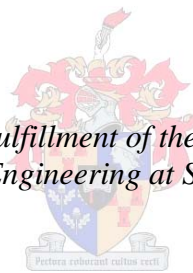


# **Development of an Integrated Avionics Hardware System for Unmanned Aerial Vehicle Research Purposes**

by  
Robin van Wyk

*Thesis presented in partial fulfillment of the requirements for the degree  
Master of Science in Engineering at Stellenbosch University*



Supervisor: Dr. Iain K. Peddle  
Department of Electrical & Electronic Engineering

March 2011

# Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: March 2011

Copyright © 2011 Stellenbosch University

All rights reserved

# Abstract

The development of an integrated avionics system containing all the required sensors and actuators for autopilot control is presented. The thesis analyzes the requirements for the system and presents detailed hardware design. The architecture of the system is based on an FPGA which is tasked with interfacing with the sensors and actuators. The FPGA abstracts a microprocessor from these interface modules, allowing it to focus only on the control and user interface algorithms. Firmware design for the FPGA, as well as a conceptualization of the microprocessor software design is presented. Simulation results showing the functionality of firmware modules are presented.

# Uittreksel

Die ontwikkeling van 'n geïntegreerde avionika-stelsel wat al die vereiste sensors en aktueerders vir outoloods-beheer bevat, word voorgestel. Die tesis analiseer die vereistes van die stelsel en stel 'n hardware-ontwerp voor. Die argitektuur van die stelsel bevat 'n FPGA wat 'n koppelvlak met sensors en aktueerders skep. Die FPGA verwyder die mikroverwerker weg van hierdie koppelvlak modules en stel dit sodoende in staat om slegs op die beheer en gebruikerskoppelvlak-algoritmes te fokus. Sagteware-ontwerp vir die FPGA, asook die konseptualisering van die sagteware-ontwerp vir die mikroverwerker, word aangebied. Simulasie resultate wat die funksionaliteit van die FPGA-sagteware modules aandui, word ook voorgestel.

# Acknowledgements

I would like to extend my gratitude and appreciation to everybody who assisted, supported and encouraged me. In particular, I would like to thank the following people:

- Dr. Iain Peddle for your assistance, guidance, patience and constant support for the entire duration of this project. The encouragement you provided will always be appreciated.
- Armscor and the National Research Foundation for funding this project.
- My colleagues and friends in the Electronic Systems Laboratory for their support and technical assistance.
- To all who provided technical assistance on this project, especially Mr. Arno Barnard.
- Mr Johan Arendse and Mr Quintis Brandt for their contributions to the sourcing of the components and building of the hardware in the project.
- My flat mates, Grant Leukes and Günther Kassier, for their assistance, sanity checks and great friendship.
- My friends and those special to me who supported and assisted me through their kind words. Each one played a significant role.
- My parents, Chris and Sherine Van Wyk and my brother, Adrian and sister Liesl, especially for the final encouragement and support on the last stretch. The great love and encouragement I experienced will always remain with me.
- I express my sincere thanks to my Heavenly Father for the inspiration and the strength received. Without His divine help, this thesis would not have been possible.

Finally, to all those with words of encouragement and support, a great thank you to you as all these moments helped me reach my final goal.

# Contents

1. INTRODUCTION .....	1
1.1 Background.....	1
1.2 Commercial Autopilot & Avionics Systems .....	1
1.3 Avionics Systems in the Electronic Systems Laboratory (ESL) .....	2
1.4 User Requirements Specification .....	6
1.5 Thesis Outline .....	7
2. SYSTEM REQUIREMENTS SPECIFICATION .....	9
2.1 Sensors in the Avionics System .....	9
2.1.1 Pressure sensors .....	9
2.1.2 Inertial measurement sensors .....	11
2.1.3 Magnetometer .....	12
2.1.4 GPS sensor module .....	12
2.1.5 Secondary sensors .....	13
2.2 The Microprocessor.....	13
2.3 Aircraft Interface .....	15
2.4 Interfaces.....	15
2.5 Further Considerations.....	16
2.6 Traceability of System Requirements .....	17
3. CONCEPTUAL DESIGN .....	18
3.1 Conceptualization based on user requirements specification analysis .....	18
3.2 High level system overview .....	21
3.3 Conceptualization based on system requirements specification analysis.....	22
4. DETAILED HARDWARE DESIGN .....	26
4.1 Component choices and considerations .....	26
4.1.1 Microprocessor .....	26
4.1.2 Sensors .....	30
4.1.2.1. GPS.....	30
4.1.2.2. Inertial Measurement Unit (IMU).....	33
4.1.2.3. Pressure Sensors.....	37
4.1.2.4. Secondary Sensors: Temperature and Current .....	38
4.1.3 SD-Card Interface .....	39
4.1.4 USB and CAN Interfaces .....	39
4.1.5 FPGA.....	40
4.1.6 Prototype Development Overview .....	41

4.2 Schematic Design of Development Prototype .....	41
4.2.1 Power Distribution and Considerations.....	43
4.2.2 IMU Considerations .....	47
4.2.3 Pressure Sensor Considerations .....	48
4.2.4 GPS Module Considerations .....	48
4.2.5 Secondary Sensors .....	48
4.2.6 FPGA Configuration.....	49
4.3 Printed Circuit Board Layout .....	49
4.3.1 Number of PCB Layers and PCB Dimensions .....	49
4.3.2 PCB Layer Stackup Planning.....	50
4.3.3 Component Placement Considerations .....	51
4.3.4 Track and Routing Considerations .....	53
5. DETAILED FIRMWARE DESIGN .....	57
5.1 Serial Peripheral Interface (SPI) Module .....	60
5.1.1 SPI Protocol Description.....	60
5.1.2 SPI Master Module Design.....	62
5.1.3 SPI Master Module Algorithmic State Machine (ASM).....	63
5.2 PWM Module .....	66
5.3 PWM Capture Module.....	69
5.4 UART Module .....	70
5.4.1 UART Protocol Description .....	70
5.4.2 UART Module Requirements .....	71
5.4.3 UART Module Implementation .....	71
5.5 Dual Port Ram.....	76
5.6 Bus Interface Controller Module.....	77
5.7 IMU Controller Module .....	83
5.7.1 ADIS 16350 Operation.....	83
5.7.2 Implementation .....	84
5.8 ADC Controller.....	88
5.9 GPS Controller .....	91
5.10 Conclusion .....	93
6. SOFTWARE DESIGN .....	94
7. TEST AND SIMULATION RESULTS.....	98
7.1 BIC and DPRAM module .....	98
7.2 SPI Master Module .....	98
7.3 PWM Module .....	101

7.4 PWM Capture Module.....	101
7.5 UART Module .....	101
7.6 IMU Controller Module .....	102
7.7 ADC Controller Module .....	102
7.8 GPS Controller Module.....	102
8. CONCLUSION, RECOMMENDATIONS AND FUTURE WORK .....	121
8.1 Conclusion .....	121
8.2 Limitations .....	123
8.3 Recommendations and future work .....	123
BIBLIOGRAPHY .....	125
APPENDIX A: SPECIFICATIONS OF THE ADIS 16360 AND 16365 .....	128
APPENDIX B: SCHEMATIC DIAGRAMS.....	129
APPENDIX C: FPGA Active Serial and JTAG Configuration Scheme Schematics .....	137
APPENDIX D: COMPONENT PLACEMENT.....	138
APPENDIX E: PRINTED CIRCUIT BOARD.....	139
APPENDIX F-1: PARTIALLY COMPLETED MAIN SYSTEM-BUILDUP .....	141
APPENDIX F-2: View A OF MAIN SYSTEM BUILD UP .....	142
APPENDIX F-3: VIEW B OF MAIN SYSTEM BUILD UP .....	143
APPENDIX G – PICTURES OF THE HARDWARE .....	144



# List of Figures

Figure 1.1 - High level overview of the <i>Micro-avionics</i> system developed in the ESL.....	3
Figure 1.2 - High level overview of the CAN-based PC104 avionics system developed in the ESL. ....	5
Figure 1.3 - A simplified high level overview of the avionics system developed to replace the PC104 Stack. ....	5
Figure 1.4 - An overview of the design process using some systems engineering principles. ....	7
Figure 2.1 - Top-level System Requirements .....	9
Figure 2.2 - Diagram showing the body axis system of an aircraft (Figure taken from [13]) .....	12
Figure 3.1 - Conceptualization of an avionics system in response to URS1 .....	18
Figure 3.2 - Conceptualization of CAN bus backward-compatibility (URS2) and extendibility (URS3) .....	19
Figure 3.3 - Conceptualization of the interchangeability requirement (URS5) .....	19
Figure 3.4 - Conceptual overview of Abstraction Layers and Reconfigurability.....	20
Figure 3.5 - High level overview of the system architecture .....	21
Figure 3.6 - Architecture and overview of the system .....	24
Figure 3.7 - High-level overview of the system timing and data flow .....	25
Figure 4.1 - Diagram illustrating the decision-making process for the microprocessor.....	28
Figure 4.2 - Form-factors of u-blox ANTARIS4 GPS modules.....	31
Figure 4.3 - Diagram illustrating the decision-making process for the GPS sensor.....	32
Figure 4.4 - IMU configurations .....	33
Figure 4.5 - Selection process overview of the IMU Sensor .....	37
Figure 4.6 - Pressure sensors selection process .....	38
Figure 4.7 - Prototype Development Overview .....	41
Figure 4.8 - Schematic Overview of the Hardware System .....	42
Figure 4.9 - Power Supply Distribution .....	43
Figure 4.10 - Power Regulation Strategy .....	44
Figure 4.11 - PTN78020W switching regulator .....	46
Figure 4.12 - Input and Output power to the switching regulators for the worst case efficiency of 85% and a supply input voltage of 9V.....	47
Figure 4.13 - Drawing of the IMU mounted on PCB .....	47
Figure 4.14 - PCB Layer Stackup.....	50
Figure 4.15 - Component Placement Strategy overview of the Top PCB Layer .....	52

Figure 4.16 - Component Placement Strategy overview of the Bottom PCB Layer (Viewed through the Top Layer) .....	52
Figure 4.17 - Screenshot of <i>Agilent AppCAD</i> Utility used to calculate the line impedance for the GPS antenna track. ....	54
Figure 4.18 - PCB layout indicating the certain high current carrying tracks and their widths.....	55
Figure 5.1 - Control and datapath partitions in a synchronous sequential system.....	57
Figure 5.2 - High level Overview of the Main VHDL Modules Required.....	59
Figure 5.3 - Timing diagrams showing the different modes of SPI.....	61
Figure 5.4 - High level concept of the SPI Master Module .....	62
Figure 5.5 - Algorithmic state machine of the SPI master module.....	65
Figure 5.6 - Relationship between the servo control pulse and angle [24].....	67
Figure 5.7 - Block diagram of the PWM module with ports and ASM shown. ....	68
Figure 5.8 - PWM Capture module ASM and ports. ....	69
Figure 5.9 - UART Transmission Character Frame [34].....	70
Figure 5.10 - UART Module Implementation.....	72
Figure 5.11 - UART TX sub-module ASM.....	74
Figure 5.12 - UART RX sub-module ASM.....	75
Figure 5.13 - Dual Port RAM .....	76
Figure 5.14 - Read cycle behavior of the DPRAM module [36] .....	76
Figure 5.15 - Write cycle behavior of the DPRAM module [36] .....	77
Figure 5.16 - The SH7201 BSC .....	78
Figure 5.17 - BSC Read Operation Basic Bus Timing (taken from [37]) .....	79
Figure 5.18 - BSC Write Operation Basic Bus Timing (taken from [37]) .....	80
Figure 5.19 - Bus Interface Controller ASM .....	81
Figure 5.20 - The operation of the SPI Interface for the ADIS 16350 IMU (taken from [25]) .....	84
Figure 5.21 - Interface between the IMU Controller and SPI Master Module.....	84
Figure 5.22 - Algorithmic State Machine of the IMU Controller Module (normal mode).....	85
Figure 5.23 - Data flow of the transaction between the IMU Controller Module and the ADIS 16350 IMU device during one complete cycle (11 frames).....	87
Figure 5.24 - Conceptual ASM of the IMU Controller Module for both modes (normal and configuration).....	88
Figure 5.25 - High level block diagram showing the interface between the SPI Master Module and the SPI ADC Controller Module.....	89

Figure 5.26 - SPI frame of the ADS8344 ADC device (taken from [38]).....	90
Figure 5.27 - GPS Controller.....	91
Figure 5.28 - GPS Controller Module ASM1.....	92
Figure 5.29 - GPS Controller Module ASM2.....	93
Figure 6.1 - Software Conceptual Design.....	94
Figure 6.2 - High level flow chart of the software design concept.....	96
Figure 7.1 - Interaction between the BIC and DPRAM modules .....	104
Figure 7.2 - Write cycle outputs of the microprocessor to the Bus Interface subsystem with wait states inserted.....	105
Figure 7.3 - Read cycle outputs of the microprocessor to the Bus Interface subsystem with wait states inserted.....	105
Figure 7.4 - Simulation result showing the functionality of the BIC to DPRAM subsystem .....	106
Figure 7.5 - Simulation Result showing Read Data Hold Time ( $t_{RDH}$ ) .....	106
Figure 7.6 - SPI Master Module Simulation 1 .....	107
Figure 7.7 - SPI Master Module Simulation 2 .....	107
Figure 7.8 - SPI Master Module Simulation 3 .....	108
Figure 7.9 - SPI Master Module Timing Parameters .....	108
Figure 7.10 - PWM Module Simulation (3.4MHz and pwmcount = 2040) .....	109
Figure 7.11 - PWM Module Simulation (3.4MHz and pwmcount = 8191) .....	109
Figure 7.12 - PWM Module triggering .....	109
Figure 7.13 - PWM Capture Module simulation .....	110
Figure 7.14 - PWM Capture Module simulation showing the strobe signal.....	110
Figure 7.15 - Baud Rate (9600 baud) Simulation .....	111
Figure 7.16 - TX and BRG (9600 baud) Simulation (see Figure 7.18).....	111
Figure 7.17 - UART Module Simulation with serial output to serial input loopback (See Figure 7.19) .....	112
Figure 7.18 - UART BRG and TX sub-module connections.....	112
Figure 7.19 - Complete UART Module (TX to RX loopback test).....	113
Figure 7.20 - IMU Controller and SPI Implementation .....	114
Figure 7.21 - Simulation of the interface between the IMU Controller and the SPI Master Module .....	115
Figure 7.22 - ADC Controller and SPI Master Module implementation .....	116

Figure 7.23 - Simulation of the interface between the ADC Controller and the SPI Master Module .....	117
Figure 7.24 - Frame 1 of ADC Controller to SPI Simulation (see Figure 7.23).....	117
Figure 7.25 - GPS Controller Simulation (part 1) .....	118
Figure 7.26 - GPS Controller Simulation (part 2) .....	118
Figure 7.27 - GPS Controller Simulation (part 3) .....	119
Figure 7.28 - GPS Controller Simulation (part 4) .....	119
Figure 7.29 - GPS Controller Simulation (part 5) .....	120

# List of Tables

Table 2.1 - Summary of the primary sensors to be used in the avionics platform and their respective base requirements.....	13
Table 2.2 - Traceability matrix showing tracing the System Requirements back to the User Requirements Specification.....	17
Table 4.1 - Microprocessors considered for this project and their selected specifications.....	27
Table 4.2 - List of u-blox LEA-4x GPS modules considered in this project.....	31
Table 4.3 - Specifications of six-degree-of-freedom IMU candidates.....	34
Table 4.4 - Devices interfacing with the FPGA and number of pins required.....	40
Table 4.5 - Recommended PCB Track Widths.....	55
Table 5.1 - SPI Modes of operation.....	61
Table 5.2 - List of the SPI devices used in this project.....	62
Table 5.3 - SPI Master module port descriptions.....	66
Table 5.4 - PWM module port descriptions.....	68
Table 5.5 - Port descriptions of the PWM Capture Module.....	70
Table 5.6 - UART BRG sub-module ports and descriptions.....	72
Table 5.7 - UART TX sub-module ports and descriptions.....	73
Table 5.8 - UART RX sub-module ports and descriptions.....	75
Table 5.9 - Port Descriptions of the Bus Interface Controller.....	82
Table 5.10 - 22-bit data word format outputted by the IMU Controller Module's data_out line....	87
Table 5.11 - Identifier on data_out line of ADC Controller.....	90
Table 7.1 - Timing Parameters of Simulation Result .....	100
Table 7.2 - SPI timing requirements vs simulated timing results.....	101
Table A.1 - Analog Devices ADIS 16360 and 16365 IMU Specifications.....	128

# Acronyms

ADC – Analog to Digital Conversion

ASM – Algorithmic State Machine

BSC – Bus State Controller

CAN – Controller Area Network

DGPS – Differential Global Positioning System

DSP – Digital Signal Processor

EKF – Extended Kalman Filter

ESL – Electronic Systems Laboratory

FLOPS – Floating-Point Operations per Second

FPGA – Field –Programmable Gate Array

FPU – Floating-Point Unit

GPS – Global Positioning System

HIL – Hardware-in-the-Loop

I/O – Input / Output

IMU – Inertial Measurement Unit

MEMS – Micro-Electromechanical Systems

MIPS – Million Instructions per Second

OBC – Onboard Computer

PCB – Printed Circuit Board

RF – Radio Frequency

SRS – System Requirements Specification

UART – Universal Asynchronous Receiver/Transmitter

UAV – Unmanned Aerial Vehicle

URS – User Requirements Specification

USB – Universal Serial Bus

WAAS – Wide Area Augmentation System

# 1. INTRODUCTION

## 1.1 BACKGROUND

Unmanned aerial vehicles, or UAVs, are used to perform various tasks which were previously done by human-piloted aerial vehicles and are becoming more widespread and common. A UAV can loosely be defined as an aircraft which has no human pilot onboard and is capable of autonomous flight. From an autonomous flight perspective, the UAV system is centered around the avionics hardware onboard which acts as the 'brain' of the system and substitutes the role of the onboard human pilot.

The avionics platform is basically an embedded system with a specialized function. Depending on the complexity of the required task to be performed by the aircraft, the avionics platform can become increasingly more complex and powerful. However, the overall task performed by the avionics hardware remains common to all systems: It gathers data from its surroundings, processes it into useful information and, according to a set of control guidelines, generates appropriate actuation commands in order to stabilize and guide the aircraft.

In industry, there is a large amount of development being done in terms of avionics hardware. This project aims to develop an avionics platform for UAV research being done in the Electronic Systems Laboratory (ESL) at Stellenbosch University. A few of the commercially available systems, as well as other research based systems, will be outlined in this chapter. Furthermore, the background of the systems currently used in the ESL will also be outlined, all with the aim of showing where this project fits into the greater picture.

It is important at this point to clarify the difference between the terms avionics and autopilot. For the purposes of this thesis, the term avionics refers to the actual hardware onto which the control algorithms are to be loaded and together, the avionics plus the control system, form an autopilot system.

## 1.2 COMMERCIAL AUTOPILOT & AVIONICS SYSTEMS

*MicroPilot* [1] offers a commercially available range of UAV autopilot systems called the *MP2028 Series*. These are very small in size and lightweight. The feature product of this series is the *MP2128<sup>HELI</sup>* which makes provision for both fixed wing and vertical takeoff and landing (VTOL) UAVs. It incorporates all the hardware for full autopilot capabilities. This includes sensors to measure altitude (up to 12000m), acceleration (2g maximum), angular rate (maximum of 150°/s)

and navigation data (GPS). It can be used for autonomous takeoff and landing, airspeed control, attitude control and navigation control. This is a very flexible autopilot and allows the user access to control certain lower level configurations such as the ability to load user code. It also includes ground control software and telemetry capabilities.

*Cloud Cap Technology* [2] offers the *Piccolo* autopilot systems for UAVs. Their range of products also incorporates hardware and software for full autopilot capability. Among the data provided by the sensors are GPS navigation information, accelerations (10g maximum), angular rates (up to 300°/s), static pressure (15 to 115kPa range) and differential pressure (4kPa maximum). It also allows a user to interface to it for a certain amount of control and is extendible if further hardware is required. It is also quite small and lightweight. The smallest system in the range weighs 124grams and has a size of 131x55.6x19mm.

*Adaptive Flight Inc.* [3] offers the *FCS20 Integrated Flight Control System* which was developed in collaboration with Georgia Institute of Technology [4]. This is a system which uses DSP technology to provide a large amount of processing power and an FPGA to provide flexibility in the design. The hardware contains sensors for navigation control and communication interface circuitry. The physical size of the hardware is two credit-card sized printed circuit boards stacked on each other and it is also light in weight.

Another autopilot system is the *wePilot2000* by *weControl* [5]. It also has full UAV capabilities and is comparable to the systems above in terms of its features.

Many of these systems have full autopilot capabilities but do not allow the user the ability to have full control over the avionics package. This can become a limiting factor in research environments and for this reason, the design and development of custom avionics platforms are implemented to have full control over the functioning of the system and how it is used. An added advantage is that it is possible to reduce costs during development.

### **1.3 AVIONICS SYSTEMS IN THE ELECTRONIC SYSTEMS LABORATORY (ESL)**

In this section a brief overview of the development history of the current avionics systems available in the ESL will be discussed with the aim of more clearly establishing where this project fits into the greater picture.



The first formal avionics system in the ESL was developed by [6] and is known as the *Micro-avionics* package. This system is basically comprised of different boards which are plugged into each other to create an avionics platform, namely:

- Controller Board – This is the heart of the system which interfaces to all the boards and executes the main control for the autopilot system. It also interacts with the RC flight platform, i.e. the actuators and RC receiver.
- Inertial Measurement Unit (IMU) Board – This provides the system with the inertial state of the vehicle.
- Airdata Board – This contains the pressure sensors for the avionics system.
- RF Module – This contains the hardware for communication with the ground station.
- GPS Receiver Module – It is responsible for providing GPS data to the system.
- Battery and Power Distribution Board – This powers the avionics package.

Furthermore, a ground station was also developed which allows a PC to communicate with the avionics package through an RF Transceiver Module connected to the PC. The software and interfaces for the ground station were also developed. A high level overview of this system is shown in Figure 1.1. The architecture of this avionics system basically represents a ‘star’ network in which the sensors are all connected to a central controller board.

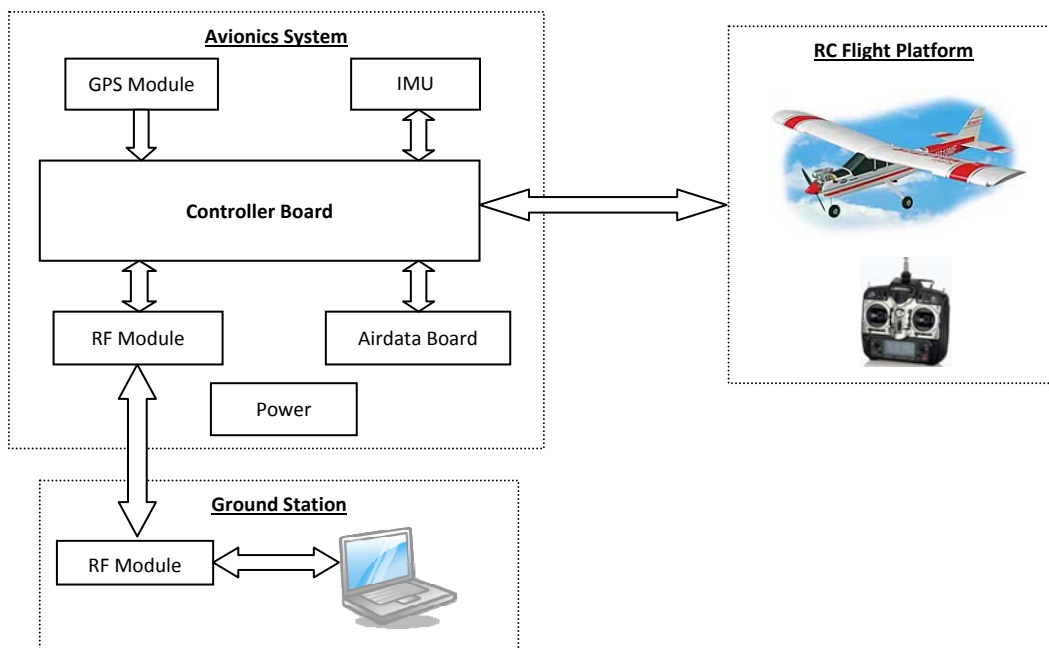


Figure 1.1 - High level overview of the *Micro-avionics* system developed in the ESL.

The main limitations of this system is that it has a limited amount of processing power available for more complex control algorithms and the system is also not easily extendible. This led to the

development of an avionics architecture utilizing a Controller Area Network (CAN) bus configuration [7], [8], [9] & [10].

This system consists of a PC104 (form factor) Stack of PCBs with many nodes connected to it via a CAN bus. The advantage of this system is that it provides extendibility as more or fewer nodes can be connected depending on the requirements of the application. The PC104 Stack around which the system is built consists of three modules:

- Onboard Computer (OBC) – This provides all the processing power for the control system. This is a commercially available PC104 hardware board of which there is a vast variety of choices on the market, each with different features and cost implications. It also provides large processing power capabilities: for example, the PC104 board used by [9] has a 300 MHz Intel Pentium III processor onboard.
- PC104/CAN Controller – This board is responsible for the timing of the entire system and is the link between the OBC and the CAN bus. For a discussion surrounding the timing sequence and the functioning of this board, see [9].
- GPS and RF Link Daughter Board – This board interfaces with the OBC and provides it with GPS data and the RF connection to the ground station.

On the CAN bus, as stated previously, many nodes can be connected:

- Servo Board CAN Node – This is the most important node on the CAN bus as it is the interface between the avionics system and the RC flight platform (actuators and RC receiver), as shown in Figure 1.2, and is always present in the system. It is also the switch between autopilot and human pilot mode. It employs important safety features which allow the aircraft to again be manually controlled in the case of any system failures.
- IMU CAN Node – It provides inertial measurements to the system.
- Magnetometer/Pressure CAN Node – It contains pressure sensors and a magnetometer.
- Hardware-in-the-loop (H.I.L.) CAN Node – This connects to a PC and allows the autopilot to be tested prior to an actual flight and thereby verify the correct functioning of the control system.

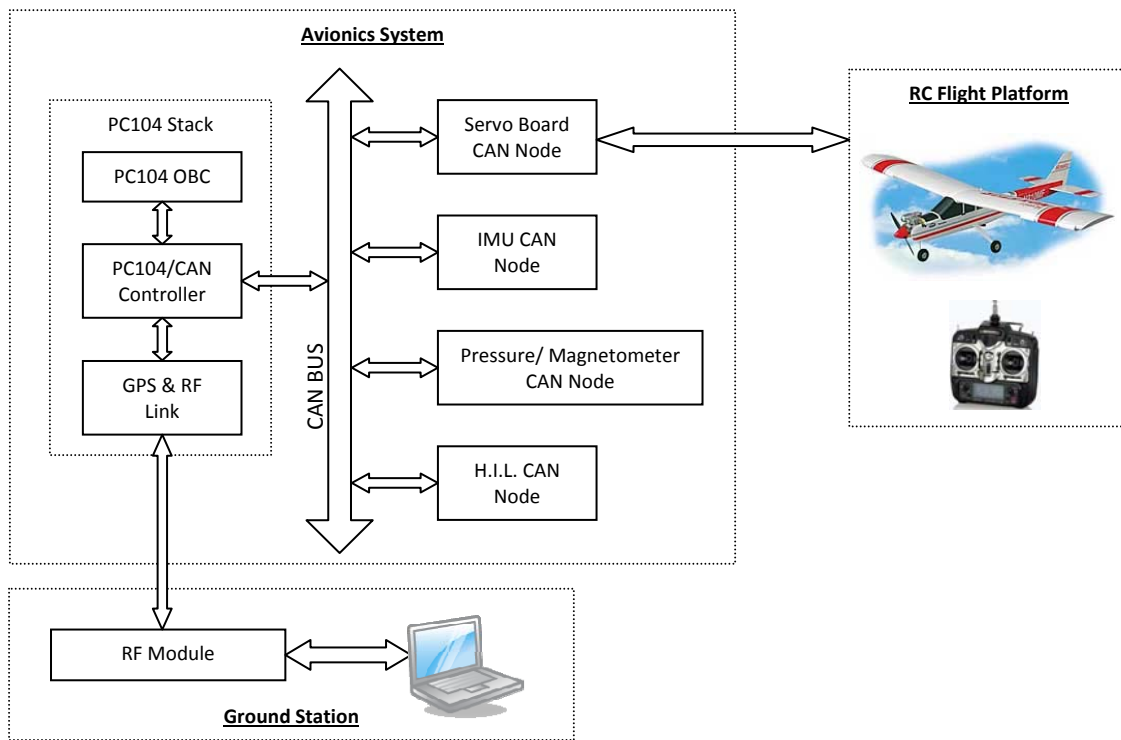


Figure 1.2 - High level overview of the CAN-based PC104 avionics system developed in the ESL.

An avionics system utilizing the CAN bus structure but replacing the PC104 Stack was then developed by [11]. The system is controlled by the Main Avionics CAN Node which has much less processing power than the PC104 Stack but still enough to provide adequate control for certain applications. It is also considerably smaller, uses less power and has significantly less RF noise than the PC104 Stack. As it is a replacement for the PC104 Stack, it also makes provision for the GPS and RF communication with the ground station.

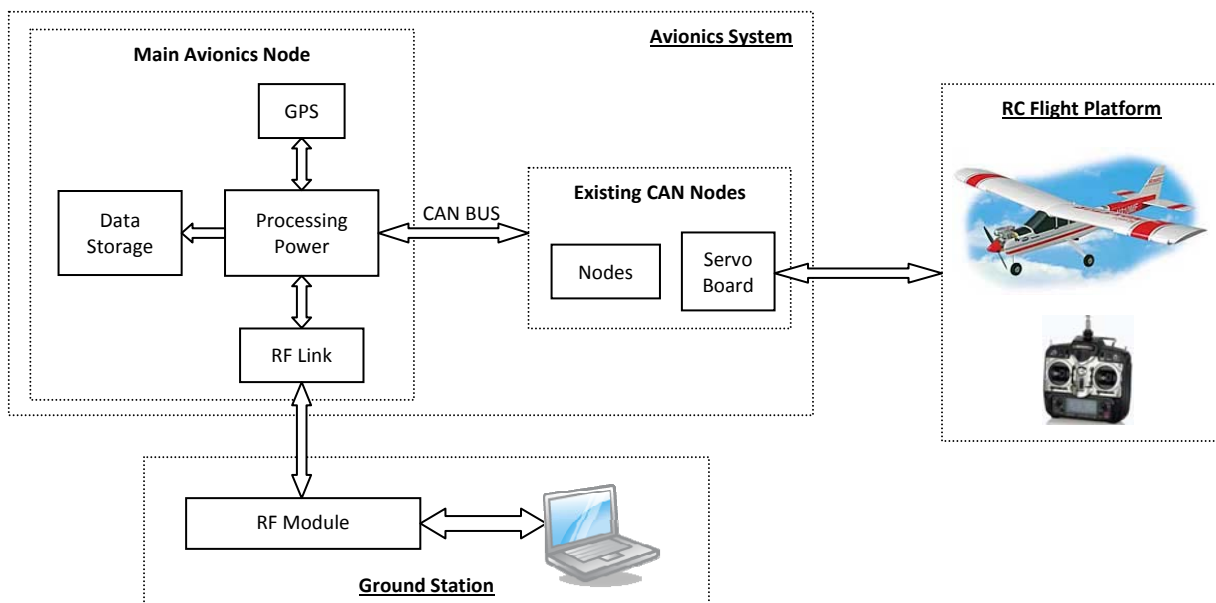


Figure 1.3 - A simplified high level overview of the avionics system developed to replace the PC104 Stack.

Given the outline of the systems above, the need arose within the ESL for an avionics platform which has the processing capabilities of the PC104 with the necessary hardware integrated into it for compactness. The system also needed to remain backward compatible with CAN architecture used in order to allow for easy extension. This project is a step towards developing such a system.

#### 1.4 USER REQUIREMENTS SPECIFICATION

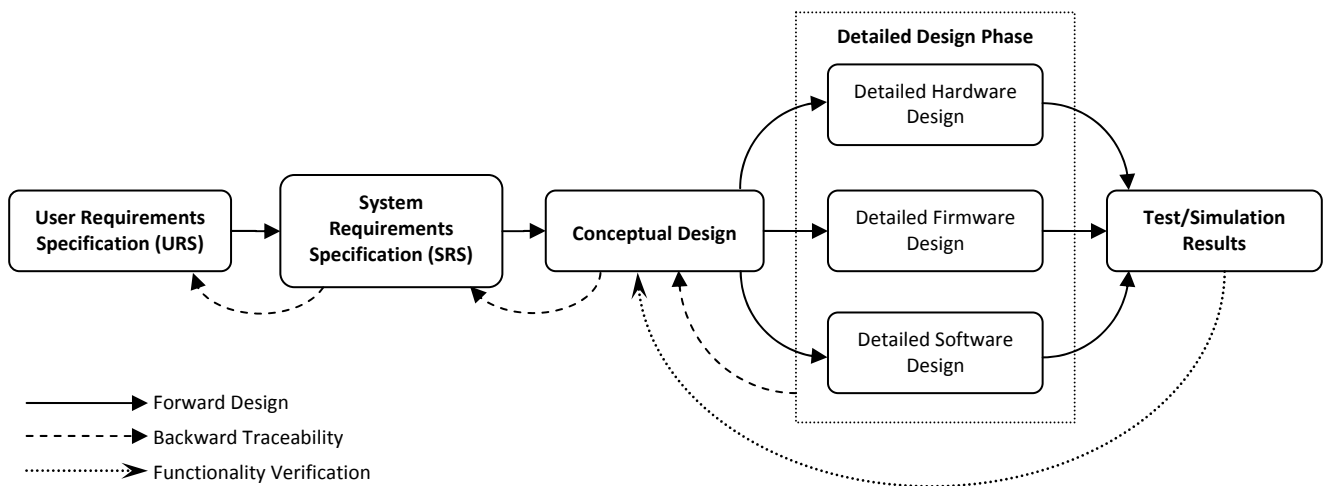
In order to formalize the avionics system design in this project, a basic system engineering approach is adopted. This approach has three phases: a requirements analysis phase, a design phase and a test or simulation phase. The first step would therefore be to do the *requirements analysis* and thereby set up the *User Requirements Specification (URS)*:

- The highest level requirement of this project is that it must be an **avionics system** and not an autopilot system. The difference between the two is stated earlier. The control algorithms are therefore not part of the scope of this project. The hardware, however, must support and provide a platform for autopilot implementations and must therefore consider all the factors needed for such a system. **(URS1)**
- It must be **backward-compatible (URS2)**, as previously stated, with the current avionics systems and structures already in place in the ESL. The main feature here is the CAN bus configuration which exists and provides **extendibility (URS3)**. Other structures to consider are current control algorithms which exist, the ground station and H.I.L. simulations.
- The **processing power** should be in the mid-performance range and capable of running certain current algorithms developed in previous projects in the ESL. **(URS4)**
- Following on the point above, the design should make provision for the processing power to be scaled up or down without needing major changes to the schematic-level design of the system. This will be termed as the **interchangeability** of processing units and provides flexibility to the system. Furthermore, the system should contain a certain amount of flexibility in order to update hardware. **(URS5)**
- The avionics platform is mainly for **research purposes** for projects in the ESL and thus does not need to be compliant with any stringent standards (e.g. military specifications). However, good engineering practices should be followed. **(URS6)**
- The hardware design should be focused towards eventually producing a **single printed circuit board (PCB)** with all the necessary hardware integrated on it. **(URS7)**

- The system should be **reconfigurable** for different aircraft platforms. This means that a user should have a certain amount of access to reload different control systems onto the avionics platform. (**URS8**)
- Following on URS8, the system should have well-defined interfaces in order to provide certain **layers of hardware abstraction**. (**URS9**)
- The system should be **low-cost, small in size and lightweight**. These parameters are in relation to the current avionics packages used in the ESL. (**URS10**)
- The hardware components should readily be available on an **off-the-shelf** basis to promote ease of duplication within minimal time. (**URS11**)

## 1.5 THESIS OUTLINE

The structure of this thesis follows the same steps used in the design process. System engineering techniques are employed as illustrated in Figure 1.4.



**Figure 1.4 - An overview of the design process using some systems engineering principles.**

The User Requirements Specification (URS) has already been discussed in **Section 1.4**. From the URS, the System Requirements Specification (SRS) for this project are drawn up and discussed in **Chapter 2**. This gives a very high-level overview of what the requirements are in terms of technical specifications. In **Chapter 3**, the Conceptual Design for the system is discussed. This describes the design philosophy and approach employed and also gives an abstract description of the system in terms of how the hardware, firmware and software interact. **Chapter 4** focuses on the design of the hardware in detail. This includes a discussion surrounding the component choices and the schematic level designs of the hardware. **Chapter 5** describes the FPGA firmware designs implemented. The software design concept for the candidate microprocessor in this project is

presented in **Chapter 6**. Finally, Test/Simulation results are provided and discussed in **Chapter 7**. In this thesis, traceability will be shown in order to justify the major decisions in the design processes and also verify the functionality of the implementations. This technique aims to keep this project, as well as this thesis document, focused on the design objectives and requirements. **Chapter 8** concludes this thesis with an overall summary of the project and traces the project back to the user requirements.

## 2. SYSTEM REQUIREMENTS SPECIFICATION

As part of the requirements analysis phase, it is necessary to convert the User Requirements Specification (URS) to System Requirements Specification (SRS). These are the technical specifications and guidelines to be used in the design phases of the project. This conversion process will be the focus of this chapter.

The most top-level requirement of the URS is that this project develops an avionics platform which provides support for UAV autopilot systems to be implemented. It must therefore contain all the components of such a system: sensors to gather data about the aircraft's state, an interface with the aircraft's actuators, a processing unit and the interfaces for interaction with the user, as illustrated in Figure 2.1. These components will be discussed and further expanded upon in the subsections which follow.

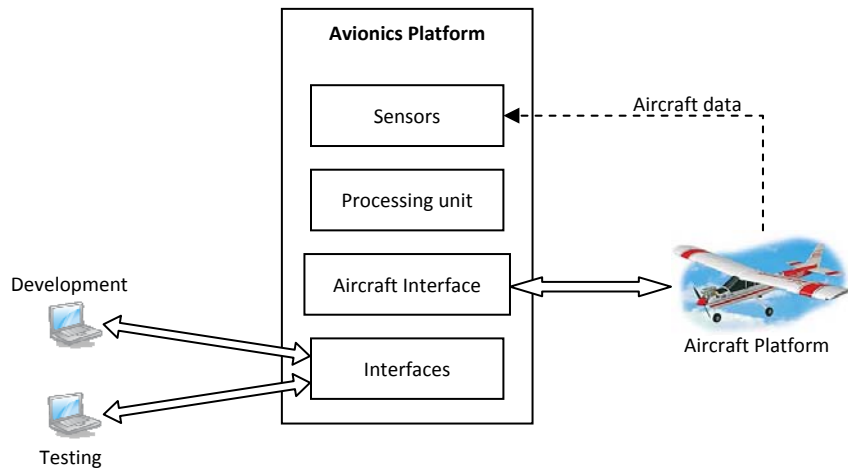


Figure 2.1 - Top-level System Requirements

### 2.1 SENSORS IN THE AVIONICS SYSTEM

The sensors must deliver the measurements which the specific control system requires. Among the factors which need to be considered are the required range and resolution. The sensors required for this project are also a function of the backward-compatibility requirement of the URS as the system must make provision for current control system algorithms. The fundamental requirements of each sensor will be addressed in this section.

#### 2.1.1 Pressure sensors

Pressure measurements are an important aspect of any avionics system. There are two important pressure sensors used, namely, a static air pressure sensor and a differential air pressure sensor.

The **static air pressure sensor** is used to calculate the barometric altitude of an aircraft. An increase in altitude results in a decrease in static pressure, as well as a decrease in temperature and this is given by the barometric formula for a standard atmosphere [12],

$$p(h) = p(0) \left(1 - \frac{\beta h}{T_0}\right)^{\frac{Mg}{R\beta}} \quad (2.1)$$

where,  $p(0)$  is the static pressure at sea-level (101.325 kPa),  $\beta$  is the vertical linear lapse rate of temperature with height (0.0065 K/m for the troposphere i.e. below 11km),  $h$  is the altitude in meters,  $T_0$  is the standard temperature at sea-level (288.16 K),  $M$  is the molecular mass of the Earth's atmosphere (0.02897 kg/mol),  $g$  is the gravitational constant (9.80665 m/s<sup>2</sup>) and  $R$  is the universal gas law constant for air (8.3145 J/K/mol).

The pressure measurement range requirements are governed by the altitude of the location which the aircraft will fly at. As this project will be used for research purposes, the altitude range requirement for this project was set at a range from sea-level up to 3000m above sea-level. This provides flexibility in terms of the locations at which the avionics platform can be flown. This was substituted into equation (2.1) yielding a static pressure range of 70.105 kPa to 101.325 kPa. Allowance would also have to be made for high pressure days at sea-level and thus an upper limit of 105 kPa for the pressure range was chosen [6].

Another important factor to consider for the barometric pressure measurement is the resolution required. In order to determine the resolution, the pressure gradient is required at the height of interest. Since Equation 2.1 is a non-linear function, an increase in height causes a decrease in pressure resolution i.e. less pressure change is observed for each fixed step in height with an increase in altitudes. The aim is thus to determine the worst case pressure resolution required by the absolute pressure sensor. This would occur at the highest altitude specification of the flight envelope, i.e. at 3000m. Differentiating equation 2.1 and evaluating it at an altitude of 3000m thus yields a pressure gradient of approximately 8.9 Pa/m around that specific point. The altitude resolution specification for this project was set at a value of less than 1 ft (0.3048m). Thus, in order to realize this specification, the pressure measurement should be resolved to a value less than 2.71 Pa.



The **differential air pressure sensor** is used to measure dynamic pressure using a pitot-static tube system, as described by [6]. From this measurement the airspeed of an aircraft can be calculated using Bernoulli's equation,

$$q = \frac{1}{2} \rho V^2$$

or, when rearranged,

$$V = \sqrt{\frac{2q}{\rho}} \quad (2.2)$$

where,  $q$  is the dynamic or differential pressure in Pa,  $\rho$  is the air density (1.2250 kg/m<sup>3</sup> at sea-level) and  $V$  is the indicated airspeed in m/s.

The maximum airspeed requirement for this project was set at 60 m/s. This provides a sufficient airspeed range for the aircraft used in the ESL. Substituting this into equation (2.2) yields a dynamic pressure of 2.205 kPa at sea-level. The density of air decreases with altitude which causes a decrease in the pressure reading at a higher altitude for a given airspeed. The pressure value of 2.205 kPa is thus the maximum sensor measurement capability requirement.

In order to find the airspeed resolution, equation 2.2 is differentiated. This gives the differential pressure gradient in pascals per m/s,

$$\frac{\partial q}{\partial V} = \rho V \quad (2.3)$$

and thus the resolution can be found by,

$$\Delta q = \rho V \times \Delta V \quad (2.4)$$

Since there is less pressure difference at lower speeds, the worst case resolution occurs at the lowest speed to be measured. The above equation (2.4) is thus evaluated at this point. For this application, this speed is set at 5m/s. Furthermore, the desired resolution at this speed is 0.5m/s. Substituting these parameters into the equation (2.4) yields a differential pressure resolution of less than 3.06 Pa to be measured at sea-level.

### 2.1.2 Inertial measurement sensors

These are sensors which can be used to determine the orientation and track the motion of a body in three dimensional space. They are fundamental for inner-loop stabilization control of an aircraft. For this purpose, **gyroscopes** are used to measure the angular rates about an axis and

**accelerometers** are used to measure the linear specific acceleration along the axis. In aircraft applications, it is important to know these angular rates (roll-, pitch- and yaw-rate) and linear accelerations (x-, y- and z-axis accelerations) relative to the body axis system shown in Figure 2.2.

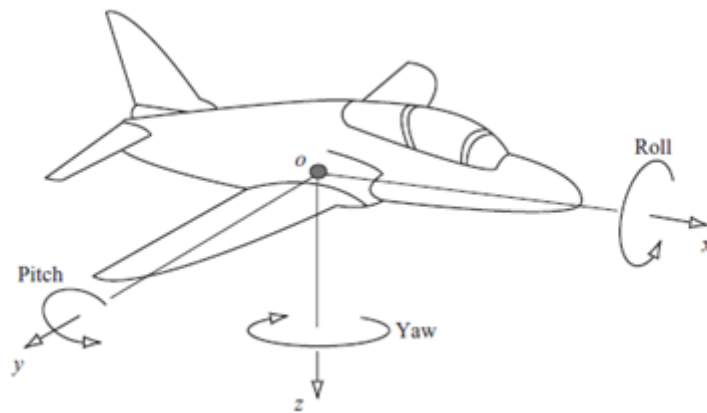


Figure 2.2 - Diagram showing the body axis system of an aircraft (Figure taken from [13])

In embedded system applications, under which the avionics platform can ultimately be classified, MEMS inertial sensors are extensively used as they offer low power consumption, are relatively low in cost and are small in size. In order to provide the avionics platform with flexibility with regard to different types of projects, the inertial sensors would have to be capable of measuring rates for a wide variety of flight envelopes from conventional flight to acrobatic flight envelopes. On inspection of previous projects done in the ESL, the requirements of the rate gyros and the accelerometers were set at  $300^\circ/\text{s}$  and  $10g$ 's respectively, [6] and [9]. Gyroscopes and accelerometers also suffer from certain noise effects which will be considered during the selection process.

### 2.1.3 Magnetometer

A magnetometer is needed to determine the exact orientation of the aircraft with respect to the Earth's magnetic field. This is also used by current control algorithms (i.e. the kinetic state estimator) in the ESL and thus needs to be considered and provided for as part of the backward-compatibility URS.

### 2.1.4 GPS sensor module

A GPS sensor provides essential data for navigation control and provides the system with its absolute position and velocity with respect to the Earth's co-ordinate system. The GPS sensor also provides data such as altitude and heading, amongst others. Low-cost GPS modules however have relatively slow update rates with respect to the frequency of the control system command outputs and thus it is used in conjunction with other sensors to make up for this shortfall. The sensors used

in the ESL have an update rate of 4Hz and this specification should be matched or improved. The sensor must also adhere to the flight envelope requirements for this project, i.e. a velocity range up to 60 m/s and an altitude range up to 3000m. Almost all GPS sensors adhere to this requirement but it must still be verified during the component selection process. The GPS module should also provide DGPS and WAAS capabilities.

### 2.1.5 Secondary sensors

In embedded applications it is useful to include a **temperature sensor** to monitor the temperature of certain onboard components and ultimately prevent component and system failure. A temperature sensor will thus be included in this system. A **current sensor** will also be included in the design to measure the total power consumption of the avionics platform and thereby act as a “fuel gauge” for the batteries. A **battery voltage measurement** is also required in order for a user to monitor the battery voltage and thereby prevent it from dropping below its minimum operating voltage (this voltage is discussed later in this chapter).

**Table 2.1 - Summary of the primary sensors to be used in the avionics platform and their respective base requirements**

Device	Requirement
Static Pressure Sensor	Measurement range of 70.105 to 105 kPa with a resolution of 3.386 Pa
Differential Pressure Sensor	Measurement range of 0 to 2.205 kPa with a resolution of 3.0625 Pa
Rate gyroscopes (MEMs)	0 to 300°/s
Accelerometers (MEMs)	10g upper limit
Magnetometer	Not Specified
GPS sensor	4Hz update rate

## 2.2 THE MICROPROCESSOR

Choosing a processor can prove to be a difficult task during the design phase of a project and one therefore needs certain parameters relevant to the project by which the performance can be measured and comparisons between the vast selections of processors can be drawn. The main parameters which were set up for this purpose are outlined below:

**Computational measure: Floating-point operations** – Some of the control and estimation algorithms which are intended to be run on the processor have a number of floating-point intensive calculations. One specific type of algorithm, used as a benchmark for the processor selection in this project, is the Extended Kalman Filter (EKF) algorithm. This requirement also therefore traces back to the backward-compatibility URS.

A numerical algorithm's computational performance can be evaluated by counting the amount of floating-point operations it uses to obtain an answer. It would thus be advantageous for the microprocessor to have a dedicated Floating-Point Unit (FPU) in order to speed up the throughput of floating-point calculations. The floating-point capabilities of a processor are measured in FLOPS (*Floating-Point Operations per Second*). The minimum FLOPS specification for this application is calculated by determining how many floating-point operations are done in the EKF and dividing it by the desired amount of time available for the completion of the calculation, as shown below. Different EKF implementations can have a different amount of floating-point operations but the calculation was done to find an approximate estimate and then factor in a margin during component selection to account for more intensive calculations. Considering Kalman Filter implementations done in previous projects [10] and [9], the number of FLOPS required was calculated as shown below. A high-level overview of the EKF is given below, the actual details and equations of the Extended Kalman Filter are beyond the scope of this project

An EKF basically consists of two main steps, shown below. There are numerous amounts of matrix and vector computations to be performed. The approximate number of floating-point operations required for each step, sourced from [10], is:

- 1) A state and covariance propagation step with

$$4n^3 + 3n^2 + 2nm + n \text{ floating-point operations, and}$$

- 2) A measurement update step with

$$4n^3 + n^2(8p + 1) + n(6p^2 + 4p + 1) + 2p^3 + 4p^2 + 7p - 1 \text{ floating-point operations,}$$

where,  $n$  is the number of states,  $m$  the number of inputs and  $p$  the number of measurement outputs of the EKF.

From these equations, it can be seen that the amount of floating-point operations increase substantially with an increase in EKF dimensions due to the presence of the second and third order terms. Substituting the dimensions for the EKF implementation by [9], which is the most intensive EKF algorithm within the ESL, in the above equations, an approximation of the number of floating-point operations for this EKF can be found. With these parameters (i.e.  $m = 6$ ,  $n = 16$  and  $p = 9$ ) the approximate amount of floating point operations required is in the order of  $62.644 \times 10^3$ .

The desired amount of time within which to perform these operations was conservatively selected as 5ms, which is also the amount of time the current CAN avionics system uses to execute control algorithms. The minimum desired amount of floating-point operations per second is thus 12.53MFLOPS. A 50% safety margin was added which yielded a value of about 25MFLOPS.

**Peripherals** – The URS requires the system to have interchangeability in terms of the processing power. A common microprocessor interface is thus required in order to connect different microprocessors to the system. Any processor chosen and integrated into future designs of this system must therefore adhere to this requirement.

## 2.3 AIRCRAFT INTERFACE

The avionics system must be able to interact with the aircraft platform in two ways: Firstly, it must have the ability to interface with the RC receiver as part of a safety-pilot feature enabling the system to be switched between autopilot and human-pilot mode. Secondly, it requires an interface with the actuators of an RC aircraft.

**RC Receiver Interface** – The system is required to interface with common RC receiver units which output PWM signals of 5V amplitude. Provision for eight PWM channels must be made in the design.

**Actuators: RC Servos** – The servos are controlled by PWM signals which command the servo angle. This will be discussed in more detail during the design phase of the system. A PWM interface is required to drive up to eight servos simultaneously and this would make it compatible with many current applications in the ESL.

## 2.4 INTERFACES

**Existing Ground Station** – A ground station structure exists in the ESL and the avionics system is required to communicate with it. A 2.4 GHz RF link is therefore required in the system.

**SD-Card Interface** – During flight tests in autopilot systems, it is necessary to log the flight data for analysis thereafter. An SD-card interface is currently used within the ESL structure.

**Existing CAN Interface** – In the ESL, a number of existing CAN nodes have been designed and built in previous projects, as discussed in Chapter 1. The embedded hardware board in this project thus needs to make provision for a CAN bus with a data rate of 800 kbps, as used in the ESL.

**USB Interface** – It was decided that it would be useful to have an easily accessible USB port to allow communication between the system and a PC. USB was decided upon rather than UART because of the higher data transfer rates possible. The USB interface is useful during the design phase for debugging and during the testing phase for downloading flight data. Since the SD-card is sometimes not easily accessible within the aircraft, this allows flight data to be read without having to remove the SD-Card. It can also be used to alter configurable system parameters and this traces back to the reconfigurability requirement of the URS.

## 2.5 FURTHER CONSIDERATIONS

During the component selection phase, close attention also needs to be given to the following aspects:

**Power consumption** – This is a very important factor in UAV applications as power usage is directly linked to flight time and thus battery life is an important commodity. The avionics platform will be powered by lithium-polymer (Li-Po) batteries which are available in 3.7V cells. It is desired to connect Li-Po batteries of between three and six cells to the system. Furthermore, special consideration will need to be given to “power hungry” devices during component selection.

**Physical Size** – In UAV applications, especially the target aircraft for this project, room for extra hardware is limited. Furthermore, a huge increase in weight and payload can significantly alter the dynamics of the aircraft. Careful consideration thus has to be given to component choices in trying to limit the physical size of the completed hardware.

**Cost** – As this is a research project, cost needs to be factored into the equation during component selection. The cost of the development tools required for programmable components also needs to be considered.

**Availability and support** – This is another factor to be taken into account in any hardware project in order to prevent unnecessary complications during the hardware design process. As far as possible, components which are currently being used in the ESL should be strongly considered unless a better and more viable solution can be found.

**Ease of Integration** – This deals with the ease with which the hardware components can be integrated into the system with regards to the supporting hardware needed for it to function. This needs to be considered to prevent design complications and minimize development time.

## 2.6 TRACEABILITY OF SYSTEM REQUIREMENTS

As part of the system engineering approach adopted in this project, a traceability matrix from the SRS to the URS is drawn up and shown below.

		System Requirements Specification														
		CPU		Sensors						Interfaces				Aircraft Interfaces		
		25 MFLOPS	Common Peripheral Interface	Static Pressure	Differential Pressure	Gyroscopes (x3)	Accelerometers (x3)	Magnetometer	GPS	Secondary Sensors	2.4 GHz RF link	SD-Card Interface	800 kbps CAN	USB Interface	Servos	Receiver Interface
User Requirements Specification	Avionics			x	x	x	x	x	x			x			x	x
	Backward-compatibility	x		x	x	x	x	x	x		x	x	x			
	Processing power	x														
	Interchangeability		x													
	Research purposes											x		x		
	Good engineering practice									x				x		
	Single PCB															
	Reconfigurable													x		
	Hardware Abstraction															
	Low-cost, small & lightweight															
	Off-the-shelf components															

Table 2.2 - Traceability matrix tracing the System Requirements back to the User Requirements Specification.

### 3. CONCEPTUAL DESIGN

Following the completion of the requirements analysis phase, the design phase of the project commenced. The first step in the design phase is to conceptualize a high-level implementation of the system. This conceptual design describes the overview of the system architecture as well as the interaction between the different components in the system in terms of control and data-flow.

#### 3.1 CONCEPTUALIZATION BASED ON USER REQUIREMENTS SPECIFICATION ANALYSIS

The most important consideration in conceptualizing a system implementation is the fulfillment of the requirements stated in the URS, as these are the guidelines which essentially govern the direction of the design. The most relevant requirements will thus be listed below and a simple conceptual design for each realized individually, eventually combining the ideas of each into the overall system conceptualization.

##### **Avionics system (URS1):**

The conceptualization for an avionics system has already partly been discussed in Chapter 2. The simplest avionics system would gather sensor data, pass it on to a microprocessor which executes the programmed control and in turn commands the aircraft servos in order to realize the control. Such a system is shown in Figure 3.1.

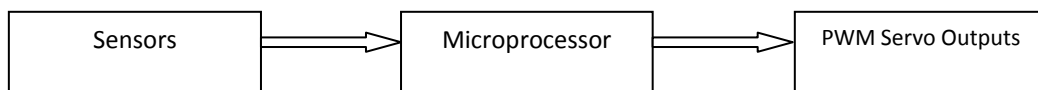


Figure 3.1 - Conceptualization of an avionics system in response to URS1

##### **Backward-compatibility (URS2) and Extensibility (URS3):**

As highlighted previously, backward-compatibility is a very important aspect as the final design concept must fit into the current avionics structures in place in the ESL. The main consideration for this is the current CAN bus centered architecture as discussed in section 1.3. The design must therefore provide a port for the CAN bus to plug into and adhere to the functioning and protocol of the CAN system. This also allows the potential for more CAN nodes to be developed and added to the system. A conceptual design for such a system is shown in Figure 3.2.



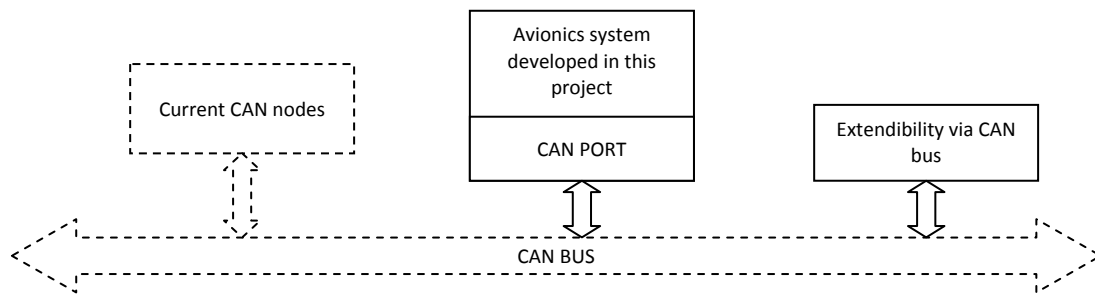


Figure 3.2 - Conceptualization of CAN bus backward-compatibility (URS2) and extendibility (URS3)

### Interchangeability / adaptability (URS5)

As part of the User Requirement Specification, it is desired to have a certain level of flexibility in the system which allows subsequent developments of this system to be changed without major alterations on a hardware level. From a simple high-level view, an embedded system for control applications typically resembles the following dataflow: data is acquired by various sensors, a microprocessor processes this data and then determines appropriate output commands to drive actuators. To realize complete interchangeability between these elements, a component is required to hold the system together and provide the necessary interfaces between them. FPGA's provide this flexibility as they can be reconfigured to interact with different components. Figure 3.3 shows the high level architecture for such a system.

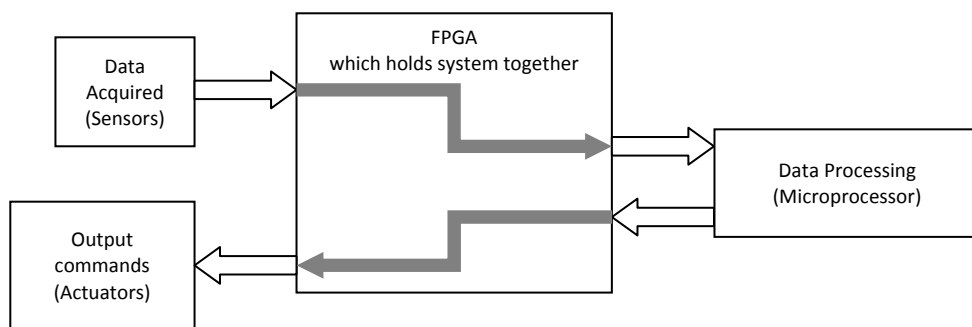
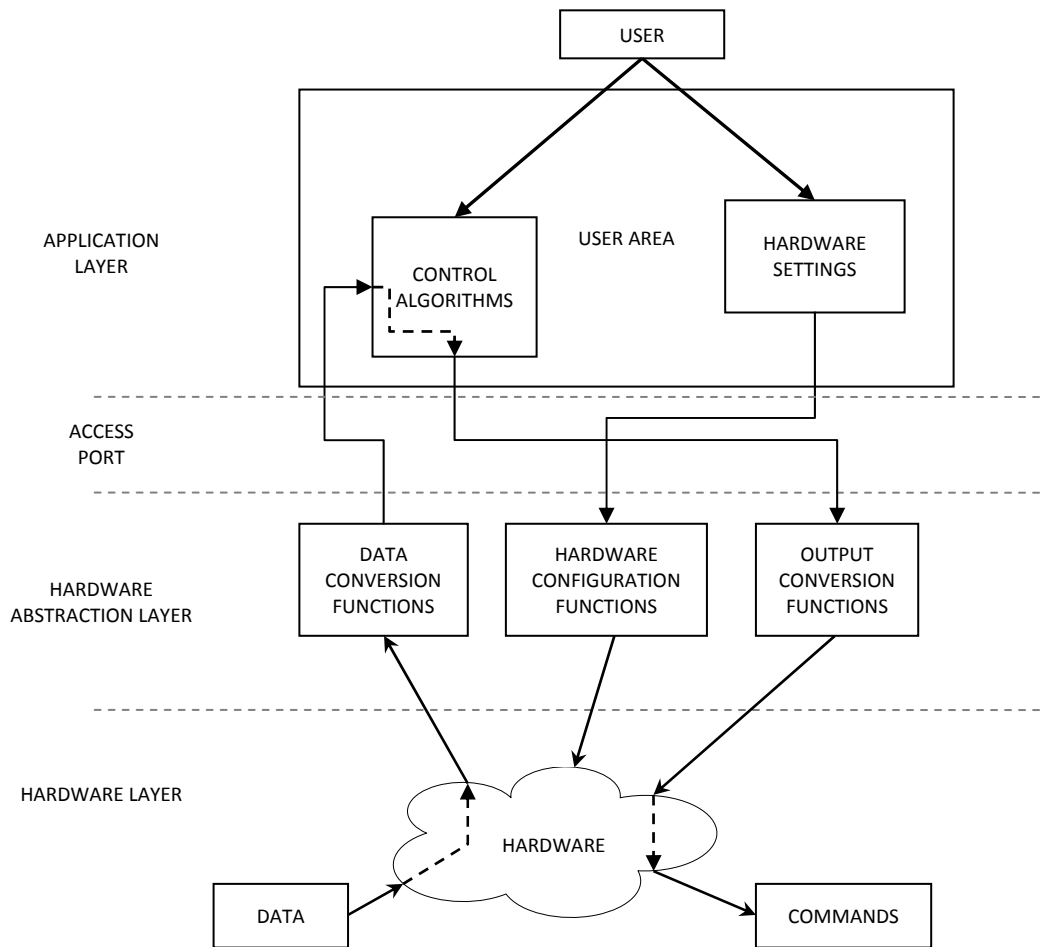


Figure 3.3 - Conceptualization of the interchangeability requirement (URS5)

### Reconfigurability (URS8) and Abstraction Layers (URS9)

These two requirements are inherently connected for this project. The user must be able to change or reconfigure the control system onboard while being abstracted from the low-level

hardware layer. The user simply has access to a port through which data is received in a defined form and must be sent in a certain form, thus creating a user area.



**Figure 3.4 - Conceptual overview of Abstraction Layers and Reconfigurability**

At the top-level of the abstraction layers is the application layer. This is the environment in which the user has access to the user area where control algorithms are loaded. The data appears through a port in a structure and form which is familiar to the user. The data is then processed by the control algorithms and the outputted commands are sent back through the port in a format which is also familiar to the user. In the user area, certain settings can also be changed. These include settings such as configurable parameters for certain hardware, such as sensors for example, that may be customizable.

The next layer is the hardware abstraction layer. This is where all the functions exist to abstract the user from the hardware. It is ultimately the link between the user and the hardware and provides the user with high-level access to the hardware layer.

The lowest layer of the system is the physical hardware layer and this is essentially where data is collected and commands are executed.

### 3.2 HIGH LEVEL SYSTEM OVERVIEW

Having performed a system conceptualization based on certain URS items in isolation, the concepts and ideas are combined to formulate the system architecture, shown in Figure 3.5.

The system is centered around an FPGA which, as mentioned earlier, provides large flexibility to the system. It creates a platform for any components to be selected and added to the system as interfaces for them can be built within the FPGA and thereby also realizes the interchangeability requirement of the URS. Hardware components are constantly being developed and the use of an FPGA allows these to be integrated into the system in future projects. This, however, comes at a certain cost; the schematic and PCB design would have to be changed in order to accommodate new components. The use of an FPGA also allows an amount of freedom during the component selection phase as most interfaces to components can be created in firmware.

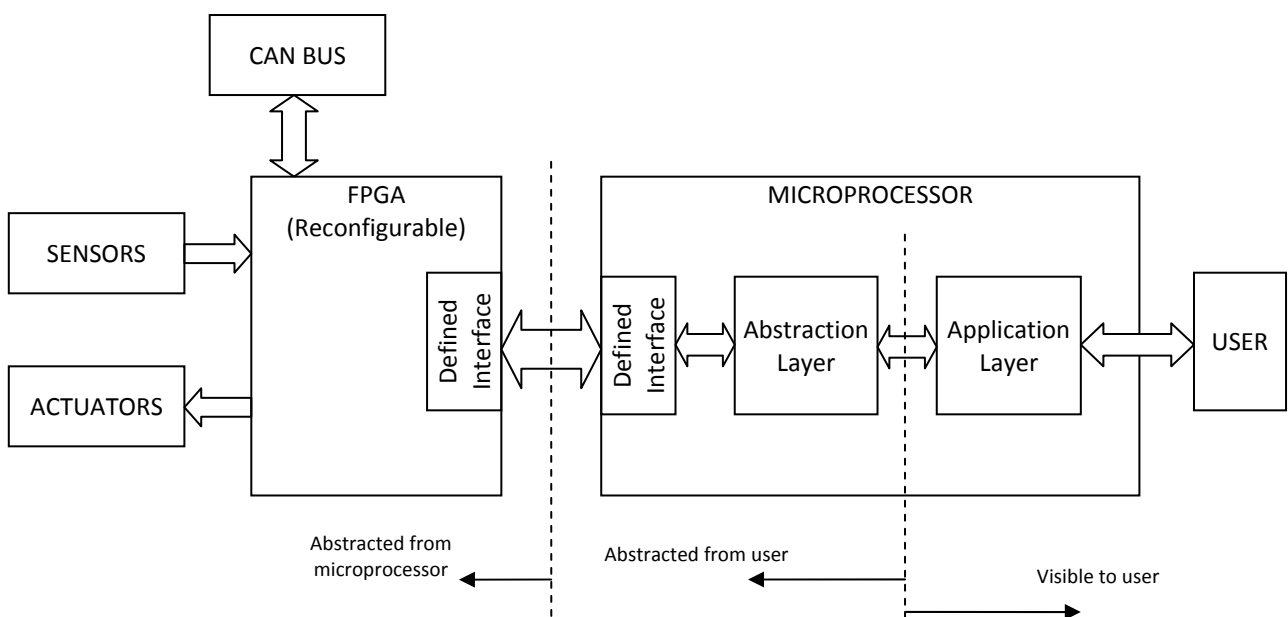


Figure 3.5 - High level overview of the system architecture

From Figure 3.5 above, certain levels of abstraction can be observed. The user is abstracted from the lower hardware level and only receives high level information in a user friendly format for use in the application layer. Furthermore, the FPGA, on the lowest level, is abstracted from the microprocessor. It only “sees” a block of addressable data/memory which is visible to it through a defined interface. The interchangeability of microprocessors can thus be realized if it adheres to

the interface requirements. The FPGA is reconfigurable and thus certain changes can also be implemented to accommodate the new microprocessor, if needed.

The microprocessor is mainly tasked with executing the control algorithms. This separates the control application from the data collection. Modules within the FPGA can operate simultaneously and this therefore speeds up data collection (note, the microprocessor would collect the data sequentially which would thus take a longer amount of time).

### **3.3 CONCEPTUALIZATION BASED ON SYSTEM REQUIREMENTS SPECIFICATION ANALYSIS**

Having conceptualized the system in response to the URS, the lower level system architecture concept can be formulated. This is driven by, firstly, the above conceptualization derived from the URS and, secondly, an analysis of the SRS. The final system architecture is shown in Figure 3.6 below.

It is shown in the high level system overview above, as well as in the microprocessor peripheral requirement specification, that a common and defined interface is required between the microprocessor and the FPGA. Since a fairly large amount of data is required to be transferred, it was decided to use a parallel bus for this. In the current avionics systems in the ESL, data is resolved into 16-bit values. It was subsequently decided to implement a 16-bit wide parallel bus with a minimum data rate of 10MHz.

The FPGA is also required to interface with a number of hardware modules stated in the SRS. This includes a PWM interface to drive up to eight servos in parallel. An interface to capture PWM signals from a standard eight channel RC receiver must also be provided for. Further interfaces required are the CAN, USB and SD-card interface. In addition to data logged, it was decided to allow the user to configure parameters of any reconfigurable hardware by loading them onto the SD-card. This abstracts the user from having to reprogram the FPGA each time new hardware configurations are desired.

The required sensors listed in the SRS must also be connected to the FPGA. These include, amongst others, a static pressure sensor, a differential pressure sensor and a GPS sensor. An inertial measurement unit (IMU) with three angular rate gyroscopes and three accelerometers, measuring the angular rates and accelerations in the axis system previously shown in Figure 2.2, is also required. Furthermore, a magnetometer is required. It was decided to use the onboard CAN

interface to enable the connection to the existing CAN magnetometer node. A temperature and current sensor are also integrated into the hardware design.

It was decided to place the RF module required for communication with the ground station on the microprocessor side of the system. This is done because all the data to be sent over this link to the ground station is of a high level nature i.e. it is data relevant to the application layer of the system. It would thus make more sense to interface the RF module with the microprocessor, which has all this data available.

Within the FPGA there are interface blocks and controller blocks. These handle all the interface specifications and protocols for the hardware, as well as controlling their functionality. This will be discussed in more detail in Chapter 5. Data in the FPGA is stored in a block of memory registers, each with a register address mapped to it. This can be accessed by the microprocessor over the parallel bus and the bus interface handles the protocol specifications and control thereof. As stated earlier, the microprocessor is thus abstracted from all the hardware connected to the FPGA and essentially, only sees a block of memory on the other side of the parallel bus. The microprocessor can thus be substituted with another one as long as it conforms to the parallel bus specification, as stated in Chapter 2.

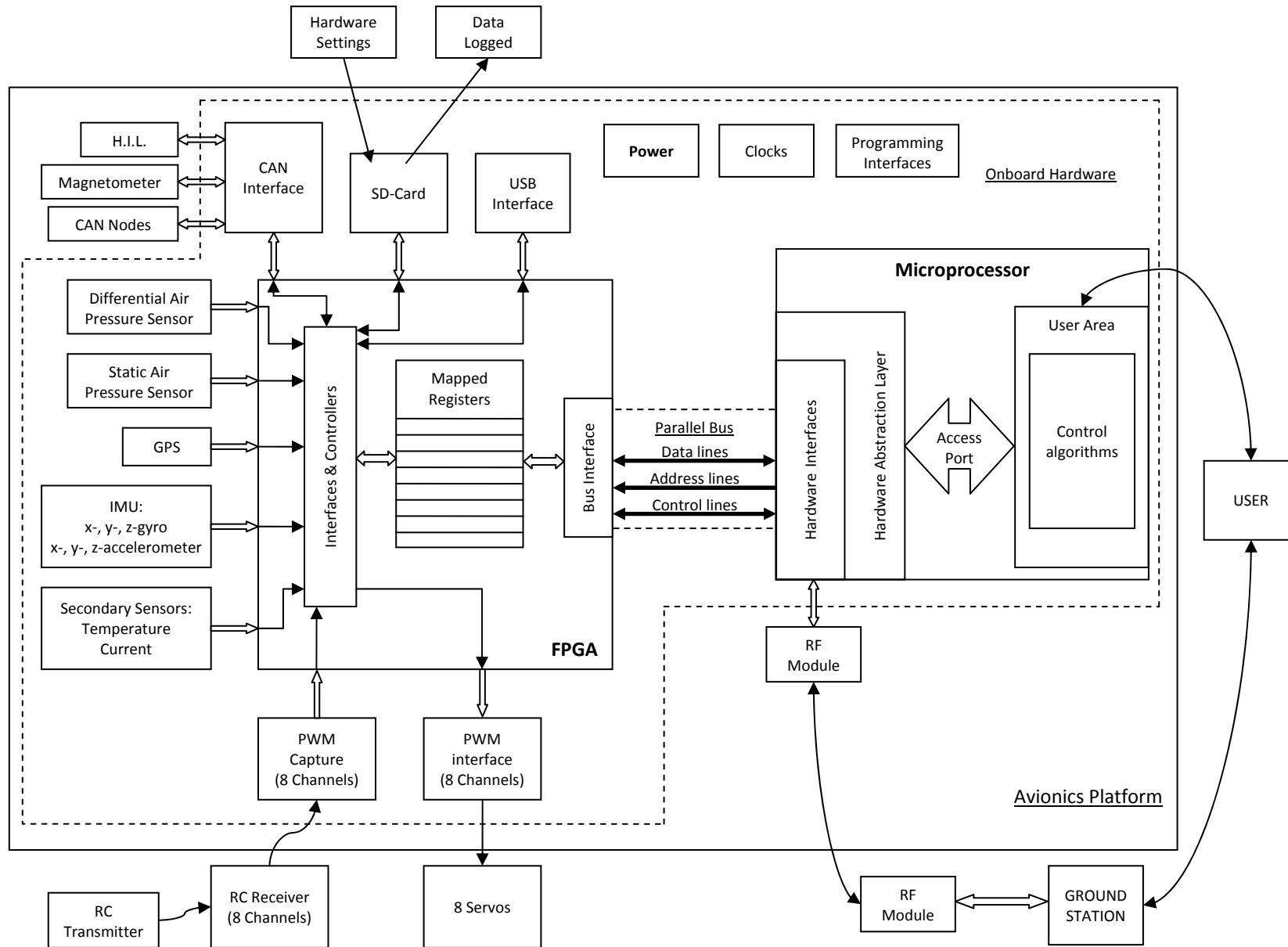
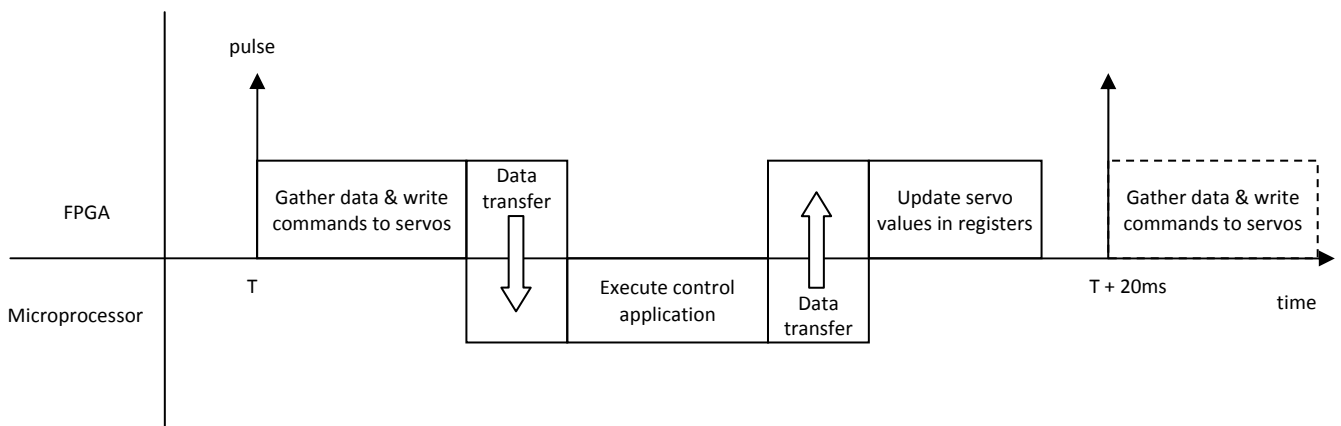


Figure 3.6 - Architecture and overview of the system

Another important consideration in conceptualizing the system is its timing and data flow. The strategy behind the timing of the system is centered on maintaining backward-compatibility with the nodes in the current CAN bus structure and its procedures [9]. This requires the FPGA to control the system timing and provide a timing pulse to synchronize the system. Sensor and RC pulse data is then gathered by the FPGA and the servo commands outputted based upon the commands from the previous cycle. The gathered data is then sent to the microprocessor, which executes the user application functions and control. The microprocessor then sends actuator commands back to the FPGA which subsequently updates the servos on the next synchronization pulse. This process is repeated every 20ms. A high-level overview of the timing functionality and data flow in the system is shown in the figure below.



**Figure 3.7 - High-level overview of the system timing and data flow**

This chapter established the architecture and conceptual functioning of the system in response to the User Requirements Specification and System Requirements Specification. This creates the platform for the detailed design phases to be commenced and this will be discussed in the chapters which follow.

## 4. DETAILED HARDWARE DESIGN

In this chapter, all the design details concerning the hardware design will be discussed. This includes component choices, schematic design and the printed circuit board layout.

### 4.1 COMPONENT CHOICES AND CONSIDERATIONS

The URS, SRS and Conceptual Design were used to guide the component selection process. In order to make the design as compact as possible in terms of component population and PCB routing, surface mount components were chosen whenever possible.

#### 4.1.1 Microprocessor

An extensive search was conducted to find a suitable microprocessor for this project based upon the system requirements specification (SRS), as well as the conceptual design set out in the previous chapters (i.e. an FPU with a minimum of 25 MFLOPS processing capability, a parallel bus that is at least 16-bits wide with a minimum data rate of 10 MHz and a UART peripheral port). The datasheets of microprocessors were studied in detail, as well as similar embedded hardware projects, in order to find candidate microprocessors. A number of these are tabulated below with selected specifications.

A few of the parameters listed in Table 4.1 are discussed below:

- **Instruction Throughput:** this is a measure of how many instructions the processor can perform in a given time. It is usually measured in Million Instructions Per Second (MIPS) and is a function of the core speed and architecture (e.g. parallel instruction execution, namely, superscalar architecture).
- **FPU:** indicates whether or not the processor has an integrated Floating-Point Unit (FPU) onboard and the type of floating-point numbers it supports.
- **FLOPS:** a measure of the number of floating point operations per second (FLOPS) the processor can perform.

*(continues after Table 4.1)*



**Table 4.1 - Microprocessors considered for this project and their selected specifications (found in manufacturers' datasheets)**

Parameter	Renesas SH7201 (CPU1)	Renesas SH7203 (CPU2)	Renesas SH7760 (CPU3)	Analog Devices TigerSHARC ADSP-TS201S (CPU4)	Freescall MPC555 (CPU5)	Texas Instruments TMS320C6713B-167 (CPU6)	Analog Devices SHARC ADSP-21065L (CPU7)	Intel PXA255 (CPU8)	Atmel AT572D740 (CPU9)
<b>Architecture</b>	32-bit RISC SH2A superscalar(2x)	32-bit RISC SH2A superscalar(2x)	32-bit RISC SH4 superscalar	32-bit VLIW & superscalar(4x) DSP	32-bit PowerPC & superscalar	32-bit VLIW & superscalar(8x) DSP	32-bit Harvard DSP	32-bit ARM core	Dual core: 32-bit ARM7TDMI RISC & VLIW DSP
<b>Core Speed</b>	120 MHz	200 MHz	200 MHz	600 MHz	40 MHz	167 MHz	66 MHz	400 MHz	DSP core: 100MHz & ARM7TDMI core: 50MHz
<b>Instruction Throughput</b>	288 MIPS (2.4MIPS/MHz)	480 MIPS (2.4MIPS/MHz)	360 MIPS	2400 MIPS	52.7 MIPS (Dhrystone 2.1)	1336 MIPS	66 MIPS	480 MIPS (Dhrystone 2.1)	- 1.5GOPS for DSP core - Not Available for ARM7TDMI core (Assume 50MIPS)
<b>FPU</b>	YES Single- & double-precision supported	YES Single- & double-precision supported	YES Single- & double-precision supported	YES Single- & extended-precision supported	YES Single- & double-precision supported	YES Single- & double-precision supported	YES Single- & extended-precision supported	NO	YES Single- & extended-precision supported
<b>FLOPS</b>	200 MFLOPS	200 MFLOPS	1.4 GFLOPS	3.6 GFLOPS	Not Available 40 MFLOPS (Assumption)	1 GFLOPS	198 MFLOPS	Poor due to no FPU	1 GFLOPS
<b>External Interface/ Parallel bus</b>	32-bit data @ 60MHz	32-bit data @ 66.67MHz	32-bit data @ 67MHz	64-bit data @ 1GB/s	32-bit data @ 40MHz	16-bit data @ 100MHz	32-bit data @ 33 MHz	32-bit data @ 100 MHz	16-bit @ 25 MHz
<b>Approximate Maximum Power Usage</b>	180mA x 3.3V @ 120MHz ≈ 0.594W	400mA(max) x 1.2V(core) @ 200MHz ≈ 0.480W	730mA (max) x 1.5V + 190mA(max) x 3.3V @ 200MHz ≈ 1.722W	≈ 4.609W (max) @ 600MHz	≈ 1.091W (max) @ 40MHz	≈ 0.8235W @ 167MHz	3.6V x 510mA (max) ≈ 1.836W(max) @ 66MHz	≈ 2.598W @ 400MHz	≈ 2.5W @ full core speeds
<b>FLOPS/Watt<sup>2</sup> (Normalized: 0 to 1)</b>	0.384	0.589	0.320	0.115	0.023	1	0.040	N/A	0.109
<b>Size</b>	176-pin QFP package 24 x 24 mm	240-pin QFP package 32mm x 32mm	256-pin BGA package 21mm x 21mm	576-pin BGA package 25mm x 25mm	272-pin BGA package 27mm x 27mm	208-pin QFP package 28mm x 28mm	196-pin BGA package 15mm x 15mm	256-pin BGA package 17mm x 17mm	352-pin BGA package 35mm x 35mm
<b>Qualitative Measure of Hardware Complexity</b>	Low	Low	Medium	High	Low	High	Medium	Medium	High

(Parameter explanation continued)

- **FLOPS/(Watt<sup>2</sup>):** This is a simple quantity to reflect how expensive it is in terms of power in order to gain FLOPS performance. All the processors subjected to this calculation would already comply with the FLOPS requirement and thus the power is squared in the equation to increase its weight in the analysis. The final values are normalized between 0 and 1.
- **Qualitative Measure of Hardware Complexity:** A qualitative measure of the complexity of each processor was determined by studying the respective data sheets and design guides. This also factors in the availability of hardware development support and tools. Further factors taken into account, amongst others, were PCB layout in terms of device packaging (e.g. BGA packages require more complex PCB layouts) and whether the device has onboard programming memory or not. The microprocessors in Table 4.1 were weighed up against each other and given a rating in terms of their complexity in relation to each other, with a low rating being more favourable.

### Selection Process & Traceability:

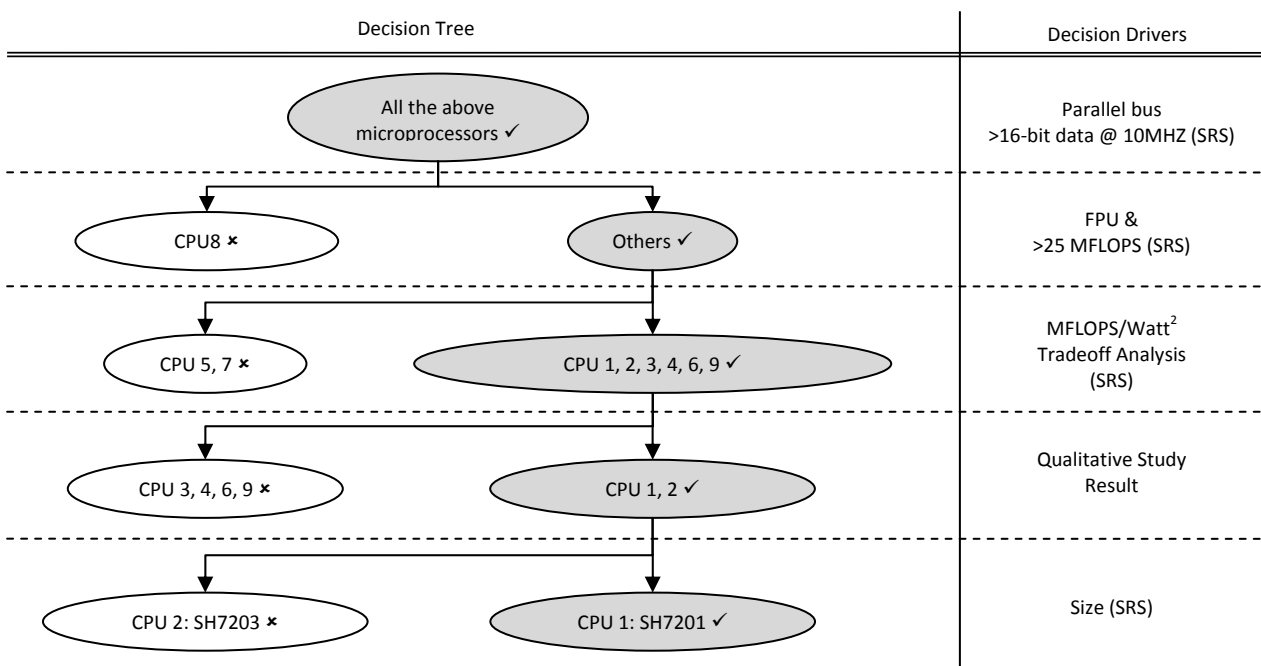


Figure 4.1 - Diagram illustrating the decision-making process for the microprocessor

The microprocessor chosen for this project is the **Renesas SH7201 (CPU1)** as shown in Figure 4.1 above. This processor complies with all the System Requirement Specifications set out in Chapter 2. However, it is a ROMless device and therefore does not have built-in FLASH program memory.

In order to aid the hardware and firmware design process, a development kit (called the Renesas Starter Kit for SH7201) for the device was purchased. It allows access to all the peripherals and features of the SH7201. It was also decided to develop the hardware system in two stages: First, a prototype board would be developed which is connected to the microprocessor development board via a ribbon cable and secondly, the microprocessor would then be integrated onboard and all necessary hardware updates would be implemented (time allowing).

The SH7201 development kit (RSK) purchased included an E8 emulator for programming and debugging. This works in conjunction with the RSK development board and a bootloader preloaded by the manufacturer into external flash on the board. The E8 alone is thus not suitable for developing one's own standalone board and thus an E10A programmer/debugger, or a similar third-party tool, is required to program external flash memory via the SH7201 microprocessor. This tool is however very expensive. Thus the strategy of first creating a prototype system which is connected to the RSK board creates a platform for testing and benchmarking the processor first and the feasibility of purchasing the E10A tool can be evaluated. Thereafter a standalone system with the microprocessor incorporated onboard could be developed.

#### **Recommendations and Alternatives:**

The advancement of microprocessors increases significantly over short periods of time. New processors are thus constantly entering the market. The following are such examples, which were not available at the time of hardware development, and should be considered in future developments.

The Renesas SH7216 is a very similar microprocessor to the SH7201 and therefore also complies with the microprocessor requirements of this project. It is also based on the SH2A architecture with FPU. It offers similar performance and has the added advantage of built-in FLASH program memory, thus simplifying the hardware design process somewhat.

For avionics applications requiring less processing power, the PIC32 range of microprocessors from Microchip should be considered. These have no FPU unit though and are thus only recommended for algorithms which are not floating-point intensive. A parallel bus is available among its peripherals for integration into the system.

## 4.1.2 Sensors

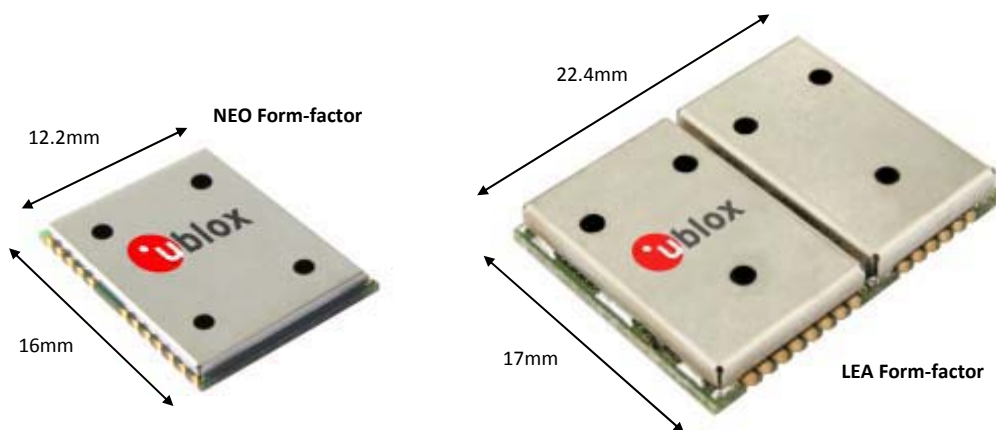
### 4.1.2.1 GPS

The current avionics hardware systems in the ESL use GPS receiver boards which plug into the system. These boards contain a GPS sensor module on them as well as all its supporting hardware. However, as this system is required to be compact in size, a decision was made to integrate the GPS sensor module onboard.

The GPS sensor modules currently used in the ESL belong to the *u-blox* ANTARIS-4 family. Various other GPS sensor modules were investigated for use in this project but it was decided to find a GPS sensor within this family of modules as support is readily available and these modules are also familiar within the ESL.

The ANTARIS-4 series of GPS modules have a navigation update rate of 4Hz (among the faster update rates available for the purposes of this project) with a 16 Channel GPS engine. They have relatively low power consumption and operate from 2.7V to 3.3V supply voltages. They also support the DGPS, WAAS, EGNOS and MSAS augmentation systems.

There are basically three different form-factor sizes within this range of GPS modules. These are the TIM form-factor (25.4mm x 25.4mm x 3mm), the LEA form-factor (17mm x 22.4mm x 3mm) and the NEO form-factor (12.2mm x 16mm x 2.8mm). The main differences between these form-factors, other than the size, are the amount of available pins for communication and configuration [14]. This difference, however, is inconsequential in this project as all contain at least one port for communication as required.



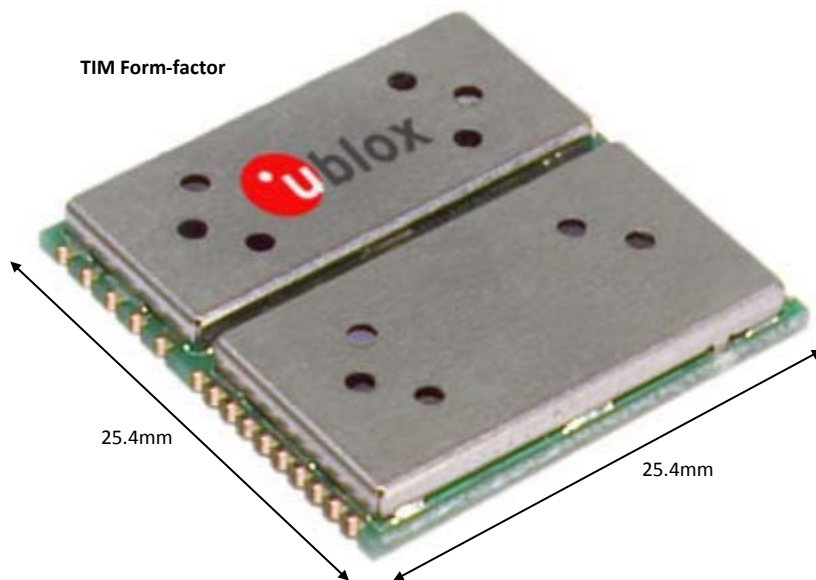


Figure 4.2 - Form-factors of u-blox ANTARIS4 GPS modules

Furthermore, within the different form-factors, each can be divided into two more groups: Firstly, a lower cost ROM-based module which has limitations in terms of configuration and firmware, and secondly, a FLASH memory-based programmable module which allows configuration settings to be permanently stored within the flash memory. The NEO form factor, however, is only available in one ROM-based module.

The ROM-based modules are not desirable for this project and were thus not considered. Only modules of the smaller LEA form-factor with configurable FLASH-based architecture were considered for use in this project and are listed in the table below [14] & [15]. The **LEA-4H** module was chosen as it provides a good balance between GPS signal sensitivity, cost-effectiveness and power consumption.

Table 4.2 - List of u-blox LEA-4x GPS modules considered in this project.

Module	SuperSense <sup>1</sup>	Power Usage <sup>2</sup>	Cost <sup>3</sup>
LEA-4H	✓	0.126W	R377.57
LEA-4P	x	0.117W	R375.52
LEA-4R	x	0.153W	Not Available
LEA-4T	✓	0.126W	R1177.85

**1) SuperSense** – This is a u-blox proprietary technology which provides better sensitivity and is useful when GPS signals are weak.  
**2) Power Usage** – This is calculated using the sustained supply current in Table 17 of the datasheet [15] and adding the additional current (3mA for 4Hz update rate) in Table 11 of System Integration Manual [14].  
**3) Cost** – Supplier: RF Design [16].

**Selection & Traceability:**

In order to justify the selection of the LEA-4H GPS module, the decision-making process is shown in the diagram below and traced back to the User Requirements Specification and the System Requirements Specification.

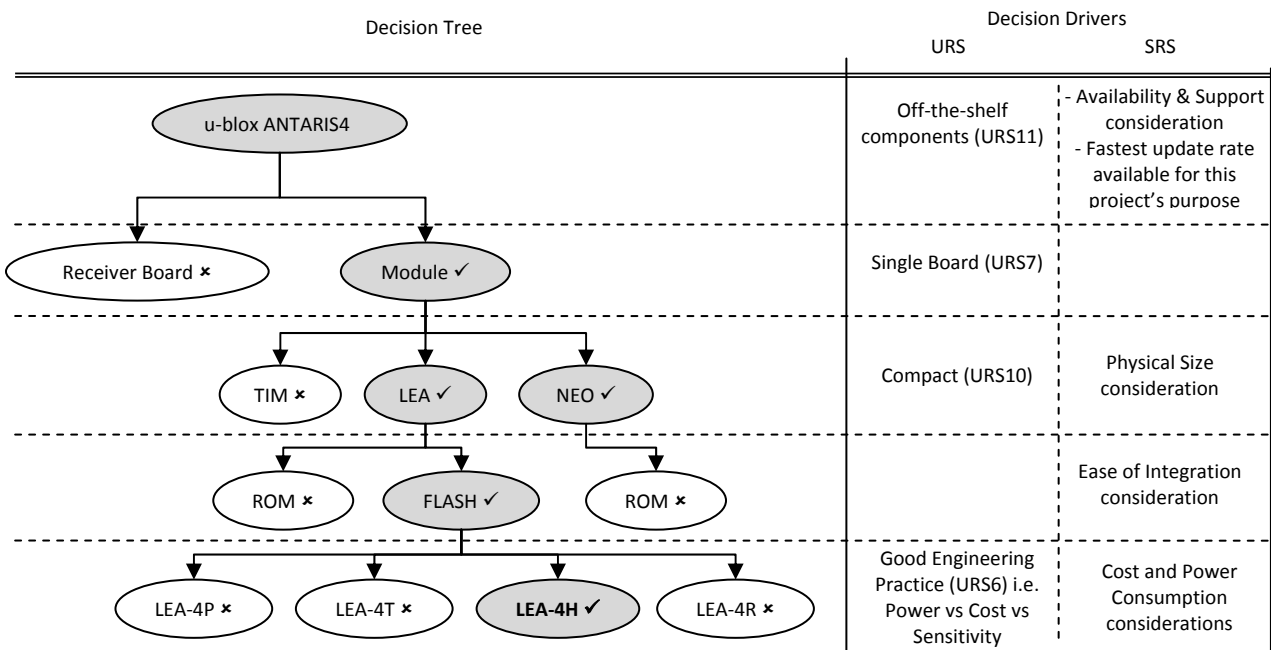


Figure 4.3 - Diagram illustrating the decision-making process for the GPS sensor

The decision to choose the FLASH-based modules over the ROM-based modules is classified as an 'Ease of Integration' consideration. This is due to the fact that the ROM-based modules require different pin configurations in hardware for different settings, whereas the FLASH-based modules can easily be reprogrammed via the communication port to be used.

**Alternatives:**

For future designs, newer alternative GPS modules could be considered which were not available at the time of the design phase of this project. Two such modules from u-blox are the LEA-5H and newer LEA-6H modules, which allow easy hardware and firmware migration from the LEA-4H. These are 50-channel GPS devices thus allowing connection to more satellites than the LEA-4H. It also provides support for the GALILEO satellite system being developed. They however only have a 2Hz update rate and are only recommended if this bandwidth is sufficient. Both modules are currently less expensive than the LEA-4H.

#### 4.1.2.2. Inertial Measurement Unit (IMU)

In the System Requirements Specification, it is shown that three angular rate gyroscopes and three accelerometers are required to measure angular rates and accelerations as shown in Figure 2.2. The selection process for the gyroscopes and accelerometers commenced by investigating two possible hardware configurations, as shown in Figure 4.4 below.

The first shows a gyroscope and accelerometer pair placed on two individual boards and mounted orthogonally to each other as well as the main PCB. The third gyroscope/accelerometer pair would be mounted on the main PCB itself.

The second, and more ideal solution, is a fully integrated tri-axis IMU which allows one to interface through some or other port or hardware pins, and this can be placed directly on the PCB. The latter configuration is less complex in terms of hardware mounting and integration, and also has better orthogonal alignment, thus providing more accurate measurements. It also takes up less physical hardware space.

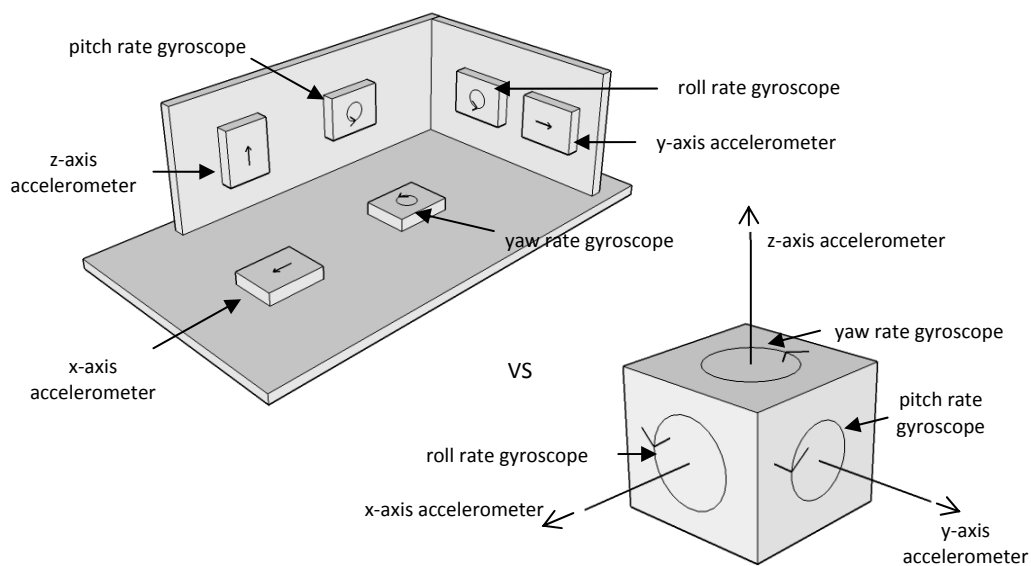


Figure 4.4 - IMU configurations

It was decided to implement the second configuration above and a search was conducted to find a fully integrated tri-axis IMU. Three possible candidates were found: the AccelRate3D from MEMSense, the ADIS 16350 from Analog Devices as well as the ADIS 16355 also from Analog Devices. The next step was to verify that these sensors comply with the requirements of the SRS, and then to choose the most suitable candidate. The relevant specifications of each IMU sensor were extracted from their respective datasheets and are tabulated below.

Table 4.3 - Specifications of six-degree-of-freedom IMU candidates

	Specification	MEMSense AccelRate3D AR10-0300S050	Analog Devices ADIS 16350	Analog Devices ADIS 16355
	Output Type	Analog	Digital (SPI)	Digital (SPI)
GYRO SPECIFICATIONS per axis	Dynamic Range	$\pm 300^\circ/s$	$\pm 300^\circ/s$ ; $\pm 150^\circ/s$ ; $\pm 75^\circ/s$ User selectable	$\pm 300^\circ/s$ ; $\pm 150^\circ/s$ ; $\pm 75^\circ/s$ User selectable
	Bias Instability	-	$0.015^\circ/s$ ( $1\sigma, 25^\circ C$ )	$0.015^\circ/s$ ( $1\sigma, 25^\circ C$ )
	Bias Temperature Coefficient	-	$0.1^\circ/s/^\circ C$	$0.01^\circ/s/^\circ C$
	Angular Random Walk	Not Available	$4.2^\circ/\sqrt{hr}$	$4.2^\circ/\sqrt{hr}$
	Nonlinearity	$\pm 0.1\%$ of FS	$\pm 0.1\%$ of FS	$\pm 0.1\%$ of FS
	Noise Density	$0.1^\circ/s/\sqrt{Hz}$	$0.05^\circ/s/\sqrt{Hz}$ rms	$0.05^\circ/s/\sqrt{Hz}$ rms
	Bandwidth	50Hz	350Hz	350Hz
	Sensitivity	$5mV/^\circ/s$	$0.07326^\circ/s/LSB$ ( $\pm 300^\circ/s$ ) $0.03663^\circ/s/LSB$ ( $\pm 150^\circ/s$ ) $0.01832^\circ/s/LSB$ ( $\pm 75^\circ/s$ )	$0.07326^\circ/s/LSB$ ( $\pm 300^\circ/s$ ) $0.03663^\circ/s/LSB$ ( $\pm 150^\circ/s$ ) $0.01832^\circ/s/LSB$ ( $\pm 75^\circ/s$ )
ACCELEROMETER SPECIFICATIONS per axis	Dynamic Range	$\pm 10g$	$\pm 10g$	$\pm 10g$
	Bias Instability	-	$0.7mg$ ( $1\sigma, 25^\circ C$ )	$0.7mg$ ( $1\sigma, 25^\circ C$ )
	Velocity Random Walk	-	$2.0 m/s/\sqrt{hr}$	$2.0 m/s/\sqrt{hr}$
	Bias Temperature Coefficient	-	$4 mg/^\circ C$	$0.5 mg/^\circ C$
	Sensitivity	$400mV/g$	$2.522mg/LSB$	$2.522mg/LSB$
	Bandwidth	50Hz	350Hz	350Hz
	Noise Density	$35\mu g/\sqrt{Hz}$ (x & y axis) $65\mu g/\sqrt{Hz}$ (z axis)	$1.85mg/\sqrt{Hz}$ rms	$1.85mg/\sqrt{Hz}$ rms
	Nonlinearity	Typically: $\pm 0.4\%$ of FS Maximum: $\pm 1.0\%$ of FS	$\pm 0.2\%$ of FS	$\pm 0.2\%$ of FS
	Cost (at selection)	-	\$275	\$359

Inertial navigation sensors (i.e. gyroscopes and accelerometers) suffer from certain temperature and noise effects which lead to navigation errors. It is thus essential to understand and analyze these noise effects. A mathematical or statistical technique known as Allan Variance can be used to evaluate gyroscope and accelerometer performance. It is especially useful to analyze the effect noise has on integrating inertial sensor outputs i.e. integrating angular rate and acceleration to obtain orientation and position, respectively. The actual method of this technique is beyond the scope of this thesis. The results of the analysis describe the following important characteristics of the inertial sensors [17], [18], [19]:



- Gyroscope Angular Random Walk (ARW): This is an indication of how the noise (modeled as white noise) on the output signal of a gyroscope affects the angle measurement (i.e. the integration of the angular rate signal) and is given in  $^{\circ}/\sqrt{hr}$ . For the ADIS IMUs above, the ARW of  $4.2^{\circ}/\sqrt{hr}$  gives a standard deviation of the integration result (angular) error of  $4.2^{\circ}$  after 1 hour integration time,  $\sqrt{2} \times 4.2^{\circ} = 5.93^{\circ}$  after 2 hours integration time based on the outline of [18]. For shorter integration times typical to aircraft systems, one could convert  $^{\circ}/\sqrt{hr}$  to  $^{\circ}/\sqrt{sec}$  by dividing by  $\sqrt{3600}$ , or 60. This reveals an ARW of  $0.07^{\circ}/\sqrt{sec}$  for the ADIS devices.
- Gyroscope Bias Instability: This is an indication of how the bias of a gyroscope fluctuates over time due to the presence of flicker noise in electronics, observed at low frequencies. It is usually modeled as a random walk [18]. Integrating this causes a second-order random walk in the angular measurement. The measurement given by manufactures is typically given for a 100s time period with  $1\sigma$  standard deviation with units  $^{\circ}/s$  (under constant conditions) as explained in [18]. Therefore the  $0.015^{\circ}/s$  ( $1\sigma, 25^{\circ}C$ ) value for the ADIS gyroscopes means that the expected bias value after 100s has a standard deviation of  $0.015^{\circ}/s$  i.e. 68% of the time the deviation is expected to be within  $0.015^{\circ}/s$ . This is a better indication of instability over short periods of time because biases are usually constrained within a certain range for longer periods of time. For more information see [18].
- Accelerometer Velocity Random Walk (VRW): Similarly to ARW for a gyroscope, this is a measure of the effect noise has on integrating the output of an accelerometer (to calculate velocity) and is given in  $m/s/\sqrt{hr}$ .
- Accelerometer Bias Instability: This is similar to the bias instability measurement of a gyroscope explained above and is given in  $m/s$ . Integration causes “a second order bias random walk in velocity” and “a third order random walk in position” [18].

Further specifications include:

- Noise Density: This is a measure of the “output noise” variation of the sensor as a “function of the bandwidth” [19]. As manufacturers give noise specifications in different ways, it can be calculated from the ARW with the following formula [18], [19]:

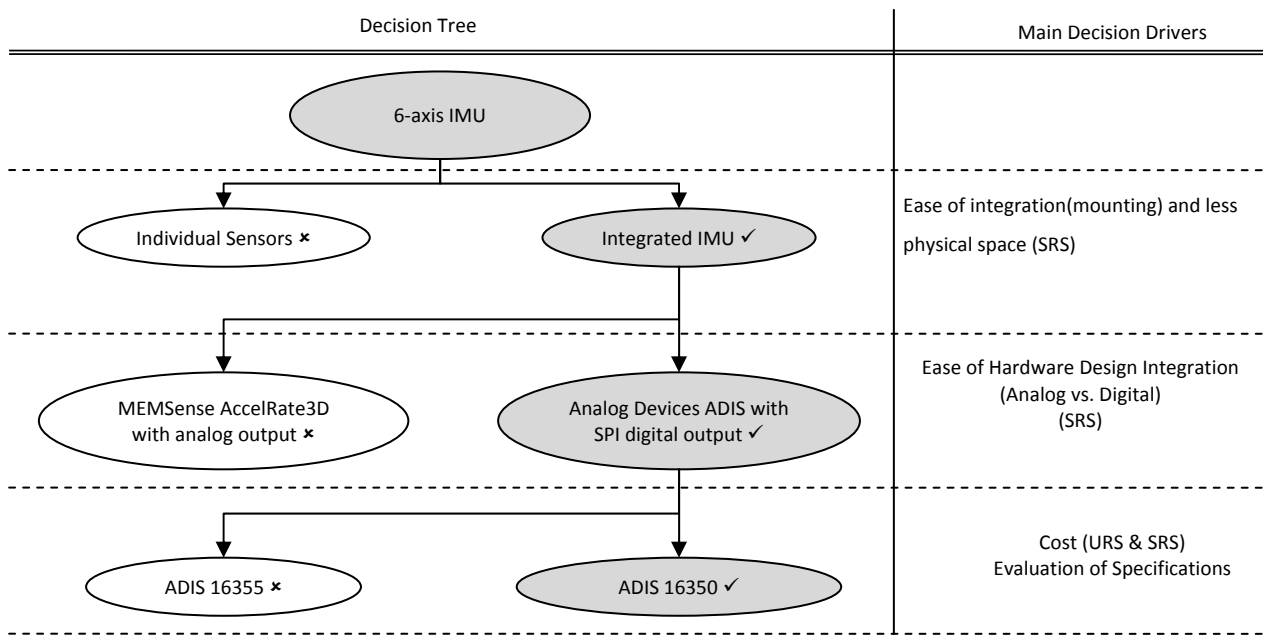
$$Gyro: ARW \text{ in } ^{\circ}/\sqrt{hr} = \frac{1}{60} \times FFT[Noise \text{ Density in } ^{\circ}/hr/\sqrt{Hz}]$$

- Temperature Bias Drift: Inertial sensors suffer from bias errors due to environmentally-induced and self-induced temperature changes. This is usually a linear increase and is given as the temperature coefficient in the table above.
- Nonlinearity: This is a measure of how close to linear the output of the sensor is with respect to the measured angular rate or acceleration. It is expressed as a percentage error of the linear full-scale range [20].

Both the MEMSense and ADIS sensors comply with the requirement of  $\pm 300^\circ/\text{s}$  (gyro) and 10g (accelerometer) set out in the System Requirements Specification. The ADIS Analog Devices IMUs were preferred over the MEMSense due to the fact that they provide an SPI digital interface. This allows less complex integration with an FPGA as no further ADC devices are required to digitize and condition the signals.

The ADIS devices have built-in digitally configurable filters (Bartlett Window FIR filter) for reducing noise and therefore no additional signal conditioning circuitry is required. The ADIS devices have a built-in temperature sensor and digitally controllable bias calibration settings which help overcome bias drift. The sample rate can also be set digitally. The ADIS 16355 has higher calibration precision than the ADIS 16350, but is more expensive. As lower cost is a consideration established in the requirements, the ADIS 16350 device was selected and the bias errors can be somewhat corrected in software. The gyro dynamic range can also be set as seen in the table above which deliver different respective sensitivities.

**Selection and Traceability:**



**Figure 4.5 - Selection process overview of the IMU Sensor**

**Recommendations for future designs:**

The ADIS 16350 device has become obsolete during the duration of this project and thus it is recommended that a newer IMU device be used in future designs. There are newer IMUs in the ADIS family offered by Analog Devices which comply with the specifications of this project, namely, the ADIS 16360 and 16365 devices. These offer even better accuracy and noise performance than the ADIS 16350. A table of these device specifications is included in Appendix A. Also to be considered are the ADIS 16400 and 16405 6DOF IMU with integrated magnetometer.

**4.1.2.3. Pressure Sensors**

Two pressure sensors are required as discussed in the System Requirements Specification, namely, an absolute pressure and a differential pressure sensor. The sensors used in the current avionics systems in the ESL are the MPXV5004GP Differential Pressure sensor and the MPX4115A Absolute pressure sensor from Freescale Semiconductor. These sensors provide an analog output signals which must be conditioned and then digitized by an ADC device. However, in order to simplify the integration of the pressure sensors with the FPGA, a search for sensors with digital outputs was undertaken.

Two sensors which provide pressure readings via an I<sup>2</sup>C digital output were found. The HCA0611AR absolute pressure sensor from Sensortech provides a static pressure range from 60 to 110 kPa.

The HCLA0025EU differential pressure sensor, also from Sensortech, provides a dynamic pressure range from 0 to 2.5kPa.

The HCA0611AR sensor output provides a digital resolution of 1.9074 Pa per count and the HCLA0025EU sensor provides a digital resolution of 0.0954 Pa per count. These comply with the specifications determined in the SRS (i.e. a static pressure resolution less than 2.71 Pa per count and a differential pressure resolution of less than 3.06 Pa per count) and provide even better resolution accuracy.

**Traceability:**

The figure below justifies the decision process for the sensor selection by tracing it back to the specified requirements.

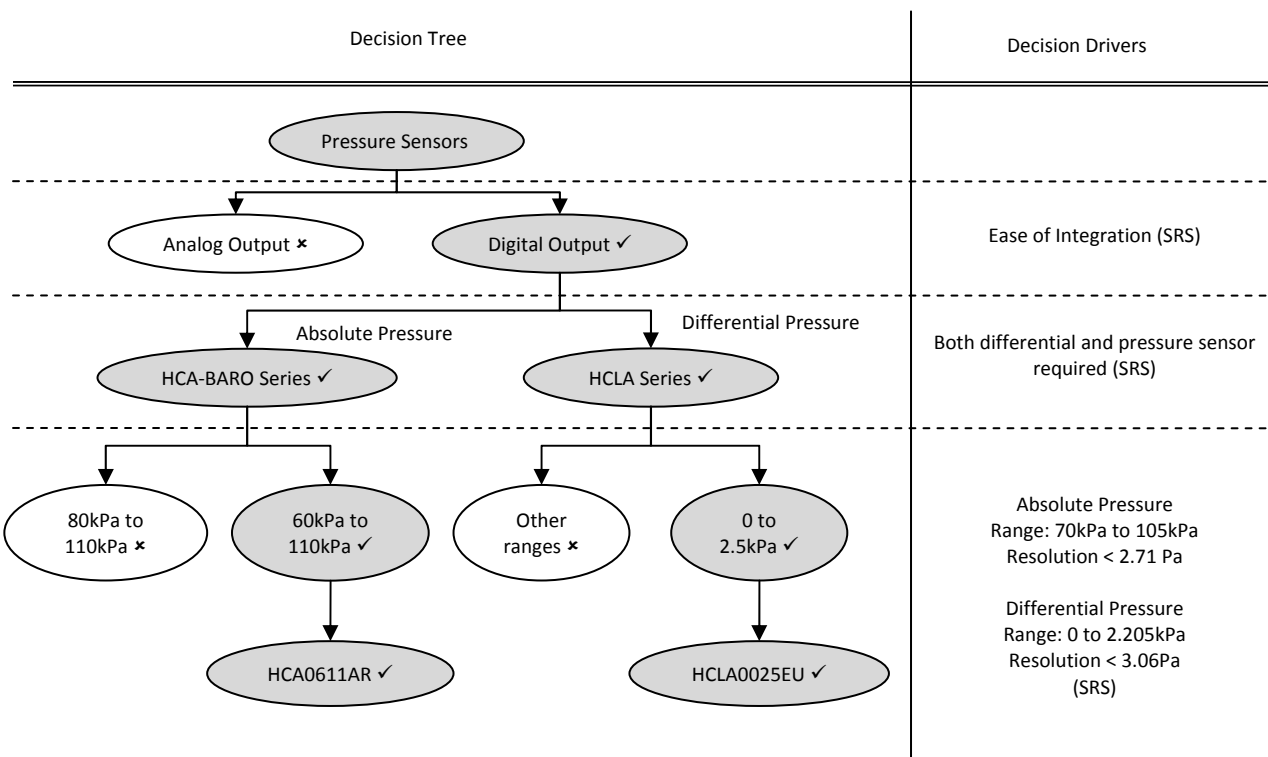


Figure 4.6 - Pressure sensors selection process

**4.1.2.4. Secondary Sensors: Temperature and Current**

In order to get an indication of the temperature of the PCB, the MAX6613 temperature sensor is used. The PCB will be subjected to an enclosed environment within an aircraft and thus the temperature sensor output will be the average temperature of the hardware PCB. The analog output of the sensor, found on the manufacturer’s datasheet, is fairly linear and is approximately given by:

$$Temp(^{\circ}C) = (1.8455 - V_{OUT}) \div 0.01123V$$

The total current consumption of the system is also an important factor, as stated in the SRS, and thus a current sensor is required. For this purpose, the LT6100 was chosen. It is a current sensing amplifier with selectable gain. Current from the source flows through a low-ohmic, high power current sensing resistor. This creates a sense voltage which is then amplified by the device to provide an accurate output voltage from which the current can be computed.

The above sensors will be discussed further during the schematic design.

### 4.1.3 SD-Card Interface

A Secure Digital memory card, or SD card, interface is required in the system to store flight data. SD cards support two fundamental modes of operation: SD Mode and SPI Mode. The SD Mode has two further sub-modes: a 1-bit protocol and a 4-bit protocol. [21]

The SD protocol specification is controlled by the SD Card Association and the full specification is only available under a purchased license. A simplified subset of this protocol specification is however freely available. SD Mode of operation requires the full specification while the slower SPI Mode allows a system to use the simpler, freely available specification protocol and implement a fully functioning SD card interface. This system will thus provide for the SPI mode.

### 4.1.4 USB and CAN Interfaces

As stated in the SRS, a USB and CAN interface is required. There are two possible ways of achieving this. The first is to implement the respective USB and CAN protocols within the FPGA. This is however very complex and time was not available for this implementation. The second way, implemented in this project, is to use a hardware device which handles the protocol and signal leveling and “converts” it to a simpler protocol i.e. SPI in this case. This also allows easier integration into the system (SRS).

The Maxim MAX3420E USB Peripheral Controller with SPI Interface was chosen to add USB capability to the system. It allows a full-speed USB (rev 2.0) to be implemented. The SPI interface can operate at frequencies up to 26MHz.

The Microchip MCP2515 Stand-alone CAN Controller with SPI Interface was selected to provide CAN (v2.0B) capabilities. It can transmit and receive standard and extended frames. The SPI interface can operate at speeds up to 10MHz. Together with this, the SN65HVD230 CAN

Transceiver from Texas Instruments was also chosen to handle the voltage leveling of the CAN signals.

#### 4.1.5 FPGA

The evaluation and selection of an FPGA was conducted last as its I/O pin requirements were dependent on all the components which were to interface with it. Other factors taken into account were the amount of logic elements, component size, speed grade, cost, availability and development support.

The first requirement for the FPGA selection was to determine the number of I/O pins required. The table below lists all the devices which interface with the FPGA, together with the protocols and number of pins required.

**Table 4.4 - Devices interfacing with the FPGA and number of pins required**

Device	Interface	Lines	FPGA I/O Pins Required
<b>SH7201 microprocessor</b>	Parallel Bus Interface	16 Data Lines 10 Address Lines 5 Control Lines 1 Clock Line	32
<b>LEA-4H GPS module</b>	UART	TX, RX	2
<b>ADIS 16350 IMU</b>	SPI	nCS, DIN, DOUT, SCLK	4
<b>HCA</b>	I <sup>2</sup> C	SDA, SCL	2
<b>HCLA</b>	I <sup>2</sup> C	SDA, SCL	2
<b>CAN</b>	SPI	nRESET, nCS, SO, SI, SCK, nINT	6
<b>USB</b>	SPI	nCS, MOSI, MISO, SCLK	4
<b>SERVOS</b>	PWM	CH0-7	8
<b>RC RECEIVER</b>	PWM	CH0-7	8
<b>SD-CARD</b>	SPI	nCS, MOSI, MISO, SCLK	4
<b>ADC</b>	SPI	nCS, DIN, DOUT, DCLK, BUSY	5
<b>DEBUG</b>	-	-	10 (minimum)
<b>TOTAL:</b>			<b>87</b>

An important factor in FPGA selection is determining the amount of Logic Elements required in implementing a system. An analysis was done for this project by investigating the VHDL modules to be implemented on the FPGA as well as investigating projects of a similar or larger size; [22], [4] and [23].

The EP2C8Q208C7 FPGA from the Altera Cyclone II family was subsequently selected. It has 8256 logic elements which were estimated as being sufficient for this project. The 208-pin PQFP package has 138 available user I/O pins, which is more than sufficient for this project. The selected device also has a speed grade of 7, which is the fastest speed grade available in this package. Support, development tools and software for this device are also readily available within the ESL and this is an added advantage during development. The FPGA configuration device required (EPCS4) and configuration schemes will be discussed later in this chapter.

#### 4.1.6 Prototype Development Overview

As stated previously, a development prototype will first be designed. Below is a high level overview of the system.

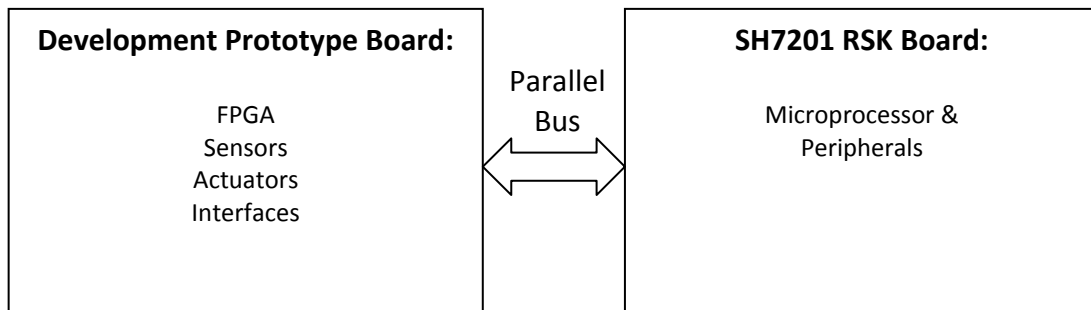


Figure 4.7 - Prototype Development Overview

## 4.2 SCHEMATIC DESIGN OF DEVELOPMENT PROTOTYPE

As stated earlier, it was decided to first develop a hardware prototype without the microprocessor onboard. It is connected to a purchased development board via a ribbon cable. If time allows, a second iteration of the hardware board would be developed to incorporate the microprocessor onboard. The first board could then be used to benchmark and test all the different parts of the systems. A block diagram of the system is shown in Figure and depicts a simplified version of the schematic (refer to Appendix B for full schematic diagrams). Most of the component choices were made to promote the ease of integration consideration in the SRS. Wherever possible, components with digital outputs or communication lines were chosen. This simplifies the circuitry and allows a direct connection to the FPGA. This section will therefore only discuss certain aspects of the schematic design.

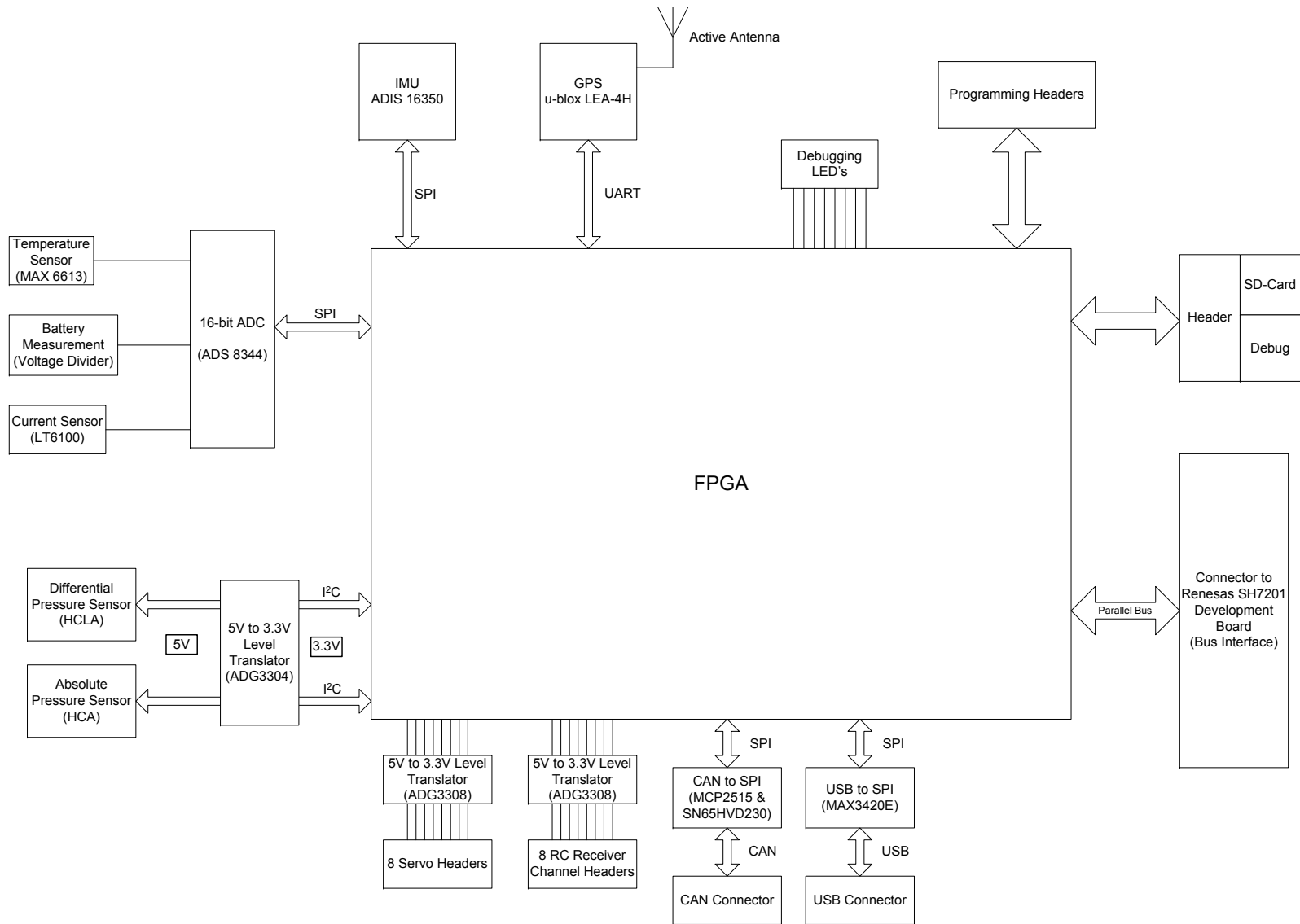


Figure 4.8 - Schematic Overview of the Hardware System



### 4.2.1 Power Distribution and Considerations

The system will make provision for between three and six cell Li-Po batteries to be connected to it for power, as stated in the System Requirements Specification. The nominal input voltage range is thus 11.1V to 22.2V (3.7V per cell). However, provision has to be made for fully charged batteries, as well as partially discharged batteries. Lithium Polymer cells are rated as having a typical safe voltage range of 3.0V (when fully discharged) to 4.2V (when fully charged).

Different voltages are required to power different sections of the hardware. The power distribution is shown in the block diagram overview below.

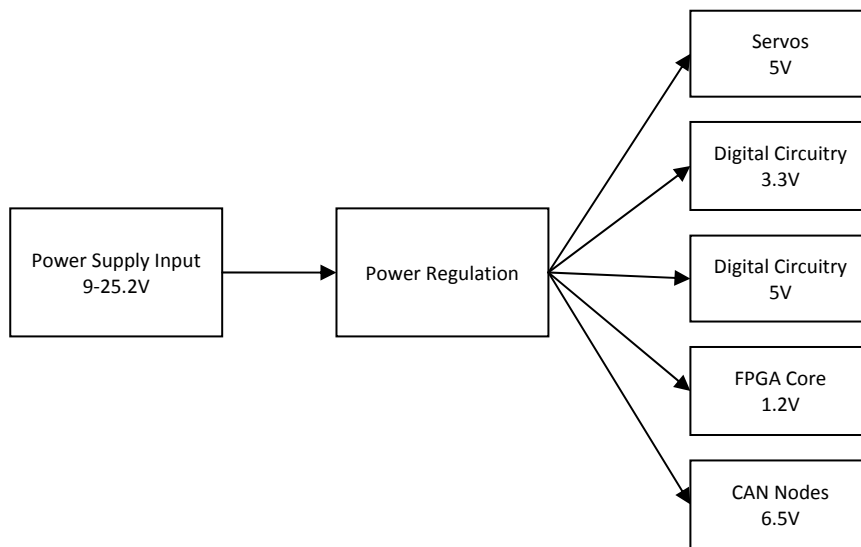


Figure 4.9 - Power Supply Distribution

The difference between the input voltage range and the output voltages of the Power Regulation block required is quite large. Typical linear voltage regulators are thus not capable or suitable to provide the output voltages on their own and it was decided to use switching regulators in conjunction with linear regulators. Switching regulators also provide better power efficiency than linear regulators.

The strategy employed in delivering the different voltages can be seen in the figure below. The battery voltage is stepped down by two parallel high-efficiency integrated switching regulators (ISRs). The first one drives the servos while the second outputs 6.5V. This is used to power the nodes on the CAN bus and also the rest of the circuit. This 6.5V is then reduced to 5V by a low-dropout (LDO) linear regulator which powers the 5V digital circuitry. This 5V is also fed to a 3.3V low-dropout linear regulator to provide power to the 3.3V digital circuitry. Finally, the 3.3V signal is also stepped down to 1.2V by another low-dropout linear regulator to power the FPGA core.

This gradual step down procedure provides cleaner power regulation in terms of noise by separating subsystems.

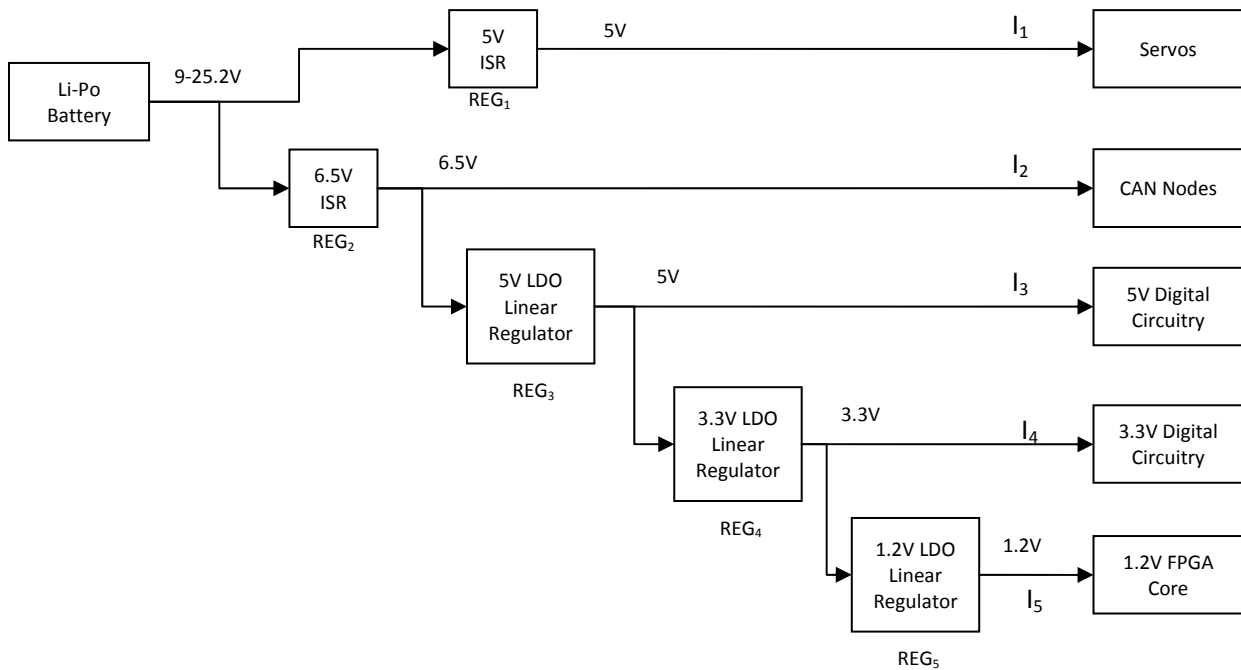


Figure 4.10 - Power Regulation Strategy

An important factor to consider in the regulator selection process, besides the input voltage range of each regulation component, is the current consumption of the different power subsections. These are number  $I_1$  to  $I_5$  in Figure 4.10 above. An analysis and estimation of the current consumption was performed in order to select an appropriate regulation device:

$I_1$ : **Servos** typically consume a maximum of about 3.5W each which translates to about 0.7A at 5V [24]. The eight servo channels in this system would thus consume a maximum current of 5.6A.

$I_2$ : The maximum current consumption of the **CAN Nodes** was set at 1A. This was done upon inspection of the previous CAN avionics power supply circuit.

$I_3$ : The **5V Digital Circuitry** current requirement was calculated from the following maximum supply currents obtained from the respective datasheets:

- ADIS 16350 → 57mA
- RC Receiver → unknown
- HCL → Not stated in the datasheet.

- HCLA → Not stated in the datasheet.
- ADG3304 → 10mA x 4 channels = 40mA
- ADG3308 x 2 → 2 x 10mA x 8 channels = 106mA
- 2.4GHz MaxStream Telemetry Module (provisionally included for future design) = 150mA
- Upon inspection of the above components, a total current consumption not exceeding 1A was estimated.

**I<sub>4</sub>:** The **3.3V Digital Circuitry** current requirement was calculated by adding up the following maximum supply currents obtained from the respective component datasheets:

- MCP2515 → 10mA
- SN65HVD230 → 50mA
- MAX3240E → 50mA
- ADG3304 → 12mA
- LEA-4H → 40mA
- ADS8344 → 75mA
- ADG3308 x 2 → 2 x 24mA = 48mA
- FPGA → 500mA (estimation)
- Total current consumption of the above components is 785mA. However, in order to provide for future incorporation of the microprocessor (with a current consumption of 180mA) onboard, a total current of 1A was decided upon.

**I<sub>5</sub>:** The **1.2V FPGA Core** requires a maximum estimated current supply of 0.5A. This was calculated upon investigation of similar-sized, as well as larger-sized, FPGA based projects.

The following regulators were thus chosen for the Power Regulation circuit above:

**REG<sub>1</sub>:** The PTN78020W high-efficiency step-down ISR from Texas Instruments is used to provide power to the servos. It can deliver a maximum current of 6A. They have a wide input voltage range (7V to 36V) and the conditional adjustable output (2.5V to 12.6V) can be set to 5V. This regulator thus meets all the requirements for the servo power supply.

**REG<sub>2</sub>:** The PTN78020W is also used to step down the input battery voltage to 6.5V. This regulator must be able to deliver the remaining current (I<sub>2</sub> to I<sub>5</sub>) of 3.5A to the system. It therefore complies with the requirements.



Figure 4.11 - PTN78020W switching regulator

**REG<sub>3</sub>:** This regulator must be able to deliver the currents  $I_3$  to  $I_5$  (2.5A) at a voltage of 5V. For this purpose, the UCC283-5 low-dropout (LDO) 3A linear regulator from Texas Instruments was used.

**REG<sub>4</sub>:** This regulator must supply currents  $I_4$  and  $I_5$  which is equal to a total of 1.5A. The TL1963A-33 LDO 1.5A linear regulator from Texas Instruments was thus used.

**REG<sub>5</sub>:** The FPGA core voltage of 1.2V and current of 0.5A is provided by the FAN1112 LDO 1A linear regulator from Fairchild Semiconductor.

In the design of the power circuitry, it was decided to implement removable links/jumpers before each subsystem. This is done to enable the power distribution circuits to be tested before connecting them to any components, thus reducing the risk of damage to the components.

In order to determine the total amount of current drawn from the batteries, the efficiency of the ISRs needs to be taken into account. Switching regulators typically exhibit better efficiency at higher output voltage specifications for constant input voltage because the ratio of power provided to the system to the power consumed by the ISR itself is smaller. The input to output voltage ratio, however, also has an effect on the efficiency.

If we assume the worst case efficiency for the regulator of 85%, the input power and current can be calculated. The power specifications on the output side of the regulator are constant. These are 28W (i.e. 5.6A at 5V) for the REG<sub>1</sub> and 22.75W (i.e. 3.5A at 6.5V) for REG<sub>2</sub>. From this the input power can be calculated by dividing by the efficiency: 33W for REG<sub>1</sub> and 27W for REG<sub>2</sub>. The total power, for the worst case, at the input side of the system is thus 60W. In order to find the maximum current, this total power must be divided by the smallest possible supply voltage, i.e. 9V. This yields a maximum current of 6.67A.

It is important to note that this is the theoretical absolute maximum power usage estimated for the system. The actual power consumption would typically be much less as most components will not draw the maximum supply current constantly nor simultaneously. The figure below illustrates the current usage on the input side of the regulators.

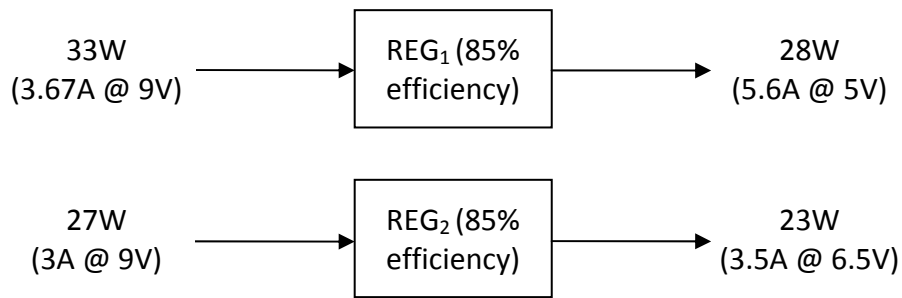


Figure 4.12 - Input and Output power to the switching regulators for the worst case efficiency of 85% and a supply input voltage of 9V

**Recommendation:** During hardware testing of the power circuit on the PCB, problems in terms of reliability were found with the FAN1112 regulator. A possible replacement in future designs is the NCP565 1.2V 1.5A low-dropout linear regulator from ON Semiconductor.

#### 4.2.2 IMU Considerations

The ADIS 16350 IMU device interfaces physically to hardware through a special Samtec connector. Care was taken in alignment of the mating connector on the PCB, as well as mounting holes for keeping the device in place. Even though the device requires a 5V power supply, the digital I/O pins are driven by an internally regulated 3.3V supply, as stated in the datasheet [25]. No level translation is therefore required and the necessary I/O pins for SPI communication can be routed directly to the FPGA.

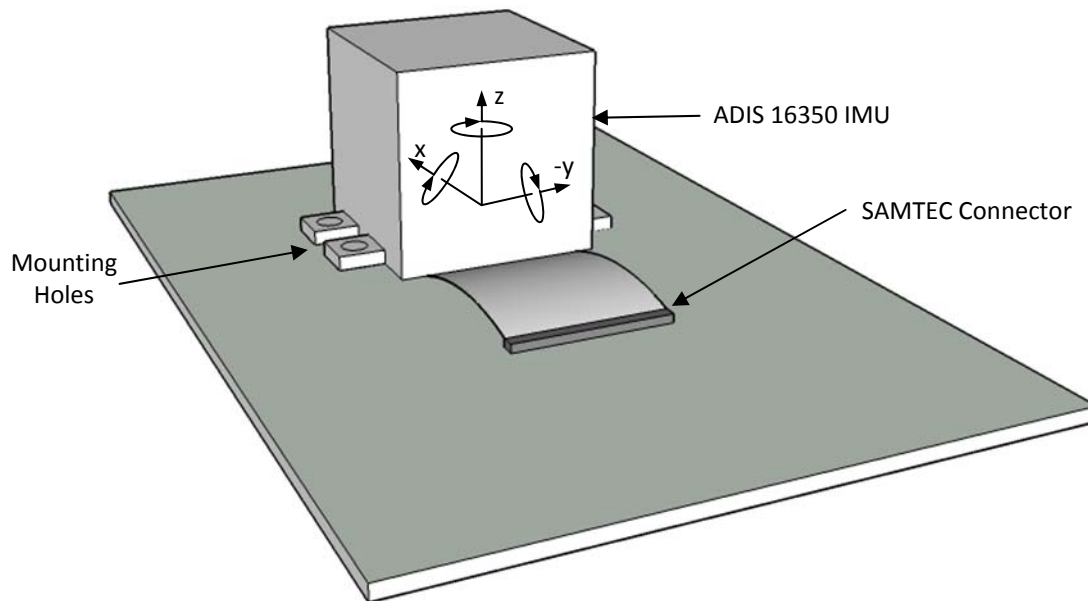


Figure 4.13 - Drawing of the IMU mounted on PCB

### 4.2.3 Pressure Sensor Considerations

The application circuit for both the Sensortech HCLA and HCA pressure sensors was taken from the device application note [26]. This includes the necessary pull-up and series resistors required for the I<sup>2</sup>C communication lines. The pressure sensors are driven by a 5V power supply and therefore the I<sup>2</sup>C lines of the devices require voltage level translation to 3.3V between itself and the FPGA. The ADG3304 Bidirectional Logic Level Translator from Analog Devices was included in the design for this purpose.

### 4.2.4 GPS Module Considerations

The circuit for the LEA-4H was designed using the design guide in [14]. The GPS module operates off a supply of 3.3V and the UART communication pins can thus be routed directly into the FPGA. The GPS module uses an active antenna which connects to the circuit via an MMCX connector. Special considerations need to be taken into account when placing the GPS module on the board and routing the tracks as it is very sensitive to RF noise. This will be discussed later in this chapter during the PCB layout section.

### 4.2.5 Secondary Sensors

The ADS8344 16-bit ADC device from Analog Devices is used to digitize the analogue outputs of the various secondary sensors. These include the MAX 6613 Temperature sensor, the LT6100 Current sensor and a voltage divider circuit to measure the battery voltage. An accurate 3V reference signal, provided by the REF3130 device, is fed to the V<sub>REF</sub> pin of the ADC device. This allows the signal to be digitized into 45.78uV intervals. The ADS8344 ADC device has 3.3V logic levels and can be routed straight to the FPGA.

The voltage divider circuit is designed to linearly scale the battery supply voltage (V<sub>BATT</sub>) down by a factor of 10. This brings the 11.1V to 22.2V range down to 1.11V and 2.22V respectively, and thus within the measurable 3V (V<sub>REF</sub>) range of the ADC device.

The LT6100 current sensor monitors the current by measuring the voltage across a current sense resistor (R<sub>SENSE</sub>). It then outputs a voltage which is proportional to the magnitude of this current. R<sub>SENSE</sub> is a special low-ohmic current sensing resistor designed to handle large currents. The value of this resistor is calculated as follows:

The maximum current which R<sub>SENSE</sub> must handle: I<sub>SENSE\_max</sub> = 6.67A (See power distribution section above)

The maximum voltage output of the sensor: V<sub>OUT\_max</sub> = 3V (Due to ADC Reference Voltage)

Now:  $V_{OUT} = V_{SENSE} \times A_V$

and  $V_{SENSE} = I_{SENSE} \times R_{SENSE}$

therefore  $V_{OUT\_max} = I_{SENSE\_max} \times R_{SENSE} \times A_V$

$R_{SENSE} = 3V \div (6.67A \times AV)$

Choose  $A_V = 20$  (datasheet), therefore  $R_{SENSE} = 22.49m\Omega$

Now, the resistor power rating is:  $P_{SENSE\_max} = I_{SENSE\_max}^2 \times R_{SENSE} = 1 \text{ W}$  resistor needed

This  $R_{SENSE}$  value is close to the standard value of  $22m\Omega$  which was used. This current sense resistor is also available in a 1W maximum power rating available from the supplier.

#### 4.2.6 FPGA Configuration

The Cyclone II device requires configuration data to be loaded into its volatile SRAM memory at every startup. There are three possible configurations which can be used, namely, the active serial (AS), passive serial (PS) or Joint Test Action Group (JTAG) configuration schemes. The schematic design details for each can be obtained from [27].

For this design, it was decided to implement the AS configuration scheme as the programming tools are readily available in the ESL. This configuration uses the EPCS4 serial configuration device to store and load the FPGA configuration data at power up. It was also decided to make provision for JTAG as the development tools can also be obtained but with less frequency of use. The schematic diagrams for each configuration are included in the Appendix C.

Furthermore, debugging pins are made available in the schematic design of the circuit to assist hardware testing and development. These pins can be used to add offboard components to the system if needed.

### 4.3 PRINTED CIRCUIT BOARD LAYOUT

This section discusses the main aspects taken into account during the PCB layout process. This includes the amount of layers required, placement of components in terms of noise coupling and ease of routing and track considerations (e.g. track impedances and widths).

#### 4.3.1 Number of PCB Layers and PCB Dimensions

The first decision which was made before the routing process was the amount of layers needed for the PCB design. An increase in PCB layers has the following advantages: simplifies the routing process as there are more layers to route on with dedicated power and ground planes, reduces overall PCB dimensions as components can be more densely populated; but it also adversely leads

to greater manufacturing costs. A tradeoff thus had to be reached between these aspects as both cost and size are specified requirements for this project.

The most commonly manufactured PCBs are 2-, 4-, 6- and 8-layer boards. It was decided that a 4-layer board would provide the best solution for this hardware system in terms of cost and size. This was due to the fact that top and bottom layer PCB real estate was required to place physical components as densely packed as possible. It would have been too crowded in terms of track routing to place power and ground tracks on these layers as well.

The PCB is the size of a standard PC104 form-factor board (9.017cm x 9.589cm).

### 4.3.2 PCB Layer Stackup Planning

In a 4-layer PCB layout, it is standard practice to dedicate one layer to a ground plane and another to a power plane, with occasional signal tracks routed on them. The top and bottom layers are then mainly used to route the signals. The layer stackup for the PCB is shown in the figure below. The parameters are taken from the manufacturer specification document [28].

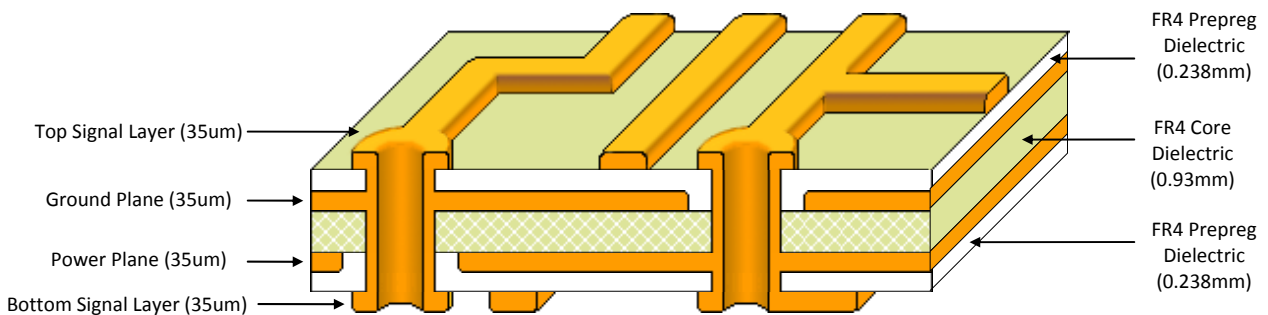


Figure 4.14 - PCB Layer Stackup

The ground plane is placed above the power plane (i.e. closer to the top layer) in the PCB stackup mainly for the following reasons:

- The FPGA and the parallel bus tracks will be routed on the top layer. Placing the ground plane directly below these signals should minimize possible cross-talk between data switching tracks and improve EMC performance of the board [29].
- The GPS module (on the top layer) requires a coplanar waveguide transmission line between the antenna MMCX connector and the module. Placing the ground plane directly beneath the top signal layer makes this possible.



### 4.3.3 Component Placement Considerations

There is a large number of ways in which the components can be arranged on the PCB. The component placement strategy was thus executed according to the following factors:

- Place the components with the straightest possible routing path between them.
- Centre the component placement around the FPGA's position.
- Place the larger, bulkier components first.
- Place RF sensitive components away from possible noise sources and towards the outer edges of the PCB.
- Place all connectors and headers along the edges of the board for ease of accessibility.
- Place all the components with a larger height on the same side of the board, i.e. on the top PCB layer, so that the hardware protrudes mainly towards one side.
- Place power supply decoupling capacitors as close to component pins as is physically possible. Place the many FPGA decoupling capacitors in the area directly below the FPGA on the bottom layer.
- Place all power supply regulators towards one side of the board.

Using the above strategy, the component placement shown in the figures below was decided upon. These figures only show the most crucial components and factors considered in order to create a starting point for the PCB routing process. Other components were then placed around this, but still following the strategy above. A complete view of the component placement is shown in Appendix D.

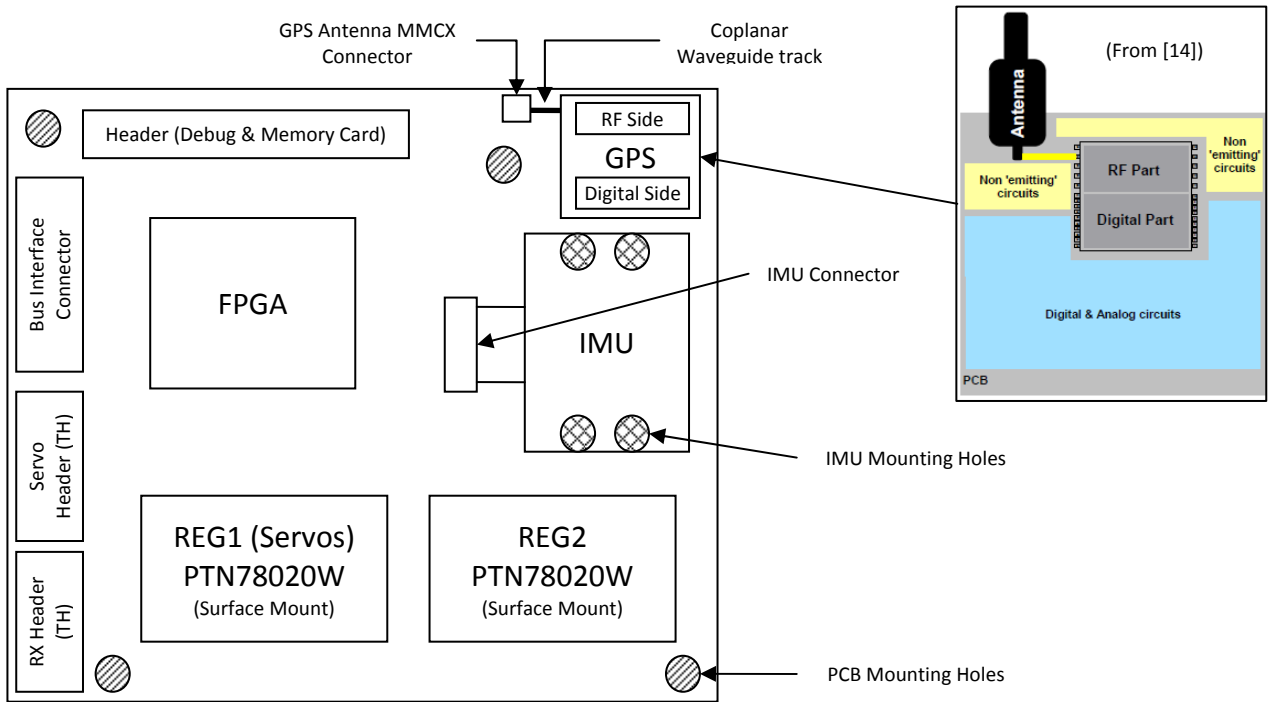


Figure 4.15 - Component Placement Strategy overview of the Top PCB Layer

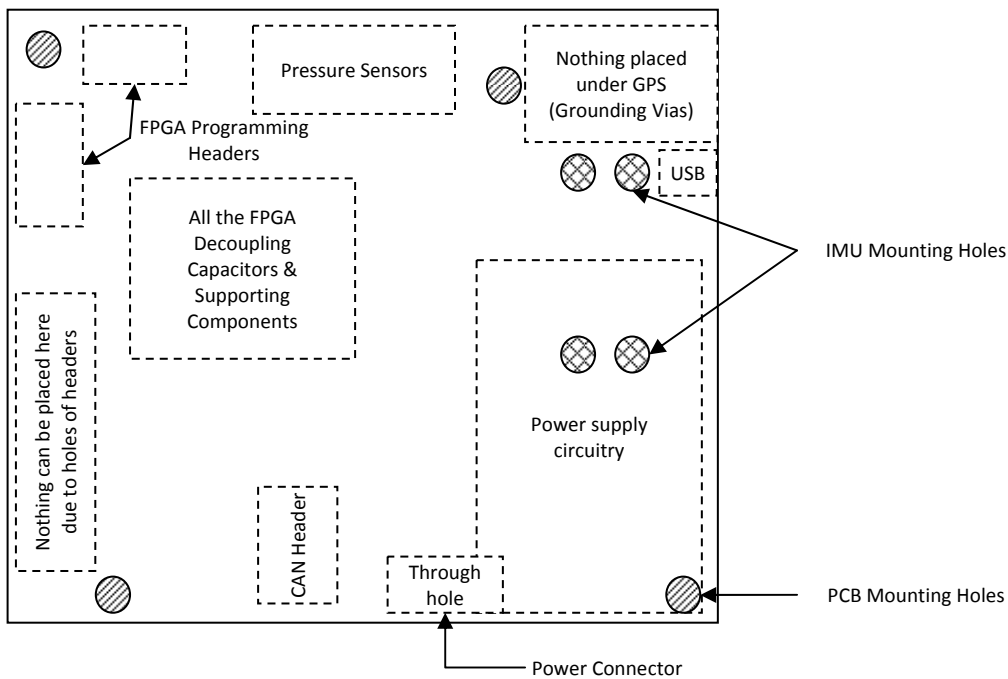


Figure 4.16 - Component Placement Strategy overview of the Bottom PCB Layer (Viewed through the Top Layer)

It was decided to remove the memory card hardware from the main board and place it on a secondary board to be created, which plugs into the available header pins, as there was no space for it onboard. It would be incorporated on the final board which would be slightly larger. To minimize the effect of overall size when placing more components (for example, the

microprocessor and SD-card) on the final board system, a 6-layer PCB should be considered in order to place components even more densely on the board.

#### 4.3.4 Track and Routing Considerations

Various considerations and decisions had to be made during the actual routing of the PCB tracks. Below a few of the key considerations are highlighted.

##### GPS Antenna Track Impedance Control

The GPS module's RF input pin desires an input impedance of  $50\Omega$ . The GPS antenna and MMCX RF connector also have  $50\Omega$  impedances. It is thus necessary for the PCB track between the connector and the module's RF antenna input to have a controlled impedance of  $50\Omega$  as well. This can be achieved by various PCB waveguide solutions. The two transmission line configurations considered for this application were microstrip lines and coplanar waveguides (CPWs). It was decided to use the CPW transmission line model as it offers more advantages over microstrip lines, as well as being recommended by [14].

The AppCAD utility software from Agilent Technologies was used to determine the required track dimensions of the CPW line. To obtain the desired track impedance of  $50\Omega$ , the following known parameters were entered:

- *Frequency band: 1575.42 MHz*
- *Dielectric: FR-4 prepreg material with  $\epsilon_r=4.6$*
- *Height above ground plane (H): 0.238mm*
- *Copper weight (T): 35um (1 oz.)*

The track width (W) and ground gap (G) were then adjusted to obtain a track impedance as close to the desired  $50\Omega$  as possible. The resulting parameters are a track width of 18mils (0.4572mm), a ground gap of 18mils (0.4572mm) and an impedance of  $49.8\Omega$ . The final configuration is shown in Figure 4.17 below.

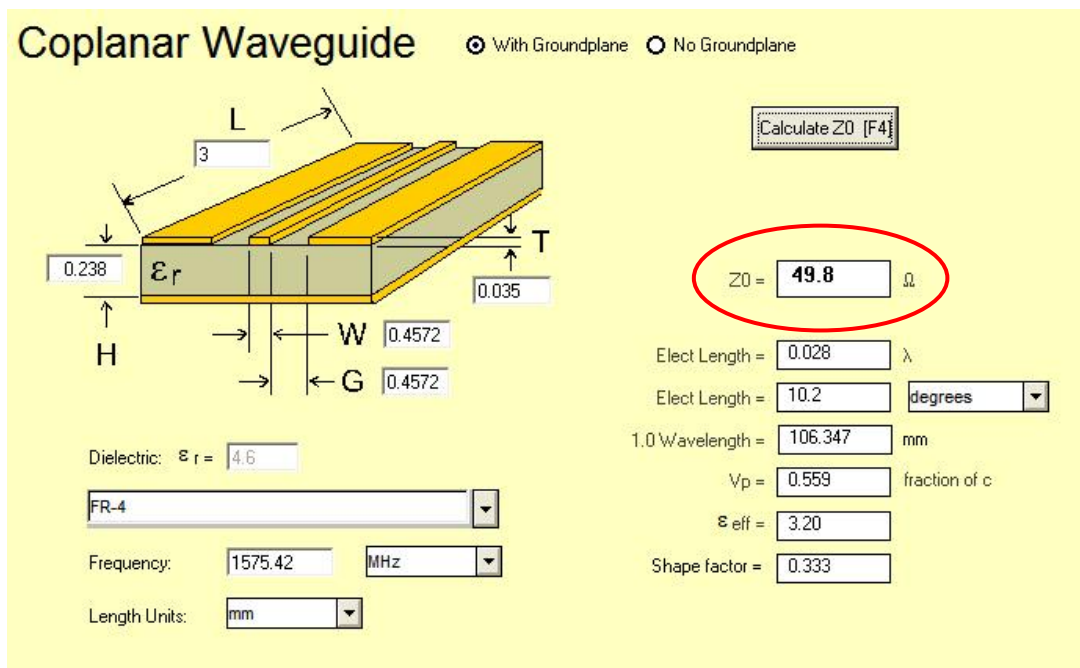


Figure 4.17 - Screenshot of Agilent AppCAD Utility used to calculate the line impedance for the GPS antenna track.

## PCB Track Widths

A number of different track widths were used on the PCB. The main aspects which governed the track widths were:

- The maximum amount of current the supply tracks would conduct (i.e. the larger the possible current, the larger the track width requirement).
- The amount of space available.
- The manufacturer specifications and capabilities [reference].

The widths for PCB tracks carrying large amounts of current on the outer layers were determined according to the IPC-2221: “Generic Standard on Printed Board Design” design standards. An online calculator [30] which implements this standard was sourced and used. Below is a table showing the supply currents in the PCB design and the track widths. The temperature rise of the track used in the calculation is also shown. These values were used as the minimum track widths as far as possible.

Table 4.5 - Recommended PCB Track Widths

Maximum Current	Recommended Track Width	Temperature Rise
9.1A	163mils	20°C
5.6A	84mils	20°C
3.5A	44mils	20°C
2.5A	28mils	20°C
1.5A	21mils	10°C
1A	12mils	10°C

The remaining power and signal tracks are routed with a track width of 10 mils. This is safely within the specification range of the PCB manufacturer. A 10 mil track can carry a current of about 1A with a 10°C rise in track temperature which is more than sufficient.

A recommendation for future designs is that the internal layer power tracks should be made larger as to decrease the temperature rise under full current load.

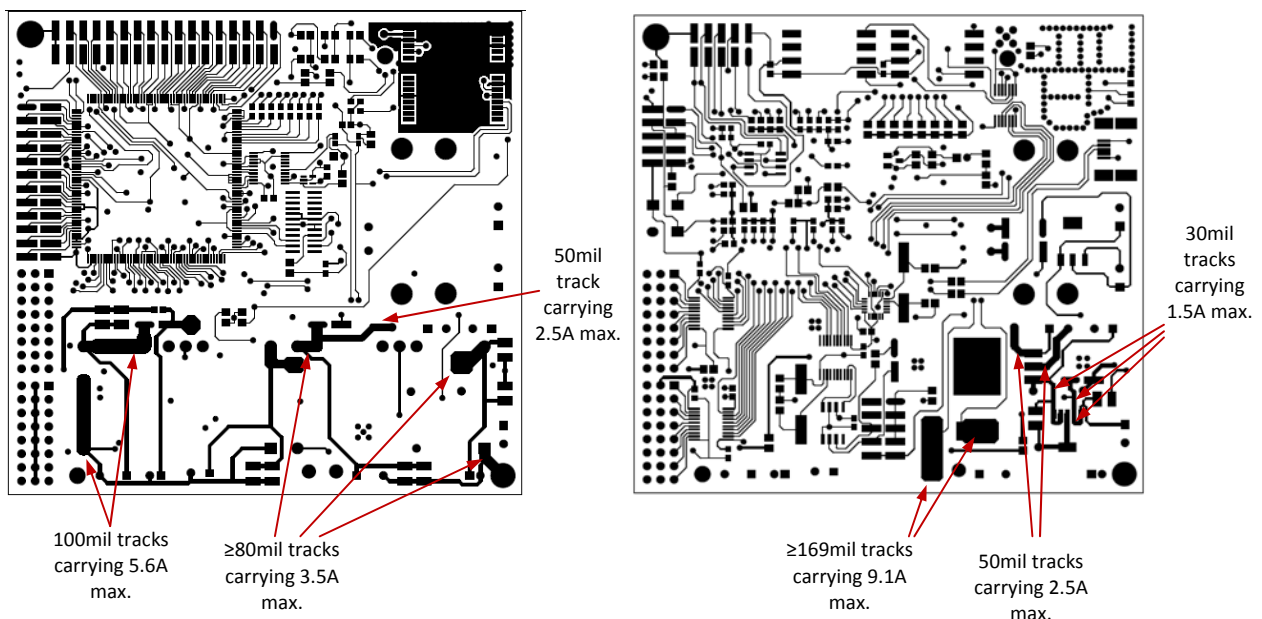


Figure 4.18 - PCB layout indicating the certain high current carrying tracks and their widths

Only plated through hole vias are used in the PCB layout. No blind or buried vias are used as this complicates the manufacturing process of the PCB and increases costs. Vias which lie under

certain components with exposed metal parts were covered with the soldermask, called tenting, to prevent them from touching, which could lead to short-circuits.

The final PCB design layout for the development prototype can be seen in Appendix E. The PCB designed in this chapter was manufactured by Trax Interconnect. Once populated with the components, this board can be used to test the overall functioning of the system. A final PCB incorporating all the components of the avionics embedded hardware system would then be created. Pictures of the hardware are included in Appendix G.

## 5. DETAILED FIRMWARE DESIGN

This chapter describes the VHDL firmware implementation in the FPGA. It describes the modules used or developed in this project. The first requirement within this design phase is to establish the basic building blocks of this system. The devices which interface with the FPGA have a set of protocols which in some cases are common between them (e.g. the SPI protocol). Furthermore, each device then has a specific operation structure which controls its functioning. The strategy was to first create and find the basic communication protocol modules, then the device-specific controller modules and use these to build up the system. This assists in creating certain levels of abstraction between modules and also promotes re-usability.

A design approach for synchronous sequential digital systems is shown in Figure 5.1 [31] & [32]. It consists of two partitions: the controller and the datapath. These communicate with the signals shown. This implementation is realized with the use of finite state machines. Datapath blocks are those modules which transfer data and perform operations on it.

This design approach was considered in the design of modules in this system as far as possible. The basic idea is that each lower level module created uses this approach internally. It then becomes a datapath block to a higher level controller. The system is thus expanded by interfacing the controller and datapath blocks with each other.

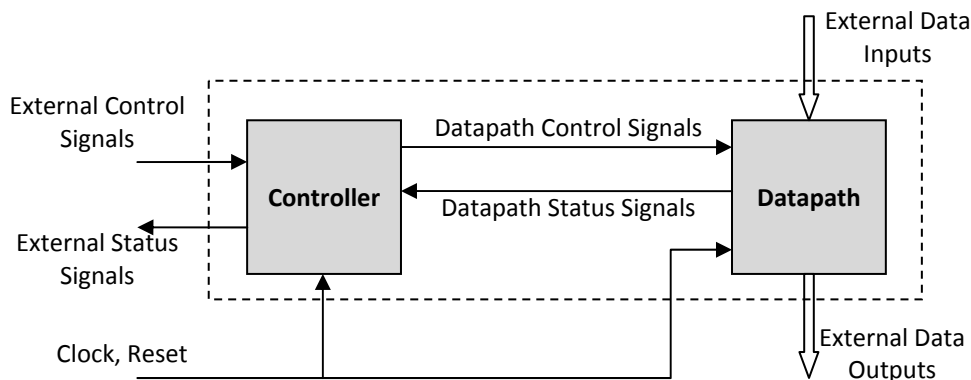


Figure 5.1 - Control and datapath partitions in a synchronous sequential system

The design of the VHDL modules is done with algorithmic state machines, or ASMs. The clock signal controls state transitions. The output signals remain in a current state until explicitly changed. In most cases in this design, an asynchronous reset is used to enter the state machine and the outputs into known states.

A simplified block diagram of the required FPGA modules is shown in Figure 5.2. The need for each module can be traced back to the component choices and conceptual design of the system. This **traceability** is summarized as follows:

- The SPI, I<sup>2</sup>C and UART interface modules: This is the respective communication protocol required by certain devices, as seen in the figure below. This firmware development requirement thus traces back to the component choices in the hardware design phase.
- The Bus Interface Controller module: This module is required to allow the FPGA system to communicate with the external bus of the SH7201 microprocessor. This requirement can be traced back to the conceptual design of the system. The specific functioning of this module traces back to the microprocessor selected.
- Device Controller modules: These modules control the device-specific operations and functioning and are traced back to the component selected in each case.
- PWM and PWM Capture modules: This firmware implementation requirement is traced back to the System Requirements Specification discussed in Chapter 2. The PWM module drives the control signals of the servos while the PWM capture module reads the signals outputted by an RC receiver hardware module.
- Dual Port RAM: Memory registers are required within the FPGA to hold data. The microprocessor is abstracted from the internal functioning of the FPGA system and essentially only sees this block of memory registers through the bus interface. This can also be traced back to the conceptual design of the system.
- Main Controller and Timing modules: This is a high level block which consists of all the modules which control the overall functioning and dataflow of the system. It would also be responsible for the signals which trigger the overall system functioning, as discussed in the conceptualization of the system in Chapter 3, as well as the control of the lower level modules discussed above.



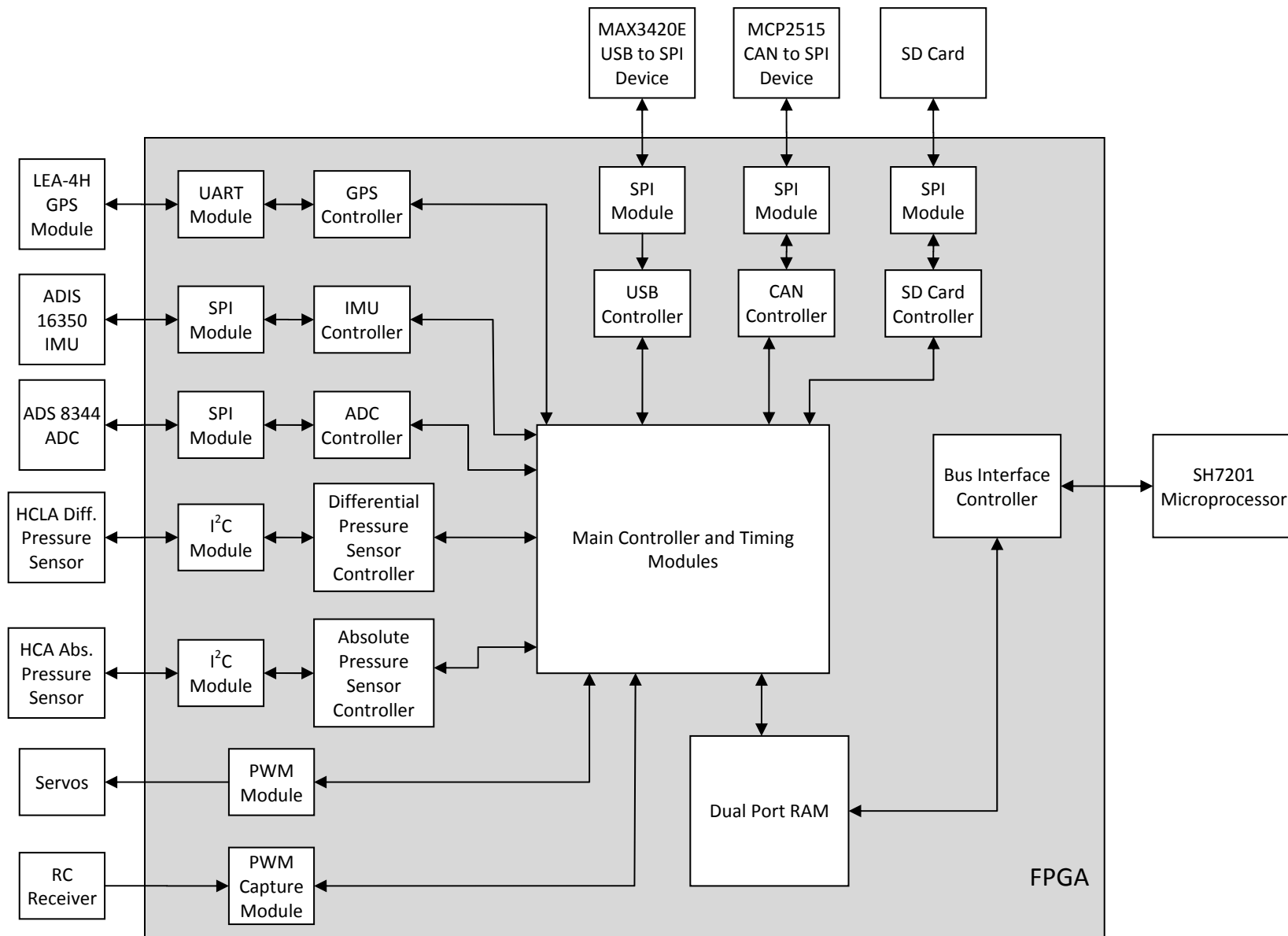


Figure 5.2 - High level Overview of the Main VHDL Modules Required

## 5.1 SERIAL PERIPHERAL INTERFACE (SPI) MODULE

### 5.1.1 SPI Protocol Description

SPI works on the principle of two shift registers synchronously exchanging data in a master-slave configuration. It is a four wire serial interface and consists of the following signal lines for communication:

- Serial Clock (SCLK) – the master generates this clock signal which synchronizes and controls the data transfer over the bus. Data is changed on one edge of the clock and read/sampled on the next. This is done to ensure data stability when the data is read. The clock phase (CPHA) and the clock polarity (CPOL) play a fundamental role in the way in which data is transferred between the master and the slave on the bus. This is discussed later in this section.
- Master Output Slave Input (MOSI) – This is the data line which carries data from the master to the slave.
- Master Input Slave Output (MISO) – This is the data line which carries data from the slave to the master.
- Chip Select (nCS) – Selects which slave the master wants to communicate with as it is possible to connect more than one slave on the SPI bus. Transactions may only take place between the master and one slave at a time by asserting the specific nCS line low.

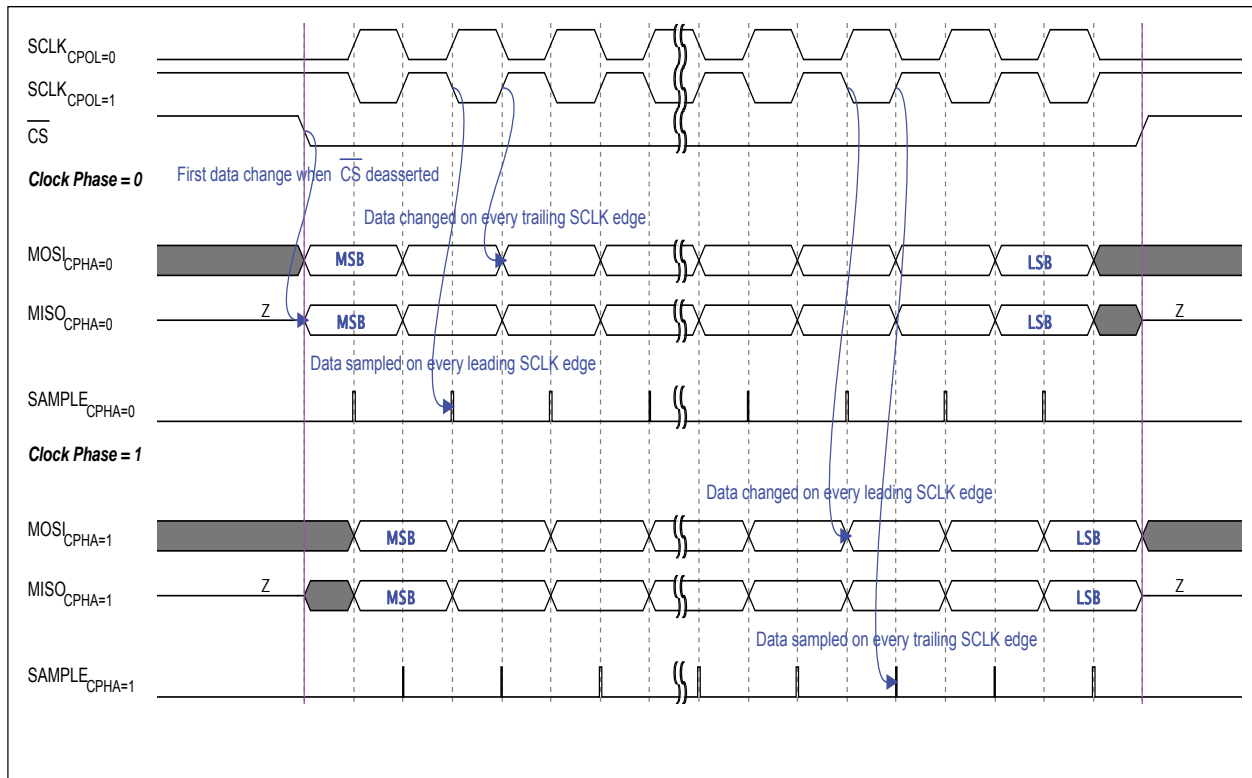
There are two fundamental modes of SPI operation depending on the CPOL property. The first mode is when the clock phase equals zero (CPHA=0). In this mode, both the master and the slave sample the data on every leading edge, as seen in the figure below, and the data is changed on every trailing edge. As the devices expect to sample data on the first leading clock edge, the data must be available before this point. The first data change thus occurs on the low assertion of the nCS signal. Before this point, the MISO line is in a high impedance (hi-Z) state and the MOSI in a “don’t care” state. When the nCS line is asserted again, MISO returns to a hi-Z state. The second mode of SPI occurs when the clock phase is set to one (CPHA = 1). In this mode, data is changed on the leading edge of the clock and sampled on the trailing edge, as shown in the figure below.

Furthermore, the polarity of the clock (CPOL) also needs to be considered. This is the idle state of the clock and will thus be low for a polarity of zero (CPOL=0) and high for a polarity of one (CPOL=1). This can also be seen in the figure below.

**Table 5.1 - SPI Modes of operation**

SPI MODE	CPOL	CPHA
Mode 0	0	0
Mode 1	0	1
Mode 2	1	0
Mode 3	1	1

These two properties of the clock (i.e. CPOL and CPHA) combine to select the specific mode of SPI for communication, and both master and slave must adhere to these requirements for correct operation. This is shown in Table 5.1.



**Figure 5.3 - Timing diagrams showing the different modes of SPI.**

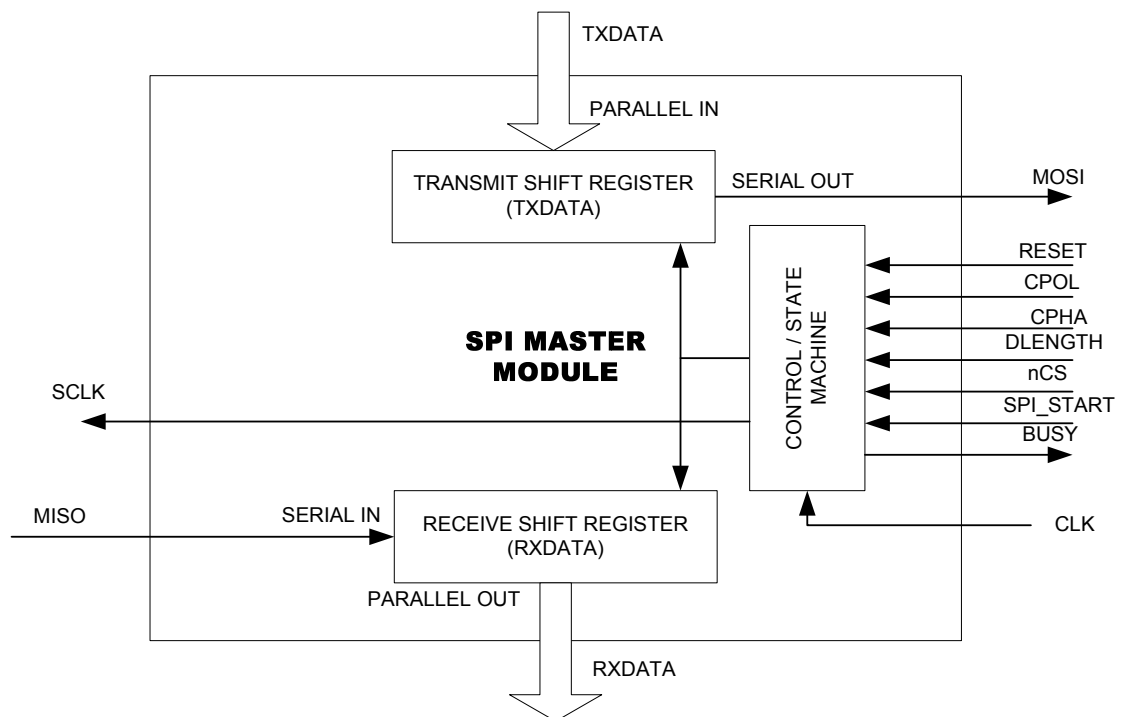
The length of the SPI data transfer depends on the requirements for specific device. The speed of the SPI bus is variable and relatively fast. It is dependent on the timing requirements for the specific devices. Below the SPI devices used in this project is tabulated.

**Table 5.2 - List of the SPI devices used in this project:**

Device	Description	SPI Mode	Data Transfer Length per transfer frame	Max. SPI Clock Speed
ADIS 16350	High Precision Tri-Axis Inertial Sensor	Mode 3	16 bits	2 MHz
MAX 3420E	USB Peripheral Controller with SPI Interface	Mode 0 or 3	8 bits	26 MHz
MCP 2515	Stand-Alone CAN Controller with SPI Interface	Mode 0 or 3	Varies but max. is 32 bits	10 MHz
ADS 8344	16-Bit, 8-Channel Serial Output Sampling Analog-To-Digital Converter	Mode 0	32 bits	2.4 MHz
SD Card	SPI Interface for SD card	Mode 0	8 bits	25 MHz

### 5.1.2 SPI Master Module Design

An SPI master module is needed to communicate with the SPI slave devices in this project. It was subsequently decided to design one SPI master module which implements all the modes of the SPI protocol. This would then create an SPI master which can be used to communicate with any SPI slave device, no matter what the mode of operation. This promotes re-usability within this project as well as in future projects. The implementation of the ASM of this module is based on the mode (0, 0) example and VHDL code sourced from [33] but this example was significantly modified and adapted to implement all the modes of SPI.



**Figure 5.4 - High level concept of the SPI Master Module**

### 5.1.3 SPI Master Module Algorithmic State Machine (ASM)

The SPI master module has an asynchronous reset line. When a reset occurs, the outputs are set to their default values as shown in Figure 5.5 and the SPI module enters the *spi\_idle* state after the reset. In this state, the *ncs* line is held high and the *sclk* line is set to the *cpol* (0 or 1) value as the SPI bus is idle. It waits in this state until the *spi\_start* signal is asserted for at least one state machine clock cycle (this is important to note as *spi\_start* line may be driven by a module/state machine with a faster clock and thus the start pulse may not be detected if it is too short. It is also important to note that the *spi\_start* signal must be negated again before the state machine has completed one full SPI transfer frame and returned to the *spi\_idle* state). The state machine then checks the value of the *cpha* register and sets the *busy* line high before transitioning to the next relevant state in one of the two following ASM branches:

- For a *cpha* value of '0', the state machine enters the *loadbit0* state. The first time the state machine enters this state, the *ncs* line is asserted low to indicate the beginning of the SPI transfer frame on the SPI bus and because the *index* value is '0' initially, the first bit to be transferred (*txdata[0]*) is loaded onto the *mosi* line, while the *sclk* line remains in its idle state. Later, when the state machine enters this state again, the relevant data bit will be loaded onto the *mosi* line depending on the *index* value. The *ncs* line will then still be held low and the *sclk* line toggled accordingly. After this, the state machine enters a wait-state, *wait0*, in which nothing happens and all output signals remain unchanged. This wait-state is important to ensure the correct functioning of the state machine. The state machine then transitions to the *transf0* state in which the *sclk* line is inverted and the data from the slave on the *miso* line is read and placed into the *rxdata* register. The next state, *chkdone0*, is then entered in which the counter/index value is checked to see if the SPI transfer is complete based on the data length value, *dlength*. If the transfer is not complete, the index value is incremented and the state machine returns to the *loadbit0* state and repeats the process. The index value serves a dual purpose. It acts as a pointer for the shift registers as well as a counter to count the amount of bits that have been transferred. Once the SPI transfer is complete, the state machine returns to the *spi\_idle* state and waits for the next SPI transaction to occur.
- For a *cpha* value of '1', the *wait1a* state is entered in which the *ncs* line is asserted low. This wait-state causes a delay of one clock cycle between the time the *ncs* line goes low and the first *sclk* clock edge. This delay can also be seen in Figure 5.3. The state machine then transitions to the *loadbit1* state in which the *sclk* line is inverted and the data bit to be

transferred, according to the *index* value, is placed on *mosi*. The *wait1b* state is then entered in which no change to the outputs occur. The state machine then transitions to the *transf1* state in which the *sclk* line is toggled and the data bit on the *miso* line placed into the *rxdata* shift register. The state machine now enters the *chkdone1* state and this is functionally the same as the explained for a *cpha* of '0'.

The core states in the ASM which are responsible for the correct operation of the SPI module are the *loadbit0*, *wait0*, *transf0* and *chkdone0* states for *cpha* equal to '0' and the *loadbit1*, *wait1b*, *transf1* and *chkdone1* states for *cpha* equal to '1'. In an ASM, each state block occupies one clock cycle. Thus these four states in each case use four clock cycles to complete one SPI bit transfer. The SPI bus clock, *sclk*, thus runs at a frequency four times slower than the state machine clock. This has to be taken into account when choosing a clock input for the SPI master module state machine and is given by:

$$f_{sclk} = \frac{f_{clk\ of\ SM}}{4} \quad (5.1)$$

It is also important to note that the parallel loaded data registers *txdata*[31..0] and *rxdata*[31..0] are reversed from the order in which the data is serially received. The controller module implemented in VHDL was simulated to test its functionality and the results are shown in Chapter 7. From these simulations it was shown that this module works correctly and can thus be used by other controller modules in further development of the system. Below is a figure of the ASM followed by a table which describes the ports of the SPI Master module.

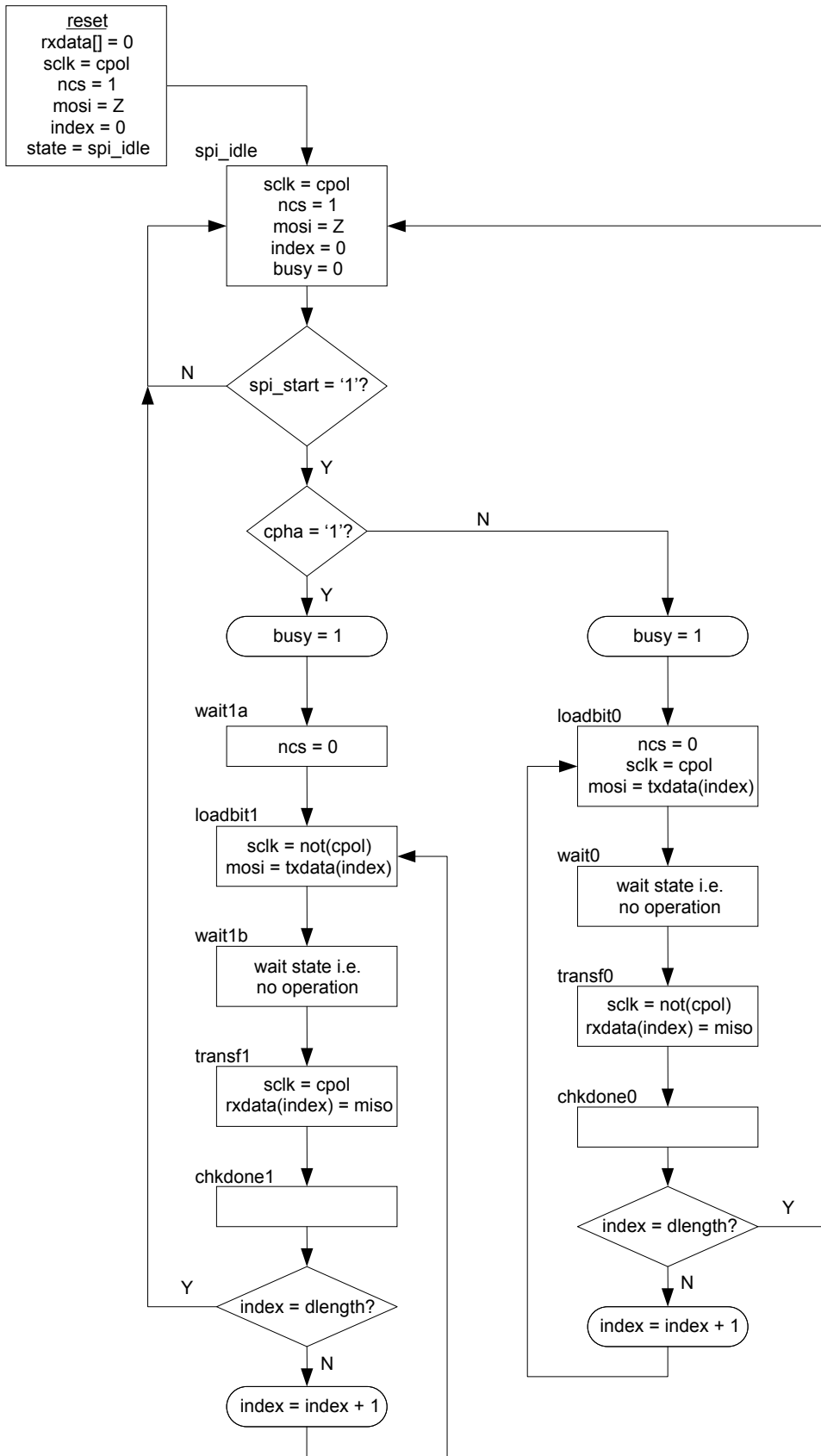


Figure 5.5 - Algorithmic state machine of the SPI master module

**Table 5.3 - SPI Master module port descriptions**

Port Name	Direction	Description
reset	in	Provides the module with an asynchronous reset to initialize it.
clk	in	Connects the module to a clock with controls the state machine.
cpol	in	Selects the clock polarity for the SPI mode of operation.
cpha	in	Selects the clock phase for the SPI mode of operation.
dlength	in	Integer value with a range of 0 to 31 and selects the number of bits per SPI transfer frame. The dlength value is one less than the amount of bits in the data transfer. The maximum data transfer length is 32 bits.
sclk	out	SPI clock generated by the module.
ncs	out	The nCS line of the SPI bus.
mosi	out	The data line which serially transfers data from the master module to a slave device.
miso	in	The data line which serially connects a slave device to the master module..
txdata[31:0]	in	The port/register which holds the data to be transferred from the master to the slave (32-bits wide). [bits in reverse order]
rxdata[31:0]	out	The port/register which holds the data which the master module receives from the slave (32-bits wide). [bits reverse order]
busy	out	This line is high while the SPI Master module is busy with a transfer and not idle.
spi_start	in	This is the line which starts the SPI transfer frame. It must be kept high for at least one <i>clk</i> cycle and negated before the SPI transfer is complete.

## 5.2 PWM MODULE

As established in the SRS and conceptualization of the system, a PWM interface to control RC servos is required. An RC servo is controlled by sending a pulse every 20ms, or 50Hz, to the servo. The duration or width of the pulse determines the angle output by the servo. The specifications of servos may vary between manufacturers. The diagram below illustrates the relationship between the servo angle and the pulse width for JR servos as given in [24].



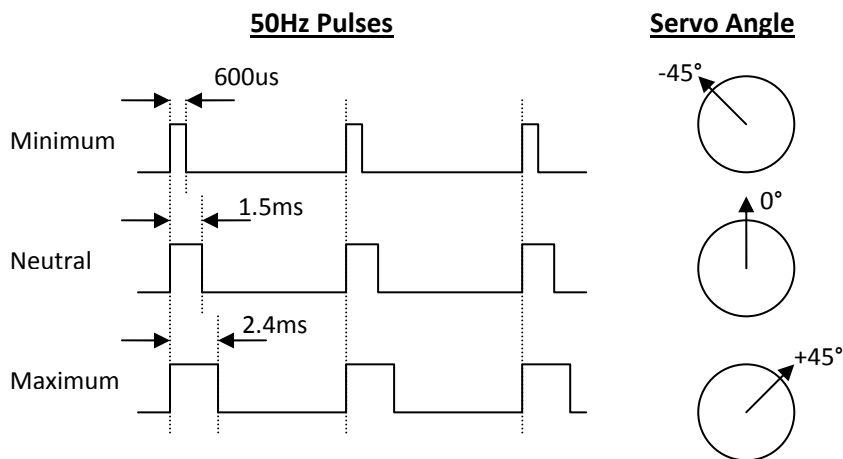


Figure 5.6 - Relationship between the servo control pulse and angle [24].

The PWM module generates servo pulses for one servo. Eight of these can then be placed in parallel to provide the required eight servo channels. The block diagram below shows the inputs and outputs of the PWM module, as well as the ASM.

The module simply functions by outputting one pulse every time its `pwm_inpulse` line is triggered. The module will be connected to a higher level timer which pulses the `pwm_inpulse` line every 20ms to create the 50Hz servo control pulses. This higher level timer forms part of the Main Controller and Timer Modules block seen in Figure 5.2. The duration of the pulse is determined by the `pwmcount` value and the frequency of the module clock (`clk`) and is calculated as follows:

$$t_{pw} = (\text{pwmcount value}) \times t_{clk}, \quad (5.2)$$

where  $t_{pw}$  is the pulse length and  $t_{clk}$  is the clock period

The typical range of the servo pulse length is about 600us to 2.4ms. The `pwmcount` value can be set to a value between 0 and 8191. This means, the maximum frequency allowed to gain the best possible resolution over the full range is  $1/(2.4ms \div 8191) = 3.4\text{MHz}$ . This means that each `pwmcount` value takes 293ns. This means that the servo can be controlled to angles with a  $14.64 \times 10^{-6}^\circ$  resolution. This is more than sufficient.

The description of each port of the PWM module is given in the table below. The PWM module was simulated and the functionality verified. The results are displayed and discussed in Chapter 7.

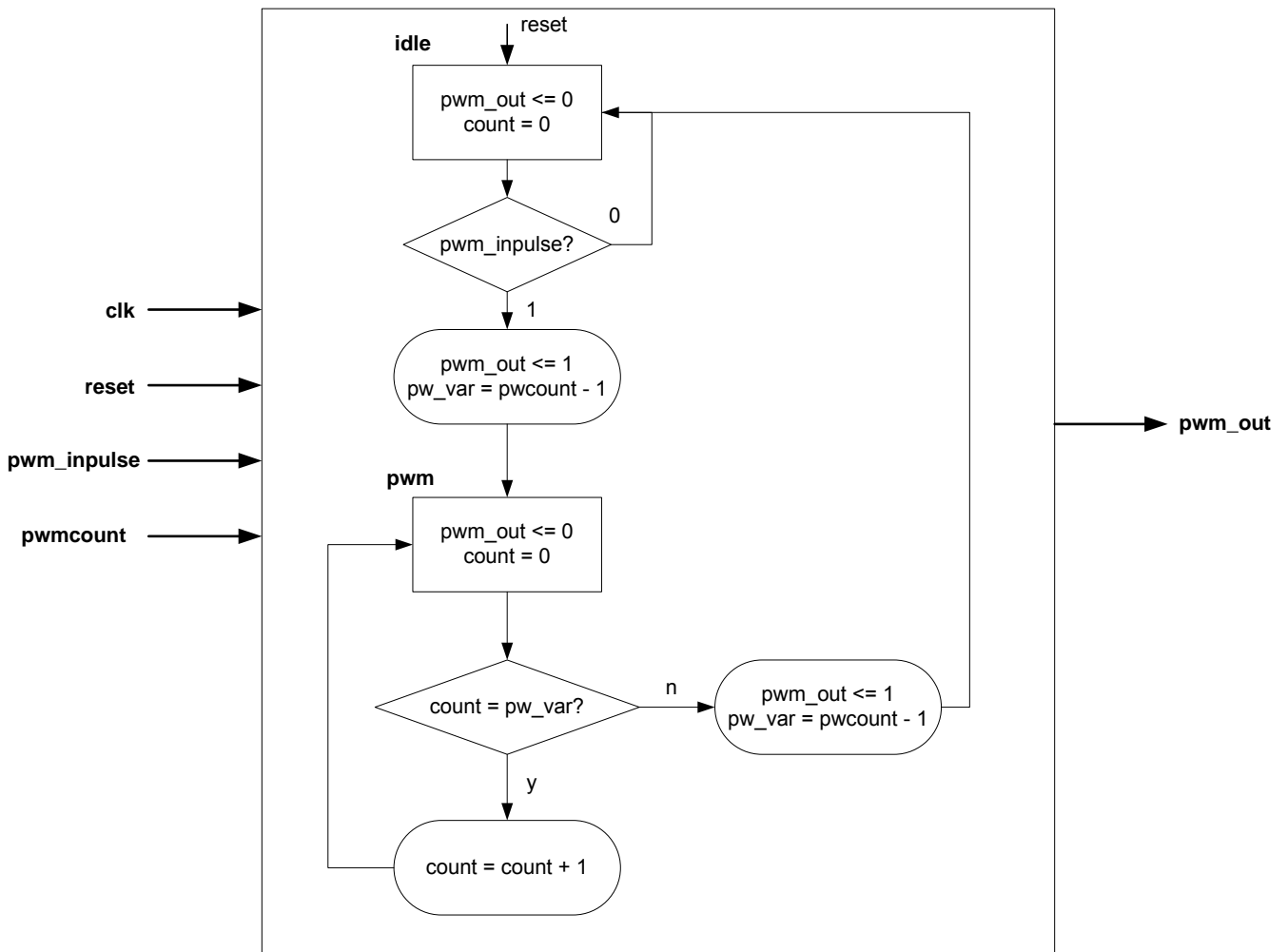


Figure 5.7 - Block diagram of the PWM module with ports and ASM shown.

Table 5.4 - PWM module port descriptions

Port	Direction	Description
clk	in	This provides the clock to the PWM module. It can be calculated as shown above. The clock period is also equal to the resolution of the output pulse.
reset	in	It provides the module with an asynchronous reset. It initializes the state machine and signals.
pwm_inpulse	in	This triggers the start of the pulse output.
pwmcount	in	This register sets the duration of the pulse. It can be set to a value between 0 and 8191. A further explanation is given above.
pwm_out	out	This is the line on which the pulse is generated and can be connected to a servo.

### 5.3 PWM CAPTURE MODULE

The PWM Capture Module is required for connection of the system with an RC receiver. The receiver usually outputs pulses to control servos as described in Section 5.3 i.e. pulses between 600us and 2.4ms long. This module needs to capture these pulses. The block diagram below shows the algorithmic state machine implementation of this module.

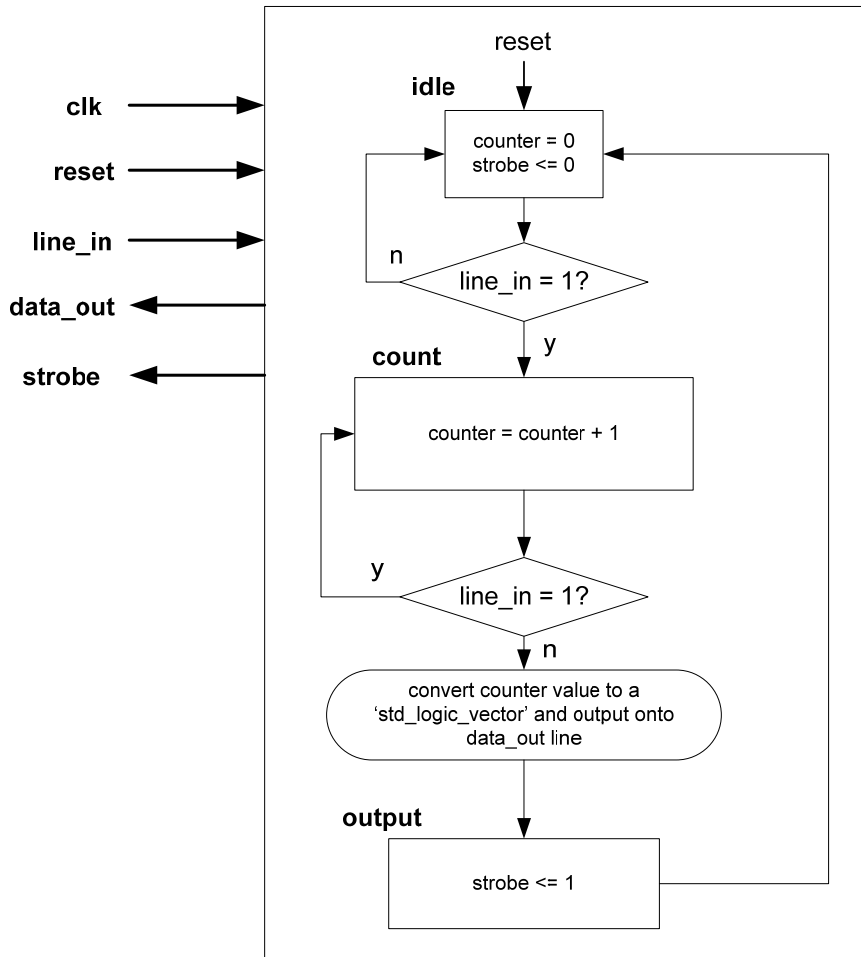


Figure 5.8 - PWM Capture module ASM and ports.

The module starts the capture process as soon as a pulse is received. It operates by counting the number of clock cycles between the start of the pulse and the end. The period of the clock thus determines the resolution of the signal capturing. The frequency of the clock fed into the module is thus an important factor. The data value outputted onto the data\_out line at the end of a pulse capture is used to calculate the pulse length by the following formula:

$$t_{pulse} = t_{clk} \times (data\_out\ value) \quad (5.3)$$

The strobe line is pulsed after the captured data value has been outputted on the data\_out line. The result for the simulation of this module is also shown in Chapter 7. The table which follows summarizes the input and output ports of the PWM Capture module

**Table 5.5 - Port descriptions of the PWM Capture Module.**

Port Name	Direction	Description
clk	in	This port provides the PWM Capture module with the clock which controls the synchronous operation of the state machine.
reset	in	This asynchronously resets the module.
line_in	in	This is the input line on which the pulse to be captured is received.
data_out	out	The data value from the counter which is output on through this port.
strobe	out	This line creates a short strobe pulse to indicate that the module has finished capturing the pulse and the data is ready to be read.

## 5.4 UART MODULE

### 5.4.1 UART Protocol Description

The Universal Asynchronous Receiver/Transmitter (UART) is a serial communications protocol used to transfer data. It allows data to be transferred in both full- and half-duplex modes of communication. Data is sent in bursts of characters which are framed as shown in the figure below.



**Figure 5.9 - UART Transmission Character Frame [34]**

As it is an asynchronous protocol, the UART interface contains a set of parameters which must be agreed upon and set by both devices before transmission can take place. These parameters are:

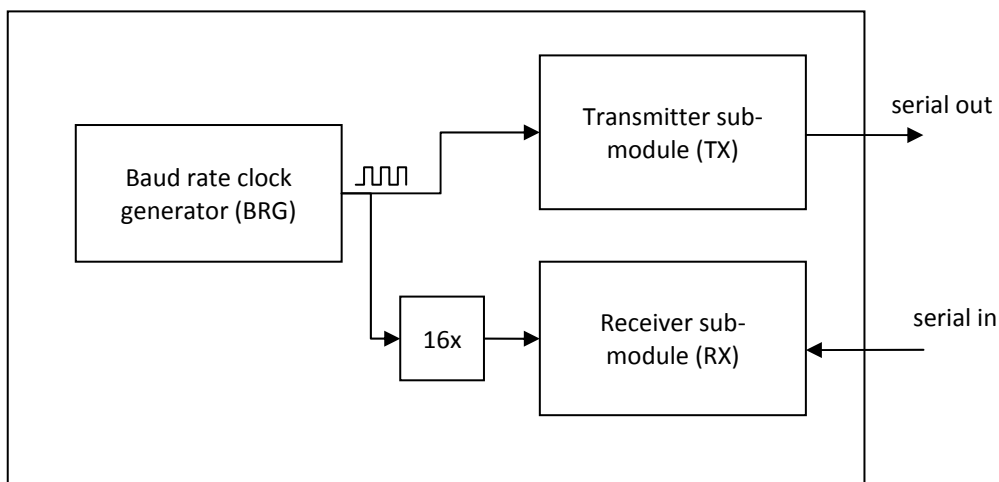
- The baud rate i.e. the speed of data transmission in bits-per-second.
- The number ( $n$ ) of data bits in a character frame. This can be between 5 and 8 bits.
- Odd, even or no parity. If parity is considered, the parity bit is included after the data bits.
- The stop bit length of 1, 1.5 or 2 bits.

Each UART module within a device has a transmitter module and a receiver module. The simplest form of UART interface is formed by connecting the transmitter of one UART module to the receiver of the other. When the UART is idle, the transmitter module keeps the line high. To indicate the beginning of a transmission frame, the transmitter sends the start bit. Transmission of the data bits then takes place. The transmitter then sends the appropriate number of stop bits to indicate to the receiver that the transmission is complete.

The baud rate parameter is important in UART communication as there is no clock to govern the process. The transmitter sends data at the baud rate. The receiver module uses the baud rate parameter to oversample the signal line, usually 16 times faster than the baud rate, and determines if a start bit has been sent. It then samples the data bits according to the agreed parameter (number of data bits  $n$ ). It then samples the stop bits to determine that the transmission is correct.

### 5.4.2 UART Module Requirements

In order to implement the UART Module within an FPGA, it can be seen from the protocol description above that transmitter, receiver and baud rate generator sub-modules are required. A block diagram of the required sub-modules is shown in the figure below. The baud rate generator creates a clock which is used by the transmitter to send data and by the receiver to oversample (16x) the incoming signal.



### 5.4.3 UART Module Implementation

The UART Module implementation can be seen in the figure below. The UART sub-modules (BRG, TX and RX) for this project were taken from [35]. The VHDL code was used and modified appropriately.

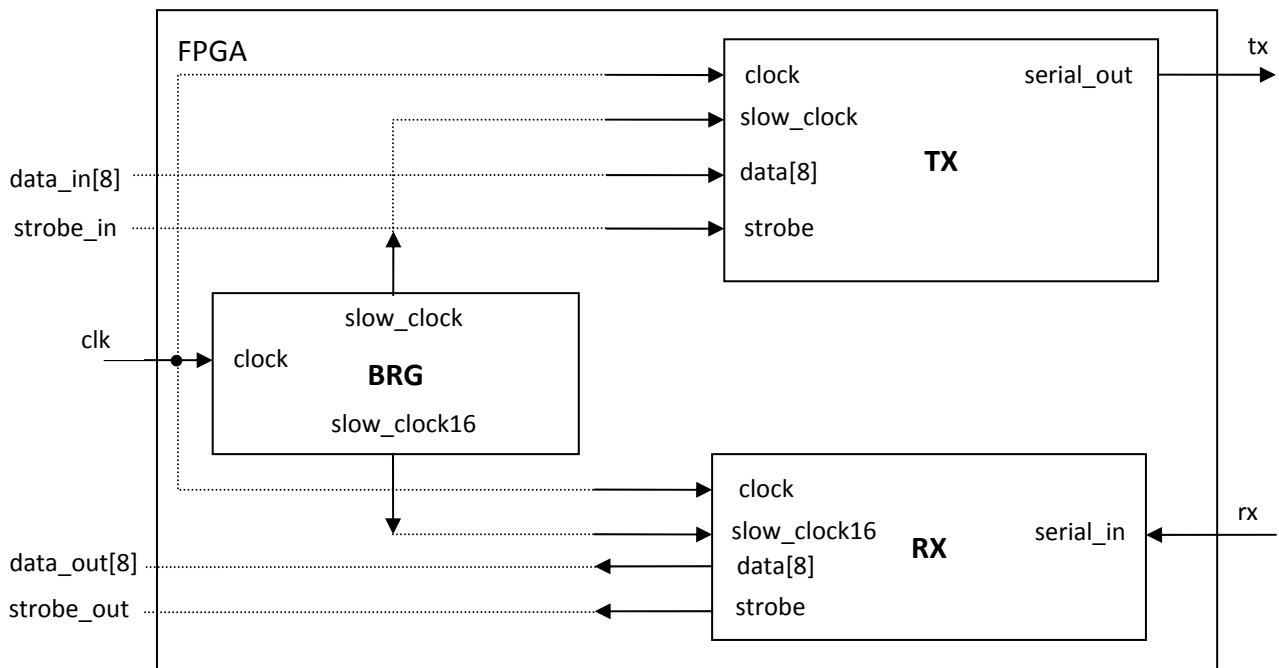


Figure 5.10 - UART Module Implementation

The individual sub-modules (BRG, TX and RX) work together to create the UART implementation. The functioning of each sub-module is described below:

**BRG sub-module:** The BRG basically consists of counters which count the number of clock cycles. When a certain number of cycles have past, it generates a pulse on the `slow_clock16` line. This is the faster signal generated. Furthermore, the number of `slow_clock16` pulses are counted and after 16 of these, a pulse is outputted on the `slow_clock` line. The simulation results of this can be seen in Chapter 7.

For proper baud rate generation, the number of clock cycles to be counted (*count*) to generate the `slow_clock16` pulses is calculated as follows:  $count = 1/16 \times (t_{bit}/t_{clk})$ , where  $t_{bit}$  is the period for one bit transmitted (i.e.  $1/BaudRate$ ).

The ports of the BRG sub-module are shown in the table below.

Table 5.6 - UART BRG sub-module ports and descriptions

Port Name	Direction	Description
clock	in	Provides the module with the clock used to generate the baud rate pulses.
slow_clock	out	Provides the baud rate pulses needed by the transmitter to transmit data correctly.
slow_clock16	out	This port provides pulses at a rate 16 times faster than the <code>slow_clock</code> pulses. It is used by the receiver to sample the incoming bits.

**TX sub-module:** The state machine for the TX sub-module is shown below. It basically waits in an idle state until the strobe signal is pulsed. During the idle state, it is responsible for keeping the serial output line high, as stated in Section 5.5.1. Once the strobe signal is pulsed, it readies the character frame to be sent out. The start bit and one stop bit is inserted into the frame as well as the 8-bit data to be transferred. It then transits to the transmit state.

In the transmit state, each time a `slow_clock` pulse is received, one of the frame bits are outputted. A counter is used to count the number of bits sent and return to the idle state once the transmission of the frame is complete. The simulation of this sub-module is also discussed in Chapter 7 and its correct functioning verified.

The ports of the TX sub-module are shown in the table below, followed by an illustration of the ASM.

**Table 5.7 - UART TX sub-module ports and descriptions**

Port Name	Direction	Description
clock	in	This is the clock is responsible for the state machine operation.
slow_clock	in	This input line provides the TX sub-module with the baud rate pulses. This is used to transmit the data.
data[8]	in	Contains the data byte to be sent out serially.
strobe	in	This line is pulsed to indicate the start of a UART transmission.
serial_out	out	The data is sent out serially on this line.

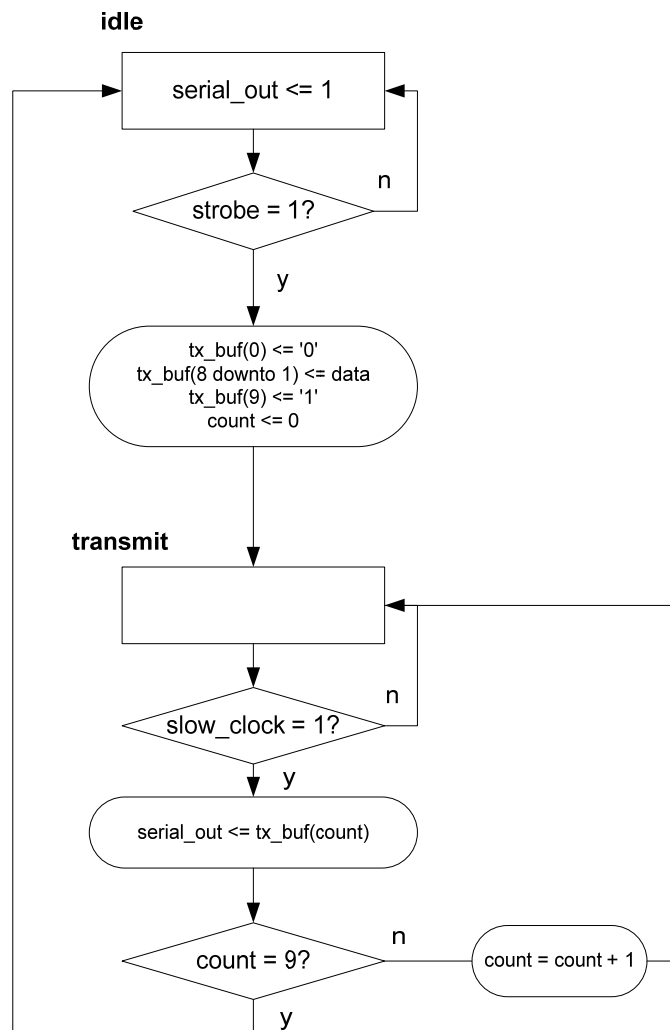


Figure 5.11 - UART TX sub-module ASM

**RX sub-module:** The state machine for the RX sub-module is shown in the figure below. The RX sub-module samples the serial\_in line on every slow\_clock16 pulse while in the idle state. Once it detects a low signal, it transitions to the start\_bit state. The midpoint of the start bit is then located by counting eight slow\_clock16 pulses. This is the reference point used to sample the rest of the incoming bits in the next state (data\_bits). In this state, 16 slow\_clock16 pulses are counted each time and the data sampled. This ensures that data is sampled on or close to the midpoint of the bit each time. The data is packed into a receive buffer (rxbuf) and after the final data bit is received, the state machine transits to the next state (stop\_bit). In this state, another 16 slow\_clock16 pulses are counted until the data in the buffer outputted through the data port. The strobe line is also pulsed to indicate that the data is ready. The functioning of this sub-module is shown in the simulation results in Chapter 7.



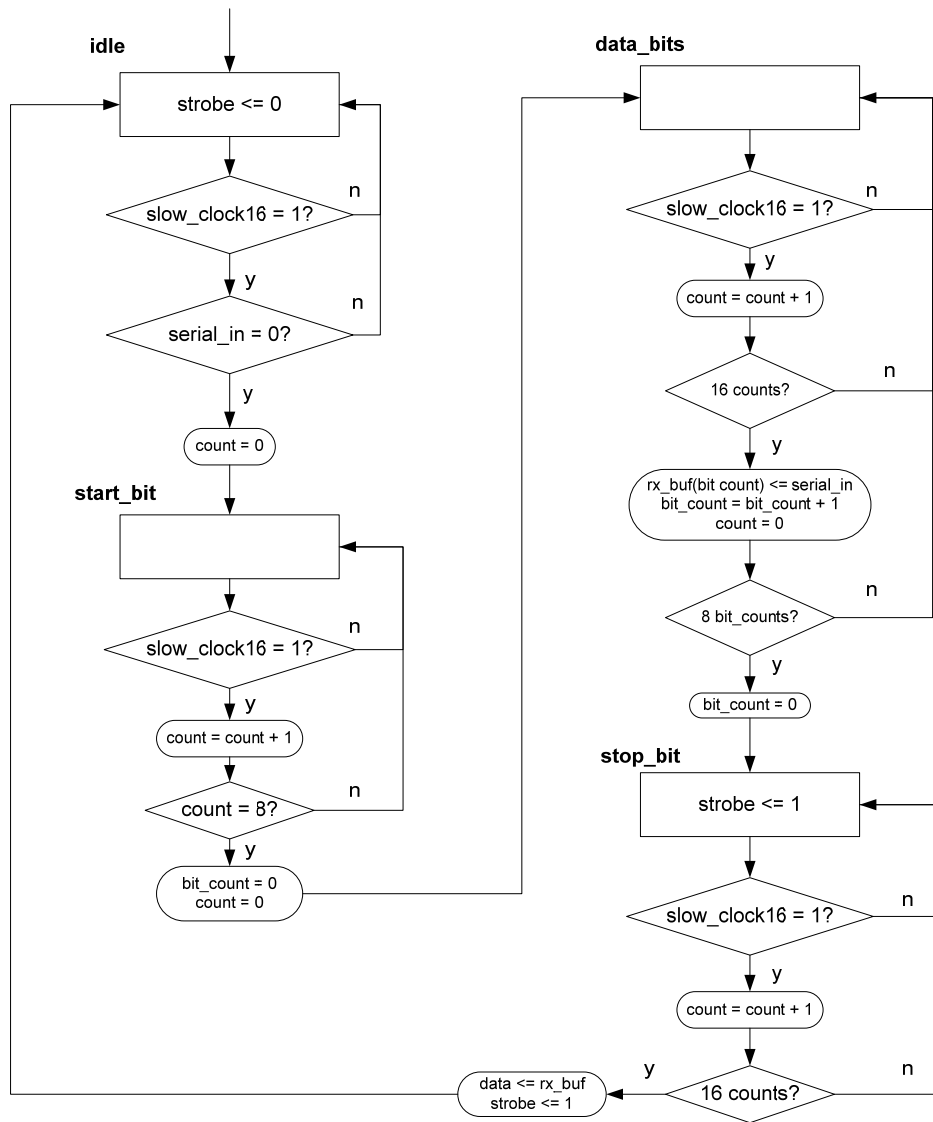


Figure 5.12 - UART RX sub-module ASM

The table below describes the ports of the UART RX sub-module.

Table 5.8 - UART RX sub-module ports and descriptions

Port Name	Direction	Description
clock	in	This is the clock is responsible for the state machine operation.
slow_clock16	in	Provides pulses which are 16 times faster than the baud rate to the RX sub-module.
serial_in	in	Receives the data sent during UART transmission.
data[8]	out	Contains the data byte received during UART communication.
strobe	out	Indicate that the data reception is complete and the data is ready.

## 5.5 DUAL PORT RAM

A memory module is required to hold all the data within the FPGA. It was decided to use the built-in dual port RAM module in the LPM (Library of Parameterized Module) functions of the Altera Quartus II development software (called the *altsyncram* megafunction). A block diagram of this module is shown in the figure below.

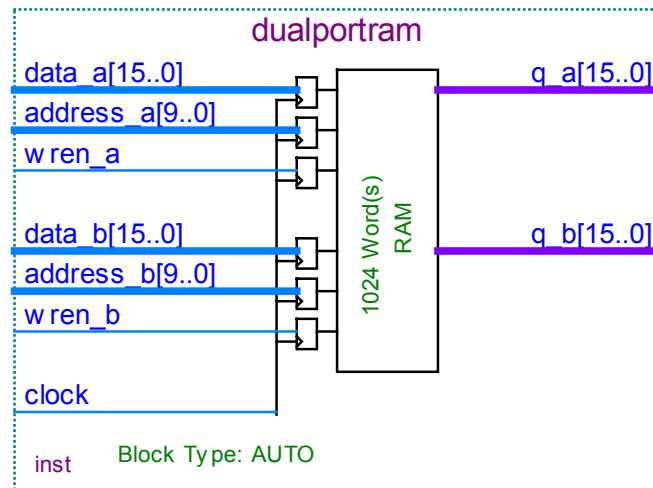


Figure 5.13 - Dual Port RAM

Each port (A and B) has access to 1024 words of 16-bit data. The waveforms below show the behavior of the dual port RAM (DPRAM) module.



Figure 5.14 - Read cycle behavior of the DPRAM module [36]

Data is clocked out on every rising edge, as can be seen in the waveform in Figure 5.14 above. The value of the data is dependent on the address input to the DPRAM. The write enable signal (wren)

of each port controls the write operation to the memory registers in the RAM. The write cycle starts on the rising edge of the clock when wren is high and ends on the next rising edge. The data is however written to the memory registers on the falling edge, as seen at marker 1 below. When the wren signal is high, the data on the output ports q are the same as the data on the input ports (see marker 2). When one port tries to read from a register being written to by the other port, the output is unknown (see marker 3).

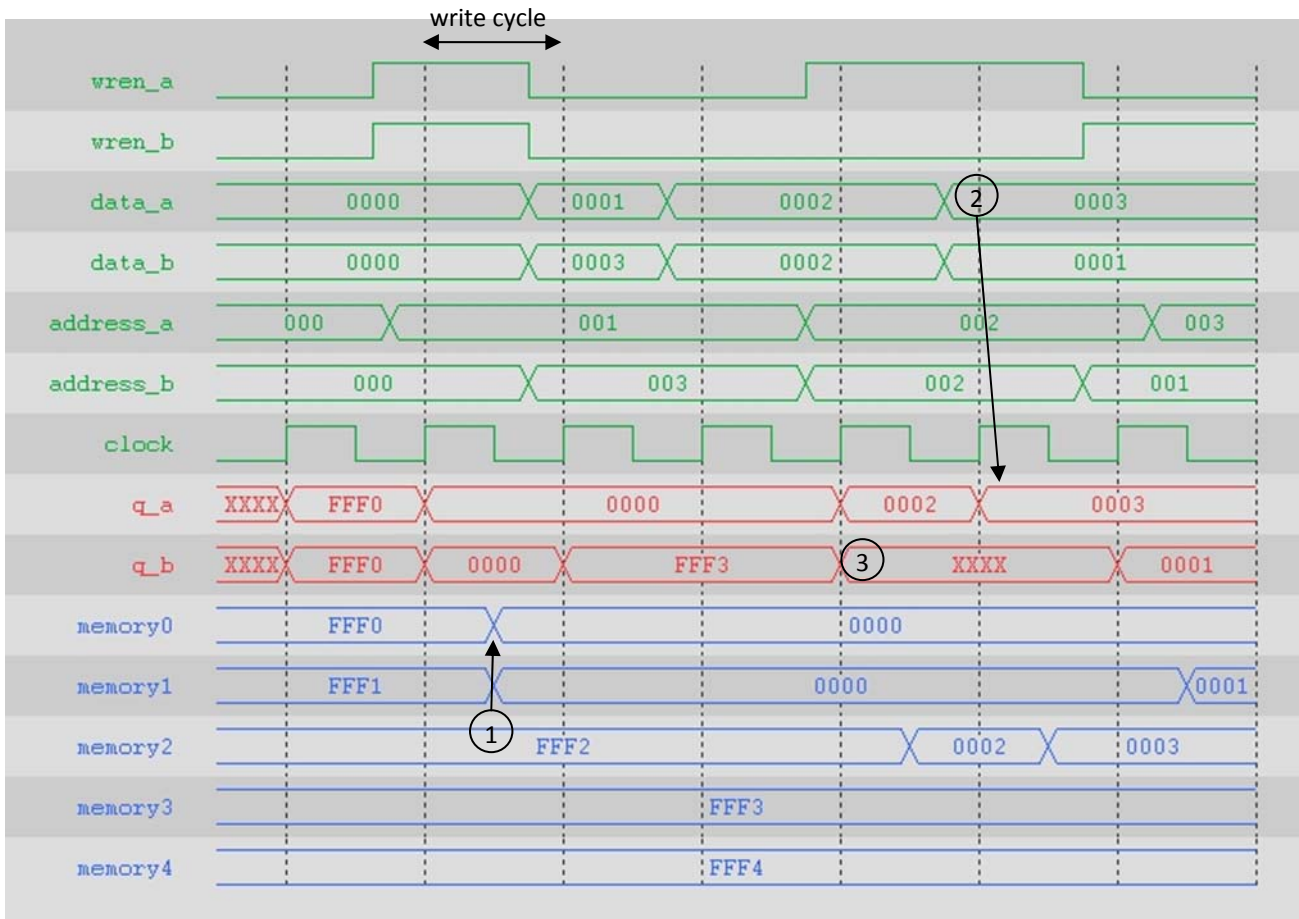


Figure 5.15 - Write cycle behavior of the DPRAM module [36]

## 5.6 BUS INTERFACE CONTROLLER MODULE

The Bus Interface Controller (BIC) module is responsible for controlling the parallel bus interface between the FPGA system and the SH7201 microprocessor. In order to develop this module, an understanding of the bus protocol on the microprocessor side is required. In the SH7201 microprocessor, the Bus State Controller (BSC) peripheral module is responsible for controlling the data and control signals of the bus. The relevant details of the BSC will be discussed in this section while a full detailed description of its features and functioning can be found in the datasheet [37]. The figure below shows a block diagram of the SH7201 BSC and the FPGA module connections for

this system (the additional signals of the BSC are not shown as they are not used). The BIC and DPRAM modules work together and form a subsystem. The SH7201 essentially sees the FPGA as “memory” and this abstracts it from the internal operations of the FPGA modules.

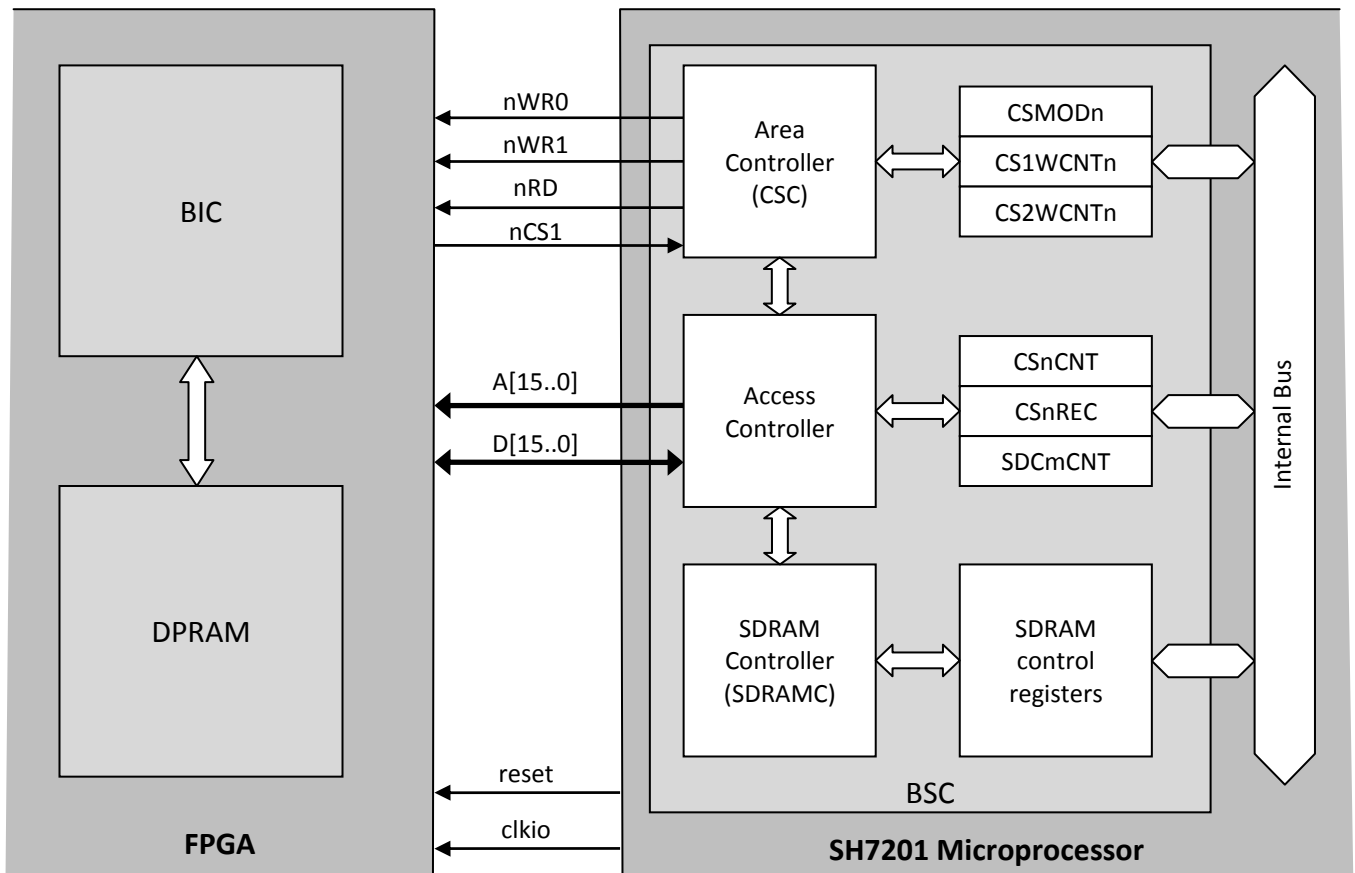


Figure 5.16 - The SH7201 BSC

The BSC allows various memory storage devices and external devices to be connected to the SH7201. It is flexible and can be configured in various ways, especially in terms of data operation and bus signal timing. The use of wait-states is employed to achieve different data and control timing waveform configurations. All these configuration settings are done via registers within the microprocessor. For this application, the normal read/write operation mode is chosen and this is explained in the figures and discussion below [37].

## BSC Read Operation (Normal Access Mode):

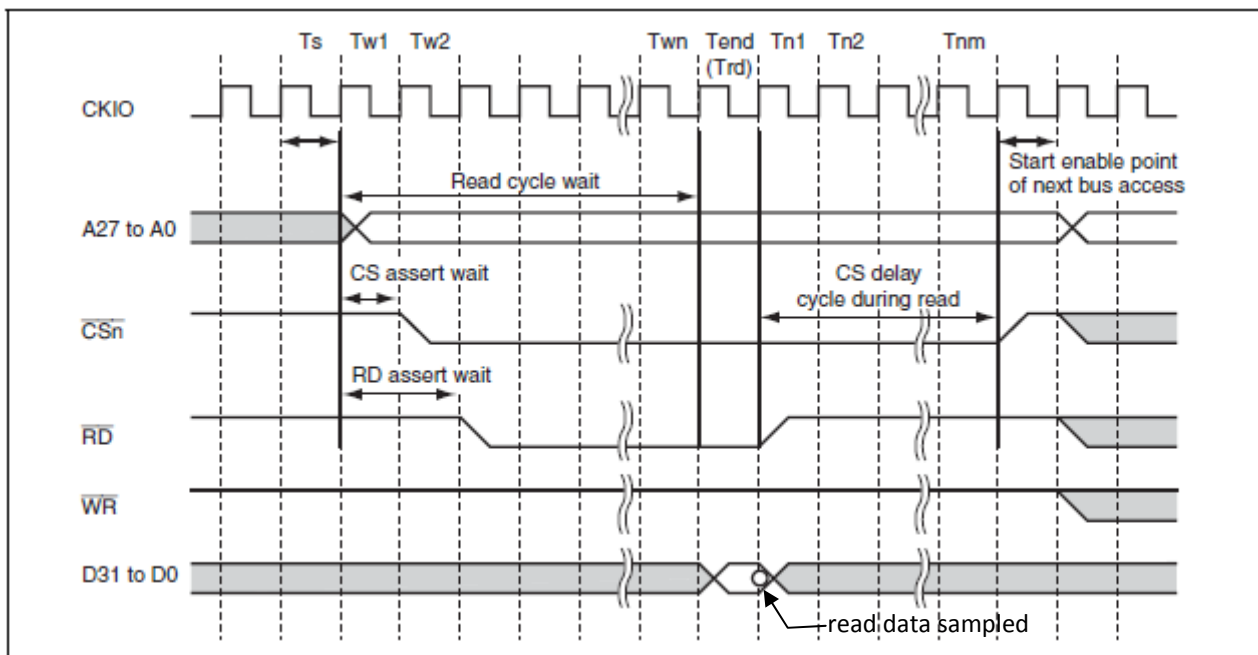


Figure 5.17 - BSC Read Operation Basic Bus Timing (taken from [37])

- 1) **Ts**: the start of the external bus access cycle. The address signals are set in the next clock cycle.
- 2) **Tw1 to Twn**: this is the number of read cycle wait-states selected by setting the CSRWAIT register (0 to 31 clocks). For this application, the maximum value of 31 clocks is chosen. The nCS and nRD signals are driven low (asserted) according to the number of wait-state set in the CSN and RDON registers respectively (0 to 7 clocks).
- 3) **Tend/Trd**: this is the end of the read cycle wait period and also the read sampling cycle (Trd). The data to be read needs to be ready during this cycle and is sampled or read on the next clock edge, as indicated in the figure above. It also needs to remain stable with the required setup and hold times around the sampling edge. The nRD signal is also negated high at the end of this cycle.
- 4) **Tn1 to Tnm**: this is the number of clock cycles (0 to 7 clocks) by which the nCS signal is delayed after the wait end cycle (Tend) before being driven high (negated) again. It is set in the CSROFF register. The next bus access cycle can start after this period.

## BSC Write Operation (Normal Access Mode):

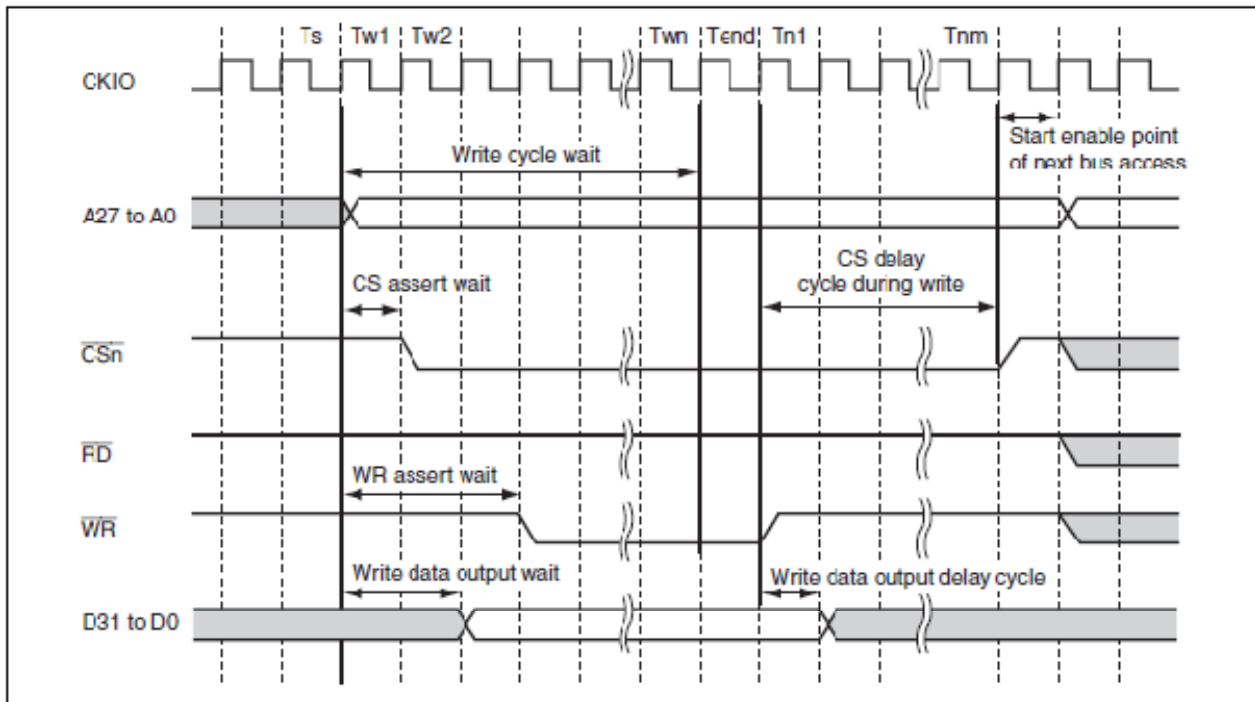


Figure 5.18 - BSC Write Operation Basic Bus Timing (taken from [37])

- 1)  **$T_s$ :** the start of the external bus access cycle. The address signals are set in the next clock cycle.
- 2)  **$T_{w1}$  to  $T_{wn}$ :** this is the number of write cycle wait-states selected by setting the CSWAIT register (0 to 31 clocks). For this application, the maximum of 31 clock cycles was chosen. The  $nCS$  and  $nWR$  signals are asserted low according to the value set in the CSO<sub>N</sub> and WRON registers respectively (0 to 7 clock cycles). The data is also outputted during this cycle according to the number of write data output wait-states selected, set in the WDON register.
- 3)  **$T_{end}$ :** this is the end of the write cycle wait period. The  $nWR$  signal is negated high at the end of this clock cycle.
- 4)  **$T_{n1}$  to  $T_{nm}$ :** this is the number of clock cycles (0 to 7 clocks) by which the  $nCS$  signal is delayed after the wait end cycle ( $T_{end}$ ) before being driven high (negated) again. For the write operation, this is set in the CSWOFF register. During this period, the data signal negation can also be controlled by the setting the write data output delay register (WDOFF) to a value between 0 and 7 cycles. After this period ( $T_{n1}$  to  $T_{nm}$ ), the bus is available for the next access cycle.

Furthermore, care must be taken during the wait-state selection as there are certain conditions which govern the wait-state settings:

- $CSON \leq \text{minimum}(CSRWAIT, CSWWAIT)$
- $RDON \leq CSRWAIT$
- $WRON \leq CSWWAIT$
- $WDON \leq CSWWAIT$
- $WDOFF \leq CSWOFF$
- $RDON < CSON$  ( This is an extra condition derived for this specific system as the state machine developed triggers on the nCS signal and therefore the nRD signal must be stable before this)
- $WRON < CSON$  (Derived for the same reason as stated directly above but with respect to the nCS and nWR signals)

To achieve communication with the microprocessor on the external bus, the BIC module was designed and implemented in VHDL. It is basically the interface between the external bus and the dual port RAM. The algorithmic state machine for the BIC module is shown below, along with the module input and output ports. A description of each port is given in Table 5.9.

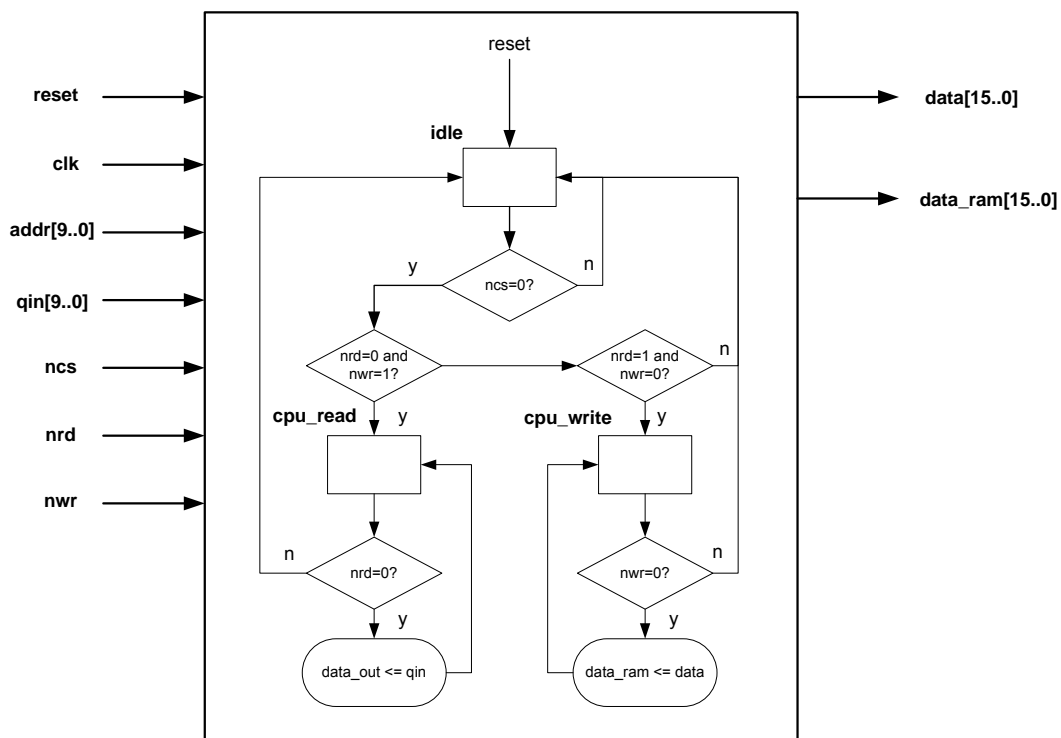


Figure 5.19 - Bus Interface Controller ASM

The state machine remains in the idle state (after a reset) until the ncs signal from the microprocessor is asserted low. It then checks the control signals nrd and nrw to determine if a read or a write cycle is being executed by the microprocessor. The microprocessor signals are also fed to the DPRAM, as can be seen in Figure 7.1, and thus the BIC operates in parallel with the DPRAM.

The data flow on the bidirectional external bus, connected to data[15..0], is controlled by the BIC and its functioning is dependent on whether a read or write cycle is to be executed. During a write cycle, the data received from the microprocessor is sent to the DPRAM through port data\_out while during a read cycle, the data requested by the microprocessor is received through the qin port and placed on the external bus.

**Table 5.9 - Port Descriptions of the Bus Interface Controller**

Port Name	Direction	Description
reset	Input	Asynchronous reset of the module
clk	Input	Provides the clock to the module and is responsible for the synchronous state transitions.
addr[9..0]	Input	This is the 10-bit wide address which comes from the microprocessor. It is used to read a specific data word.
data[15..0]	Bidirectional	This is the 16-bit port which connects to the microprocessor data bus signals to transfer the data.
data_ram[15..0]	Output	This is a 16-bit wide port and provides the connection to the DPRAM which stores the data received from the microprocessor.
qin[15..0]	Input	This is a 16-bit wide port and is connected to the DPRAM to get the data to be transferred to the microprocessor.
ncs	Input	Connected to the nCS signal of the microprocessor and triggers the data transfer operations.
nrd	Input	Connected to the nRD signal of the microprocessor and controls the read data transfer operation.
nwr	Input	Connected to the nWR signal of the microprocessor and controls the read data transfer operation.

The interaction between the Bus Interface Controller Module and the Dual Port Ram is shown in Figure 7.1 in Chapter 7. The test and simulation results are also shown here.



## 5.7 IMU CONTROLLER MODULE

### 5.7.1 ADIS 16350 Operation

In order to interface the ADIS 16350 (IMU) with the system through the SPI Master Module, a device-specific controller is required. The data sheet [25] of the IMU device was studied to determine the manner in which the device operates.

The IMU has internal registers which are mapped to a specific address. Each register consists of an upper and lower byte, each of which is addressable. Data gathered by the IMU can be read from the data registers. The device also has certain programmable features which are controlled by writing to the appropriate control registers. The control registers can also be read to determine their current settings. In order to implement this, two defined modes of operation are defined: a *normal mode* in which the IMU gathers data and it is read by the FPGA and a *configuration mode* in which the FPGA configures the control registers.

The operation of the SPI interface (mode 3) for the IMU is shown in Figure 5.20 below. Each SPI transmission frame is 16 clock cycles long and the first data bit received determines whether the IMU enters a read or write cycle.

During a write cycle, the 6-bit address of the register followed by an 8-bit data value can be written, as seen in Figure 5.20A. Note that for a write command, the first bit is set to zero. The second bit is always zero for all SPI sequences. In order to write to the both the upper and lower byte of a register (i.e. the entire 16-bits), two transmission frames are required.

For a read cycle, the 6-bit address is sent to the IMU. The frame bit sequence is shown in the Figure 5.20B. As each register has an upper or lower address, either can be used to read the full data content of the 16-bit register. During the next frame, the data will be outputted on the DOUT line.

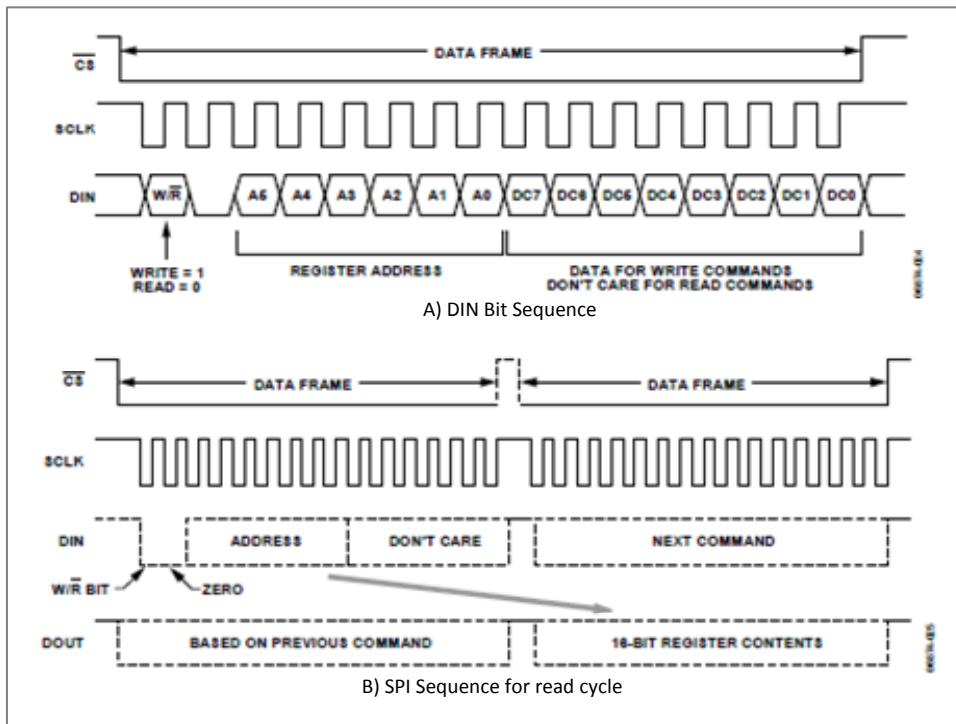


Figure 5.20 - The operation of the SPI Interface for the ADIS 16350 IMU (taken from [25])

### 5.7.2 Implementation

The IMU controller interfaces with the implemented SPI Master Module as shown in the figure below.

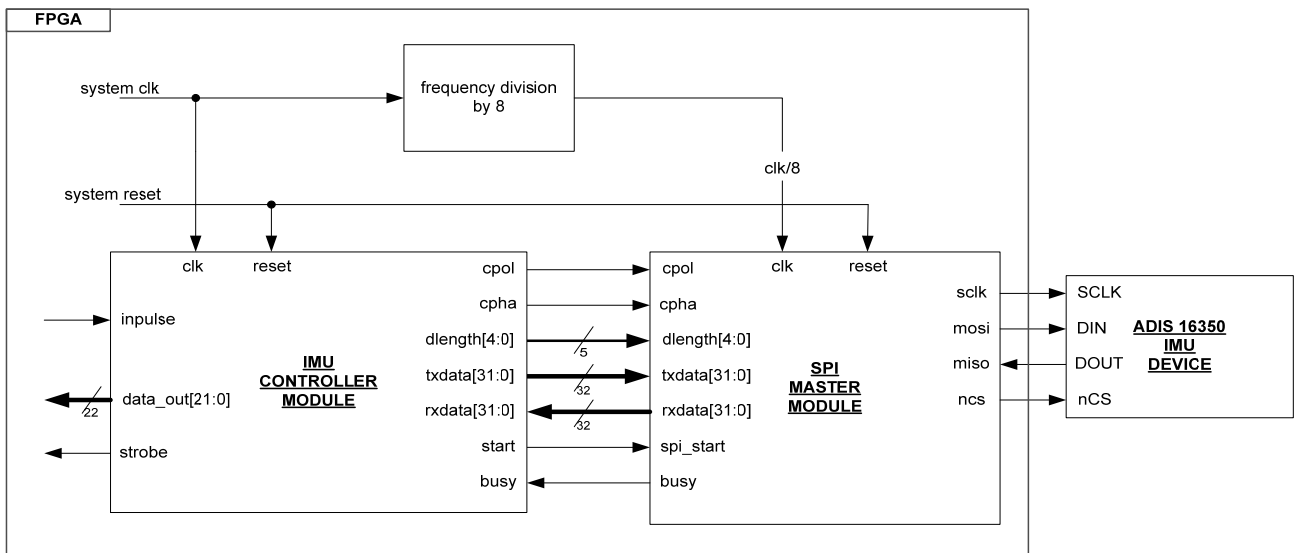


Figure 5.21 - Interface between the IMU Controller and SPI Master Module.

The algorithmic state machine (ASM) for the *normal mode* of operation for the IMU Controller is shown in Figure 5.22 below. A discussion of the ASM also follows.

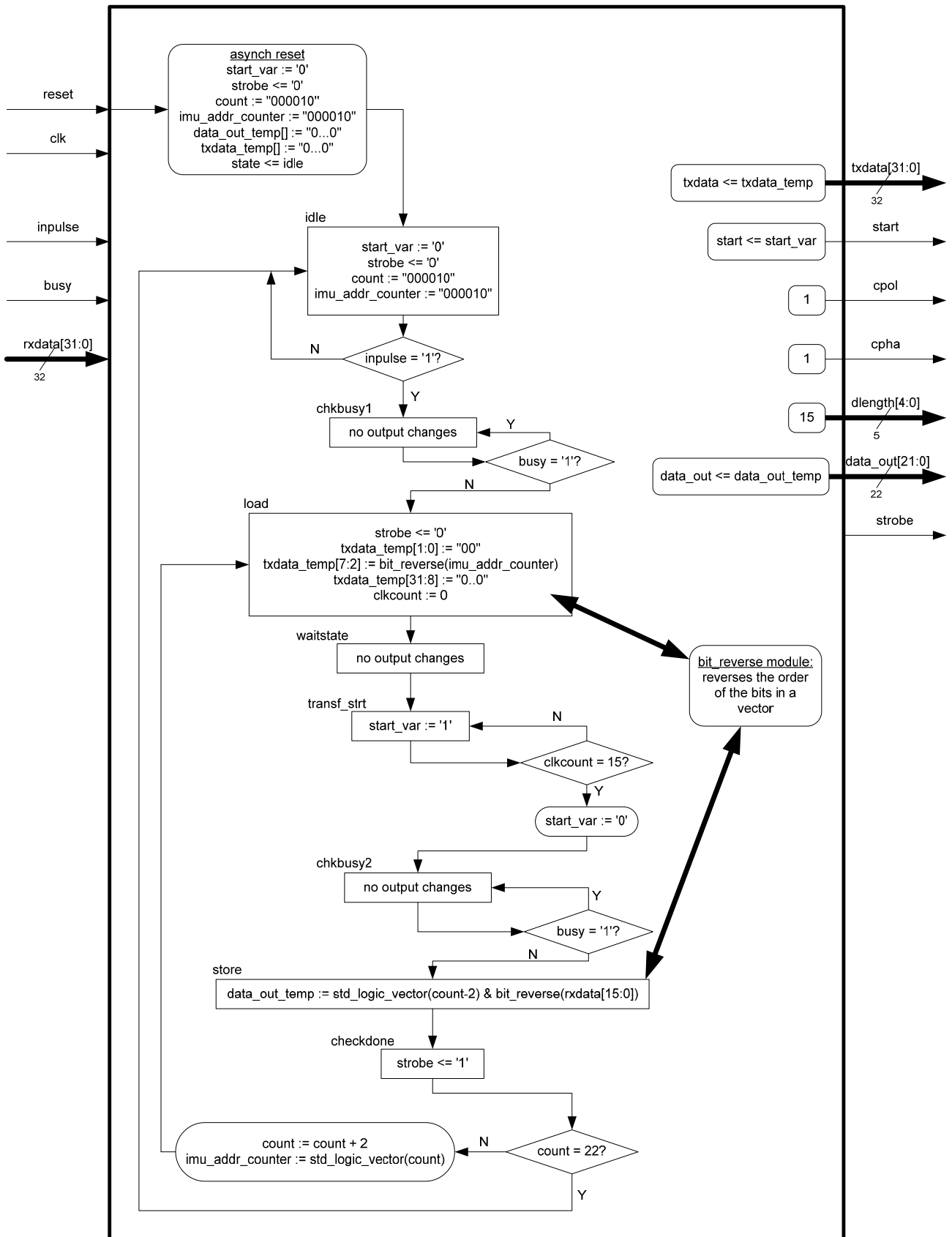


Figure 5.22 - Algorithmic State Machine of the IMU Controller Module (normal mode)

The ASM is asynchronously reset by asserting the *reset* line. This sets the output signals and the variables to their default values as shown in Figure 5.22 and also places the ASM in the *idle* state after reset. The ASM uses a pointer called *imu\_addr\_counter* to point to the correct register in the ADIS 16350 device. This pointer is always set to “000010” initially as a result of the register addressing explained above. A binary *count* variable is used to keep track of the amount of registers read in the device and this is also set to “000010”. The *start* and *strobe* signals are also asserted low in this state. The ASM remains in the *idle* state until the *inpulse* line is asserted.

The ASM then transitions to the *chkbusy1* state in which it checks that the SPI Master Module is not currently busy.

It then enters the *load* state. In this the *txdata* register is loaded with the correct current addressing value for SPI transmission to the IMU device. The SPI Master Module accepts a data word through its *txdata* port with the least significant bit first and the most significant bit last. It is for this reason that the data word is first subjected to the *bit\_reverse* function in order to reverse its bit order.

The ASM then transits to a wait state and this is done to ensure that all the relevant registers have finished loading before the SPI transmission is triggered.

The *transf\_strt* state is then entered in which the *start* signal is asserted to begin the SPI transfer. As alluded to earlier, the current ASM and the SPI Master Module are fed by clocks of different frequencies and it is thus imperative that the *spi\_start* signal is held high for the correct duration. As seen in Figure 5.21, the SPI Master Module runs eight times slower than the IMU Controller Module. The *spi\_start* signal is thus held high for 16 IMU Controller Module ASM clock cycles which equates to two SPI Master Module SM clock cycles.

The ASM then enters the *chkbusy2* state in which it waits until the SPI transfer has completed by checking the status of the *busy* line.

The ASM then transits to the *store* state in which the data is outputted by the module in the format shown in the table below.

Table 5.10 - 22-bit data word format outputted by the IMU Controller Module's *data\_out* line

Bit:	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	6-bit Address[5:0]						16-bit Data[15:0]															
	0x00						Auxiliary Analog Input Data															
	0x02						Power Supply Measurement															
	0x04						X-axis Gyroscope Output Measurement															
	0x06						Y-axis Gyroscope Output Measurement															
	0x08						Z-axis Gyroscope Output Measurement															
	0x0A						X-axis Acceleration Output Measurement															
	0x0C						Y-axis Acceleration Output Measurement															
	0x0E						Z-axis Acceleration Output Measurement															
	0x10						X-axis Gyroscope Sensor Temperature Measurement															
	0x12						Y-axis Gyroscope Sensor Temperature Measurement															
	0x14						Z-axis Gyroscope Sensor Temperature Measurement															

A strobe signal, which is asserted for one clock signal in the next state, namely the *checkdone* state, is used to signal that the data on the output port *data\_out* is ready. This state also evaluates the counter to determine whether the next ASM frame should begin or the ASM should return to the *idle* state. One ASM frame returns one 16-bit data value from the IMU device and there are 11 frames in all. The figure below shows the data flow during these frames.

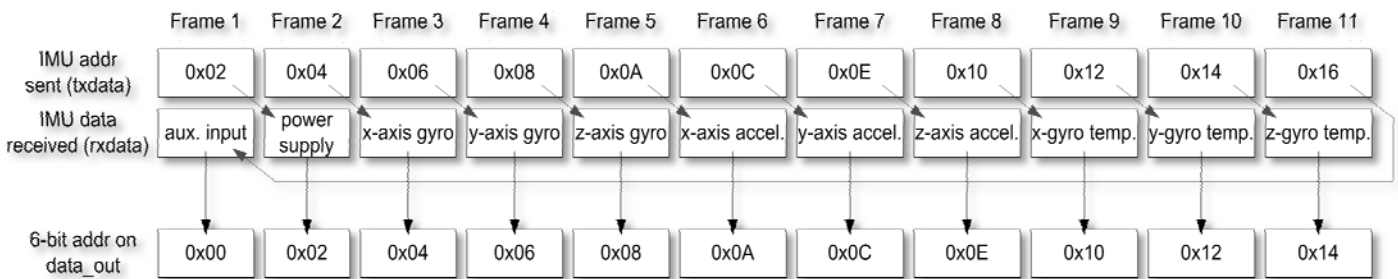


Figure 5.23 - Data flow of the transaction between the IMU Controller Module and the ADIS 16350 IMU device during one complete cycle (11 frames)

As explained earlier, the IMU device outputs data based upon an address sent to it in the previous frame. A frame consists of a complete SPI transaction between the IMU device and the IMU Controller state machine. Each time the *inpulse* line is triggered during the *idle* state, the ASM starts a cycle which consists of 11 data frames. In each frame it sets up the address of the register to be read in IMU and sends it via SPI to the device. Simultaneously data based upon the previous

address requested in the previous frame is sent to the IMU Controller Module in the FPGA. The data is then stored to the address shown in Table 5.10 and in Figure 5.23. In frame 1, the data received is the contents of the last register address sent to the IMU device and this is address 0x16 i.e. the last address set up before the ASM exits the cycle and enters the *idle* state again. Upon an IMU device reset, this data will be an unknown value as the IMU Controller Module would not have completed a full cycle yet. Therefore, the very first time it enters frame 1, an unknown value will be received. This is not significant as this value is the contents of the auxiliary analog input data register and this is not used in this project.

The *configuration mode* of the IMU Controller Module was not implemented as time did not allow for this. A conceptualization of an ASM containing both *normal and configuration* modes are shown below. It uses a *mode* value of either 0 or 1 to determine the respective mode for the IMU Controller Module.

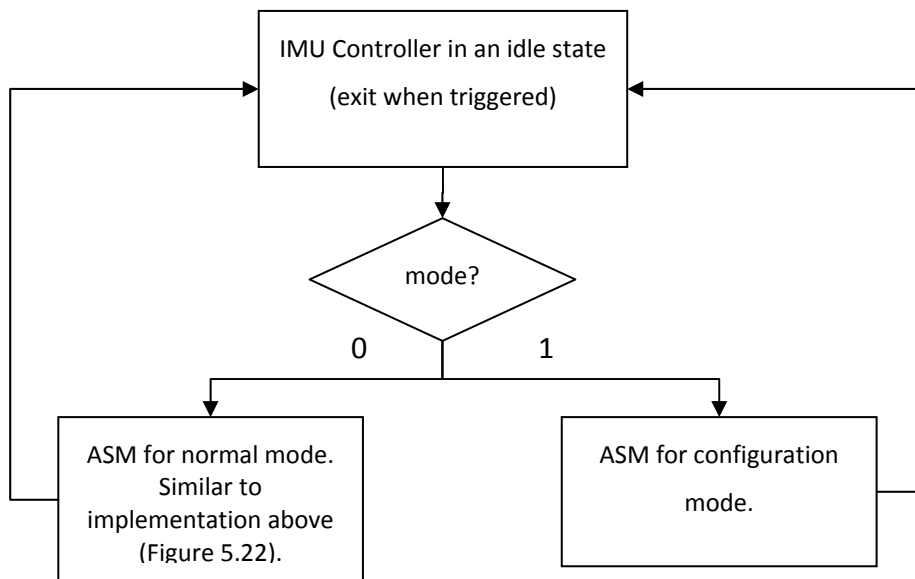
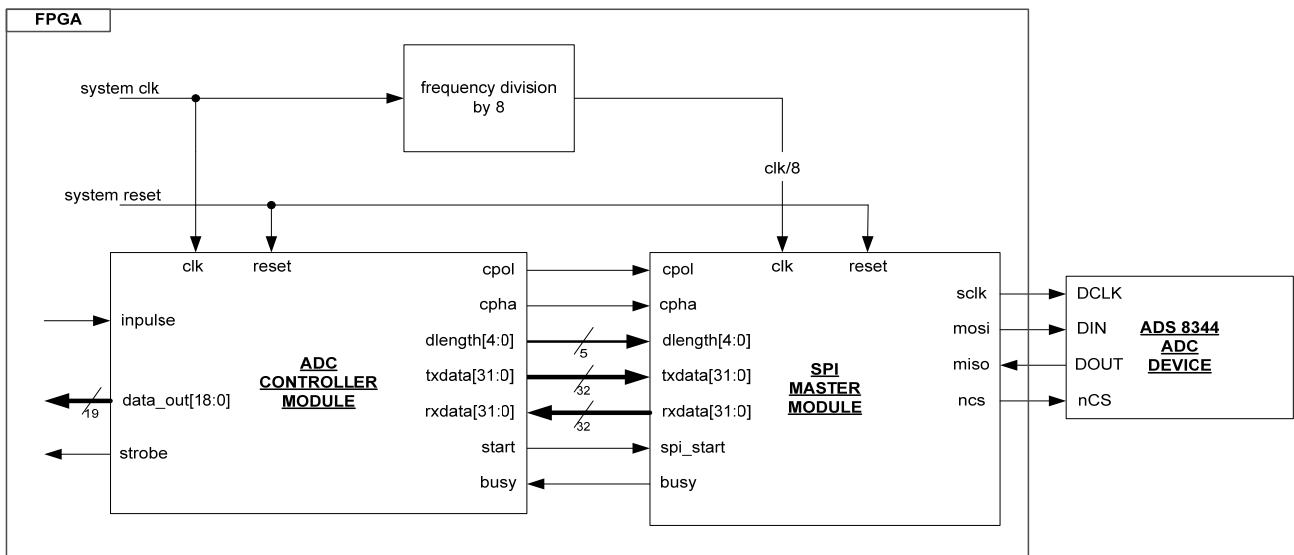


Figure 5.24 - Conceptual ASM of the IMU Controller Module for both modes (normal and configuration)

## 5.8 ADC CONTROLLER

The operation of the ADS8344 ADC device also uses SPI to communicate. A controller is thus needed to operate in conjunction with the SPI Master Controller implemented in this project to interface with the device, as shown in Figure 5.25 below. This is the function of the ADC Controller.



**Figure 5.25 - High level block diagram showing the interface between the SPI Master Module and the SPI ADC Controller Module.**

A description of the manner in which the ADC device operates can be found in the data sheet [38]. The device is to be configured in External Clock Mode and the frame sequence for SPI transmission is shown Figure 5.26 below. The SPI clock signal is used to perform the analog-to-digital conversion.

An ASM for the controller was designed and implemented. It basically operates by waiting for a pulse signal to trigger the beginning of a cycle. It then uses the SPI Master Module to send the addresses of the channels to the ADC device which returns the data. After each channel data received, the ADC Controller module outputs the data (with an address) and a strobe signal to indicate the data is ready. It can then be stored in memory registers. After all the channels have been received and outputted by the ADC Module, the ADC Controller module returns to the idle state and waits for the next cycles to be triggered.

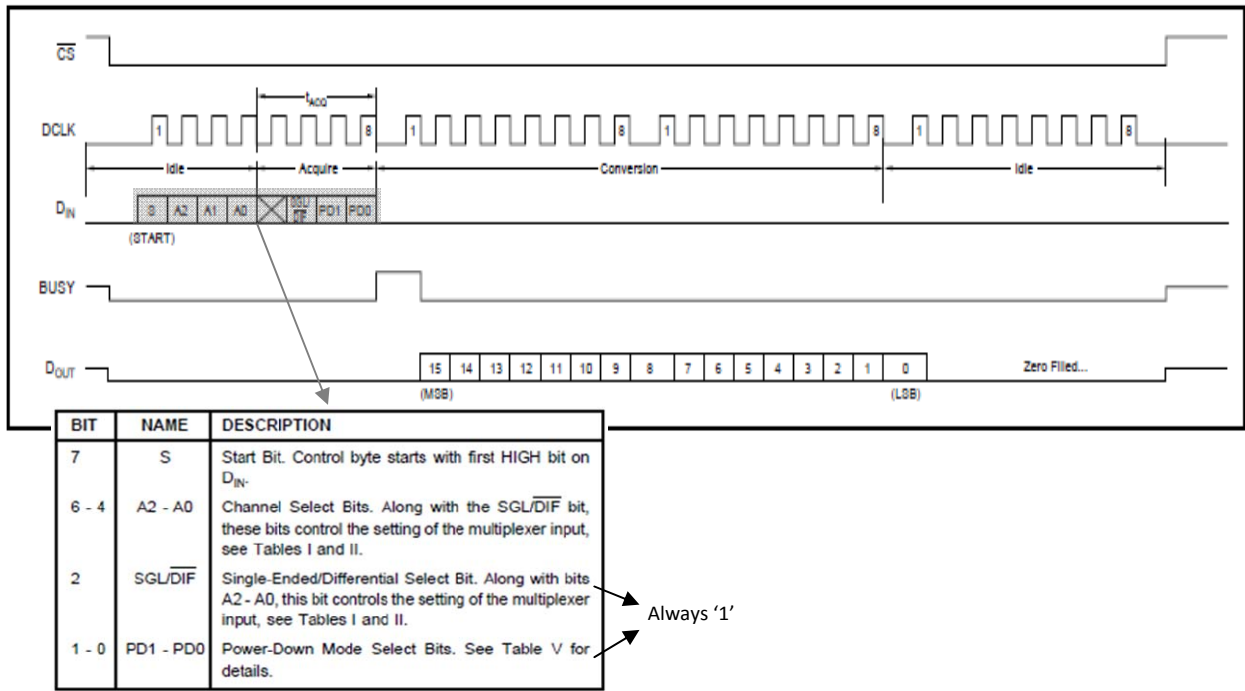


Figure 5.26 - SPI frame of the ADS8344 ADC device (taken from [38])

The data outputted on the data\_out line is identified by an address to indicate which channel data of the ADC is being sent. This is shown in the table below.

Table 5.11 - Identifier on data\_out line of ADC Controller

ADC Channel	Identifier Address
CH0	0x00A
CH1	0x00B
CH2	0x00C
CH3	0x00D
CH4	0x00E
CH5	0x00F
CH6	0x010
CH7	0x011

The VHDL implementation of the ADC Controller was completed and simulated together with the SPI Master Module to verify that it functions correctly. The simulation results can be seen in Chapter 7.



## 5.9 GPS CONTROLLER

The LEA-4H GPS sensor is to be configured to operate using the u-blox proprietary UBX protocol. The details of which can be found in [14] and [39]. The relevant details were studied and an ASM implemented to interpret/parse the protocol. Only the NAV-POSLLH, NAV-STATUS and NAV-VELNED messages are to be implemented as this provides all the necessary information required. The GPS Controller is intended to be used together with the UART Module to interface with the GPS sensor.

As the GPS sensor does not make provision for flow control, possibilities could thus arise where the GPS sends data faster than the controller module can process. A FIFO buffer is thus used in an attempt to prevent this. The block diagram below shows the GPS Controller implementation.

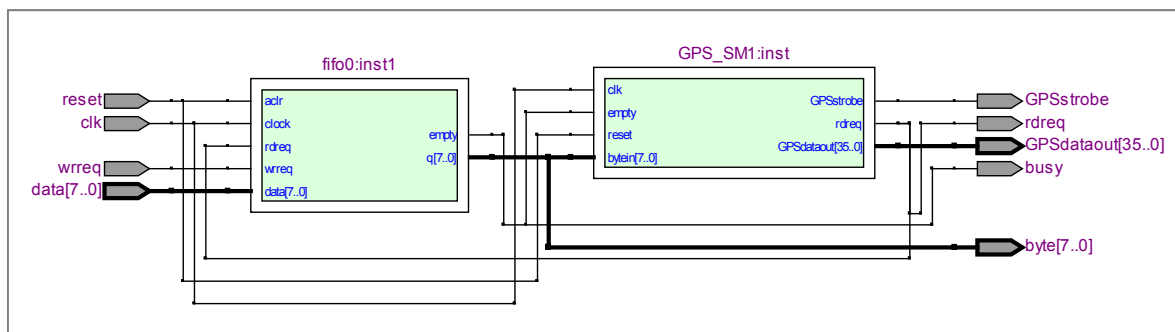


Figure 5.27 - GPS Controller

An overview of the ASMs implemented for the controller is shown in Figure 5.28 and 5.29 and the VHDL implementation was completed. The ASM1 is responsible for servicing the FIFO buffer and only packing the necessary data into an array of memory (internal to the GPS Controller). It then invokes the ASM2 into operation. In this state machine, data is basically read from the memory array and placed into packets, which are sent out. These packets have an ID which identifies it.

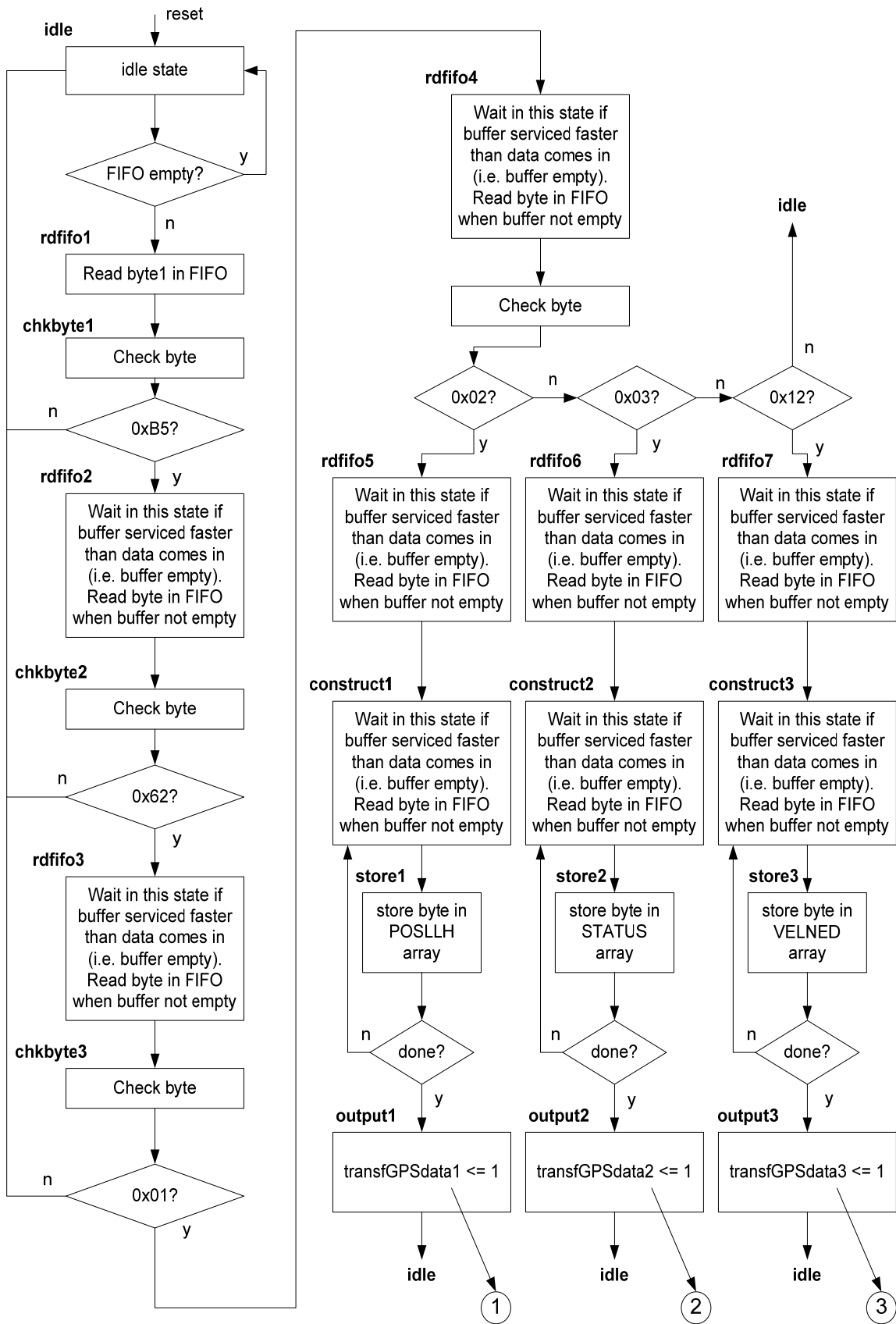


Figure 5.28 - GPS Controller Module ASM1

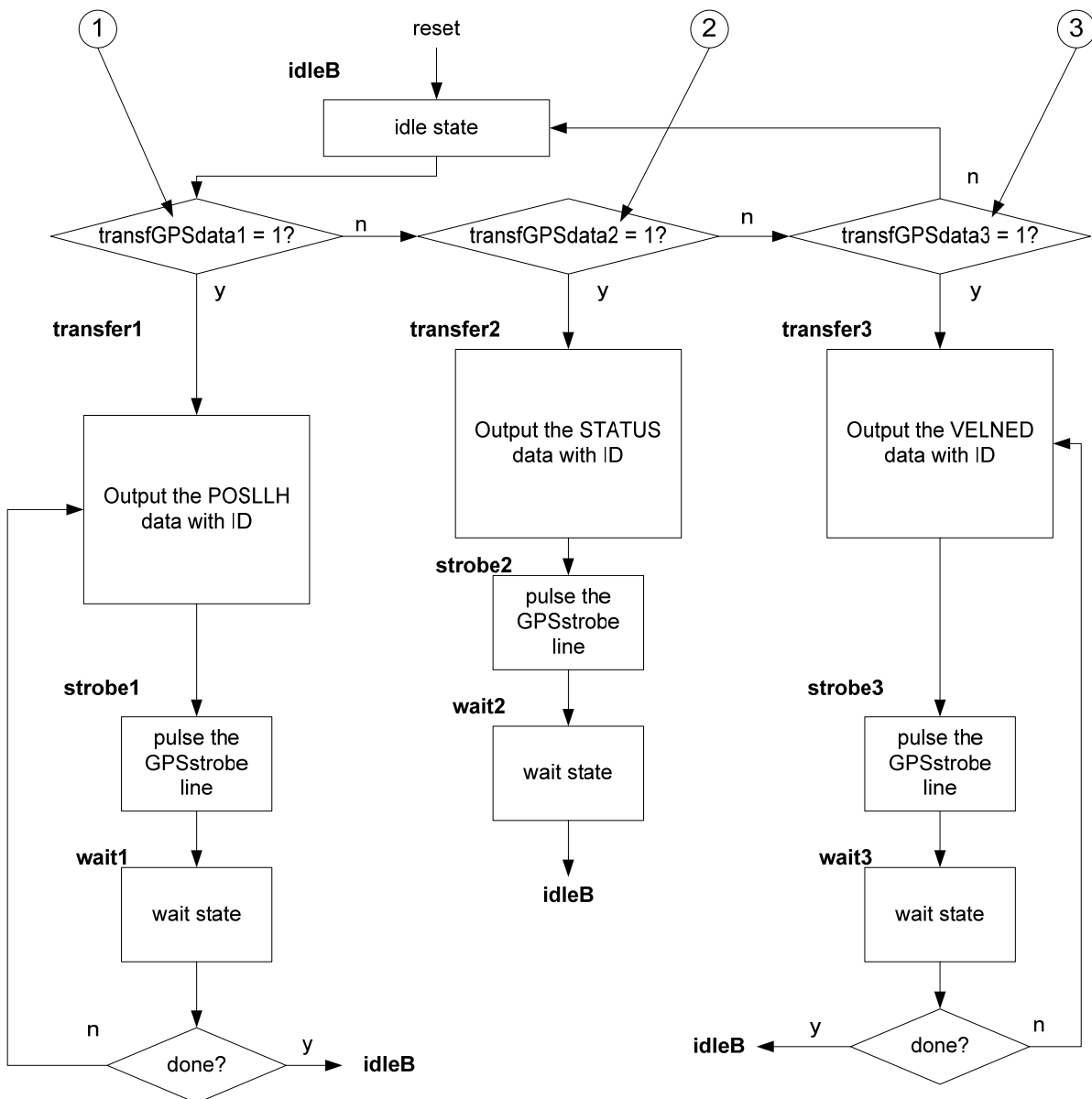


Figure 5.29 - GPS Controller Module ASM2

The GPS Controller module was simulated and the results are shown in Chapter 7. The controller should also be modified in future to enable it to configure the programmable features of the LEA-4H GPS Device.

## 5.10 CONCLUSION

In this Chapter, firmware interface modules and controllers were designed and implemented in VHDL. The FPGA firmware implementation completed in this project forms a platform for future development of the system. However, certain modules were not implemented due to time constraints. One of these is the main state machine which controls the overall functioning of the system. This has been partly implemented and is available in Appendix F for review. Its main function is to implement high level timing control and facilitates data flow in the system.

## 6. SOFTWARE DESIGN

The software design pertains to the source code, or C-code, required for the operation of the SH7201 microprocessor. In this chapter, the conceptual design of microprocessor functioning, in terms of data flow, is presented. The abstraction layer concept developed in Chapter 3 is a fundamental aspect of this design phase. It aims to create a shell in which the user can place control code and is thereby not concerned with the low-level hardware functionality. The microprocessor is abstracted from the FPGA-side and essentially only observes it as addressable memory. The main function of the microprocessor is to execute the control algorithms. The figure below illustrates an overview of the software conceptual design.

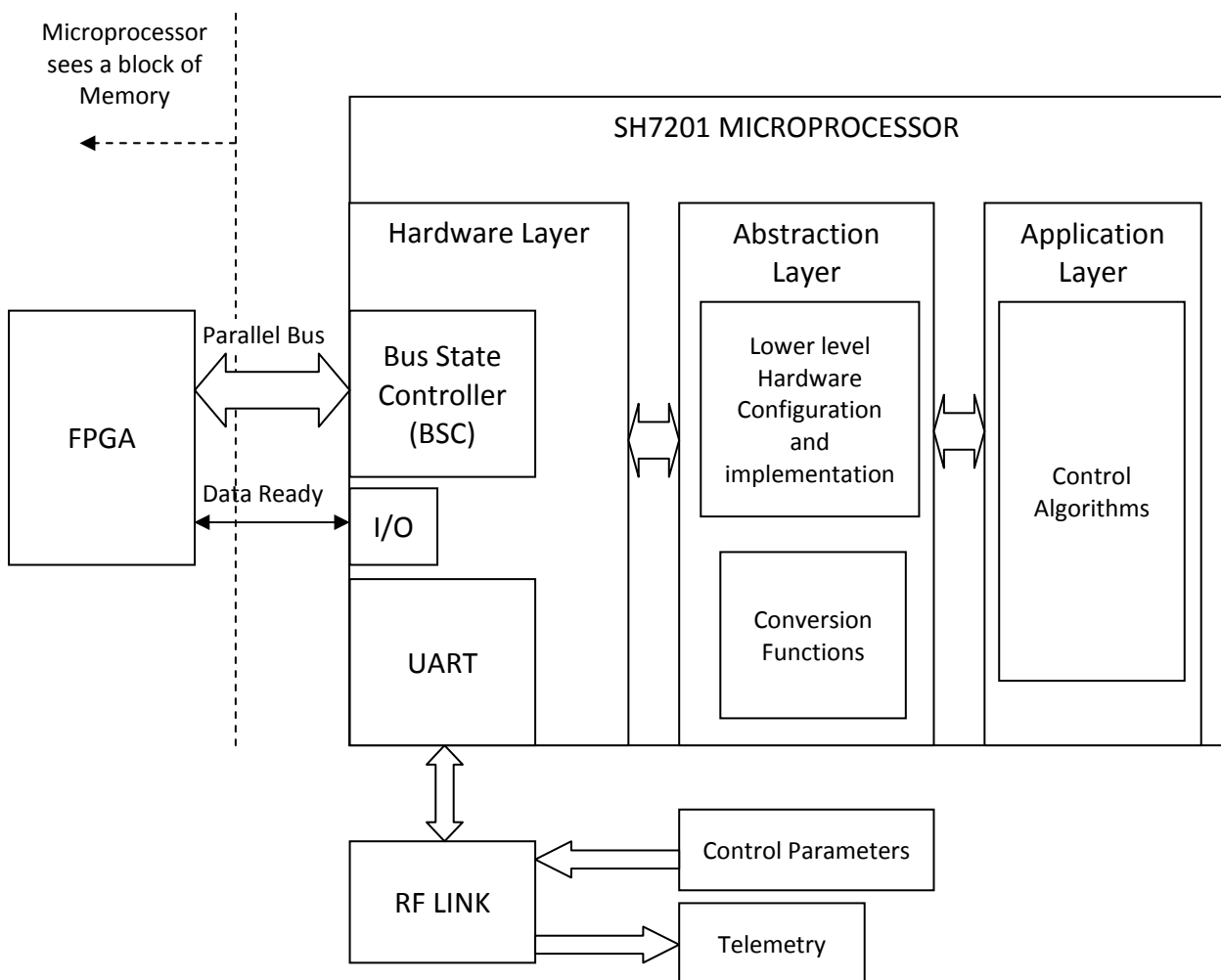


Figure 6.1 - Software Conceptual Design

An important design driver behind the timing and data flow functionality of the system is the backward-compatibility requirement with the current CAN avionics system, as well as the timing

thereof, which can be seen in [9]. A high level flow chart of the software design in this system is shown in Figure 6.2. The following steps support this figure and explain the concepts of the software functionality on the microprocessor.

1. The microprocessor waits for a signal from the FPGA indicating that the data collected from the sensors is ready. It then essentially takes control of system functioning.
2. The microprocessor then reads the data by addressing the memory locations on the FPGA.
3. The microprocessor also processes any control parameter updates (such as feedback loop gains for example) which have been received through the UART port via the RF link and updates the control algorithms.
4. Conversion functions are executed to convert the raw data to useful information in a user friendly format.
5. The control algorithms are then executed and actuator commands generated.
6. Servo actuator commands generated by the control system are again converted from the user friendly units to data values required on the hardware level.
7. This command data is then written to the respective memory register locations on the FPGA.
8. Data packets of all the relevant information is constructed and sent to the ground station through the RF link.
9. The microprocessor then signals the FPGA via the data ready signal to indicate that the servo data is ready.

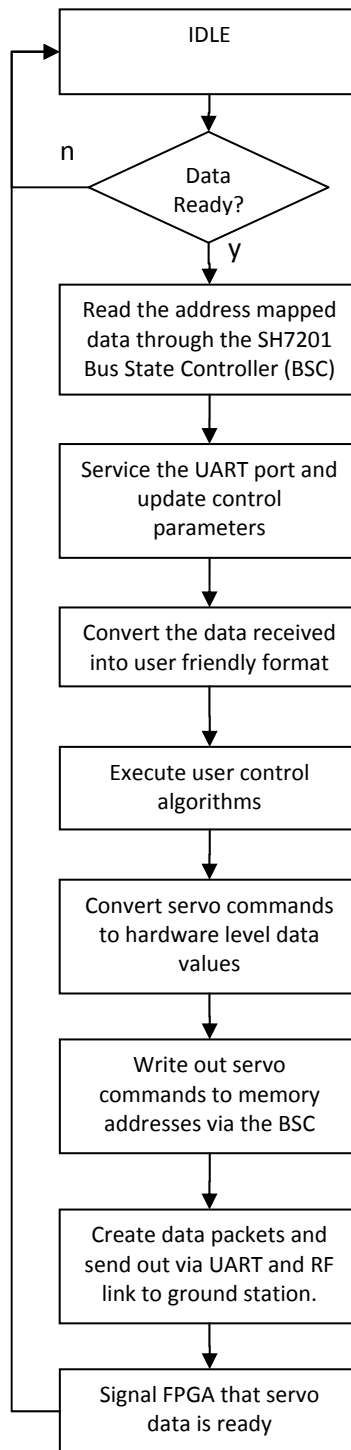


Figure 6.2 - High level flow chart of the software design concept.

The proposed conceptual design captures the key concepts and functionality to be implemented in the microprocessor software. Due to time constraints this was not possible, however, considering the conceptualization of the system and the investigation of the microprocessor functioning, implementing this architecture should not require further complex design considerations as the foundation of the functionality has been developed.

**Traceability:** The application layer (control algorithms) offers the user a simplification of the underlying execution which occurs in the hardware layer (UART, I/O and BSC). The abstraction layers (conversion functions) allows for an interface between these two layers. This traces back to main conceptual design discussed in chapter 3.

## 7. TEST AND SIMULATION RESULTS

This chapter discusses the simulations and results of the FPGA firmware modules designed and implemented in VHDL. The simulations were executed using the simulation tool (in *timing analysis mode*) of the *Quartus II* development software. The relative figures for each simulation case are shown at the end of this chapter.

### 7.1 BIC AND DPRAM MODULE

Figure 7.1 shows the connections between the BIC and DPRAM modules implemented to test this subsystem. The simulation signals from the microprocessor are shown in Figure 7.2 for the write cycle and Figure 7.3 for the read cycle. The wait-states as explained in Chapter 5 have also been inserted in the simulation. The above simulation is run at 40MHz.

The subsystem was tested by simulating a microprocessor write cycle and then a microprocessor read cycle with the same addresses. From the simulation results in Figure 7.4 it can be seen that the modules function correctly. The data written to the subsystem (displayed in HEX values) is the same data read out later, as indicated by the arrows.

The datasheet of the SH2701 shows that the data must be stable for at least 13ns after the read sample point. The Read Data Hold time ( $t_{RDH}$ ), seen in Figure 7.5, is more than two clock cycles. This is more than enough as the maximum bus frequency that the SH7201 can be run at is 60MHz, which means that 2 clock cycles equal a time of 33.33ns, which is sufficient.

### 7.2 SPI MASTER MODULE

The simulation results for the SPI Master Module are shown below. The module was tested using different SPI modes and data transfer lengths. The signal and data ports were loaded with values and the simulation outputs verified for correct functioning of the SPI Master module. Below are the different simulation parameters used for each simulation, as well as a discussion on the results:

#### 1. Simulation 1 (Figure 7.6)

- Data transfer length of 16 bits (dlength = 15)
- SPI Mode 0: cpol=0,cpha=0
- Simulation data from slave device (serial): miso = 1010'1111'0101'0001
- Simulation data to be transfer to slave: txdata[15:0] = 0111'0001'1000'1010



The simulation result verifies the correct functioning of the module for the input parameters. From the figure it can be seen that the sclk output is correct and it is low during the bus idle period, corresponding to cpol=0. The data on the mosi data line also becomes available when the ncs signal is asserted low. The data is also changed on the trailing edge of the sclk. This corresponds to the cpha=1 setting. The data outputted on the mosi data line corresponds to the data in the txdata register. The data in the rxdata register at the end of the SPI transfer frame also corresponds to the serial data on the miso data line. This shows that the data is being sampled correctly by the module. The bit reversed order of the rxdata and txdata registers can also be observed in the simulation results. This simulation result therefore corresponds to the SPI protocol description given in Chapter 5. The spi\_start triggering signal and the busy signal also function correctly.

## 2. Simulation 2 (Figure 7.7)

- Data transfer length of 16 bits (dlength = 15)
- SPI Mode 3: cpol=1,cpha=1
- Simulation data from slave device (serial): miso = 1010'1111'0101'0001
- Simulation data to be transfer to slave: txdata[15:0] = '0111'0001'1000'1010

All the simulation inputs except the cpol and cpha values are kept the same as in Simulation 1 in order to test the functionality of the different SPI modes. From Figure 7.7 it can be observed that the module functions correctly for the second fundamental mode of SPI (cpha = 1). Data changes on the mosi data line change on the leading edge and this is correct. Once again the data in the rxdata register and on the mosi data line correspond with the values of the miso and txdata input parameter respectively. The sclk idle value of one also corresponds to the cpol value of one. The results of Simulation 1 and Simulation 2 verify that the SPI Master module functions correctly for different modes of SPI.

## 3. Simulation 3(Figure 7.8)

- Data length of 32 bits (dlength = 31)
- SPI Mode 2: cpol=1,cpha=0
- Simulation data from slave device (serial): miso =  
1010'0110'0100'1101'1010'0110'0100'1101
- Simulation data to be transfer to slave: txdata[32:0] =  
1100'1010'0000'1111'1100'1010'0000'1111

This simulation verifies the correct functioning of the module at different data transfer frame lengths and once again the correct functioning of the SPI mode settings is verified. Further simulations were run and the functionality of the SPI Master module was correct in all these cases.

4. Simulation 4 (Figure 7.9):

This simulation was executed to verify that the signal timing requirements of the SPI Master module adhere to the specified values in the respective datasheet. The relevant timing data was extracted from the simulation results and are tabulated below. Each SPI Master Module operates at a different clock speed and thus the values of the parameters are given in terms of the number of clock cycles.

**Table 7.1 - Timing Parameters of Simulation Result**

Parameter	Description	Value
t <sub>1</sub>	nCS falling edge to first SCLK edge (when cpha=0)	2 clock cycles
t <sub>2</sub>	Data valid before sample edge of SCLK (data setup time)	2 clock cycles
t <sub>3</sub>	Data hold after SCLK sample edge	2 clock cycles
t <sub>4</sub>	nCS rising edge after last SCLK rising edge(when cpha=1)	2 clock cycles

The specific timing requirements for the SPI devices can be found were found in the datasheets of each component. A summary of the requirements and simulation values are shown below. The simulated values calculated show that the timing requirements of all the devices, except the MAX3420E, are met at their maximum clock speed. The MAX3420E specifications t<sub>1</sub> and t<sub>2</sub> are not met at the highest SPI bus speed of 26MHz, as seen in Table 7.2. In order to meet the 30ns specification in both cases, a clk period of 15ns (i.e. 30ns divided by 2) is required. This translates to a maximum SPI SCKL-speed of 8.33 MHz at which the device can communicate.

**Table 7.2 - SPI timing requirements vs simulated timing results**

Device	clk cycle	t <sub>1</sub>		t <sub>2</sub>		t <sub>3</sub>		t <sub>4</sub>	
		req. (min)	sim.	req. (min)	sim.	req. (min)	sim.	req. (min)	sim
ADS 8344	104.2ns	100ns	208ns	100ns	208ns	10ns	208ns	NA	NA
ADIS 16350	125ns	48ns	250ns	24.4ns	250ns	48.8ns	250ns	5 ns	250ns
MAX 3420E	9.62ns	30ns	19.24ns	5ns	19.24ns	10ns	19.24ns	30ns	19.24ns
MCP 2515	25ns	50ns	50ns	10ns	50ns	10ns	50ns	50ns	50ns

[KEY - req. (min) = minimum requirement (from data sheet) ; sim. = simulation result]

### 7.3 PWM MODULE

The PWM module was simulated at different clock frequencies. Figure 7.10 and 7.11 show the simulation for a frequency of 3.4 MHz. This gives a resolution of 294ns per pwmcount value. The values of 2040 and 8191 were set in the pwmcount register and the pulse output verified. This is shown in Figure 7.10 and Figure 7.11 respectively. It can be seen that the correct pulse lengths of 600us and 2.4ms are outputted on the pwm\_out line. Figure 7.12 also shows the pwm\_inpulse trigger signal.

### 7.4 PWM CAPTURE MODULE

The simulation results for this module are shown in Figure 7.13 and Figure 7.14. The clock frequency was set to 20MHz yielding a period of 50ns. The module was driven with a 600us and 2.4ms pulse signal on the line\_in port. From the simulation results, it can be seen that the data\_out values are “12000” and “48000” for the respective signals. This verifies the correct operation of the PWM Capture module. Figure 7.14 focuses in on the end of the capture period to show the strobe signal more clearly.

### 7.5 UART MODULE

The sub-modules which create the UART implementation were simulated in order to verify their correct functioning. The results of this simulation can be seen in Figures 7.15 to 7.17.

The baud rate simulation was run at a BRG module clock frequency of 20MHz and a 9600 baud rate implementation. The result can be seen in Figure 7.15. The difference between the slow\_clock pulses is 104us and this verifies the 9600 baud rate generation. The 16 slow\_clock16 pulses can also be seen between each slow\_clock pulse.

The TX sub-module, together with the BRG, was simulated to verify the functioning. A block diagram of the connections between the sub-modules is shown in Figure 7.18. The result of the simulation can be seen in Figure 7.16. It shows that the data value entered in the *data* register is correctly transmitted on the *serial\_out* line. The TX and BRG sub-modules thus function correctly.

In order to test the RX module, all the sub-modules were implemented and the serial data from the TX module was looped back into the UART Module. This configuration is shown in Figure 7.19. From the simulation result in Figure 7.17, it is seen that the data in the *data\_in* and *data\_out* register correspond. This shows that the RX module functions correctly.

## 7.6 IMU CONTROLLER MODULE

The IMU Controller Module was simulated together with the SPI Master Module, as shown in Figure 7.20. The simulation results are shown in Figure 7.21. From this result it can be seen that the IMU Controller Module functions correctly.

Arbitrarily simulation data entered on the *miso* line of the SPI Master Module is correctly outputted from the module with the correct address, as highlighted in Figure 7.21. Furthermore, the correct address is also sent to the IMU device as shown.

## 7.7 ADC CONTROLLER MODULE

Figure 7.22 shows the combined implementation of the ADC Controller and SPI Master Module. The result of the simulation of this implementation can be seen in Figure 7.23. The *inpulse* line was triggered to begin the operation cycle. Figure 7.24 focuses on frame 1 to show that the data is sent over the SPI bus correctly. The other frames were also verified. Arbitrary data entered on the *miso* line is also correctly outputted by the ADC Controller module and the strobe line pulsed. It is addressed with the value shown on *data18to15*.

## 7.8 GPS CONTROLLER MODULE

The simulation result for the GPS Controller is shown below. Due to the simulation length, the result is spread across Figures 7.25 to 7.29. Signals resembling GPS data is inputted to the FIFO buffer. This simulates the function performed by the UART module. The *data[7]* and *wrreq* signals were set up as if it were connected to the UART module. This is observed by the pulses on the *wrreq* line and the data values on the *data* lines.

As soon as the GPS controller observes data in the FIFO buffer, it begins reading the data. This can be seen by the *rdreq* pulses (Figure 7.25). The data is stored in memory arrays and written out once all the data has been received. The correct functionality is observed upon inspection of the output data with the correct identifier (Figure 7.29). An example of this is seen by looking at the “05 – 06 – 07 – 08” bytes of data on the input line. This is data which contains desired GPS information in the packet. This is observed on the GPSdataout line with a respective identifier (05) attached at the front i.e. “5-05-06-07-08” (Figure 7.29).

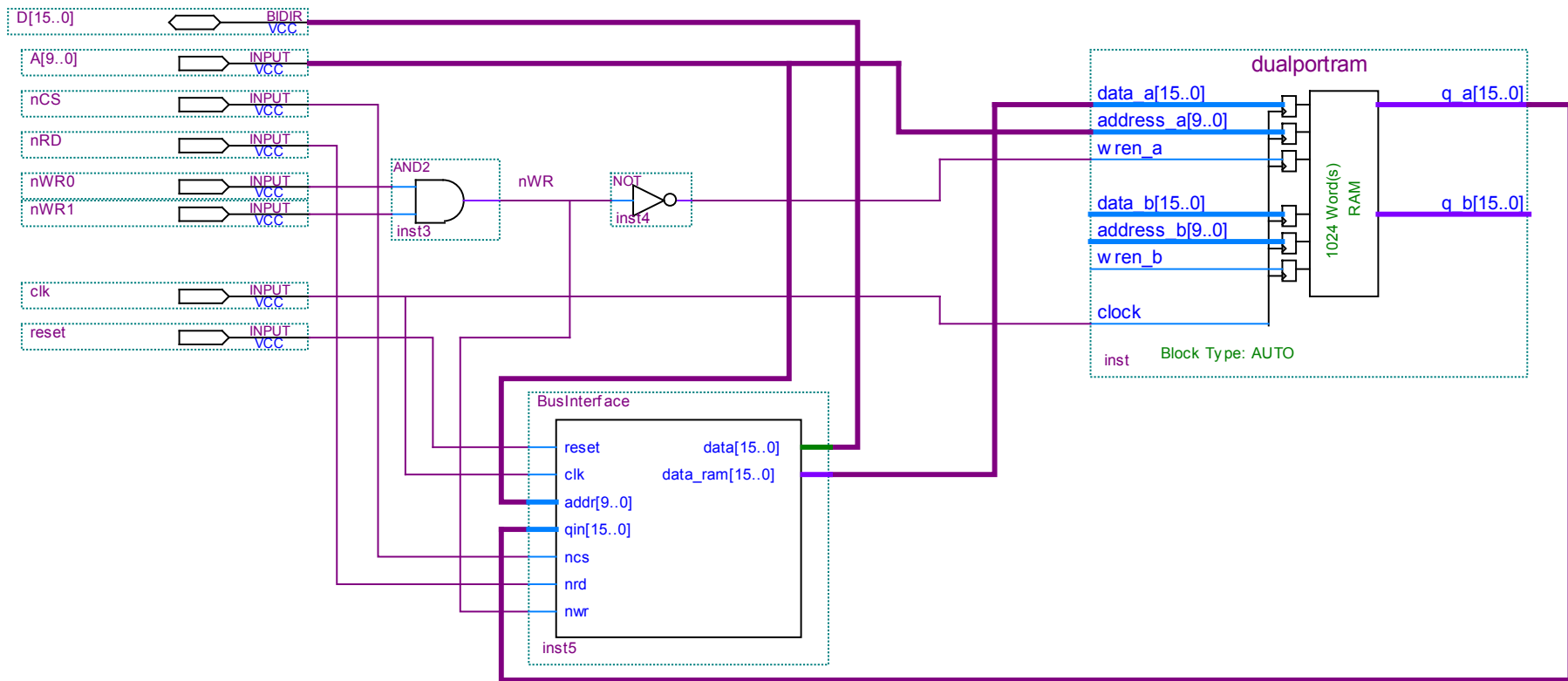


Figure 7.1 - Interaction between the BIC and DPRAM modules

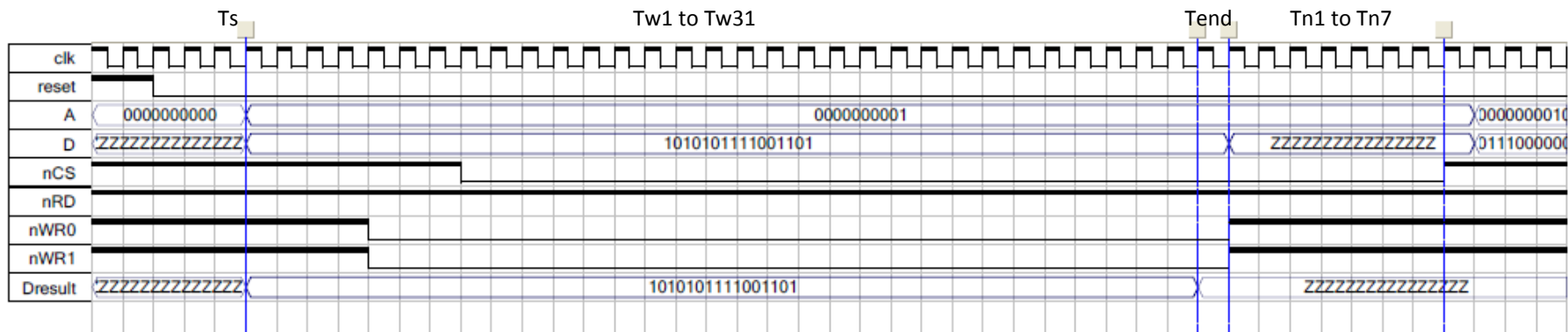


Figure 7.2 - Write cycle outputs of the microprocessor to the Bus Interface subsystem with wait states inserted

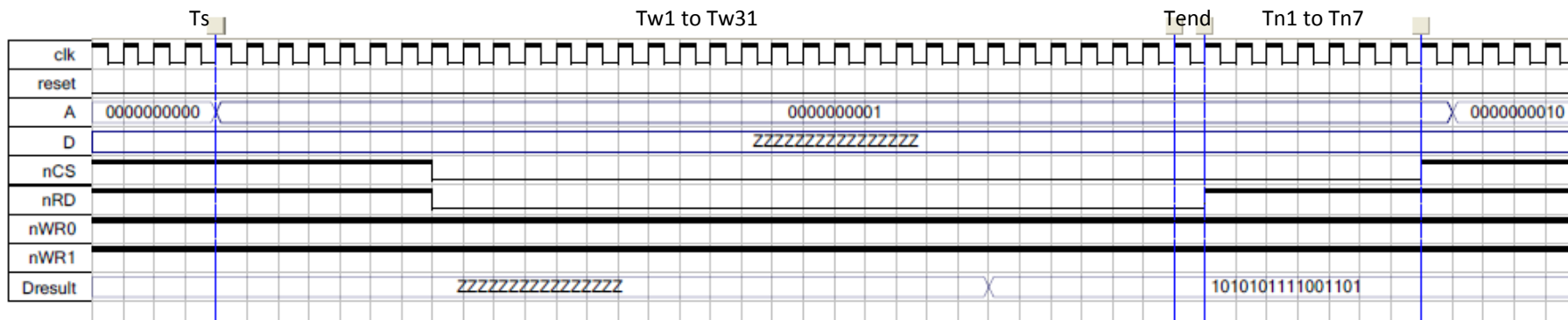


Figure 7.3 - Read cycle outputs of the microprocessor to the Bus Interface subsystem with wait states inserted

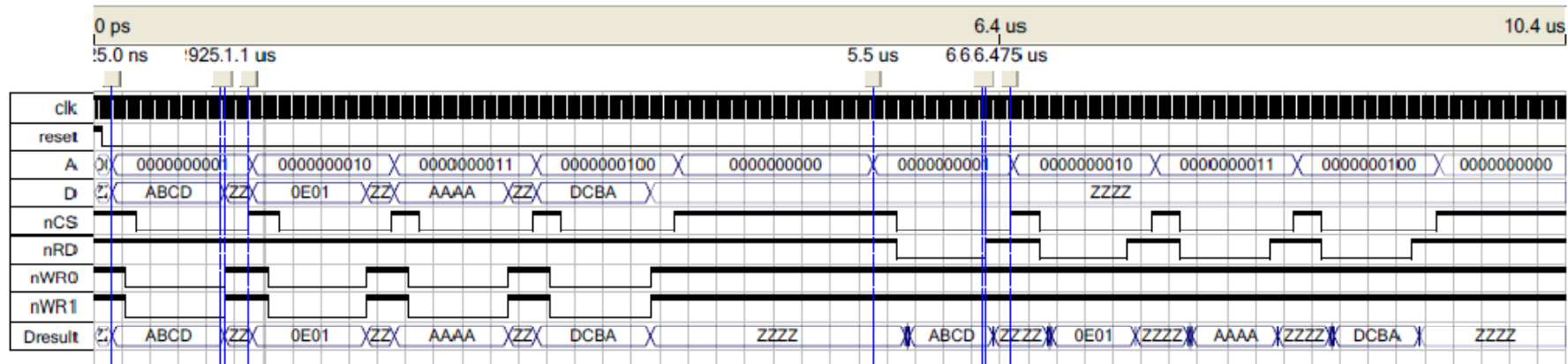


Figure 7.4 - Simulation result showing the functionality of the BIC to DPRAM subsystem

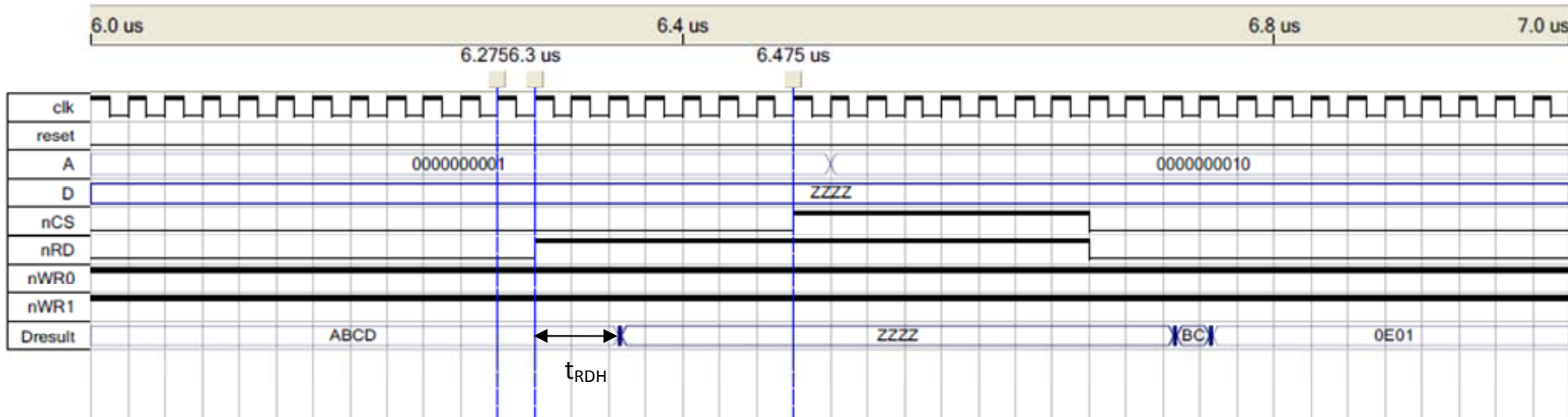


Figure 7.5 - Simulation Result showing Read Data Hold Time ( $t_{RDH}$ )



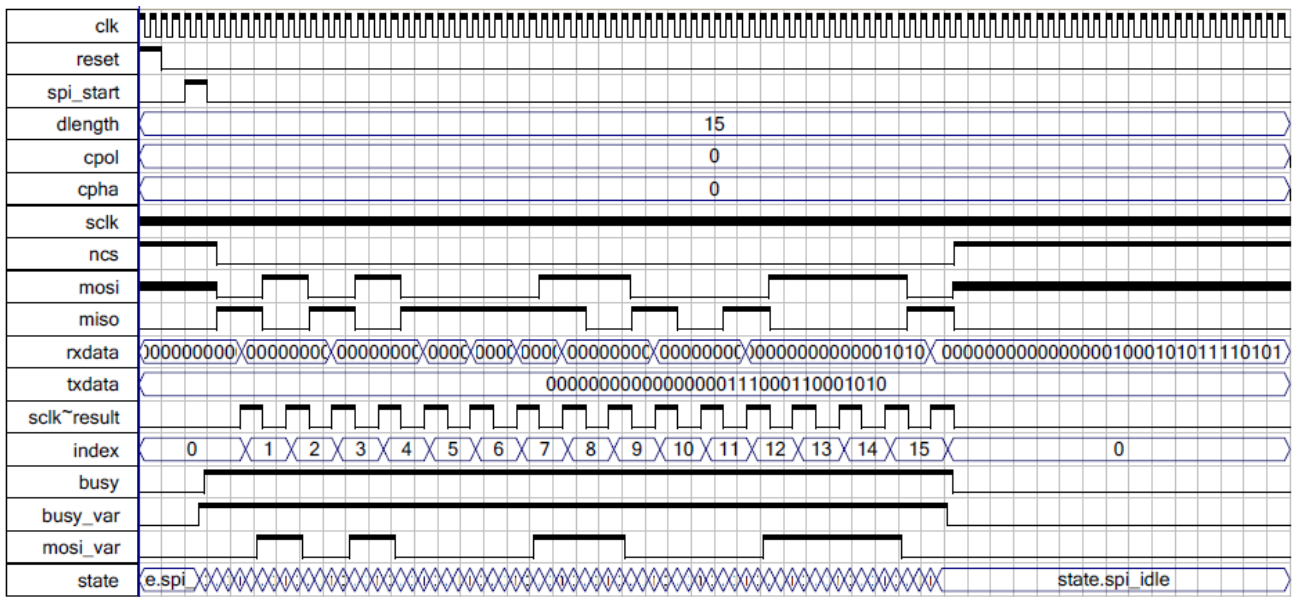


Figure 7.6 - SPI Master Module Simulation 1

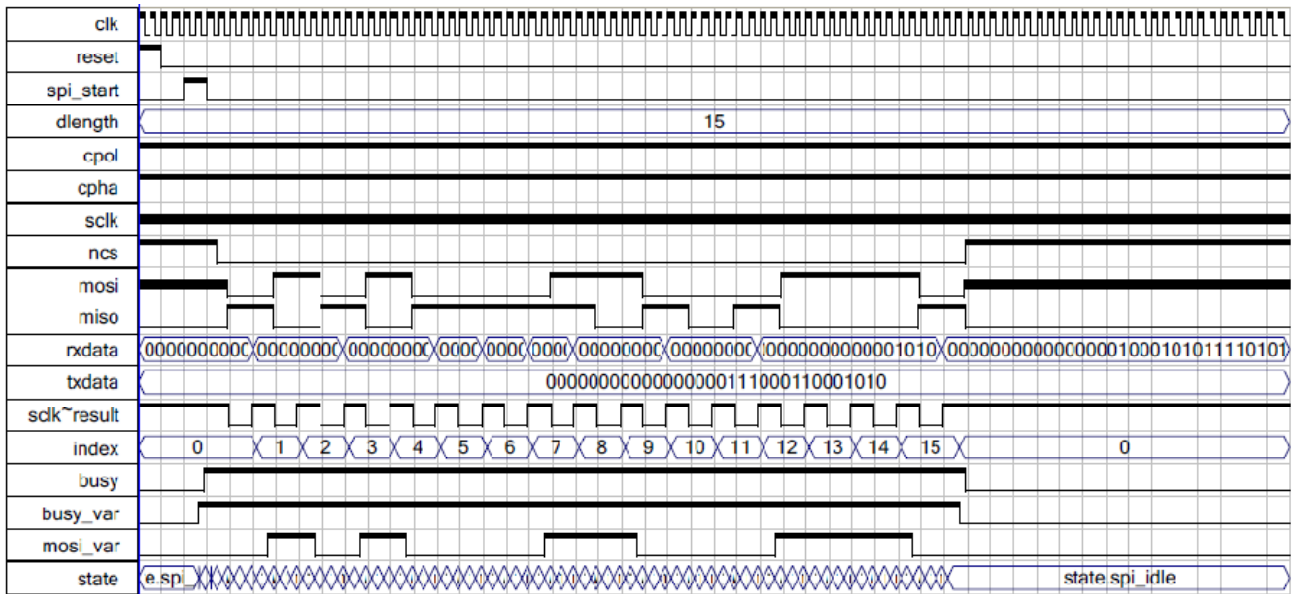


Figure 7.7 - SPI Master Module Simulation 2

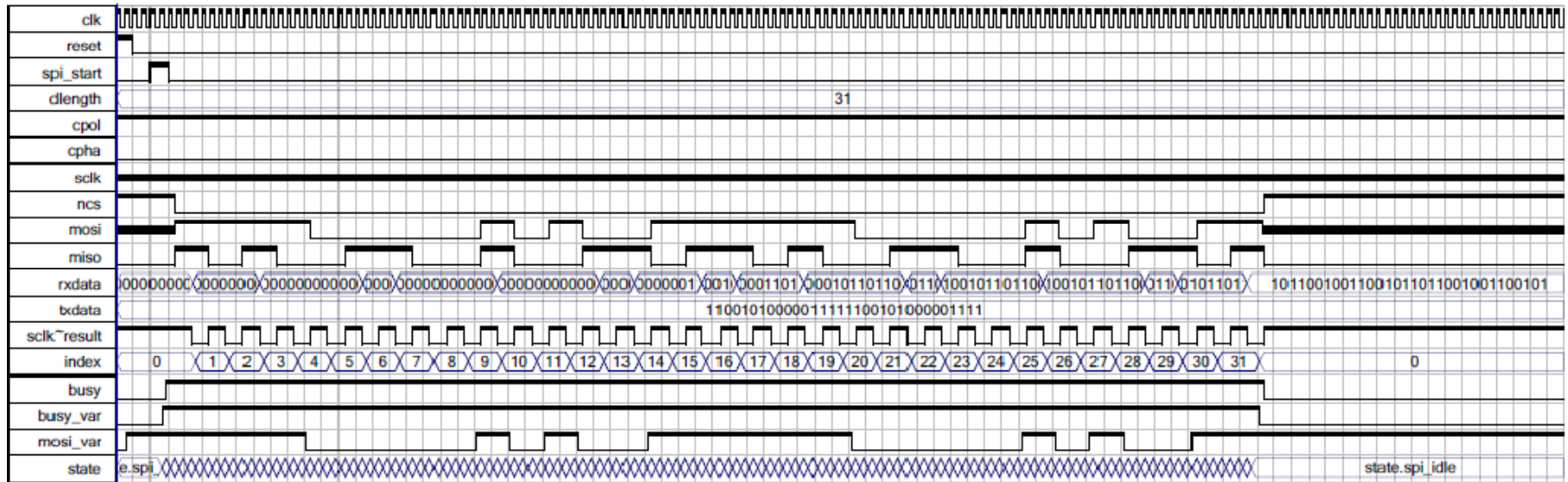


Figure 7.8 - SPI Master Module Simulation 3

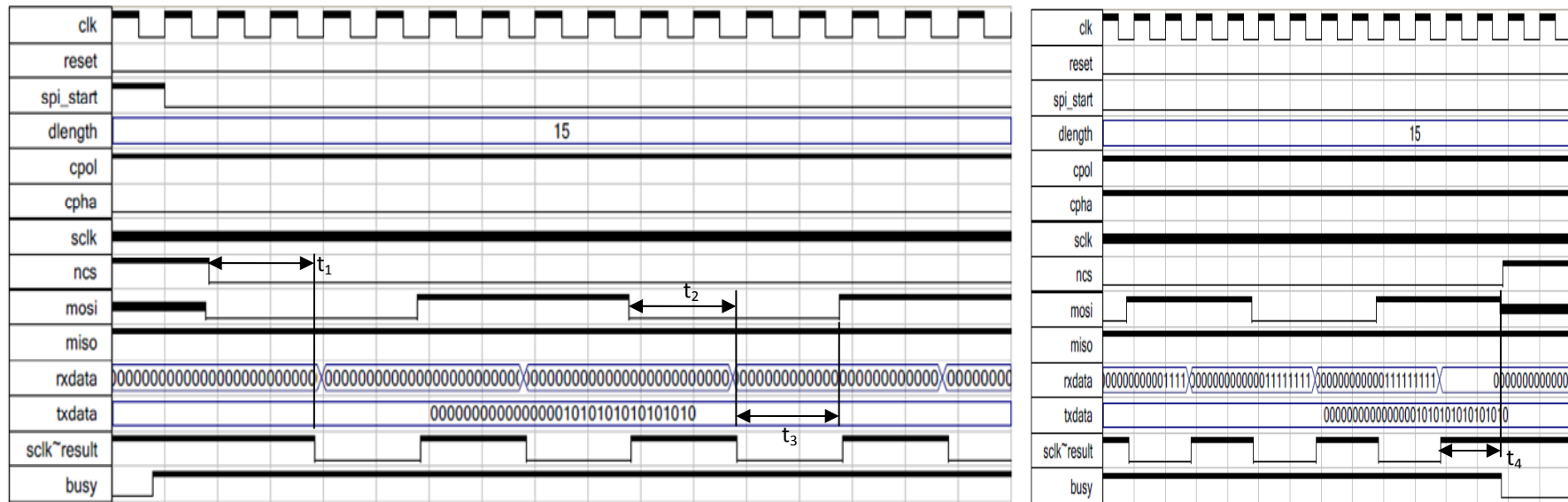


Figure 7.9 - SPI Master Module Timing Parameters

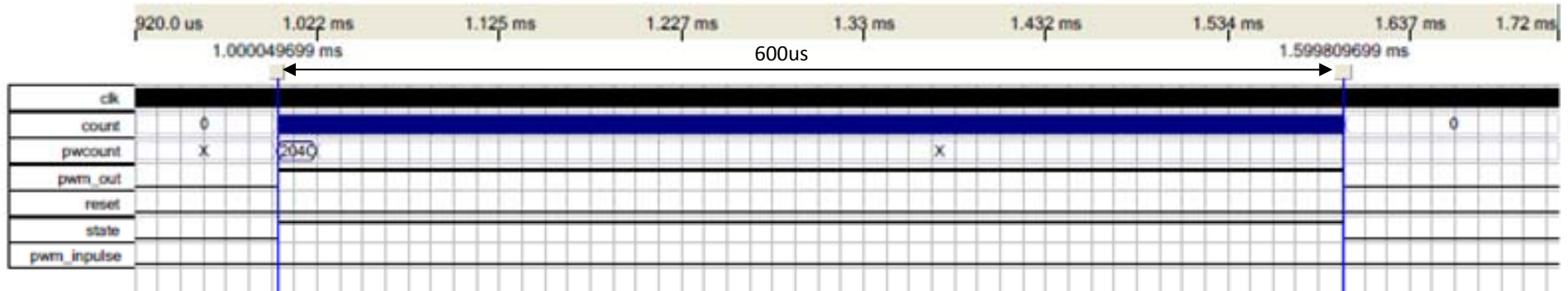


Figure 7.10 - PWM Module Simulation (3.4MHz and pwmcount = 2040)

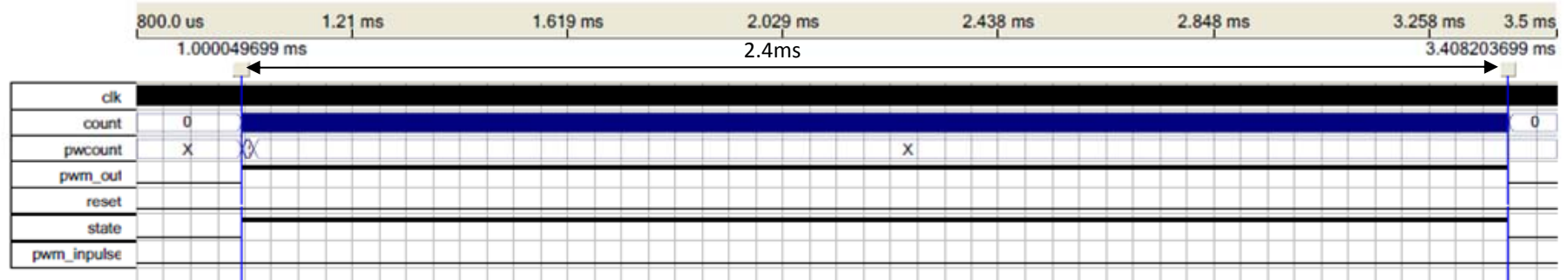


Figure 7.11 - PWM Module Simulation (3.4MHz and pwmcount = 8191)

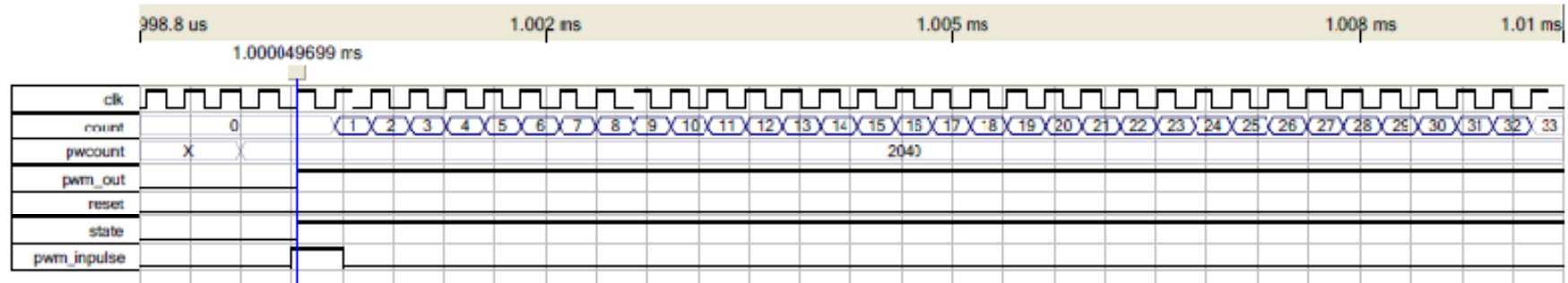


Figure 7.12 - PWM Module triggering

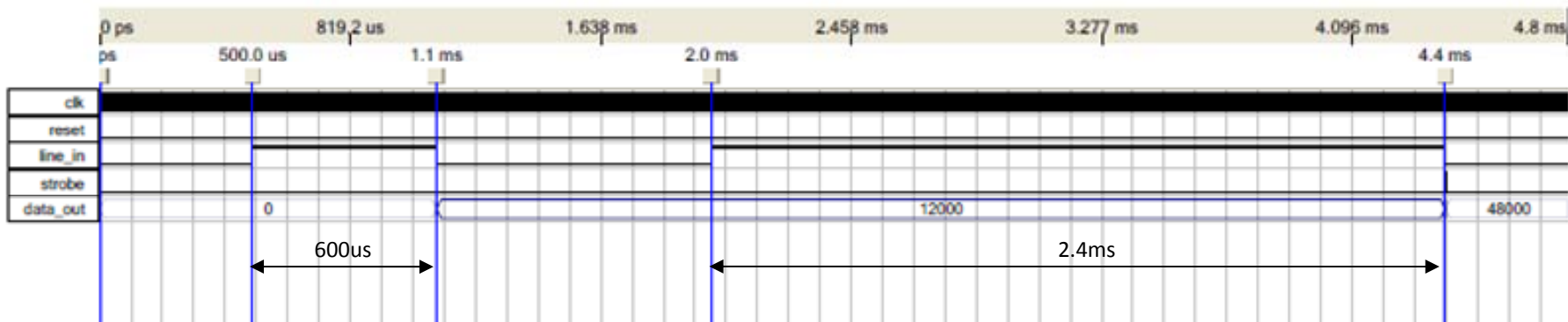


Figure 7.13 - PWM Capture Module simulation

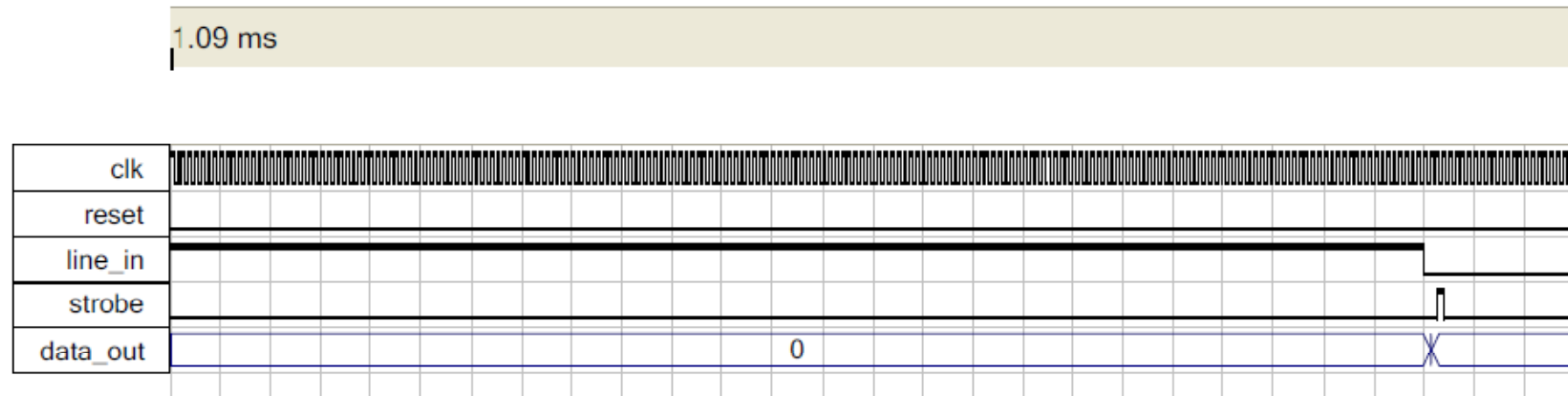


Figure 7.14 - PWM Capture Module simulation showing the strobe signal

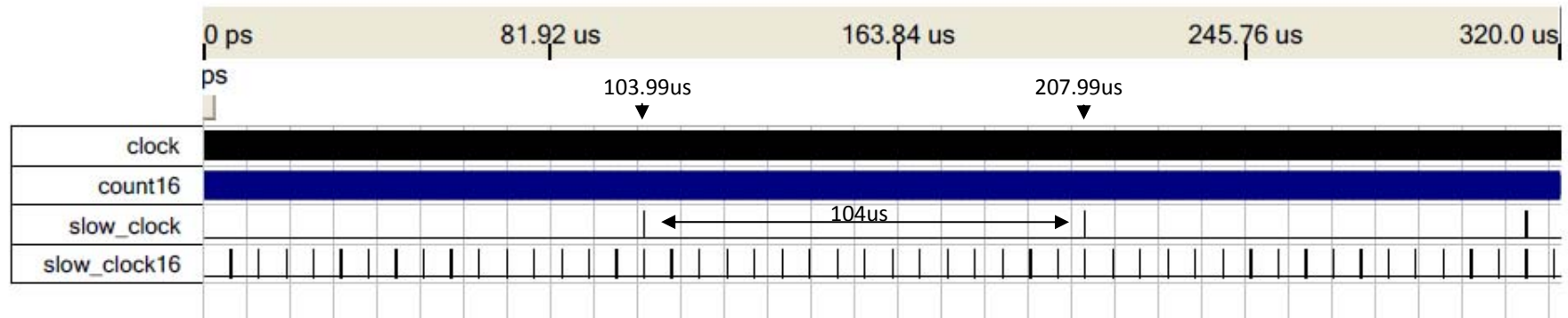


Figure 7.15 - Baud Rate (9600 baud) Simulation

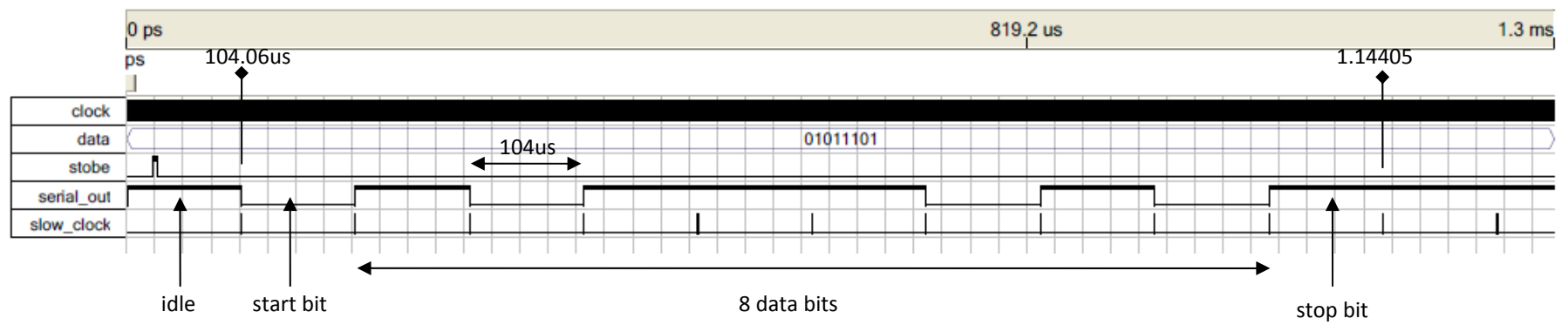


Figure 7.16 - TX and BRG (9600 baud) Simulation (see Figure 7.18)

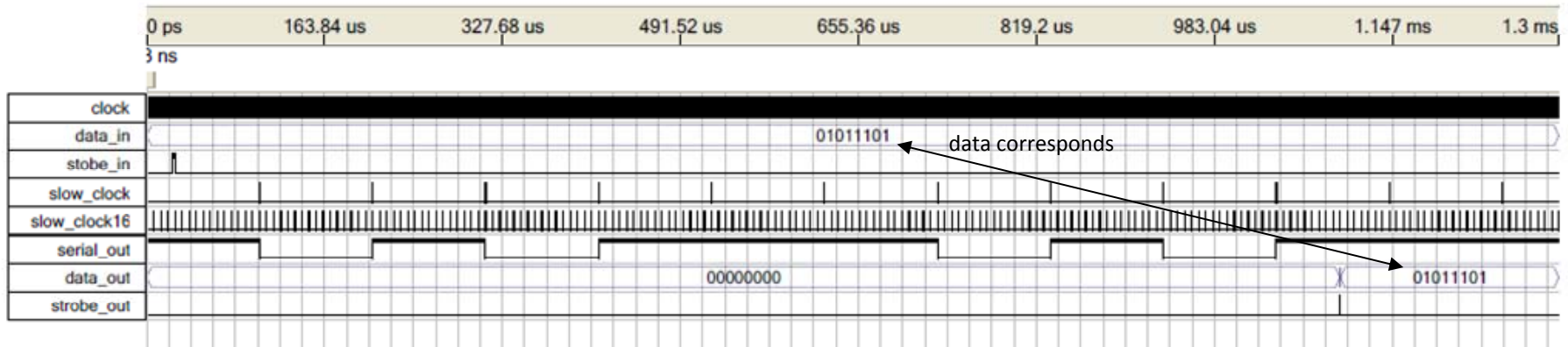


Figure 7.17 - UART Module Simulation with serial output to serial input loopback (See Figure 7.19)

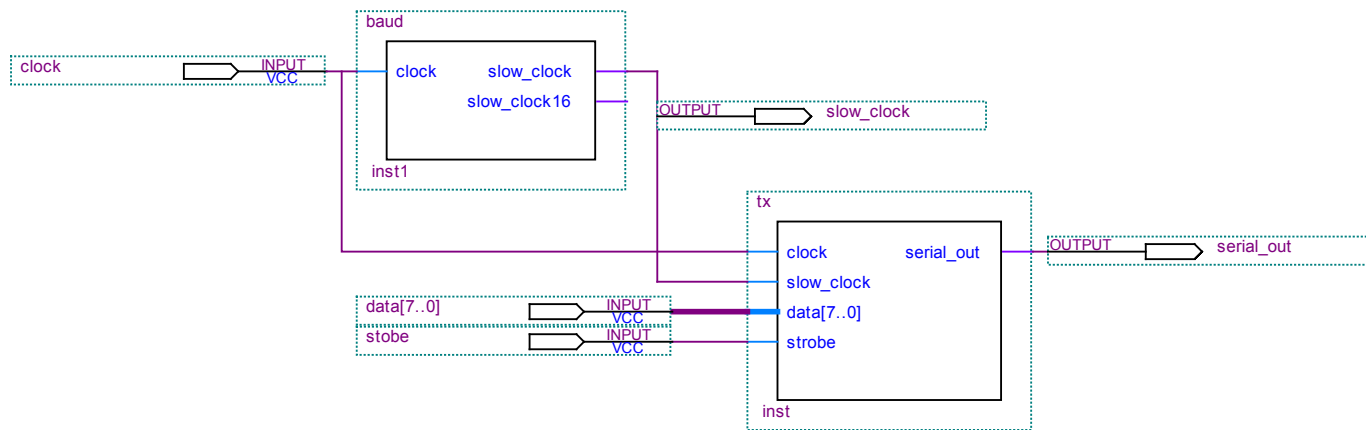


Figure 7.18 - UART BRG and TX sub-module connections

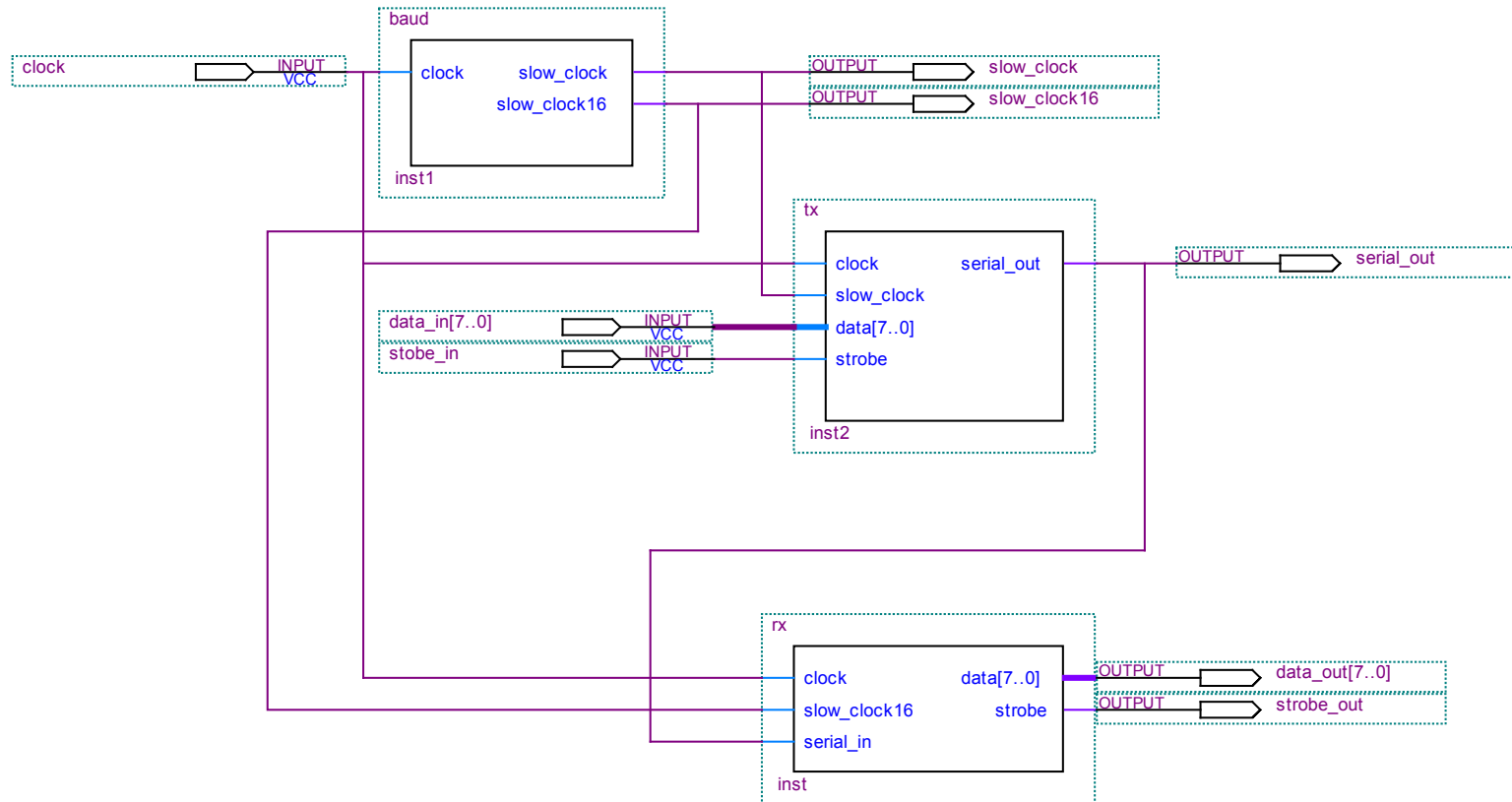


Figure 7.19 - Complete UART Module (TX to RX loopback test)

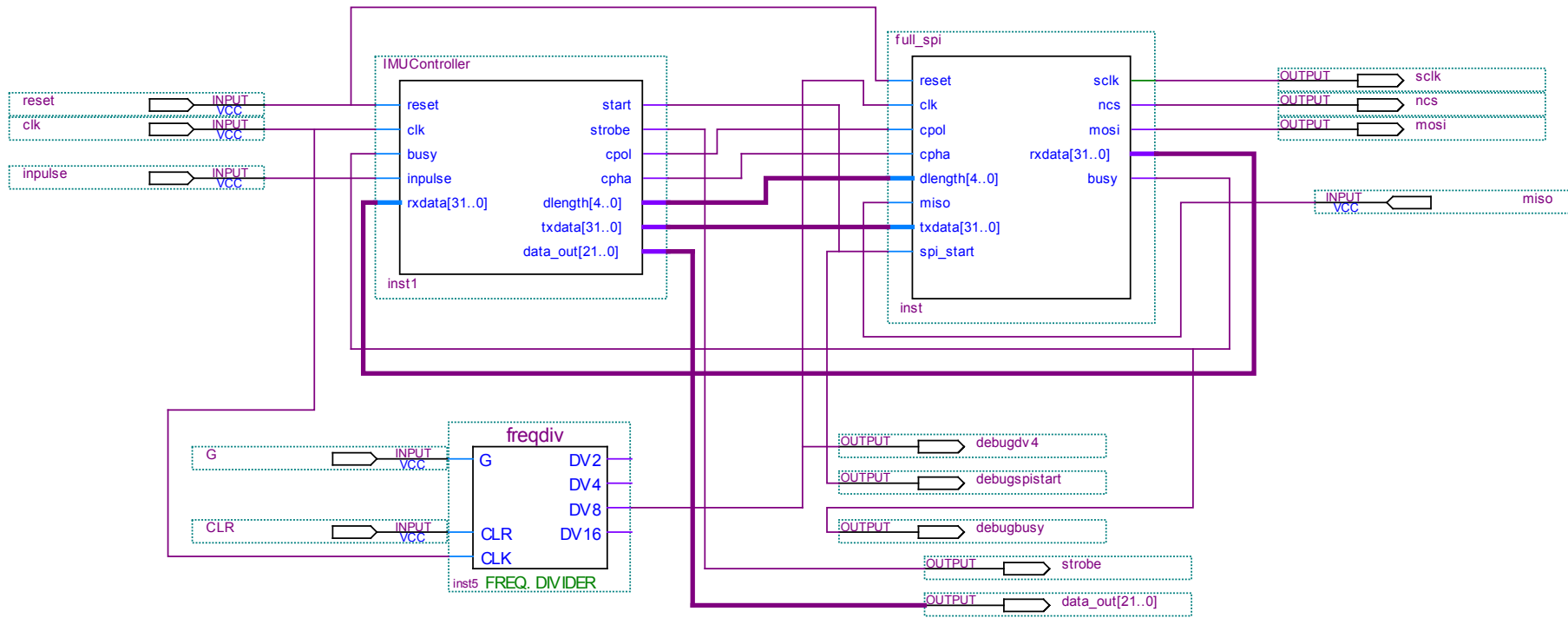


Figure 7.20 - IMU Controller and SPI Implementation



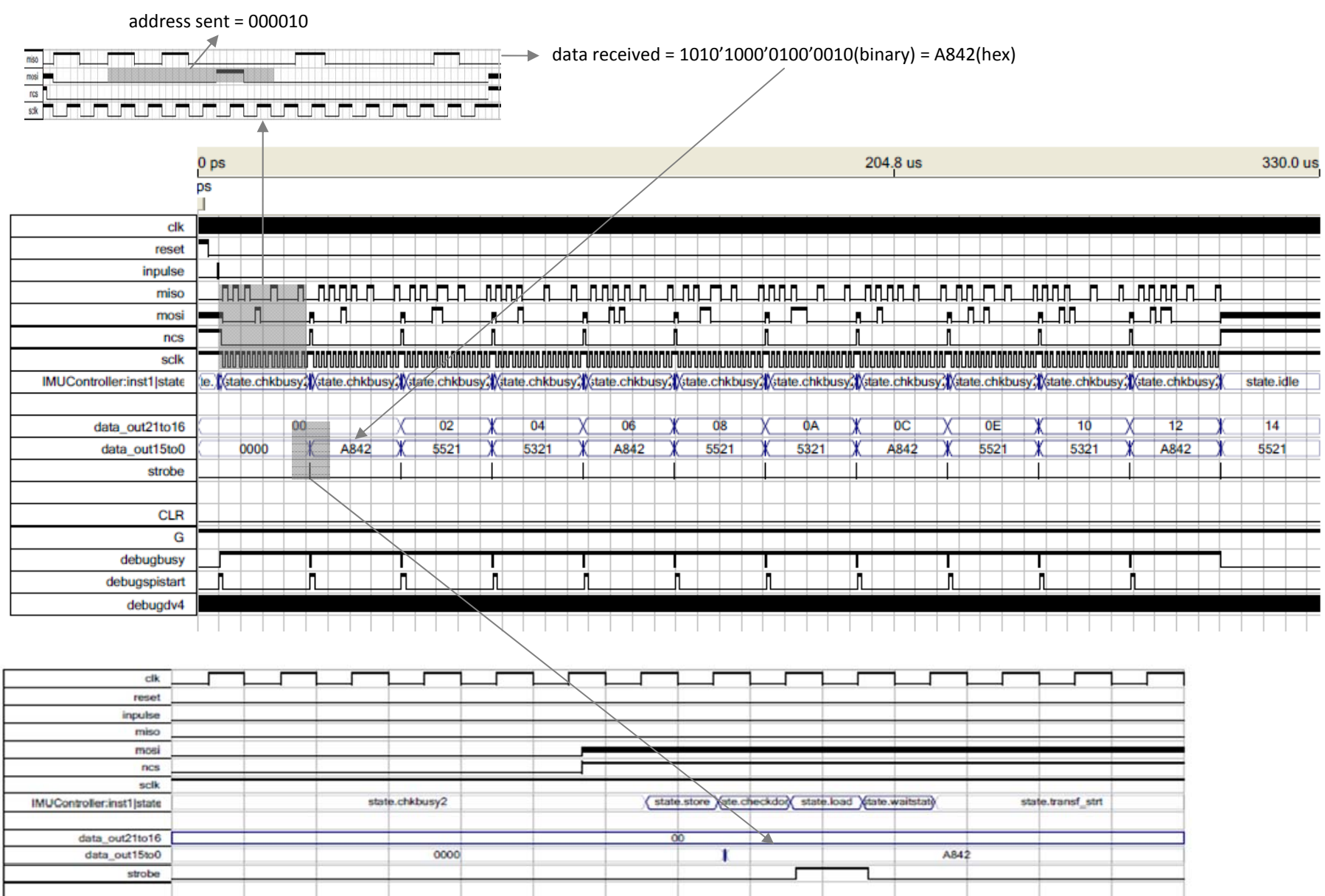


Figure 7.21 - Simulation of the interface between the IMU Controller and the SPI Master Module

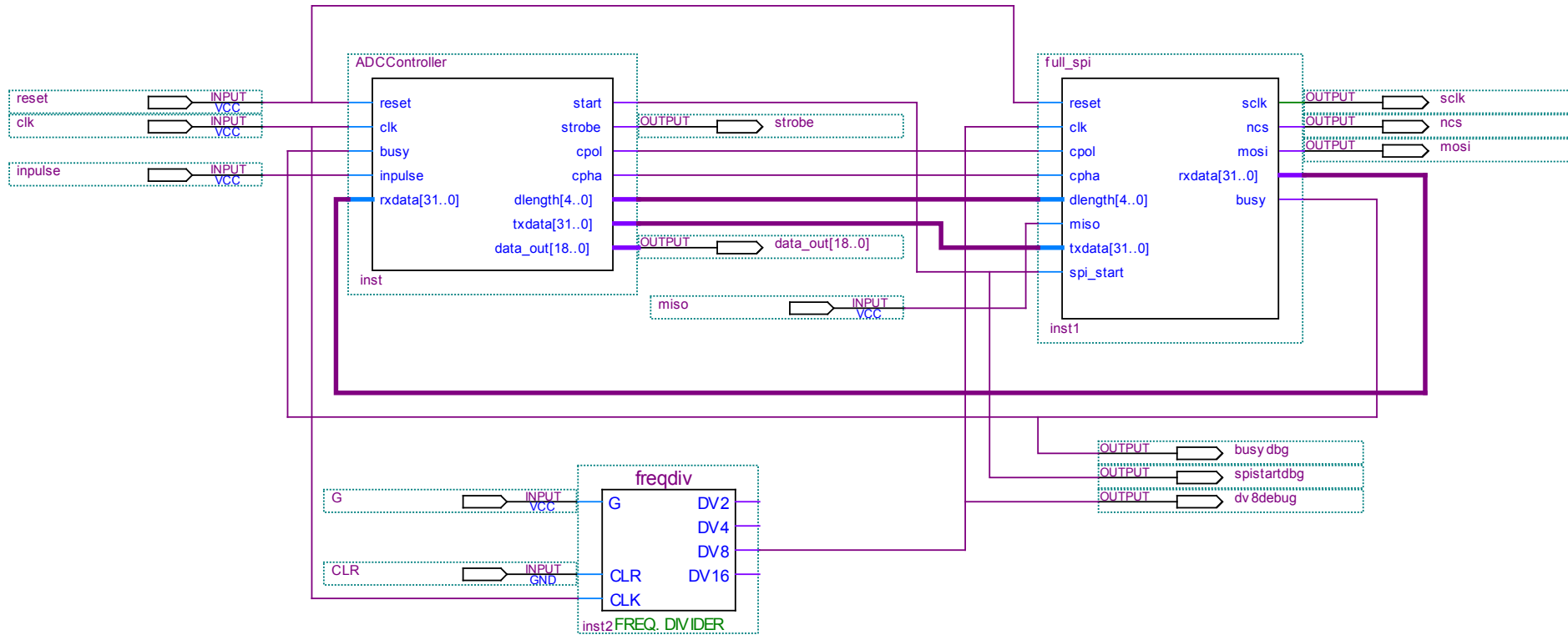


Figure 7.22 - ADC Controller and SPI Master Module implementation

see Figure 7.24

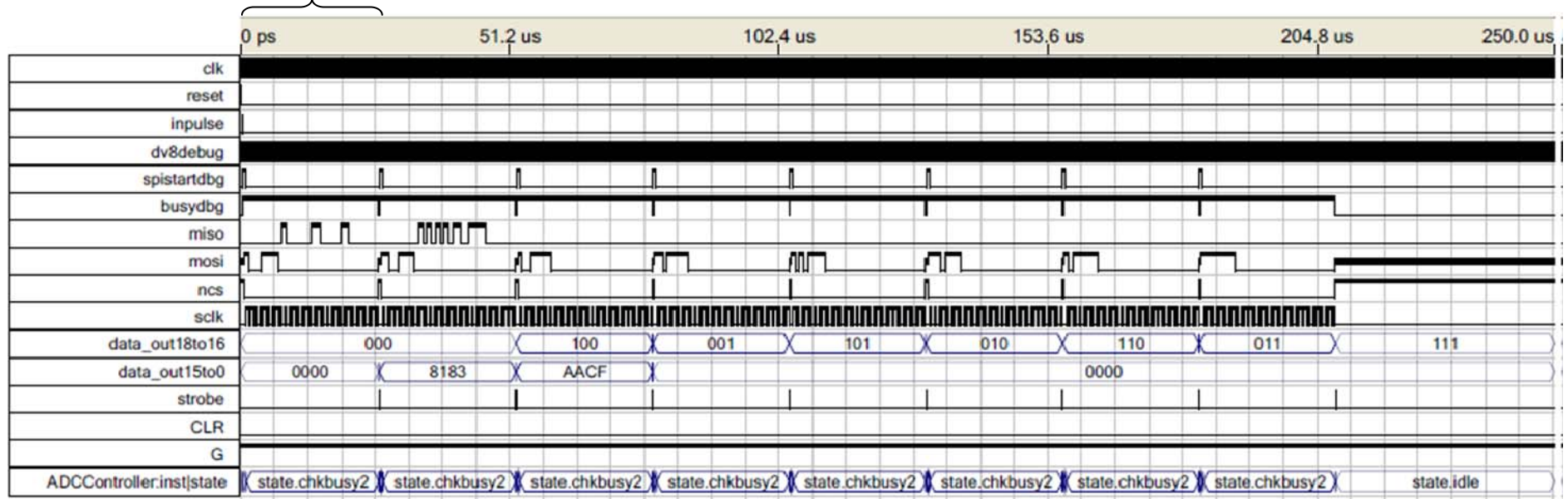


Figure 7.23 - Simulation of the interface between the ADC Controller and the SPI Master Module

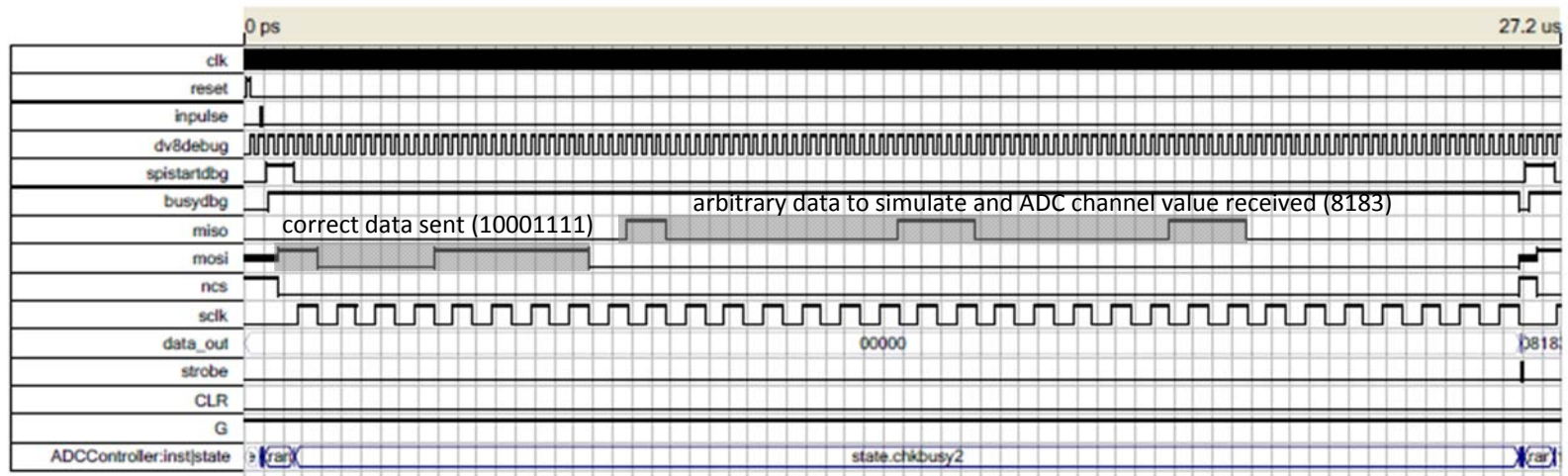


Figure 7.24 - Frame 1 of ADC Controller to SPI Simulation (see Figure 7.23)

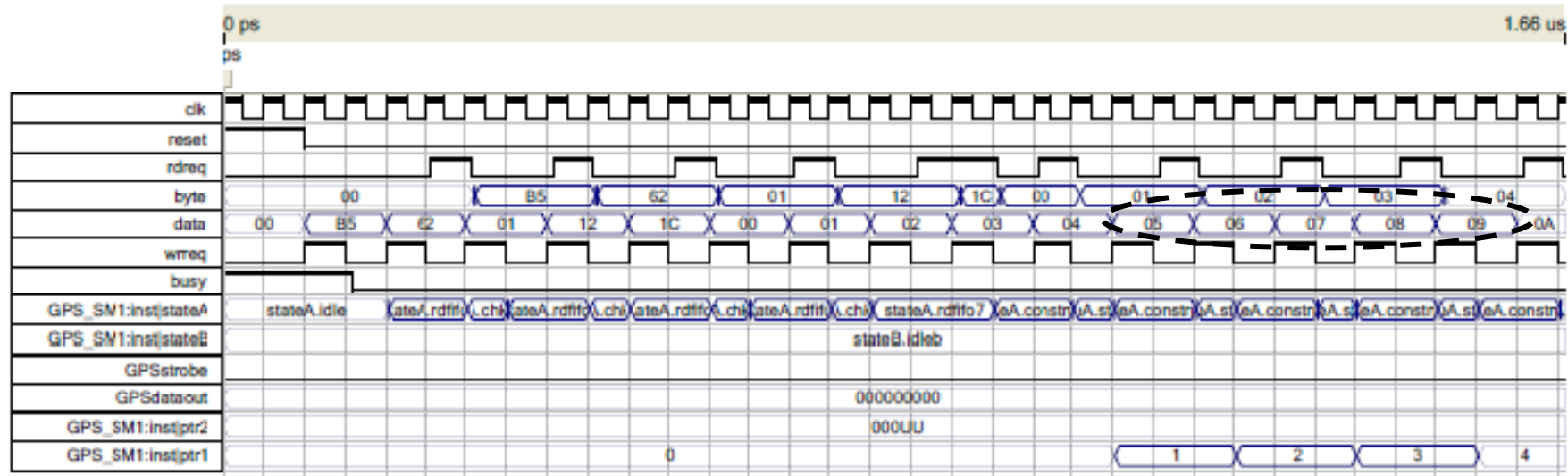


Figure 7.25 - GPS Controller Simulation (part 1)

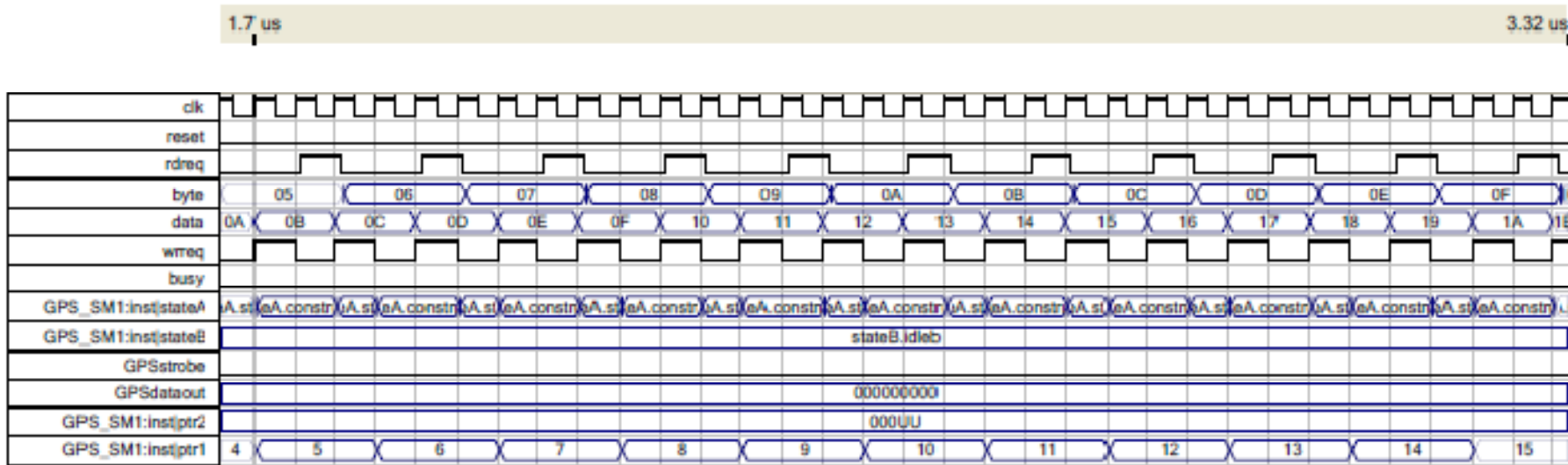


Figure 7.26 - GPS Controller Simulation (part 2)

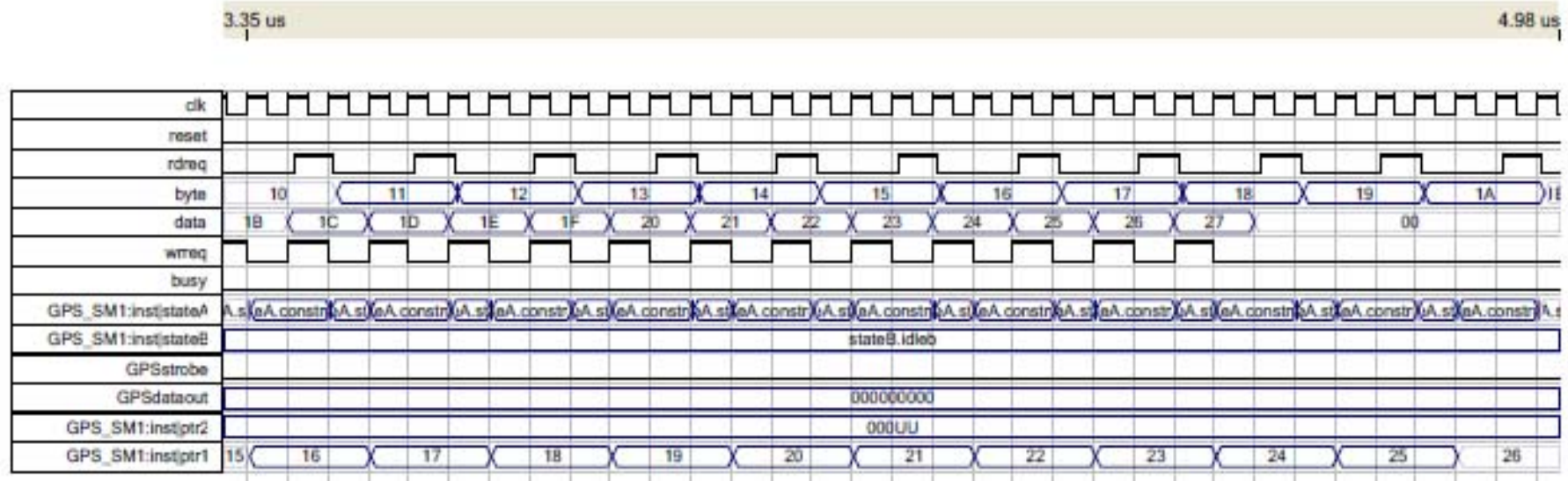


Figure 7.27 - GPS Controller Simulation (part 3)

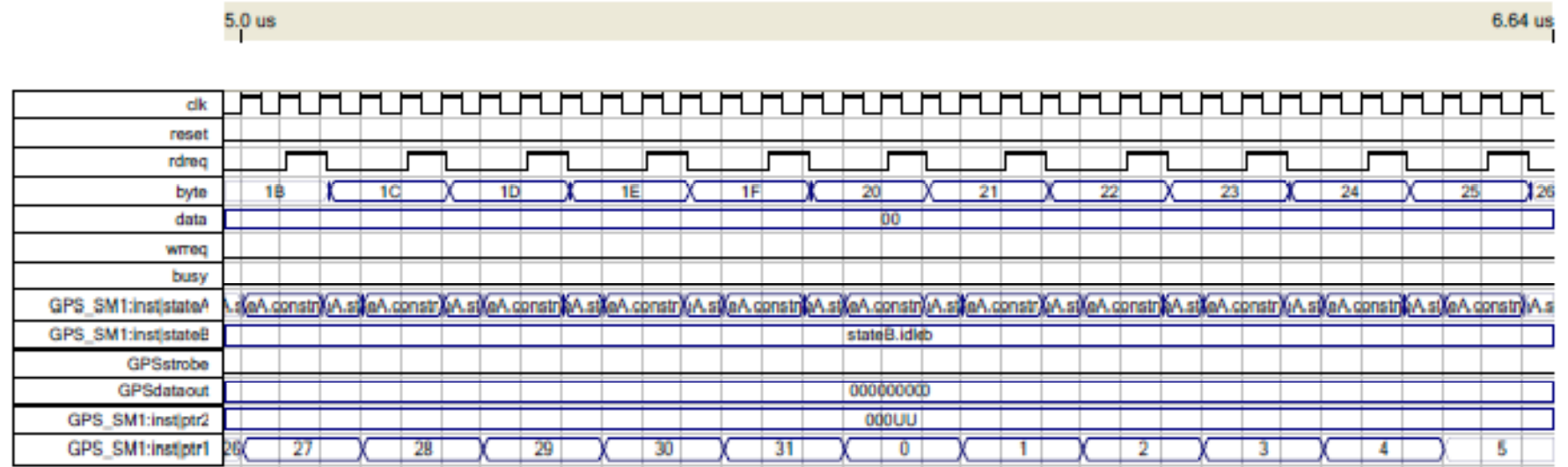


Figure 7.28 - GPS Controller Simulation (part 4)

6.65 us

8.3 us

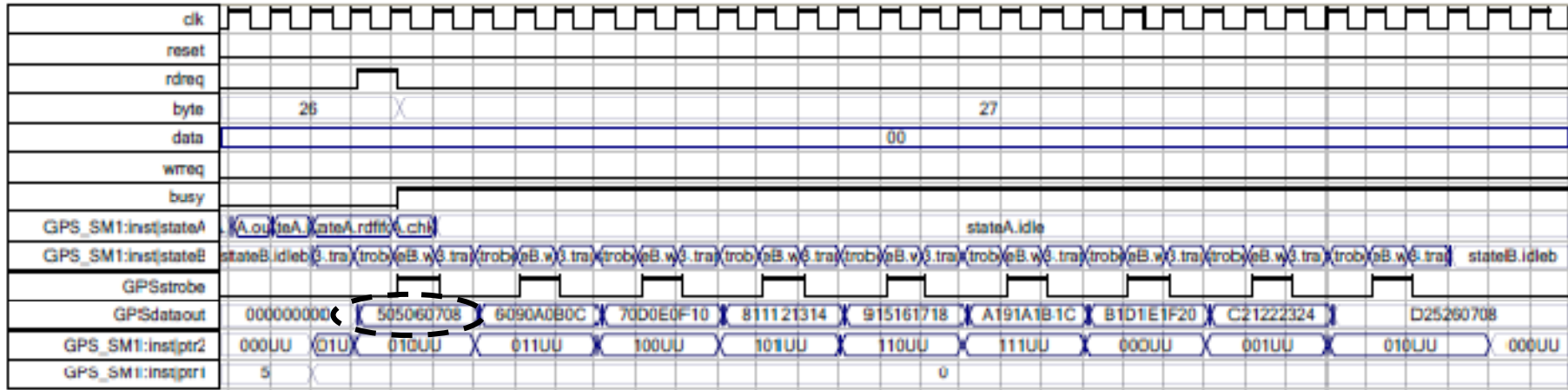


Figure 7.29 - GPS Controller Simulation (part 5)



## 8. CONCLUSION, RECOMMENDATIONS AND FUTURE WORK

### 8.1 CONCLUSION

This thesis reported the process undertaken to develop an integrated avionics platform for research purposes. A basic System Engineering approach was adopted and that defined three distinct phases: a requirements analysis phase, a design and implementation phase and a simulation and testing phase.

A requirements analysis phase was completed, in which the User Requirement Specification (URS) was drawn up, in order to lay the foundation for the project. This included fairly high-level specifications which had to be complied with. The URS was then used to establish the System Requirements Specification (SRS). This specified all the lower-level requirements for the system and was the main drivers in the design phase of the project.

At the commencement of the design phase, a conceptual design of the system was realized in response to the URS and SRS established previously. One concept, among others, taken into account was to create different layers of abstraction which removes the user from the low-level hardware functioning and allows the user high-level access to an application layer, where control algorithms can be implemented. The architecture of the system was also presented and this incorporated an FPGA and microprocessor which communicate via a parallel bus. A functional description of the data flow and timing of the system was also included.

A detailed design of the hardware then followed. This started with the selection of components based upon the SRS. Traceability back to the SRS was also shown in order to justify component choices. Among the main components selected for the design were the Altera Cyclone II FPGA and the Renesas SH7201 microprocessor with integrated FPU. The schematics and printed circuit board layout for a development prototype was then drawn up and the PCB manufactured. The aim of the development prototype board was to use it for testing and development in order to evaluate the feasibility of purchasing the tools, then finally to implement a single stand-alone board incorporating the microprocessor onboard.

A high level overview of the firmware modules required for the functioning of the FPGA was presented followed by the commencement of the VHDL implementation of modules. An important implementation completed was the SPI module. This was implemented to support all the modes

of SPI operation and created a reusable module within the system which attests to the design principle of reusability.

The modules designed and implemented were tested by simulating them in the Altera Quartus II FPGA development software. The results are presented to show the functionality of the modules.

The conceptualization of the software design was presented as well as a flow chart describing its functionality.

Some firmware modules and a final hardware board were not completed due to time constraints.

It is evident from the overall conceptual design, that layers of abstraction were realized through an application layer and offered the user the ability to implement an autopilot system while being unaware to the execution tasks in the hardware layer. Through these layers of abstraction, the user is conceptually presented with user-friendly data which can be used for control purposes. The URS1 and URS9 established in the requirements phase are thus satisfied.

In order to implement backward-compatibility (URS2), a CAN interface was designed so that the avionics system developed in this project could incorporate the existing CAN bus configuration in the ESL.

The microprocessor used in the avionics systems was the Renesas SH7201 which meets the processing power specification set out in URS4 in terms of the floating-point intensive operations required.

The FPGA configuration allowed for scalability of the microprocessor with minimal changes required to the schematic-level design of the system. Essentially one could interchange any component as the FPGA allows for this to be implemented through its flexibility i.e. modules can be written for any new interfaces required. This satisfied the requirements for URS5.

Good engineering practices were observed during all the phases of this project as required in URS6. This is evident through using a system engineering methodology and observing traceability from specification to design as well as implementation to specification. This avionics platform was mainly for research purposes for projects in the ESL. This satisfies the URS6.

Conceptually the system is reconfigurable for different aircraft platforms made possible by the user application layer and therefore allows the user certain amount of access to reload different control systems onto the avionics platform. This complies with URS8.



Components selection and hardware design considered URS10 which required that the system be low-cost, small in size and lightweight. In concept this would allow that the final integrated board to be smaller and lightweight.

URS11 was satisfied through selecting hardware components that were available off-the-shelf and would allow for ease of duplication within minimal time although recommendations are highlighted in the section below in this regard.

This system has created a starting point for further development towards an integrated avionics system. A hardware board was also created which, once populated with components, can be used for further development of the avionics system.

## **8.2 LIMITATIONS**

A key limitation to this study was the overall size of the project which included a final integrated standalone avionics board. A more focused approach in the implementation of the system could have yielded a more integrated system.

## **8.3 RECOMMENDATIONS AND FUTURE WORK**

Recommendations which can be considered in future development of the system are listed in point form below:

- The SH7216 microprocessor can be considered in future designs. It has the same architecture as the SH7201 used in this project but has the added advantage of onboard FLASH memory for programming.
- The level translator which converts the voltages on the I<sup>2</sup>C bus between the FPGA and pressure sensors should be upgraded to a device suitable for I<sup>2</sup>C communication.
- In the PCB design, the internal layer power tracks should be made larger as to decrease the temperature rise under full current load.
- Newer ublox GPS modules such as the LEA-6H and LEA-5H could be considered. These however only have a 2Hz update rate.
- The ADIS 16350 device has become obsolete during the duration of the project and therefore the newer ADIS 16360 and 16365 devices should be used.
- Replace the FAN1112 regulator with the NCP565 regulator.

Among future work to be done includes:

- Populate the PCB with components and test current firmware modules, such as SPI and bus interface for example.
- Complete firmware modules not implemented in this project.
- Implement the microprocessor software and test it on the development board purchased.
- Design a single stand-alone board incorporating the microprocessor as well.

## BIBLIOGRAPHY

- [1] MicroPilot. Products: Autopilots. [Online]. <http://www.micropilot.com/products-mp2028-autopilots.htm>
- [2] Cloud Cap Technology. Autopilots. [Online]. [http://www.cloudcaptech.com/piccolo\\_system.shtm](http://www.cloudcaptech.com/piccolo_system.shtm)
- [3] Adaptive Flight Inc. Products: FCS20. [Online]. [http://www.adaptiveflight.com/products\\_fcs20.html](http://www.adaptiveflight.com/products_fcs20.html)
- [4] H.B. Christophersen, W.J. Pickell, A.A. Koller, S.K. Kannan, and E.N. Johnson, "Small Adaptive Flight Control Systems for UAVs using FPGA/DSP Technology (Georgia Institute of Technology)," *American Institute of Aeronautics and Astronautics*, 2004.
- [5] weControl. Flight Control Systems: wePilot2000. [Online]. <http://www.wecontrol.ch/>
- [6] I. K. Peddle, "Autonomous Flight of a Model Aircraft," Stellenbosch University, Masters Thesis 2005.
- [7] S. Groenewald, "Development of a Rotary-Wing Test Bed for Autonomous Flight," Stellenbosch University, Masters Thesis 2005.
- [8] J. Venter, "Development of an Experimental Tilt-Wing VTOL Unmanned Aerial Vehicle," Masters Thesis 2005.
- [9] W.J. Hough, "Autonomous Aerobatic Flight of an Unmanned Aerial Vehicle," Stellenbosch University, Masters Thesis 2006.
- [10] J. Bijker, "Development of an Attitude Heading Reference System for an Air Ship," Stellenbosch University, Masters Thesis 2006.
- [11] D. Blaauw, "Flight Control System for a Variable Stability Blended-Wing-Body Unmanned Aerial Vehicle," Stellenbosch University, Masters Thesis 2009.
- [12] M.N. Berberan-Santos, E.N. Bodunov, and L. Pogliani, "On the Barometric Formula," *American Journal of Physics*, vol. 65, no. 5, May 1997.

- [13] M.V. Cook, *Flight Dynamics Principles*, 2nd ed. MA: Elsevier Ltd, 2007.
- [14] ublox, "ANTARIS4 GPS Modules System Integration Manual (SIM)," Datasheet [Rev. A1], 2007.
- [15] ublox, "LEA-4x ANTARIS4 GPS Modules ," Data Sheet [Rev. 2], 2008.
- [16] RF Design. GPS Products: Antaris 4 Modules. [Online]. <http://www.rfdesign.co.za>
- [17] Silicon Sensing. Glossary. [Online]. <http://www.siliconsensing.com/information/glossary>
- [18] O.J. Woodman, "An Introduction to Inertial Navigation," University of Cambridge, Technical Report ISSN 1476-2986, 2007.
- [19] W. Stockwell. "Angle Random Walk," Crossbow Technology. [Online]. <http://www.xbow.com/pdf/AngleRandomWalkAppNote.pdf>
- [20] Memsense. Glossary of Technical Terms. [Online]. <http://www.memsense.com/index.php/Support-Pages/inertialtermsh.html>
- [21] F. Foust, "Secure Digital Card Interface for the MSP430," Michigan State University, Application Note 2004.
- [22] W. Duckitt, "The Design of a High-Performance, Floating-Point Embedded System for Speech Recognition and Audio Research Purposes," Stellenbosch University, Masters Thesis 2007.
- [23] B.J. Visser, "The Precision Landing of an Unmanned Aerial Vehicle," Stellenbosch University, Masters Thesis 2008.
- [24] J. Venter, The A to Z of Servos, 2005, ESL Aero meeting presentation, Stellenbosch University.
- [25] Analog Devices, "Tri Axis Inertial Sensor: ADIS 16350/16355," Datasheet [Rev. A].
- [26] Sensortech, "I2C Bus Communication with Sensortech's Digital HCLA, HCA and HDI Pressure Sensors," Application Note E/11155/A.
- [27] Altera Corporation, "Cyclone II Device Handbook," Datasheet [Volume 1] 2007.
- [28] Trax Interconnect, "Material Lay Up Tables (Multilayer Panels)," Datasheet [Document No. 7.2.5.13.3.1] 2010.

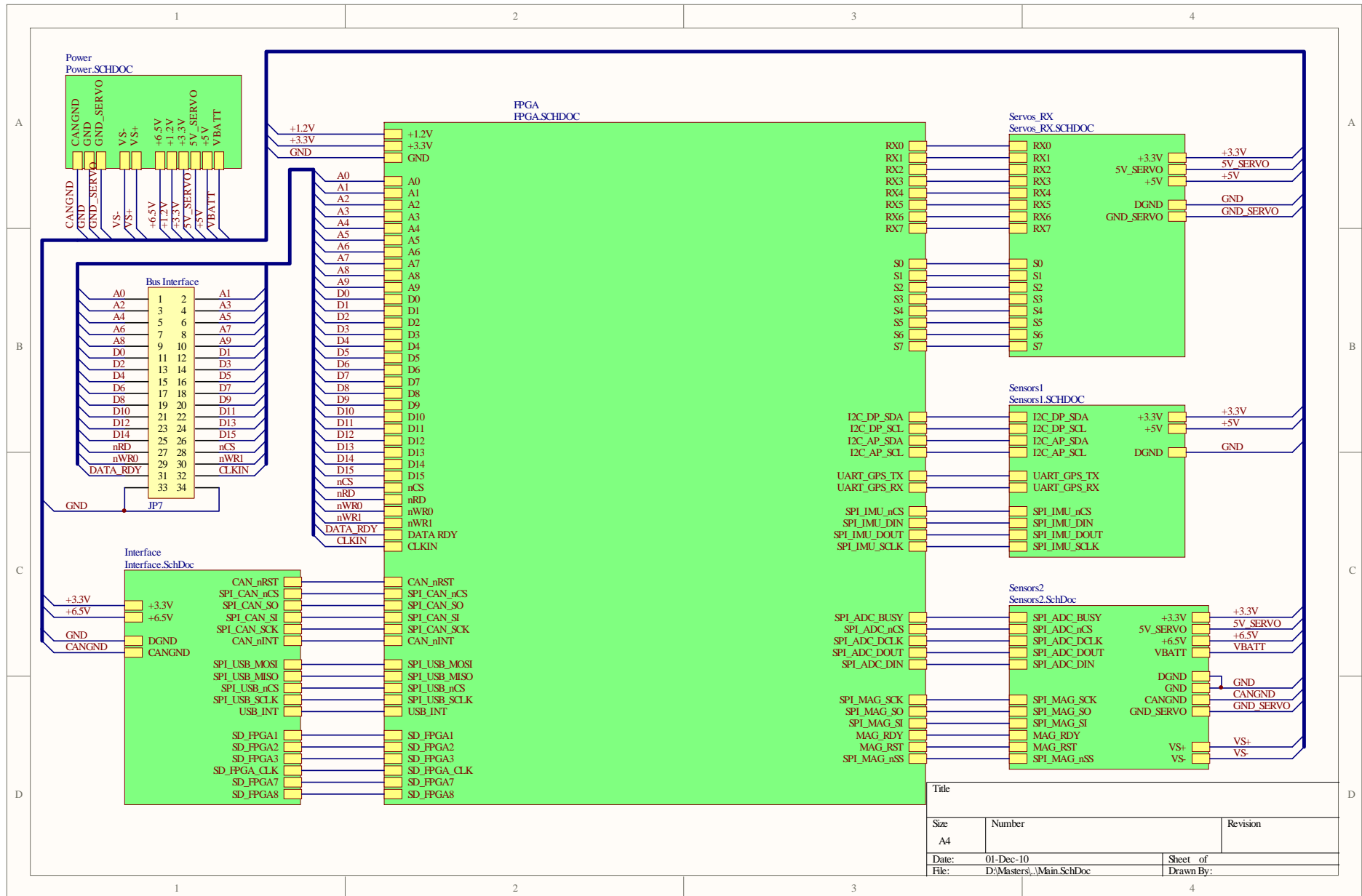
- [29] Barry Olney, "Multilayer PCB Stackup Planning," In-Circuit Design Pty Ltd, Application Note.
- [30] PCB Trace Width Calculator. [Online].  
<http://www.circuitcalculator.com/wordpress/2006/01/31/pcb-trace-width-calculator/>
- [31] Mark Zwoliński, *Digital System Design with VHDL*, 2nd ed. Essex, England: Pearson Education Limited, 2004.
- [32] Dr. M.E.S. Elrabaa. Fundamentals of Computer Engineering (Course Material: "Digital System Design Using Data path (DP) and Control Unit (CU)"). [Online].  
[http://www.ccse.kfupm.edu.sa/~elrabaa/coe202/DP\\_CU.pdf](http://www.ccse.kfupm.edu.sa/~elrabaa/coe202/DP_CU.pdf)
- [33] Tom Scott. (2007, May) Mission Technologies Webpage: Simple SPI Master Slave VHDL example code. [Online]. <http://www.missiontech.co.nz/index.php?page=read-article&articleId=76>
- [34] W. Wolf, *Computer As Components: Principles of Embedded Computing System Design*. San Francisco: Morgan Kaufmann Publishers, 2001.
- [35] I.K. Peddle, Computer Systems 214 Course, Practical Notes, Department of Electrical & Electronic Engineering, Stellenbosch University, 2005.
- [36] Altera Corporation, Sample behavioral waveforms for design file dualportram.vhd, Automatically generated waveforms by Quartus II LPM wizard.
- [37] Renesas Tehchnology, "SH7201 Group Hardware Manual," Datasheet [Rev. 1.00] 2006.
- [38] Texas Instruments, "ADS8344: 16-Bit, 8-Channel Serial Output Sampling Analog-to-Digital Converter," Datasheet [Rev. E] 2006.
- [39] ublox, "ANTARIS Positioning Engine: NMEA and UBX Protocol Specification," Specification Document [Doc ID: GPS.G3-X-03002-D] 2003.

# APPENDIX A: SPECIFICATIONS OF THE ADIS 16360 AND 16365

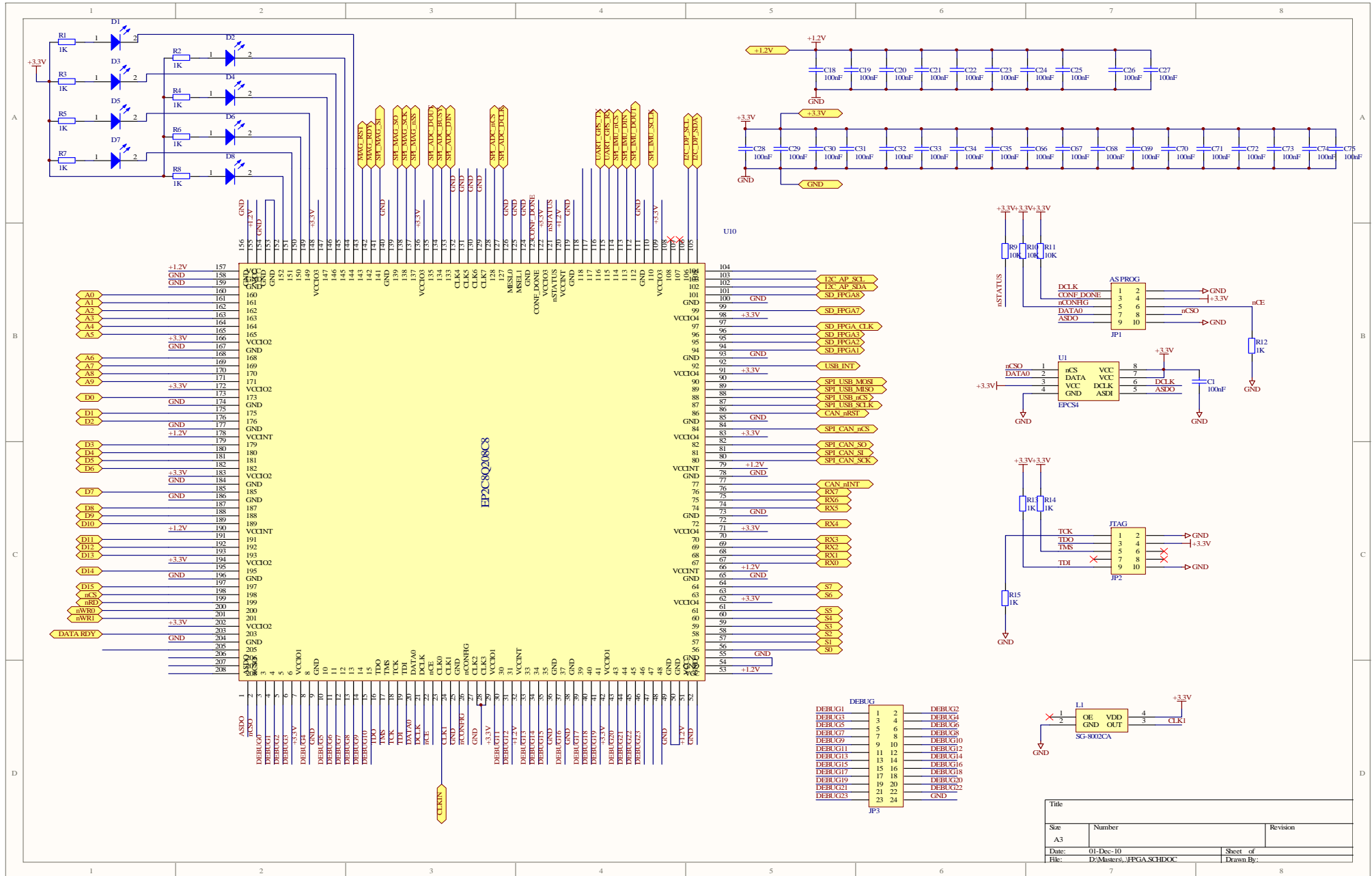
Table A.1 - Analog Devices ADIS 16360 and 16365 IMU Specifications

	Specification	ADIS 16360	ADIS 16365 (High Precision Calibration)
	Output Type	Digital (SPI)	Digital (SPI)
GYRO SPECIFICATIONS per axis	Dynamic Range	$\pm 300^\circ/\text{s}$ ; $\pm 150^\circ/\text{s}$ ; $\pm 75^\circ/\text{s}$	$\pm 300^\circ/\text{s}$ ; $\pm 150^\circ/\text{s}$ ; $\pm 75^\circ/\text{s}$
	Bias Instability	$0.007^\circ/\text{s}$ ( $1\sigma, 25^\circ\text{C}$ )	$0.007^\circ/\text{s}$ ( $1\sigma, 25^\circ\text{C}$ )
	Bias Temperature Coefficient	$0.025^\circ/\text{s}/^\circ\text{C}$	$0.01^\circ/\text{s}/^\circ\text{C}$
	Angular Random Walk	$2.0^\circ/\sqrt{\text{hr}}$ ( $1\sigma$ )	$2.0^\circ/\sqrt{\text{hr}}$ ( $1\sigma$ )
	Nonlinearity	$\pm 0.1\%$ of FS	$\pm 0.1\%$ of FS
	Noise Density	$0.044^\circ/\text{s}/\sqrt{\text{Hz}}$ rms	$0.044^\circ/\text{s}/\sqrt{\text{Hz}}$ rms
	Bandwidth	330Hz	330Hz
	Sensitivity	$0.05^\circ/\text{s}/\text{LSB}$ ( $\pm 300^\circ/\text{s}$ ) $0.025^\circ/\text{s}/\text{LSB}$ ( $\pm 150^\circ/\text{s}$ ) $0.0125^\circ/\text{s}/\text{LSB}$ ( $\pm 75^\circ/\text{s}$ )	$0.05^\circ/\text{s}/\text{LSB}$ ( $\pm 300^\circ/\text{s}$ ) $0.025^\circ/\text{s}/\text{LSB}$ ( $\pm 150^\circ/\text{s}$ ) $0.0125^\circ/\text{s}/\text{LSB}$ ( $\pm 75^\circ/\text{s}$ )
ACCELEROMETER SPECIFICATIONS per axis	Dynamic Range	$\pm 18\text{g}$	$\pm 18\text{g}$
	Bias Instability	$0.2\text{mg}$ ( $1\sigma$ )	$0.2\text{mg}$ ( $1\sigma$ )
	Velocity Random Walk	$0.2\text{ m/s}/\sqrt{\text{hr}}$ ( $1\sigma$ )	$0.2\text{ m/s}/\sqrt{\text{hr}}$ ( $1\sigma$ )
	Bias Temperature Coefficient	$\pm 4\text{ mg}/^\circ\text{C}$	$\pm 0.3\text{ mg}/^\circ\text{C}$
	Sensitivity	$3.33\text{mg}/\text{LSB}$	$3.33\text{mg}/\text{LSB}$
	Bandwidth	330Hz	330Hz
	Noise Density	$0.5\text{mg}/\sqrt{\text{Hz}}$ rms	$0.5\text{mg}/\sqrt{\text{Hz}}$ rms
	Nonlinearity	$\pm 0.1\%$ of FS	$\pm 0.1\%$ of FS

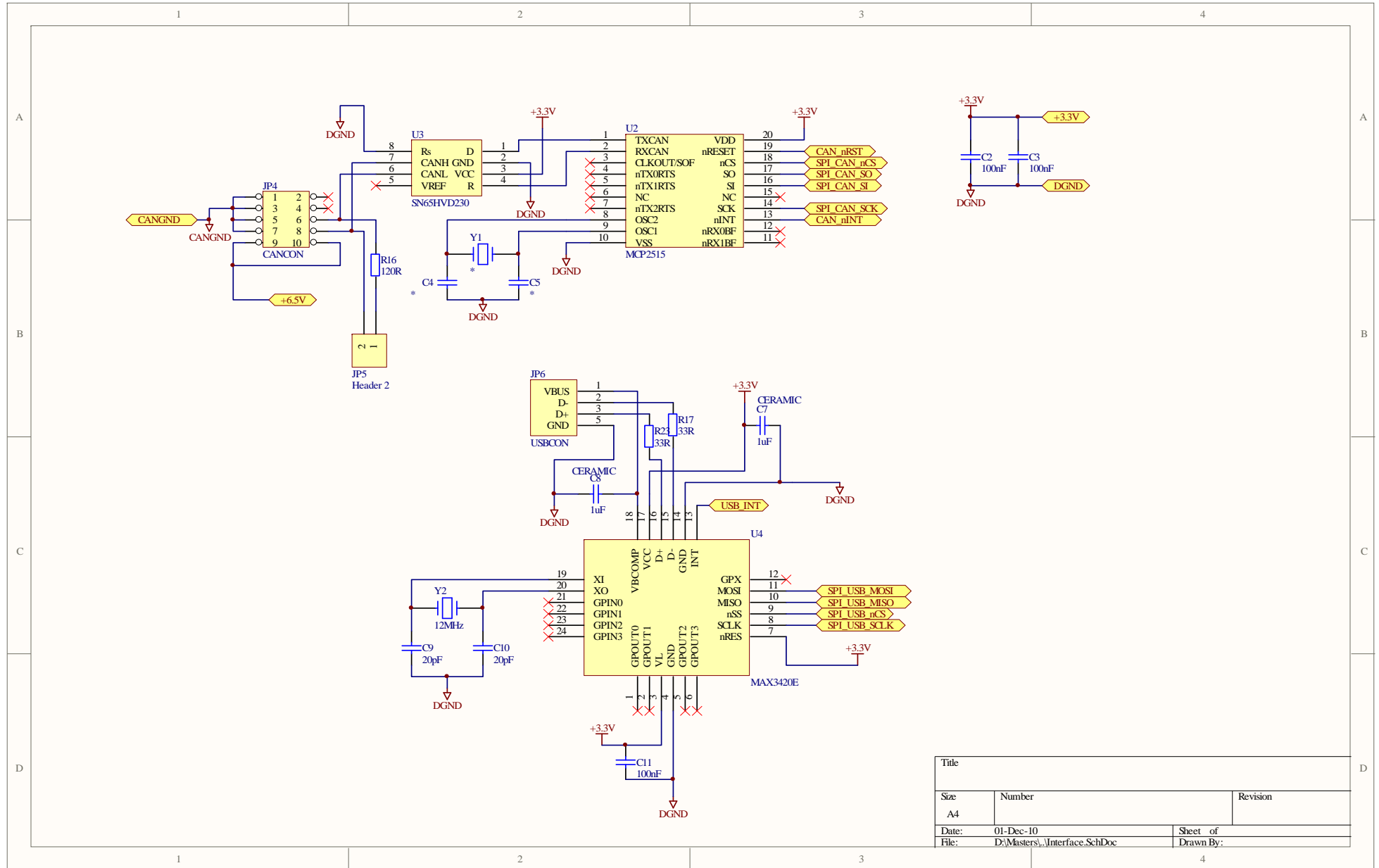
## **APPENDIX B: SCHEMATIC DIAGRAMS**



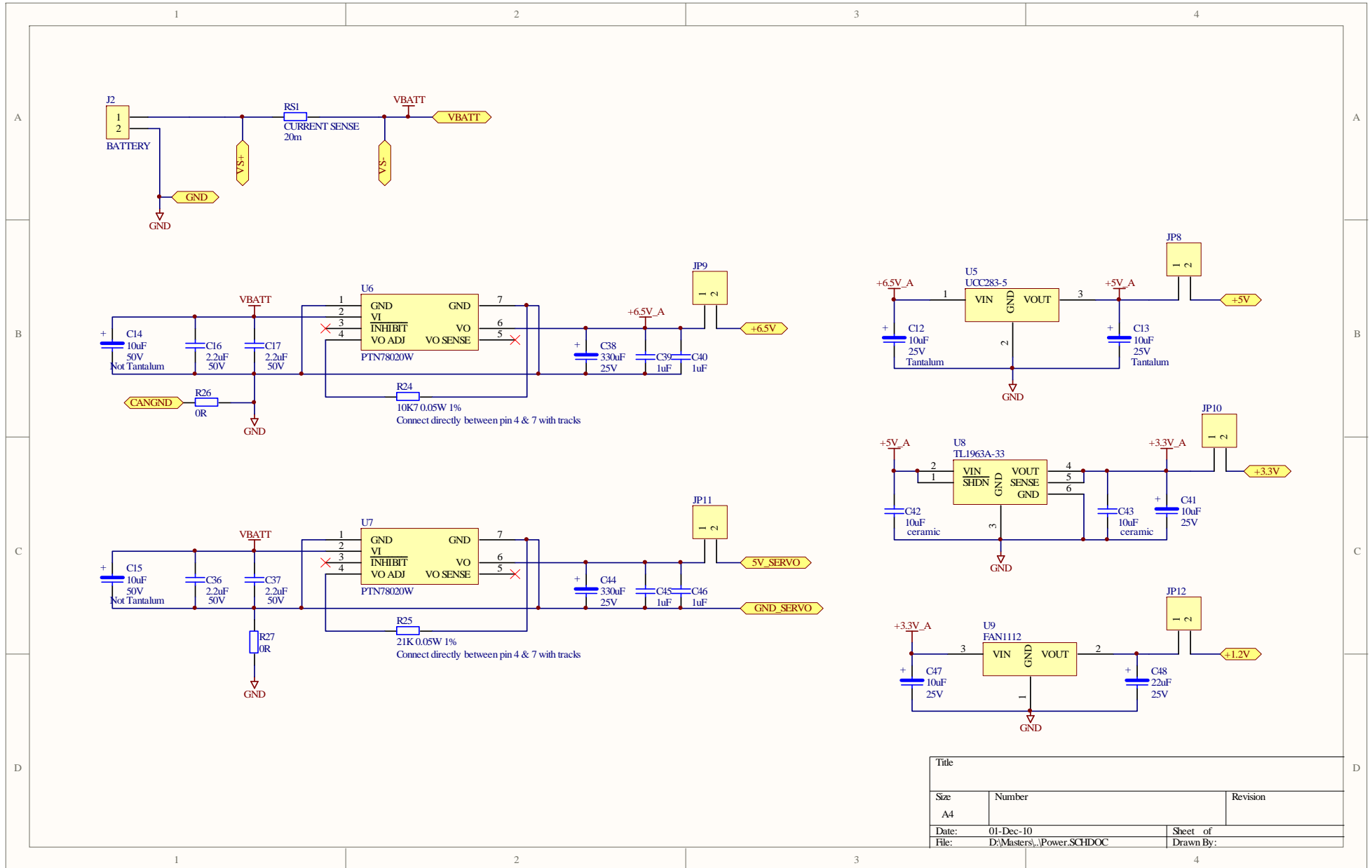




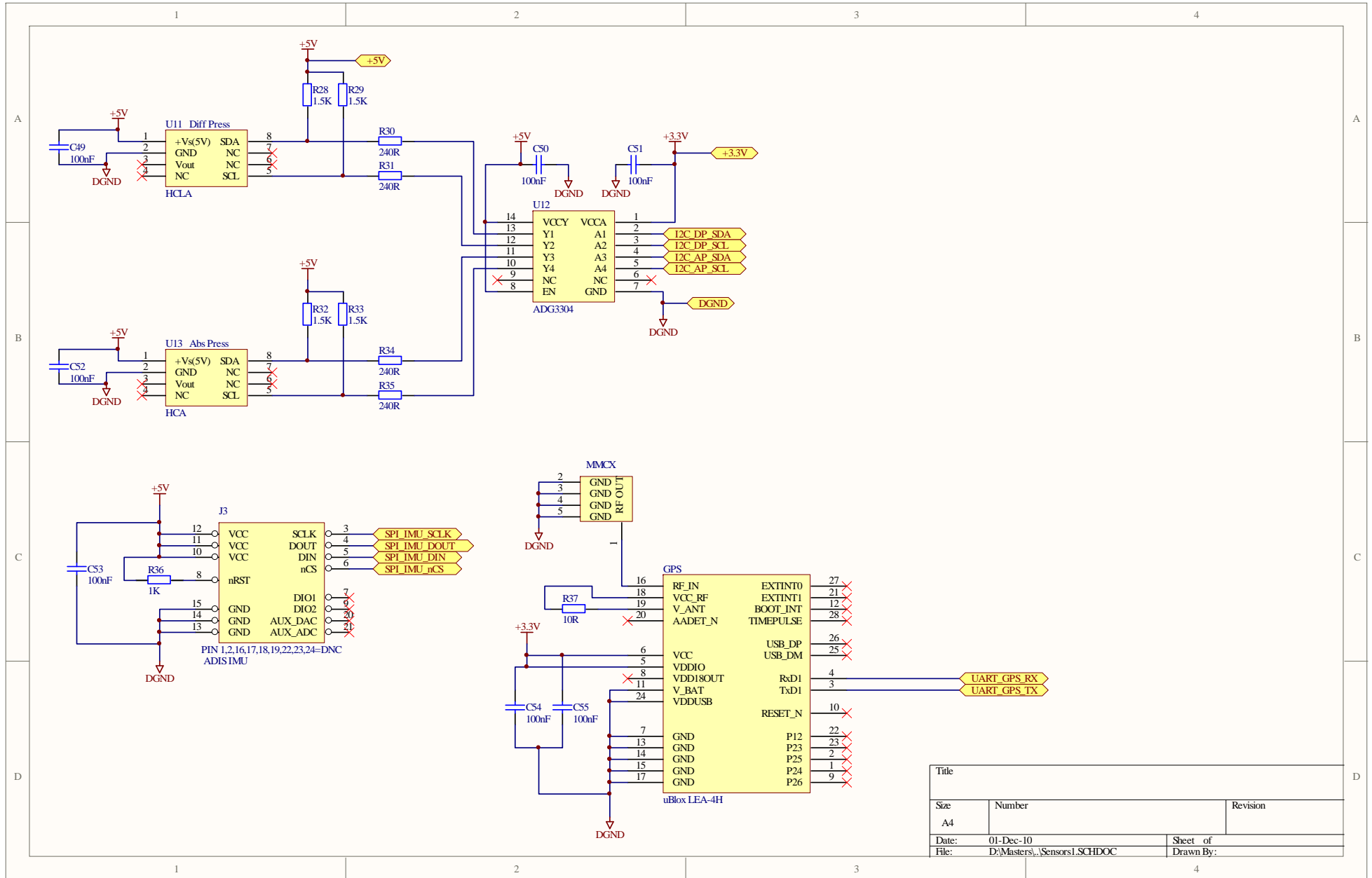
Title		
Size	Number	Revision
A3		
Date:	01-Dec-10	Sheet of
File:	D:\Masters\...FPGA.SCHDOC	Drawn By:



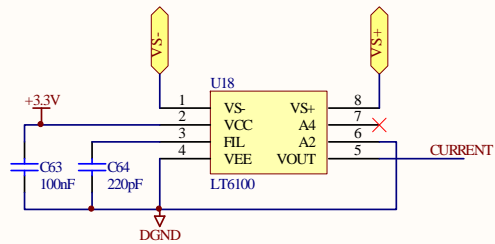
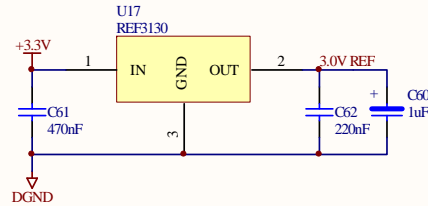
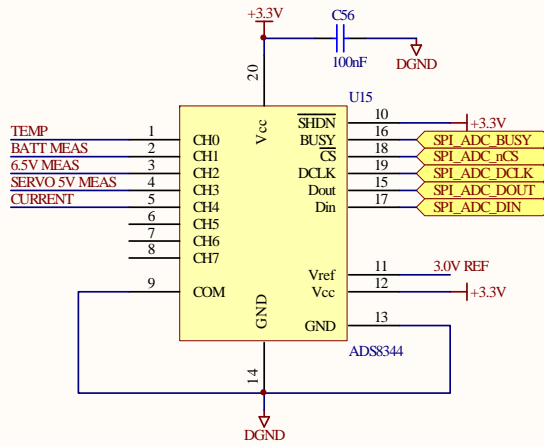
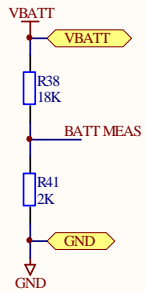
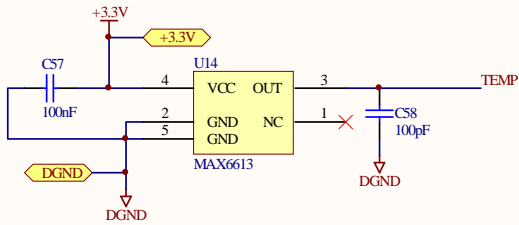
Title		
Size	Number	Revision
A4		
Date:	01-Dec-10	Sheet of
File:	D:\Masters\...Interface.SchDoc	Drawn By:



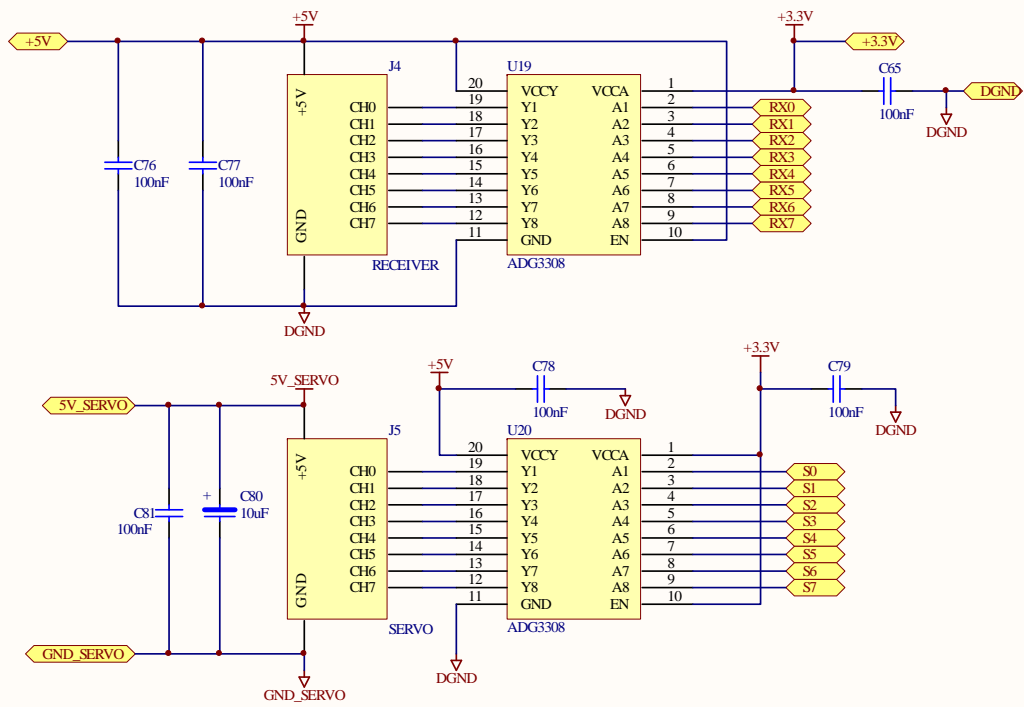
Title		
Size	Number	Revision
A4		
Date:	01-Dec-10	Sheet of
File:	D:\Masters\...\Power.SCHDOC	Drawn By:



Title		
Size	Number	Revision
A4		
Date:	01-Dec-10	Sheet of
File:	D:\Masters\1_Sensors\1.SCHDOC	Drawn By:



Title		
Size	Number	Revision
A4		
Date:	01-Dec-10	Sheet of
File:	D:\Masters\...\Sensors2.SchDoc	Drawn By:



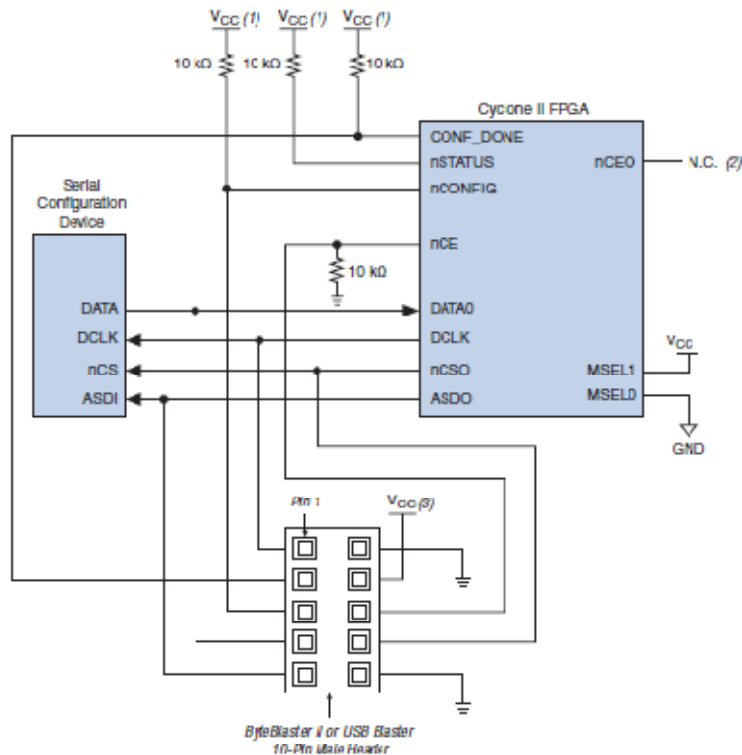
Title		
Size A4	Number	Revision
Date: 01-Dec-10	Sheet of	
File: D:\Masters\ Servos RX.SCHDOC	Drawn By:	

# APPENDIX C: FPGA ACTIVE SERIAL AND JTAG CONFIGURATION SCHEME SCHEMATICS

(Excerpts taken from [27])

## Active Serial Configuration (Serial Configuration Devices)

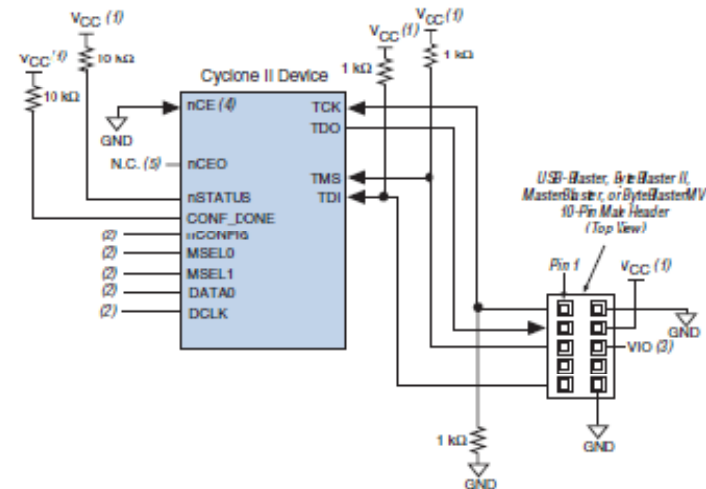
Figure 13-7. In-System Programming of Serial Configuration Devices



**Notes to Figure 13-7:**

- (1) Connect these pull-up resistors to 3.3-V supply.
- (2) The nCEO pin can be left unconnected or used as a user I/O pin when it does not feed other device's nCE pin.
- (3) Power up the ByteBlaster II or USB Blaster cable's V<sub>CC</sub> with a 3.3-V supply.

Figure 13-22. JTAG Configuration of a Single Device Using a Download Cable

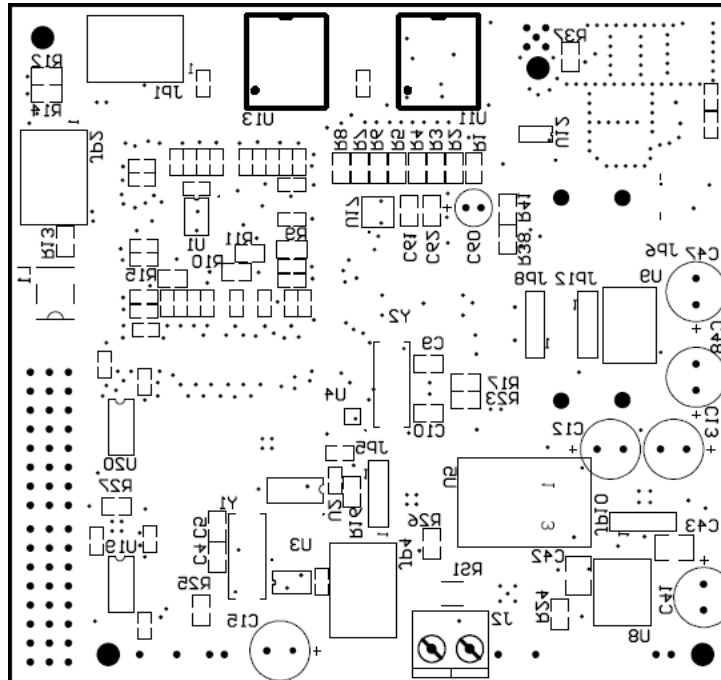


**Notes to Figure 13-22:**

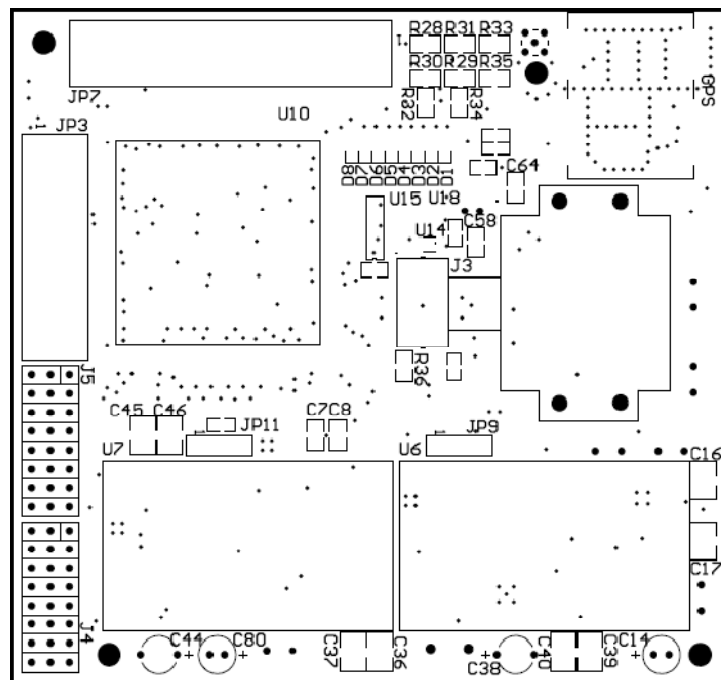
- (1) The pull-up resistor should be connected to the same supply voltage as the USB-Blaster, MasterBlaster (V<sub>IO</sub> pin), ByteBlaster II, or ByteBlasterMV cable.
- (2) Connect the nCONFIG and MSEL [1 . . 0] pins to support a non JTAG configuration scheme. If only JTAG configuration is used, connect the nCONFIG pin to V<sub>CC</sub>, and the MSEL [1 . . 0] pins to ground. In addition, pull DCLK and DATA0 to either high or low, whichever is convenient on your board.
- (3) Pin 6 of the header is a V<sub>IO</sub> reference voltage for the MasterBlaster output driver. V<sub>IO</sub> should match the device's V<sub>CCIO</sub>. Refer to the MasterBlaster Serial/USB Communications Cable Data Sheet for this value. In the ByteBlasterMV, this pin is a no connect. In the USB-Blaster and ByteBlaster II, this pin is connected to nCE when it is used for AS programming, otherwise it is a no connect.
- (4) nCE must be connected to GND or driven low for successful JTAG configuration.
- (5) The nCEO pin can be left unconnected or used as a user I/O pin when it does not feed other device's nCE pin.

## APPENDIX D: COMPONENT PLACEMENT

The figure below shows the bottom layer silkscreen overlay of the PCB with the component placement viewed through the top layer (not to scale).

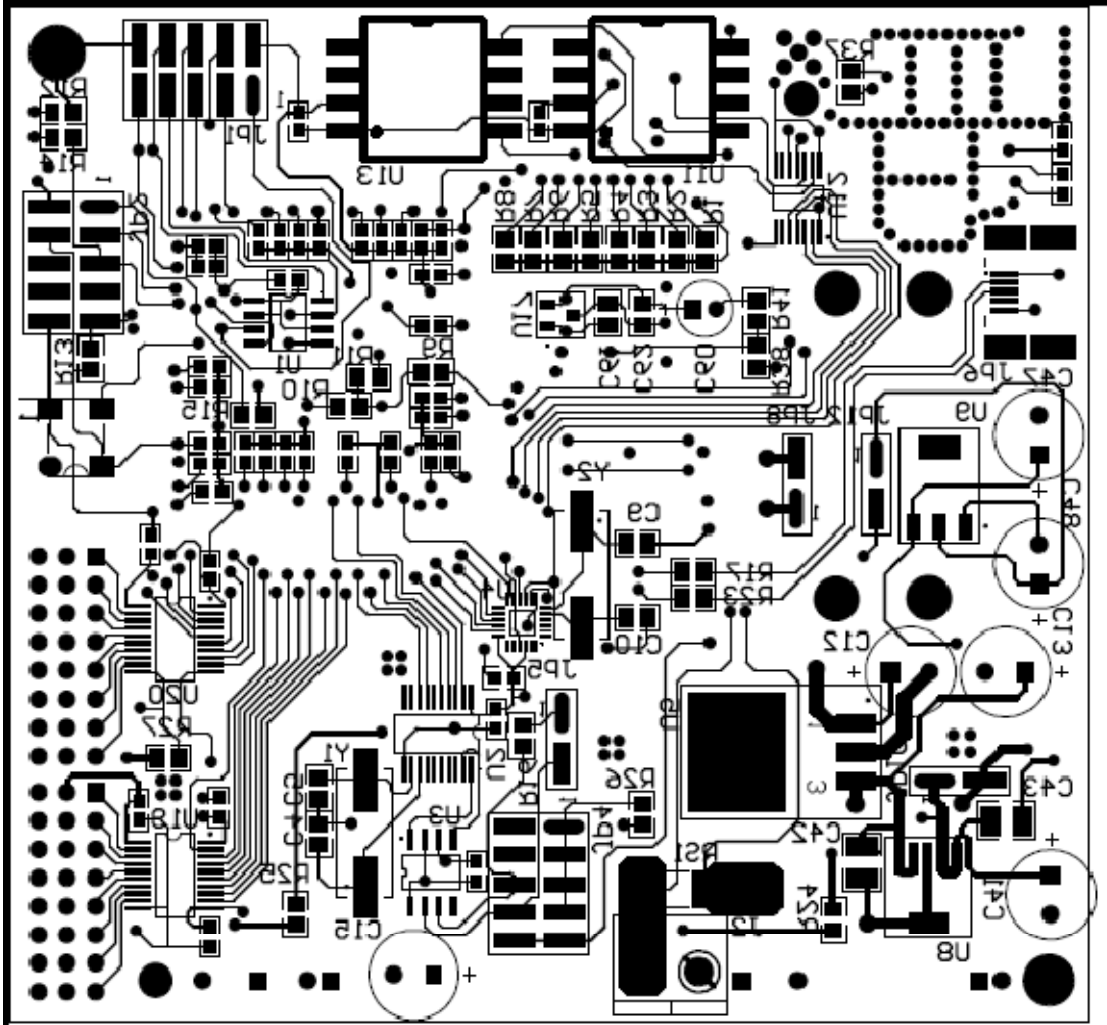


The figure below shows the top layer component placement of the PCB (not to scale).

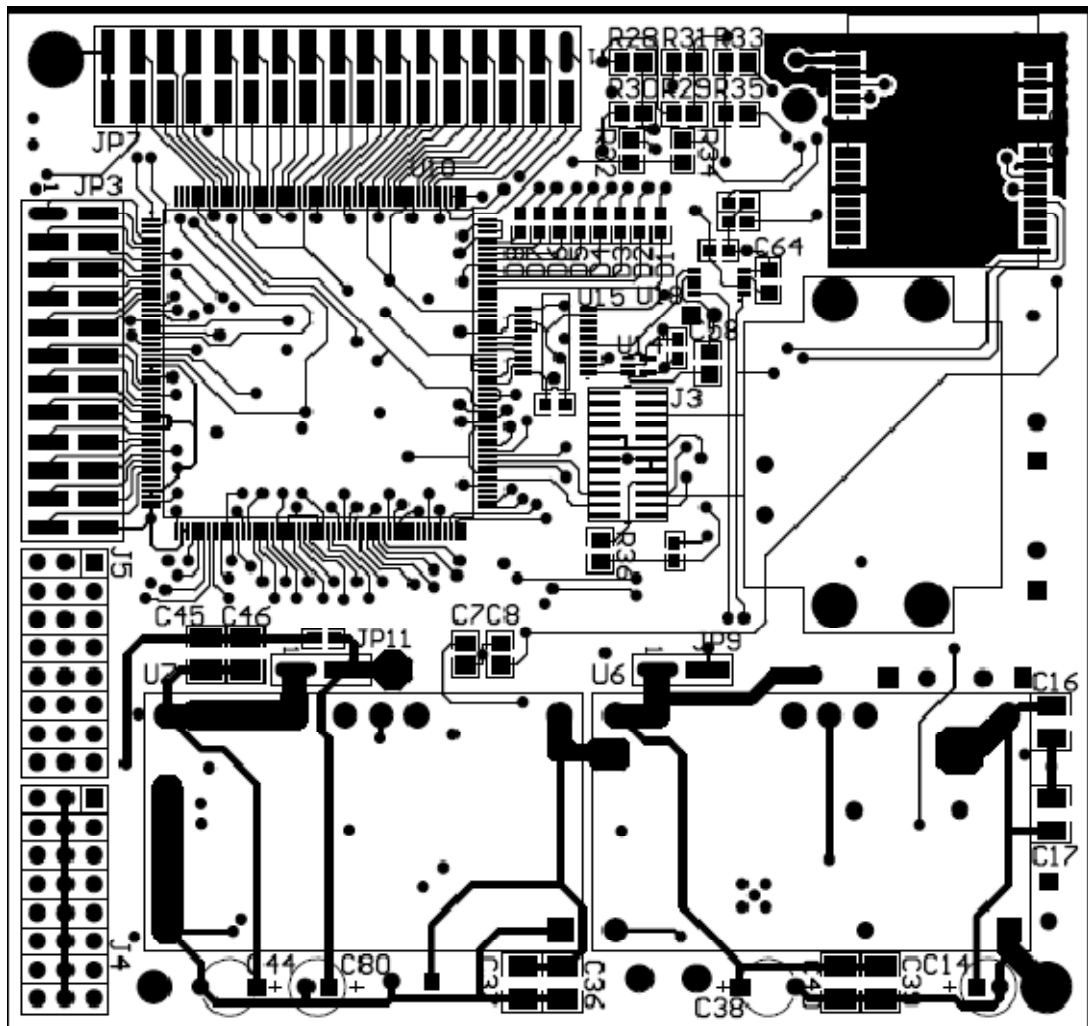




APPENDIX E: PRINTED CIRCUIT BOARD

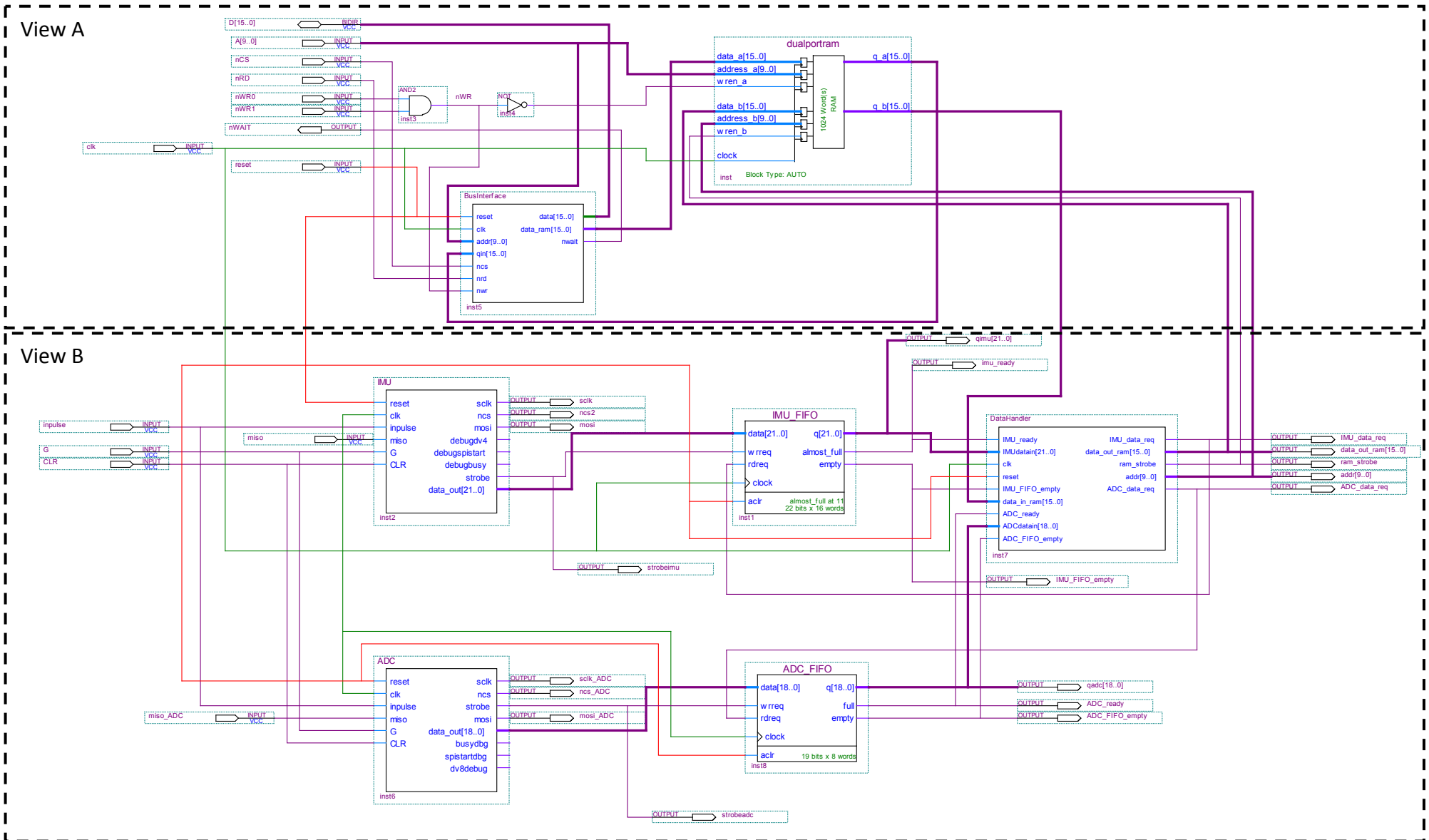


Bottom Layer of the PCB (Not to scale)

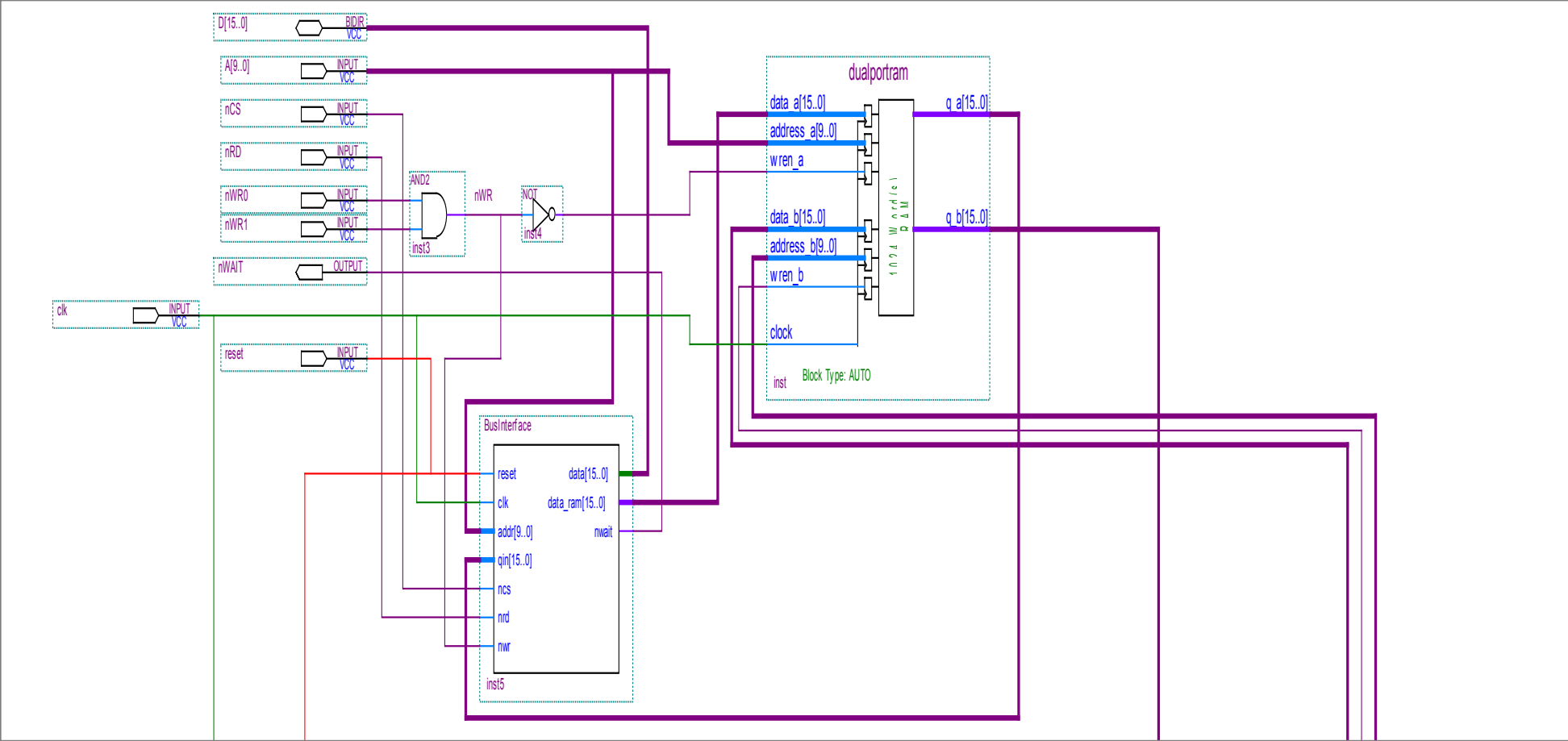


Top Layer of PCB (Not to scale)

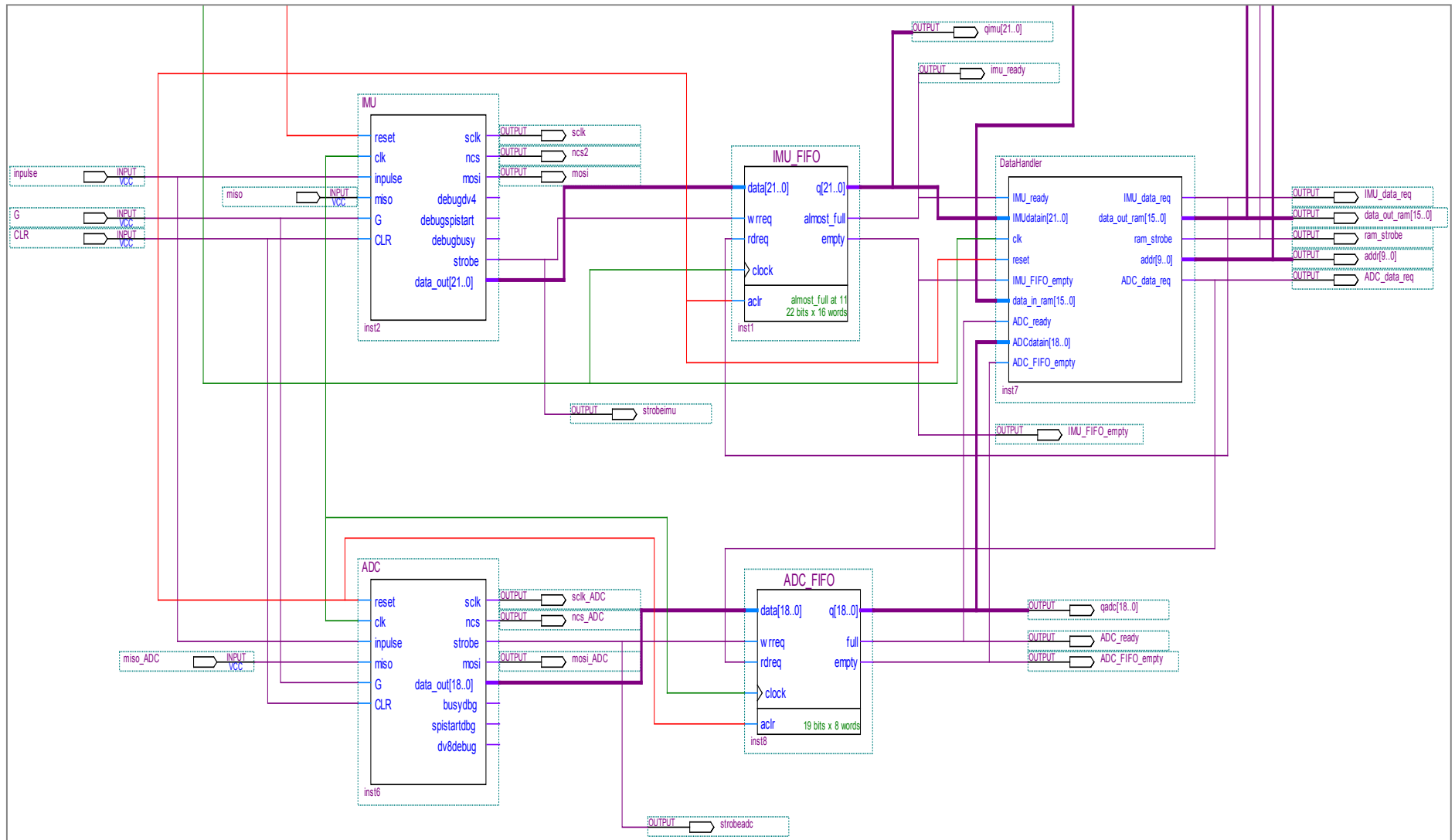
# APPENDIX F-1: PARTIALLY COMPLETED MAIN SYSTEM-BUILDUP



# APPENDIX F-2: View A OF MAIN SYSTEM BUILD UP



# APPENDIX F-3: VIEW B OF MAIN SYSTEM BUILD UP



## APPENDIX G – PICTURES OF THE HARDWARE

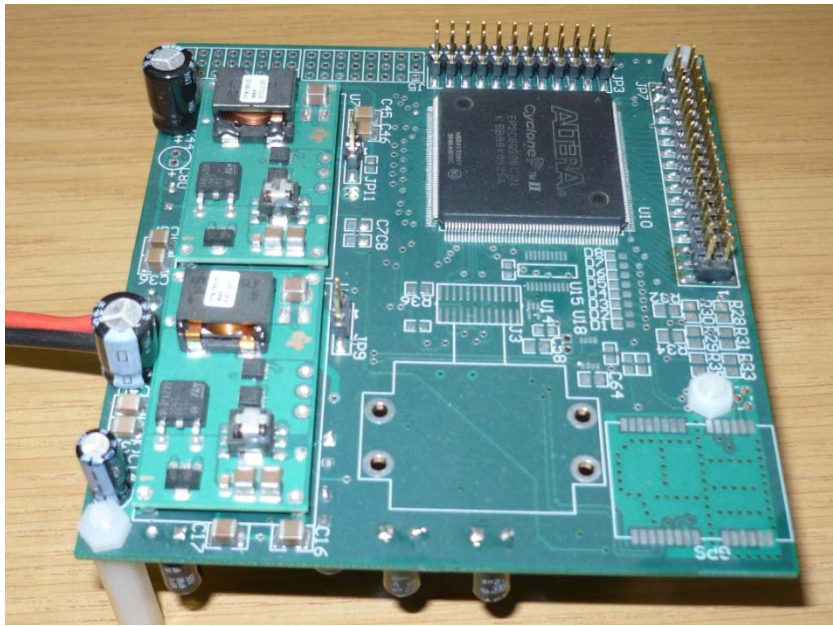


Photo of the Top layer of the Prototype Development Printed Circuit Board designed in this project (not fully populated with components).

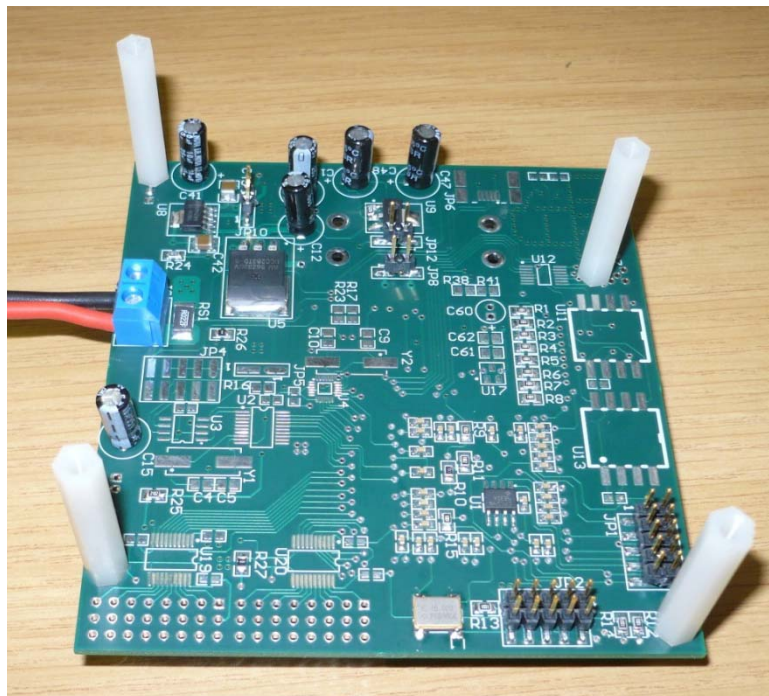
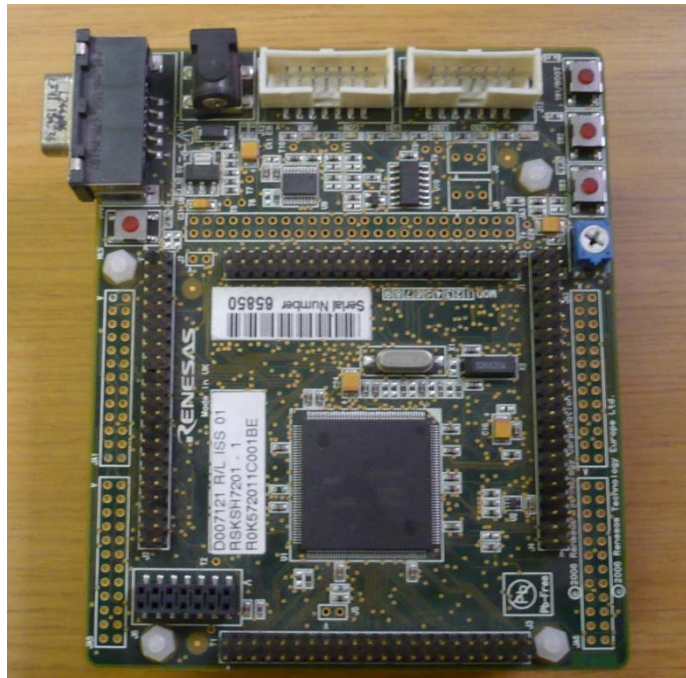
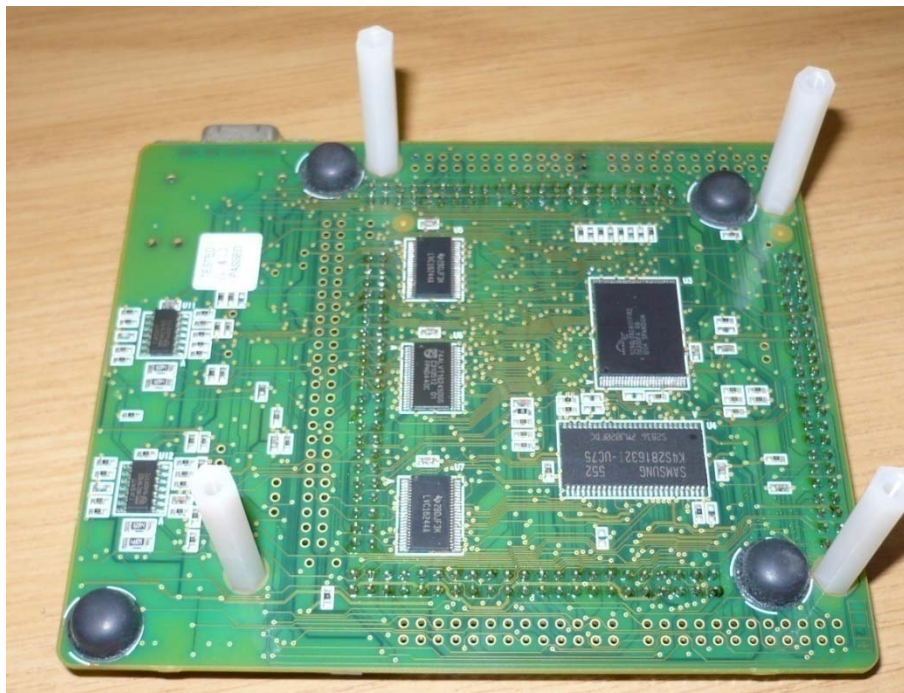


Photo of the Bottom layer of the Prototype Development Printed Circuit Board designed in this project (not fully populated with components).





**Photo of the Purchased Renesas SH7201 Development Board (Top)**



**Photo of the Purchased Renesas SH7201 Development Board (Bottom)**