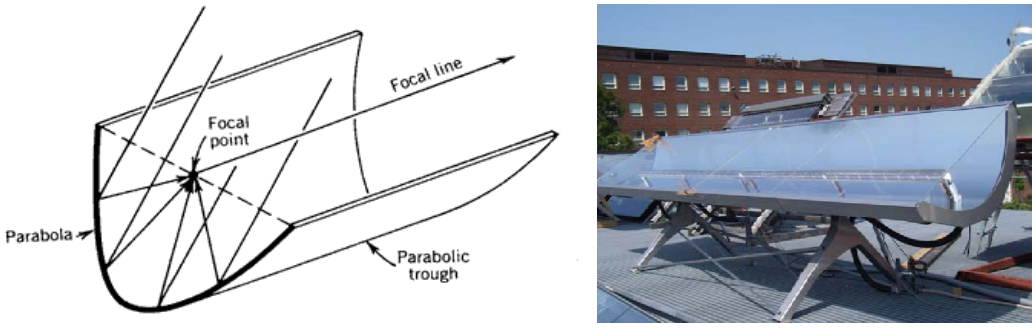


Figure 9 - Parabolic trough concentrating collector

c) Parabolic dish concentrator. It is a paraboloid of revolution, which concentrates sun light on a single point. In practice, however, the focus of the dish is not a point but an extended area, which is usually in the approximate shape of a circle. The reason for the extended focus area is due to the fact that the sun's rays are not exactly parallel as well as deviations of the dish from a perfect parabolic shape. A suitable receiver is placed at the focal region of the dish. One type of receiver is of a thermal type in which circulates a heat transfer fluid. A Stirling engine is another type of receiver used (which converts heat into electricity). In turn, in some parabolic dish based solar cooking applications, the receiver is the cooker itself. In Figure 10 below, a drawing shows a parabolic dish collector.

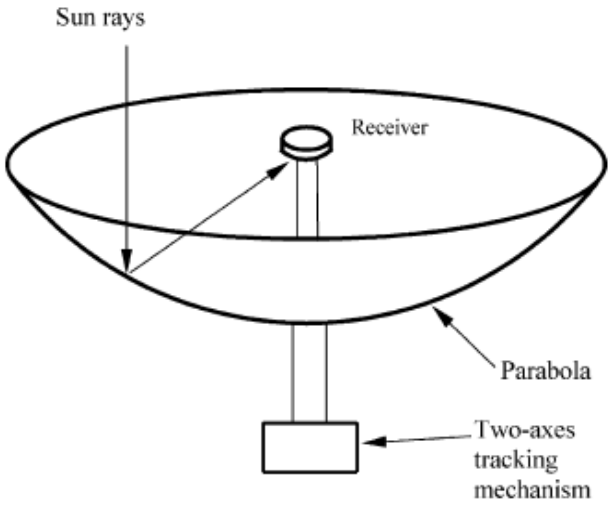


Figure 10 - A parabolic dish collector (Kalogirou, 2004)

d) Compound parabolic concentrator (CPC). Like a PTC, it concentrates sun light onto a (heat, PV or hybrid) linear receiver. The drawing in Figure 11 shows a CPC.

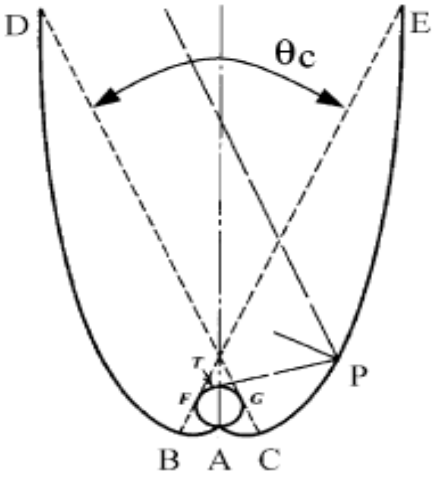


Figure 11 - Schematic diagram of a compound parabolic concentrator (Kalogirou, 2004)

e) Central (heliostatic) receiver field and Fresnel linear reflector (FLR).

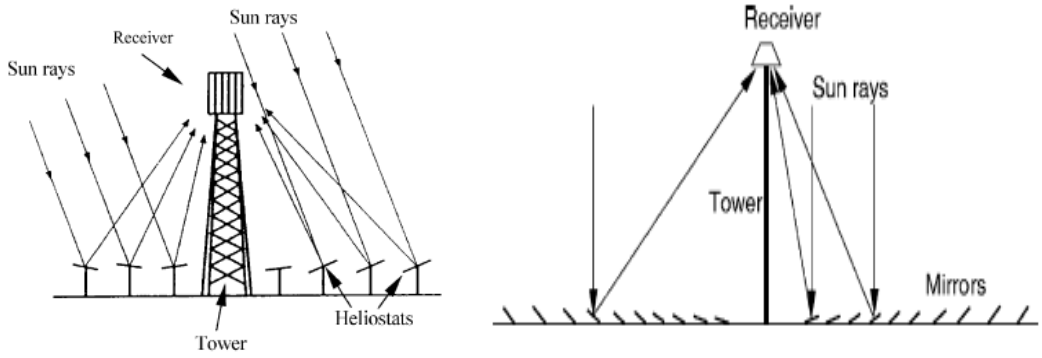


Figure 12 - Central receiver and FLR (Kalogirou, 2004)

There are other collector types, geometries and spatial collective arrangements,

each one with its own characteristics which may be appropriate for some particular application. Just to mention some: cylindrical trough concentrator, spherical concentrator, evacuated tube collector, etc., as addressed in various literature. See Kalogirou (Kalogirou, 2004) for a detailed discussion of these.

2.1.3.2 Solar position detection and alignment

For the purpose of tracking the sun for maximum solar energy collection, there is a need for detecting the position of the sun, aiming the concentrator at it and tracking its position during the day and throughout the year. It is possible to detect and aim at the sun using a number of light or position sensor devices (including photodiodes, phototransistors, LDRs, photocells, CCDs, potentiometers, bimetallic strips, fluid-mechanical devices, etc.), arranged in a certain way. Prapas, D.E. (Prapas, et al., 1986) used four LDRs for sun detection and alignment to provide azimuthal single-axis tracking of a parabolic trough concentrator. LDRs, however, are criticized by their inability in distinguishing between direct (desired) and diffuse sunlight. It was found that this drawback, however, can be overcome by using compensating resistors (Kalogirou, 1996). Khalifa, A. and Al-Mutawalli, S. (Khalifa, et al., 1998) used phototransistors to provide a dual axis tracking of a CPC assembly. Roth, Georgiev and Boudinov (Roth, et al., 2005) used rotary potentiometers as position sensors, whilst K. Aiuchi et al. (Aiuchi, et al., 2006) used photocells to perform the solar detection and aiming.

2.1.3.3 Solar aiming and tracking

There are different types of solar aiming and/or tracking strategies, which are divided into two branches: passive versus active tracking. Passive tracking uses neither motor nor any power consuming device to perform the sun tracking. It uses the received solar energy to perform the tracking assembly motion.

An example of a passive tracker is one developed by (Clifford, et al., 2004). A similar example is represented in Figure 13, developed by Zomeworks (Zomeworks, 2008). It uses a fluid that is heated by the sun light, causing it to migrate and accumulate at the opposite side where it condenses. This causes the surface to tilt, seeking a new gravitic equilibrium.

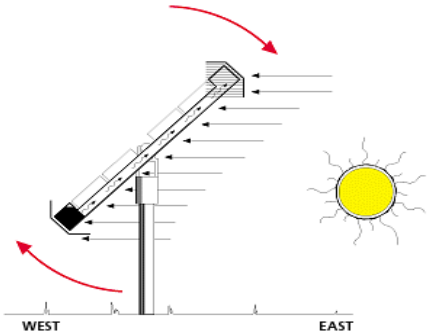


Figure 13 - Passive sun tracking system (Zomeworks, 2008)

Active tracking techniques (normally electrical/electronic based) involve the use of an active actuating device to provide the desired movement, commonly, motors of different types (stepper, DC, etc.). A monitoring and control apparatus is also required to properly control the actuator.

2.1.3.3.1 Types of collector positioning / sun tracking

Collector positioning is the task of aiming the collecting surface at the sun so as to null the light incident angle and thereby maximize the energy collecting efficiency.

It should be pointed out that the choice of a tracking strategy is influenced by collector type, along with a trade-off between performance and cost. Comments will be added later about the suitability or fitness of a particular collector for specific types of control strategies.

There are three major types of collector positioning / sun tracking: Fixed positioning (non-tracking), one axis tracking and two axes tracking. We describe each of these methods briefly:

2.1.3.3.1.1 Fixed (non-tracking)

In this configuration the collector assembly is fixed. The collector surface is tilted by the latitude angle of the location and the normal to the surface is made coplanar with the local meridian. In most cases the collector assembly is strictly fixed, with no adjustment whatsoever. However, in some cases, there may be seasonal adjustments of the collector's tilt angle so as to account for seasonal variations of the solar light incident angle, where the gain in efficiency is considerable: For mid-latitudes, tilt angle adjustments at every three months, only increases the annual energy production by less than 5%; (Wenham, et al., 2007).

Fixed (strictly or seasonally adjusted) tracking is only suitable for collector shapes and configurations whose efficiency is not greatly affected by the cosine of the light incident angle.

2.1.3.3.1.2 One axis tracking:

In one of the possible configurations, the axis is made parallel to the earth's rotation axis (i.e., it is a polar axis) and the corresponding tracking angle is made equal to the hour angle (ω). To make the axis parallel to the earth's polar axis, the axis should be coplanar with the local meridian and its tilt angle (slope) should be made equal to the latitude of the location (see Figure 14). As in the case of fixed tracking, in some cases, there may be the need for seasonal adjustments of the collectors tilt angle so as to account for seasonal variations of the solar light incident angle. This is done in case where the gain in efficiency is considerable

compared against tilt adjustment implementation costs.

An azimuthal one axis tracking is also possible (Prapas, et al., 1986), where the axis is made vertical, i.e., collinear to the zenith line; the corresponding tracking angle is than the azimuth angle (γ).

It should be pointed out however, that for some types/shapes of solar collectors (like a PTC or a), a one axis tracking, at the least, is compulsory.

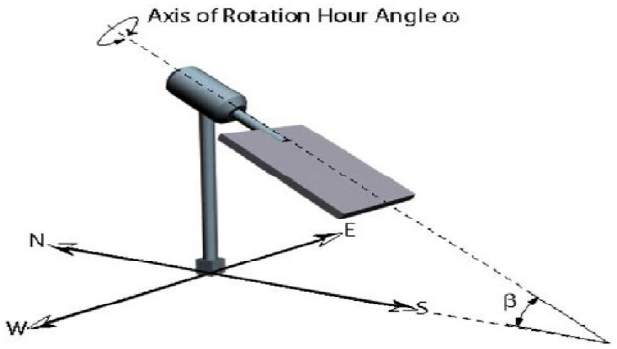


Figure 14 - One axis sun tracking systems with tilt angle equal to the latitude angle (Alata, et al., 2005)

2.1.3.3.1.3 Dual axis tracking

There are two major types of two axes tracking: the polar (equatorial) (or hour-declination) tracking (Figure 15) and the azimuth-elevation (azimuth-altitude) tracking (Figure 16). Dual axis tracking may increase the annual energy production around 30% (Wenham, et al., 2007). It should be pointed out that for some types/shapes of solar collectors, the point focus type collectors (like the paraboloidal type concentrator), dual axis tracking is mandatory.

2.1.3.3.1.4 Polar (equatorial) tracking

In this type of tracking (see also Figure 15), the tracking angles are the hour (ω) and declination (δ) angles. The main axis (hour) is made parallel to the earth's polar axis and perpendicular to the equator's plane. This is equivalent to orienting this axis north-south and tilting it by the latitude angle of the location. When this axis rotates (eastwards or westwards) the tracking angle is the hour angle (null at solar noon when the sun is at the local meridian; positive westward and negative eastward), while the other axis is perpendicular to the main one and when it rotates (northwards or southwards), the tracking angle is the declination angle (null at equinoxes; negative southwards from September to March equinox and positive northwards from March to September).

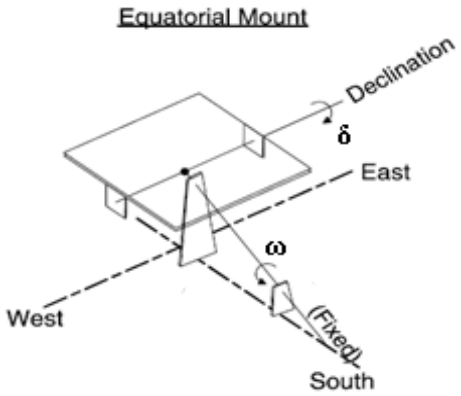


Figure 15 - Dual axis equatorial tracking (north hemisphere) (adapted from Aiuchi, et al., 2006).

2.1.3.3.1.5 Azimuth elevation tracking

In this type of dual axis tracking (see Figure 16), the tracking angles are the azimuth (γ) and altitude (α) angles. In one of the possible approaches, the one axis is made vertical (i.e., collinear to the zenith line) and the corresponding tracking angle is then equal to the solar azimuth angle (γ_S), while the other axis is perpendicular to the first one and the corresponding tracking angle is equal to the elevation angle (Alata, et al., 2005).

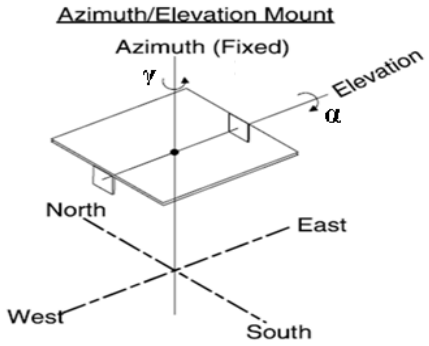


Figure 16 - Dual axis azimuth/elevation tracking (adapted from Aiuchi, et al., 2006)

2.1.3.3.2 Polar axis versus azimuth-elevation axis tracking.

The major advantage of polar axis tracking (over the azimuth-altitude approach) is the slow variation of declination tracking angle. This allows for very long (time) step adjustments and thus saving power and allowing for longer motor and tracking assembly life expectancy. This same characteristic (of declination tracking angle low variation) is what favours the existence of a one-axis (polar axis) tracking needing only seasonal tilt angle adjustments.

Conversely, in the azimuth-altitude approach, both tracking angles (azimuth and altitude) have moderate to fast variations during one single day. That makes this tracking approach somehow power hungry and not very appropriate for collectors that are cosine-effect sensitive. In general that approach should be discouraged where low cost, sustainable and energy-centric solutions are desired.

2.1.3.3.3 Manual versus automatic tracking

Solar tracking may be performed manually or automatically.

In manual tracking, a human is in charge of periodically or seasonally changing the position of the tracking assembly aiming it at the sun, guided by a schedule. In automated tracking, a machine (a controller and actuator) is used to perform the tracking tasks. The major drawbacks of manual/human executed tracking are that the periodic step adjustments cannot be as short as can be with automated tracking and on the other hand they are subject to possible human timing and positioning errors. Manual tracking is mostly likely to be used with collectors whose efficiency is not greatly affected by the cosine of the sun light incident angle (θ).

2.1.3.3.4 Open loop versus closed loop tracking

Open loop tracking: the relative position of the sun is mathematically determined using the solar time and the suitable tracking angle equations. This does not account for possible disturbances, and other types of perturbations, widely addressed in next sections. This type of tracking is suitable for simple systems where a desired positioning accuracy is guaranteed and the effect of disturbances is negligible. Abdallah and Nijmeh (2004) built a PLC based open loop dual axis tracking system that was found to be efficient.

Closed loop tracking: the actual alignment of the controlled surface to the sun position is measured and compared to the desired position (the set point), and a corrective action taken to minimize this error. A number of closed loop systems for providing one axis and dual axis tracking were mentioned above including: (Aiuchi, et al., 2006), (Kalogirou, 1996), (Khalifa, et al., 1998), (Prapas, et al., 1986) and (Roth, et al., 2005).

2.1.3.3.5 Continuous versus step tracking

When the variation of the tracking angles is relatively slow and/or the degradation of collector efficiency relative to the variation of the tracking angle is considerably low, a step tracking is advisable, since, most likely it saves power. Only for very accurate tracking needs (like in automatic pyrheliometer applications) a continuous tracking is compulsory. For parabolic or paraboloidal concentrators, the collector's acceptance angle determines the maximum size of the tracking step. The tracking step may be defined in degrees or time (minutes), deriving it from the equations that relates solar time with the tracking angle.

2.2 Basic Control theory concepts

In this section, we discuss some concepts of control systems, which are the basis for understanding the design of the data acquisition and control system, addressed in the chapters ahead.

2.2.1 Open loop VS closed loop control system

The most generic presentation of a system (a plant or generally a process) is as illustrated in Figure 17:

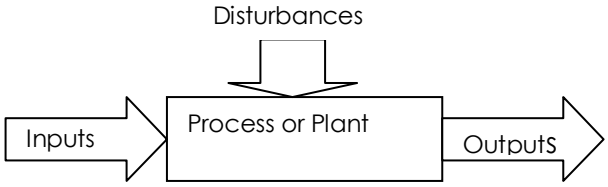


Figure 17 - Conceptual diagram of a plant or process

The inputs are the applied signals and unwanted disturbances. The outputs are the desired output signals, in other words, the system behaviour. This behaviour may be a good one or an unsuitable behaviour. A controller comes into reality for the aim of driving the system into, and maintaining it, at the desired behaviour. The new system is depicted in the conceptual diagram of Figure 18:

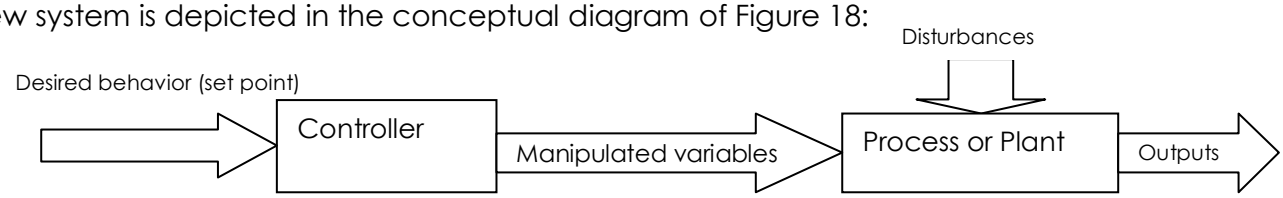


Figure 18 - Conceptual diagram of an open loop control system

This type of control system (in Figure 18) is called an open loop system as it does not check if the outputs (process behaviour) are the desired /expected ones. What may generally happen is that, due mainly to disturbances, the outputs may deviate from the desired/expected values. This is a major drawback of open loop systems, as they do not take into account the effect of possible disturbances, as well as possible internal variation of system parameters (due for example to mechanical or electromechanical wear or components wear). In the presence of a disturbance, an open loop system will fail performing the desired task: the controlled variable will deviate from the desired value. An open loop approach is suitable for processes where the relation between the inputs and output is known and there are no disturbances (Ogata, 2002).

A better system that accounts for possible disturbance inputs, is a closed loop (feedback) control system, in which the outputs (system behaviour) are monitored (measured) and compared with the desired input variables (set points) and then the difference (error) is used to adjust the manipulated variables, thus acting against disturbances. The conceptual diagram of such a system is shown in the Figure 19.

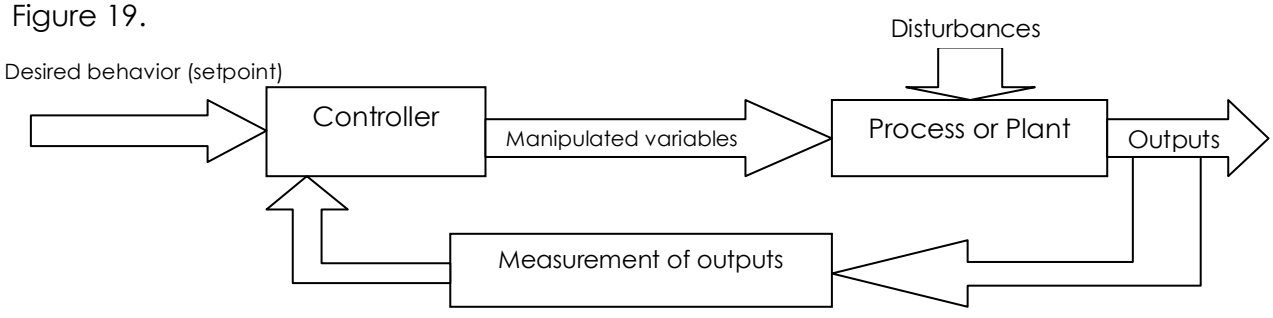


Figure 19 - Conceptual diagram of a closed loop (feedback) control system

A more formal block diagram of a feedback control system is depicted in Figure 20 below, where $V_s(t)$ is the set point variable (the desired reference input) value; $V_c(t)$ the actual (output) value, $e(t)$ is the error (difference between the desired and the actual values, and $V_m(t)$ is the manipulated variable, which represents an action to correct (to null or normally to minimize to an acceptable value) the error $e(t)$, and t is time.

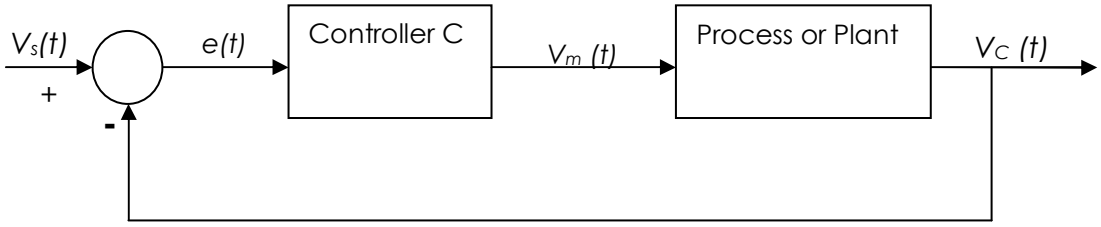


Figure 20 - Block diagram of a feedback controlled system

2.2.2 System identification, Process model, process behaviour and performance characteristics.

2.2.2.1 System identification and choice of control technique.

In order to design a controller for a given process, one needs, in general terms, to know its (uncontrolled) behaviour (in open loop) along with the desired behaviour and performance specifications.

For doing that, the first step is a system identification, or say, to build a knowledge about the process variables: the number and nature of its inputs and outputs (linear or non-linear, static or dynamic, time-invariant or time-varying, discrete or continuous, etc.) the way that they dynamically relate to each other and with the operating environment.

Indeed according to the type and complexity of the particular process, the number and nature of its inputs and outputs, the operating environment, the desired behaviour and performance specifications, a process controller can be chosen to be open-loop or feedback, analogue or digital, etc. For instance:

a) Processes that are single input and single output (SISO), linear and time-invariant (LTI), can be dealt with in the field of classical control systems, that is, using transfer functions / Laplace (or also Fourier) transforms representations and analysis/synthesis techniques. Then, ordinary feedback control strategies (with P, I, PI, PD or PID) may be used as controllers; whilst,

b) For processes that are multiple inputs and multiple outputs (MIMO) with linear, including time-varying behaviour, state space representations are the appropriate choice.

c) In turn, processes that are SISO or MIMO but non-linear or, rather, with an unknown or poorly defined mathematical model, can be addressed by advanced control techniques in the branch of computer based control

techniques, like fuzzy logic control, neural networks based control, finite state machines based control, PLCs, etc. Also, the combination between these control techniques may be used, like: Fuzzy-PID or Neuro-Fuzzy [as with (Linkens, et al., 1998)].

On the other hand, there are special control techniques, aimed at enhancing the controller's performance, particularly for the systems mentioned in a) and b). Such enhancement is achieved by addressing uncertainty, linearity deviation, perturbations in the system parameters, complexity. Enhancement is also achieved by looking for robustness, better or optimal performance. Such special control techniques include: feedforward techniques, gain scheduling techniques, model predictive control, internal model control, optimal control, H_∞ control, etc. These techniques can be found in many control engineering textbooks. Some of these are well considered and applied by Camacho, Berenguel and Rubio in their work on advanced control of solar plants (Camacho, et al.).

It is worth noting that the modern trend of control systems is the widening use of computer based control devices and techniques, from tiny microcontroller based sensors - like the smart dust (Elsevier, 2001) (Crossbow_Technology, 2009) (Dust_Networks, 2009) to PLCs and computer supervisory control (just to list some).

2.2.2.2 Process model

A common and most effective way of describing a process is determining its process model, a mathematical representation of the dynamic system behaviour in form of differential equations. A system may have many mathematical models depending on the perspective. Also a mathematical model may assume different forms (Ogata, 2002).

Once a mathematical model is defined the system behaviour can then be studied by using analytical and/or graphical methods. A number of system characteristics may then be determined, such as stability, observability, controllability, etc. However, not all the processes can have their mathematical models easily determined. In such a case, experimental methods come to the rescue for determining their behaviour to an acceptable approximation.

Once the system behaviour is discovered and the desired performance characteristics are defined, a controller can then be designed and added to the system. Regardless of having or not a mathematical model, a system behaviour is what is definitely necessary to design and tune a control system that will affect the particular process in a way that it will assume a desired behaviour and performance to an acceptable degree.

2.2.2.3 Process behaviour

A common way used for determining the behaviour of a process, is studying its open loop transient and steady state response to a number of possible input signals (step, ramp, parabolic, etc.). This can be done analytically (when a

mathematical model exists) or experimentally (regardless of the existence of a mathematical model).

Frequently, as a common starting point, the open loop transient response of the process to the unit step input is determined. Then a number of common behaviours/characteristics are analyzed like: delay time[8], rise time, time constant, overshoot (ratio), settling time, steady state error, etc., as suggested in Figure 21:

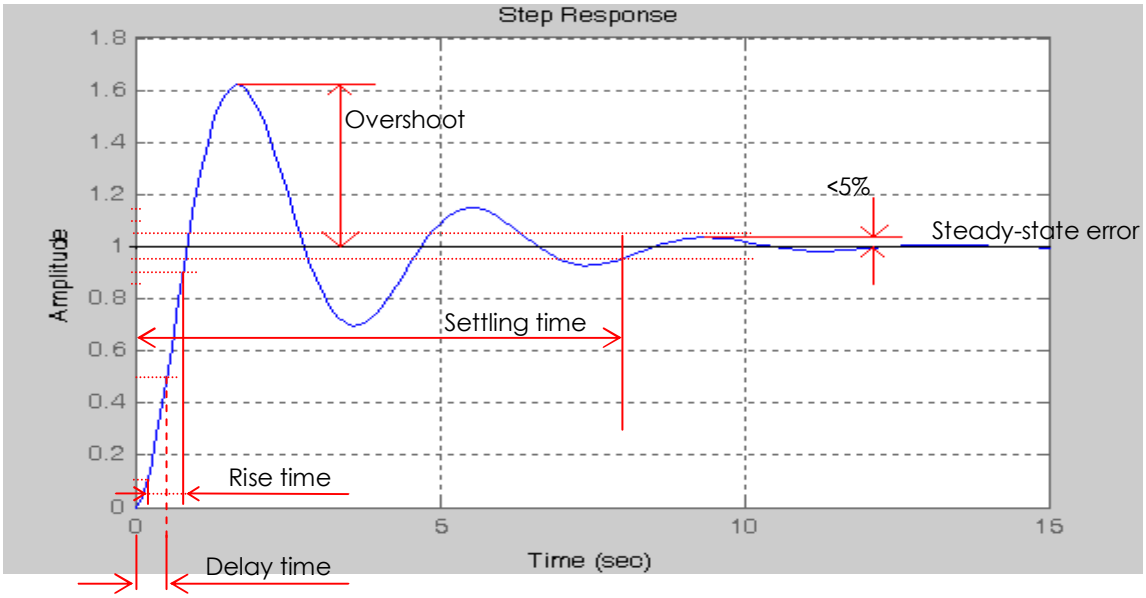


Figure 21 – Behavioural / performance characteristics of a process in response to a unit step input

2.2.3 Some Traditional and Advanced Control Techniques

It is worth discussing some of the most commonly used control techniques. First we present conventional control techniques in the industry and later followed by some advanced/computer based control strategies. This is to highlight their importance in the arena of applied control systems and introduce a possible choice for the current application in the field of solar tracking and solar thermal applications.

[8] Definitions (some respecting to Figure 21, a 2nd order underdamped system):
Error: the error of the output relative to the setpoint, in a specific instant of time.
Steady state: is the value of the output as time approaches infinity.
Steady state error: is the error (of the output) as time approaches infinity.
Delay time: the time it takes for the output to become 50% of the setpoint.
Rise time: the time it takes for the output to transit from %10 to 90% of the setpoint.
Settling time: the time it takes for the output to die to below 5% of error (absolute value).
Overshoot (ratio): is the normalized value of the first peak over the setpoint.
Time constant: Is the convergence speed of the output approaching the setpoint.

2.2.3.1 On-Off Control

On-off (or also called bang-bang) control is one of the most widely used control techniques. It is a single input (the reference or set point) and single output (the controlled variable) system where the value of the set point must be maintained at the output. A good example of a common application of on-off control is the thermostat control of the temperature of an electrical stove that can be frequently found in many domestic and industrial temperature control for stoves, furnaces, etc. This type of controller is represented in Figure 22 and Figure 23.

This type of control technique is best defined by the relations and diagram depicted below:

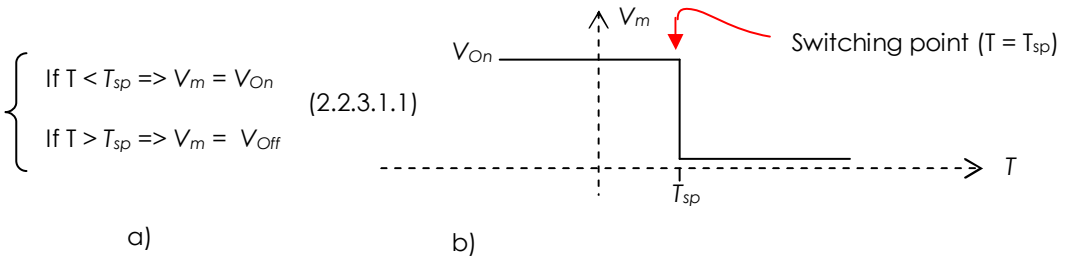


Figure 22 - Working principles of a simple on-off controller

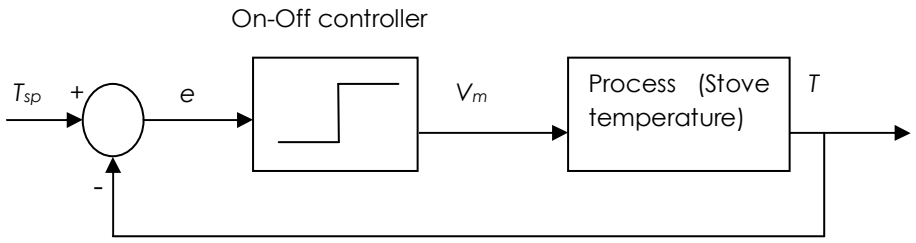


Figure 23 - Block diagram of a on/off feedback control system

In Figure 22 and Figure 23, T_{sp} is the switching point temperature (the set point); T is the stove temperature; V_m the manipulated variable (control action: it switches On or Off the stove power, according to the value of ***e***); ***e*** is the error: $e = T_{sp} - T$;

The major drawback of the above described On-Off feedback control system is that real life systems (ex. temperature control) do have some degree of inertia on the output and/or a delay in the feedback loop, which causes the system to fail in following the set point. As a result, the controller may switch On and Off around the set point in an oscillatory fashion. This oscillating behaviour may cause excessive power consumption by the actuator (like a relay, motor drive, etc.).

2.2.3.2 Differential Gap (hysteretic) On-Off Control

An improved representation of on-off control, which addresses the oscillatory and power consumption shortcomings of the above described On-Off controller, is the one that includes a differential gap. This means having two switching points around the desired set point: one for going from low to high state and vice-versa. The output in this case is driven to lie inside this interval. The switching behaviour is thus path dependent assuming a form of a hysteresis loop. That can be best

viewed in the Figure 24 below:

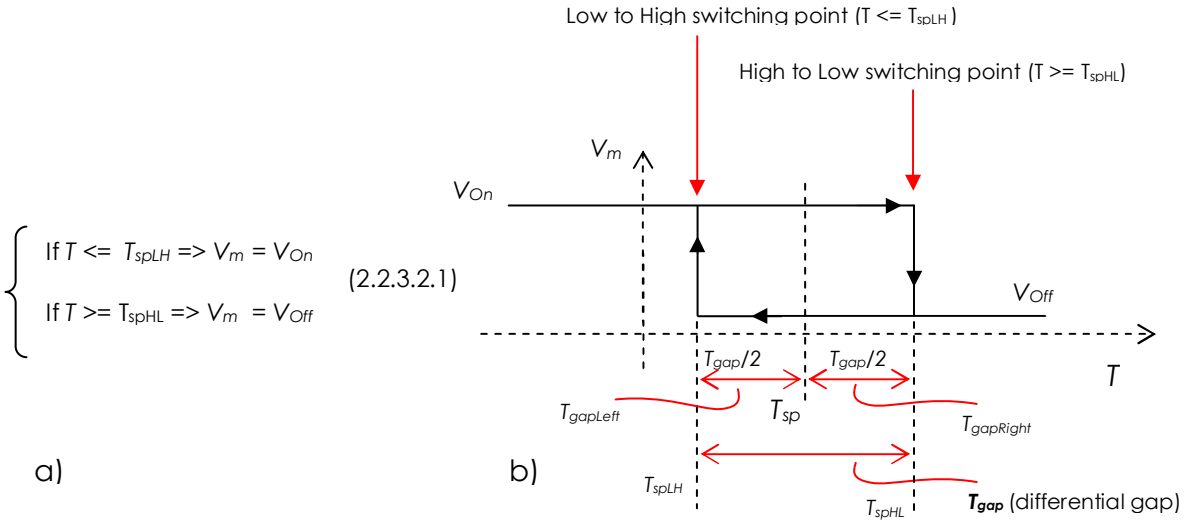


Figure 24 – Working principles of a Hysteretic (differential gap) on-off controller

In Figure 24, T_{sp} is the desired set point, T_{spLH} is the lower set point (the lower limit of the differential gap), while T_{spHL} is the upper set point (the upper limit of the differential gap), T_{gap} is the differential gap, and V_m the manipulated variable whose switching values are V_{on} and V_{off}. On the other hand, in the figure, T_{sp} it is at the centre of T_{gap}, however, this is not compulsory.

When T travels from low to high (while $V_m = V_{On}$) the value of the manipulated variable V_m only will change to V_{Off} when T reaches T_{spHL} and then V_m will remain V_{Off} until T decreases to a value not less than T_{spLH} in which case than V_m switches to V_{On}. Due to this differential gap (or hysteresis), the controller assumes a hysteretic behaviour thus avoiding oscillation and allowing for less power consumption on the actuators.

It should be pointed out however that, this system is not of great accuracy and can only be suitable for situations where a considerable error between the plant output and the set point can be tolerated, otherwise the use of the On/Off control strategy is not recommended. Figure 25 shows the block diagram representation of an hysteretic On/Off controller.

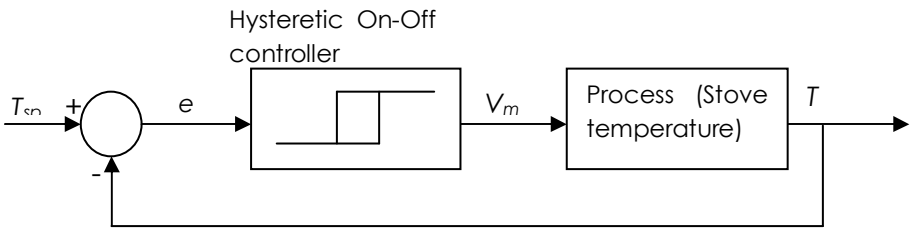


Figure 25 - Block diagram of a differential gap on-off feedback control system

2.2.3.3 PID Feedback Control System

A good example of a feedback control system is that using a PID controller, which is the most widely used closed loop control system in industrial process control, with a share of about 95%, according to Astrom K. J. and Hagglund T. H as cited by Jinghua Zhong (Zhong, 2006).

The PID controller is composed of three terms, which can be distinguished in the following expression:

$$V_m = K_p e + K_i \int e dt + K_d \frac{de}{dt} \qquad (2.2.3.3.1)$$

That may also be represented as:

$$V_m = K_p \left(e + \frac{K_i}{K_p} \int e dt + \frac{K_d}{K_p} \frac{de}{dt} \right) \qquad (2.2.3.3.2)$$

or:

$$V_m = K_p \left(e + \frac{1}{T_i} \int e dt + T_d \frac{de}{dt} \right) \qquad (2.2.3.3.3)$$

where K_p, K_i and K_d, are the proportional, the integral and the derivative gains respectively. T_i and T_d, are the integral and derivative times, respectively. The variable **e** is the error, on which the PID controller action is calculated, which will be eventually applied to the system so as to drive it into the desired behaviour. Its block diagram is as follows:

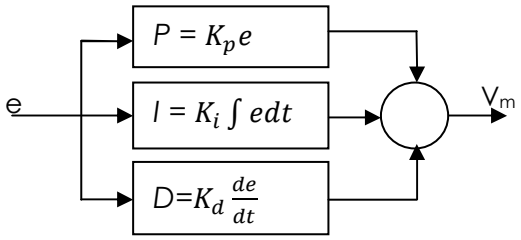


Figure 26 - Block diagram of a PID controller

Following is the Laplace transform of the above PID controller:

$$V_m(s) = \frac{(K_i + K_p s + K_d s^2)}{s} \qquad (2.2.3.3.4)$$

or otherwise:

$$V_m(s) = K_p \left(1 + \frac{1}{T_i s} + T_d s \right) \qquad (2.2. .3.3.5)$$

A PID feedback control system is the one that uses a PID controller, whose block

diagram is as follows:

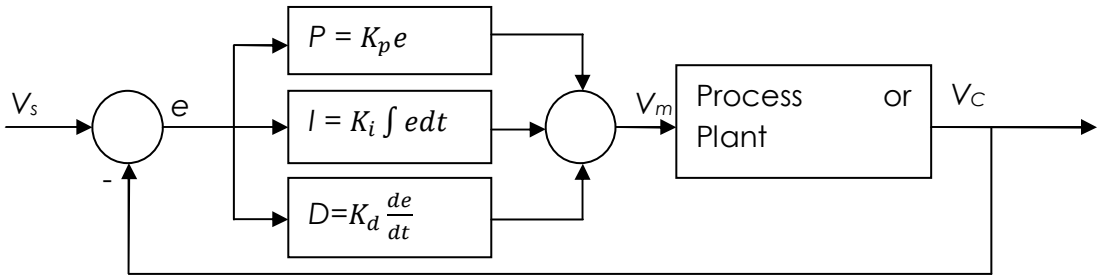


Figure 27 - Block diagram of a PID feedback control system

where V_s is the set point variable (the desired value), V_m is the manipulated variable; V_c is the actual process output value (as a result of the manipulated variables action over the process plus any possible disturbances), **e** is the error (difference between V_s and V_c). The manipulated variable V_m is the PID controller output and represents the controller's action to null the error **e**.

It is worth noting that it is not always necessary that a control system will use all these terms to achieve the desired performance characteristics. Some will need only one of the three controllers most likely the proportional or the integral controllers only, while some others may need a combination of the two, most likely PI or PD.

The individual effects of each one of the PID components on the control actions are the following:

- P – The proportional term outputs a proportion of the error, thus giving a reaction to the current error to the control the system. However, the proportional component introduces an offset error into the system.
- I – The integral component provides a reset action against the offset error created by the proportional component. It performs so by integrating (summing) all the 'historical' errors . But as a result, it may lead to what is called integral windup, where the integral term, keeps increasing indefinitely.
- D – the derivative term reduces the rate of change of the error **e**. This may lead to noise amplification producing system instability.

2.2.3.3.1 PID Tuning

The manner in which the individual PID terms affect the overall manipulated variable, depends on the respective gain. The gains are tuning parameters, whose optimal values are the ones that guaranties short rise time, null or small overshoot, fast settling time and small (ideally null) steady state error. Tuning gains may be determined as follows:

a) Analytically, if a mathematical model exists. Additionally, a computer aided calculation may be used; well addressed in the literature, as in (Ogata, 2002).
b) Adjusting the system behaviour manually in real time (done by an expert such as an experienced technician);

c) With the help of many well established methods, well considered in various textbooks, like the reaction curve Ziegler-Nichols method, well addressed by various authors like Leigh (Leigh, 1988), (Ogata, 2002), etc.
d) Through self tuning methods, by using computer based methods embedded inside the controller implementation, like the evolutionary concepts (genetic algorithms) used by Jin-Sung Kim et al. (Kim, et al., 2008).

2.2.3.3.1.1 Ziegler-Nichols reaction curve method

The reaction curve method (known as the First Method) for determining the optimal P, PI and PID controller gains, is only applicable to processes with neither integrators nor dominant complex-conjugate poles (Ogata, 2002). This method consists of obtaining the delay time *L* and the time constant *T* from the plant's open loop unit step transient response, whose graphical representation resembles the S-shaped curve of Figure 28. The PID gains are then found by computing the relations from Table 2.2 below.

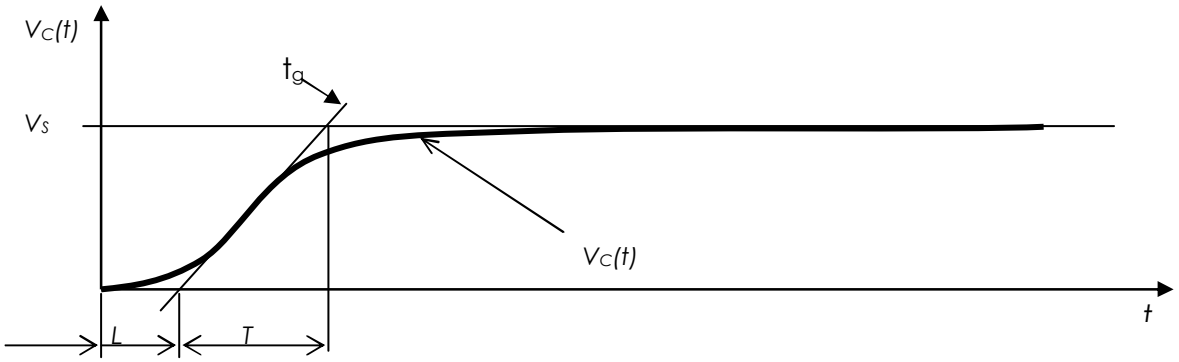


Figure 28 - Reaction curve (plant's open loop transient response to unit step input)

In Figure 28, $V_c(t)$ is the measured process output (the S-shaped curve); t_g is a tangent at the S-curve inflection point; *L* is the delay time and *T* the time constant.

Resulting gains	K_p	T_i	T_d
P	*T*/*L*	∞	0
PI	0.9**T*/*L*	*L*/0.3	0
PID	1.2 * *T*/*L*	2*L*	*L*/2

Table 2.1 – Ziegler – Nichols reaction curve coefficients (Ogata, 2002).

performance specifications. Such specifications are normally application-specific. They will, in turn, tell how far to take and what to focus in the fine tuning.

Fine tuning is performed by further adjusting each one of *Kp*, *Ki*, and *Kd* until a desired overall response is obtained. When performing the fine tuning, one should take into account that the increase of a given gain, aiming at improving a certain characteristic may lead to worsening of other characteristics, namely:

Kp – when increased: decreases the rise time, however, this may worsen offset (steady-state) error.

Ki – when increased: decreases the steady-state error but may lead to integral windup, where the integral term (summing of the errors), keeps increasing indefinitely.

Kd – when increased: it reduces the overshoot and settling time, though it may perform amplification of noise induced into the feedback loop, resulting in system instability.

The best fine tuning process, however, is that performed by a computer program, either performed outside the control system or built in as a self tuning mechanism.

Figure 29 shows a closed loop unit step response curve (Matlab generated) of a hypothetical process, whose open loop transfer function is

$$G(s) = \frac{1}{s^3+6s^2+5s}$$

and whose closed loop (with PID controller) transfer function is

$$G_c(s) = \frac{6.3223s^2+18s+12.811}{s^4+6s^3+11.3223s^2+18s+12.811}$$

and PID gains are *Kp*=18, *Ki*=18/1.405, *Kd*=18x0.35124. The PID gains (and thereby the closed loop transfer function) were found through the second Ziegler-Nichols tuning method (Ogata, 2002). It can be observed that, some of the performance characteristics, namely the overshoot (ratio is more than 50%) will need improvement, by further fine tuning steps, as suggested above.

Figure 29 - Closed loop step response of a system

2.2.3.4 Digital Control Systems

A digital controller is a system in the realm of a discrete time, discrete data (inputs and outputs) subspace of the generic continuous state space. In simple terms, its inputs and outputs are quantized discrete time data. So, to interface a digital controller to a process (see Figure 30), it will need analogue to digital (A/D) and

digital to analogue (D/A) converter. The conversion is a discretization process that takes normally two steps: sampling (which is a linear operation) and quantification (which is non-linear). The sampling process is governed by the sampling theorem, according to which the sampling frequency should be at least twice the highest frequency of the variable being sampled. In practice, the sampling is commonly done by a device called sample and hold (S/H) that has a sampler and normally a zero-order-hold (z.o.h) device. From samples obtained through a z.o.h device and with a frequency governed by the sampling theorem, the original signal can be reconstructed to an acceptable approximation. In turn, the quantification process is a non-linear discretization of the sampled data, where the higher the quantization resolution, the smaller the quantization error. The quantization process should take less than the sampling period (the inverse of the sampling frequency as determined by the sampling theorem).

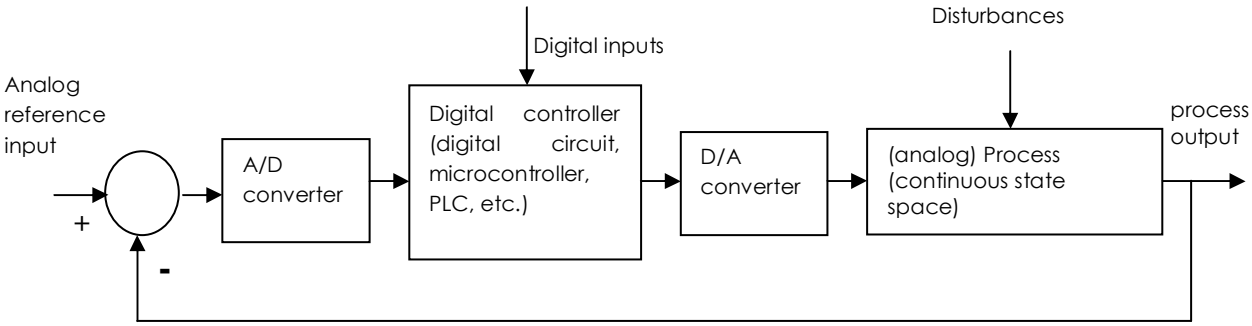


Figure 30 - Conceptual diagram of a digital controller based system

If the sampling theorem is not fulfilled, then an error (frequency aliasing) is introduced over the higher frequencies of sampled data. In practice as high frequency noise is normally present in any analogue input, then, whenever possible, anti-aliasing analogue filters should be placed between the sampling devices and the sampled variables. Further digital filtering may be done at software level processing of the signal, inside the computer based controller.

It is worth noting that, in order to fulfil the requirements of the sampling theorem and other concepts addressed above, A/Ds or D/As are not the only devices to take into account. In a computer based digital data acquisition and control system, many devices are tailored together, each one playing a role in the overall timing and synchronization. For example in a microcontroller based system, where the A/D has multiplexed inputs, the sampling frequency will be affected by the number of multiplexer channels. In turn, the microprocessor ability to cope with the sampling frequencies is also a constraint to take into account. A broad treatment of digital control systems, where the above concepts and many other are well addressed, can be found in (Kuo, 1980).

With the aid of (but not to limit to) the diagram of Figure 30, it should be pointed out that, the control strategy housed inside the digital controller, may be of various kinds: one of the PID family of controllers in digital form, an On-Off controller, a fuzzy logic controller (FLC), a finite state machine based controller, a neural network based controller, etc. or even a combination of the above, like

neuro-fuzzy, fuzzy-PID, fuzzy-state-machine, etc.; not excluding feedforward, adaptive and other types of techniques mentioned (or not) before.

With software/firmware based implementations, there is the versatility of controller strategy update as well as self tuning and / or learning capabilities that may be embedded into neuro, fuzzy and state machine implementations. It is this versatility that makes digital systems based control techniques the evolving trend of control system today.

2.2.3.5 Fuzzy Logic Control

As mentioned before, the traditional control concepts, are applicable mainly to linear time-invariant systems. Although the traditional (continuous state space) feedback control systems can perform well with some level of disturbances and/or some degree of change in the internal system parameters, that capability is limited. To cope with non-linearity, uncertainty and varying or ill-defined processes, the best choice are computer based control systems (ranging from simple algorithms to artificial intelligence techniques), like the Fuzzy logic controller (FLC) technique (Verbruggen, et al., 1997) (Yan, et al., 1994).

In fuzzy logic based control design, the complex mathematical process models are not a compulsory requirement, thus providing a non-analytic control design alternative to the classical analytic control design methodology. Fuzzy control design is rather based on an expert knowledge and representation of process behaviour, what is conveniently related to fuzzy representation and interpretation of crisp inputs; and the production (inference) of the final crisp outputs (control actions).

A fuzzy logic controller is based on the theory of fuzzy sets first proposed and developed by Zadeh (Zadeh, 1965). A fuzzy logic controller is a rule based, linguistic and human-like approach. It consists basically of the following parts and concepts, as depicted in Figure 31 and well covered by Yan, Ryan and Power (Yan, et al., 1994) :

1. A fuzzification unit, which converts the crisp inputs to their fuzzy equivalents, to be later used by the fuzzy reasoning (called also the inference engine). The fuzzification process is done trough the concept of membership functions;
2. A knowledge base, which is a collection of control rules (rule base) and data base, that are manipulated and related to the fuzzy input data by the fuzzy inference engine. The rules are generally expressed in the form of "IF-THEN" statements, which is appropriate to express expert knowledge and is also appropriate to computer based implementation;
3. A fuzzy inference engine (also called the fuzzy reasoning mechanism), that infers (makes decisions of) control actions from the

fuzzy inputs using the rule and data base;

4. A deffuzification unit, which converts the inferred fuzzy actions into the corresponding crisp control actions.

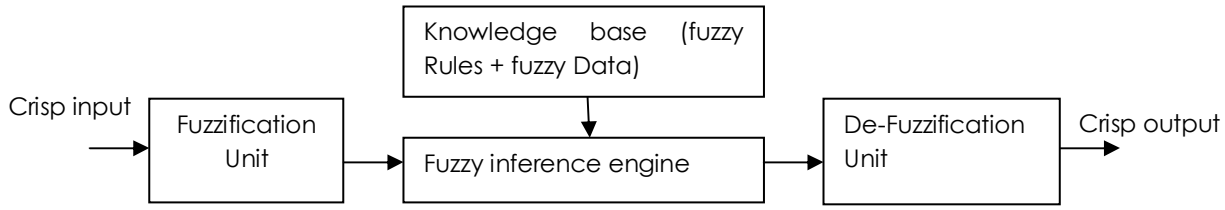


Figure 31 - Conceptual diagram of a Fuzzy Logic Controller (FLC)

FLCs versatility make them appropriate for application in a wide range of control implementations, ranging from the most basic controller to supervisory, as addressed by Jantzen (Jantzen, 1998). It can be frequently found in combined implementation with other control techniques, as with Alata (Alata, et al., 2005) or Linkens (Linkens, et al., 1998).

2.2.3.6 Finite State Machine Based Control

Finite state machines (FSMs) are an old control technique, widely used in the field of digital systems and computers. Indeed, finite state machines (FSMs) are the bricks that build digital systems (from a single flip/flop, a simple counter, to a complex CPU sequencer and controller).

Basic approaches of finite state machines (FSMs) or automata are well considered in many digital sequential systems design textbooks. In the stand point of control systems, they may be regarded as a finite subspace of the discrete time state space, where a FSM is a finite number of discrete states, a finite number of discrete inputs and outputs, a set of initial states and a finite number of interstate transitions, governed by conditions or guards. A transition condition is a logical function of discrete time, the external inputs and the current internal state.

Besides those basic definitions, there are additional specifications and/or constraints that further particularize FSMs into specific types such as deterministic FSMs, non-deterministic FSMs, stochastic FSMs, etc. Each specific type may be appropriate for application in a specific field. For instance, many real life event driven control problems, will require the use of a deterministic finite state machine.

An adequate mathematical and control systems treatment of finite state machines as elements of the discrete state space, as opposed by elements of a continuous state space, is considered by Mensah (Mensah, 2008). Here a hybrid controller is built by tailoring together a FSM based discrete space control subsystem with a continuous state space control subsystem. In this way, he could successfully address the problem of optimal control of systems with both

continuous dynamics and discrete events (what is indeed, the nature of many real processes).

In the stand point of control system design, FSMs control design may be regarded as similar to fuzzy logic controller design, in that no system mathematical model is mandatory for the design of a controller for a given process. In the absence of a mathematical model, the process under consideration may be treated as a black box. In such a case, the set of its relevant inputs and outputs, are then the required prerequisites for representing the system and perform the controller design.

In the realm of control systems, FSM control design is the most effective for controlling discrete event systems, where either timed (periodic) events or asynchronous (aperiodic or sporadic) events occur. Moreover, as with FLCs (as well as neural and evolutionary techniques) it can successfully cope with process non-linearity, uncertainty and other unpredictable system changes. Figure 32 below, shows a state flow diagram representation of an On-Off finite state automaton.

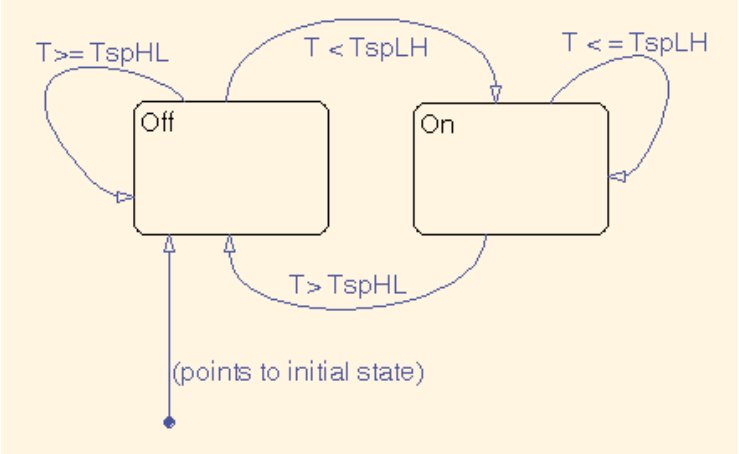


Figure 32 - FSM model (in flow diagram form) for an On-off controller

2.2.3.6.1 FSMs applications

While In the past FSMs were only found in the realm of digital sequential systems, today they have been deserving a wider and wider role in different fields like:

- artificial intelligence applications (pattern recognition, game programming, etc.),
- basic and advanced control systems design, including taking a role on a supervisory control framework that change and adapt the behaviour of conventional controllers. This may also combine together various control devices and techniques to address the control of complex, heterogeneous systems (or systems with ever changing or non-linear behaviours) and cope with uncertainty and unpredicted disturbances .

As a good example of the direct application of state machines in control design, Pakanen and Karjlainen used state machines to model and control the heating

process of a building (Pakanen, et al., 2009). While, (Osornio-Rios, et al., 2008) used a FSM to perform the control of a digital PID on a FPGA based CNC milling machine controller.

The Matlab Stateflow is another good point of reference of FSMs application. This is a computer based design tool that helps develop state-machines based control. It provides "an efficient environment for designing embedded systems that contain control, supervisory, and mode logic" (Mathworks, 2009).

A state machine can be implemented in bare (fixed) logic, in configurable logic (example FPGAs), in computer based hardware and/or software (examples: a conventional PC, an industrial computer, a PLC, a microcontroller, etc.).

2.2.3.7 Programmable Logic Controllers (PLCs)

A PLC is a digital controller, a type of (but not a general purpose) computer, tailored and customized for control applications. Despite its limited resources as compared to general purpose computers, it can be programmed to perform all the basic control paradigms discussed above, including some advanced techniques up to a level allowed by these resources. Before the microcontrollers boom, PLCs where the most widely used digital computer based controllers, in the realm of industrial applications. Although PLCS are rather expensive compared to microcontroller based implementations, PLCs are, indeed, the fastest and easiest digital control implementation. This may be one of the reasons of sometimes being preferred over MCUs (that are cheaper) as with (Abdallah, et al., 2004) in their sun tracker implementation.

Chapter III – Model of the pre-existing solar thermal energy system at UKZN

In this chapter we describe the thermal energy system pre-existing at UKZN. We discuss its basic working principles and safe operation. This will be a means of determining the monitoring and control requirements for the data acquisition subsystem that will be designed and described in the following chapters.

3.1 Models of solar energy systems

Solar energy systems models may be of many different types. They have however a common start point and objective: use of solar radiation as source of energy, convert it normally to electricity or thermal energy. Whichever the case, generically any system is composed of three parts: (i) an energy capture, (ii) energy storage and (iii) energy utilization. The storage however may be suppressed where the captured energy goes straight to utilization for simplicity and low cost.

The solar based renewable energy systems are divided into two major families: PV based and thermal systems. The prototype solar energy system on the roof of UKZN/Physics - henceforth also referred to as ST-KZN[9] - is an example of a thermal type solar energy technology, which we describe in the following sections.

3.2 Model of the pre-existing solar thermal energy system at UKZN

The solar thermal energy system at the University of KwaZulu-Natal, is currently being redesigned and rebuilt. An identical one is also being built at the University Eduardo Mondlane (UEM), Maputo, Mozambique. We now perform a short description of the system, whose conceptual diagram is shown in Figure 33.

This system is composed of:

1. **Energy capture subsystem**: paraboloidal half dish tracking concentrator with heat receiver (see also Figure 34);
2. **Energy storage subsystem**: a rock bed thermal energy storage (TES);
3. **Energy utilization subsystem**: A simple pot with heat exchanger (could be any user heat utilization system, like water warming, room heating, etc.);
4. **Data acquisition and control subsystem**.

Figure 33, shows a conceptual model of the above system. See also in appendix E photographs of some relevant system components.

[9] ST-KZN – Abbreviated name for the system under consideration. See also the specific list of acronyms and abbreviations, on page xxii.

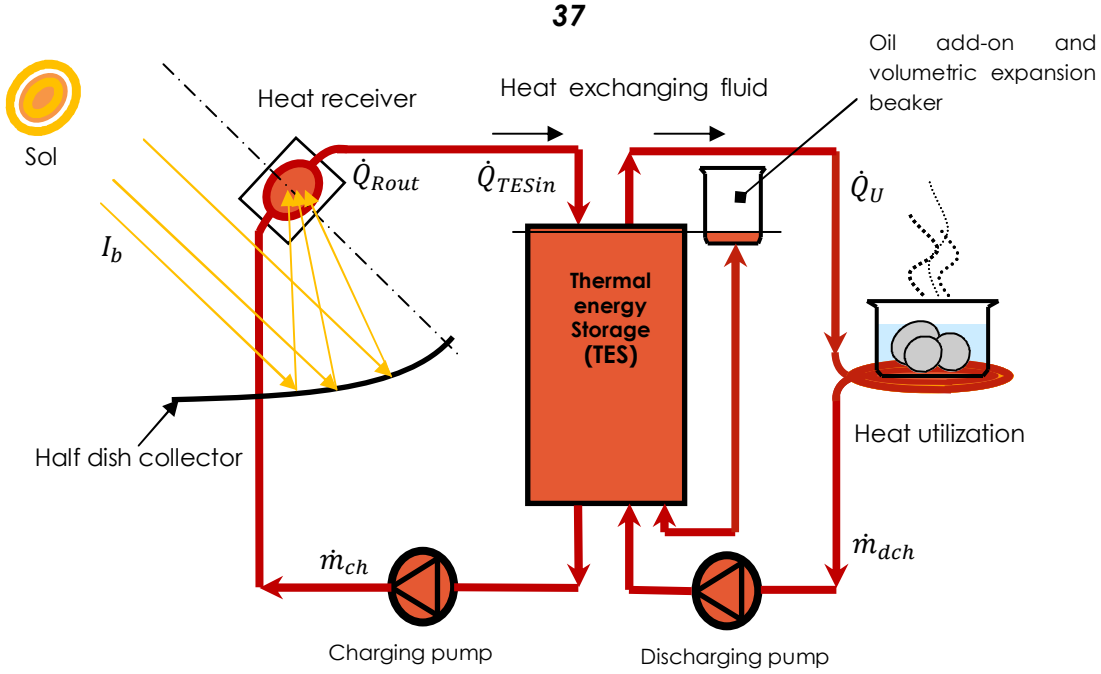


Figure 33 - Conceptual model of the solar thermal energy system pre-existing at UKZN

3.2.1 System description

The plant under consideration is a small solar thermal system, intended for home use, such as water warming and food preparation or other home scale applications in rural communities. This was a project aimed at improving the UKZN former air based system addressed by (Heetkamp, 2002). Following is a summary description of the system components, which will give a basic understanding of the overall working principles of the system and help in understanding the system behaviour. This would assist in the design of a MCU based monitoring and control system.

3.2.1.1 The energy capture subsystem

This is composed of the following subcomponents:

1. A paraboloidal half dish concentrating collector with an aperture diameter of about 2.4m. The reflecting surface (built by another team member) is composed of trapezoidal tiles. Figure 34 shows two photographs of the concentrator.
2. A heat receiver/exchanger: It is a spiral coiled steel pipe of 10mm external diameter (and 8mm internal), forming a 20cm circumference of receiving surface. The receiver's steel pipe is part of the overall circuit followed by the heat exchanging fluid in the charging cycle, according to what is depicted in the Figure 33.
3. A supporting and tracking assembly, which is a dual axis polar mount tracking system, compatible with the one described in the section 2.1.3.3.1.4 "Polar (equatorial) tracking". One DC motor fitted on each axis, being the actuators that provide the angular movement.

It should be mentioned that the paraboloidal half dish and the receiver were not actually pre-existing. They were built, in parallel with the present work by another team member.

Figure 34 - Photos showing 2 details of the colector and tracking assembly at roof Physics, UKZN

3.2.1.2 The thermal energy storage (TES) subsystem

It is made of a cylindrical steel can (of approximately 20 litre), filled with pebble and the heat exchanging fluid. The TES inlet is at the top and the outlet at the bottom. At the bottom there is also an auxiliary oil inlet/outlet to a transparent and graduated small (~2.5l) beaker for adding oil to the storage or for volumetric expansion compensation that may arise from the heat exchanging process. As with the receiver, the TES is part of the heat exchanging fluid charging circuit, according to what is represented in the Figure 33.

3.2.1.3 Energy utilization subsystem.

The heat utilization is a spiral coiled copper pipe that allows a normal sized pot to be placed over it (this arrangement can however be replaced by any other suitable home application like water heating). The heat transfer fluid (the same as of the charging circuit) circulates between the TES and this pot, forced by the discharging pump at a speed that must be governed by heat transfer laws in accordance to the cooking process needs.

3.2.1.4 Fluid / heat transport

Following is a description of the parts that perform the fluid and heat transport among the different subsystems. The fluid/heat transport system is composed of:

- A heat transfer fluid: Calfo AF (further addressed below),
- The fluid charging circuit (left loop in Figure 33). This links the receiver to the TES,
- The fluid discharging circuit (right loop in Figure 33). This links the TES to the

heat utilization subsystem,

- A charging circuit pump, fitted between the TES outlet and the receiver's inlet. This is the most suitable position, considering that, it lies on the cold pipe of the charging circuit (this remains true until pump's critical temperature is reached in the charging process), and
- A discharging circuit pump, fitted between the TES outlet and the receiver's inlet. It lies also on the colder side of the discharging circuit.

These pumps are cheap and of low operating temperature. No datasheet or any other information about their operating settings were found. They are presumably of 100ºC/150ºC operating temperature. Each pump is driven by a DC motor aimed at producing flow rates at such levels required by control laws, according to the particular heat transfer needs (at charging and/or discharging processes). On the other hand, it is worth noting that the pumps and their motor drives are old and needing replacement.

3.2.1.5 The Heat Transfer Fluid

It is important to describe briefly the characteristics of the heat transfer fluid to understand the manner in which it can affect the TES charging and discharging process or the overall system dynamics.

The fluid used, Calflo AF oil, has the summary properties listed below. Details can also be found in its datasheet in appendix D, and (Petro-Canada, 2006). The datasheet includes plots that relate temperature to heat capacity, density, thermal conductivity and viscosity. The main properties of the Calflo AF heat transfer fluid are:

i. Maximum operating temperature of 316ºC: not that good for the intended applications (ex: baking is at about 250ºC);
ii. Thermal conductivity of 0.142 to 0.127 W/(m K): low and variable;
iii. Specific heat capacity of 1.89 to 2.88 KJ/(Kg K): variable;
iv. Viscosity of 32.1 to 0.7 cSt: high at low temperatures and variable;
v. Density of 0.88 to 0.68 Kg/L: variable.

On the other hand, as can be seen (from the plots), density, heat capacity and thermal conductivity vary linearly with temperature over the operating range, while viscosity has a non linear variation, being high at low temperatures and vice-versa.

This variability of fluid characteristics introduces additional complexity to the system dynamic model. In addition, some of the properties (or their conjunction) may compromise system performance in a moderate to severe fashion. In effect, as also addressed by Løvseth (Løvseth, 2008), for efficient heat transfer with this oil, a turbulent flow (implies Reynolds number $R_e > 4000$) is mandatory. It is difficult to achieve a turbulent flow below 250ºC, due to the Calflo AF's high viscosity at low temperatures (because Re is in inverse proportion to the viscosity).

The need for achieving such turbulent flows will require high pumping power at low

temperatures, to counteract the high viscosity and high density that is characteristic of the oil at low temperatures. In this way, the already built pumping system may or (most probably) may not be able to perform up to such requirements. This may prevent the control system from performing the expected work to a satisfactory level.

Also, the maximum temperature of 316ºC, is not only a performance constraint but also a safety constraint as addressed below (section 3.2.1.7).

3.2.1.6 Data acquisition and control subsystem

The system that was developed previously by Mawire (Mawire, A., 2005) and also by Robert van den Heetkamp (2002) included a data acquisition and control subsystem composed of:

a) 2 HP/Agilent 34970A data loggers, that performed data readings of relevant plant's outputs and DAC generated actuating outputs to the plant;
b) 2 Desktop Windows PCs running data logger interfacing software, both for the data input and control data output.
    - An electrical hot plate simulating a solar heat collector.

All these components along with the heat storage and the dish collector, formed part of a complete system in which heat was supplied by either the dish collector (Heetkamp, 2002) or the electrical hot plate (Mawire, 2005).

However, the system developed by Heetkamp and Mawire was mostly dismantled. In particular, the data acquisition and control system was completely dismantled and attempts to rebuild it were fruitless.

The present work therefore, attempts rebuilding the data acquisition and control system, using an embedded microcontroller, together with a newly built half dish solar collector and the existing rock bed with its thermocouple sensors.

3.2.1.7 Safety considerations

Some precautions have to be taken to insure that the system is operated inside safety conditions. The intention here is to underline the temperature related safety considerations, where the control system can play a role. Each one of the system components described above, can only withstand a limited value of temperature:

(a) The heat transfer fluid should not be heated above 316ºC or its properties cannot be guaranteed thereafter. It is also important to observe that besides losing the normal operating properties there is also a risk of auto ignition (from 343ºC).
(b) The charging pump: It is located at the cold side of the charging circuit. Nevertheless, it is important to insure that the fluid outlet temperature from the TES is not allowed to exceed the pump's maximum operating

temperature (about 100/150ºC), otherwise the pump may be damaged. This has the negative implication that the TES cannot be fully charged to temperatures above that mentioned above.

(c) Also the discharging outlet temperature from the utilization system should not be allowed to exceed the discharging pump's maximum operating temperature.

(d) The receiver's steel pipes: To insure that the melting temperature of steel (about 1370 °C) will not be attained, a stagnation (and higher) temperature should not be allowed to develop in the receiver. Such levels of temperatures can develop at the receiver if it is focused to the sun and the fluid flow is left null (pumping speed is zero).

(e) The receiver's surface should not be allowed to exceed the maximum operating temperature of the absorbing material (about 300^0C);

(f) The conducting pipes: There are flexible conducting pipes at the inlet and outlet of the receiver, made of ptfe (Teflon), with a maximum safe operating temperature of about 260ºC. So, it is important to insure that the fluid temperature is maintained below 260ºC.

Other oil conducting pipes are made of either steel or copper (about 1085°C melting point). So, if the previous and most of the other constraints are met, these steel and copper pipes are safe.

All these safety operating conditions, and others not mentioned, must be taken into consideration for a safe operation.

It is essential that the monitoring and control system be given that knowledge base and made capable of performing the required preventive actions or taking corrective actions whenever a critical condition is detected.

3.2.2 Summary of physical working principles of the thermal energy system

A good understanding of the system working principles and the relevant underlying physical laws is the basis of controller design and implementation. So, in this section, we perform a brief description of the basic working principles of the system.

Solar radiation strikes the paraboloidal half dish collector. The direct component (beam) is concentrated towards the receiver surface where it forms a spot, supplying an energy flux of $\dot{Q}_c$ which generates heat. This is then transmitted to the heat transfer fluid (as $\dot{Q}_{Rout}$) and partially lost to the environment. The heat transfer fluid eventually takes the heat to the TES where it is absorbed and stored. The law that relates $\dot{Q}_c$ to the beam radiation I_b is:

$$\dot{Q}_c = \eta_C \cdot A_{eff} \cdot I_b \qquad (3.1)$$

where η_C is the collector efficiency, which accounts for collector losses due to a number of imperfections, namely, geometric, alignment, optical properties and

disturbances (like poor cleanness), etc. A_{eff} is the effective surface area of the collector aperture, taking into account the following subtractive factors: the gaps between the trapezoidal mirrors and collector's aperture shading by the supporting and the receiver assemblies.

On the other hand, as mentioned earlier, $\dot{Q}_c$ equals the energy flux effectively absorbed by the receiver $\dot{Q}_{Rout}$ plus the receiver's heat losses $\dot{Q}_{Rloss}$ to the environment, as expressed by:

$$\dot{Q}_c = \dot{Q}_{Rout} + \dot{Q}_{Rloss} \tag{3.2}$$

$\dot{Q}_{Rout}$, the effective energy output from the receiver, which is also the energy flux passed to the oil, can be inferred from the following law:

$$\dot{Q}_{Rout} = \dot{m}_{ch} \int_{T_{Rin}}^{T_{Rout}} c_f(T) dT \tag{3.3}$$

Where $\dot{m}_{ch}$ is the oil's charging circuit mass flow rate (see also eq. 3.10 below), $c_f(T)$ is the oil's temperature dependant specific heat capacity, T_{Rin} and T_{Rout} are the oil's inlet and outlet temperatures of the receiver, respectively. There are many factors that determine the magnitude of $\dot{Q}_{Rloss}$, namely, the characteristics of the absorbing material, the geometry and construction of the absorber (which includes insulation), atmospheric and environmental conditions (which includes outside temperature and wind), as well as radiactive losses (which increase with temperature).

The effective energy Q_{Rout} gained by the oil is further depleted by losses between the receiver's outlet and the storage inlet:

$$\dot{Q}_{TESin} = \dot{Q}_{Rout} - \dot{Q}_{RSloss} \tag{3.4}$$

where $\dot{Q}_{TESin}$ is the TES inlet energy flux and $\dot{Q}_{RSloss}$ are the mentioned losses between the receiver's outlet and the storage inlet.

In turn, the heat storage has its own energy losses ($Q_{TESloss}$), so the effective energy Q_{TESeff} stored in a specific period of time, may be expressed as:

$$Q_{TESeff} = \eta_{TES} \cdot Q_{TESin} \tag{3.5}$$

where η_{TES} is the TES efficiency, which can be expressed as:

$$\eta_{TES} = \frac{Q_{TESin} - Q_{TESloss}}{Q_{TESin}} \tag{3.6}$$

In turn, the maximum energy that can be stored in a pebble bed TES is a function of its physical and geometrical properties (height, cross section, pebbles size, pebbles heat capacity, pebbles porosity, etc).

The theoretical maximum energy Q_{TESmax} that can be absorbed by the TES by raising its temperature from an initial temperature of T_{TESo} to a final temperature

that equals the inlet temperature T_{TESin}, can be expressed as:

$$Q_{TESmax} = c_p.m_p.(T_{TESin} - T_{TES0}) \quad (3.7)$$

Or:

$$Q_{TESmax} = c_p.\rho_p.V_{TES}(1 - \varepsilon_p)(T_{TESin} - T_{TES0}) \quad (3.8)$$

Where c_p is the specific heat capacity of rock bed material, m_p its total effective mass, ρ_p its density, V_{TES} the total TES volume and ε_p is the pebbles void fraction which accounts for the porosity between the pebbles.

The above energy is gained as a result of heat exchange with the oil that traverses the TES, whose energy flux $\dot{Q}_f$ is:

$$\dot{Q}_f = \rho_f(T).\dot{V}_{ch}.c_f(T).(T_{TESin} - T_{TES0}) \quad (3.9)$$

If $Q_{TESloss}$ can be neglected, that is, the efficiency approaches 100%, then, the integral of $\dot{Q}_f$ over the charging period (the time that took the TES temperature to be raised from T_{TESo} to T_{TESin}) equals Q_{TESin} and Q_{TESmax}.

For a more detailed study of the heat storage dynamics, taking into account the above mentioned TES constructive characteristics and the heat losses, other variables and models should be considered. The Schumann as well as Mumma and Marvin models, addressed by (Duffie, et al., 2006), may be a good starting point. Worth considering for the same purpose is the recent oil-pebble TES system simulation by (Mawire, et al., 2008).

The charging pump, with a pumping constant of approximately $\alpha = 6.12$ml per revolution, forces heat transfer oil to travel from the TES to the receiver and from this, back to the TES, with a mass flow rate $\dot{m}_{ch}$, which can be expressed as:

$$\dot{m}_{ch} = \rho_f(T).\dot{V}_{ch} = \rho_f(T).\alpha.\omega_{ch} \quad (3.10)$$

Where $\rho_f(T)$ is the density of Calflo oil as function of temperature, $\dot{V}_{ch}$ is the oil's charging circuit volumetric flow rate, and ω_{ch} is the rotating speed of the charging pump in revolution per unit time. From the previous equation, the pump speed can be derived:

$$\omega_{ch} = \frac{\dot{m}_{ch}}{\rho_f(T).\alpha} \quad (3.11)$$

This law can be used as set point for controlling the pump speed (for example to keep the mass flow rate constant, independently of the temperature).

In turn, the laws that govern the discharging process as compared to that of the charging are identical, although the source of heat is now the TES. Thus, for the discharging process, a relation can also be derived to control the discharging pump rotational speed as a means of controlling the mass or volumetric flow rates to fit the required values within time, according to the desired temperature profile

to be followed. This profile is defined by the specific user application (boiling water for tea, bake a bread, etc.). It is worth noting that the amount of heat stored in the TES will affect the controller's ability to follow the required temperature profile. The higher the temperature difference (TES – user load) and the higher the stored heat, then, the higher the performance of the utilization system. That is why the system performance will be affected by fluid's max temperature of 316ºC, as well as other constraints.

Further discussion of the system working will be done in the next sections, where the specific design of the controller system is addressed.

Chapter IV – Proposed model for microcontroller based monitoring and control.

In this chapter we present the layout of the data acquisition system: the set of inputs from the plant and the set of controller outputs to the plant. In a later point we address the controller design for the sun tracking system and for the charging and discharging processes. After choosing and designing the model of the controllers, implementation issues are finally discussed.

4.1 Problem definition

The main problem that is being addressed in this work can be described as:

- We have solar radiation reaching an absorber. Heat from the absorber is transferred to a fluid which conveys the heat to a rock bed. The fluid flow through the absorber must be maintained at a certain minimum speed or else the absorber will burn out;
- Simultaneously, heat must be extracted from the rock bed for utilization. The fluid flow through the utilization heat exchanger must such that the required heat/temperature is delivered to the utilization system;
- During sunlight hours, there is also the need of tracking the sun and insuring safe operation.

These processes require accurate monitoring and/or control of the absorber temperature, rock bed temperatures, utilization temperatures, solar dish aiming and many other system variables. Therefore, the model of a data acquisition and control system, for addressing the above problem, is discussed and developed in the following sections.

4.2 System layout. Inputs and outputs

For the purpose of performing data acquisition and control of this system, there is the need for monitoring the behaviour of the following system variables (see Figure 35 below for further visual details):

- The receiver's inlet and outlet temperatures (two input[10] variables);
- The receiver's surface temperatures (receiving side and the opposite side) (at least two input[10] variables);
- The thermal energy storage(TES) inlet and outlet temperatures (two input[10] variables);
- The TES nine level temperature profiles from the inlet side to the outlet end of the storage as well as in five different radial zones of the same level (9x5=45 input[10] variables. At least the 9 level temperatures);
- The charging pump (feedback) speed from which the heat transfer fluid charging circuit mass flow rate is inferred (one input[10] variable);

[10] Inputs in the stand point of the control system; actually plant's outputs.

- The discharging pump (feedback) speed from which the heat transfer fluid discharging circuit mass flow rate is inferred (one input[10] variable);
- The actual hour angle measured from the solar noon zero degrees reference (one input[10] variable);
- The actual declination angle which should be equal to the angle between the equator's plane and the sun beam for a particular day of the year (one input[10] variable);

The system described above also has plant's inputs, through which the controller will apply its control actions, namely:

- The tracker assembly positioning and speed control outputs[11], whose number and nature are defined according to the architecture and implementation of the tracker assembly motion driver in later sections;
- The charging pump speed control output[11], which is used to control the heat transfer fluid mass or volumetric flow rate and hence influence the behaviour of the charging heat transfer process;
- The discharging pump speed control output[11], which is used to control the heat transfer fluid volumetric or mass flow rate and hence influence the behaviour of the energy utilization heat transfer process;

These controller outputs may be further modified or broken down into other sets of variables according to the general MCU system and specific controller design and implementation needs.

In the same way, other controller outputs and inputs may be eventually added later as new control needs arise from the aim of reaching control objectives, system performance specification or from system safety needs.

Besides the input and output variables discussed above there are other relevant parameters and inputs that are necessary for full implementation of a control system. These are:

- The geographical coordinates (latitude and longitude) and time zone of the plant's location. They are used for deriving solar time and other relevant angles, such as sunrise and sunset angles.
- The local time, which can be generated by using internal MCU timers plus time update procedures and/or by using an external real time clock (RTC) with backup battery to insure timekeeping in the event of power failures. Local time as described earlier is used in the calculation of solar time.
- Finally, human interface, data logging interface (local and remote) and communications interface should be included. Single buttons or an entire keyboard may be the means to provide such human interface. In turn, communications ports - RS232 to PC, M2M wireless interfaces, may be the means to provide communications with other machines.

[11] Outputs for the controller, however they are inputs for the plant.

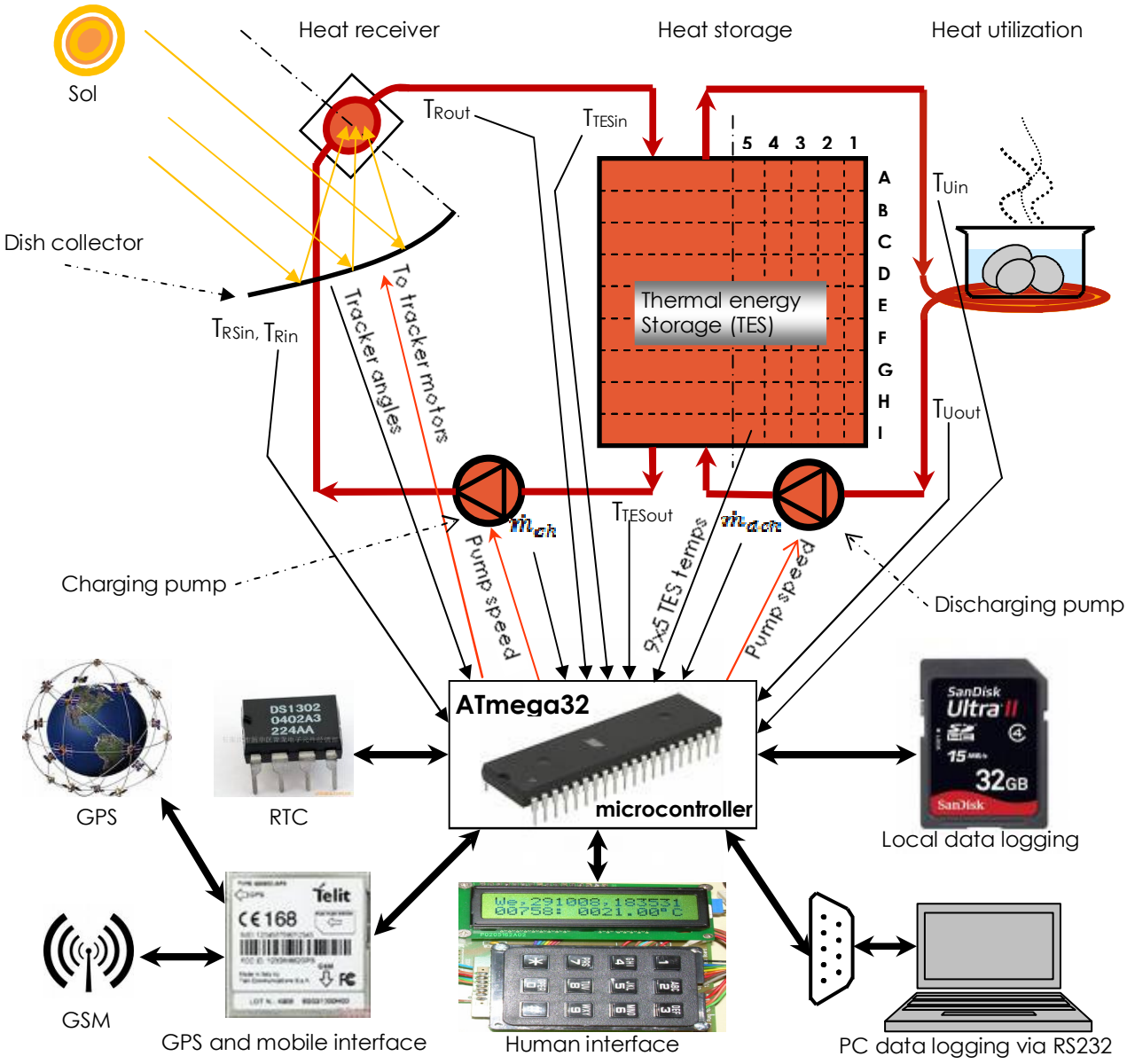


Figure 35 - Conceptual diagram of the MCU controlled Solar thermal energy system (Solar Tech), illustrating input and output control variables.

4.3 Control system design and implementation

The monitoring and control tasks have been defined above and well depicted in Figure 35. A further analysis in the control point of view leads to the conclusion that it is a very complex, multiple inputs and multiple outputs system with non linearities, time-variance and event driven behaviour. Further discussion on these characteristics will be done when designing a specific controller.

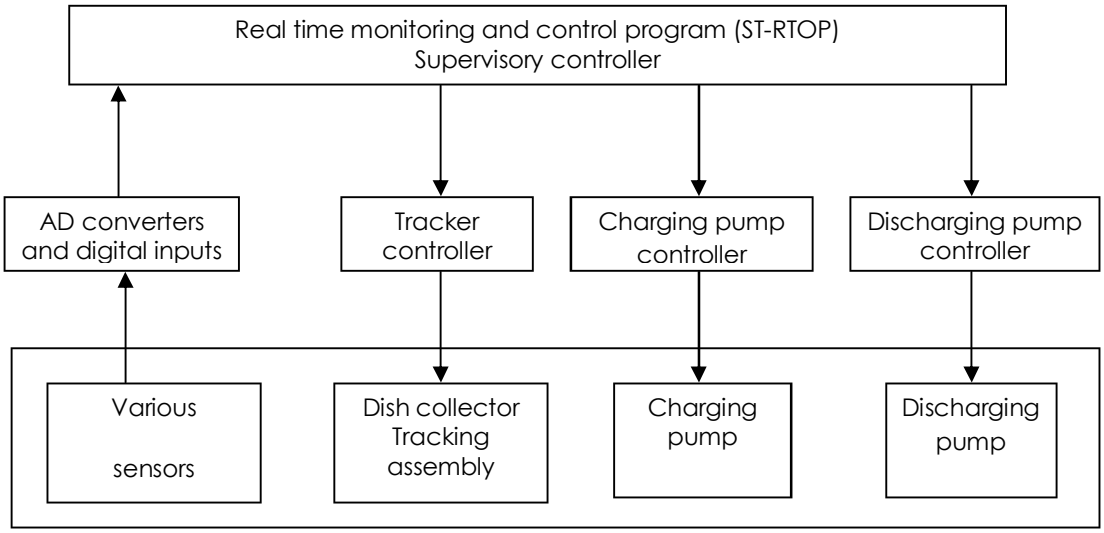


Figure 36 - ST-KZN Controller framework

The above considerations lead us to the following choice: to use a microcontroller based control system so as to address complexity at lower cost[12] (as mentioned earlier). Furthermore, we have chosen to address the specific controllers of Figure 36 above, as separate control problems (to a certain level) and use the real time control program as a supervisory controller that will integrate together the partial control solutions. A diagram describing the controller framework is depicted in Figure 36. The description of this controller framework in the software implementation stand point can be found on chapter VI. Also, further discussion on what are the forms of supervisory and how they are exercised, will follow later.

4.3.1 Tracker Controller

The controller for performing the sun tracking functions can be built in a number of ways, as addressed earlier. In many cases (and for the present case too) the choice of controller may be influenced by an existing tracking assembly. In our case, there is a solar dish collector with its motors installed, over a polar mount tracking configuration.

[12] The former UKZN system used 2 HP/Agilent data loggers and 2 PCs: Their cost together (hardware+software) is of thousands of dollars (more than $4000). The cost of the basic components used to build the prototype microcontroller based system (computed approximately from its bill of materials) is of about $1350. Although this does not account for intellectual properties and any other design costs, we are sure that after design optimization and also mass production, the end user distribution cost should be dramatically lower. As we can see, there is a cost advantage of the MCU system compared to using 2 PCs and 2 conventional data loggers. Cost is not the only advantage.

4.3.1.1 Model and parameter identification and controller choice

The tracking assembly is composed of two polar mount rotating axes (see section 2.1.3.3.1.4). The hour axis is driven by a 36V geared DC motor (Jaeger SMR-90U). The declination axis is driven by a 30VDC motor linear actuator (Jaeger HARL-3018). As a result we decided using 30VDC as the maximum supply for the motor driver, which is common for both motors. We did not find any further technical information about the motors, which could be used for model determination. In fact, DC motor models are well know and well addressed in various literature. However, in the present case, for such models to be useful, we lack many system (motor) parameters.

Experimental determination of motor parameters, fall out of the scope or this work. In fact, the option of experimental determination was considered, but it was eventually set aside when we realized that while some parameters could be easily determined (like motor resistance), others could be lengthy and very difficult to determine. Indeed, determination of some of the parameters could involve the removal of the motors from the tracker assembly and further detaching them out of their gear box assemblies (just to mention some of the obstacles). This prevented us from obtaining important system parameters and thence the system models, thus narrowing the range of controller design alternatives.

Before coming up with an eventual choice of control strategy, it is important to discuss the basic performance characteristics of dual axis polar mount, parabolic dish sun tracking system and look deeper into the input/output requirements for the fulfilment of such performance characteristics.

4.3.1.2 Basic performance and working requirements for the control system

The control system should have the following working characteristics and capabilities:

(a) Be able to rotate the hour tracking angle (track the sun) at an angular speed not less than 0.25°/min (which is the sun's apparent mean speed). In practice, as the actual hour angular position may be far away from the actual sun's position (0° to 180° distanced), it is required that the hour axis tracking speed be much higher than 0.25°/min. The maximum tracking speeds measured experimentally for the 30V operating voltage have been found to be approximately 0.17^0/S (about 10^0/min) for the hour axis and approximately 0.94^0/S (about 56.5^0/min) for the declination axis.
(b) Be able to rotate the declination axis tracking angle at speeds comparable to those of the hour axis, considering also that the arbitrary declination position may be far away from the required one.
(c) Be able to aim at the sun with an accuracy error angle not greater than half the collector's acceptance angle. The acceptance angle is a collector's constructive parameter. The actual collector's acceptance angle was found to be not better than the default 0.53°, due to sun spot

image size and geometry not fitting inside the receiver's aperture.

(d) Be able to support asynchronous (event driven) changes of set point input variables, at any arbitrary instant of the solar time, and to arbitrary values, other than the real time ones, derived from solar time. In fact, the tracker may be sent to any arbitrary position, such as:

- To a safety position in the occurrence of a safety event. For instance, if the system overheats (refer to safety considerations, in section 3.2.1.7), then the collector should be defocused to prevent it from further collecting heat, until system temperature falls within the safe operating boundaries.
- To rest position, at night (hour angle is above sunset hour angle or it is below sunrise hour angle). The choice of the rest position should be led by safety considerations, namely, it should be a defocused and wind safe collector position, etc.
- To any other arbitrary position that may arise from any control need, including human generated supervisory control events. This should include a human manually operated remote control.

The last requirements (d) and their nature (asynchronism and arbitrariness of set point changes) enforce the need of an overall supervisory control strategy, as discussed earlier and represented in Figure 36.

There are further performance and working requirements, some implied from the system description that has been being discussed, others not explicitly mentioned at this point.

4.3.1.3 Further analysis and change of system variables for fulfilment of the above specifications

Controller inputs:

i. Set point hour angle,
ii. actual hour angle (also a process output),
iii. set point declination angle,
iv. actual declination angle (also a process output), and
v. Fine alignment photodiodes (consisting of two pairs).

Where, the actual angles are measurements taken from the real hour and declination angles. To make such measurements possible, we have chosen and we implemented the use of absolute position angular sensors, built from single turn linear potentiometers that we fitted on the axes. The set point angles are MCU calculated values, as functions of solar time. In turn, the photodiodes are additional angular position variables for fine sun aiming. Fine aiming should be done after a coarse aiming is performed based on variables i to iv. Fine aiming is

to enforce the fulfilment of requirement (c).

It is worth noting that, since solar time is a varying time axis (refer to eqs.2.1.1.7 through 2.1.1.11), the set point tracking angles are governed by time-varying laws: It is good to remember that the determination of solar time involves local time, time zone and the equation of time (the time-varying component).

Controller outputs (also process inputs):

1. Axis select (= motor select or also the abbreviated MSel): For reasons discussed earlier (see section 2.1.3.3.5 "Continuous versus step tracking"), we have chosen a step tracking strategy, rather than be tracking the sun continually. On the other hand, taking advantage of the fact that the declination angle is a slow varying magnitude, we determined that only one motor actuator will function at a time (only one axis will rotate at a time, not both simultaneously). This last choice streamlines tracker driver circuit implementation in the support of both MCU/automatic control and human manually operated remote control, insuring fulfilment of requirements defined at last point of (d) above.

2. Motor direction (2 variables): To provide forward or backward tracking movements, eg. for angular position control, forward is when actual angle is below the set point and backward is when actual angle is ahead of the set point angle. To simplify design, we have chosen to break down direction into 2 variables, see function table below.

Direction Output variable		Movement
Reverse (Rev)	Forward (Fwd)	
0	0	Stop = No movement
0	1	Forward movement
1	0	Backward movement
1	1	Stop = No movement

Table 4.1 – Definition of direction control inputs

3. Motor speed: This determines or affects the speed at which the tracking assembly rotates about the selected axis and specified direction of movement. As actuators are DC motors, we have chosen pulse width modulation (PWM) for speed control implementation. How PWM is implemented at microcontroller level will be addressed in the relevant sections ahead. It is worth noting also that this output may perform the "Stop – No movement" function mentioned in the table above. This is not an exact redundancy, once the electrical implementation of DC speed control involves the use of a charge storage circuit that may introduce some output inertia, hence delaying the stop function. Indeed, whilst the direction outputs depicted in table above are boolean outputs, the speed output at motor level, is rather an analogue variable.

4.3.1.4 Control strategy

As can be concluded from the above analysis and synthesis of input and output

variables, the tracker subsystem is itself a multiple inputs, multiple outputs, time-varying and event driven system.

The use of model based control design was discussed and discarded earlier. At this point, the further breaking down of the tracker control solution into partial controllers (a controller for each motor/axis) was also discarded, once, rather than reducing complexity, it would actually end up in adding complexity to the overall control framework.

All the above, led to the conclusion that the choice of controller should be among the following types: finite automaton, fuzzy or neural network, including their combination with PID type control strategies or with each other. The main reason of this choice as discussed progressively in earlier sections, lies in the fact that with a digital controller of these types, one can address the complexity of a MIMO system, non-linearity and time variance, as also the absence of a properly defined process model. Also, controller implementation in the realm of the MCU based system, could also be straightforward, including future modifications of controller behaviour to leverage system performance.

4.3.1.5 Finite state machine (FSM) controller approaches

Our primary choice has fallen over a finite state machine controller (FSMC), and addressing the control design of the tracking subsystem as a single controller (in lieu of 2). Many different scenarios were considered for the FSM implementation, which range from simpler to more complex approaches. However, only 3 of the possible scenarios were eventually found to be appropriate for a straightforward and gradual FSM design and implementation. They are presented and discussed gradually in the controller block diagrams that follow in next sections.

4.3.1.5.1 FSM based tetra-directional On/Off tracker controller

The first scenario is depicted in Figure 37, which is a FSM based four-directional On/Off controller, a multiple inputs and multiple outputs (MIMO) controller. It is a unique controller for both the hour and the declination axis tracker motions.

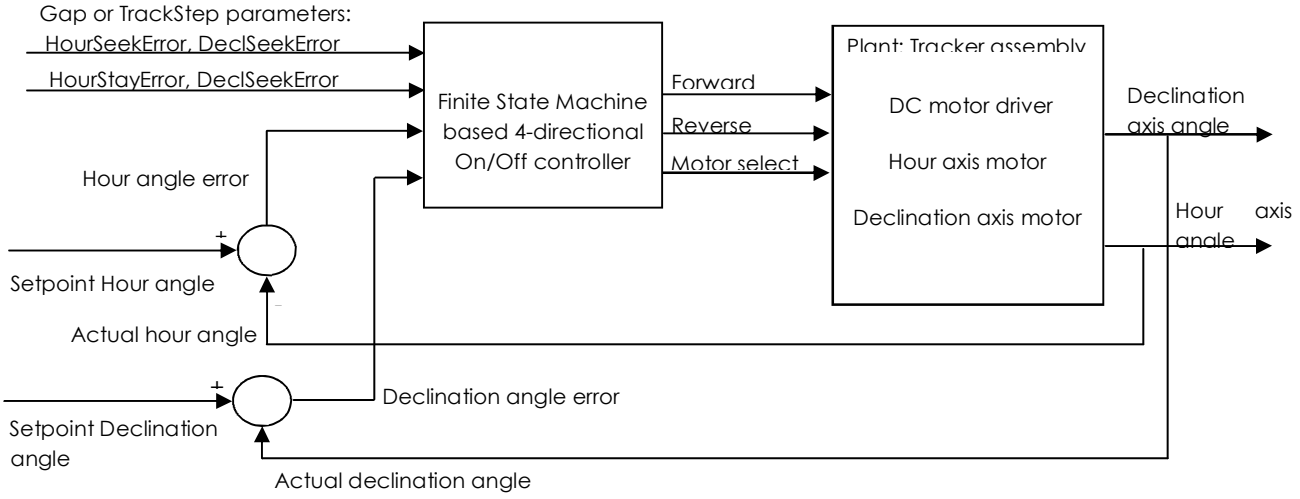


Figure 37 – Diagram of the FSM based tetra-directional On/Off tracker controller

Where, HourStayError or DeclStayError is the magnitude of the tracking step (should be not greater than half the collector's acceptance angle) and SeekError is half the size of the On/Off differential gap, which is the allowed error when seeking the set point. SeekError should be less than or equal to StayError. The direction variables (forward, reverse, motor select) are as discussed in section 4.3.1.3 a) and b) detailed in tables 4.1 and 4.2. The actual angle is the measured value of either the hour or the declination angle.
However, for a straightforward presentation of the controller scenarios, we depict a simplified diagram shown in Figure 38, below:

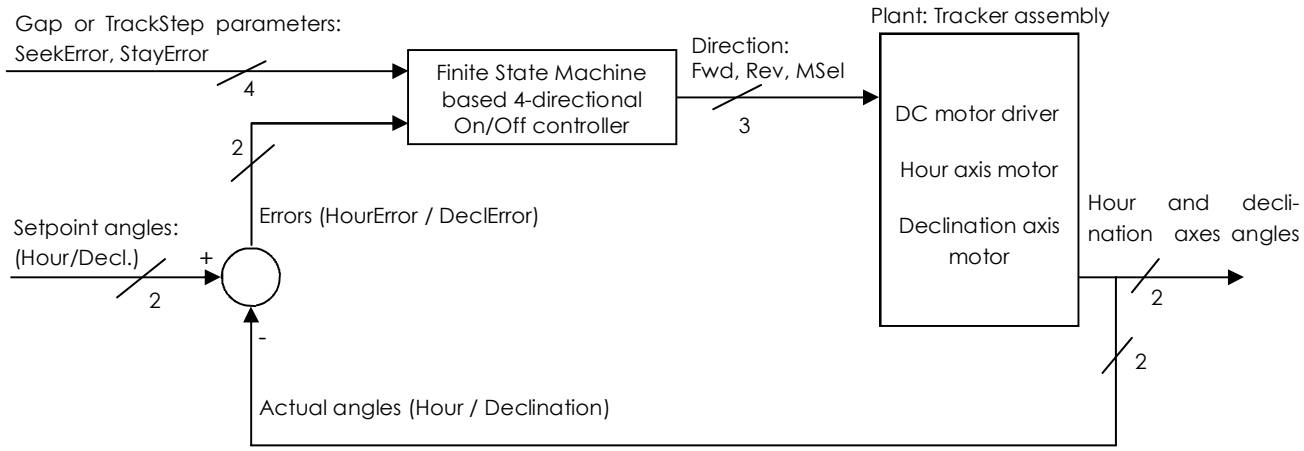


Figure 38 – Simpified diagram of the FSM based tetra-directional On/Off tracker controller of the previous figure.

4.3.1.5.2 FSM-PID tetra-directional tracker controller

The second scenario, shown in Figure 39, is motivated by the fact that an On/Off controller is prone to errors (as discussed in sections 2.2.3.1 and 2.2.3.2). So, from the first scenario (Figure 37 and Figure 38) we derived the second one, consisting of the addition of a PID block, to control the speed at which the tracker approaches a certain set point angle in the selected axis. The PID components (discussed in section 2.2.3.3) will add efficiency by conveniently producing a speed according to the error between the set point and the actual angle.

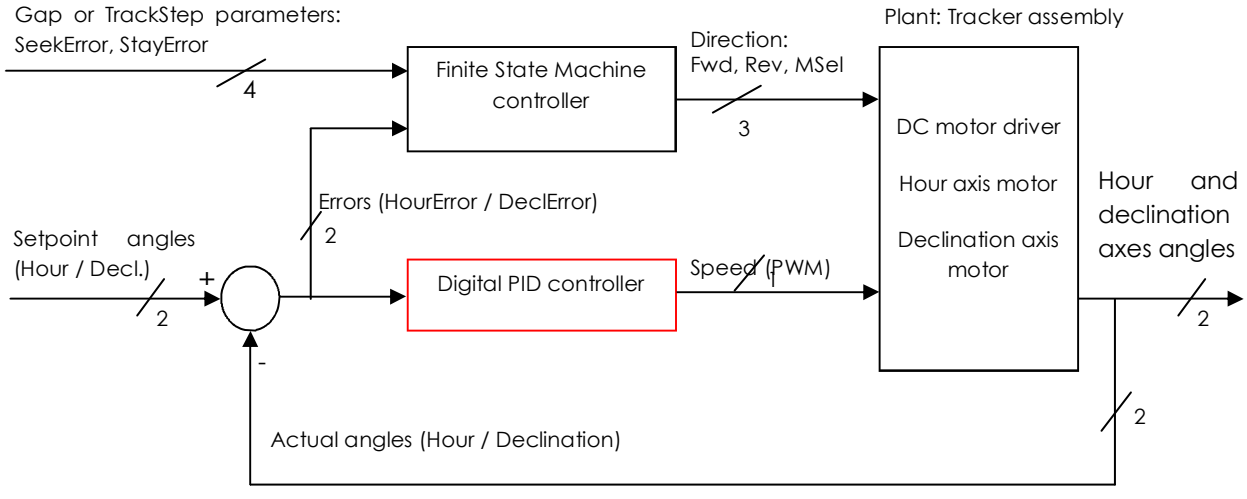


Figure 39 - Finite State Machine - PID tetra-directional tracker controller.

4.3.1.5.3 Fuzzy finite state machine - PID tetra-directional tracker controller

After the PID enhancements introduced in the second scenario (Figure 39), the resulting tracking system may still be subject to possible inaccuracies and oscillations on seeking the set point, due to the following possible factors:

(a) The crisp nature of the set point and measured variables along with the crisp nature of the FSM boolean inferencing mechanism;

(b) The variation and non-linearity of the mechanical load, caused by:

(i) the displacement and variation of the concentrator's centre of gravity with respect to the supporting structure;

(ii) mechanical imperfections (lack of cleanliness, unevenness and rusting) of the motion transmission system, as well as

(iii) sudden and unpredictable changes of the environment conditions (like the wind);

(c) Poorly defined PID gains: When a process model is not defined the PID gains have to be determined experimentally, which is not straightforward and subject to errors, moreover,

(d) Large PID gains may be good for large errors but improper for small ones (this could be overcome if the gains could be adjusted on the fly). Also, the fitness of the PID gains may also be compromised by the variability and non-linearity of the mechanical load, as discussed above.

The list above, suggested enhancing the previous approach by incorporating a fuzzy functionality. This can be achieved by (among other ways) turning the simple FSM controller into a fuzzy FSM (FFSM). See section 2.2.3.6 for an introduction to fuzzy logic controllers.

Figure 40 presents the FFSM approach (the 3rd scenario). A fuzzy functionality would avoid oscillation around the set point, due to the non crisp nature of the fuzzified inputs along with the fuzzy inferencing mechanism (as opposed to the Boolean inferencing mechanism of a simple FSM). For instance, a value slightly above or slightly below the desired seeking set point position, will still be considered equal to the desired position. On the other hand, as can be seen in the Figure 40, with the use of a fuzzy approach, the real time adjustment of the PID gains (or directly the speed itself), discussed above, can also be addressed (see the red arrows from the FFSMC block to the PID block).

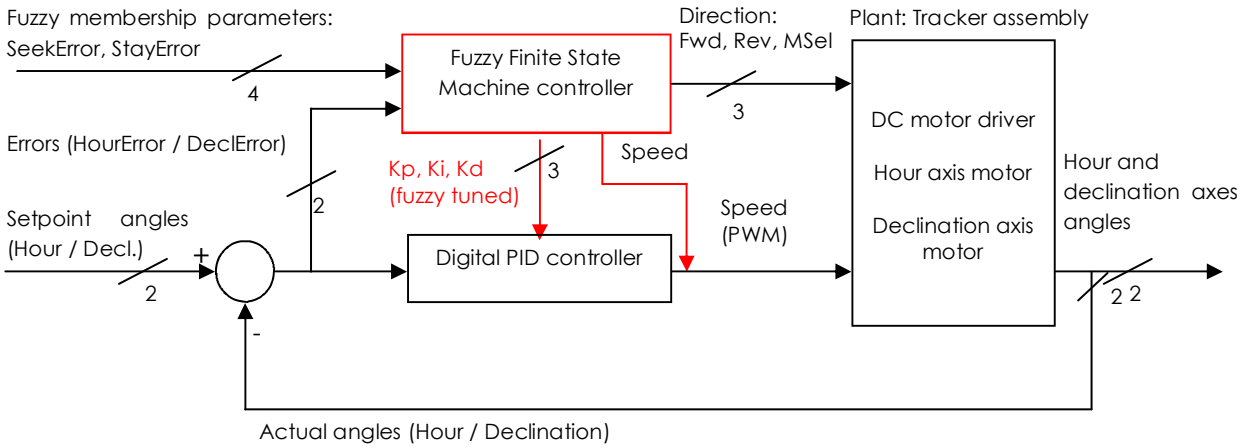


Figure 40 - Fuzzy Finite State Machine - PID tetra-directional tracker controller.

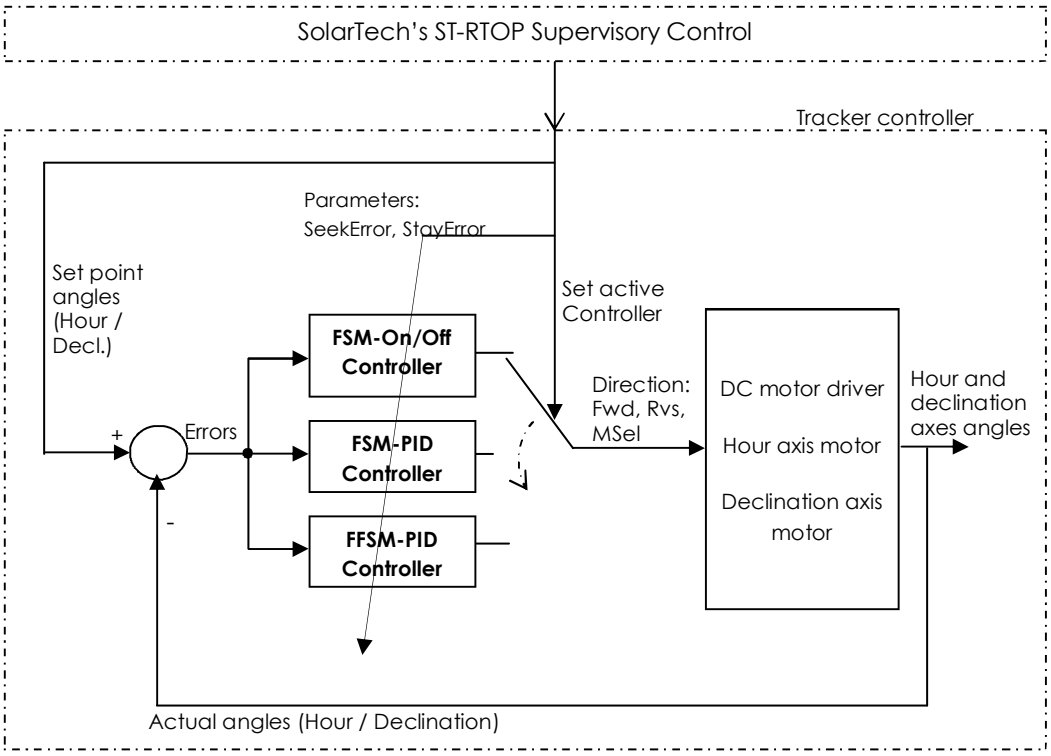


Figure 41 – The 3 considered tracker controllers switched under supervisory control

4.3.1.6 Controller Choice and Implementation: FSM-On/Off Tracker Controller

Based on the discussion above, it was decided to design a control system in such a manner, that incorporates all the above strategies. Figure 41 illustrates the integrated control. In this way, it is possible to implement each approach independently. The overall operator or controller (this could be a high level program) can choose the control algorithm to use. This is indicated by the dashed semicircular arrow in Figure 41 resembling a rotary switch.

This approach of design also facilitates the implementation of the simplest scheme first and more advanced algorithms can be added at later stages as and when required.

4.3.1.7 Design of the FSM Based Tetra directional On/Off Controller

In Figure 45, we present the targeted FSM controller in state flow diagram. Such diagram however, is a result of incremental changes applied over the first simplest state diagram. So, in order to simplify the discussion, we will perform here the same incremental presentation, starting from the most simplified approach in Figure 42 to the targeted final state machine diagram in Figure 45.

For the first approach we assume that the dish assembly is either stopped or moving, in accordance with the magnitude of error (setpoint_angle – actual_angle). We then obtain the state flow diagram of Figure 42.

Note, however, that there is a major challenge:

1) The dish assembly is not just moving:
- it may be rotating either westwards or eastwards, on the hour axis, according to the magnitude and sign of the hour angle error;
- it may be rotating either northwards or southwards, on the declination axis, according to the magnitude and sign of the declination angle error.

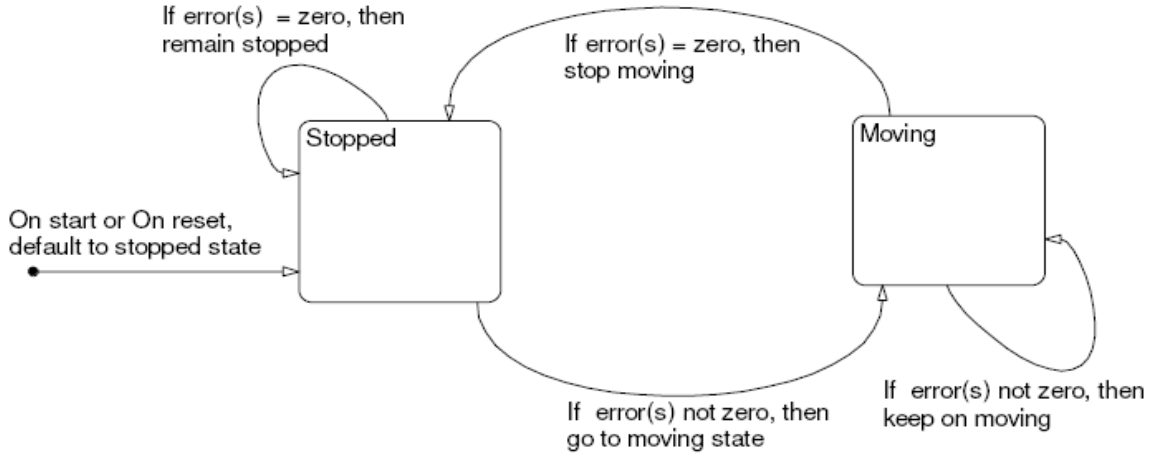


Figure 42 - Initial approach of the FSM, not taking in consideration the axis and orientation of motion.

From the last consideration we evolve to braking down the single "Moving" state into four states, namely:

- MovingEastwards – when the hour angle error is negative (the actual hour angle is ahead of the set point);
- MovingWestwards – when the hour angle error is positive (the actual hour angle is below the set point);
- MovingSouthwards – when the declination angle error is negative (the actual declination angle is ahead of the set point);
- MovingNorthwards – when the declination angle error is positive (the actual declination angle is below the set point).

The resulting state flow diagram is presented in Figure 43. Note that, since there are no simultaneous motions of the two axes (reasons discussed in section 4.3.1.3), from any moving state, the machine should always transit to the Idle (stopped) state. Only from this idle state it can switch to any of the moving states.

Now, observing the state flow diagram of Figure 43 we can easily notice the following new or remaining shortcomings:

2) It is not deterministic, since, in the Idle state, two transition conditions may become true simultaneously. So, there is a need of establishing transition priorities, as a means of providing determinism, or else, there will be a situation when the system will go to an unpredictable state.
3) It does not yet define the state outputs, that will select the active axis (motor select) and which orientation (forward or reverse);

4) The crisp nature of the FSM boolean inferencing which evaluates these transition conditions, will cause the dish assembly to oscillate around the set point. To address possible oscillation, there is a need for introducing the differential gap, hysteretic On/Off control discussed in the section 2.2.3.2.

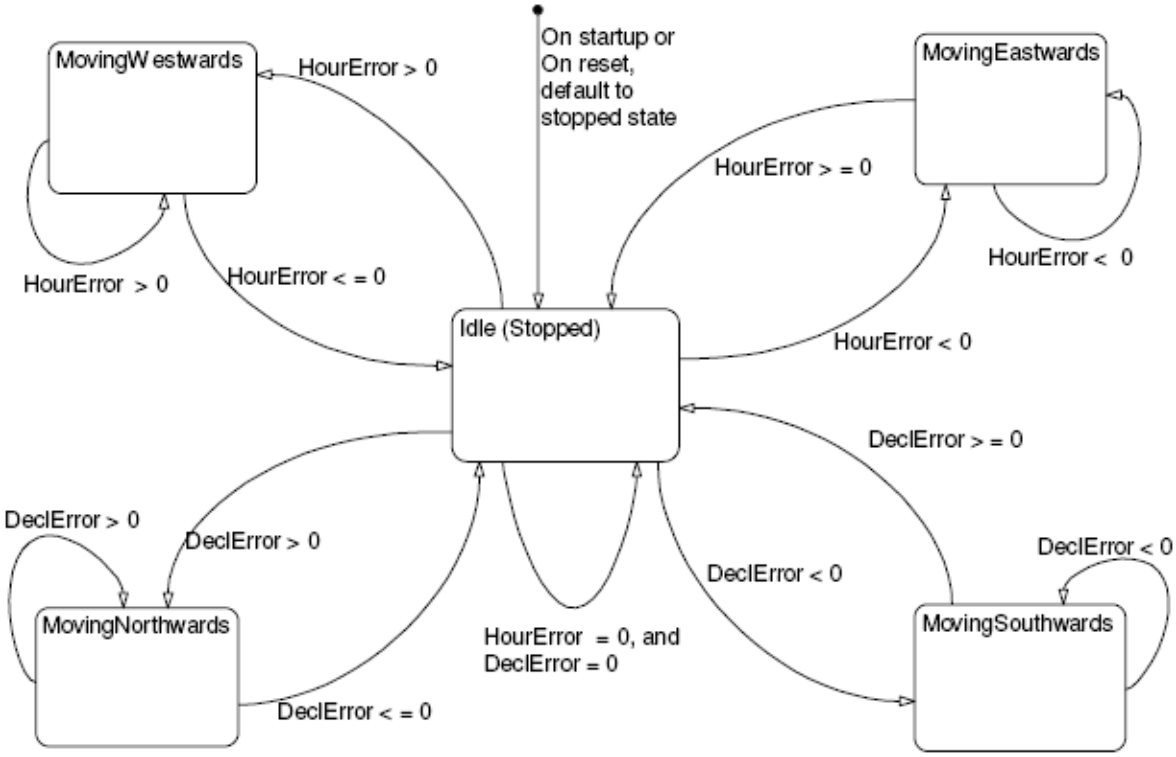


Figure 43 - Second approach of the FSM, now taking in consideration the axis and orientation of the dish assembly motion.

To address the issue of non deterministic behaviour, at this starting point, we have given the hour axis motion precedence over that of the declination axis. In such a way whenever both the HourError and the DeclError are non null, hour axis motion is performed first and followed by the declination axis motion. This also means that, if HourError becomes non null while declination axis is in motion, this will be suspended, to grant the priority to the hour axis as described.

Concerning the state outputs they are partly defined on the table 4.1. The complete definition is as follows:

Axis Select (MSel)	Direction Output variable		Selected motor (axis) and motion orientation
	Reverse (Rev)	Forward (Fwd)	
0	0	0	Stop = No movement
0	0	1	Hour Motor Westwards rotation
0	1	0	Hour Motor Eastwards rotation
0	1	1	Stop = No movement
1	0	0	Stop = No movement
1	0	1	Declination Motor Northwards rotation
1	1	0	Declination Motor Southwards rotation
1	1	1	Stop = No movement

Table 4.2 - FSM state outputs.

In turn, for introducing the differential gap behaviour, we use the HourSeekError, for the hour axis motion, and DeclSeekError, for the declination axis. Either variable

will be made, in run time, equal to half the size of the desired differential gap (see section 2.2.3.2). HourSeekError and DeclSeekError have also been discussed in section 4.3.1.5.1.

Furthermore, for the implementation of step tracking strategy, we use the HourStayError for the hour axis motion and DeclStayError for the declination axis. Either variable will be made, in run time, equal to the size of the desired tracking step in degrees of arc. HourStayError and DeclStayError have also been discussed in the section 4.3.1.5.1.

It is very important noting that, both variables, the SeekError (= half the gap) and the StayError (the tracking step) play roles in both the differential gap and track step features: SeekError is the accuracy while in motion, seeking the set point, whilst StayError is the allowed error while stopped, staying for the next tracking step.

To avoid oscillation the SeekError should be left sufficiently less than the StayError. This is equivalent to the difference |StayError – SeekError| being made as big as required to avoid oscillation. This is because, a stop action is triggered when the differential gap boundary is crossed. However, if at the same time we are beyond the step size, a moving action for the opposite direction will be immediately triggered, which equates to oscillation. This means, in other words, that the differential gap should lie inside and centred into the double step interval.

We should point out that during the development phase we considered having only the tracking step parameter (StayError) and thereby deriving the differential gap size as a fixed percentage of the step size. However, for the sake of simplicity, we have chosen to keep the two concepts as separated by default, and leaving the supervisory operator the freedom of specifying them at run time, including that of making them equal. We did this during the experimentation, and the oscillatory behaviour was confirmed when we made SeekError equal to StayError.

After incorporating the last considerations over the diagram of Figure 43, the resulting state flow diagram is shown in Figure 44. Once we obtained the last approximation (Figure 44) we can now see that:

5) it still does not satisfy all the working and performance requirements, namely, the ones mentioned in 4.3.1.2.(d).

So, to fulfil these requirements, there is a need for incorporating supervisory controller actions, that will change the normal FSM behaviour governed by the transitions conditions in the Figure 43. Introducing supervisory actions can only be done by modifying the current transition conditions. We did so by incorporating modes of operation for the tracking process control, as described in the table 4.3.

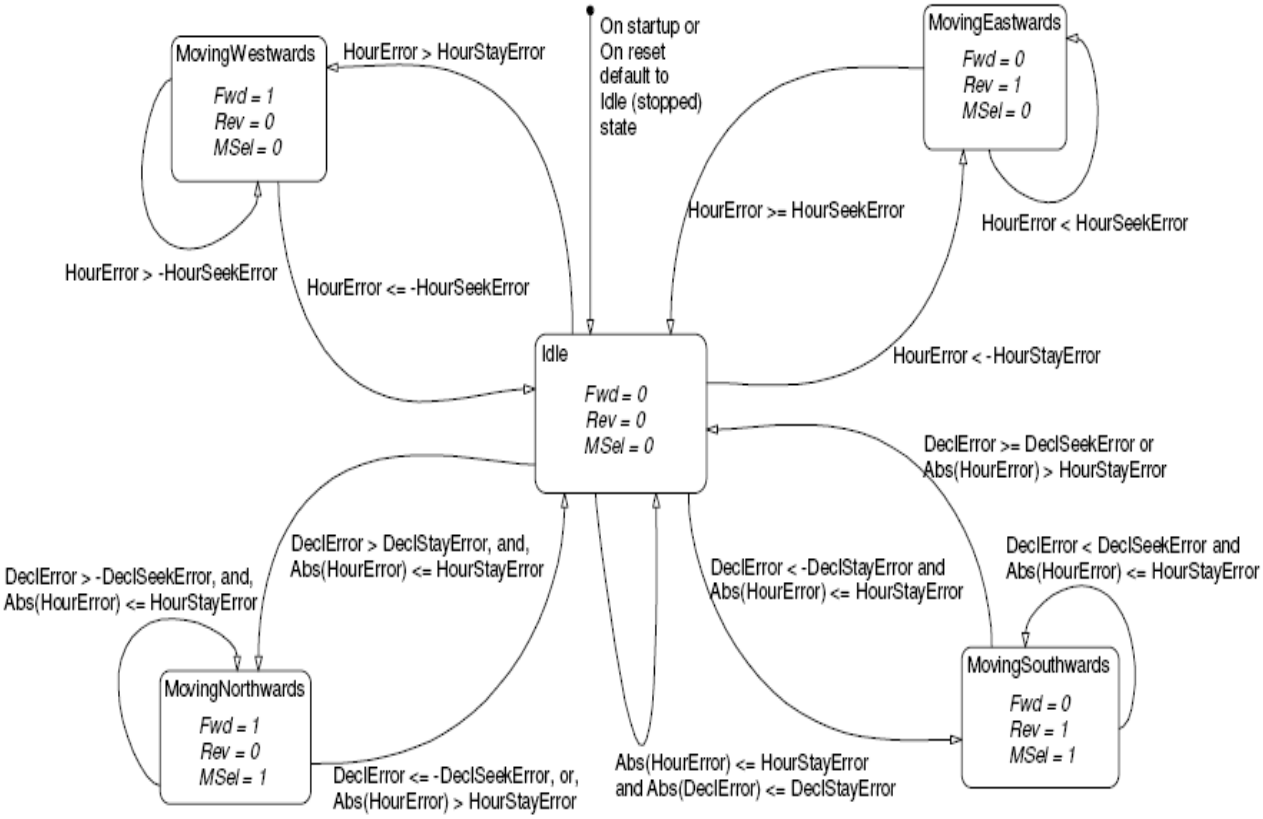


Figure 44 - Third approach of the FSM

In the chart of Figure 44, the FSM now incorporates the step tracking strategy and the differential gap/hysteretic On/Off control, as well as it gives the hour axis motion precedence over any declination axis condition.

Mode	Description	State	Fwd,Rev,MSel
Idle	Do nothing, insure no movement	Idle	11X
Manual	Do nothing, allow manual control	Idle	000
Auto	Automatically perform tracking control	Idle MEW MWW MNW MSW	11x 010 100 101 011
Safety	Perform tracking automatically with safety set points		

Table 4.3 – Definition of mode supervisory control inputs and some related outputs, where Fwd, Rev and MSel are the state and mode dependent outputs (see table 4.2).

In the table, the abbreviations are as follows: MEW (Moving Eastward), MWW(Moving Westward), MNW(Moving Northward), MNW(Moving Southward).

Brief explanation of the modes:

(a) When *idle* mode is selected, the machine is sent to the Idle (stopped) state regardless of the current tracking error(s). Once in the idle state, it will be kept there until any mode change occurs. Meanwhile, the state outputs Fwd, Msel are set to 1 to insure no motor movement (see table 4.1).

(b) When *manual* mode is selected, the machine will have the same behaviour as for the *idle* mode, except for the state outputs: they will all be cleared (=0) to grant the user the freedom of controlling the motor drive through a wired remote button or even trough moving the tracking assembly manually.

(c) When *auto* mode is selected, the machine takes its normal tracking behaviour, based on the current tracking errors and basic transition conditions as proposed by the flow diagram of 3rd approach in Figure 44.

(d) When *safety* mode is selected, normal tracking behaviour will happen although the set point hour and declination angles will be set to their safety values.

After incorporating the supervisory controller actions into the state transition conditions and the state outputs, we obtain the flow diagram shown in Figure 45.

At later stages, there will be further modifications to facilitate and make possible the software implementation of the tracker control itself and the supervisory control program as a whole. Also, experimental observation of the tracker functionality may eventually suggest further corrections and or enhancements.

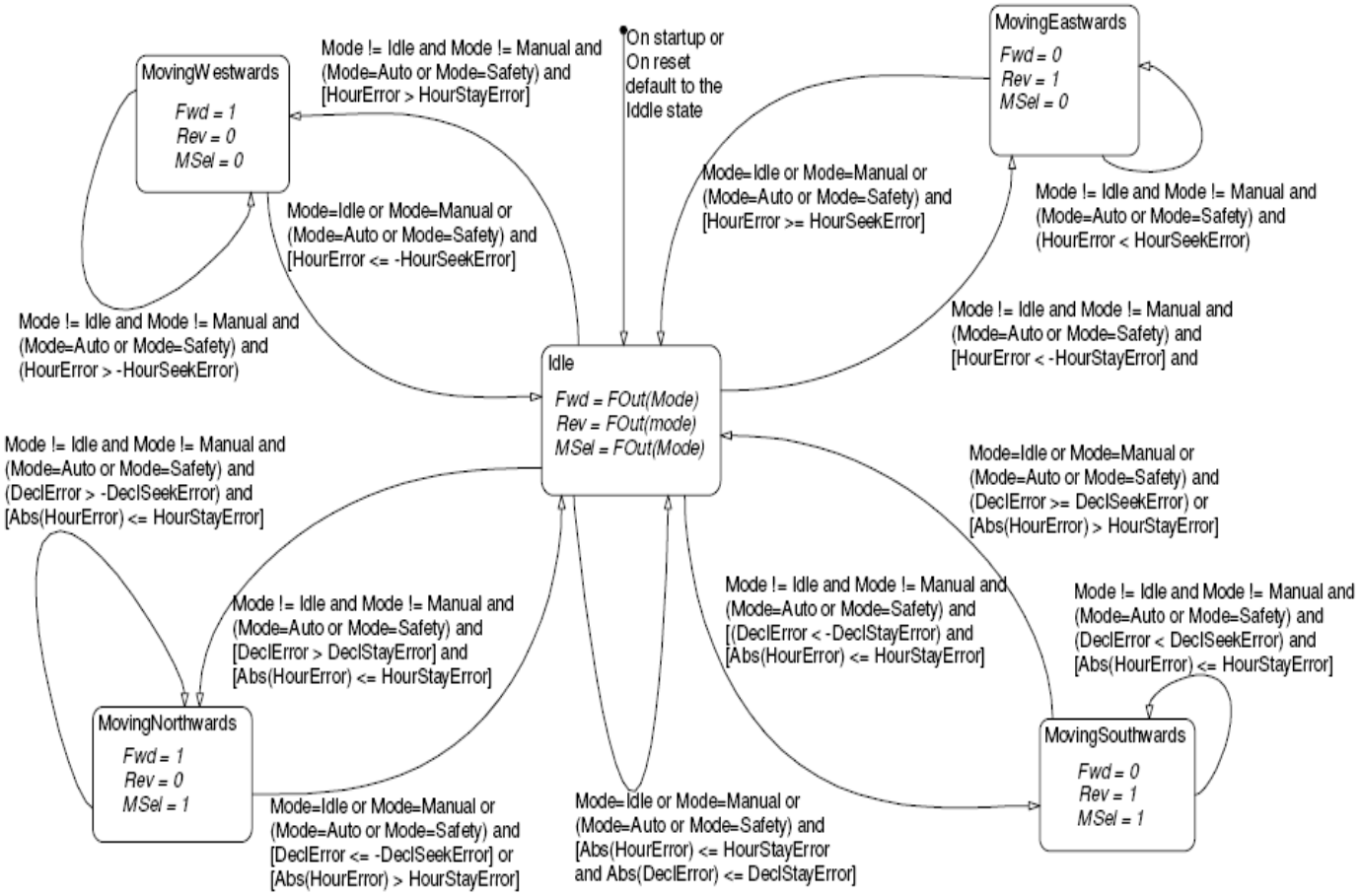


Figure 45 - Final (design stage) FSM flow diagram

4.3.2 Charging Pump Controller

For the control of the charging pump we consider a PID controller at a first stage and a Fuzzy-PID controller at an advanced one. The inclusion of the fuzzy component in the latter scenario will provide self tuning of PID gains as well as other measures to counteract system non linearity. The first scenario is presented in Figure 46, while the Fuzzy-PID scenario is shown in Figure 47. We have chosen to implement the simple digital PID approach. Despite the choice, as with the tracker controller, we left both approaches included in the control framework, being the supervisory controller (under user choice) responsible of selecting the active controller, as depicted in Figure 48.

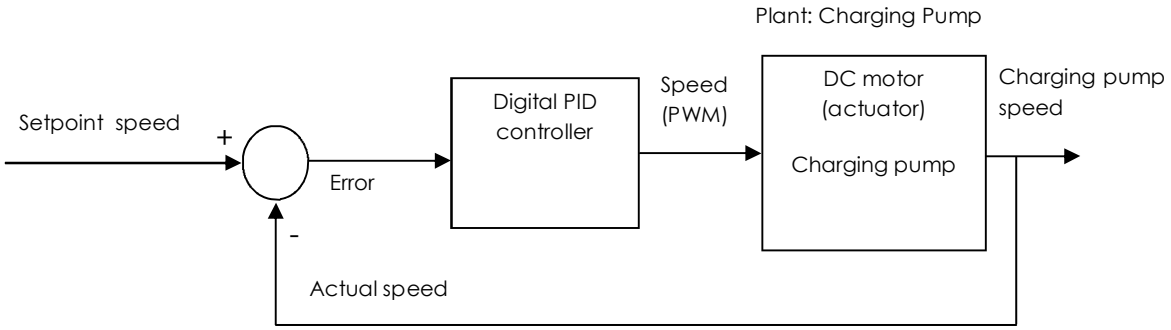


Figure 46 - Charging pump PID controller

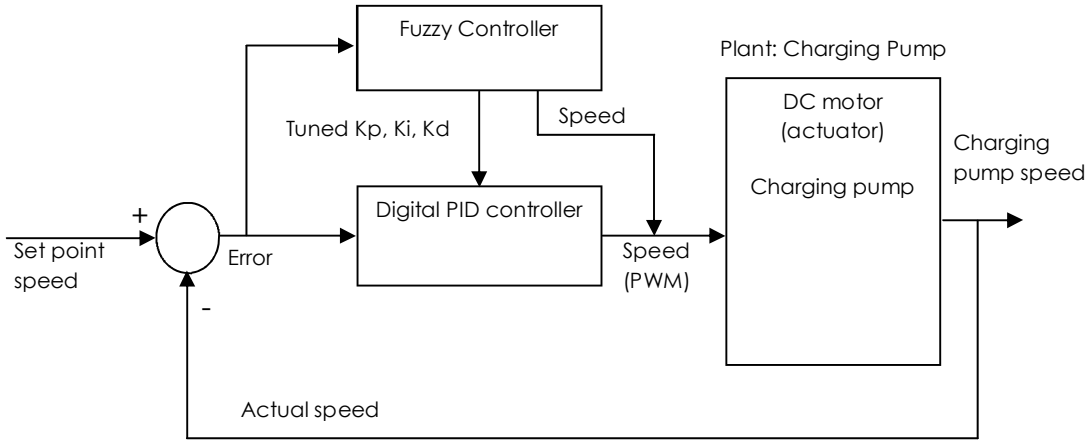


Figure 47 - Charging pump Fuzzy - PID controller

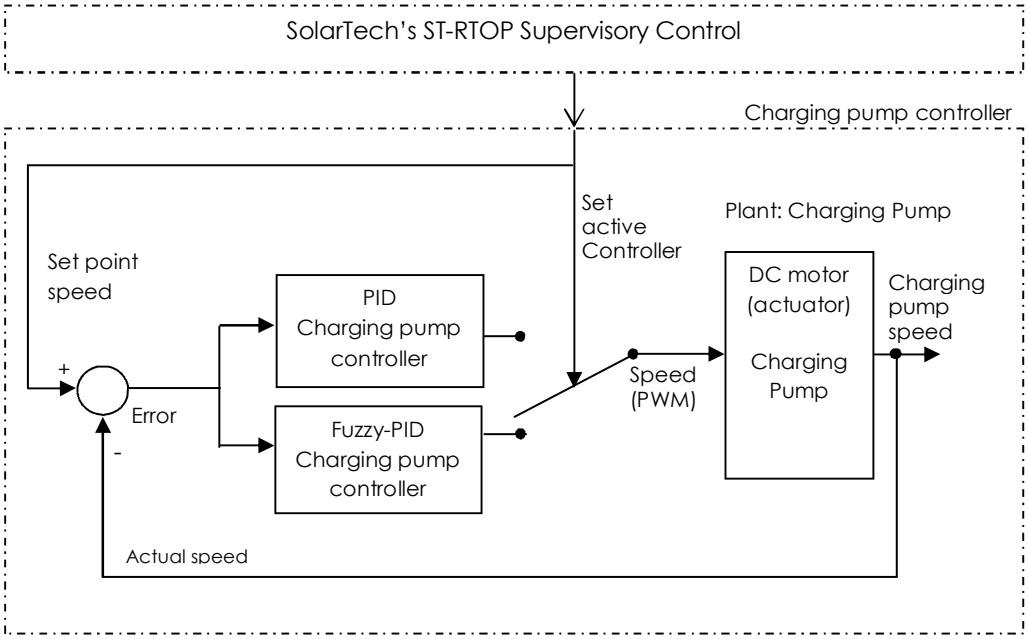


Figure 48 - Charging pump controller under supervisory control

4.3.3 Discharging Pump Controller

For the control of the discharging pump we consider the same scenarios addressed for the control of the charging pump and depicted in Figure 46 through Figure 48. What will eventually differ are the underlying control laws and hence the parameters like PID gains, set points, etc.

4.4 Chapter summary

In this chapter we have walked the following steps:

(1) We have developed a conceptual model of the data acquisition and control system based on the set of required inputs and outputs with respect to the plant, along with its basic working and performance requirements;
(2) We have chosen and designed a tracker controller , whose implementation will be carried out later at software level on chapter VI;
(3) Conceptual models for the charging and discharging pump controllers have also been discussed and their implementation will also be performed at software level on chapter VI.

The next chapter follows, with the hardware design of the data acquisition and control system (sensors + controller + actuators) which has been discussed and proposed in this chapter.

Chapter V – Design of the experimental prototypes of the microcontroller based system according to the models addressed in chapter IV

In this chapter we will perform the electronic design of the data acquisition and control system which comprises (i) the hardware interfaces for reading analogue inputs, (ii) the microcontroller base unit with all interfaces, and (iii) hardware interfaces for the control actuators. The design will be guided by discussion in the previous chapter, particularly the conceptual model of Figure 35 as well as all the discussions concerning the controller design.

5.1 Basic considerations. MCU and other components choice

The design and implementation of the microcontroller system itself is presented in the following pages.

It is based on the input and output needs described in the earlier chapters and guided by the generic idea of low cost, local availability, ease of prototyping and straightforward development and implementation, as well as considering a degree of expandability and ease of future modification. However, complying with all these ideas was not always easy as the ideas of low cost and simplicity may diverge with that of expandability, etc. Before we could come up with a choice of MCU, we performed a deeper analysis of the resources requirements (inputs/outputs and memory).

5.2 Hardware design level re-evaluation of input/output requirements

5.2.1 Analogue to digital conversion inputs.

The table below provides a list of analogue input variables that will be monitored by the system and then they will have to be digitized through an AD converter.

#	Description	Analogue inputs
1	Actual hour angle (read from an angle to voltage converter potentiometer – for coarse alignment)	1
2	Actual declination angle (read from an angle to voltage converter potentiometer – for coarse alignment)	1
3	2 triple axis analogue output MEMS accelerometers (for an alternative coarse alignment)	6
4	Fine alignment photodiodes	4
5	Wind direction / orientation (for safety monitoring)	1
	Wind speed / load (for safety monitoring)	1
6	Onboard (MCU) ambient temperature	1
7	User heat utilization set point temperature (to be read from an angle to voltage converter potentiometer)	1
8	There may be further ADC inputs that may arise from any need during future the system development.	?
	Subtotal	16

Table 5.1 - ADC inputs

All ordinary low end and mid range microcontrollers have a maximum of about 8 ADC inputs. From the requirement of at least 16 ADC inputs we conclude that an external analogue multiplexer or a pair of external ADC and analogue multiplexer will be required, unless a high end MCU is used.

The 16 ADC inputs require a 16 channel multiplexer (mux), which involves the use of 4 bit address and 1 bit strobe for storing the address. For future expandability reasons 5 address outputs are used. This is also to take into account line 8 of the table 5.1 above. So, the ADC interface end up requiring 5 mux address bits (digital outputs), 1 address strobe bit (digital output) and 1 ADC analogue input (from the analogue mux to the MCU), as depicted in Figure 49 below.

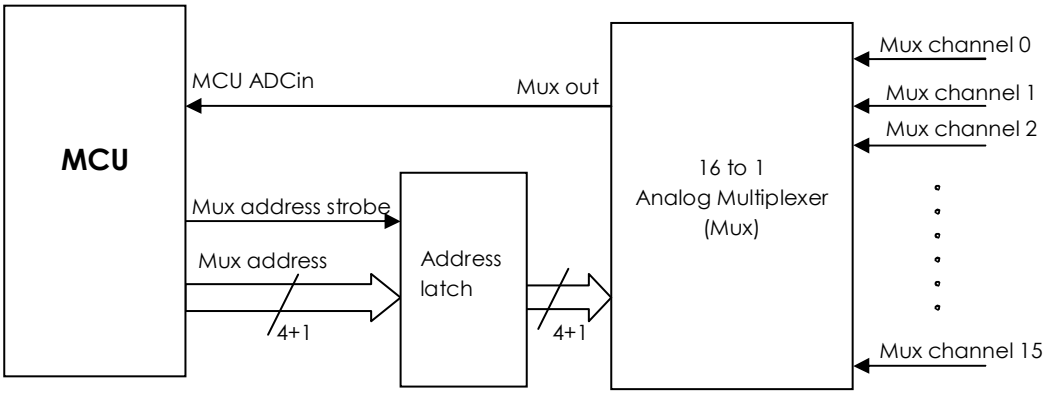


Figure 49 - Externally multiplexed ADC inputs

5.2.2 Thermocouple to digital conversion (TDC) inputs.

The measurement of high temperatures requires the use of thermocouples. This is the case of the temperatures to be read from the ST-KZN's receiver, the TES and the utilization subsystems. Thermocouples of the K type are already built into the mentioned subsystems. The table 4.2 below summarizes the thermocouple to digital conversion input requirements.

#	Description	Analogue inputs	Notes
1	Receiver's inlet temperature	1	
2	Receiver's outlet temperature	1	
3	Receiver's surface temperatures	2	Up to ≈ 8
4	TES inlet temperature	1	
5	TES outlet temperature	1	
6	TES profile temperatures (9 levels x 5 radial points)	45	
7	User heat utilization inlet temperature	1	
8	User heat utilization outlet temperature	1	
9	User heat utilization surface temperatures	1	
10	Plant (outside) ambient temperature	1	
11	Additional thermocouple inputs that may arise from any need during future system development	?	
	Subtotal	55	

Table 5.2 - Thermocouple to digital converter inputs

The high number of thermocouple inputs (55) enforces the need for an external multiplexing solution, regardless of the microcontroller choice. In particular, the use

of a specific purpose ADC: a K type thermocouple to digital converter (TDC), MAX6675 (see further details in section 5.7.4 and in the datasheet), is a good solution, since it simplifies and streamlines both the hardware and the software design and development. The use of AD595 cold junction compensated thermocouple amplifier was also considered.

The 55 thermocouple inputs require a 64 differential channel multiplexer: 6 bit address and 1 bit strobe for storing the address. Also, 1 more bit for address is considered for future expandability (at TDC board), although, we keep the 6 bits address on the MCU as there are 9 (=64-55) spare thermocouple channels.

So, the TDC interface ends up requiring 6 mux address bits, 1 address strobe bit and 1 SPI bus select bit (other 2 SPI bus bits are shared among other SPI devices) , as depicted in Figure 50 below.

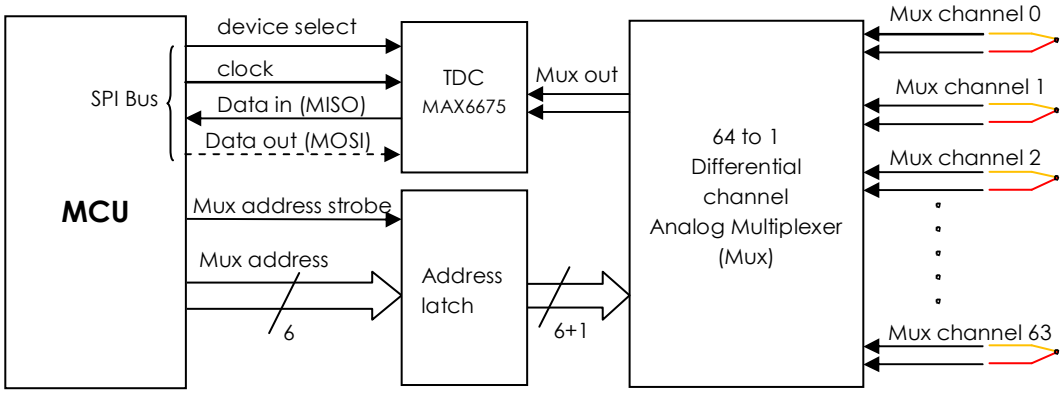


Figure 50 - Multiplexed thermocouple to digital conversion

5.2.3 Global analogue and digital inputs and outputs requirements

Besides the analogue and digital inputs/outputs discussed above, there are a number of I/Os that deal with miscellaneous devices interfacing. The following table 5.3 summarizes the global I/O needs at MCUs ports level, including the interfacing needs (for ADC and TDC) arisen from the previous discussions.

#	Description	Inputs/outputs	Notes
1	SPI bus: 3 bits shared among SPI devices.	3	
2	Keyboard i/o: 2 control and 4 data	6	4 bits sharable
3	LCD i/o: 2 control and 4 data	6	4 bits sharable
4	RS232 interface	2	
5	Sun Tracker: *Fwd, Rev, MSel, pwmSpeed*	4	3 sharable
6	Charging Pump: *pwmSpeed, SpeedFeedBack*	2	
7	Discharging Pump: *pwmSpeed, SpeedFeedBack*	2	
8	Real time clock: 2 control and 1 data	3	2 bits sharable
9	ADC: 1 analogue input, 5 mux address and 1 strobe	7	5 bits sharable
10	TDC: 6 mux address ,1 strobe, 1 SPI select,3 SPI bus;	8	6 bits sharable
11	SD Card: 1 SPI select, 3 shared on SPI bus	1	
12	M2M interface: 1 SPI select, 3 shared on SPI bus	1	
13	Further i/o's to arise from future design steps	?	
	Total	45	

Table 5.3 – Global list of analogue and digital input/output requirements at MCU ports level.

In table 5.3, sharable bits are the ones that can be multiplexed with other bits, thus

sharing the same microcontroller pins. Sharing pins is a way of narrowing the I/O window requirement.

5.3 Basic system architecture and microcontroller choice

As we can see in the table 5.3, the required total number of I/O pins is 45. These 45 miscellaneous I/Os, suggested the use of a microcontroller with 64 pin or with a greater pin count. However, high pin count MCUs are generally expensive and are packaged in fine pitch SMDs (requiring special soldering and handling precautions as they are normally more sensitive to electrostatic discharge and moisture). This compromises the features of low cost and ease of prototyping, required for this development stage.

In order to use a 40 pin microcontroller, which are of lower cost, it was necessary to use additional multiplexing to cater for the 45 i/o connections.

There are other practical advantages of using a standard 40 pin microcontroller. These devices are less sensitive to moisture and ESD and allow for easy handling and they are easily catered in designs using through-hole-plating technology. To perform the narrowing of the I/O window from 45 to 32, we created 2 local buses of shared I/Os as follows (see also table 5.4):

- Local Bus 1 (LB1): shared between the LCD data and keyboard data;
- Local Bus 2 (LB2): shared among ADC, TDC, RTC and sun tracker's interface.

We should point out that one single local bus could be possible; however we wanted reducing possible timing and resources conflicts on the shared bus. Anyhow, bus arbitring mechanisms and/or semaphores were required and used for resolving conflicts on the use of shared buses and resources.

#	Description	**Non shared I/O pins**	**Shared I/O pins**
1	SPI bus: 3 shared bits plus 1 for each device	-	**3- SPI bus**
2	Keyboard i/o: 2 control and 4 data	**2**	**(4 on LB1)**
3	LCD i/o: 2 (+1) control and 4 data	**2**	**(4 on LB1)**
4	RS232 interface	2	
5	Sun Tracker: ***Fwd, Rev, MSel, pwmSpeed***, **1 strobe**	**2**	**(3 on LB2)**
6	Charging Pump: *pwmSpeed, SpeedFeedBack*	2	
7	Discharging Pump: *pwmSpeed, SpeedFeedBack*	2	
8	Real time clock: **2 control** and 1 data	**1**	**(2 on LB2)**
9	ADC: 1analog input, **5 mux address** and 1 strobe	**2**	**(5 on LB2)**
10	TDC: **6 mux address** (shared on LB1), 1 strobe, 1 SPI select and 2 shared on SPI bus	**2**	**(6 on LB2) (2 on SPI)**
11	SD Card: 1 SPI select and 2 shared on SPI bus	1	
12	M2M interface: 1 SPI select and 3 shared on SPI bus	1	
13	**Local Bus 2 (LB2): 6 shared digital I/Os**	**-**	**6 – LB2**
14	**Local bus 1 (LB1) 4 shared digital I/Os**	**-**	**4 – LB1**
15	Further i/o's to arise from future design steps	?	
	Subtotal: 19 + 13 = 32 I/Os	**19 non shared**	**13 shared**

Table 5.4 – Modified list of analogue and digital input/output requirements

It is worth noting that, if envisaging lower pin count (like 28 pins) MCUs, further

narrowing of the I/O window to below 32 bits is still possible, at the expense of simplicity and performance.
So, for the sake of simplicity and performance, we decided for a 32 bit I/O window applicable to 40 pin MCUs, as a good compromise for the realization of the first prototype of this solar controller.

The resulting system architecture, applicable to any 40 (or higher) pins microcontroller, is shown in Figure 51 below.

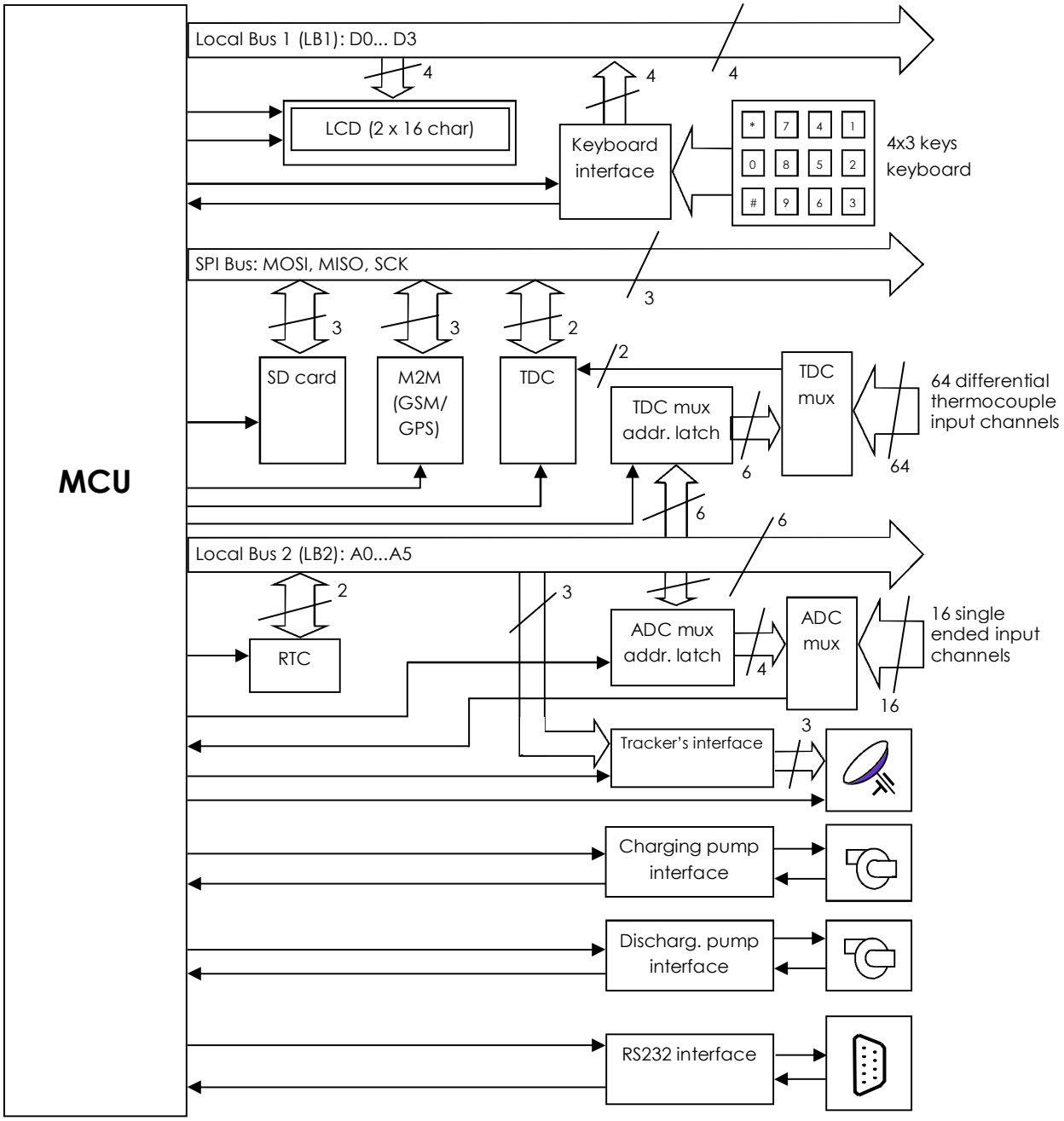


Figure 51 - Basic (MCU independent) architecture of the SolarTech Controller

In Figure 51 above as well as in Figure 54, the number alongside each arrow indicates the number of digital lines required or used.

At this point we are ready to choose the right microcontroller. At the beginning, many MCU families and architectures were considered, namely: Microchip PICs, Atmel ATmegas, Atmel Xmegas, Atmel ARMs, NXP ARMs, Texas Instruments DSPs/ARMs, Analog Devices, ST, etc. However our choice eventually fell on the Atmel AVR ATmega family of microcontrollers, particularly the ATmega32 MCU, depicted in Figure 52 below.

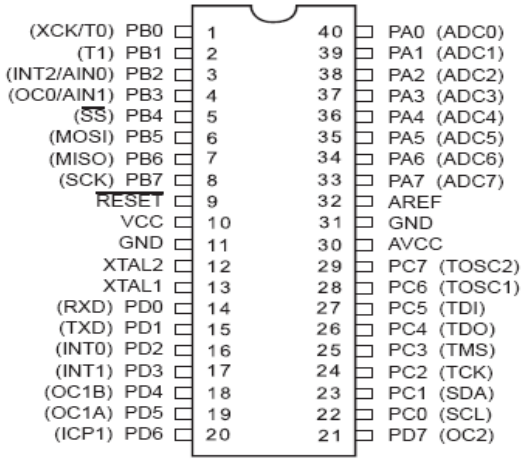


Figure 52 - Atmega32 MCU pinout configuration (DIP package)

The preference was based on some comparative advantages of the ATmega32, like the robust and simple architecture of the AVRs, higher memory and I/O resources (of the ATmega32), ease of prototyping: use of through hole plating technology, more tolerant handling (with respect to ESD and moisture sensitivity), straightforward programming languages and development tools, low cost, local availability (including that of development tools), etc. We should point out however, that there may be some other MCU that may possibly satisfy these requirements.
The alternative use of the ATmega 644P MCU (40 pin DIP) was also considered. This is for taking advantage of its higher memory and I/O handling capabilities, although slightly more expansive and not fully pin and function compatible with the ATmega32 subfamily, which would require small (most likely software level) changes. In the future, if additional requirements arises, then high end ATmegas (such as 128, 1280, 1281, 256, 2560, 2561, etc.) or other architectures/brands, may be considered.

Although the choice was the ATmega32, the design began with the ATmega8535 (pin and function compatible except JTAG) and progressively passed to the ATmega16 and finally to the ATmega32 as memory requirements evolved.

5.5 Final architecture, schematic and components layout diagrams

5.5.1 Basic considerations

The design of the final system architecture (Figure 54) and specific schematic diagrams were based on and/or guided/helped by the following aspects:

- The basic (MCU independent) architecture of Figure 51;

- The architecture and functions of the ATmega32 MCU (Figure 52);
- The architecture and functions of the different IC and components chosen to implement the generic diagrams of Figure 49 and Figure 50, including their datasheets/application notes; and
- In general: all the theoretical and practical considerations discussed in earlier sections, as well as, basic and engineering concepts in the fields of electronics, digital systems and computer science, and CADD/CAE/CAM.

5.5.2 Final system architecture

The final system architecture, evolved from the one in Figure 51 as a result of developing it over the ATmega32 and other chosen components, is presented on the diagram of Figure 54.

Whilst all the schematics and component layouts will be presented, we find no absolute need for doing so for PCB track layouts. Figure 53 shows all the pc boards that the system is comprised of. It is important to note that these boards can be easily identified on the overall system architecture presented in the Figure 54, as they are highlighted using the same colour as in the block diagram below.

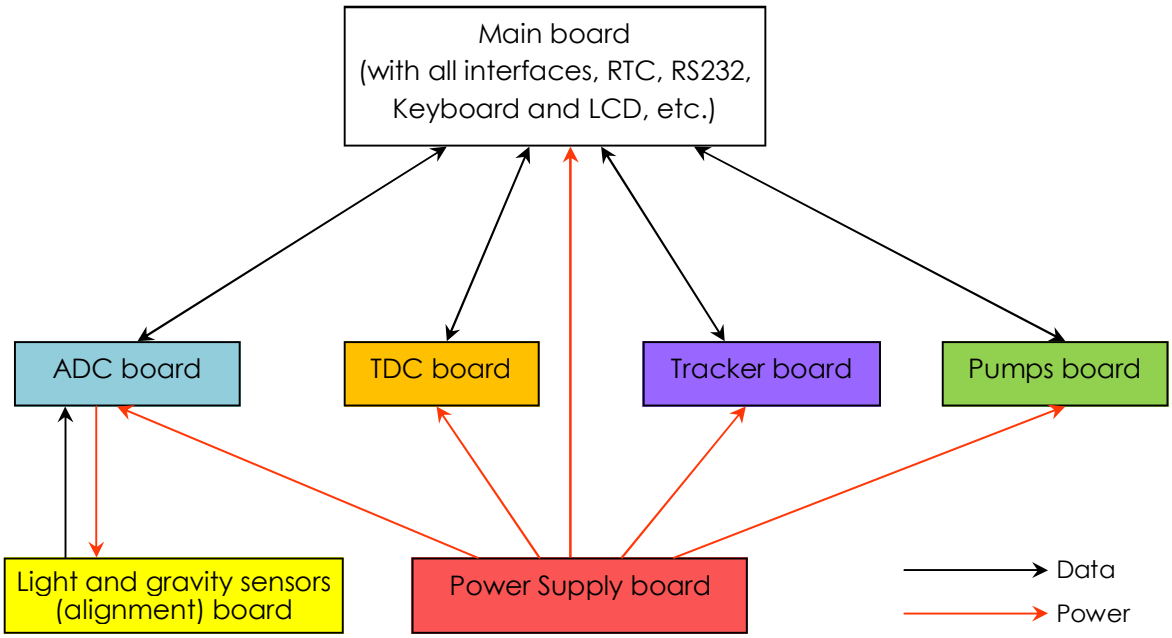


Figure 53 - The Microcontroller system – the pcboards composing it

5.6 Implementation Schedule

It has been decided that, taking into account the overall system complexity and the other tasks to be accomplished, the implementation be firstly focused on the tracking subsystem as well as the basic MCU system functions that are indispensable, namely the ADC and TDC data acquisition and logging.

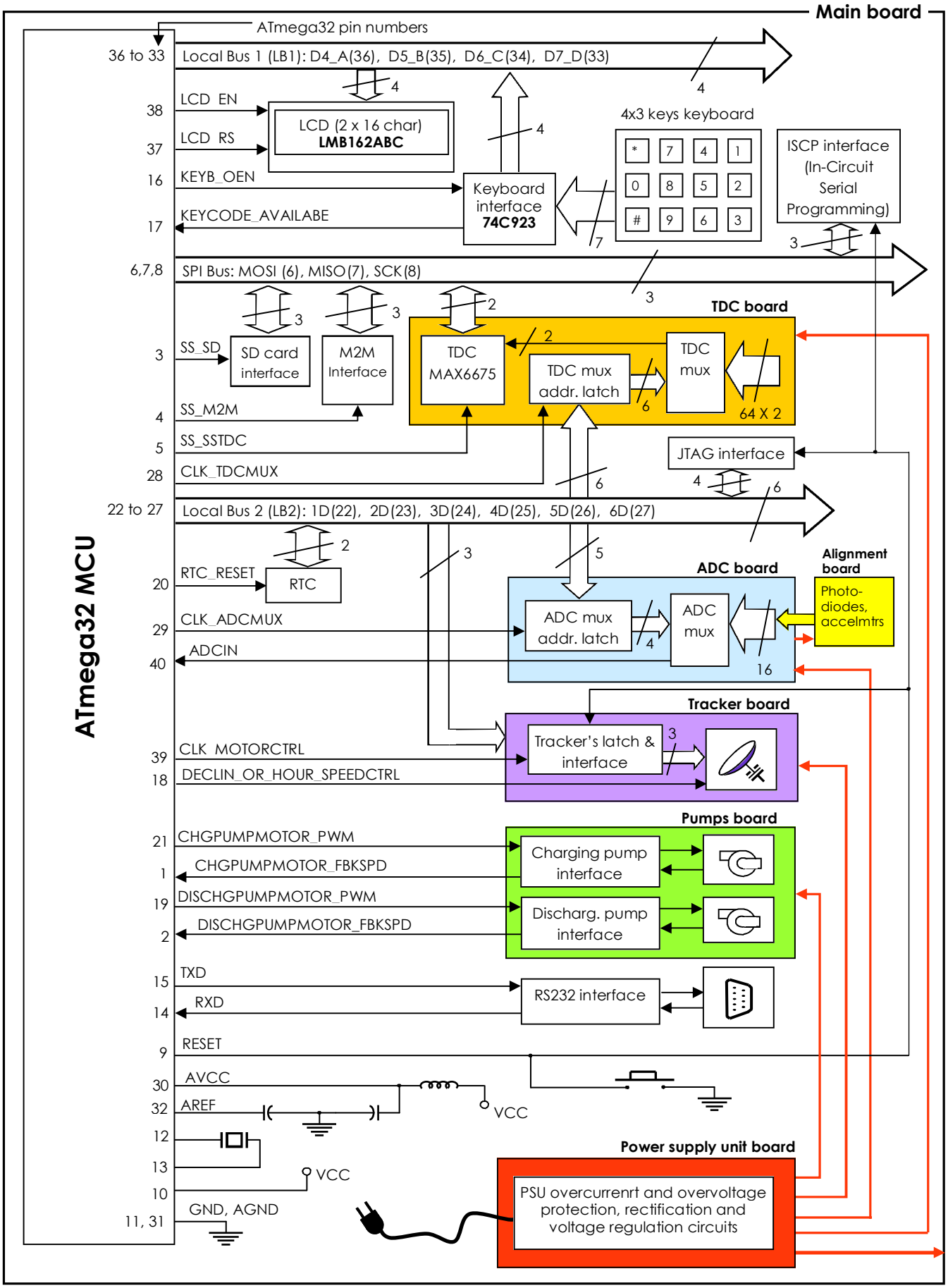


Figure 54 – Final, ATmega32 based, SolarTech controller system architecture

5.7 Circuit diagrams

We now discuss the circuit diagrams of the various subsystems as given in Figure 53.

5.7.1 Main board circuit design considerations

The circuit diagram for the main board is shown in Figure 55. Referring to Figure 55, the following points can be noted:

The ATmega32 MCU has four 8 bits ports. Each port and port bit has a general purpose, as well as one or more specific purpose functions. This was taken into account in all cases where a specific purpose function was required, namely:

- The pins 5, 6, 7 and 8 are SPI bus pins. They were used as such.
- The *ADCin* pin (for AD conversion), the RXD and TXD pins (for RS232), the pins OC2, OC1A and OC1B (for sun tracker and pump motor PWM speed control), the INT1 pin (for keyboard interrupt request).

It is worth noting that there are miscellaneous purpose specific power and timing pins (*Vcc, AVcc, ARef, Gnd, AGnd, Reset, Xtal1* and *Xtal2*). All other port pins were used as general purpose pins.

We now draw brief considerations about different devices and/or interfaces that make up the main board:

- A MAX232 line driver/receiver has been used for RS232 interfacing;
- The keyboard is a pair of a 4 lines x 3 columns keypad and a 74C923 encoder. The 74C923 raises an interrupt request line (KEYCODE_AVAILABE) as soon as a keystroke is detected. The MCU lowers a KEYB_OEN (keyboard output enable) line and then it reads the 4 bits wide scan code from the 74C923. These 4 bits are the ones shared with the LCD on local bus 1 (LB1), as discussed earlier.
- The LCD is interfaced to the MCU in 4 bits mode using the above mentioned LB1 for data bits and LCD_EN (enable) and LCD_RS (register select) as control bits, whilst the LCD_RW (read/write) is tied to GND.
- The Dallas-MAXIM DS1302 was chosen as real time clock. It is powered by both the 5V system supply and a non rechargeable 3.6V coin cell battery backup, capable of providing several years of failsafe timekeeping in the absence of the normal 5V source.
- A latch (the 74HC273N) is used for storing the state variables for the sun tracker. The 74HC273N clear input is tied to the system RESET to ensure that the tracker motors are in the stopped-mode at start up, as a failsafe security measure.
- For system board ambient temperature monitoring, a LM35 is used. As it normally requires a differential ADC input, its ground reference was shifted up by about 2.4V to enable its interface with the MCU's ADC that is being used in single ended configuration.
- The SD card interface is done via the SPI bus. As the SD cards use 3V supply and signalling levels, a LP2950 regulator is used to provide it with 3.3V supply. Voltage dividers are used for 5V to 3V level adaptation;
- Several connectors are provided for linking to the different daughter boards, namely: the ADC board, the TDC board, the tracker's board, the pumps

board and the power supply board.

Other circuits details can be found on the board's schematics diagram (Figure 55), as well as relevant datasheets may also be referenced. The components layout and printed circuit board layout can be seen in Figure 56 and Figure 57, respectively.

5.7.1.1 Main board's circuit schematics, components and pcb layouts

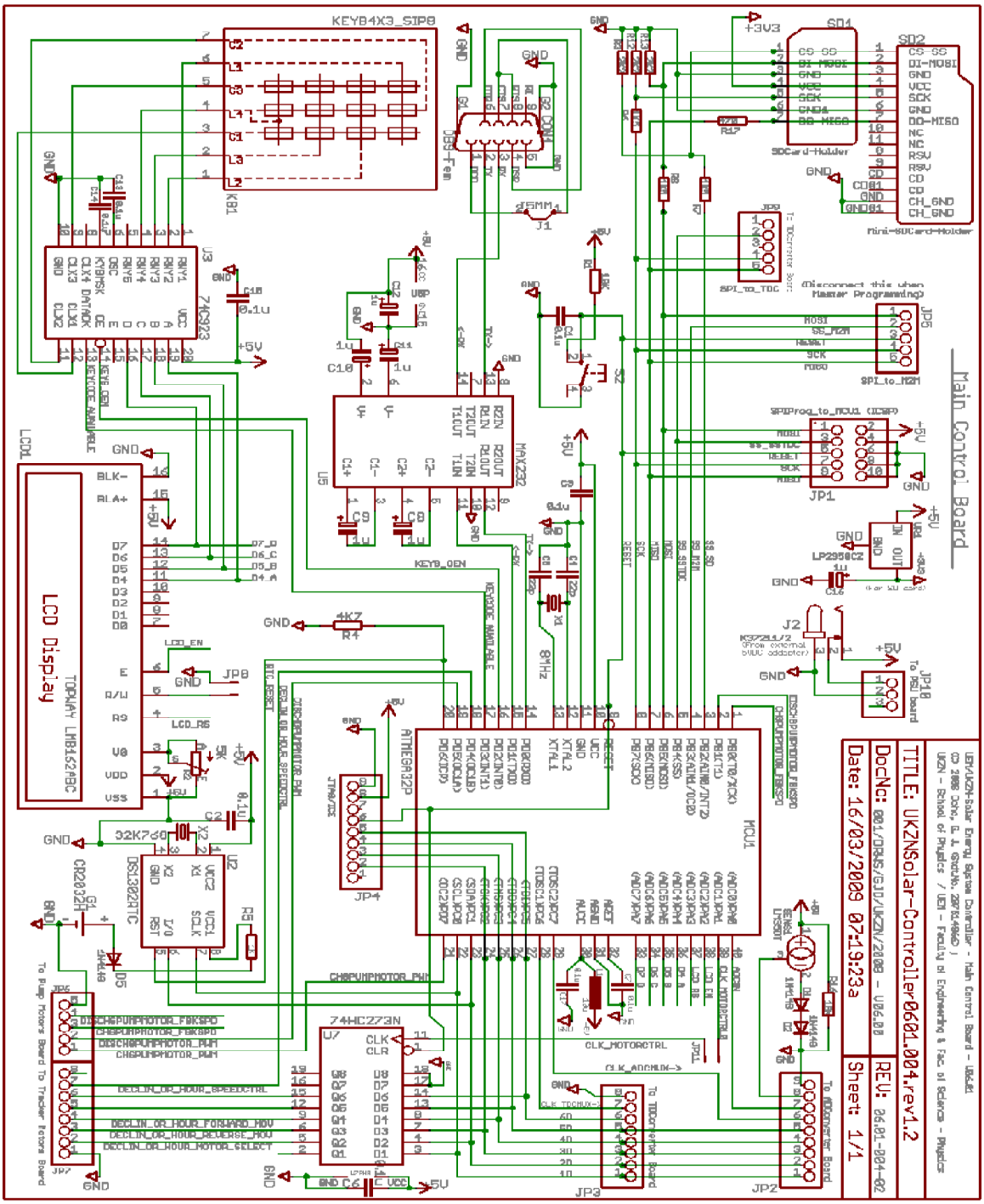


Figure 55 - Main board's circuit schematics

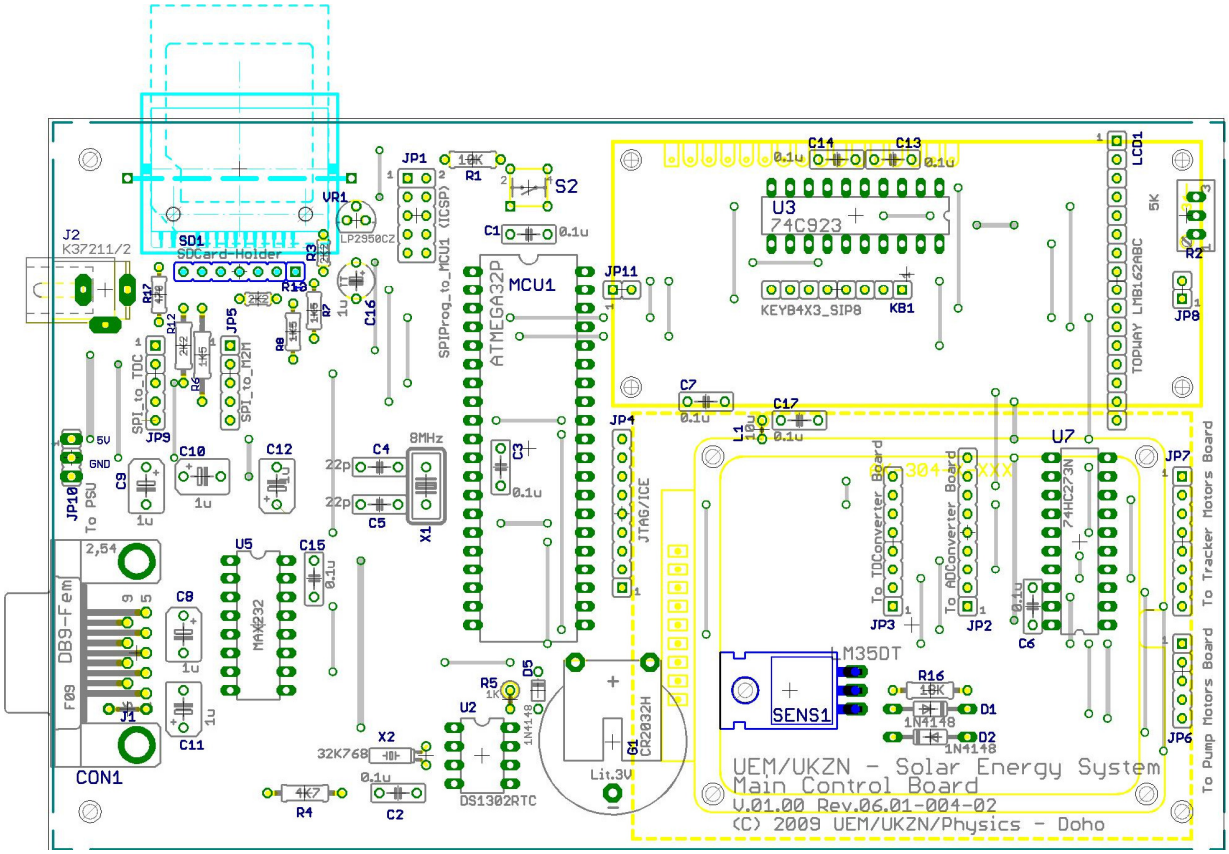


Figure 56 - Main board's components layout

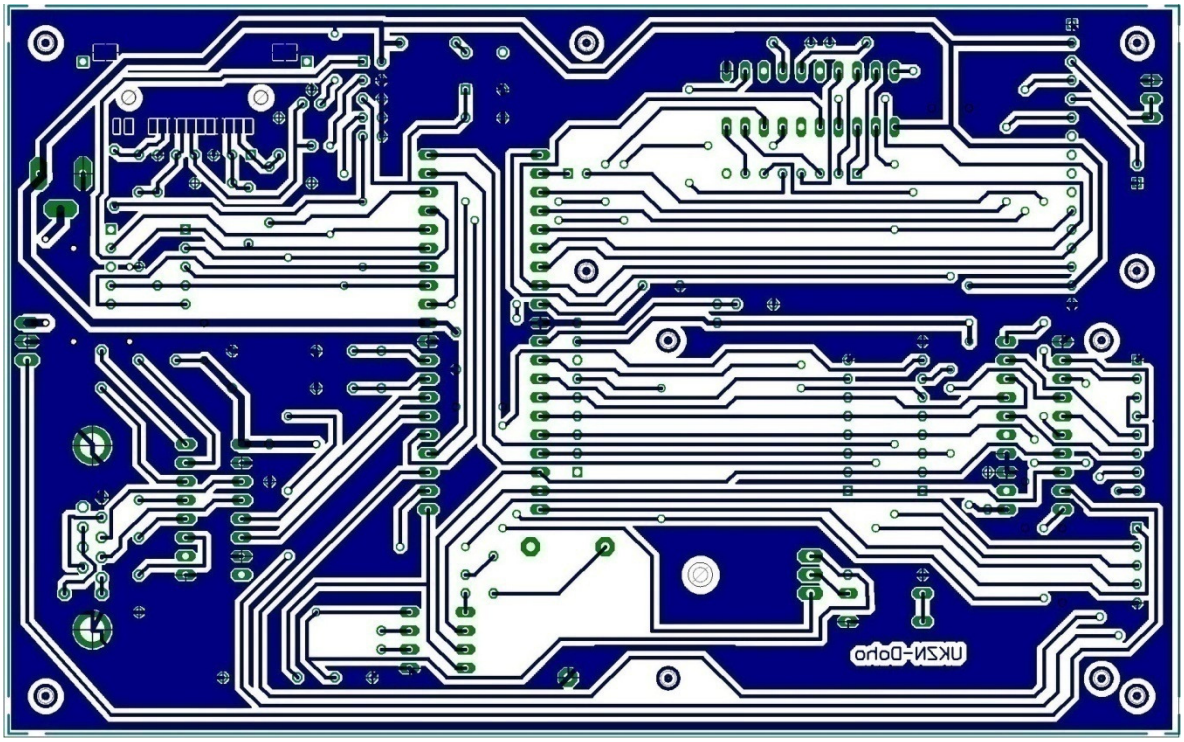


Figure 57 - Main board's PCB layout

5.7.2 ADC board circuit design considerations

The circuit diagrams for the ADC board are shown in Figure 58 and Figure 59. Concerning the Figure 59, the following points are noted:

- The circuit schematics is based on the multiplexed AD conversion interface discussed earlier, whose conceptual design is depicted in Figure 49;
- A 16 to 1 channel analogue multiplexer DG406 is used, paired with a 74HC374N 8 bits register for address latching. A NOT gate (logic inverter) made of a transistor is used for decoding the 5th address line. This allows for the future addition of the upper page of another 16 channels;
- An onboard temperature sensor (thermistor) is used as alternative temperature monitoring device to the main board's LM35 sensor, although it shares the same channel as the user's heat utilization set point temperature potentiometer (whose target board was not scheduled for this stage implementation);
- Several connectors are provided for linking to the target sensors / boards, namely: the hour and declination axes angle to voltage converter potentiometers (fitted on the tracker axes), the light (photodiodes) and gravity (accelerometers) sensors (from the alignment board), the wind sensors (target interface not implemented) and the user heat utilization set point temperature potentiometer (target interface not implemented);
- Other connectors link to either the PSU or the main board.

Other circuit's details can be found on the board's circuit schematics (Figure 59), as well as in relevant datasheets. The ADC board's components layout and the schematics are presented in Figure 58 and Figure 59 respectively.

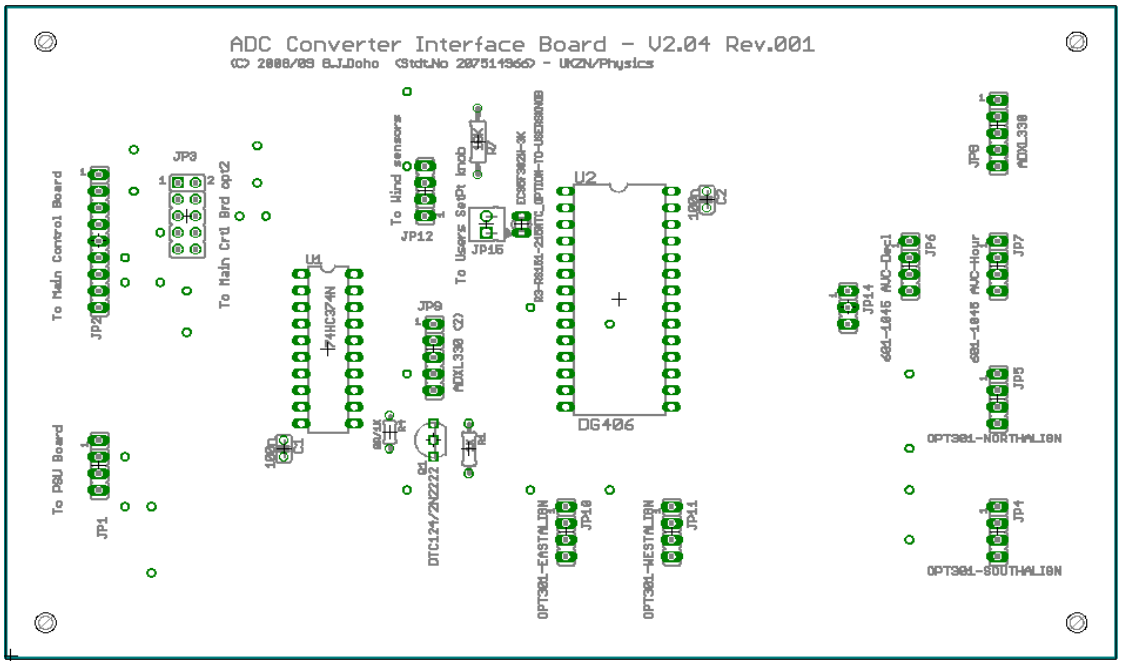


Figure 58 - ADC board's components layout

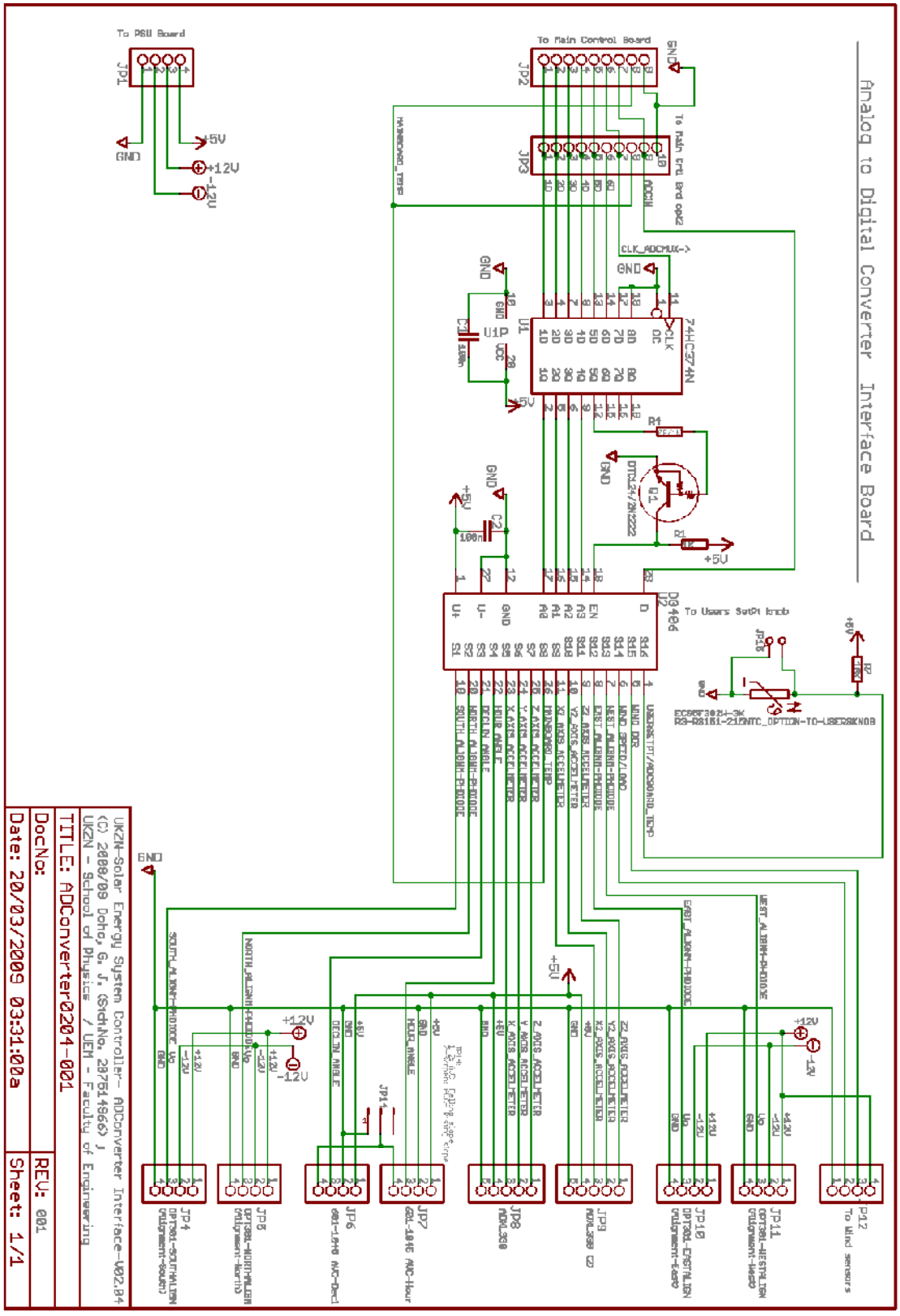


Figure 59 - ADC board's circuit schematics

5.7.3. Alignment board circuit design considerations

The circuit diagrams for the alignment (gravity and light sensors) board are shown in Figure 60 and Figure 61. With respect to Figure 61, the following points are noted:

- The alignment (light and gravity sensors) board, houses 4 photodiodes (OPT301M) physically arranged in a way to provide fine alignment to the sun tracking dish. One pair is for the hour axis (East-West) fine alignment, while the other pair is for the declination axis (South - North) fine alignment. When the dish is aligned in respect to both axes the four photodiodes are equally illuminated, yielding the same output to the AD converter. Once the dish gets misaligned in some of the 2 directions of motion, the corresponding pair of photodiodes is unequally illuminated: one photodiode is getting shadowed while the other one is getting more illuminated. So, they yield unbalanced outputs to the AD converter. This information is feedback to the tracking controller to correct the dish position angular accordingly;

- The gravitic sensors are two ADXL330 triple axis accelerometers/inclinometers. They are intended to provide coarse alignment to the dish collector as an alternative to the angle to voltage potentiometers tied to the tracking axes. Moreover, being 3-axis accelerometers, besides giving the actual dish's 3D position in the space, they can as well give acceleration and vibration information, very useful for safety purposes. Acceleration and vibration outside the safety operating area may be caused by wind or some system malfunction.

The alignment board's components layout and the schematics are presented below, in Figure 60 and Figure 61 respectively.

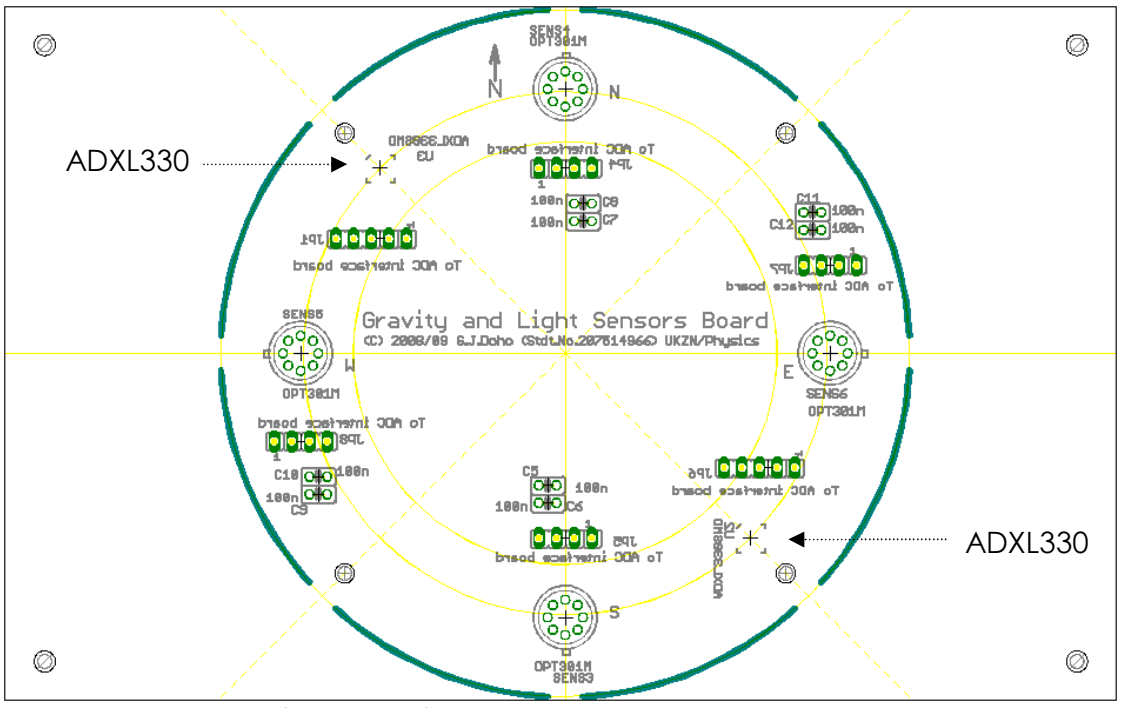


Figure 60 - Alignment board's components layout

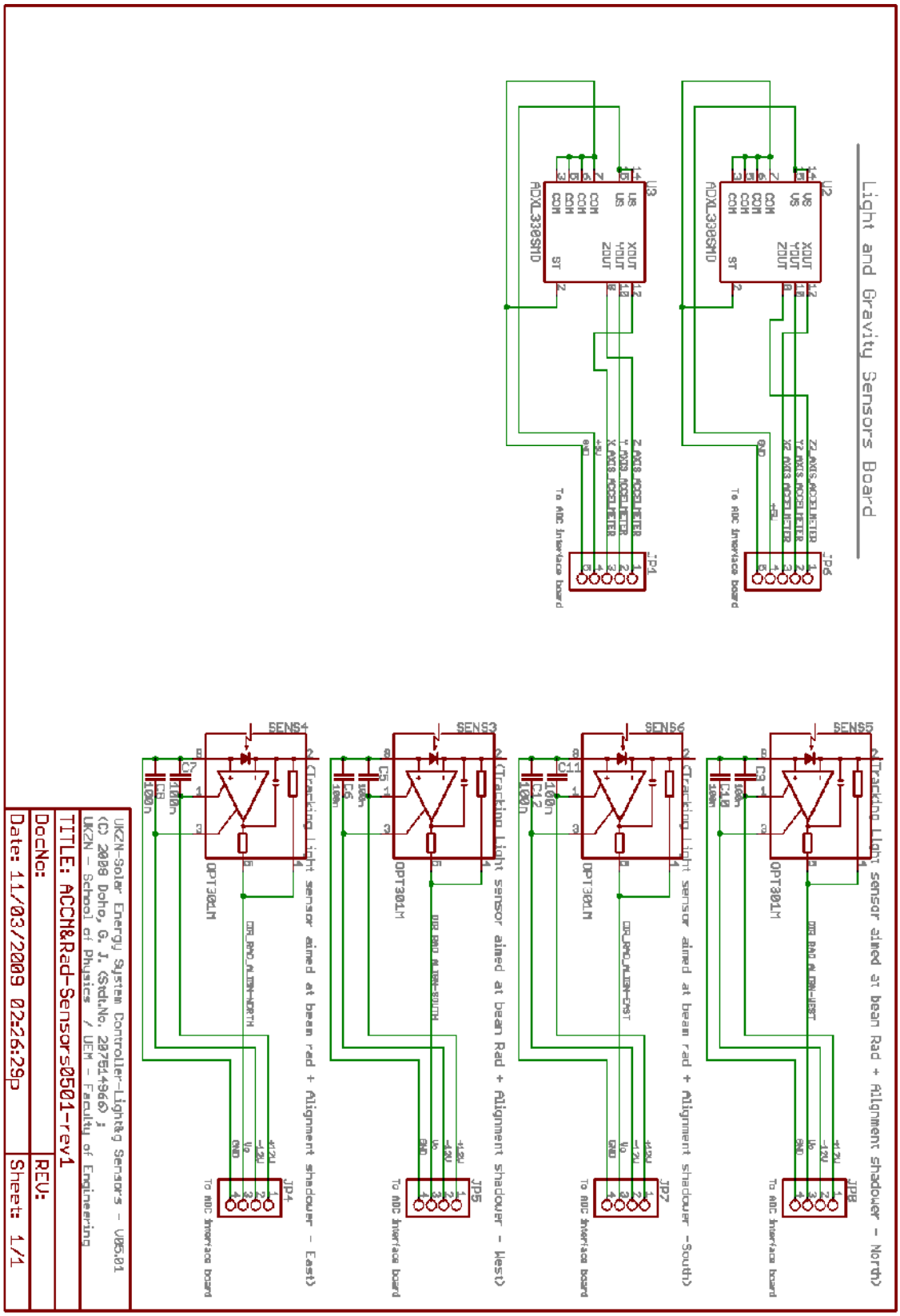


Figure 61 - Alignment board's circuit schematics

5.7.4 TDC board circuit design considerations

The circuit diagrams for the thermocouple to digital conversion (TDC) board are shown in Figure 62 and Figure 63. With respect to the schematics in Figure 63, the following points are noted:

A MAX6675 cold junction compensated K type thermocouple to digital converter (TDC) is used. This TDC has 12 bits, 0.25 ^{0}C/LSB resolution, being able to read temperatures up to 1023.75^{0}C. It has a thermocouple accuracy of about 8LSBs (2^{0}C) for temperatures ranging from 0 to 700^{0}C. The conversion time is about 220 milliseconds which yields about 4Hz sampling rate. All these characteristics are roughly acceptable for the current application. In fact higher performing ADCs (like the AD7793 sigma-delta ADC) were considered before we eventually decided for the MAX6675, aiming at cutting complexity for both the hardware and the software development.

The TDC and other components are interfaced according to the discussion of earlier sections (see also Figure 50). A DG407, an 8 to 1 differential channel analogue multiplexer is used. On the other hand, as 64 channels are required, a 3 to 8 decoder (74HC238N) is used for providing device selection for the required 8 DG407 multiplexers, whilst a 74HC374N is used to store the multiplexer address received from the main board.

Different connectors are used for providing connections to the thermocouples, the main board and the power supply.

Further details can be found on the circuit schematics and components layout in the figures Figure 62 and Figure 63, which follow below, as well as relevant datasheets may also be referenced in the appendix D.

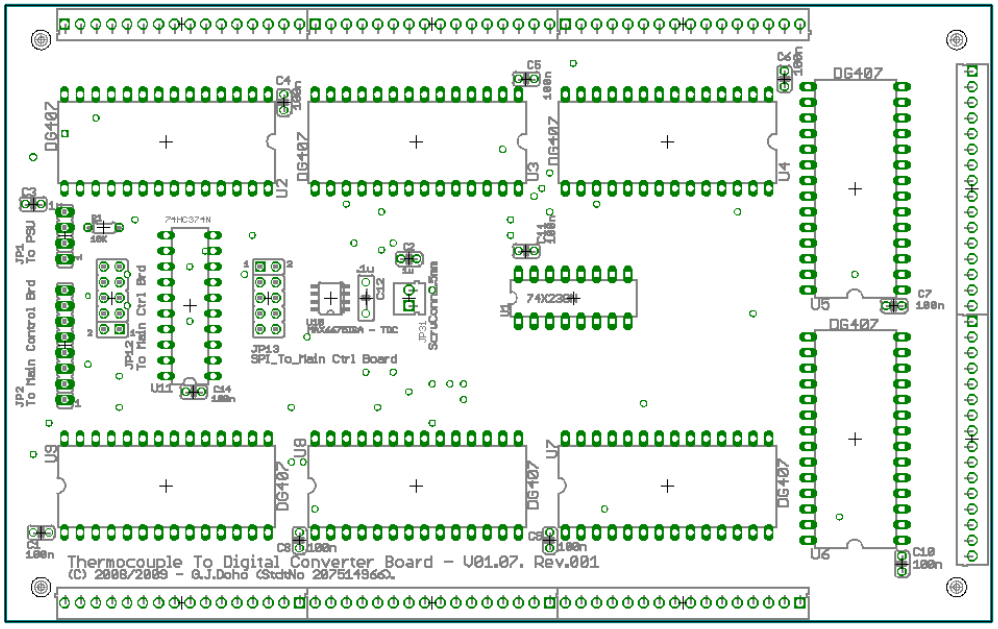


Figure 62 - TDC board's components layout

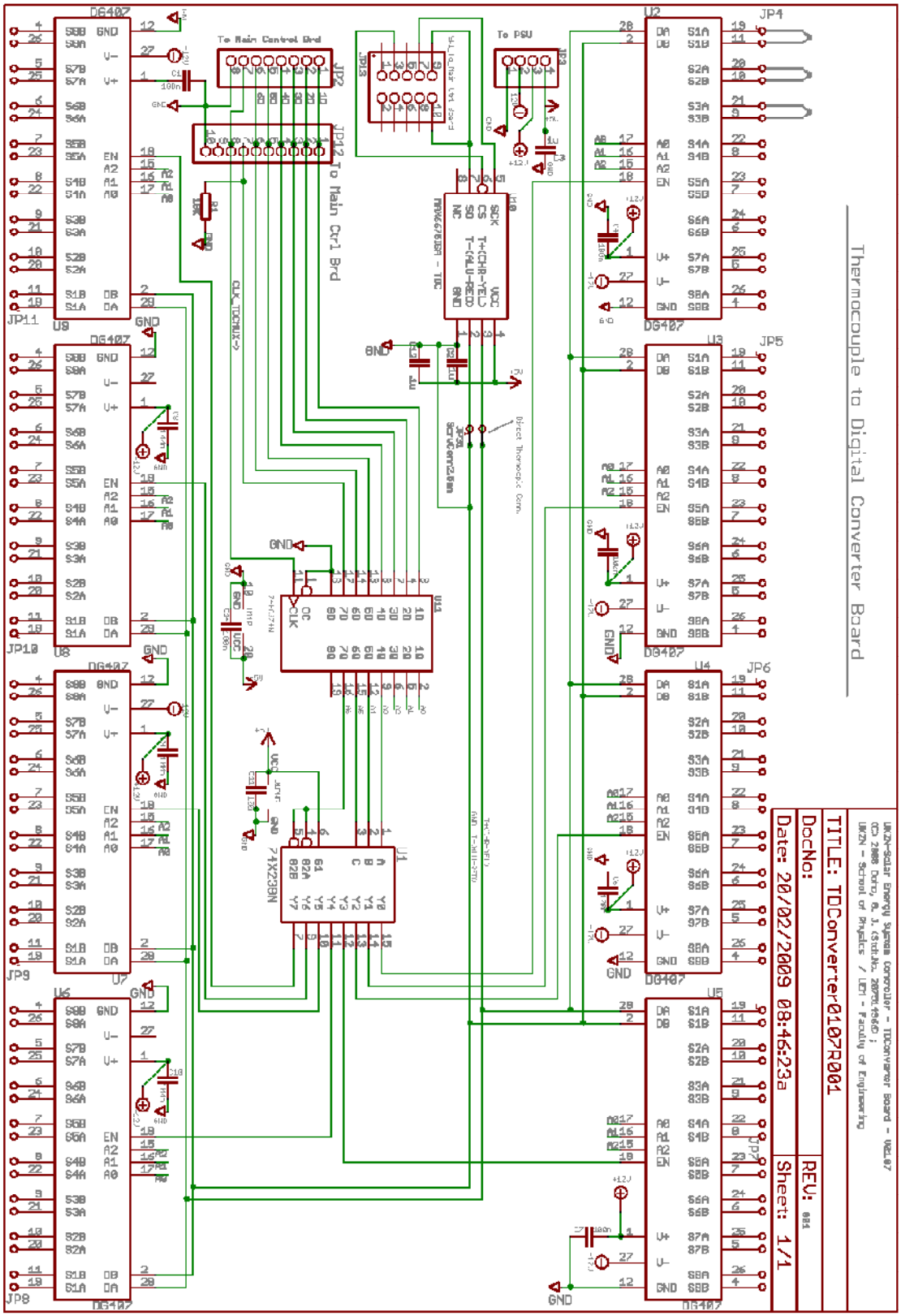


Figure 63 - TDC board's circuit schematics

5.7.5 Tracker's board circuit design considerations

The circuit diagrams for the sun tracker's interface board are presented in Figure 64 and Figure 65.

Two hardware implementations were considered: (i) A relay based design (the one presented here) and (ii) a dual H-bridge connected power mosfet based design. The first one was implemented successfully. The 2nd one, although a more attractive design, did not behave as expected due mainly to lack of the right components. Eventually, time constraints forced this approach to be set aside.

The design presented here is based on the discussions in earlier sections, particularly section 4.3.1. It is worth noting that this circuit can be used as the actuator part for all control strategies represented in Figure 41.

With reference to Figure 65, the following points are noted:

- Low cost DPDT relays and low cost general purpose transistor relay drivers were used for implementing the actuator part of the FSM 4-directional controller discussed in earlier sections;
- The relays are used for implementing the selective on/off control functions of Fwd, Rev and MSel variables (as described in table 4.1 and later sections), whilst the PID or FUZZY-PID speed control functionality is provided by a PWM scheme that varies the supply to the selected DC motor thereby controlling its speed;
- For providing feedback of the actual hour and declination angles, a precision single turn potentiometer was attached to the shaft of each of the 2 axes. To interface it to the ADC we electrically connected the potentiometer to behave as an angle to voltage converter (0-290^0 to 0-5V).

Further details can be found in the schematics (Figure 65).

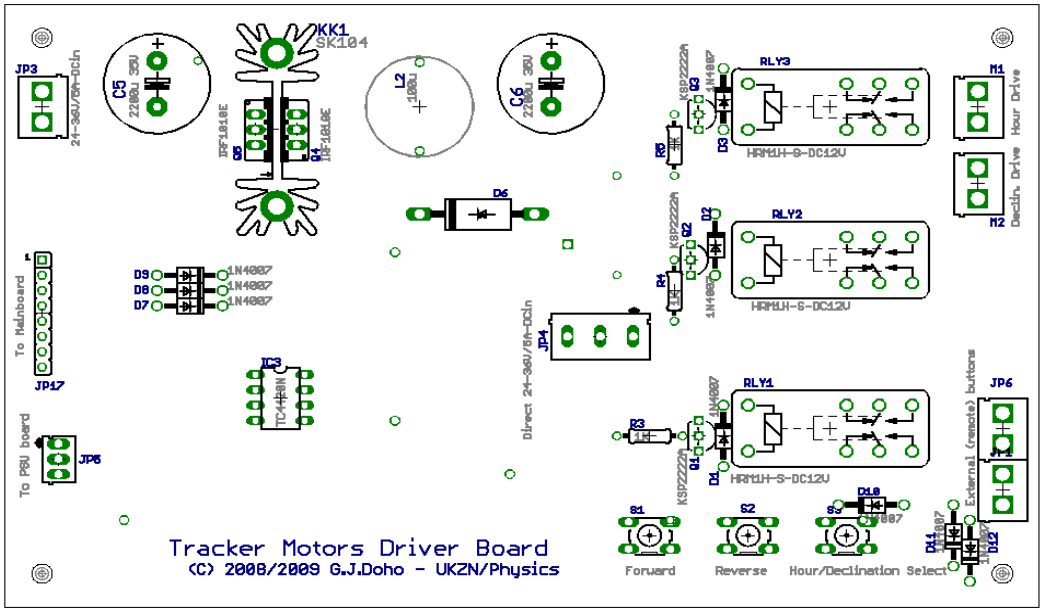


Figure 64 - Tracker's board components layout

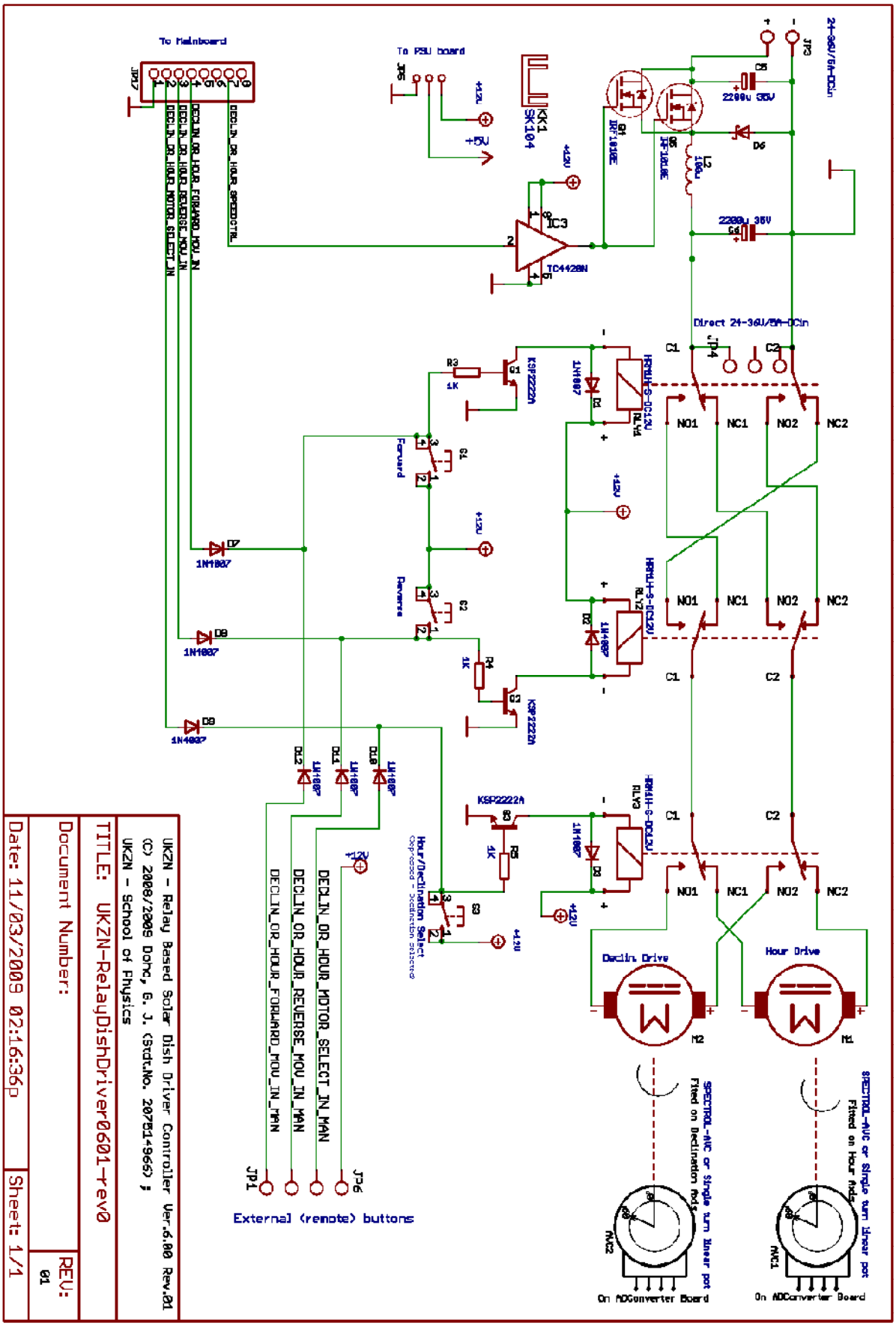


Figure 65 - Tracker's board circuit schematics

5.7.6 Charging and discharging pumps circuit design considerations

The circuit diagrams for the pumps' interface board are presented in Figure 66 and Figure 67. With respect to Figure 67, the following points are noted:

The circuit implements the controller(s) discussed in section 4.3.2 and depicted in Figure 46. Two separate circuits are provided to allow the pumps to work independently, as discharging process may be concurrent to the charging.

As with the tracker motors speed control, a PWM based scheme is used for providing the desired pump motor speed control as can be observed in detail on the circuit schematics (Figure 67).

For pump motor actual speed feedback to the system, an optocoupler based scheme is provided, where each pump motor revolution will trigger a fixed number of (n) impulses on the required MCU port pin. Such pulses are produced by a slotted disc rotating with the pump motor shaft, together with a light beam. Each time the slot passes the light beam, light falls on a photo-detector which generates an electrical pulse. The number n of pulses is the number of slots on the disc. The pulses collected on the MCU port pin can then be used to recover the pump rotating speed and thereby other system variables.

As an alternative to the above described optocoupler based speed feedback, a reed switch scheme is also provided, where a magneto is tied to the pump motor shaft which will toggle the reed switch for each shaft's revolution thus triggering a pulse on the desired MCU port pin. In this case the number of pulses counted during a specified time slot is the number of revolutions per such time slot. This can be used for deriving other variables.

Shown below in Figure 66 and Figure 67, are the pump board's components layout electrical schematic, respectively.

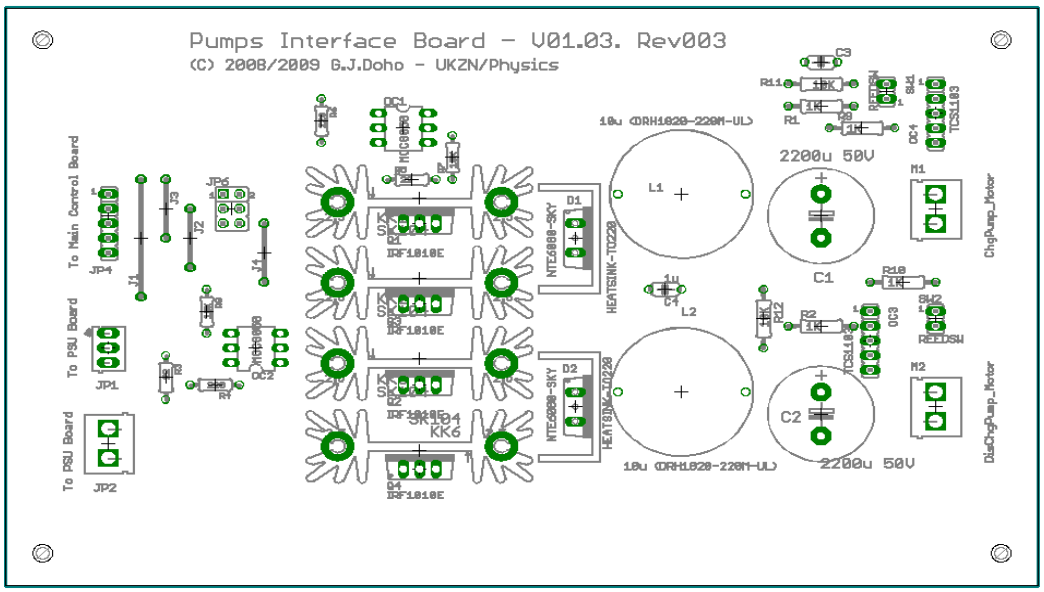


Figure 66 - Pumps' board components layout

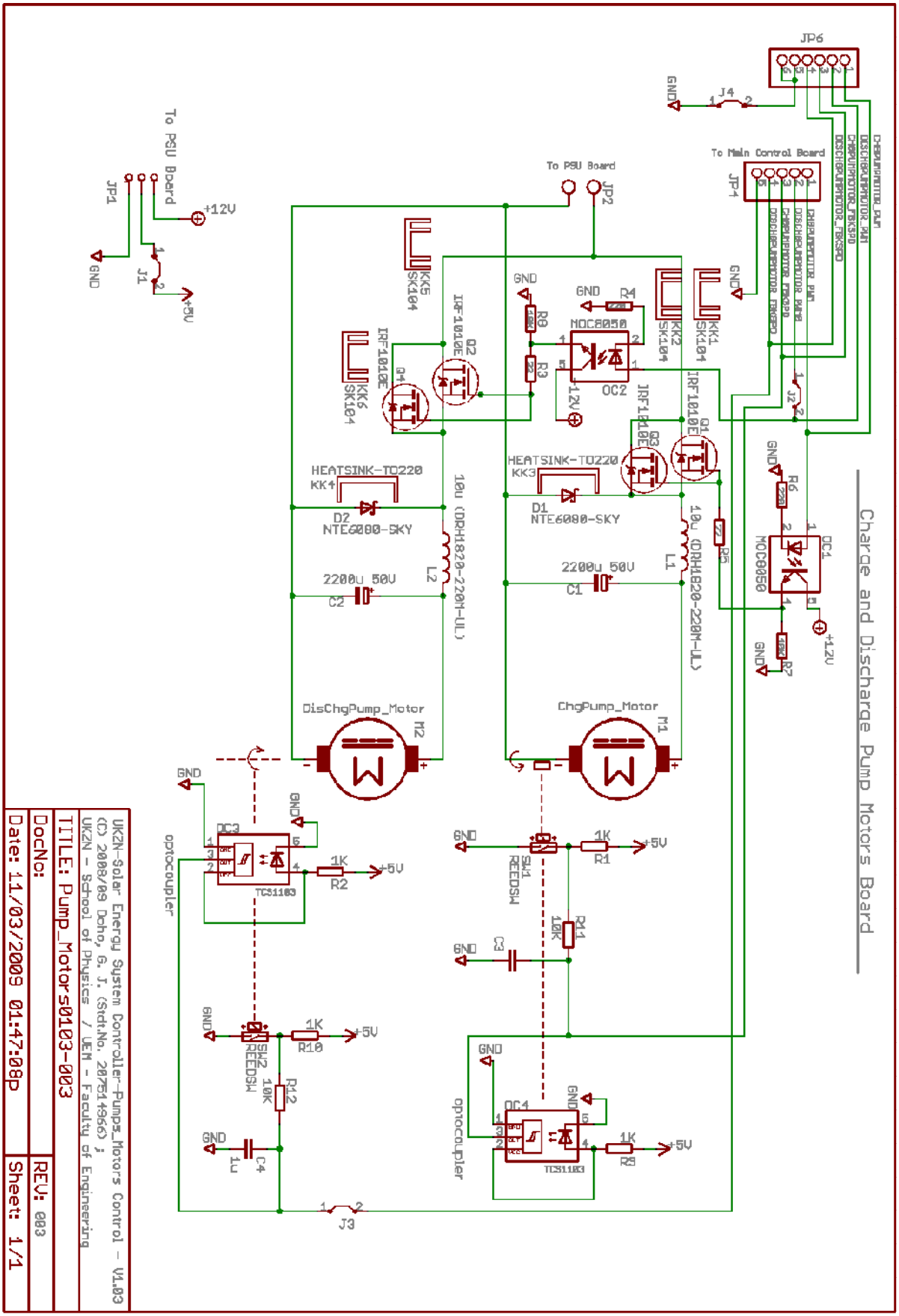


Figure 67 - Pumps' board circuit schematis

5.7.7 Power supply unit (PSU) board design considerations.

From Figure 68 through Figure 71 we present the design of the power supply board. The power supply unit is composed of:

(i) An input protection to safeguard the PSU and the system from incoming overvoltage and/or over current conditions. Fuses/thermal cut outs (TCOs), metal oxide varistors (MOVs) and gas arrestors (GDTs) are used;
(ii) Step-down transformer(s) for converting from the line voltage of 220VAC to 18 and 36VAC which are interfaced later to rectifying bridges;
(iii) Bridge rectifiers (full wave type);
(iv) Voltage regulators, for +5V, +12V, -12V and dual +30V. Ordinary voltage regulators are used: the LT1084 for all positive voltages and the LM337 to provide the -12V.

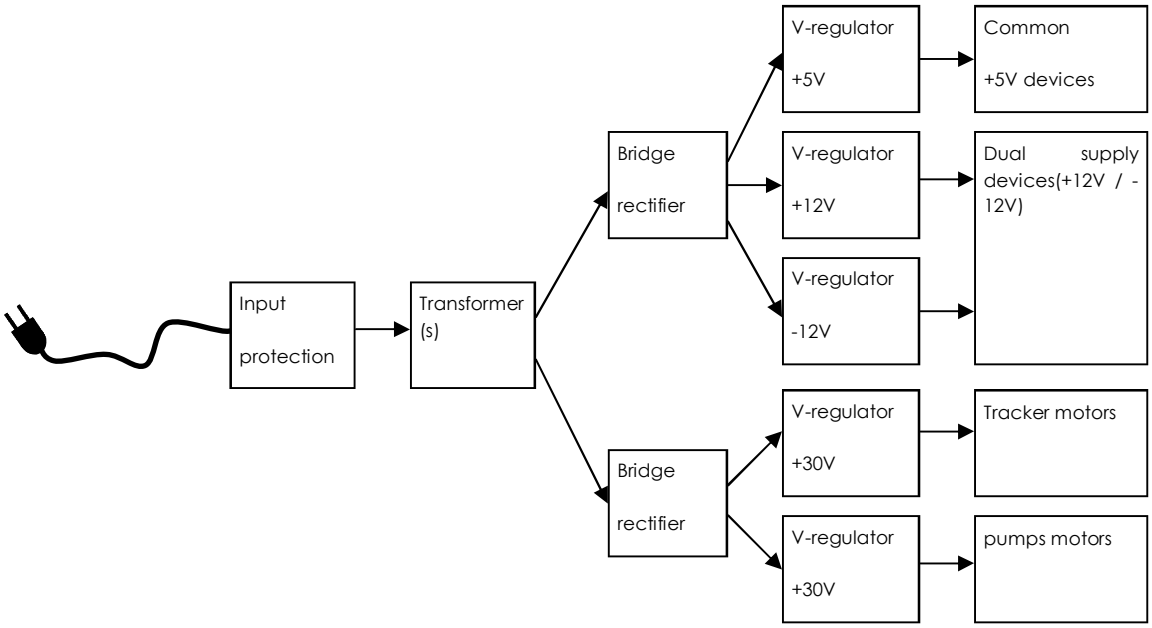


Figure 68 PSU and consumers block diagram

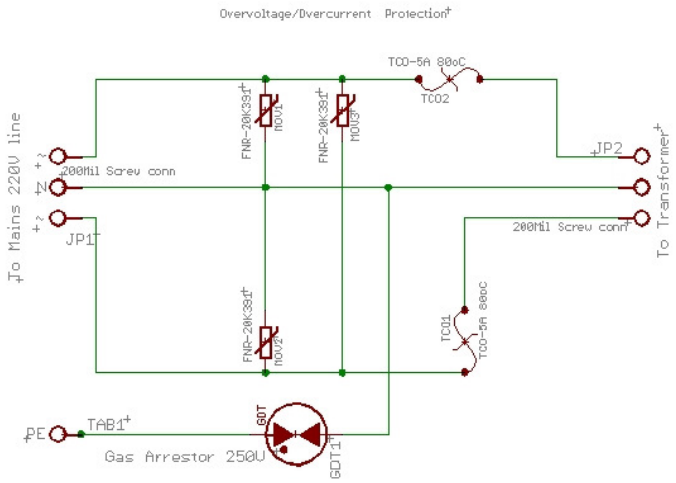


Figure 69 - Schematics of the PSU's protection unit

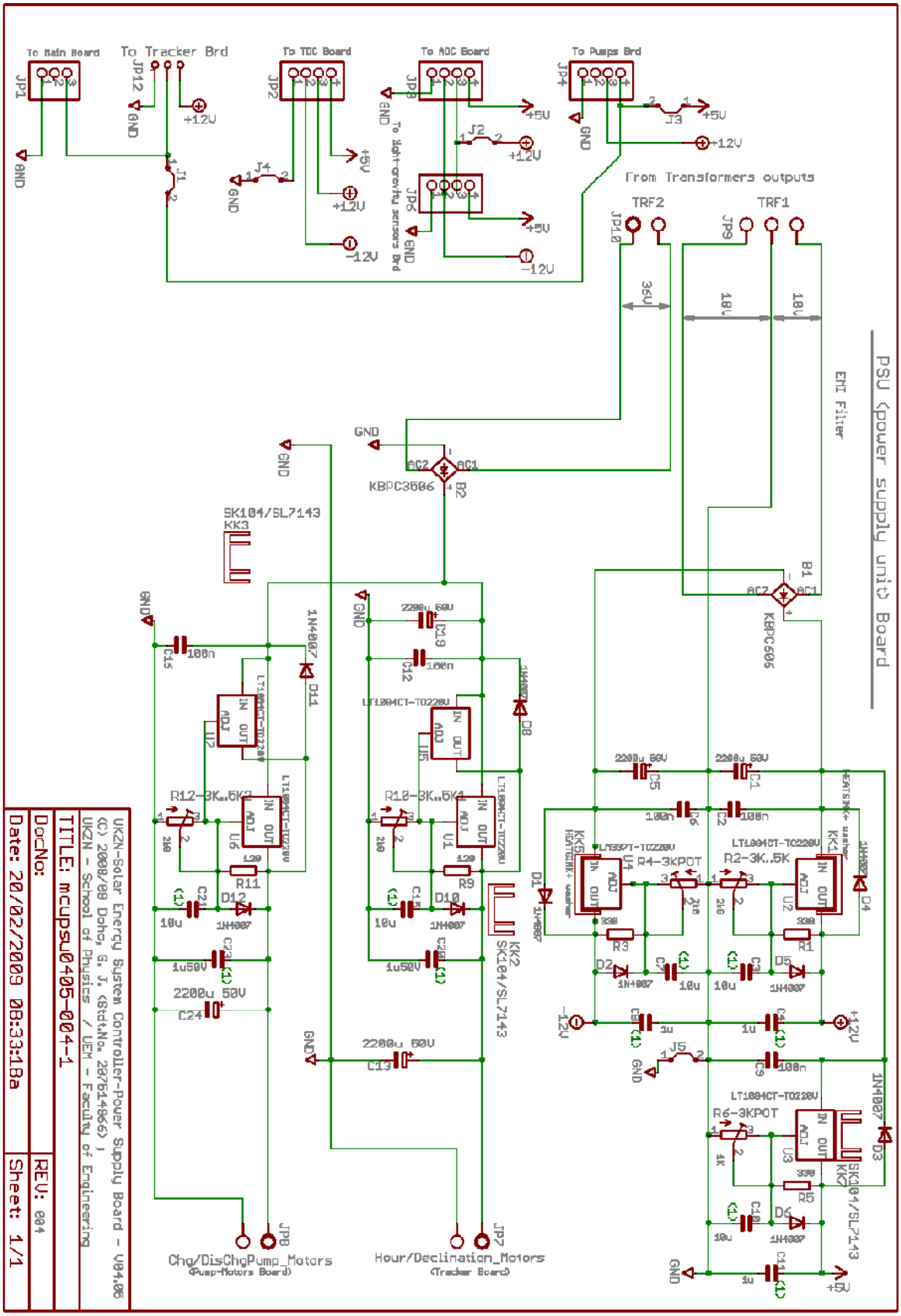


Figure 70 - PSU board's chematics

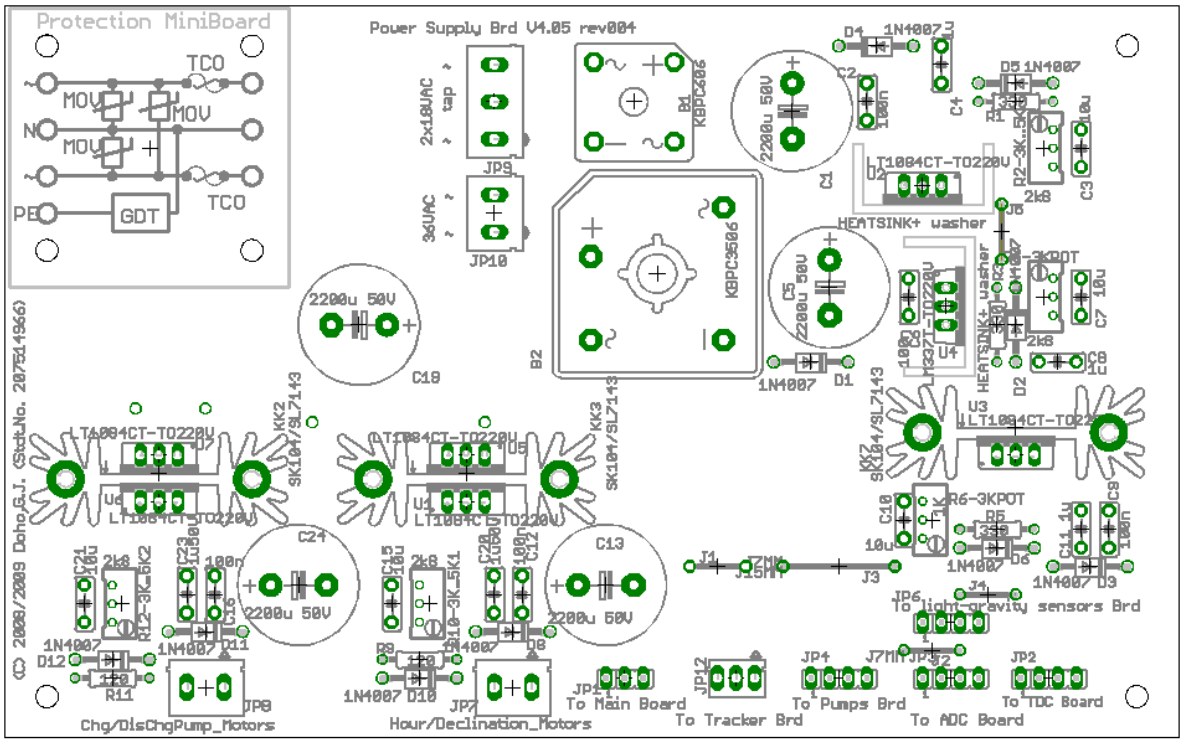


Figure 71 - PSU board's components layout

5.8 Chapter summary

In this chapter we have walked the following steps:

(1) We have designed the architecture of the microcontroller based system main unit, based on the monitoring and control needs discussed in previous chapter.

(2) We have designed the different circuit boards based on that architecture. As results of the boards design process, we produced using a CAD/CAE/CAM package, the schematics, components and printed circuit boards layouts, as well as Gerber (computer aided manufacturing) files which were sent to a PCB manufacturer where were produced. We afterwards assembled the prototype boards (with the relevant components). Find in appendix E some sample boards photographs.

In next chapter we address the design and implementation of the real time operating program, which is the logical component of the system designed in this chapter.

Chapter VI – The real time monitoring and control program (ST-RTOP)

In this chapter we present the design of the microcontroller program, which is a sort of operating system at the embedded microcontroller scale. We firstly discuss the generic characteristics and architecture of the program. Then we address the design and implementation of specific software components, from basic input-output and system timing, to the software implementation of the controllers discussed in chapter IV. Closing this chapter, the PC side data logging program is also described.

6.1. Real time control program generic characteristics

This control program (named abreviatedly ST-RTOP[13] for simplicity) embodies the basic functionalities and features of a generic real time operating system (RTOS) for embedded devices.

The main features of a real time operating system are generically the capability of responding to external events and correctly perform the desired tasks within finite and specified times or deadlines. More specifically, such capabilities may be:

(1) Concurrency: which means multitasking (in real life control problems many tasks need to be executed simultaneously). In multiprocessor / multi-memory (MIMD) environments, real multitasking (or real parallelism) can be performed. However, in a single processor / single memory environment (SISD) as with the present case, concurrency has to be achieved by convenient scheme for sharing processor time and other resources among the scheduled tasks, insuring that within a satisfactorily short period of time all tasks should have run their specific time slot.

Providing and managing concurrency, may involve the following concepts:

a. task scheduling, task triggering, task suspending and preemption;
b. resources sharing, mutual exclusion, intertask communication, etc.

(2) Within the general arena of task concurrency, insure the periodic execution of time-critical tasks (the real time task scheduling or triggering component, is responsible of timedly firing periodic tasks, either directly or indirectly).

(3) Have a provision for detecting and servicing aperiodic / sporadic events: like human generated (like: keyboard initiated) events, communication events, etc. These types of tasks are called event triggered activities whilst the tasks described in point 2 above, are time triggered activities.

(4) Provide and control the orderly use of shared resources, like: processor time (as suggested above), memory, input and output devices, buses and any other specific resources.

[13] ST-RTOP, abbreviated name for the real-time operating program. See also acronyms on page xxii.

(5) Other capabilities or characteristics.

6.2. ST-RTOP's characteristics

The present real time operating program has the following specific characteristics:

(a) It has a task scheduler/triggerer component, the timer0 interrupt service routine (see also page 141 of appendix A) that directly triggers the execution of some of the periodic activities, and indirectly triggers the execution of other periodic activities, through the use of task triggering flags. The flags are used by a background loop (the main loop) to execute the activities. The use of flags in this way is a form of intertask communication, coordination and synchronization.
(b) It has components for servicing aperiodic, event driven activities, like the keyboard interrupt service routine and complementary keystroke processing routines (see page 142 of appendix A);
(c) Various tasks are executed simultaneously, with strict respect to task deadlines, with the help of different specific purpose hardware components (like the TDC, the ADC, the PWM generating timers, the RTC, etc.).
(d) It has a simple but very efficient semaphore system, used to control and coordinate the orderly use of shared buses and other resources.

From the above, it can be observed that, this ST-RTOP has all the roles of a real time operating system (RTOS), although not tailored and featured in the conventional way that the standard RTOSs are.

6.3. The ST-RTOP as a control framework

The real time operating program (ST-RTOP), on the one hand is the basic operating system for the microcontroller system and, on the other hand, embodies the software based control strategies discussed in chapter IV. Indeed, the entire ST-RTOP program is tailored to the specific purpose of being a controller for the thermal energy system under study.

To be precise, let us discuss the mapping between what is depicted in Figure 36, to the software modules in the operating program. The ST-RTOP as a whole, is the supervisory controller. The 3 specific controllers (The FSM based tracker controller, the charging pump and the discharging pump controllers) are mapped to specific functions as presented in the table 6.1.

Control framework component	**Software component name**
Supervisory controller	The whole program, specially its main routine
FSM 4-directional On/Off tracker controller	`FSMtrack` (plus the state machine data structure)
Charging pump PID controller	`PIDctrl` (plus the charging pump PID control data structure)
Charging pump PID controller	`PIDctrl` (plus the discharging pump PID control data structure)

Table 6.1 – The mapping between the control framework and the relevant software routines

6.4. ST-RTOP Structural diagram

The main structures that make up the MCU's monitoring and control program are depicted below in Figure 72. This diagram is a guide for all the major software components comprising the ST-RTOP.

The ST-RTOP program has been written in C programming language (AVR Studio/WinAVR/avr-gcc development environment) and in assembler (for avr-gcc) for some time critical hardware interfacing functions (like the real time clock).

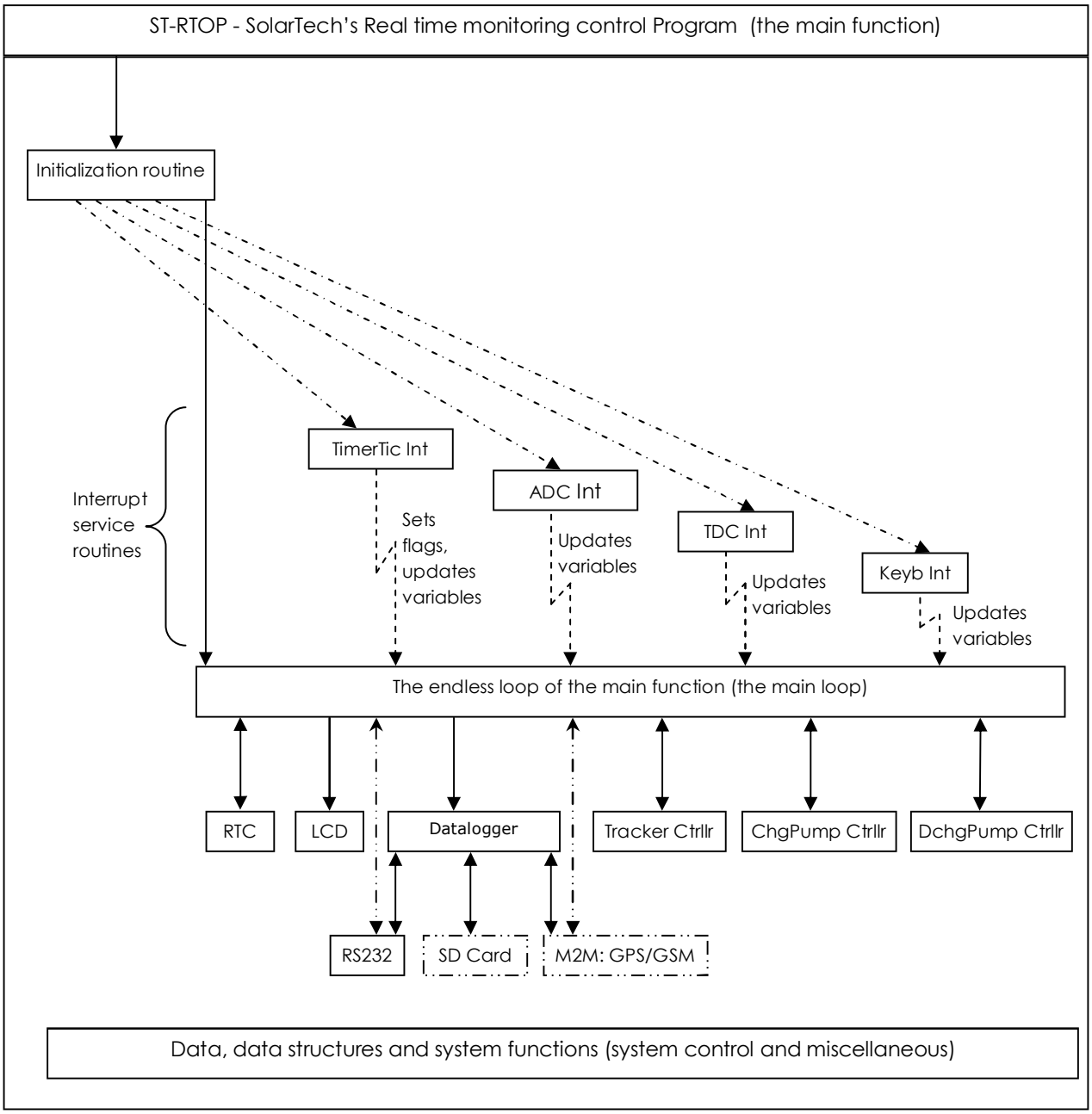


Figure 72 - Architecture of the MCU's real time monitoring and control program

In Figure 72, the double dot - dashed boxes and/or arrows indicate not implemented (no code written) software components and/or interfaces, whilst the dashed or dot-dashed arrows above the main loop indicate non-direct calls.

The corresponding generalized operating flowchart is shown on the Figure 73 below. See appendix A, especially the main() function on page 155 for details. Other generic flow charts, showing the functionality of some of particular subcomponents of the diagram in Figure 72, are also shown in the next sections.

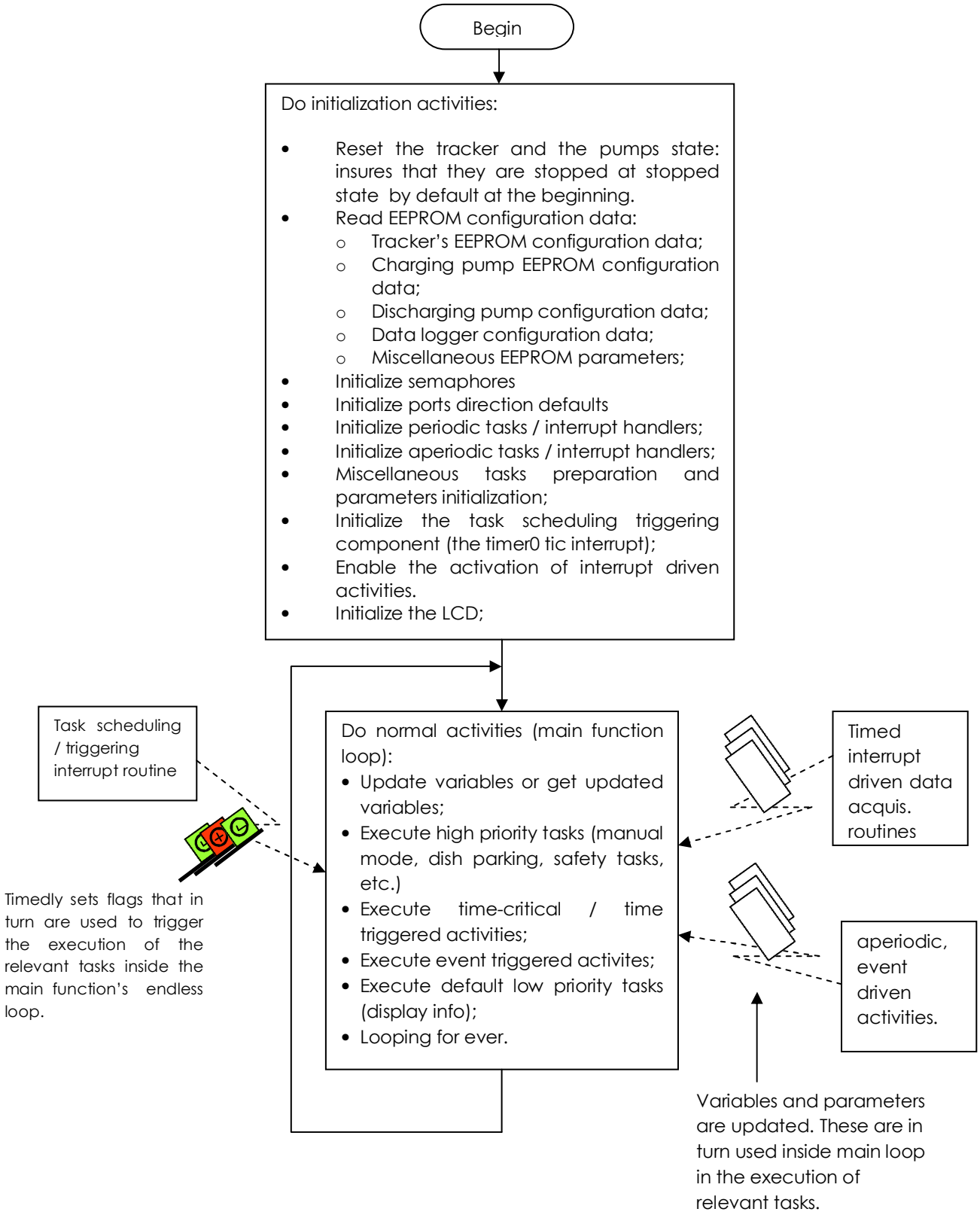


Figure 73 - Generalized flow chart of the Real Time Operating Program (ST-RTOP)

6.5. Description of some of the main software components.

The ST-RTOP program required the creation of many types of routines, ranging from simple data processing functions to hardware interfacing software components. The description of all these functions would be extremely lengthy. So, only the most important and high level components will be explicitly addressed. Anyhow, we recommend the reference of appendices A and B, where we include all the source code.

6.5.1. Description of some system control and timing functions.

The most important functions of the ST-RTOP are certainly the system control and timing routines, since, they insure the timed execution of real time tasks. In this section we describe such functions and related hardware devices or features.

6.5.1.1. The timer tic[14] and system timing and synchronization

The real time clock (RTC) is a time backup device that insures time keeping in case the normal power becomes unavailable. However, its time resolution is only 1 second. This cannot be used for the task triggering and synchronization of some devices (like the TDC, the ADC, etc.) requiring resolutions of milliseconds or lower. So, we solved this shortcoming by configuring a timer tic interrupt of 1 KHz frequency (timer0Tic1k). This *tic*, is conveniently used to synchronously and periodically trigger the execution of time driven events. Such tasks include:

- It increments and maintains a 1000 Hz tic counter (timer0tic1k count) being the main tic counter;
- Based on timer0tic1k, it maintains a 100 Hz tic counter (timer0tic count) used as a master tic for the timing of the TDC, the ADC, and other timed events;

  It is important to note that this timer0tic count is made equal to the total number of seconds (times 100) from the beginning of the year. Also, it is synchronized with the RTC time at program start up and at solar midnight;
- It periodically triggers the TDC start conversion at each 250 milliseconds and the respective digital data readout after 230 milliseconds from start conversion;
- It sets various flags at timed intervals to be used by the main loop for triggering the execution of periodic events;
- It performs polling (at 1KHz freq) of the feedback data bits from the charging and discharging pumps to compute the pump motor's actual rotating speed. The hardware interface for this, was addressed on section 5.7.6 on page 82.

[14] Timer tic is a frequent term in microprocessors, used to name a timer periodic interrupt.

The Figure 74 below, illustrates the timer tic interrupt as a timing axis, and the tasks being triggered at specific intervals of such axis. In the figure, the arrows of the same colour or pattern, represent a certain task being triggered at timed intervals.

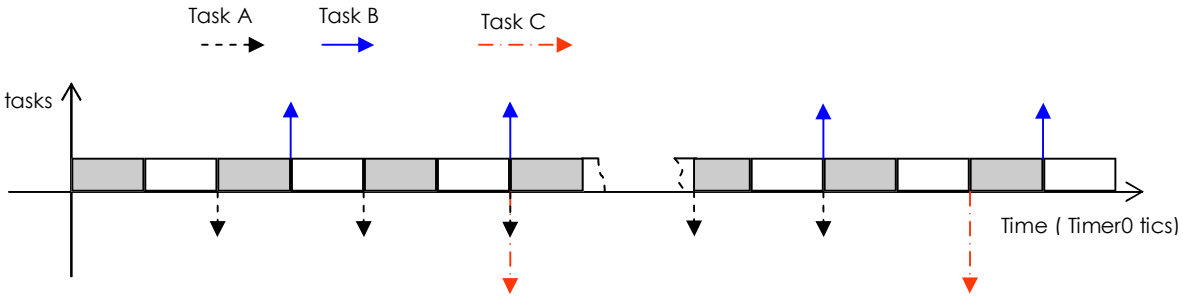


Figure 74 - Conceptual diagram of the system task scheduling, triggering and synchronization provided under the use of the timer 0 tic periodic interrupt.

6.5.1.2. The real time clock (RTC) software interface implementation

As mentioned previously, the RTC is a time backup device that insures time keeping in case the normal power becomes unavailable. Besides being a backup device it plays an active role in the system time awareness, timing and synchronization, namely:

- When the system is restarted or at each solar midnight, the timer0tic is synchronized to the RTC time (see previous section).
- The local time component for the calculation of solar time (and thereby derive other variables) is the RTC time.

The DS1302 RTC uses a 3 wires serial interface with specifications and communications protocol well defined in its datasheet. Based on the datasheet information, a software interface was designed. The low level hardware interfacing functions were written in assembly language. This was to avoid timing race conditions, considering that, the RTC along with its 3-wire serial interface is a time critical device. These low level routines are then called safely by higher level functions, from either C or assembly code.

The software interface is made of hierarchized levels of functions, as follows:

- At the lowest level, `WrByte` and `RdByte` for serially writing or reading a byte respectively to/from the RTC;
- At the intermediate level, `RTC_WriteByteAtAddress` and `RTC_ReadByteAtAddress` for writing or reading a specified byte to/from a specified RTC register address and,
- At also the intermediate level, `WrClockBurst` and `RdClockBurst` for writing or reading to/from the RTC, multiple bytes of data, in burst mode.
- At higher level, there are many functions (like RTC_Reset). This 3rd level is also the common place for any code calling the low or the intermediate level RTC functions. Higher level functions may also access directly the RdByte and WrByte lowest level functions.

The block diagram in the Figure 75 below, illustrates the hierarchy. The code implementation can be seen on pages 175 and 185 on the appendix A. The datasheet can also be referenced for interfacing / protocol details.

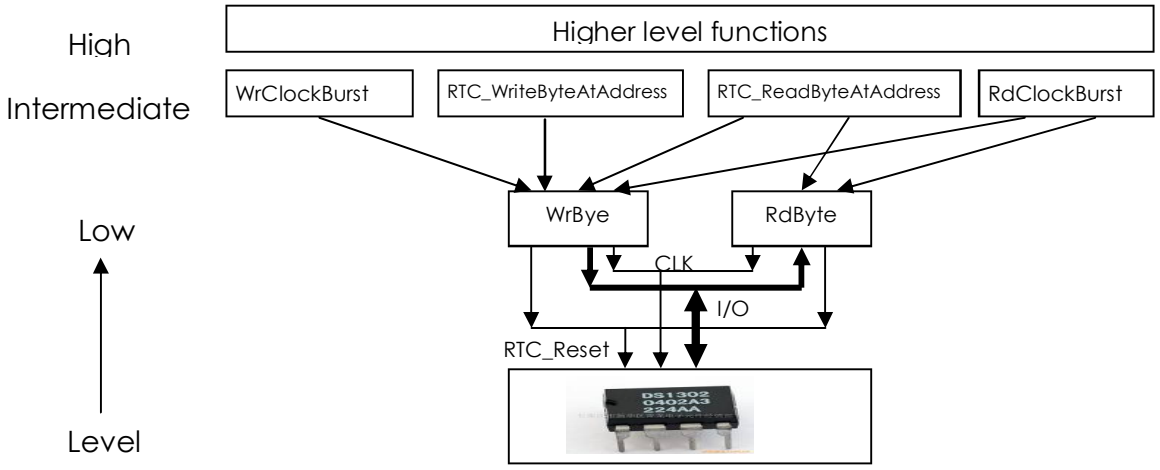


Figure 75 - RTC software interfacing modules hierarchy.

6.5.1.3. The arbitration of the use of shared resources. The semaphores

As mentioned before, the existence of shared resources implies the existence of an arbitration scheme to avoid conflicts. This is the case of the present system where there are 3 shared buses (LB1: on port A, LB2 on port C, SPI on port B. See Figure 54 for details). Other resources were also added to the list of shared resources (see page 189), although not frequently used in a shared fashion.

6.5.1.3.1 The semaphores architecture:

(a) There is a list of shared resources: the list is composed of a number of **S** semaphore IDs, numbered from 1 upwards (semaphore ID 0 is system reserved);

(b) For each shared resource **R** on the list mentioned in (a), there is a list of registered users. This list is composed of the UserIDs for the semaphore R, numbered 1, 2, …n. The lower the ID the higher the priority;

(c) For each semaphore ID there is a semaphore request word, whose length **U** in bits should not be less than the number of user IDs plus 1 (a byte was used in the ST-RTOP's semaphores architecture, since, the number of users for all semaphore IDs is less than 8);

(d) For each semaphore ID there is another flag word, the semaphore owner's ID word, whose length **U** is the same as the semaphore request word. This word records the ID of the user currently busy with the resource;

(e) Each UserID owns a bit in the semaphore flag word. The bit position is equal to the userID itself which equals to its priority in the use of the relevant resource;

(f) The bit 0 of the semaphore flag word is the busy flag: when 0, resource is free, otherwise, resource is busy => someone is using it;

(g) The complete data structure for all semaphores is a 3-dimensional matrix **S**x**U**x2, where **S** is the number of semaphore IDs, i.e. the number of shared resources, **U** is the length of each semaphore word (as addressed above) and 2 represents (1)the plane **S**x**U** of semaphore request words and (2) the plane **S**x**U** of semaphore owner's ID words.
For U=8, as bits are not declared on the dimension structure, we end up with a 2-dimensional matrix of Sx2. In the case of the ST-RTOP a number of S=16 semaphore IDs is reserved, although only 9 semaphores IDs were established (see the complete list on the section "File: Task_control.h" in Appendix A). The Figure 76 below, shows an example with 8 bits words.

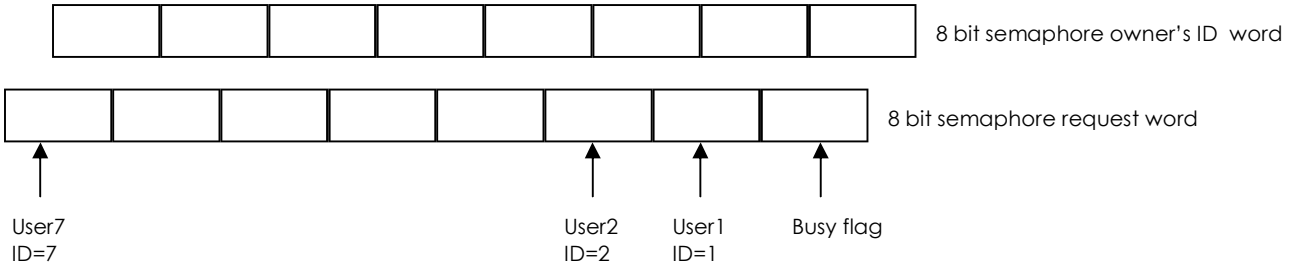


Figure 76 - 8 bits semaphore request and owner's ID words

6.5.1.3.2 Protocol for using a semaphorised shared resource

The protocol is described by the following flow charts:

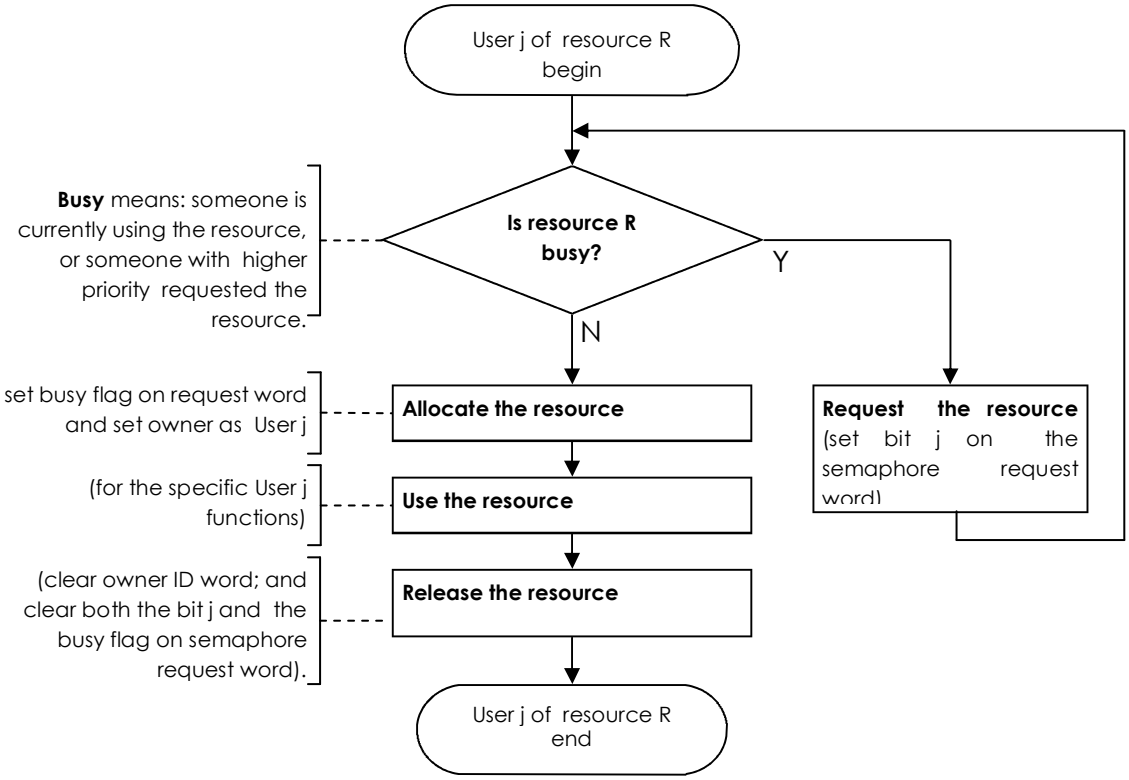


Figure 77 - Semaphores protocol for the use of a shared resource - for routines on or called from the main function (other than interrupt service routines).

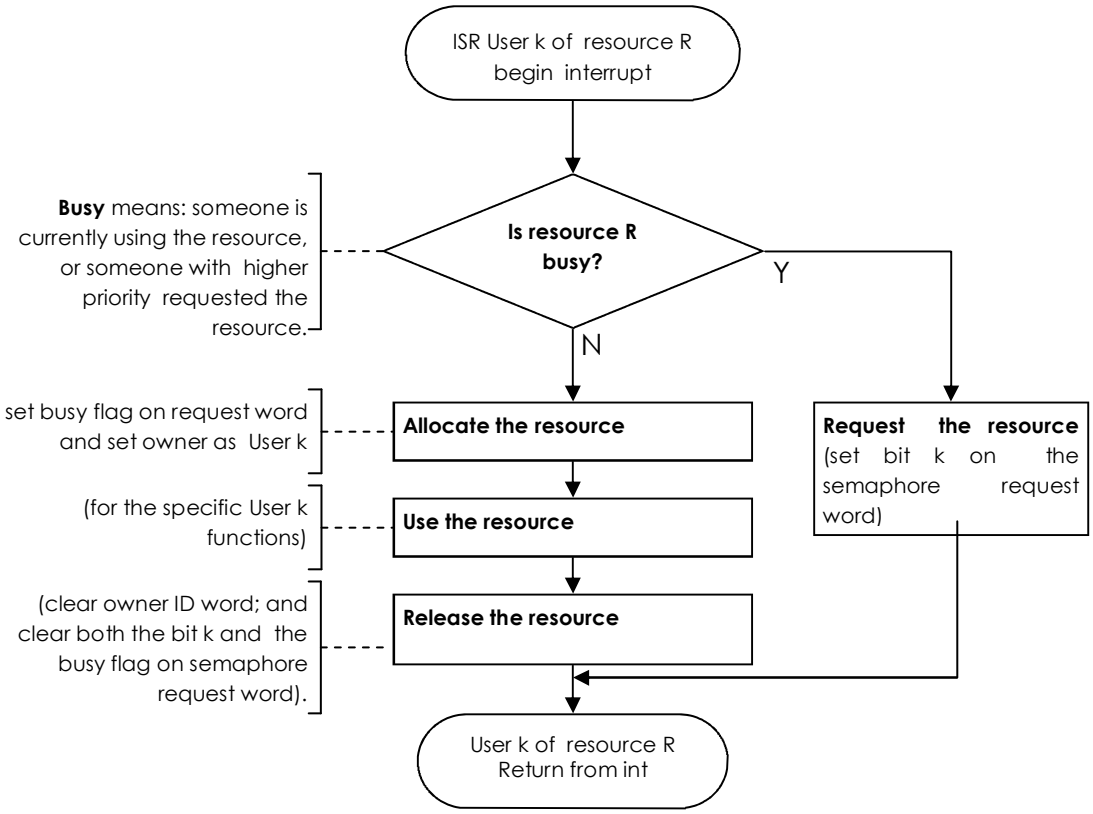


Figure 78 - semaphores protocol for the use of a shared resource - for time critical interrupt service routines or routines called from inside such interrupt service routines.

The relevant code for executing the above protocol (i.e., for checking or requesting a semaphore and releasing it) is implemented by the set of functions checklisted below and code listed on page 138 in appendix A:

- `InitSemaphores` – it releases all semaphores;
- `AllTheOnesToTheRightOf` – it returns a bit mask for testing whether someone with higher priority is requesting the specified semaphore;
- `GetSemaphore`– It test for the semaphore status not busy. If not busy it allocates the semaphore. Otherwise, it just requests the semaphore. In any case, it returns with the status;
- `ReleaseSemaphore` – it releases a selected semaphore.

We should point out that the development and later utilization of this architecture, protocol and code (original of the author), was one of the main points that provided program stabilization. Let us cite two examples of what happened before or without the use of semaphores:

First example: the ADC showed strange values when multiplexed but right ones when on a fixed channel, and the clock time on the LCD, was being corrupted for no reason. The explanation for both problems was: someone else was accessing the bus (port C = Local Bus 2) while it was being used, writing strange data to the bus, thus corrupting someone else's data. This was solved by the orderly use of the

Local Bus 2 resource through the above discussed architecture and protocol.

Second example: We observed, in data logs collected by the ST-RTOP, that in some sporadic lines, the local time was corrupted. We had no explanation for only some sporadic lines being corrupted, until we discovered that we missed placing a semaphore for accessing the RTC, by mistake, in a place where it was required. The spurious error immediately disappeared once we placed a semaphore on the required position. See the partial data log on the Figure 79 below showing the mentioned spurious corrupt data, as a factual example of bus conflicts and their solution through bus arbitring.

C1 fx LocalTime

	A	B	C	D	E
1	Timer0Tic	LastSampleTic	LocalTime	SolarTime	LocalTimeMins
155	4075	4075	St,180409,142358	141717	863.97
156	4098	4075	St,180409,142358	141717	863.97
157	4128	4125	??,;?;?;?,7?7?7?	141718	863.98
158	4151	4150	St,180409,142359	141718	863.98
159	4175	4175	St,180409,142359	141718	863.98
227	5873	5850	St,180409,142417	141736	864.28
228	5903	5900	St,180409,142417	141736	864.28
229	5926	5925	Tu,0<8204,0:2417	141736	864.28
230	5950	5950	St,180409,142418	141737	864.3
316	8098	8075	St,180409,142440	141759	864.67
317	8128	8125	St,180409,142440	141759	864.67
318	8151	8150	Tu,188204,142441	141800	864.68
319	8175	8175	St,180409,14:441	141800	864.68
395	10074	10050	St,180409,142501	141820	865.02
396	10103	10100	St,180409,142501	141820	865.02
397	10126	10125	Tu,0<8204,142501	141820	865.02
398	10149	10150	St,180409,142501	141820	865.02
415	10577	10575	St,180409,142506	141825	865.1

Figure 79 - Screen shot of a datalog (of Sat, 18Apr09)

In Figure 79, highligthed lines show corrupted local time. Note that there are some (non corrupted) lines that were hidden for showing more records.

6.5.2. The console user interface (the keyboard and the LCD)

6.5.2.1. The Keyboard software interface implementation

The keyboard hardware implementation has been addressed in the chapter V. We present here some highlights about its logical functionality:

(a) When a keyboard key is pressed an interrupt event is generated over the INT pin of the ATmega32 MCU. If there are no other time-critical interrupt service routines running, the keyboard encoder is then accessed and a 4 bits wide scancode is retrieved. From the scancode an ascii code is produced which corresponds to the key pressed on the keypad map.

(b) There are only 12 (4 x 3) keys. They are insufficient for executing a number of commands for effective user interface. An effective interface with many commands, is of great importance to enable her/him to exercise supervisory control over the system, like request manual mode, request immediate dish parking, etc. A list of the currently programmed keyboard commands is presented on table 6.2.

(c) To address the shortage mentioned above, we implemented an extended

command interface. While 11 commands are single stroke actions, the extended commands are triggered by pressing a combination of keys in the format *nn#, where the asterisk is the command prologue (when pressed it indicates that an extended command is beginning), "nn" is a number representing the command, and the hash is the command epilogue (command terminator). A list of extended commands and their description can be found in table 5.2 below.

extended Cmd Nr	Single char equivalent	Command description	Notes (Cmd = Command)
	*	Begin an extended command	
**	ESC	Abort extended command, set menu option to "1"	A double star sequence
	#	If keyboard is in buffer (multichar) mode (including extended command) then terminate it; Otherwise: adjust the RTC time;	Equivalent to CR (carriage return)
	0	Reset LCD once; show Local time from RTC chip;	
	1	Show day,date,time, onboard temperature and system (it is the start up default option)	
02	2	Show TES / Receiver's inlet and outlet temperatures	
03	3	Show current set point / actual hour and decl. angles	
04	4	Up arrow: show next command	
05	5	Show TES / Receiver's inlet and outlet temperatures	
06	6	Down arrow: execute next command	
07	7	Charging pump manually adjusted PWM control for visual debugging display charging pump actual and set point speeds.	
08	8	Display discharging pump actual and set point speeds.	
09	9	Heat utilization display (not implemented) set temporarily to tracker control monitoring	
00	A	Set tracker control to auto mode.	
01	C	Adjust the RTC time	
11	d	Download data logging configuration from RS232 (not implemented)	
12	D	Download data logging configuration from SD card (not implemented)	
20	e	Specify seekError (=tracking accuracy error)	
21	E	Specify StayError (=tracking step)	
22	F	Request fuzzy_FSM tracker control(not implemented).	
23	g	Show M2M info (not implemented).	
24	G	Datalog to M2M (not implemented).	
25	L	Specify the data logging interval in seconds	
55	M	Set tracker to manual mode.	
99	p	Set tracker to auto mode & set point to parking position	
66	Q	Request Fuzzy-PID pump control(not implemented).	
70	R	Request system reset (restart the system) (not implemented)	
77	s	Show SD card information (not implemented)	
78	S	Data log to SD card (not implemented)	

Table 5.2 – List of menu options and the respective keyboard commands.

It is worth noting that this table can be improved, by reassigning and or reordering the commands. Also, new commands can be added.

The basic functionality of the keyboard event handler, as briefly described in point (a) of this section, is represented on the flow chart of the Figure 80 below.

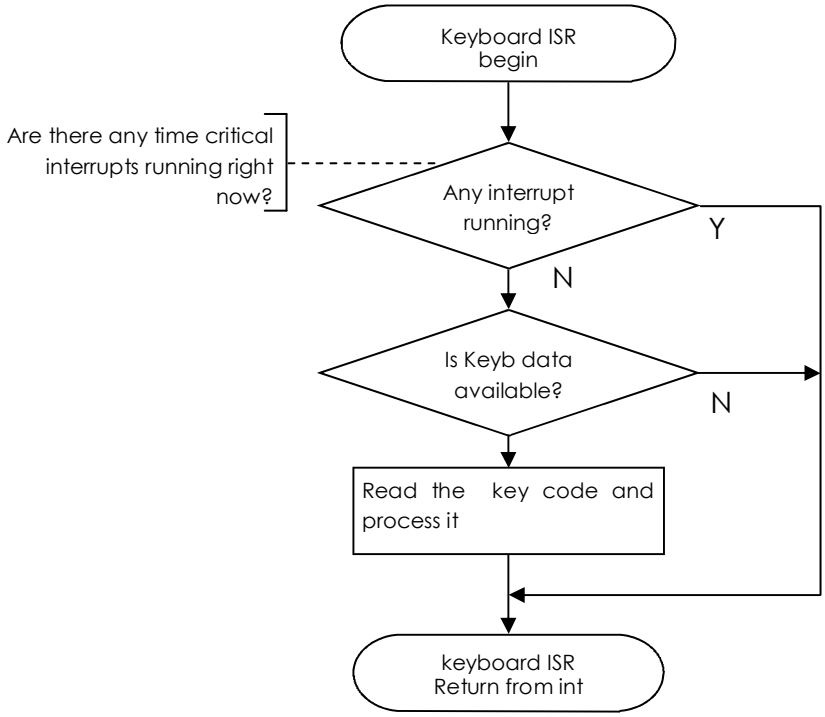


Figure 80 - Generic functionality of the keyboard event handler and comand processor

A more detailed flow chart could be represented. Though, a listing of the source code can be found in the appendix A. Both the keyboard interrupt handler and the auxiliary extended command processor can be found in page 142.

6.5.2.2. The LCD software interface implementation

The LCD is a common type, 2 lines x 16 dot matrix characters display (LMB162ABC). It is interfaced with the MCU using 4 data bits mode, as described in section 5.7.1.

Based in its datasheet information a software interface was written. The protocol for writing/reading to/from the LCD is well documented in the datasheet, including the flowcharts for initializing the LCD for either 4 bits or 8 bits data mode. We used this to write the list of interfacing function described below.

As with the real time clock, the low level hardware interfacing functions were written in assembly language, aiming at fulfilling the time-critical requirements of the LCD initialization protocol (see the datasheet in appendix D). The LCD handling functions at low level, are:

- `LCDreset` - performs LCD module initialization for 4bits interface;
- `LCDwriteCMD` - sends a command byte to the LCD;
- `LCDwriteDATA` - this sends a data byte to the LCD;

- `LCDaddress` - sets the active address (line, column) at which the next character will be written;
- `dly_1ms` – generates a delay of about 1 millisecond;
- `dly_10ms` – generates a delay of 10 milliseconds;
- `dly_100us` – generates a delay of 100 microseconds.

The only high level function is `LCDwriteMsg`, which is written in C language. The source code for this function can be found in page 150, while the source code for the low level functions listed above, are found in page 171 of the appendix A.

6.5.3 Analogue to Digital Conversion interface functions

The AD conversion interface has a separate time triggering to insure high responsivity to fast changing inputs. This was possible taking advantage of the built in ADC capabilities of auto retriggering and interrupt request on end-conversion.

There are 2 functions: The ADC_Init (the initialization function) and the ADC interrupt handler (ADC_vect). They are written following the interface and protocol defined in the relevant section of the ATmega32 datasheet. Implementation details can be seen in the appendix A, starting from page 144.

6.5.4 Thermocouple to Digital Conversion interface functions

The AD conversion for the K type thermocouples (Thermocouple to Digital conversion) is scheduled, triggered and synchronized by the timer0tic based task scheduler/triggerer, discussed in section 6.5.1.1 ("The timer tic and system timing and synchronization interface"). These timed tasks include: the start conversion task and the read converted data. This is because the TDC does not fire an interrupt at end of conversion, as well as it does not have an auto-trigger after end of conversion.

There are 3 TDC interface functions:

- The TDC_Trigger (triggers the start of a new conversion);
- TDC_ReadTrigger, which activates the SPI interrupt handler, and ;
- The SPI interrupt handler (SPI_STC_vect), reads and processes the data word.

All the 3 functions were also written following the interface and protocol specifications found in the datasheets of both the ATmega32 and the MAX6675 TDC. Details can be found on page 146, in the appendix A.

6.5.5 The sun tracking interface implementation

6.5.5.1 The tracking interface mathematical model software implementation.

At supervisory control level, there are background functions based on the

underlying laws of sun tracking. These functions perform solar time calculations and derive the set point hour and declination angles. They also calculate sunrise / sunset angles, which are used to evaluate whether it is day or night and determine the right sun tracking action to be performed accordingly.
We should revisit the sun tracking mathematical model (in part addressed in the section 2.1, equations 2.1.1.6 through 2.1.1.11) to better understand its software implementation:

The sun tracking has the goal of aiming the collector at the sun, which has an East-West apparent angular motion ω, called the hour angle. The hour axis rotates according to the hour angle. ω is proportional to the solar time (S_{time}):

$\omega = S_{time}/4 - 180°$ (which is the eq. 2.1.1.11) (S_{time} expressed in minutes)

In turn, the solar time is derived from the local time, the local and standard meridians and the equation of time. The solar time in minutes is:

$$S_{time} = L_{time} + 4(L_{st} - L_{loc}) + E \quad \text{(Which is the eq. 2.1.1.8)}$$

The standard meridian L_{st} is a product of 15^0 by the Timezone of the location under consideration (Time zone is the difference between the local time t_{loc} and the GMT time t_{GMT}): $L_{st}=15*(t_{loc}-t_{GMT})$ or $L_{st}=15$ * Timezone (which is the eq. 2.1.1.6)

The local time L_{time} in minutes is derived from the clock time (we use the RTC time). The Equation of time (eq. 2.1.1.7) is a corrective magnitude that accounts for perturbations on the Earth's rotation:

$$E = 229.2 * (0.000075 + 0.001868 \cos B - 0.032077 \sin B - 0.014615 \cos 2B - 0.040849 \sin 2B)$$

The sun has also a North-South apparent angular motion δ, called declination angle. The declination axis rotates according to the declination angle δ, which varies according to the Spencer's equation (eq. 2.1.1.10.1):

$$\delta = 0.006918 - 0.399912 \cos B + 0.070257 \sin B - 0.006758 \cos 2B + 0.000907 \sin 2B - 0.002679 \cos 3B + 0.00148 \sin 3B$$

Where, B is the fractional year at the current day d: $B = (d-1)*360/365$ (eq.2.1.1.2.2)

Sunrise and sunset angles:

- SunsetAngle = acos(-tan(*LatitudeAngle*pi*/180) * tan(*DeclinationAngle*))
- SunriseAngle = -SunsetAngle

Durban/UKZN plant's location data:
Longitude $L_{st}=30^0$; $L_{loc}=30^0\,56'40.0''E$; *Latitude*=$29^0\,49'2.0''S$ and *Altitude* ≈ 200m.

This mathematical model is implemented through the following set of functions:

- CalcSolarTime - calculates the solar time and also derives the current set point hour and declination angles;
- SunsetHourAngle – calculates the sun set angle;

- EQofTime – calculates the equation of time;
- FractionalYear_B – calculates the fractional year;
- CurTime2Secs - converts clock time from MM:dd:hh:mm:ss to seconds;
- YearDays – calculates the current day of the year;
- IsLeapYear – return true if the current year is a leap year;
- MonthDays – returns the number of days of the current month.

Implementation details can be found in pages 148 onwards, in appendix A.

6.5.5.2 Tracker's Finite state machine controller software implementation

The FSM sate flow diagrams have been discussed in chapter IV and illustrated in Figure 42 through Figure 45. The software implementation of the FSMC was derived from the state flow diagram of Figure 45. However, experimental observation forced further alterations, ending up in the flow diagram of Figure 81. below.

In fact, we introduced a change in the precedence of state transitions sequence, now, giving precedence to the northwards motion over the rest. This was caused by the dish parking procedure. The resting position is: hourangle=0^0, and declinationangle=-30^0. From this position, the dish cannot move any further southwards and cannot move the hour axis (either directions). This is like parking or unparking a car in a narrow space: once parked one cannot further move forward or to the sides. This applies to the solar dish assembly. On the other hand, in the diagram below we include abbreviated state transition conditions to make it easy to represent them on the flow chart of Figure 82 (next page).

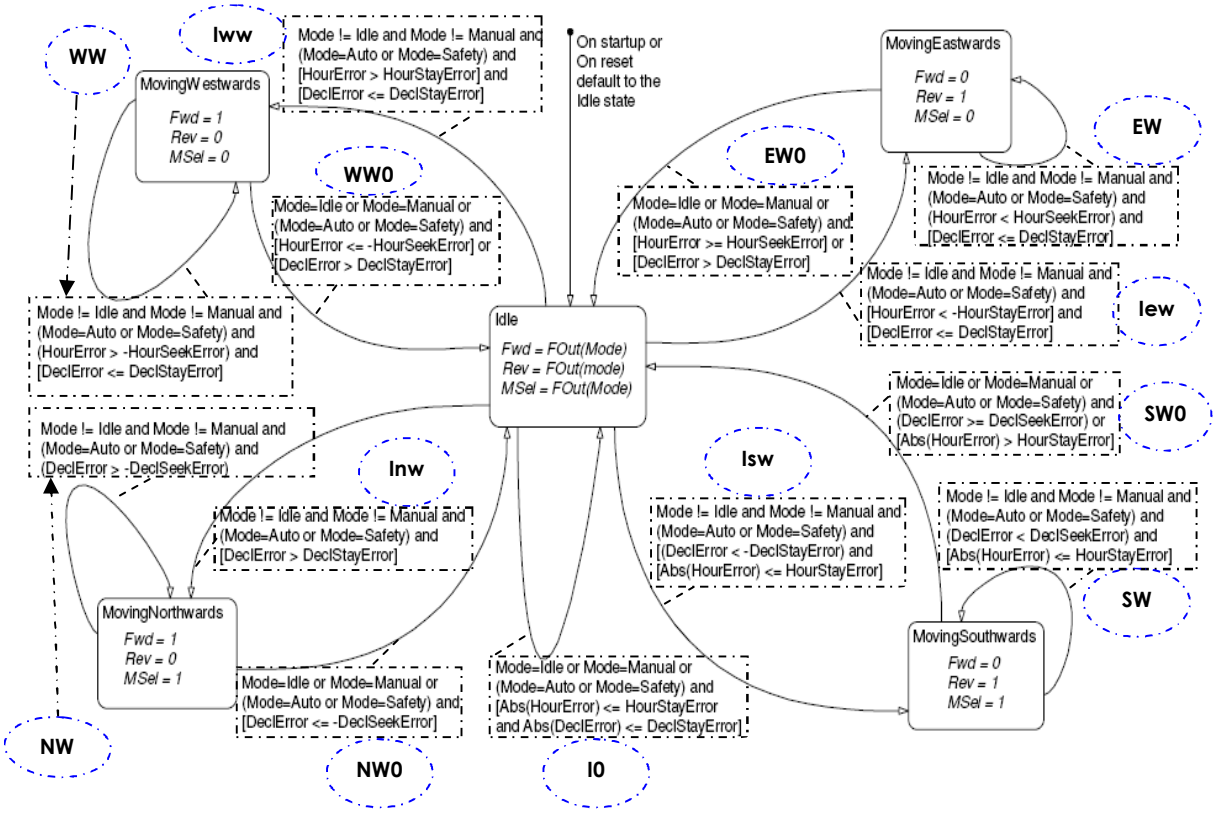


Figure 81 –FSM state flow diagram derived from that of Figure 45, with some transitions precedence changes.

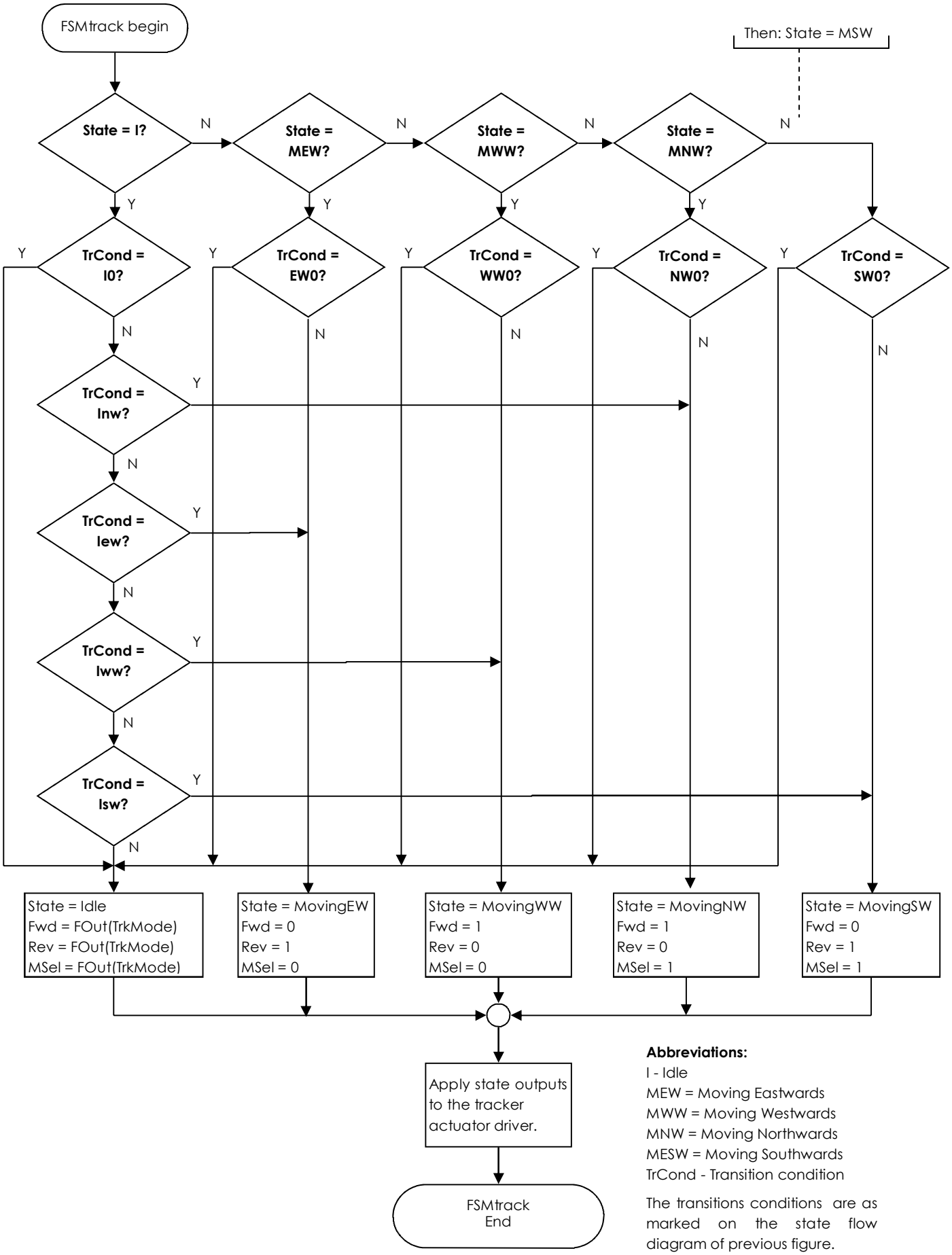


Figure 82 - (Algorithmic State Machine) Flow Chart for the FSM based tracker controller

6.5.6. The PID controller software implementation

There are various PID controllers embodied into the control strategies discussed on chapter IV. So to avoid the repetition of code the PID state values and parameters specific to the controllers, are stored in respective data structures and a common PID function is then used to generate the successive and iterative PID outputs. The function is illustrated on the flow chart of the Figure 83. See function PIDctrl on page 155 of the appendix A for source code details.

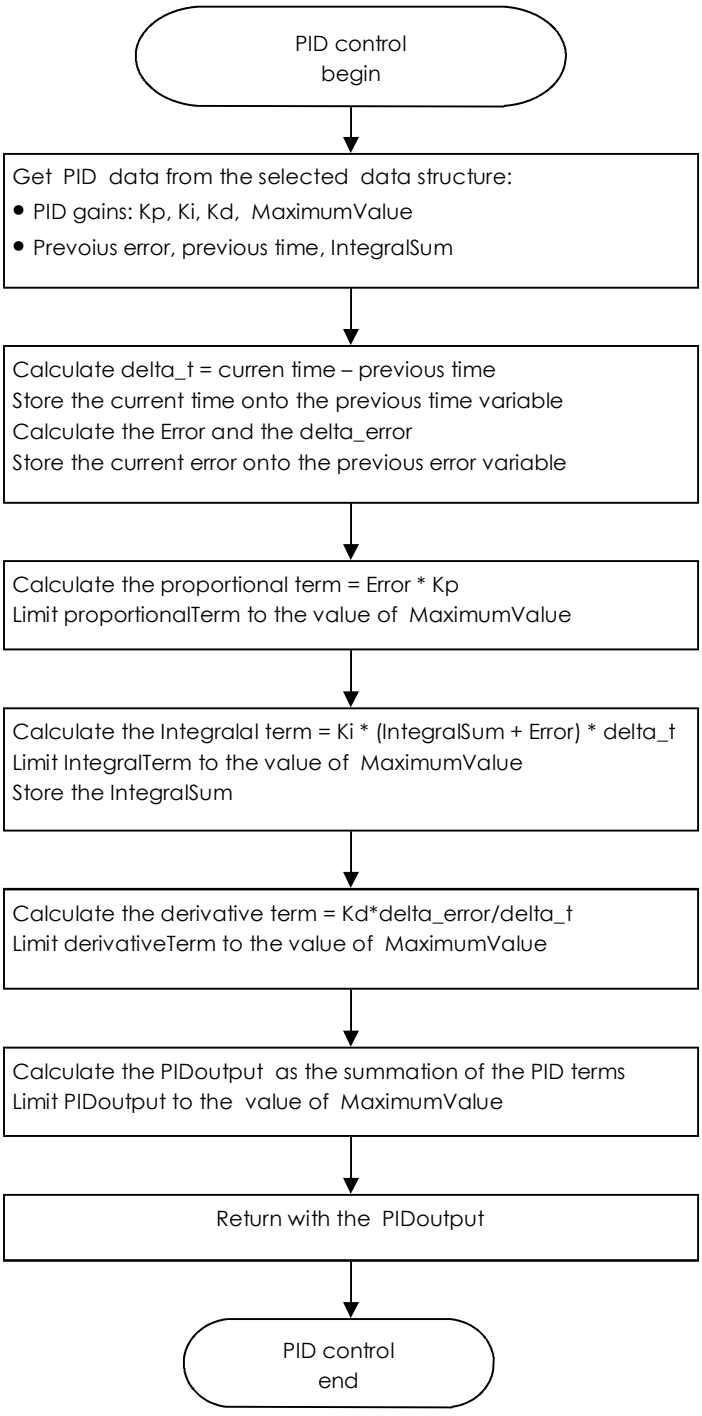


Figure 83 - PID controller software implemention

6.5.7. The MCU side data logging software interface implementation

One of the key components for a successful standalone data acquisition and control functionality, is surely the built in data logger. On the other hand, this component, far from only data logging the results of experiments, it was important for the monitoring and debugging, during the software/hardware development process. Indeed, besides logging experimental data it logged also internal system variables at time resolution of milliseconds, allowing us to evaluate what was happening at such short time scales.

The data logging architecture and functionality is as follows:

(1) There are 3 different data logging destinations (only one at a time) namely:
   - The SD card (local data logging),
   - the RS232 (remote data logging to any compatible device), and
   - M2M (machine to machine wireless interface, through the mobile network).

   However, the 1st one to be implemented was the RS232 data logging to a PC. The other 2 (SD and M2M) were not implemented.

(2) The data logging format chosen and used is the "comma separated values" (.CSV), which is readable and easily processed by many packages including spreadsheet processors, like the Microsoft Excel. The CSV protocol states that each record should have a fixed number of fields separated by commas, spaces or tabs (we used commas). Each record should terminate by a CR-LF (carriage return and line feed ascii characters). Also, the first record sent should be the column titles (if any).

(3) Stored in the MCUs EEPROM memory, there is a master table of variables that can be data logged. This list is a complete data structure with definitions, describing the variables, to be data logged. One line of that table is defined by the following C language **struct** :

```
typedef struct DataLogLine{
        char CSV_ColumnTitle[13];
        uint8_t CheckList1;
        uint8_t CheckList2;
        uint8_t CheckList3;
        void *fieldAddress;
        uint8_t fieldType;
}LogLine;
```

   where:

   - `CSV_ColumnTitle` is the name of the variable, sent in the 1st record of the .CSV file;
   - CheckList1, CheckList2 and CheckList3 are data logging configuration checklists (used for choosing which field is included in the current data logging record);
   - filedAddress is the MCU's RAM memory address of the variable, where its real time value is stored and retrieved from, to be data

logged; and,

- fieldType is the data type of the variable to be logged: integer, float, character, etc. (see *datalogger.h,* page 164, for the list of field types and other details).

The Checklists are the data logging configurations. They are a provision for telling the MCU which variables it should datalog. They are also a provision for allowing the user the freedom of choosing which fields should be included in data logging. A Checklist field is just 1 byte containing either the logical 0 or 1 (zero causes the relevant variable to be excluded from the data logging, when that Checklist is selected). The specific uses of the checklist are:

(a) Checklist 0 is a special list with all fields = 1 (this means all variables are data logged when this Checklist is selected). For saving memory Checklist0 is not stored in the EEPROM table;
(b) Checklists 1 and 3 are predefined data logging configurations, not modifiable by the user. Checkilist3 is system/developer reserved; whilst
(c) Checklist 2 is reserved for user specified configuration, although no interface for this functionality was developed.

The following flow chart (Figure 84) shows the generic functionality of the main function of the data logging interface. Details can be found in the files datalogger.c in page 164 and datalogger.h in page 164 of the appendix A.

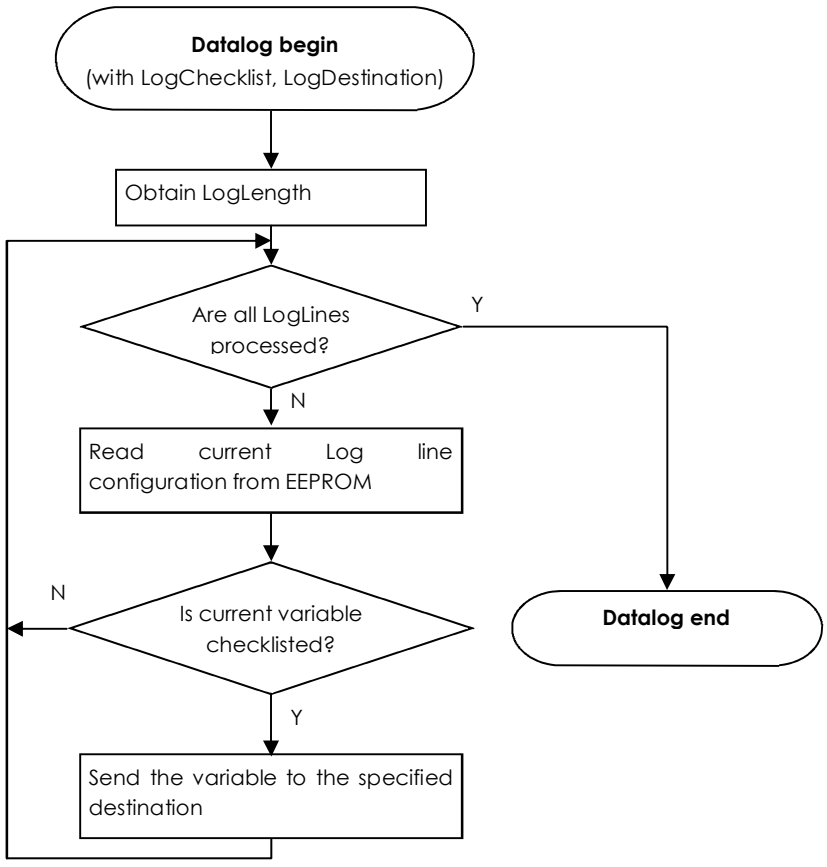


Figure 84 - Simplified flow chart of the main data logging function

6.6 The PC side data logging program

The data logging to a remote counterpart requires the existence of another program on the counterpart that receives and saves the collected data.

For the data logging to the PC via RS232 interface, we have designed and implemented a PC side data logging Windows program. This program interacts with the MCU for the process of data logging, using the RS232 protocol. The following figure is a flow chart in generic lines, of what the program does.

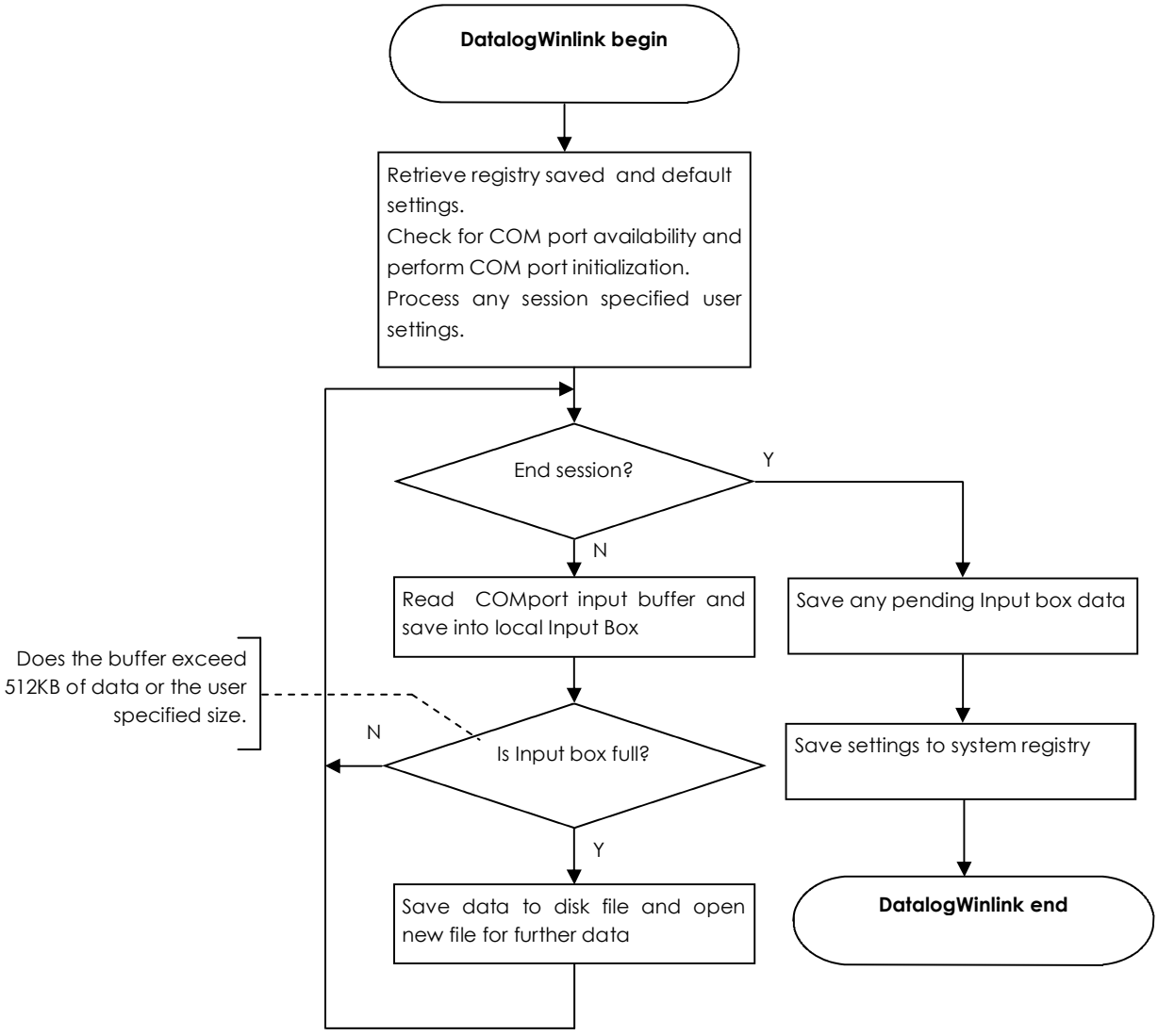


Figure 85 - Generic flow chart of the PC side data logging program (DatalogWinlink)

The program has been written in Visual Basic 6, having been successfully used to datalog the experimental data presented in next sections.

Further implementation details can be found in the source code in the appendix B. The screen shot of the Figure 86, shows a detail of the program running on a PC during one experiment held on the 22 July 2009.

Details of user interface and real time settings used in that session can be

distinguished on the screenshot, namely:

- COM port (4) and baud rate (set to 38400 bauds);
- The input box auto-save features;
- The RS232 input auto-receive feature;
- The automatic processing of the received input to extract the CSV header;
- Etc.

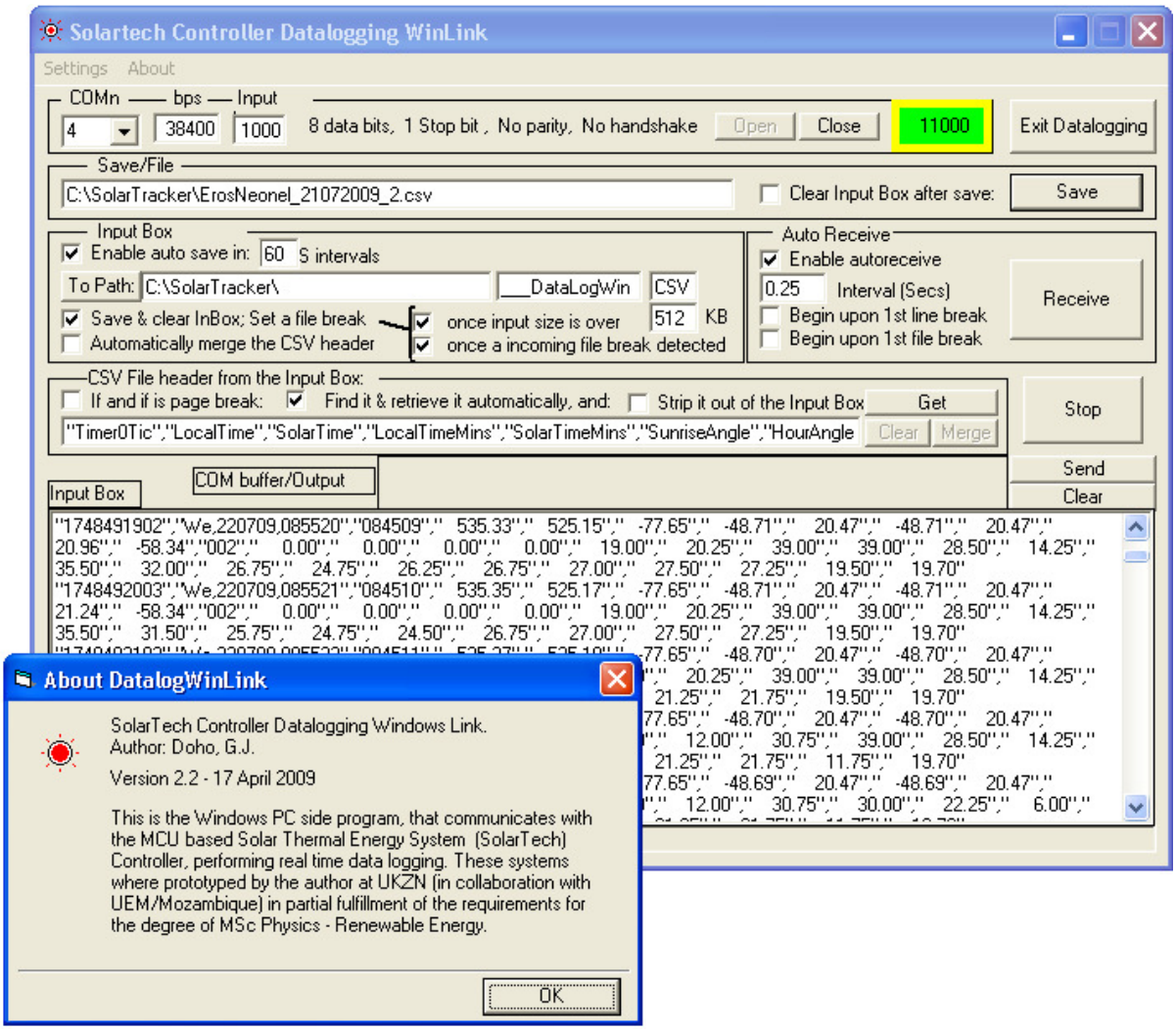


Figure 86 - Screen shot of the PC side Solar Tech data logging interface

6.7 Chapter summary

In this chapter we described the architecture and implementation of the real type operating program ST-RTOP, a tiny operating system for the embedded microcontroller system. On the other hand, we briefly presented the PC side data logging Windows program, which communicates with the ST-RTOP to jointly perform data logging of the collected data to the PC via serial port RS232.

Chapter VII – Description of Experimental Setups

In this chapter we describe the list of experiments that we performed to evaluate the functionality of the system and how it can positively influence the heat collection process. The results of these experiments will be presented in the next chapter.

7.1 Experiments performed

From the design, assembly, programming, testing, up to the real-time data collection, various experiments were performed. The experiments were conducted to:

- test the basic working of elementary hardware and software subsystems;
- calibrate or determining specific data or parameters of hardware components or subsystems;
- Evaluate the higher level functionality and/or reliability of integrated subsystems and the entire system;
- Evaluate the behaviour of the system in real time (on the plant);
- Evaluate the behaviour of real time system (plant) variables under the control of the newly built system.

We present here only the ones that can be validated through the experimental data that was collected.

7.2 The schedule of experiments

As mentioned earlier, it has been determined that the main focus of the implementation and experiments schedule be directed to: (i) the building of the basic MCU system with the basic system tasks of data acquisition and data logging; (ii)the realization of the sun tracking functionality.

The following list of experiments were scheduled and performed.

Nr	Short description of the experiment.	Experiment objectives.
1	Basic data acquisition and logging experiment.	Test that the newly built system has the basic capabilities of data acquisition through the internal ADC and of logging the acquired data to a PC via RS232.
2	Thermocouple data acquisition and logging experiments.	Test that the newly built system has capabilities of thermocouple temperature acquisition through the external TDC and logging to PC via RS232.
3	The ice to boiling experiment.	Test the reliability of the thermocouple data readouts through the observation of water temperature from the ice to the boiling conditions, under normal pressure.

Nr	Short description of the experiment.	Experiment objectives.
4	Basic Tracker control functionality experiments.	(i)Measure the tracking axes maximum angular mean speeds and (ii) evaluate that the newly built system performs the basic sun tracker control functions in real time over the plant: the sun tracking dish assembly.
5	Tracker control with temperature logging experiments.	(i)Evaluate that the newly built system is capable of performing the basic sun tracker control while concurrently reading and logging thermocouple temperatures and other system variables, and also (ii) evaluate in which manner the sun tracking positively influences the heat collection.

7.3 Description of experimental setups.

7.3.1 Setup of experiments 1: Basic data acquisition and logging experiments.

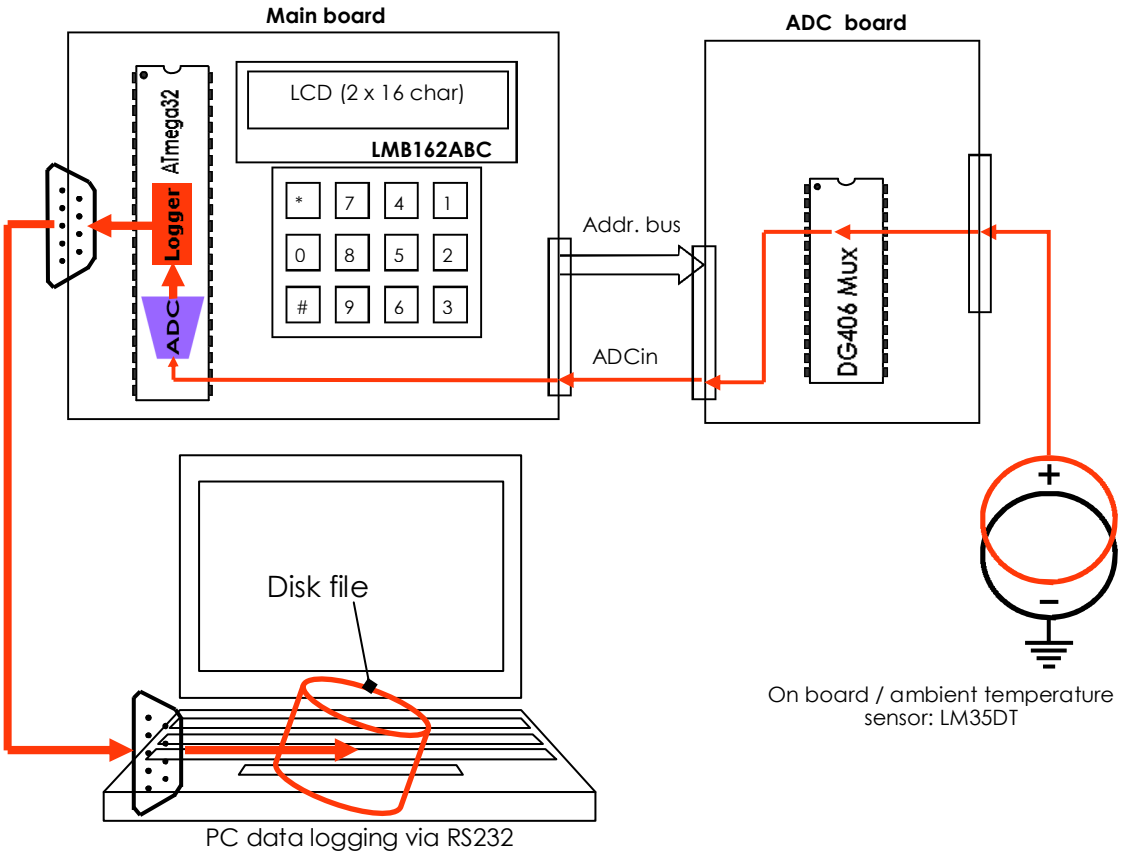


Figure 87 - Simplified diagram of the setup for the ambient temperature acquisition and logging experiment

Figure 87 shows a simplified diagram for the setup of experiment 1. In the figure, the temperature sensor LM35DT placed on the main board is read through the analogue multiplexer on the ADC board. The relevant channel is selected by sending a channel select word through the address bus (local 2 = LB2).

The AD conversion is performed using the internal MCU's ADC. The data are read and temporarily stored in a buffer in MCU's RAM. The data are later sent to the PC via serial port RS232. The data logging is performed by the data logging MCUs software component and the DatalogWinlink PC side program, all of which were described in the previous chapter VI. The results are presented in the next chapter. Sample tables of the recorded data can be found in the appendix C.

7.3.2 Setup of experiments 2: Thermocouple data acquisition and logging experiments.

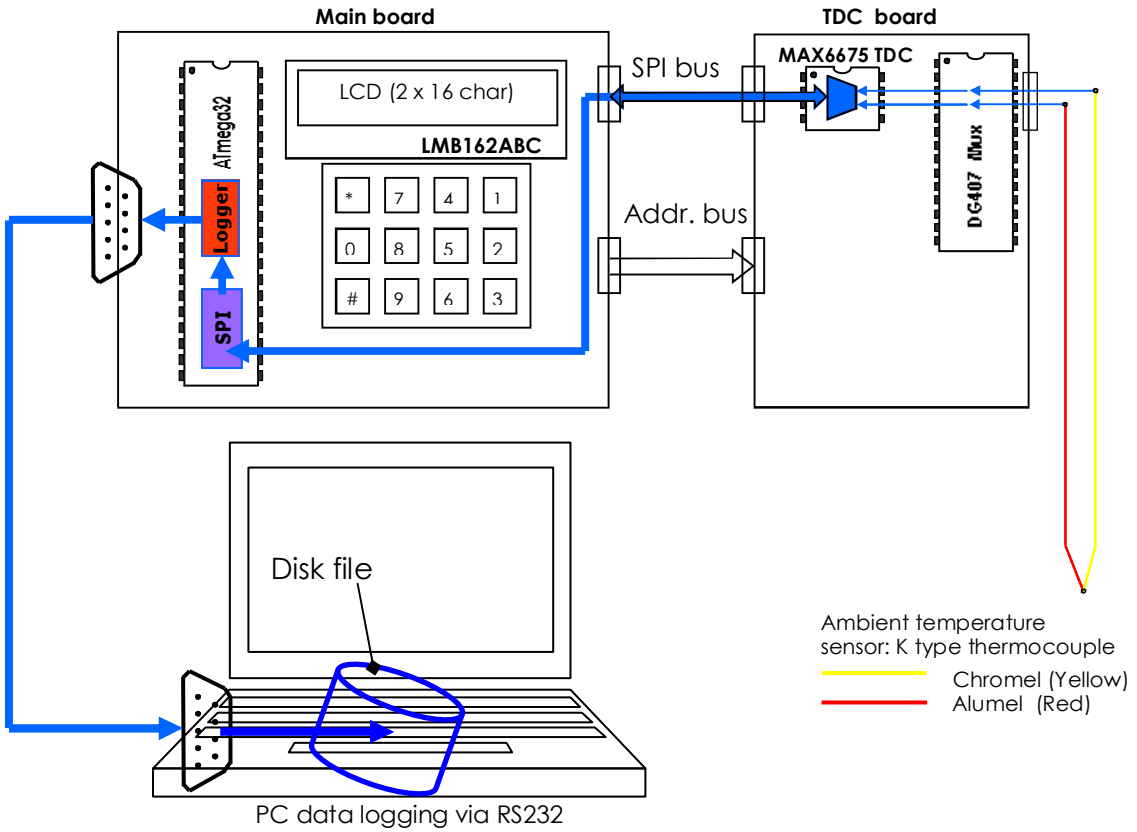


Figure 88- Simplified diagram showing the setup for the thermocoulple acquisition and logging experiments.

The figure above shows the basic setup for this experiment. In the figure, a K type temperature measuring junction is placed in air (of the lab room). The thermocouple is interfaced to the thermocouple-to-digital converter through a differential channels analogue multiplexer (DG407) placed on the TDC board. The relevant channel is selected by sending a channel select word through the address bus (local 2 = LB2).

The TD conversion is performed by the MAX6675 TDC. The data are read via the SPI bus and temporarily stored in a buffer in MCU's RAM. The data are afterwards sent to the PC via serial port RS232. The data logging is performed by the data logging MCUs software component and the DatalogWinlink PC side program, all of them described in the previous chapter VI. Sample tables of the recorded data can be found in the appendix C.

7.3.3 Setup of experiment 3: The "ice to boiling" experiment.

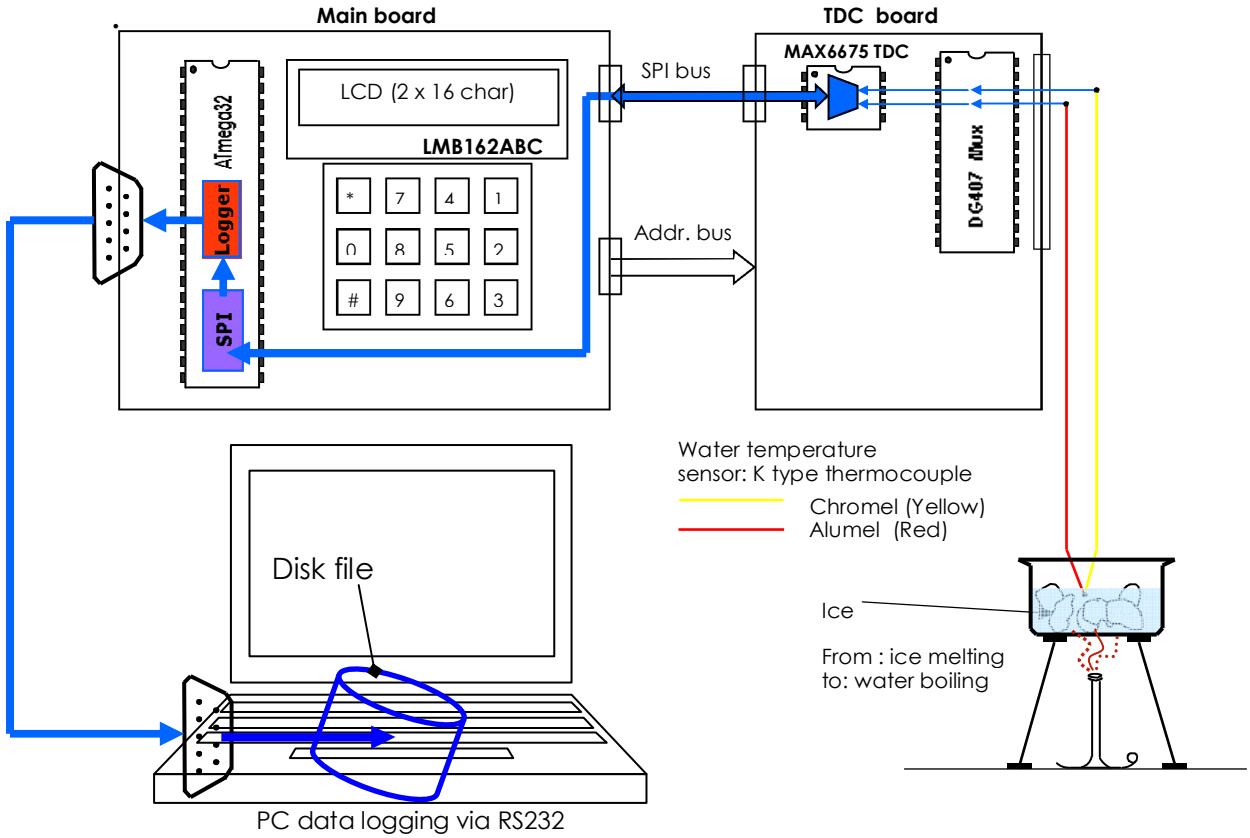


Figure 89 - Simplified diagram showing the setup for the "ice to boiling experiment"

7.3.3.1 Experimental setup description

Figure 89 above shows the basic setup for this experiment. The behaviour of water temperature when it is heated from ice (solid) state up to boiling (vapour/gas) is known. If the water is pure and the experiment is done at sea level under normal pressure, then, the melting and boiling points are known to be 0 and 100 degrees Celsius, respectively. Assuming the above conditions, the temperature behaviour is as follows:

- The water in ice state is heated from an initial temperature that is below zero (i.e., ice melting has not started);
- The water (ice) temperature increases until it reaches 0^0C, where the ice starts melting (phase change from solid to liquid). Liquid water and ice coexist during phase change;
- The temperature remains constant and equal to zero while melting is taking place until all the ice is completely melt. The energy received from the heating is spent on the phase change process;
- Once melting is finished the water temperature starts increasing up to 100^0C. At this point, water is boiling and a phase change of water from liquid to gas (vapour) is taking place;
- The boiling water temperature will remain constant and equal to 100^0C until all the water is evaporated.

This can be used to roughly evaluate the reliability of and calibrate the thermocouple temperature reading scheme, assuming that the conditions are respected.
The lab (UKZN, physics) is at approximately 200m above the sea level. The vapour pressure of water at 200m above sea level is about 99.35^{0}C [for calculation see (Nave, 2005)].

Procedure:

A pot (beaker) with ice at a temperature below zero is heated continually. The pot temperature increases until the ice starts melting;

A K type thermocouple measuring junction is immersed into the melting ice. The thermocouple is interfaced to the thermocouple-to-digital converter (MAX6675) through a differential channels analogue multiplexer (DG407) both placed on the TDC board. The relevant channel is selected by sending a channel select word through the address bus (local bus 2 = LB2).

TD conversion is performed by the MAX6675 TDC and data are read by the MCU via the SPI bus and temporarily stored in a buffer in RAM. The data are afterwards sent to the PC via serial port RS232. The data logging is performed by the data logging MCUs software component and the DatalogWinlink PC side program, all of them described in the previous chapter VI. Soon after boiling starts taking place the thermocouple is removed from the pot and placed in ambient air. Sample tables of the recorded data can be found in the appendix C.

Below is a figure showing the expected behaviour of water temperature from ice melting to water boiling conditions.

7.3.3.2 Expected results

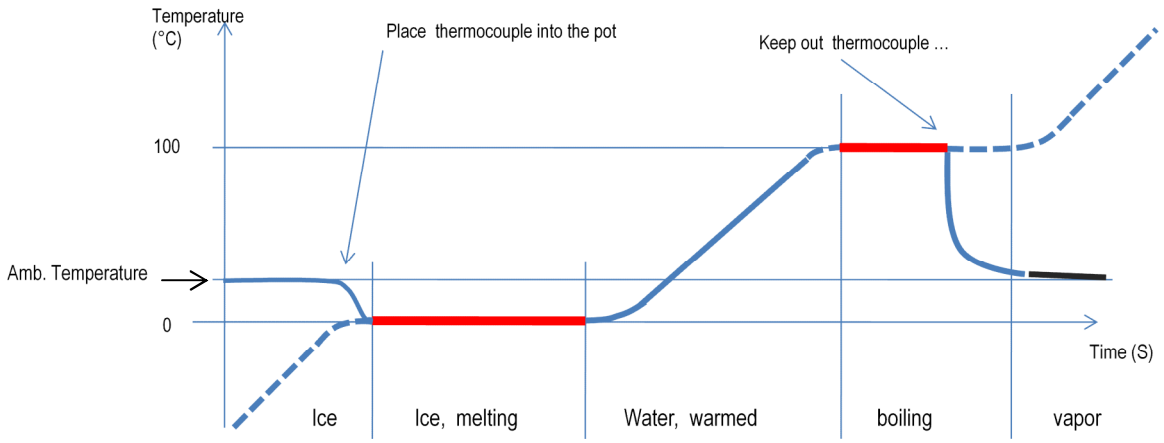


Figure 90 - Expected thermocouple temperature behavior for the "ice to boiling experiment"

7.3.4 Setup of experiment 4: "Basic Tracker control functionality experiments".

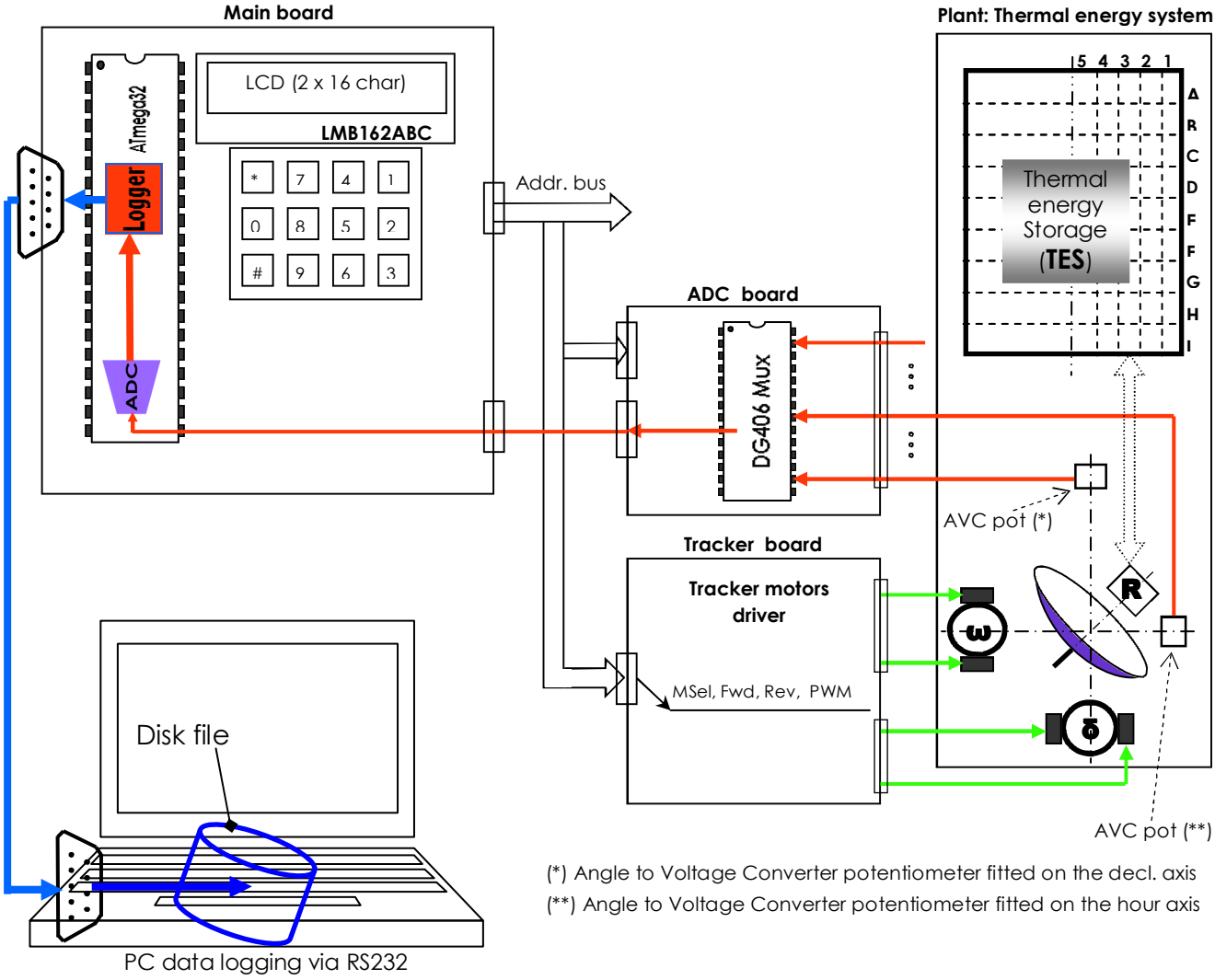


Figure 91 - Simplified diagram showing the setup for the "basic tracker control functionalityexperiments"

The figure above shows the basic setup for this experiment. In the figure, the tracker controller drives the motion of the dish (according to the control laws and supervisory controller inputs discussed earlier, see section 6.5.5.1). A single turn linear potentiometer is attached to each of the tracking axes. This is connected to act as an angle to voltage converter (AVC). In this way, the dish angular position in both axes is recorded by the AVC potentiometers. The AVCs are read through the analogue multiplexer on the ADC board. The relevant channels are selected by sending a channel select word through the address bus (local bus 2 = LB2).

The AD conversion is performed by the internal MCU's ADC. The data are read and temporarily stored in a buffer in MCU's RAM and soon afterwards sent to the PC via serial port RS232. Sample tables of the recorded data can be found in the appendix C.

7.3.5 Setup of experiments 5: Integrated tracker control with temperature acquisition and logging experiment.

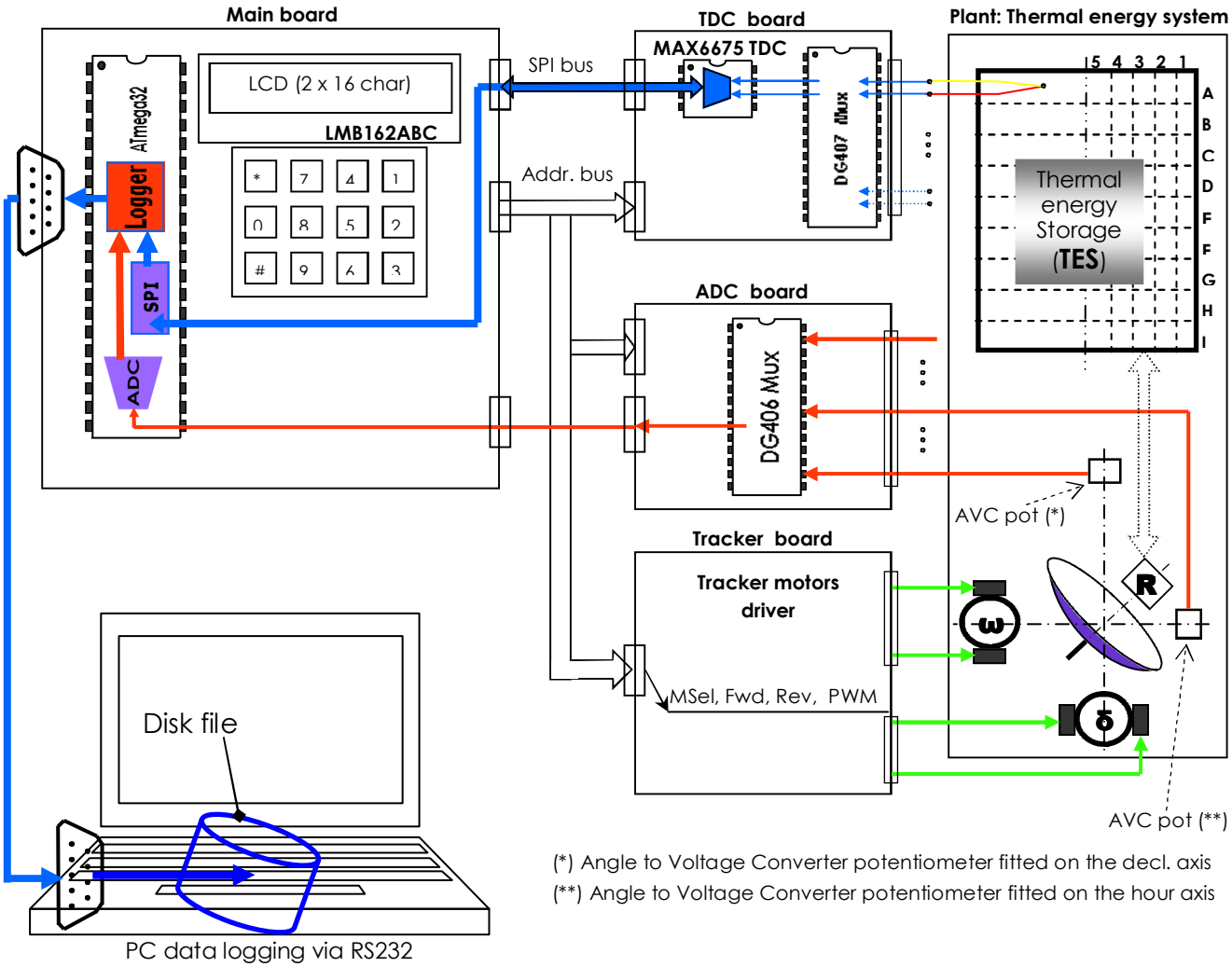


Figure 92 - Simplified diagram for the integrated tracker control and temperature acquisition experiment.

Figure 92 shows the basic setup for this experiment (which is the same as the previous one, except that the TDC board and the SPI interface are now part of the experiment). In the figure, as with the previous setup, the tracker controller drives the motion of the dish. While some of the 2 axes rotates, it drives a rigidly fitted AVC potentiometer, as a means of giving back the dish angular position of the relevant axis. The AVCs are read through the analogue multiplexer on the ADC board. The relevant channels are selected by sending a channel select word through the address bus. The AD conversion is performed by the MCU's ADC.

In turn, K type thermocouples measuring junctions are placed on various points of interest (Receiver's inlet and outlet, Receiver's outer surface, TES inlet and outlet, some TES inside profile levels an ambient/air). The thermocouples are interfaced to the MAX6675 TDC through the DG407 analogue multiplexer. TD conversion takes place and data is read by the MCU via the SPI bus and temporarily stored in a buffer in RAM. Both the ADC and the TDC data are sent to the PC via the RS232 serial interface.

Chapter VIII – Results and Discussion

This chapter presents the results obtained from the experiments, whose setups were described in the previous chapter. It is worth noting that all the plots presented here are based on recorded experimental data, which were logged using the data logging software components designed and implemented during the development of this work, as described in earlier chapters/sections.

8.1 Results of the basic data acquisition and logging experiment (using the MCU's built in ADC).

8.1.1 Ambient temperature acquisition and logging, on the 13/Aug/07.

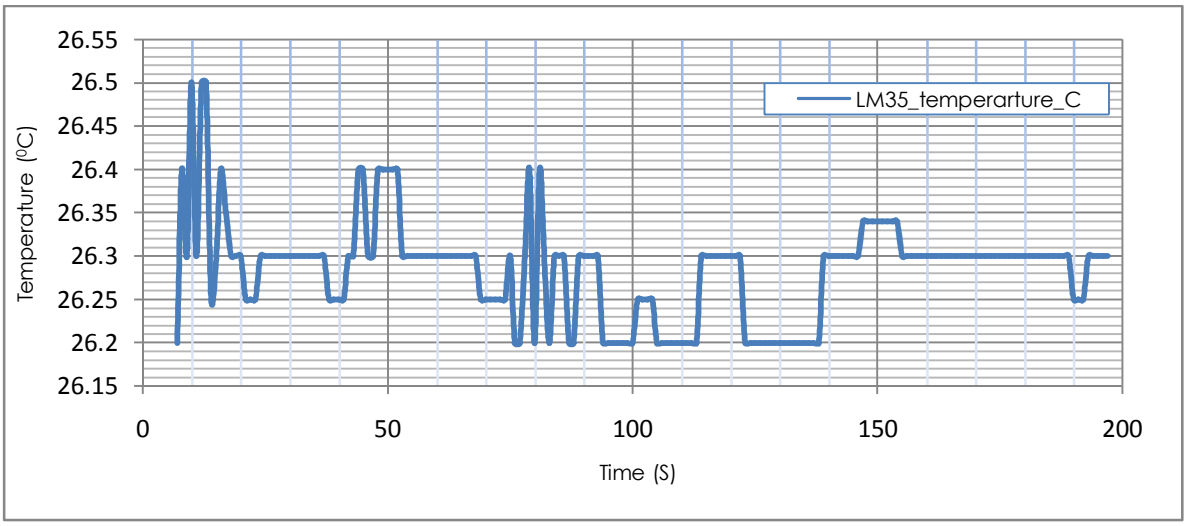


Figure 93 - Plot of ambient temperature read from the on-board LM35DT temperature sensor

The plot in Figure 93 above, shows ambient temperature, sensed by the onboard LM35DT sensor a temperature to voltage converter which yields 10mV/^{0}C. The AD conversion was underdone by the MCU's internal ADC. The data read by the ADC is stored temporarily in MCU's RAM and soon afterwards it was sent to the serial port by the data logging component of the real time operating program. The baud rate was set to 9600 baud (although in the very last experiments 38400 baud was the rate practiced). At the PC side, a windows program was put in charge of receiving, parsing and saving the CSV formatted data. The ambient temperature read from an alternate instrument, the IR gun, was about 26.25^{0}C±0.5.

Basically, this was probably one of our first (recorded) observation of desired system functionality concerning basic data acquisition from the internal ADC as well as the first observation of the stand alone capability for remote PC data logging (no conventional data logger used).

8.2 Results of the thermocouple data acquisition and logging experiments.

Figure 94 shows a plot of the ambient temperature measured on the 23/10/07, 9h, while Figure 95 shows a plot of ambient temperature collected in the same day at 14h00. In both figures the red trace is an average temperature calculated from the collected one (trace in blue).

8.2.1 Thermocouple temperature acquisition experiment A, on the 23/08/07(9h).

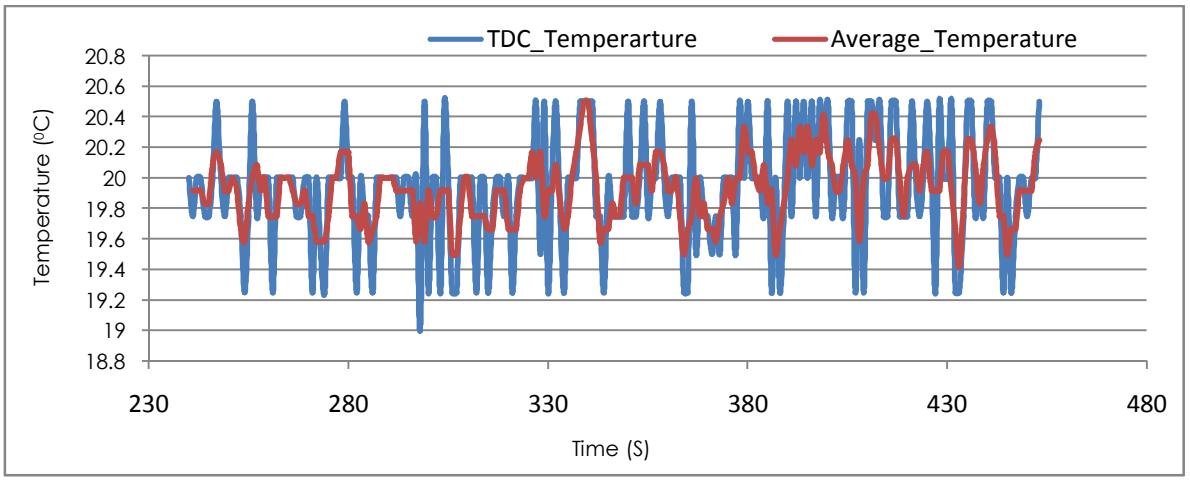


Figure 94 - Ambient temperature read from K type thermocouple (A)

8.2.2 Thermocouple temperature acquisition experiment B, on the 23/08/07 (14h).

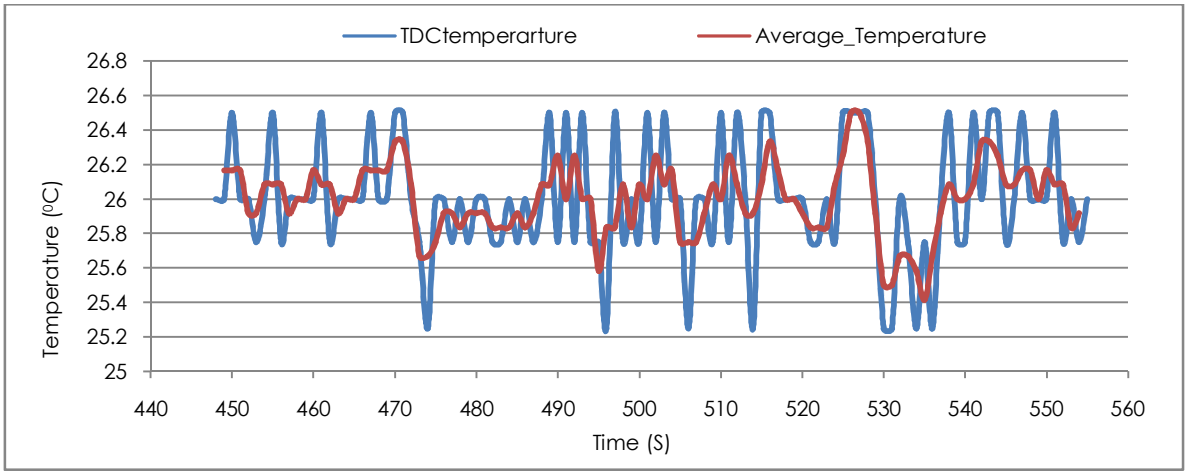


Figure 95 - Ambient temperature read from K type thermocouple (B)

For the experiment A, the ambient temperature read from an IR gun was about 20⁰C, whilst for the experiment B the ambient temperature was about 26⁰C±0.5. This set of experiments were intended to test that the newly built system has capabilities of thermocouple temperature acquisition through the TDC and logging to PC via RS232.

We found that the thermocouple temperature readings fall roughly into the actual ambient temperatures, although the readouts seemed to oscillate more than actual temperatures do (ambient temperature changes are smooth). However,

we found the reasons for such oscillatory behaviour as lying on the following facts (i) K type thermocouples do have temperature errors of about 1 to 2^0C as well as (ii) the MAX 6675 TDC does have an accuracy limited to 8LSBs (= 8x0.25^{0}C = 2^0C).

In summary, we could observe that the system was capable of reading temperatures using the MAX6675 K type TDC, and that the recorded data is sent to a remote PC via RS232, without the use of a conventional data logger.

This was our second step towards one of our overall goals: getting rid of conventional data loggers, aiming at low cost for the system being designed.

8.3 Results of the "ice to boiling" experiment, held the 23/Jul/08.

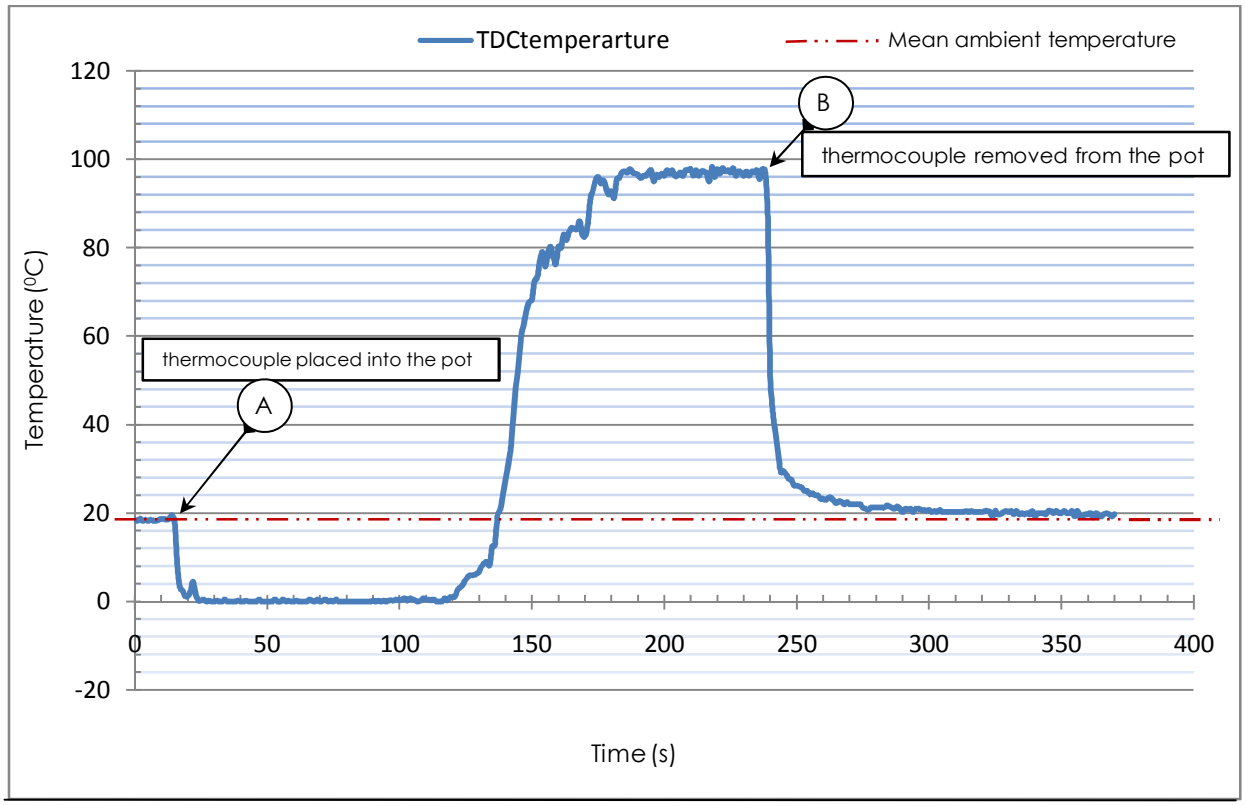


Figure 96 - Plot of water temperature from ice melting to water boiling (between points A and B).

From the plot in Figure 96 above, we can see that the results fit the expected behaviour as discussed on the section 8.3.3. In fact the recorded temperature for the ice melting point was 0^{0}C and the one for the boiling point was 98^{0}C. These values are roughly close to the expected ones. We also observe that the water temperature during both the melting and the boiling points remains roughly constant as supposed earlier.

From these results we can conclude that the TD conversion scheme that was used does not introduce a considerable offset error to the thermocouple measurements. On the other hand and once again, we positively tested the functionality and reliability of the data acquisition and built in data logging capabilities of the newly built system.

8.4 Results of the "basic tracker control functionality experiments".

8.4.1 Basic sun tracking experiment (A), on the 23/Jun/09.

Figure 97 shows the graphic results of sun tracking, where we compare the set point hour and declination angles with their actual values on the solar time axis.

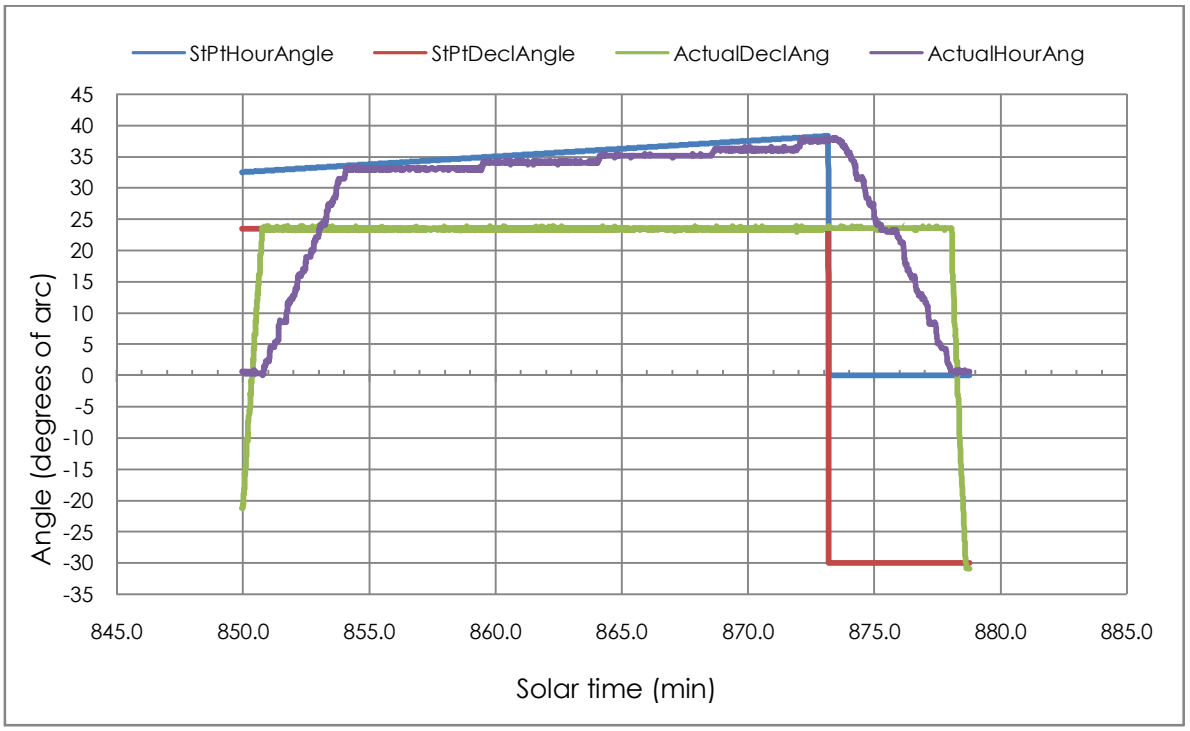


Figure 97a - Plot of the basic sun tracking experiment 1

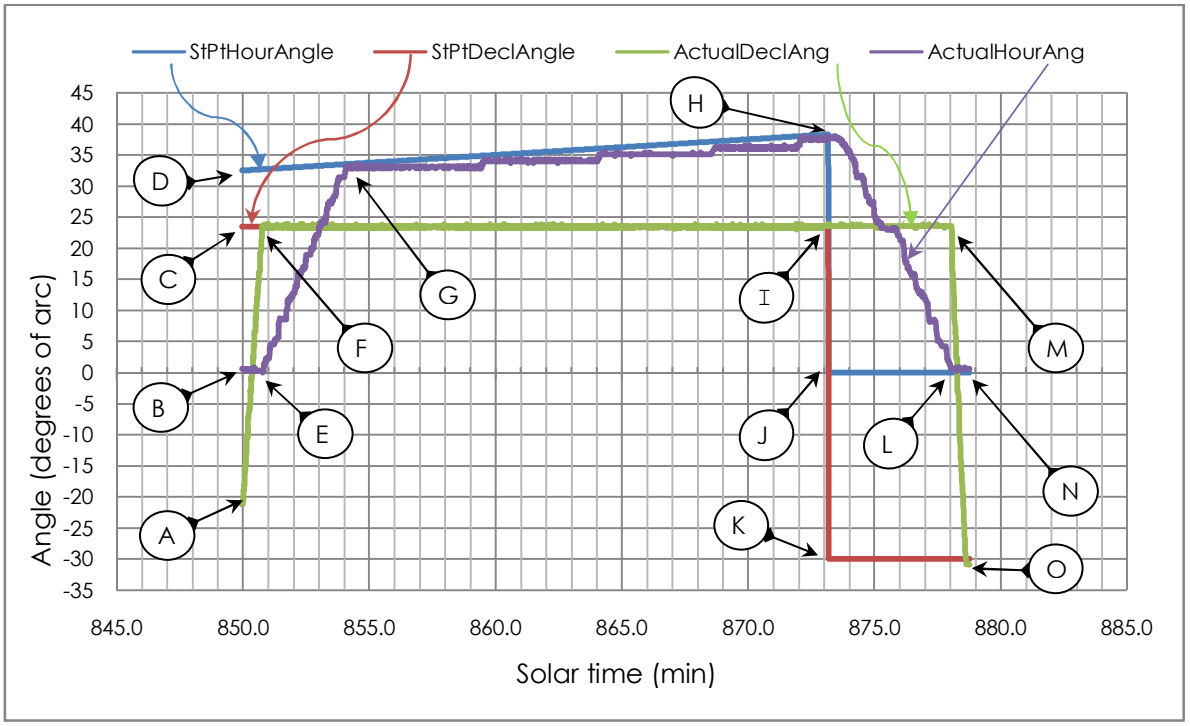


Figure 97b - Plot of the basic sun tracking experiment 1 (with further legends)

For clarity, in the Figure 97b, we add more legends to help discussing the plots and finding the results. In the figures, the various traces are as follows:

(a) ActualDeclAng represents the actual declination angle (green graph).
(b) ActualHourAng represents the actual hour angle (violet graph);
(c) StPtDeclAngle represents the set point declination angle (red graph); and
(d) StPtHourAngle represents the set point hour angle (blue graph);

Observing the figure(s) we find the following:

1) The tracking process begins at 850 min (points A, B, C, D) of the solar time axis (which equates to about 14h10 of solar time) and stops at about 878 min (≈14h39). From 850min to 873min (points H, I, J, K) a normal/automatic tracking mode was taking place and from point H (873 min) to the end, a parking process has taken place, as a result of a sudden supervisory command to park immediately and unconditionally (= send the dish assembly to its resting position). These observations apply to all curves;

2) The set point hour angle (blue curve), from points D to H, is a straight line with a slope of about 0.25^0/min. This is because the set point hour angle is a linear function of solar time (see eq. 2.1.1.11), whilst from points J to N, the set point hour angle is a straight line parallel (also collinear) to the time axis. This is because the dish's resting position set point hour angle is fixed (and equal to zero);

3) In turn, the set point declination angle (curve in red) is a straight line parallel to the time axis, at the value of about 23.5^0. This was the declination angle for 23th June. The value of declination angle does not change in a single day, which is the reason for the parallelism with the time axis. From point K afterwards, the value of set point declination angle remains at -30^0 which is the set point declination angle for the dish's resting position;

4) The tracking motion begins from the declination axis. The value of the actual declination angle (green curve) was about -21^0 at the beginning (point A, 850min). The dish moves northwards at full angular speed until it reaches the set point (on point F, about minute 851 of time axis). From point F to I the actual declination angle remains roughly overlapped with its set point, until this changes to the value of dish resting position from points I/K onwards;

5) From the experimental data we found that the value of the declination axis angular speed between points A and F is 0.94^0/S (about 56.5^0/min). This is approximately the maximum mean angular speed developed by the declination axis under normal load conditions (neglecting the variation of tracker assembly load due to the displacement of its centre of gravity during the motion);

6) At minute 851, once the declination angle matched the set point, the tracking process passes to the hour axis. The value of the actual hour angle (violet curve) was about 0^0 from the beginning (from point B to E). From point E onwards, the dish moves westwards at full speed, until it reaches the set point (on point G, about minute 854 of time axis);

7) From the experimental data, we found that the value of the hour axis angular speed between points E and G is 0.17^0/S (about 10^0/min). This is approximately the maximum mean angular speed developed by the hour axis under normal load conditions (neglecting the variation of tracker assembly load due to the displacement of its centre of gravity during the motion);

8) From point G to H we observe a stair case evolution of the actual hour angle (violet curve), following the slope of the set point hour angle. The stair case is due to the step tracking strategy embodied into the FSM controller as discussed in earlier sections;

9) As mentioned above, on points H and J (curve in blue)as well as I and K (curve in red), the set points are changed to their resting position values (0^0 for the hour angle and -30^0 for the declination);

10) From point H to L, the hour axis is rotated eastwards (curve in violet) until the resting set point (0^0) is reached. At this time, from point M to O, the declination axis is rotated southwards (curve in green), until the declination angle matches the resting set point (-30^0).

All the above observations fit the expected behaviour of the sun tracking process, under the control performed by the ST-RTOP program, particularly the FSM tracker controller software component.

There are some aspects however that deserve remarking:

(a) The plot lines where the tracking assembly is rotating freely, towards a given set point, e.g. segments AF (green), EG (violet), HL (violet), MO (green), etc., show that the rotating motion is not smooth. This lack of smoothness is more noticeable on the hour axis (violet curve). As discussed in section 4.3.1.5.3(b), this may have to do with the variation and non-linearity of the mechanical load, caused mainly by:
   (i) mechanical imperfections (lack of cleanliness, unevenness and severe rusting) of the motion transmission system, as well as;
   (ii) sudden and unpredictable changes of the environment conditions (like the wind) and also;
   (iii) The displacement and variation of the concentrator's centre of gravity with respect to the supporting structure;
   (iv) The limited resolution of the MCU's internal ADC (10 bits) for reading the AVC potentiometers (with a scale of 290^0 of arc). In this case, the 10 bits

ADC can only yield a mere 0.28^0/LSB as opposed to the required minimum resolution of 0.25^0. Also, the limited accuracy of the AVC potentiometers may be contributing to this issue.

It is worth noting that the problems mentioned in (i) several times caused the tracking assembly to get stuck and needed human intervention for releasing it and restarting the automatic tracking;

(b) Although respecting the step tracking strategy, the value of the actual hour angle (violet) in the interval GH is always below the set point. That results in a non null mean error. This equates to: "the dish is almost all the time slightly defocused". The desired behaviour would be: "the dish is almost all the time focused (and only defocused in short whiles)". This unwanted behaviour can be well observed on the plots of Figure 98 as well.

(c) To address the shortcoming discussed in (b) we have taken measures in time, to improve the tracking behaviour accordingly. Such improvements can be observed in the plots of Figure 99 and Figure 100.

8.4.2 Basic sun tracking experiment (B), on the 24/Jun/09.

In Figure 98, the curves represent a section of the sun tracking experiment carried out on the 24th June. The various traces are as follows:

(a) ActualDeclAng represents the actual declination angle (green curve).
(b) ActualHourAng represents the actual hour angle (violet curve);
(c) StPtDeclAngle represents the set point declination angle (red curve); and
(d) StPtHourAngle represents the set point hour angle (blue curve);

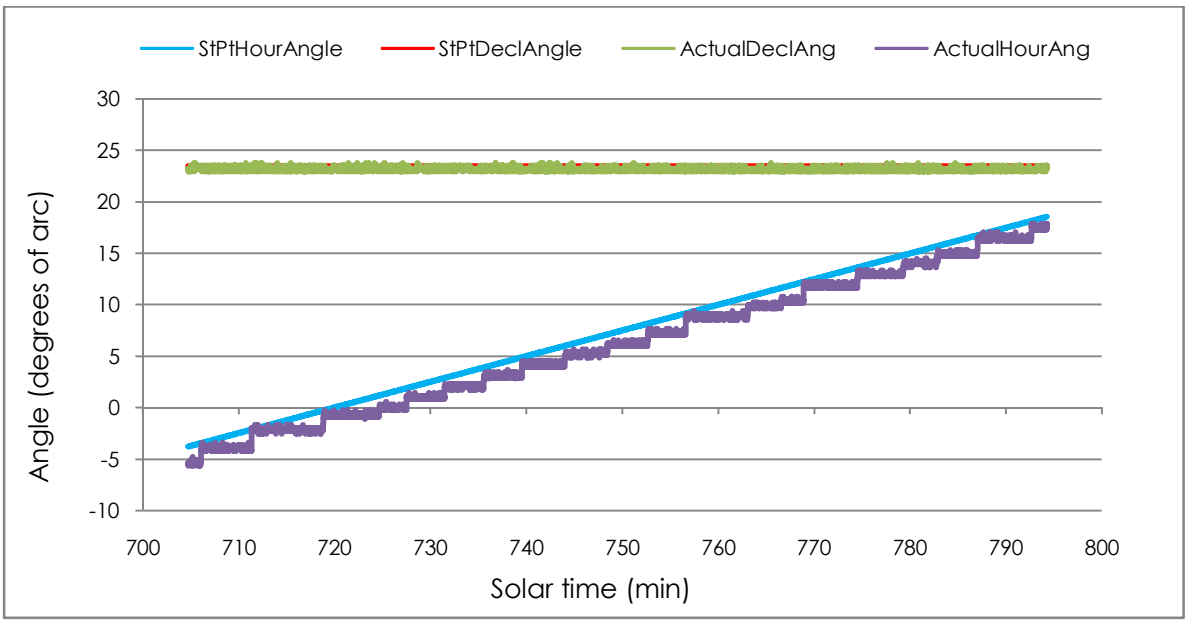


Figure 98 - Plots showing a section of the basic tracking experiment of 24/Jun/09

As mentioned in previous section, the staircase (actual hour angle, violet trace) in

the plot of Figure 98, is always below the set point. We now present (in Figure 99) a section of the results of the 25th June experiment and compare their characteristics. The various traces are named and coloured as in Figure 98.

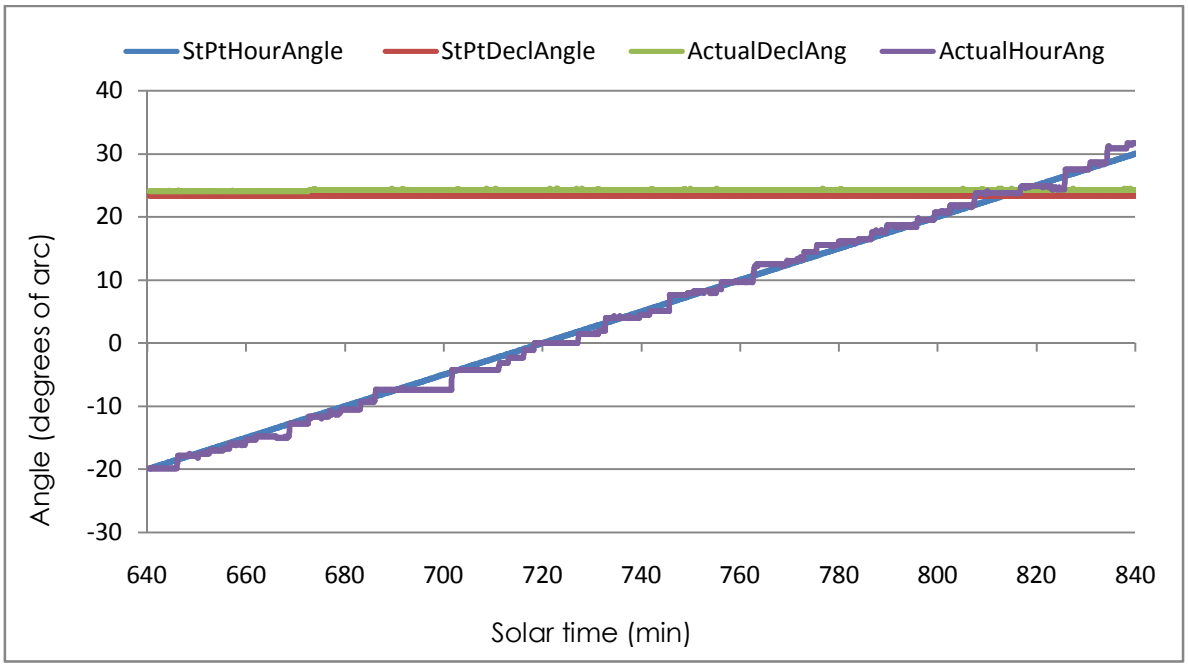


Figure 99 - Plots showing a section of the integrated sun tracking experiment, on the 25/Jun/09.

In the Figure 99 above, we can see that, the actual hour angle staircase plot walks around (above and below) the set point straight line, which now yields a mean error that, is closer to zero (see table 8.1). In fact the calculated errors for the plot in Figure 99 show evident improvement as compared to those of Figure 98. The maximum error should be neglected because it may be a result of a sporadic disturbance, not a persistent error.

Date of experiment / Variable		Maximum Error	Mean Error	RSM Error
24th June (Figure 98)	Hour angle	2.75	1.22	1.32
25th June (Figure 99)	Hour angle	3.73	0.23	0.97

Table 8.1 – Actual hour angle errors relative to its set point in normal tracking (automatic mode tracking, parking mode excluded).

On the other hand, in Figure 99, we see that although the tracking step was set to 0.5^0 and the accuracy to 0.25^0, in some sections of the hour angle plot noticeable deviations or lack of uniformity can be observed. These have to do with the shortcomings discussed in section 8.4.1 (a) above, which can be overcome by improving the system (plant) condition and/or the control strategy to the ones suggested in chapter IV.

8.5 Results of the integrated tracker control with temperature logging experiment, on the 25/Jun/09.

Figure 100 shows the full plot for the 25th June experiment. There are two groups of plots. The upper group shows:

(i) Direct solar radiation (Irrad - light blue trace) logged with a separate HP data logger;
(ii) Receiver's frontal surface temperature (RcvOuterTemp - dark blue trace) collected from a thermocouple placed on the receiver's hot spot zone;
(iii) Rock bed storage inlet level A temperature (TESinletTemp - red trace);

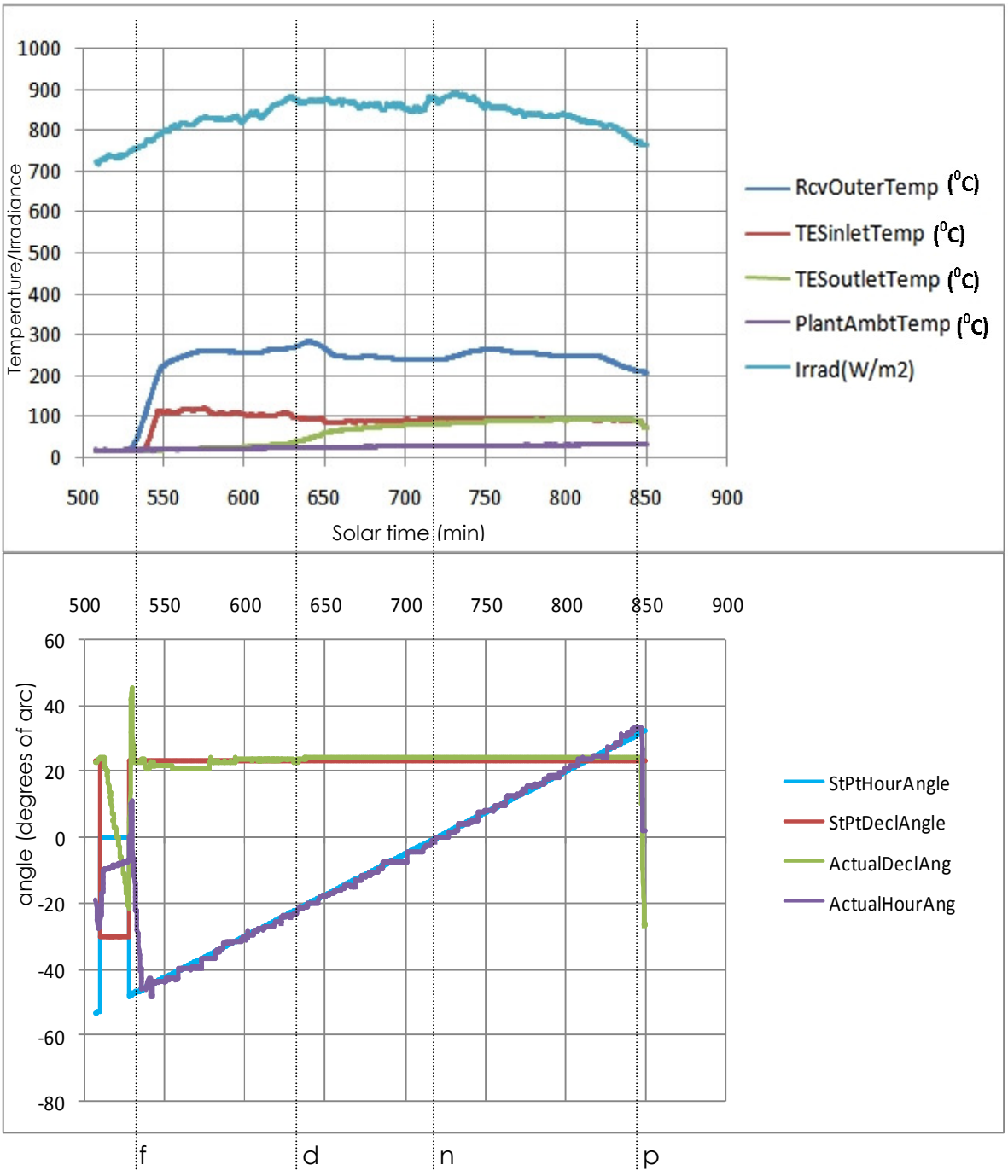


Figure 100 - Integrated sun tracking and temperature acquisition experiment

(iv) Rock bed storage outlet level I temperature (TESoutletTemp - green trace);
(v) Plant/ambient temperature (PlantAmbtTemp - violet trace) collected from a thermocouple sensor, measuring the environment temperature.

The lower group of plots represent sun tracking angles, in which relevant curves are named and coloured as in previous sun tracking plots, namely:

(vi) The set point (StPtDeclAng) and actual (ActualDeclAng) declination angles (red and green traces respectively), and
(vii) The set point (StPtHourAng) and actual (ActualHourAng) hour angles (blue and light violet traces respectively).

All the data were collected by the system except the solar radiation data as indicated above.

Observing the figure, we can learn the following:

1) The data collection begins about 510 min of the solar time axis;
2) From the beginning up to slightly before line f, the tracker was being adjusted manually because it got stuck, due to some areas of the transmission system that are in bad mechanical condition (as mentioned earlier).
3) At about 530 min (line f) the declination axis gets focused (lower group violet trace overlaps the green). Slightly afterwards, the hour axis gets also focused (lower group green trace matches the red);
4) About the same time that the axes get focused, heat collection begins. This can be seen from the receiver's surface temperature that begins increasing at line f (upper group dark blue trace). Also, some 10 min later, the rock bed level A temperature starts increasing too (upper red trace);
5) From about 630 min (line d) the rock bed's lower level temperature (upper green trace) starts showing an increase towards the temperature of the upper level (upper group red trace), as a result of the rock bed collecting energy. Though, about this time (line d), there is a noticeable decrease in the temperature of the rock bed's upper level temperature (upper red trace);
6) About solar noon (line n, 720 min) the rock bed's lower level temperature approaches the one of the upper level;
7) At about 840 min of solar time (line p) the sun tracking and data collection is stopped (see lower group traces).

The errors of this sun tracking, were addressed in the previous section while comparing the hour tracking errors and discussing shortcomings and gradual improvements from one experiment to other.

We can see that the sun tracking and focusing, positively influenced the heat collection process. On the other hand, we should remember that, as the dish is a point focus collector, almost no heat could have been collected without the sun tracking performed by the control system.

We should point out that, among others, there was a negative influence of the charging pump, which may be the main reason for the decrease in the heat transfer to the storage, observable (on the upper red trace) from line d on the Figure 100. The pump was unable to maintain a constant speed. Sometimes, the pump was almost not rotating at all. This is due to its poor condition, which could not be fixed, suggesting its replacement along with other equally poor condition components.

Despite some negative influences, we can conclude that the system can perform the monitoring and logging of important plant variables and perform the control of the sun tracking process and thereby positively influence the heat collection process.

8.6 Chapter summary

In this chapter we have presented and discussed the experiments results, reaching the final milestone of the work. At this point we are ready to make the conclusions of the work, which can be found in the next chapter.

Chapter IX – Conclusions and outlook of further work

9.1 The roadmap and the main obstacles faced.

It has been a long roadmap: from the simple first contact with the thermal energy system, passing to the study, design, implementation, testing, up to the real time control implementation and data collection.

The main difficulties were, among others, the obsolete condition of the thermal energy system including the fact that it was mostly dismantled. Many important system components could no longer be brought back to their full working state. This greatly compromised the success of this work as with the work of others.

Also, the fact that we were studying and building a MCU based control system while also studying another system (the thermal energy plant), brought some increased difficult to the work, by crowding and lengthening our activities roadmap.

Anyhow, we could walk against the majority of these obstacles and develop a work that we fill it is not perfect, but left us with many learning subsidies.

9.2 Conclusions. Achievements and failures.

1. A microcontroller based data acquisition and control subsystem for a solar thermal energy system, has been designed and built, based on (but not limited to) the characteristics of the target plant, the UKZN's Solar Thermal Energy System being developed at School of Physics.
2. The newly built system is capable of performing basic data acquisition and control tasks: direct digital inputs/outputs and analogue inputs interfaced to either the MCU internal ADC (generic inputs) or the external TDC (for K type thermocouple inputs);
3. The newly built system has built in capabilities of performing data logging to a remote PC, without the aid of a conventional data logger;
4. The newly built system is capable of providing automatic sun tracking in dual axis polar mount configuration, of point focus dish collector. This tracking capability can be used with any sun tracking system, requiring only minor changes to the hardware and/or software;

5. The system showed some limitation in the tracking, with accuracies in the order of 0.25^0. This is due to many reasons: (i) the poor condition of the tracking assembly, (ii) the influence of environmental condition (like the wind causing disturbances). This can be improved by improving the plant's condition and using better control strategies, like the proposed Fuzzy-FSM.

9.3 Recommendations

Due to limited time and financial resources, not all hardware and software features mentioned in the design phase have been implemented. So, there are gaps that must be filled in, to have the system working up to its desired capabilities and performance. Such gaps and/or improvements, among other, are:

(a) To avoid the negative influence of poor condition of the thermal energy plant on the reliability and performance of the control subsystem (and the whole system), the plant condition should be improved or even rebuilt from scratch. In fact almost everything needs replacement: the TES, the pumps, etc. This should be done as the first action, before any other suggested activities are performed;

(b) For improving tracking accuracy and the effective dish to sun aiming, the fine alignment subsystem (photodiodes) should be completed. In turn, the resolution of the coarse alignment provided by the potentiometers fitted on the tracking axes, can be improved by using more accurate potentiometers and using an external ADC with more than the 10-bits resolution offered by the MCU's internal ADC. We should remark however that, improving coarse alignment is not necessary with the fine alignment subsystem working.

(c) For increasing the independence and further lowering the cost of data logging, the software interface for the local SD card data logging should be implemented;

(d) To complete the system control functionality the pumps controllers and the overall control system should be completed and/or improved; also,

(e) The implementation of the already proposed better control strategies is also a follow-up recommendation for such competition or improvement.

Bibliography /References

Abdallah, S. and Nijmeh, S. 2004. Two axes sun tracking system with PLC control. *Elsevier / Energy Conversion and Management 45 (2004) 1931–1939*. 2004, Vols. 45 / pp. 1931–1939.

Aiuchi, K., et al. 2006. Sensor-controlled heliostat with an equatorial mount. *Elsevier/Solar Energy*. 2006, Vols. 80 (2006) 1089–1097.

Alata, Mohanad, Al-Nimr, M.A. and Qaroush, Yousef. 2005. Developing a multipurpose sun tracking system using fuzzy control. *Energy Conversion and Management*. 46 (2005) 1229-1245, 2005.

Al-Karaki, Jamal N. and Al-Mashaqbeh, Ghada A. 2007. Energy-centric routing in wireless sensor networks. *Elsevier - Microprocessors and Microsystems*. February, 2007, 31.

B. Ekwurzel . 2007. Findings of the IPCC Fourth Assessment Report. [Online] 2007. http://www.ucsusa.org/assets/documents/global_warming/IPCC-WGI-UCS-summary-72dpi.pdf.

Brogren, Maria. 2004. *Optical Eficiency of Low-Concentrating Solar Energy Systems With Parabolic Reflectros*. Uppsala : Uppsala Univeristy, Sweden, 2004.

Brooks, David. R. 2006. Monitoring Solar Radiation and Its Transmission Through the Atmosphere. *Prof David R. Brooks Web Page*. [Online] Drexel University, August 2006. [Cited: 5 June 2007.] http://www.pages.drexel.edu/~brooksdr/DRB_web_page/papers/UsingTheSun/using.htm.

Camacho, E, Berenguel, M and Rubio, F. *Advanced Control of Solar Plants*. s.l. : Springer.

Clifford, M.J. and Eastwood, D. 2004. Design of a novel passive solar tracker. *Solar Energy*. 77, 2004, 269–280.

Crossbow_Technology. 2009. Crossbow Technology. *Crossbow Technology Wireless Sensor Networks Overview*. [Online] 2009. [Cited: 8 6 2009.] http://www.xbow.com/Technology/Overview.aspx.

Cuamba, B. at al. 2006. A solar energy resources assessment in Mozambique. *Journal of Energy in Southern Africa*. November, 2006, Vol. 17, 4.

Cumbe, Fabiao; Sharma, Deepak and Lucas, Carlos. 2008. The Status of "Clean Cooking Fuels" in Mozambique. 2008.

Duffie, J.A. and Backman, W.A. 2006. *Solar Engineering of Thermal Processes, 3rd*

Ed. s.l. : John Wiley & Sons, 2006.

Dust_Networks. 2009. Dust Networks. [Online] 2009. www.dustnetworks.com.

Ehrhart, Charles and Twena, Michelle. 2006. *Climate Change and Poverty in Mozambique.* [http://www.care.dk/multimedia/pdf/web_english/Climate%20Change%20and%20Poverty%20in%20Mozambique-Country%20Profile.pdf] s.l. : CARE International, 2006.

Elsevier. 2001. Smart Dust Could Conserve Energy. *Elsevier / Materials Today*. 2001, Vol. 4, 4.

Geyer, M., et al. 2008. *Solar Power and Chemical Energy Systems - SolarPACES Report 2007*. Köln/Germany : International Energy Agency (IEA), 2008.

Gruian, Flavius. 2002. *Energy-Centric Scheduling for Real-Time Systems.* Lund : LTH - Lund Institute of Technology, 2002. ISBN: 91-628-5494-1.

Hankins, Mark. 2009. A Renewable Energy Plan for Mozambique. [Online] 9 2009. www.internationalrivers.org/files/Clean%20Energy%20for%20MZ%2030_9_09.pdf.

Heetkamp, R.R.J. 2002. The development of Small Solar Concentrating Systems with Heat Storage for Rural Food Preparation. *Physica Scripta.* 2002.

IPCC. 2007. Combining Evidence of Anthropogenic Climate Change. [Online] 2007. http://www.ipcc.ch/publications_and_data/ar4/wg1/en/ch9s9-7.html.

Jantzen, J. 1998. Fuzzy Supervisory Control. *Technical University of Denmark, Department of Automation.* [Online] 1998. http://www.iau.dtu.dk/~jj/pubs/proc.pdf.

Jet Propulsion Laboratory, NASA. 1998. *Solar Absorption Spectrometry.* [mark4sun.jpl.nasa.gov/solar.html] s.l. : JPL-NASA, 1998.

Kalogirou, S.A. 2004. Solar thermal collectors and applications. *Elsevier / Progress in Energy and Combustion Science* . 2004, Vol. 30, 231–295.

Kalogirou, Soteris A. 1996. Design and construction of a one axis sun tracking system. *Elsevier/Solar Energy*. 1996, Vol. 57 / Nr 6.

Khalifa, A. and Al-Mutawalli, S. 1998. Effect of Two-Axis Sun Tracking on the Performance of Compound Parabolic Concentrators. *Elsevier/Energy Convers. Mgmt*. 1998, Vols. 39 / pp. 1073-1079,, 10.

Kim, Jin-Sung, et al. 2008. Auto Tuning PID Controller based on Improved Genetic Algorithm for Reverse Osmosis Plant. *http://www.waset.org/ijist/v3/v3-4-37.pdf*. [Online] International Journal of Intelligent Systems and Technologies 3;4, 2008.

Kuo, Benjamin C. 1980. *Digital Control Systems.* New York : HRW - Holt, Rinehart and

Winston, Inc., 1980. 0-30-057568-0.

Leigh, J.R. 1988. *Temperature Measurement and Control.* London : Peter Peregrinus, Ltd, 1988. ISBN: 0-86341-111-8.

Levermann, Anders, et al. 2005. Dynamic sea level changes following changes in the thermohaline circulation. *Nature*. February, 2005.

Linkens, D.A. and Chen, Minyou. 1998. A hybrid neuro-fuzzy PID controller. *Elsevier/ Fuzzy Sets and Systems* . 1998, Vol. 99 , 27-36.

Løvseth, J. 2008. On Heat Transfer Methods from Absorber to Storage. *Annual Small Scale Concentrating Solar Energy Systems Workshop.* Kampala, Makerere University, Uganda : s.n., 2008.

Mathworks, The. 2009. Stateflow - State chart design and development environment- Simulink.htm. *www.mathworks.com.* [Online] The Mathworks, 2009. [Cited: 5 June 2009.] http://www.mathworks.com/products/stateflow/?BB=1.

Mawire, A. 2005. *A data acquisition and control system for a solar thermal energy storage (TES) and cooking system.* Durban : UKZN Libarary, 2005.

Mawire, A.; McPherson, M. and Heetkamp, R.R.J. 2008. Simulated energy and exergy analyses of the charging of an oil–pebble bed thermal energy storage system for a solar cooker. *Elesevier - Solar Energy materials & Solar Cells.* 2008.

Mensah, Edoe F. 2008. Logic-Based Optimal Control for Shipboard Power System Management - PhD Thesis. *Drexel University*. [Online] 2008. http://dspace.library.drexel.edu/bitstream/1860/2765/1/Mensah_Edoe.pdf.

Nave, Rod. 2005. Vapor Pressure - Boiling Point Varaition. *HyperPhysics.* [Online] R.Nave - Dep of Physics and Astronomy - Georgia State University, 2005.

NEIC(EIA-DOE-USA). 2009. Greenhouse Gases, Climate Change, and Energy. [Online] EIA-DOE-USA, 2009. www.eia.doe.gov/oiaf/1605/ggccebrochapter1.html.

Ogata, K. 2002. *Modern Control engineering.* 4th. s.l. : Prentice Hall, 2002. ISBN 0130432458.

Osornio-Rios, R.A, et al. 2008. The application of reconfigurable logic to high speed CNC milling machines controllers. *Elsevier/Control Engineering Practice*. 2008, Vol. 16.

Pakanen, J. and Karjalainen, S. 2009. A state machine approach in modelling the heating process of a building. *Elsevier.* 2009, Vol. 49.

Petro-Canada. 2006. CALFLO family of products. [Online] 2006. http://www.calflo.com/family.htm.

Prapas, D.E., Norton, B. and Probert, S.D. 1986. Sensor System for Aligning a Single-

Axis Tracker with Direct Solar Insolation. 1986, Applied Energy 25 (1986) 1-8.

Root, Terry L., et al. 2003. Fingerprints of global warming on wild animals and plants. *Nature.* January, 2003, 421 (57-60).

Roth, P.; Georgiev, A. and Boudinov, H. 2005. Cheap two axis sun following device. *Elsevier / Energy Conversion and Management 46 (2005) 1179–1192.* 2005, Vol. 46.

Schiermeier, Q. 2006. Climate change: A sea change. *Nature.* 2006, 439.

Shafiee, S and Topal, E. 2009. When will fossil fuel reserves be diminished? *Energy Policy.* January, Pages 181-189, 2009, Vol. 37, 1.

Sollar_Millenium_AG. 2009. Sollar Millenium AG. [Online] 2009. http://www.solarmillennium.de/technology/solar-thermal-power-plants/locations/index.html.

Stine, William B. and Geyer, Michael. 2008. Power From The Sun. *Power From The Sun.* [Online] 2008. http://www.powerfromthesun.net/book.htm.

UN_Statistics_Division. 2009. *Environment Statistics Country Snapshot: Mozambique.* s.l. : United Nations Statistics Division, 2009.

Ungeheuer, Udo. 2005. *SCHOTT Memorandum on Solar Thermal Power Plant Technology.* Mainz / Germany : Schott AG, 2005.

Verbruggen, H.B. and Bruijn, P.M. 1997. Fuzzy control and conventional control. What is (and can be) the real contribution of Fuzzy Systems? *Elsevier / Fuzzy Sets and Systems* . 1997, Vol. 90, 151-160.

Wenham, S., et al. 2007. *Applied Photovoltaics.* s.l. : earthscan, 2007. ISBN 1844074013.

World_Bank. 2009. Mozambique at a glance. [Online] December 2009. devdata.worldbank.org/AAG/moz_aag.pdf.

Yan, J., Ryan, M. and Power, J. 1994. *Using Fuzzy Logic.* London : Prentice Hall, 1994. ISBN 0-13-102732-8.

Zadeh, Lofti A. 1965. 1965 Seminal Work on Fuzzy Sets. [Online] 1965. http://www-bisc.cs.berkeley.edu/Zadeh-1965.pdf.

Zhong, Jinghua. 2006. www.purde.edu. *PID Controller Tuning: A Short Tutorial.* [Online] 2006. web.ics.purdue.edu/~jzhong/gttc/lesson/part1.pdf.

Zomeworks. 2008. How Trackers Work - Zomeworks Passive Solar Products. *Zomeworks Passive Solar Products.* [Online] Zomeworks, 2008. http://zomeworks.com/products/pv-trackers/how-trackers-work.

Appendix A : Listing of ST-RTOP - Solar Tech's Real Time Operating Program

```
/*---------
```

File: SolarTechController3-14-5.c (The main c program of the ST-RTOP)

```
*///-------------------------------------------------------------------------
// ST-RTOP (SolarTech - Real Time Operating Program): A specific purpose RTOS-like,
// real time monitoring and control program for the MCU based solar thermal energy
// system (Solar Tech) controller at UKZN.
// Author: Doho, G.J. (2007/2009). All rights reserved.
//
// Programmed and tested with the ATmega32 at 8MHz.
// Progam was actually developed consecutivelly on the ATmegas 8535, 16 and 32;
// however this final code is actually resources compatible with the mega32.
// It will be applicable to ATmegas 324/644 subfamily (with some changes).
//
// general obs: assume option 0 (no optimization) compiling; no volatiles defined.

//includes;
#include <stdint.h>
#include <stdlib.h>
#include <stdio.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <string.h>
#include <math.h>
#include <avr/eeprom.h>
#include "portconfig3_14.h"
#include "Tracker.h"
#include "Task_control.h"
#include "pumps.h"
#include "Datalogger.h"
#include "USART.h"
#include "1302rtc.h"

#define MCUXX 0              //mega16/32 subfamily
#define MCUXX4 1       //mega324/644 subfamily

#if !defined(MCU_TYPE)         //directives for MCU addaptive compiling
        #if defined(__AVR_ATmega16__)
        #       define MCU_TYPE MCUXX
        #elif defined(__AVR_ATmega32__)
        #       define MCU_TYPE MCUXX
        #elif defined(__AVR_ATmega64__)
        #       define MCU_TYPE MCUXX
        #elif defined(__AVR_ATmega164__)
        #       define MCU_TYPE MCUXX4
        #elif defined(__AVR_ATmega324__)
        #       define MCU_TYPE MCUXX4
        #elif defined(__AVR_ATmega644__)
        #       define MCU_TYPE MCUXX4
        #elif defined(__AVR_ATmega164P__)
        #       define MCU_TYPE MCUXX4
        #elif defined(__AVR_ATmega324P__)
        #       define MCU_TYPE MCUXX4
        #elif defined(__AVR_ATmega644P__)
        #       define MCU_TYPE MCUXX4
        #else
        #  error "Unsupported MCU!"
        #endif
#endif

#if !defined(F_CPU)    // comment/uncomment as needed
//#     define F_CPU 4000000UL //MCU clock freq in Hertz
#       define F_CPU 8000000UL //MCU clock freq in Hertz
//#     define F_CPU 10000000UL       //MCU clock freq in Hertz
//#     define F_CPU 16000000UL       //MCU clock freq in Hertz
//#     define F_CPU 20000000UL       //MCU clock freq in Hertz
#endif
```

```
/*
//select ADC Prescaler at compile/build time
//select ADC Prescaler and Timer0 Preload Count at prscaller 1
//assume timer prescaler table  set to 1 (no prescaling)
#if (F_CPU == 20000000UL)
#  define ADC_prescalMASK ADC_prescalMASK20MHZ
#  define TIMER0_PRELOADCNT 236
#elif (F_CPU == 16000000UL)
#  define ADC_prescalMASK ADC_prescalMASK16MHZ
#  define TIMER0_PRELOADCNT 240
#elif (F_CPU == 10000000UL)
#  define ADC_prescalMASK ADC_prescalMASK10MHZ
#  define TIMER0_PRELOADCNT 246
#elif (F_CPU == 8000000UL)
#  define ADC_prescalMASK ADC_prescalMASK8MHZ
#  define TIMER0_PRELOADCNT 248
#elif (F_CPU == 4000000UL)
#  define ADC_prescalMASK ADC_prescalMASK4MHZ
#  define TIMER0_PRELOADCNT 252
#else
#  error "define Prescaller mask for the specified F_CPU!"
#endif
*/


//select ADC Prescaler at complie/build time
//assume timer prescaler for this table set to 8 (fcpu/8)
/*#if (F_CPU == 20000000UL)
#  define ADC_prescalMASK ADC_prescalMASK20MHZ
#  define TIMER0_PRELOADCNT 6
#elif (F_CPU == 16000000UL)
#  define ADC_prescalMASK ADC_prescalMASK16MHZ
#  define TIMER0_PRELOADCNT 56
#elif (F_CPU == 10000000UL)
#  define ADC_prescalMASK ADC_prescalMASK10MHZ
#  define TIMER0_PRELOADCNT 131
#elif (F_CPU == 8000000UL)
#  define ADC_prescalMASK ADC_prescalMASK8MHZ
#  define TIMER0_PRELOADCNT 156
#elif (F_CPU == 4000000UL)
#  define ADC_prescalMASK ADC_prescalMASK4MHZ
#  define TIMER0_PRELOADCNT 206
#else
#  error "define Prescaller mask for the specified F_CPU!"
#endif */

//the active table
//select ADC Prescaler at complie/build time
//assume timer prescaler for this table set to 64 (fcpu/64)
#if (F_CPU == 20000000UL)
#  define ADC_prescalMASK ADC_prescalMASK20MHZ
#  error "for fcpu 20MHz, prescaler 64 not supported on Timer0!"
#elif (F_CPU == 16000000UL)
#  define ADC_prescalMASK ADC_prescalMASK16MHZ
#  define TIMER0_PRELOADCNT 6
#elif (F_CPU == 10000000UL)
#  define ADC_prescalMASK ADC_prescalMASK10MHZ
#  define TIMER0_PRELOADCNT 100
#elif (F_CPU == 8000000UL)
#  define ADC_prescalMASK ADC_prescalMASK8MHZ
#  define TIMER0_PRELOADCNT 131
#elif (F_CPU == 4000000UL)
#  define ADC_prescalMASK ADC_prescalMASK4MHZ
#  define TIMER0_PRELOADCNT 194
#else
#  error "define Prescallers mask for the specified F_CPU!"
#endif

#define pi  3.14159265358979323846264 3
#define noDisp 1
#define toDisp 0
#define cPadZero '0'
#define cPadSpace 32
```

```
//function prototypes;
void RTC_on(void);
void RTC_off(void);
void WrByte(char byteData);
uint8_t RdByte(void);
char RTC_ReadByteAtAddress(uint16_t Addr);
void RTC_WriteByteAtAddress(uint16_t Addr, char byteData);
char RTC_GetSecs(void);
void RdClockBurst(char * prtClockDataString);
void WrClockBurst(char *ptrClockDataString);
void RTC_ResetInit(void);
void RTC_Adjust(uint8_t *ptrClkData);
char *RTC_Disp(char *ptrClockData, char line, uint8_t DispOrNot); //zero:disp; 1: no disp
char BCD2Ascii(char byteData);
char Hex2Ascii(char byteData);
char Ascii2BCD(const char msbAscii, const char lsbAscii);

void LCDreset(void);
void LCDwriteDATA(uint8_t LCDdata);
void LCDwriteCMD(uint8_t LCDdata);
void LCDaddress(uint8_t LCDline, uint8_t LCDcol);
void LCDwriteMsg(char *MsgPtr, char line, char col);

void ADC_init(void);
void int2Ascii(uint16_t, char *);
void I2A(uint32_t *inVal, char *cResult, uint8_t nDigs, char cLeftPad);
void IO_Initialization(void);
void CalcSolarTime(char *sSolarTime);
void dly_10us(void);
void dly_100us(void);
void dly_1ms(void);
void dly_10ms(void);
void dly_100ms(void);
void dly_1hS(void);
void dly_1s(void);
void dly_1sTM(void);

uint8_t NextAction(char back_forth);
char ChkXtended_menuaction(void);
void TDC_Trigger(void);
void TDC_ReadTrigger(void);
void ADC_Trigger(uint8_t xChannel, uint8_t xSamplMode);

uint32_t CurTime2Secs(void);
uint8_t IsLeapYear(uint16_t xYear);
uint8_t MonthDays(uint8_t xMonth, uint16_t xYear);
uint16_t YearDays(uint16_t xYear);

uint32_t CurTime2Secs(void);
double FractionalYear_B(uint16_t xYearDay, uint8_t xHour);
double EQofTime(double xB); //xB is fractonal Year in radians
void BlinkDot(uint8_t *lToggle, uint8_t line, uint8_t col);
uint8_t Dec2Bin(uint8_t dDig);
uint8_t BIT(uint16_t xword, uint8_t bitPos);
float GetMaxtemperature(void);
int CheckFeedbackError(float processValue, float SetPointValue, float xError);
int CheckFeedbackErrorRgt(float processValue, float SetPointValue, float xError);
int CheckFeedbackErrorLft(float processValue, float SetPointValue, float xError);

uint8_t GetSemaphore(uint8_t semaphoreID, uint8_t semaPriorityNo); //semaPriorityNo must be 0 to 7
for 8 bits
//semaphore words, ...
void ReleaseSemaphore(uint8_t semaphoreID, uint8_t semaPriorityNo);
uint8_t AllTheOnesToTheRightOf(uint8_t bNumber); //bNumber must be 0 to 7 for 8 bits semaphore
words, ...
void SetSysFlag(uint8_t sysFlagUserID);
int sign(int);

uint8_t InitDataLog(uint8_t LogChecklist, uint8_t LogDestination);
uint8_t DataLog(uint8_t LogChecklist, uint8_t LogDestination);
void DataLogSendString(char *str, uint8_t sLen, uint8_t LogDestination);
void DataLogSendChar(char xChar, uint8_t LogDestination);
uint8_t FOut(uint8_t TrkMode);
```

```
//vars and consts;
//generic temporary scrachpad or debug purposes vars
static uint8_t tempByte;
double xTemp;
char *sTemp = "01234567890123456789", *dummyPtr;

uint8_t OscCalByte = 10;      //for Fcpu frequency calibration when using internal oscillator.
uint8_t EEMEM  OscCalByte4MHz;
uint8_t EEMEM  OscCalByte8MHz;

//following definition can be amended to greater values (width and/or lengt) according to new needs;
//the second byte is for reccording the current user.(when semaphore in busy state) ...
//... or  last (when free state).
uint8_t SysStatUserIDs = 0;
uint8_t Semaphores[16][2];

//System safety parameters - required and determined by the Safe Operating Areas (SOA) of the plant
//components and devices;

//in degrees celcius the lowest of Maximum temps. supported among the plant devices and conduits.
float MaxTemp = 250;
//float MaxWindLoad; //Max wind load for accurate and safe operation;
//the wind load safety control not implement in this version.

//-274.0: an initial invalid temp., only used for avoiding confusion with real temperatures.
float AmbientTemp_inCelsius = -274.0;

uint32_t timer0Tic=0, tempTic = 0, maxTic=0, synchroCnt=0, LastSampTic, TDC_SampTic;
uint16_t timer0Tic1k=0;
uint8_t timer0Tic_rehitFlag=0, timer0Tic_rehitCount=0;
uint8_t ADCtrig_cnt=0;
uint16_t ADCautoTrigCnt = 0;
uint8_t TDCtrig_cnt=0;
uint8_t intsRunning_cnt=0;
uint8_t ADC_xmuxAddr = 0, TDC_xmuxAddr = 0;
uint16_t ADCresult;
uint8_t toggle, lTicBlink=0;
uint8_t sregsave, ddrsave;
uint8_t lastScan =0, inScan = 0;
uint8_t lastKey =0, inKey = 0;
uint8_t lastSecs=0;
uint8_t lClearLCD;

float LatitudeAngleLoc = -(29.0+49.0/60.0+2.0/3600.0);//29deg49'2.0"S; for solar plant at roof  ukzn
float LongitudeAngleLoc = 30.0+56.0/60.0+40.0/3600.0 ;//30deg56'40.0"E  //
uint8_t TimezoneLoc = 2.0;    //for solar plant at roof  ukzn = for all RSA
char *sLocalTime="Su,161108,072030", *sSolarTime="0000064530";
char *sLogLocalTime;
char sWrRTCtime[17] = {'s','s','m','m','h','h','d','d','M','M','0','D','y','y','0','0'};
char *ssWrRTCtime = "ssmmhhddMM0Dyy0000";
uint8_t dWrRTCtime[10] = {0x00,0x12,0x22,0x21,0x06,0x01,0x09,0x00,0,0};
uint16_t year;
float SolarTime_inMinutes, LocalTime_inMinutes;
uint16_t CurDayOfYear;
double FractionalYear, EqOfTime_Minutes;
float acceptanceAngleDg = 0.5; //concentrator's acceptance angle in degrees
uint16_t trackResolFine = 0.25;
float StHourAngle, StDeclAngle, StDeclAngleDeg; //Solar time calculated hour and declination angles.
float StPtHourAngle, StPtDeclAngle;//generic setpoint (desired=destination) hour and declin. angles.
float SafetyHourAngle=0, SafetyDeclAngle=-30;//safety position setpoint hour and declination angles;
//these may dinamically vary with specific location, weather and environm. conditions.
float RestHourAngle=0, RestDeclAngle=-30;  //Rest position setpoint hour and declination angles;
float ActualDeclAngle ; //Current (actual) tracker's  declination angle (readout from ADC);
float ActualHourAngle ; //Current (actual) tracker's hour angle (readout from ADC);
float MaxDeclAngle= 23.46;//degrees; maximum value for declin.angle; minimum value  is the simetric.
float MaxHourAngle = 180; //degrees; maximum value for hour angle; minimum value  is the simetric.
uint32_t ActualHourAngleSampTime, ActualDeclAngleSampTime;

float SunriseAngle;     //= -Sunset angle
uint8_t IsSolarMidNightFlag = FALSE, notSynchronized = FALSE;
uint8_t CheckSafetyTemperatureFlag = FALSE;
```

```
uint8_t collectorDefocusFlag = FALSE;
uint8_t trackingIntervalFlag = FALSE;
uint8_t ParkFlag = FALSE;




//tracker's
uint8_t curTrackersPWM_DC = 128; //Mid speed by default.
uint8_t TrackerMode = TRACK_MODE_IDLE;
uint8_t TrackersState;
float xSeekFineError, xStayCoarseError;
struct TrackerMachine
{
        //system state variables.     //see tracker.h for tracker related definitions
        uint8_t TrackersCTRLmode;     //FSM_OnOff/FSM_PID/Fuzzy FSM_PID
        uint8_t state;
        float HourAngle_SeekError;//Allowed seek error=Setpoint-Actualvalue,on seeking the setpoint;
        float HourAngle_StayError;//when error: within 1/2 the acceptance angle,tracking is stopped.
        float DeclAngle_SeekError;//Allowed seek error=Setpoint-Actualvalue,on seeking the setpoint;
        float DeclAngle_StayError;//when error <= SeekError, the tracking is stopped.

        //state outputs: //inputs for the tracker motor driver.
        uint8_t Forward;
        uint8_t Reverse;
        uint8_t AxisSelect;
};

struct ChgPumpStateData
{
        //system state variables. //see pumps.h for pumps related definitions
        uint8_t ChgPumpCTRLmode;
        uint8_t state;  //Idle, Auto or Manual;
        //state outputs:
        uint8_t Forward;
        uint8_t Reverse;//actually not used: Charging pump is set to forward only movement.
};

/*struct DisChgPumpStateData
{
        //system state variables. //see pumps.h for pumps related definitions
        uint8_t DisChgPumpCTRLmode;
        uint8_t state;  //Idle, Auto or Manual;
        //state outputs:
        uint8_t Forward;
        uint8_t Reverse;
};*/

struct PIDctrlData
{       //Hour axis, Declination axis and (dis)charging pump  PID variables
        uint32_t LastSampTime;
        float MaxValue;
        float MaxPWM;
        float LastError; //the previous error: to use for calculating I and D terms.
        float IntegralSum;     //the integral of the errors
        float Kp;              //Kp of PID angle control
        float Ki;              //Ki of PID angle control
        float Kd;              //Kd of PID angle control
};

//read from eeprom all at initialization.
uint8_t EEMEM EETrackersCTRLmode = TRACKER_CTRLMODE_FSM_ONOFF_ANGULARPOSITION;

//Hour Axis EEPROM settings
float EEMEM EEMaxHourAngle = 360*pi/180; //360 deg.
float EEMEM EEMaxHourPWM=255;
float EEMEM EEHourAngleKp = 100;
float EEMEM EEHourAngleKi=0;
float EEMEM EEHourAngleKd=0;
float EEMEM EEHourAngle_SeekError = 0.25;
float EEMEM EEHourAngle_StayError = 0.5;

float EEMEM EEHourAngularSpeedKp=1;   //Kp of PID angular velocity control;
float EEMEM EEHourAngularSpeedKi=0;   //Ki of PID angular velocity control
float EEMEM EEHourAngularSpeedKd=0;   //Kd of PID angular velocity control
float EEMEM EEMaxHourAngularSpeed = 1;  //1 deg per second.
```

```
//Declination Axis EEPROM settings
float EEMEM EEDeclAngleKp = 1;
float EEMEM EEDeclAngleKi=0;
float EEMEM EEDeclAngleKd=0;
float EEMEM EEDeclAngle_SeekError = 0.25;
float EEMEM EEDeclAngle_StayError = 0.5;
float EEMEM EEMaxDeclAngle;
float EEMEM EEMaxDeclAngularSpeed;
float EEMEM EEMaxDeclPWM = 255;

//Charging Ppump EEPROM stored variables
uint8_t EEMEM EEChgPumpCTRLmode = CHGPUMP_CTRLMODE_PID_ANGULARSPEED;
float EEMEM EEChgPpMaxSpeed = 2*pi*81.4; //2pi rev/sec:  yields a max flowrate value of 0.5l/sec.
float EEMEM EEChgPpMaxPWM = 255;
float EEMEM EEChgPpSpeedKp = 1;
float EEMEM EEChgPpSpeedKi = 0;
float EEMEM EEChgPpSpeedKd = 0;

//Discharging Ppump EEPROM settings
/*uint8_t EEMEM EEDisChgPumpCTRLmode = DISCHGPUMP_CTRLMODE_PID_ANGULARSPEED;
float EEMEM EEDisChgPumpKp = 1;
float EEMEM EEDisChgPumpKi=0;
float EEMEM EEDisChgPumpKd=0;
float EEMEM EEDisChgPpMaxSpeed = 2*pi*81.4; // 2pi*rev/sec.
float EEMEM EEDisChgPpSpeedKp = 1;
float EEMEM EEDisChgPpSpeedKi = 0;
float EEMEM EEDisChgPpSpeedKd = 0;*/

;//these prototypes could not be declared before the struct declaration.
void FSMtrack(struct TrackerMachine *Tracker);
void FFSMtrack(struct TrackerMachine *Tracker);
void    ResetTrackerState(struct TrackerMachine *Tracker, uint8_t resetSource);
float PIDctrl(float StPtValue, float ActualValue, uint32_t sampTime, struct PIDctrlData *pidData);
void ResetPIDdata(float inMaxValue, int inMaxPWM, float inKp, float inKi, float inKd, struct
PIDctrlData *pPIDdata, uint8_t resetSource);

uint8_t ChgPumpFBK_bitStatus = 0, ChgPpSpeedIncrement = 0;
uint8_t curChgPumpPWM_DC=0, ChgPumpMinPWM_DC=10;
uint32_t ChgPump_curTic, LastChgPpSampTime;
uint16_t ChgPumpRPScount = 0, ChgPumpRPSLastcount, ChgPumpRPMcount = 0, ChgPumpRPMLastcount;
float ActualChgPumpSpeed, StPtChgPumpSpeed, MaxChgPumpSpeed;
uint8_t ChgPumpIntervalFlag = FALSE;

uint8_t DisChgPumpFBK_bitStatus = 0, DisChgPpSpeedIncrement = 0;
uint8_t curDisChgPumpPWM_DC=0, DisChgPumpMinPWM_DC=10;
uint32_t DisChgPump_curTic;
uint16_t DisChgPumpRPScount = 0, DisChgPumpRPSLastcount;
uint16_t DisChgPumpRPMcount = 0, DisChgPumpRPMLastcount;
float ActualDisChgPumpSpeed, StPtDisChgPumpSpeed, MaxDisChgPumpSpeed;
uint8_t DisChgPumpIntervalFlag = FALSE;

char menu_action = ASCII_one, lastAction;
int8_t cur_DispAction = 0;
uint8_t disp_actions[7]={ASCII_one,ASCII_two,ASCII_three,ASCII_five,ASCII_seven,
ASCII_eight,ASCII_nine};
#define lBACK 0
#define lFORTH 1

uint8_t KeybBuffInput_IsON = FALSE, KeybBuffIndex=0;
char KeybBuffer[18];
//nKey order: according to the maping of the key on the keypad. See data sheet
char nKey[16] = {0x33, 0x32, 0x31, 0, 0x36, 0x35, 0x34, 0, 0x39, 0x38, 0x37, 0, '#', 0x30, '*'};
/* char aKey[7][16] = {  // this is for possible implementation of multi-stroke accessed chars.
       {' ', 'A', 'D', 0, 'G', 'J', 'M', 0, 'P', 'T', 'W', 0, '#', '.', '*'},
       {'-', 'B', 'E', 0, 'H', 'K', 'N', 0, 'Q', 'U', 'X', 0, '@', ',' ,'/'},
       {'=', 'C', 'F', 0, 'I', 'L', 'O', 0, 'R', 'V', 'Y', 0, '&', ';', '+'},
       {'_', 0xE0,'É', 0, 'Í', '$', 'Ó', 0, 'S', 'Ú', 'Z', 0, '[', ':', '-'},
       {'^', 0xE2,'"', 0, ':', '¥', 'Ó', 0, '®', 'Ú', '%', 0, ']', '?',0x5C},
       {'`', 'ª', '"', 0, 'Í', '£', 0xDF,0, '©', 'Ú', '<', 0, '{', '!', '('}, //0xDF = degree
       {0x27,'º', 'É', 0, 0x7F,'¢', 0x7E,0, 'r', 'Ú', '>', 0, '}', '|', ')'} //0x27=sngl qte,
                                                                              // 0x7E-F arrows
};*/

uint8_t TDC_SPI_transferStatus = OFF;
uint8_t TDC_SS_toggleStatus = OFF;
uint8_t TDC_is1stByte = TRUE;
```

```
static uint16_t TDCword, lastTDCword;
float TDC_LastSample[2][64]; //64 tdc channels
float TDC_CalibrOffset = 15.5; //Measured TDC offset error with Maxim MAX397 mux;
//No error observed on direct thermocouple connection to TDC
uint8_t TDC_muxAddress=0;
uint32_t TDC_SampleCounter=0; //increments 1 for each sample
float dummySpacer[4];




//Vishay-Spectrol smart (angular) position sensor settings; //now, not used.
//float AVC_Vmax = 4.87; //volt
//float AVC_Vmin = 0.13; //volt

//PB050 angular position  sensing, rotary single/turn linear pot settings;
float AVC_Vmax = 5.00; //volt
float AVC_Vmin = 0.10; //volt

//MCU ADC settings;
#define ADC_Vref 5  //5 Volts
#define ADCmax 1024 //the value for a Vin = Vref;

//ADC logging
uint32_t ADC_LastSample[2][16]; //for 16 ADC channels
uint8_t ADC_muxAddr=0;

uint8_t EEMEM EELogChecklist = CHECKLIST1;   //Logging profile: checklist and destination in EEPROM
uint8_t EEMEM EELogDestination = TO_RS232;   //Logging profile: checklist and destination in EEPROM
uint8_t DataLogFlag = FALSE, notLogged = FALSE, notInitLogged=TRUE;
//, ADCnotLogged=FALSE, TDCnotLogged=FALSE;
uint16_t DatalogInterval = 100, xDatalogInterval = 100;     //defaults to 1 sec interal

/*

Functions
*/
#include "datalogger.c"

//system malfunction status flags handling
void SetSysFlag(uint8_t sysFlagUserID) {
        SysStatUserIDs |= _BV(sysFlagUserID);
}

/* Semaphores handling functions */
void InitSemaphores(void) //releases all semaphores
{
        uint8_t i;
        for (i=0; i<16; i++){
                Semaphores[i][0] = 0;
                Semaphores[i][1] = 0;
        }
}

//This func returns a Nr that is all 1's at right of the priority bits: [2^(semaPriorityNo-1) -1]
uint8_t AllTheOnesToTheRightOf(uint8_t bNumber){ //bNumber must be 0 to 7 for 8 bits semaphore words
        uint8_t retval;
        retval = ~((~_BV(bNumber))+1);
        return retval;
}

uint8_t GetSemaphore(uint8_t semaphoreID, uint8_t semaPriorityNo)
{//semaPriorityNo must be 0 to 7 for 8 bits semaphore words, ...
        uint8_t retVal;
        if      ((Semaphores[semaphoreID][0] & AllTheOnesToTheRightOf(semaPriorityNo)) == 0)
        {       //if no bit set on the right of semaPriorityNo, ...
                //...then no one with higher priority is currently using the resource ...
                //if there are pending requests (at left) => someone recently released the resource.
                Semaphores[semaphoreID][0] |= _BV(0); // set the busy flag asap.
                Semaphores[semaphoreID][0] |= _BV(semaPriorityNo); //set userID/priority bit
                 //now, identify yourself as the ower of current access.
                Semaphores[semaphoreID][1] = _BV(semaPriorityNo);
                retVal = TRUE; //grant the resource
        }
        else // someone is already using the resource: cant grant the resource;
        {
```

```
                retVal = FALSE; //not granted
                Semaphores[semaphoreID][0] |= _BV(semaPriorityNo); //anyway, place your request.
        }
        return retVal;
}

void ReleaseSemaphore(uint8_t semaphoreID, uint8_t semaPriorityNo)
{
        //clear the respective userID/priority bit ...
        Semaphores[semaphoreID][0] &= ~_BV(semaPriorityNo);
        Semaphores[semaphoreID][0] &= ~_BV(0);//... as well as the resource busy flag.
}


//-----------------------------------------------
// input/out initialization routine
void IO_Initialization(void)
{
        uint8_t dummyReg, i;

        /* lets work with external osc for now
        #if (F_CPU == 8000000UL)      //settings to be recorded only for 4 and 8MHz
                OscCalByte = eeprom_read_byte(&OscCalByte8MHz);
        #elif (F_CPU == 4000000UL)
                OscCalByte = eeprom_read_byte(&OscCalByte4MHz);
        #else
        #       error "specified cpu frequency not supported!"
        #endif
        OSCCAL=OscCalByte;//this is calibration byte; should be written to eeprom at MCU programming
         */

        cli();  //desable interrupts
        InitSemaphores(); //release all semaphores

        //Port direction default initialization //see main control board signals
        DDRA = 0xFE; //bit 0 is ADC_in:input; rest outputs
        DDRB = 0xBC; //
        //lowers SS SPI line, to be ready for the upcoming start conversion command
        TDC_DataPort &= ~_BV(TDC_SS);
        DDRC = 0xFF; //
        PORTC = 0;
        DDRD = 0xF6;
        PORTD &= 0xF6;   //disable pullups on port D


        // keyboard (INT1) initialization
        inScan = 0; // default scan code (= no key pressed);
        //MCUCR &= ~_BV(ISC11); //ISC11:ISC10=01 => Any logic change triggers an int1.
        MCUCR |= _BV(ISC11); //ISC11:ISC10=11 => rising edge triggers an int1.
        MCUCR |= _BV(ISC10);
        //GICR &= ~(_BV(INT0) | _BV(INT2));// desable external INT0 and INT2
        GICR |= _BV(INT1);     //enable external INT1 pin interrupts (from keyboard)

        //TDC - Thermocouple to Digital Converter (MAX6675) interface initialization;
        //MAX6675TDC as slave; MCU as master; SS controlled by software; 16 (8x2) bit frame
        //bit7=1 SPIE:enable SPI ints; bit6=1 SPE:enable SPI; bit5=0 DORD:shift order MSB 1st;
        SPCR = 0xD4;//bit4=1 enable master mode; bit3=0 rise to fall polarity; bit2=1 SCK phase:=>
        //=>read data on trailling edge(to follow MAX6675 protocol); bits1-0: SPI prescaler selector
        //for SCK in SPI master mode.Ignore in slave mode. set to 00 for  Fosc/4 in master mode:
        //this yields fsck=4MHz max for fosc=16MHz; 4.1MHz is max freq for MAX6675 data collection.
        SPSR  &=  ~_BV(SPI2X); //clear SPI2X: so that even if SPR1:SPR0=11 (fosc/128) =>
                               //=> we insure tdcCLK <= 4MHz for fOSC<=16MHz
        dummyReg = SPSR;
        dummyReg = SPDR;       //by reading SPISR and SPIDR we clear SPI IFlag
        SPCR |= _BV(SPE) | _BV(SPIE);         //enable SPI along with, enable SPI interrupts
        for (i=0; i<40; i++)
        {
                TDC_LastSample[0][i]=-276;
                TDC_LastSample[1][i]=0;
        }
        TDCtrig_cnt = 0;
        TDC_muxAddress = 0;
        TDC_SampleCounter = 0;


        //ADC ---
        // ADC dataacq.is done at higher freqs than the 1KHz of the task triggering Tmr0 interrupt.
```

```
        // this gave a better response for the need of many externally multiplexed channels.
        ADC_init(); //this initializes ADC interrupt handling routines.


        //Timer 1 PWM initialization
        //Tracker's motors Trackers_PWM0 on Timer1 (on OC1B/PD4)
        DDRD |= _BV(PORTD4); //set Trackers_PWM0 = OC1B/PD4 data dir to output
        DDRD |= _BV(PORTD5); //set Discharge pump_PWM 0n OC1A/PD5 data dir to output
        DDRB |= _BV(PORTB3); //set Trackers_PWM1 on OC0/PB3 data dir to output;
        TCCR1B = 0; //stop timer1 => stop motors
        PORTB &= ~_BV(PORTB3); //clear Trackers_PWM1 bit on PORTB3; => stop motors;
        OCR1A = 0;// PWM_DC =0  => OC1A output low; that means stop motors or keep them stoped;
        OCR1B = 0;// PWM_DC =0  => OC1B output low; that means stop motors or keep them stoped;
        TCCR1A = 0xA1; //8 bit, Non inverting, Phase correct PWM; => ...
        TCCR1B = 0x01; // => PWM_Freq=15,686.275Hz @ 8MHz fcpu.no precaling;
        //The PWM freqs for other fcpus's: just mul these PWMfreqs by their ratio to 8.
               //TCCR1B = 0x02; // => PWM_Freq=1960.78Hz @ 8MHz fcpu. ; prescal. 8
               //TCCR1B = 0x03; // => PWM_Freq=490.20Hz @ 8MHz fcpu. ; prescal. 32
               //To control OC1A PWM_DC and thence the motor speed ...;
                      //... just vary OCR1A between 0 (stoped) and 255 (full speed);
               //To control OC1B PWM_DC and thence the motor speed ...;
                      //... just vary OCR1B between 0 (stoped) and 255 (full speed);


        //Charge pump PWM on Timer2
        DDRD |= _BV(PORTD7);//set Charge pump motor PWM control pin (OC2) data dir bit to output;
        TCCR2 = 0; //stop timer2 => stop Charge pump motor
        OCR2 = 0;       // PWM_DC =0  => OC2 output low; that means stop motor or keep it stoped;
        TCCR2= 0x61;//Non inverting Phase correct PWM, no prescaling; => PWM_Freq=15,686.27Hz @ 8MHz
        //TCCR2=0x62;//Non inverting Phase correct PWM,precaler 8;=> PWM_Freq=1,960.78Hz@ 8MHz fcpu.
        //TCCR2=x63;//Non inverting Phase correct PWM, precaler 32;=> PWM_Freq=490.20Hz@ 8MHz fcpu.
        //TCCR2=0x64;//Non inverting Phase correct PWM, precaler 64;=> PWM_Freq=245.10Hz@ 8MHz fcpu.
        //TCCR2=0x65; //Non inverting Phase correct PWM, precaler 128; => PWM_Freq=122.55Hz @ 8MHz.
        //TCCR2=0x66;//Non inverting Phase correct PWM, precaler 256;=> PWM_Freq=61.27Hz@ 8MHz fcpu.
        //TCCR2= x67; //Non inverting Phase correct PWM, precaler 1024; => PWM_Freq=15.32Hz @ 8MHz.
        //To control OC2 PWM_DC and thence the motor speed...
        // ... just vary OCR2 from 0 (stoped) to 255 (full speed);

        //Timer0: timer tic interrup: for task triggering, synchronizing and coordination;
        //must be the last initialization
        //Initialize the 8bit Tmr/Cntr0 for 1KHz int tic to be used to synchronously trigger events.
        TCCR0 = 0; //stop timer0
        // see preload counts defs; //!:remember to do this preload on the timer0 ISR at each tic;
        TCNT0 = TIMER0_PRELOADCNT;
        SFIOR |= _BV(PSR10);   //reset prescaller (for timer 0 and this is also for timer 1)
        timer0Tic = CurTime2Secs() * 100; //synchronize with Real time clock.
        TCCR0 = 0;      //normal mode
        TIMSK |= _BV(TOIE0); //enable interrupts on timer0 overflow
        //TCCR0 = 0;    //normal mode; prescaller = 0 //no clock = timer stopped;
        //TCCR0 = 1;    //normal mode; prescaller = 1 //start timer 0 with no prescaling;
        //TCCR0 = 2;    //normal mode; prescaller = 8 //start timer 0 with 8 prescaling divider;
        TCCR0 = 3;      //normal mode; prescaller = 64 //start timer with 64 prescaling;
        //TCCR0 = 4;    //normal mode; prescaller = 256 //start timer with 64 prescaling;
        //TCCR0 = 5;    //normal mode; prescaller = 1024 //start timer with 64 prescaling;
        //TCCR0 = 6;    //normal mode; external clock at falling edge
        //TCCR0 = 7;    //normal mode; external clock at rising edge

        //set the timertic rehit verification counter to zero.
        // This is incremented by 1 at entrance of timer0tic ISR and
        // decremented by 1 at exit. If it hapens this counter is greater then 1 =>
        // => there are further tics rehiting the ISR, before the prior hit exiting.
        timer0Tic_rehitFlag = 0;
        timer0Tic_rehitCount = 0;
        timer0Tic1k=65001;     //this insures it will be reset to 0 at first int service.
        synchroCnt = 0;        //for RTC and internal timer tic periodic realignment functionality
        intsRunning_cnt=0;     //used for allowing/desallowing keyb int handling.

        DataLogFlag = FALSE;

        //all startup initiallization done, so ...
        sei();  //globally enable ints. This fires the real "starting of the machine".
        //
}


//timer0 overflow interrupt: The Timer Tic interrupt. Configured to hapen at 1KHz frequency
```

```
//This is the task control loop that timedly sets flags for periodic tasks and /or
//timedly reads or modifies system variables.
ISR(TIMER0_OVF_vect)
{
        uint8_t bitStat;

        //these 2 vars monitor the system timing delays;
        intsRunning_cnt++;
        timer0Tic_rehitFlag++;
        TCNT0 = TIMER0_PRELOADCNT;    //reload Timer init count
        timer0Tic1k--;

        if (timer0Tic1k ==0) {
                timer0Tic1k = 65000; //max 16 bit number multiple of 1000
        }



        //trigger 1 milisecond (higher time) resolution activities
        //read Pump motors feedback speeds
        //to this pt, only charging pump is processed.
        if ((timer0Tic1k % 1000) == 0 )       //at the beginning of each 1 second period
        {
                ChgPump_curTic = 0;
                ActualChgPumpSpeed = ChgPumpRPScount;
                LastChgPpSampTime = timer0Tic1k;             //verify rehits
                //ChgPumpRPSLastcount = ChgPumpRPScount;
                ChgPumpRPScount = 0;
                //ActualChgPumpSpeed = ChgPumpRPSLastcount;
                /*if (((timer0Tic1k - 1) % 60000) == 0 ) //beginning of each 60 seconds period
                {
                        ChgPumpRPMLastcount = ChgPumpRPMcount;
                        ChgPumpRPMcount = 0;
                }*/
        }
        bitStat = PINB & _BV(MOT_ChgPump_FeedbackBit); // check the chg pump bit status
        if ((ChgPumpFBK_bitStatus) && (bitStat == 0)) //if true, falling edge detected
        {
                ChgPumpRPScount++;
                //ChgPumpRPMcount++;
        }
        ChgPumpFBK_bitStatus = bitStat;

        if ((timer0Tic1k % 10) == 0 ) //i.e. at 100Hz
        {       //trigger 10 milisecond lower time resolution activities
                timer0Tic++; //increment the 100Hz timer0 tic counter.
                //Acquisition / monitoring tasks, are before control tasks;
                if (((timer0Tic) % 25) == 0) { //4Hz action:trigger TDC conversion
                        LastSampTic = timer0Tic;

                        if (TDCtrig_cnt == 0){ //this var controls TDC triggering synchronization
                                        //in the event of system delays, it avoids TDC retriggering.
                            TDCtrig_cnt++;
                             TDC_Trigger(); //triggers the TDC; 220ms conversion time =  22 tics.
                        }
                }
                if (TDCtrig_cnt == 255){      //no semaphore last time: try again.
                        TDCtrig_cnt=1;
                        TDC_Trigger();//trigger thermocouple to DC; 220ms conversion time =  22 tics.
                }
                 //after 23tics (230mS), read TDC.
                if (((TDC_SampTic+23) == timer0Tic )  && (TDCtrig_cnt==2)){
                        TDC_ReadTrigger();
                }

                // control tasks: process read / logged inputs and actuate on the relevant outputs
                // this is done by seting flags, which are used by the main normal loop ..
                //...for firing the respective tasks

                //Activities:
                //A.Safety activities: highest priority control activities
                //1. Defocus and brake: if any temperature eq or exceeds MaxTemp: the system ..
                // ... maximum safe temp (within SOA).
                if  ((timer0Tic % 100) == 0)  {
                        {CheckSafetyTemperatureFlag = TRUE;}
                }//endif
                //2. Defocus and brake: in the event of a wind load superior to MaxWindSpeed;
```

```
                //  not implemented in this version;
                //B. Track sun:
                //1. If within day time => track;
                //2.    Reposition to sunrise: If time >= sunset and not in sunrise position
                if ((timer0Tic % 100) == 0)
                { //in 1 secs intervals: trigger tracking action.
                        trackingIntervalFlag = TRUE;
                }
                //control pumps speeds
                if ((timer0Tic % 100) == 0)//at freq of 1 Hz:trigger charg.pump speed control action.
                        {ChgPumpIntervalFlag = TRUE;}
                if ((timer0Tic % 100) == 0)  //in 1 secs intervals (freq of 1/2 Hz):
                        //trigger discharging pump speed control action.
                        {DisChgPumpIntervalFlag = TRUE;}
                if (((timer0Tic % DatalogInterval) == 0) && (notLogged)) {
                        DataLogFlag = TRUE; //(ADCnotLogged && TDCnotLogged);
                }
        }


        //then predecrement and test the mis-synchronization audit flag.
        if (--timer0Tic_rehitFlag != 0) {//if !=0 => excessive delay inside timer tic ISR code.
                timer0Tic_rehitCount++;
                SetSysFlag(SYS_STAT_TIMERTIC);
        } //endif
        intsRunning_cnt--;
}


/* This form of declaration - instead of the macro ISR(), instructs the compiler to place a SEI just
at the
* begining of the handler, to allow other ints to occur. All ints (except the keyboard's INT1) are
of
* absolute "no miss" need. A flag is used to signal INT1 handler to relinquish the processor control
* if another int handler is running. Keyb ints are asynchronous, sporadic events;
*/
void INT1_vect(void) __attribute__((interrupt));
void INT1_vect(void)
{
        uint8_t chkScan, i;

        if (intsRunning_cnt==0)
        {  //if no higher priority interrupt is currently runing, process keystroke.
                chkScan = KEYB_CTRLPORT;
                if (KEYB_CTRLPORTPIN & _BV(KEYCODE_AVAILABLE)) { //is a keystroke available?
                        //KEYB_DATAPORT &= 0x0E;
                        KEYB_DDRPORT &= 0x0E; //set bits 7:4 of keybort to inputs

                        KEYB_CTRLPORT &= ~_BV(KEYB_OEN);  //output enable the keyb encoder
                        chkScan = 0;
                        //chkScan |= (PINA);   //read scancode
                        chkScan = (PORTA);     //read scancode

                        lastScan = inScan;
                        inScan |= KEYB_DATAPIN;        //read from pin port
                        KEYB_CTRLPORT |= _BV(KEYB_OEN);//set back the kbencoder outs to HiZ
                        KEYB_DDRPORT = 0xFE;


                        inScan = (inScan >> 4);        //scan code is in hi nible
                        inScan &= 0x0F;                //strip and store
                        lastKey = inKey;
                        inKey = nKey[inScan];
                        lastAction = menu_action;

                        if ((lastKey == inKey) && inKey == ASCII_star) { //let double star be = ESC;
                                KeybBuffInput_IsON = FALSE;//abort current keyb input mode, and ...
                                inKey = ASCII_one;             //... default to menu option 1.
                        }
                        if (KeybBuffInput_IsON) { //Keybord input in buffer mode, in progress.
                                if ((KeybBuffIndex >= 16) && (inKey != ASCII_hash)
                                //keyb input buffer is 16+1 chars long.
                                {// shift down the keyboard buffer: discard oldest key;
                                        for (i=0; i<15; i++){
                                                KeybBuffer[i] = KeybBuffer[i+1];
                                        }
                                        KeybBuffIndex = 15;
```

```
                        }
                        KeybBuffer[KeybBuffIndex++] = inKey;
                        KeybBuffer[KeybBuffIndex] = 0;
                        if (inKey == ASCII_hash) {
                                inKey = CR;
                                menu_action = ChkXtended_menuaction();
                                KeybBuffInput_IsON = FALSE;
                        }
                }
                else { // normal processing
                        lClearLCD = (inKey == ASCII_zero) ? 1:0;
                        //lastAction = menu_action;
                        menu_action = inKey;
                } //endif

        } //endif
    } /* else: there is an int handler running: relinquish CPU control. */
}



//extended keyboard command processor
//this processes commands of type *nn#, where nn is the command; * the prolog and # the epilog.
//When it detects a good command sequence, it assigns to a single byte m_action ...
// .. required by the menu handling section of the main loop.
char ChkXtended_menuaction(void){
        //char m_action = ASCII_one;
        char m_action = menu_action;
        inKey = 0;
        if (strncasecmp ((const char*)KeybBuffer, "*01#",4) == 0) {
                m_action = 'C';
                inKey = 'C';
        }
        else if (strncasecmp ((const char*)KeybBuffer, "*02#",4) == 0) {
                m_action = '2';
                inKey = '2';
        }
        else if (strncasecmp ((const char*)KeybBuffer, "*03#",4) == 0) {
                m_action = '3';
                inKey = '3';
        }
        else if (strncasecmp ((const char*)KeybBuffer, "*04#",4) == 0) {
                m_action = '4';
                inKey = '4';
        }
        else if (strncasecmp ((const char*)KeybBuffer, "*05#",4) == 0) {
                m_action = '5';
                inKey = '5';
        }
        else if (strncasecmp ((const char*)KeybBuffer, "*06#",4) == 0) {
                m_action = '6';
                inKey = '6';
        }
        else if (strncasecmp ((const char*)KeybBuffer, "*07#",4) == 0) {
                m_action = '7';
                inKey = '7';
        }
        else if (strncasecmp ((const char*)KeybBuffer, "*08#",4) == 0) {
                m_action = '8';
                inKey = '8';
        }
        else if (strncasecmp ((const char*)KeybBuffer, "*09#",4) == 0) {
                m_action = '9';
                inKey = 0;
        }
        else if (strncasecmp ((const char*)KeybBuffer, "*00#",4) == 0) {
                m_action = 'A';
                inKey = 'A';
        }
        else if (strncasecmp ((const char*)KeybBuffer, "*20#",4) == 0) {
                m_action = 'e';
                inKey = 'e';
        }
        else if (strncasecmp ((const char*)KeybBuffer, "*21#",4) == 0) {
                m_action = 'E';
                inKey = 'E';
```

```
        }
        else if (strncasecmp ((const char*)KeybBuffer, "*25#",4) == 0) {
                m_action = 'L';
                inKey = 'L';
        }
        else if (strncasecmp ((const char*)KeybBuffer, "*55#",4) == 0) {
                m_action = 'M';
                inKey = 'M';
        }
        else if (strncasecmp ((const char*)KeybBuffer, "*99#",4) == 0) {
                m_action = 'P';
                inKey = 'P';
        }
        else {
                inKey = CR;
        }
        return m_action;
}




//arrow keys (4 and 6) context menu processeor
uint8_t NextAction(char back_forth){
        uint8_t retAction;
        if (lastAction == ASCII_three)
        {
                ActualHourAngle +=  ((back_forth == lBACK) ? -10: +1);
                retAction = ASCII_three;
        }
        else if (lastAction == ASCII_seven)
        {
                ChgPpSpeedIncrement =  ((back_forth == lBACK) ? -1: +10);
                retAction = ASCII_seven;
        }
        else if (back_forth == lBACK) {
                cur_DispAction--;
                if (cur_DispAction < 0) {cur_DispAction = 0;}
                retAction = disp_actions[cur_DispAction];
        }
        else
        {
                cur_DispAction++;
                if (cur_DispAction > 6) {cur_DispAction = 6;}
                retAction = disp_actions[cur_DispAction];
        }
        return retAction;
}


// finds the current maximum temperature, for saffety control.
float GetMaxtemperature(void)
{
        uint8_t i=0;
        float maxTemp=0;
        for (i=0; i<16; i++)
        {
                maxTemp = ((maxTemp >= TDC_LastSample[0][i]) ? maxTemp : TDC_LastSample[0][i]);
        }
        return maxTemp;
}

void ADC_init(void)
{// this is called from IO_initialization, so, at this point, no need of semaphores.

        uint8_t i;
        uint8_t saveDDRC, savePORTC;
        uint8_t SamplingModeMask = 7; //free running mode by default

        DDRA = 0xFE; //ADC_inDDR = 0xFE; //bit 0 is ADC_in:input; rest outputs
        ADC_inPort &= ~_BV(ADC_inBit);   //desable pullup
        SFIOR &= SamplingModeMask; //actually noefect with ADATE = 0;
        //ADMUX = 0x1E; //band gap ref;
        ADMUX = ADC_ChannelZero_AVCCref;//set internal ADC to channel 0 and Vref  to AVCC by default
```

```
        //ADCSRA bits: 7:ADEN; 6:ADSC; 5:ADATE; 4:ADIF; 3:ADIE; 2tru0: ADPS: ADC prescaller select
        ADCSRA = (1<<ADEN) | (7); //(ADC_prescalMASK) ;
        //ADCprescaller: bits 2:0: 7 is prescaller 128. valid for any fcpu>=8MHz

        ADCtrig_cnt=0;
        for (i=0; i<16; i++)
        {
                ADC_LastSample[0][i]=-276;
                ADC_LastSample[1][i]=0;
        }

        //check semafore for accessing portC
        while (GetSemaphore(SEMAF_PORTC, PORTCUSER_ADCADDRESS)==FALSE) {;}
        saveDDRC = DDRC;
        savePORTC = PORTC;
        DDRA = 0xFE;
        DDRC = 0xFF;
        PORTC = 0;  //select external channel 0
        asm("nop"); //wait at least 100nS for 374 setup time
        asm("nop");
        asm("nop");
        asm("nop");
        ADC_xmuxAddrClockPort |= _BV(ADC_xmuxAddrClockBit);
        asm("nop"); //wait at least 50nS for 374 clock width time
        asm("nop");
        asm("nop");
        asm("nop");
        ADC_xmuxAddrClockPort &= ~_BV(ADC_xmuxAddrClockBit);

        asm("nop");//wait at least 100nS for 374 pd and transition times
        asm("nop");//and wait at least another 400 for DG40x transition time
        asm("nop");
        asm("nop");
        asm("nop");
        asm("nop");
        asm("nop");
        asm("nop");
        asm("nop");
        asm("nop");
        asm("nop");
        asm("nop");
        DDRC = saveDDRC;
        PORTC = savePORTC;
        ReleaseSemaphore(SEMAF_PORTC, PORTCUSER_ADCADDRESS); //release portC semafore
        ADCautoTrigCnt = 0;
        ADC_xmuxAddr = 0;
        ADCSRA  |= ((1<<ADIE) | (1<<ADSC)) | (1<<ADATE); // enable ADC int and  start conversion
}

ISR(ADC_vect)
{
        static uint8_t okToNext = FALSE;
        uint8_t saveDDRC, savePORTC;
        intsRunning_cnt++;
        ADCresult = ADCW;
        ADCtrig_cnt--;
        if (((timer0Tic1k % 31) == 0) || okToNext){
                ADCautoTrigCnt=0;
                //check semafores for accessing portA and C
                if ((GetSemaphore(SEMAF_PORTC, PORTCUSER_ADCINT)))
                {
                        ADC_xmuxAddr++;
                        if (ADC_xmuxAddr > 15) {ADC_xmuxAddr = 0;};
                        saveDDRC = DDRC;
                        savePORTC = PORTC;
                        okToNext = FALSE;
                        PORTC = ADC_xmuxAddr;  //select external channel
                        asm("nop"); //wait at least 100nS for 374 setup time
                        asm("nop");
                        asm("nop");
                        asm("nop");
                        ADC_xmuxAddrClockPort |= _BV(ADC_xmuxAddrClockBit);
                        asm("nop"); //wait at least 50nS for 374 clock width time
                        asm("nop");
                        asm("nop");
                        asm("nop");
                        ADC_xmuxAddrClockPort &= ~_BV(ADC_xmuxAddrClockBit);
```

```
                        asm("nop");//wait at least 100nS for 374 pd and transition times
                        asm("nop");//and wait at least another 400 for DG40x transition time
                        asm("nop");
                        asm("nop");
                        DDRC = saveDDRC;
                        PORTC = savePORTC;
                        //release portC semaphore
                        ReleaseSemaphore(SEMAF_PORTC, PORTCUSER_ADCINT);
                }
                else {
                        okToNext = TRUE;
                }
        }
        ADCautoTrigCnt++;
         //next, read and store at 10th autotrigger period...
         // ..  This assures stabilization after mux channel switching
         if (ADCautoTrigCnt == 10) {
                ADC_LastSample[0][ADC_xmuxAddr] = ADCresult;  // record the (single) result.
                ADC_LastSample[1][ADC_xmuxAddr] = timer0Tic1k/10; // record the sample time
        }
        else if (ADCautoTrigCnt == 12){
                //release portA semafore
                ReleaseSemaphore(SEMAF_PORTA, PORTAUSER_ADC_DATA);
        }
        ADCtrig_cnt++;
        intsRunning_cnt--;
}
//this function is intended to be called from the task scheduler ISR
void TDC_Trigger(void)
{
        uint8_t saveDDRC, savePORTC; //, saveDDRB, savePORTB;
        if (TDCtrig_cnt == 1) {
                //check semafores for accessing portA and C
                if ((GetSemaphore(SEMAF_PORTC, PORTCUSER_TDCADDRESS)))
                {
                        TDCtrig_cnt++;
                        saveDDRC = DDRC;
                        savePORTC = PORTC;
                        DDRC = 0xFF;
                        PORTC = TDC_muxAddress;  //select external channel
                        asm("nop"); //wait at least 100nS for 374 setup time
                        asm("nop");
                        asm("nop");
                        asm("nop");
                        PORTC |= _BV(TDC_xmuxAddrClockBit); //toggle address port clock line
                        asm("nop"); //wait at least 50nS for 374 clock width time
                        asm("nop");
                        asm("nop");
                        asm("nop");
                        PORTC &= ~_BV(TDC_xmuxAddrClockBit);
                        asm("nop");//wait at least 100nS for 374 pd and transition times
                        asm("nop");//and wait at least another 400nS for DG40x transition time
                        asm("nop");
                        asm("nop");
                        asm("nop");
                        asm("nop");
                        asm("nop");
                        asm("nop");
                        asm("nop");
                        asm("nop");
                        asm("nop");
                        asm("nop");

                        DDRC = saveDDRC;
                        PORTC = savePORTC;
                        //release portC semafore
                        ReleaseSemaphore(SEMAF_PORTC, PORTCUSER_TDCADDRESS);

                        //semaphores for SPI bus:
                        //semaphores conversation supressed: TDC is so far the only user of SPI;
                        //TODO: When other modules (like SDcard) will be present, ...
                        //... then semaphores should be used for accessing SPI.
                        //while (GetSemaphore(SEMAF_PORTB,PORTBUSER_TDC_SPIBUS)== FALSE){;}
                                        //wait until get semafore for accessing portB
                        //
                        DDRB = 0xBC;
                        //next line: raises SS SPI line this triggers MAX6675 TDC conversion
```

```
                        TDC_DataPort |= _BV(TDC_SS) ;
                        TDC_SampTic = timer0Tic;

                        TDC_SPI_transferStatus = OFF;
                        TDC_SS_toggleStatus = OFF;
                        TDC_is1stByte = TRUE;
                }
                else {
                        TDCtrig_cnt = 255;
                }
        }
        else { // a timing/coordination mismatch
                SetSysFlag(SYS_STAT_TDCTRIGGER);
        }
}

void TDC_ReadTrigger(void)
{
        uint8_t dummyReg;
        //uint8_t saveDDRB, savePORTB;

        if (TDCtrig_cnt == 2) {
                //semaphores conversation supressed: TDC is so far the only user of SPI;
                //TODO: When other modules (like SDcard) will be present, ...
                //... then semaphores should be used for accessing SPI.
                //while (GetSemaphore(SEMAF_PORTB, PORTBUSER_TDC_SPIBUS) == FALSE) {;}
                        //wait until get semafore for accessing portB
                DDRB = 0xBC;
                //release portB semafore
                //ReleaseSemaphore(SEMAF_PORTB, PORTBUSER_TDC_SPIBUS);

                //while (GetSemaphore(SEMAF_SPIBUS, SPIUSER_TDC) == FALSE){;}
                        //wait until get semafore for accessing SPI BUS

                TDC_SPI_transferStatus = ON;
                TDC_SS_toggleStatus = OFF;
                TDC_is1stByte = TRUE;

                TDC_DataPort &= ~_BV(TDC_SS); //lowers SS SPI line this enables SPIs slave transfer
                asm("nop");  //4 nops = 1uS at 4MHz; .5uS at 8; 250 nS at 16; 200ns at 20MHz;
                             //to insure tDV and tCC of MAX6675 (100nS); 2 nops could be enough
                asm("nop");
                asm("nop");
                asm("nop");

                dummyReg = SPSR;   //by reading SPISR and SPIDR we clear SPI IFlag
                dummyReg = SPDR;
                SPDR =  dummyReg;  //triger SPI
        }
        else { // a timing/coordination mismatch
                SetSysFlag(SYS_STAT_TDCREADTRIGGER);
        }
}


//SPI int handler: used for TDConverter;
//Also to be used for SD interface (this part not implemented).
ISR(SPI_STC_vect)
{
        static uint8_t tempReg;
        float xTempTDC;
        uint16_t xTDCword;

        intsRunning_cnt++;
        tempReg = SPDR;
                if (TDC_SPI_transferStatus)  //if SPI transfer triggered running then ...
                {
                        if (TDC_is1stByte)
                        {       // 1st byte was on SPI receive buffer (it is now in tempReg)
                                TDCword = 0;
                                TDCword |= tempReg;
                                TDCword = (TDCword     << 8);
                                TDC_SS_toggleStatus = ON;
                                TDC_is1stByte = OFF;
                                SPDR = tempReg; //trigger SPI read of 2nd byte of the 16 bit frame
                        }
                        else //Is2ndByte:
```

```
                {
                        TDCword |= tempReg;

                        if (TDCword & 0x0002) //bit 1 set => invalid device id=> invalid input
                        {//signal that by the invalid temperature of -274.
                                xTempTDC = -274.0;
                        }
                        else if (TDCword & 0x0004) //if bit 2 set=> thermocple inputs are open
                        {//signal by a zero abs (0 Kelvin is to  mean thermocouple leads open)
                                xTempTDC = -273.0;
                        }
                        else if (TDCword & 0x8000)//sign bit set (should be 0)=> invalid input
                        {//signal by an invalid temperature.
                                xTempTDC = -275.0;
                        }
                        else
                        {//strip bits 15 and 2tru0. //no need. could leave
                                TDCword = (TDCword & 0x7FF8);
                                xTDCword = TDCword;
                                xTempTDC = (xTDCword >> 5);  //recover the int part, and …
                                //add frac part: the bits 3 and 4 of TDCword
                                xTempTDC+=0.5 * BIT(TDCword,4) + 0.25 * BIT(TDCword, 3);
                                xTempTDC -= TDC_CalibrOffset; //deducts the offset error
                                lastTDCword = xTempTDC;
                        }
                        TDC_LastSample[0][TDC_muxAddress] = xTempTDC;
                        TDC_LastSample[1][TDC_muxAddress] = TDC_SampTic;

                        TDC_muxAddress++;
                        TDC_SampleCounter++;
                        //if beyond the last channel set back to channel 0
                        if (TDC_muxAddress>15) //39) //this vers.: only 16 (out of 64) active
                        {
                                TDC_muxAddress = 0;
                        }

                        notLogged = TRUE;
                        TDCtrig_cnt = 0;
                        //clear SPItransfer flag: no transfer in course
                        TDC_SPI_transferStatus = OFF;
                        TDC_SS_toggleStatus = OFF;
                } //endif
        }//endif
        //ReleaseSemaphore(SEMAF_SPIBUS, SPIUSER_TDC);
        //release semafore for accessing SPI BUS
        intsRunning_cnt--;

} // return from SPI int

//calculate solar time and thence related angles (hour and declination)
void CalcSolarTime(char *sSolarTime)
{
        uint32_t xYearDay, solHrs, solMins, solSecs; //, iSolarTime_inMinutes;
        double xxB;
        uint32_t xHour;
        char *curLocalTime = "0000120304060900";
        xHour = ((timer0Tic / 100 / 3600) % 24);
        xYearDay = (timer0Tic / 100 / 3600 / 24)+1;
        CurDayOfYear = (uint16_t)xYearDay;
        xxB = FractionalYear_B(xYearDay, xHour);
        //LocalTime_inMinutes = (timer0Tic/6000.0) - (xYearDay-1) * 24 * 60; timertic: right but..
        //.. missaligned with RTC. so, rather use time fromRTC:

        //wait until obtaining semafore for accessing portC
        while (GetSemaphore(SEMAF_PORTC, PORTCUSER_RTC_CLKIO) == FALSE) {;}
        //burst read Local time from the DS1302 RTC; store the readout into curLocalTime
        RdClockBurst(curLocalTime);
        ReleaseSemaphore(SEMAF_PORTC, PORTCUSER_RTC_CLKIO); //release portC semafore
        LocalTime_inMinutes = Dec2Bin(*curLocalTime)/60.0+ Dec2Bin(*(curLocalTime+1)) +
                        Dec2Bin(*(curLocalTime+2)) * 60;
        //Solartime in minutes = EQofTime/*minutes*/ + 4(Lstdmeridian-Lloc) + LocalTime/*minutes*/;
        EqOfTime_Minutes = EQofTime(xxB);
        SolarTime_inMinutes = EqOfTime_Minutes  + 4*(TimezoneLoc * 15 - LongitudeAngleLoc) +
                        LocalTime_inMinutes;
        if (SolarTime_inMinutes < 0) {
                SolarTime_inMinutes =  24*60 + SolarTime_inMinutes;  //avoid negative solar time.
        }
```

```
        else if (SolarTime_inMinutes > 24*60) {
                SolarTime_inMinutes -=  24*60;  //avoid a solar time > 24h.
        }
        StHourAngle = SolarTime_inMinutes / 4 - 180.0;  //once zero degrees is at solar noon
        //Spencer's Declination angle (in radians).
        StDeclAngle = 0.006918 - 0.399912 * cos(xxB) + 0.070257 * sin(xxB) - 0.006758 * cos(2*xxB)
                              +0.000907 * sin(2*xxB) - 0.002697*cos(3*xxB) + 0.00148*sin(3*xxB);
        StDeclAngleDeg = StDeclAngle*180/pi; //in degrees
        solHrs = SolarTime_inMinutes /60;
        solMins = (SolarTime_inMinutes - solHrs * 60);
        solSecs = (SolarTime_inMinutes  - solHrs * 60 - solMins) * 60;
        IsSolarMidNightFlag = (((solHrs+solMins+solSecs)==0) && (notSynchronized));
        notSynchronized = notSynchronized || (solSecs != 0);
        //convert to ascii solar time to the format: hhmmss for display and datalogging;
        I2A(&solHrs,sSolarTime,2,cPadZero);
        I2A(&solMins,sSolarTime+2,2,cPadZero);
        I2A(&solSecs,sSolarTime+4,2,cPadZero);
}

float SunsetHourAngle(void) //returns the sun set angle. in radians.
{
        return acos(-tan(LatitudeAngleLoc*pi/180) * tan(StDeclAngle));
}

double EQofTime(double xB) //xB is fractonal Year in radians
{
        double EoT=0; //in minutes

        //Duffie & Beckman's Spencer equation of time
        EoT = 229.2 * (0.000075 + (0.001868 * cos(xB)) - (0.032077* sin(xB)) -
                        (0.014615 * cos(2 * xB)) - (0.04089 * sin(2 * xB)));
        return EoT;
}
double FractionalYear_B(uint16_t xYearDay, uint8_t xHour)  //returns fractional year in radians
{       //the B of Spencer equations of Time and Declination in Duffie & Beckman's
        //double xB;
        //FractionalYear = ((2 * pi / 365.2425) * ((xYearDay - 81) + ((xHour - 12) / 24)));
//radians
        FractionalYear = 2.0 * pi  * (xYearDay - 1) / YearDays(year); //radians
        return FractionalYear;
}

uint32_t CurTime2Secs(void) //this converts current local time from MM:dd:hh:mm:ss format to
seconds.
{
        char *curLocalTime = "0000120304060900"; // "ssmmhhddMMwwyycc";
        //seconds, minutes, hours, date, month, day, year, ctrlreg;
        uint8_t secs, mins, hours, date, month, nMonths;
        uint16_t totDays; //uint16_t year: declared as global, to be available to other activities
        uint32_t totSecs, totMins, totHours;

        //wait until obtaing semafore for accessing portC
        while (GetSemaphore(SEMAF_PORTC, PORTCUSER_RTC_CLKIO) == FALSE) {;}
        //burst read current Local time from the DS1302 time keeping chip;
        //store the readout into curLocalTime
        RdClockBurst(curLocalTime);
        ReleaseSemaphore(SEMAF_PORTC, PORTCUSER_RTC_CLKIO); //release portC semafore
        secs = Dec2Bin(*curLocalTime);
        mins = Dec2Bin(*(curLocalTime+1));
        hours = Dec2Bin(*(curLocalTime+2));
        date = Dec2Bin(*(curLocalTime+3));
        month = Dec2Bin(*(curLocalTime+4));
        nMonths = month - 1;   //the number of fully elapsed months
        year = Dec2Bin(*(curLocalTime+6))+2000; //assume year is from 2000
        totDays = 31*((nMonths%2) + (nMonths/2)) + 30*(nMonths/2) + ((nMonths%7) % 2) -
                 ((nMonths<2)?0:((IsLeapYear(year))?1:2)) + date - 1;
        totHours = totDays * 24 + hours;
        totMins = totHours * 60 + mins;
        totSecs = totMins * 60 + secs;
        return totSecs;
}

uint16_t YearDays(uint16_t xYear)
{//returns the number of days of the year according to the gregorian classification of leap years.
        return (IsLeapYear(xYear)) ? 366 : 365;
}
```

```
//returns true (1) if year is leap, or zero otherwise...
// ... according to gregorian callendar
uint8_t IsLeapYear(uint16_t xYear) {
return ((((xYear %4) == 0) && ((xYear % 100) != 0)) || ((xYear % 400) == 0)) ? 1:0;
}

//returns the number of days of the specified  month of the year.
uint8_t MonthDays(uint8_t xMonth, uint16_t xYear){
        uint8_t nMonthDays[12] = {31,28,31,30,31,30,31,31,30,31,30,31};
        return (((xMonth == 2) && IsLeapYear(xYear)) ? 29 : nMonthDays[xMonth]);
}

//bitwise tests the bitPos of the xWord word
uint8_t BIT(uint16_t xWord, uint8_t bitPos)
{
        return (xWord & _BV(bitPos)? 1:0);
}


//--------------------
int sign(int xValue) { //returns: -1 if xValue < 0; 0 if xValue =0; 1 if xValue > 0
        int xRet;
        if (xValue < 0) {
                xRet = -1;
        }
        else if (xValue > 0) {
                xRet = +1;
        }
        else {
                xRet = 0;
        }
        return xRet;
}


// hi level LCD messaging function.
void LCDwriteMsg(char *MsgPtr, char line, char col)
{
        char TempData;
        int i=0;
        LCDaddress(line,col);
        //TempData=(*(MsgPtr+i));
        TempData=(*MsgPtr);
        while (TempData != 0 && i++<40)
        {
                LCDwriteDATA(TempData);
                TempData=(*(MsgPtr+i));
        }
}

//blinks a dot on LCD specified position at 1Hz freq;
void BlinkDot(uint8_t *lToggle, uint8_t line, uint8_t col)
{
        uint8_t curSecs;
        char *dotOrBlank=".";
        //wait until obtaing semafore for accessing portC
        while (GetSemaphore(SEMAF_PORTC, PORTCUSER_RTC_CLKIO) == FALSE) {;}
        curSecs = RTC_GetSecs();
        ReleaseSemaphore(SEMAF_PORTC, PORTCUSER_RTC_CLKIO); //release portC semafore
        if (curSecs != lastSecs)
        {
                lastSecs = curSecs;
                if (*lToggle)
                {
                        dotOrBlank = " ";
                        *lToggle = 0;
                }
                else
                {
                        dotOrBlank = ".";
                        *lToggle = 1;
                }
                LCDwriteMsg(dotOrBlank,line,col);
        }
}

//calculates a power ot 10. This avoids mem hungry std library funcs.
```

```
uint32_t powOf10(uint8_t xXp)
{
        uint32_t pwOf10 = 1;
        uint8_t i;
        for (i=0; i<xXp; i++)
        {
                pwOf10 *= 10;
        }
        return pwOf10;
}

//this is a short and special purpose itoa like library's func.
//converts the int32 word to ascii padding left with any specified char;
void I2A(uint32_t *inVal, char *cResult, uint8_t nDigs, char lcPad)
{
        uint8_t i, cVal, xpn=0, gotLeftMost = FALSE;
        uint32_t qVal, dVal, nVal;
        nVal = *inVal;
        for (i=0; i<nDigs; i++)
        {
                xpn = nDigs-1-i;
                dVal = powOf10(xpn);
                qVal = nVal / dVal;
                gotLeftMost = (qVal > 0) ? TRUE : gotLeftMost;
                cVal = (((nVal==0) && (gotLeftMost==FALSE)) ? lcPad : (qVal + 48));
                *(cResult + i) = cVal;
                nVal = nVal - (qVal * dVal);
        }
        *(cResult + nDigs) = 0;  //asciiz string
}

uint8_t Dec2Bin(uint8_t dDig)
{
        uint8_t tDig;
        tDig = dDig >> 4;
        return ((tDig * 10) + (dDig & 0x0F));
}
//=============================================================================
//CheckFeedbackError: is a FSM transitional subfunction
//Compares the difference between the supplied process and setpoint values with xError;
// where xError is either the coarse (stay) error or the fine (seek) error.
//Returns an integer (-1, 0, 1) according to the conditions below.
int CheckFeedbackError(float processValue, float SetPointValue, float xError) {
        if (abs(processValue - SetPointValue) <=  abs(xError))
                {return 0;}
        else if (processValue > SetPointValue )
                {return 1;}
        else //if(processValue < SetPointValue)
                {return -1;}
}

//error at rith boundary
int CheckFeedbackErrorRgt(float processValue, float SetPointValue, float xError) {
        if ((processValue > SetPointValue) && ((processValue - SetPointValue) <= abs(xError)))
                {return 0;}
        else if (processValue > SetPointValue )
                {return 1;}
        else //if(processValue < SetPointValue)
                {return -1;}
}

//error at left boundary
int CheckFeedbackErrorLft(float processValue, float SetPointValue, float xError)
{
        if ((processValue < SetPointValue) && ((processValue - SetPointValue) >= -abs(xError)))
                {return 0;}
        else if (processValue > SetPointValue )
                {return 1;}
        else //if(processValue < SetPointValue)
                {return -1;}
}

void ResetPIDdata(float inMaxValue, int inMaxPWM, float inKp, float inKi, float inKd,
struct PIDctrlData *pPIDdata, uint8_t resetSource){
        pPIDdata->IntegralSum=0;      //the integral of the errors
        pPIDdata->LastError=0; //the previous error: to use for calculating I and D terms..
        if (resetSource == FLASH_DEFAULTS)    //reset with original program defaults;
```

```
        {
                pPIDdata->MaxValue = 2*pi*81.4; //rad/sec this yields a flow rate of 0.5l/sec
                pPIDdata->MaxPWM = 255; // for 8 bit phase correct PWM
                pPIDdata->Kp = 1;               //proportional gain
                pPIDdata->Ki = 0;               //integral gain
                pPIDdata->Kd = 0;               //derivative gain
        }
        else if (resetSource == EEPROM_DEFAULTS){//reset with learned or tuned eeprom saved
defaults;
                pPIDdata->MaxValue = inMaxValue;
                pPIDdata->MaxPWM = inMaxPWM; // for 8 bit phase correct PWM
                pPIDdata->Kp = inKp;            //proportional gain
                pPIDdata->Ki = inKi;            //integral gain
                pPIDdata->Kd = inKd;            //derivative gain
        }
}


void    ResetTrackerState(struct TrackerMachine *Tracker, uint8_t resetSource){
         // state variable. //see tracker.h for tracker related definitions
        Tracker->state = TRACK_STATE_IDLE;
        //state outputs:
        Tracker->Forward=0; //no fwd movement
        Tracker->Reverse=0; //no reverse movement
        Tracker->AxisSelect=0; //hour axis selected by default

        if (resetSource == FLASH_DEFAULTS)    //reset with original program defaults;
        {
                //Hour Axis variables
                Tracker->TrackersCTRLmode = TRACKER_CTRLMODE_FSM_PID_ANGULARPOSITION;
                Tracker->HourAngle_SeekError = 0.25; //.25 degree. should be <= ..
                                                     // .. required tracking resolution.
                Tracker->HourAngle_StayError = .5; //

                //Declination Axis variables
                Tracker->DeclAngle_SeekError=.25;  //should be <= required tracking resolution.
                Tracker->DeclAngle_StayError=.5;  //should be <= required tracking resolution.
        }

        else if (resetSource == EEPROM_DEFAULTS)//reset with learned or tuned eeprom saved defaults;
        {
                //Read control mode settings from EEPROM
                eeprom_read_block( &Tracker->TrackersCTRLmode, &EETrackersCTRLmode,
                        sizeof(EETrackersCTRLmode));
                //Read learned Hour Axis Seek/Stay Errors settings from EEPROM
                eeprom_read_block( &Tracker->HourAngle_SeekError, &EEHourAngle_SeekError,
                        sizeof(EEHourAngle_SeekError));
                eeprom_read_block( &Tracker->HourAngle_StayError, &EEHourAngle_StayError,
                        sizeof(EEHourAngle_StayError));
                //Read learned Decl Axis Seek/Stay errors settings from EEPROM
                eeprom_read_block( &Tracker->DeclAngle_SeekError, &EEDeclAngle_SeekError,
                        sizeof(EEDeclAngle_SeekError));
                eeprom_read_block( &Tracker->DeclAngle_StayError, &EEDeclAngle_StayError,
                        sizeof(EEDeclAngle_StayError));
        }
}



uint8_t FsOut(uint8_t TrkrMode) //this function is to insure selective state output when state =
Idle.
{
        if (TrkrMode == TRACK_MODE_IDLE)
                {return 1;}     //this insure no movement, blocking manual remote control commands.
        else if (TrkrMode == TRACK_MODE_MANUAL)
                {return 0;}     //insures state machine will not interfere on manual remote commands.
        else if (TrkrMode == TRACK_MODE_AUTOTRACK)
                {return 0;}     //in fact the return value could also be = 1 (dont care).
        else if (TrkrMode == TRACK_MODE_SAFETY)
                {return 0;}     //in fact the return value could also be = 1 (dont care).
        else
                {return 0;}     //in fact the return value could also be = 1 (dont care).
}


//Boolean Finite State Machine (FSM) Tracker control
//Multiple inputs and multiple output system
```

```
//Boolean inferencing of transitional and output functions
void FSMtrack(struct TrackerMachine *Tracker)
{
//          uint8_t tempByte;
            switch (Tracker->state)
            {
                  case TRACK_STATE_IDLE:              //tracking assembly not moving.
                        if ((TrackerMode == TRACK_MODE_IDLE)              ||
                              (TrackerMode == TRACK_MODE_MANUAL) ||
                        (((TrackerMode==TRACK_MODE_AUTOTRACK)||(TrackerMode==TRACK_MODE_SAFETY)) &&
                        (CheckFeedbackError(ActualHourAngle, StPtHourAngle,  Tracker->HourAngle_StayError)==0) &&
                          (CheckFeedbackError(ActualDeclAngle, StPtDeclAngle,  Tracker->DeclAngle_StayError)==0)))
                              {
                                    //Tracker->state = TRACK_STATE_IDLE;  //keep on idle
                                    Tracker->Forward = FsOut(TrackerMode);//state outputs
                                    Tracker->Reverse = FsOut(TrackerMode);
                                    Tracker->AxisSelect = FsOut(TrackerMode);
}
                        else if (((TrackerMode == TRACK_MODE_AUTOTRACK) || (TrackerMode == TRACK_MODE_SAFETY)) &&
                          /*(CheckFeedbackError(ActualHourAngle, StPtHourAngle,  Tracker->HourAngle_StayError)==0) && */
                          (CheckFeedbackError(ActualDeclAngle, StPtDeclAngle,  Tracker->DeclAngle_StayError)<0))
                              {
                                    Tracker->state = TRACK_STATE_MOVING_NORTHWARD;
                                    Tracker->Forward =1;
                                    Tracker->Reverse =0;
                                    Tracker->AxisSelect =1;
                              }
                        else if (((TrackerMode == TRACK_MODE_AUTOTRACK) || (TrackerMode == TRACK_MODE_SAFETY)) &&
                          (CheckFeedbackError(ActualHourAngle, StPtHourAngle,  Tracker->HourAngle_StayError)>0))
                              {
                                    Tracker->state = TRACK_STATE_MOVING_EASTWARD;
                                    Tracker->Forward =0;
                                    Tracker->Reverse =1;
                                    Tracker->AxisSelect =0;
                              }

                        else if (((TrackerMode == TRACK_MODE_AUTOTRACK) || (TrackerMode == TRACK_MODE_SAFETY)) &&
                          (CheckFeedbackError(ActualHourAngle, StPtHourAngle,  Tracker->HourAngle_StayError)<0))
                              {
                                    Tracker->state = TRACK_STATE_MOVING_WESTWARD;
                                    Tracker->Forward =1;
                                    Tracker->Reverse =0;
                                    Tracker->AxisSelect =0;
                              }
                        else if (((TrackerMode == TRACK_MODE_AUTOTRACK) || (TrackerMode == TRACK_MODE_SAFETY)) &&
                          (CheckFeedbackError(ActualHourAngle, StPtHourAngle,  Tracker->HourAngle_StayError)==0) &&
                          (CheckFeedbackError(ActualDeclAngle, StPtDeclAngle,  Tracker->DeclAngle_StayError)>0))
                              {
                                    Tracker->state = TRACK_STATE_MOVING_SOUTHWARD;
                                    Tracker->Forward =0;
                                    Tracker->Reverse =1;
                                    Tracker->AxisSelect =1;
                              }

                              break;

                  case TRACK_STATE_MOVING_EASTWARD: //= moving backward; actual hour angle decreasing
                  // ... earthwise rotation movement).
                        if ((TrackerMode == TRACK_MODE_IDLE)              || (TrackerMode == TRACK_MODE_MANUAL) ||
                          (((TrackerMode == TRACK_MODE_AUTOTRACK) || (TrackerMode == TRACK_MODE_SAFETY)) &&
                          ((CheckFeedbackError(ActualHourAngle, StPtHourAngle,  Tracker->HourAngle_SeekError)<=0)))) // ||
                          //(CheckFeedbackError(ActualHourAngle, StPtHourAngle,  Tracker->HourAngle_SeekError)<0))))
                              {
                                    Tracker->state = TRACK_STATE_IDLE;
                                    Tracker->Forward =0;
                                    Tracker->Reverse =0;
                                    Tracker->AxisSelect =0;
                              }
                        else if (((TrackerMode == TRACK_MODE_AUTOTRACK) || (TrackerMode == TRACK_MODE_SAFETY)) &&
                          (CheckFeedbackError(ActualHourAngle, StPtHourAngle,  Tracker->HourAngle_SeekError)>0))
                              {
                                    //Tracker->state = TRACK_STATE_MOVING_EASTWARD; //keep moving easthwards
                                    Tracker->Forward =0;
                                    Tracker->Reverse =1;
                                    Tracker->AxisSelect =0;
                              }

                              break;

                  case TRACK_STATE_MOVING_WESTWARD: //= moving forward; actual hour angle increasing; ...
                  // ... anti-earthwise rotation movement.
                        if ((TrackerMode == TRACK_MODE_IDLE)              || (TrackerMode == TRACK_MODE_MANUAL) ||
                          (((TrackerMode == TRACK_MODE_AUTOTRACK) || (TrackerMode == TRACK_MODE_SAFETY)) &&
                          ((CheckFeedbackErrorRgt(ActualHourAngle, StPtHourAngle,  Tracker->HourAngle_SeekError)>=0) )))
                              {
                                    Tracker->state = TRACK_STATE_IDLE;
                                    Tracker->Forward =0;
```

```
                                        Tracker->Reverse =0;
                                        Tracker->AxisSelect =0;
                                }
                        else if (((TrackerMode == TRACK_MODE_AUTOTRACK) || (TrackerMode == TRACK_MODE_SAFETY)) &&
                          (CheckFeedbackError(ActualHourAngle, StPtHourAngle,  Tracker->HourAngle_SeekError)<0))
                                {
                                        //Tracker->state = TRACK_STATE_MOVING_WESTWARD; //keep moving westwards
                                        Tracker->Forward =1;
                                        Tracker->Reverse =0;
                                        Tracker->AxisSelect =0;
                                }

                                break;

                case TRACK_STATE_MOVING_SOUTHWARD: //= moving backward; actual declination angle decreasing
                                if ((TrackerMode == TRACK_MODE_IDLE)                || (TrackerMode == TRACK_MODE_MANUAL) ||
                                  (((TrackerMode == TRACK_MODE_AUTOTRACK) || (TrackerMode == TRACK_MODE_SAFETY)) &&
                                  ((CheckFeedbackError(ActualDeclAngle, StPtDeclAngle,  Tracker->DeclAngle_SeekError)<=0) )))
                                {
                                        Tracker->state = TRACK_STATE_IDLE;
                                        Tracker->Forward =0;
                                        Tracker->Reverse =0;
                                        Tracker->AxisSelect =0;
                                }
                                else if (((TrackerMode == TRACK_MODE_AUTOTRACK) || (TrackerMode == TRACK_MODE_SAFETY)) &&
                                  (CheckFeedbackError(ActualHourAngle, StPtHourAngle,  Tracker->HourAngle_StayError)==0) &&
                                  (CheckFeedbackError(ActualDeclAngle, StPtDeclAngle,  Tracker->DeclAngle_SeekError)>0))
                                {
                                        //Tracker->state = TRACK_STATE_MOVING_SOUTHWARD; //keep moving southwards
                                        Tracker->Forward =0;
                                        Tracker->Reverse =1;
                                        Tracker->AxisSelect =1;
                                }

                                break;

                case TRACK_STATE_MOVING_NORTHWARD: //=moving forward; actual declination angle increasing
                                if ((TrackerMode == TRACK_MODE_IDLE)                || (TrackerMode == TRACK_MODE_MANUAL) ||
                                  (((TrackerMode == TRACK_MODE_AUTOTRACK) || (TrackerMode == TRACK_MODE_SAFETY)) &&
                                  ((CheckFeedbackError(ActualDeclAngle, StPtDeclAngle,  Tracker->DeclAngle_SeekError)>=0) )))
                                {
                                        Tracker->state = TRACK_STATE_IDLE;
                                        Tracker->Forward =0;
                                        Tracker->Reverse =0;
                                        Tracker->AxisSelect =0;
                                }
                                else if (((TrackerMode == TRACK_MODE_AUTOTRACK) || (TrackerMode == TRACK_MODE_SAFETY)) &&
                                  /*(CheckFeedbackError(ActualHourAngle, StPtHourAngle,  Tracker->HourAngle_StayError)==0) &&*/
                                  (CheckFeedbackError(ActualDeclAngle, StPtDeclAngle,  Tracker->DeclAngle_SeekError)<0))
                                {
                                        //Tracker->state = TRACK_STATE_MOVING_NORTHWARD; //keep moving northwards
                                        Tracker->Forward =1;
                                        Tracker->Reverse =0;
                                        Tracker->AxisSelect =1;
                                }

                                break;
        }


        //apply state outputs to the tracker module

        //place a semaphore to prevent access to the port C while writing motor control info to the tracker module;
        while (GetSemaphore(SEMAF_PORTC, PORTCUSER_TRACKADDRESS) == FALSE)
        {;}             //wait until obtaing semafore for accessing portC
        while (GetSemaphore(SEMAF_PORTA, PORTAUSER_TRACK_CLK) == FALSE)
        {;}             //wait until get semafore for accessing portA
        tempByte = PORTC;
        tempByte &= 0xC0; //clear 6 lsbits.
        tempByte |= (Tracker->Forward << 3) | (Tracker->Reverse << 2) | (Tracker->AxisSelect << 1);
        DDRC = 0xFF; //all bits outputs
        PORTC = tempByte;
        asm("nop");
        asm("nop"); //wait at least 100nS for 374 setup time
        PORTA |= _BV(PORTA1); //= _BV(MOT_CtrlLatchClockBit);
        asm("nop"); //wait at least 50nS for 374 clock width time
        PORTA &= ~_BV(PORTA1);  //= ~_BV(MOT_CtrlLatchClockBit);
        asm("nop");//wait at least 100nS for 374 pd and transition times
        asm("nop");
        asm("nop");
        asm("nop");
        OCR1B = 255; //curTrackersPWM_DC;     //ToDo: Assign this tru setting curTrackersPWM_DC via eeprom.
        curTrackersPWM_DC = 255;

        ReleaseSemaphore(SEMAF_PORTA, PORTAUSER_TRACK_CLK);   //release portA semafore
        ReleaseSemaphore(SEMAF_PORTC, PORTCUSER_TRACKADDRESS); //release portC semafore
}
```

```
float PIDctrl(float StPtValue, float ActualValue, uint32_t sampTime, struct PIDctrlData *pidData)
{
	float PIDoutput, proportionalTerm, integralTerm, derivativeTerm, delta_t, Error;

	delta_t = (sampTime - pidData->LastSampTime)/100; //100 is the timer0Tic int freq.in hertz.
	pidData->LastSampTime = sampTime;
	Error = StPtValue - ActualValue;

	proportionalTerm = pidData->Kp * Error;
	if (abs(proportionalTerm) > pidData->MaxValue)
	{
		proportionalTerm = pidData->MaxValue * sign(Error); //avoid overflow
	}

	integralTerm = pidData->Ki *(pidData->IntegralSum + Error) * delta_t;
	if (abs(integralTerm) > pidData->MaxValue) { //avoid overflow / integral windup
		integralTerm = sign(Error) * pidData->MaxValue;
	}
	pidData->IntegralSum = integralTerm;

	derivativeTerm = pidData->Kd * (Error - pidData->LastError) / delta_t;
	if (abs(derivativeTerm) > pidData->MaxValue)
	{
		derivativeTerm = pidData->MaxValue * sign(Error); //avoid overflow
	}

	PIDoutput =  proportionalTerm + integralTerm + derivativeTerm;
	if (abs(PIDoutput) > pidData->MaxValue)
	{
		PIDoutput = pidData->MaxValue * sign(PIDoutput); //avoid overflow
	}

	pidData->LastError = Error;

	return PIDoutput;
}
```

```
/* ========================================================================
/* The main function.
/* Code execution starts here:
/*     1st is the initialization and
/*     2nd is the forever loop.
/* ------------------------------------------------------------------------*/
int main(void){
	/
	char *sUKZNwelcome ="UKZN Solar Ctrlr";
	char *sGoodDay ="Good Day! :-)";
	char sDegrees[4]={0xDF,'C','-'}, *sSysOKStat="SystemOK!", *sCheckSys="ChkSystm!";
	char *sTimerTic="0123456789012345";
	char *sLt; // pointer to LocalTime
	char *sCurSolarTime = "St:000000;000000 ";
	char *sAmbientTemp="+101.25oC;      ";
	float xHA, xDA;
	char *sHourAngle="HA:+50.7;t:+55.9  ";
	char *sDeclAngle="DA:+50.7;t:+55.9  ";
	char *sRcvTemp="RcvI150.7;O240.9  ";
	char *sTEStemp = "TesI150.7;O240.9  ";
	char *sStPtChg = "SetPtRPS:+1240.9  ";
	char *sChgPump = "ChgPpRPS:+1240.9  ";
	char *sStPtDisChg = "SetPtRPS:+1240.9  ";
	char *sDisChgPump = "DChgPpRPS+1240.9  ";
	uint8_t chgPpStarted = FALSE;
	uint8_t i, blkCol = 9;

	uint8_t LogChecklist, LogDestination; //Logging profile: checklist and destination
	float MaxHourAngle, MaxHourPWM, HourAngleKp, HourAngleKi, HourAngleKd;
	float MaxDeclAngle, MaxDeclPWM, DeclAngleKp, DeclAngleKi, DeclAngleKd;

	uint8_t ChgPumpCTRLmode;
```

```
float ChgPpMaxSpeed, ChgPpMaxPWM, ChgPpSpeedKp, ChgPpSpeedKi, ChgPpSpeedKd;

//uint8_t DisChgPumpCTRLmode;
//float DisChgPpMaxSpeed, DisChgPpSpeedKp, DisChgPpSpeedKi, DisChgPpSpeedKd;
//not implemented

float PIDout;
struct TrackerMachine TrackersStateData;

ResetTrackerState(&TrackersStateData, EEPROM_DEFAULTS); //normal mode
TrackerMode = TRACK_MODE_AUTOTRACK;

struct PIDctrlData HourAnglePIDdata;
eeprom_read_block((void *)&MaxHourAngle, (const void *)&EEMaxHourAngle, 4);
eeprom_read_block( &MaxHourPWM, &EEMaxHourPWM, sizeof(EEMaxHourPWM));
eeprom_read_block( &HourAngleKp, &EEHourAngleKp, sizeof(EEHourAngleKp));
eeprom_read_block( &HourAngleKi, &EEHourAngleKi, sizeof(EEHourAngleKi));
eeprom_read_block( &HourAngleKd, &EEHourAngleKd, sizeof(EEHourAngleKd));
ResetPIDdata(MaxHourAngle, MaxHourPWM, HourAngleKp, HourAngleKi,
       HourAngleKd, &HourAnglePIDdata, FLASH_DEFAULTS);

struct PIDctrlData DeclAnglePIDdata;
eeprom_read_block( &MaxDeclAngle, &EEMaxDeclAngle, sizeof(EEMaxDeclAngle));
eeprom_read_block( &MaxDeclPWM, &EEMaxDeclPWM, sizeof(EEMaxDeclPWM));
eeprom_read_block( &DeclAngleKp, &EEDeclAngleKp, sizeof(EEDeclAngleKp));
eeprom_read_block( &DeclAngleKi, &EEDeclAngleKi, sizeof(EEDeclAngleKi));
eeprom_read_block( &DeclAngleKd, &EEDeclAngleKd, sizeof(EEDeclAngleKd));
ResetPIDdata(MaxDeclAngle, MaxDeclPWM, DeclAngleKp, DeclAngleKi,
       DeclAngleKd, &DeclAnglePIDdata, FLASH_DEFAULTS);


struct PIDctrlData ChgPpPIDdata;
struct ChgPumpStateData ChgPpState;
eeprom_read_block( &ChgPumpCTRLmode, &EEChgPumpCTRLmode, sizeof(EEChgPumpCTRLmode));
eeprom_read_block( &ChgPpMaxSpeed, &EEChgPpMaxSpeed, sizeof(EEChgPpMaxSpeed));
eeprom_read_block( &ChgPpMaxPWM, &EEChgPpMaxPWM, sizeof(EEChgPpMaxPWM));
eeprom_read_block( &ChgPpSpeedKp, &EEChgPpSpeedKp, sizeof(EEChgPpSpeedKp));
eeprom_read_block( &ChgPpSpeedKi, &EEChgPpSpeedKi, sizeof(EEChgPpSpeedKi));
eeprom_read_block( &ChgPpSpeedKd, &EEChgPpSpeedKd, sizeof(EEChgPpSpeedKd));
ResetPIDdata(ChgPpMaxSpeed, ChgPpMaxPWM, ChgPpSpeedKp,
       ChgPpSpeedKi, ChgPpSpeedKd, &ChgPpPIDdata, FLASH_DEFAULTS);

/*struct PIDctrlData DisChgPpPIDdata;
eeprom_read_block(&DisChgPumpCTRLmode, &EEDisChgPumpCTRLmode, sizeof(EEDisChgPumpCTRLmode));
eeprom_read_block( &DisChgPpMaxSpeed, &EEDisChgPpMaxSpeed, sizeof(EEDisChgPpMaxSpeed));
eeprom_read_block( &DisChgPpMaxPWM, &EEDisChgPpMaxPWM, sizeof(EEDisChgPpMaxPWM));
eeprom_read_block( &DisChgPpSpeedKp, &EEDisChgPpSpeedKp, sizeof(EEDisChgPpSpeedKp));
eeprom_read_block( &DisChgPpSpeedKi, &EEDisChgPpSpeedKi, sizeof(EEDisChgPpSpeedKi));
eeprom_read_block( &DisChgPpSpeedKd, &EEDisChgPpSpeedKd, sizeof(EEDisChgPpSpeedKd));
ResetPIDdata(DisChgPpMaxSpeed, DisChgPpMaxPWM, DisChgPpSpeedKp,
DisChgPpSpeedKi, DisChgPpSpeedKd, &DisChgPpPIDdata, FLASH_DEFAULTS);
*/

LogChecklist = eeprom_read_byte((const uint8_t *)&EELogChecklist);
LogDestination = eeprom_read_byte((const uint8_t*)&EELogDestination);
notInitLogged = InitDataLog(LogChecklist, LogDestination);

IO_Initialization();
LCDreset();
LCDwriteMsg(sUKZNwelcome, 0, 0);
LCDwriteMsg(sGoodDay, 1, 0);
dly_1s();

//default display action is 'ONE': show RTC (local)time, ambiente temp. and system status.
if (inScan == 0) {menu_action = ASCII_one;}


//the main normal control loop
while(TRUE) {  //loop for ever

       //check system activity event flags and fire the respective tasks  accordingly:

       //update trackers input'variables
       CalcSolarTime(sCurSolarTime);
       strcpy(sSolarTime, sCurSolarTime);
       //reset timer0Tic to zero at each  day begining (at 00:00:00 solar time);
       //sincronize the timertic with RTC at midnight solar time.
```

```
if (IsSolarMidNightFlag && notSynchronized)
{
        //wait for currently triggered conversions to terminate.
        while ((ADCtrig_cnt+TDCtrig_cnt) != 0) {;}
        notSynchronized = FALSE; //to insure sync only once
        timer0Tic = CurTime2Secs() * 100;  //synchronize with RTC
        synchroCnt++;  //to verify sync only once functionality.
}

StPtHourAngle = StHourAngle;          //it is already in degrees of arc
StPtDeclAngle = StDeclAngleDeg;       //in degrees.

SunriseAngle = - SunsetHourAngle()*180/pi; //calculate sunrise angle

//retrieve Hour angle from ADC buffer, and convert from voltage to degrees.
xHA = ADC_LastSample[0][ADC_xmuxAddr_HourAngle];
ActualHourAngleSampTime = ADC_LastSample[1][ADC_xmuxAddr_HourAngle];
//ActualHourAngle = (((360.0 * ADC_Vref * xHA) / (ADCmax * (AVC_Vmax - AVC_Vmin))) -
//((360.0 * AVC_Vmin) /(AVC_Vmax - AVC_Vmin))); //For Spectrol AVC.
ActualHourAngle = (290* xHA/1024) - 145 ; // for PB050 pot; ADCref=5V; VmaxPot=5V;
xDA = ADC_LastSample[0][ADC_xmuxAddr_DeclinationAngle];
ActualDeclAngleSampTime = ADC_LastSample[1][ADC_xmuxAddr_DeclinationAngle];
//ActualDeclAngle= (((360.0 * ADC_Vref * xDA) / (ADCmax * (AVC_Vmax - AVC_Vmin))) -
//((360.0 * AVC_Vmin) /(AVC_Vmax - AVC_Vmin)));
ActualDeclAngle = (290* xDA/1024) - 145 ;//for PB050 pot; ADCref=5V; VmaxPot=5V;

//manual mode & parking, are the highest priority, so they override everything else.
if ((TrackerMode != TRACK_MODE_MANUAL) && (ParkFlag == FALSE)) {
        if (CheckSafetyTemperatureFlag == TRUE) {
                if (GetMaxtemperature() >= MaxTemp) {
                //used for either defocusing or sending dish to a safe/rest position
                   collectorDefocusFlag = TRUE;
                }
        }

        if (collectorDefocusFlag)
        {//move collector to defocussed position;
                TrackerMode = TRACK_MODE_SAFETY;
                StPtHourAngle = SafetyHourAngle;
                StPtDeclAngle = SafetyDeclAngle;
                trackingIntervalFlag = TRUE;
        }
        else if ((StHourAngle > -SunriseAngle) || (StHourAngle < SunriseAngle))
        //if it is night: //move collector to either the rest position ...
        { ... or the start (focused to extreme east) position;
                if ((StHourAngle > -SunriseAngle)||(StHourAngle-SunriseAngle < -10))
                { //if it is night and...
                  // (before solar midnight; or after midnight but far from sunrise)
                        TrackerMode = TRACK_MODE_AUTOTRACK;
                        StPtHourAngle = RestHourAngle;//send dish to rest position
                        StPtDeclAngle = RestDeclAngle;
                        trackingIntervalFlag = TRUE;
                }
                else if ((StHourAngle<SunriseAngle)&&(StHourAngle-SunriseAngle >= -10))
                // if it is night and near sunrize
                {//move collector to sunrise  position;
                        TrackerMode = TRACK_MODE_AUTOTRACK;
                         //send dish to sunrise position.
                        StPtHourAngle = SunriseAngle;
                        StPtDeclAngle = StDeclAngleDeg;
                        trackingIntervalFlag = TRUE;
                }
        }

}
//Manual mode overrides parking.
else if ((TrackerMode != TRACK_MODE_MANUAL) && (ParkFlag == TRUE))
{//move collector to parking position;
        StPtHourAngle = RestHourAngle;
        StPtDeclAngle = RestDeclAngle;
        trackingIntervalFlag = FALSE;
        TrackerMode = TRACK_MODE_AUTOTRACK;
}
//Manual mode is a "do nothing"
else if (TrackerMode == TRACK_MODE_MANUAL)
{//set the trackingIntervalFlag, so as to send the FSM to IDLE state;
        trackingIntervalFlag = TRUE;
```

```
}

//Now, after updating variables and firing the execution of critical tasks, datalog.
if ((DataLogFlag==TRUE) && notLogged)
{//Datalog the current set of checklisted variables.
        //wait until obtaing semafore for accessing portC
        while (GetSemaphore(SEMAF_PORTC, PORTCUSER_RTC_CLKIO) == FALSE) {;}
        sLogLocalTime = RTC_Disp(sLocalTime, 0, noDisp); //get datalog time from RTC
        ReleaseSemaphore(SEMAF_PORTC, PORTCUSER_RTC_CLKIO); //release semaphore
        //Log data using specified checklist
        notLogged = DataLog(LogChecklist, LogDestination);
        DataLogFlag = FALSE;
}

//and now, perform tracking tasks if flag is set.
if (trackingIntervalFlag == TRUE)
{
        trackingIntervalFlag = FALSE;
        collectorDefocusFlag = FALSE;
        if(TrackersStateData.TrackersCTRLmode ==
                TRACKER_CTRLMODE_FSM_ONOFF_ANGULARPOSITION) {
                FSMtrack(&TrackersStateData);
                //StPtHourAngle,ActualHourAngle, StPtDeclAngle, ActualDeclAngle; ...
        }       //are passed as well, through their global scope/visibility.
        else if (TrackersStateData.TrackersCTRLmode ==
                TRACKER_CTRLMODE_FSM_PID_ANGULARPOSITION) {
                PIDout=PIDctrl(StPtHourAngle,ActualHourAngle,ActualHourAngleSampTime,
                  &HourAnglePIDdata);
                if (PIDout == 0){
                        TrackersStateData.Forward = 0;
                        TrackersStateData.Reverse = 0;
                }
                else if (PIDout < 0){
                        TrackersStateData.Forward = 0;
                        TrackersStateData.Reverse = 1;
                }
                else if (PIDout < 0){
                        TrackersStateData.Forward = 1;
                        TrackersStateData.Reverse = 0;
                }
                PIDout = PIDout / (ActualHourAngleSampTime -
                        HourAnglePIDdata.LastSampTime); //obtain speed

                PIDout = (HourAnglePIDdata.MaxPWM * abs(PIDout) /
                         HourAnglePIDdata.MaxValue) ; //to PWM
                OCR1B = PIDout;
        }
        else if (TrackersStateData.TrackersCTRLmode ==
                TRACKER_CTRLMODE_FFSM_PID_ANGULARPOSITION) {
                //not implemented
        }//endif
        TrackersState = TrackersStateData.state;
}
// now execute Charging pump speed control.
if (ChgPumpIntervalFlag == TRUE){

        if (ChgPumpCTRLmode == CHGPUMP_CTRLMODE_PID_ANGULARSPEED) {
              PIDout = PIDctrl(StPtChgPumpSpeed, ActualChgPumpSpeed,
                        LastChgPpSampTime, &ChgPpPIDdata);
                if (PIDout == 0){
                        ChgPpState.Forward = 0;
                        ChgPpState.Reverse = 0;
                }
                else if (PIDout < 0){
                        ChgPpState.Forward = 0;
                        ChgPpState.Reverse = 1;
                }
                else if (PIDout > 0){
                        ChgPpState.Forward = 1;
                        ChgPpState.Reverse = 0;
                }
                PIDout = ChgPpPIDdata.MaxPWM * abs(PIDout) / ChgPpPIDdata.MaxValue;
                curChgPumpPWM_DC =
                        (PIDout > = ChgPumpMinPWM_DC)? PIDout : ChgPumpMinPWM_DC;
                OCR2 = curChgPumpPWM_DC;
        }
        else if (ChgPumpCTRLmode == CHGPUMP_CTRLMODE_FUZZYPID_ANGULARSPEED)
```

```
        {
                //not implemented
        }
}

// and now execute DisCharging pump speed control.
if (ChgPumpIntervalFlag == TRUE){
        //Discharge pump control not implemented

}//endif


//Finaly: check for user's keyboard triggered (or default) menu actions ..
// ..  and execute them accordingly.
switch (menu_action)
{
case ASCII_zero: //zero key pressed: Reset LCD once; show Local time from RTC chip;
        if (lClearLCD) {LCDreset();}
        lClearLCD = 0;
        //wait until obtaing semafore for accessing portC
        while (GetSemaphore(SEMAF_PORTC, PORTCUSER_RTC_CLKIO) == FALSE) {;}
        RTC_Disp(sLocalTime, 0,toDisp);
        ReleaseSemaphore(SEMAF_PORTC, PORTCUSER_RTC_CLKIO); //release portC semafore
        LCDwriteMsg("TmTic:",1,0);
        tempTic = timer0Tic;
        I2A(&tempTic, sTimerTic, 10, cPadZero); //32bits => 10 decimal dgits
        LCDwriteMsg(sTimerTic,1,6);
        //wait until obtaing semafore for accessing portC
        while (GetSemaphore(SEMAF_PORTC, PORTCUSER_RTC_CLKIO) == FALSE) {;}
        if (RTC_GetSecs() & 0x80){;           //if clock halted
                RTC_WriteByteAtAddress(RTC_SECS_WRITE,0);    //start clock
        } //endif
        ReleaseSemaphore(SEMAF_PORTC, PORTCUSER_RTC_CLKIO); //release portC semafore
        blkCol = 5;
        //menu_action = ASCII_one;
        break;
case ASCII_one://1 pressed: show day,date,time, onboard temperat.and system status.
        //obtain semaphore for using portC (LB2)
        while (GetSemaphore(SEMAF_PORTC, PORTCUSER_RTC_CLKIO) == FALSE) {;}
        RTC_Disp(sLocalTime, 0, toDisp);
        ReleaseSemaphore(SEMAF_PORTC, PORTCUSER_RTC_CLKIO); //release portC semafore
        AmbientTemp_inCelsius = ADC_LastSample[0][ADC_xmuxAddr_MainBoardTemp];
        AmbientTemp_inCelsius = ((AmbientTemp_inCelsius  * 500) / 1024)-97;
        //calibrated from the onboard LM35DT temp sensor; at Vref=Vdd=4.9V;
        dtostrf(AmbientTemp_inCelsius, 4, 1, sAmbientTemp);
        strcpy(sAmbientTemp+4, sDegrees);
        strcpy(sAmbientTemp+8, (SysStatUserIDs==0)?sSysOKStat:sCheckSys);
        LCDwriteMsg(sAmbientTemp,1,0); //

        blkCol = 6;
        break;
case ASCII_two://2 pressed: show current local and solar times
        //wait until obtaing semafore for accessing portC
        while (GetSemaphore(SEMAF_PORTC, PORTCUSER_RTC_CLKIO) == FALSE) {;}
        sLt = RTC_Disp(sLocalTime, 0, noDisp);
        ReleaseSemaphore(SEMAF_PORTC, PORTCUSER_RTC_CLKIO); //release portC semafore
        LCDwriteMsg("Lt:",0,0);
        sLt += 10;
        LCDwriteMsg(sLt,0,3);
        dtostrf(LocalTime_inMinutes,7,1,sLocalTime);
        LCDwriteMsg(sLocalTime,0,9);
        LCDwriteMsg(";",0,9);

        LCDwriteMsg("St:",1,0);
        LCDwriteMsg(sCurSolarTime,1,3);
        dtostrf(SolarTime_inMinutes,7,1,sCurSolarTime);
        LCDwriteMsg(sCurSolarTime,1,9);
        LCDwriteMsg(";",0,9);
        blkCol = 9;
        break;
case ASCII_three: //3 pressed: show current setpoint / actual hour and decl.angles
        strcpy(sHourAngle, "HA");
        dtostrf(StHourAngle, 6,1,  sHourAngle+2);
        strcpy(sHourAngle+8, ";t");
        xHA = ActualHourAngle;
        dtostrf(xHA, 6,1,  sHourAngle+10);
        LCDwriteMsg(sHourAngle,0,0);
```

```
        strcpy(sDeclAngle, "DA");
        dtostrf(StDeclAngleDeg, 6,1,  sDeclAngle+2);
        strcpy(sDeclAngle+8, ";t");
        xDA = ActualDeclAngle;
        dtostrf(xDA, 6,1,  sDeclAngle+10);
        LCDwriteMsg(sDeclAngle,1,0);
        blkCol = 2;
        break;
case ASCII_four:       //4 pressed: up arrow
        menu_action = NextAction(lFORTH);
        break;
case ASCII_five://5 pressed:  show TES / Receiver's inlet and outlet temperatures
        xTemp = TDC_LastSample[0][TDC_xmuxAddr_TRcvrIn];
        dtostrf(xTemp, 5,1,  sRcvTemp+4);
        strcpy(sRcvTemp+9, ";O");
        xTemp = TDC_LastSample[0][TDC_xmuxAddr_TRcvrOut];
        dtostrf(xTemp, 5,1,  sRcvTemp+11);
        LCDwriteMsg(sRcvTemp,0,0);
        xTemp = TDC_LastSample[0][TDC_xmuxAddr_TESin];
        dtostrf(xTemp, 5,1,  sTEStemp+4);
        strcpy(sTEStemp+9, ";O");
        xTemp = TDC_LastSample[0][TDC_xmuxAddr_TESout];
        dtostrf(xTemp, 5,1,  sTEStemp+11);
        LCDwriteMsg(sTEStemp,1,0);
        blkCol = 9;
        break;
case ASCII_six: //6 pressed: down arrow
        menu_action = NextAction(lBACK);
        break;;
case ASCII_seven: //7 pressed: //charging pump manual control for visual debuging

        if (!chgPpStarted)
        {
                OCR2 = curChgPumpPWM_DC;
                chgPpStarted = TRUE;
        }
        else if (ChgPpSpeedIncrement != 0)
        {
                curChgPumpPWM_DC +=    ChgPpSpeedIncrement;
                if ((ChgPpSpeedIncrement >0)&&(curChgPumpPWM_DC<ChgPumpMinPWM_DC)){
                        curChgPumpPWM_DC = ChgPumpMinPWM_DC;
                }
                //ChgPpState.PWM = curChgPumpPWM_DC
                OCR2 = curChgPumpPWM_DC;
        }
        if ((curChgPumpPWM_DC == 0) || ( curChgPumpPWM_DC < ChgPumpMinPWM_DC)) {
                curChgPumpPWM_DC = 0;
                ChgPpState.state = PUMP_STATE_IDLE;
                ChgPpState.Forward = 0;
                ChgPpState.Reverse = 0;
        }
        //display charging pump actual and set poit speeds.
        dtostrf(ActualChgPumpSpeed, 5,1,  sChgPump+8);
        dtostrf(curChgPumpPWM_DC, 3,0,  sChgPump+13);
        LCDwriteMsg(sChgPump,0,0);
        dtostrf(StPtChgPumpSpeed, 7,1,  sStPtChg+9);
        LCDwriteMsg(sStPtChg,1,0);
        blkCol = 7;
        ChgPpSpeedIncrement = 0;
        break;
case ASCII_eight://8 pressed: //display Discharging pump actual and set poit speeds.
        dtostrf(ActualDisChgPumpSpeed, 7,1,  sDisChgPump+9);
        LCDwriteMsg(sDisChgPump,0,0);
        dtostrf(StPtDisChgPumpSpeed, 7,1,  sStPtDisChg+9);
        LCDwriteMsg(sStPtDisChg,1,0);
        blkCol = 9;
        break;
case ASCII_nine:
        /*LCDwriteMsg("Heat utilization",0,0);*/ //not implemented

         //set temporariliy for tracker control Monitoring visual debug: begin
        dtostrf(TrackerMode, 2,0,  sHourAngle); //show current trackercontrol mode
        //show current tracker's FSM state
        dtostrf(TrackersStateData.state, 2,0,  sHourAngle+2);
        //show current tracker set point hour angle
        dtostrf(StPtHourAngle, 6,1,  sHourAngle+4);
        xHA = ActualHourAngle;
```

```
        dtostrf(xHA, 6,1,  sHourAngle+10);
        LCDwriteMsg(sHourAngle,0,0);
        //show current tracker Motor select output
        dtostrf(TrackersStateData.AxisSelect, 1,0,  sDeclAngle);
         //show current tracker state Reverse output
        dtostrf(TrackersStateData.Reverse, 2,0,  sDeclAngle+1);
        //show current tracker state Forward output
        dtostrf(TrackersStateData.Forward, 1,0,  sDeclAngle+3);
        //show current tracker set point declination angle
        dtostrf(StPtDeclAngle, 6,1,  sDeclAngle+4);
        xDA = ActualDeclAngle;
        dtostrf(xDA, 6,1,  sDeclAngle+10);
        LCDwriteMsg(sDeclAngle,1,0);
        blkCol = 3;
        //tracker control Monitoring debug: end

        blkCol = 9;
        break;
case 'C'://*01# extended command pressed:  real time clock adjust (menu option 'C')
        menu_action = 'C';
        //clock registers order: secs, mins, hours, date, month, day, year, ctrlreg;
        if (KeybBuffInput_IsON == FALSE) {
                if (inKey == CR) {
                        if (KeybBuffIndex >=13){
                                for (i=0; i<8; i++)
                                { //each pair of two ascii bytes = 1 byte of BCD data
                                        *(sWrRTCtime+i) = //convert to hexadecimal
                                           Ascii2BCD((const char)KeybBuffer[2*i],
                                           (const char)KeybBuffer[2*i+1]);
                                }
                                //now: audit the user input:
                                if (Dec2Bin(*(sWrRTCtime)) > 59)
                                        {*(sWrRTCtime) = 00;} //default to 00 secs
                                if (Dec2Bin(*(sWrRTCtime+1)) > 59)
                                        {*(sWrRTCtime+1) = 00;}        //00 mins
                                if (Dec2Bin(*(sWrRTCtime+2)) > 23)
                                        {*(sWrRTCtime+2) = (1<<4) + 2;} //12h
                                if (Dec2Bin(*(sWrRTCtime+4)) > 12)
                                        {*(sWrRTCtime+4) = 03;}        //March
                                if (Dec2Bin(*(sWrRTCtime+5)) > 59)
                                        {*(sWrRTCtime+5) = 02;}        //the 2nd
                                if (Dec2Bin(*(sWrRTCtime+6)) > 59)
                                        {*(sWrRTCtime+6) = 9;} //2009
                                if (Dec2Bin(*(sWrRTCtime+3)) >
                                    MonthDays(Dec2Bin(*(sWrRTCtime+4)),
                                    Dec2Bin(*(sWrRTCtime+6))+2000))
                                        {*(sWrRTCtime+3) = 1<4+7;} //17
                                if (Dec2Bin(*(sWrRTCtime+7)) !=80)
                                        {*(sWrRTCtime+7) = 00;}//null ctrl  reg
                                while (GetSemaphore(SEMAF_PORTC,PORTCUSER_RTC_CLKIO)
                                        ==FALSE) {;}
                                RTC_Adjust((uint8_t *)sWrRTCtime);  // RTC update
                                ReleaseSemaphore(SEMAF_PORTC, PORTCUSER_RTC_CLKIO);
                                //synchronize timer0tic with RTC
                                timer0Tic = CurTime2Secs() * 100;
                        }
                        menu_action = ASCII_one;
                }
                else {
                        LCDwriteMsg(ssWrRTCtime,0,0);
                        for (i=0; i<17; i++){
                                KeybBuffer[i] = 0;
                        }
                        KeybBuffIndex = 0;
                        KeybBuffInput_IsON = TRUE;
                }
        }
        KeybBuffer[16]=0;
        KeybBuffer[KeybBuffIndex]=0;
        LCDwriteMsg(KeybBuffer,1,0);


        break;
case 'A': //   *00#    command: set tracker to auto mode.
        TrackerMode = TRACK_MODE_AUTOTRACK;
        ParkFlag = FALSE;
        menu_action = ASCII_one;
```

```
        break;
case 'M':       //      *55#    command: set tracker to manual mode.
        TrackerMode = TRACK_MODE_MANUAL;
        ParkFlag = FALSE;
        menu_action = ASCII_one;
        break;
case 'P'://*99# command: set tracker to auto mode & setpoint to parking position.
        ParkFlag = TRUE;
        TrackerMode = TRACK_MODE_AUTOTRACK;
        menu_action = ASCII_one;
        break;
case 'L':       //      *25# command: specify the datalogging interval in seconds
        menu_action = 'L';
        if (KeybBuffInput_IsON == FALSE) {
                if (inKey == CR) {
                        if (KeybBuffIndex >=4){
                                xDatalogInterval = 0;

                                for (i=0; i<KeybBuffIndex-1; i++){
                                        xDatalogInterval += ((KeybBuffer[i] - '0') *
                                              powOf10(KeybBuffIndex-2-i));
                                }

                                if (xDatalogInterval > 360000) {//if interval > 1h
                                        DatalogInterval = 10000;//then default to 10s
                                }
                                else {
                                        DatalogInterval = xDatalogInterval;
                                }
                        }
                        menu_action = ASCII_one;
                }
                else {
                        LCDwriteMsg("Interval<=360000",0,0);
                        for (i=0; i<17; i++){
                                KeybBuffer[i] = 0;
                        }
                        KeybBuffIndex = 0;
                        KeybBuffInput_IsON = TRUE;
                        //I2A(&DatalogInterval, KeybBuffer, 16, cPadZero);
                        dtostrf(DatalogInterval, 6,0,  KeybBuffer);
                        KeybBuffer[16]=0;
                        LCDwriteMsg(KeybBuffer,1,0);
                }
        }
        else {
                KeybBuffer[16]=0;
                KeybBuffer[KeybBuffIndex]=0;
                LCDwriteMsg(KeybBuffer,1,0);
        }
        break;
case 'E':       //      *21#    command: inputs Stay error
        menu_action = 'E';
        if (KeybBuffInput_IsON == FALSE) {
                if (inKey == CR) {
                        if (KeybBuffIndex >=3){
                             xStayCoarseError = strtod((const char *)KeybBuffer,
                                      &dummyPtr) / 100;
                                if ((xStayCoarseError <= 0) || xStayCoarseError > 5)
                                {//if error > 5deg, then: default to 0.25 deg of arc
                                        xStayCoarseError = 0.25
                                }
                        }
                        TrackersStateData.HourAngle_StayError = xStayCoarseError;
                        TrackersStateData.DeclAngle_StayError = xStayCoarseError;
                        menu_action = ASCII_one;
                }
                else {
                        LCDwriteMsg("0<CoarseErr<500%",0,0);
                        for (i=0; i<17; i++){
                                KeybBuffer[i] = 0;
                        }
                        KeybBuffIndex = 0;
                        KeybBuffInput_IsON = TRUE;
                        dtostrf(TrackersStateData.HourAngle_StayError,
                                5,2,KeybBuffer);
                        KeybBuffer[5]=0;
```

```
                        LCDwriteMsg(KeybBuffer,1,0);
                }
        }
        else {
                KeybBuffer[16]=0;
                KeybBuffer[KeybBuffIndex]=0;
                LCDwriteMsg(KeybBuffer,1,0);
        }
        break;
case 'e':       //      *20# command: inputs seekError
        menu_action = 'e';
        if (KeybBuffInput_IsON == FALSE) {
                if (inKey == CR) {
                        if (KeybBuffIndex >=3)
                        {
                                xSeekFineError = strtod((const char *)KeybBuffer,
                                        &dummyPtr) / 100;
                                if ((xSeekFineError <= 0) || xSeekFineError > 5)
                                { //if error > 5 degrees
                                        xSeekFineError = 0.25;//then default to 0.25deg
                                }
                        }
                        TrackersStateData.HourAngle_SeekError = xSeekFineError;
                        TrackersStateData.DeclAngle_SeekError = xSeekFineError;
                        menu_action = ASCII_one;
                }
                else {
                        LCDwriteMsg("0%<FineErr<=500%",0,0);
                        for (i=0; i<17; i++){
                                KeybBuffer[i] = 0;
                        }
                        KeybBuffIndex = 0;
                        KeybBuffInput_IsON = TRUE;
                        dtostrf(TrackersStateData.HourAngle_SeekError,
                                5,2,KeybBuffer);
                        KeybBuffer[5]=0;
                        LCDwriteMsg(KeybBuffer,1,0);
                }
        }
        else {
                KeybBuffer[16]=0;
                KeybBuffer[KeybBuffIndex]=0;
                LCDwriteMsg(KeybBuffer,1,0);
        }
        break;
case ASCII_hash://hash pressed:  real time clock reset (now desabled) and adjust
        //enable the commented lines for activating RTC reset to firmware default.
        /*wait until obtaing semafore for accessing portC
        while (GetSemaphore(SEMAF_PORTC, PORTCUSER_RTC_CLKIO) == FALSE) {;}
        {
         RTC_ResetInit();
              WrClockBurst((char *)dWrRTCtime);
              RTC_Adjust((uint8_t *)dWrRTCtime);
        }
        ReleaseSemaphore(SEMAF_PORTC, PORTCUSER_RTC_CLKIO); //release portC semafore
        menu_action = ASCII_one;
        */
        menu_action = 'C'; //trigger real time clock adjust
        inKey = 0;
        break;
case ASCII_star://star pressed, then:
        //begin an exetended command in the form: *nn#  where:
        // 'nn' is the command and '*' is the prolog and '#' the epilog.
        LCDwriteMsg("Enter Cmand:*nn#",0,0);
        if (KeybBuffInput_IsON == FALSE) {
                for (i=1; i<17; i++){
                        KeybBuffer[i] = 0x20; //blank
                }
                KeybBuffer[0] = '*';
                KeybBuffIndex = 1;
                KeybBuffInput_IsON = TRUE; //
        }
        KeybBuffer[16]=0;
        LCDwriteMsg(KeybBuffer,1,0);
        menu_action = ASCII_star;
        break;
}
```

```
                //this gives a visual signal that the system is still looping normally.
                BlinkDot(&toggle, 1,blkCol);
                blkCol = 9;
        };
        return 0;
}//end of program
/*---------
```

File: Datalogger.h (definitions file for datalogging interface functions)

```
*///-------------------------------------------------------------------------
#define CHECKLIST0 0  //this include all data. not included in the profile checklist table
#define CHECKLIST1 1  //see profile checklist table
#define CHECKLIST2 2  //see profile checklist table
#define CHECKLIST3 3  //see profile checklist table
#define TO_SDCARD 0    //datalog to local media SD card via SPI
#define TO_RS232 1     //datalog to external device (PC) via serial port RS232.
#define TO_M2M 2       //datalog to external device (PC) via (SPI to) M2M wireless interface.
#define LOG_FIELDTYPE_NIL 0
#define LOG_FIELDTYPE_UINT8 1
#define LOG_FIELDTYPE_UINT16 16
#define LOG_FIELDTYPE_UINT32 32
#define LOG_FIELDTYPE_UINT64 64
#define LOG_FIELDTYPE_INT 2
#define LOG_FIELDTYPE_FLOAT 4
#define LOG_FIELDTYPE_DOUBLE 8 //although for the time being, avr-gcc doubles are same as floats
#define LOG_FIELDTYPE_1CHAR 3
#define LOG_FIELDTYPE_MCHAR 5
/*---------
```

File: Datalogger.c (source code of MCU's side c datalogging functions)

```
*///-------------------------------------------------------------------------
typedef struct DataLogLine{
        char CSV_ColumnTitle[13];
        uint8_t CheckList1;
        uint8_t CheckList2;
        uint8_t CheckList3;
        void *fieldAddress;
        uint8_t fieldType;
}LogLine;

//Profile Checklist table
LogLine EEMEM EEtimer0Tic             = {"Timer0Tic" ,       1, 0, 1, (uint32_t*)&timer0Tic,
        LOG_FIELDTYPE_UINT32};
LogLine EEMEM EELastSampTic           = {"LastSampleTic", 0, 0, 0, (uint32_t*)&LastSampTic,
        LOG_FIELDTYPE_UINT32};
LogLine EEMEM EELocalTime             = {"LocalTime",        1, 0, 1, (char*)&sLogLocalTime,
        LOG_FIELDTYPE_MCHAR};
LogLine EEMEM EESolarTime             = {"SolarTime",        1, 0, 1, (char*)&sSolarTime,
        LOG_FIELDTYPE_MCHAR};
LogLine EEMEM EELocalTimeMins         = {"LocalTimeMins", 1, 0, 1, (float*)&LocalTime_inMinutes,
        LOG_FIELDTYPE_FLOAT};
LogLine EEMEM EESolarTimeMins         = {"SolarTimeMins", 1, 0, 1, (float*)&SolarTime_inMinutes,
        LOG_FIELDTYPE_FLOAT};
LogLine EEMEM EECurDayOfYear  = {"CurDayOfYear",     0, 0, 1, (uint16_t*)&CurDayOfYear,
        LOG_FIELDTYPE_UINT16};
LogLine EEMEM EEFractionalYear        = {"FractionlYear", 0, 0, 1, (float*)&FractionalYear,
        LOG_FIELDTYPE_FLOAT};
LogLine EEMEM EEEquatnOfTime  = {"EquatnOfTime",     0, 0, 1, (float*)&EqOfTime_Minutes,
        LOG_FIELDTYPE_FLOAT};
LogLine EEMEM EESunriseAngle  = {"SunriseAngle",     1, 0, 1, (float*)&SunriseAngle,
        LOG_FIELDTYPE_FLOAT};
LogLine EEMEM EEHourAngle     = {"HourAngle",        1, 0, 1, (float*)&StHourAngle,
        LOG_FIELDTYPE_FLOAT};
LogLine EEMEM EEDeclAngle     = {"DeclAngle",        1, 0, 1, (float*)&StDeclAngleDeg,
        LOG_FIELDTYPE_FLOAT};
LogLine EEMEM EEStPtHourAngle         = {"StPtHourAngle", 1, 0, 0, (float*)&StPtHourAngle,
        LOG_FIELDTYPE_FLOAT};
LogLine EEMEM EEStPtDeclAngle         = {"StPtDeclAngle", 1, 0, 0, (float*)&StPtDeclAngle,
        LOG_FIELDTYPE_FLOAT};
LogLine EEMEM EEtrakDeclAngle         = {"ActualDeclAng", 1, 0, 0, (float*)&ActualDeclAngle,
        LOG_FIELDTYPE_FLOAT};
```

```
LogLine EEMEM EEtrakHourAngle        = {"ActualHourAng", 1, 0, 0, (float*)&ActualHourAngle,
        LOG_FIELDTYPE_FLOAT};
LogLine EEMEM EEtrakCurPWM    = {"TrackersPWM",      0, 0, 0, (float*)&curTrackersPWM_DC,
        LOG_FIELDTYPE_FLOAT};
LogLine EEMEM EEtrakState     = {"TrackersState", 1, 0, 0, (uint8_t*)&(TrackersState),
        LOG_FIELDTYPE_UINT8};
LogLine EEMEM EEChgPpSetSpeed        = {"ChgPpSetSpeed", 0, 0, 0, (float*)&StPtChgPumpSpeed,
        LOG_FIELDTYPE_FLOAT};
LogLine EEMEM EEChgPpRealSpd  = {"ChgPpRealSpd",     1, 0, 0, (float*)&ActualChgPumpSpeed,
        LOG_FIELDTYPE_FLOAT};
LogLine EEMEM EEChgPpPWM      = {"ChgPpCurPWM",      0, 0, 0, (float*)&curChgPumpPWM_DC,
        LOG_FIELDTYPE_FLOAT};
LogLine EEMEM EEPyrHeliometer        = {"PyrHelioMtrVi", 1, 0, 0,
        (float*)&(ADC_LastSample[0][ADC_xmuxAddr_PyrHeliometer_Vi]), LOG_FIELDTYPE_FLOAT};
LogLine EEMEM EETrackers_Vout        = {"Trackres_Vout", 0, 0, 0,
        (float*)&(ADC_LastSample[0][ADC_xmuxAddr_Trackers_Vout]), LOG_FIELDTYPE_FLOAT};
LogLine EEMEM EETrackers_Iout        = {"Trackers_Iout", 0, 0, 0,
        (float*)&(ADC_LastSample[0][ADC_xmuxAddr_Trackers_Iout]), LOG_FIELDTYPE_FLOAT};
LogLine EEMEM EESouthPhDiodVi        = {"SouthPhDiodVi", 1, 0, 0,
        (float*)&(ADC_LastSample[0][ADC_xmuxAddr_SouthAlignPHDiode]), LOG_FIELDTYPE_FLOAT};
LogLine EEMEM EENorthPhDiodVi        = {"NorthPhDiodVi", 1, 0, 0,
        (float*)&(ADC_LastSample[0][ADC_xmuxAddr_NorthAlignPHDiode]), LOG_FIELDTYPE_FLOAT};
LogLine EEMEM EENTCambient_Vi        = {"NTCambient_Vi", 0, 0, 0,
        (float*)&(ADC_LastSample[0][ADC_xmuxAddr_NTCambientTemp_Vi]), LOG_FIELDTYPE_FLOAT};
LogLine EEMEM EERcvrInletTemp        = {"RcvrInletTemp", 1, 0, 0,
        (float*)&(TDC_LastSample[0][TDC_xmuxAddr_TRcvrIn]), LOG_FIELDTYPE_FLOAT};
LogLine EEMEM EERcvrSurfTempIn       = {"RcvSurfTempIn", 1, 0, 0,
        (float*)&(TDC_LastSample[0][TDC_xmuxAddr_TRcvrSurfIn]), LOG_FIELDTYPE_FLOAT};
LogLine EEMEM EERcvrOutltTemp        = {"RcvrOutltTemp", 1, 0, 0,
        (float*)&(TDC_LastSample[0][TDC_xmuxAddr_TRcvrOut]), LOG_FIELDTYPE_FLOAT};
LogLine EEMEM EERcvrSurfTempOut      = {"RcvSurfTempOu", 1, 0, 0,
        (float*)&(TDC_LastSample[0][TDC_xmuxAddr_TRcvrSurfOut]), LOG_FIELDTYPE_FLOAT};
LogLine EEMEM EETESinletTemp  = {"TESinletTemp",     1, 0, 0,
        (float*)&(TDC_LastSample[0][TDC_xmuxAddr_TESin]), LOG_FIELDTYPE_FLOAT};
LogLine EEMEM EETESoutletTemp        = {"TESoutletTemp", 1, 0, 0,
        (float*)&(TDC_LastSample[0][TDC_xmuxAddr_TESout]), LOG_FIELDTYPE_FLOAT};
LogLine EEMEM EETESprfTemp1_1        = {"TESprfTemp_A1", 1, 0, 0,
        (float*)&(TDC_LastSample[0][TDC_xmuxAddr_TES_LevA1]), LOG_FIELDTYPE_FLOAT};
LogLine EEMEM EETESprfTemp2_1        = {"TESprfTemp_B1", 1, 0, 0,
        (float*)&(TDC_LastSample[0][TDC_xmuxAddr_TES_LevB1]), LOG_FIELDTYPE_FLOAT};
LogLine EEMEM EETESprfTemp3_1        = {"TESprfTemp_C1", 1, 0, 0,
        (float*)&(TDC_LastSample[0][TDC_xmuxAddr_TES_LevC1]), LOG_FIELDTYPE_FLOAT};
LogLine EEMEM EETESprfTemp4_1        = {"TESprfTemp_D1", 1, 0, 0,
        (float*)&(TDC_LastSample[0][TDC_xmuxAddr_TES_LevD1]), LOG_FIELDTYPE_FLOAT};
LogLine EEMEM EETESprfTemp5_1        = {"TESprfTemp_E1", 1, 0, 0,
        (float*)&(TDC_LastSample[0][TDC_xmuxAddr_TES_LevE1]), LOG_FIELDTYPE_FLOAT};
LogLine EEMEM EETESprfTemp6_1        = {"TESprfTemp_F1", 1, 0, 0,
        (float*)&(TDC_LastSample[0][TDC_xmuxAddr_TES_LevF1]), LOG_FIELDTYPE_FLOAT};
LogLine EEMEM EETESprfTemp7_1        = {"TESprfTemp_G1", 1, 0, 0,
        (float*)&(TDC_LastSample[0][TDC_xmuxAddr_TES_LevG1]), LOG_FIELDTYPE_FLOAT};
LogLine EEMEM EETESprfTemp8_1        = {"TESprfTemp_H1", 1, 0, 0,
        (float*)&(TDC_LastSample[0][TDC_xmuxAddr_TES_LevH1]), LOG_FIELDTYPE_FLOAT};
LogLine EEMEM EETESprfTemp9_1        = {"TESprfTemp_I1", 1, 0, 0,
        (float*)&(TDC_LastSample[0][TDC_xmuxAddr_TES_LevI1]), LOG_FIELDTYPE_FLOAT};
LogLine EEMEM EEPlantAmbtTemp        = {"PlantAmbtTemp", 1, 0, 0,
        (float*)&(TDC_LastSample[0][TDC_xmuxAddr_PlantAmbientTemp]), LOG_FIELDTYPE_FLOAT};
LogLine EEMEM EEMBrdAmbtTemp         = {"MBrdAmbtTemp",     1, 0, 0, (float*)&AmbientTemp_inCelsius,
        LOG_FIELDTYPE_FLOAT};
LogLine EEMEM EESysStatUserIDs       = {"SysTimingFlgs", 0, 0, 1, (float*)&SysStatUserIDs,
        LOG_FIELDTYPE_UINT8};
uint8_t EEMEM EEListBottom = 0;
const void *TopLine = &EEtimer0Tic;          //pointer to top of List.
const void *SecondLine = &EELocalTime;       //pointer to 2nd element of the List.
const void *BottomLine = &EEListBottom;

//This function reads the CSV header from EEPROM and data logs it to specified destination
uint8_t  InitDataLog(uint8_t LogChecklist, uint8_t LogDestination)
{
        LogLine TitLogLine;
        uint8_t i, LogLen, log_it= FALSE, nRet = 0;
        uint16_t iOffset;

        //initialize the destination device.
        switch (LogDestination)
        {
        case TO_RS232:
                USARTinit();   //initialize the serial port RS232.
```

```
                break;
        case TO_SDCARD:
                //not implemented
                break;
        case TO_M2M:
                //not implemented
                break;
        }

        //LogLen = ((&EETimer0tic - &EEListBottom) / sizeof(LogLine)) ;
        LogLen = ((BottomLine - TopLine) / sizeof(LogLine)) ;
        for (i = 0; i < LogLen; i++) {//traverse the LogList table for the supplied checklist.
                //retrieve next LogLine Definition struct.
                iOffset = i * sizeof(LogLine);
                eeprom_read_block((void*)&TitLogLine,(const void*)&EEtimer0Tic+iOffset,
                        sizeof(LogLine));

                switch (LogChecklist)
                {
                        case CHECKLIST0:        //Datalog every field.
                                log_it = TRUE;
                                break;
                        case CHECKLIST1:        //Datalog fields marked 1 on Checklist1 field;
                                log_it = (TitLogLine.CheckList1);
                                break;
                        case CHECKLIST2:        //Datalog fields marked 1 on Checklist2 field;
                                log_it = (TitLogLine.CheckList2);
                                break;
                        case CHECKLIST3:        //Datalog fields marked 1 on Checklist3 field;
                                log_it = (TitLogLine.CheckList3);
                                break;
                }

                if (log_it)
                {
                //if the current field is checklisted (included) in this datalogging configuration;
                        nRet++;
                        if (nRet > 1){  //if field number  > 1;
                                DataLogSendChar(',', LogDestination); //send field separator
                        }
                        DataLogSendString(TitLogLine.CSV_ColumnTitle,
                                sizeof(TitLogLine.CSV_ColumnTitle),  LogDestination);
                }
        }

        if (nRet > 0)//then:
        {//we are at end of line: if at least one field checklisted, send a line epilog.
                //nRet++;
                //DataLogSendString(LineEpilog, 2, LogDestination);
                DataLogSendChar(CR, LogDestination);  //send line epilog carriage
                DataLogSendChar(LF, LogDestination);  //send line epilog linefeed
        }
        return (nRet>0)?0:1;
};
```

```
//The following function reads the current (real time) data from RAM and ...
// data logs to specified destination; data set defined by datalogging configuration from EEPROM.
uint8_t DataLog(uint8_t LogChecklist, uint8_t LogDestination)
{
        LogLine xLogLine;
        uint8_t i, LogLen, log_it= FALSE, nRet = 0, sLen, sreg;
        uint16_t iOffset;
        char *sValue = "+1.2345678901234567890", *cValue;
        uint32_t uiValue;
        float fValue, *pValue;
        //LogLen = ((&EETimer0tic - EEListBottom) / sizeof(LogLine)) ;
        LogLen = ((BottomLine - TopLine) / sizeof(LogLine)) ;
        for (i = 0; i < LogLen; i++) //traverse the LogList table for the supplied checklist.
        {
                //retrieve next LogLine Definition struct.
                iOffset = i * sizeof(LogLine);
                sreg = SREG;
                cli();
                eeprom_read_block((void *)&xLogLine, (const void*)&EEtimer0Tic+iOffset,
                        sizeof(LogLine));
                SREG = sreg;
                switch (LogChecklist)
                {
                case CHECKLIST0:        //Datalog every field.
                        log_it = TRUE;
                        break;
                case CHECKLIST1:        //Datalog fields marked 1 on Checklist1 field;
                        log_it = (xLogLine.CheckList1);
                        break;
                case CHECKLIST2:        //Datalog fields marked 1 on Checklist2 field;
                        log_it = (xLogLine.CheckList2);
                        break;
                case CHECKLIST3:        //Datalog fields marked 1 on Checklist3 field;
                        log_it = (xLogLine.CheckList3);
                        break;
                }
                if (log_it)
                {
                        nRet++;
                        switch (xLogLine.fieldType)
                        {
                        case LOG_FIELDTYPE_NIL:
                                sLen = 1;
                                sValue = " "; //send space
                                break;
                        case LOG_FIELDTYPE_UINT8:
                                sLen = 3;
                                uiValue = *((uint8_t *)(xLogLine.fieldAddress));
                                I2A(&uiValue, sValue, 3, cPadSpace);
                                break;
                        case LOG_FIELDTYPE_UINT16:
                                sLen = 5;
                                uiValue = *((uint16_t*)(xLogLine.fieldAddress));
                                I2A(&uiValue, sValue, 5, cPadSpace);
                                break;
                        case LOG_FIELDTYPE_UINT32:
                                sLen = 10;
                                I2A((xLogLine.fieldAddress), sValue, 10, cPadSpace);
                                break;
                        case LOG_FIELDTYPE_UINT64:
                                sLen = 20;
                                I2A((xLogLine.fieldAddress), sValue, 20, cPadSpace);
                                break;
                        case LOG_FIELDTYPE_INT:
                                sLen = 6;
                                fValue = *((int*)(xLogLine.fieldAddress));
                                dtostrf(fValue, 6,0,  sValue);
                                break;
                        case LOG_FIELDTYPE_FLOAT:
                                sLen = 9;
                                pValue = (float*)(xLogLine.fieldAddress);
                                dtostrf(*pValue, 9,2,  sValue);
                                break;
                        case LOG_FIELDTYPE_DOUBLE:
                                //code same as for float: cause: avr-gcc doubles are same as floats
                                sLen = 9;
                                pValue = (float*)(xLogLine.fieldAddress);
```

```
                        dtostrf(*pValue, 9,2,  sValue);
                        break;
                case LOG_FIELDTYPE_1CHAR:
                        sLen = 1;
                        sValue = (char*)(xLogLine.fieldAddress);
                        break;


                case LOG_FIELDTYPE_MCHAR:
                        sLen = 16;
                        cValue = (xLogLine.fieldAddress);
                        sValue = *(cValue+1) * 256 + *cValue;
                        sLen = strlen(sValue);
                        break;
                }
                if (nRet > 1){ //if field Nr > 1;
                        DataLogSendChar(',', LogDestination); //send field separator
                }
                DataLogSendString(sValue, sLen, LogDestination);     //send the field itself
        }
    }
    if (nRet > 0)//we are at end of line: if at least one field checklisted, send a line epilog.
    {
            //nRet++;
            //DataLogSendString(LineEpilog, 2, LogDestination);
            DataLogSendChar(CR, LogDestination);  //send line epilog carriage
            DataLogSendChar(LF, LogDestination);  //send line epilog linefeed
    }
    return (nRet>0)?0:1;
}


void DataLogSendString(char *str, uint8_t strLen, uint8_t LogDestination)
{
    uint8_t j;
    char xChar;
    j=0;
    xChar = *str;
    if (xChar != 0) {      //if not a null string
            DataLogSendChar('"', LogDestination); //send CSV field prolog
            while((xChar) && (j<strLen)) //ASCIIZ string: traverse it until got the null char.
            {
                    DataLogSendChar(xChar, LogDestination);      //send CSV field current char

                    xChar = (*(str + ++j));
            }
            DataLogSendChar('"', LogDestination); //send CSV field epilog
    }
};

void DataLogSendChar(char xChar, uint8_t LogDestination)
{
    if (xChar != 0) {      //if not the null char, send it
            switch (LogDestination)
            {
            case TO_RS232:
                USART_SendByte(xChar); //send the char to serial port RS232.
                break;
            case TO_SDCARD:
                //not implemented (out of the scope)
                break;
            case TO_M2M:
                //not implemented (out of the scope)
                break;
            }
    }
};

/*----------
```

File: USART.S (RS232 - MCU to PC - interface functions)

```
*///-------------------------------------------------------------------------;
;//USART.S file
;// RS232 serial interface routines
;//Author: Doho, G.J. (Aug2007); updated April2009 for mixed c - asm environment

#include "mixed_C_asm.h"
#include <avr/io.h>

#define genTemp R16
#define genTemp2 R17


//=============================================================================
.global USARTinit
;//void USARTinit(void); //declaration to be put in main c program
USARTinit:
        push genTemp;
        push genTemp2;
        ;//URSEL is bit 7 of both UCSRC and BaudRateRegister_H shared locus registers
        //URSEL=1;enable access to UCSRC
        ldi genTemp, 0x86;     //URSEL=1, Assync, no parity, 1 stop, 8 data
        out UCSRA, genTemp;

        //URSEL = 0, enable access to BaudRateRegister_; clear bits 11:8
        // of baud rate in BaudRateRegister_H
        ldi genTemp, 0;
        out  UBRRH, genTemp

        ;//Baudrate settings for 8MHz. For other freqs/bauds, refer the datasheet.
        ;//128000bps:
        //ldi genTemp, 02;     //U2X = 1
        //ldi genTemp2, 7;     //BaudRateRegister_H118:BaudRateRegister_L = 0x07 ;
        //and bits11:8 = 0; U2X=1 in UCSRCA, 8MHz) request 128000 baud


        ;//115200bps:
        //ldi genTemp, 02;     //U2X = 1
        //ldi genTemp2, 8;     //BaudRateRegister_H118:BaudRateRegister_L = 0x08 ;
        //and bits11:8 = 0; U2X=1 in UCSRCA, 8MHz) request 115200 baud

        ;//57600bps:
        //ldi genTemp, 02;     //U2X = 1
        //ldi genTemp2, 16;    //BaudRateRegister_H118:BaudRateRegister_L = 16 ;
        //and bits11:8 = 0; U2X=1 in UCSRCA, 8MHz) request 57600 baud

        ;//38400bps: the highest proven to work at 8MHz. Higher have framing errors. …
        //… Causes: see baud generation tables
        ldi genTemp, 02;       //U2X = 1

        ldi genTemp2, 25;      //BaudRateRegister_H118:BaudRateRegister_L = 25 ;
        //and bits11:8 = 0; U2X=1 in UCSRCA, 8MHz) request 38400 baud

        ;//19200bps:
        //ldi genTemp, 02;     //U2X = 1
        //ldi genTemp2, 51;    //BaudRateRegister_H118:BaudRateRegister_L = 51 ;
        //and bits11:8 = 0; U2X=1 in UCSRCA, 8MHz) request 19200 baud

        ;//9600bps:
        //ldi genTemp, 02;     //U2X = 1
        //ldi genTemp2, 103;   //BaudRateRegister_H118:BaudRateRegister_L = 103
        // and bits11:8 = 0; U2X=1 in UCSRCA, 8MHz) request 9600 baud

        //clear flags: RxC, TxC, UDRE, FE, DOR, PE; and bit MPCM; set or clear bit
        out UCSRA, genTemp; U2X.
        out UBRRL, genTemp2

        ldi genTemp, (1<<TXEN);  //|(1<<RXEN);   //now, only transmission enabled.
        out UCSRB, genTemp
        pop genTemp2
        pop genTemp
        ret
```

```
;====================================
.global USART_SendByte
//void USART_SendByte(uint8_t xByte);  //declaration to be put in main c program
//input: R24  with byte to transmit
USART_SendByte:               //USART_TxD:
        //push genTemp
//waitUDDRE:
1:      sbis UCSRA, UDRE
        rjmp 1b //waitUDDRE
        out UDR, R24
        ret

;====================================
.global USART_ReceiveByte
//output: R25:R24  = 0:R24 with received byte
USART_ReceiveByte:            //USART_RxD:
//waitRXC:
2:      sbis UCSRA, UDRE
        rjmp 2b //waitRXC
        in R24, UDR
        eor R25,R25
        ret

;==============================
.global USART_SendPage
//void USART_SendPage(char *ptrPageBuffer,uint8_t nBytes);
USART_SendPage: //inputs: R25:R24 with page or bufer location; and R22 with pageLength
        push YH
        push YL
        push genTemp
        mov YH, R25
        mov YL, R24
        mov genTemp, R22
//repTXD:
3:      ld R24, Y+
        rcall USART_SendByte //USART_TXD
        dec genTemp
        brne 3b //repTXD
        pop genTemp
        pop YL
        pop YH
        ret

;==================================
.global USART_ReceivePage
//void USART_ReceivePage(char *ptrPageBuffer,uint8_t nBytes);
USART_ReceivePage: //inputs: R25:R24 with page or bufer location; and R22 with pageLength
        push YH     //upon return, the specified page buffer locus, is filled with the received data.
        push YL
        push R25
        push R24
        push genTemp
        mov YH, R25
        mov YL, R24
        mov genTemp, R22
//repRXD:
4:      rcall USART_ReceiveByte   //USART_RXD
        st  Y+, R24
        dec genTemp
        brne 4b //repRXD
        pop genTemp
        pop R24
        pop R25
        pop YL
        pop YH
        ret


/*---------
```

File: lcd.S (LCD interface routines)

```
*///-------------------------------------------------------------------------//LCD.s file
//Autor: Doho, G.J. (July 2007)
//updated Nov/08 to run on avr-gcc mixed  c - asm developm/ environment
//
//LCD handling functions for 4 bits, write only (WE pin tied to GND by hardware)
//proven to work at 4 to 8MHz. For best performance at very low or very high speeds,
//may need adjusting the delay routines accordingly.
//The use of a different LCD module may also require delays revision.
// ------------------------------------------------
// GCC calling conventions:
//      arg1: r25:r24 (r24 for 8bit); arg2: r23:r22; ... arg9: r9:r8; more args: pushed on stack;
//      return value: r25:r24 (8 bit extended with sign); r25:r22 (32bit); r25:r18 (64bit);
//      Declare each function as .global; as well as: prototype the function in the C main file;
//   be sure to save all variables prior to use them and rstore prior to ret or iret;
//   If calling any C function, know that C procedures freely use R18-R27, R30-R31 so save them
//       prior to call a C function.
//
;============================================================================
#include "mixed_C_asm.h"
#include <avr/io.h>
//----------------------------
#define LCDdata R24
#define LCDline R24
#define LCDcol R22
#define genTemp R16
#define genTemp2 R17

#define LCDDDR_WRITE  0xFE    ; Data direction for LCD write
#define LCDport PORTA
#define LCDportDDR  DDRA
#define LCDFUNCTIONSET_CMD 0x28       ;DL=0=>4bits, N=1=>2rows, F=0=>5x8 dotmtx
#define LCDON_CURSOFF_CMD 0x0C        ;D=1=>dply on, C=0=>cursor off; B=0 => no blink
#define LCDON_CURSON_CMD 0x0F ;D=1=>dply on, C=1=>cursor on, B= 1 => blink
#define LCDCURS_SCR_SHIFT_CMD  0x14   ;S/C=0=>shift cursor, R/L=1 => shiftright
#define LCDENTRYMODE_CMD 0x06 ;I/D=1=>inc AC,  S=0 no screen shifting
#define LCDRETHOME_CMD 0x02   ;return to LCD home pos.
#define LCDCLEAR_CMD 0x01             ;clear LCD and return to home pos.
#define LCDADDRESS_CMD  0x80  ;bit7=1, followed by 7 bit ddram address
#define LCD_EN 0x02
//#define LCD_RW  1    ;now this LCD pin fixed to GND(implies: LCD writes only)
#define LCD_RS 0x03                   ;

;void LCDreset(void)  // to declare on function prototypes area, in main c file
.global LCDreset;

LCDreset:       //this performs LCD module initialization for 4bits interface
        push genTemp
        push LCDdata
        ldi genTemp, LCDDDR_WRITE             ;data direction for LCD write
        out     LCDportDDR,genTemp
        ldi genTemp, 0x00
        out LCDport, genTemp
        call dly_10ms
        call dly_10ms
        call dly_10ms
        call dly_10ms
        call dly_10ms
        ldi LCDdata, LCDFUNCTIONSET_CMD;28  Function set (request 4 bit i/o)
        rcall LCDwriteCMD //Hnibble           ;(hi nibble)
        call dly_10ms
        call dly_10ms
        ldi LCDdata, LCDFUNCTIONSET_CMD;28 Function set DL=0=>4bits, N=1=>2rows, F=0=>5x8 dotmtx
        rcall LCDwriteCMD
        call dly_10ms
        ldi LCDdata, LCDON_CURSOFF_CMD ;0C D=1=>dply on, C=0=>cursor off; B=0 => no blink
        rcall LCDwriteCMD
        call dly_10ms
        ldi LCDdata, LCDCLEAR_CMD             ;01 clear LCD and return to home pos.
        rcall LCDwriteCMD
        call dly_10ms
        ldi LCDdata, LCDENTRYMODE_CMD ;06 I/D=1=>inc AC,  S=0 no screen shifting
        rcall LCDwriteCMD
        call dly_10ms
        pop LCDdata
```

```
        pop genTemp
        ret
;=============================================================================
;void LCDwriteCMD(uint8_t LCDdata)  // to declare on function prototypes area, in main c file
.global LCDwriteCMD;

LCDwriteCMD:   //inputs: LCDdata is R24
                        //LCDdata contains 8 bit scommand to be writen to LCD;
        //parse and send the high Nibble to the LCD
        push genTemp
        push LCDdata

        ldi genTemp, LCDDDR_WRITE            ;data direction for LCD write
        out     LCDportDDR,genTemp
        mov genTemp, LCDdata
        andi genTemp, 0xF0     ;zero the low Nibble
        out LCDport, genTemp
        cbi LCDport, LCD_RS
        nop
        nop
        nop
        nop
        nop
        sbi LCDport, LCD_EN     //assert enable: set bit 1 on LCDport
        nop     //wait 290ns   ;290ns< 2nops at 4MHz and < 5 nops at 16MHz
        nop
        nop
        nop
        nop
        cbi LCDport, LCD_EN //pull enable down: clear bit 1 on LCDport
        nop     //wait 344ns   ;344ns< 2nops at 4MHz and < 5 nops at 16MHz
        nop
        nop
        nop
        nop

        // now parse and send the low Nibble to the LCD
        mov genTemp, LCDdata
        swap genTemp
        andi genTemp, 0xF0     ;zero the low Nibble
        out LCDport, genTemp
        nop
        nop
        nop
        nop
        nop
        sbi LCDport, LCD_EN     //assert enable: set bit 1 on LCDport
        nop     //wait 290ns   ;290ns< 2nops at 4MHz and < 5 nops at 16MHz
        nop
        nop
        nop
        nop
        cbi LCDport, LCD_EN //pull enable down: clear bit 1 on LCDport
        nop     //wait 344ns   ;344ns< 2nops at 4MHz and < 5 nops at 16MHz
        nop
        nop
        nop
        nop
        sbi LCDport, LCD_RS
        call dly_100us
        pop LCDdata
        pop genTemp
        ret

;=============================================================================
;void LCDwriteDATA(uint8_t LCDdata)  // to declare on function prototypes area, in main c file
.global LCDwriteDATA;

LCDwriteDATA:          //inputs: LCDdata is R24
        //LCDdata contains 8 bit data to be writen to LCD;
        //parse and send the high Nibble to the LCD
        push genTemp
        push LCDdata


        ldi genTemp, LCDDDR_WRITE            ;data direction for LCD write
        out     LCDportDDR,genTemp
```

```
        mov genTemp, LCDdata
        andi genTemp, 0xF0      ;zero the low Nibble
        out LCDport, genTemp
        sbi LCDport, LCD_RS
        nop
        nop
        nop
        nop
        nop
        sbi LCDport, LCD_EN     //assert enable: set bit 1 on LCDport
        nop     //wait 290ns    ;290ns< 2nops at 4MHz and < 5 nops at 16MHz
        nop
        nop
        nop
        nop
        cbi LCDport, LCD_EN //pull enable down: clear bit 1 on LCDport
        nop     //wait 344ns    ;344ns< 2nops at 4MHz and < 5 nops at 16MHz
        nop
        nop
        nop
        nop
        // now parse and send the low Nibble to the LCD
        mov genTemp, LCDdata
        swap genTemp
        andi genTemp, 0xF0      ;zero the low Nibble
        out LCDport, genTemp
        sbi LCDport, LCD_RS
        nop
        nop
        nop
        nop
        nop
        sbi LCDport, LCD_EN     //assert enable: set bit 1 on LCDport
        nop     //wait 290ns    ;290ns< 2nops at 4MHz and < 5 nops at 16MHz
        nop
        nop
        nop
        nop
        cbi LCDport, LCD_EN //pull enable down: clear bit 1 on LCDport
        nop     //wait 344ns    ;344ns< 2nops at 4MHz and < 5 nops at 16MHz
        nop
        nop
        nop
        nop
        sbi LCDport, LCD_RS
        call dly_1ms
        pop LCDdata
        pop genTemp
        ret


;=============================================================
;void LCDaddress(uint8_t LCDline, uint8_t LCDcol)
.global LCDaddress;

LCDaddress:     //inputs: LCDline in R24, and LCDcol in R22
                        //LCDline(Y=0 or 1), and LCDcol(X=0 to 15)
        push genTemp
        push genTemp2
        push LCDdata
        push LCDline
        push LCDcol
        nop
        nop
        nop
        nop
        nop
        mov genTemp, LCDline
        mov genTemp2, LCDcol
        neg genTemp //neg 0 =0x00; neg 1 =0xFF

        // andi with 00=00h; andi with 0xFF = 40h;
        // which are the 0 positions of line 0 and line 1 respectiv/
        andi genTemp, 0x40
        add genTemp, genTemp2  // adds char position (0 to 15) into line
        ori genTemp, LCDADDRESS_CMD
        mov LCDdata, genTemp   ;request adress write.
```

```
        rcall LCDwriteCMD
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        call dly_1ms
        pop LCDcol
        pop LCDline
        pop LCDdata
        pop genTemp2
        pop genTemp
        ret

////-------------------------------------------------
// delay routines
// general observation: these delays were programed for 4MHz. This means that,
// for 8MHz thay yield half the delay time thay were suposed to produce.
// We did not adapt them for 8MHz once, the LCD used (LMB162ABC) was not affected.
// However, it should be pointed out that higher (or even very low) system frequencies
// may require revisions on the delays, so as to avoid race conditions or any other
// unwanted and unpredictable LCD behavior.

.global dly_1s
//void dly_1s(void);
dly_1s:         //revew this: added pushes and pops and not only
        push genTemp
        ldi     genTemp, 100
1: //loop1s:
        call dly_10ms
        dec     genTemp
        brne    1b //loop1s
        pop genTemp
        ret
;================================
.global dly_1hS
//void dly_1hS(void);
dly_1hS:                //revew this: added pushes and pops and not only
        push genTemp
        ldi     genTemp, 50
2: //loop1hs:
        call dly_10ms
        dec     genTemp
        brne    2b //loop1hs
        pop genTemp
        ret
;==============================
;100us //4 MHz
.global dly_100us
//void dly_100us(void);
dly_100us:      //revewed: added pushes and pops and not only
        push genTemp
        ldi     genTemp,131
4: //loop100us:
        dec     genTemp
        brne    4b //loop100us
        pop genTemp
        ret
;================================
.global dly_1ms
//void dly_1ms(void);
dly_1ms:         //4 MHz
```

```
        push r25
        push r24
        ldi r25, 6      ;from above
        ldi r24, 0      ;0 means 256: decrements to 255 before brne test
5: //loop1ms:
        dec r24         ;inner loop
        brne 5b //loop1ms      ;inner loop
        dec r25         ;outer loop
        brne 5b //loop1ms      ;outer loop
        ldi r24, 45     ;
6: //loop1msadj:
        dec r24         ;single loop
        brne 6b //loop1msadj   ;single loop
        pop r24
        pop r25
        ret
;=====================================
.global dly_10ms
//void dly_10ms(void);
dly_10ms:       //4 MHz
        push r25
        push r24
        ldi r25, 52     ;
        ldi r24, 0      ;0 means 256: decrements to 255 before brne test
7: //loop10ms:
        dec r24         ;inner loop
        brne 7b //loop10ms     ;inner loop
        dec r25         ;outer loop
        brne 7b //loop10ms     ;outer loop
        ldi r24, 238    ;
8: //loop10msadj:
        dec r24         ;single loop
        brne 8b //loop10msadj  ;single loop
        pop r24
        pop r25
        ret

/*---------
```

File: 1302rtc.S (Real Time Clock interface routines)

```
*///-------------------------------------------------------------------------
;=========================================================================
// Maxim-Dallas Semi DS1302 RTC 3-wire interface handling routines for AVR
// Note: initial timing considerations based on: 5V supply and 4MHz clock speed.
// routines were further updated and tested and are proven to work as well up to 8MHz.
// updated Nov/08 to run on avr-gcc mixed  c - asm developm/ environment
// Author: Doho, G.J. (Aug/Oct 2008)
//
// ------------------------------------------------
// AVR-GCC calling conventions:
//      arg1: r25:r24 (r24 for 8bit); arg2: r23:r22; ... arg9: r9:r8; more args: pushed on stack;
//      return value: r25:r24 (8 bit extended with sign); r25:r22 (32bit); r25:r18 (64bit);
//      Declare each function as .global; as well as: prototype the function in the C main file;
//   be sure to save all variables prior to use them and restore prior to ret or iret;
//   If calling any C function, know that C procedures freely use R18-R27, R30-R31 so save them
//       prior to call a C function.
//
;=========================================================================
#include "mixed_C_asm.h"
#include <avr/io.h>
//---------------------------
#define genTemp R16
#define genTemp2 R22          //will be also LSbyte of arg2
#define LCDline R24
#define LCDcol R22

//----------------------
//#define LOW(x) lo8(x)
//---------------
// In the following 6 routines (and not only), the nop's are for DS1302 timing, so as to insure
// for instance, the following timing settings: tCWH, tCC, tCCH, tDC, tCH, tCL, tCDH, tR, tF, etc.
// the calculated nops were duplicated to avoid any race conditions.
.global RTC_on
//void RTC_on(void); //declaration to be put in main c program
```

```
RTC_on: // Enables the RTC: keeps it ou from the reset state.
        sbi RTC_CTRLDDR, RTC_CS
        nop
        nop
        nop
        nop
        nop
        nop
        cbi RTC_CTRLPORT, RTC_CS
        nop
        nop
        nop
        nop
        nop
        nop
        cbi RTC_DATAPORT, RTC_SCK
        nop
        nop
        nop
        nop
        nop
        nop
        sbi RTC_CTRLPORT, RTC_CS;
        nop
        nop
        nop
        nop
        nop
        nop
        ret
//---------------------------------------
.global RTC_off
//void RTC_off(void); //declaration to be put in main c program
RTC_off: //setles the RTC back to the reset state
        nop
        nop
        nop
        nop
        nop
        nop
        cbi RTC_CTRLPORT, RTC_CS;
        nop
        nop
        //cbi RTC_DATAPORT, RTC_SCK;
        nop
        nop
        nop
        nop
        nop
        nop
        ret
//---------------------------------
.global RTC_readON
RTC_readON: //togles RTC IO line direction to input
        nop
        nop
        nop
        nop
        nop
        nop
        cbi RTC_DDRPORT, RTC_IO
        nop
        nop
        nop
        nop
        nop
        nop
        ret
        //---------------------------------
.global RTC_writeON
RTC_writeON:  //togles RTC IO line direction to output
        push genTemp
        ldi genTemp, 0xF
        nop
        nop
        nop
        nop
```

```
        nop
        nop
        //sbi RTC_DDRPORT, RTC_IO
        sbi RTC_DDRPORT, RTC_IO
        nop
        nop
        nop
        nop
        nop
        nop
        pop genTemp
        ret
//---------------------------------
.global WrByte
//void WrByte(char byteData); //declaration to be put in main c program
WrByte:  //serialy sends to the RTC a byte of data from LSBit .
//input byte in RTC_byte

        push R24 //RTC_byte
        push genTemp
        in genTemp, SREG
        push genTemp
        //cli
        ldi genTemp, 8
        rcall RTC_writeON
//SndNxtBit:
1:        // check data bit
        ror RTC_Byte;
        brcc 2f //clearBit      // if bit <> 0 then =>
        sbi RTC_DATAPORT, RTC_IO  // set rtc_data pin
        rjmp 30f  //sendBit
2: //clearBit:
        cbi RTC_DATAPORT, RTC_IO  // clear rtc_data pin
30: //sendBit:         // send a rising clock edge
        nop
        nop
        nop
        nop
        nop
        nop
        cbi RTC_DATAPORT, RTC_SCK
        nop
        nop
        nop
        nop
        nop
        nop
        sbi RTC_DATAPORT, RTC_SCK
        nop
        nop
        nop
        nop
        nop
        nop
        dec genTemp
        brne 1b //SndNxtBit
        pop genTemp
        out SREG,genTemp
        pop genTemp
        pop R24//RTC_Byte
        ret

//-----------------------------------------
.global RdByte
//char RdByte(void); //declaration to be put in main c program
RdByte: //serialy colects from the RTC a byte of data; from LSBit.
// return byte goes into RTC_Byte
        push genTemp
        ldi genTemp, 8
        rcall RTC_readON
3: //RcvNxtBit:
        nop
        nop
        nop
        nop
        nop
        nop
```

```
        sbi RTC_DATAPORT, RTC_SCK;// send a falling clock edge
        nop
        nop
        nop
        nop
        nop
        nop
        cbi RTC_DATAPORT, RTC_SCK;
        nop
        nop
        nop
        nop
        nop
        sec
        sbis RTC_PINPORT, RTC_IO
        clc
        ror RTC_Byte;
        dec genTemp
        brne 3b //RcvNxtBit
        ldi R25,0
        pop genTemp
        ret

//-----------------------------------------
.global RTC_ReadByteAtAddress
//char RTC_ReadByteAtAddress(uint16_t Addr); //declaration to be put in main c program
RTC_ReadByteAtAddress: //It reads a byte from a specified RTC address.
//input addr in RTC_byte
//return  RTC_Byte
        rcall RTC_on;
        rcall WrByte;  //firstly send the address
        rcall RdByte;  // now read data
        rcall RTC_off;
        ret
//-----------------------------------------
.global RTC_WriteByteAtAddress
//void RTC_WriteByteAtAddress(uint16_t Addr, char byteData);
RTC_WriteByteAtAddress: //It sends a byte to a specified RTC address.
// input addr in RTC_byte(arg1), input data in R22 (arg2).
        rcall RTC_on;
        rcall WrByte;  //firstly send send address
        mov RTC_Byte, genTemp2
        rcall WrByte;  // now send data
        rcall RTC_off;
        ret
//-----------------------------------------------
// gets the current secs from the RTC
.global RTC_GetSecs
//char RTC_GetSecs(void);  //declaration to be put in main c program
RTC_GetSecs:
        ldi RTC_Byte, RTC_SECS_READ
        rcall RTC_ReadByteAtAddress
        ldi R25,0 //null the MSbyte of return register pair (R25:R24).
        ret
//-----------------------------------------------
.global RdClockBurst
//void RdClockBurst(char * prtClockDataString); //declaration to be put in main c program
RdClockBurst:  //burst read RTC clock registers
//input: R25:R24 destination address in MCU RAM for read clock data
//clock registers read order: seconds, minutes, hours, date, month, day, year (, ctrlreg);
        push genTemp;
        push RTC_Byte
        push ZH
        push ZL
        push R25
        push R24
        mov ZH, R25
        mov ZL, R24
        ldi genTemp,7 ; (8; could be 8 for includiing control reg)
        rcall RTC_on;
        ldi RTC_Byte, RTC_CKBURST_READ //clock burst read address command
        rcall WrByte;  //write clock burst command
4: //repRdClock:  // burst read
        rcall RdByte
        st Z+, RTC_Byte
        dec genTemp
        brne 4b //repRdClock
```

```
        rcall RTC_off
        pop R24
        pop R25
        pop ZL
        pop ZH
        pop RTC_Byte
        pop genTemp
        ret
//-----------------------------------------------
.global WrClockBurst
//void WrClockBurst(char *ptrClockDataString); //declaration to be put in main c program
WrClockBurst:  // burst write RTC clock registers (including control reg).
//input: R25:R24 source address of write clock data
//clock registers write order: seconds, minutes, hours, date, month, day, year, ctrlreg;
        push genTemp;
        push genTemp2
        push RTC_Byte
        push ZH
        push ZL
        push R22
        push R24
        push R25
        mov ZH, R25
        mov ZL, R24

        sbi DDRD, RTC_CS       //set CS bit on Ctrl port to output
        sbi RTC_DDRPORT, RTC_SCK  //set SCK bit on data port to output
        sbi RTC_DDRPORT, RTC_IO  //set I/O bit on Data port to output

        //ldi R22, 0   //clear data protection flag
        //ldi RTC_Byte, RTC_CTRLREG_WRITE
        //rcall RTC_WriteByteAtAddress  //these 3 lines commented because pre-desabling write prot.
        // is useless for burst mode clock writes.

        //7clkdata+ 1ctrlreg: must write to ctrlreg in burst mode (with 0 to clear write prot.)
        ldi genTemp,8
        ldi RTC_Byte, RTC_CKBURST_WRITE //clock burst command
        //rcall RTC_on;
        rcall WrByte;  //write clock burst command
5: //reprWrtClockBrst:
        ld RTC_Byte, Z+
        rcall WrByte
        dec genTemp
        brne 5b //reprWrtClockBrst
        rcall RTC_off

        //clr genTemp2 //start clock
        ldi R22, 00
        ldi RTC_Byte, RTC_SECS_WRITE
        rcall RTC_WriteByteAtAddress

        ldi R22,80     //set data protection flag
        ldi RTC_Byte, RTC_CTRLREG_WRITE
        rcall RTC_WriteByteAtAddress

        pop R25
        pop R24
        pop R22
        pop ZL
        pop ZH
        pop RTC_Byte
        pop genTemp2
        pop genTemp
        ret



//-----------------------------------------
//tested. not used in this version
//RTC clock initialization to default firmware init time
.global RTC_Adjust
//void RTC_Adjust(uint8_t *ptrClkData); //declaration to be put in main c program
//input: R25:24 with ptr to clockData
RTC_Adjust:
        push genTemp
        in genTemp, SREG
```

```
        //cli
        push genTemp
        push genTemp2
        push RTC_Byte
        push ZH
        push ZL
        push R22
        mov     ZH,R25  ; Load high byte of secs address word into ZH
        mov     ZL,R24  ; Load low byte of secs address word into ZL

        sbi DDRD, RTC_CS       //set CS bit on Ctrl port to output
        sbi RTC_DDRPORT, RTC_SCK  //set SCK bit on data port to output
        sbi RTC_DDRPORT, RTC_IO  //set I/O bit on Data port to output

        //rcall RTC_on;
        ldi R22, 0     //clear data protection flag
        ldi RTC_Byte, RTC_CTRLREG_WRITE
        //rcall RTC_on;
        rcall RTC_WriteByteAtAddress

        ldi R22, 0//desable tricle charger (this is for a non-rechargeable batt, ex.: a CR2032)
        ldi RTC_Byte, RTC_TRICLE_WRITE
        //rcall RTC_on;
        rcall RTC_WriteByteAtAddress


        //ldi R22, 0   //clear data protection flag
        //ldi RTC_Byte, RTC_CTRLREG_WRITE
        //rcall RTC_WriteByteAtAddress //pre-enabling write access is useless in burst mode as
        // crtlreg must be written in burst mode.

        //7clkdata+ 1ctrlreg: must write to ctrlreg in burst mode (with 0 to clear write prot.)
        ldi genTemp,8
        ldi RTC_Byte, RTC_CKBURST_WRITE //clock burst command
        rcall RTC_on;
        rcall WrByte;  //write clock burst command
99: //reprWrtClockBrst:
        ld RTC_Byte, Z+
        rcall WrByte
        dec genTemp
        brne 99b //reprWrtClockBrst
        rcall RTC_off

        //clr genTemp2 //start clock
        ldi R22, 00
        ldi RTC_Byte, RTC_SECS_WRITE
        rcall RTC_WriteByteAtAddress

        ldi R22,80     //set data protection flag
        ldi RTC_Byte, RTC_CTRLREG_WRITE
        rcall RTC_WriteByteAtAddress

        pop R22
        pop ZL
        pop ZH
        pop RTC_Byte
        pop genTemp2
        pop genTemp
        out SREG, genTemp
        pop genTemp
        ret



//-----------------------------------------
//RTC clock initialization to default firmware init time
.global RTC_ResetInit
//void RTC_ResetInit(void); //declaration to be put in main c program
RTC_ResetInit: //First time (only once) RTC initialization
// this routine assumes: RTC_ResetInitTime is preset with the RTC initial clock data.
        push genTemp
        in genTemp, SREG
        //cli
        push genTemp
        push genTemp2
        push RTC_Byte
        push ZH
```

```
        push ZL
        push R22

        sbi DDRD, RTC_CS       //set CS bit on Ctrl port to output
        sbi RTC_DDRPORT, RTC_SCK  //set SCK bit on data port to output
        sbi RTC_DDRPORT, RTC_IO  //set I/O bit on Data port to output

        //rcall RTC_on;
        ldi R22, 0     //clear data protection flag
        ldi RTC_Byte, RTC_CTRLREG_WRITE
        //rcall RTC_on;
        rcall RTC_WriteByteAtAddress

        ldi R22, 0//desable tricle charger (this is for a non-rechargeable batt, ex.: a CR2032)
        ldi RTC_Byte, RTC_TRICLE_WRITE
        //rcall RTC_on;
        rcall RTC_WriteByteAtAddress

        ldi     ZH,hi8(RTC_ResetInitTime)     ; Load high byte of secs address word into ZH
        ldi     ZL,lo8(RTC_ResetInitTime)     ; Load low byte of secs address word into ZL

        //ldi R22, 0   //clear data protection flag
        //ldi RTC_Byte, RTC_CTRLREG_WRITE
        //rcall RTC_WriteByteAtAddress //pre-enabling write access is useless in burst mode as
        // crtlreg must be written in burst mode.

        //7clkdata+ 1ctrlreg: must write to ctrlreg in burst mode (with 0 to clear write prot.)
        ldi genTemp,8
        ldi RTC_Byte, RTC_CKBURST_WRITE //clock burst command
        rcall RTC_on;
        rcall WrByte;  //write clock burst command
9: //reprWrtClockBrst:
        lpm RTC_Byte, Z+
        rcall WrByte
        dec genTemp
        brne 9b //reprWrtClockBrst
        rcall RTC_off

        //clr genTemp2 //start clock
        ldi R22, 00
        ldi RTC_Byte, RTC_SECS_WRITE
        rcall RTC_WriteByteAtAddress

        ldi R22,80     //set data protection flag
        ldi RTC_Byte, RTC_CTRLREG_WRITE
        rcall RTC_WriteByteAtAddress

        pop R22
        pop ZL
        pop ZH
        pop RTC_Byte
        pop genTemp2
        pop genTemp
        out SREG, genTemp
        pop genTemp
        ret
//
//---------------------------------------
.global RTC_Disp
//void RTC_Disp(char *ptrClockData, line, toDisp); //declaration to be put in main c program
//inputs: R25:R24 with ptrClockdata scratchpad area to read clockdata to;
//inputs: R22 with line of LCD where to display the 16 char clock update info on default col 0.
//inputs: R20 with logical 1 if not to send result to LCD.
//returns: pointer to clockdisplay string
RTC_Disp: //updates the LCD display with the current RTC clock time;
        push genTemp
        push genTemp2
        push ZH
        push ZL
        push YH
        push YL
        push XH
        push XL
        //push R24  //= push RTC_Byte
        //push R25
        push R22
        push R23
```

```
        push R21
        push R20
        mov XH, R25
        mov XL, R24


        or R20, R20; cheq if 1: chek if noDisp = 1
        brne DoNotDisp
        mov R24, R22   //line into arg1 of address func call, next.
        clr R25
        clr R22 //disp on Col 0
        clr R23
        call LCDaddress        //LCDaddress(line, 0);
DoNotDisp:
        mov R24, XL
        mov R25, XH
        rcall RdClockBurst

        ldi     YH,hi8(RTC_CurrentTime)
        ldi     YL,lo8(RTC_CurrentTime)


        clr genTemp
        adiw XL,RAMday
        ld RTC_Byte, X
        sbiw XL,RAMday
        dec RTC_Byte   //because weekdays are 1 to 7; this offset read is 0 to 6.
        ldi     ZH,hi8(WeekDays)
        ldi     ZL,lo8(WeekDays)
        lsl RTC_Byte //x2: weekday abreviation of 2 chars
        add ZL, RTC_Byte
        adc ZH, genTemp
        lpm RTC_Byte, Z+ //read 1st weekday letter


        st Y+, RTC_Byte
        lpm RTC_Byte, Z+ //read 2nd weekday letter
        st Y+, RTC_Byte
        ldi RTC_Byte, ',' //place a separating comma
        st Y+, RTC_Byte

        adiw XL,RAMdate
        ld RTC_Byte, X
        sbiw XL,RAMdate
        rcall BCDAsciiStore

        adiw XL,RAMmonth
        ld RTC_Byte, X
        sbiw XL,RAMmonth
        rcall BCDAsciiStore
        adiw XL,RAMyear
        ld RTC_Byte, X
        sbiw XL,RAMyear
        rcall BCDAsciiStore
        ldi RTC_Byte, ',' //place a separating comma
        st Y+, RTC_Byte

        adiw XL,RAMhours
        ld RTC_Byte, X
        sbiw XL,RAMhours
        rcall BCDAsciiStore
        adiw XL,RAMmins
        ld RTC_Byte, X
        sbiw XL,RAMmins
        rcall BCDAsciiStore
        adiw XL,RAMRTC_secs
        ld RTC_Byte, X
        sbiw XL,RAMRTC_secs
        rcall BCDAsciiStore

        or R20, R20; check if 1: chek wether noDisp = 1
        brne DoNotDispX

        ldi genTemp, 16
        ldi YH,hi8(RTC_CurrentTime)    ; Load high byte of address word
        ldi YL,lo8(RTC_CurrentTime)    ; Load low byte of address word
10: //repDispRTC:      //display RTC current time on LCD
```

```
        ld      RTC_Byte, Y+;Load byte from RAM memory into LCDdata = RTC_Byte and post increment Y
        call LCDwriteDATA
        dec genTemp
        brne 10b //repDispRTC

DoNotDispX:
        ldi     R25,hi8(RTC_CurrentTime) //return clock dispstring
        ldi     R24,lo8(RTC_CurrentTime)
        pop R20
        pop R21
        pop R23
        pop R22
        //pop R25
        //pop R24       //= pop RTC_Byte
        pop XL
        pop XH
        pop YL
        pop YH
        pop ZL
        pop ZH
        pop genTemp2
        pop genTemp
        ret

//---------------------------------
// not globally  visible.
.global BACDAsciiStore
BCDAsciiStore:  // converts bcd to ascii and store result. It assumes input byte is BCD formated.
//Input: RTC_Byte with byte to convert to ascii and Y with the store destination addr in MCU RAM
        push genTemp
        push RTC_Byte
        swap RTC_Byte  //isolate BCD MSdigit
        andi RTC_Byte, 0x0F
        rcall BCD2Ascii
        st Y+, RTC_Byte
        pop RTC_Byte
        andi RTC_Byte, 0x0F    //isolate BCD LSdigit
        rcall BCD2Ascii
        st Y+, RTC_Byte
        pop genTemp
        ret


//-------------------------
.global BCD2Ascii
//char BCD2Ascii(char byteData); //declaration to be put in main c program
// input and output on R24 = RTC_Byte
BCD2Ascii:  // converts low nible from bcd to ascii.
//it assumes input byte is high nible null and low nible is BCD (0-9).
        push genTemp
        ldi genTemp, 0x30      //ascii base for decimal digits
        add RTC_Byte, genTemp //bcd to ascii
        clr R25 //clear R25, the MSbyte of return.
        pop genTemp
        ret


//---------------
.global Hex2Ascii
//char Hex2Ascii(char byteData); //declaration to be put in main c program
//input and output on R24
Hex2Ascii:  // converts hex to ascii. it assumes input byte is hex (0-9, A-F)
        push genTemp
        ldi genTemp, 0x30      //ascii base for decimal digits (= hex digits 0-9)
        cpi RTC_Byte, 10
        brcs 11f //AsciiAdj
        ldi genTemp, 0x37      //37h ascii base for hex digits A to F
11: //AsciiAdj:
        add RTC_Byte, genTemp //hex to ascii
        clr R25 //clear MSbyte of return.
        pop genTemp
        ret


//---------------
//converts two ascii bytes to a BCD hex byte. Assume: input byte are BCD digits in ascii(30h-39h)
.global Ascii2BCD
```

```
//char Ascii2BCD(const char msbAscii, const char lsbAscii);
//inputs R24 and R22 with ascii MS and LSBytes to be converted to a BCD hex byte.
//outputs on R24
Ascii2BCD:
        push genTemp
        ldi genTemp, 0x30      //ascii base for decimal digits
        sub R24, genTemp
        swap R24
        sub R22, genTemp
        add R24, R22
        clr R25 //clear MSbyte of return.
        pop genTemp
        ret
//-------------------------
```

```
;*********************************************************
;in flash mem
;---------------------------------------
;Real Time Clock data
//RTC default initialization clock time: see remarks on RTC_ResetInitTime routine header
RTC_ResetInitTime:
//clock registers order: seconds, minutes, hours, date, month, day, year, ctrlreg;
.byte 0x00,0x05,0x11,0x21,0x06,0x01,0x09,0x00
//RTC_ResetInitTimeEnd:
WeekDays:
.ascii "SuMoTuWeThFrSt"
.byte 0
.byte 0


.data //in RAM area

RTC_CurrentTime:
.ascii "Th,091008,014500"
.byte 0
//.global RAMRTC_clock
RAMRTC_clock:
.byte 0x00;    0 RTC active; 0 secs
.byte 0x30
.byte 0x00; 24h mode, 0h
.byte 0x09
.byte 0x10
.byte 0x04
.byte 0x08
.byte 0x00     //write protect flag on ctrl reg
RAMRTC_ClockEnd:


/*---------
```

File: 1302rtc.h (1302 RTC definitions)

```
*///-------------------------------------------------------------------------------
// DS1302 Real Time Clock defs
#define RTC_Byte R24          //is the LSByte of arg1 - for all single arg functions
#define RTC_CTRLPORT  PORTD
#define RTC_CTRLDDR  DDRD
#define RTC_DATAPORT  PORTC
#define RTC_PINPORT  PINC
#define RTC_DDRPORT  DDRC
#define RTC_CS  6 ; bit 6 of port D
#define RTC_IO  1 ; bit 1 of port C
#define RTC_SCK  0 ; bit 0 of port C
#define RTC_SECS_WRITE  0x80
#define RTC_SECS_READ  0x81   //in gereral any clock read address = the writeaddress+1
#define RTC_MINS_WRITE  0x82
#define RTC_HRS_WRITE  0x84
#define RTC_DATE_WRITE  0x86
#define RTC_MONTH_WRITE  0x88
#define RTC_DAY_WRITE  0x8A
#define RTC_YEAR_WRITE  0x8C
#define RTC_YEAR_READ  0x8D
#define RTC_CTRLREG_WRITE  0x8E
#define RTC_TRICLE_WRITE  0x90
#define RTC_CKBURST_WRITE  0xBE
#define RTC_CKBURST_READ  0xBF
#define RTC_RAMBASE_WRITE  0xC0
#define RTC_RAMUPPER_WRITE  0xFC
#define RTC_RAMBURST_WRITE  0xFE
#define RTC_RAMBURST_READ  0xFF
#define READ_ADDR  1
//---------------------------------------
#define RAMRTC_secs 0
#define RAMmins 1
#define RAMhours 2
#define RAMdate 3
#define RAMmonth 4
#define RAMday 5
#define RAMyear 6
#define RAMctrl_reg 7
```

```
/*---------
```

File: USART.h (RS232 definitions)

```
*///-------------------------------------------------------------------------
void USARTinit(void);
void USART_SendByte(uint8_t xByte);
uint8_t USART_ReceiveByte(void);
void USART_SendPage(char *ptrPageBuffer, uint8_t nBytes);
void USART_ReceivePage(char *ptrPageBuffer, uint8_t nBytes);

#define  CR 0x0D       //carriage return;
#define  LF 0x0A       //line feed;

/*---------
```

File: mixed_C_asm.h (header file)

```
*///-------------------------------------------------------------------------//mixed_C_asm.h
header file //
//
// For avr-gcc to recognize Z, Y, and X naming of pointer registers in assembly modules.
//

#define __SFR_OFFSET 0
#define ZH R31
#define ZL R30
#define YH R29
#define YL R28
#define XH R27
#define XL R26


/*---------
```

File: portConfig3_14.h (port and other configuration definitions)

```
*///-------------------------------------------------------------------------
// Overall Port Configuration header file //
//
// For SolarController3-14
//
//Author: Doho, G.J.

#define ON 1
#define OFF 0

#define TRUE 1
#define FALSE 0


//SPI Config
#define SPI_Port PORTB
#define SPI_DDR DDRB
#define MISO PORTB6
#define MOSI PORTB5
#define SCK PORTB7
#define SS_MCU1 PORTB4


//ADC - Analog to digital converter interface port configuration
#define ADC_ChannelZero_AVCCref 0x40 //01 0 00000: 01 AVCC; 0 RghtJustif result; 00000 chann 0:
PortA0
#define ADC_prescalMASK4MHZ 0x05 //fcpu/32
#define ADC_prescalMASK8MHZ 0x06 //fcpu/64
#define ADC_prescalMASK10MHZ 0x06 //fcpu/64
#define ADC_prescalMASK16MHZ 0x07 //fcpu/128
#define ADC_prescalMASK20MHZ 0x07 //fcpu/128
#define ADC_inPort PORTA
#define ADC_inBit PORTA0
#define ADC_xmuxAddrClockPort PORTC
```

```
#define ADC_xmuxAddrClockBit PORTC7
#define ADC_xmuxAddrWordPort PORTC
#define ADC_xmuxAddr_SouthAlignPHDiode 0x0
#define ADC_xmuxAddr_NorthAlignPHDiode 0x1
#define ADC_xmuxAddr_DeclinationAngle 0x2
#define ADC_xmuxAddr_HourAngle 0x3
#define ADC_xmuxAddr_XaxisACCM 0x4
#define ADC_xmuxAddr_YaxisACCM 0x5
#define ADC_xmuxAddr_ZaxisACCM 0x6
#define ADC_xmuxAddr_MainBoardTemp 0x7
#define ADC_xmuxAddr_X2axisACCM 0x8
#define ADC_xmuxAddr_Y2axisACCM 0x9
#define ADC_xmuxAddr_Z2axisACCM 0xA

//#define ADC_xmuxAddr_EastAlignPHDiode 0xB
//#define ADC_xmuxAddr_WestAlignPHDiode 0xC
//#define ADC_xmuxAddr_WindSpeed_Load 0xD
//#define ADC_xmuxAddr_WindDir 0xE
//#define ADC_xmuxAddr_ADCbrd_or_UserSetPointTemp 0xF

#define ADC_xmuxAddr_PyrHeliometer_Vi 0xB
#define ADC_xmuxAddr_Pyranometer_Vi 0xC
#define ADC_xmuxAddr_Trackers_Vout 0xD
#define ADC_xmuxAddr_Trackers_Iout 0xE
#define ADC_xmuxAddr_NTCambientTemp_Vi 0xF

//TDC - Thermocouple to digital converter interface port configuration
#define TDC_MISO MISO
#define TDC_MOSI MOSI //actually: no use of MOSI for MAX6675 TDC
#define TDC_SCK SCK
#define TDC_SS PORTB4 //
#define TDC_DataPort PORTB
#define TDC_DdrPort DDRB

#define TDC_xmuxAddrClockPort PORTC
#define TDC_xmuxAddrClockBit PORTC6
#define TDC_xmuxAddrWordPort PORTC

#define TDC_xmuxAddr_TRcvrIn 0x00     //receiver's inlet temp
#define TDC_xmuxAddr_TRcvrSurfIn 0x01 //receiver surfacial temp at the inlet end.
#define TDC_xmuxAddr_TRcvrOut 0x02    //receiver's outlet temp
#define TDC_xmuxAddr_TRcvrSurfOut 0x03       //receiver surfacial temp at the outlet end
#define TDC_xmuxAddr_TESin 0x04       //TES inlet temp
#define TDC_xmuxAddr_TESout 0x05      //TES outlet temp

#define TDC_xmuxAddr_TES_LevA1 0x06   //TES profile temps (level A radial pos 1)
#define TDC_xmuxAddr_TES_LevB1 0x07   //TES profile temps (level B radial pos 1)
#define TDC_xmuxAddr_TES_LevC1 0x08   //TES profile temps (level C radial pos 1)
#define TDC_xmuxAddr_TES_LevD1 0x09   //TES profile temps (level D radial pos 1)
#define TDC_xmuxAddr_TES_LevE1 0x0A   // ....
#define TDC_xmuxAddr_TES_LevF1 0x0B
#define TDC_xmuxAddr_TES_LevG1 0x0C
#define TDC_xmuxAddr_TES_LevH1 0x0D
#define TDC_xmuxAddr_TES_LevI1 0x0E  //TES profile temps (level I radial pos 1)
#define TDC_xmuxAddr_PlantAmbientTemp 0x0F


//Motion Control interface port configuration

#define MOT_CtrlLatchClockPort PORTA
#define MOT_CtrlLatchClockBit PORTA1
#define MOT_CtrlLatchPort PORTC

//#define MOT_xxxxxx   PORTB3 //OCR0          //no use for this config
#define MOT_PWM_DECLIN_OR_HOUR_SPEEDCTRL PORTD4  //OCR1B
//#define MOT_xxxxxx   PORTC5  //no use for this config
//#define MOT_xxxxxx   PORTC4  //no use for this config
#define MOT_DECLIN_OR_HOUR_REVERSE_MOV PORTC3
#define MOT_DECLIN_OR_HOUR_FORWARD_MOV PORTC2
#define MOT_DECLIN_OR_HOUR_MOTOR_SELECT PORTC1
//#define MOT_AllMot_ONOFF PORTC0            //no use for this config

//Mot control/Pumps

#define MOT_Pump_PWMPort PORTD
#define MOT_Pump_PWMDDR DDRD
#define MOT_Pump_FeedbackPort PORTB
```

```
#define MOT_Pump_FeedbackDDR DDRB
#define MOT_ChgPump_PWMBit PORTD7     //OC2
#define MOT_DisChgPump_PWMBit PORTD5  //OC1A
#define MOT_ChgPump_FeedbackBit PORTB0
#define MOT_DisChgPump_FeedbackBit PORTB1


//SD memory card interface config: via SPI interface with SS on PB2
#define SD_MISO MISO
#define SD_MOSI MOSI //actually: no use of MOSI for MAX6675TDC
#define SD_SCK SCK
#define SD_SS PORTB2

//Keyboard interface
#define KEYB_OEN  PORTD2 //PD2
#define KEYCODE_AVAILABLE  PORTD3  //PD3
#define KEYB_DDRPORT  DDRA
#define KEYB_DATAPORT  PORTA
#define KEYB_DATAPIN PINA
#define KEYB_CTRLPORT  PORTD
#define KEYB_CTRLPORTPIN  PIND

//MC74HC923 keyb-encoder with a 3col x 4rows keypad
#define SCAN_zero 0x0D
#define SCAN_one 0x02
#define SCAN_two 0x01
#define SCAN_three 0x00
#define SCAN_four 0x06
#define SCAN_five 0x05
#define SCAN_six 0x04
#define SCAN_seven 0x0A
#define SCAN_eight 0x09
#define SCAN_nine 0x08
#define SCAN_star 0x0E
#define SCAN_hash 0x0C

//ASCII codess for the kaypad
#define ASCII_zero 0x30
#define ASCII_one 0x31
#define ASCII_two 0x32
#define ASCII_three 0x33
#define ASCII_four 0x34
#define ASCII_five 0x35
#define ASCII_six 0x36
#define ASCII_seven 0x37
#define ASCII_eight 0x38
#define ASCII_nine 0x39
#define ASCII_star '*'
#define ASCII_hash '#'

/*---------
```

File: pumps.h (pumps interface definitions)

```
*///-------------------------------------------------------------------------// header file for
Charging and discharging pumps//
#define CHGPUMP_CTRLMODE_PID_ANGULARSPEED 0 //in revol/second (1 revol/second = 2.pi rads/second).
//#define DISCHGPUMP_CTRLMODE_PID_ANGULARSPEED 0
#define  CHGPUMP_MODE_IDLE    0 //stop pump motor, go to (stay on) idle state;
#define  CHGPUMP_MODE_AUTO    1 //system controlled movement;
#define  CHGPUMP_MODE_MANUAL  2 //keep idle and allow manual control of the pump.

#define  PUMP_STATE_IDLE      0
#define  PUMP_STATE_ROTATING_CW
#define  PUMP_STATE_ROATING_CCW 2 //yields the forward fluid movement

//for pumps state reset behavior
#define  FLASH_DEFAULTS 0     //reset with original program (non tuned) defaults;
#define  EEPROM_DEFAULTS 1    //reset with learned or tuned eeprom saved defaults;


/*---------
```

File: Tracker.h

```
*///-------------------------------------------------------------------------
//Tracker.h header file //
//solar tracker control defs
//
//Author: Doho, G.J.

//#ifndef _TRACKER_DEFS_
//#define  _TRACKER_DEFS_

#define HOUR_AXIS 0
#define DECL_AXIS 1

#define  TRACK_STATE_IDLE     0
#define  TRACK_STATE_MOVING_EASTWARD 1 //earthwise rotation movement (actual hour angle decreasing).
#define  TRACK_STATE_MOVING_WESTWARD 2 //anti-earthwise rotation movement (hour angle increasing).
#define  TRACK_STATE_MOVING_SOUTHWARD 5 //(actual declination angle decreasing)
#define  TRACK_STATE_MOVING_NORTHWARD 6 //(actual declination angle increasing)

#define  TRACK_MODE_IDLE      0       //stop all tracking motors, go to (stay on) idle state;
#define  TRACK_MODE_AUTOTRACK 1 //system controlled tracking movement;
#define  TRACK_MODE_SAFETY 2 //go to (stay on) safety position;
#define  TRACK_MODE_MANUAL 3 //keep idle and allow manual control of the tracking machine.
//#define  TRACK_MODE_PARK 4 //go to or keep on parking position; keep while trackermode = park;


//for tracker state reset behavior
#define  FLASH_DEFAULTS 0     //reset with original program (non tuned) defaults;
#define  EEPROM_DEFAULTS 1    //reset with learned or tuned eeprom saved defaults;


//define Tracker control modes.
#define  TRACKER_CTRLMODE_P_ANGULARPOSITION          0
#define  TRACKER_CTRLMODE_PIDANGULARPOSITION 1
#define  TRACKER_CTRLMODE_PIDANGULARSPEED            2

/*---------
```

File: Task_control.h

```
*///-------------------------------------------------------------------------------//Task_control.h
//For real time task scheduling / coordination related defs

//Semaphores;

//Semaphore IDs
#define SEMAF_SYS_RESERVED 0//sytem reserved
#define SEMAF_PORTA 1 //embodies LB1
#define SEMAF_PORTB 2 //
#define SEMAF_PORTC 3 // embodies LB2
#define SEMAF_PORTD 4 //
#define SEMAF_ADC 5 //Analog to Digital converter
#define SEMAF_RTC 6 //
#define SEMAF_SPIBUS 7 //
#define SEMAF_SDCARD 8 //
#define SEMAF_RS232 9 //
#define SEMAF_LCD 10 //



//Semaphore users

//For SEMAF_PORTA //
#define PORTAUSER_BUSYFLAG 0  //while=1=> someone is using the shared resource
#define PORTAUSER_RTC_CLKIO 1
#define PORTAUSER_ADC_DATA 2
#define PORTAUSER_TDC_SS 3
#define PORTAUSER_TRACK_CLK 4
#define PORTAUSER_LCD_CTRL 5
#define PORTAUSER_M2M_INT 6

//For SEMAF_PORTB 2 //
#define PORTBUSER_BUSYFLAG 0  //while=1=> someone is using the shared resource
```

```
#define PORTBUSER_TDC_SPIBUS 1
#define PORTBUSER_TRACK_CLK 2
#define PORTBUSER_CHGPP_CTRL 3
#define PORTBUSER_DISCHGPP_CTRL 4
#define PORTBUSER_SD_SPIBUS 5
#define PORTBUSER_M2M_SPIBUS 6

//For SEMAF_PORTC 3 //
#define PORTCUSER_BUSYFLAG 0  //while=1=> someone is using the shared resource
#define PORTCUSER_RTC_CLKIO 1
#define PORTCUSER_TDCADDRESS 2
#define PORTCUSER_TRACKADDRESS 3
#define PORTCUSER_ADCADDRESS 4
#define PORTCUSER_ADCINT 5
#define PORTCUSER_LCDDATA 6
#define PORTCUSER_KEYBDATA 7

//For SEMAF_PORTD 4 //
#define PORTDUSER_BUSYFLAG 0  //while=1=> someone is using the shared resource
#define PORTDUSER_RTC_CTRL 1  //on Ver.1.6.4
#define PORTDUSER_RS232_IO 2
#define PORTDUSER_KEYB_CTRL 3
#define PORTDUSER_TRACK_CTRL 4
#define PORTDUSER_CHGPP_CTRL 5
#define PORTDUSER_DISCHGPP_CTRL 6


//For SEMAF_ADC 5 //Analog to Digital converter
#define ADCUSER_BUSYFLAG 0    //while=1=> someone is using the shared resource
#define ADCUSER_1SAMPLE 1
#define ADCUSER_MULTISAMPLES 2


//For SEMAF_RTC 6 //


//For SEMAF_SPIBUS 7 //
#define SPIUSER_BUSYFLAG 0    //while=1=> someone is using the shared resource
#define SPIUSER_TDC 1
#define SPIUSER_SD 2
#define SPIUSER_M2M 3
#define SPIUSER_OPT 4

//For SEMAF_SDCARD 8 //

//For SEMAF_RS232 9 //

//For SEMAF_LCD 10 //


// system malfunction status flags and IDs (SysStatUserIDs)
// these are bit number in the 8 bits flag word SysStatUserIDs; ...
// ... increase wordlength to add more userIDs.
//
//      these can help finding the causes of system timing malfunctions.
#define SYS_STAT_TIMERTIC 0
#define SYS_STAT_ADCREADTRIGGER 1
#define SYS_STAT_ADCINT 2
#define SYS_STAT_TDCTRIGGER 3
#define SYS_STAT_TDCREADTRIGGER 4
#define SYS_STAT_TDCINT 5
#define SYS_STAT_INITLOG 6
#define SYS_STAT_DATALOG 7
```

Appendix B - Listing of the PC side Data logging program: DatalogWinLink

File: DataLogWinLink.frm

```
' SolarTech Datalogging Windows Link main form (frmDataLogWinLink.frm)
' Author: Doho, G.J., 2007/2009 (UKZN StudentNo: 207514966)
Option Explicit
 Public gotHeader As Boolean, nFilebreakDetected As Integer
 Public CrLf As String, gotBeginning As Boolean
 Public lastCOM As Integer, lastBaud As Long, lastInputLen As Long
 Public lastAutoStart As Boolean, lastIconize As Boolean
 Public lastAutoSaveFileName As String, lastAutoSavePath As String
 Public lastSaveFileName As String, isNIDdeleted As Boolean

Public Sub Form_Load()
  Dim i As Integer
  CrLf = Chr$(13) + Chr$(10)
  rtxtInputData.Text = ""
  rtxtInputBuffer.Text = ""
  txtFilename.Text = "ErosNeonel_" + zTrim(Str(Day(Date$))) + zTrim(Str(Month(Date))) +
zTrim(Str(Year(Date)))
  gotHeader = False
  DataBakSchedule.Enabled = (chkAutosave.Value = vbChecked)
  DataBakSchedule.Interval = Min(Val((txtAutosaveInterval.Text)) * 1000, 65535)
  gotBeginning = False
  isNIDdeleted = True

  On Error GoTo chkNextPortNo 'check available com ports
  For i = 1 To 256
    MSComm1.CommPort = i
    MSComm1.Settings = "9600,N,8,1"
    MSComm1.InputLen = "100"
    MSComm1.RThreshold = 5
    MSComm1.PortOpen = True
    MSComm1.PortOpen = False
    cmbCOMport.AddItem i
    GoTo NextI
chkNextPortNo:
    If Err.Number <> 8002 And Err.Number <> 8005 Then
      Exit For
    Else
      Resume NextI
    End If
NextI:
  Next i

 'Retrieve last session settings from system registry.
 lastCOM = GetSetting(App.EXEName, Me.Caption, "lastCOM", cmbCOMport.List(0))
  lastBaud = GetSetting(App.EXEName, Me.Caption, "lastBaud", "9600")
  lastInputLen = GetSetting(App.EXEName, Me.Caption, "lastInputLen", "100")
  lastAutoStart = GetSetting(App.EXEName, Me.Caption, "lastAutoStart", "-1") '-1=True
  lastIconize = GetSetting(App.EXEName, Me.Caption, "lastIconize", "-1") '-1=True
  lastAutoSaveFileName = GetSetting(App.EXEName, Me.Caption, "lastAutoSaveFileName", _
       "___DataLogWinLink_")
  lastAutoSavePath = GetSetting(App.EXEName, Me.Caption, "lastAutoSavePath", App.Path + "\")
  lastSaveFileName = GetSetting(App.EXEName, Me.Caption, "lastSaveFileName", txtFilename.Text)
  Me.cmbCOMport.Text = lastCOM
  Me.txtInputLen.Text = lastInputLen
  Me.txtBauds.Text = lastBaud
  Me.txtAutosaveFilename = lastAutoSaveFileName
  Me.txtAutosaveFilePath = lastAutoSavePath
  Me.txtFilename.Text = lastSaveFileName
  Me.mnuMinimizeToTray.Checked = lastIconize
  Me.mnuAutostart_AutoReceive.Checked = lastAutoStart
  InputSchedule.Enabled = lastAutoStart
  Me.chkAutoReceive.Value = IIf(lastAutoStart, vbChecked, vbUnchecked)
  InputSchedule.Interval = Min(Val(txtAutoReceiveInterval.Text) * 1000, 65535)

  If lastAutoStart And cmbCOMport.ListCount > 0 Then
    chkAutoReceive_Click
  Else
    Me.chkAutoReceive.Value = vbUnchecked
```

```
  End If

  If MSComm1.PortOpen Then
    cmdCloseCOM.Enabled = True
    cmdOpenCOM.Enabled = False
    cmdSend.Enabled = True
    cmdReceive.Enabled = True
    cmdStop.Enabled = True
  Else
    cmdOpenCOM.Enabled = True
    cmdCloseCOM.Enabled = False
    cmdSend.Enabled = False
    cmdReceive.Enabled = False
    cmdStop.Enabled = False
  End If
  Me.Top = 500
  Me.Left = 1000

  'system taskbar notification area (tray icon) interface begin
  If lastIconize Then
    Me.WindowState = vbMinimized
  End If
  With nid
   .cbSize = Len(nid)
   .hwnd = Me.hwnd
   .uId = vbNull
   .uFlags = NIF_ICON Or NIF_TIP Or NIF_MESSAGE
   .uCallBackMessage = WM_MOUSEMOVE
   .hIcon = Me.Icon
   .szTip = Me.Caption & vbNullChar
  End With
  ' Shell_NotifyIcon NIM_ADD, nid
  'system tray icon interface end
  Me.Show
  Me.Refresh

End Sub

Private Sub Form_Resize()
  RefreshWindowState
End Sub
Private Sub RefreshWindowState()

    'prepare notification icon
    If (Me.chkAutoReceive.Value = vbChecked) Then
      nid.hIcon = Me.icoSolShine.Picture
      nid.szTip = Me.Caption & ": AutoReceiving is ON" & vbNullChar
    Else
      nid.hIcon = Me.Icon
      nid.szTip = Me.Caption & ": AutoReceiving is OFF" & vbNullChar
    End If

    If Not Me.mnuMinimizeToTray.Checked Then
      Shell_NotifyIcon NIM_DELETE, nid
      isNIDdeleted = True
    ElseIf (Me.WindowState = vbMinimized) Or Not isNIDdeleted Then
      'Hide the minimized window; keep it out from the taskbar
      If isNIDdeleted Then
        Shell_NotifyIcon NIM_ADD, nid
        isNIDdeleted = False
      Else
        Shell_NotifyIcon NIM_MODIFY, nid
      End If
      If (Me.WindowState = vbMinimized) Then Me.Hide
    End If
End Sub
'System Tray icon callbacks handler.
'quote: this procedure adapted from VB's msdn code sample - begin
Private Sub Form_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
Dim Result As Long, msg As Long
 'do not process when window is displayed
 If Me.WindowState = vbNormal Then Exit Sub

 If (Me.ScaleMode = vbPixels) Then
    msg = X
 Else
    msg = X / Screen.TwipsPerPixelX
```

```
 End If
 Select Case msg
   Case WM_LBUTTONDBLCLK    'left dbl-click: show form
      Me.WindowState = vbNormal
      Result = SetForegroundWindow(Me.hwnd)
      Me.Show
   Case WM_LBUTTONUP 'left-click: show settings context menu
      Result = SetForegroundWindow(Me.hwnd)
      Me.PopupMenu Me.mnuSettings
   Case WM_RBUTTONUP 'right-click: show restore/exit context menu
      Result = SetForegroundWindow(Me.hwnd)
      Me.PopupMenu Me.mnuSysTrayMenu
  End Select
End Sub
'quote: this procedure (above) adapted from VB's msdn code sample - end

Private Sub mnuTrayExit_Click()
 'user clicks on the tray icon's context menu Exit command
 Unload Me
End Sub
Private Sub mnuAbout_Click()
  'user clicked about menu option
  frmAbout.Show
End Sub

Private Sub mnuAutoStart_AutoReceive_Click()
  ' user selected to toggle the menu option: "On "
  mnuAutostart_AutoReceive.Checked = Not mnuAutostart_AutoReceive.Checked
End Sub
Private Sub mnuStartAutoRcvNow_Click()
  ' user selected to Start AutoReceive now
  If Me.chkAutoReceive.Value = vbUnchecked Then
    Me.chkAutoReceive.Value = vbChecked 'this triggers chkAutoReceive_Click
  End If
End Sub


Private Sub mnuMinimizeToTray_Click()
  'user selected to toggle the menu option: "On minimizing, Minimize to nTray"
  mnuMinimizeToTray.Checked = Not mnuMinimizeToTray.Checked
End Sub
Private Sub mnuTrayRestore_Click()
  'user clicks on the tray icon's context menu Restore command
  Dim Result As Long
  Me.WindowState = vbNormal
  Result = SetForegroundWindow(Me.hwnd)
  Me.Show
End Sub
Private Sub Form_Unload(Cancel As Integer)
  If MSComm1.PortOpen = True Then
    MSComm1.PortOpen = False
  End If
  SaveAllOnExit
  Shell_NotifyIcon NIM_DELETE, nid  'delete the notification tray icon
  End
End Sub

Private Sub chkAutoAddCSVHeader_Click()
  If chkAutoFindCSVheader And Not gotHeader Then
    txtCSVHeader.Locked = True
    AutoFindCSVHeader
  End If
End Sub

Private Sub chkAutoFindCSVheader_Click()
  If chkAutoFindCSVheader Then
    txtCSVHeader.Locked = True
    AutoFindCSVHeader
  Else
    txtCSVHeader.Locked = False
  End If
End Sub
Private Sub AutoFindCSVHeader()
  Dim CSVheader As String, eolPos As Long, xCrLf As String, xLen As Integer
  xCrLf = CrLf
  xLen = 2
  If chkOnlyOnPageBreak Then
```

```
    xCrLf = xCrLf + CrLf
    xLen = 4
  End If
  If Not gotHeader And chkAutoFindCSVheader Then
    eolPos = FindAtLeft(rtxtInputBuffer.Text, xCrLf)
    If eolPos > 0 Then
      CSVheader = Mid(rtxtInputBuffer.Text, 1, eolPos + 1)
      txtCSVHeader.Text = CSVheader
      gotHeader = True
      If chkStripCSVheader Then
        rtxtInputData.Text = Mid(rtxtInputData.Text, eolPos + xLen)
        chkAutoAddCSVHeader.Value = vbChecked
      Else
        chkAutoAddCSVHeader.Value = vbUnchecked
      End If
    End If
  End If
End Sub

Private Sub chkAutoReceive_Click()
  If chkAutoReceive.Value = vbChecked Then
    If Not MSComm1.PortOpen Then
      cmdOpenCOM_Click
      'chkAutoReceive.Value = vbChecked
    End If
    If MSComm1.PortOpen Then
      InputSchedule.Enabled = True
      cmdSend.Enabled = False
      cmdReceive.Enabled = False
      cmdCSVHDclear.Enabled = False
      cmdMerge.Enabled = False
      rtxtInputBuffer.Enabled = False
      mnuStartAutoRcvNow.Enabled = False
    Else
      chkAutoReceive.Value = vbUnchecked
    End If
  End If
  If (chkAutoReceive.Value = vbUnchecked) Then
    InputSchedule.Enabled = False
    cmdSend.Enabled = True
    cmdReceive.Enabled = True
    cmdCSVHDclear.Enabled = True
    cmdMerge.Enabled = True
    rtxtInputBuffer.Enabled = True
    mnuStartAutoRcvNow.Enabled = True
  End If
  RefreshWindowState
End Sub

Private Sub chkAutosave_Click()
    DataBakSchedule.Interval = Min(Val(txtAutosaveInterval.Text) * 1000, 65535)
    DataBakSchedule.Enabled = (chkAutosave.Value = vbChecked)
End Sub


Private Sub chkAutoSaveFilebreak_Click()
  chkAutoSaveFilebreakOnTextSize.Value = chkAutoSaveFilebreak.Value
  chkAutoSaveFilebreakOnIncommingFileBresk.Value = chkAutoSaveFilebreak.Value
  If chkAutoSaveFilebreak Then
    chkAutoAddCSVHeader.Value = gotHeader And chkStripCSVheader
  End If
End Sub

Private Sub chkBeginUpon1stFileBreak_Click()
  chkBeginUpon1stLineBreak.Value = IIf(chkBeginUpon1stFileBreak.Value, vbUnchecked,
chkBeginUpon1stLineBreak.Value)
End Sub
Private Sub chkBeginUpon1stLineBreak_Click()
  If chkBeginUpon1stLineBreak.Value Then
      chkBeginUpon1stFileBreak.Value = vbUnchecked
  End If
End Sub

Private Sub chkStripCSVheader_Click()
  chkAutoAddCSVHeader.Value = gotHeader And chkStripCSVheader
  If chkAutoFindCSVheader And (Len(Trim(txtCSVHeader.Text)) > 0) And _
    (Mid(rtxtInputData.Text, 1, Len(txtCSVHeader.Text)) = txtCSVHeader.Text) Then
```

```
      rtxtInputData.Text = Mid(rtxtInputData.Text, Len(txtCSVHeader.Text) + 1)
      gotHeader = True
  End If
End Sub


Private Sub cmdClear_Click()
    If Len(Trim(rtxtInputData.Text)) > 0 Then
      rtxtInputData.Text = ""
    Else
      rtxtInputBuffer.Text = ""
    End If
End Sub

Private Sub cmdCSVHDclear_Click()
  txtCSVHeader.Text = ""
End Sub

Private Sub cmdExit_Click()
  If MSComm1.PortOpen = True Then MSComm1.PortOpen = False
  SaveAllOnExit
  End
End Sub

Private Sub cmdGetCSVheader_Click()
  Dim CSVheader As String, eolPos As Long
  eolPos = FindAtLeft(rtxtInputData.Text, CrLf)
  If ((Len(Trim(txtCSVHeader.Text)) > 0) And (Mid(rtxtInputData.Text, 1, Len(txtCSVHeader.Text)) =
txtCSVHeader.Text)) Then
    MsgBox ("CSV file header already got!")
  ElseIf Len(Trim(txtCSVHeader.Text)) = 0 And Len(Trim((rtxtInputData.Text))) = 0 Then
    MsgBox ("Cannot get header: input buffer empty!")
  ElseIf Len(Trim((rtxtInputData.Text))) = 0 And Len(Trim(txtCSVHeader.Text)) > 2 And gotHeader Then
    MsgBox ("Input buffer is empty; though, we already got a header!")
  ElseIf eolPos > 1 Then
    CSVheader = Mid(rtxtInputData.Text, 1, eolPos + 1)
    txtCSVHeader.Text = CSVheader
    gotHeader = True
    If chkStripCSVheader Then
      rtxtInputData.Text = Mid(rtxtInputData.Text, eolPos + 2)
      chkAutoAddCSVHeader.Value = vbChecked
    Else
      chkAutoAddCSVHeader.Value = vbUnchecked
    End If
  Else
    MsgBox "Could not find a CSV header. You may copy/paste it or write it down manualy."
  End If
End Sub

Private Sub cmdReceive_Click()
  cmdSend.Enabled = False
  COMreceive
  'cmdSend.Enabled = True
End Sub

Private Sub cmdSend_Click()
    InputSchedule.Enabled = False
    chkAutoReceive.Value = vbUnchecked
    cmdReceive.Enabled = False
    If Len(rtxtInputData.Text) > 0 Then
      MSComm1.Output = rtxtInputData.Text
    End If
    'cmdReceive.Enabled = True
End Sub

Private Sub cmdStop_Click()
  chkAutoReceive.Value = False
  If MSComm1.PortOpen Then
    MSComm1.Output = ""
  End If
  cmdSend.Enabled = True
  cmdReceive.Enabled = True
End Sub

Private Sub cmdCloseCOM_Click()
  If MSComm1.PortOpen = True Then
    MSComm1.PortOpen = False
```

```
    lblStatusInfo.Caption = "COM Port " + Str(MSComm1.CommPort) + " now closed!"
  Else
    lblStatusInfo.Caption = "COM Port " + Str(MSComm1.CommPort) + ", was already closed!"
  End If
  If MSComm1.PortOpen Then
    cmdCloseCOM.Enabled = True
    cmdOpenCOM.Enabled = False
  Else
    cmdOpenCOM.Enabled = True
    cmdCloseCOM.Enabled = False
    cmdSend.Enabled = False
    cmdReceive.Enabled = False
    chkAutoReceive.Value = vbUnchecked
    cmdStop.Enabled = False
    gotBeginning = False
  End If
  shpCOMmonitor.BackColor = IIf(MSComm1.PortOpen, &HFF00&, &HFF&)
End Sub

Private Sub cmdMerge_Click()
Dim pos As Long
If Len(Trim(txtCSVHeader.Text)) = 0 Or Trim(txtCSVHeader.Text) = CrLf Then
  MsgBox ("No CSV file header to merge with!")
ElseIf Mid(rtxtInputData.Text, 1, Len(txtCSVHeader.Text)) = txtCSVHeader.Text Then
  MsgBox ("CSV file header already merged!")
ElseIf gotHeader Then
    rtxtInputData.Text = txtCSVHeader.Text + rtxtInputData.Text
    rtxtInputData.Refresh
Else
  MsgBox ("No (or malformated) CSV file header defined!")
End If
End Sub


Private Sub cmdOpenCOM_Click()
  On Error GoTo COM_error
  lblStatusInfo.Caption = "Opening COM Port!"
  If MSComm1.PortOpen = False Then
    MSComm1.CommPort = cmbCOMport.Text
    MSComm1.Settings = IIf(txtBauds.Text = "", "38400", txtBauds.Text) + ",N,8,1"
    MSComm1.InputLen = IIf(txtInputLen.Text = "", 1000, txtInputLen.Text)
    MSComm1.RThreshold = 5
    MSComm1.PortOpen = True
    MSComm1.DTREnable = False
    MSComm1.RTSEnable = False
    MSComm1.Handshaking = comNone
    gotBeginning = False
    lblStatusInfo.Caption = "COM Port " + Str(MSComm1.CommPort) + " now open!"
  Else
    lblStatusInfo.Caption = "COM Port " + Str(MSComm1.CommPort) + ", was already open!"
  End If

AfterCOM_error:
  If MSComm1.PortOpen Then
    cmdCloseCOM.Enabled = True
    cmdOpenCOM.Enabled = False
    cmdSend.Enabled = True
    cmdReceive.Enabled = True
    cmdStop.Enabled = True
    lastCOM = MSComm1.CommPort
    lastBaud = IIf(txtBauds.Text = "", IIf(lastBaud = 0, "38400", lastBaud), txtBauds.Text)
    lastInputLen = IIf(txtInputLen.Text = "", IIf(lastInputLen = 0, "1000", lastInputLen),
txtInputLen.Text)
  Else
    cmdStop.Enabled = False
    cmdOpenCOM.Enabled = True
    cmdCloseCOM.Enabled = False
    cmdSend.Enabled = False
    cmdReceive.Enabled = False
  End If
  shpCOMmonitor.BackColor = IIf(MSComm1.PortOpen, &HFF00&, &HFF&)
  Exit Sub
COM_error:
  lblStatusInfo.Caption = "Error Opening COM Port: " & Err.Description
  chkAutoReceive.Value = vbUnchecked
  Resume AfterCOM_error
End Sub
```

```
Private Sub cmdBrowse_Click()
  On Error GoTo CancelSelected
    If Len(txtAutosaveFilePath.Text + txtAutosaveFilename.Text) > 0 Then _
dlgFileSave.FileName = txtAutosaveFilePath.Text + txtAutosaveFilename.Text
    dlgFileSave.DialogTitle = "Specify autosave file path..."
    dlgFileSave.FilterIndex = 1
    dlgFileSave.Filter = ""
    dlgFileSave.Flags = cdlOFNHideReadOnly + cdlOFNOverwritePrompt + cdlOFNPathMustExist
    dlgFileSave.CancelError = True
    dlgFileSave.ShowOpen
    txtAutosaveFilePath.Text = Mid(dlgFileSave.FileName, 1, FindAtRight(dlgFileSave.FileName, "\"))
CancelSelected:
End Sub



Private Sub cmdSave_Click()
  On Error GoTo CancelPressed
    If Len(txtFilename.Text) > 0 Then dlgFileSave.FileName = txtFilename.Text
    dlgFileSave.DialogTitle = "Save to CSV file ..."
    dlgFileSave.FilterIndex = 1
    dlgFileSave.Filter = _
        "Comma separated values files (*.CSV)|*.csv|txt files (*.txt)|*.txt|All files (*.*)|*.*"
    dlgFileSave.Flags = cdlOFNHideReadOnly + cdlOFNOverwritePrompt + cdlOFNPathMustExist
    dlgFileSave.CancelError = True
    dlgFileSave.ShowSave
    txtFilename.Text = dlgFileSave.FileName
    DataBakSchedule.Enabled = False
    rtxtScratchPad.Text = rtxtInputData.Text
    If gotHeader And Mid(rtxtScratchPad.Text, 1, Len(txtCSVHeader.Text)) _
              <> txtCSVHeader.Text Then
        rtxtScratchPad.Text = txtCSVHeader.Text + rtxtScratchPad.Text
    End If
    rtxtScratchPad.SaveFile txtFilename.Text, rtfText
    lblStatusInfo.Caption = "File: " + txtFilename.Text + " saved!"
    If chkClearAfterSave Then
      rtxtInputData.Text = ""
    End If
    DataBakSchedule.Enabled = True
CancelPressed:
    Exit Sub
End Sub

Private Sub DataBakSchedule_Timer()
  AutoSaveInputData
End Sub

Private Sub AutoSaveInputData(Optional ByVal InSaveIndex As Variant, Optional inSaveExt As Variant)
  On Error GoTo ErrHandler
  Static saveIndex As Integer, sFilename As String
  Dim splitIt As Boolean, sExt As String, dExt As String, eolPos As Long
  dExt = "_ed"
  If Not IsMissing(InSaveIndex) Then
    saveIndex = InSaveIndex
    splitIt = IIf(saveIndex > 0, True, False)
  End If
  If Not IsMissing(inSaveExt) Then
    dExt = inSaveExt
  End If
  If (saveIndex < 0) Or (saveIndex > 2) Then saveIndex = 0
  If (chkAutoSaveFilebreak And chkAutoSaveFilebreakOnTextSize And _
      Len(rtxtInputData.Text) > Val((txtAutosaveTextSize.Text)) * 1024) Then
      splitIt = True
      saveIndex = 1
  End If
  nFilebreakDetected = FindAtLeft(rtxtInputBuffer.Text, CrLf + CrLf)
  If (chkAutoSaveFilebreak And nFilebreakDetected > 0) Then
    splitIt = True
    saveIndex = 1
  End If
  If Len(Trim(sFilename)) = 0 Then
    sFilename = txtAutosaveFilename.Text + zTrim(Str(Day(Date$))) + zTrim(Str(Month(Date))) + _
    zTrim(Str(Year(Date))) + zTrim(Str(Hour(Time$))) + zTrim(Str(Minute(Time$))) +
zTrim(Str(Second(Time)))
  End If
```

```
  If nFilebreakDetected > 0 Then
    rtxtScratchPad.Text = Mid(rtxtInputData.Text, 1, nFilebreakDetected - 1)
    rtxtInputData.Text = Mid(rtxtInputData.Text, nFilebreakDetected + 4)
    If chkOnlyOnPageBreak And chkAutoFindCSVheader Then
      eolPos = FindAtLeft(rtxtInputData.Text, CrLf)
      If Len(Trim((rtxtInputData.Text))) = 0 Then
        gotHeader = False
      ElseIf ((Len(Trim(txtCSVHeader.Text)) > 0) And (Mid(rtxtInputData.Text, 1, eolPos) =
txtCSVHeader.Text)) Then
        gotHeader = True
      ElseIf eolPos > 1 Then
        txtCSVHeader.Text = Mid(rtxtInputData.Text, 1, eolPos + 1)
        gotHeader = True
        If chkStripCSVheader Then
          rtxtInputData.Text = Mid(rtxtInputData.Text, eolPos + 2)
        End If
        'chkAutoAddCSVHeader.Value = vbChecked
      End If
    End If

  Else

    rtxtScratchPad.Text = rtxtInputData.Text
    If splitIt Then rtxtInputData.Text = ""
    If chkAutoAddCSVHeader And gotHeader And Mid(rtxtScratchPad.Text, 1, Len(txtCSVHeader.Text)) <>
txtCSVHeader.Text Then
      rtxtScratchPad.Text = txtCSVHeader.Text + rtxtScratchPad.Text
    End If

  End If
  sExt = IIf(Len(Trim(txtExt.Text)) = 0 Or Trim(sExt) = "", "CSV", sExt)
Save2New:
  rtxtScratchPad.SaveFile txtAutosaveFilePath.Text + sFilename + "__" + Trim(Str(saveIndex)) + "." +
IIf(splitIt, sExt, dExt), rtfText
  rtxtScratchPad.Text = ""
  If saveIndex >= 1 Then
    sFilename = txtAutosaveFilename.Text + zTrim(Str(Day(Date$))) + zTrim(Str(Month(Date))) +
zTrim(Str(Year(Date))) + zTrim(Str(Hour(Time$))) + zTrim(Str(Minute(Time$))) +
zTrim(Str(Second(Time)))
    saveIndex = 0
    nFilebreakDetected = 0
  End If
  Exit Sub
ErrHandler:
  If Err.Number = 75 Then ' cannot access (already open by another process or invalid).
    sFilename = txtAutosaveFilename.Text + zTrim(Str(Day(Date$))) + zTrim(Str(Month(Date))) +_
    zTrim(Str(Year(Date))) + zTrim(Str(Hour(Time$))) + zTrim(Str(Minute(Time$))) +
zTrim(Str(Second(Time)))
    saveIndex = 1
    Resume Save2New
  End If
  End Sub

'Save input box contents and save the current session relevant settings to system registry.
Public Sub SaveAllOnExit()
  On Error GoTo ErrHandler
  Dim dExt As String, saveIndex As Integer, sFilename As String
  Dim nPth As Integer
  dExt = "_ed"
  saveIndex = 2
  sFilename = txtAutosaveFilename.Text + zTrim(Str(Day(Date$))) + zTrim(Str(Month(Date))) +_
      zTrim(Str(Year(Date))) + zTrim(Str(Hour(Time$))) + zTrim(Str(Minute(Time$))) +
zTrim(Str(Second(Time)))
  rtxtScratchPad.Text = rtxtInputData.Text + rtxtInputBuffer.Text
  If chkAutoAddCSVHeader And gotHeader And Mid(rtxtScratchPad.Text, 1, _
         Len(txtCSVHeader.Text)) <> txtCSVHeader.Text Then
    rtxtScratchPad.Text = txtCSVHeader.Text + rtxtScratchPad.Text
  End If
TrySave2New:
  If Len(Trim(rtxtScratchPad.Text)) > 0 Then
    rtxtScratchPad.SaveFile txtAutosaveFilePath.Text + sFilename + "__" + Trim(Str(saveIndex)) + "."
+ dExt, rtfText
  End If
  SaveSetting App.EXEName, Me.Caption, "lastCOM", lastCOM
  SaveSetting App.EXEName, Me.Caption, "lastBaud", lastBaud
  SaveSetting App.EXEName, Me.Caption, "lastInputLen", lastInputLen
  SaveSetting App.EXEName, Me.Caption, "lastAutoStart", mnuAutostart_AutoReceive.Checked
```

```
  SaveSetting App.EXEName, Me.Caption, "lastIconize", mnuMinimizeToTray.Checked
  SaveSetting App.EXEName, Me.Caption, "lastAutoSaveFileName", Me.txtAutosaveFilename.Text
  SaveSetting App.EXEName, Me.Caption, "lastAutoSavePath", txtAutosaveFilePath.Text
  SaveSetting App.EXEName, Me.Caption, "lastSaveFileName", Me.txtFilename.Text
  Exit Sub
ErrHandler:
  If Err.Number = 75 Then ' cannot access (already open by another process or invalid.
    sFilename = txtAutosaveFilename.Text + zTrim(Str(Day(Date$))) + zTrim(Str(Month(Date))) + _
        zTrim(Str(Year(Date))) + zTrim(Str(Hour(Time$))) + zTrim(Str(Minute(Time$))) +
zTrim(Str(Second(Time)))
    Resume TrySave2New
  End If
End Sub

Public Sub InputSchedule_Timer()
  COMreceive
End Sub

Private Sub COMreceive()
    Static iTic As Integer
    Dim discard1stInput As Boolean, xLen As Integer, sTics As String
    Dim subStr As String, eolPos As Long, bolPos As Long, xCrLf As String
    sTics = "00000000010001100110011001100010000"
    iTic = IIf(iTic >= 0 And iTic < 7, iTic + 1, 1)
    lblTicTac = Mid(sTics, (iTic - 1) * 5 + 1, 5)
    If chkBeginUpon1stFileBreak.Value Then
      xCrLf = CrLf + CrLf 'file break is a dbl CrLf
      xLen = 4
    Else
      xCrLf = CrLf
      xLen = 2
    End If
    discard1stInput = chkBeginUpon1stFileBreak Or chkBeginUpon1stLineBreak
    If MSComm1.InBufferCount > 0 Then
      rtxtInputBuffer.Text = rtxtInputBuffer.Text + MSComm1.Input
    End If
    If (discard1stInput And (Not gotBeginning)) Then
      If (Len(rtxtInputBuffer.Text)) > 0 Then
        bolPos = FindAtLeft(rtxtInputBuffer.Text, xCrLf)
        If bolPos > 0 Then
          rtxtInputBuffer.Text = Mid(rtxtInputBuffer.Text, bolPos + xLen)
          gotBeginning = True
        End If
      End If
    End If
    eolPos = FindAtRight(rtxtInputBuffer.Text, CrLf)
    If eolPos > 0 Then
      subStr = Mid(rtxtInputBuffer.Text, 1, eolPos + 1)
      rtxtInputData.Text = rtxtInputData.Text + subStr
      If (Len(rtxtInputBuffer.Text) > eolPos + 1) Then
        rtxtInputBuffer.Text = Mid(rtxtInputBuffer.Text, eolPos + 2)
      Else
        rtxtInputBuffer.Text = ""
      End If
    End If
End Sub

Private Sub rtxtInputData_Change()
  AutoFindCSVHeader
End Sub
Private Sub rtxtInputData_DblClick()
  Static togSpad As Boolean
  togSpad = Not togSpad
  If togSpad Then
    rtxtInputData.Height = 2170
  Else
    rtxtInputData.Height = 2890
  End If
End Sub

Private Sub txtAutoReceiveInterval_Change()
  chkAutoReceive.Value = vbUnchecked
  If Val(txtAutoReceiveInterval.Text) > 0 Then
    InputSchedule.Interval = Min(Val(txtAutoReceiveInterval.Text) * 1000, 65535)
    If Val(txtAutoReceiveInterval.Text) * 1000 > 65535 Then
      txtAutoReceiveInterval.Text = InputSchedule.Interval / 1000
    End If
```

```
  End If
End Sub

Private Sub txtAutoReceiveInterval_LostFocus()
  If Val(txtAutoReceiveInterval.Text) > 0 Then
    InputSchedule.Interval = Min(Val(txtAutoReceiveInterval.Text) * 1000, 65535)
    If Val(txtAutoReceiveInterval.Text) * 1000 > 65535 Then
      txtAutoReceiveInterval.Text = InputSchedule.Interval / 1000
    End If
  End If
End Sub

Private Sub txtAutosaveInterval_Change()
  chkAutosave.Value = vbUnchecked
  If Val(Trim(txtAutosaveInterval.Text)) > 0 Then
    DataBakSchedule.Interval = Min(Val(txtAutosaveInterval.Text) * 1000, 65535)
    If Val(txtAutosaveInterval.Text) * 1000 > 65535 Then
      txtAutosaveInterval.Text = DataBakSchedule.Interval / 1000
    End If
  End If
End Sub

Private Sub txtAutosaveInterval_LostFocus()
  If Val(Trim(txtAutosaveInterval.Text)) > 0 Then
    DataBakSchedule.Interval = Min(Val(txtAutosaveInterval.Text) * 1000, 65535)
    If Val(txtAutosaveInterval.Text) * 1000 > 65535 Then
      txtAutosaveInterval.Text = DataBakSchedule.Interval / 1000
    End If
  End If
End Sub



Private Sub txtCSVHeader_Change()
Dim posL As Long, posR As Long, msg As String
If Right(txtCSVHeader.Text, 2) = CrLf Then gotHeader = True
posL = FindAtLeft(txtCSVHeader.Text, CrLf)
posR = FindAtRight(txtCSVHeader.Text, CrLf)
If ((posR > 0) And ((posR + 1) <> Len(txtCSVHeader.Text))) Then
  msg = "the CrLf must be at end of line"
End If
msg = IIf(Len(Trim(msg)) > 0, msg + CrLf, msg)
msg = IIf(posR = posR, msg, msg + "There must be only one CrLf in the header line")
If (Len(Trim(msg)) > 0) Then
  MsgBox msg, vbExclamation, "Malformated CSV file header:"
  gotHeader = False
End If
If (Len(Trim(txtCSVHeader.Text)) = 0) Then
  gotHeader = False
End If
End Sub

Private Sub txtCSVHeader_LostFocus()
  Dim ln As Long
  gotHeader = False
  txtCSVHeader.Text = Trim(txtCSVHeader.Text)
  ln = Len(txtCSVHeader.Text)
  If ln > 0 Then
    If ln = 2 And Right(txtCSVHeader.Text, 2) = CrLf Or _
      (ln = 1) And Not IsAlphabetic(txtCSVHeader.Text) Then
      txtCSVHeader.Text = ""
    ElseIf ln > 2 And Right(txtCSVHeader.Text, 2) = CrLf Then
      gotHeader = True
    ElseIf (ln = 1) And IsAlphabetic(txtCSVHeader.Text) Or _
      (ln = 2) And txtCSVHeader.Text <> CrLf Or _
      (ln > 2) And Right(txtCSVHeader.Text, 2) <> CrLf Then
        txtCSVHeader.Text = txtCSVHeader.Text + CrLf
        gotHeader = True
    End If
  End If
End Sub

Private Sub txtInputLen_Change()
 If Len(txtInputLen.Text) > 0 Then
  MSComm1.InputLen = txtInputLen.Text
 End If
End Sub
```

```
Private Sub txtInputLen_LostFocus()
 If Len(txtInputLen.Text) > 0 Then
  MSComm1.InputLen = txtInputLen.Text
 End If
End Sub

File: frmAbout.frm
VERSION 5.00
Begin VB.Form frmAbout
   BorderStyle     =   3  'Fixed Dialog
   Caption         =   "About DatalogWinLink"
   ClientHeight    =   2916
   ClientLeft      =   2340
   ClientTop       =   1932
   ClientWidth     =   5736
   ClipControls    =   0   'False
   LinkTopic       =   "Form2"
   MaxButton       =   0   'False
   MinButton       =   0   'False
   ScaleHeight     =   2007.592
   ScaleMode       =   0  'User
   ScaleWidth      =   5380.766
   ShowInTaskbar   =   0   'False
   Begin VB.PictureBox picIcon
      AutoSize        =   -1  'True
      BorderStyle     =   0  'None
      ClipControls    =   0   'False
      FillStyle       =   0  'Solid
      Height          =   384
      Left            =   240
      Picture         =   "frmAbout.frx":0000
      ScaleHeight     =   263.118
      ScaleMode       =   0  'User
      ScaleWidth      =   263.118
      TabIndex        =   1
      Top             =   240
      Width           =   384
   End
   Begin VB.CommandButton cmdOK
      Cancel          =   -1  'True
      Caption         =   "OK"
      Default         =   -1  'True
      Height          =   345
      Left            =   4320
      TabIndex        =   0
      Top             =   2520
      Width           =   1260
   End
   Begin VB.Line Line1
      BorderColor     =   &H00808080&
      BorderStyle     =   6  'Inside Solid
      Index           =   1
      X1              =   84.426
      X2              =   5309.473
      Y1              =   1687.451
      Y2              =   1687.451
   End
   Begin VB.Label lblDescription
      Caption         = "This is the Windows PC side program, that communicates with " & _
                        "the MCU based Solar Thermal Energy System  (SolarTech) Controller," & _
                        " performing real time data logging. These systems where prototyped " & _
                        "by the author at UKZN (in collaboration with UEM/Mozambique) in partial" & _
                " fulfillment of the requirements for the degree of MSc Physics - Renewable Energy."
      ForeColor       =   &H00000000&
      Height          =   1404
      Left            =   936
      TabIndex        =   2
      Top             =   960
      Width           =   4476
   End
   Begin VB.Label lblTitle
      Caption         =   "SolarTech Controller Datalogging Windows Link.  Author: Doho, G.J."
      ForeColor       =   &H00000000&
      Height          =   480
      Left            =   936
      TabIndex        =   3
```

```
      Top             =   120
      Width           =   4008
   End
   Begin VB.Line Line1
      BorderColor     =   &H00FFFFFF&
      BorderWidth     =   2
      Index           =   0
      X1              =   98.497
      X2              =   5309.473
      Y1              =   1697.779
      Y2              =   1697.779
   End
   Begin VB.Label lblVersion
      Caption         =   "Version 2.2 - 17 April 2009"
      Height          =   228
      Left            =   936
      TabIndex        =   4
      Top             =   600
      Width           =   4008
   End
End
Attribute VB_Name = "frmAbout"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
' SolarTech Datalogging Windows Link credits info form (frmAbout.frm)
' Author: Doho, G.J., 2007/2009 (UKZN StudentNo: 207514966)
Option Explicit

Private Sub cmdOK_Click()
  Unload Me
End Sub

Private Sub Form_Load()
    'Me.Caption = "About " & App.Title
    'lblVersion.Caption = "Version " & App.Major & "." & App.Minor & "." & App.Revision
    'lblTitle.Caption = App.Title
End Sub
```

File: modDataLogWinLink.bas (Visual Basic module)

```
Attribute VB_Name = "modDataLogWin"
' SolarTech Datalogging Windows Link VB module (modDataLogWinLink.bas)
' Author: Doho, G.J., 2007/2009 (UKZN StudentNo: 207514966)

Option Explicit

'Initialization  for minimize to taskbar notification area (sytem tray).
'as required by Shell_NotifyIcon API call interface speccification.

'user defined type for holding the tray icon
Public Type NOTIFYICONDATA
 cbSize As Long
 hwnd As Long
 uId As Long
 uFlags As Long
 uCallBackMessage As Long
 hIcon As Long
 szTip As String * 64
End Type

'constants for Tray icon handler
Public Const NIM_ADD = &H0
Public Const NIM_MODIFY = &H1
Public Const NIM_DELETE = &H2
Public Const NIF_MESSAGE = &H1
Public Const NIF_ICON = &H2
Public Const NIF_TIP = &H4
Public Const WM_MOUSEMOVE = &H200
Public Const WM_LBUTTONDOWN = &H201     'Button down
Public Const WM_LBUTTONUP = &H202       'Button up
Public Const WM_LBUTTONDBLCLK = &H203   'Double-click
```

```
Public Const WM_RBUTTONDOWN = &H204     'Button down
Public Const WM_RBUTTONUP = &H205       'Button up
Public Const WM_RBUTTONDBLCLK = &H206   'Double-click

'Taskbar Notification API calls declarations
Public Declare Function SetForegroundWindow Lib "user32" (ByVal hwnd As Long) As Long
Public Declare Function Shell_NotifyIcon Lib "shell32" Alias "Shell_NotifyIconA" _
(ByVal dwMessage As Long, pnid As NOTIFYICONDATA) As Boolean
Public nid As NOTIFYICONDATA



Public Function FindAtLeft(aString As String, aSubStr As String) As String
Dim i As Integer
  FindAtLeft = 0
  If Len(aString) < Len(aSubStr) Then Exit Function
  For i = 1 To (Len(aString) - Len(aSubStr) + 1)
  'For i = 1 To Len(aString)
    If (Mid(aString, i, Len(aSubStr)) = aSubStr) Then
      FindAtLeft = i
      Exit For
    End If
  Next i
End Function

Public Function FindAtRight(aString As String, aSubStr As String) As String
Dim i As Integer
  FindAtRight = 0
  If Len(aString) < Len(aSubStr) Then Exit Function
  For i = 1 To (Len(aString) - Len(aSubStr) + 1)
    If (Mid(aString, (Len(aString) - Len(aSubStr) + 2) - i, Len(aSubStr)) = aSubStr) Then
      FindAtRight = (Len(aString) - Len(aSubStr) + 2) - i
      Exit For
    End If
  Next i
End Function

Public Function zTrim(aString As String, Optional ByVal nWid As Variant, Optional cPad As Variant,
Optional ByVal bLeft As Variant) As String
Dim xNull As Integer, xMiss As Integer, xEmpty As Integer
  If IsMissing(cPad) Then
    cPad = "0"
  End If
  If IsMissing(nWid) Then
    nWid = 2
  End If
  If IsMissing(bLeft) Then
    bLeft = True
  End If
  zTrim = Trim(aString)
  While Len(zTrim) < nWid
    zTrim = IIf(bLeft, cPad + zTrim, zTrim + cPad)
  Wend
End Function

Public Function Max(lVal As Variant, rVal As Variant) As Variant
  Max = IIf(lVal < rVal, rVal, lVal)
End Function

Public Function Min(lVal As Variant, rVal As Variant) As Variant
  Min = IIf(lVal < rVal, lVal, rVal)
End Function

Public Function IsAlphabetic(ByVal aChar As String) As Boolean
IsAlphabetic = ((Asc(aChar) >= Asc("A") And Asc(aChar) <= Asc("Z")) Or _
                (Asc(aChar) >= Asc("a") And Asc(aChar) <= Asc("z")))
End Function
```

Appendix C – Sample Tables of collected and logged data

In this appendix we show sample tables of some of the experiments listed and discussed in chapters VII and VIII. They are intended only for giving an idea of what the datalogs are.

SampleTime_secs	LM35_temperarture_C
7	26.2
8	26.4
9	26.3
10	26.5
11	26.3
12	26.5
13	26.5
14	26.25
15	26.3
16	26.4
17	26.35
18	26.3
19	26.3
20	26.3
21	26.25
22	26.25
23	26.25
24	26.3
25	26.3
26	26.3
27	26.3
28	26.3
29	26.3
30	26.3
30	26.3
31	26.3
32	26.3
33	26.3
34	26.3
35	26.3
36	26.3
37	26.3
38	26.25
39	26.25
40	26.25
41	26.25
42	26.3
43	26.3
44	26.4
45	26.4
46	26.3
47	26.3
48	26.4
49	26.4
50	26.4
51	26.4
52	26.4
53	26.3
54	26.3

Table C.1 – Ambient temperature acquisition and logging - 13080713h35 (sample records)

SampleTime _secs	TDCtemperarture (⁰C)	Average_Temperature (⁰C)
448	26	
449	26	26.17
450	26.5	26.17
451	26	26.17
452	26	25.92
453	25.75	25.92
454	26	26.08
455	26.5	26.08
456	25.75	26.08
457	26	25.92
458	26	26.00
459	26	26.00
460	26	26.17
461	26.5	26.08
462	25.75	26.08
463	26	25.92
464	26	26.00
465	26	26.00
466	26	26.17
467	26.5	26.17
468	26	26.17
469	26	26.17
470	26.5	26.33
471	26.5	26.33
472	26	26.08
473	25.75	25.67
474	25.25	25.67
475	26	25.75
477	25.75	25.92
478	26	25.83
479	25.75	25.92
480	26	25.92
481	26	25.92
482	25.75	25.83
483	25.75	25.83
484	26	25.83
485	25.75	25.92
486	26	25.83
487	25.75	25.92
488	26	26.08
489	26.5	26.08
490	25.75	26.25
491	26.5	26.00
492	25.75	26.25
493	26.5	26.00
494	25.75	26.00
495	25.75	25.58
496	25.25	25.83
497	26.5	25.83
498	25.75	26.08
499	26	25.83
500	25.75	26.08
501	26.5	26.00

Table C.2 – Thermocouple temperature acquisition and logging experiment B – 23/Aug/2007 14h (sample records)

SampleTime_secs	TDC_WaterTemperarture (^{0}C)
0	18.5
1	18.25
2	18.75
3	18.25
4	18.5
5	18.25
6	18.5
7	18.25
8	18.25
9	18.5
10	18.75
11	18.75
12	18.5
13	18.75
14	19.5
15	18.25
16	8.5
17	3.25
18	2.5
19	1.25
20	1.25
21	2.25
22	4.5
23	1.75
24	0.25
25	0.25
26	0.5
27	0
28	0.25
29	0
30	0
31	0.25
32	0
33	0
34	0.5
35	0
36	0
37	0
38	0
39	0.25
40	0
41	0
42	0.25
43	0
44	0
45	0
46	0
47	0.5
48	0
49	0
50	0
51	0
52	0
53	0.5
54	0

Table C.3 – "Ice-to-boiling" experiment - 23/Jul/2008 (sample records)

LocalTime	SolarTime	SolarTimeMins	StPtHourAngle	StPtDeclAngle	ActualDeclAng	ActualHourAng
Tu,230609,141542	140957	849.96	32.49	23.44	-21.24	0.57
Tu,230609,141543	140958	849.98	32.49	23.44	-21.24	0.57
Tu,230609,141543	140958	849.98	32.49	23.44	-20.67	0.57
Tu,230609,141544	140959	850	32.5	23.44	-20.67	0.57
Tu,230609,141544	140959	850	32.5	23.44	-20.67	0.57
Tu,230609,141544	140959	850	32.5	23.44	-18.97	0.57
Tu,230609,141544	140959	850	32.5	23.44	-18.97	0.57
Tu,230609,141545	141000	850.01	32.5	23.44	-18.97	0.57
Tu,230609,141545	141000	850.01	32.5	23.44	-18.97	0.57
Tu,230609,141545	141000	850.01	32.5	23.44	-18.97	0.57
Tu,230609,141546	141001	850.03	32.51	23.44	-18.97	0.57
Tu,230609,141546	141001	850.03	32.51	23.44	-18.97	0.57
Tu,230609,141546	141001	850.03	32.51	23.44	-18.13	0.57
Tu,230609,141546	141001	850.03	32.51	23.44	-18.13	0.57
Tu,230609,141547	141002	850.05	32.51	23.44	-18.13	0.28
Tu,230609,141547	141002	850.05	32.51	23.44	-18.13	0.28
Tu,230609,141547	141002	850.05	32.51	23.44	-18.13	0.28
Tu,230609,141547	141002	850.05	32.51	23.44	-15.86	0.28
Tu,230609,141548	141003	850.06	32.52	23.44	-15.86	0.28
Tu,230609,141548	141003	850.06	32.52	23.44	-15.58	0.28
Tu,230609,141548	141003	850.06	32.52	23.44	-15.86	0.28
Tu,230609,141548	141003	850.06	32.52	23.44	-15.86	0.28
Tu,230609,141549	141004	850.08	32.52	23.44	-15.86	0.57
Tu,230609,141549	141004	850.08	32.52	23.44	-15.86	0.28
Tu,230609,141549	141004	850.08	32.52	23.44	-15.86	0.28
Tu,230609,141549	141004	850.08	32.52	23.44	-15.29	0.28
Tu,230609,141550	141005	850.1	32.52	23.44	-14.73	0.28
Tu,230609,141550	141005	850.1	32.52	23.44	-14.73	0.28
Tu,230609,141550	141005	850.1	32.52	23.44	-14.73	0.28
Tu,230609,141550	141005	850.1	32.52	23.44	-14.73	0.28
Tu,230609,141551	141006	850.11	32.53	23.44	-14.73	0.28
Tu,230609,141551	141006	850.11	32.53	23.44	-14.73	0.28
Tu,230609,141551	141006	850.11	32.53	23.44	-13.31	0.28
Tu,230609,141552	141007	850.13	32.53	23.44	-13.03	0.28
Tu,230609,141552	141007	850.13	32.53	23.44	-13.03	0.57
Tu,230609,141552	141007	850.13	32.53	23.44	-12.74	0.57
Tu,230609,141552	141007	850.13	32.53	23.44	-12.74	0.57
Tu,230609,141553	141008	850.15	32.54	23.44	-12.18	0.28
Tu,230609,141553	141008	850.15	32.54	23.44	-12.18	0.28
Tu,230609,141553	141008	850.15	32.54	23.44	-10.2	0.28
Tu,230609,141553	141008	850.15	32.54	23.44	-10.48	0.28
Tu,230609,141554	141009	850.16	32.54	23.44	-10.48	0.28
Tu,230609,141554	141009	850.16	32.54	23.44	-10.48	0.28
Tu,230609,141554	141009	850.16	32.54	23.44	-10.48	0.28
Tu,230609,141554	141009	850.16	32.54	23.44	-10.48	0.28
Tu,230609,141555	141010	850.18	32.54	23.44	-10.48	0.28
Tu,230609,141555	141010	850.18	32.54	23.44	-10.48	0.28
Tu,230609,141555	141010	850.18	32.54	23.44	-9.63	0.28
Tu,230609,141555	141010	850.18	32.54	23.44	-9.35	0.28
Tu,230609,141556	141011	850.2	32.55	23.44	-9.35	0.28
Tu,230609,141556	141011	850.2	32.55	23.44	-9.35	0.28
Tu,230609,141556	141011	850.2	32.55	23.44	-9.63	0.28
Tu,230609,141556	141011	850.2	32.55	23.44	-7.36	0.28
Tu,230609,141557	141012	850.21	32.55	23.44	-7.36	0.28
Tu,230609,141557	141012	850.21	32.55	23.44	-7.36	0.57

Table C.4 – Basic sun tracking experiment A – 23/06/09 (sample records)

Appendix D – Datasheets of the main electronic components

Datasheet of ATmega32 IC – 8bit MCU (pages 1 and 2)

Features

- High-performance, Low-power AVR® 8-bit Microcontroller
- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single-clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 16 MIPS Throughput at 16 MHz
  - On-chip 2-cycle Multiplier
- Nonvolatile Program and Data Memories
  - 32K Bytes of In-System Self-Programmable Flash
    Endurance: 10,000 Write/Erase Cycles
  - Optional Boot Code Section with Independent Lock Bits
    In-System Programming by On-chip Boot Program
    True Read-While-Write Operation
  - 1024 Bytes EEPROM
    Endurance: 100,000 Write/Erase Cycles
  - 2K Byte Internal SRAM
  - Programming Lock for Software Security
- JTAG (IEEE std. 1149.1 Compliant) Interface
  - Boundary-scan Capabilities According to the JTAG Standard
  - Extensive On-chip Debug Support
  - Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Four PWM Channels
  - 8-channel, 10-bit ADC
    8 Single-ended Channels
    7 Differential Channels in TQFP Package Only
    2 Differential Channels with Programmable Gain at 1x, 10x, or 200x
  - Byte-oriented Two-wire Serial Interface
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated RC Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby and Extended Standby
- I/O and Packages
  - 32 Programmable I/O Lines
  - 40-pin PDIP, 44-lead TQFP, and 44-pad QFN/MLF
- Operating Voltages
  - 2.7 - 5.5V for ATmega32L
  - 4.5 - 5.5V for ATmega32
- Speed Grades
  - 0 - 8 MHz for ATmega32L
  - 0 - 16 MHz for ATmega32
- Power Consumption at 1 MHz, 3V, 25°C for ATmega32L
  - Active: 1.1 mA
  - Idle Mode: 0.35 mA
  - Power-down Mode: < 1 µA

8-bit AVR® Microcontroller with 32K Bytes In-System Programmable Flash

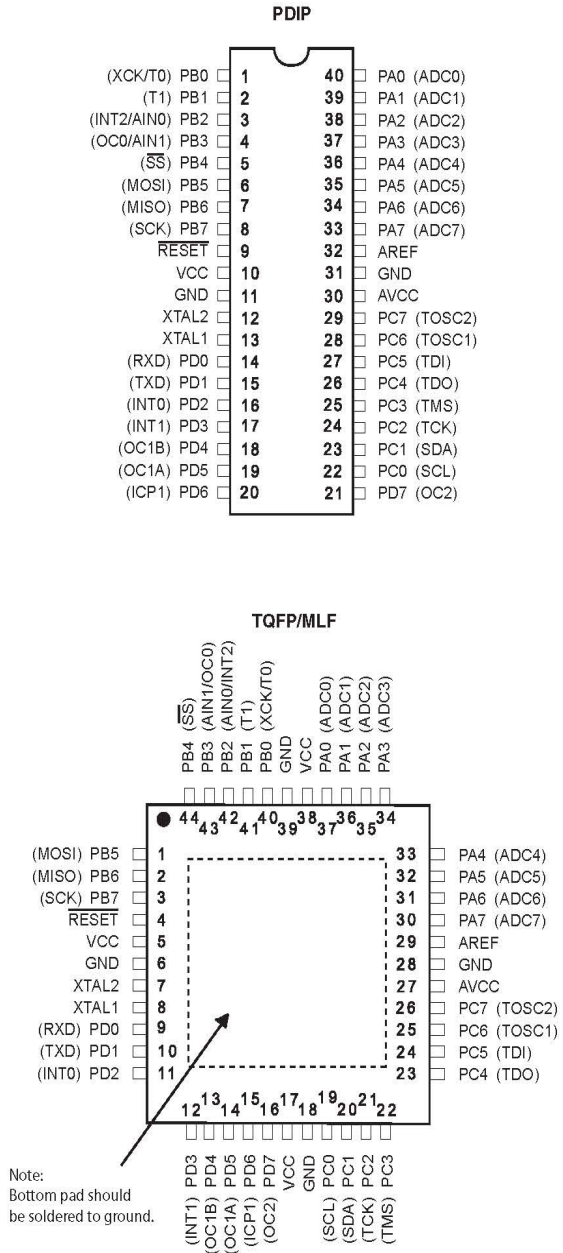
ATmega32
ATmega32L

Summary

2503IS–AVR–04/06

Pin Configurations

Figure 1. Pinout ATmega32



Datasheet of DS1302 IC - Real time clock (some relevant pages)

DALLAS SEMICONDUCTOR

DS1302

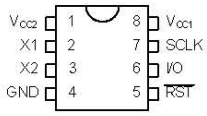
Trickle Charge Timekeeping Chip

www.dalsemi.com

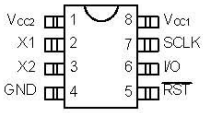
FEATURES

- Real time clock counts seconds, minutes, hours, date of the month, month, day of the week, and year with leap year compensation valid up to 2100
- 31 x 8 RAM for scratchpad data storage
- Serial I/O for minimum pin count
- 2.0–5.5 volt full operation
- Uses less than 300 nA at 2.0 volts
- Single–byte or multiple–byte (burst mode) data transfer for read or write of clock or RAM data
- 8–pin DIP or optional 8–pin SOICs for surface mount
- Simple 3–wire interface
- TTL–compatible (V_{CC} = 5V)
- Optional industrial temperature range –40°C to +85°C
- DS1202 compatible
- Added features over DS1202
  - Optional trickle charge capability to V_{CC1}
  - Dual power supply pins for primary and backup power supplies
  - Backup power supply pin can be used for battery or super cap input
  - Additional scratchpad memory (7 bytes)

PIN ASSIGNMENT



DS1302
8-Pin DIP (300-Mil)



DS1302S 8-Pin SOIC (200-Mil)
DS1302Z 8-Pin SOIC (150-Mil)

PIN DESCRIPTION

X1, X2	– 32.768 kHz Crystal Pins
GND	– Ground
$\overline{RST}$	– Reset
I/O	– Data Input/Output
SCLK	– Serial Clock
V_{CC1}, V_{CC2}	– Power Supply Pins

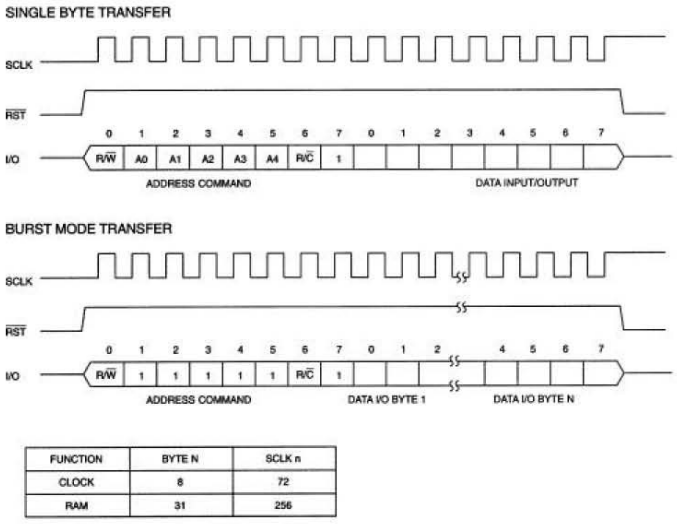
ORDERING INFORMATION

PART #	DESCRIPTION
DS1302	Serial Timekeeping Chip; 8–pin DIP
DS1302S	Serial Timekeeping Chip; 8–pin SOIC (200-mil)
DS1302Z	Serial Timekeeping Chip; 8–pin SOIC (150-mil)

DESCRIPTION

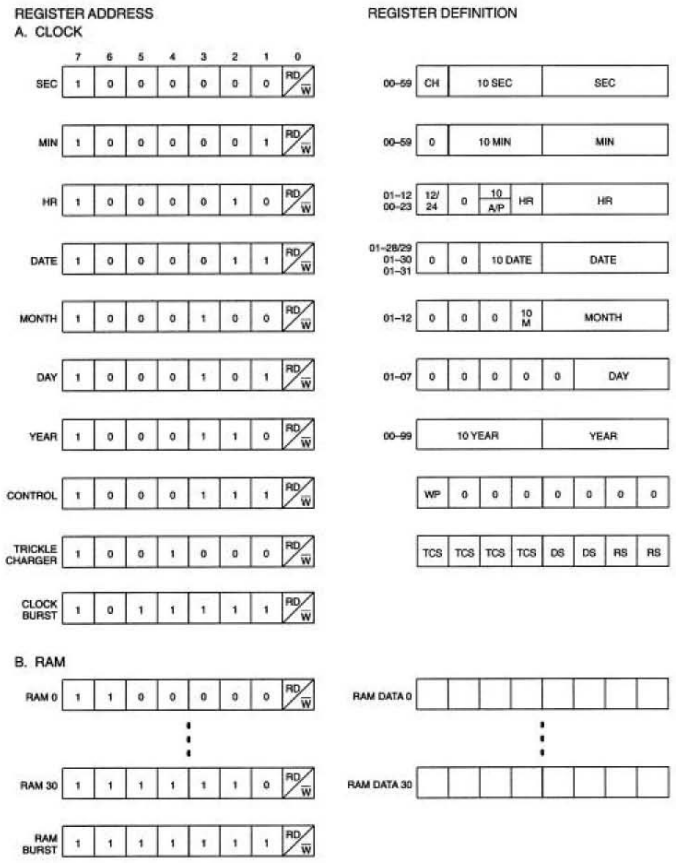
The DS1302 Trickle Charge Timekeeping Chip contains a real time clock/calendar and 31 bytes of static RAM. It communicates with a microprocessor via a simple serial interface. The real time clock/calendar provides seconds, minutes, hours, day, date, month, and year information. The end of the month date is automatically adjusted for months with less than 31 days, including corrections for leap year. The clock operates in either the 24–hour or 12–hour format with an AM/PM indicator.

DATA TRANSFER SUMMARY Figure 3



FUNCTION	BYTE N	SCLK n
CLOCK	8	72
RAM	31	256

REGISTER ADDRESS/DEFINITION Figure 4



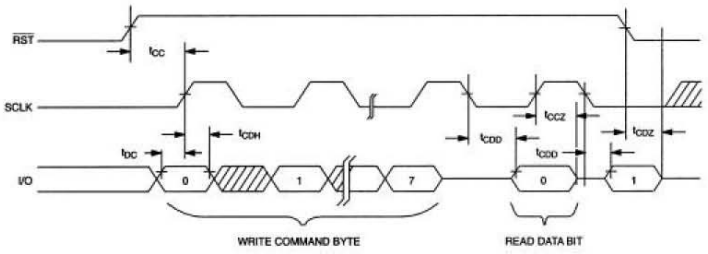
DS1302

AC ELECTRICAL CHARACTERISTICS (cont'd) (0°C to 70°C; V_{CC} = 2.0 to 5.5V*)

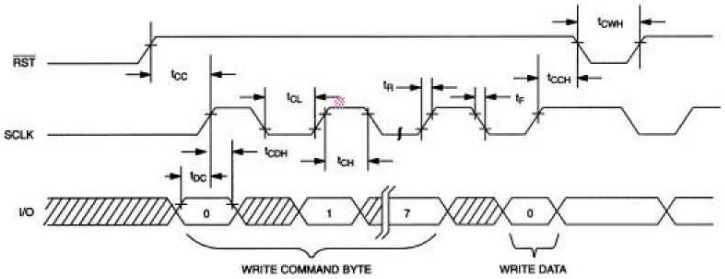
CLK Frequency	t_{CLK}	V_{CC}=2.0V			0.5	MHz	7
		V_{CC}=5V	DC		2.0		
CLK Rise and Fall	t_R, t_F	V_{CC}=2.0V			2000	ns	
		V_{CC}=5V			500		
$\overline{RST}$ to CLK Setup	t_{CC}	V_{CC}=2.0V	4			µs	7
		V_{CC}=5V	1				
CLK to $\overline{RST}$ Hold	t_{CCH}	V_{CC}=2.0V	240			ns	7
		V_{CC}=5V	60				
$\overline{RST}$ Inactive Time	t_{CWH}	V_{CC}=2.0V	4			µs	7
		V_{CC}=5V	1				
$\overline{RST}$ to I/O High Z	t_{CDZ}	V_{CC}=2.0V			280	ns	7
		V_{CC}=5V			70		
SCLK to I/O High Z	t_{CCZ}	V_{CC}=2.0V			280	ns	7
		V_{CC}=5V			70		

*Unless otherwise noted.

TIMING DIAGRAM: READ DATA TRANSFER Figure 5



TIMING DIAGRAM: WRITE DATA TRANSFER Figure 6



Datasheet of 74C923 IC – Keyboard encoder (some relevant pages)

National Semiconductor

July 1993

MM54C922/MM74C922 16-Key Encoder
MM54C923/MM74C923 20-Key Encoder

General Description

These CMOS key encoders provide all the necessary logic to fully encode an array of SPST switches. The keyboard scan can be implemented by either an external clock or external capacitor. These encoders also have on-chip pull-up devices which permit switches with up to 50 kΩ on resistance to be used. No diodes in the switch array are needed to eliminate ghost switches. The internal debounce circuit needs only a single external capacitor and can be defeated by omitting the capacitor. A Data Available output goes to a high level when a valid keyboard entry has been made. The Data Available output returns to a low level when the entered key is released, even if another key is depressed. The Data Available will return high to indicate acceptance of the new key after a normal debounce period; this two-key rollover is provided between any two switches.

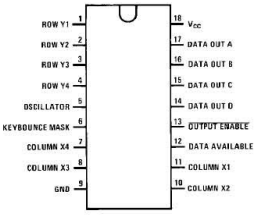
An internal register remembers the last key pressed even after the key is released. The TRI-STATE® outputs provide for easy expansion and bus operation and are LPTTL compatible.

Features

- 50 kΩ maximum switch on resistance
- On or off chip clock
- On-chip row pull-up devices
- 2 key roll-over
- Keybounce elimination with single capacitor
- Last key register at outputs
- TRI-STATE outpust LPTTL compatible
- Wide supply range 3V to 15V
- Low power consumption

Connection Diagrams

Pin Assignment for Dual-In-Line Package

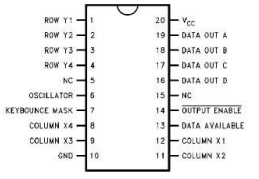


TL/F/6037–1

Top View

Order Number MM54C922 or MM74C922

Pin Assignment for SOIC

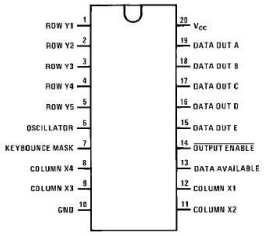


TL/F/6037–14

Top View

Order Number MM74C922

Pin Assignment for DIP and SOIC Package



TL/F/6037–2

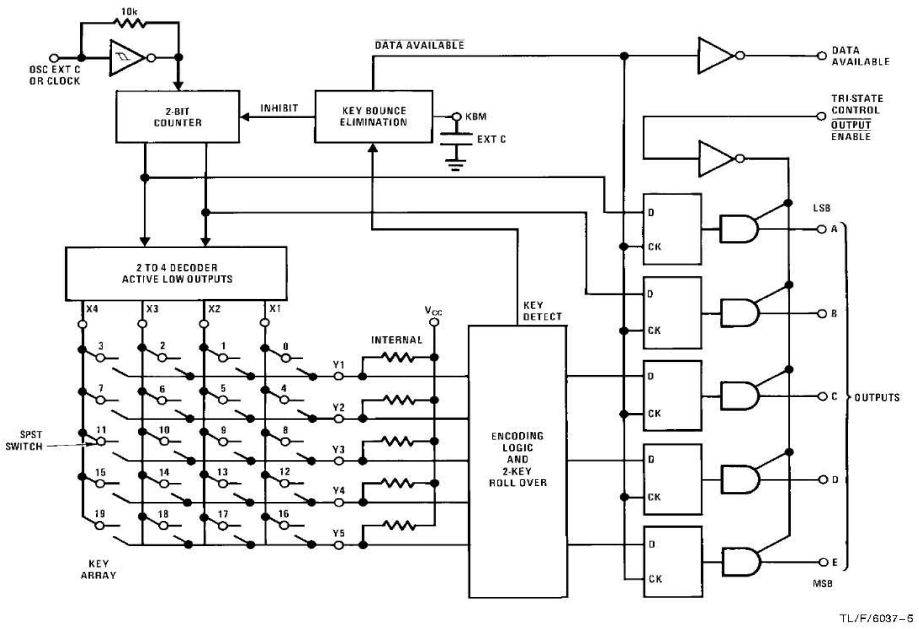
Top View

Order Number MM54C923 or MM74C923

TRI-STATE® is a registered trademark of National Semiconductor Corporation.

RRD-B30M105/Printed in U. S. A.

Block Diagram



TL/F/6037–5

Truth Table

Switch		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Position		Y1,X1	Y1,X2	Y1,X3	Y1,X4	Y2,X1	Y2,X2	Y2,X3	Y2,X4	Y3,X1	Y3,X2	Y3,X3	Y3,X4	Y4,X1	Y4,X2	Y4,X3	Y4,X4	Y5*,X1	Y5*,X2	Y5*,X3	Y5*,X4
D																					
A	A	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
T	B	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
A	C	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
O	D	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0
U	E*	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
T																					

*Omit for MM54C922/MM74C922

Datasheet of AK-XXXX keypads (some relevant pages)

STANDARD KEYPADS

GENERAL SPECIFICATION

- Contact rating: 20mA, 24VDC
- Contact resistance: 200 ohm max
- Life: 1,000,000 cycles per key
- Operating Temperature: -20°C to +60°C
- Storage Temperature: -40°C to +65°C

KEYPAD PART NUMBER INFORMATION

AK-X X X X - X - X X X - X X - X X
(a) (b) (c)(d)(e) (f) (g)

(a) **MODEL NUMBER:** 101, 102, 103, 104, 207, 304, 304-FM, 507, 707, 804, 1604, 1607

(b) **LEGEND (CHARACTER) TYPE:** A (ALPHA-NUMERIC LEGEND)
N (NUMERIC LEGEND)
B (BLANK)
AR (ARROW)

(c) **BEZEL COLOR:** W: WHITE COLOR
B: BLACK COLOR
G: GRAY COLOR
I: IVORY COLOR
S: SILVER COLOR FOR METAL BEZEL

(d) **KEYTOP COLOR:** W: WHITE COLOR
B: BLACK COLOR
G: GRAY COLOR
I: IVORY COLOR
S: SILVER COLOR FOR METAL KEYTOP

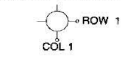
(e) **LEGEND (CHARACTER) COLOR:** W: WHITE COLOR
B: BLACK COLOR
Y: YELLOW COLOR
L: LED LIGHTING (BACKLIGHTING)

(f) **WATER PROOF:** WP

(g) **METAL BEZEL, METAL KEYTOP:** MM

STANDARD MATRIX CIRCUIT DIAGRAM

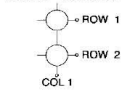
CIRCUIT DIAGRAM



AK-101

OUTPUT ARRANGEMENT	
OUTPUT PIN NO.	SYMBOL
1	COL 1
2	ROW 1

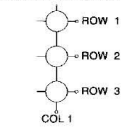
CIRCUIT DIAGRAM



AK-102

OUTPUT ARRANGEMENT	
OUTPUT PIN NO.	SYMBOL
1	ROW 1
2	ROW 2
3	COL 1

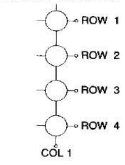
CIRCUIT DIAGRAM



AK-103

OUTPUT ARRANGEMENT	
OUTPUT PIN NO.	SYMBOL
1	ROW 1
2	ROW 2
3	ROW 3
4	COL 1

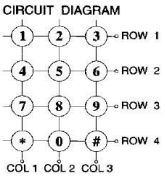
CIRCUIT DIAGRAM



AK-104

OUTPUT ARRANGEMENT	
OUTPUT PIN NO.	SYMBOL
1	ROW 1
2	ROW 2
3	ROW 3
4	ROW 4
5	COL 1

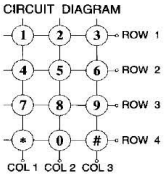
CIRCUIT DIAGRAM



AK-207, 304, 507, 707, 804

OUTPUT ARRANGEMENT	
OUTPUT PIN NO.	SYMBOL
1	COL 2
2	ROW 1
3	COL 1
4	ROW 4
5	COL 3
6	ROW 3
7	ROW 2

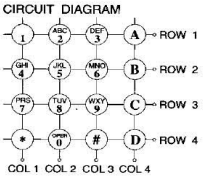
CIRCUIT DIAGRAM



AK-304-FM

OUTPUT ARRANGEMENT	
OUTPUT PIN NO.	SYMBOL
1	ROW
2	ROW 2
3	ROW 3
4	ROW 4
5	COL 1
6	COL 2
7	COL 3

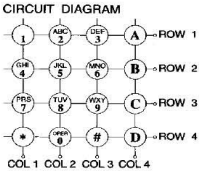
CIRCUIT DIAGRAM



AK-1604

OUTPUT ARRANGEMENT	
OUTPUT PIN NO.	SYMBOL
1	COL 1
2	COL 2
3	COL 3
4	COL 4
5	ROW 1
6	ROW 2
7	ROW 3
8	ROW 4

CIRCUIT DIAGRAM

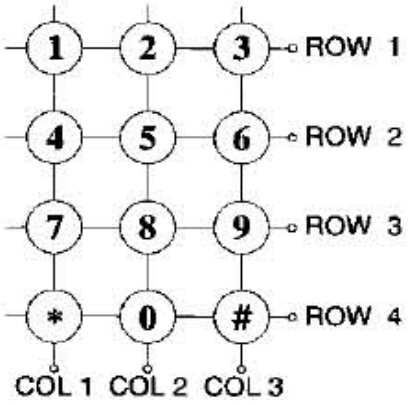


AK-1607

OUTPUT ARRANGEMENT	
OUTPUT PIN NO.	SYMBOL
1	ROW 2
2	ROW 3
3	COL 1
4	ROW 4
5	COL 2
6	COL 3
7	COL 4
8	ROW 1

AK-304NB

CIRCUIT DIAGRAM



AK-207, 304, 507, 707, 804

OUTPUT ARRANGEMENT	
OUTPUT PIN NO.	SYMBOL
1	COL 2
2	ROW 1
3	COL 1
4	ROW 4
5	COL 3
6	ROW 3
7	ROW 2

Datasheet of RS232 UART IC (page 1)

HARRIS SEMICONDUCTOR

ICL232

August 1997

+5V Powered, Dual RS-232 Transmitter/Receiver

Features

- Meets All RS-232C and V.28 Specifications
- Requires Only Single +5V Power Supply
- Onboard Voltage Doubler/Inverter
- Low Power Consumption
- 2 Drivers
  - ±9V Output Swing for +5V Input
  - 300Ω Power-off Source Impedance
  - Output Current Limiting
  - TTL/CMOS Compatible
  - 30V/µs Maximum Slew Rate
- 2 Receivers
  - ±30V Input Voltage Range
  - 3kΩ to 7kΩ Input Impedance
  - 0.5V Hysteresis to Improve Noise Rejection
- All Critical Parameters are Guaranteed Over the Entire Commercial, Industrial and Military Temperature Ranges

Applications

- Any System Requiring RS-232 Communications Port
  - Computer - Portable and Mainframe
  - Peripheral - Printers and Terminals
  - Portable Instrumentation
  - Modems
- Dataloggers

Description

The ICL232 is a dual RS-232 transmitter/receiver interface circuit that meets all EIA RS-232C and V.28 specifications. It requires a single +5V power supply, and features two onboard charge pump voltage converters which generate +10V and -10V supplies from the 5V supply.

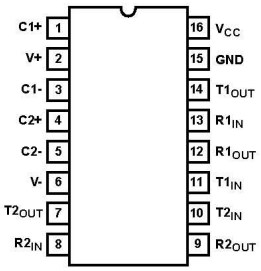
The drivers feature true TTL/CMOS input compatibility, slew-rate-limited output, and 300Ω power-off source impedance. The receivers can handle up to +30V, and have a 3kΩ to 7kΩ input impedance. The receivers also have hysteresis to improve noise rejection.

Ordering Information

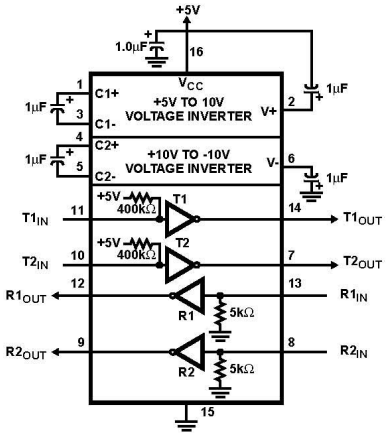
PART NUMBER	TEMP. RANGE (^{o}C)	PACKAGE	PKG. NO.
ICL232CPE	0 to 70	16 Ld PDIP	E16.3
ICL232CBE	0 to 70	16 Ld SOIC	M16.3
ICL232IPE	-40 to 85	16 Ld PDIP	E16.3
ICL232IJE	-40 to 85	16 Ld CERDIP	F16.3
ICL232IBE	-40 to 85	16 Ld SOIC	M16.3
ICL232MJE	-55 to 125	16 Ld CERDIP	F16.3

Pinout

ICL232 (PDIP, CERDIP, SOIC)
TOP VIEW



Functional Diagram



CAUTION: These devices are sensitive to electrostatic discharge. Users should follow proper IC Handling Procedures.

File Number **3020.5**

Datasheet of 74HC273 IC – PIPO register with master reset (some relevant pages)

PIN DESCRIPTION

PIN NO.	SYMBOL	NAME AND FUNCTION
1	$\overline{MR}$	master reset input (active LOW)
2, 5, 6, 9, 12, 15, 16, 19	Q_0 to Q_7	flip-flop outputs
3, 4, 7, 8, 13, 14, 17, 18	D_0 to D_7	data inputs
10	GND	ground (0 V)
11	CP	clock input (LOW-to-HIGH, edge-triggered)
20	V_{CC}	positive supply voltage

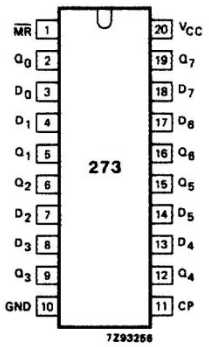


Fig.1 Pin configuration.

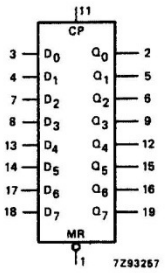


Fig.2 Logic symbol.

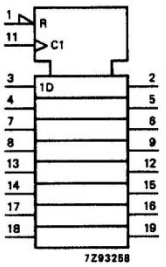


Fig.3 IEC logic symbol.

Datasheet of LM35 IC – Celsius temperature sensor (some relevant pages)

National Semiconductor

July 1999

LM35
Precision Centigrade Temperature Sensors

General Description

The LM35 series are precision integrated-circuit temperature sensors, whose output voltage is linearly proportional to the Celsius (Centigrade) temperature. The LM35 thus has an advantage over linear temperature sensors calibrated in ° Kelvin, as the user is not required to subtract a large constant voltage from its output to obtain convenient Centigrade scaling. The LM35 does not require any external calibration or trimming to provide typical accuracies of ±¼°C at room temperature and ±¾°C over a full −55 to +150°C temperature range. Low cost is assured by trimming and calibration at the wafer level. The LM35's low output impedance, linear output, and precise inherent calibration make interfacing to readout or control circuitry especially easy. It can be used with single power supplies, or with plus and minus supplies. As it draws only 60 µA from its supply, it has very low self-heating, less than 0.1°C in still air. The LM35 is rated to operate over a −55° to +150°C temperature range, while the LM35C is rated for a −40° to +110°C range (−10° with improved accuracy). The LM35 series is available packaged in hermetic TO-46 transistor packages, while the LM35C, LM35CA, and LM35D are also available in the plastic TO-92 transistor package. The LM35D is also available in an 8-lead surface mount small outline package and a plastic TO-220 package.

Features

- Calibrated directly in ° Celsius (Centigrade)
- Linear + 10.0 mV/°C scale factor
- 0.5°C accuracy guaranteeable (at +25°C)
- Rated for full −55° to +150°C range
- Suitable for remote applications
- Low cost due to wafer-level trimming
- Operates from 4 to 30 volts
- Less than 60 µA current drain
- Low self-heating, 0.08°C in still air
- Nonlinearity only ±¼°C typical
- Low impedance output, 0.1 Ω for 1 mA load

Typical Applications

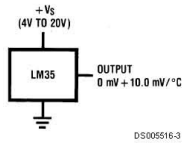
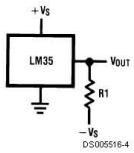


FIGURE 1. Basic Centigrade Temperature Sensor (+2°C to +150°C)



Choose $R_1 = -V_S/50\ \mu A$
$V_{OUT} = +1{,}500$ mV at +150°C
$= +250$ mV at +25°C
$= -550$ mV at −55°C

FIGURE 2. Full-Range Centigrade Temperature Sensor

TRI-STATE® is a registered trademark of National Semiconductor Corporation.

Typical Applications

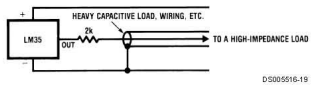


FIGURE 3. LM35 with Decoupling from Capacitive Load

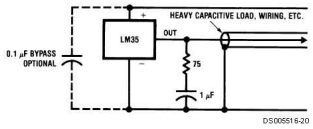


FIGURE 4. LM35 with R-C Damper

CAPACITIVE LOADS

Like most micropower circuits, the LM35 has a limited ability to drive heavy capacitive loads. The LM35 by itself is able to drive 50 pf without special precautions. If heavier loads are anticipated, it is easy to isolate or decouple the load with a resistor; see *Figure 3*. Or you can improve the tolerance of capacitance with a series R-C damper from output to ground; see *Figure 4*.

When the LM35 is applied with a 200Ω load resistor as shown in *Figure 5*, *Figure 6* or *Figure 8* it is relatively immune to wiring capacitance because the capacitance forms a bypass from ground to input, not on the output. However, as with any linear circuit connected to wires in a hostile environment, its performance can be affected adversely by intense electromagnetic sources such as relays, radio transmitters, motors with arcing brushes, SCR transients, etc, as its wiring can act as a receiving antenna and its internal junctions can act as rectifiers. For best results in such cases, a bypass capacitor from V_{IN} to ground and a series R-C damper such as 75Ω in series with 0.2 or 1 μF from output to ground are often useful. These are shown in *Figure 13*, *Figure 14*, and *Figure 16*.

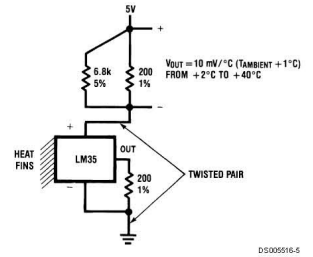


FIGURE 5. Two-Wire Remote Temperature Sensor (Grounded Sensor)

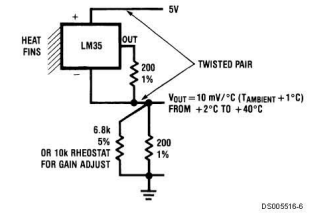


FIGURE 6. Two-Wire Remote Temperature Sensor (Output Referred to Ground)

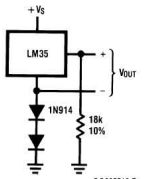


FIGURE 7. Temperature Sensor, Single Supply, −55° to +150°C

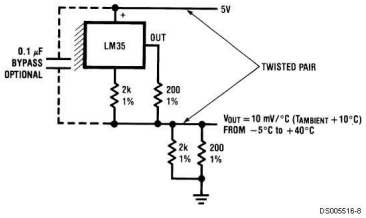


FIGURE 8. Two-Wire Remote Temperature Sensor (Output Referred to Ground)

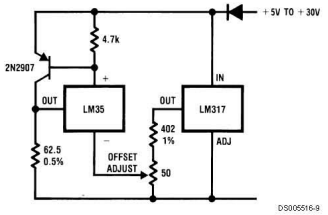


FIGURE 9. 4-To-20 mA Current Source (0°C to +100°C)

Datasheet for LM162ABC – LCD module (some relevant pages)

1. BASIC SPECIFICATIONS

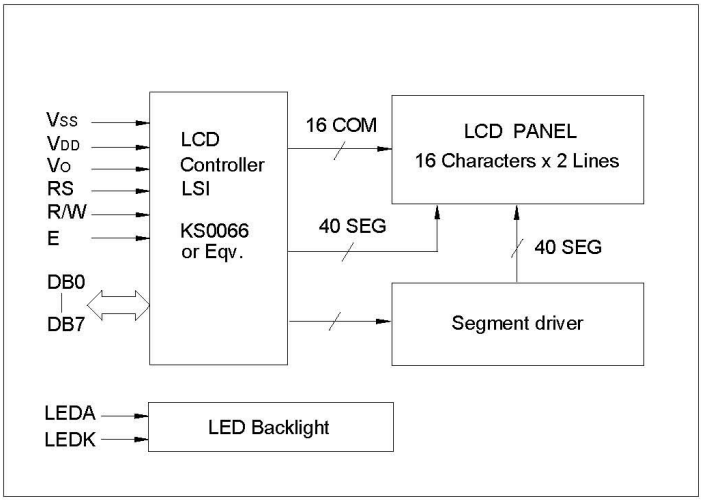
1.1 Display Specifications

LCD Mode	:	STN—Positive—Transflective
Display Color	:	Dark Blue
Background Color	:	Yellow-Green
Driving Duty	:	1/16 Duty
Viewing Direction	:	6:00
Backlight	:	LED

1.2 Mechanical Specifications

Outline Dimension	:	80.0(W) X 36.0(H) X 14.0(T)	mm
Viewing Area	:	64.6(W) X 16.0(H)	mm
Number of Characters	:	16 Characters X 2 Lines	
Character Size	:	2.95 X 5.55	mm
Dot Size	:	0.55 X 0.65	mm
Weight	:		

1.3 Block Diagram



1.4 Terminal Functions

Pin No.	Symbol	Level	Function
1	VSS	-	Ground
2	VDD	-	Power Supply for Logic (+5V)
3	VO	-	Power Supply for LCD
4	RS	H/L	Register Selection H: Display Data L: Instruction Code
5	R/W	H/L	Read/Write Selection H: Read Operation L: Write Operation
6	E	H, H→L	Enable Signal. Read data when E is "H", write data at the falling edge of E.
7	DB0	H/L	In 8-bit mode, used as low order bi-directional data bus. In 4-bit mode, open these terminals.
8	DB1	H/L	
9	DB2	H/L	
10	DB3	H/L	
11	DB4	H/L	In 8-bit mode, used as high order bi-directional data bus. In 4-bit mode, used as both high and low order data bus.
12	DB5	H/L	
13	DB6	H/L	
14	DB7	H/L	
15	LEDA	--	LED Power Supply (+5V)
16	LEDK	--	LED Power Supply (0v)

2. ABSOLUTE MAXIMUM RATINGS

Item	Symbol	Min.	Max.	Unit
Supply Voltage(Logic)	VDD-VSS	-0.3	7.0	V
Supply Voltage(LCD)	VDD-VO	-0.3	13.0	V
Input Voltage	VI	-0.3	VDD+0.3	V
Operating Temp.	Topr	-20	70	℃
Storage Temp.	Tstg	-30	80	℃

3. ELECTRICAL CHARACTERISTICS

3.1 DC Characteristics (VDD=5.0V±10%, Ta=25℃)

Item	Symbol	Condition	Min.	Typ.	Max.	UNIT
Supply Voltage (Logic)	VDD		4.5	5.0	5.5	V
Supply Voltage (LCD Drive)	VDD-VO		--	5.0	--	V
Input High Voltage	VIH		2.2	--	VDD	V
Input Low Voltage	VIL		-0.3	--	0.6	V
Output High Voltage	VOH	IOH=-0.2mA	2.4	--	VDD	V
Output Low Voltage	VOL	IOL=1.2mA	0	--	0.4	V
Supply Current (Logic)	IDD	VDD=5.0V	--	1.5	3.0	mA

3.2 Interface Timing Chart (VDD=5.0V±10%, Ta=25℃)

Mode	Characteristic	Symbol	Min.	Typ.	Max.	Unit
Write Mode Refer to fig.1	E Cycle Time	t_C	500	--	--	ns
	E Rise/Fall Time	t_R, t_F	--	--	20	
	E Pulse Width (High,Low)	t_W	230	--	--	
	R/W and RS Setup Time	t_{SU1}	40	--	--	
	R/W and RS Hold Time	t_{H1}	10	--	--	
	Data Setup Time	t_{SU2}	80	--	--	
	Data Hold Time	t_{H2}	10	--	--	
Read Mode Refer to fig.2	E Cycle Time	t_C	500	--	--	ns
	E Rise/Fall Time	t_R, t_F	--	--	20	
	E Pulse Width (High,Low)	t_W	230	--		
	R/W and RS Setup Time	t_{SU}	40	--		
	R/W and RS Hold Time	t_H	10	--		
	Data Output Delay Time	t_D	--	--	120	
	Data Hold Time	t_{DH}	5	--	--	

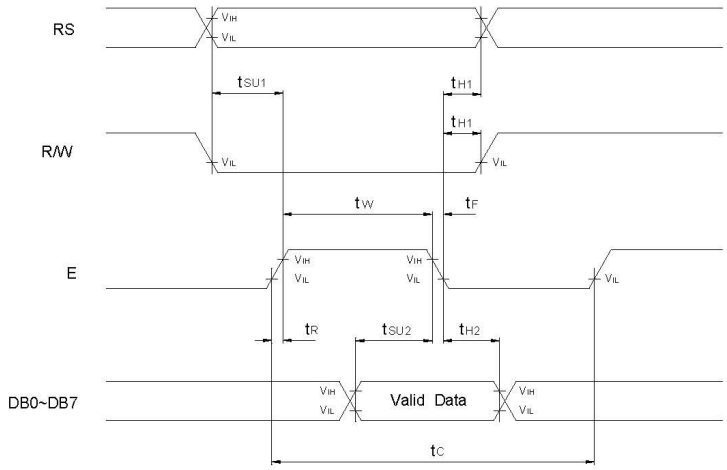


Fig.1 MPU Write Timing

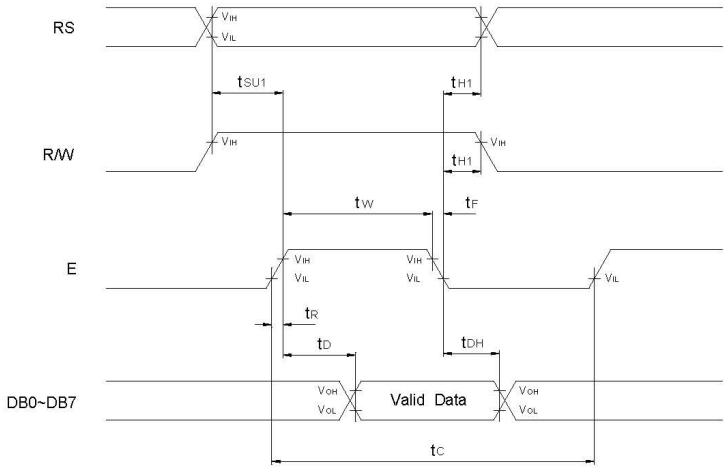
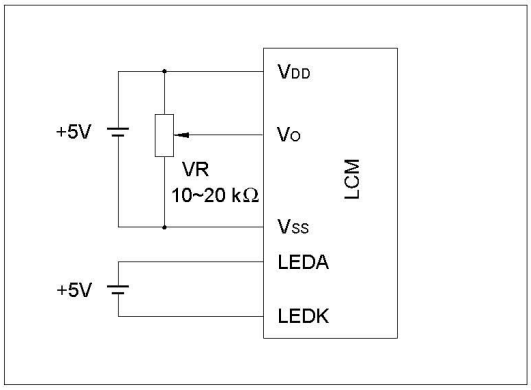


Fig.2 MPU Read Timing

3.3 LED Backlight Characteristics (Ta=25℃)

Item	Symbol	Condition	Min.	Typ.	Max.	UNIT
Forward Voltage	Vf		3.9	4.1	4.3	V
Forward Current	If	Vf=4.1V	--	110	--	mA
Peak Wave Length	λp	If=110mA	--	568	--	nm
Luminance	Lv	If=110mA	--	100	--	cd/m^2

3.4 Power Supply



5. MPU INTERFACE

5.1 General

(1). The LCD controller can be operated in either 4 or 8 bits mode. Instructions/Data are written to the display using the signal timing characteristics found in section 3.2.
When operating in 4-bit mode, data is transferred in two 4-bit operations using data bits DB4~DB7. DB0~DB3 are not used. When using 4-bit mode, data is transferred twice before the instruction cycle is complete. The higher order 4 bits (contents of DB4~DB7 when interface data is 8 bits long) is transferred first, then the lower order 4 bits (contents of DB0~DB3 when interface data is 8 bits long) is transferred. Check the busy flag after 4-bit data has been transferred twice (one instruction). A 4-bit two operation will then transfer the busy flag and address counter data.
(2). When operating in 8-bit mode, data is transferred using the full 8-bit bus DB0~DB7.

5.2 Initialization

5.2.1 Initialization by the Internal Reset Circuit

The display can be initialized using the internal reset circuit when the power is turned on. The following instructions are executed in initialization. The busy flag (BF) is kept in busy state until initialization ends. The busy flag will go active 10ms after Vcc rises to 4.5V.
(1). Display Clear
(2). Function set:
DL = 1 : 8 bit interface operation
N = 0 : 1 - line display
F = 0 : 5 x 7 dot character font
(3). Display ON/OFF Control:
D = 0 : Display OFF
C = 0 : Cursor OFF
B = 0 : Blink OFF
(4). Entry Mode Set
I/D = 1 : +1 (Increment Mode)
S = 0 : No Display Shift operation
If the internal power supply reset timing cannot be met (0.1ms<trcc<10ms), the internal reset circuit will not operate normally and initialization will not be performed. In this case, the display must be initialized by software.

5.2.2 Software Initialization

Although software initialization is not mandatory, it is recommended that this procedure always be performed. When the internal power supply reset timing cannot be met, then the display must be initialized using one of the following procedures.

(2). 4-Bit Initialization:

Power on

↓

Wait for more than 30ms after V_{DD} rises to 4.5V.

↓

Function Set									
RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	1	0	X	X	X	X
0	0	0	0	1	0	X	X	X	X
0	0	N	F	X	X	X	X	X	X

↓

Wait for more than 39 μ s

↓

Display ON/OFF Control									
RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	X	X	X	X
0	0	1	D	C	B	X	X	X	X

↓

Wait for more than 39 μ s

↓

Display Clear									
RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	X	X	X	X
0	0	0	0	0	1	X	X	X	X

↓

Wait for more than 1.53ms

↓

Entry Mode Set									
RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	X	X	X	X
0	0	0	1	I/D	SH	X	X	X	X

↓

End of initialization

6. DISPLAY CONTROL INSTRUCTION

Table 6.1 Instructions

Instruction	Instruction code										Description	Execution time (fosc=270KHz)
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear Display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DDRAM address to 00H.	1.53ms
Return Home	0	0	0	0	0	0	0	0	1	-	Sets DDRAM address to 00H in AC and returns shifted display to its original position. The contents of DDRAM remain unchanged.	1.53ms
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	SH	Sets cursor move direction and enable the shift of entire display. These operations are performed during data write and read.	39 μ s
Display ON/ OFF Control	0	0	0	0	0	0	1	D	C	B	Set ON/OFF of entire display (D), cursor ON/OFF(C), and blinking of cursor position character(B).	39 μ s
Cursor or Display Shift	0	0	0	0	0	1	S/C	R/L	-	-	Moves cursor and shifts display without changing DDRAM contents.	39 μ s
Function Set	0	0	0	0	1	DL	N	F	-	-	Sets interface data length (DL: 8-bit/4-bit), numbers of display line (N: 2-line/1-line), and display font type (F: 5x11dots/5x8dots)	39 μ s
Set CGRAM Address	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0	Set CGRAM address in address counter.	39 μ s
Set DDRAM Address	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Set DDRAM address in address Counter.	39 μ s
Read Busy Flag and Address	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Reads busy flag (BF) indicating internal operation is being performed and reads address counter contents.	0 μ s
Write data to CG or DD RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Write data into internal RAM (DDRAM/CGRAM).	43us
Read data from CG or DD RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Read data from internal RAM (DDRAM/CGRAM).	43us

"-" : don't care

Note: 1. Make sure to check the busy flag before sending the instruction to the display. If the busy flag is not checked, the time between first and next instruction must be longer than the instruction execution time list in the Table 6.1.

2. After execution of CG RAM/DD RAM data write or read instruction, the RAM address counter is increased or decreased by 1. The RAM address counter is updated after the busy flag turns off.

Sample SD card specifications (some relevant pages)

2.6. Physical Specifications

Refer to Table 2-6 and to Figures 2-1 through 2-3 for SD Card physical specifications and dimensions.

Table 2-6. Physical Specifications

Weight	2.0 g. maximum
Length:	32mm ± 0.1mm
Width:	24mm ± 0.1mm
Thickness:	2.1mm ± 0.15mm (in substrate area only, 2.25mm maximum)

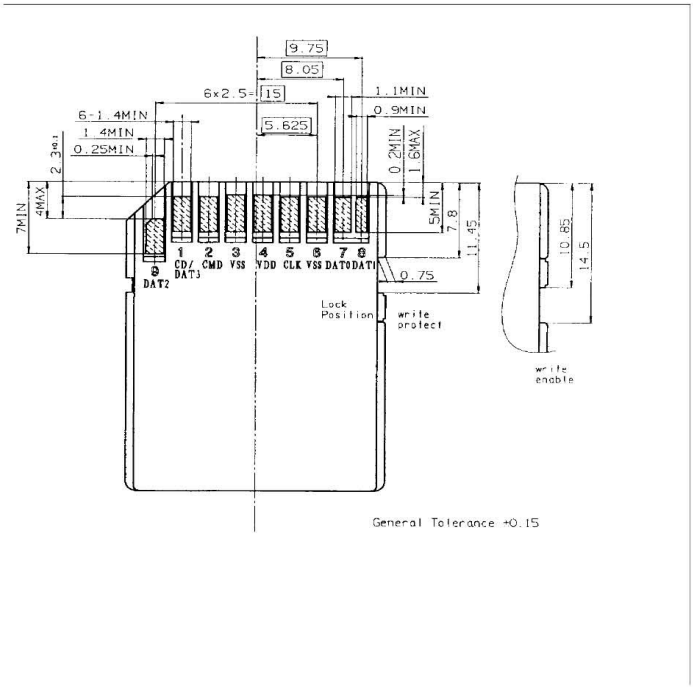


Figure 2-1. SD Card Dimensions

3.1.2. Pin Assignments in SPI Mode

Table 3-2 lists the pin assignments and definitions in SPI Mode.

Table 3-2. SPI Bus Mode Pad Definition

Pin #	Name	Type[1]	SPI Description
1	CS	I	Chip Select (Active low)
2	DataIn	I	Host to Card Commands and Data
3	VSS1	S	Supply Voltage Ground
4	VDD	S	Supply Voltage
5	CLK	I	Clock
6	VSS2	S	Supply Voltage Ground
7	DataOut	O	Card to Host Data and Status
8	RSV[2]	I	Reserved
9	RSV[2]	I	Reserved

NOTES: 1) S=power supply; I=input; O=output.
2) The 'RSV' pins are floating inputs. It is the responsibility of the host designer to connect external pullup resistors to those lines. Otherwise non-expected high current consumption may occur due to the floating inputs.

Each card has a set of information registers (refer to Table 3-3). Detailed descriptions are provided in Section 3.5.

Table 3-3. SD Card Registers

Name	Width	Description
CID	128	Card identification number: individual card number for identification.
RCA[1]	16	Relative card address: local system address of a card, dynamically suggested by the card and approved by the host during initialization.
CSD	128	Card specific data: information about the card operation conditions.
SCR	64	SD Configuration Register: information about the SD Card's special features capabilities.
OCR	32	Operation Condition Register

NOTE: 1) The RCA register is not available in SPI Mode.

The host may reset the cards by switching the power supply off and on again. The card has its own power-on detection circuitry which puts the card into an idle state after the power-on. The card can also be reset by sending the GO_IDLE (CMD0) command.

3.4.2. Bus Operating Conditions

SPI Mode bus operating conditions are identical to SD Card mode bus operating conditions. Table 3-4 lists the power supply voltages. The CS (chip select) signal timing is identical to the input signal timing (see Figure 3-7).

Table 3-4. Power Supply Voltage

General					
Parameter	**Symbol**	**Min.**	**Max.**	**Unit**	**Remark**
Peak voltage on all lines		-0.3	VDD+0.3	V	
All Inputs					
Input Leakage Current		-10	10	µA	
All Outputs					
Output Leakage Current		-10	10	µA	
Power Supply Voltage					
Parameter	**Symbol**	**Min.**	**Max.**	**Unit**	**Remark**
Supply Voltage	V_{DD}	2.0	3.6	V	CMD0, 15, 55, ACMD41 commands
Supply Voltage		2.7	3.6	V	Except CMD0, 15, 55, ACMD41 commands
Supply voltage differentials (V_{SS1}, V_{SS2})		-0.3	0.3	V	
Power up Time			250	mS	From 0V to V_{DD} Min.

3.4.3. Bus Signal Line Load

The total capacitance CL of the CLK line of the SD Card bus is the sum of the bus master capacitance CHOST, the bus capacitance CBUS itself and the capacitance CCARD of each card connected to this line:

$$CL = CHOST + CBUS + N*CCARD$$

Where N is the number of connected cards. Requiring the sum of the host and bus capacitances not to exceed 30 pF for up to 10 cards, and 40 pF for up to 30 cards, the values in Table 3-5must not be exceeded.

Table 3-5. Signal Line's Load

Parameter	Symbol	Min.	Max.	Unit	Remark
Pull-up resistance	R_{CMD} R_{DAT}	10	100	kΩ	To prevent bus floating
Bus signal line capacitance	C_L		250	pF	$f_{PP} \leq 5$ MHz, 21 cards
Bus signal line capacitance	C_L		100	pF	$f_{PP} \leq 20$ MHz, 7 cards
Single card capacitance	C_{CARD}		10	pF	
Maximum signal line inductance			16	nH	$f_{PP} \leq 20$ MHz
Pull-up resistance inside card (pin 1)	R_{DAT3}	10	90	kΩ	May be used for card detection

Datasheet of MAX6675 TDC

19-2235; Rev 1; 3/02

MAXIM

Cold-Junction-Compensated K-Thermocouple-to-Digital Converter (0°C to +1024°C)

MAX6675

General Description

The MAX6675 performs cold-junction compensation and digitizes the signal from a type-K thermocouple. The data is output in a 12-bit resolution, SPI™-compatible, read-only format.

This converter resolves temperatures to 0.25°C, allows readings as high as +1024°C, and exhibits thermocouple accuracy of 8LSBs for temperatures ranging from 0°C to +700°C.

The MAX6675 is available in a small, 8-pin SO package.

Features

- ♦ Direct Digital Conversion of Type -K Thermocouple Output
- ♦ Cold-Junction Compensation
- ♦ Simple SPI-Compatible Serial Interface
- ♦ 12-Bit, 0.25°C Resolution
- ♦ Open Thermocouple Detection

Ordering Information

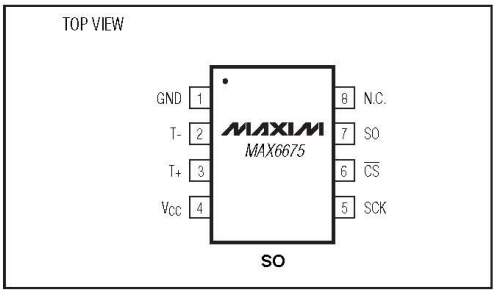
PART	TEMP RANGE	PIN-PACKAGE
MAX6675ISA	-20°C to +85°C	8 SO

Applications

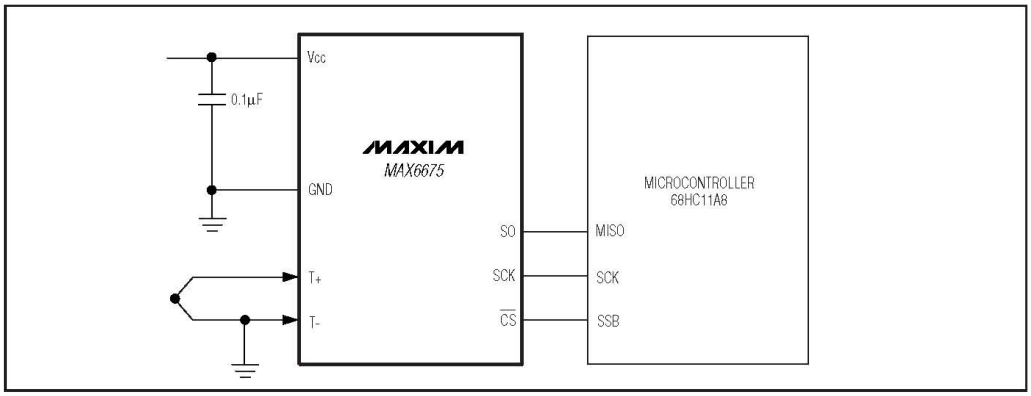
Industrial
Appliances
HVAC
Automotive

SPI is a trademark of Motorola, Inc.

Pin Configuration



Typical Application Circuit



Cold-Junction-Compensated K-Thermocouple-to-Digital Converter (0°C to +1024°C)

ABSOLUTE MAXIMUM RATINGS

Supply Voltage (V_{CC} to GND) ..... -0.3V to +6V
SO, SCK, $\overline{CS}$, T-, T+ to GND .....-0.3V to V_{CC} + 0.3V
SO Current 50mA
ESD Protection (Human Body Model) ..... ±2000V
Continuous Power Dissipation (T_A = +70°C)
8-Pin SO (derate 5.88mW/°C above +70°C) 471mW
Operating Temperature Range .....-20°C to +85°C
Storage Temperature Range .....-65°C to +150°C
Junction Temperature ..... +150°C
SO Package
Vapor Phase (60s) .....+215°C
Infrared (15s) .....+220°C
Lead Temperature (soldering, 10s) ..... +300°C

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of the specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

ELECTRICAL CHARACTERISTICS

(V_{CC} = +3.0V to +5.5V, T_A = -20°C to +85°C, unless otherwise noted. Typical values specified at +25°C.) (Note 1)

PARAMETER	SYMBOL	CONDITIONS		MIN	TYP	MAX	UNITS
Temperature Error		$T_{THERMOCOUPLE}$ = +700°C, T_A = +25°C (Note 2)	V_{CC} = +3.3V	-5		+5	LSB
			V_{CC} = +5V	-6		+6	
		$T_{THERMOCOUPLE}$ = 0°C to +700°C, T_A = +25°C (Note 2)	V_{CC} = +3.3V	-8		+8	
			V_{CC} = +5V	-9		+9	
		$T_{THERMOCOUPLE}$ = +700°C to +1000°C, T_A = +25°C (Note 2)	V_{CC} = +3.3V	-17		+17	
			V_{CC} = +5V	-19		+19	
Thermocouple Conversion Constant					10.25		µV/LSB
Cold-Junction Compensation Error		T_A = -20°C to +85°C (Note 2)	V_{CC} = +3.3V	-3.0		+3.0	°C
			V_{CC} = +5V	-3.0		+3.0	
Resolution					0.25		°C
Thermocouple Input Impedance					60		kΩ
Supply Voltage	V_{CC}			3.0		5.5	V
Supply Current	I_{CC}				0.7	1.5	mA
Power-On Reset Threshold		V_{CC} rising		1	2	2.5	V
Power-On Reset Hysteresis					50		mV
Conversion Time		(Note 2)			0.17	0.22	s
SERIAL INTERFACE							
Input Low Voltage	V_{IL}					0.3 x V_{CC}	V
Input High Voltage	V_{IH}			0.7 x V_{CC}			V
Input Leakage Current	I_{LEAK}	V_{IN} = GND or V_{CC}				±5	µA
Input Capacitance	C_{IN}				5		pF

Cold-Junction-Compensated K-Thermocouple-to-Digital Converter (0°C to +1024°C)

ELECTRICAL CHARACTERISTICS (continued)

(V_{CC} = +3.0V to +5.5V, T_A = -20°C to +85°C, unless otherwise noted. Typical values specified at +25°C.) (Note 1)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
Output High Voltage	V_{OH}	I_{SOURCE} = 1.6mA	V_{CC} - 0.4			V
Output Low Voltage	V_{OL}	I_{SINK} = 1.6mA			0.4	V
TIMING						
Serial Clock Frequency	f_{SCL}				4.3	MHz
SCK Pulse High Width	t_{CH}		100			ns
SCK Pulse Low Width	t_{CL}		100			ns
CSB Fall to SCK Rise	t_{CSS}	C_L = 10pF	100			ns
CSB Fall to Output Enable	t_{DV}	C_L = 10pF			100	ns
CSB Rise to Output Disable	t_{TR}	C_L = 10pF			100	ns
SCK Fall to Output Data Valid	t_{DO}	C_L = 10pF			100	ns

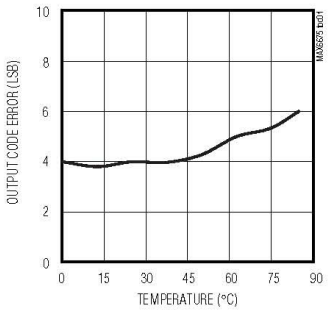
Note 1: All specifications are 100% tested at T_A = +25°C. Specification limits over temperature (T_A = T_{MIN} to T_{MAX}) are guaranteed by design and characterization, not production tested.

Note 2: Guaranteed by design. Not production tested.

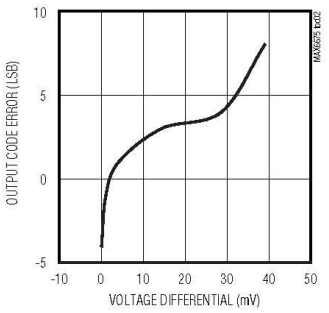
Typical Operating Characteristics

(V_{CC} = +3.3V, T_A = +25°C, unless otherwise noted.)

OUTPUT CODE ERROR vs. AMBIENT TEMPERATURE



OUTPUT CODE ERROR vs. VOLTAGE DIFFERENTIAL



Cold-Junction-Compensated K-Thermocouple-to-Digital Converter (0°C to +1024°C)

Pin Description

PIN	NAME	FUNCTION
1	GND	Ground
2	T-	Alumel Lead of Type-K Thermocouple. Should be connected to ground externally.
3	T+	Chromel Lead of Type-K Thermocouple
4	VCC	Positive Supply. Bypass with a 0.1µF capacitor to GND.
5	SCK	Serial Clock Input
6	$\overline{CS}$	Chip Select. Set $\overline{CS}$ low to enable the serial interface.
7	SO	Serial Data Output
8	N.C.	No Connection

Detailed Description

The MAX6675 is a sophisticated thermocouple-to-digital converter with a built-in 12-bit analog-to-digital converter (ADC). The MAX6675 also contains cold-junction compensation sensing and correction, a digital controller, an SPI-compatible interface, and associated control logic.

The MAX6675 is designed to work in conjunction with an external microcontroller (µC) or other intelligence in thermostatic, process-control, or monitoring applications.

Temperature Conversion

The MAX6675 includes signal-conditioning hardware to convert the thermocouple's signal into a voltage compatible with the input channels of the ADC. The T+ and T- inputs connect to internal circuitry that reduces the introduction of noise errors from the thermocouple wires.

Before converting the thermoelectric voltages into equivalent temperature values, it is necessary to compensate for the difference between the thermocouple cold-junction side (MAX6675 ambient temperature) and a 0°C virtual reference. For a type-K thermocouple, the voltage changes by 41µV/°C, which approximates the thermocouple characteristic with the following linear equation:

$$V_{OUT} = (41\mu V / °C) \times (T_R - T_{AMB})$$

Where:

V_{OUT} is the thermocouple output voltage (µV).

T_R is the temperature of the remote thermocouple junction (°C).

T_{AMB} is the ambient temperature (°C).

Cold-Junction Compensation

The function of the thermocouple is to sense a difference in temperature between two ends of the thermocouple wires. The thermocouple's hot junction can be read from 0°C to +1023.75°C. The cold end (ambient temperature of the board on which the MAX6675 is mounted) can only range from -20°C to +85°C. While the temperature at the cold end fluctuates, the MAX6675 continues to accurately sense the temperature difference at the opposite end.

The MAX6675 senses and corrects for the changes in the ambient temperature with cold-junction compensation. The device converts the ambient temperature reading into a voltage using a temperature-sensing diode. To make the actual thermocouple temperature measurement, the MAX6675 measures the voltage from the thermocouple's output and from the sensing diode. The device's internal circuitry passes the diode's voltage (sensing ambient temperature) and thermocouple voltage (sensing remote temperature minus ambient temperature) to the conversion function stored in the ADC to calculate the thermocouple's hot-junction temperature.

Optimal performance from the MAX6675 is achieved when the thermocouple cold junction and the MAX6675 are at the same temperature. Avoid placing heat-generating devices or components near the MAX6675 because this may produce cold-junction-related errors.

Digitization

The ADC adds the cold-junction diode measurement with the amplified thermocouple voltage and reads out the 12-bit result onto the SO pin. A sequence of all zeros means the thermocouple reading is 0°C. A sequence of all ones means the thermocouple reading is +1023.75°C.

Cold-Junction-Compensated K-Thermocouple-to-Digital Converter (0°C to +1024°C)

Applications Information

Serial Interface

The *Typical Application Circuit* shows the MAX6675 interfaced with a microcontroller. In this example, the MAX6675 processes the reading from the thermocouple and transmits the data through a serial interface. Force $\overline{CS}$ low and apply a clock signal at SCK to read the results at SO. Forcing $\overline{CS}$ low immediately stops any conversion process. Initiate a new conversion process by forcing $\overline{CS}$ high.

Force $\overline{CS}$ low to output the first bit on the SO pin. A complete serial interface read requires 16 clock cycles. Read the 16 output bits on the falling edge of the clock. The first bit, D15, is a dummy sign bit and is always zero. Bits D14–D3 contain the converted temperature in the order of MSB to LSB. Bit D2 is normally low and goes high when the thermocouple input is open. D1 is low to provide a device ID for the MAX6675 and bit D0 is three-state.

Figure 1a is the serial interface protocol and Figure 1b shows the serial interface timing. Figure 2 is the SO output.

Open Thermocouple

Bit D2 is normally low and goes high if the thermocouple input is open. In order to allow the operation of the open thermocouple detector, T- must be grounded. Make the ground connection as close to the GND pin as possible.

Noise Considerations

The accuracy of the MAX6675 is susceptible to power-supply coupled noise. The effects of power-supply noise can be minimized by placing a 0.1µF ceramic bypass capacitor close to the supply pin of the device.

Thermal Considerations

Self-heating degrades the temperature measurement accuracy of the MAX6675 in some applications. The magnitude of the temperature errors depends on the thermal conductivity of the MAX6675 package, the mounting technique, and the effects of airflow. Use a large ground plane to improve the temperature measurement accuracy of the MAX6675.

The accuracy of a thermocouple system can also be improved by following these precautions:

- Use the largest wire possible that does not shunt heat away from the measurement area.
- If small wire is required, use it only in the region of the measurement and use extension wire for the region with no temperature gradient.
- Avoid mechanical stress and vibration, which could strain the wires.
- When using long thermocouple wires, use a twisted-pair extension wire.
- Avoid steep temperature gradients.
- Try to use the thermocouple wire well within its temperature rating.
- Use the proper sheathing material in hostile environments to protect the thermocouple wire.
- Use extension wire only at low temperatures and only in regions of small gradients.
- Keep an event log and a continuous record of thermocouple resistance.

Reducing Effects of Pick-Up Noise

The input amplifier (A1) is a low-noise amplifier designed to enable high-precision input sensing. Keep the thermocouple and connecting wires away from electrical noise sources.

Chip Information

TRANSISTOR COUNT: 6720

PROCESS: BiCMOS

Cold-Junction-Compensated K-Thermocouple-to-Digital Converter (0°C to +1024°C)

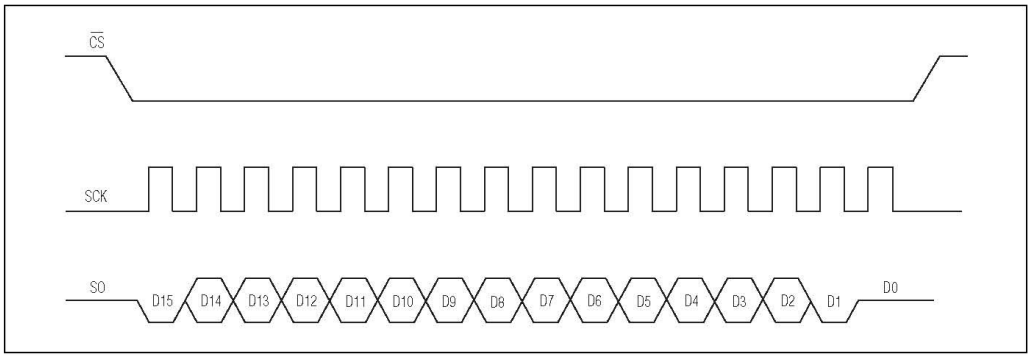


Figure 1a. Serial Interface Protocol

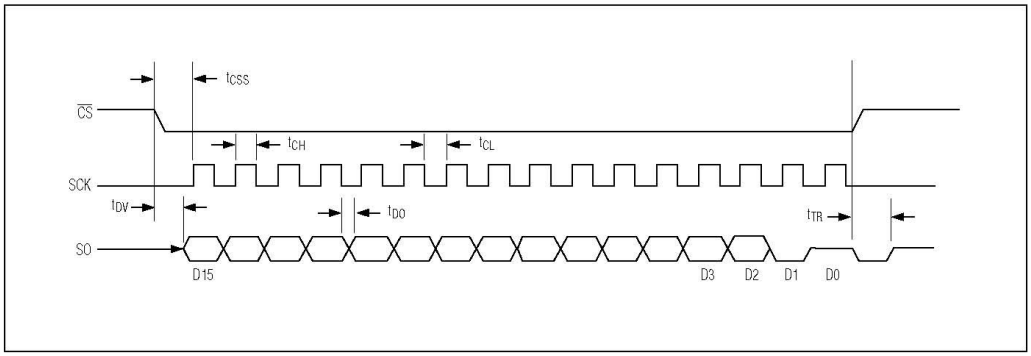


Figure 1b. Serial Interface Timing

BIT	DUMMY SIGN BIT	12-BIT TEMPERATURE READING												THERMOCOUPLE INPUT	DEVICE ID	STATE
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	MSB											LSB		0	Three-state

Figure 2. SO Output

Datasheet for DG406 and DG407 multiplexers

19-4729; Rev 5; 8/04

MAXIM

Improved, 16-Channel/Dual 8-Channel, CMOS Analog Multiplexers

DG406/DG407

General Description

Maxim's redesigned DG406 and DG407 CMOS analog multiplexers now feature guaranteed matching between channels (8Ω, max) and flatness over the specified signal range (9Ω, max). These low on-resistance muxes (100Ω, max) conduct equally well in either direction and feature guaranteed low charge injection (15pC, max). In addition, these new muxes offer low input off-leakage current over temperature—less than 5nA at +85°C.

The DG406 is a 1 of 16 multiplexer/demultiplexer and the DG407 is a dual 8-channel multiplexer/demultiplexer. Both muxes operate with a +4.5V to +30V single supply and with ±4.5V to ±20V dual supplies. ESD protection is guaranteed to be greater than 2000V per Method 3015.7 of MIL-STD 883. These improved muxes are pin-compatible plug-in upgrades for the industry standard DG406 and DG407.

Applications

Sample-and-Hold Circuits
Test Equipment
Guidance and Control Systems
Communications Systems
Data-Acquisition Systems
Audio Signal Routing

Features

- Pin-Compatible Plug-In Upgrade for Industry Standard DG406/DG407
- Guaranteed Matching Between Channels, 8Ω (max)
- Guaranteed On-Resistance Flatness, 9Ω (max)
- Guaranteed Low Charge Injection, 15pC (max)
- Low On-Resistance 100Ω (max)
- Input Leakage, 5nA (max) at +85°C
- Low Power Consumption, 1.25mW (max)
- Rail-to-Rail Signal Handling
- Digital Input Controls TTL/CMOS Compatible
- ESD Protection >2000V per Method 3015.7

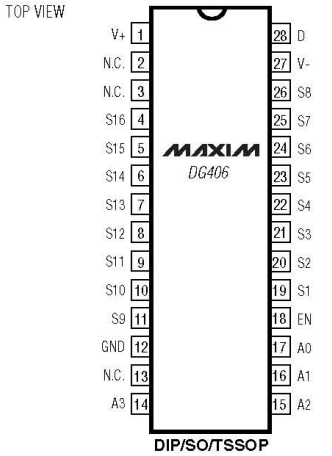
Ordering Information

PART	TEMP RANGE	PIN-PACKAGE
DG406CJ	0°C to +70°C	28 Plastic DIP
DG406CWI	0°C to +70°C	28 Wide SO
DG406C/D	0°C to +70°C	Dice*
DG406DJ	-40°C to +85°C	28 Plastic DIP
DG406EWI	-40°C to +85°C	28 Wide SO
DG406DN	-40°C to +85°C	28 PLCC
DG406AK	-55°C to +125°C	28 CERDIP
DG406EUI	-40°C to +85°C	28 TSSOP

Ordering Information continued at end of data sheet.

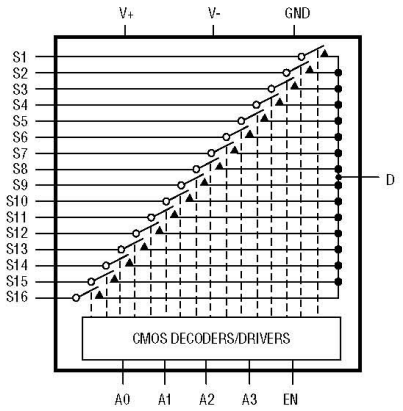
** Contact factory for dice specifications.*

Pin Configurations



Pin Configurations continued at end of data sheet.

Functional Diagrams



DG406 16-CHANNEL SINGLE-ENDED MULTIPLEXER

Functional Diagrams continued at end of data sheet.

Improved, 16-Channel/Dual 8-Channel, CMOS Analog Multiplexers

ABSOLUTE MAXIMUM RATINGS

(Voltage Referenced to V-)
V+ .....-0.3V, 44V
GND .....-0.3V, 25V
Digital Inputs, S, D (Note 1).....(V- - 2V) to (V+ + 2V) or 30mA (whichever occurs first)
Continuous Current (any terminal).....30mA
Peak Current, S or D
(pulsed at 1ms, 10% duty cycle max).....100mA
Continuous Power Dissipation (T_A = +70°C)
28-Pin Plastic DIP (derate 9.09mW/°C above +70°C)..727mW
28-Pin Wide SO (derate 12.50mW/°C above +70°C)...1000mW
28-Pin PLCC (derate 10.53mW/°C above +70°C).....842mW
28-Pin CERDIP (derate 16.67mW/°C above +70°C)...1333mW
28-Pin TSSOP (derate 12.8mW/°C above +70°C).....1025mW
Operating Temperature Ranges
DG406/DG407C_ .....0°C to +70°C
DG406/DG407D_ .....-40°C to +85°C
DG406/DG407AK .....-55°C to +125°C
Storage Temperature Range .....-65°C to +150°C
Lead Temperature (soldering, 10s).....+300°C

Note 1: Signals on S_, D_, A0, A1, A2, A3, or EN exceeding V+ or V- are clamped by internal diodes. Limit forward current to maximum current ratings.

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of the specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

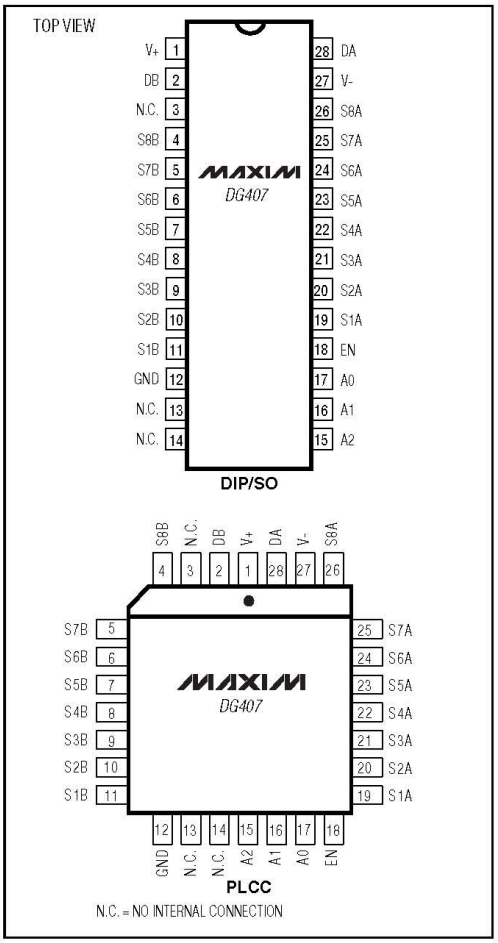
ELECTRICAL CHARACTERISTICS—Dual Supplies

(V+ = 15V, V- = -15V, GND = 0V, V_{AH} = +2.4V, V_{AL} = +0.8V, T_A = T_{MIN} to T_{MAX}, unless otherwise noted.)

PARAMETER	SYMBOL	CONDITIONS				MIN	TYP (Note 2)	MAX	UNITS
SWITCH									
Analog Signal Range	V_{ANALOG}	(Note 3)				-15		+15	V
Drain-Source On-Resistance	$R_{DS(ON)}$	I_S = -1.0mA, V_D = ±10V		T_A = +25°C			60	100	Ω
				T_A = T_{MIN} to T_{MAX}				125	
On-Resistance Matching Between Channels	$\Delta R_{DS(ON)}$	I_S = -1.0mA, V_D = ±10V (Note 4)		T_A = +25°C			1.5	8	Ω
				T_A = T_{MIN} to T_{MAX}				10	
On-Resistance Flatness	R_{FLAT}	I_S = -1.0mA, V_D = ±5V or 0V		T_A = +25°C			1.8	9	Ω
				T_A = T_{MIN} to T_{MAX}				12	
Source-Off Leakage Current (Note 5)	$I_{S(OFF)}$	V_D = +10V, V_S = ±10V, V_{EN} = 0V		T_A = +25°C		-0.5	+0.01	+0.5	nA
				T_A = T_{MIN} to T_{MAX}	C, D	-5		+5	
					A	-50		+50	
Drain-Off Leakage Current (Note 5)	$I_{D(OFF)}$	V_D = ±10V, V_S = +10V, V_{EN} = 0V	DG406	T_A = +25°C		-1	+0.02	+1	nA
				T_A = T_{MIN} to T_{MAX}	C, D	-40		+40	
					A	-200		+200	
		V_D = +10V, V_S = ±10V, V_{EN} = 0V	DG407	T_A = +25°C		-1	+0.02	+1	
				T_A = T_{MIN} to T_{MAX}	C, D	-20		+20	
					A	-100		+100	
Drain-On Leakage Current (Note 5)	$I_{D(ON)}$ + $I_{S(ON)}$	V_D = ±10V, V_S = ±10V, sequence each switch on	DG406	T_A = +25°C		-1	+0.02	+1	nA
				T_A = T_{MIN} to T_{MAX}	C, D	-40		+40	
					A	-200		+200	
			DG407	T_A = +25°C		-1	+0.02	+1	
				T_A = T_{MIN} to T_{MAX}	C, D	-20		+20	
					A	-100		+100	

Improved, 16-Channel/Dual 8-Channel, CMOS Analog Multiplexers

Pin Configurations/Functional Diagrams/Truth Tables (continued)



A3	A2	A1	A0	EN	ON Switch
X	X	X	X	0	None
0	0	0	0	1	1
0	0	0	1	1	2
0	0	1	0	1	3
0	0	1	1	1	4
0	1	0	0	1	5
0	1	0	1	1	6
0	1	1	0	1	7
0	1	1	1	1	8
1	0	0	0	1	9
1	0	0	1	1	10
1	0	1	0	1	11
1	0	1	1	1	12
1	1	0	0	1	13
1	1	0	1	1	14
1	1	1	0	1	15
1	1	1	1	1	16

DG406

LOGIC "0" $V_{AL} \leq 0.8V$, LOGIC "1" = $V_{AH} \geq 2.4V$

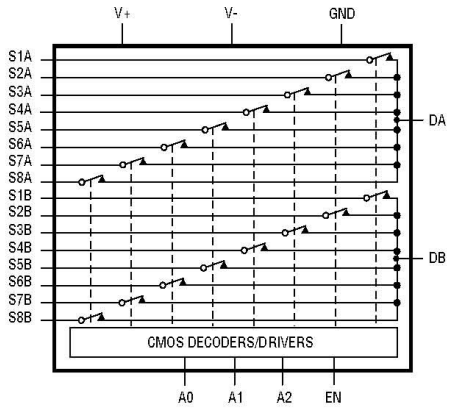
A2	A1	A0	EN	ON Switch
X	X	X	0	None
0	0	0	1	1
0	0	1	1	2
0	1	0	1	3
0	1	1	1	4
1	0	0	1	5
1	0	1	1	6
1	1	0	1	7
1	1	1	1	8

DG407

LOGIC "0" $V_{AL} \leq 0.8V$, LOGIC "1" = $V_{AH} \geq 2.4V$

Improved, 16-Channel/Dual 8-Channel, High-Performance, CMOS Analog Multiplexers

_Functional Diagrams (continued)



DG407 8-CHANNEL DIFFERENTIAL MULTIPLEXER

_Ordering Information (continued)

PART	TEMP RANGE	PIN-PACKAGE
DG407CJ	0°C to +70°C	28 Plastic DIP
DG407CWI	0°C to +70°C	28 Wide SO
DG407C/D	0°C to +70°C	Dice*
DG407DJ	-40°C to +85°C	28 Plastic DIP
DG407EWI	-40°C to +85°C	28 Wide SO
DG407DN	-40°C to +85°C	28 PLCC
DG407AK	-55°C to +125°C	28 CERDIP
DG407EUI	-40°C to +85°C	28 TSSOP

** Contact factory for dice specifications.*

Datasheet of Calfo AF Heat Transfer Fluid

Thermal Data

PROPERTY	TEMPERATURE			
	15°C (59°F)	38°C (100°F)	260°C (500°F)	316°C (600°F)
Density, kg/L (lb/ft^3)	0.867 (54.1)	0.852 (53.2)	0.715 (44.7)	0.681 (42.5)
Thermal Conductivity, W/m K (BTU/hr.°F.ft)	0.142 (0.082)	0.141 (0.082)	0.130 (0.075)	0.127 (0.073)
Heat Capacity, kJ/kg K (BTU/lb. °F)	1.89 (0.45)	1.96 (0.47)	2.69 (0.64)	2.88 (0.69)
Vapour Pressure, kPa (psia)	0.00 (0.00)	0.00 (0.00)	3.78 (0.55)	15.32 (2.20)

For detailed heat transfer calculations please refer to our Engineering Assistant software which is available at no cost from your Petro-Canada representative.

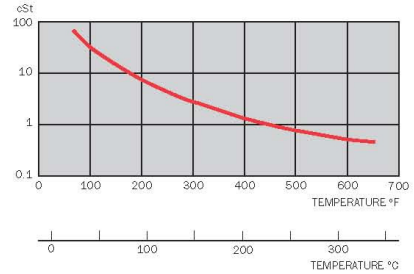
Typical Performance Data

PROPERTY	TEST METHOD	RESULTS
Colour	ASTM D1500	<0.5
Pour Point, °C (°F)	ASTM D97	-42 (-44)
Flash Point, COC, ° C (°F)	ASTM D92	225 (437)
Fire Point, °C (°F)	ASTM D92	240 (464)
Autoignition Temperature, °C (°F)	ASTM E659	343 (649)
Viscosity, cSt at 40°C (104°F) cSt at 100°C (212°F) cSt at 316°C (600°F)	ASTM D445	32.1 5.4 0.7
Average Molecular Weight		371
Neutralization Value, TAN, mg KOH/g	ASTM D664	< 0.1
Sulfur by XRF, wt%	ASTM D4294	< 0.0001
Conradson Carbon Residue, wt %	ASTM D189	0.01
Coefficient of Thermal Expansion, %/°C (%/°F)		0.0907 (0.0504)
Distillation Range, °C (°F) 10% 50% 90%	ASTM D2887	 365 (689) 417 (783) 475 (887)

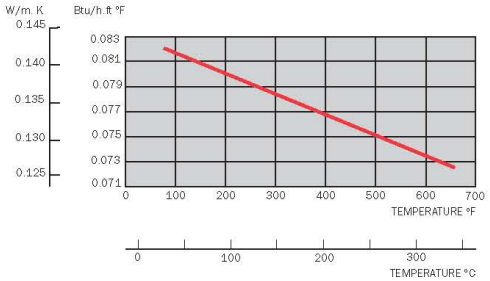
The values quoted above are typical of normal production. They do not constitute a specification.

*non-toxic defined as non-controlled under WHIMIS, non-hazardous under OSHA and non-dangerous under EUDPD.
**Any transport and disposal practice must be in compliance with federal, state, provincial and/or local laws and regulations.

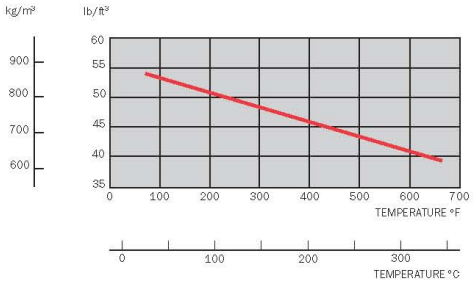
CALFLO AF VISCOSITY



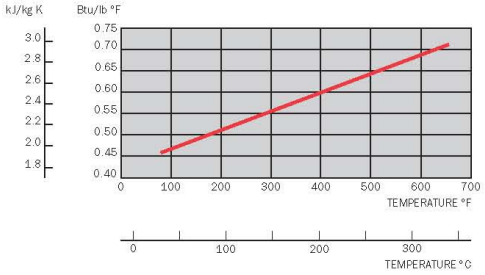
CALFLO AF THERMAL CONDUCTIVITY



CALFLO AF DENSITY



CALFLO AF HEAT CAPACITY



Health and Safety

To obtain Material Safety Data Sheet (MSDS), contact one of Petro-Canada's TechData Info Lines.

TechData Info Lines

TechData sheets for Petro-Canada's Cleaning Fluid and Flushing Fluid and Technical Bulletins regarding guidelines for system cleaning, flushing and change-out are also available. If you would like to know more about Petro-Canada's CALFLO™ AF Heat Transfer Fluid, or any other product in our complete line of quality lubricants, please contact us at:

Lubricants Head Office
Petro-Canada
2310 Lakeshore Road West
Mississauga, Ontario
Canada L5J 1K2

CERTIFIED
ISO 9001
ISO 14001
ISO/TS 16949

Canada - West . . . . . Phone 1-800-661-1199
- East (English) . . . . Phone 1-800-268-5850
(French) . . . . . Phone 1-800-576-1686
Other Areas . . . . . Phone (416) 730-2408
E-mail . . . . . lubecsr@petro-canada.ca
Internet . . . . . lubricants.petro-canada.ca

Petro-Canada Europe Lubricants
The Manor, Haseley Business Centre
Warwick, Warwickshire
CV35 7LS
United Kingdom
Phone . . . . . +44 (0) 2476-247294
Fax . . . . . +44 (0) 2476-247295

Petro-Canada America Lubricants
980 North Michigan Avenue
Suite 1400, #1431
Chicago, Illinois
USA 60611
Phone 1-888-284-4572
Fax . . . . . (708) 246-8994
E-mail . . . . . email@petro-canadaamerica.com

IM-7852E (06.08)

Beyond today's standards. PETRO-CANADA

Appendix E – Sample photographs of system components

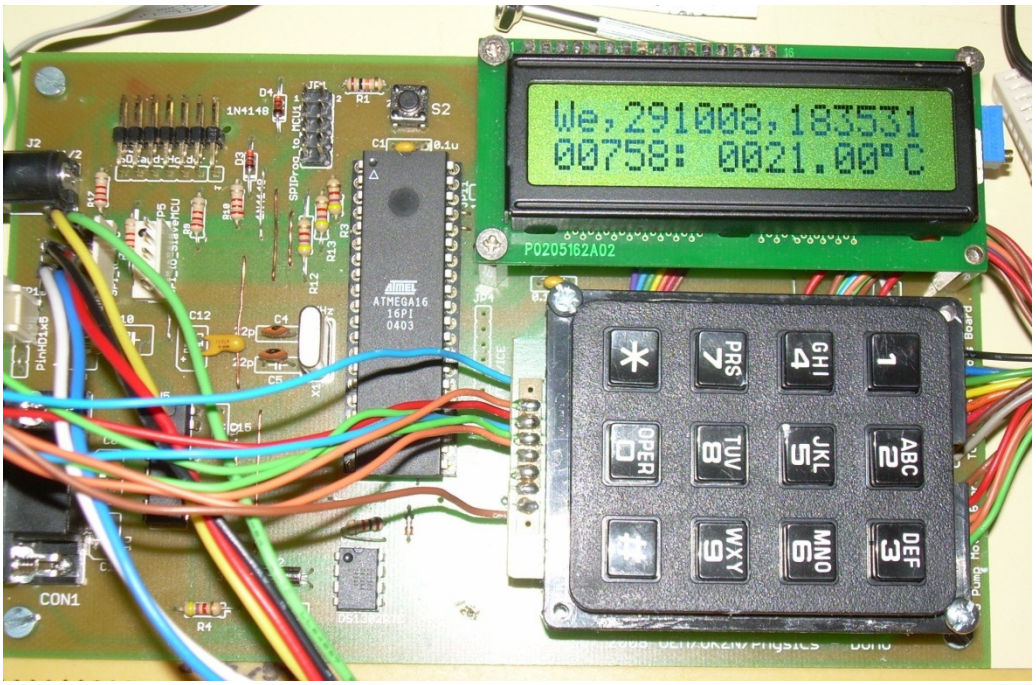


Figure 101 - Photograph of the main board's first prototype, showing RTC time (day, date, hours) and ambient temperature

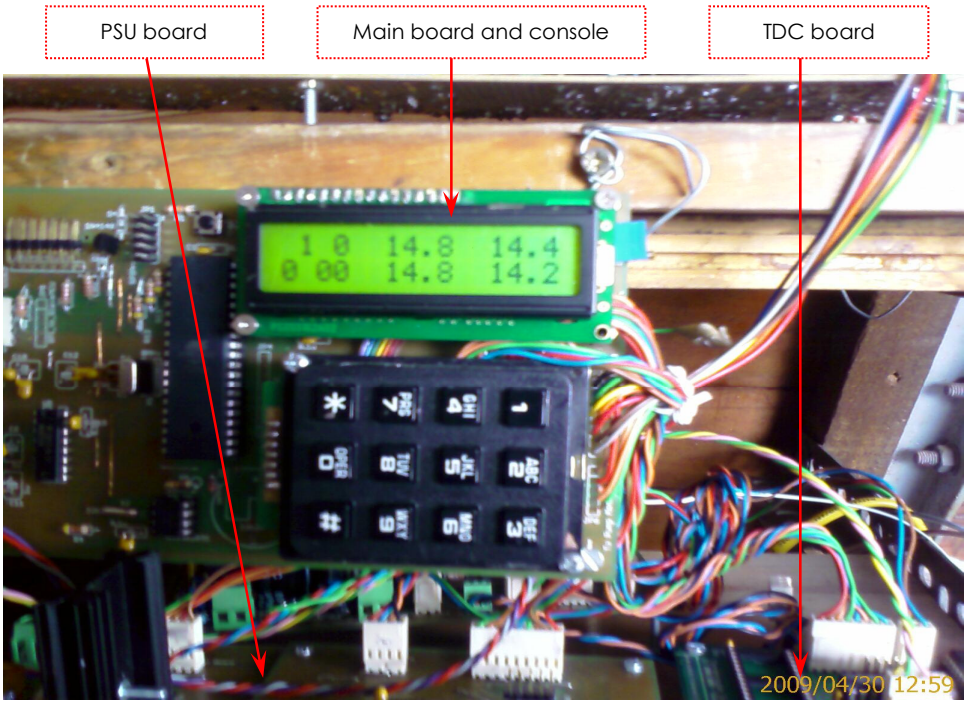


Figure 102 - Photograph of the integrated system on the plant, showing tracking state information.

Figure 103 - Photograph of the 2 very first PCB prototype boards: the main (upper) board and the TDC board, being assembled

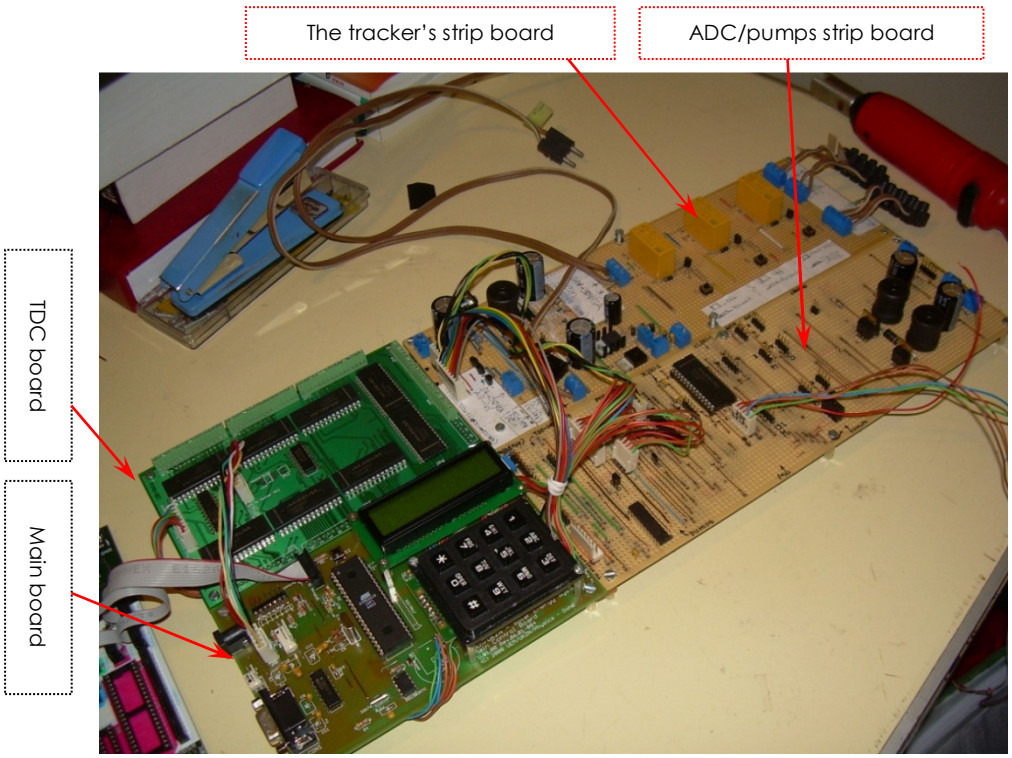


Figure 104 - Photograph showing the 2 PCBs in Figure 103, connected to stripboard prototypes (tracker's and ADC/pumps')

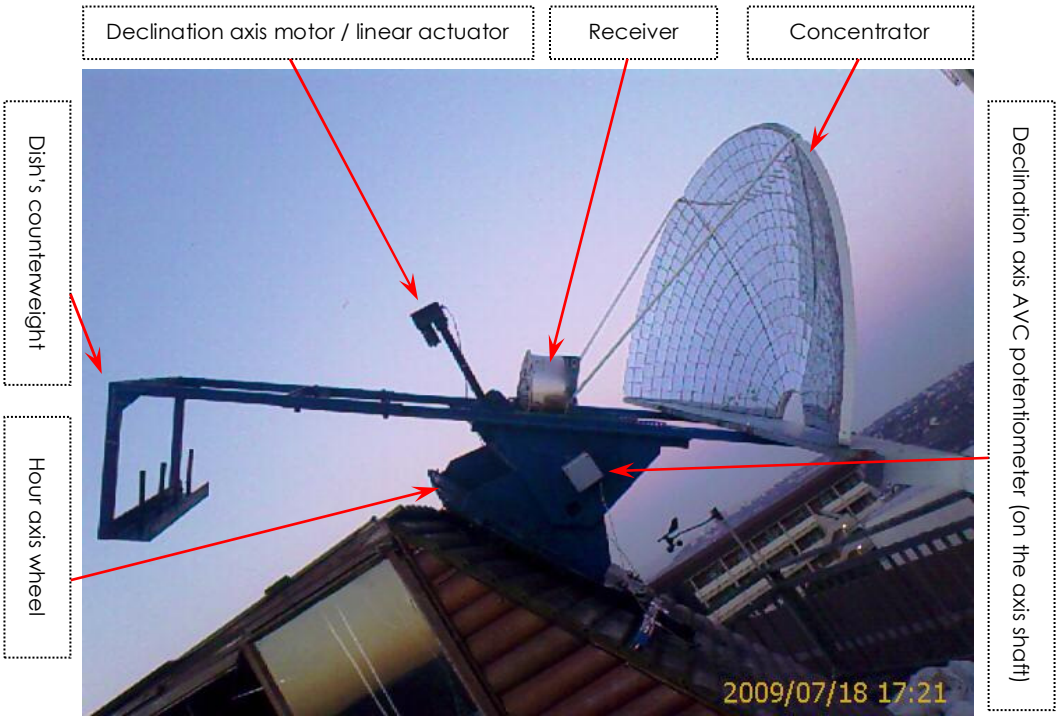


Figure 105 – A detail of the tracking assembly where various components are distinguishable. See also Figure 107.

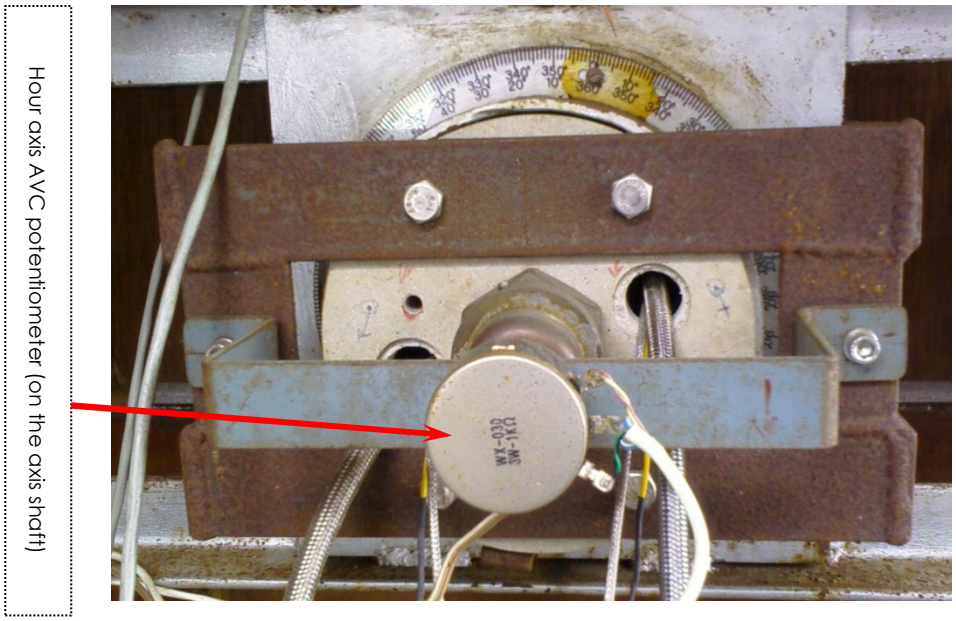


Figure 106 - Detail of the hour axis tail, showimng the AVC potentiometer for the actual hour angle

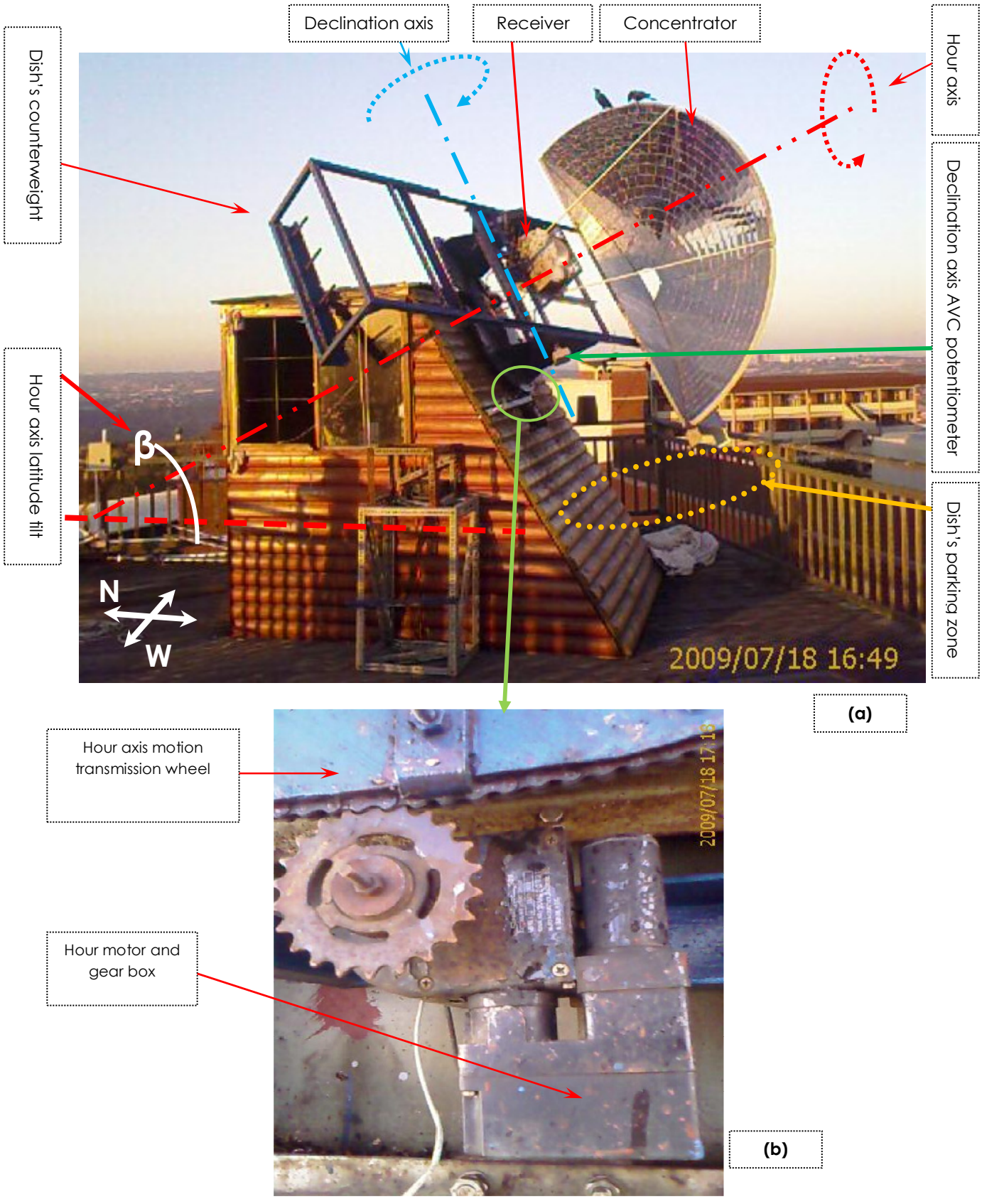


Figure 107 - (a) Various details of the ST-KZN solar thermal energy system. (b) A detail of the hour axis motor driver.

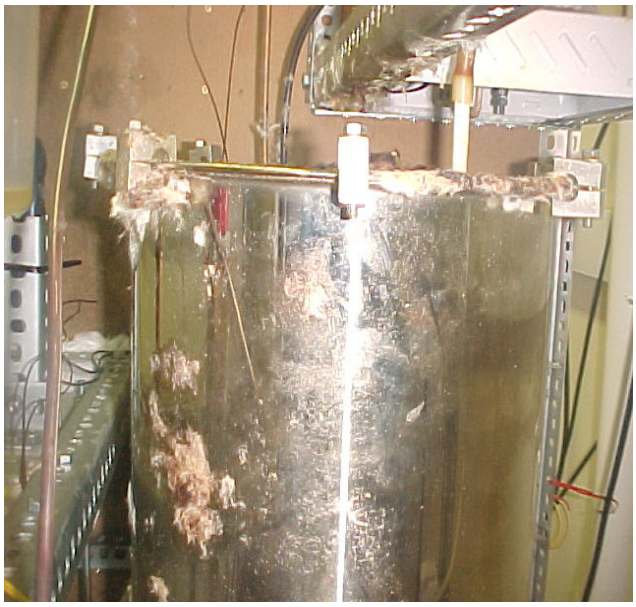


Figure 109 - A detail of the rock bed heat storage before it was insulated.

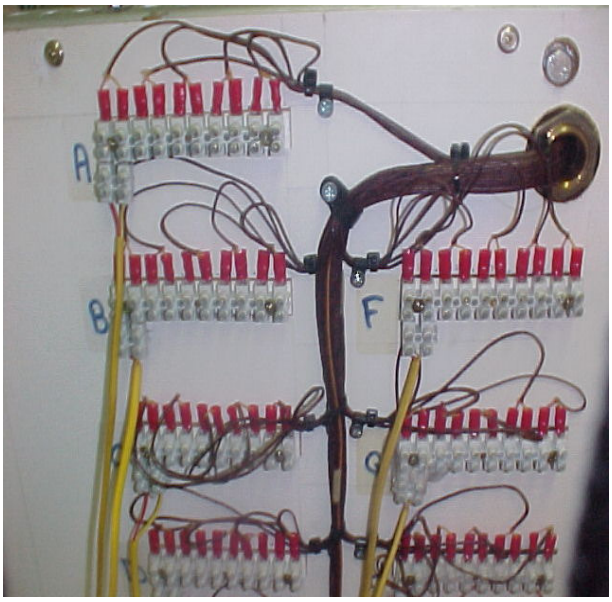


Figure 108 - Thermocouple sensor connectors for the rock bed storage, where A, B, etc. are the levels and 5 pairs (5 sensors) in each level.

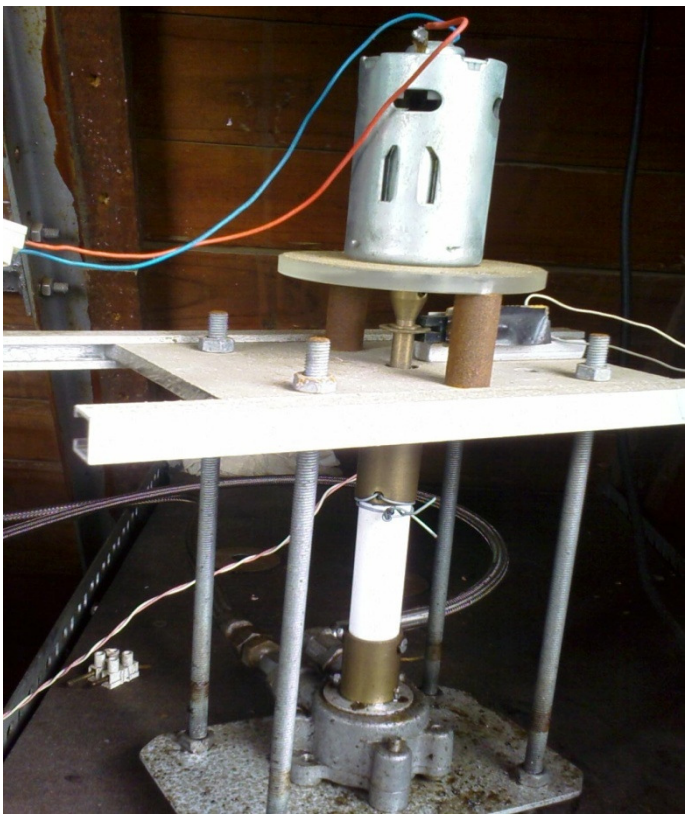


Figure 110 – A detail of the charging pump and motor driver.