Development of an onboard computer (OBC) for a CubeSat

by

LWABANJI TONY LUMBWE

Thesis submitted in fulfilment of the requirements for the degree

Master of Technology: Electrical Engineering

in the Faculty of Engineering
at the Cape Peninsula University of Technology

Supervisor: Prof Richardt Wilkinson
Co-supervisor: Prof Elmarie Biermann

Bellville
May 2013

DECLARATION

I, Lwabanji Tony Lumbwe, declare that the contents of this thesis represent my own unaided work, and that the thesis has not previously been submitted for academic examination towards any qualification. Furthermore, it represents my own opinions and not necessarily those of the Cape Peninsula University of Technology.

Signed **Date**

ABSTRACT

Over the past decade, the satellite industry has witnessed the birth and evolution of the CubeSat standard, not only as a technology demonstrator tool but also as a human capacity development platform in universities. The use of commercial off the shelf (COTS) hardware components makes the CubeSat a cost effective and ideal solution to gain access to space in terms of budget and integration time for experimental science payloads.

Satellite operations are autonomous and are essentially based on the interaction of interconnected electronic subsystems exchanging data according to the mission requirements and objectives. The onboard computer (OBC) subsystem is developed around a microcontroller and plays an essential role in this exchange process as it performs all the computing tasks and organises the collection of onboard housekeeping and payload data before downlink during an overpass above the ground station.

The thesis here presented describes the process involved in the development, design and implementation of a prototype OBC for a CubeSat. An investigation covering previously developed CubeSat OBCs is conducted with emphasis on the characteristics and features of the microcontroller to be used in the design and implementation phases. A set of hardware requirements are defined and according to the current evolution on the microcontroller market, preference is given to the 32-bit core architecture over both its 8-bit and 16-bit counterparts. Following a well defined selection process, Atmel's AT91SAM3U4E microcontroller which implements a 32-bit Cortex-M3 core is chosen and an OBC architecture is developed around it.

Further, the proposed architecture is implemented as a prototype on a printed circuit board (PCB), presenting a set of peripherals necessary for the operation of the OBC. Finally, a series of tests successfully conducted on some of the peripherals are used to evaluate the proposed architecture.

Keywords: CubeSat, OBC, COTS, Microcontroller.

ACKNOWLEDGEMENTS

The completion of this thesis would not have been feasible without the support, inspiration and encouragement of the following people and organisations to whom I would like to express my sincere gratitude:

- My supervisors Prof Richardt Wilkinson and Prof Elmarie Biermann for their patience, guidance and support;
- The French South African Institute of Technology (F'SATI) in particular Prof Robert Van Zyl, Prof Elmarie Biermann, Mr Francois Visser and Mr Ian Van Zyl for giving me the opportunity to complete my Masters degree on a full-time basis;
- The Centre for Instrumentation Research (CIR) staff, store members and students, in particular Khaleel Jooste for his ongoing assistance;
- My family in particular my brother Sean (Zed) Lumbwe and my sister Lydia (LaDiva) Lumbwe for their never ending motivation and encouragement;
- Everyone who took interest in my project;
- And most importantly, my Saviour, Lord Jesus Christ, for giving me the strength to complete this thesis in time.

Special thanks to my colleagues Gavin Mutch and Gary de Villiers.

The financial assistance of the National Research Foundation and F'SATI towards this research is acknowledged. Opinions expressed in this thesis and the conclusions arrived at, are those of the author, and are not necessarily to be attributed to the respective mentioned establishments.

DEDICATION

This thesis is dedicated to my mother LWABANDJI KINDJA ESTHER, who taught me that the best kind of knowledge to have is that which is learned for its own sake and even the largest task can be accomplished if it is done one step at a time.

To my late cousin CHOGO CA LWABANDJI, Rest in Peace.

TABLE OF CONTENTS

GLOSSARY

ADC	Analogue to Digital Converter
ADCS	Attitude and Determination Control System
ALU	Arithmetic Logic Unit
COTS	Commercial-off-the-shelf
CPUT	Cape Peninsula University of Technology
CRC	Cyclic Redundancy Check
CSLI	CubeSat Launch Initiative
C&DH	Command and Data Handling
DIP	Dual In-line Package
DMA	Direct Memory Access
EBI	External Bus Interface
EEPROM	Electrically Erasable Programmable Read-Only Memory
EPS	Electrical Power System
EWARM	Embedded Workbench for ARM
FFPI	Fast Flash Programming Interface
F'SATI	French South African Institute of Technology
FSW	Flight Software
FWUP	Fast Wake Up
GPIO	General Purpose Input/Output
GPS	Global Positioning System
HF	High Frequency
HSMCI	High Speed Multimedia Card Interface

IAC	International Astronautical Congress
IC	Integrated Circuit
ICE	In Circuit Emulator
IDE	Integrated Development Environment
I/O	Input/Output
JTAG	Joint Test Action Group
kSps	Kilosamples per second
LEO	Low Earth Orbit
LGA	Land Grid Array
LSB	Least Significant Bit
MMC	Multimedia Card
MSps	Megasample per second
NASA	National Aeronautics and Space Administration
OBC	Onboard Computer
PC	Personal Computer
PCB	Printed Circuit Board
PDC	Peripheral DMA Channel
PGA	Pin Grid Array
PIO	Parallel Input/Output
PLL	Phase-Locked Loop
PPM	Pluggable Processor Module
PSRAM	Pseudo Static Random Access Memory
PWM	Pulsewidth Modulation
P-POD	Poly Picosatellite Orbital Deployer

QFP	Quad Flat Package
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
ROM	Read Only Memory
RF	Radio Frequency
RTOS	Real Time Operating System
SAM-BA	Smart ARM Microcontroller-Boot Assistant
SCL	Serial Clock Line
SDA	Serial Data
SMT	Surface Mount Technology
SOIC	Small-Outline Integrated Circuit
SPI	Serial Peripheral Interface
TC	Timer Counter
TCCLKS	Timer Counter Clock Select
TT&C	Telemetry Tracking and Command
TWI	Two Wire Interface
UART	Universal Asynchronous Receiver Transmitter
USART	Universal Synchronous Asynchronous Receiver Transmitter
UTC	Unix Time Counter
UTMI	USB2.0 Transceiver Macrocell Interface
UV	Ultraviolet
WDT	Watch Dog Timer

LIST OF FIGURES

LIST OF TABLES

CHAPTER ONE

INTRODUCTION

1.1. Overview

The trend presented by the exploration and utilisation of space is growing on a daily basis and the interest around it has given room for numerous useful applications such as remote sensing[1], global positioning system[2] (GPS) and relay telecommunication[3]. These applications require carefully planned space missions that are often very expensive and take years to develop. Satellites that carry out these missions must operate autonomously for the largest part of their service life, occasionally receiving commands from the ground operator.

Satellite operations are essentially based on subsystems interacting with one another and exchanging data according to the mission requirements and objectives. Virtually all satellites require an onboard computer (OBC) to manage autonomous operations of the satellite and to interact with the ground operators. The study here presented focuses on the design and development of the OBC subsystem for a CubeSat.

This introductory chapter covers all relevant background topics relating to the CubeSat programme and focuses on the CubeSat subsystem architecture, particularly on the OBC, listing its functions and requirements. Also highlighted are the research objectives and the motivation towards this study. Finally, the methodology, the research delineation and the thesis outline are covered.

1.2. Background

The rapid advancement of space technology could not pass unnoticed over the recent years and in this regard, academic institutions across the world have grown interest in space science. More focus and attention were specifically given to the CubeSat platform which was developed and proposed in 1999 by the California Polytechnic State

[1]See for example http://ciesin.columbia.edu/TG/RS/RS-home.html
[2]See for example http://hyperphysics.phy-astr.gsu.edu/hbase/gps.html
[3]See for example http://www.satellites.spacesim.org/english/function/communic/index.html

University (CalPoly) and the Stanford University (Heidt *et al*., 2000). A CubeSat is a cubic satellite of dimensions 10 cm x 10 cm x 10 cm (also referred to as 1U) which can be expanded according to given mission specifications and payloads (Lee *et al*., 2005:8).

In 2009, the Cape Peninsula University of Technology (CPUT) initiated a programme in Space Science and Technology under the auspices of the French South African Institute of Technology (F'SATI). The primary focus of this programme is human capacity development in science and engineering in order to support the national space agenda (DefenceWeb, 2012). A group of CPUT postgraduate students under the leadership of several academics and specialists from industry developed the first South African CubeSat codenamed ZACUBE-1, set to be launched in 2013. Two major payloads are accommodated on ZACUBE-1, namely a matrix imager and a high frequency (HF) beacon transmitter (South Africa. Department of Science and Technology, 2010: 3-4).

CubeSats comprise of several subsystems, each performing a dedicated task. According to Gildeh's (2003) representation of the CubeSat internal organisation, subsystems normally include the following:

- An electrical power system (EPS) supplied by solar panels;
- A payload (e.g. beacon transmitter or camera) with its control unit;
- An attitude determination and control system (ADCS);
- A radio frequency (RF) communication subsystem;
- A memory module generally associated with the OBC and
- An OBC which enables the communication between the different subsystems.

With the high price associated with space grade (class S) and radiation hardened components required for building satellites, it is prohibitively expensive for educational institutions to build traditional spacecrafts. CubeSats were conceived to make access to space more affordable by making use of commercial-off-the-shelf (COTS) components.

ZACUBE-1's OBC is based on the FM430 CubeSat Kit from Pumpkin®. This is a ready to fly kit comprising of a development board, an aluminium skeleton chassis and a pluggable processor module (PPM) based OBC to which additional subsystems designed in-house are stacked on top of each other (Pumpkin Inc., 2008).

The OBC is considered the brain of the CubeSat and essentially consists of a microcontroller connected to subsystems through a serial data bus and additional peripheral hardware. A real time operating system (RTOS) managing all the software applications runs on the microcontroller and constitutes the CubeSat's flight software (FSW). The surface area of the solar panels is proportional to the CubeSat's size. This constraint makes the available onboard power a scarce resource and requires that the microcontroller primarily has low average power consumption and at the same time possesses enough processing power to handle all data transfers according to the mission requirements.

The major functions of the OBC are as follows (Wells *et al*., 2003):

- Recording and storage of telemetry and satellite payload data for transmission to the ground station for analysis;
- Encoding and decoding of data packets to and from the ground station;
- Processing of telecommands from the ground station, including time delay commands received on the uplink channel;
- Monitoring of subsystems, implementing watchdog functions and resetting certain critical subsystems if necessary.

In some cases, power supply management (checking battery level and shutting down subsystems when necessary) is attributed to the OBC but generally this is done by an independent power management system in conjunction with the EPS subsystem (Hidayat, 2010).

1.3. Research focus and motivation

The satellite programme at F'SATI is focused on growing human capacity and expertise within the South African space domain. Future strategies include the development and launch of a CubeSat constellation to be used in specialised applications such as disaster management. To achieve this, research and development of in-house software and hardware components, such as the OBC, is essential. The choice of the CubeSat Kit from Pumpkin® for ZACUBE-1 CubeSat mission was highly motivated by the time frame

restriction since the engineering model had to be unveiled at the International Astronautical Congress (IAC) held in October 2011 in Cape Town, South Africa.

The OBC included in the CubeSat Kit is useful for its flexibility (different processors are available as PPMs). It has also accumulated space heritage in low earth orbit (LEO) CubeSat missions such as Delfi-C3[4], Libertad-1[5] and RAX[6], among others (Pumpkin, 2010). The microcontroller used for these missions as well as for the ZACUBE-1 mission was Texas Instruments' MSP430. This microcontroller is known to combine high performance with low power consumption, making it the ideal choice for developing embedded systems where power consumption has to be kept at a minimum as is the case for a CubeSat's OBC (Albus *et al*., 2009). Pumpkin's Salvo RTOS was selected as the operating system for ZACUBE-1 and is the uniform platform for upcoming F'SATI CubeSat missions.

In order to make future F'SATI CubeSat missions independent of limitations presented by development kits in terms of performance, peripherals, data buses and instruction size, a microcontroller core architecture needs to be investigated for the development of an in-house OBC. A significant evolvement has occurred in the microcontroller market and the migration from 8 and 16-bit core to 32-bit core architectures is common in embedded electronic designs. The trend towards the 32-bit core architecture is driven by the addition of several interfaces and features associated with the need to deliver more processing power for better value for money and flexibility in code re-use across projects using high level languages that lacked in its predecessors (Povey, 2008).

The designer wishing to develop an OBC is spoilt for choice with the wide variety of 32-bit microcontrollers available on the market. Hence, an adequate selection process has to be conducted according to a set of requirements and specifications prior to the design.

In addition to the selection process, the design and development of a working OBC prototype around the selected microcontroller on a printed circuit board (PCB) conforming to the CubeSat physical standards constitutes a challenge. Furthermore,

[4]See http://www.delfic3.nl/
[5]See http://www.usergioarboleda.edu.co/proyecto_espacial/
[6]See http://rax.engin.umich.edu/

peripherals of the developed OBC should be tested for functionality in order to evaluate the prototype.

1.4. Objectives of the research

The primary objective of this research is to design and develop a prototype OBC subsystem for CubeSat applications, based on a selected microcontroller architecture.

In order to achieve this, two milestones have to be reached chronologically, which can also be identified as secondary objectives of this research:

- Defining a set of requirements and specifications for the OBC subsystem which will lead to the selection of the microcontroller to be used for the design of the architecture.
- Designing, implementing and testing the OBC architecture around the chosen microcontroller according to the specifications defined in the first phase.

Additional circuitry should be accommodated alongside the microcontroller for peripheral support and a PC/104 format PCB of the final OBC architecture should be designed, populated and functionally tested.

With the main research objective mentioned, the following research questions were significant to pose:

- Today's microcontroller market provides three major ranges to choose from according to the instruction, data and bus sizes, notably 8, 16 and 32-bit. For the purpose of developing a CubeSat OBC, which option should be considered the best and why?
- What requirements and specifications can be considered ideal for a particular microcontroller when compared to others in its range in order to develop the OBC around it?
- What major improvements will the proposed OBC architecture bring compared to previously designed OBCs?

1.5. Significance and contribution

Satellite engineering is currently emerging in South Africa and the CubeSat platform is viewed as being a convenient tool for human capacity development in this field.
F'SATI provides an academic platform to develop and train engineers in the domain of satellite engineering. The study covered in this thesis will contribute to the South African space programme development by delivering one of the first locally developed, tested and functional CubeSat OBC prototypes.

In addition, a CubeSat will serve as an inexpensive vehicle to test this technology in space, hence contributing to the CubeSat programme.

1.6. Methodology

The workflow process illustrated in Figure 1.1 describes the method that will be followed throughout this research. All the major steps involved in the development of the prototype OBC are presented in the following points:

- A study will be conducted around the CubeSat platform with the goal of gaining a deep understanding of the concept behind it. Focus will specifically be directed to the OBC subsystem and its components. Research comparing previously developed and deployed OBCs will also be conducted in order to define the set of specifications and requirements necessary for engaging in the microcontroller selection process.
- Following the selection, the architectural design will be derived around the microcontroller. The selected microcontroller will be evaluated by looking at each one of its peripherals before creating the OBC architecture that will be implemented around it.
- The developed architecture will be implemented as a hardware prototype and the design will be verified for some peripherals of the OBC.

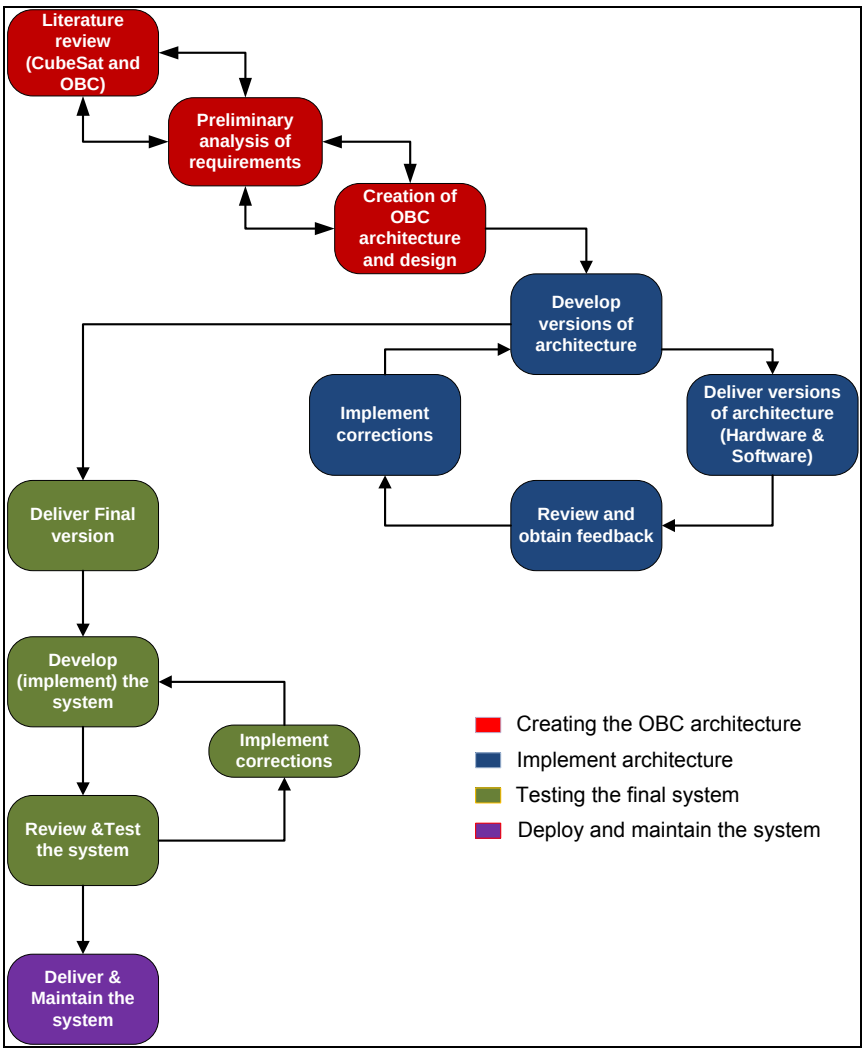


Figure 1.1: Workflow process from concept to delivery

1.7. Research delineation

This thesis will present the design and development of an OBC for a CubeSat. A research process is presented which will lead to the selection of the microcontroller. Further, the OBC architecture is proposed according to the selected microcontroller and finally, the design, implementation and testing of the proposed architecture around the selected microcontroller are presented.

It is important to note that the final OBC prototype will be developed with hardware specifications only since it is not intended for any specific mission. If it had been, a complete flight software (FSW) model would have to be developed, which would have

fallen outside the scope of this project. This research will be regarded as a proof of concept and will deliver an engineering model of the OBC which can be implemented as a flight model once a specific mission has been defined.

1.8. Thesis Outline

This document is structured as follows:

- Chapter 2 covers a detailed literature review of the CubeSat platform, subsystems organisation and space environmental effects on satellite missions.
- Chapter 3 discusses the hardware selection process with a focus on the microcontroller.
- Chapter 4 discusses the architecture development of the OBC around the selected microcontroller.
- Chapter 5 presents all the steps involved in the design and implementation of the OBC prototype.
- Chapter 6 lists all the tests and measurements performed on the prototype.
- Chapter 7 concludes and summarises the study and makes recommendations for future work.

1.9. Summary

The research objectives were defined and the process involved was broken down into significant key points essential for the execution of the project. The significance, contribution and limitations of the study were pointed out and the methodology was subsequently discussed. Finally, the contents of chapters to follow within this thesis were outlined in detail.

The upcoming chapter covers a detailed look at the CubeSat platform and each of its subsystems is presented. Investigations into OBC hardware used for previous CubeSat missions are conducted and the effects of the space environment on semi-conductive components are also outlined.

CHAPTER TWO

LITERATURE REVIEW

2.1. Introduction

In addition to the cost involved in developing a satellite mission, two more constraints can be added, notably the size of the satellite and the risks associated when launching or experimenting with new technologies in space. The introduction of the CubeSat to the satellite market is primarily regarded as an innovation because it simplifies the complexity of access to space by using COTS components in the development phase as opposed to radiation hardened and space qualified components. Governmental organisations, such as the National Aeronautics and Space Administration (NASA), have started to develop CubeSats as a test-bed for larger missions by providing launch opportunities through the CubeSat Launch Initiative [7](CSLI) (NASA, 2010). The CubeSat also opens doors to new markets specifically focused on CubeSat subsystems development and related services.

This chapter introduces the CubeSat by describing its specifications and internal architecture. Emphasis is placed on the OBC subsystem and associated components and a study of previously developed OBCs is also presented. Finally, elements of the space environment that an influence LEO CubeSat space missions are briefly discussed.

2.2. The CubeSat platform

A CubeSat is a standard type of nanosatellite for which a very precise set of specifications exists. As stated in Section 1.2 of Chapter 1, these specifications were developed in 1999 and the initial purpose was to provide a standard design for nanosatellites in order to reduce both the cost and development time and at the same time increase accessibility to space for small payloads and sustain frequent launches. As seen in Figure 2.1, the most restrictive aspect of the CubeSat is its size which is of 1000cm³ (10 cm x 10 cm x 10 cm) and also referred to as a 1U. This can be extended to 2U (10 cm x 10 cm x 20 cm) or 3U (10 cm x 10 cm x 30 cm) form factors depending on

[7] See http://www.nasa.gov/directorates/heo/home/CubeSats_initiative.html

the mission. Associated with the volume is a restricted mass of 1.33 kg for the 1U (Lee *et al.*, 2009).

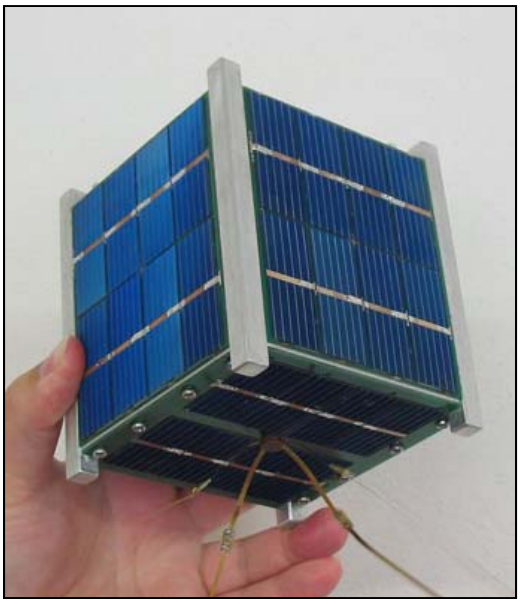


Figure 2.1: The Norwegian Ncube-2 CubeSat
(Adapted from Rupprecht, 2011)

2.2.1. CubeSat architecture and subsystems

In its very basic form, a CubeSat's architecture is similar to what is presented in Figure 2.2. The OBC is at the centre of all communications between other subsystems via a serial bus interface. The OBC generally includes the memory subsystem because it has the additional function to record housekeeping parameters and telemetry payload data collected at given timestamps or coordinates, before initiating the transmission to the ground station during an overpass (Hardy, 2009).

Each subsystem is assigned a dedicated task. A brief description of each subsystem is given in the points that follow.

2.2.1.1. Electrical power system

According to Wertz and Larson (2008), the EPS accomplishes the following functions:

- Supplies a continuous source of electrical power to satellite loads during the mission life;
- Controls and distributes electrical power to the satellite;
- Supports power requirements for average and peak demands from electrical loads;

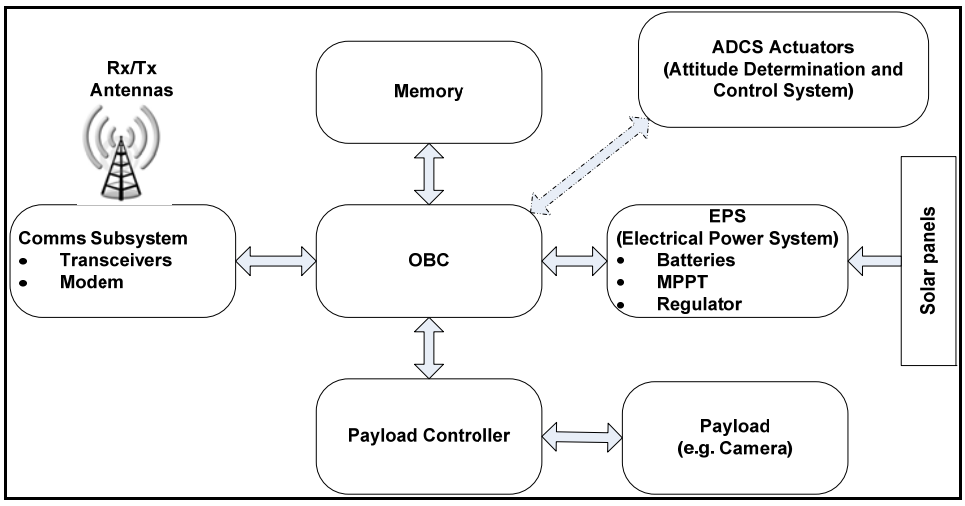


Figure 2.2: Basic CubeSat architecture
(Adapted from Gildeh, 2003)

- Provides command and telemetry capability for EPS health and status as well as control by ground station or an autonomous system;
- Suppresses transient bus voltages and prevent bus faults.

The source that generates the electrical power is an array of photovoltaic solar cells that make up one or more solar panels which convert solar radiation into electrical energy. Satellites orbiting Earth go through a shaded region on the other side of Earth, away from the sun, defined as eclipse. Depending on the type of orbit, eclipse lasts for a few minutes up to a few hours and during this time, the solar panels do not produce electrical power, leaving the satellite unpowered. (Koskienen *et al.*, 1999).

A backup source has to be available in these conditions and electrical energy needs to be stored during sun exposure in order to be used during eclipse. A common practice is to use banks of rechargeable batteries. A series of results published in a study conducted by Clark and Simon (2007) on battery technologies used in small satellite applications recommends the use of Lithium Ion, Lithium Polymer and Nickel Cadmium battery cells.

2.2.1.2. Communications Subsystem

Also referred to as the Telemetry Tracking and Command (TT&C) subsystem, the communications subsystem provides the interface or link between the satellite and the

ground segment. According to Louis and Ippolito (2008), the RF link is essentially defined by two one-way free space or air link channels between the ground segment and the satellite, namely the uplink and the downlink channel. The uplink channel is the portion of the link from Earth to the satellite (upload of telecommands) and the downlink channel the portion from the satellite to the ground station (download of telemetry).

The communications link is made possible by RF transmitters and receivers coupled to high gain antennas onboard the satellite and on the ground station. Telecommands from the ground segment to satellite are sent through the uplink channel every time the orbiting satellite appears at the horizon and becomes visible to the ground station's antenna.

In a similar way, onboard telemetry and payload data are downloaded via the downlink channel to the ground segment. The pass above the ground station only happens for a limited time during which it is a requirement for the receiving antenna to have a clear field of view and enough gain to allow proper data transfer (Pierlot, 2009).

2.2.1.3. ADCS subsystem

Once the CubeSat is deployed, it will randomly tumble while orbiting Earth. The ADCS subsystem is a stabilisation mechanism made of actuators and sensors connected in a loop in order to keep the satellite's orientation steady and ensure it operates efficiently (Hales and Pedersen, 2002).

A closed feedback control loop is used where the attitude of the satellite is first determined by various data collected by sensors and then compared to the desired attitude through complex control algorithms. Actuators are then used to perform the necessary manoeuvres to achieve the required attitude for the satellite and a continuous loop is maintained. Possible sensors include earth, sun and star sensors, gyroscopes, magnetometers and directional antennas while actuators could include magnetic thrusters and wheels which in turn can be reaction wheels, momentum wheels or control moment gyros (Wertz & Larson, 2008).

It is common for CubeSats to make use of reaction wheels because 3-axis stabilisation is achievable within the limited physical space offered by the structure and frequently,

the payload includes a camera that needs to be continuously pointing down to Earth in order to take usable images (Oland & Schlanbusch, 2009).

2.2.1.4. Payload subsystem

According to Denier (2010), satellite payloads can include telecommunication systems, imagers or scientific measurement probes. Satellite missions are therefore classified as communications, imaging or scientific missions. In order to increase effectiveness in terms of cost and integration, it is common to combine two or more types of payloads on a single satellite.

Three types of payload data are associated with communications satellites, namely voice, video and data (e.g. text files and Morse code). Signals are sent from the ground station to the satellite and retransmitted either to the ground or to another satellite (relay) through a channel at a selected frequency band. Applications of communications satellites range from cellular telecommunication to television transmission. A practical example will be the communication with airplanes, ships, trains and trucks just to name a few (Denier, 2010).

The payload onboard imaging satellites consists of a camera which is classified according to the resolution of the image that can be reproduced. The imager is required to be capable of capturing images of Earth or other universal bodies over different spectral bands according to the application. Filtering techniques are then applied to present the image in the usable frequency range. Applications of imaging satellites can range from mapping (cartography), disaster management (fire detection), meteorology or universal observation (Navalgund *et al.*, 2007).

Scientific payloads include measurement instrumentation assisting in collecting specific scientific data for analysis. They can provide information about sun spot activities, solar winds and storms, map carbon dioxide concentrations and evaluate the ozone layer in the atmosphere which can lead to a better understanding of our universe and its components (Denier, 2010).

2.2.1.5. OBC and memory module

Satellite systems are generally engaged in the collection and transfer of information or commands among all of their subsystems. The OBC is at the heart of this data transfer and acts as the onboard communications link between all other subsystems. It is an embedded computer dedicated to command and data handling (C&DH) of the satellite. The major OBC specifications need to be determined and optimised to ensure secure and efficient data flow since different transfer rates are used between subsystems (Hidayat, 2010).

Onboard parameters such as temperature and power consumption in different sections of the satellite are constantly measured and stored in the OBC's memory module and constitute telemetry data to be downloaded to the ground station during an overpass. From the ground station, newer versions of the FSW can be uploaded via the uplink channel and new orbital parameters can be defined and implemented. This is also accomplished by the OBC which decodes these telecommands and executes them by in turn commanding the other subsystems to achieve the desired action (Polaschegg, 2005).

The OBC also makes use of the data from ADCS sensors to determine the attitude of the orbiting satellite and controls the ADCS actuators to control the satellite's attitude. This is done by loading a pre-defined control algorithm into the microcontroller and executing manoeuvres whenever necessary (Hales *et al.*, 2001).

As illustrated in Figure 2.3, the OBC is composed of both hardware and software components. Such a global visualisation is very useful to associate all the OBC elements with their respective functions and build the OBC architecture based on these elements (Noergaard, 2005). These two components are presented in more details:

a. Hardware component

Also called the physical layer, the hardware component contains all the physical electronic components that are located on a PCB. It is the lowest level in the OBC model and consists essentially of a microcontroller with various peripheral interfaces and supporting hardware with a memory module to store programs and data.

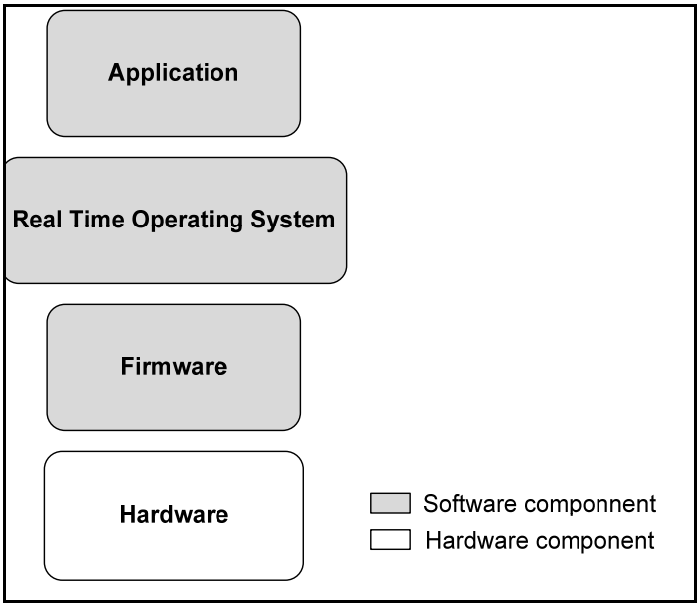


Figure 2.3: OBC subsystem model
(Adapted from Castoulis, 2005)

Castoulis (2005) defines a microcontroller as a central processing unit (CPU), memory and some Input/Output (I/O) devices all contained within a single integrated circuit (IC) and interconnected by an internal bus system. The processor is central to the microcontroller unit and represents the most important part of the system. It manipulates data in a way specified by a sequence of instructions defined in the software layer and intended to suit a given application or program. Selecting the ideal microcontroller is crucial since it constitutes the backbone of the OBC design.

In general, microcontrollers are classified according to their instruction set, memory and internal bus width. Singh (2004) states that the nature of the arithmetic logic unit (ALU) determines if the microcontroller is classified as 8-bit, 16-bit or 32-bit depending on logical and arithmetic operations which will then, respectively, be performed on a byte (8-bit), a word (16-bit) or a double word (32-bit).

The internal memory (also referred to as on-chip memory) of the microcontroller is divided into two types which are defined by Botma (2011) as follows:

- Flash memory: It is a variation of electrically erasable programmable read-only memory (EEPROM) which is accessible in blocks of data. It is the non-volatile portion of the memory, meaning that it retains its value even when power is removed in the system. It has the advantage of presenting fast access time and has a high tolerance to radiation effects.

- Static Random Access Memory (SRAM): It is the volatile portion of the memory, meaning it loses data when power is removed. The read and write operations on SRAM are very fast; it is thus used to store program data which are temporarily allocated and constantly changing during the run of the program.

On-chip memory is usually insufficient and in most cases, microcontrollers provide support for external memory devices in order to expand the memory capacity of the system. In an OBC subsystem, the selected microcontroller should allow for external memory expansion since housekeeping and payload data are usually of sizes bigger than the space available on-chip. The external memory module can also be used to store program applications that would run on the microcontroller and that may require additional memory space (Sakoda, Horning & Moseley, 2005). Various components are available that can be used as memory expansion modules such as secure digital/multimedia cards (SD/MMC) and memory ICs.

The I/O devices (or peripherals) are integrated in the microcontroller and are used by the processor to communicate with other external devices (Catsoulis, 2005). In the case of the CubeSat, these peripherals create an interface between the OBC and the other subsystems. The CubeSat architecture requires a standard serial data bus protocol to be used for fast data rate transfers between subsystems. This data transfer is initialised and monitored by the OBCs' microcontroller. Common serial interfaces found in microcontrollers include universal asynchronous receive-transmit (UART), inter-integrated circuit (I^2C) (also known as two wire interface (TWI)) and serial peripheral interface (SPI).

Additional supporting hardware components to the microcontroller are necessary to ensure the OBC runs efficiently. According to the work published by Clausen *et al.* (2001), it is recommended for the OBC subsystem to include an overcurrent protection

circuitry, a battery backed-up real time clock (RTC) and one or more on-board temperature sensors. The overcurrent protection is needed to monitor the current drawn by the subsystem and disconnect the supply in case of an overcurrent condition. The RTC keeps track of the time and helps to synchronise onboard operations. The RTC also keeps onboard data synchronised with the ground operations. Time is typically encoded in a timing format known as the Unix Time Counter (UTC[8]) system and interrupts are used to synchronise events between subsystems. The RTC's power is backed up with a battery so time is not lost when the OBC's power is removed for any reason. The on-board temperature sensors are necessary to monitor the variations in temperature at different points of the CubeSat which are exposed to the harsh space environment.

b. Software component

The software component commands the processor and controls its operability and functionality. There are three layers of software that are represented by a common shade in Figure 2.3, each interacting only with the layer(s) directly above or below it.

When the microcontroller powers up, it is necessary that the processor's internal registers be configured to an initial state in order to operate correctly. The piece of software responsible for this process is known as the firmware and is the lower software component layer that communicates directly with the hardware component layer and the operating system. The operating system is the system software component that manages the use of and access to the memory and other peripherals within the microcontroller. It acts as a host for applications and is responsible for the management and coordination of activities and sharing of other resources in the OBC subsystem, making it easier for the user to write and run applications (Catsoulis, 2005).

Applications within the OBC are organised in tasks to which a priority level and a deadline are assigned. These tasks are co-ordinated by a real time operating system (RTOS) which according to Whilmshurst (2007) accomplishes three major functions when implemented in a system:

- It decides which task should run and for how long;

[8]See http://www.unixtimestamp.com/

- It provides communication and synchronisation between tasks and
- It controls the use of resources shared between tasks, for example memory and hardware peripherals.

The implementation of several applications (or multi-tasking applications) based on a RTOS constitutes the CubeSat's flight software (FSW). Hidayat (2010) defines the FSW as *"the central intelligent system of the modern satellite that can run on a single chip"*. Enright (2002) further summarises the major flight software functions as follows:

- It schedules satellite activities autonomously when the satellite is away from the ground station radiation coverage, such as scheduling the payload imager to take a picture of a certain location at a particular time;
- It responds to ground user telecommands during the ground station overpass. The FSW should be able to interact with the ground user by accepting all the valid commands and execute commands based on their parameters;
- It performs data logging by taking measurements of the housekeeping parameters (telemetry) and stores them into the memory device until they can be sent to the ground station. Data logging also includes the logging of events that took place during flight ;
- It performs data management, including data structure and protocol.

When the use of a specific RTOS is required, the selected microcontroller needs to be supported by it. An adequate integrated development environment (IDE) to write and compile applications or tasks is also needed in order to program the microcontroller or load the RTOS.

2.2.2. Previously deployed CubeSat OBCs

To date, the number of successful CubeSat missions orbiting Earth amounts to more than fifty and there are still a large number of them currently under development by different organisations and universities worldwide such as TechEdSat[9], StudSat[10] and EQUiSat[11]. However, not all CubeSat missions have been successful in the past and a

[9] See http://www.amsatuk.me.uk/iaru/finished_detail.php?serialnum=228
[10] See http://www.nmit.ac.in/Centerforsmallsatellite.aspx?LinkId=407
[11] See http://amsat-uk.org/2012/09/30/equisat-optical-beacon-cubesat

number of failures have been recorded. These failures were either associated with a technical fault from the launch operator or malfunction of a subsystem onboard the satellite.

Each CubeSat is unique in the mission that it has to perform and the subsequent payload it has to carry. Depending on the initial mission requirements, the OBC subsystem needs to be designed around hardware components that fit within the defined specifications.

In order to illustrate this idea and to facilitate the derivation of the OBC hardware specifications, information was collected and compiled and as a result, the classification found in Table A.1 in Appendix A was drawn. In this table, a compilation of CubeSat missions is presented according to the mission overview, the mission status and the microcontroller used for the OBC subsystem. It shows that most of the previously launched CubeSats had their OBCs designed using 8- and 16-bit microcontrollers as opposed to 32-bit microcontrollers which only started to make their entry recently and whose CubeSats are due for launch. It is also important to consider the fact that the complexity associated with CubeSat missions has started to increase and more than one payload can now be accommodated. This in turn adds to the demand in processing capabilities and, amongst other, microcontroller specifications.

2.2.3. Hardware kits

The success of the CubeSat as a low-cost solution for space experiments has led to the commercialisation of subsystems and modules, giving birth to a new emerging market. Several companies have been established as subsystems and solution providers for CubeSat missions notably, Tyvak Nano-Satellite Systems, GomSpace, Stras Space, Space Micro, Blue Canyon Technologies and Sequoia Space, just to name a few. As an example of services delivered, three of the most successful companies are listed with their specific products or solutions:

- Pumpkin® is the manufacturer of the CubeSat kit which is the most popular and user-friendly CubeSat development kit on the market because of its high success rate in space. It requires the addition and integration of other subsystems around an MSP430 based OBC from the kit which also includes a modular skeleton chassis, a development board for prototyping and the Salvo RTOS as shown in Figure 2.4.

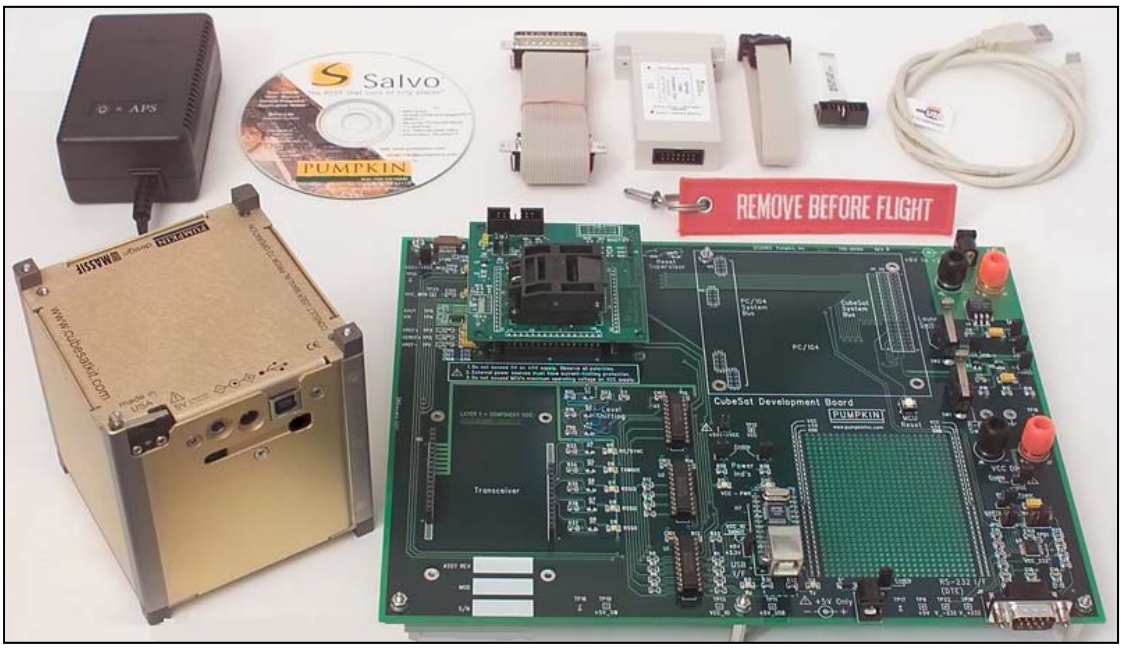


Figure 2.4: CubeSat Kit by Pumpkin

(Adapted from Pumpkin, 2003)

On several missions, the CubeSat kit has proven to be reliable and is currently a commercial success (Pumpkin, 2003).

- Clyde Space is a Scottish based company and a world leading CubeSat subsystem manufacturer and vendor. Their main focus lays in designing and developing high performance EPS, lithium polymer batteries and high efficiency solar panel arrays. The company was founded in 2005 with the primary objective of CubeSats power subsystems development. It has since expanded to small size satellites, larger than CubeSats. Figure 2.5 illustrates a 1U Clyde Space EPS which includes an integrated battery charge management system among other features (ClydeSpace, 2011).

- Innovative Solution In Space (ISIS) was established in 2006 and holds its roots from the Delfi-C3 Dutch CubeSat mission (Delft University of Technology) which was successfully put into orbit in 2008 and is still operational. The company is renowned mostly for delivering high quality telecommunication products (RF subsystem and antennas) and modular CubeSat structures.

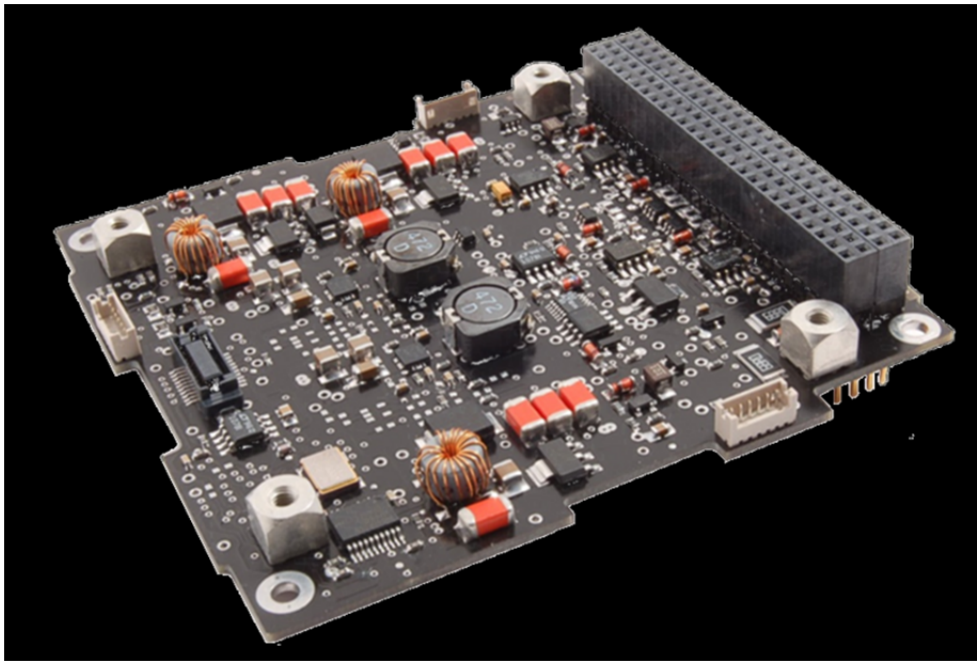


Figure 2.5: 1U CubeSat EPS from Clyde Space (ClydeSpace, 2011)

One of their most successful product to date is a complete ground station kit which represents a solution necessary to provide support in terms of tracking, upload and download exchanges with the CubeSat during a mission (ISIS, 2008).

2.2.4. PC/104 standard

One of the most critical constraints in the CubeSats standards remains its size. The PC/104 form factor which is illustrated in Figure 2.6 is an embedded computer standard configuration which was adopted by the California Polytechnic State University and Stanford University when developing the CubeSat standards. It is restricted by the following factors defined by the PC/104 Embedded Consortium (2003):

- Each PCB has to be 90.2 × 95.9 mm in size;
- Spacing between stacked modules should not be less than 15mm;
- Power consumption typically has to be of the order of 2W maximum per board with a reduced bus drive current of not more than 4mA;
- Bus connectors are optionally *"stack-through"* or *"non-stack-through"* depending on the termination and the module the PCB connects to;
- The modules are designed to be stacked, thus not requiring the use of a backplane;

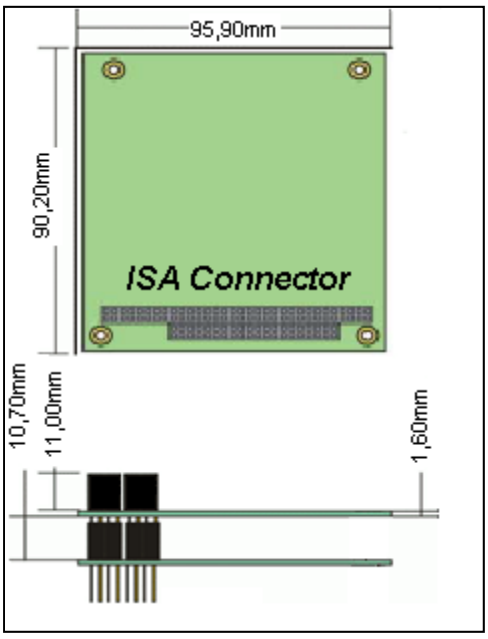


Figure 2.6: PC/104 Standard PCB form factor dimensions

(Adapted from PC/104 Embedded Consortium, 2003)

- The data bus should be configurable for 8-bit, 16-bit and 32-bit;
- The bus connectors should be about 11mm high and 5mm wide. They consist of 1 row for an 8-bit bus and 2 rows for 16-bit or 32-bit buses.

2.2.5. CubeSat deployment system

Due to the high costs of rocket launchers, CubeSats are usually launched alongside commercial satellites as secondary payloads (piggy-back payloads). A standard deployment system known as the Poly Picosatellite Orbital Deployer, or P-POD, is used to eject the CubeSat once in space. It is an aluminium box capable of housing three 1U or, alternatively, one 3U CubeSat.

Figure 2.7 shows a side view and a look inside the P-POD. After the launch, the CubeSat is released from the P-POD through a door mechanism that opens via telecommand.

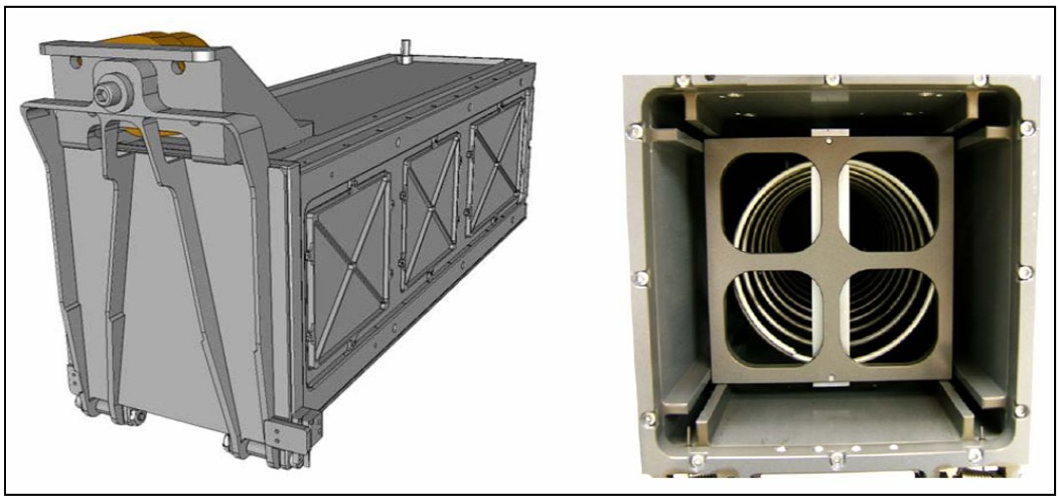


Figure 2.7: Side and internal view of P-POD deployment system (Adapted from Lee *et al.*, 2009)

2.3. Space environment effects

Once a satellite is launched and put into orbit, its operations remain largely autonomous. During the orbital flight, it will be exposed to a variety of mechanical, thermal and electromagnetic effects which may have a direct impact on its operations. A summary of disturbances that CubeSats can be exposed to when orbiting Earth are explained in the following sections.

2.3.1. Atmospheric drag and atomic oxygen

CubeSats are intended to fly in LEO and are exposed to the upper atmospheric layer which applies a braking force on the satellite. This atmospheric drag limits the lifespan of CubeSats. The atmospheric layer heats up and expands during high solar activities, causing a significant increase in atmospheric density and leading to a higher atmospheric drag (Gaposchkin and Coster, 1988).

Similarly, solar panels are exposed to atomic oxygen in LEO which can be highly corrosive to structural materials. At high altitude, the exposure to ultraviolet (UV) rays causes photodissociation of molecular oxygen, i.e. the molecules are split into single atoms (O_1). The atomic oxygen is very reactive and degrades material properties and hence, performance. (Pierlot, 2009).

2.3.2. Solar wind and solar flares

Han and Kim (2006) define solar wind as a constant flow of electrically charged gasses (also known as plasma) and magnetic field emanating from the sun, created when gases in the sun's atmosphere are heated enough to achieve escape velocity and fly off into the solar system. This wind can reach speeds in excess of 400km/s and can be particularly devastating to semi-conductive materials and photovoltaic cells that are directly exposed to it.

Solar flares are a phenomenon that occurs when magnetic energy that has built up in the solar atmosphere is suddenly released. This energy causes emission of radiations that affect the entire electromagnetic spectrum, from radio waves at the long wavelength end, through optical emission to X-rays and gamma rays at the short wavelength end. This increases the ionisation of the upper atmosphere causing interference with telecommunication networks (GPS, GSM, short wave radio), increasing the heat within the outer atmospheric layer and consequently, increasing the atmospheric drag on satellites in LEO (Holman & Benedict, 2007).

2.3.3. Electromagnetic waves in free space

During solar storms which can be unpredictable, electromagnetic wave activity is accelerated within the neighbouring regions of the sun and can get carried away by the flares and wind. From long wavelengths to gamma, X-rays, UVs and infrared waves, effects can be devastating (Hardy, 2009).

2.3.4. Semiconductors in space

The exposure of COTS components to radiation and extreme temperature changes in LEO can have numerous consequences within CubeSats, such as:

- Progressive degradation of components in terms of efficiency;
- Bit errors or bit flips during data transfer (telemetry and telecommands);
- Latchups which could lead to short-circuits (Hardy, 2009).

Figure 2.8: Space debris representation around Earth
(Adapted from Gunter, 2008)

In space, the exterior of the spacecraft is cycled between extreme temperatures from about +150°C on the sun-facing side down to -150°C on the shadow-facing side. Inside the satellite, the temperature ranges are less severe. A minimum operating temperature range for each electronic component used in a CubeSat is usually specified between +85°C and -40°C which is an industrial temperature range (Han & Kim, 2006).

2.3.5. Space debris

A multitude of random materials are orbiting Earth and represent what is known as space debris or space junk. They are a result of satellites being abandoned at the end of their useful life or launch vehicles' upper stages being left in orbit after they have served their purpose and are no longer serving any function (Mehrholz *et al.*, 2002). Figure 2.8 is an artist's impression illustrating space debris currently orbiting Earth.

These objects travel at speeds similar to that of orbiting operational satellites, ranging between 6 and 8 km/s. A major contributor to the orbital debris is and remains object breakup. These breakups are caused by explosions and collisions with other objects in space and danger arises when an operating satellite is involved in the collision because that does not only increase the amount of junk in space, but also represent a loss in investment for the corporation responsible for design, launch and operations of the satellite (Senechal, 2007).

2.4. Summary

An overview of the CubeSat platform and its specifications were given in this chapter and its architectural design was covered in more detail. Each subsystem was identified and defined according to its functions in order to give an overview of the internal CubeSat organisation. A breakdown of the OBC subsystem was presented, listing all essential hardware and software components in order to establish basic requirements and give a global view and orientation to this study. Furthermore, a table of previously deployed CubeSats' OBCs was drawn in order to identify the ideal microcontroller for the upcoming design.

Available services and solutions related to the CubeSat market were discussed and three industry leading companies were presented according to their specific products. Finally, environmental constraints and their impact on the CubeSats' operations in LEO were discussed and some precautions to be taken in order to avoid malfunction or failures during operation were presented.

The next chapter covers the hardware requirements for the microcontroller to be used in the architecture design and implementation of the OBC. These requirements will lead to a selection process based on a range of available microcontrollers, each with individual specifications, features and capabilities.

CHAPTER THREE

MICROCONTROLLER SELECTION

3.1. Introduction

Establishing a set of requirements for the hardware to be used is of high significance in the design process of the CubeSat's OBC. These requirements constitute the basis of the OBC architecture design. With the microcontroller being the major hardware component, a specific path to follow has to be defined in order to select the option that best suits these requirements.

This chapter essentially focuses on the analysis of the hardware requirements and covers the process involved in the selection of the appropriate microcontroller around which the OBC architecture will be implemented.

3.2. Analysis of selection criteria

Once a satellite mission has been defined, a set of requirements needs to be established in order to begin the design process. For this study, the OBC architecture development was based on a flexible mission context because a specific mission was still to be proposed; hence, specific mission requirements were not defined. However, it was possible to identify common attributes for the OBC hardware for upcoming missions based on the information presented in Table A.1 of Appendix A.

The OBC is an embedded system which means that it combines both hardware and software and is intended to accomplish a particular function. When designing embedded systems, Castoulis (2005) and Botma (2011) highlight a series of common constraints that have to be considered. Each one of these criteria is presented, analysed and discussed in the following sections.

3.2.1. Power consumption (peak power and stand by)

A low power scheme is the major requirement in satellite systems due to the fact that electrical power is extremely limited, particularly in CubeSats where a total power budget

of 1W is available for a 1U and 5W for a 3U due to the minimal external surface area covered by solar panels (Lee *et al.*, 2009).

In addition to a low power consumption figure, the microcontroller should accommodate a feature whereby power can be saved by disabling peripherals which are not in use during a particular phase of the mission. It is common for microcontrollers to feature different modes of operation and use more than one clock source, allowing various operating modes and saving on power consumption.

3.2.2. Operating temperature range

The variations of temperature in LEO are quite significant. When the CubeSat orbits the sunny side of Earth, the surface of the satellite gets very warm and reaches temperatures up to 150°C. Similarly, when covering the zone of eclipse, the temperature can drop as low as -150°C (Koskienen *et al.*, 1999).

These extreme changes in temperature can cause significant damage to electronic components inside the CubeSat which exclusively uses COTS components and may cause mission failure. For this reason, all the hardware components of the OBC to be implemented should be rated for an operating range between -40°C and 85°C which is the standard for industrial electronic equipment.

3.2.3. Operating voltage

The EPS in a CubeSat normally provides an unregulated bus voltage varying between 6V and 8.3V. In order to avoid the implementation of several voltage regulators (buck or boost) which may add to the weight, size and complexity of subsystems onboard the CubeSat, most of the active components including the OBC's microcontroller, should be chosen to operate at or below a value of 3.3V.

3.2.4. Packaging

Consideration is also given to the size and weight of the chosen components. The electronic packaging of ICs and of all other different components, will have a direct impact on the overall size and weight of the OBC subsystem.

As shown in Figure 3.1, different types of IC packages are available to choose from, depending on the application. Preference is given to surface mount technology (SMT) over dual in-line package (DIP) due to volume. Furthermore, on the packaging for ICs, small-outline integrated circuit (SOIC) and quad flat package (QFP) are preferred over pin grid array (PGA) and land grid array (LGA) for ease of integration during the soldering process.

3.2.5. I/O and serial bus compatibility

The variety of I/Os (digital and analogue) available on the microcontroller are necessary in terms of connectivity. Being able to interface the final OBC prototype with all other subsystems within the CubeSat is mandatory and is directly related to the microcontroller chosen to implement the design.

The data transfer between subsystems is made possible by the OBC's microcontroller's integrated serial interfaces (UART, I^2C or UART). Selecting a microcontroller which possesses one or more of each of the serial interfaces is highly recommended since each of them is constrained by their data transfer rates and data or address bus sizes (Castoulis, 2005).

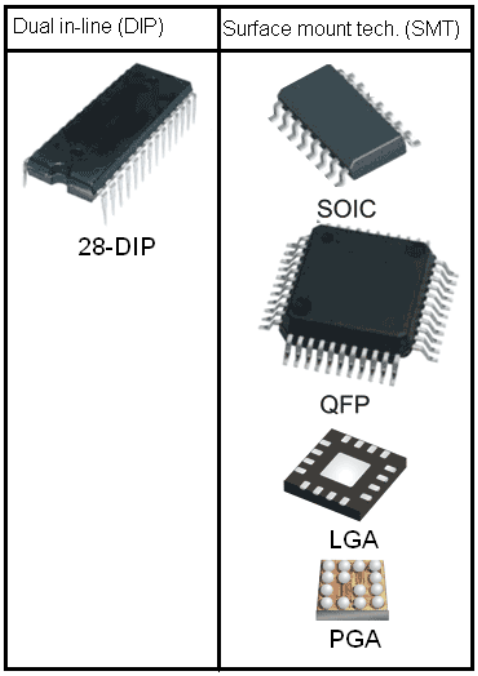


Figure 3.1: Integrated circuits packaging types

3.3. Selection process

The microcontroller can be viewed as the core of the OBC hardware component since all processing operations take place in it. It is composed of a CPU, memories, timers and different sets of interfaces and peripherals allowing interconnections to different modules externally as shown in Figure 3.2.

Depending on the tasks to be completed and how fast the operations need to be accomplished, microcontrollers are available from different manufacturers in variants supporting 8-bit, 16-bit and 32-bit word length.

3.3.1. Migrating from 8-bit and 16-bit to 32-bit

The goal when selecting the microcontroller for the OBC is to be able to implement a prototype that is able to deliver maximum processing capability while using a minimum of power and also being capable of accomplishing all of the required OBC functions, mainly the execution of programs and commands with a minimum response time (Castoulis, 2005).

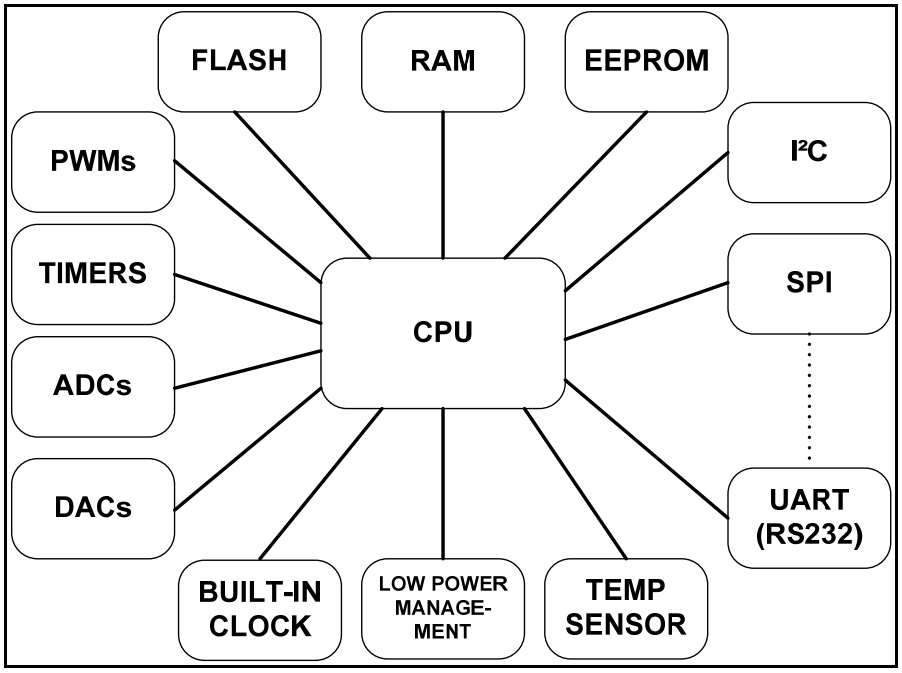


Figure 3.2: Internal conceptual diagram of a microcontroller

In the past, CubeSats have mostly used 8-bit and 16-bit microcontrollers. 32-bit core based microcontrollers were initially introduced in 1985 by Intel and because of the complexity they presented they were only adopted by a limited number of system integrators. 8-bit and 16-bit architectures were still favoured because many of the embedded and real time applications at the time were not critically dependant on memory, power or speed and the amount of data to handle was sufficient. With time, more products and applications started to require increased processing capability. It became clear that a migration from 8 and 16-bit to 32-bit core architecture was necessary, although the complexity remained an issue (Khan, 2008).

Today, the 32-bit core architecture's complexity has been significantly reduced and has been made efficient and capable of handling 8-bit, 16-bit and 32-bit instructions and data. In addition to the reduction in complexity, multiple features have been added and customised by different core manufacturers. On this subject, Povey (2008) states that the trend of migrating toward the 32-bit is mainly driven by the need to deliver increased processing power and flexibility in code re-use across projects using high level languages. These lacked in the 8-bit and 16-bit architectures.

3.3.2. Previous 32-bit implementations

The 32-bit core architecture has been successfully tested on CubeSats for several missions, such as NCube2 and CanX-1.

NCube2 which is illustrated in Figure 2.1 is a product of the Norwegian University of Science and Technology. Its payload consisted of an automatic identification system for the marine industry where the AIS protocol which is a standard for tracking ships, is used to identify, position and exchange data messages between ships. Its C&DH OBC used a 32-bit ATmega32L from Atmel with an AVR32 core and 4kB of ROM. Although contact was never established after launch, the flight-model operated properly on the ground (Eide & Iilstad, 2003).

CanX-1 from the University of Toronto was launched with three experimental payloads onboard: a low cost CMOS horizon sensor, a star tracker and a GPS receiver. Its C&DH OBC used the AT91SAM7 microcontroller which is a 32-bit ARM7 core, from Atlmel with

512kB of static random access memory (SRAM), 32MB of Flash-RAM and with a maximum operating frequency of 40MHz (Stras, *et al.*, 2003).

More examples can be found in Table A.1 of Appendix A which gives details in terms of RAM, ROM and operating frequency of the microcontrollers used for the C&DH OBC of all listed CubeSat missions. It can be seen from this table that 32-bit microcontrollers represent a minority as opposed to 8-bit and 16-bit microcontrollers.

3.3.3. Microcontroller selection

The method followed for the selection of the microcontroller consists of four steps. Because of the large number of core manufacturers in the microcontroller market and the variety associated with them, a process of elimination is followed by comparing each device according to a set of criteria.

Step 1

The search engine provided by the embedded developer website (EmbeddedDeveloper) is used as an initial step to shortlist reputable core design manufacturers. Six manufacturers have been shortlisted, namely: Atmel, ARM, MIPS, Freescale, Renesas and Texas Instruments. From the range of core architectures that are produced and licensed by these manufacturers, only 32-bit ranges were considered.

Figure 3.3 shows a classification of the 32-bit core manufacturers and the architecture that they implement. As an example, ARM is a 32-bit core manufacturer who developed the ARM7TDMI architecture which is implemented by STMicroelectronics and by Atmel (also a core manufacturer). Similarly, Freescale is a 32-bit core manufacturer who developed the COLDFIRE V1 architecture which is also implemented by them.

Step 2

The second step involves a comparison between each semiconductor manufacturer's part numbers. This comparison is done according to three of the previously mentioned selection criteria:

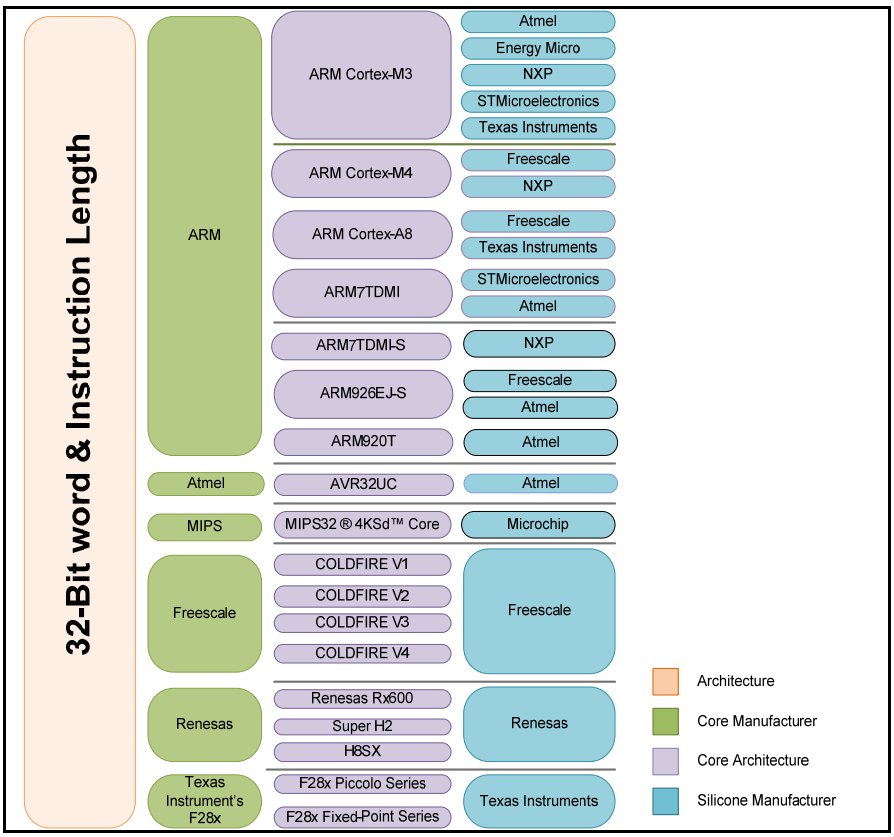


Figure 3.3: 32-bit core architecture classification

- The operating temperature range for the chosen microcontroller should vary between -40°C and 85°C.
- The IC packaging is also important. The QFP package is preferred since the leads extend on each of the four sides of the IC, making it practical and easier to solder unlike its counterparts which do not only require expensive equipment for soldering reliably but also are not recommended for prototyping.
- The operating voltage should not exceed 3.3 V. Since power consumption is directly proportional to the operating frequency, the operating frequency will be limited to 100 MHz to keep the maximum power consumption at a suitable level.

A comparison of the microcontrollers according to these criteria is shown in Table 3.1. This table classifies the semiconductor manufacturer, the core architecture that they implement and their respective part numbers according to the maximum operating frequency, the operating voltage and temperature ranges and the packaging options available for a particular part. The rows highlighted with a light green colour indicate the

parts that did not qualify for the next step and the cells highlighted with a darker green colour indicate the requirement that was not met.

Table 3.1: Microcontrollers requirements criteria

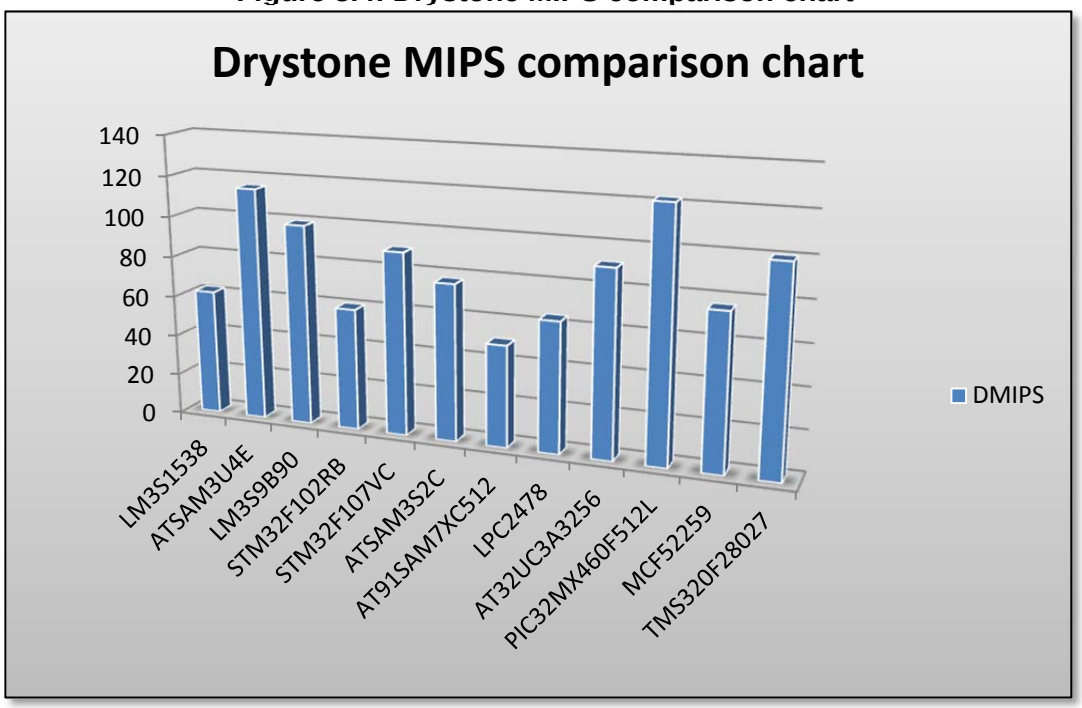
Manufacturer	Core Variant	Part Number	Freq(MHz)	V Min	V Max	T° range	Package Options
Texas Instruments	ARM Cortex-M3	LM3S1538	50	3	3.3	-40 to 85	LQFP100
NXP	ARM Cortex-M3	LPC1343	70	2	3.3	0 to 70	HVQFN33, LQFP48
Atmel	ARM Cortex-M3	AT91SAM3U4E	96	1.6	3.3	-40 to 85	LQFP 144, LFBGA 144
Texas Instruments	ARM Cortex-M3	LM3S9B90	80	3	3.3	-40 to 85	LQFP100
Texas Instruments	ARM Cortex-M3	LM3S9B95	100	3	3.3	-40 to 85	LQFP100
STMicroelectronics	ARM Cortex-M3	STM32F102RB	48	2	3.3	-40 to 85	TQFP 64
STMicroelectronics	ARM Cortex-M3	STM32F107VC	72	2	3.3	-40 to 105	TQFP 100
Energy Micro	ARM Cortex-M3	EFM32TG840F32	32	1.8	3.8	-40 to 85	QFN 64
Atmel	ARM Cortex-M3	ATSAM3S2C	64	1.6	3.3	-40 to 85	LQFP 100, LFBGA 100
NXP	ARM Cortex-M3	LPC1769	100	2.4	3.6	-40 to 85	LQFP100
Freescale	ARM Cortex-M4	MK20N512V	100	1.7	3.6	-40 to 105	PBGA 144
NXP	ARM Cortex-M4	LPC4350	150	2.2	3.3	-40 to 85	LQFP 208, BGA 180
Texas Instruments	ARM Cortex-A8	OMAP3530	600	1.8	3	-40 to 105	FCBGA 423,BGA 515
Freescale	ARM Cortex-A8	MCIMX537	800	0	3.3	-40 to 85	TEPBGA 529
STMicroelectronics	ARM7TDMI	STR755FV2	60	3	5.5	-40 to 85	TQFP100, LFBGA100
Atmel	ARM7TDMI	AT91SAM7XC512	55	3	3.3	-40 to 85	LQFP 100, TFBGA100
NXP	ARM7TDMI-S	LPC2478	72	3	3.3	-40 to 85	LQFP208, TFBGA208
Freescale	ARM926EJ-S	MCIMX27	400	1.7	2.8	-20 to 85	MAPBGA404
Atmel	ARM926EJ-S	AT91SAM9XE512	210	1.6	3.3	-40 to 85	PQFP208, LBFBGA217
Atmel	ARM920T	AT91RM9200	180	1.6	3.3	-40 to 85	QFP 208, LFBGA 256
Atmel	AVR32 UC	AT32UC3A3256	66	3	3.3	-40 to 85	TFBGA 144, LQFP 144
Microchip	MIPS32®4KSd™Core	PIC32MX460F512L	80	2.3	3.3	-40 to 85	TQFP100
Freescale	ColdFire V4	MCF5407	220	2.7	3.3	-40 to 85	FQFP 208
Freescale	ColdFire V3	MCF5372	180	2.7	3.3	-40 to 85	QFP 160
Freescale	ColdFire V2	MCF52259	80	3	3.3	-40 to 85	LQFP144, MAPBGA144
Freescale	ColdFire V1	MCF51JM128	50	2.7	5.5	-40 to 105	LQFP64, QFP64,QFP44
Renesas	H8SX	R5F61668	50	3	3.3	-20 to 75	QFP144, LFBGA176
Renesas	SH2	DF71253	50	4.5	5.5	-20 to 85	LQFP64
Renesas	Renesas RX 600	R5F562N8B	100	2.7	3.3	-40 to 85	LFBGA 176,LQFP 144,
Texas Instruments	F28x Fixed-point Series	TMS320R2811	150	3.1	3.3	-40 to 125	LQFP 128
Texas Instruments	F28x Piccolo Series	TMS320F28027	60	3.1	3.3	-40 to 125	LQFP 48

Step 3

The third step involves an additional comparison; according to processing power referenced to a standard benchmark. The commonly used benchmark is the Dhrystone MIPS or DMIPS and microcontroller manufacturers always classify their products by running the DMIPS benchmark test and awarding them with a score which appears on the IC's datasheet (Weiss, 2002). This benchmark is directly proportional to the processing power of the chosen microcontroller which Steinmetz and Wherle (2005) define as being a representation of the measure of how much computation can be handled by the microcontroller within a given system.

A chart comparing the twelve short-listed microcontrollers' DMIPS scores is shown in Figure 3.4. A value above 90 DMIPS is recommended for designing embedded systems requiring real-time operations as it is the case for the CubeSat's OBC. Five out of the twelve microcontrollers met this criterion, namely: Atmel's ATSAM3U, Texas Instruments' LM3S9B90, Atmel's AT32UC3A3256, Microchip's PIC32MX460F512L and Texas Instruments' TMS320F28027.

Figure 3.4: Drystone MIPS comparison chart



Step 4

The final step in the selection involves an individual comparison of each of the final five microcontrollers according to the available onboard memory, the presence of internal and external peripherals and the serial interfaces. Table 3.2 identifies areas of strength and weaknesses of each microcontroller and presents each one's characteristics.

Table 3.2: Refined microcontrollers comparison

Part Number	AT91SAM3U4E	LM3S9B90	AT32UC3A3256	PIC32MX460F512L	TMS320F28027
Manufacturer	Atmel	Texas Instruments	Atmel	Microchip	Texas Instruments
Core Variant	ARM Cortex-M3	ARM Cortex-M3	AVR32 UC	MIPS32®4KSd™Core	F28x Piccolo
Flash (Bytes)	262144	262144	262144	524288	65536
ROM (Bytes)	16384	0	0	0	2048
RAM (Bytes)	53248	98304	131072	32768	12288
Floating Point Operation	x	x	x	x	x
Memory Management Unit	√	x	√	x	√
External Bus Interface	√	√	√	x	√
A/D resolution	10	10	10	10	12
A/D Ch	16	16	8	10	13
PWM	4x16-bit	8x16-bit	N/A	N/A	8x16-bit
Timers	16-bit Digital Programable Timer/Counter	5xGen. Purp. Timer Watchdog Timer SysTick	2x16-bit Timer/Counter Real-Time Clock Watchdog Timer	5x16-bit digital timers	1xWD 3x32-Bit GP
Timer Bits	16	32	16	16	16
Serial Interfaces	4xUSART 1xUART 1xSPI 1xHSMCI	3xUART 2xSSI	2xSPI 4xUSART	2xUART 2xSPI	1xSCI 1xSPI
I^2C Interfaces	2	2	2	2	1
USB Interfaces	USB 2.0 480 Mbps	2.0 OTG/Host	Full-Speed + OTG	USB 2.0 OTG + PHY	N/A
CAN Interfaces	0	2	0	0	0
CAN Type	N/A	2.0 A/B	N/A	N/A	N/A
Ethernet Interfaces	N/A	10/100 MAC/PHY	N/A	N/A	N/A
Encryption	N/A	AES	N/A	N/A	N/A
DMA Channels	21	32	0	4	0
Maximum GPIO	96	60	110	85	22

After evaluation, the AT91SAM3U4E was selected. It is Atmel's implementation of the Cortex-M3 core and this choice is motivated by the following reasons:

- The AT91SAM3U from Atmel presents a significant advantage in terms of ROM (16kB) which is absent or minimal in its counterparts (2kB for Texas Instruments' TMS320F28027).
- In addition to the ROM, it also has the capability to perform memory management via a memory management unit (MMU) and offers the possibility to extend the memory if needed via the external bus interface (EBI) and the high-speed multimedia card interface (HSMCI).
- The absence of a control area network (CAN) interface, an Ethernet interface and encryption on Atmel's AT91SAM3U4E does not represent a handicap when compared opposed to Texas Instrument's LM3S9B90 since they are not required in a CubeSat. The presence of more than one of each of the common serial peripherals further justifies the choice of the AT91SAM3U4E.
- A total of 96 general purpose I/Os (GPIO) with 16 ADC channels are present.

Additional features of the AT91SAM3U4E are discussed in detail in Chapter 4.

3.4. Summary

Requirements related to the hardware to be used in the OBC design were identified and listed and the selection process for the microcontroller to be used was explained in detail in four steps. Comparison tables were drawn, identifying strengths and weaknesses of each one of the selected microcontrollers. After proceeding by elimination, Atmel's implementation of the Cortex M-3 core, the AT91SAM3U4E, was selected as the microcontroller to design and implement the CubeSat's OBC.

The upcoming chapter presents the AT91SAM3U4E according to its features, notably the CPU, memories, peripherals and different modes of operation. The proposed OBC architecture to be designed around the microcontroller is also presented.

CHAPTER FOUR

PROPOSED OBC ARCHITECTURE

4.1. Introduction

In order to proceed with the design and implementation of the OBC hardware around the microcontroller, the initial phase involves the development of the architecture. The selected microcontroller provides a number of features which need to be implemented and used accordingly in order to accomplish the list of services required by the OBC subsystem.

The first step in defining an OBC architecture consists of defining hardware requirements and analysing the available resource, which in this case is the microcontroller. Background regarding Atmel's AT91SAM3U is provided in this chapter and its features and onboard peripherals are also presented. The proposed OBC architecture is introduced as a fairly simple but effective solution since an increase in complexity would further increase the complexity in implementing the architecture as a hardware prototype.

4.2. Presentation of Atmel's AT91SAM3U4E

Features and characteristics of the AT91SAM3U4E are highlighted before the OBC architecture is developed. The points to be discussed present the processing core, the internal memory unit, the peripherals and the power consumption of the microcontroller in different modes of operation.

4.2.1. Processing core

A block diagram of the AT91SAM3U4E microcontroller showing its internal structure is presented in Figure 4.1. It is an implementation of ARM's Cortex-M3 core processor which is a high performance core with a 3-stage pipeline Harvard architecture (using separate instruction and data buses), making it ideal for demanding embedded systems applications. It implements a version of the Thumb® instruction set based on the Thumb-2 technology, ensuring high code density and reduced program memory requirements

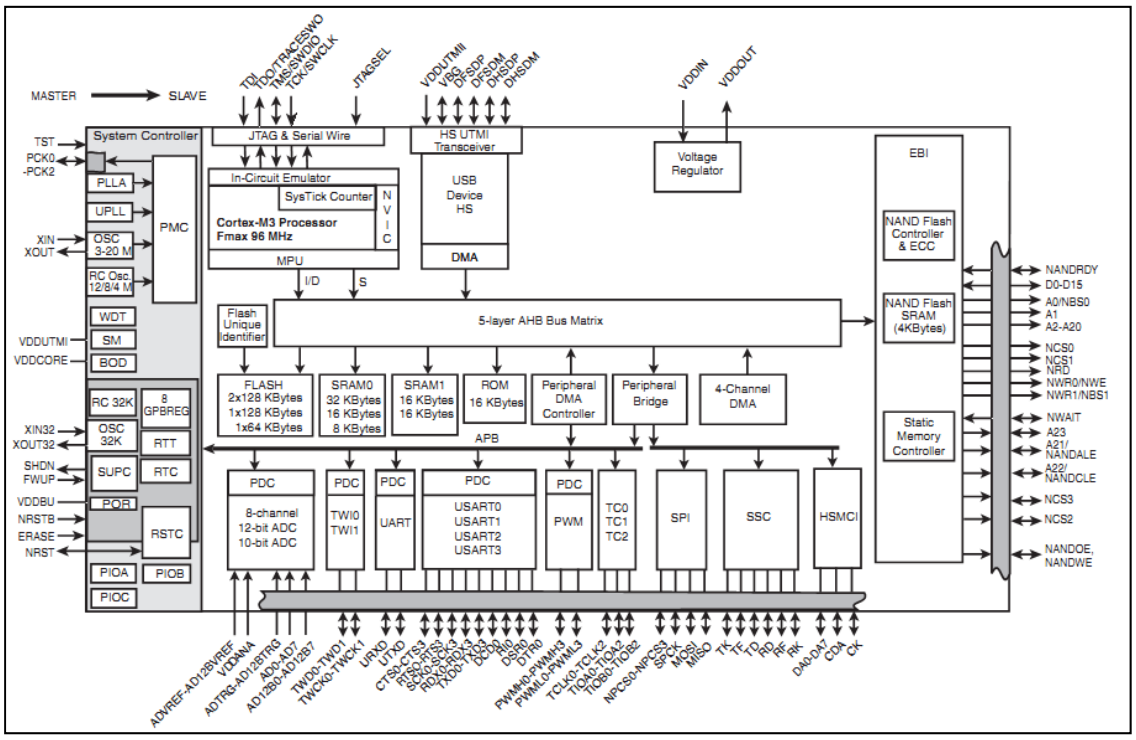


Figure 4.1: Internal Block diagram of the SAM3U4E (Atmel Corporation, 2009)

meaning that it provides the exceptional performance expected of a modern 32-bit architecture with the high code density of 8-bit and 16-bit microcontrollers (ARM, 2010).

In real-time applications such as the CubeSat's OBC, it is crucial to be able to service interrupts and exceptions within the minimum time (interrupt latency). This time is largely determined by the interrupt controller circuit and its configuration. The Cortex-M3 core is closely integrated with a nested vector interrupt controller (NVIC) which facilitates low-latency in exceptions and interrupt handling and represents a key feature of this core. It supports nesting (stacking) of interrupts, allowing an interrupt to be serviced earlier by exerting high priority and it also supports dynamic reprioritisation of interrupts. The vector table which contains addresses of interrupt service routines is of a re-locatable type; this means that the address located at address zero can be altered and relocated by simply programming a register (ARM, 2010).

The main features of the NVIC include:

- A configurable number of external interrupts, from 1 to 240;

- A configurable number of bits of priority, from three to eight bits (256 priority levels);
- Eight levels of pre-emption priority for interrupts;
- Level and pulse interrupt support;
- Dynamic reprioritisation of interrupts;
- Priority grouping and
- Support for tail-chaining of interrupts.

A memory protection unit (MPU) is also available. This feature allows memory protection and restricts access to particular memory locations of the memory map (ARM, 2010).

4.2.2. Memory capacity and access

The implementation of Harvard architecture in the Cortex-M3 represents a major advantage in that instructions and data fetches can occur concurrently, and the size of an instruction is not set by the size of the standard unit (word) combined with the Thumb® instruction support, data can be 8-, 16- or 32-bits wide (Catsoulis, 2005).

A memory-mapped I/O configuration is used, where I/O devices exist within the same linear space as memory devices and the same instruction is used to access each of them. The following is provided in terms of internal memory:

- 256kB of flash organised in a dual plane configuration with 512 pages;
- 48kB of high-speed SRAM (32kB SRAM0 and 16kB SRAM1) and
- 16kB of ROM containing the SAM-BA boot and Fast Flash Programming Interface (FFPI) program (ARM, 2010).

An external bus interface (EBI) is available and allows the microcontroller to interface with an external memory module via a 16-bit parallel data bus which can also be adapted for 8-bit devices. The EBI provides for 4 chip selects and up to a 24-bit address bus.

Also provided is a high speed multimedia card interface (HSMCI) which supports a large number of memory card formats and is 8-bits wide. It is compatible with the following:

- MultiMedia card (MMC);
- SD Memory cards;
- SDIO and CE-ATA cards.

4.2.3. Peripherals

The AT91SAM3U4E embeds a large set of user peripherals distributed on two separate bridges (high speed and low speed bridges). The advantage of this configuration is that concurrent accesses can be made on both bridges.

Peripherals include:

- High speed USB 2.0 device up to 480Mbps;
- 4 USART (Universal Synchronous Asynchronous Receiver Transmitter);
- 1 UART;
- 1 HSMCI (8-bit wide);
- 2 TWI or I²C;
- 1 SPI;
- 1 SSC (serial synchronous controller);
- 4-channel 16-bit PWM (pulse width modulation) controller;
- 8-channel 12-bit ADC (analogue to digital converter) with differential input mode and programmable gain stage;
- 8-channel 10-bit ADC;
- 96 I/O lines with external hardware interrupt capability;
- 3-channel 16-bit timer/counter (TC) used for capture, compare and PWM;
- 32-bit real time timer (RTT) and a real time clock (RTC) with calendar and alarm features (Atmel Corporation, 2009).

All the peripherals are on the low-speed bridge except the SPI, the SSC and the HSMCI which are on the high-speed bridge.

Direct memory access (DMA) is an additional feature of the microcontroller. This enables hardware peripherals linked to its channels to bypass the core processor and access the system's memory independently. During this process, the DMA controller takes the place of the CPU and act as a bridge during data transfer while the CPU can continue with

normal operations. This process improves the overall performance of the microcontroller, making it faster (Catsoulis, 2005). The AT91SAM3U4E has a total of 19 peripheral DMA channels and an additional 4-channel central DMA.

4.2.4. Power consumption and operating modes

The microcontroller embeds an internal voltage regulator which supplies a stable voltage to the core processor and which can also be used to supply external circuitry. It can operate in two modes, namely:

- Normal mode: the regulator draws around 150mA and consumes 700μA of static current;
- Shutdown mode: The output of the voltage regulator is driven to ground and its current consumption is of the order of less than 1μA (Atmel Corporation, 2009).

The microcontroller normally operates in active mode, where the core clock is running from a chosen clock source and peripheral clocks can be enabled or disabled depending on the application. Alternatively, it also has three low power modes of operation, namely:

- Backup mode: This mode achieves the lowest power consumption by performing periodic wake-ups to perform tasks (interrupts).
- Wait mode: The purpose of the wait mode is to achieve very low power consumption while maintaining the device in a powered state for a start-up time of less than 10μs. In this mode, the clocks of the core, peripherals and memories are stopped. However, the core, peripherals and memories power supplies are still powered. From this mode, a fast start-up is available.
- Sleep mode: The purpose of sleep mode is to optimise power consumption of the device versus response time. In this mode, only the core clock is stopped. The peripheral clocks can be enabled (Atmel Corporation, 2009).

4.3. Proposed OBC architecture

The OBC architecture is essentially based on the connectivity between subsystems within the CubeSat. This simply means that the microcontroller's peripherals are configured according to the data flow within the CubeSat's computing scheme. A basic

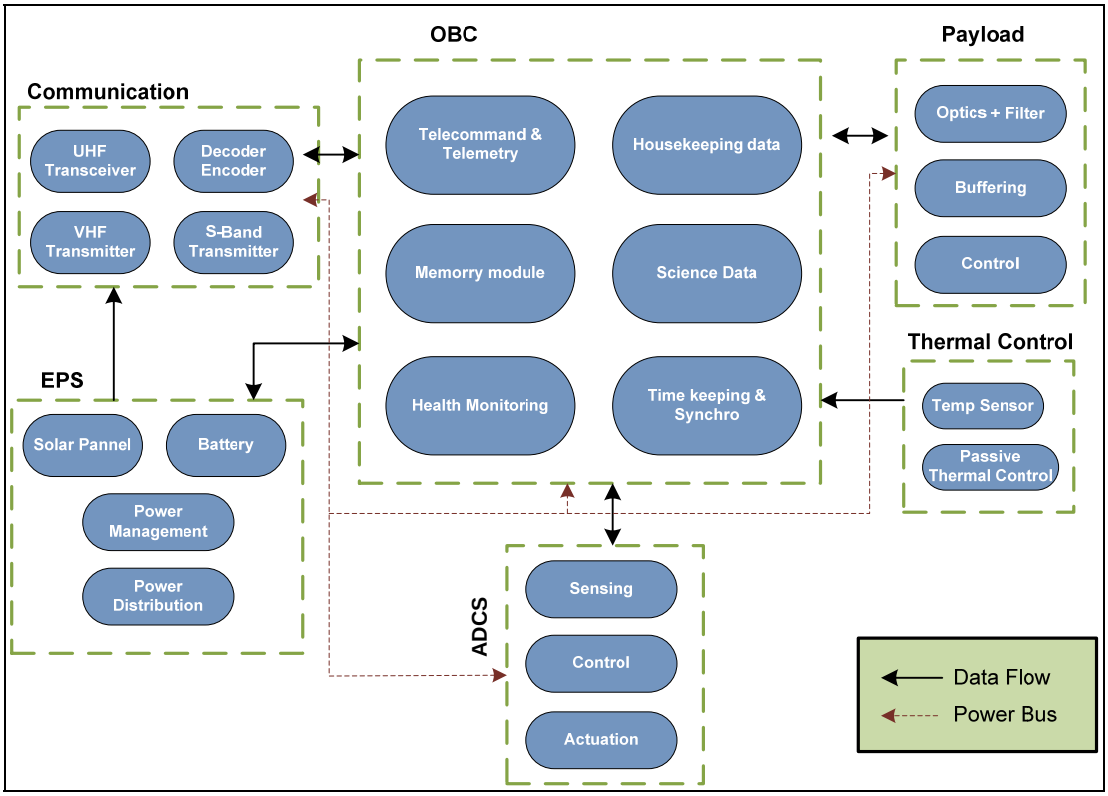


Figure 4.2: Basic functional diagram of a CubeSat internal organisation

functional diagram of the CubeSat's internal organisation is shown in Figure 4.2 where the direction in which data flows and the power distribution network are clearly shown.

Based on the data flow network, the power distribution network and the characteristics of the microcontroller, it is possible to establish specifications for the OBC that are presented in Table 4.1. They can be summarised in the following points:

- The board size should comply with the PC/104 standard;
- An operating temperature between -40°C and 85°C should be assumed and every component constituting the OBC should be able to operate within this range;
- An onboard temperature sensor must be available to keep track of temperature changes onboard the OBC;
- A regulated voltage of 3.3V should be available on the local power bus;
- An external storage device should be made available in order to store housekeeping and telemetry data before transmission to the ground station;
- The presence of a backup battery is essential in order to evaluate the back-up mode for date and time whenever the main power is removed.

Table 4.1: General OBC specifications

Size	PC/104 compliant (form factor of 90 x 95 mm)
Operating temperature	Between -40°C and 85°C
Programming interface	JTAG connector or USB2.0 port (in system programming)
External memory storage media	SD/MMC card on 8-bit HSMCI bus
Over voltage & Current protections	Implemented using MAX4374F
RTOS compatibility	Salvo compatible
Onboard oscillators	12MHz main crystal and 32kHz low-power for RTC or device clock
Maximum operating frequency	Operates up to 96MHz
Other peripherals	2 X USART
	1 X UART
	2 X TWI (I²C)
	1 X SPI
	8 channel 12-bit ADC and 8 channel 10-bit ADC
	1 x Temperature sensor (I²C connected)
	1 x Backup battery
	96 x GPIOs
	1 x USB
Supply line voltage	Regulated at 3.3V
Overall power consumption	To be determined.

- An over voltage and current protection should be implemented in order to trip the power and reset the OBC in case of a short circuit or instability in voltage levels;
- A maximumm number of peripherals should be available in order to facilitate serial data communication between the OBC and the other subsystems, sensors and possibly actuators. It is important to have serial communication architecture available since the data transfer rate to and from the OBC changes from one subsystem to another; hence the increase in flexibility.

The resulting architecture is presented in Figure 4.3 and is a representation of all major peripherals necessary for operation and communication of the OBC subsystem.

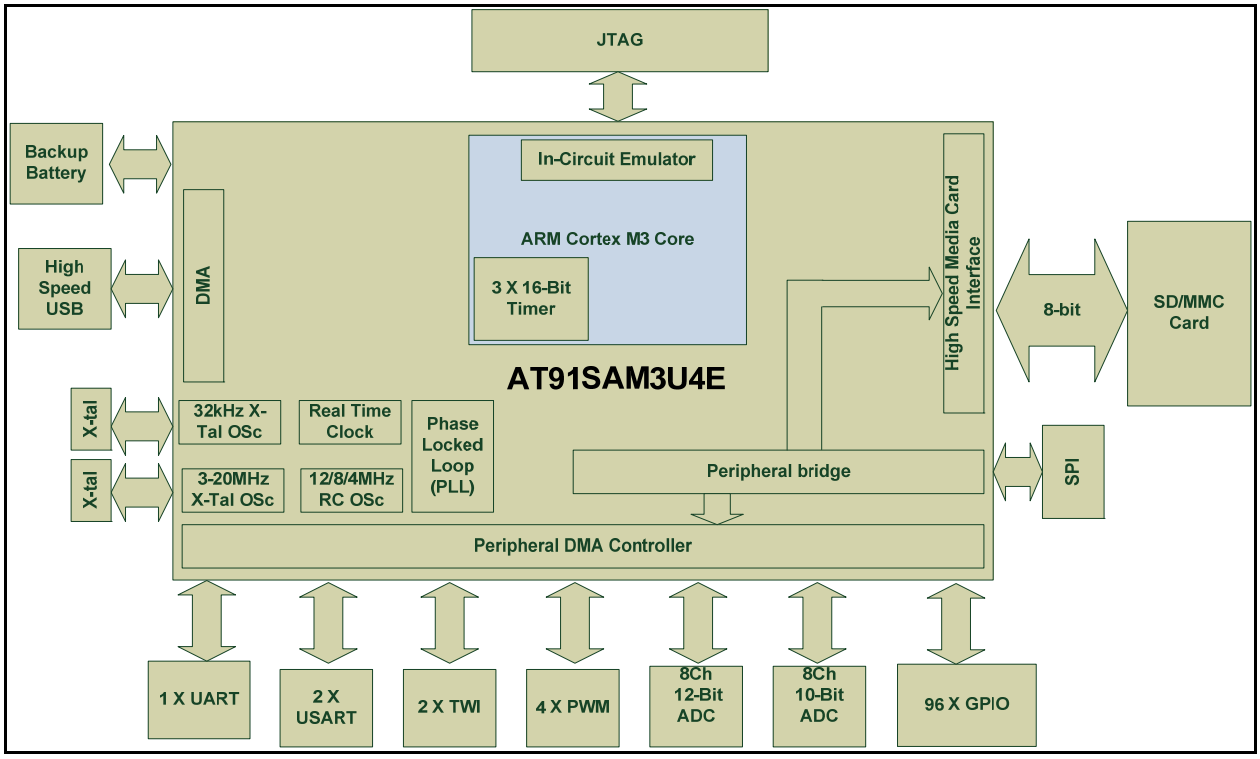


Figure 4.3: Proposed OBC architecture

4.4. Summary

The AT91SAM3U4E microcontroller was introduced and presented in terms of its processing core, memory capacity, memory access, peripherals, power consumption and operating modes.

Following the detailed characterisation of the microcontroller, the internal connectivity between the OBC and the different subsystems in terms of data flow and power distribution was presented.

Finally, a table listing characteristics according to which the OBC hardware will be implemented was presented.

In the upcoming chapter, a walkthrough of the design and implementation processes of the OBC hardware will be covered. Details regarding the choice of each major component and the implementation of each section of the OBC will be covered and a physical PC/104 board layout will be completed and presented.

CHAPTER FIVE

DESIGN AND IMPLEMENTATION

5.1. Introduction

The flow of data between the OBC and the various subsystems onboard the CubeSat is made possible via the microcontroller's peripherals which were introduced in section 4.2.3 of Chapter 4. This data flow occurs at various rates depending on the peripheral used to individually communicate between the OBC and the other subsystems. The AT91SAM3U4E has the advantage of possessing numerous peripherals from which the OBC architecture was drawn in Chapter 4.

This chapter covers the detailed design and implementation of the presented OBC architecture. Each component of the OBC subsystem is analysed individually and schematic diagrams of each subsection are also included. Finally, the PC/104 format implementation of the PCB prototype is presented.

5.2. OBC components

It is important to list the services that the OBC will be providing in terms of data transfer to and from the different CubeSat subsystems. According to Brand (2007), these services are very common to CubeSats and can be summarised as follows:

- Providing a low-speed redundant communication bus for information exchange between peripherals;
- Providing a high-speed communication interface for communication to the payload;
- Performing general housekeeping tasks;
- Providing the ability to run user processes (ADCS or image processing).

A block diagram showing all different components of the OBC is represented in Figure5.1.

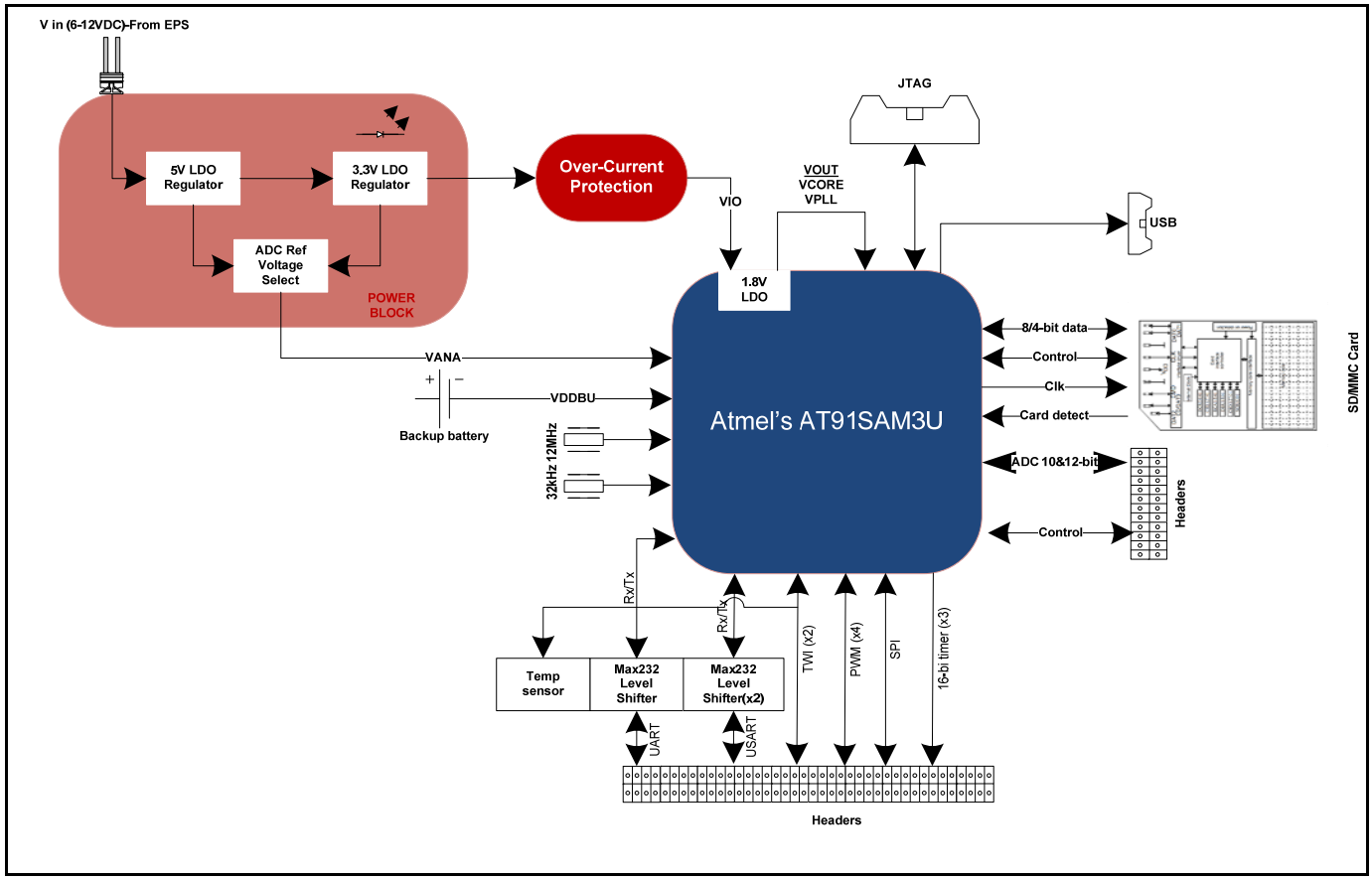


Figure 5.1: Detailed view of bus connections and OBC main components

This diagram can be broken down into two basic building blocks, notably:

- The power system block: It supplies a regulated voltage to the OBC and provides circuitry for over voltage and overcurrent protection.
- The peripheral block: It represents all inputs and outputs to and from the microcontroller. Each peripheral is assigned to one or more registers and to a base address which are respectively configured and called within the execution of the program.

The sections to follow cover each block in detail and discuss the hardware implementation associated with each section of the OBC.

5.2.1. Power system block

The CubeSat architecture provides an external regulated DC bus voltage from the EPS subsystem which supplies all other subsystems including the OBC. This voltage varies between 8V and 15V depending on how charged the batteries are. It is necessary for each subsystem to have its own and independent voltage regulation and protection in order to operate within the required range and to distribute the power according to the provided power budget.

The power system block of the proposed OBC is divided into two sections, namely the voltage regulation and the overcurrent protection.

Voltage regulation

The AT91SAM3U4E microcontroller has a total of 8 supply points, each serving a particular function. These supply points are presented in Table 5.1 which provides a function and typical voltage levels for each one of them.

The internal voltage regulator has a nominal output voltage of 1.8V that appears at the VDDOUT pin. This voltage is used uniquely to power the core through the VDDCORE pin and the phase locked loop (PLL) through the VDDPLL pin. VDDIN is the input to the internal regulator. The additional supply points have input voltages that fall within the specified value of 3V. These are VDDUTMI, VDDANA and VDDIO. VDDBU can be powered by the same line or alternatively, by a backup battery. This supply point is necessary when running the microcontroller in backup mode (Atmel Corporation, 2009). The different modes of operation will be covered in the upcoming chapter.

The LM117 series of low dropout voltage regulators from National Semiconductor Corporation provides a range of fixed voltage outputs including 3.3V and 5V. It was selected to be implemented in the design as the regulating device due to the multiple features it presents, notably:

- A maximum output current of 800mA;
- An operating temperature range between -40° C and 125°C;

Table 5.1: Supply points on the AT91SAM3U4E

Pin Name	Function	Typical voltage
VDDCORE	Powers the core, embedded memories and peripherals.	1.62V to 1.95V
VDDIO	Powers peripherals I/O lines.	1.62V. to 3.6V
VDDIN	Powers internal voltage regulator.	3V to 5V
VDDOUT	Output of internal voltage regulator.	1.8V(V out)
VDDBU	Powers the slow clock oscillator. Needs to be supplied at the same time as VDDIO and VDDCORE.	1.6V to 3.6V
VDDPLL	Powers the programmable PLL and UPLL as well as the 3-20MHz oscillator.	1.62V to 1.95V
VDDUTMI	Powers the UTMI.	3.0V to 3.6V
VDDANA	Powers the ADC cells.	2.4V to 3.6V

- An optional current limiting and thermal shutdown feature;
- Availability in SOT-223 package;
- 0.2% line regulation and 0.4% load regulation (National Semiconductor Corporation, 2006).

An LM117 3.3V regulator placed at the input line is used to supply the voltage needed at VDDUTMI, VDDANA and VDDIO. In addition, a 5V regulator inserted in parallel with the bus voltage input is present in order to give the possibility to select the ADC reference voltage (ADVREF) between 2.5V and 3V. An LM4040 shunt voltage reference which uses Zeener regulation is used to attenuate and regulate the 5V output down to 2.5V which can be selected as ADVREF (National Semiconductor Corporation, 2000). The ADC is discussed in a further section of this chapter.

A schematic of the voltage regulation section and the reference voltage input to the ADC is shown in Figure 5.2 where U2 and U3 represent the LM117 voltage regulators.

Table 5.2: Jumper arrangement for ADCVREF

Connector ID	Pins 1-2	Pins 2-3	Pins 1-3
P6	ADC 10-Bit @ 2.5Vref	ADC 10-Bit @ 3.3Vref	N/A
P7	ADC 12-Bit @ 2.5Vref	ADC 12-Bit @ 3.3Vref	N/A

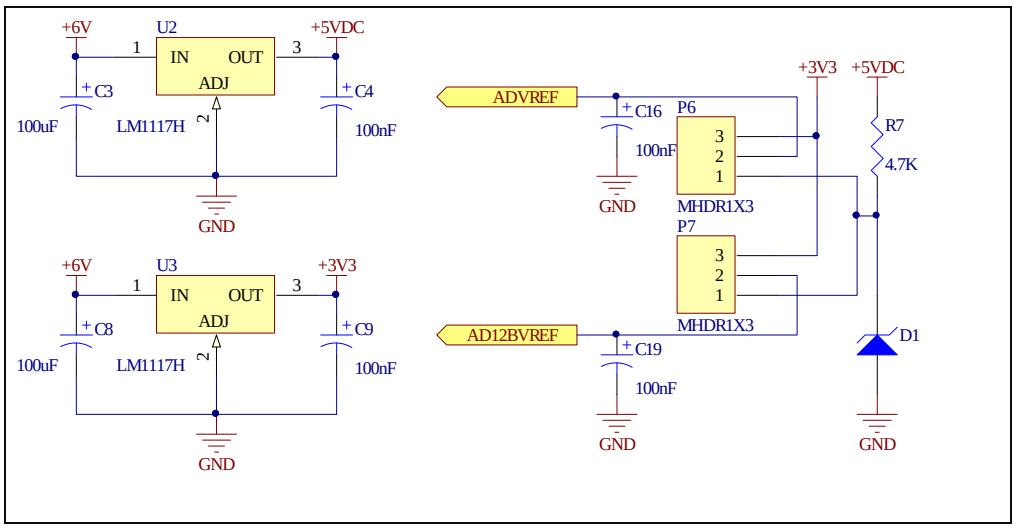


Figure 5.2: Voltage regulation and ADVREF selection

P6 and P7 are selection jumpers that set ADVREF either to 3.3V or to 2.5V for both the 10-bit and the 12-bit ADC channels according to Table 5.2. D1 represents the LM4040 shunt voltage reference forming a voltage divider with resistor R7. The capacitors are used for decoupling the supply line.

Overcurrent protection

In case of a surge in current which may occur in the event of an onboard short circuit, it is necessary to implement a protecting circuitry on the OBC. Out of many techniques used, the high-side current sense principle was decided upon, where a shunt resistor commonly known as the sense resistor, is connected in series with the supply line and a voltage measurement is taken across it, which is directly proportional to the current flowing in the circuit. This choice is highly motivated by the advantages it presents when compared to its low-side current sense counterpart:

- The load is directly connected to ground;
- The load is directly protected and cannot be activated by an accidental short at power connection;
- In case of a short-circuit occurrence, the high load current can be detected (Maxim Integrated Products, 2001).

The MAX4374 from Maxim Integrated Products is chosen to accomplish this task. It is a low-cost and micro-power high-side current sense IC equipped with an amplifier, an

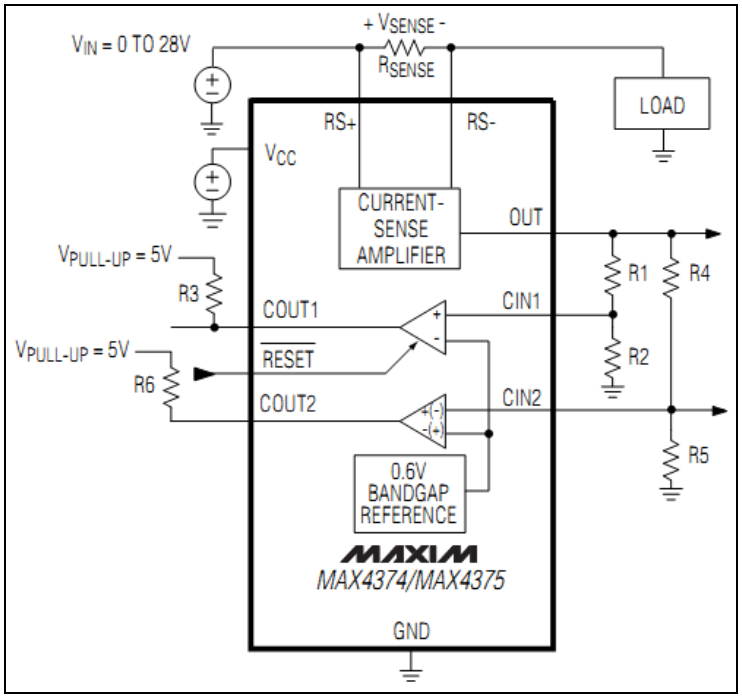


Figure 5.3: Functional block diagram of Max4374 (Maxim Integrated Products, 2000)

internal bandgap voltage reference and a comparator with a latching output to monitor the rail voltage supply line and shut it down without any oscillation through the comparator. The functional block diagram setup of the MAX4374 is shown in Figure 5.3. The maximum current to the load is determined by the value of the sense resistor R_{SENSE}. The voltage drop across R_{SENSE} represents the differential input to the current sense amplifier and should not exceed the maximum value of 300mV.

Three variations of the IC are available. Their difference lies in the value of the gain of the current-sense amplifier: MAX4374T (Gain: 20V/V), MAX4374F (Gain: 50V/V) and MAX4374H (Gain: 100V/V).

An internal comparator is used to compare the output voltage of the current-sense amplifier to a 0.6V bandgap reference. R1 and R2 are used as a voltage divider to ensure an input voltage at C_{IN1} in the order of 0.6V or less which will give a negative output at C_{OUT1}. This is an open collector output which requires a pull-up resistor (R3 in this case) in order to raise the output voltage when the internal transistor is turned off. In case an overcurrent condition occurs, the output

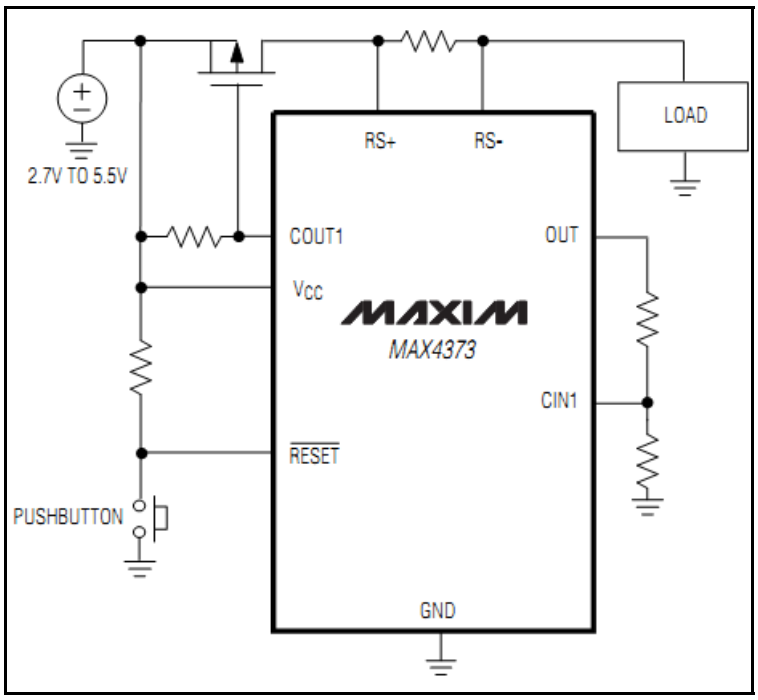


Figure 5.4: MAX4373 overcurrent protection circuit (Maxim Integrated Products, 2000)

voltage of the current-sense amplifier after the voltage divider will exceed the 0.6V limit at C_{IN1} and the comparator's output will toggle from a negative to a positive state.

Figure 5.4 shows the high-side current sense technique implemented around the MAX4374. The circuit arrangement shows the voltage source connected to the supply rail and the gate of a p-channel MOSFET connected to the output of the comparator of the MAX4374. Initially, the MOSFET will be in its ON state which closes the path of the current flow from the source to the load. In this condition, R_{SENSE} will provide a means of measuring the current as discussed.

As soon as an overcurrent condition is detected, the following sequence of events will happen:

- The output of the comparator will switch from a negative to a positive state, causing the MOSFET to switch to its OFF state and opening the current path;
- The load will be disconnected from the supply until a low signal is applied to the RESET input which will restore the line by forcing the comparator's output to a

negative state. This will turn the MOSFET back to its ON state and restore the supply line to the load (Maxim Integrated Products, 2000).

The SHDN (shutdown) output pin which is available on the AT91SAM3U4E will output a low level signal whenever the microcontroller is configured to run in backup mode. When running in this mode, the main supply to the processing core is disconnected and only the backup battery supplies the slow clock oscillators through VDDBU. The high-side current sense circuitry is placed in series with the output of the 3V regulator and the SHDN pin is used to reset the supply line. This lets the MAX4374 perform a double action: current sensing and supply shutdown. The schematic design of the overcurrent circuit is shown in Figure 5.5. The circuit was simulated using Linear Technology's LTspice circuit simulator and values of passive components were based on an estimated line current of 500mA. The calculations and the simulation results are presented and discussed in Appendix B.

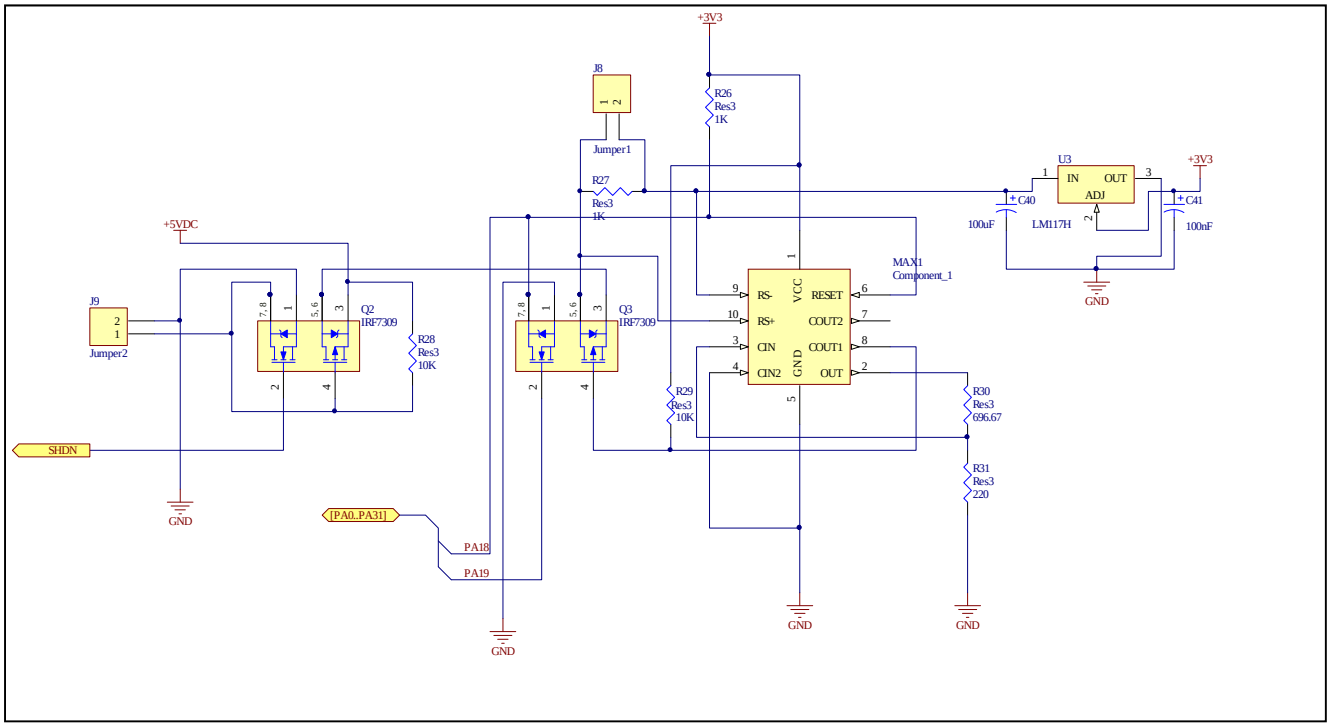


Figure 5.5: Overcurrent protection circuit schematic

5.2.2. JTAG

The JTAG port provides access to the internal core architecture of the microcontroller. Its major function is to allow real-time debugging of the OBC hardware and software. Once a program has been written, compiled and downloaded on the AT91SAM3U4E, single or multi-steps through the code can be performed during runtime.

The boundary scan principle is used where external subsystems and connecting points (I/Os) can individually be toggled to check the functionality of the microcontroller. Breakpoints can be set within the code to verify the state or contents of a particular address and timing can be made accurate in the same way (Catsoulis, 2005). The JTAG port consists of five signal lines, each performing a dedicated function as indicated in Table 5.3.

The AT91SAM3U4E microcontroller supports JTAG and a standard 20-pin connector is used to connect the microcontroller to an emulator (in-circuit emulator or ICE) for programming and debugging purposes. The schematic diagram showing the connection from the microcontroller pins to the JTAG connector is presented in Figure 5.6.

Resistors R1 through R5 are used as pull up resistors. They are required to avoid a floating state on the signal lines since the JTAG input and output pins have an open collector internal configuration. C1 and C2 are decoupling capacitors.

Table 5.3: JTAG signal lines and functions (Catsoulis, 2005).

Pin ID	Name	Function
TCK	Test Clock	Synchronises the initial state machine operations.
TMS	Test Mode State	Sampled at the rising edge of TCK to determine the next state.
TDI	Test Data In	Represents the data shifted into the device's test or programming logic. Samples occur at the rising edge of TCK when the internal state machine is in the correct state.
TDO	Test Data Out	Represents the data shifted out of the device's test or programming logic and is valid on the falling edge of TCK when the internal state machine is in the correct state.
NRST or TRST	Test Reset	An optional pin which when available, can reset the TAP controller's state machine.

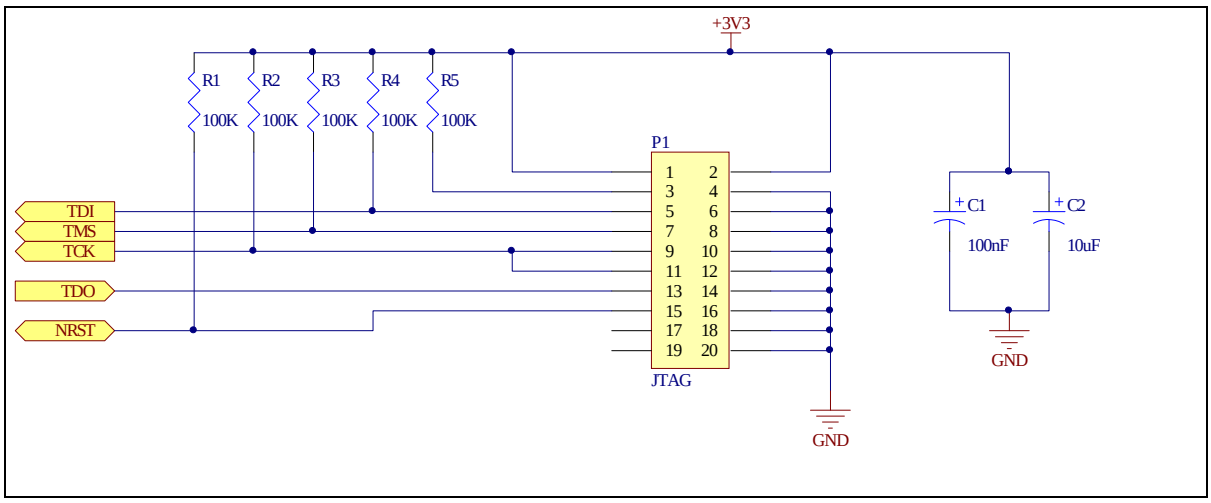


Figure 5.6: JTAG header connection circuit schematic

5.2.3. UART and USART

The need for serial exchange of data in a CubeSat represents a major requirement. This type of transmission allows for one bit to be transmitted or received at a time between two devices and involves the transfer of data over a single wire for each direction. The general operation of serial interfaces consists of converting parallel data from a microcontroller to a serial bit stream and vice versa (Catsoulis, 2005).

The UART and USART are both transmissions of a serial type. Asynchronous transmission implies that both the transmitter and the receiver have individual local clock signals which are determined before the start of the communication as opposed to synchronous, where the transmission is synchronised on both receiving and transmitting ends by means of a common clock.

As their names imply, a USART is capable of both synchronous and asynchronous communication as opposed to a UART which can only perform asynchronous communication. They are capable of data transfers at rates in the ranges between 2400 and 115200 bits per second.

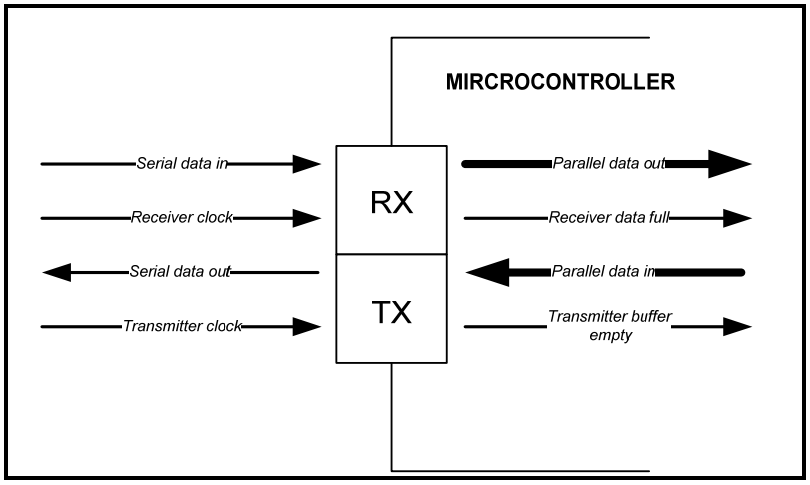


Figure 5.7: Data flow of UART/USART (Catsoulis, 2005)

In a CubeSat architecture, the UART and USART lines enable the data exchange between the OBC and the telecommunication subsystem via a set of modems, hence enabling the downlink and uplink of telemetry or telecommands to or from Earth (Brand, 2007).

The diagram in Figure 5.7 represents the flow of data in a UART/USART interface. Two sections can be identified:

- The transmitter (TX) which converts parallel data from the microcontroller into a serial form for transmission. It is a parallel-to-serial converter with a shift-register loaded in parallel and shifting out each bit sequentially on each pulse of the serial clock.

- The receiver (RX) which converts a serial bit stream from external subsystems to parallel data for the microprocessor. It accepts a serial bit stream and loads it into a shift register which is read out in parallel by the microcontroller (Catsoulis, 2005).

The UART and USART mode of communication transfers data in blocks of bits (usually 8 bits), also called frames, which are user-configurable in terms of contents. The USART can operate in both synchronous and asynchronous modes, depending on the application (Eady, 2004).

The AT91SAM3U4E has a total of four USARTs and one UART which are incorporated on-chip. The RS-232 interface standard for serial communication is generally adopted for UART and USART transmission in microcontrollers. It is an unbalanced interface which means that the voltage level of a data bit being transmitted is referenced to a local ground. The interface is negatively active, meaning that a logic high falls in the range of -5V to -15V (typically -12V), and a logic low falls between +5V and +15V (typically +12V). These logic voltage levels differ from one system to another and transistor-transistor-logic (TTL) is often used as a standard. In order to implement the RS-232 interface on the OBC subsystem, level shifters need to be incorporated.

The MAX3232 is an IC from Maxim Integrated Circuit that acts as a charge-pump based voltage generator in order to reach the -12V and +12V required for bit reading in a normal RS-232 transmission (Catsoulis, 2005). It is done with no additional external voltage regulator but only by connecting pump capacitors as shown in Figure 5.8.

A more detailed section of the datasheet of the MAX3232 can be found in Appendix C where typical values for the charge pump capacitors are given. The serial to parallel and parallel to serial conversions are done within the microcontroller by the on-chip UART and USART converters. In order to add flexibility to the OBC architecture being developed, a total of two USARTs and one UART were implemented in the design in case more than one serial communication line is required using these protocols.

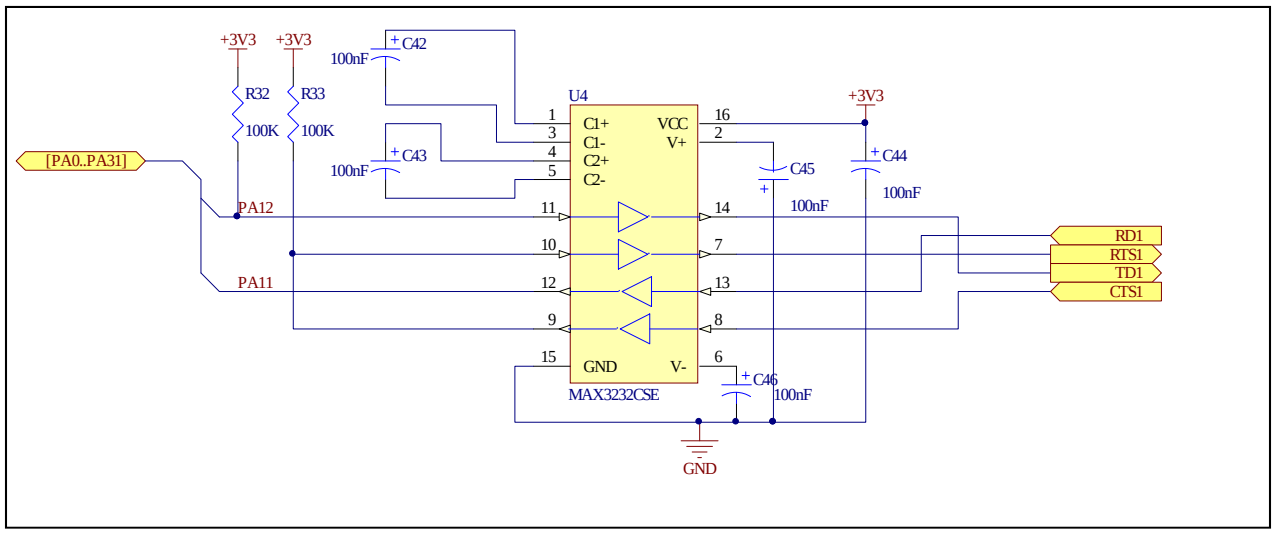


Figure 5.8: MAX3232 setup with DB9 interface to AT91SAM3U4E

5.2.4. Universal Serial Bus (USB)

The AT91SAM3U4E microcontroller embeds an integrated USB2.0 Transceiver Macrocell Interface (UTMI) device, allowing faster serial upload and download of data when needed and, hence, saving space in the implementation. The presence of a high speed peripheral bridge in the architecture makes it possible to reach serial data transfer rates up to 480 Mbps which is qualified as high-speed transfer (Universal Serial Bus Specification Revision 2.0, 2000).

The USB device is associated with a PLL which is used as a frequency synthesiser, giving the ability to run the UTMI at a frequency multiple of the main oscillator (Chenakin, 2010). Besides the JTAG interface, the AT91SAM3U4E's internal architecture can be accessed using in-system programming (ISP) via the USB port. The microcontroller embeds 16kB of ROM of which half is allocated to the smart Smart ARM Microcontroller-Boot Assistant (SAM-BA). This method allows for fast and secure memory programming and can be used as an alternative to JTAG programming. In this instance, the host computer which should be running the SAMBA graphical user interfaces, can display memory and peripherals contents but does not include a debugging option available (Atmel Corporation, 2006).

A type B USB port is used on the OBC board and the schematic diagram is shown in Figure 5.9. Resistors R16 and R17 form a 90 Ω differential impedance with the 5 Ω output impedance of the high-speed channel drivers. Plug-in detection is implemented by the divider bridge made by R18 and R19 from VBUS (5V) which is lowered to the required value of 3.3V. C37 and C38 are bypass capacitors.

5.2.5. Two wire interface (TWI)

The TWI, also known as I²C, is a half-duplex, synchronous, multi-master bus which was developed by Philips Semiconductors to provide communications link between multiple integrated circuits (ICs). An on-chip interface is embedded in every TWI compatible device, allowing data transfers to occur (Philips Semiconductors, 2003). The bus only requires two bi-directional signal lines:

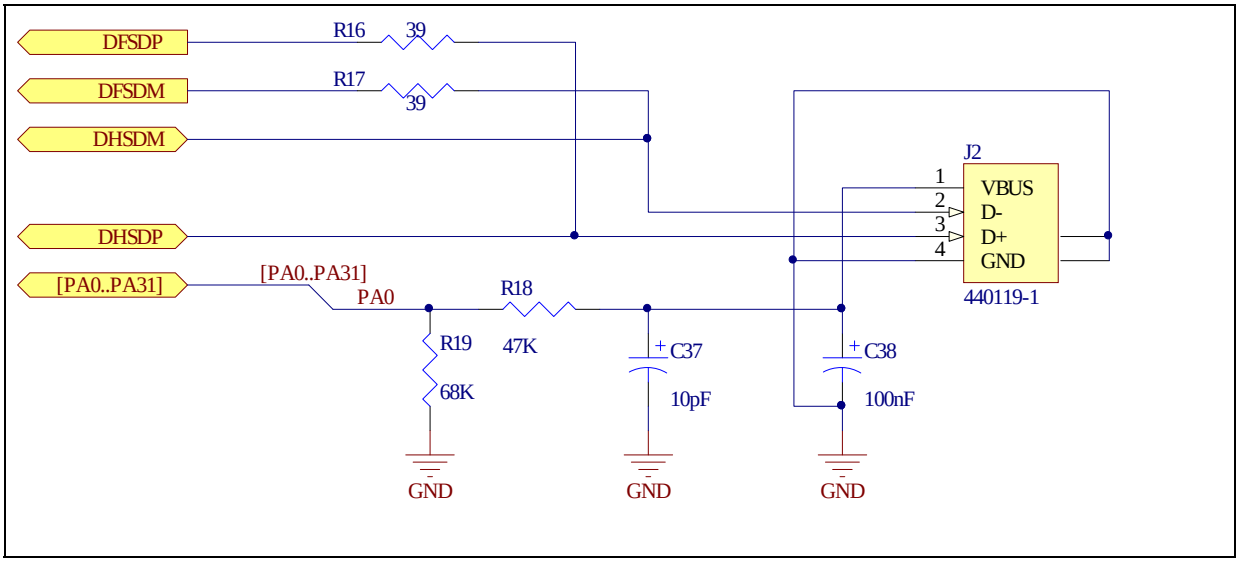


Figure 5.9: USB connection schematic diagram

- Serial Data (SDA): used to transfer data between the master and the slave(s);
- Serial Clock (SCL): provides a synchronous clock between the master and the slave(s).

The SDA and SCL lines are pulled to logic high by means of pull-up resistors and are controlled by the master IC which will normally be the microcontroller, via open-drain drivers. More than one slave device can be connected to the bus and the master IC initiates communication and provides the clock signal (SCL) and transfer rates of up to 400kbps can be reached. Each device on the bus including the master is identified by an individual 7-bit or 10-bit address which enables the line for the specified slave and the data transfer is based on an 8-bit (1 byte) sequence (Philips Semiconductors, 2003).

Two TWI are provided with the AT91SAM3U4E. In order to prove functionality of the bus, it was decided to establish communication with a TWI compatible device. The MCP9800 temperature sensor from Microchip® is an integrated TWI sensor which meets the hardware requirements of the OBC and was chosen to accomplish two functions:

Table 5.4: MCP9800 features

Characteristic	Value
Operating voltage	2.7V to 5.5V
Operating current	200μA
Accuracy	User selectable resolution between 9 and 12-bit
Temperature range	-55°C to 125°C
Device slave address	0b100100

- Monitor the OBC onboard temperature;
- Evaluate the TWI on the AT91SAM3U4E microcontroller.

Table 5.4 gives its specifications in more detail (Microchip Technology Inc, 2010). In addition to the SDA and SCL lines, the MCP9800 has an alert pin which outputs an alert signal when the ambient temperature goes beyond a user-programmed temperature limit. As for any TWI communication, a standard process is followed to accomplish a data transfer:

- The master generates a start condition (S) to initiate data transfer;
- The master sends the 7-bit slave device address with a read or write bit on the bus to identify which operation will be conducted;
- The addressed slave device sends an acknowledge (ACK) bit to the master to confirm the reception of the previous byte if a write operation was selected, otherwise the master sends an ACK signal to the slave in case of a read operation;
- The byte to be transferred is loaded onto the bus and sent or received by the master or the slave and for each byte transferred, an ACK bit is sent from the slave to the master or vice versa;
- The communication is terminated by a stop (P) condition generated by the master (Philips Semiconductors, 2003).

This process is illustrated in Figure 5.10 for a read operation during which the master must also signal an end of data to the slave by generating a “not ACK” bit (NAK). This allows the slave device to release the data line in order for the master to generate the stop condition.

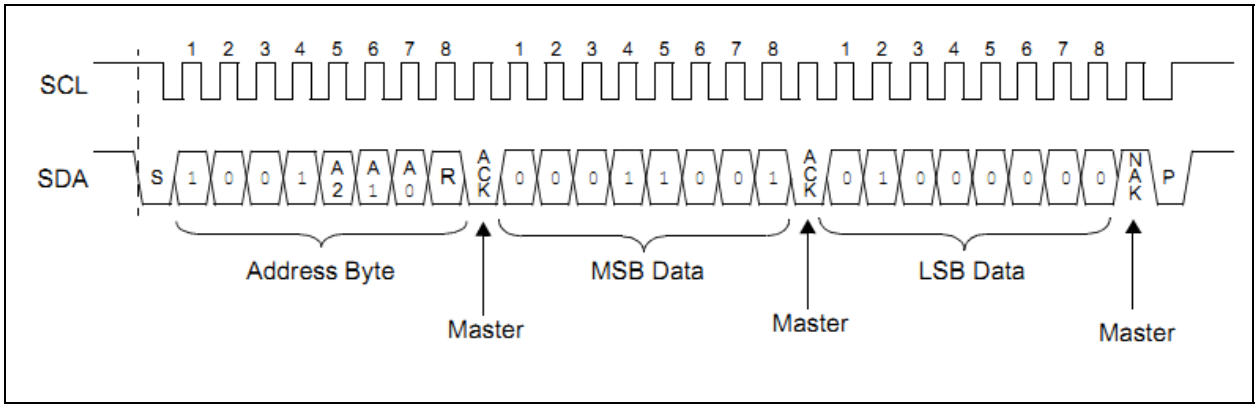


Figure 5.10: Read operation from MCP9800 to master IC
(Philips Semiconductors, 2003)

The MCP9800 is made of four registers that can be configured by the user for a specific operation:

- The ambient temperature register (T_A);
- The temperature limit-set register (T_{SET});
- The temperature hysteresis register (T_{HYST});
- The device configuration register (CONFIG).

Each register is set independently and is accessed by sending an 8-bit register pointer to the device which has to initially be configured for a write operation. Only the two least significant bits (LSB) are used to select the register and the other bits have to be cleared. Each register's function is elaborated in Table 5.5 and register pointer bits associated with them are also shown.

One of the two available TWI buses on the AT91SAM3U4E was used to connect the MCP9800 as illustrated in the schematic diagram shown in Figure 5.11. The alert pin previously mentioned connects to a GPIO configured as an input to acknowledge the alert signal in case the ambient temperature drifts from a preset threshold value. The resolution of the ADC is set in the configuration register. The device is factory calibrated with a 9-bit resolution.

Table 5.5: MCP9800 registers description and functions

Register	Attribute	Function	Pointer bits	Size
Temperature register (T_A)	Read only	Access the ambient temperature. ADC data is loaded in parallel in this register.	0b00	16-bit; only uses 9 to 12-bit data in two's compliment.
Temperature limit-set register (T_{SET})	Read/Write	Provides user-programmable temperature limit.	0b01	16-bit; only uses the 9 MSBs.
Temperature hysteresis register (T_{HYST})	Read/Write	Provides user-programmable hysteresis.	0b10	16-bit; only uses the 9 MSBs.
Configuration register (CONFIG)	Read/Write	Provides user- access to the device's different features.	0b11	8-bit; each bit sets a feature.

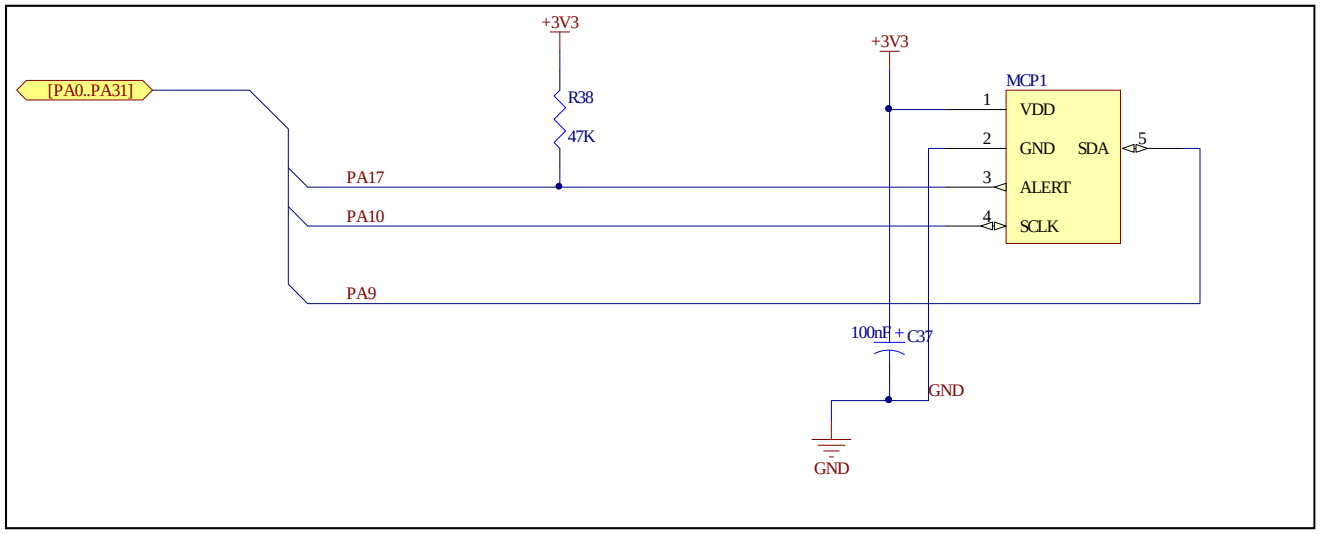


Figure 5.11: MCP9800 TWI temperature sensor schematic diagram

At power up, the ambient temperature is set to 0°C, the temperature limit is set to 80°C and the hysteresis is set to 75°C. Additional settings associated to each one of the register can be found in the device's data sheet which is provided in Appendix C.

5.2.6. Serial peripheral interface (SPI)

The AT91SAM3U4E also provides for an SPI data bus which will add to the features and flexibility of the OBC in terms of serial data transfer. This bus is also known as four wire interface and can be used to interface the microcontroller with external memory modules amongst other devices, such as LCD drivers, audio chips sensors, and other processors (Catsoulis, 2005).

The SPI transmission is a synchronous protocol where all transmissions are referenced to a common clock generated by the processor which is identified as the master. The receiving peripheral is known as the slave and makes use of the same clock signal to synchronise the acquisition of the serial bit stream of data. One master can connect to several slave devices on the same bus and unlike the TWI, the slave device is not identified by pointing at its own address but instead an assertion is made to the slave's chip select input. Four data lines can be found on an SPI device as shown in Figure 5.12:

- Master in slave out (MISO): This is a serial data stream generated by the master and received by the slave during a write operation;
- Master out slave in (MOSI): This is a serial data stream generated by the slave and received by the master during a read operation;
- Serial clock (SCLK or SCK): This signal synchronises data transfer between the master and the slave devices;
- Chip select (CS): This line is driven by using a spare I/O pin from the master which makes the selected slave active on the bus (Maxim Integrated Products, 2007).

The operating principle is based on the shift registers exchange principle where the master starts a byte transfer by writing it to its shift register and as the register transmits the byte to the slave on the MOSI line, the slave transfers the contents of its shift register back to the master on the MISO line, making it a simultaneous read and write operation.

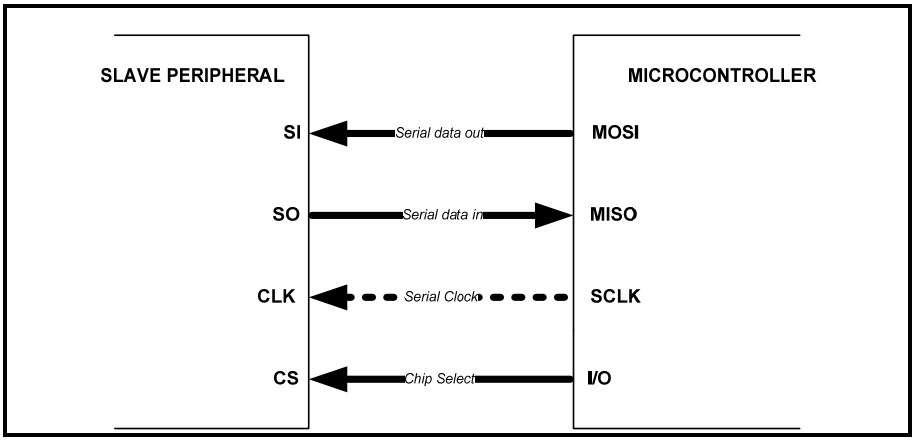


Figure 5.12: Basic master-slave SPI configuration

In case a read operation is to be performed, a dummy byte is transferred from the master in order to initiate a slave transmission and similarly, for a write operation, the master will ignore the bytes it receives and only empty out its register (Catsoulis, 2005).

In previous OBC designs, SPI was mainly used for interfacing the memory module with the microcontroller. The AT91SAM3U4E provides a 16-bit wide external bus interface (EBI) as well as an 8-bit wide high speed multimedia card interface (HSMCI) to directly interface memory modules with the microcontroller. This allows the SPI to be reserved for other purposes. The SPI lines will run directly from the microcontroller to the header pins of the OBC to make them available for future use.

A total of four chip select lines equipped with external decoder support permit communication with a total of 15 peripherals at a time. The interface is reconfigurable (microcontroller as master or slave) and the bus interface has a data length of 8 to 16-bit programmable per chip select.

5.2.7. SD/MMC Card

The memory module in the CubeSat architecture is generally associated with the OBC and according to Hidayat (2010), it serves two major functions:

- Recording and storing housekeeping parameters and payload data for downlinking;
- Storing flight software updates from the uplink channel.

The choice for the SD/MMC solution over a memory chip is highly motivated by the presence of an EBI on the AT91SAM3U4E microcontroller. The SD/MMC solution is an integrated flash-based removable memory card with serial and random access capability. Its reliability in terms of security, performance and capacity has proven to be efficient in many handheld devices and its bus interface which can easily be integrated in any design, has been adapted to work with most microcontrollers on the market (SANDISK Corporation, 2003).

Two alternative protocols can be used for communication at a variable clock rate:

- SPI mode: As described in section 5.2.6 of this chapter, this interface provides a serial synchronous bus capable of interfacing a host controller (master) with several selectable slave devices with four data lines (CLK, CS, MOSI and MISO);
- SD mode: This mode is based on command and data bit streams initiated by a start bit and terminated by a stop bit, and a dynamic configuration of the number of data lines from 1 to 4 bi-directional parallel data signals (Kingmax Digital Inc., 2000).

Preference for the transmission mode was given to SD mode which proved to be much faster (parallel data transmission). In addition to speed, the SD mode allows full support for wait states, hence eliminating the need for continuous polling which consequently increases power consumption. A block diagram of the SD/MMC's internal structure is shown in Figure 5.13 and each pin is described in Table 5.6 for the SD mode (SANDISK Corporation, 2003).

Table 5.6: Signal description on SD card pins in SD mode

Pin N°	Name	Description
1	CD/DAT[3]	Data line 3
2	CMD	Command line
3	VSS1	Ground
4	VDD	Supply voltage
5	CLK	Clock
6	VSS2	Ground
7	DAT[0]	Data line 0
8	DAT[1]	Data line 1 or Interrupt (optional)
9	DAT[2]	Data line 2 or Read Wait (optional)

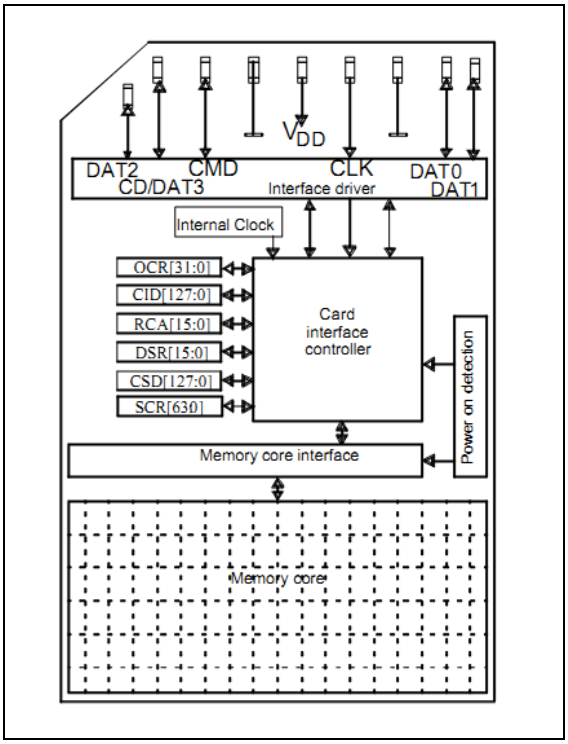


Figure 5.13: SD/MMC Block diagram (Kingmax Digital Inc., 2000)

Once a card is inserted into an SD/MMC socket, the card powers-up and the initialisation phase can start. By default, only the DAT[0] line is used for data transfer and after initialisation, it is possible for the master microcontroller to change the bus width and use additional data lines according to the user's choice. The clock starts running and the initialising sequence can be sent. It consists of a stream of logical '1's on the CMD line which is responsible for all commands and response tokens when running in SD mode. This is done in order to avoid all possible power-up synchronisation mismatches that may occur between the internally generated clock and the host clock. These commands are typical read and write operations and sent from the host to the card which, in turn, sends a response on the same line before a read/write operation begins on the data lines (Kingmax Digital Inc., 2000).

Data transfers to and from the card are always done in multiple blocks to increase throughput. Each block of data is always appended with cyclic redundancy check (CRC) bits which helps in detecting errors by means of sending a positive or negative acknowledgement to the microcontroller whether or not the check bits that were

generated on the sending end agree with the data at the receiving end. This way, it is possible to detect errors and in case of a negative acknowledgement, a re-read of the block of data is initiated until a positive acknowledgement is received (Peterson, 1961).

A basic multiple block write operation is illustrated in Figure 5.14 showing commands and response tokens on the CMD line and data blocks and CRC bits on the DAT line. A start and stop command on the CMD line are necessary to, respectively, initiate or halt operations on the data bus.

The AT91SAM3U4E is equipped with an MMC/MMCPlus high-speed interface capable of accommodating a total of 8 data lines (8-bit). For availability reasons, it was decided to fall back to a regular 4-bit parallel data bus since the MMCPlus socket was scarce and hard to obtain at the time of the design.

The connection from the SD card socket to the microcontroller is done according to the schematic diagram shown in Figure 5.15 where two additional pins can be identified:

- Card_Detect (CD): Active-low signal to detect card insertion;
- SD_Write _Protected (SD_WP): Active low signal indicating write-protection on the card.

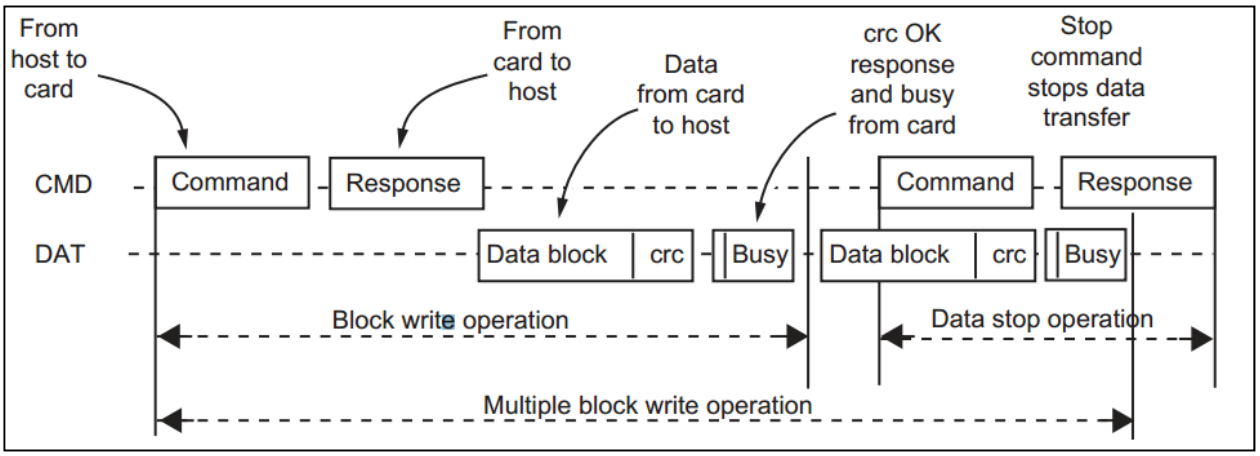


Figure 5.14: Multiple block write operation (SANDISK Corporation, 2003)

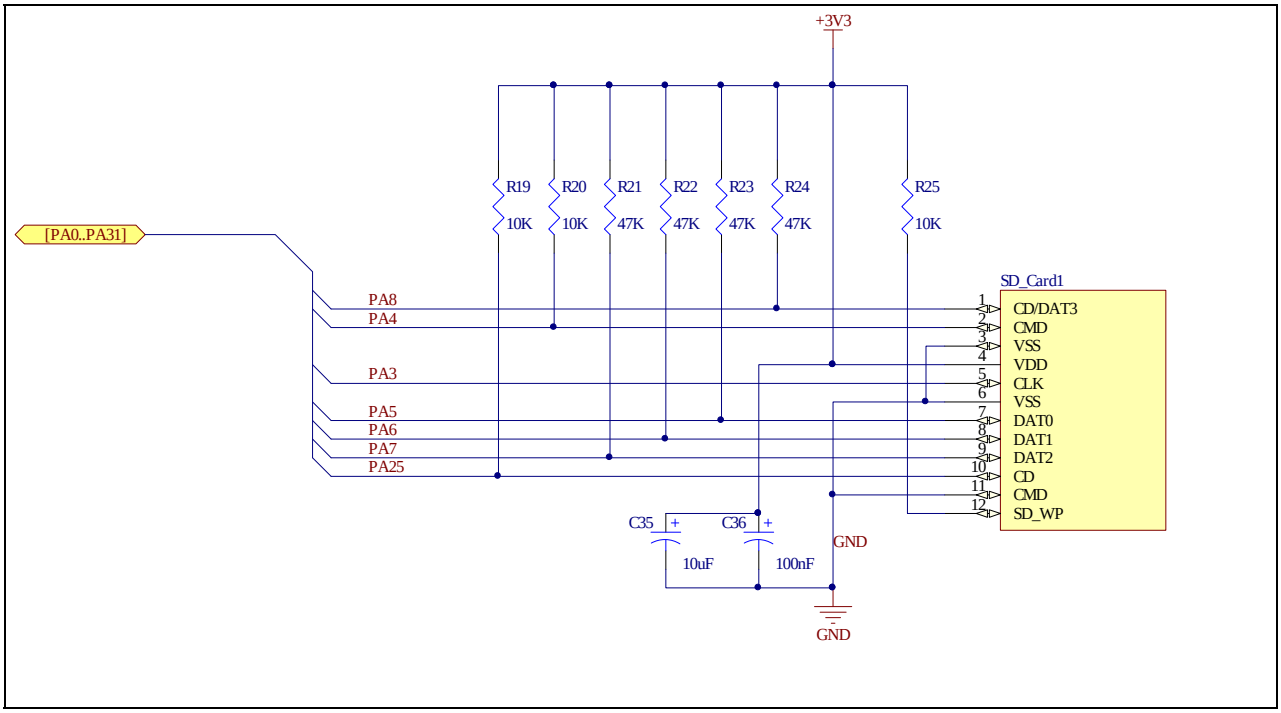


Figure 5.15: SD Card socket schematic diagram

All data lines are active low and require pull-up resistors to avoid floating signals on the lines. Resistors R19 through R25 accomplish this purpose while C35 and C36 are decoupling capacitors. The SD/MMC interface is supported by the DMA controller which minimises the processor intervention for large buffer transfers by enabling a direct interaction between the memory card and the peripheral associated with it. This makes it possible to reach data transfer rates of up to 55MHz (Atmel Corporation, 2009).

5.2.8. Analogue to digital converters

In order to convert analogue voltage levels from different transducers onboard the CubeSat into digital quantities for processing, ADCs are needed. The AT91SAM3U4E embeds two 8-channel ADCs: ADC0 and ADC1. They feature an automatic sleep mode after conversion on enabled channels and possess multiple hardware and software trigger sources to automatically wake up the ADC when a conversion is needed. They individually have additional characteristics:

- ADC0: It has a 12-bit resolution and a maximum sample rate of 1 MSps and can be reconfigured to 10-bit and 2 MSps, respectively, if need arises. In addition to the

high resolution and data sampling rate, it is possible to select between a single ended or differential input voltage mode and the reference voltage is externally set for better accuracy on low input voltages.

- ADC1: It has a 10-bit resolution and a maximum sample rate of 460 KSps, also reconfigurable to 8-bit at 660 KSps respectively. It only gives ability to perform single ended voltage measurement and its reference voltage is also set externally (Atmel Corporation, 2009).

The ADC has a set of 13 registers, each performing a specific function on each channel. By setting the analogue control register bits, it is possible to configure an ADC channel's parameters (gain, operating mode, input offset and bias current control) depending on the ADC application. There is also a set of registers available which allow to enable or disable unused ADC channels and to provide interrupt handling (Atmel Corporation, 2009).

5.2.9. Timer/ counter (TC)

In their basic form, timers are counters that increment or decrement based on a clock cycle and a pre-scaler (clock divider). The advantage of using a timer lies in the fact that the clock input and the operation of the timer are totally independent of the running program. The resolution (Res) associated with a timer corresponds to the maximum value the timer can reach (Max_Val_Cnt) and is calculated by the following formula:

$$Max\_Val\_Cnt = 2^{Res} - 1 \quad (1)$$

When the maximum value is reached, an overflow occurs and the counter is reset to zero. A flag in the status register is set and can be used to trigger an event or set an interrupt. It is possible to pre-load the counter with a value in order to reach the overflow status earlier than the maximum count and hence, shortening the cycle and making it possible to generate different time intervals. Timers can be configured to count up or down (Atmel Corporation, 2002).

The AT91SAM3U4E has a total of three timer-counter (TC) channels, each with a resolution of 16-bit which according to equation (1) makes their maximum count:

$$Max\_Val\_Cnt = 2^{16} - 1$$

$$=65535 \quad (2)$$

Each channel is user-configurable in terms of the pre-scaler value and can be synchronised to either the internal clock (five inputs available) or the external clock (three inputs available). The main advantage of having multiple timers is to give flexibility between applications. These timers can be used in a range of functions including frequency and interval measurement, pulse width modulation, pulse or signal generation and delay timing (Atmel Corporation, 2009).

In CubeSat applications, timers are ideal to use since synchronisation between different subsystems can occur at different times intervals which can be generated by the timers at different frequencies and with different pre-scaler values.

5.2.10. Pulse width modulation (PWM)

PWM is a technique used in analogue speed control of actuators. It can be defined as a way of digitally encoding analogue signal levels. This modulation technique generates variable width pulses which represent the amplitude of an analogue input signal. By using the timer counters previously discussed to generate triangular signals, it is possible to alter the duty cycle of the square wave according to a specific analogue signal (Barr, 2001).

The AT91SAM3U4E embeds an on-chip PWM controller with 4 channels, each using one of the 16-bit TCs. Each channel is independently programmable and can also run in synchronous channel mode where the same TC is shared (Atmel Corporation, 2009).

In order to generate a PWM sequence on an output, it is necessary to follow these general steps:

- Set the period of the timer providing the modulating square wave;
- Set the on-time of the pulse in the control register;
- Select the output (direction) of the PWM;
- Start the timer;
- Enable the PWM controller (Barr, 2001).

The main advantage of using the PWM technique to control actuators in CubeSats applications is that the output signal remains digital, which means that it is either a '1' or a '0' that is being sent, hence minimising the noise and radiation effects which could be devastating in case of analogue signal transmission to a load. The PWM signal can be used to precisely control magnetorquer rods or servo motors in the ADCS subsystem of a CubeSat (Wertz & Larson, 2008).

5.2.11. I/O peripheral set

Each of the previously mentioned embedded peripherals of the AT91SAM3U4E microcontroller belong to an I/O set which is controlled by three 32-bit parallel input/output (PIO) controllers: PIOA, PIOB and PIOC. Each controller is fully programmable through individual registers and each I/O line is multiplexed for two peripheral functions (A and B) besides being used as a GPIO.

An internal programmable 100kΩ pull up resistor is present on every I/O line and can be managed by the PIO controllers. After power-up, all the I/O lines are inputs by default and pull-up resistors are enabled. Appendix C.3 gives a detailed table of all three PIO controllers and each I/O line as well as their respective peripheral functions (Atmel Corporation, 2009).

5.3. PCB integration

The transfer from schematics to PCB was made possible by the electronic computer aided design (ECAD) software called Altium Designer. This software package works on a unified approach which allows importing schematic designs directly onto PCB design and also allows a visualisation in 3D. The components libraries are available per manufacturers and custom defined component libraries can also be added when required.

A double sided board with two layers was preferred for the OBC design due to the ease of access presented to both the power and the ground tracks. This approach also minimises the overall amount of through-holes on the board and saves a large amount of space, making it possible to fit all the components on the physical space provided.

In addition to the number of layers to be used, it was decided to place all ICs and active components on the top layer in order to make direct connections to the microcontroller and use the bottom layer strictly for passive components such as resistors and filtering capacitors. The AT91SAM3U4E microcontroller which comes in a 144 pin QFP package represents the main component on the board and was placed in the centre of the top layer.

As recommended by Schmitz and Wong (2007), it is important to place all bypass capacitors for the microcontroller on the bottom layer directly under it and as close as possible to each supply pin. This is done to ensure steadiness of each supply pin which can easily be driven to an unstable state due to fast switching signals around those pins. The line inductance and series resistance which are proportional to the length of the conducting track are reduced and the voltage delivered is cleaner and steady.

The power block is placed at the bottom left corner of the top layer to facilitate the distribution of the power bus. For power consumption measurement purposes, a number of removable jumpers are made available to measure the power drawn by the core in different power mode configurations. An additional jumper is available to bypass the overcurrent protection circuitry and run the OBC directly from the main power supply.

The SD card socket can be found on the bottom layer of the board at the top right corner in order to facilitate insertion of the card as well as routing of the data bus from the microcontroller. Other connectors include the header pins, the USB socket, the JTAG interface and the battery holder. These are strategically placed on the different sides around the PCB to facilitate access to their respective peripherals.

The routing of the PCB was done using different track widths for data and address bus tracks (0.2mm), for interconnecting tracks (0.5mm) and for power and ground tracks (1mm) as shown in Figure 5.14.

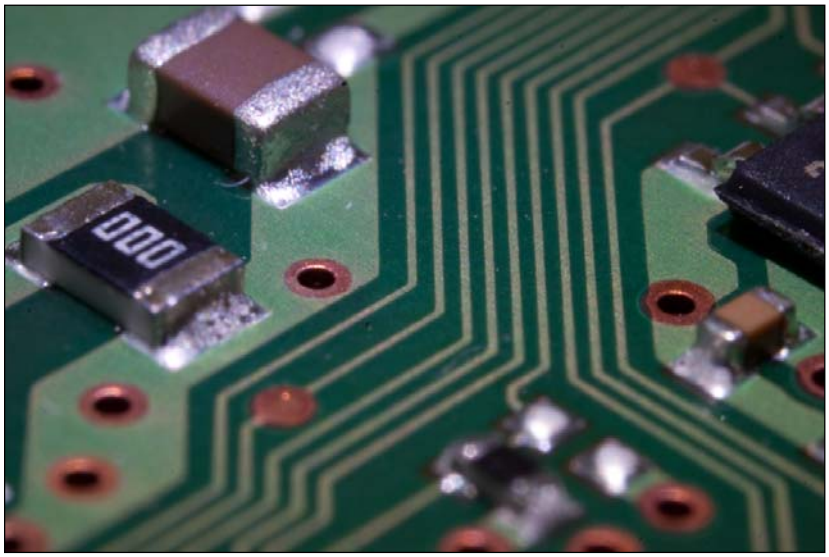


Figure 5.16: Close up of power/ground and data tracks on PCB

A solid copper polygon pour was placed over both the top and bottom layer with a net connection to the ground plane in order to reduce the chances of potential electromagnetic interference.

The complete schematic of the OBC can be found in Appendix D which also contains views of the top and bottom layers during the design phase and after the components were populated on the physical PC/104 board.

5.4. Summary

General specifications of the OBC were highlighted according to the characteristics of the chosen microcontroller which were presented in Chapter 4. An overview of the OBC subsystem was presented, showing every component associated to it.

The power system block was presented and analysed. The MAX4374 was selected as the active device for this section of the OBC and its operations were investigated.

Peripherals and interfaces of the microcontroller to be implemented on the OBC were presented in terms of performance and possible application on a CubeSat platform. When applicable, operating procedures were provided in addition to detailed schematic diagrams.

Finally, the PCB implementation of the design was presented and the placement of components on the PC/104 form factor board was justified.

In the upcoming chapter, a series of tests will be conducted in order to evaluate the designed prototype. These tests will essentially be based on the functionality of some peripherals and features of the OBC.

CHAPTER SIX

DESIGN VERIFICATION

6.1. Introduction

In order to validate the OBC design, each feature that was implemented in the architecture needs to be tested for functionality. It is mandatory to provide an experimental setup to gather data from each test and to be able to evaluate and troubleshoot the hardware whenever it is necessary. For each test, the microcontroller is loaded with a program which is written and compiled with the aid of an integrated development environment (IDE) tool. The program is written in a high level programming language (C or C++).

This chapter presents the experimental setup and the IDE which was adopted in order to conduct the tests. The results of each test are also presented.

6.2. Experimental setup

The proposed experimental setup in order to evaluate the OBC is presented in Figure 6.1. A personal computer running Windows® XP provides an interface to the OBC to simulate the communication with other subsystems. It is also used as a running platform for the interactive user interface and software necessary to provide the code to program the microcontroller. IAR's Embedded Workbench for ARM (EWARM[12]) was the chosen IDE to program the microcontroller. This software incorporates chip-specific optimising capabilities and integrates the necessary tools for embedded hardware and software development which are:

- A C/C++ compiler and an assembler;
- A linker;
- A text editor;
- A librarian and a project manager;
- A high-level language debugger (C-SPY®) (IAR Systems, 2007).

[12] See http://www.iar.com/en/Products/IAR-Embedded-Workbench/ARM/

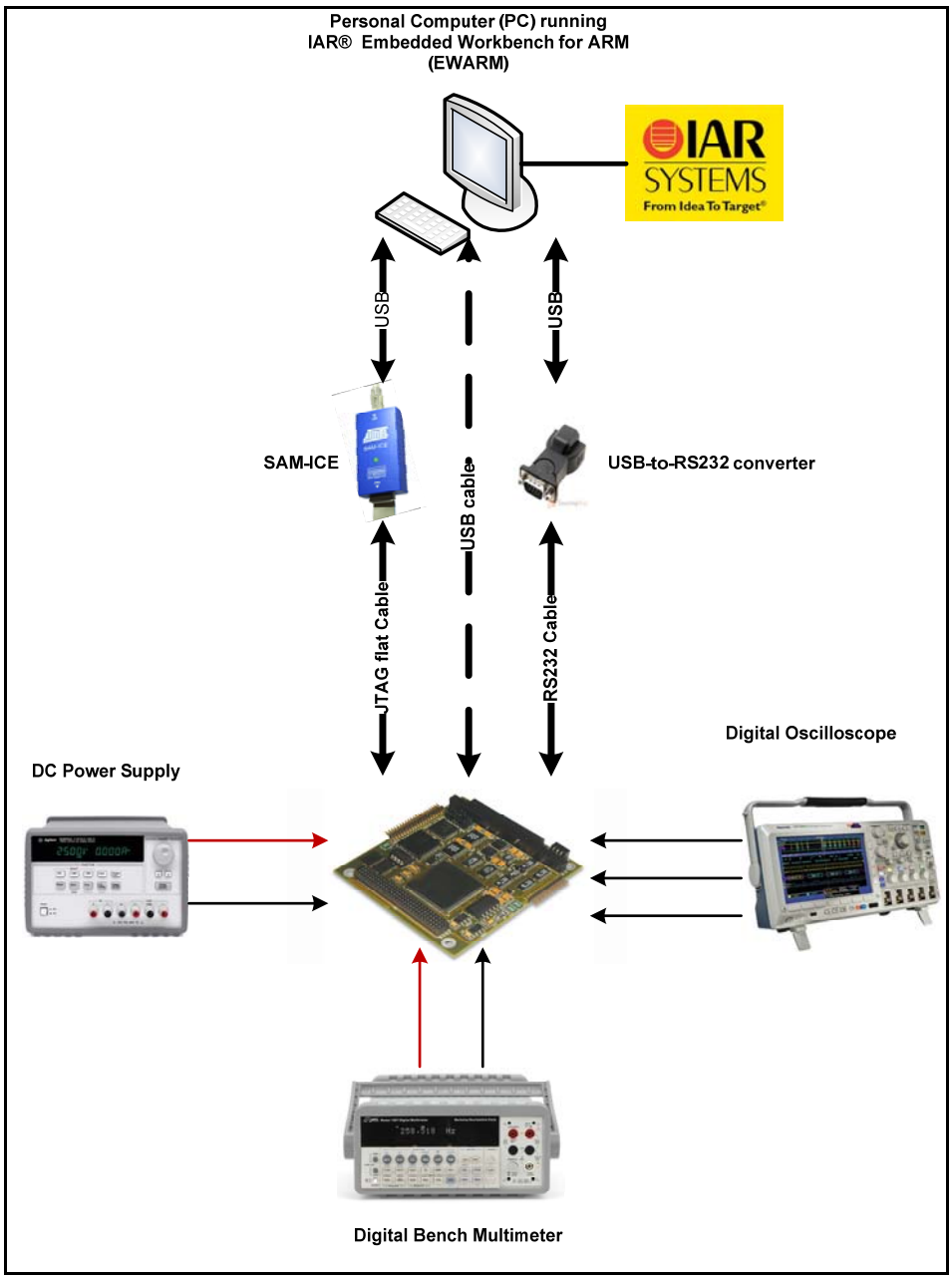


Figure 6.1: Experimental setup

The choice for IAR EWARM was highly motivated by the advantages it presents when compared to other IDEs. The following attributes justify this choice:

- A modular and extensible IDE with powerful project management component allowing more than one active project in one session;
- Extensive ARM-core devices technical support;

- Support for Thumb®1 and Thumb®2 instructions sets;
- Available C-SPY driver for JTAG target system;
- RTOS support by C-SPY debugger (including Salvo);
- An unlimited number of breakpoints in flash memory with interrupt visualisation when debugging;
- A powerful macro assembler with built-in C language pre-processor;
- ISO/ANSI C and C++ libraries with full source included with listing of entry points and symbolic information (IAR Systems, 2007).

The debugging and downloading interfaces between the PC and the OBC are implemented using the SAM-In circuit emulator (SAM-ICE). This emulator is designed by Atmel for use exclusively with AT91ARM cores of which the Cortex M3 is a close implementation. It has a JTAG connector integrated with a 20-pin connection compatible with the one offered by the OBC's AT91SAM3U4E microcontroller interface. By emulating the microcontroller, the ICE allows one to access, examine and change the contents of the registers, memory and I/Os of the targeted microcontroller after programming. Similarly, it is possible to stop the program at precise locations or at given conditions and observe the state of internal registers at that particular point in the code, hence making the software development less time consuming (Ganssle, 1999).

As stated in section 4.3 of Chapter 4, serial communication is of high importance in a CubeSat. In order to simulate data transmission from the OBC (CubeSat) to the ground station, a UART link is established between the OBC and the PC. In a CubeSat, the UART transmission link is interfaced with a modem which in turn is connected to a transmitting or receiving antenna via the transceiver subsystem. The transmission and reception simulations are important in order to validate the functionality of the UART and USART interfaces by having the board send data packets to the PC and vice-versa. Due to the absence of a serial DB9 port on modern PCs, a USB-to-RS232 adapter is used between the OBC and the PC, allowing the connection and data transmission to take place.

The high-speed USB2.0 device integrated on the AT91SAM3U is capable of serial data transfer rates up to 480 Mbps and can also be used in conjunction with the internal ROM

for in-series-programming (ISP). A USB connection from the OBC to the host PC is enabled by a USB-to-USB cable.

A regulated bench DC power supply is used to supply power to the OBC board with a fixed voltage of 8V. Onboard regulators on the OBC regulate this voltage down to constant values of 3.3V and 5V.

Digital signals from the OBC are analysed by a digital oscilloscope and the power consumption is measured in different operating modes with the help of a digital bench multimeter.

6.3. IAR EWARM Toolchain

The EWARM modular IDE presents an interface divided into a series of windows representing the active workspace as shown in Figure 6.2. A tree representation of the active project containing organised library folders for all files associated with the running project is accessible through the workspace navigation window (see Figure 6.2)

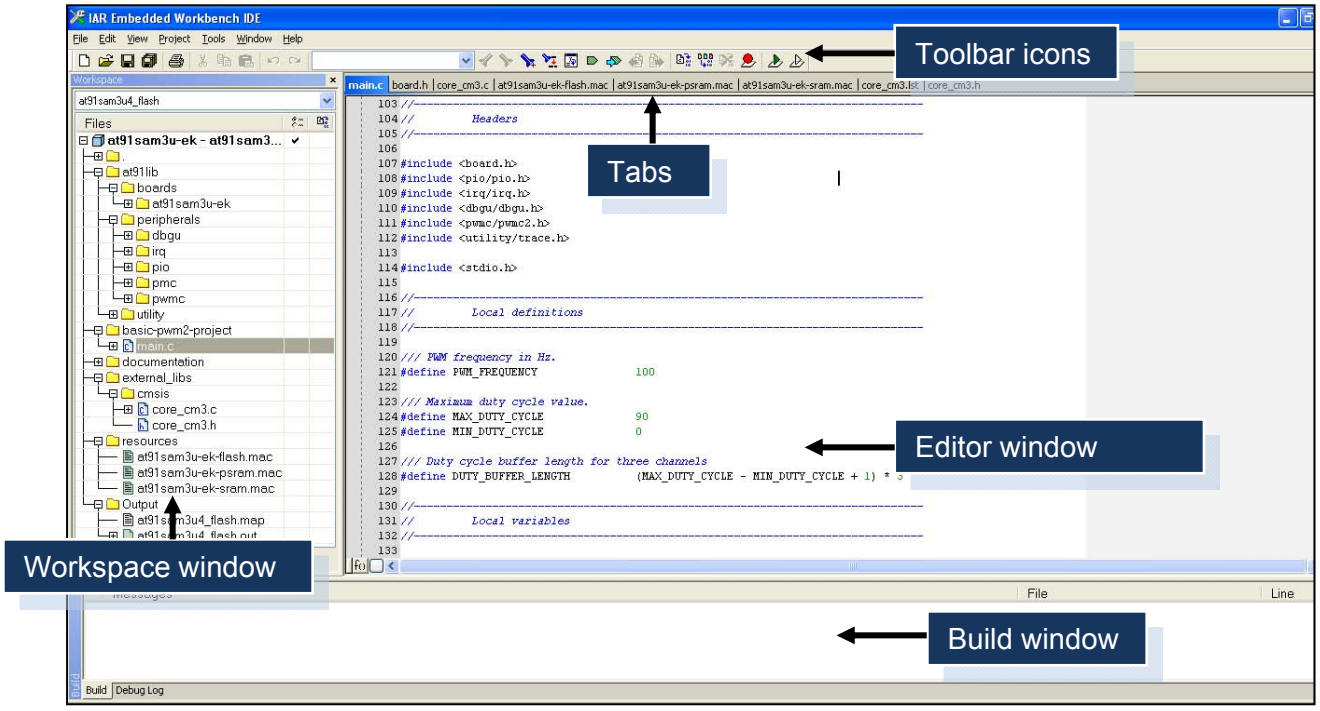


Figure 6.2: IAR EWARM IDE main programming interface

The editor window allows for writing or editing of source files or header files found in the workspace navigation window.

The *AT91lib* directory found in the active project tree is associated with a folder of the same name that appends to the project's working directory path and includes all the library files called in the main source file. Three sub-directories can be found in the *AT91lib* folder: boards, peripherals and utilities. Each of these directories contains files which accomplish a particular function.

The "boards" sub-directory contains all the header files necessary to configure the microcontroller in terms of features, software application programming interface (API) definitions, operating frequency and I/O pins definitions. The header file *board.h* is particularly important because it contains all of the definitions and functions for using the AT91SAM3U4E-dependant I/O pins and external components interfacing with it. It also contains definitions for the operating frequencies, ADC clock frequencies and I/O pins.

An example of frequency definitions for the main oscillator and the 10-bit ADC are as follows:

```
// Frequency of the board main oscillator (in Hz)
#define BOARD_MAINOSC         12000000
// ADC clock frequency, at 10-bit resolution (in Hz)
#define ADC_MAX_CK_10BIT      5000000
```

The I/O pins definitions follow a different format which associates a pin to a constant according to the following convention:

- `PIN_*` for a constant defining a single pin instance;
- `PINS_*` for a list of pin instances.

An example for the SPI0 MISO pin declaration and the SPI0 pin instances declaration will be in the following form:

```
// SPI0 MISO pin definition (Refer to a single pin)
#define PIN_SPI0_MISO  {1<<13, AT91C_BASE_PIOA, AT91C_ID_PIOA, PIO_PERIPH_A,
PIO_DEFAULT}

// List of SPI0 pin definitions (MISO, MOSI & SPCK)
#define PINS_SPI0      PIN_SPI0_MISO, PIN_SPI0_MOSI, PIN_SPI0_SPCK
```

In order to re-assign functions to particular pins, these definitions must be changed accordingly.

The “peripherals” sub-directory contains configuration files for peripherals directly associated with the running project. The files contained in this location should never be changed as they define every function that is necessary to run the associated peripheral.

Finally, the “utility” sub-directory contains three files necessary for configuring output methods for reporting debug information, warnings and errors. These files are in no case mandatory for the program to be compiled and can be left out but should never be modified.

The main source file is located in a folder containing the project’s name. The file is accessible through the workspace navigation window and contains local definitions, variables and functions to be manipulated through the program.

6.4. Experimental tests and results

The points to follow present the series of tests that were conducted on the OBC prototyping board. These tests are run on each peripheral that is present on the board. Measurements are also made in regard to the power consumption in different modes of operation.

6.4.1. Power considerations

After the integration of the final prototype, it is necessary to verify the functionality of the overcurrent protection circuit in case a short circuit condition arises. The overall power consumption figures for the OBC subsystem when running in different power modes should also be measured.

6.4.1.1. Overcurrent protection circuit

In section 5.2.1 of Chapter 5, details are provided regarding the overcurrent protection circuit design around the MAX4374 and Appendix B includes related calculations, simulations and results about this section of the OBC.

The circuit was physically tested using two resistors connected in a parallel configuration, one of which represents the OBC's load impedance and the other, the overload. With an assumption of a total drawn current of 500mA and a line voltage of 3.3V, the load resistance was calculated to be:

$$R_{load} = \frac{V_{line}}{I_{max}}$$
$$= \frac{3.3}{0.5}$$
$$= 6.6\Omega \quad (3)$$

A value of 6.8Ω was used for both the load and the overload resistances. The parallel resistor configuration implies that a higher current value will be drawn by the load which is halved to 3.4Ω. This condition causes the sense resistor to have a higher voltage drop across it and the input to the comparator to be above the recommended value of 0.6V.

Initially, the P-channel MOSFET driven by the output of the comparator is switched on, allowing power to flow from the power supply to the OBC circuitry through the sense resistor. The increase in current caused by the addition of the parallel load causes the output of the comparator to go to a high state, hence switching off the MOSFET and leaving the power line in an open circuit state. As the output of the comparator changes its state, the supply line is disconnected. In order to restore the power to the circuit, an incoming pulse from the OBC which now runs in BACKUP mode must trigger the RESET input. This is done via the FWUP pin of the AT91SAM3U4E and an N-channel MOSFET after removing the overload resistor.

During power off the microcontroller automatically enters the BACKUP mode where the onboard backup battery takes over from the rail supply and leaves only the internal slow clock oscillator to run. This operation requires a BAT54C Schottky barrier double diode which can only conduct one of the two voltage sources at the time to power the internal slow clock oscillator.

The RESET input of the MAX4374 is directly connected to the FWUP pin of the microcontroller which remains active during BACKUP mode. This pin is used to reset MAX4374 back to its normal operating mode.

6.4.1.2. OBC Power consumption

The different operating modes of the AT91SAM3U4E microcontroller were covered and explained in Section 4.2.4 of Chapter 4. In order to draw power consumption figures for the OBC in these operating modes, current measurements were taken by inserting a digital multimeter in series with the VDDIN input.

During the design phase, an open circuit connection was created by inserting a jumper (J-Vin) to break the supply line to facilitate these measurements. Knowing that the operating voltage of the OBC is 3.3V and having measured the current drawn by the system in different modes, it was possible to draw the overall power consumption figures for each mode and the results are presented in Table 6.1.

6.4.2. Peripherals

All the peripherals listed in section 5.2 of Chapter 5 had to be tested individually. Test codes were written, compiled, downloaded and run on the OBC board for each peripheral. These tests and their results are presented in the following section.

6.4.2.1. Digital I/O, timer and UART configuration

The initial test involves a simple digital control to verify that the board does communicate with the PC via its JTAG port and the SAM-ICE. Two user LEDs on the OBC allow to visualise a change of state as one LED which is configured as an output, was programmed to toggle states at a constant rate.

The UART communication is also enabled and is capable of sending and receiving commands to and from the PC every time a change of state occurs. The rate at which the LED toggles is set by configuring one of the three timer counters (`TC0`) in up-counting mode in order to generate an interrupt every time its value overflows. A flowchart of this program is shown in Figure 6.3.

Table 6.1: OBC system power consumption in different operating modes

Operating Mode	Current Drawn	Power Consumption
Active mode	286.36mA	944.98mW
Backup mode	35.97mA (on VDDBU)	118.7mW
Sleep mode	(35.42mA at 8MHz) Frequency dependant	116.89mW
Wait mode	30.25mA	99.82mW

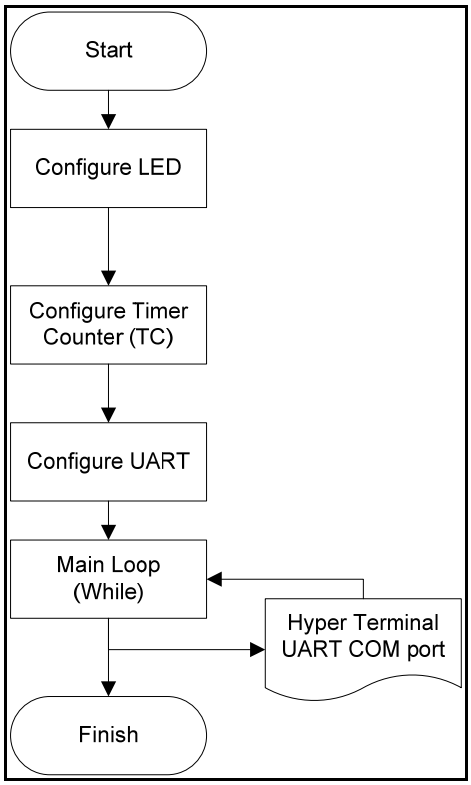


Figure 6.3: Blink and UART program flowchart

In order to set the LED as an output pin, a library function is used. This function belongs to the led library which is included in the main program and whose functions can be found in the utility sub-directory. The LED is configured by calling the following function:

```
LED_Configure(0);
```

This function assigns a pin instance to an LED constant in the first position (position '0') in a list of LED pin instances located in the header file *board.h*.

The configuration of timer `TC0` is done in four phases occurring sequentially within a locally defined function called `ConfigureTc`:

- The peripheral clock is enabled for `TC_0` by using the following statement:
  ```
  AT91C_BASE_PMC->PMC_PCER = 1 << AT91C_ID_TC0;
  ```

  The power management controller (`PMC`) base address points to the peripheral clock enable register `PCER` and assigns the timer counter 0 (`TC0`) to it.

- `TC0` is then enabled by shifting its register values one bit to the left. `TC0` can now be configured for a chosen frequency and its value can be compared with Register C of the timer counter channel interface to release a trigger signal to indicate that the desired frequency has been reached. The three following statements are used:

  ```
  TC_FindMckDivisor(4, BOARD_MCK, &div, &tcclks);
  TC_Configure(AT91C_BASE_TC0, tcclks | AT91C_TC_CPCTRG);
  AT91C_BASE_TC0->TC_RC = (BOARD_MCK / div) / 1;
  ```

  The function `TC_FindMckDivisor` takes four arguments: the desired frequency (4Hz in this case), the master clock frequency (defined in *board.h*), the divisor value and the timer counter clock select (TCCLKS)). This function automatically computes the best value for a divisor (`div`) for the clock in order to attain the desired frequency. The chosen divisor is guaranteed to satisfy the equation:

$$freq = \frac{\text{MCK}}{(\text{div} \times 65536)} \quad (4)$$

where `div` represents the highest possible value. The TCCLKS is necessary to determine the appropriate source for the counter and tell whether it is clocked by an internal or an external clock source.

The function `TC_Configure` configures the timer to operate in RC compare mode. This mode allows the timer to generate a trigger which resets the counter whenever its count reaches the pre-loaded RC value. The function `TC_Configure` takes two arguments: a pointer to a timer instance (`AT91C_BASE_TC0` in this case) and the operating mode (`tcclks | AT91C_TC_CPCTRG`).

In the third statement of the timer configuration phase, the base address of `TC0` points to register C (`RC`)'s API definition. It is then assigned a value equivalent to the desired frequency by using the previously computed value of the variable `div`.

- The interrupt on `TC0` needs to be enabled and configured on occurrence of the RC compare trigger since it is disabled and cleared at power-up. This is done by calling the following functions:

```
IRQ_ConfigureIT(AT91C_ID_TC0, 0, TC0_IrqHandler);
AT91C_BASE_TC0->TC_IER = AT91C_TC_CPCS;
IRQ_EnableIT(AT91C_ID_TC0);
```

The function `IRQ_ConfigureIT` is located in the *nvic.h* header file which contains all the methods and definitions for configuring interrupts by using the NVIC. It configures an interrupt by taking three arguments into consideration: the interrupt source to configure (`AT91C_ID_TC0`), the pre-emption priority (it is set to '`0`' in this case in order to disable the interrupt vector) and the interrupt handler function (`TC0_IrqHandler`). Next, the timer counter interrupt enable register (`TC_IER`) is pointed to `TC0`'s base address and is assigned to the RC compare bit (`AT91C_TC_CPCS`) of the TC Channel status register. The two previous statements were mandatory in order to enable the incoming interrupt from the unique source of `TC0` (`AT91C_ID_TC0`).

- Finally, `TC0` can be started, on the condition that the LED is enabled, with the following `if` statement:

```
if (pLedStates[0])
{
    TC_Start(AT91C_BASE_TC0);
}
```

  The function `TC_Start` enables `TC0` and performs a software reset to start the count in the condition that the `if` statement is true. Every time this condition occurs, the interrupt is serviced by resetting the RC value in the counter.

The UART configuration is done by calling the following library function:

```
TRACE_CONFIGURE(DBGU_STANDARD, 115200, BOARD_MCK);
```

This function initialises the UART communication and takes three arguments which represent, the standard UART operating mode (asynchronous, 8-bit, no parity, 1 stop bit), the baud rate and the master clock frequency.

The UART is configured to display the frequency at which the LED toggles and to transmit a text: “Blink at x Hz” via the serial port. The Windows Hyperterminal application is used to visualise the output on the PC at the rate specified in the UART setup. A screen shot of the Hyperterminal window is presented in Figure 6.4 showing the continuous message being transmitted every time the LED toggles.

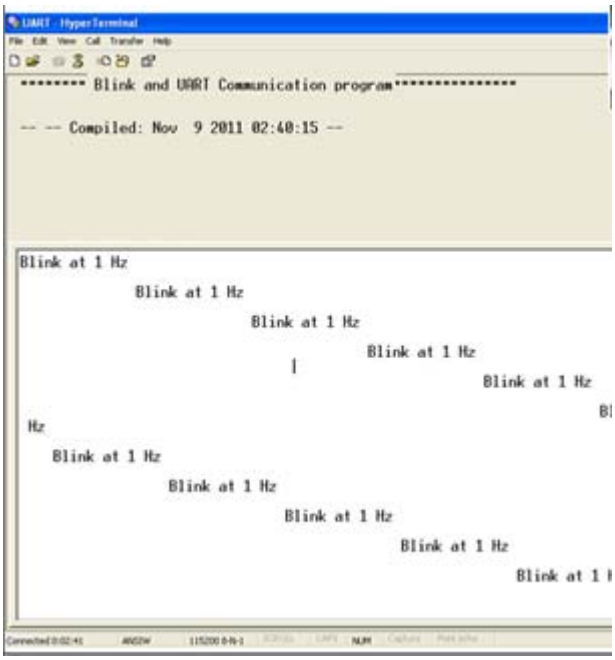


Figure 6.4: Blink and UART communication program output

6.4.2.2. 10-bit and 12-bit ADCs

The two sets of ADCs available on the AT91SAM3U4E are presented in section 5.2.8 of Chapter 5. In order to validate the functionality of this peripheral, one ADC channel is tested.

An externally connected potentiometer is used on one of the 12-bit ADC channels of the OBC (`ADC12_CHANNEL_3`) and the 3.3V reference is selected in order to vary the input. The UART communication is once again used to display a menu which allows varying the ADC gain and displaying the exact value of the voltage being sampled.

Figure 6.5 presents a flowchart of the program which is divided into three sections: the configuration of the ADC, the configuration of the interrupt with its handler and the analogue to digital sampling and conversion.

In the first step of the configuration process, the 12-bit ADC is initialised by calling the following function:

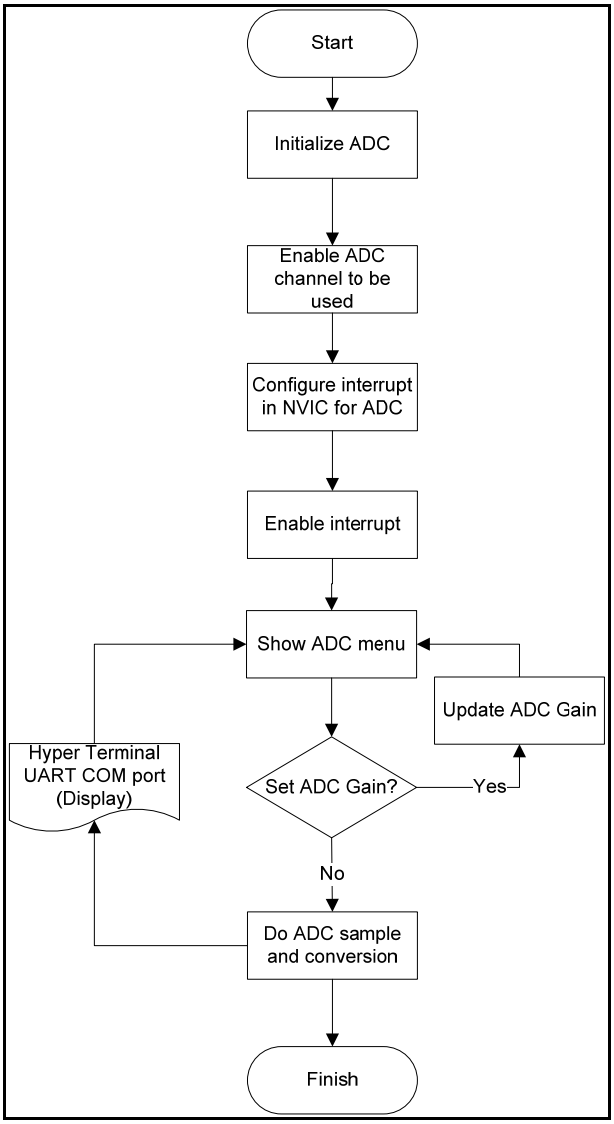


Figure 6.5: ADC program flowchart

```
ADC12_Initialize(AT91C_BASE_ADC, AT91C_ID_ADC, AT91C_ADC12B_MR_TRGEN_DIS,
                 0,AT91C_ADC12B_MR_SLEEP_NORMAL, AT91C_ADC12B_MR_LOWRES_12_BIT,
                 BOARD_MCK,BOARD_ADC_FREQ, 10, 1200);
```

This function takes ten arguments which represent the following parameters: a pointer to the 12-bit ADC instance, the software trigger mode, the sleep mode of operation, a 12 bits resolution, a master clock frequency of 48MHz, a selected ADC frequency, 10us startup time and 1.2µs sample and hold time. The ADC channel to be used is enabled in the second step of the configuration process by calling a function which refers to channel 3 of the 12-bit ADC:

```
ADC12_EnableChannel(AT91C_BASE_ADC, ADC12_CHANNEL_3);
```

A software interrupt is then configured and enabled for channel 3 of the 12-bit ADC. The interrupt handling function will set a flag variable which will indicate that the analogue to digital conversion has ended. As for the previous program, the configuration of the interrupt is done by calling the following function:

```
IRQ_ConfigureIT(AT91C_ID_ADC, 0, ADCC0_IrqHandler);
```

The interrupt source to configure is `AT91C_ID_ADC` and the pre-emption priority is set to '`0`'. The interrupt handler is the function `ADCC0_IrqHandler` which is defined as follows:

```
void ADCC0_IrqHandler (void)
{
  unsigned int status, i=1;
  status = ADC12_GetStatus(AT91C_BASE_ADC);

     if(ADC12_IsChannelInterruptStatusSet(status, channels[i]))

        {
            ADC12_DisableIt(AT91C_BASE_ADC, 1<<channels[i]);
            conversiondone |= 1<<channels[i];
        }
}
```

This function retrives the value of the status register of the ADC (`ADC12_GetStatus(AT91C_BASE_ADC)`), then tests if the interrupt status bit of channel 3's register is set (`ADC12_IsChannelInterruptStatusSet(status, channels[i])`). The status bit indicates the end of the conversion for this channel. The "end of conversion" flag can be cleared and disabled for the concerned channel (`ADC12_DisableIt(AT91C_BASE_ADC, 1<<channels[i])`).

The final section of this program displays a menu which allows one to set the gain of the ADC through the UART and explore the other characteristics of the ADC. Initially, the analogue control register value is extracted and assigned to a variable:

```
adc_acr = ADC12_GetAnalogCtrlReg(AT91C_BASE_ADC);
```

This register is defined in the API list of registers found in the library header file *AT91SAM3U4.h* and is found at address 0x400A8064. Table 6.2 presents all the variables necessary to set the register and indicates the value associated with these variables as well as their function. The input gain is represented by the three LSBs as indicated by the field value of 0x3. Other variables can also be changed within the register as indicated by their field values and functions on the table.

The value of the gain is read by performing a logical AND operation between the entire register and the three LSBs:

```
gain = adc_acr & 0x3;
```

The menu is displayed after configuring the UART. Values for the gain can be selected and a measurement can be performed by entering a key on the keyboard (G for gain selection and any other key to perform a measurement). A case statement associated to the gain value is used to select the desired gain (1, 2 or 4).

The conversion process is initiated on the ADC 12-bit by the following function:

```
ADC12_StartConversion(AT91C_BASE_ADC);
```

Table 6.2: API variable definitions, field and function

Analog Control Register variable (ADC12B_ACR)	Field Value	Function
AT91C_ADC12B_ACR_GAIN	0x3 << 0	Input gain
AT91C_ADC12B_ACR_IBCTL	0x3 << 6	Bias current control
AT91C_ADC12B_ACR_DIFF	0x1 << 16	Differential mode
AT91C_ADC12B_ACR_DIFF_SINGLE	0x0 << 16	Single ended mode
AT91C_ADC12B_ACR_DIFF_FULLY	0x1 << 16	Fully differential mode
AT91C_ADC12B_ACR_OFFSET	0x1 << 17	Input offset

The interrupt handler previously defined indicates the end of the conversion on the channel which is associated with it. In this case channel 3 and the program will constantly be polling to verify if the conversion is done before displaying the value which was sampled in mV. This is done in the following while loop:

```
 while (conversiondone != ((1<<ADC2)));
{
    id_channel = 1;
    advalue = ADC12_GetConvertedData(AT91C_BASE_ADC, CHANNEL_3);
    printf("\rChannel %u : %u mV", id_channel, convertHex2mv(advalue));
}
```

The sampled analogue data value is returned on the selected channel and is assigned to a local variable called `advalue` which represents the raw value on the ADC channel. The UART output is displayed using Hyper Terminal as shown in Figure 6.6.

6.4.2.3. TWI

The reliability of the TWI is vital in the CubeSat architecture since it has often been the selected serial interface for communication between the OBC and other subsystems. In section 5.2.5 of chapter 5, a TWI enabled temperature sensor (the MCP9800) is introduced and its features are presented. The sensor serves its purpose by measuring the temperature of the OBC and validating the functionality of the TWI communication.

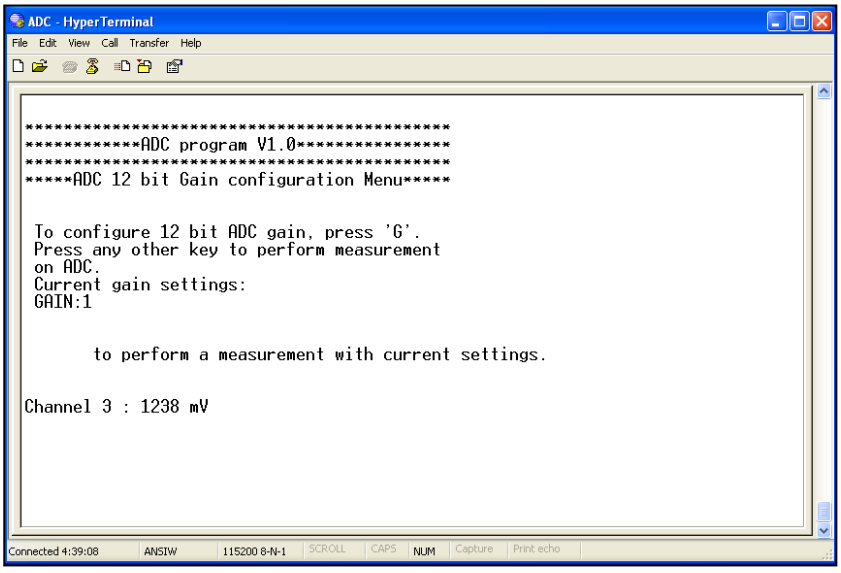


Figure 6.6: UART output of ADC program

The AT91SAM3U4E is configured as the master device on `TWI0` by calling the following function:

```
TWI_ConfigureMaster(AT91C_BASE_TWI0, TWCK, BOARD_MCK);
```

This function takes three arguments and configures the first peripheral (`TWI0`) by calling its API instance (`AT91C_BASE_TWI0`). The desired clock frequency is also defined as:

```
TWCK=1000000 Hz.
```

After configuring the microcontroller as the master, an initialisation of the TWI driver must take place. The following function accomplishes this task:

```
TWID_Initialize(&twid, AT91C_BASE_TWI0);
```

The drivers for TWI communication are called by this specific function which identifies the driver handler (`&twid`) as well as its location at base address `AT91C_BASE_TWI0`.

The temperature is displayed in a Hyper terminal window and data is sent to the PC via a UART serial communication link. The MCP9800's pointer bits are defined for each one of its registers in section 5.3.5 and each register's function is also given. Two of the four registers are being used in this particular program: the configuration register and the temperature register.

The configuration register is an 8-bit read/write register which provides user-access to the device's array of features. These features are set by the user but have defaualt settings at power up as shown in Table 6.3.

Table 6.3: MCP9800 configuration register power-up default settings

Feature	Bit number	Power-up default setting	Register value
One-Shot	7	Disabled	0
ADC Resolution	5-6	9-bit or 0.5°C	00
Fault Queue	3-4	1	00
Alert Polarity	2	Active Low	0
Comp/Int	1	Comparator mode	0
Shutdown	0	Disabled	0

In the program, the configuration register bits are arranged in a structure called `Confg_Reg` which contains declarations for all six features as structure members:

```
typedef struct
{
  unsigned char bOneShot;
  unsigned char bRes;
  unsigned char bFaultQueue;
  unsigned char bAlrtPol;
  unsigned char bMode;
  unsigned char bShutdown;
}Confg_Reg;
```

The first statement in the main function of the program performs a read operation on the configuration register:

```
TWID_Read(&twid, MCP9800_ADDRESS, CONF_REG, 0x01, Tdata, 0x01, 0);
```

The slave address of the MCP9800 is locally defined as `MCP9800_ADDRESS = 0x48` or in binary `0b1001000`. The function also indicates that it must perform a read operation on the configuration register (`CONF_REG`) and determines the address size of the register in bytes (`0x01`). The data that is being read must be stored in a buffer (`Tdata`) and the number of bytes to be read is also specified (`0x01`). The final argument in the function simply indicates an asynchronous data transfer. The data buffer `Tdata` is defined at the beginning of the main function as a one dimensional array of two elements. The stored buffer data from the configuration register is then used to fill in the register itself with selected values which in this case will be the default values and will only be using the first eight bits from the buffer:

```
FillConfigReg(Tdata[0],& Confg_Reg);
```

The temperature register on the other hand, is a 16-bit read only register which depending on the resolution, can contain 9 to 12 bits of temperature data. Bit assignments and corresponding resolution for this register are shown in Figure 6.7. Another read operation is performed but, this time on the temperature register (`TEMP_REG`):

Upper Half:							
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
Sign	2^6 °C	2^5 °C	2^4 °C	2^3 °C	2^2 °C	2^1 °C	2^0 °C
bit 15							bit 8

Lower Half:							
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
2^{-1} °C/bit	2^{-2} °C	2^{-3} °C	2^{-4} °C	0	0	0	0
bit 7							bit 0

Figure 6.7: MCP9800 temperature register bits (Microchip Technology Inc, 2010)

```
TWID_Read(&twid, MCP9800_ADDRESS, TEMP_REG, 0x01, Tdata, 0x02, 0);
```

This function's arguments are similar to the ones found in the read operation performed on the configuration register. The only major difference is that the buffer size (`Tdata`) is two bytes (16 bits) long as indicated by the size of the temperature register.

The final step is to process the gathered data. The temperature register data is formatted in two's complement and is converted into a decimal number depending on the selected resolution. An operation is performed on the buffer where only the upper byte is filtered out and assigned to a variable called `temp`. This variable is then converted into a decimal value (`integer`) and displayed with a decimal identifier corresponding to the selected resolution:

```
temp = ((Tdata[0]<<8)|(Tdata[1]));
temp = GetRealTemp(temp,confg.bRes);
index = temp%(2<<(confg.bRes-8-1));
index =  index <<(4-(confg.bRes-8));
integer =   temp/(2<<(confg.bRes-8-1));
printf("%d.%s C\r",integer,sixteenths[index]);
```

The program output is shown in Figure 6.8 where the ambient temperature is displayed in a Hyper terminal window via UART.

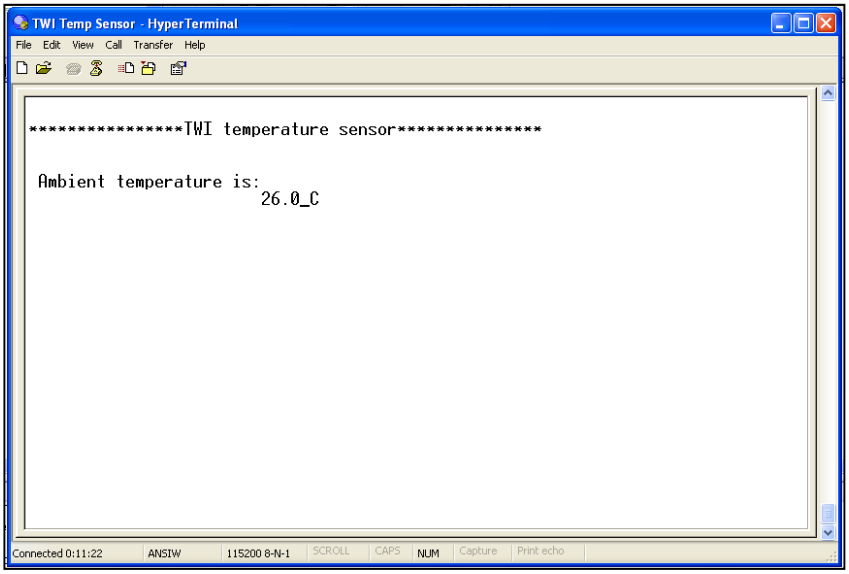


Figure 6.8: UART output of TWI temperature sensor program

6.4.2.4. USB

This section presents a discussion of the functional tests conducted on the OBC's USB port. The integrated UTMI device was presented in the previous chapter in section 5.2.4 and its properties and use on the OBC were covered. The test in this case involves the implementation of a UART to USB bridge using a communication device class (CDC).

The strategic position of the USB port on the OBC board allows access to the port after integration within the CubeSat structure. The purpose of the CDC application in this case would be to be able to emulate the UART interface by using the UTMI device.

The OBC appears as a serial device and a new COM port is made available once the CDC program is downloaded and the OBC is connected to the host PC with a USB cable as shown in Figure 6.9. A UART-to-USB driver is loaded into the microcontroller to replace the UART driver and initiate CDC operation which has the advantage of being unrestricted by hardware changes and independent of the running application on the PC. All the driver functions are located in a configuration file which is included in the CDC library subfolder by the following pre-processor directive:

```
#include <usb/device/cdc-serial/CDCDSerialDriver.h>
```

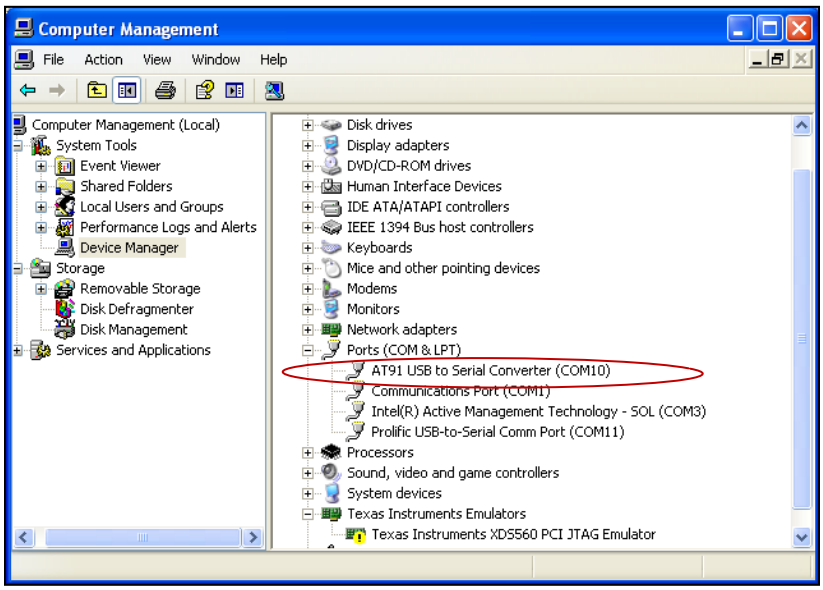


Figure 6.9: USB-to-UART COM port

Full-duplex communications can then occur between the PC and the OBC via the UTMI device as it would have been for a UART connection. The application running on the PC uses the same APIs as for UART communications but the CDC drivers act as a bridge between these APIs and the USB hardware. On the other hand, the loaded UART-to-USB driver interfaces the OBC application to the USB hardware and initiates the serial transmission from the running OBC application (Atmel Corporation, 2008).

Figure 6.10 illustrates the migration changes from USB to UART communication. A full duplex file transfer application program is downloaded on the microcontroller as part of the written program. This can be seen as another way to emulate telemetry and housekeeping data transfers from the OBC to the modems (or vice-versa) during integration.

The host PC reads or writes data from/to the OBC through the serial COM port (serial CDC) connected to the UTMI device and this data is either stored in a buffer or read from it. An example of the read function implementation by the OBC is shown in the following statement:

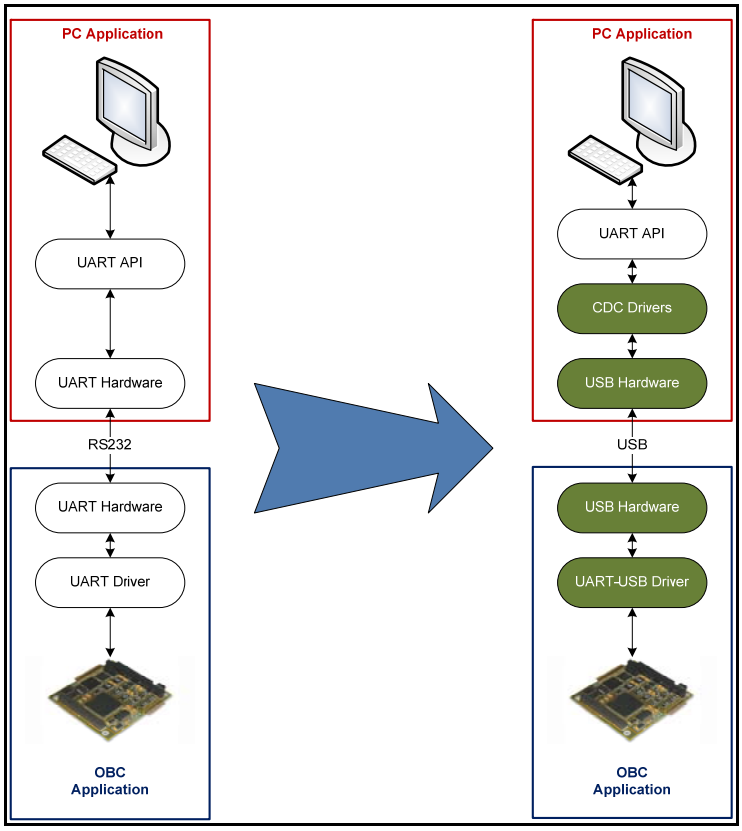


Figure 6.10: USB to UART migration (Atmel Corporation, 2008)

```
CDCDSerialDriver_Read(usbBuffer,DATABUFFERSIZE,(TransferCallback)
UsbDataReceived,0);
```

The argument `usbBuffer` represents the data buffer to store all the incoming received data while `DATABUFFERSIZE` specifies the size of the data buffer in bytes. These two parameters have previously been defined in the program and an indication of the transfer being completed triggers an optional call-back function which accommodates the interrupt handler for the end of data transfer condition (`(TransferCallback) UsbDataReceived`).

The received buffered data is in turn transmitted by using a serial transfer through a regular USART connection to the same host PC by using the following function:

```
USART_WriteBuffer(BASE_USART, usbBuffer, received);
```

The data content of `usbBuffer` is sent through the USART peripheral specified by its particular base address (`BASE_USART`) and the size of the receiving buffer (`received`) which are both pre-defined in the program. These two processes emulate a housekeeping data packet being transferred from a given subsystem to the OBC and transmitted to the modems for downlink. By opening two Hyper Terminal windows and having the PC connected to the OBC using their respective COM port connections (USART and USB-to-UART), a file transfer can be successfully initiated and accomplished. An initialisation of the file transfer is illustrated in Figure 6.11 while the file being transferred to the indicated destination path is shown in Figure 6.12.

6.4.2.5. PWM

The AT91SAM3U4E embeds an on-chip PWM controller with 4 channels; each uses one of the 16 bit resolution up/down counters and each channel is independently programmable. The user LED which is incorporated in the OBC design is used to visualise the output pulse provided by one of the PWM channels. The duty cycle value of the pulse is automatically updated by the peripheral direct memory access controller (PDC).

The interface used to configure the PWM controller peripheral is included as a header file in the peripherals subfolder of the IAR toolchain workspace window by the following statement:

```
#include <pwmc/pwmc2.h>
```

The frequency of the PWM is selected to be 1 kHz and the maximum duty cycle is set to 90% and stored in a buffer of a predefined length. All of the above definitions are implemented by the following statements:

```
#define PWM_FREQUENCY        1000
#define MAX_DUTY_CYCLE       90
#define MIN_DUTY_CYCLE       0
#define DUTY_BUFFER_LENGTH   (MAX_DUTY_CYCLE - MIN_DUTY_CYCLE + 1)
```

The PDC buffer is defined as an array of `DUTY_BUFFER_LENGTH` elements as follows:

```
unsigned short dutyBuffer[DUTY_BUFFER_LENGTH];
```

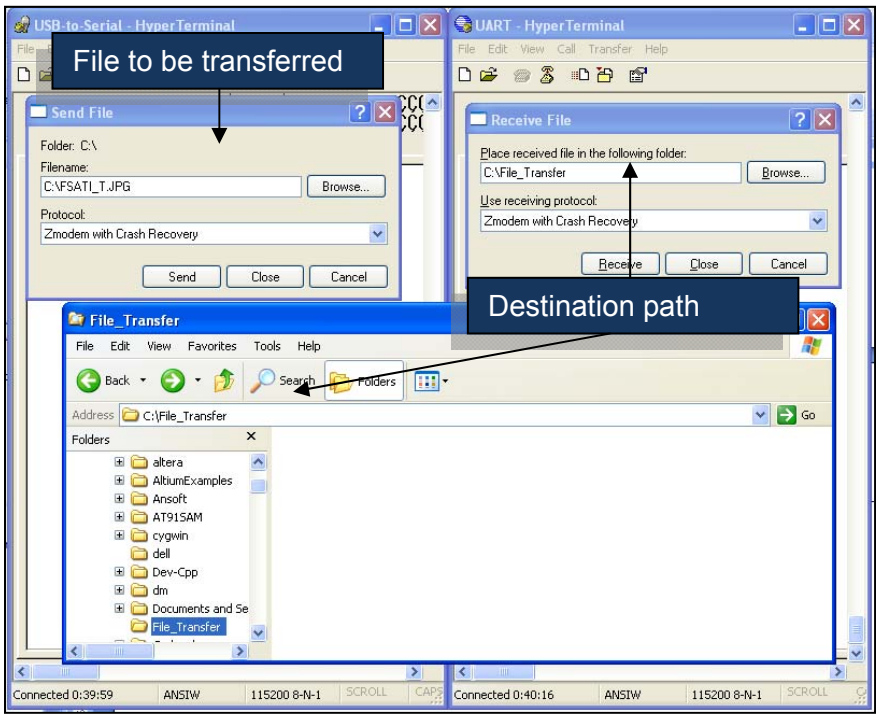


Figure 6.11: File transfer initialisation

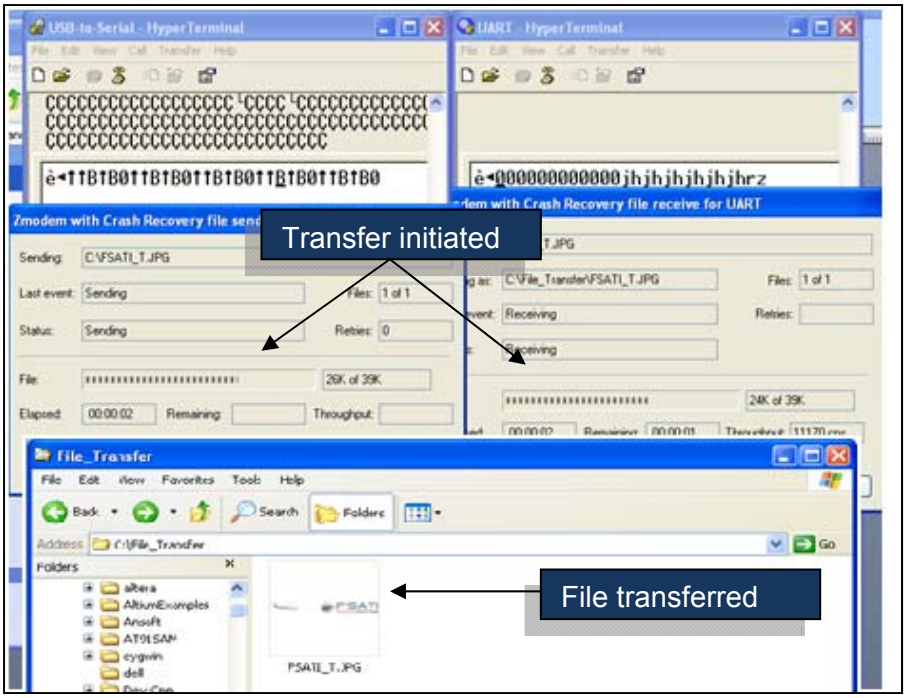


Figure 6.12: File transfer progress and completion

The main function of this test program contains a sequence of instructions which configure the PWM channel and PWM controller before initiating the PDC transfer and write from the buffer to the PWMC.

The first step is to enable the peripheral clock in the peripheral_clock_enable_register (PCER) and assigning it to the PWM controller ID:

```
AT91C_BASE_PMC->PMC_PCER = 1 << AT91C_ID_PWMC;
```

This operation is necessary since it gives the onboard clock access to the peripherals which in this case are the PWM controller, the PWM channel and the timers. The next step involves the configuration of the timer which will be set to run at a frequency equal to `PWM_FREQUENCY * MAX_DUTY_CYCLE` and the following function accomplishes this task by finding the right MCK divisor and prescaler values:

```
PWMC_ConfigureClocks(PWM_FREQUENCY * MAX_DUTY_CYCLE, 0, BOARD_MCK);
```

After configuring the timer, the PWM controller channel also needs to be configured and assigned to LED0. In the same line, the period, the duty cycle and the dead-time values of the PWM channel have to be set according to the chosen channel. Four functions can be identified which accomplish the abovementioned tasks:

```
PWMC_ConfigureChannelExt(CHANNEL_PWM_LED0, AT91C_PWMC_CPRE_MCKA, 0, 0, 0,
AT91C_PWMC_DTE, 0, 0);
PWMC_SetPeriod(CHANNEL_PWM_LED0, MAX_DUTY_CYCLE);
PWMC_SetDutyCycle(CHANNEL_PWM_LED0, MIN_DUTY_CYCLE);
PWMC_SetDeadTime(CHANNEL_PWM_LED0, 5, 5);
```

This program is essentially based on the execution of an interrupt in the NVIC with a source address at the PWM controller. The interrupt is enabled and serviced at the end of data transfer (duty cycle increment) from the duty cycle buffer to the PDC and this action is repeated according to the pre-defined PWM frequency. The interrupt is first configured with a priority '`0`' and then enabled by the following statements:

```
IRQ_ConfigureIT(AT91C_ID_PWMC, 0, PWM_IrqHandler);
IRQ_EnableIT(AT91C_ID_PWMC);
```

With the interrupt source selected as `AT91C_ID_PWMC`, the next step is to enable it in a peripheral and initiate it under the “end of buffer transfer” condition. This is accomplished by the following function:

```
PWMC_EnableIt(0, AT91C_PWMC_ENDTX);
```

where `AT91C_PWMC_ENDTX` represents the end of the PDC buffer transfer in the interrupt enable register of the PWM controller interface.

Before beginning the PDC transfer of the buffer contents, it is essential to fill the duty cycle buffer for the concerned channel (Channel 0) by first enabling it in the PWM controller interface:

```
PWMC_EnableChannel(CHANNEL_PWM_LED0);
```

The duty cycle buffer is filled with values going from `MIN_DUTY_CYCLE` to `MAX_DUTY_CYCLE` by implementing the following `for` loop:

```
for (i = 0; i < DUTY_BUFFER_LENGTH; i++)

            {
                  dutyBuffer[i] = (i + MIN_DUTY_CYCLE);
            }
```

Finally, the transfer can be initiated by sending the contents of the buffer through the PWM controller peripheral, using the PDC to take care of the transfer. This is accomplished by the following function:

```
PWMC_WriteBuffer(AT91C_BASE_PWMC, dutyBuffer, DUTY_BUFFER_LENGTH);
```

The function takes three arguments: the pointer to the source address of the PWM controller instance, the data buffer whose contents will be sent and the length of the data buffer. This function is initiated by the abovementioned statement and perpetuated on in a loop by the interrupt handler function `PWM_IrqHandler` which was previously defined. As a result, LED0 on the OBC will intermittently glow with a duty cycle varying from 0% to 90% at a frequency of 1kHz.

6.5. Summary

This chapter covered all the different tests and results conducted on the OBC prototype. The overcurrent protection circuit was simulated and successfully tested. Power consumption figures were deduced for all different operating modes by measuring the current in series at the VDDIN jumper which was included in the design for this purpose. Additionally, peripherals associated with the OBC were tested for functionality by writing, compiling and downloading relevant test programs into the microcontroller.

CHAPTER SEVEN

CONCLUSIONS AND RECOMMENDATIONS

7.1. General conclusions

In the first chapter, an introduction to the CubeSat concept and architecture was given. The objectives and significance of the study were defined and overviews of the methodology and the structure of the thesis were also presented.

A presentation of the CubeSat standard was given and each one of the CubeSat subsystems was defined in terms of their functions. Being the main focus of this research, more emphasis was put on the OBC subsystem and a detailed analysis was conducted in terms of hardware and software, listing all essential components.

The space environment and its effects on electronic equipment were discussed in order to set environmental constraints for selecting components for the design and implementation.

Following a thorough literature research which considered previously deployed CubeSats (Appendix A), it was necessary to select a suitable microcontroller in the 32-bit core architecture range to carry out all data transfers between the OBC peripherals and all other CubeSat subsystems. Selection criteria were identified and preference was given to the AT91SAM3U4E which is Atmel's implementation of ARM's Cortex M3 core.

A specific OBC architecture was then designed around the selected microcontroller and all relevant features of the microcontroller in relation to the CubeSat architecture were pointed out.

The OBC specifications were presented and upon the completion of the architectural design, the hardware implementation of the proposed OBC architecture was developed. A circuit was implemented in series with the supply rail by means of a MAX4374 high-side current sense IC in order to protect the OBC from short circuits during operations by shutting down the supply line.

Each peripheral was presented and implemented. In the case of the TWI, the MCP9800 from Microchip, which is a TWI driven temperature sensor, was used to prove the functionality of the TWI bus. Similarly, the ADC, USB, PWM, SD/MMC card, SPI and UART interfaces were presented. Consequently, a set of schematic diagrams for each section and peripheral of the OBC was developed and combined in the final design. This phase was necessary before completing the PCB implementation which was done using Altium designer CAD software.

A series of hardware and software tests were conducted in the final step after populating the PCB. Power consumption figures were derived for all different operating modes and the trip function of the overcurrent protection circuitry which was implemented in series with the supply rail was verified in short circuit and overcurrent conditions. In addition to the power supply management, a series of source codes were written, compiled and downloaded to the microcontroller in order to individually test some of the peripherals for functionality.

In the end, the research conducted which lead to the final development of the OBC prototype responded positively to the addressed research questions from the introductory chapter. This can be summarised as follows:

- The choice of a 32-bit microcontroller, more precisely Atmel's AT91SAM3U4E, was justified over 16-bit and 8-bit microcontrollers following a literature study and an analysis of selection criteria which were explained in detail in Chapter 3.
- The OBC subsystem was analysed and important peripherals were listed. Further, each one of these peripherals was implemented in the OBC architecture as well as in the design which were respectively presented in Chapter 4 and Chapter 5.
- The developed OBC prototype possesses multiple peripherals which in the future can be customised according to a given mission requirements. This represents an exclusive advantage when compared to previously developed OBCs which all had mission-specific designs. Future subsystems can be developed for upcoming F'SATI missions around an OBC for which peripherals and power consumption figures would have already been successfully tested.

7.2. Issues encountered

7.2.1. Hardware

A number of challenges were encountered during the implementation of the OBC architecture. The final PCB was gradually populated, starting with the voltage regulation section and the overcurrent protection circuit, followed by all passive components, connectors, other active devices and the microcontroller.

Soldering of the microcontroller, which is a 144 pin fine pitch component, was particularly challenging. The overcurrent protection circuit proved its relevance more than once by detecting short-circuits between a power pin and a GPIO pin of the microcontroller and shutting down the voltage supply before causing any damage to the microcontroller.

A permanent communication error was detected on the JTAG port when attempting to program the microcontroller.The SAM-ICE indicated a connection to a non-powered device. What seemed to be a software problem was in fact hardware related and research about the topic established that when shipped from factory, the AT91SAM3U4E among other microcontrollers is always in deep-sleep operating mode by default. This presents the lowest power figures since only the internal clock of the core is running and every other peripheral clock as well as the core itself are not being powered. The way around this was to perform a Forced-Wake-Up (FWUP) to the microcontroller by pulling the associated FWUP pin to ground for more than 200ms. This was necessary to awaken the core and the internal peripheral clocks, hence making communication possible.

7.2.2. Software

A fair amount of time was invested in becoming familiar with the IAR EWARM toolchain. A particular challenge was to understand the mechanism behind the NVIC as well as the way libraries are organised.

The documentation provided the available sample programs and the reliable customer support and the diversity of online forums allowed a good understanding of the IDE software package and implementation of experimental programs.

7.3. Future work

The prototype OBC that was developed had proven to function correctly after running a series of tests on all necessary and accessible peripherals. An SD/MMC card was selected as the memory device but the card holder was perceived to be potentially unreliable in vibration environments (i.e. during launches or P-POD ejection). An investigation can be conducted in order to select an appropriate SD/MMC card holder which would be reliable enough and not cause any disconnection when exposed to mechanical vibrations.

In order to save board space, the mini SD/MMC card may be replaced with a micro SD/MMC card which is a quarter of the size. Similarly the CR2031 lithium cell can be replaced with a smaller and lighter CR1225 cell which has similar capacity.

Finally, a file system for data transfer can be implemented with a specific data format for logged housekeeping parameters as well as telemetry and can be integrated in the OBC's flight software.

BIBLIOGRAPHY

Albus, Z., Valenzuela, A. & Buccini, M. *Ultra-Low Ppower Comparison :MSP430 vs. Microchip XLP Tech Brief-A Case for Ultra-Low Power Microcontroller Performance.* Available at: http://www.ti.com/lit/wp/slay015/slay015.pdf

ARM. 2010. *Cortex-M3 Devices Generic User Guide.* Available at: http://infocenter.arm.com/help/topic/com.arm.doc.dui0552a/DUI0552A_cortex_m3_dgug.pdf [6 August 2010]

Atmel Corporation. 2009. *AT91ARM M3 Cortex-based processor.* Available at: http://www.atmel.com/images/doc11020.pdf [6 August 2010]

Atmel Corporation. 2002. *AVR130: Setup and Use the AVR Timers.* Available at: http://www.atmel.com/images/doc2505.pdf [6 August 2010]

Atmel Corporation. 2008. *AVR272: USB CDC Demonstration UART to USB Bridge.* Available at: http://www.atmel.com/dyn/resources/prod_documents/doc7619.pdf [6 August 2010]

Atmel Corporation. 2006. *SAM Boot Assistant (SAM-BA) User Guide.* Available at: http://symbrion.eu/tiki-download_file.php?field=19 [6 August 2010]

Atmel Corporation. 2010. *SAM3U-EK Development board User Guide.* Available at: http://www.atmel.com/Images/doc6478.pdf [6 August 2010]

Barr, M. 1999. *Programming Embedded Systems in C and C++*. O'Reilly.

Botma, P. J. 2011. *The Design and Development of an ADCS OBC for a CubeSat,* Master's thesis, University of Stellenbosch, Stellenbosch.

Brand, C. J. 2007. *The Development of an ARM-based OBC for a Nanosatellite.* Master's thesis, University of Stellenbosch, Stellenbosch.

Catsoulis, J. 2005. *Designing Embedded Hardware.* Sebastopol: O'Reilly.

Chenakin, A. 2010. *Frequency Synthesizers: Concept to Product.* Norwood: Artech House Inc.

Clark, C. S. & Simon, E. 2007. Evaluation of Lithium Polymer Technology for Small Satellite Applications, Proceedings of *The 21st Annual AIAA/USU Conference on small satellites.* 2007 Utah, USA (SSC07-X-9).

ClydeSpace. 2011. *1U CubeSat EPS*. Available at: http://www.clyde-space.com/cubesat_shop/eps/8_1u-cubesat-eps [7 January 2011]

Dasgupta, R. 2008. *Low Power MCU Selection Criteria and Sleep Mode Implementation Using Hardware/Software CoDesign Technique.* Available at: http://www.eetimes.com/electrical-engineers/education-training/tech-papers/secure/tata-consultancy-services/4137624?isSurveySuccess=True [23 June 2011]

Davidsen, P. 2008. *Satellite Systems and Design: Architecture of On-Board Systems.* Available at: http://www.space.aau.dk/cubesat/documents/DTU_kursus_Architecture_of_onboard_systems.ppt [21 June 2012]

DefenceWeb. 2012. *South Africa's CubeSat promoting space ambitions*. Available at: http://www.defenceweb.co.za/index.php?option=com_content&view=article&id=23165&catid=74&Itemid=30 [19 February 2011]

Delft University of Technology. n.d. *Delfi-C3 2 year anniversary.* Available at: http://www.delfic3.nl/index.php?option=com_content&task=view&id=105&Itemid=71 [4 July 2011]

Denier, W. 2010. *Satellite Payload Systems: Satellite Operations and 3D modeling* Available at: http://waynedenier.com/?p=144 [22 June 2011]

South Africa. Department of Science and Technology. 2010. *A Journey for Tomorrow, Watch This Space.* DST,3 (2), February.

Eady, F. 2004. *Networking and Internetworking with Microcontrollers.* Oxford: Elsevier.

Eide, E. & Iilstad, J. 2003. *Ncube-1, The First norwegian CubeSat Student Satellite*. Proceedings of The 16th ESA Symposium on European Rocket and Balloon Programmes and Related Research,pp. 85 - 88. St Gallen, Switzerland.

EmbeddedDeveloper. n.d. *Microcontroller selection.* Available at: http://www.embeddeddeveloper.com [5 March 2011]

Enright, J. P. 2002. *A Flight Software Development and Simulation Framework for Advanced Space Systems.* PhD thesis, Massachusetts Institute of Technology (MIT), Cambridge.

Ganssle, J. 1999. *ICE Technology Unplugged: EE Times-India* , pp. 109-113, November. Available at: http://www.eetindia.co.in/STATIC/PDF/199910/EEIOL_19990CT02_EMS_TEST_TA.pdf?SOURCES=DOWNLOAD [25 March 2011]

Gildeh, D. 2003. *Final Report: Design and development of OBC for Pico-Sat PALMSAT*. University of Surrey, Guildford. http://www.davidgildeh.com/sites/davidgildeh.com/files/projects/Palmsat_Report.pdf [5 June 2010]

Gunter, P. 2008. *Space Junk.* Frankfurter Allgemeine Zeitung Available at: http://www.faz.net/aktuell/wissen/weltraum/weltraummuell-schrott-so-weit-das-auge-reicht-1540458.html [12 February 2011]

Hales, J.H. & Pedersen, M. 2002. Two-Axis MOEMS Sun Sensor for Pico Satellites. Proceedings of *The 16th Annual AIAA/USU Conference on small satellites.* 2002 Utah, USA (SSC02-VI-6).

Han, J.H. & Kim, C.G. 2006. *Low earth orbit space environment simulation and its effects on graphite/epoxy composites*. Composie Structures. 72(7): pp. 218-226. Elsevier Ltd.

Hardy, J. 2009. *Implementation du Protocol AX.25 a Bord du Nanosatellite OUFTI-1.* Master's thesis, University of Liege, Liege.

Heidt, H., Suari, J. P., Moore, A. S., Nakasuka, S. & Twiggs, R. J. 2000. CubeSat: A new Generation of Picosatellite for Industry Low-Cost Space Experimentation. *Proceedings of the 14th Annual Small Satellite Conference*, Salt Lake City, Utah, August, 2000.

Hidayat, N. D. 2010. *Development of Flight Software for The Nanosatellite Onboard Computer Based on FM430 and Real-Time Operating System.* Master of Technology thesis, Cape Peninsula University of Technology, Cape Town.

Holman, G.& Benedict, S. 2007. *Solar Flare Theory Educational Web Pages*.Nasa's Goddard Space Flight Center
Available at: http://hesperia.gsfc.nasa.gov/sftheory/frame1.htm [6 June 2011]

IAR Systems. 2007. *IAR Embedded Workbench IDE User Guide for Advanced RISC Machines Ltd's ARM Cores.*
Available at: http://www.efo.ru/doc/Atmel/pdf/EWARM_UserGuide.pdf [17 June 2011]

ISIS. 2008. *Small Satellite Ground Stations.*
Available at:
http://www.isispace.nl/media/products/GSKIT/Brochure_ISIS_GroundStation.pdf
[12 March 2011]

Khan, M. (2008). *Computer Revolution.* BP Network.
Available at: http://bpnetwork.ca/whitepaper_files/Computer-Revolution.pdf [8 April 2011]

Kingmax Digital Inc. (2000). *SD Card Specification.*
Available at: http://www.kingmax.com/m,k [29 August 2011]

Koskienen, H., Eliasson, L., Holback, B., Anderson, L., Eriksson, A., Malkki, A., Norberg, O., Oulkkinen, T., Viljanen, A., Wahlund, J. E. & Wu, J. G. 1999.*Final Report: Space Weather and Interactions with Spacecraft.* SPEE.
Available at: http://space.fmi.fi/spee/docs/spee-final1.pdf

Lee, S., Hutputanasin, A., Torrian, A., Lan, W. & Munakata, R. 2009. *CubeSat Design Specification: Technical report. California Polytechnic State University, California.*
Available at: http://www.cubesat.org/images/developers/cds_rev12.pdf [12 May 2010]

Louis, J. & Ippolito, J. 2008. *Satellite Communications Systems engineering: Atmospheric Effects, Satellite Link Design and System Performance.* Washington DC: Wiley.

Maxim Integrated Products. 2007. *SPI/ I^2C Bus Lines Control Multiple Peripherals. Application note 4024.*
Available at: http://pdfserv.maxim-ic.com/en/an/AN4024.pdf

Maxim Integrated Products. 2001. *High-Side Current-Sense Measurement: Circuits and Principles.* Application note 746.

Available at:http://pdfserv.maxim-ic.com/en/an/AN746.pdf

Maxim Integrated Products. 2000. *Low-Cost, Micropower, High-Side Current-Sense Amplifier +Comparator +Reference ICs: MAX4373/MAX4374/MAX4375.*
Available at: http://datasheets.maxim-ic.com/en/ds/MAX4373-MAX4375.pdf

Mehrholz, D., Leushacke, L., Flury, W., Jehn, R., Klinkrad, H. & Landgraf, M. 2002. Detecting, Tracking and Imaging Space Debris. *ESA bulletin 109*, 128-134, February 2002.
Available at: http://www.esa.int/esapub/bulletin/bullet109/chapter16_bul109.pdf

Microchip Technology Inc. 2010. *MCP98000/1/2/3 2-Wire High-Accuracy Temperature Sensor*
Available at: http://ww1.microchip.com/downloads/en/DeviceDoc/21909d.pdf

Nagi, C. 2003. *Embedded Systems Design using the TI MSP430 Series* : Newness.
NASA. 2010. Human Exploration and Operations. *The CubeSat Launch Initiative*.
Available at: http://www.nasa.gov/directorates/heo/home/CubeSats_initiative.html

National Semiconductor Corporation. 2006. *LM1117/LM1117I 800mA Low-Dropout Linear Regulator.*
Available at: http://www.ti.com/lit/ds/symlink/lm1117-n.pdf

National Semiconductor Corporation. 2000. *LM4040 Precision Micropower Shunt Voltage Reference.*
Available at: http://www.ti.com/lit/ds/symlink/lm4040-n.pdf [3 April 2010]

Navalgund, R. R., Jayaraman, V. & Roy, P. S. 2007. *Remote sensing applications: An overview*. Current Science, *93* (12), pp. 1747-1766, November.

Noergaard, T. 2005. *Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers.* Burlington: Newnes.

Oland, E., & Schlanbusch, R. 2009. *Reaction wheel design for CubeSats*. Proceedings of The 4th IEEE International Conference on Recent Advances in Space Technology, pp. 778-83, Istanbul, Turkey.

PC/104 Embedded Consortium. 2003. *PC/104 standard and specifications.*
Available at: http://versalogic.com/support/pdf/PC/104Specv246.pdf [11 February 2011]

Peterson, W. W. 1961. Cyclic *Codes for Error Detection.* Proceedings of The IRE pp. 228-235. Gainesville, Florida..

Philips Semiconductors. 2003. *I²C Manual. Application note AN10216-01*.
Available at: http://www.philips.com/appnotes/ I^2C _Manual [3 April 2010]

Pierlot, G. 2009. Oufti-1: *Flight System Configuration and Structural Analysis.* Master's thesis, University of Liege, Liege.

Polaschegg, M. 2005. *Study of a CubeSat Mission.* Master's thesis, University of Graz, Graz, Australia.

Povey, H. 2008. *32-Bit Microcontrollers in an 8-Bit World*. Information Quartely , 28-31.
Available at: http://www.asdjhakjfajk.com [12 November 2010]

Pumpkin Inc. 2003. *An Overview of the CubeSat Kit*
Available at: http://www.cubesatkit.com/content/overview.html [7 January 2011]

Pumpkin Inc. 2008. *CubeSat Kit FM430 Flight Module.*
Available at: http://www.cubesatkit.com/docs/datasheet/DS_CSK_FM430_710-00252-C.pdf [7 January 2011]

Pumpkin Inc. 2010. *History & Performanceof Pumkin's Products In Space*.
Available at: http://www.cubesatkit.com/content/space.html [1 October 2011]

Rupprecht, M. 2011. *NCube-2 CubeSat,*DK3WN
Available at: http://www.dk3wn.info/sat/afu/sat_ncube.shtml [12 March 2011]

Sakoda, D., Horning, J. A. & Moseley, S. D. 2005. *Naval Postgraduate School NPSAT1 Small Satellite.* Monterey.
Available at: http://sp.nps.edu/npsat1/technical/371468_NPS.pdf [21 March 2011]

SANDISK Corporation. 2003. *Secure Digital Card Product Manual Rev 1.7.*
Availabel at: http://www.sandiskcorp.com/cardspec [5 September 2011]

Schmitz, T., & Wong, M. 2007. *Choosing and Using bypass Capacitors.* Application note 1325.0. Intersil.
Available at: http://www.intersil.com/content/dam/Intersil/documents/an13/an1325.pdf [24 April 2011]

Senechal, T. 2007. *Orbital Debris: Drafting, Negociating, Implementing a Convention.* Massachusetts.

Singh, KB. 2004. *Microcontroller and Embedded System*. New Delhi: New Age International.

Steinmetz, R., & Wehrle, K. 2005. *Peer-to-Peer Systems and Applications.* Berlin: Springer.

Stras, L., Kekez, D. D., Wells, G. J., Jeans, T., Zee, R. E., Pranajaya, F.& Foisy D. 2003. Final Report:*The Design and Operation of The Canadian Advanced Nanospace eXperiment (CanX-1).*
Available at: http://www.canx1.ca/ [15 December 2010]

Universal Serial Bus. 2000. *Specification Revision 2.0.*
Available at: http://www.usb.org/2.0 [14 May 2011]

Weiss, A. R. (2002). *Dhrystone Benchmark.* Austin.

Wells, G. J., Stras, L.. & Jeans, T. 2003. *Canada's Smallest Satellite; The Canadian Advanced Nanospace eXperiment (Can X-1).* University of Toronto Institute for Aerospace Studies, pp. 4-7. Toronto, Canada.

Wertz, J. R. & Larson, W. J. 2008. *Space Mission Analysis And Design.* New York: Microcosm and Springer.

Whilmshurst, T. 2007. *Designing Embedded Systems with PIC Microcontrollers "Principles and Applications".* Oxford: Elsevier Ltd.

Appendix A: Previous CubeSat missions

Table A.1. Previous CubeSat missions OBCs features

CubeSat	University or company	Mission overview	Status	TT&C OBC microcontroller
CUTE-1	Tokyo Institute of Technology	Test platform based on COTS components. Deployable solar cells, piezoelectric vibrating gyroscope (4 pcs), dual axis accelerometers (4 pcs) and CMOS camera used as sun sensor. The camera pictures could not be transmitted to the ground.	The satellite is still operating nominally in early December 2008 (5 years, 5 months after launch)	H8/300(Renesas Technology) 8-bit operation 32kB ROM 1kB RAM Operates at 30MHz
XI-IV	University of Tokyo	Test platform based on COTS components. Included a camera to take pictures of the earth.	Latest telemetry analysis dated September 20th, 2007. The satellite beacon was last heard in early December 2008 (5 years, 5 months after launch).	PIC 16F877 (Microchip) 8-bit microcontroller 32kB ROM 1kB RAM Operating at 40MHz
CanX-1	University of Toronto	Space-testing key technologies for future missions: Low-cost CMOS horizon sensor and star-tracker, GPS receiver.	Radio contact never established	AT91SAM (ARM7 fromAtmel) 32-bit microcontroller 512kB SRAM 32MB Flash-RAM Operating at 40MHz
DTUsat	Technical University of Denmark	MEMS sun sensors and a 600 m tether used to change the orbit. A color CCD camera and electron emitter were not ready on time for launch.	Radio contact never established	AT91M40800 (ARM7 from Atmel) 32-bit microcontroller 1MB RAM 16 KB ROM 2MB Flash-RAM Operating at 16MHz
AAUCubeSat	Aalborg University	Color CMOS camera	Antennas Short Circuited, resulting in poor comms performance. Batteries died after a month due to poor packaging	C161(Siemens) 4MB RAM 512kB PROM 256kB Flash-ROM Operating at 10MHz

QuakeSat	Stanford University and Quakesat	Detect ELF radio emission of seismic activity during earthquakes. Had deployable solar panels, and a magnetometer mounted on a 60 cm boom. The s/c was designed using COTS components.	Designed for 6 months, it worked flawlessly until at least June 6th, 2004 (more than 11 months).	ZFx86 (Siemens on a PrometheusPC/104 board) 32-Bit microcontroller 16MB RAM 1MB Flash-ROM Operates at 60MHz
NCube2	Norwegian Univesrity of Science and Technology	Similar to NCube1, the payload consists of an Automatic Identification System. AIS is a mandatory system on all larger ships, which transmits identification and position data messages. The satellite will redirect these messages along with messages from Norwegean reindeer collars.	Radio contact never established	ATmega32L (Atmel AVR) 32-bit microcontroller 32KB Flash 4kB ROM Operating at 16MHz
UWE-1	University of Wurzburg	Testing a communication protocol, test of GaAs cells in space, running micro Linux	Nominal operations until November 17th, 2005, when it was last heard. Since then contact has been lost completely.	Hitachi H8S-2674R (Hitachi) No Flash/ROM (All external) Operating at 16MHz
XI-V	University of Tokyo	Test of CIGS and GaAs solar cells, increased resolution of camera and an introduction of rapid shooting mode for estimating attitude motion. A morse message transmission service for radio amateurs was been added.	Nominal operations, first image received November 22nd, 2005. From December 2005, the images are showing some problems. The satellite was still operating 3 years, 2 months after launch.	PIC 16F877 (Microchip) 8-bit microcontroller 32kB ROM 1kB RAM Operating at 40MHz
Cute 1.7	Tokyo Institute of Technology	Test of charged particle detector (Avalance Photo Diode sensor module), made by Tokyo Institute of Technology. Due to low perigee (about 300 km at launch) expected lifetime is significantly less than one year. Experimental 10m tether and ectron emitter to change orbit	Nominal operations until early March 2006, it was since fully recovered, but since April 2006 no longer responds to commands from the ground.	ARMV4I (Onboard PDA) 16-bit microcontroller 32MB RAM 128MB Flash (SD Card) Operating at 400MHz

ION	University of Illinios	Features a Photomultiplier Tube (PMT) to observe airglow phenomenon in the earth's upper atmosphere (mesosphere). Also has a low-thrust, electric propulsion system and a CMOS camera for Earth imaging.	Destroyed due to launch failure.	SH2 (Hitachi Processor) 16-bit microcontroller 1MB External SDRAM 8MB Flash 256kB EEPROM Operating at 7MHz
Sacred	University of Arizona	This is the third CubeSat by the University of Arizona (the second to be launched). Produced by Montpelier University and Alcatel, the payload will measure the total amount of high-energy radiation over a two-year span and will test the radiation properties of four commercial integrated circuits.	Destroyed due to launch failure.	PIC16C77 (Microchip) 8-bit microcontroller 168kB RAM 64kB FRAM Operating at 20MHz
KUTEsat Pathfinder	University of Kansas	Measure the radiation in LEO and take photographs with an onboard camera	Destroyed due to launch failure.	PIC18F4220 (Microchip) 16-bit microcontroller 32kB Flash 4B SRAM Operates at 40MHz
SEEDS	Nihon University	Contains a gyro sensor for accurate determination of attitude motion	Destroyed due to launch failure.	NO MISSION INFORMATION AVAILABLE
HAUSAT1	Hankuk Aviation University	GPS receiver, experiment on deployment mechanism of solar panel and space verification of home made sun sensor	Destroyed due to launch failure.	AT91LS8535 (Atmel) 32-bit microcontroller 512 B internal SRAM 512 Bytes EEPROM Operating at 60MHz
NCube1	Norwegian University of Science and Technology	Similar to NCube2, the payload consists of an Automatic Identification System. AIS is a mandatory system on all larger ships, which transmits identification and position data messages. The satellite will collect these messages along with messages from Norwegian reindeer collars.	Destroyed due to launch failure.	AVR ATmega640 (Atmel) 8-bit microcontroller 64KB Flash 8B SRAM Operates at 8MHz

ICE Cube 1	Cornell University	Perform GPS scintillation science by measuring fluctuations in the signals that the GPS satellites emit when the signals pass through the ionosphere. Identical to ICE Cube 2	Destroyed due to launch failure.	NO MISSION INFORMATION AVAILABLE
RINCON 1	University of Arizona	This is the second CubeSat by University of Arizona (the first, which was just a s/c bus, is not scheduled for launch) The payload is a low-power beacon system, which provides a redundant means of relaying sensor data in analogue form if the primary (digital) transmitter fails.	Destroyed due to launch failure.	PIC16C77 (Microchip) 16-bit microcontroller 64KB FRAM 368Bytes of RAM Operates at 4MHz
MEROPE	Montana State University	Radiation experiment	Destroyed due to launch failure.	MC68HC812A4 (Motorola) 16-bit microcontroller 4kB EEPROM 1kB RAM Operating at 2MHz
AeroCube-1	The Aerospace Corporation	Short life satellite (10 days), using Lithium batteries as primary batteries (no recharging). Mission is to test a communication system and the system bus plus a suite of CMOS cameras done by Harvey Mudd College. The satellite has no deployables. Instead an omnidirectional patch antenna is used.	Destroyed due to launch failure.	NO MISSION INFORMATION AVAILABLE
CP2	California Politechnic Institute	Energy Dissipation Experiment. First mission based on what is supposed to be a "standardised" bus (though CP3 features an updated bus)	Destroyed due to launch failure.	PIC18LF6720 (Microchip) 16-bit microcontroller 1kB ROM 4kB RAM 128kB Flash Operating at 4MHz
ICE Cube2	Cornell University	Perform GPS scintillation science by measuring fluctuations in the signals that the GPS satellites emit when the signals pass through the ionosphere. Identical to ICE Cube 1	Destroyed due to launch failure.	NO MISSION INFORMATION AVAILABLE

MeaHuaka	University of Hawaii	To test a 5.8-GHz active antenna (grid oscillator) for high bandwidth communication (does not require deployment of antenna)	Destroyed due to launch failure.	RCM200 and2300(Rabbit Core) 16-bit microcontroller 256KB Flash 512KB SRAM Operates at 25.8MHz
CP1	California Politechnic Institute	Test of sun sensor developed by Optical Energy Technologies	Destroyed due to launch failure.	PIC18LF6720(Microchip) 8-bit microcontroller 128KB Flash (1kB boot ROM) 4KB RAM Operates at 4MHz
GeneSat-1	Center for Robotic Exploration and Space Technologies	Perform experiment on E. Coli bacteria in space, first CubeSat to carry a biological experiment.	96-hour experiment completed successfully on December 22nd, 2006. Last confirmed telemetry reception on April 11th, 2008 (2 year, 4 months after launch)	NO MISSION INFORMATION AVAILABLE. Only mention is that the C&DH subsystem used a PUC-class microcontroller.
CP4	California Politechnic Institute	Second flight unit of CP2, which was destroyed during the previous DNEPR launch	CP4 has been heard from numerous ground stations, but is not responding to telecommands. Seems it never reached full functionality. It is no longer operational	PIC18LF6720 (Microchip) 16-bit microcontroller 1kB ROM 4kB RAM 128kB Flash Operating at 4MHz
AeroCube-2	The Aerospace Corporation	Similar to AeroCube-1, except added charging system for the Lithium batteries. Mission is to test a communication system and the system bus plus a suite of CMOS cameras done by Harvey Mudd College. The satellite has no deployable. Instead an omnidirectional patch antenna is used.	Solar up-converter failed shortly after launch. Batteries dead. Shortly after ejection, AeroCube-2 also took a picture of CP4	NO MISSION INFORMATION AVAILABLE

CSTB-1	The Boeing Company	Testbed for components for future Boeing small-sat missions. Redundant radios, deployable antenna, various non-disclosed sensors	NO MISSION INFORMATION AVAILABLE	
CAPE-1	University of Louisiana	Camera payload (not more information could be sourced).	Has power system problems and is semi-operational (battery appears to be dead; currently only operates in the sun). CW has been received by several ground stations. 9600 bps TM packets have not been received by any station	NO MISSION INFORMATION AVAILABLE
Libertad-1	University of Sergio Arboleda	Camera and transmission of one stanza of the Colombian national anthem. Note: Powered by primary batteries only. They will last for about 52 days. This is the first Colombian satellite	Last confirmed telemetry on May 4th, 2007 (weak, undecoded until May 7, 2007). Designed with a 50 day lifespan.	Pumpkin Kit based on MSP430F149 (T. I) 16-bit microcontroller 60KB+256B of Flash 2KB of RAM Operates up to 10MHz
MAST	Theters Unlimited	Tether experiment (1 km Hoytether). ~1 million USD for the entire program	Only had contact to one of the two modules. Tether may have deployed "a little", but definitely not fully.	PIC 18F8720 (Microchip) 16-bit microcontroller 1KB of EEPROM Up to 1MB of Flash Operates up to 40MHz
CP3	California Polytechnic Institute	Three-axis magnetorquer experiment	Radio never established	NO MISSION INFORMATION AVAILABLE
CanX-2	University of Toronto	Will test instrumentation for future CanX missions including a propulsion system, momentum wheel, sun sensors, GPS receiver, CMOS camera (star tracker), and a new communication protocol. Scientific instrumentation and goals includes: Atmospheric spectrometer, GPS occultation experiment, and atomic oxygen material degradation experiment	TM received	2 x ARM7 (Atmel) 32-bit microcontrollers 2MB of SRAM 16MB of Flash Operates up to 15MHz

AAUsat-2	Alborg University	Detect gamma ray bursts by a gamma ray detector developed by the Danish National Space Centre	The satellite is tumbling, at one time up to 85 RPM, but using the magnetorquers the tumbling has been reduced to 35 RPM. The tumbling is making communication a challenge, they are currently using a 1 kW (yes, kilo watt) PA on the ground station. The satellite suffers from spontaneous reboots every 1-4 hours (typically). The payload has been powered on briefly, but no science data has been downlinked. They are in the process of uploading new software to the satellite.	AT91SAM7A1 (Atmel) 32-bit microcontroller 4KB of RAM External Bus Interface Operates up to 40MHz
Compass One	Fachhochschule Aachen	Technology demonstration of a miniature GPS receiver, and a transceiver for fast RF communication. A colour camera is implemented for p/r purposes	Pictures (although saturated) downlinked. High speed telemetry (4800 MSK) confirmed working	C8051F123 (Silicon Laboratory) 8448 Bytes of RAM 128KB of Flash Operates up to 100MHz
SEEDS (2)	Nihon University	Rebuild of the SEEDS CubeSat which was destroyed during June 26th, 2006 DNEPR launch failure. Contains a gyro sensor for accurate determination of attitude motion	Nominal operations, December 7th, 2008	NO MISSION INFORMATION AVAILABLE
AeroCube-3	Aerospace Corporation	New solar power subsystem to replace the one failing on AeroCube-2. Two foot diameter semi-spherical (8-panel) balloon that can serve as a de-orbit device as well as a tracking aid. The change in orbit life is estimated to be from 1-3 years (depending on atmosphere assumptions) without a balloon to 2-3 months with the balloon inflated. A VGA-resolution camera pointing in the direction of the balloon will photograph its state of inflation. 200' tether attached to the upper stage. AeroCube-3 will measure the dynamics while at the end of this tether. The tether will be cut some time into the mission	To be determined	NO MISSION INFORMATION AVAILABLE

Hawksat-1	Hawk Institute of Space Sciences	CubeSat platform demonstrator mission	To be determined	Pumpkin Kit based on MSP430F149 (T. I) 16-bit microcontroller 60KB+256B of Flash 2KB of RAM Operates up to 10MHz
Pharmasat-1	Santa Clara University	Follow on from GeneSat-1, will study the influence of microgravity on yeast resistance to an antifungal agent	To be determined	NO MISSION INFORMATION AVAILABLE
Polysat CP6	California Polytechnic Institute	Test attitude determination and control using magnetometers and magnetorquers. Also has two cameras and tether for deorbit experiment	To be determined	NO MISSION INFORMATION AVAILABLE
KySat	Couple of kentucky Universities	Build technological interest in students, much of the satellite is based on commercially available parts, including from the Pumpkin CubeSat kit	Awaiting launch	Pumpkin Kit based on MSP430F149 (T. I) 16-bit microcontroller 60KB+256B of Flash 2KB of RAM Operates up to 10MHz
AtmoCube	university of Trieste	Space weather monitoring. Contains a (supposedly modified for space use(?)) GPS receiver (Trimble M-Loc MPM Module), a radiation sensor, and a chip-based magnetometer based on magnetoresistance	Awaiting launch	H8/38076R (Renesas) 16-bit microcontroller 48 KB ROM 2KB RAM Operates up to 60MHz
e-st@r	Polytechnico di Torino	Demonstration of active 3-axis attitude control system inertial measurement unit.	Awaiting launch	Pumpkin Kit based on MSP430F149 (T. I) 16-bit microcontroller 60KB+256B of Flash 2KB of RAM Operates up to 10MHz
Goliat	university of Bucharest	Imaging of the Earth surface using digital camera and in-situ measurements of radiation dose and micrometeoroid flux (50x37 mm piezo-film). Part of the system is based on the Pumpkin CubeSat kit (2 pcs MSP430)	Awaiting launch	2xMSP430 16-bit microcontroller 60KB+256B of Flash 2KB of RAM Operates up to 10MHz
Oufti-1	University of Liege	Test use of D-STAR amateur radio digital communication protocol in space	Awaiting launch	Pumpkin Kit based on MSP430F149 (T. I) 16-bit microcontroller 60KB+256B of Flash 2KB of RAM Operates up to 10MHz

PW-Sat	Warsaw Universty of Technology	Test deployable atmospheric drag de-orbiting device	Awaiting launch	AT91SAM7X (Atmel) 16-bit microcontroller 8KB SRAM
Robusta	University of Montpelier	Test radiation effects on bipolar transistors	Awaiting launch	NO MISSION INFORMATION AVAILABLE
SwissCube	Polytechnical School of Lausanne	Observe oxygen emission in order to characterise the airglow intensity	Awaiting launch	AT91M55800A (Atmel) 32-bit microcontroller 128KB SRAM 512KB Flash
UNICubeSat	University of Rome	Performing in-situ measurements of atmospheric neutral density using Broglio drag balance instrument	Awaiting launch	Pumpkin Kit based on MSP430F149 (T. I) 16-bit microcontroller 60KB+256B of Flash 2KB of RAM Operates up to 10MHz
XaTcobeo	University of Vigo and INTA	Demonstrate software-defined radio and solar panel deployment	Awaiting launch	Virtex-II FPGA

Appendix B: Calculations and simulations (Overcurrent/voltage protection)

Simulations were conducted using Linear Technology Corporation's simulation package, LTSpice. From the available range of MAX4374 available, it was decided to go for the mid-range which is the MAX4374F with a current-sense amplifier gain of 50V/V and a maximum voltage drop of 300mV across $\mathrm{R_{SENSE}}$.

The first step in the overcurrent protecting circuit design using the MAX4374 is to give a current limit to the load which in this case is the OBC. When operating at full power, the microcontroller alone pulls a total of 400mA and assuming all other components average a total of 100mA, this brings the total to:

$$\mathrm{I_{max}} = 400\mathrm{mA} + 100\mathrm{mA} = 500\mathrm{mA} \tag{1}$$

With the maximum voltage drop $\mathrm{V_{RSENSE\_Max}}$ being 300mV, choosing a value of 0.100Ω for $\mathrm{R_{SENSE}}$ shall result in

$$\begin{aligned} \mathrm{V_{Rsense_{Max}}} &= \mathrm{I_{max}} \times \mathrm{R_{SENSE}} \\ &= 0.500 \times 0.100 \\ &= 0.05\mathrm{V} \end{aligned} \tag{2}$$

As mentioned previously, the MAX4374F was selected. With a voltage gain of 50V/V, the output to the current sense amplifier with $\mathrm{V_{RSENSE_{Max}}}$ being the differential input voltage is given by

$$\begin{aligned} \mathrm{V_{Out}} &= \mathrm{V_{RSENSE_{Max}}} \times 50 \\ &= 0.05 \times 50 \\ &= 2.5\mathrm{V} \end{aligned} \tag{3}$$

From equation (3), it is possible to determine the values of the voltage divider resistors $\mathrm{R_1}$ and $\mathrm{R_2}$ necessary to give an input voltage value of 0.6V at $\mathrm{C_{IN1}}$. This is done by choosing an arbitrary value for $\mathrm{R_2}$, say 220Ω, and then calculating the value of $\mathrm{R_1}$.

$$C_{IN1} = V_{Out}\left(\frac{R_2}{R_1 + R_2}\right)$$

$$0.6 = 2.5\left(\frac{220}{R_1+220}\right)$$

$$R_1 + 220 = \frac{550}{0.6}$$

$$R_1 = 916.67 - 220$$

$$= 696.67\Omega \quad (4)$$

This value can be practically implemented with a 680Ω resistor in series with a 15Ω resistor. These are main components necessary for the simulation.

The state of the output $C_{OUT}$is determined by the comparison between C_{IN1} and the internally generated reference of 0.6V. C_{OUT} is an open collector output and necessitates a pull-up resistor. The state of C_{OUT} varies according to table B.1. This signal is used to drive the n-channel MOSFET which is placed on the distribution line and which opens or closes the circuit depending on the value of the current being drawn.

Table B.1. Voltage levels at C_{in1} and C_{out} after RESET

Overload	V-Load-after RESET	C_{IN1}-after RESET	C_{OUT}-after RESET
Low	High	↑ > 0.6V	Low
High	Low	↓ < 0.6V	High

The short circuit condition will need to be simulated by means of two load resistors representing, respectively, the OBC's input impedance and the overload (short-circuit) condition. In order to make the overall current drawn by the OBC equal to 400mA which does not exceed the limit of 500mA, the total input impedance of the OBC, R_{Load} ,was given a value of

$$R_{Load} = \frac{V_{Supply}}{400mA}$$

$$= \frac{3.3}{400mA}$$

$$= 7.33\Omega \quad (5)$$

The short circuit condition which occurs when I_{Max} exceeds 500mA can be reached by adding a resistor in parallel with R_{Load}, making the overall resistance smaller and the current being drawn bigger. A value of 2.2Ω was chosen (R7) and a P-channel MOSFET was placed in series with a randomly generated pulse at the gate and the overload resistor at the drain, overloading the OBC at different intervals.

The resulting circuit schematic is shown in Figure A.1. An active low signal is necessary at the Reset input in order to pull C_{OUT} to a low state and re-activate the distribution line via n-channel MOSFET M5. The resulting waveforms are shown in Figure A.2 where each signal is represented by a different colour as a result of probes being placed at specific points during the run of the simulation.

It can be observed that for each time interval where the overload (navy blue mark) is in a high state (active), the value at C_{IN1} (red mark) drop to a level below 0.6V which represents the threshold value of the comparator. At this point, V_{load} (green mark) which represent the line voltage, will toggle its state from high to low. This will remain the case until a high pulse on the RESET input (purple mark) is observed. When this pulse occurs, the line voltage is restored and the comparator value at C_{IN1} (red mark) will rise back to 0.6V.

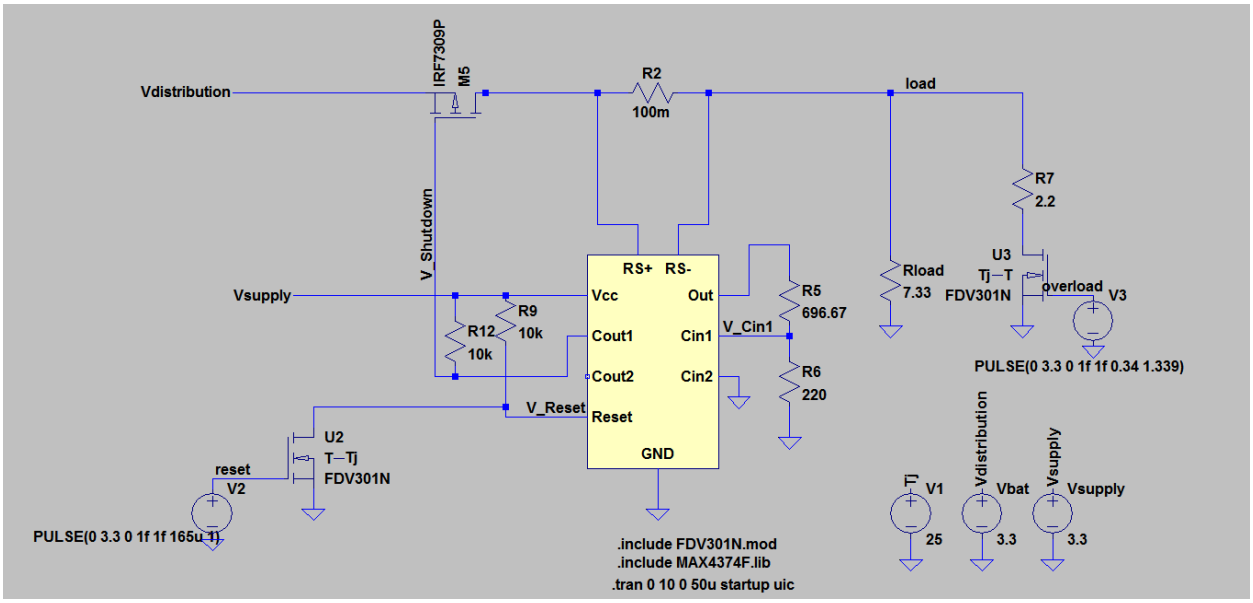


Figure B.1. Simulated overcurrent protection schematic

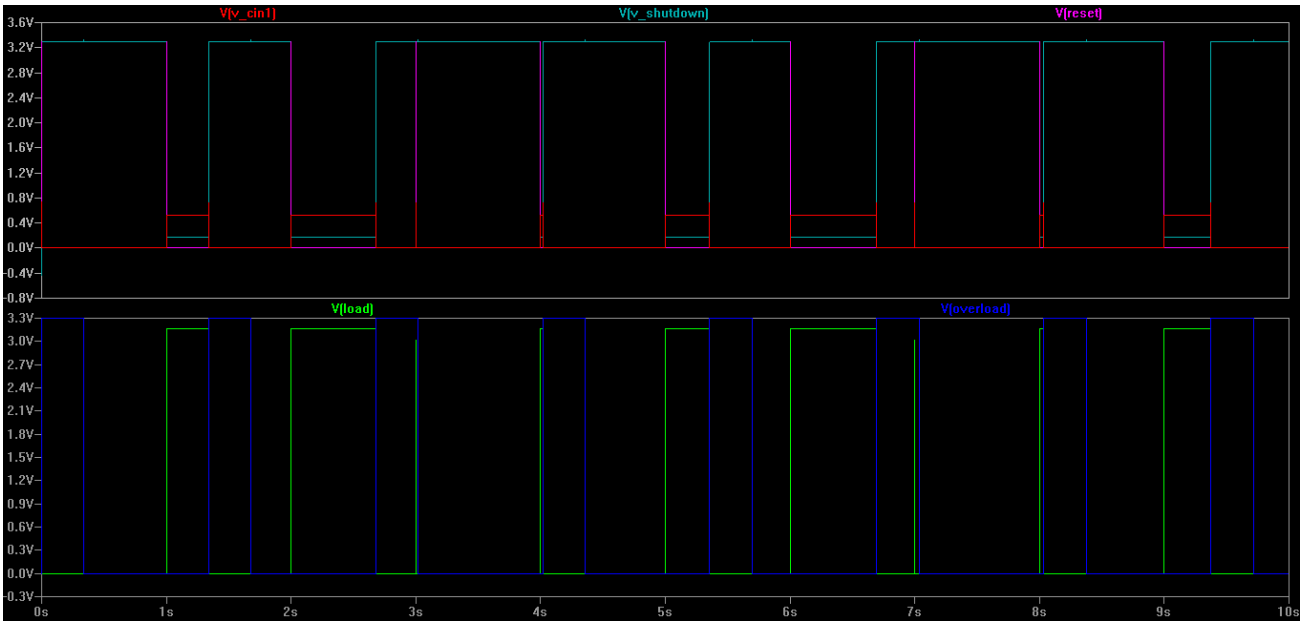


Figure B.2. Overcurrent protection simulation results waveforms

Appendix C: Datasheet Information

C.1. MCP9800 Temperature sensor

MCP9800/1/2/3

5.3 Registers

The MCP9800/1/2/3 has four registers that are user-accessible. These registers are specified as the Ambient Temperature (T_A) register, the Temperature Limit-set (T_{SET}) register, the Temperature Hysteresis (T_{HYST}) register and device Configuration (CONFIG) register.

The Ambient Temperature register is a read-only register and is used to access the ambient temperature data. The data from the ADC is loaded in parallel in the register. The Temperature Limit-set and Temperature Hysteresis registers are read/write registers that provide user-programmable temperature limits. If the ambient temperature drifts beyond the programmed limits, the MCP9800/1/2/3 outputs an alert signal using the ALERT pin (refer to **Section 5.3.4.3 "ALERT Output Configuration"**). The device Configuration register provides access for the user to configure the MCP9800/1/2/3's various features. These registers are described in further detail in the following sections.

The registers are accessed by sending Register Pointers to the MCP9800/1/2/3 using the serial interface. This is an 8-bit pointer. However, the two Least Significant bits (LSbs) are used as pointers and all other bits need to be cleared <0>. This device has additional registers that are reserved for test and calibration. If these registers are accessed, the device may not perform according to the specification. The pointer description is shown below.

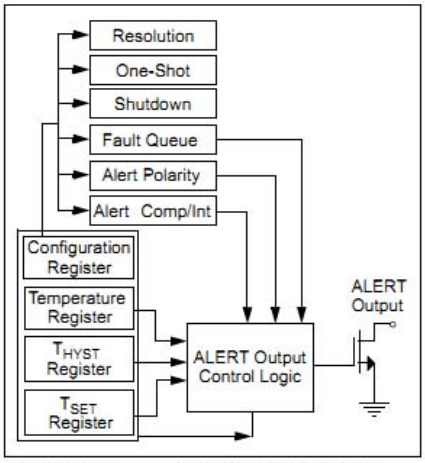


FIGURE 5-2: *Register Block Diagram.*

REGISTER 5-1: REGISTER POINTER

U-0	U-0	U-0	U-0	U-0	U-0	R/W-0	R/W-0
0	0	0	0	0	0	P1	P0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7-2 **Unimplemented:** Read as '0'

bit 1-0 **Px<1:0>:** Pointer bits
- 00 = Temperature register (T_A)
- 01 = Configuration register (CONFIG)
- 10 = Temperature Hysteresis register (T_{HYST})
- 11 = Temperature Limit-set register (T_{SET})

5.3.1 AMBIENT TEMPERATURE REGISTER (T_A)

The MCP9800/1/2/3 has a 16-bit read-only Ambient Temperature register that contains 9-bit to 12-bit temperature data. (0.5°C to 0.0625°C resolutions, respectively). This data is formatted in two's complement. The bit assignments, as well as the corresponding resolution, is shown in the register assignment below.

The refresh rate of this register depends on the selected ADC resolution. It takes 30 ms (typical) for 9-bit data and 240 ms (typical) for 12-bit data. Since this register is double-buffered, the user can read the register while the MCP9800/1/2/3 performs Analog-to-Digital conversion in the background. The decimal code to ambient temperature conversion is shown in Equation 5-2:

EQUATION 5-2:

$$T_A = \text{Code} \times 2^{-4}$$

Where:

T_A = Ambient Temperature (°C)
Code = MCP9800 output in decimal

REGISTER 5-2: AMBIENT TEMPERATURE REGISTER (T_A) – ADDRESS <0000 0000>b

Upper Half:							
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
Sign	2^6 °C	2^5 °C	2^4 °C	2^3 °C	2^2 °C	2^1 °C	2^0 °C
bit 15							bit 8

Lower Half:							
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
2^{-1} °C/bit	2^{-2} °C	2^{-3} °C	2^{-4} °C	0	0	0	0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Note 1: When the 0.5°C, 0.25°C or 0.125°C resolutions are selected, bit 6, bit 7 or bit 8 will remain clear <0>, respectively.

5.3.2 SENSOR CONFIGURATION REGISTER (CONFIG)

The MCP9800/1/2/3 has an 8-bit read/write Configuration register that allows the user to select the different features. These features include shutdown, ALERT output select as comparator or interrupt output, ALERT output polarity, fault queue cycle, temperature measurement resolution and One-shot mode (single conversion while in shutdown). These functions are described in detail in the following sections.

REGISTER 5-3: CONFIGURATION REGISTER (CONFIG) – ADDRESS <0000 0001>b

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
One-Shot	Resolution		Fault Queue		ALERT Polarity	COMP/INT	Shutdown
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **ONE-SHOT** bit
- 1 = Enabled
- 0 = Disabled (Power-up default)

bit 5-6 **ΣΔ ADC RESOLUTION** bits
- 00 = 9 bit or 0.5°C (Power-up default)
- 01 = 10 bit or 0.25°C
- 10 = 11 bit or 0.125°C
- 11 = 12 bit or 0.0625°C

bit 3-4 **FAULT QUEUE** bits
- 00 = 1 (Power-up default)
- 01 = 2
- 10 = 4
- 11 = 6

bit 2 **ALERT POLARITY** bit
- 1 = Active-high
- 0 = Active-low (Power-up default)

bit 1 **COMP/INT** bit
- 1 = Interrupt mode
- 0 = Comparator mode (Power-up default)

bit 0 **SHUTDOWN** bit
- 1 = Enable
- 0 = Disable (Power-up default)

5.3.3 TEMPERATURE HYSTERESIS REGISTER (T_{HYST})

The MCP9800/1/2/3 has a 16-bit read/write Temperature Hysteresis register that contains a 9-bit data in two's compliment format. This register is used to set a hysteresis for the T_{SET} limit. Therefore, the data represents a minimum temperature limit. If the ambient temperature drifts below the specified limit, the MCP9800/1/2/3 asserts an alert output (refer to **Section 5.3.4.3 "ALERT Output Configuration"**).

This register uses the nine Most Significant bits (MSbs) and all other bits are "don't cares".

The power-up default value of T_{HYST} register is 75°C, or <0100 1011 0>b in binary.

REGISTER 5-4: TEMPERATURE HYSTERESIS REGISTER (T_{HYST}) – ADDRESS <0000 0010>b

Upper Half:							
R/W-0	R/W-1	R/W-0	R/W-0	R/W-1	R/W-0	R/W-1	R/W-1
Sign	2^6 °C	2^5 °C	2^4 °C	2^3 °C	2^2 °C	2^1 °C	2^0 °C
bit 15							bit 8

Lower Half:							
R/W-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
2^{-1} °C	0	0	0	0	0	0	0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

5.3.4 TEMPERATURE LIMIT-SET REGISTER (T_{SET})

The MCP9800/1/2/3 has a 16-bit read/write Temperature Limit-Set register (T_{SET}) which contains a 9-bit data in two's compliment format. This data represents a maximum temperature limit. If the ambient temperature exceeds this specified limit, the MCP9800/1/2/3 asserts an alert output. (Refer to **Section 5.3.4.3 "ALERT Output Configuration"**).

This register uses the nine Most Significant bits (MSbs) and all other bits are "don't cares".

The power-up default value of the T_{SET} register is 80°C, or <0101 0000 0>b in binary.

REGISTER 5-5: TEMPERATURE LIMIT-SET REGISTER (T_{SET}) – ADDRESS <0000 0011>b

Upper Half:							
R/W-0	R/W-1	R/W-0	R/W-1	R/W-0	R/W-0	R/W-0	R/W-0
Sign	2^6 °C	2^5 °C	2^4 °C	2^3 °C	2^2 °C	2^1 °C	2^0 °C
bit 15							bit 8

Lower Half:							
R/W-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
2^{-1} °C	0	0	0	0	0	0	0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

5.3.4.1 Shutdown Mode

The Shutdown mode disables all power-consuming activities (including temperature sampling operations) while leaving the serial interface active. The device consumes 2 µA (maximum) in this mode. It remains in this mode until the Configuration register is updated to enable continuous conversion or until power is recycled.

In Shutdown mode, the CONFIG, T_A, T_{SET} and T_{HYST} registers can be read or written to; however, the serial bus activity will increase the shutdown current.

5.3.4.2 One-Shot Mode

The MCP9800/1/2/3 can also be used in a One-shot mode that can be selected using bit 7 of the CONFIG register. The One-shot mode performs a single temperature measurement and returns to Shutdown mode. This mode is especially useful for low-power applications where temperature is measured upon command from a controller. For example, a 9-bit T_A in One-shot mode consumes 200 µA (typical) for 30 ms and 0.1 µA (typical) during shutdown.

To access this feature, the device needs to initially be in Shutdown mode. This is done by sending a byte to the CONFIG register with bit 0 set <1> and bit 7 cleared <0>. Once the device is in Shutdown mode, the CONFIG register needs to be written to again, with bit 0 and bit 7 set <1>. This begins the single conversion cycle of t_{CONV}, 30ms for 9-bit data. Once the conversion is completed, T_A is updated and bit 7 of CONFIG becomes cleared <0> by the MCP9800/1/2/3.

TABLE 5-2: SHUTDOWN AND ONE-SHOT MODE DESCRIPTION

Operational Mode	One-Shot (Bit 7)	Shutdown (Bit 0)
Continuous Conversion	0	0
Shutdown	0	1
Continuous Conversion (One-shot is ignored)	1	0
One-shot (Note 1)	1	1

Note 1: The shutdown command <01> needs to be programmed before sending a one-shot command <11>.

5.3.4.3 ALERT Output Configuration

The ALERT output can be configured as either a comparator output or as Interrupt Output mode using bit 1 of CONFIG. The polarity can also be specified as an active-high or active-low using bit 2 of CONFIG. The following sections describe each output mode, while Figure 5-7 gives a graphical description.

5.3.4.4 Comparator Mode

In Comparator mode, the ALERT output is asserted when T_A is greater than T_{SET}. The pin remains active until T_A is lower than T_{HYST}. The Comparator mode is useful for thermostat-type applications, such as turning on a cooling fan or triggering a system shutdown when the temperature exceeds a safe operating range.

In Comparator mode, if the device enters the Shutdown mode with asserted ALERT output, the output remains active during shutdown. The device must be operating in continuous conversion, with T_A below T_{HYST}, for the ALERT output to be deasserted.

5.3.4.5 Interrupt Mode

In Interrupt mode, the ALERT output is asserted when T_A is greater than T_{SET}. However, the output is deasserted when the user performs a read from any register. This mode is designed for interrupt-driven, microcontroller-based systems. The microcontroller receiving the interrupt will have to acknowledge the interrupt by reading any register from the MCP9800/1/2/3. This will clear the interrupt and the ALERT pin will become deasserted. When T_A drifts below T_{HYST}, the MCP9800/1/2/3 outputs another interrupt and the controller needs to read a register to deassert the ALERT output. Shutting down the device will also reset, or deassert, the ALERT output.

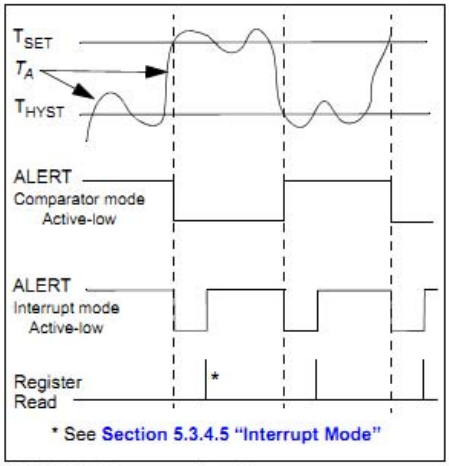


FIGURE 5-7: *Alert Output.*

5.3.4.6 Fault Queue

The fault queue feature can be used as a filter to lessen the probability of spurious activation of the ALERT pin. T_A must remain above T_{SET} for the consecutive number of conversion cycles selected using the Fault Queue bits. Bit 3 and bit 4 of CONFIG can be used to select up to six fault queue cycles. For example, if six fault queues are selected, T_A must be greater than T_{SET} for six consecutive conversions before ALERT is asserted as a comparator or an interrupt output.

This queue setting also applies for T_{HYST}. If six fault queues are selected, T_A must remain below T_{HYST} for six consecutive conversions before ALERT is deasserted (Comparator mode) or before another interrupt is asserted (Interrupt mode).

5.3.4.7 ΣΔ ADC Resolution

The MCP9800/1/2/3 provides access to select the ADC resolution from 9-bit to 12-bit (0.5°C to 0.0625°C resolution) using bit 6 and bit 5 of the CONFIG register. The user can gain better insight into the trends and characteristics of the ambient temperature by using a finer resolution. Increasing the resolution also reduces the quantization error. Figure 2-3 shows accuracy versus resolution.

Table 5-3 shows the T_A register conversion time for the corresponding resolution.

TABLE 5-3: RESOLUTION AND CONVERSION TIME

Bits	Resolution	t_{CONV} (typical)
9	0.5	30 ms
10	0.25	60 ms
11	0.125	120 ms
12	0.0625	240 ms

5.4 Summary of Power-up Condition

The MCP9800/1/2/3 has an internal Power-on Reset (POR) circuit. If the power supply voltage V_{DD} glitches down to the 1.7V (typical) threshold, the device resets the registers to the power-up default settings.

Table 5-4 shows the power-up default summary.

TABLE 5-4: POWER-UP DEFAULTS

Register	Data (Hex)	Power-up Defaults
T_A	0000	0°C
T_{SET}	A000	80°C
T_{HYST}	9600	75°C
Pointer	00	Temperature register
CONFIG	00	Continuous Conversion Comparator mode Active-low Output Fault Queue 1 9-bit Resolution

At power-up, the MCP9800/1/2/3 has an inherent 2 ms (typical) power-up delay before updating the registers with default values and start a conversion cycle. This delay reduces register corruption due to unsettled power. After power-up, it takes t_{CONV} for the TCN75A to update the T_A register with valid temperature data.

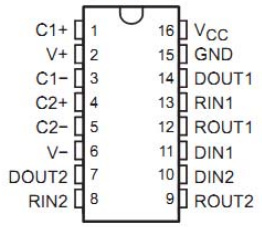
C.2. MAX3232

MAX3232
3-V TO 5.5-V MULTICHANNEL RS-232 LINE DRIVER/RECEIVER WITH ±15-kV ESD PROTECTION

SLLS410I – JANUARY 2000 – REVISED JANUARY 2004

- RS-232 Bus-Pin ESD Protection Exceeds ±15 kV Using Human-Body Model (HBM)
- Meets or Exceeds the Requirements of TIA/EIA-232-F and ITU v.28 Standards
- Operates With 3-V to 5.5-V V_{CC} Supply
- Operates Up To 250 kbit/s
- Two Drivers and Two Receivers
- Low Supply Current . . . 300 μA Typical
- External Capacitors . . . 4 × 0.1 μF
- Accepts 5-V Logic Input With 3.3-V Supply
- Alternative High-Speed Pin-Compatible Device (1 Mbit/s)
  - SNx5C3232
- Applications
  - Battery-Powered Systems, PDAs, Notebooks, Laptops, Palmtop PCs, and Hand-Held Equipment

D, DB, DW, OR PW PACKAGE
(TOP VIEW)



description/ordering information

ORDERING INFORMATION

T_A	PACKAGE†		ORDERABLE PART NUMBER	TOP-SIDE MARKING
−0°C to 70°C	SOIC (D)	Tube of 40	MAX3232CD	MAX3232C
		Reel of 2500	MAX3232CDR	
	SOIC (DW)	Tube of 40	MAX3232CDW	MAX3232C
		Reel of 2000	MAX3232CDWR	
	SSOP (DB)	Tube of 80	MAX3232CDB	MA3232C
		Reel of 2000	MAX3232CDBR	
	TSSOP (PW)	Tube of 90	MAX3232CPW	MA3232C
		Reel of 2000	MAX3232CPWR	
−40°C to 85°C	SOIC (D)	Tube of 40	MAX3232ID	MAX3232I
		Reel of 2500	MAX3232IDR	
	SOIC (DW)	Tube of 40	MAX3232IDW	MAX3232I
		Reel of 2000	MAX3232IDWR	
	SSOP (DB)	Tube of 80	MAX3232IDB	MB3232I
		Reel of 2000	MAX3232IDBR	
	TSSOP (PW)	Tube of 90	MAX3232IPW	MB3232I
		Reel of 2000	MAX3232IPWR	

† Package drawings, standard packing quantities, thermal data, symbolization, and PCB design guidelines are available at www.ti.com/sc/package.

TEXAS INSTRUMENTS
POST OFFICE BOX 655303 • DALLAS, TEXAS 75265

description/ordering information (continued)

The MAX3232 device consists of two line drivers, two line receivers, and a dual charge-pump circuit with ±15-kV ESD protection pin to pin (serial-port connection pins, including GND). The device meets the requirements of TIA/EIA-232-F and provides the electrical interface between an asynchronous communication controller and the serial-port connector. The charge pump and four small external capacitors allow operation from a single 3-V to 5.5-V supply. The devices operate at data signaling rates up to 250 kbit/s and a maximum of 30-V/µs driver output slew rate.

Function Tables

EACH DRIVER

INPUT DIN	OUTPUT DOUT
L	H
H	L

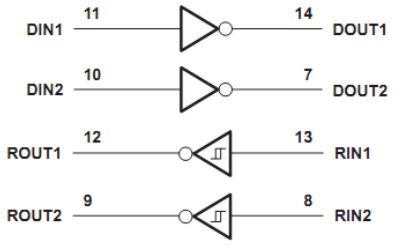
H = high level, L = low level

EACH RECEIVER

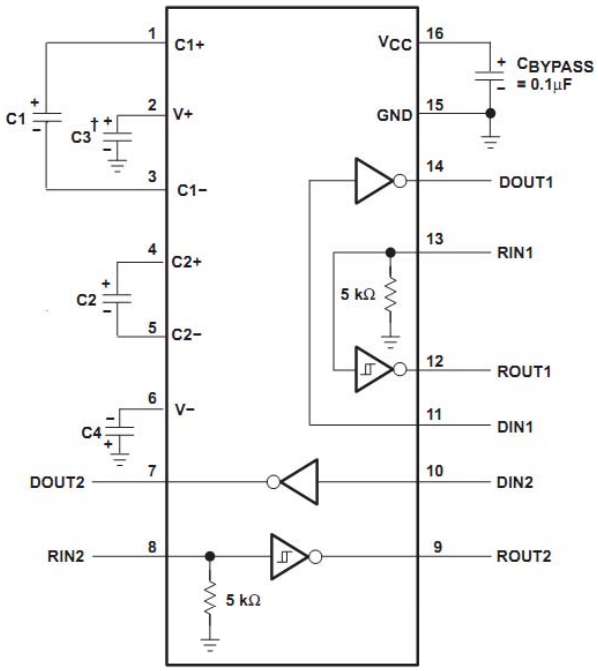
INPUT RIN	OUTPUT ROUT
L	H
H	L
Open	H

H = high level, L = low level, Open = input disconnected or connected driver off

logic diagram (positive logic)



APPLICATION INFORMATION



† C3 can be connected to V_{CC} or GND.

NOTES: A. Resistor values shown are nominal.
B. Nonpolarized ceramic capacitors are acceptable. If polarized tantalum or electrolytic capacitors are used, they should be connected as shown.

V_{CC} vs CAPACITOR VALUES

V_{CC}	C1	C2, C3, C4
3.3 V ± 0.3 V	0.1 μF	0.1 μF
5 V ± 0.5 V	0.047 μF	0.33 μF
3 V to 5.5 V	0.1 μF	0.47 μF

Figure 4. Typical Operating Circuit and Capacitor Values

C.3. AT91SAM3U4E PIO Controllers tables

10.2.1 PIO Controller A Multiplexing

Table 10-2. Multiplexing on PIO Controller A (PIOA)

I/O Line	Peripheral A	Peripheral B	Extra Function	Comments
PA0	TIOB0	NPCS1	WKUP0[1][2]	
PA1	TIOA0	NPCS2	WKUP1[1][2]	
PA2	TCLK0	AD12BTRG	WKUP2[1][2]	
PA3	MCCK	PCK1		
PA4	MCCDA	PWMH0		
PA5	MCDA0	PWMH1		
PA6	MCDA1	PWMH2		
PA7	MCDA2	PWML0		
PA8	MCDA3	PWML1		
PA9	TWD0	PWML2	WKUP3[1][2]	
PA10	TWCK0	PWML3	WKUP4[1][2]	
PA11	URXD	PWMFI0		
PA12	UTXD	PWMFI1		
PA13	MISO			
PA14	MOSI			
PA15	SPCK	PWMH2		
PA16	NPCS0	NCS1	WKUP5[1][2]	
PA17	SCK0	ADTRG	WKUP6[1][2]	
PA18	TXD0	PWMFI2	WKUP7[1][2]	
PA19	RXD0	NPCS3	WKUP8[1][2]	
PA20	TXD1	PWMH3	WKUP9[1][2]	
PA21	RXD1	PCK0	WKUP10[1][2]	
PA22	TXD2	RTS1	AD12B0	
PA23	RXD2	CTS1		
PA24	TWD1	SCK1	WKUP11[1][2]	
PA25	TWCK1	SCK2	WKUP12[1][2]	
PA26	TD	TCLK2		
PA27	RD	PCK0		
PA28	TK	PWMH0		
PA29	RK	PWMH1		
PA30	TF	TIOA2	AD12B1	
PA31	RF	TIOB2		

Notes: 1. Wake-Up source in Backup mode (managed by the SUPC).
2. Fast Start-Up source in Wait mode (managed by the PMC).

10.2.2 PIO Controller B Multiplexing

Table 10-3. Multiplexing on PIO Controller B (PIOB)

I/O Line	Peripheral A	Peripheral B	Extra Function	Comments
PB0	PWMH0	A2	WKUP13[1][2]	
PB1	PWMH1	A3	WKUP14[1][2]	
PB2	PWMH2	A4	WKUP15[1][2]	
PB3	PWMH3	A5	AD12BAB2	
PB4	TCLK1	A6	AD12BAB3	
PB5	TIOA1	A7	AD0	
PB6	TIOB1	D15	AD1	
PB7	RTS0	A0/NBS0	AD2	
PB8	CTS0	A1	AD3	
PB9	D0	DTR0		
PB10	D1	DSR0		
PB11	D2	DCD0		
PB12	D3	RI0		
PB13	D4	PWMH0		
PB14	D5	PWMH1		
PB15	D6	PWMH2		
PB16	D7	PWMH3		
PB17	NANDOE	PWML0		
PB18	NANDWE	PWML1		
PB19	NRD	PWML2		
PB20	NCS0	PWML3		
PB21	A21/NANDALE	RTS2		
PB22	A22/NANDCLE	CTS2		
PB23	NWR0/NWE	PCK2		
PB24	NANDRDY	PCK1		
PB25	D8	PWML0		Only on 144-pin version
PB26	D9	PWML1		Only on 144-pin version
PB27	D10	PWML2		Only on 144-pin version
PB28	D11	PWML3		Only on 144-pin version
PB29	D12			Only on 144-pin version
PB30	D13			Only on 144-pin version
PB31	D14			Only on 144-pin version

Notes: 1. Wake-Up source in Backup mode (managed by the SUPC).
2. Fast Start-Up source in Wait mode (managed by the PMC).

10.2.3 PIO Controller C Multiplexing

Table 10-4. Multiplexing on PIO Controller C (PIOC)

I/O Line	Peripheral A	Peripheral B	Extra function	Comments
PC0	A2			Only on 144-pin version
PC1	A3			Only on 144-pin version
PC2	A4			Only on 144-pin version
PC3	A5	NPCS1		Only on 144-pin version
PC4	A6	NPCS2		Only on 144-pin version
PC5	A7	NPCS3		Only on 144-pin version
PC6	A8	PWML0		Only on 144-pin version
PC7	A9	PWML1		Only on 144-pin version
PC8	A10	PWML2		Only on 144-pin version
PC9	A11	PWML3		Only on 144-pin version
PC10	A12	CTS3		Only on 144-pin version
PC11	A13	RTS3		Only on 144-pin version
PC12	NCS1	TXD3		Only on 144-pin version
PC13	A2	RXD3		Only on 144-pin version
PC14	A3	NPCS2		Only on 144-pin version
PC15	NWR1/NBS1		AD12B4	Only on 144-pin version
PC16	NCS2	PWML3	AD12B5	Only on 144-pin version
PC17	NCS3		AD12B6	Only on 144-pin version
PC18	NWAIT		AD12B7	Only on 144-pin version
PC19	SCK3	NPCS1		Only on 144-pin version
PC20	A14			Only on 144-pin version
PC21	A15			Only on 144-pin version
PC22	A16			Only on 144-pin version
PC23	A17			Only on 144-pin version
PC24	A18	PWMH0		Only on 144-pin version
PC25	A19	PWMH1		Only on 144-pin version
PC26	A20	PWMH2		Only on 144-pin version
PC27	A23	PWMH3		Only on 144-pin version
PC28		MCDA4	AD4	Only on 144-pin version
PC29	PWML0	MCDA5	AD5	Only on 144-pin version
PC30	PWML1	MCDA6	AD6	Only on 144-pin version
PC31	PWML2	MCDA7	AD7	Only on 144-pin version

Notes: 1. Wake-Up source in Backup mode (managed by the SUPC).
2. Fast Start-Up source in Wait mode (managed by the PMC).

Appendix D: OBC Printed Circuit Board

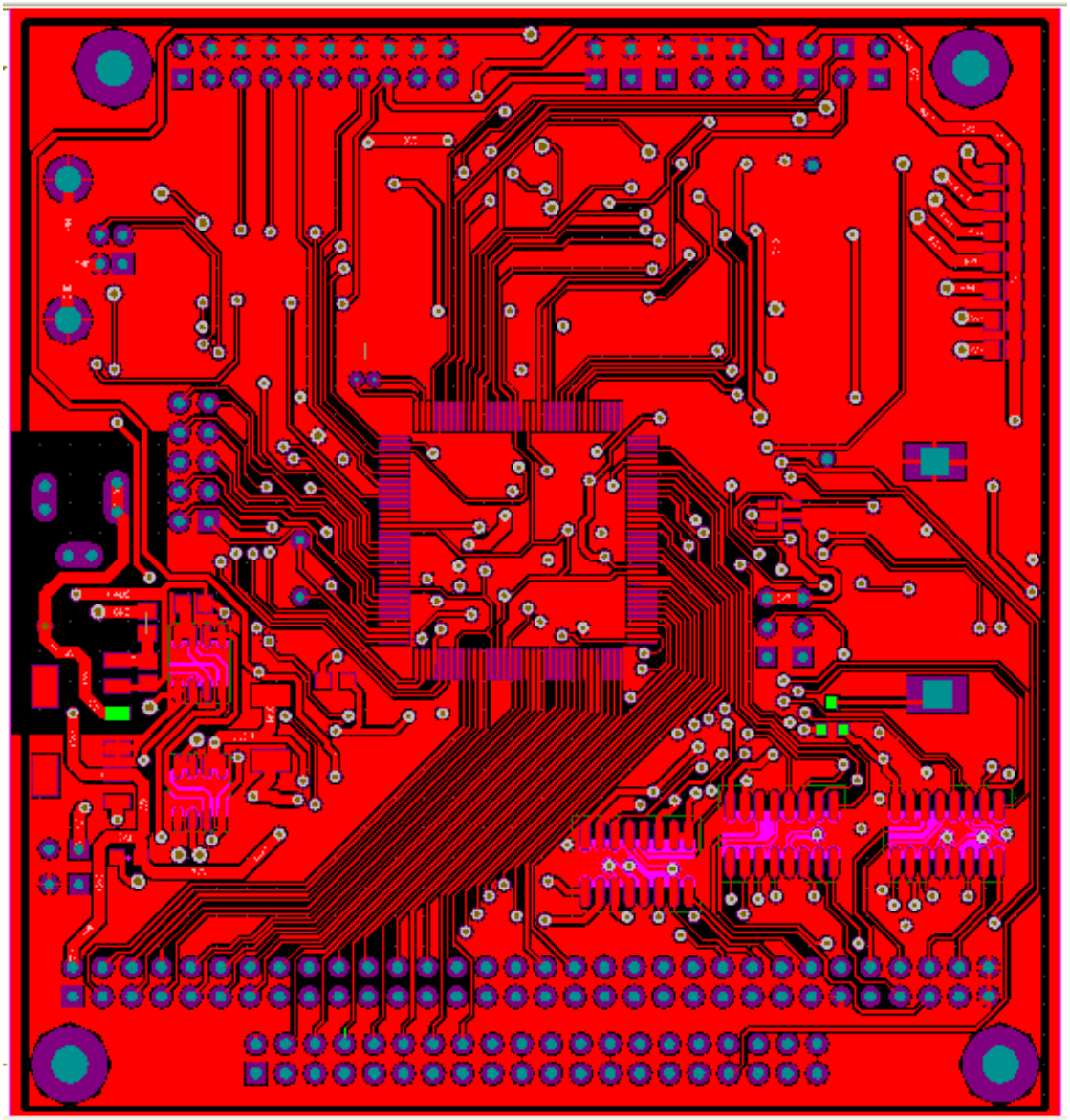


Figure D.1: OBC Top Layer (Altium designer)

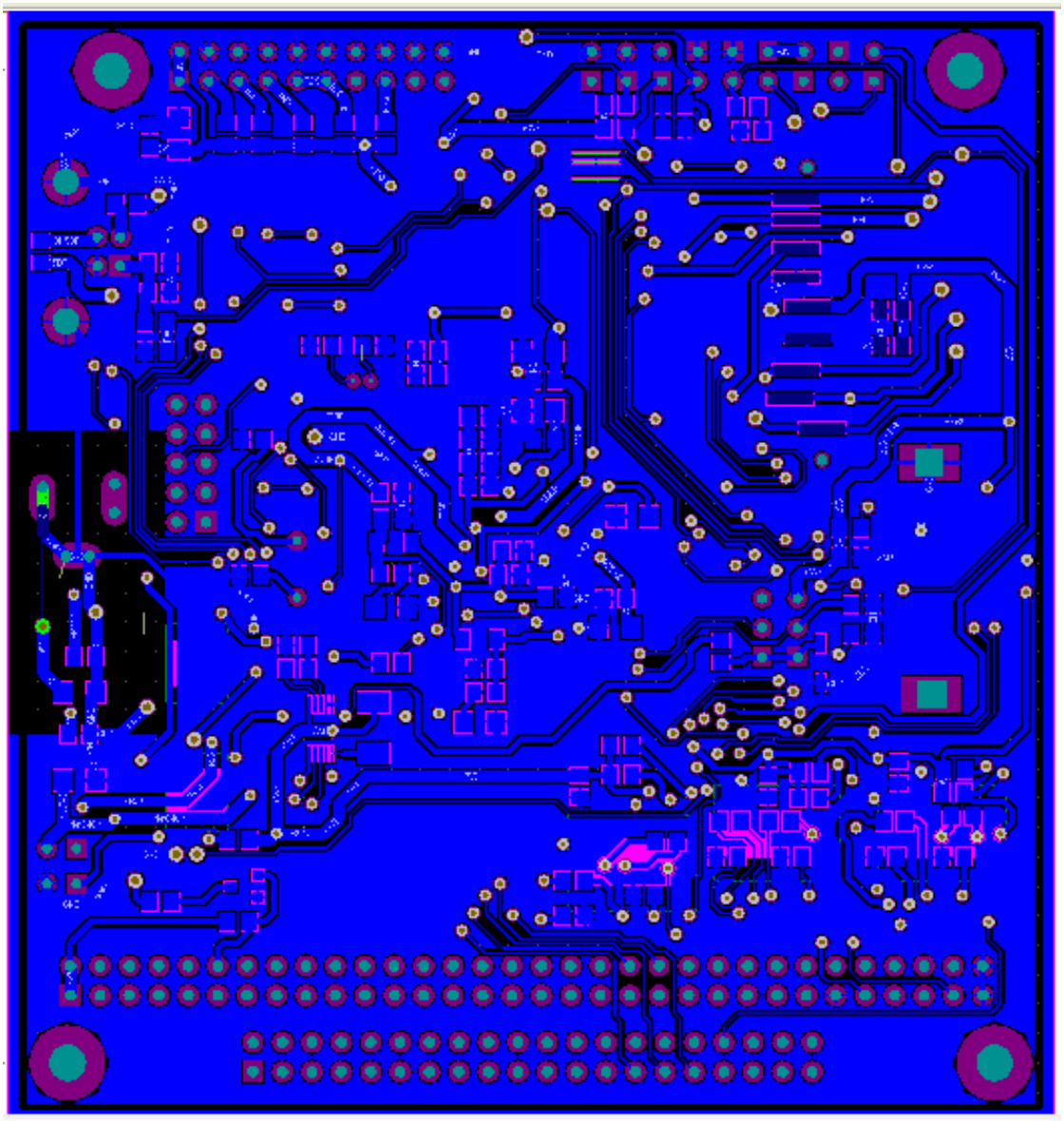


Figure D.2: OBC Bottom Layer (Altium designer)

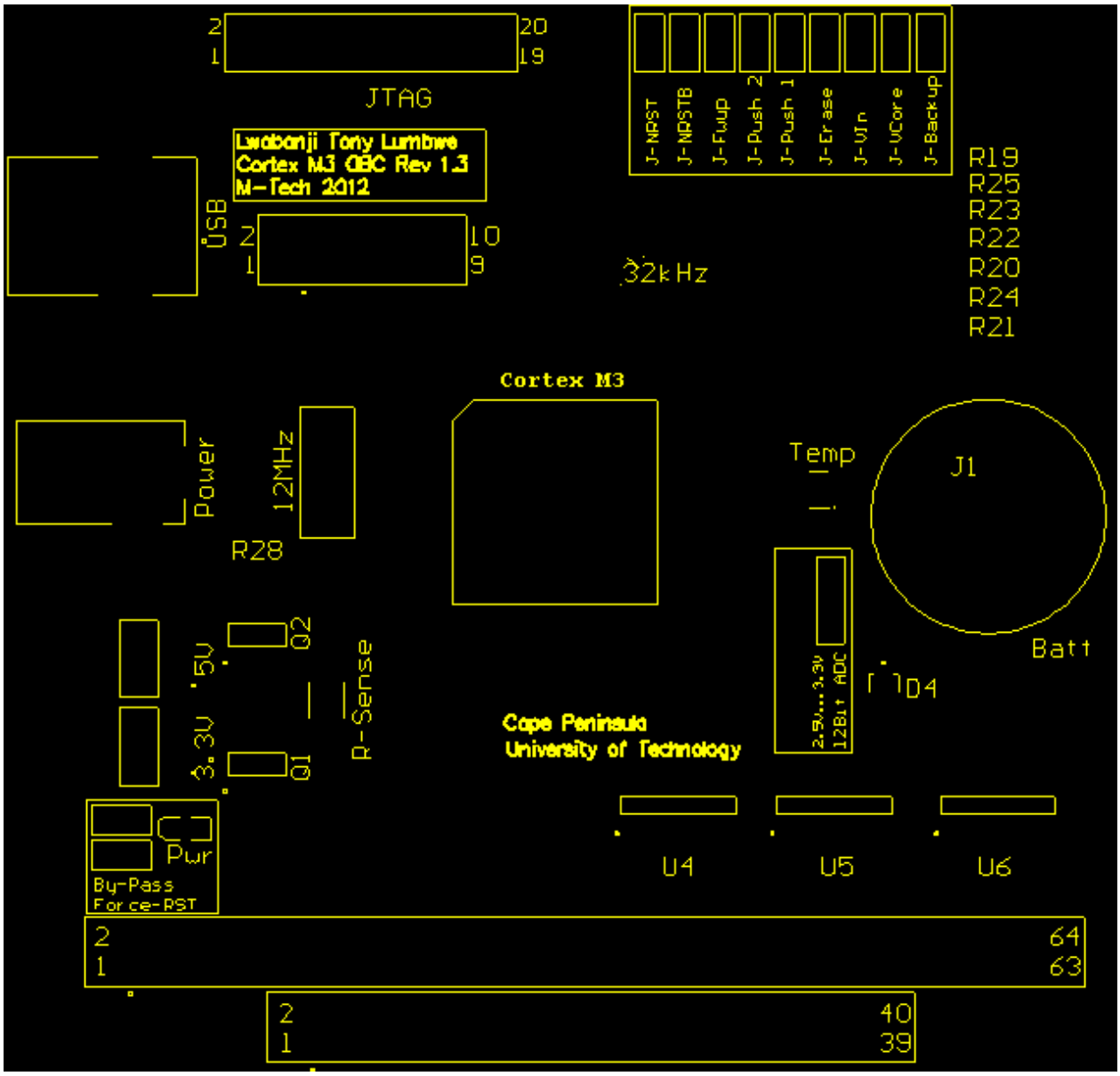


Figure D.3: OBC Top Silkscreen Overlay (Altium designer)

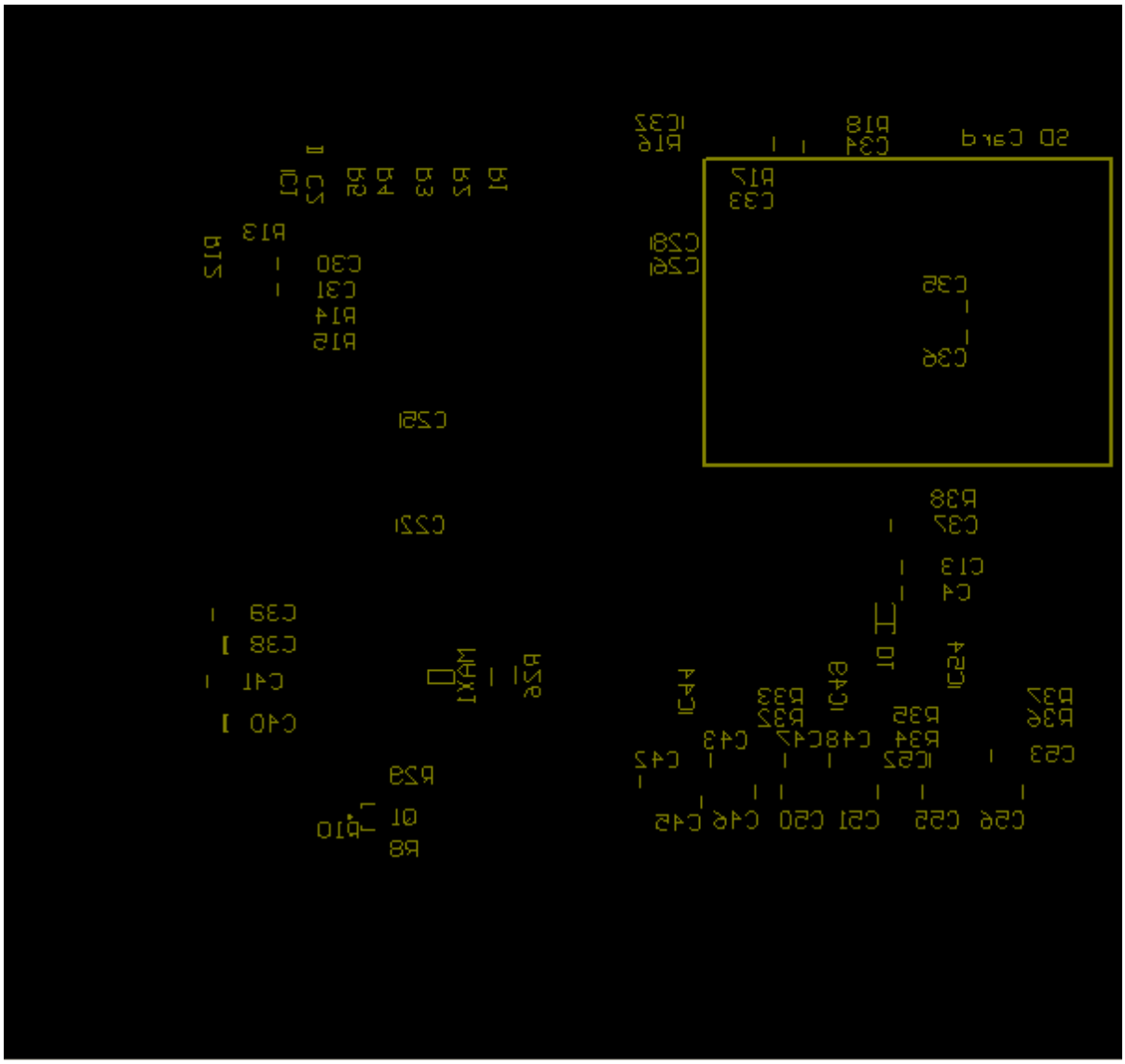


Figure D.4: OBC Top Silkscreen Overlay (Altium designer)

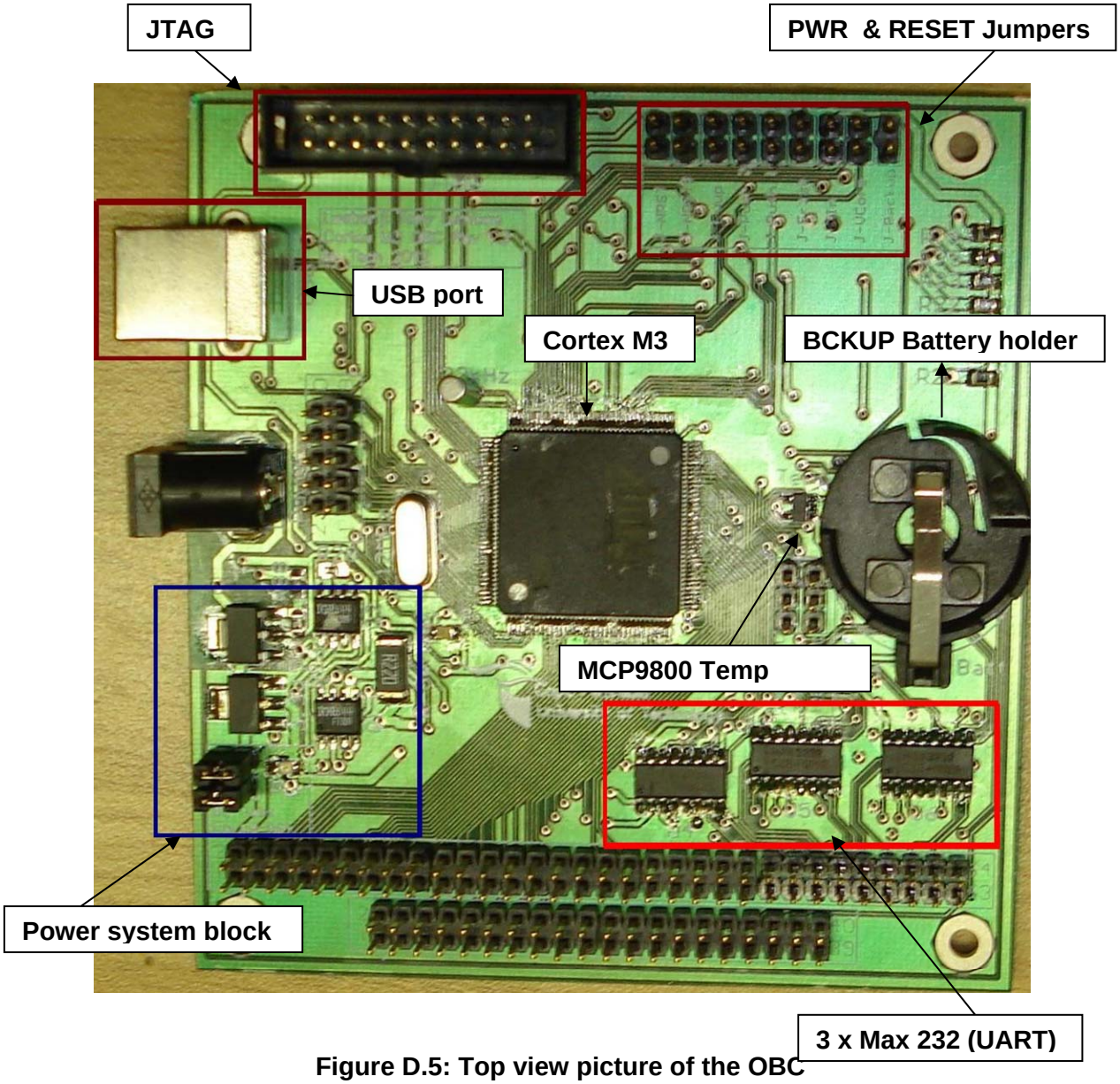


Figure D.5: Top view picture of the OBC

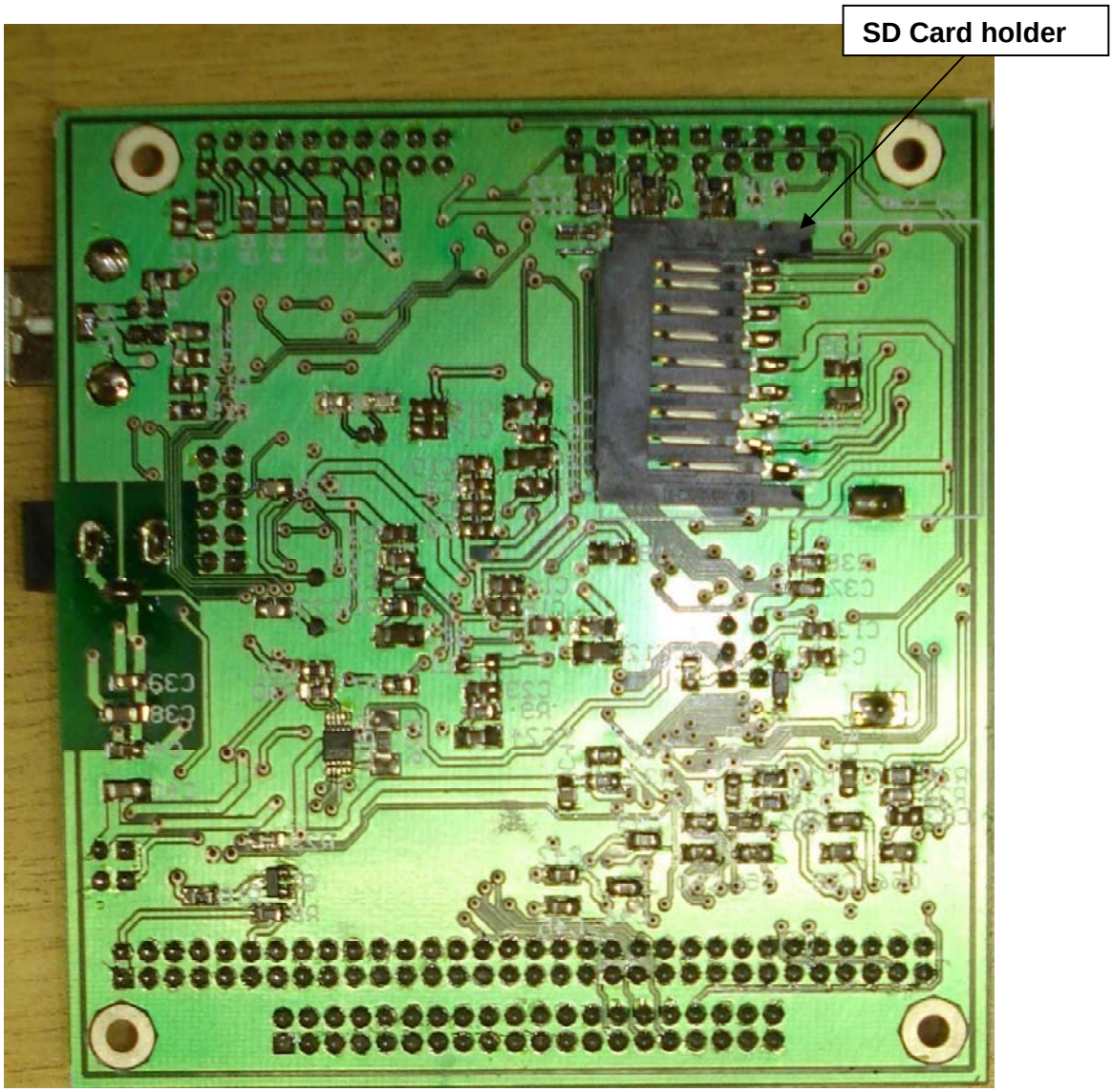
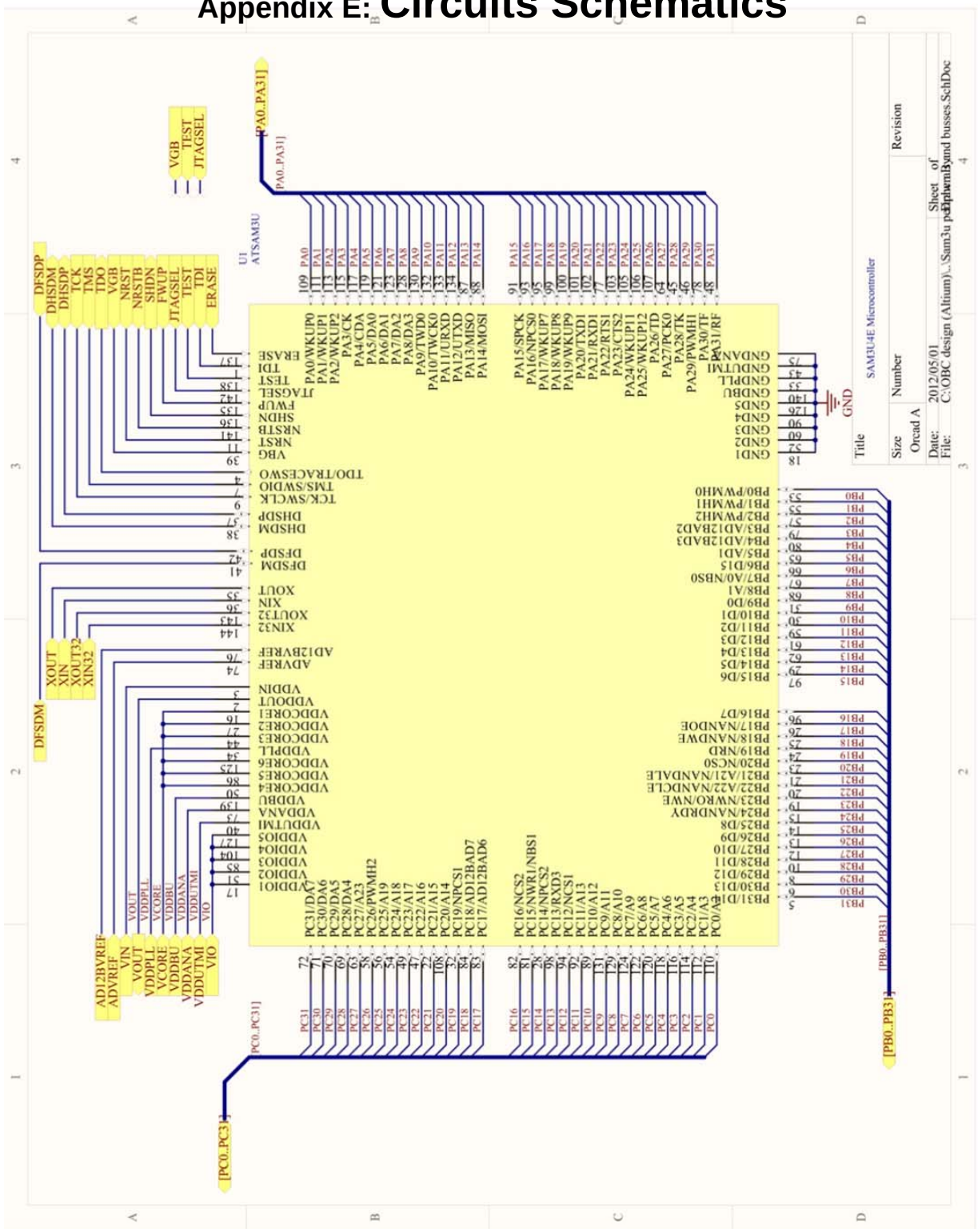
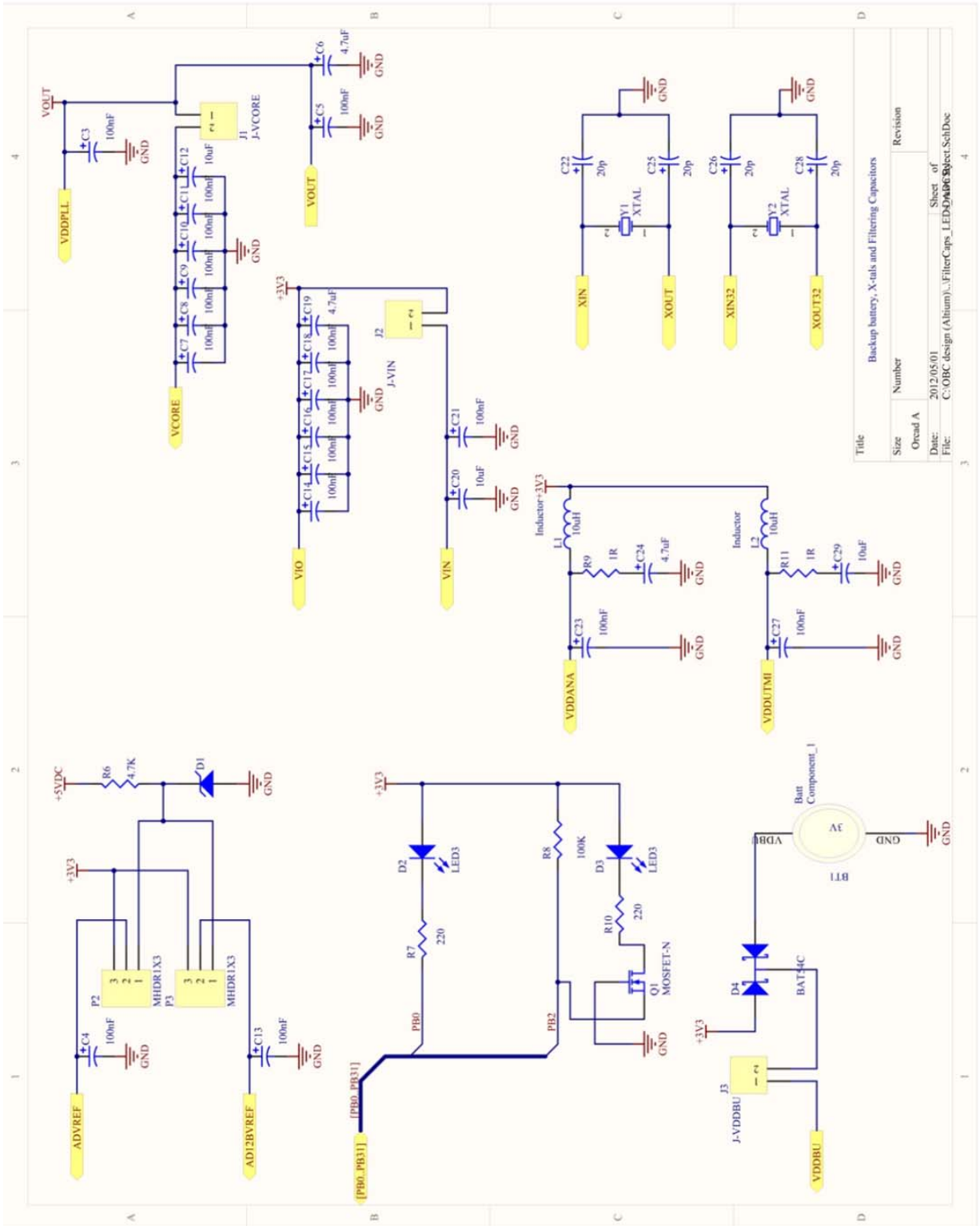
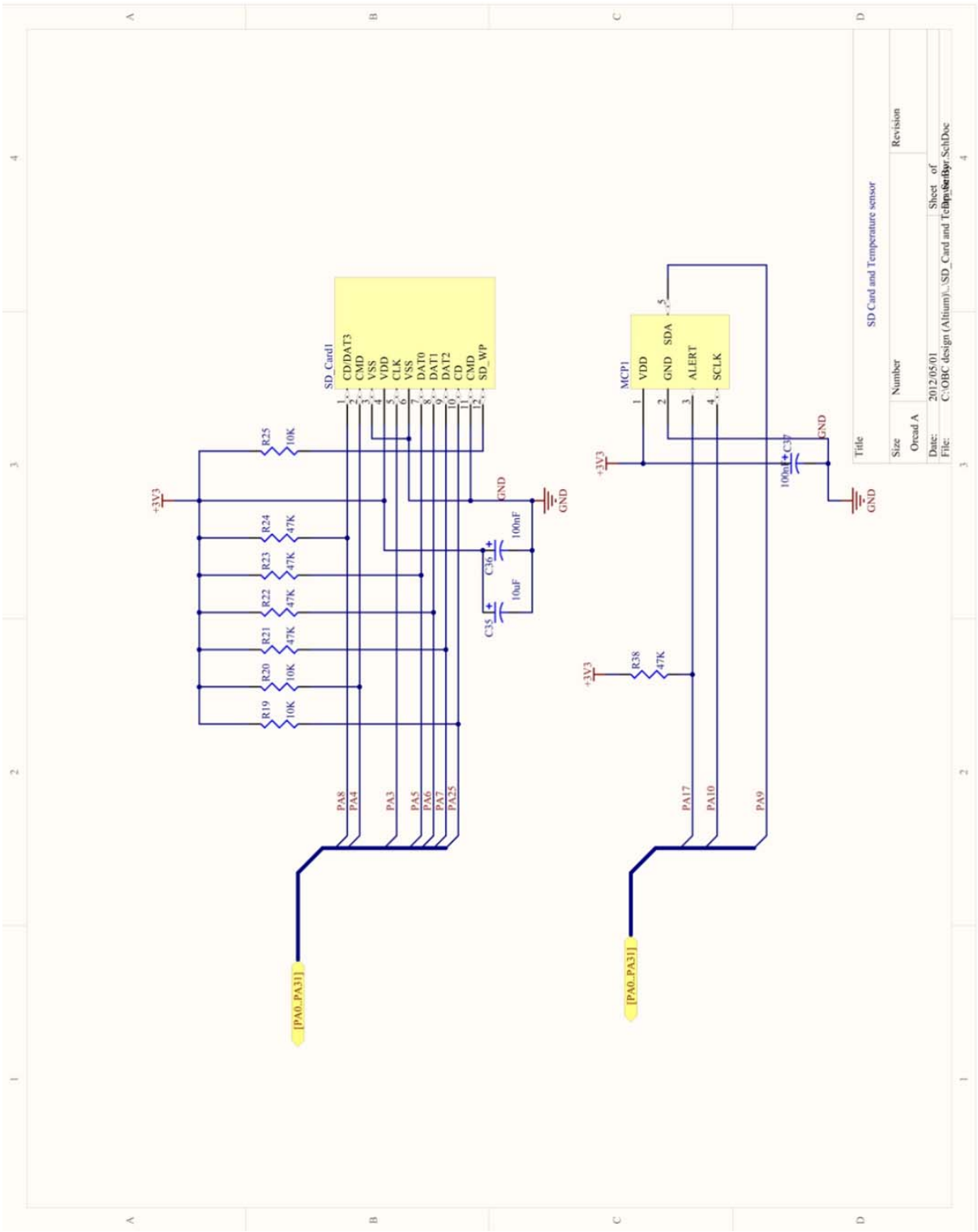


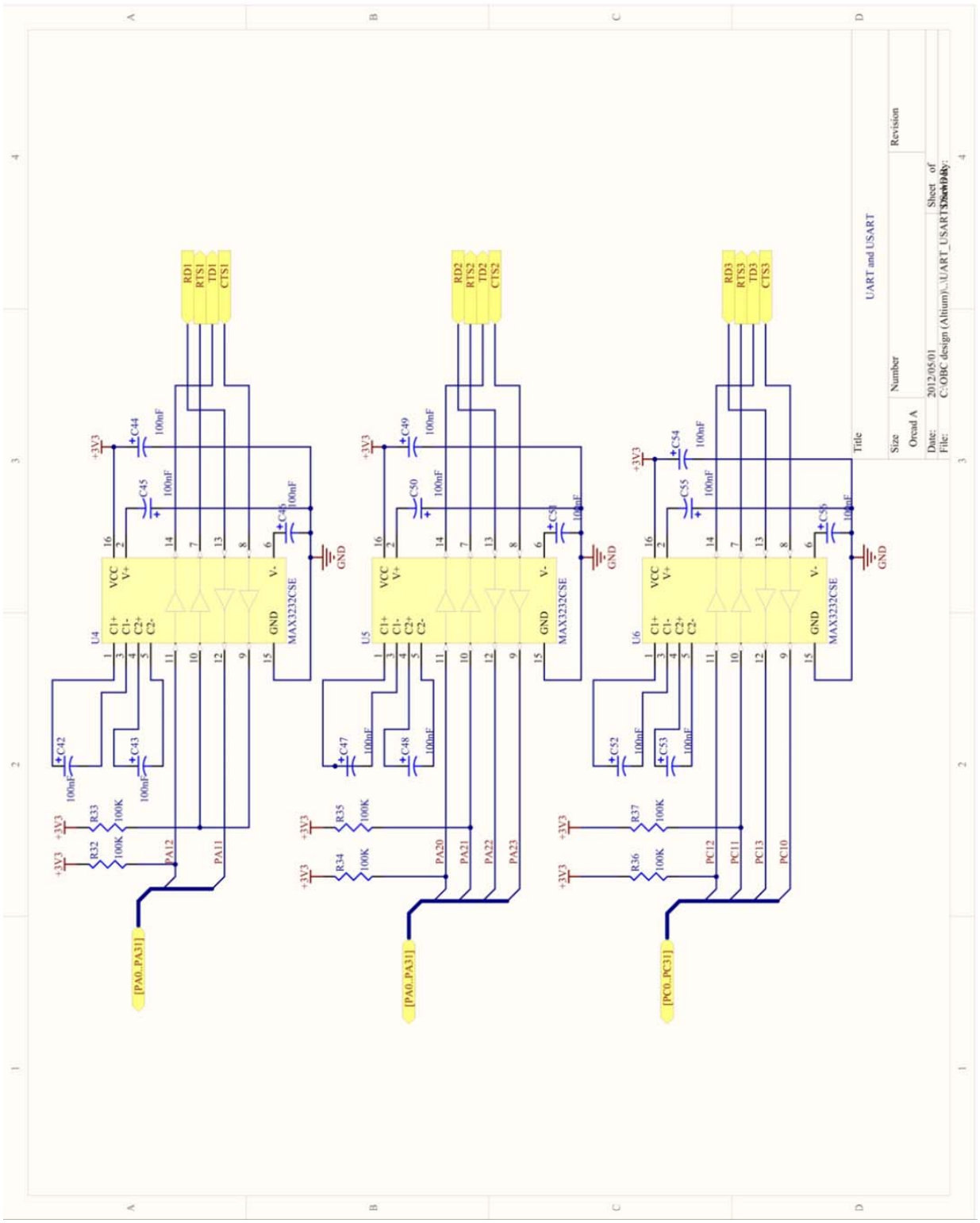
Figure D.6: Bottom view picture of the OBC

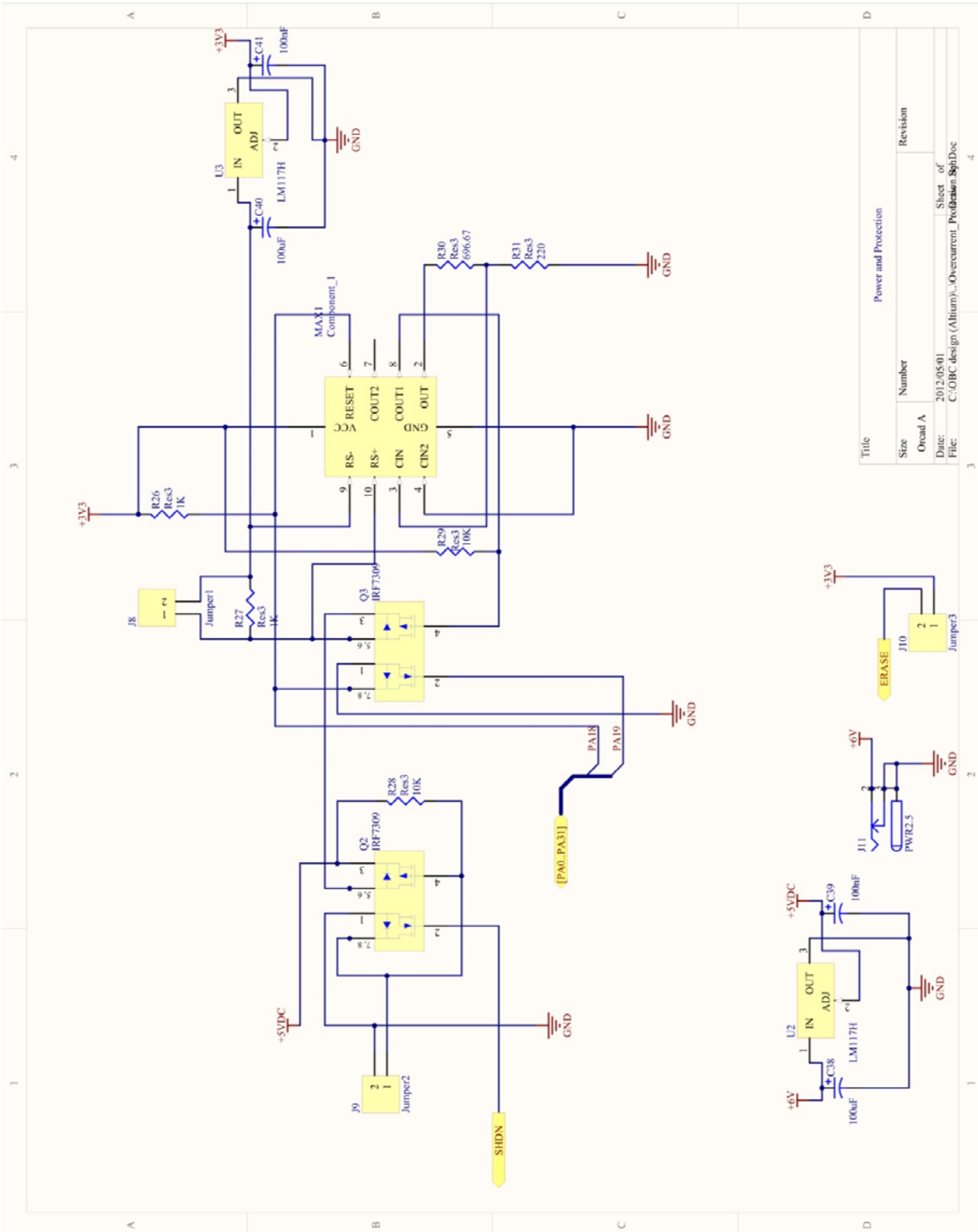
Appendix E: Circuits Schematics

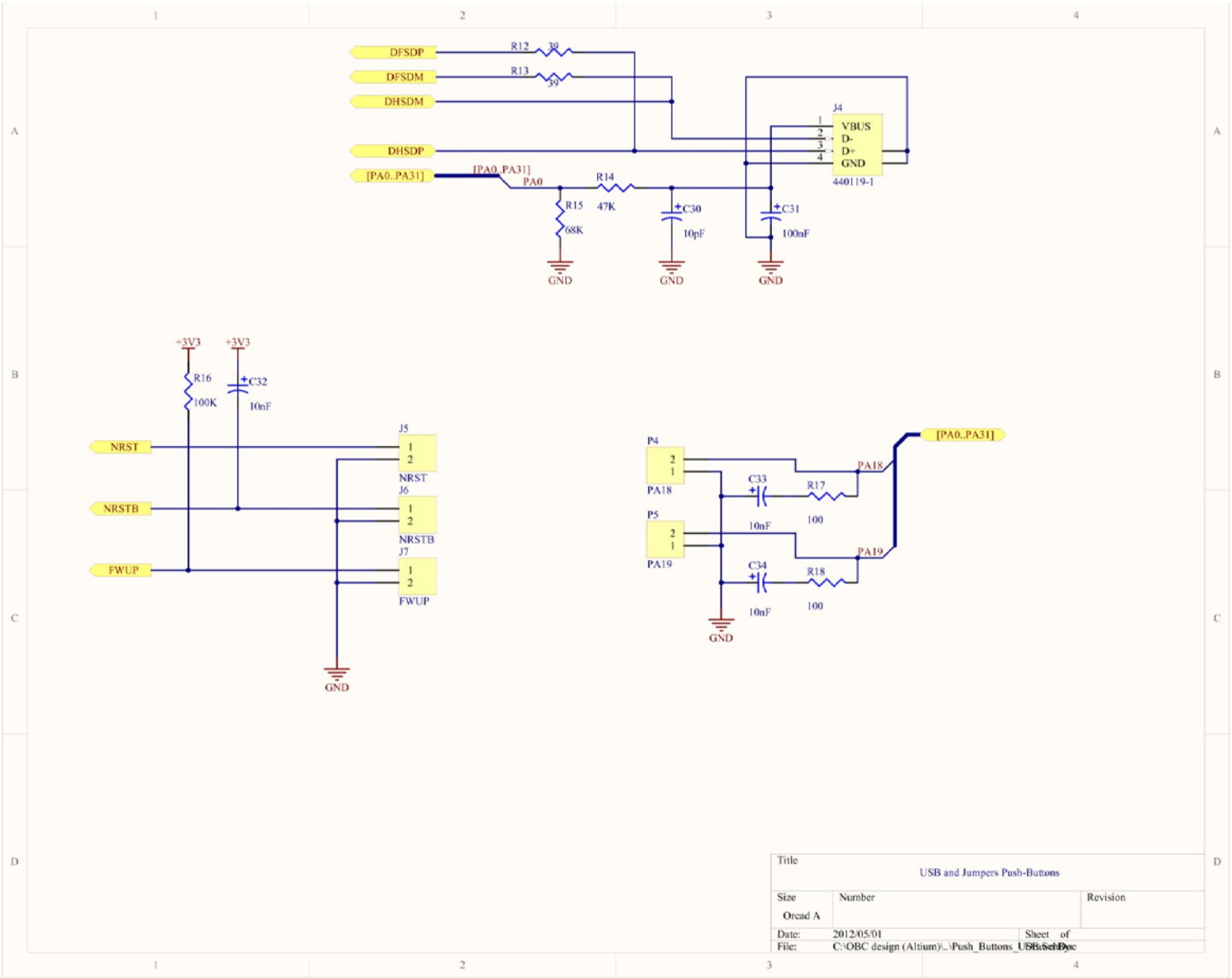


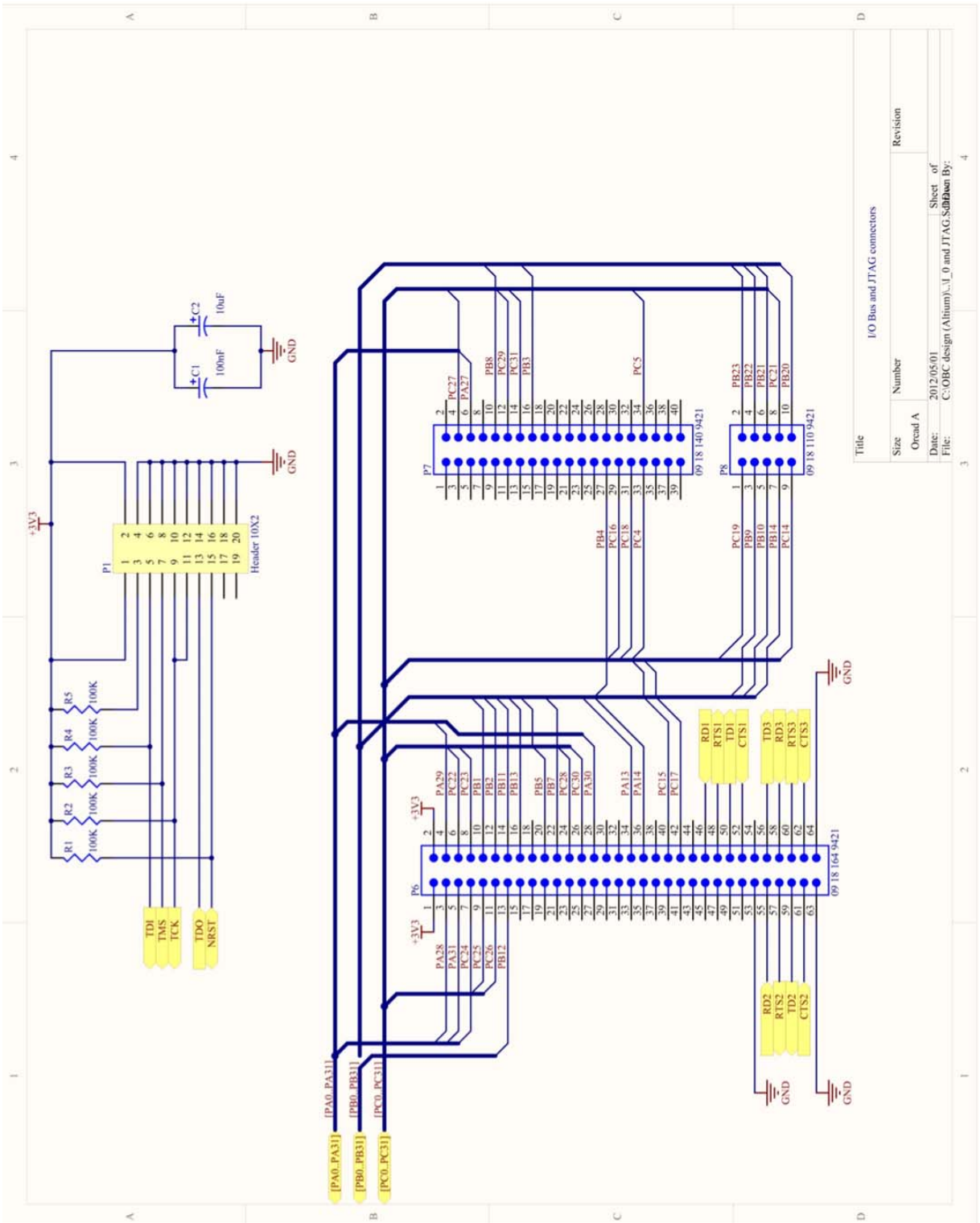

Backup battery, X-tals and Filtering Capacitors
Size
Orcad A
Number
Revision
Date: 2012/05/01
Sheet of
File: C:\OBC design (Altium)\...\FilterCaps_LED.SchDoc
VOUT
VDDPLL
VCORE
J1
J-VCORE
+3V3
VIO
VIN
J2
J-VIN
XIN
XOUT
XIN32
XOUT32
Y1
Y2
XTAL
Inductor+3V3
Inductor
VDDANA
VDDUTMI
+5VDC
ADVREF
AD12BVREF
P2
P3
MHDR1X3
D1
D2
D3
LED3
R6
4.7K
R7
220
R8
100K
R10
Q1
MOSFET-N
PB0
PB2
[PB0..PB31]
Batt
Component_1
BT1
3V
VDBU
GND
D4
BAT54C
J3
J-VDDBU
VDDBU


SD_Card1
CD/DAT3
CMD
VSS
VDD
CLK
VSS
DAT0
DAT1
DAT2
CD
CMD
SD_WP
R25 10K
R24 47K
R23 47K
R22 47K
R21 47K
R20 10K
R19 10K
+3V3
GND
C36 100nF
C35 10uF
PA8
PA4
PA3
PA5
PA6
PA7
PA25
[PA0..PA31]
MCP1
VDD
GND
SDA
ALERT
SCLK
C37 100nF
R38 47K
PA17
PA10
PA9
Title
SD Card and Temperature sensor
Size
Number
Revision
A4
Date: 2012/05/01
Sheet of
File: C:\OBC design (Altium)\...\SD_Card and Temp.SchDoc


RD1
RTS1
TD1
CTS1
RD2
RTS2
TD2
CTS2
RD3
RTS3
TD3
CTS3
U4
U5
U6
MAX3232CSE
VCC
V+
C1+
C1-
C2+
C2-
V-
GND
+3V3
C42
C43
C44
C45
C46
C47
C48
C49
C50
C51
C52
C53
C54
C55
C56
100nF
R32
R33
R34
R35
R36
R37
100K
PA12
PA11
PA20
PA21
PA22
PA23
PC12
PC11
PC13
PC10
[PA0..PA31]
[PA0..PA31]
[PC0..PC31]
Title
UART and USART
Size
A4
Number
Revision
Date:
2012/05/01
Sheet of
File:
C:\OBC design (Altium)\...\UART_USART.SchDoc
Drawn By:


Title
Power and Protection
Size
Orcad A
Number
Revision
Date:
2012/05/01
File:
U3
LM117H
IN
OUT
ADJ
C40
100uF
C41
100nF
+3V3
GND
MAX1
Component_1
RESET
COUT2
COUT1
OUT
VCC
GND
RS-
RS+
CIN
CIN2
R30
Res3
696.67
R31
Res3
220
R26
Res3
1K
R27
Res3
1K
R29
Res3
10K
R28
Res3
10K
J8
Jumper1
Q3
IRF7309
Q2
IRF7309
PA18
PA19
[PA0..PA31]
J9
Jumper2
SHDN
+5VDC
J10
Jumper3
ERASE
+6V
J11
PWR2.5
U2
LM117H
C38
100uF
C39
100nF


DFSDP
DFSDM
DHSDM
DHSDP
[PA0..PA31]
R12
39
R13
39
J4
VBUS
D-
D+
GND
440119-1
[PA0..PA31]
PA0
R14
47K
R15
68K
C30
10pF
C31
100nF
GND
GND
GND
+3V3
+3V3
R16
100K
C32
10nF
NRST
NRSTB
FWUP
J5
NRST
J6
NRSTB
J7
FWUP
GND
P4
PA18
P5
PA19
C33
10nF
R17
100
C34
10nF
R18
100
PA18
PA19
[PA0..PA31]
GND
Title
USB and Jumpers Push-Buttons
Size
Orcad A
Number
Revision
Date:
2012/05/01
Sheet of
File:
C:\OBC design (Altium)\..\Push_Buttons_USB.SchDoc


I/O Bus and JTAG connectors
Header 10X2
P1
P6
P7
P8
09 18 164 9421
09 18 140 9421
09 18 110 9421
+3V3
GND
C1
C2
100nF
10uF
R1
R2
R3
R4
R5
100K
TDI
TMS
TCK
TDO
NRST
[PA0..PA31]
[PB0..PB31]
[PC0..PC31]
RD1
RTS1
TD1
CTS1
TD3
RD3
RTS3
CTS3
RD2
RTS2
TD2
CTS2
Title
Number
Revision
Size
A3
Date:
2012-05-01
File:
C:\OBC design (Altium)\...\I_O and JTAG.SchDoc
Sheet of
Drawn By:

Appendix F: Test Programs source codes

1. Blink program and UART communication

```
//CPUT 2011
//Tony Lumbwe
//204131316
//
/*


       Blink and UART communication program V1.0 (Source file)


*/
//------------------------------------------------------------------------------
// IAR EWARM Toolchain
//------------------------------------------------------------------------------
//         Headers
//------------------------------------------------------------------------------

#include <board.h>
#include <pio/pio.h>
#include <pio/pio_it.h>
#if defined(AT91C_BASE_PITC)
#include <pit/pit.h>
#endif
#include <irq/irq.h>
#include <tc/tc.h>
#include <utility/led.h>
#include <utility/trace.h>
#include <stdio.h>
#if defined(cortexm3)
#include <systick/systick.h>
#endif


//------------------------------------------------------------------------------
//         Local variables
//------------------------------------------------------------------------------


/// Indicates the current state (on or off) for each LED.
volatile unsigned char pLedStates[2] = {1, 1};

/// Global timestamp in milliseconds since start of application.
volatile unsigned int timestamp = 0;

//------------------------------------------------------------------------------
//         Local functions
//------------------------------------------------------------------------------
//------------------------------------------------------------------------------
/// Configures LEDs \#1 and \#2 (cleared by default).
//------------------------------------------------------------------------------
void ConfigureLeds(void)
{
   LED_Configure(0);
   LED_Configure(1);
}

//------------------------------------------------------------------------------
/// Interrupt handler for TC0 interrupt. Toggles the state of LED\#2.
//------------------------------------------------------------------------------
void TC0_IrqHandler(void)
{
    volatile unsigned int dummy;
    // Clear status bit to acknowledge interrupt
    dummy = AT91C_BASE_TC0->TC_SR;

    // Toggle LED state
    LED_Toggle(1);
```

```
    LED_Toggle(0);
    printf("2");
}

//------------------------------------------------------------------------------
/// Configure Timer Counter 0 to generate an interrupt every 250ms.
//------------------------------------------------------------------------------
void ConfigureTc(void)
{
    unsigned int div;
    unsigned int tcclks;

    // Enable peripheral clock
    AT91C_BASE_PMC->PMC_PCER = 1 << AT91C_ID_TC0;

    // Configure TC for a 4Hz frequency and trigger on RC compare

    TC_FindMckDivisor(8, BOARD_MCK, &div, &tcclks);
    TC_Configure(AT91C_BASE_TC0, tcclks | AT91C_TC_CPCTRG);
    AT91C_BASE_TC0->TC_RC = (BOARD_MCK / div) / 1; // timerFreq / desiredFreq

    // Configure and enable interrupt on RC compare
    IRQ_ConfigureIT(AT91C_ID_TC0, 0, TC0_IrqHandler);
    AT91C_BASE_TC0->TC_IER = AT91C_TC_CPCS;
    IRQ_EnableIT(AT91C_ID_TC0);

    // Start the counter if LED is enabled.
    if (pLedStates[0])
    {

        TC_Start(AT91C_BASE_TC0);
    }
}

//------------------------------------------------------------------------------
//         Exported functions
//------------------------------------------------------------------------------

//------------------------------------------------------------------------------
/// Application entry point. Configures the DBGU, PIT for SAM7 & SAM9
/// microcontrollers, UART and System tick for SAM3 microcontrollers.
/// Configures TC0 and LED to loop infinitely.
//------------------------------------------------------------------------------
int main(void)
{
    // UART configuration
    TRACE_CONFIGURE(DBGU_STANDARD, 115200, BOARD_MCK);
    printf("-- Getting Started Project %s --\n\r", SOFTPACK_VERSION);
    printf("-- %s\n\r", BOARD_NAME);
    printf("-- Compiled: %s %s --\n\r", __DATE__, __TIME__);


    ConfigureTc();
    ConfigureLeds();

    // Main loop
    while (1)
    {

    }
}
```

2. 10-bit and 12-bit ADC

```
//CPUT 2011
//Tony Lumbwe
//204131316
//
/*
```

```
       10-bit and 12-bit ADC program V1.0 (Source file)


*/
//------------------------------------------------------------------------------
// IAR EWARM Toolchain
//------------------------------------------------------------------------------
//         Headers
//------------------------------------------------------------------------------

//Headers
//-------

#include <board.h>
#include <pio/pio.h>
#include <dbgu/dbgu.h>
#include <irq/irq.h>
#include <utility/led.h>
#include<utility/trace.h>
#include <adc/adc12.h>
#include <adc/adc.h>
#include <stdio.h>

//Local definitions
//-----------------
#define esc 27
#define cls printf("%c[2J",esc)
#define pos(row,col) printf("%c[%d;%dH",esc,row,col)

#define BOARD_ADC_FREQ 6000000       //5MHz ADC Frequency
#define ADC_VREF       3300    //3.3*1000

typedef struct _GainMap

{
  unsigned int diffmode;
    char *gainstring[4];
}GainMap;

//Local variables
//-----------------


//PIOs to configure.

#ifdef PINS_ADC
  static const Pin pinsADC[] = {PINS_ADC};   //implement all ADC pins in a array representing
each channel
#endif


 //Remap SAM3U4 adc 10 bit to be compatible with definition name of others

#undef AT91C_ID_ADC                   //undefine 10-bit ADC controller
#define AT91C_ID_ADC AT91C_ID_ADC12B  //use ID name of 12-bit controller for 10-bit controller
#define AT91C_BASE_ADC AT91C_BASE_ADC12B     //Use same base address for both ADC10 and ADC12-bit

//indication that the conversion is finished.

static volatile unsigned char conversiondone;


#define ADC1   ADC12_CHANNEL_2
#define ADC2   ADC12_CHANNEL_3
#define ADC3   ADC12_CHANNEL_6
#define ADC4   ADC12_CHANNEL_7


static unsigned int channels[] = {ADC1, ADC2, ADC3, ADC4};
```

```
//Local functions
//-----------------
//Converting a digital value in milivolts
//*************************************

static unsigned int convertHex2mv( unsigned int valuetoconvert )
{
  unsigned int mask;
#if defined(at91sam3u4)
  mask = 0xFFF;
#else
  mask = 0xFF;
#endif
  return ((ADC_VREF * valuetoconvert)/mask);

}



//Interupt handler for ADC. Signals that the conversion has been done by setting
//a flag variable.
//******************************************************************************
void ADCC0_IrqHandler (void)
{
  unsigned int status, i=1;

    status = ADC12_GetStatus(AT91C_BASE_ADC);

    TRACE_DEBUG("status =0x%X\n\r", status);

           if (ADC12_IsChannelInterruptStatusSet(status, channels[i]))

        {

            TRACE_DEBUG("channel %d\n\r", channels[i]);

            ADC12_DisableIt(AT91C_BASE_ADC, 1<<channels[i]);//disable EOCx interrupt

            conversiondone |= 1<<channels[i];
        }

}


//Configuration Menu for ADC
//**************************

static void ShowADC12ConfigMenu()
{
   unsigned int adc_acr;
   unsigned int gain, currentcontrol, diffmode, offset;

   GainMap gainmap [2]={{0,"1","1","2","4"},{1,"0.5","1","2","2"}};

   char *inputmode [2]= {"Single Ended","Full Differential"};

   //GainMap

   adc_acr = ADC12_GetAnalogCtrlReg(AT91C_BASE_ADC);
   gain = adc_acr & 0x3;
   currentcontrol = (adc_acr & 0x3)>>6 ;
   diffmode = (adc_acr & 0x10000) >>16;
   offset = (adc_acr & 0x20000) >>17;

printf("\n\r***********ADC 12 bit configuration Menu**************");
printf("\n\r g--Configure 12 bit ADC gain");
printf("\n\r c--Configure 12 bit ADC bias current");
printf("\n\r d--Configure 12 bit ADC Differential mode");
printf("\n\r o--Configure 12 bit ADC Offset");
printf("\n\r m--Show this menu again!\n\r");
```

```
  printf("\n\r\r Current settings:\n\r [GAIN:%u, gain = %s], [IBCTL: %u], [DIFF: %u, %s],
[OFFSET: %u]\n\r", \
      gain, gainmap[diffmode].gainstring[gain], currentcontrol, diffmode, inputmode[diffmode],
offset);
      printf("\n\rPlease enter a key to configure the ADC or any other key \r\n \
      to perform a measurement with current settings.");

}


//Global functions
//-----------------


//*****************************************************************
//Performing measurement on ADC channel 0 and display results on UART
//*****************************************************************

int main (void)

{
  unsigned int id_channel, advalue;
  char key;

  TRACE_CONFIGURE(DBGU_STANDARD, 115200, BOARD_MCK); //Initializing the standard UART
                                                     //port with 11520 baudrate and using the
board's masterclock frequency of 48MHz


#ifdef PINS_ADC
       PIO_Configure(pinsADC, PIO_LISTSIZE(pinsADC));      //Configuring the list of pin
instances defined by pinsADC and of size determined by PIO_LISTSIZE)
#endif



//Initialize the ADC with software trigger mode, sleep mode, 12 bits resolution,
//48MHz frequency(MCK), ADC frequency, 10us startup time and 1.2us sample and hold time.
ADC12_Initialize( AT91C_BASE_ADC,
                  AT91C_ID_ADC, AT91C_ADC12B_MR_TRGEN_DIS,
                 0,AT91C_ADC12B_MR_SLEEP_NORMAL,
                 AT91C_ADC12B_MR_LOWRES_12_BIT,
                 BOARD_MCK,BOARD_ADC_FREQ, 10, 1200);

//Enable ADC channels to be used

ADC12_EnableChannel(AT91C_BASE_ADC, ADC1);
ADC12_EnableChannel(AT91C_BASE_ADC, ADC2);
ADC12_EnableChannel(AT91C_BASE_ADC, ADC3);
ADC12_EnableChannel(AT91C_BASE_ADC, ADC4);

//Configure NVIC interrupt for ADC
IRQ_ConfigureIT(AT91C_ID_ADC, 0, ADCC0_IrqHandler);
//Enable given interrupt source address
IRQ_EnableIT(AT91C_ID_ADC);


//Show ADC Menu

ShowADC12ConfigMenu();

//Infinite loop

 while (1)
 {

   //wait for user input
   key = DBGU_GetChar();

   //set Gain
   if(key == 'g' || key =='G')
```

```
{
  cls;
    printf("\n\r************Select Gain option to set GAIN<0:1>**************");
    printf("\n\r 1-- 00b [Single Ended Mode: gain =1, Full Differential Mode: gain =0.5]");
    printf("\n\r 2-- 01b [Single Ended Mode: gain =1, Full Differential Mode: gain =1]");
    printf("\n\r 3-- 10b [Single Ended Mode: gain =2, Full Differential Mode: gain =2]");
    printf("\n\r 4-- 11b [Single Ended Mode: gain =4, Full Differential Mode: gain =2]");


    key == DBGU_GetChar();

    unsigned int gain = 0xff;
    switch (key)

         {
             case '1':
             gain = 0;
             break;

             case '2':
             gain = 0x1;
             break;

             case '3':
             gain = 0x2;
             break;

             case '4':
             gain = 0x3;
             break;

             default:
             printf("\n\r Wrong selection!\n\r");
             break;

          }

      if (gain!=0xff)
          {
             //change Gain field of ADC_ACR of ADC 12 bit
             gain |= (((unsigned int)ADC12_GetAnalogCtrlReg(AT91C_BASE_ADC))& 0x300c0);
             ADC12_CfgAnalogCtrlReg(AT91C_BASE_ADC, gain);

           }

     ShowADC12ConfigMenu();
     continue;


}
 //Show menu

 if (key == 'm' || key == 'M')
     {
       ShowADC12ConfigMenu();
       continue;

     }

  //set offset

if (key == 'o'|| key == 'O')
{
   printf("\n\r************Select input Offset option to set Offset bit field**************");
   printf("\n\r 1-- 0 [DIFF:0, Vrefin/4][DIFF:1, Vrefin/2]");
   printf("\n\r 2-- 1 [Vrefin/2]");
   printf("\n\r ");


 key = DBGU_GetChar();
```

```
    unsigned int offset = 0xff;
    switch(key)
      {
      case '1':
        offset = 0;
        break;

      case '2':
        offset = 0x20000;
        break;

      default :
        printf("\n\r Wrong selection!\n\r");
        break;

      }
    if (offset !=0xff)
      {
        //Change GAIN field of ADC_ACR of ADC 12bit
        offset |= (((unsigned int) ADC12_GetAnalogCtrlReg(AT91C_BASE_ADC)) &0x100c3);
        ADC12_CfgAnalogCtrlReg(AT91C_BASE_ADC, offset);

      }

      ShowADC12ConfigMenu();
      continue;

   }

 conversiondone = 0;

 ADC12_EnableIt(AT91C_BASE_ADC, 1<<ADC2);

 //Start measurement
 while (1)
 {

ADC12_StartConversion(AT91C_BASE_ADC);         //Function to start ADC conversion process
 while (conversiondone != ((1<<ADC2)));
 id_channel = 1;

printf("%c[2J",27);
advalue = ADC12_GetConvertedData(AT91C_BASE_ADC, channels [id_channel]);    //Returns the
converted channel's data and assign its value to advalue

printf("\n\rChannel %u : %u mV", id_channel, convertHex2mv(advalue));

 }
 }
}
```

3. PWM

```
//CPUT 2011
//Tony Lumbwe
//204131316
//
/*


       PWM program V1.0 (Source file)


*/
//--------------------------------------------------------------------------
```

```
// IAR EWARM Toolchain
//------------------------------------------------------------------------------
//------------------------------------------------------------------------------
//         Headers
//------------------------------------------------------------------------------

#include <board.h>
#include <pio/pio.h>
#include <irq/irq.h>
#include <dbgu/dbgu.h>
#include <pwmc/pwmc2.h>
#include <utility/trace.h>

#include <stdio.h>

//------------------------------------------------------------------------------
//         Local definitions
//------------------------------------------------------------------------------

/// PWM frequency in Hz.
#define PWM_FREQUENCY               1000

/// Maximum duty cycle value.
#define MAX_DUTY_CYCLE              90
#define MIN_DUTY_CYCLE              0

/// Duty cycle buffer length for three channels
#define DUTY_BUFFER_LENGTH          (MAX_DUTY_CYCLE - MIN_DUTY_CYCLE)

//------------------------------------------------------------------------------
//         Local variables
//------------------------------------------------------------------------------

/// Pio pins to configure.
static const Pin pins[] = {
    PINS_DBGU,
    PIN_PWM_LED0,
    PIN_PWM_LED1,
    PIN_PWM_LED2,
    PIN_PWMC_PWML0,
    PIN_PWMC_PWML1,
    PIN_PWMC_PWML2
};

/// duty cycle buffer for PDC transfer
unsigned short dutyBuffer[DUTY_BUFFER_LENGTH];

//------------------------------------------------------------------------------
//         Local functions
//------------------------------------------------------------------------------

//------------------------------------------------------------------------------
/// Interrupt handler for the PWM controller.
//------------------------------------------------------------------------------
void PWM_IrqHandler(void)
{
    unsigned int isr2 = AT91C_BASE_PWMC->PWMC_ISR2;

    if ((isr2 & AT91C_PWMC_ENDTX) == AT91C_PWMC_ENDTX)

    {
      PWMC_WriteBuffer(AT91C_BASE_PWMC, dutyBuffer, DUTY_BUFFER_LENGTH);
    }
}

//------------------------------------------------------------------------------
//         Global functions
//------------------------------------------------------------------------------

//------------------------------------------------------------------------------
/// Outputs a PWM on LED1to makes it fade in repeatedly.
```

```
/// Channel #0 is configured to have a source clock, a period and an alignment and
/// The update of the duty cycle values is made
/// automatically by the Peripheral DMA Controller (PDC).
//------------------------------------------------------------------------------
int main(void)
{
    unsigned int i;

    PIO_Configure(pins, PIO_LISTSIZE(pins));

// Enable PWMC peripheral clock
    AT91C_BASE_PMC->PMC_PCER = 1 << AT91C_ID_PWMC;

// Set clock A to run at PWM_FREQUENCY * MAX_DUTY_CYCLE (clock B is not used)
    PWMC_ConfigureClocks(PWM_FREQUENCY * MAX_DUTY_CYCLE, 0, BOARD_MCK); // Configures PWM clocks
A to run at the given frequencies. This function
                                                                        // finds the best MCK
divisor and prescaler values automatically.

// Configure PWMC channel for LED0 (left-aligned, enable dead time generator)
    PWMC_ConfigureChannelExt(CHANNEL_PWM_LED0, AT91C_PWMC_CPRE_MCKA, 0, 0, 0, AT91C_PWMC_DTE, 0,
0);
    PWMC_SetPeriod(CHANNEL_PWM_LED0, MAX_DUTY_CYCLE);
    PWMC_SetDutyCycle(CHANNEL_PWM_LED0, MIN_DUTY_CYCLE);
    PWMC_SetDeadTime(CHANNEL_PWM_LED0, 5, 5);

// Set channel #0, #1 and #2 as synchronous channels, update mode = 2
    PWMC_ConfigureSyncChannel(AT91C_PWMC_SYNC0 , AT91C_PWMC_UPDM_MODE2, 0, 0); //
|AT91C_PWMC_SYNC1| AT91C_PWMC_SYNC2

// Set Synchronous channel update period value
    PWMC_SetSyncChannelUpdatePeriod(AT91C_PWMC_UPVUPDAL);

// Configure interrupt for PDC transfer
    IRQ_ConfigureIT(AT91C_ID_PWMC, 0, PWM_IrqHandler);
    IRQ_EnableIT(AT91C_ID_PWMC);
    PWMC_EnableIt(0, AT91C_PWMC_ENDTX);

// Enable syncronous channels by enable channel #0
    PWMC_EnableChannel(CHANNEL_PWM_LED0);

// Fill duty cycle buffer for channel #1, duty cycle from MIN_DUTY_CYCLE to MAX_DUTY_CYCLE
    for (i = 0; i < DUTY_BUFFER_LENGTH/3; i++)

    {
        dutyBuffer[i] = (i + MIN_DUTY_CYCLE);
    }

    // Start PDC transfer
    PWMC_WriteBuffer(AT91C_BASE_PWMC, dutyBuffer, DUTY_BUFFER_LENGTH);

///-----------------------------------------------------------------
///Infinite loop
///-----------------------------------------------------------------

while (1)
    {

    }

}
```

4.TWI

```
//CPUT 2011
//Tony Lumbwe
//204131316
//
/*


       TWI temp sensor program V1.0 (Source file)


*/
//------------------------------------------------------------------------------
// IAR EWARM Toolchain
//------------------------------------------------------------------------------
//         Headers
//------------------------------------------------------------------------------

#include <board.h>     //defines interfaces and PIOs characteristics (-frequency,
///                                -portable PIOs definitions etc...)
#include <pio/pio.h>              //Provides API for PIO configuration and usage of user
///                               controlled pins.
#include <irq/irq.h>              //Configure an interrupt in the NVIC
#include <dbgu/dbgu.h>             // This module provides definitions and functions for using
the Debug Unit
//                                 (DBGU).
#include <twi/twi.h>               // Interface for configuration the Two Wire Interface (TWI)
peripheral.
#include <utility/math.h>          //maths functions exported

#include <utility/assert.h>

#include <utility/trace.h>         // Standard output methods for reporting debug information,
warnings and
                                   // errors, which can be easily be turned on/off.
#include <drivers/async/async.h>    // Asynchronous transfer descriptor.
#include <drivers/twi/twid.h>       //

#include <stdio.h>                    //Standard Input and Output header
#include <string.h>                   //string.h is the header in the C standard library for the
C programming
//                                      language which contains macro definitions, constants, and
declarations
//                                      of functions and types used not only for string handling
but also various
//                                      memory handling functions; the name is thus something of
a misnomer.

//------------------------------------------------------------------------------
//         Local definitions
//------------------------------------------------------------------------------

/// TWI clock frequency in Hz.

#define TWCK            100000
///TWI0 ID
#define TWI_TEMP_ID    AT91C_ID_TWI0
#define INVALID_TEMP (-110)

/// Slave address of Temperature Sensor.
#define MCP9800_ADDRESS   0x48
/// Internal register within MCP9800
//00 = Temperature register (TA)
//01 = Configuration register (CONFIG)
//10 = Temperature Hysteresis register (THYST)
//11 = Temperature Limit-set register (TSET)

#define TEMP_REG       0x0
#define CONF_REG       0x1
#define HYST_REG       0x2
```

```
#define LIMT_REG       0x3


typedef struct
{
  unsigned char bOneShot;
  unsigned char bRes;
  unsigned char bFaultQueue;
  unsigned char bAlrtPol;
  unsigned char bMode;
  unsigned char bShutdown;
}Confg_Reg;



//-----------------------------------------------------------------------------
//         Local variables
//-----------------------------------------------------------------------------

/// Pio pins to configure.
static const Pin pins_twi_temp[] = {PINS_TWI0};
/// TWI driver instance.
static Twid twid;          //TWI driver structure. Holds the internal state of the driver
/// constant represent float value
const char* sixteenths[]={
"0","0625","125","1875","25","3125","375","4375","5","5625","625","6875","75","8125","875","9375"
};

//*****************************************************************************
//         Local functions
//*****************************************************************************

//-----------------------------------------------------------------------------
/// Get the real temperature code regardless of different adc resolutions
//-----------------------------------------------------------------------------

short GetRealTemp(unsigned short reg_value,unsigned char res)
{
  if( 8 < res && 13 > res)  //since the sensor only works within limits of 9 to 12-bit,
resolution
//                            needs to be set in between these values.
  {
    return (reg_value>>(16-res)); //Shift right the reg_value to return the real temperature code

  }
  return INVALID_TEMP;
}
//-----------------------------------------------------------------------------
///  Fill the register value into specific structure
//MCP9800 configuration register:8bits
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//bit 7 ONE-SHOT bits
//bit 5-6 ¦²¦¤ ADC RESOLUTION bit
//bit 3-4 FAULT QUEUE bit
//bit 2 ALERT POLARITY bit
//bit 1 COMP/INT bit
//bit 0 SHUTDOWN bit
//
//-----------------------------------------------------------------------------

void FillConfigReg(unsigned char reg_value,Confg_Reg *conf)
{
  unsigned char res_array[]={9,10,11,12};
  unsigned char fault_queue_array[]={1,2,4,6};
  unsigned char temp;
  temp = ((reg_value>>7)&0x1);
  conf->bOneShot = temp;

  temp = ((reg_value>>5)&0x3);
  conf->bRes = res_array[temp];
```

```
  temp = ((reg_value>>3)&0x3);
  conf->bFaultQueue = fault_queue_array[temp];

  temp = ((reg_value>>2)&0x1);
  conf->bAlrtPol = temp;

  temp = ((reg_value>>1)&0x1);
  conf->bMode = temp;
  conf->bShutdown = (reg_value&0x1);

}

//------------------------------------------------------------------------------
/// Main function
//------------------------------------------------------------------------------
int main()
{
    unsigned char Tdata[2];
    short temp;
    Confg_Reg confg;

    PIO_Configure(pins_twi_temp, PIO_LISTSIZE(pins_twi_temp));
    TRACE_CONFIGURE(DBGU_STANDARD, 115200, BOARD_MCK);
    printf("%c[2J",27);
    printf("\n\r****************TWI temperature sensor***************");
    printf("\n\n\n");
    printf("\r Ambient temperature is:\n ");

    // Configure TWI
    AT91C_BASE_PMC->PMC_PCER = 1 << TWI_TEMP_ID;//AT91C_ID_TWI0;
    TWI_ConfigureMaster(AT91C_BASE_TWI0, TWCK, BOARD_MCK);
    TWID_Initialize(&twid, AT91C_BASE_TWI0); // Initializes a TWI driver instance, using the
given TWI peripheral. The
                                             // peripheral must have been initialized properly
before calling this function.
                                             /// \param pTwid  Pointer to the Twid instance to
initialize.
                                             /// \param pTwi  Pointer to the TWI peripheral to
use.
    while(1)
    {


        char index;
        short integer;
        TWID_Read(&twid, MCP9800_ADDRESS, CONF_REG, 0x01, Tdata, 0x01, 0);
        FillConfigReg(Tdata[0],&confg);
        TWID_Read(&twid, MCP9800_ADDRESS, TEMP_REG, 0x01, Tdata, 0x02, 0);
        temp = ((Tdata[0]<<8)|(Tdata[1]));
        temp = GetRealTemp(temp,confg.bRes);
        index = temp%(2<<(confg.bRes-8-1));
        index =  index <<(4-(confg.bRes-8));
        integer =   temp/(2<<(confg.bRes-8-1));
        printf("\r\t\t\t%d.%s C\r",integer,sixteenths[index]);

    }
}
```