Applications for FPGAs on Nanosatellites

Thong Thai

A thesis submitted to

the Faculty of Graduate Studies

in partial fulfillment of the requirements

for the degree of

Master of Science

Graduate Program in Earth and Space Science

York University

Toronto, Ontario

April 2014

Abstract

This thesis examines the feasibility of using a Field Programmable Gate Array (FPGA) based design on-board a CubeSat-sized nanosatellite. FPGAs are programmable logic devices that allow for the implementation of custom digital hardware on a single Integrated Circuit (IC). By using these FPGAs in spacecraft, more efficient processing can be done by moving the design onto hardware. A variety of different FPGA-based designs are looked at, including a Watchdog Timer (WDT), a Global Positioning System (GPS) receiver, and a camera interface.

Dedicated to Loan

Table of Contents

Abstract ii

Dedication iii

Contents iv

List of Tables ix

List of Figures x

Acronyms xii

Chapter 1 Introduction 1

1.1 CubeSat Architecture 2

1.2 CubeSat Missions 3

1.3 Satellite Data Processing 4

1.4 FPGAs 4

1.5 FPGAs in Space 7

1.6 FPGAs in CubeSat Applications / Current Implementations 8

1.6.1 Processing 9

1.6.2 Radio 10

1.6.3 Power 10

1.7 Research Motivation 11

1.8 Thesis Outline 12

Chapter 2 Field Programmable Gate Arrays 13
2.1 Introduction to FPGAs 14
2.2 FPGA Work Flow 15
2.3 FPGA Fabric 17
2.4 FPGA Technologies 17
2.5 Comparison to Other Technologies 19
2.5.1 Software versus Hardware 19
2.6 Microprocessors, FPGAs and ASICs 20
2.6.1 Cost 21
2.6.2 Power consumption 22
2.6.3 Security 23
2.6.4 Summary 23
2.7 System on a Chip 24
2.8 Performance 25
2.8.1 Clock Speeds 26
2.8.2 Operating Temperature 26
2.9 Radiation Concerns 27
2.10 Review of FPGAs 28
2.11 FPGA Test Cases 30
2.11.1 Power Board 30
2.11.2 GPS Signal Processing 30
2.11.3 Camera Interface 30
Chapter 3 FPGA Watchdog Timer for Electrical Power Subsystem 32
3.1 Power Board Design 32
3.2 FPGA 34

3.3 Watchdog Timer 35

3.4 Pin Monitoring 36

3.5 Serial Peripheral Interface Bus Interface 37

3.6 Resource Usage 38

3.7 Testing 38

3.7.1 Verification 39

3.7.2 Feasibility 40

3.7.3 Components 41

3.7.4 Future Improvements 41

Chapter 4 FPGA GPS Baseband Processor 44

4.1 Software Defined Radio 44

4.2 Why FPGAs for SDR 45

4.3 Background on GPS Hardware 45

4.4 GPS Baseband Processor Design 46

4.5 Test Hardware 49

4.5.1 Nios II Soft-Core Processor 51

4.5.2 System on Chip 55

4.6 GPS Data Acquisition Block 57

4.6.1 Alternatives 57

4.6.2 Implementation 58

4.6.3 Acquisition Module 59

4.6.4 State Machine 60

4.6.5 Avalon Interface 62

4.6.6 Software Driver 63

4.7 GPS Satellite Acquisition Block 64

4.7.1 Design Alternatives . . . 65

4.7.2 Implementation . . . 66

4.7.3 Multiplication and Representation of Numbers . . . 67

4.7.4 Acquisition Channel . . . 69

4.7.5 Replica Carrier Resolution . . . 69

4.7.6 Replica Carrier Frequency Resolution . . . 70

4.7.7 Numerically Controlled Oscillators . . . 72

4.7.8 Replica Signals . . . 73

4.7.9 State Machine . . . 74

4.7.10 Interface . . . 76

4.8 Testing . . . 77

4.8.1 Feasibility . . . 77

4.8.2 Future Improvements . . . 81

Chapter 5 FPGA for Camera Interfacing 83

5.1 Interfacing Components . . . 83

5.2 Limitations of Microcontroller UART . . . 84

5.2.1 Alternative Solutions . . . 85

5.3 Implementation . . . 87

5.3.1 Oversampling . . . 87

5.3.2 Verification . . . 89

5.3.3 Feasibility . . . 89

Chapter 6 Conclusions and Future Work 91

6.1 Conclusions . . . 92

6.1.1 FPGA Development . . . 92

6.1.2 In comparison to software development . . . 93

6.2 Contribution 95

6.3 Future Work 96

Chapter 7 References 98

Chapter A Appendix CubeSat Power Board Design 107

List of Tables

2.1 Technology Comparison 24

2.2 FPGA Comparison 29

3.1 Amount of FPGA resources used to implement the Watchdog Timer design with a CoreABC processor 38

3.2 Amount of microcontroller memory used to run the FreeRTOS firmware . . 38

4.1 Nios II Floating-Point Benchmark Results 54

4.2 Dhrystone Benchmark Results 54

4.3 FPGA computation using a single logic gate 68

4.4 Equivalent computation using signed multiplication 68

4.5 FPGA resource usage comparison between different alternative NCO implementations 78

4.6 FPGA resource usage comparison between the smallest alternative implementation and the 1-bit NCO implementation 79

4.7 Time to complete a serial-search on a laptop compared to Field Programmable Gate Array (FPGA) 81

List of Figures

1.1 Comparison of hardware through the decades 6

2.1 Typical design work flow 15

3.1 Completed prototype power board that was used for simulated space-environment testing 33

3.2 Overview of FPGA and microcontroller connections 35

4.1 Overview of GPS processor design implemented on the FPGA 47

4.2 Test hardware for GPS processor - Terasic DE2-115 development board with an Altera Cyclone IV FPGA, 128MB SDRAM and 2MB SRAM below, and the MAX2769 evaluation board on top 50

4.3 Compiler source code for Nios II software multiplication 52

4.4 Overview of GPS processor design implemented on the FPGA 56

4.5 Overview of the GPS Data Acquisition module. Data from the frontend is read by a 2-port FIFO, and written from the FIFO into a shift register. Each bit of the shift register is converted to an 8-bit signed integer 59

4.6 Overview of GPS Data Acquisition State Machine 61

4.7 GPS Data Acquisition Avalon interface. The Avalon interface is used to allow the data acquisition module to connect to the Avalon bus used by the processor. The processor controls the acquisition module through the registers exposed by the Avalon Slave interface. Data captured by the acquisition module is stored into external SDRAM through the Avalon Master interface. 63
4.8 Overview of GPS processor design implemented on the FPGA. Notice that this figure differs from that of Figure 4.5.2 by replacing the FFT module with the hardware-based GOS Signal Acquisition module 65
4.9 GPS Signal Acquisition module overview 67
4.10 Comparison between a) a 1-bit replica carrier signal, and b) a 64-bit replica carrier signal after cross-correlation 70
4.11 Different carrier frequencies and the effect on CA-code correlation 71
4.12 Overview of GPS Acquisition State Machine 75

5.1 LVDS Clock Timing 85
5.2 Oversampling 86
5.3 Oversampling Modules 88
5.4 Edge Detector Circuit 89

A.1 Overview of power board components and connections 108
A.2 The PCB layout of the power board 112

Acronyms

ADC	Analog-to-Digital Converter. 24, 35, 43, 44, 52, 56, 63, 64, 71, 116, 117
AFSK	Audio Frequency-Shift Keying. 20
ALU	Arithmetic Logic Unit. 66
API	Application Programming Interface. 68
ASIC	Application Specific Integrated Circuit. 17, 20, 22, 28, 33, 35–38, 58
BGA	Ball Grid Array. 48, 115
CA	Coarse Acquisition. 59, 60, 77, 85, 86, 88, 89, 103, 106
CAD	Computer-Aided Design. 28, 30, 64
CMOS	Complementary Metal Oxide Semiconductor. 41, 95, 103, 117
COTS	Commercial Off-the-Shelf. 14–16, 25, 36, 41, 97
CRC	Cyclic Redundancy Check. 41
DET	Direct Energy Transfer. 110, 112
DMA	Direct Memory Access. 69, 70, 74

DSP Digital Signal Processing. 28

DSSS Direct-Sequence Spread Spectrum. 59

EEPROM Electrically Erasable Programmable Read-Only Memory. 55, 56

EPS Electrical Power Subsystem. 24, 110, 112, 113

FFT Fast Fourier Transform. 44, 60, 68, 69, 75, 83, 84, 107

FIFO First In, First Out. 71, 72

FLOPS FLoating-point Operations Per Second. 66

FPGA Field Programmable Gate Array. 1, 15, 17, 20–25, 27, 28, 30–42, 44–46, 48–57, 61, 62, 64, 67, 72, 76, 77, 79, 91, 95, 97–100, 102–108, 110, 115, 117

FPU Floating-Point Unit. 66

GCC GNU Compiler Collection. 64

GLONASS GLONASS. 20

GNSS Global Navigation Satellite System. 63

GPIO General Purpose Input/Output. 50, 56, 62, 63

GPS Global Positioning System. 1, 17, 25, 44, 58–63, 71, 72, 75, 77, 79, 80, 85, 89, 90, 102–104, 106–108

HDL Hardware Description Language. 25, 28, 30, 31

I In-phase. 62, 71, 85, 87, 103, 106

I^{2}C Inter-Integrated Circuit. 51, 56

I/O Input/Output. 36, 40, 50–52, 55, 96, 102, 106

IC Integrated Circuit. 1, 20, 24, 38, 48, 50, 56, 62, 63, 97, 101, 102, 110, 112, 113, 116

IP Intellectual Property. 28, 35, 36, 38, 48, 51, 69, 70, 75, 76, 90, 97, 99, 102, 104, 106, 107

ITAR International Traffic in Arms Regulations. 59

JTAG Joint Test Action Group. 35, 48, 55, 115

LDO Low Dropout. 55

LE Logic Element. 30, 31

LFSR Linear Feedback Shift Register. 59, 86

LUT Look-Up Table. 31, 41

LVDS Low Voltage Differential Signal. 96–99, 101

MIPS Million Instructions per Second. 22

MOSFET Metal Oxide-semiconductor Field-Effect Transistor. 56

MPPT Maximum Power Point Tracking. 55, 56, 108, 110, 112, 113

MTBF Mean Time Before Failure. 72

NCO Numerically Controlled Oscillator. 59, 80, 84–86, 89, 90, 106, 107

NRE Non-Recurring Engineering. 22

NRZ Non-Return to Zero. 79

NVM Non-Volatile Memory. 43

OBC On-board Computer. 15, 34, 46, 112

PC Personal Computer. 61

PCB Printed Circuit Board. 48, 54–56, 114, 115

PLL Phase-Locked Loop. 64, 98, 99

PRN Pseudo Random Number. 59, 75, 77, 86, 88, 89, 91

PWM Pulse Width Modulation. 24, 56

Q Quadrature. 62, 71, 85, 87, 103, 106

RAM Random Access Memory. 51

RF Radio Frequency. 23, 44, 58, 60, 62, 63, 69, 71, 75, 76, 102, 103, 106

RTL Register-Transfer Level. 30

SDK Software Development Kit. 104

SDR Software Defined Radio. 23, 25, 58, 61, 104

SDRAM Synchronous Dynamic Random-Access Memory. 62, 71

SEE Single-Event Effect. 41

SEL Single-Event Latchup. 41

SERDES Serializer-Deserializer. 97

SEU	Single-Event Upset. 32, 41
SMA	Sub-Miniature version A. 64
SoC	System on a Chip. 22, 38, 39, 44, 67–69, 74, 105, 107
SPI	Serial Peripheral Interface. 28, 44, 51, 53, 62–64, 75, 103, 106
SRAM	Static Random Access Memory. 22, 32, 33, 36, 37, 39, 41, 62, 98, 103, 108
TMR	Triple Modular Redundancy. 15, 22, 23, 41, 108
TTL	Transistor-Transistor Logic. 41
UART	Universal Asynchronous Receiver/Transmitter. 64, 96, 99, 115
USB	Universal Serial Bus. 61
USRP	Universal Software Radio Peripheral. 61
VHDL	VHSIC Hardware Description Language. 28
WDT	Watchdog Timer. 1, 50, 52–54, 102, 106

Chapter 1

Introduction

Advancements in technology have allowed for reduction in transistor sizes, allowing for smaller and exponentially more powerful electronics. Combined with the availability of affordable Commercial Off-the-Shelf (COTS) components has allowed for the development of small satellites such as nanosatellites [1], and even smaller satellites called CubeSats [2]. However, two problems arise: heat and the effects of radiation.

As the speed of a processor increase and the surface areas decrease, cooling and power present constraints on the processing hardware that can be used. As the processor clock speeds increase, so does the power consumption and heat generation. The satellites are in vacuum-like conditions, there is not much available to cool these processors, due to the lack of convection and mass allocation for heat sinks. The other problem is the effect radiation has on the components of the satellite subsystems. COTS components are used because of their availability, and effectiveness. They are designed as commercial-grade components, without shielding and protection from the harsh environment and radiation of space in mind.

A way to solve both of these problems is through the use of FPGAs. To deal with the issue of heat and power, one solution is make the processor more efficient, by making the processing hardware do more work at slower clock speeds. This can be done through the use of custom designed hardware which is optimized for the task. To deal with radiation, either radiation tolerant hardware is used, or mitigation techniques such as Triple Modular Redundancy (TMR) are employed. The use of FPGAs allows for such tasks to be done while still being available and affordable through the use of COTS FPGAs.

The objective of this research is to explore the feasibility of using FPGA devices on nanosatellites, with a focus on CubeSats. The challenges that this thesis address in designing for small satellites are the limited electrical power, limited size and limited computing power available. Therefore, investigating the feasibility of using FPGAs on nanosatellites, looking at the feasibility of using FPGAs for computation, results in improving upon the current processing capabilities given the power and size constraints of these small satellites.

1.1 CubeSat Architecture

CubeSats come in different sizes, e.g., 1U, 2U and 3U, based on a 1U CubeSat which is approximately 10 cm x 10 cm x 10 cm in size and 1 kg in mass [1]. The standardized CubeSat size allows for different CubeSats to be launched into space using a common launching mechanism [2].

Typical components of CubeSats are: a frame, an On-board Computer (OBC), a radio transceiver, an attitude control system, a power subsystem and a payload. Most of these subsystems are implemented on a standardized PC104 form factor board, which theoretically allows for modular designs and compatibility. As such, the various components can either be bought online, or be custom designed using COTS components.

The COTS components are not rated for space environments, i.e., extreme temperatures, and radiation. The COTS components are however, affordable and once they have been successfully flown, they gain flight heritage, which makes the COTS parts more suitable for use in future designs.

1.2 CubeSat Missions

CubeSats are typically designed by university teams for education purposes, allowing students to work on designing a satellite as a part of an interdisciplinary (mechanical, hardware, software, controls) team. They are popular because the cost to design and launch the satellite is relatively small compared to larger scale satellites [3]. CubeSats are normally launched by piggy backing on a primary payload, e.g. a much larger communications satellite sitting on a launch vehicle. This situation allows for CubeSat launches to be affordable by university teams. CubeSat satellite missions are designed to last as short as 3-9 months [4].

When launched, CubeSats are normally deployed in Low-Earth Orbits. Low-Earth Orbits allow for low-cost electronics, as satellites do not receive as much radiation or experience as much of a vacuum compared to satellites operating in a higher orbit. Communications systems are also more affordable, as the satellite is relatively close to the Earth, and therefore sending or receiving a signal requires less power.

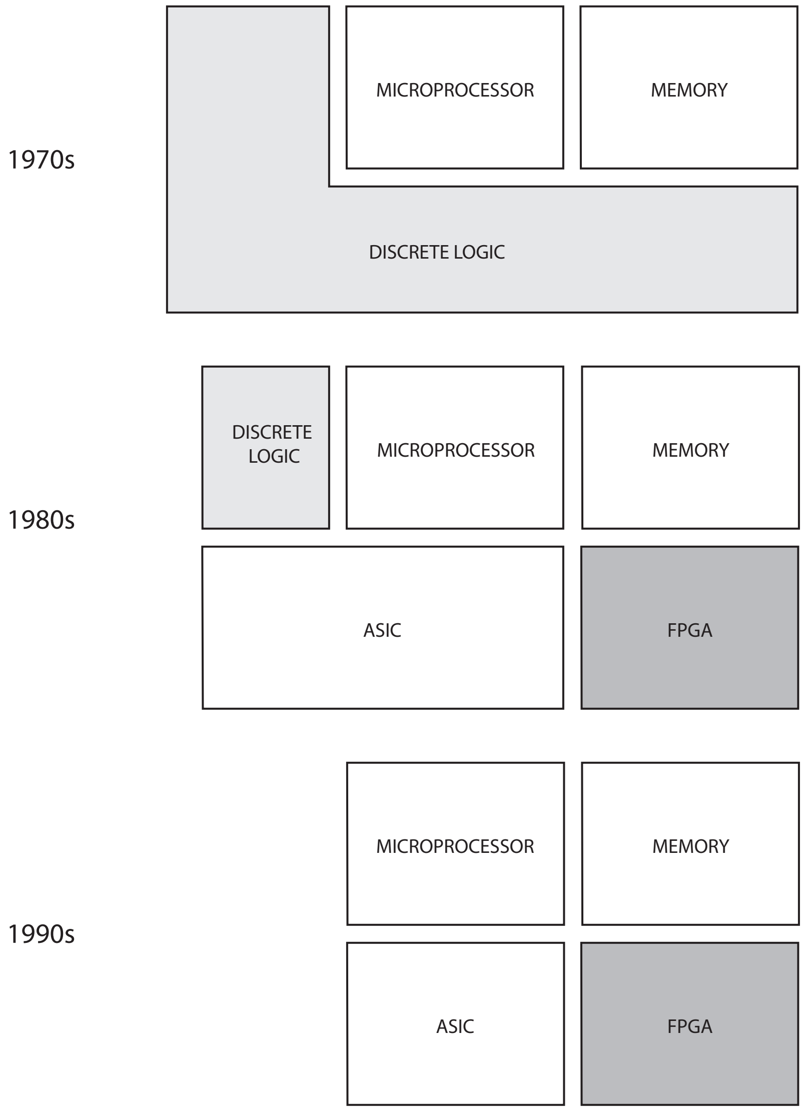
1.3 Satellite Data Processing

The processing work on a satellite can range from simple storing and relay to a ground station for later processing or processing the data onboard and streaming only the important bits of data later to a ground station.

The trade off is between the processing power available, and the amount of data bandwidth available for downlinking the data from the satellite. For the simple case of simpling collecting and storing the data on a satellite, greater bandwidth requirements are needed because the entire dataset needs to be transmitted back to Earth. With onboard processing, the data can be filtered, processed, and compressed, resulting in smaller amounts of data to be transferred.

1.4 FPGAs

With most hardware-based designs, the functionality is implemented on an Application Specific Integrated Circuit (ASIC) chip, since as the name implies, most of the required circuitry is contained on one chip. e.g., Global Positioning System (GPS) ASIC chips can be found in consumer devices and provide excellent functionality. However, if GPS receiver hardware were to be used in space, the design of the ASIC hardware chip needs to be changed (see Section 4.2 for more information) - which is not possible on an ASIC chip, as the internal circuitry is already hardwired for its intended purpose. Figure 1.1 gives an overview of the trend of designs using a smaller area and the incorporation of FPGAs as a way to replace discrete logic and ASICs. Beginning in the 1970s, custom hardware functionality was implemented using discrete logic gates. From the 1980s to the 1990s, ASICs replaced bulky discrete logic as the primary way hardware functionality was implemented. From the 2000s onwards, and as part of this thesis, FPGAs are capable of replacing ASICs and also by incorporating the processor onto the FPGA, replacing the need for a separate processor Integrated Circuit (IC).



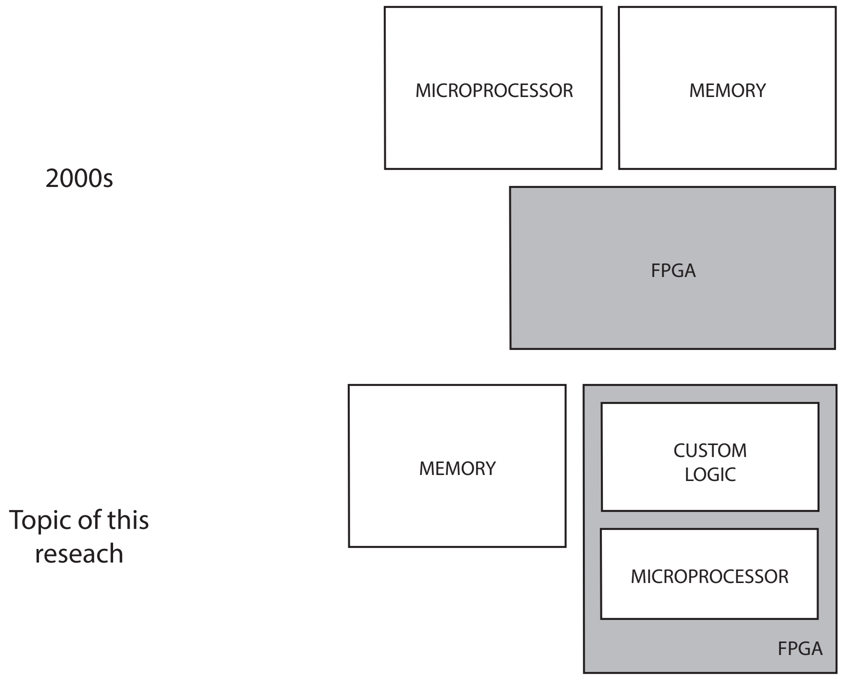


Figure 1.1: Comparison of hardware through the decades. Adapted from [5]

An FPGA is an electronic chip that allows for reconfigurable hardware elements to be programmed into the chip. For example, a GPS receiver design can be programmed onto an FPGA, and at a later date, if a function or parameter needs to be later changed, such as adding Global Navigation Satellite System (GLONASS) support, the firmware can be updated and the FPGA reprogrammed. FPGAs can be programmed multiple times, and similar to software, this flexibility allows for changes and updates to hardware to be implemented on the chip after assembly of the hardware. An FPGA can be incorporated into many subsystems of a CubeSat, as it can be programmed with different designs depending on the application.

Examples where an FPGA can be used on CubeSat subsystems include: A CubeSat transceiver board, the GOMSpace U480 [6], uses a microcontroller to process data, and an

Audio Frequency-Shift Keying (AFSK) modem chip [7] to modulate and demodulate the signals. By using a separate ASIC modem chip, changes to the radio modulation scheme would require a new board to be designed. If an FPGA were to be used, the end user would be able to use a different modulation scheme simply by uploading different firmware to the FPGA.

On a CubeSat power board, there are multiple DC-DC converter ICs to convert the available electricity on the satellite into the appropriate voltage levels that the various other subsystems require. The transistor control functionality of the individual DC-DC converters can be combined and implemented a single FPGA, allowing for one board design to be used on different CubeSats of different power requirements.

By successfully implementing an FPGA in space, the usability of the particular model of FPGA in space will be known, gaining flight heritage. This would allow for other designs to be implemented on a previously flown model of FPGA chips.

1.5 FPGAs in Space

FPGAs have already been used in space missions by NASA. Rovers sent to Mars: Spirit and Opportunity, Phoenix, and Curiosity, have some functionality controlled by an FPGA.

- The Mars Exploration Rovers, Spirit and Opportunity, use radiation-hardened FPGAs from Actel [8] and multiple radiation tolerant Xilinx FPGAs to control the motors and lander pyrotechnics [9].
- A subsystem of the Phoenix Mars rover, called the Meteorological Station uses an Actel RTAX-S, which is a radiation tolerant antifuse FPGA in its design [10].
- The Curiosity rover has a function the designers call a ”dream mode”, which allows

the rover to save power by having the main processing units be put to sleep while an FPGA with 1.2 million logic gates keeps the other components operating [11]

.

1.6 FPGAs in CubeSat Applications / Current Implementations

FPGA on CubeSats are normally used to implement microprocessors, utilizing a microprocessor soft-core on the FPGA. A soft-core processor is one in which the design of the entire microprocessor is described using a hardware description language, such as Verilog. This allows for anyone to simply add a microprocessor by adding it to their design, and having the design software synthesize it to fit on an FPGA.

Since the market for CubeSat components is relatively small, designing and manufacturing a custom ASIC unit would be expensive since there is not much demand for it in comparison to regular consumer products where the Non-Recurring Engineering (NRE) cost can be amortized over many units. With CubeSats, if someone needs a radiation-hardened microprocessor, they can simply obtain a radiation-hardened FPGA, add a soft-core processor, add TMR logic. Depending on the license of soft-core processor, the design can be customized specifically to the application by adding error checking, synchronizers, and TMR to various areas of the microprocessor to increase reliability of the design when operating in space.

1.6.1 Processing

An FPGA can be used for processing data in two ways: as a customized microprocessor running software, or as a hardware circuit running in parallel.

In the case of a microprocessor, NASA has a presentation on a implementation of a low-power processor for use in CubeSat missions [12]. The processor in this particular case is the LEON processor [13], which is based on the SPARC-V8 processor, implemented on an Actel ProAsic3 A3PE3000L FPGA. Implementing most of the design as a System on a Chip (SoC) allowed the developers to add SpaceWire interfaces and signal processing capabilities to their processor.

The processor runs at 20 MHz and performs at 17 Million Instructions per Second (MIPS), which is extremely limited in performance. To get better performance using an FPGA, the processing algorithm can be implemented in hardware.

FPGAs are typically used for real-time data processing, as they allow the designer to parallelize designs, since they can simply instantiate multiple copies of a module, and it run in parallel. Typically, due to better technology, Static Random Access Memory (SRAM)-based FPGAs are used over Flash-based ones. An example of an application using hardware-based processing is image processing while in space[14].

A general purpose processing board is the COSMIAC CubeSat FPGA Board [15], which is a combination of boards: a board with a Xilinx Spartan FPGA and Atmel AT90 microcontroller, a power board, and application specific daughter boards. The COSMIAC board implements Space Plug-and-Play Architecture (SPA) [16] to communicate with other components of the satellite.

Using an FPGA in a CubeSat design allows for better utilization of space, since it allows for customized solutions on a single chip. Implementing a radiation tolerant processor with TMR simply involves using a soft-core processor and TMR logic on one FPGA chip, instead of having a separate microcontroller and TMR.

1.6.2 Radio

One application of the COSMIAC board mentioned above is in a thesis titled "Modular FPGA-Based Software Defined Radio for CubeSats" [17]. The thesis discusses the design of a Radio Frequency (RF) daughter-board for the COSMIAC CubeSat FPGA board. The RF board is based on the Ettus Research USRP1 [18], but uses a Xilinx FPGA instead of one from Altera, which is what the USRP1 uses. By adapting the USRP1 board, their RF board is compatible with an open source software called GNU Radio [19]. GNU Radio contains different blocks, which can be used for a Software Defined Radio (SDR).

1.6.3 Power

An FPGA-based power board is used in CubeSat design from Montana State University, called Firebird. The CubeSat Electrical Power Subsystem (EPS) board used in their design is produced by a company called Tiger Innovation [20]. Their datasheet shows that the FPGA is used as a controller for the power system, taking analog measurements and sending out control signals to various components and communicating with the rest of the CubeSat through a serial interface.

In using an FPGA for a power board, the idea can be taken a step further by controlling the individual discrete transistors themselves instead of a DC-DC converter IC. In this case, the FPGA sends out a series of pulses through Pulse Width Modulation (PWM), and by varying the duty cycle, the output voltage is varied. An Analog-to-Digital Converter

(ADC), either built-into the FPGA or as a separate IC) can be used to sample the output voltage, providing feedback to the FPGA. By implementing the DC-DC converter design using an FPGA, we can have multiple converters o n one chip, and have separate converters for each solar panel, or even each solar cell, with each running at its own peak power point.

1.7 Research Motivation

With an FPGA, multiple components can be integrated and implemented on one chip, which saves space and power. The research motivation is to be able to design an FPGA solution allowing for the majority of a satellite subsystem to be contained on a single chip. The first step in the design of an all-in-one chip is to explore the possible subsystems that can be implemented on an FPGA and the performance of such systems. FPGAs allow for circuits to operate in parallel, which when compared to a processor, allows for more data to be processed per clock cycle. This means that an FPGA can perform similar amounts of work at a slower clock speed as compared to a microprocessor.

The outcome of this thesis is to develop several FPGA designs that can be used in CubeSat applications. The designs are written in a low-level Hardware Description Language (HDL) without the use of conversion tools, e.g., MATLAB HDL Coder or Synopsys Symphony C. The first design is an FPGA-based power management design that uses comparable power to a low-power microcontroller. It demonstrates the feasibility of using an FPGA on CubeSat hardware. The FPGA performs a simple task that is not mission critical, while not affecting the current system in a drastic way, i.e., low power draw, small size and low mass. Since the FPGA is a COTS component, the design of the physical device was not intended for use in the space environment, hence the device might prematurely fail. By incorporating the FPGA in the design as a secondary component, the impact of the FPGA design not working is minimized. However, once the FPGA has been successfully flown and the feasibility of using an FPGA demonstrated, the FPGA would then gain some flight heritage. Future designs can use an identical FPGA in other applications, using a component that has flight heritage but performs application specific tasks, simply by programming the chip with a different design.

Designs to be implemented on an FPGA are: a power board management module, an FPGA-based GPS satellite acquisition module that performs faster than comparable software running on a processor, and a camera interface to allow for buffering of data to a slower microprocessor.

1.8 Thesis Outline

This thesis is organized into six chapters. The first chapter is this introduction, followed by a brief look at FPGAs. Afterwards, the FPGA implementation on a power board will be looked at, which explores using an FPGA to increase the reliability of COTS microcontrollers; followed by the FPGA GPS Baseband Processor, which uses an FPGA to process raw GPS signals; and finally a FPGA Camera Interface, which interfaces a high-speed sensor with a low-speed processor. The last chapter is the conclusion which will summarize the research.

Chapter 2

Field Programmable Gate Arrays

Computers consist of hardware and software. Software exists as a sequence of instructions running on a processor, and hardware circuits control the flow of electrons through devices such as transistors. With digital hardware, the flow of electrons is either on or off, which represent binary values. These binary values can be manipulated using Boolean logic such as; AND, OR, and NOT. Their corresponding logic gates are then used as the building blocks of digital hardware, such as processors and other components inside a computer system. This section will give a brief introduction to FPGAs, the technology behind them, and how they compare to microprocessors and ASICs.

2.1 Introduction to FPGAs

FPGAs are a type of programmable logic device [21]; that is, they contain configurable logic blocks with programmable interconnections [22]. The programmable logic allows custom digital electronic circuits to be implemented on the device, allowing the circuit design within it to be customized for a specific application. It is analogous to a breadboard used for analog circuits. A circuit can be designed, called a netlist, which can then be synthesized and programmed onto the FPGA. These circuits can be simple as glue logic: simple logic functions to interface chips together, or as complex as an entire microprocessor.

Before an FPGA can be programmed, the design has to be described to a Computer-Aided Design (CAD) program, which takes it and translates it into bits to program the FPGA. The type of language used to describe the design is HDL. The two HDL languages commonly used are, Verilog and VHSIC Hardware Description Language (VHDL).

Similar to software, and software libraries, hardware descriptions can be reused and packaged in the form of Intellectual Property (IP) cores. These cores range from Digital Signal Processing (DSP) and arithmetic cores, to chip interfaces such as Serial Peripheral Interface (SPI), to processor cores such the 8051 [23] and OpenSPARC T1 processor [24].

FPGAs allows the custom design of a chip that can be similar in functionality to an ASIC, which would overwise cost millions to make [25], One simply has to take a design, let the software synthesize it, and then program it onto the FPGA. Synthesis is one of the steps done by the FPGA CAD software in implementing an FPGA design.

2.2 FPGA Work Flow

To give a describe about how a design is implemented on an FPGA, this section will provide a brief overview on the steps involved in creating a design from the HDL input to programming the FPGA. Figure 2.1 illustrates the typical design workflow. The design steps are as follows:

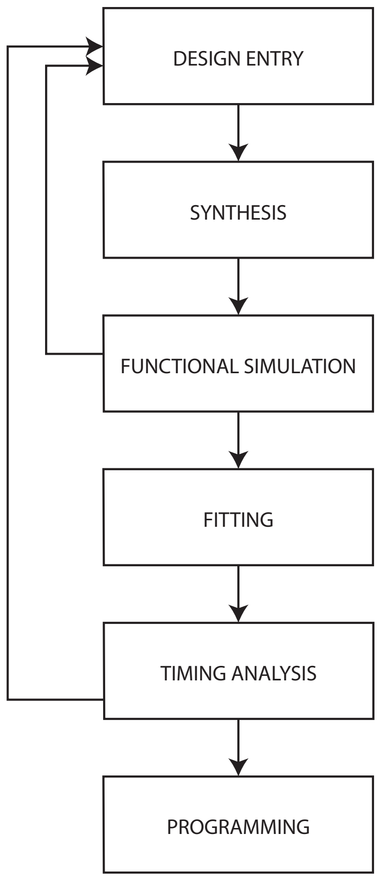


Figure 2.1: Typical design work flow. Adapted [26]

- Design Entry - The first step is to describe the hardware to be implemented using an HDL such as Verilog or VHDL. This process is similar to a software programmer writing code. Depending on the CAD tools used, the design can also be created visually by connecting blocks together in a schematic diagram.

- Synthesis - The next step is to let the CAD software interpret the HDL code and create an Register-Transfer Level (RTL) netlist. The lines of HDL code are translated into combinational Logic Elements (LEs), latches and registers. The designer can also verify the design by visually examining the netlist to see the CAD software interpretation of design described by the HDL code.
- Functional Simulation - The functionality of the design is verified through simulation and test-benches. These test-benches are written using an HDL and specify the input waveforms for the design, and the simulation produces the corresponding output waveforms. This task is performed by using another specialized CAD software, such as ModelSim [27].
- Fitting - The components of the RTL design are fitted into the LEs in the FPGA through placement and routing. The routing step determines the connections between the individual LEs and therefore the length of each wire.
- Timing Analysis - The length of each wire and the functionality of each LE determine the timing of the FPGA. The timing is important as it determines the fastest clock speed the design can handle. The timing simulation also helps to determine if there might be any glitches that might happen due to the timing of the various inputs and outputs changing state in the design.
- Programming - The final step in the CAD work flow is to generate a bitstream and program the FPGA. The bitstream contains the configuration of each routing switch and LE in an FPGA. The bits in the bitstream are then shifted into the FPGA, which programs the design onto the FPGA, implementing the HDL code in hardware.

2.3 FPGA Fabric

The FPGA fabric is the heart of the FPGA. It is the configurable layout of the chip, and is configured when a design is programmed onto the FPGA, determining the functionality of the design. The fabric is made up of thousands of programmable elements, which consists of LEs and routing switches.

Inside each LE is generally a Look-Up Table (LUT) and a register, although it may vary depending on the FPGA design. The lookup table allows the logic element to mimic different type of combinational logic. It is basically a demultiplexer with programmable inputs. The addition of the register allows any of the logic elements to be used in a sequential circuit, which is a circuit with a clock.

To connect the individual logic elements together and form a larger circuit, routing switches are used. They form a grid, and allow different pairs of wires to be connected. Since the switches are programmable, the routing and connection of logic elements vary based on the design programmed into the FPGA.

When an FPGA gets programmed, a stream of bits are shifted one by one into the FPGA. Each of these bits correspond to the configuration of a switch or logic element of the FPGA.

2.4 FPGA Technologies

The configuration of the logic elements and switches are stored directly in the logic elements / switches through the use of small memory elements. The three types of memory elements commonly found in FPGAs are SRAM, Flash (floating-gate) and antifuse [5].

SRAM cells are memory elements made from normal transistors, the basic SRAM cell is two NOT gates connected to each other in a loop, which can be made from 4 transistors, or 6 transistors when including access transistors. In order for SRAM memory to keep its state, power has to be constantly applied. Constant power usage is the cause of two of the disadvantages of SRAM FPGAs: high power consumption, and volatile memory; meaning the FPGA has to be reprogrammed if power is lost.

Flash memory operates by using a special transistor with a floating gate. The gate is isolated electrically, which makes Flash memory non-volatile, meaning a Flash FPGA can work immediately after power is applied, without having to be reprogrammed. It is also more tolerant of radiation affects such as Single-Event Upsets (SEUs) compared to SRAM FPGAs. The transistors in Flash FPGAs are larger than those of SRAM FPGAs, which result in Flash-based FPGAs having a slower maximum clock speeds, and contain less logic elements compared to SRAM FPGAs. Another disadvantage of Flash-based FPGAs is that their technology normal lags behind those of SRAM-based FPGAs. As of this time, an SRAM-based FPGA from Altera uses 28 nm technology [28], allowing for faster transistors, while a Flash-based FPGA from Actel is still based on a 130 nm process [29].

Antifuse are devices that work in the opposite manner of a fuse - they are normally non-conducting, but when enough electrical energy is applied, a permanent conducting path is formed. Due to the way antifuses work, antifuse-based FPGAs can only be programmed once. Since the connection is permanent, antifuse FPGAs are the least susceptible to radiation out of the different FPGA technologies.

These different ways to store the configuration of the FPGA also determine the application the FPGA is used in. With an SRAM-based FPGA, there is a trade off in performance, and the fact that the configuration data of the FPGA is lost when it is turned off, and has to be reprogrammed again on start up. Hence, SRAM FPGAs are better suited for non-mission critical tasks, or tasks where computational speed is importation, such as on scientific instrument payloads. With a Flash-based FPGA, its configuration data are non-volatile due to the Flash memory, so it is able to start up without reprogramming. This makes Flash-based FPGAs ideal in a supporting role to setup or assist other components, or as part of the satellite infrastructure. The last type are the antifuse types of FPGAs, that are configured by blowing the antifuses, permanently programming the design into the FPGA. Antifuse devices are ideal for mission critical components such as processors or attitude control.

2.5 Comparison to Other Technologies

FPGAs are not alone in allowing for implementations of custom designs on a single chip. Processors allow for the implementation of custom functionality through sets of instructions called software. However, the difference is that microprocessors work by interpreting instructions sequentially, while FPGAs have interconnected logic elements, which allow for independent electrical circuits to operate in parallel. ASICs are the hardwired version of FPGAs. Instead of logic elements, the entire ASIC chip consists of permanently connected transistors.

2.5.1 Software versus Hardware

A processor works by executing a series of instructions, which is the software running on the processor. A few steps are required in this process: the processor must fetch the instruction from memory; the instruction must be decoded to make sense; and the processor executes the instruction. For a more detailed explanation, refer to Section 4.8.1. Each one of these steps normally takes one clock cycle to complete on a simple processor.

Clock speeds found on OBCs used for satellites are typically low. For example, the Earth Observing 1 (EO-1) spacecraft from NASA uses a Mongoose-V processor [30]. The Mongoose-V is a 42,225 USD processor that operates at a maximum of 15 MHz [31]. The slow clock speed is because fast clock speeds use more power, switching on and off the transistors, charging and discharging their gates more often. To keep the clock frequencies low, more work should be done per cycle, meaning more instructions per cycle. An FPGA can operate on multiple tasks per clock cycle by design. A processor can accomplish more per clock cycle by pipelining or through multiple processors, or using multiple cores.

On an FPGA, the sequence of steps is normally controlled by a state machine. However, each state is hardwired to perform a specific task. It therefore does not need to have separate steps to fetch and decode each step in the state machine.

To execute multiple tasks in parallel, either multiple processors are required, or an operating system is used, and a process called time slicing splits the tasks into smaller tasks so that it appears the processor is working on multiple tasks simultaneously. This extra layer of software adds additional overhead to the already big disadvantage of operating sequentially.

2.6 Microprocessors, FPGAs and ASICs

Another type of integrated circuit is the ASICs, which is a dedicated chip designed to perform a specific set of tasks. The advantages of ASICs over microprocessors are they are faster and much more efficient. Instead of fetching and following software instructions to complete a task, ASICs have predefined circuitry to complete the task. FPGAs provide a low-cost alternative to dedicated ASIC chips. A design can be customized for a specific purpose and then programmed on to the FPGA, unlike an ASIC where the design is "hardware" and cannot be changed.

Progressing from microcontroller to FPGA to ASIC, each gets more and more application specific, and therefore more efficient. A microcontroller can have many peripherals, such as an ADC, and voltage comparators, allowing them to interact with other analog devices. These peripherals and the flexibility of software, allow one kind of microcontroller to be used in many different types of applications. An FPGA on the other hand, consists of simply logic elements, which can be combined to implement the peripherals previously mentioned, but only interact with mostly other digital electronics. With an ASIC, the logic elements of an FPGA are replaced with actual logic gates made of transistors, which are sometimes manually laid out. They represent the most optimized design and can be made to combine with analog electronics.

2.6.1 Cost

Implementing a design on an FPGA typically costs more than on a microcontroller, due to design hardware costs, design software licensing and IP licensing. Hardware is needed to program and debug the FPGA, normally a Joint Test Action Group (JTAG) programmer and logic analyzer. A free version of the design software is normally available from the FPGA vendor, but additional functionality such as implementing an soft core processor, designing for multiple FPGAs or more advanced verification tools need costly design software licenses. Another cost in FPGAs design is the licensing of various IPs used in the final design - items such as processors, memory controllers, and interface implementations.

These items can be found in COTS microcontrollers, but to implement them on an FPGA requires licensing the corresponding IPs.

Creating an ASIC design costs much more, in terms of millions of dollars and years of development[25]. The process for designing an ASIC chip includes the steps for creating an FPGA design, along with optionally laying out the individual transistors, fabricating the silicon wafers at a foundry and finally packaging the silicon into a piece of plastic with legs and leads. It is possible to produce an ASIC design on a more affordable level through companies which take different designs and create them on one wafer, and also through academic pricing [32] that ranges from 350 CDN for a 0.8 µm process, up to 18500 CDN for the 28 nm process, where prices are per mm^2.

2.6.2 Power consumption

To compare the difference in power consumption between microcontrollers and FPGAs is difficult because such a comparison depends on many variables in the design. On an FPGA, dynamic power is dependent on the Input/Output (I/O) pins, clock speeds, and what the system is doing, i.e., which component of the microcontroller or FPGA is active. Comparing static power, when the devices are idle is simple since the datasheets do specify the average static power used by each device. Flash-based FPGAs and small microcontrollers use power in the range of micro-watts, while SRAM-based FPGAs are orders of magnitude more power hungry, consuming milli-watts of power [33].

FPGAs can offer power savings since they are then typically able work at clock frequencies lower than those of microprocessors in order to perform the same task (see Section 4.8.1 for an explanation). Lower clock speeds mean lower power consumption since the transistors are switching less often. By operating at a lower frequency, less heat will also be given off. In comparison with ASICs, SRAM-based FPGAs on average consume 14 times more dynamic power compared to an equivalent ASIC design, along with being 119 times bigger in area [34].

Equation 2.1, relates the dynamic power, P to switching frequency, f, which in this case is the clock rate of the system. The other parameters are capacitance C and operating voltage V.

$$P = CV^2 f \tag{2.1}$$

2.6.3 Security

An advantage a FPGA-based design has over an ASIC is the security of the design. With an ASIC, the design has to be sent to the foundry in order to have the silicon wafer created. While at the foundry, the design can be secretly modified or simply stolen. With an FPGA, since the devices are field programmable, the design can be programmed in-house after the FPGA is purchased. This means that the design going onto the FPGA never has to be given to someone else [35].

2.6.4 Summary

Table 2.1 summarizes the comparison between microcontrollers, FPGAs and ASICs.

	Processor	SRAM FPGA	Flash FPGA	ASIC
Speed (parallel)	Low	High	Medium	**Highest**
Cost for custom design	**Lowest**	Medium	Medium	High
Development time	**Low**	Medium	Medium	High
Power usage	Medium	Medium	**Low**	Lowest

Table 2.1: Technology Comparison

2.7 System on a Chip

An application of an FPGA is a SoC: a combination of microprocessor and FPGA technology. It consists of a microprocessor implemented on an FPGA chip, either as a hard processor, or a soft processor implemented on the FPGA fabric out of logic elements. A system on a chip contains most of the circuitry that a microcontroller or microprocessor may have, such as processing cores, serial interfaces, and memory controllers. Internally, the components are connected by a bus such as the AMBA bus for ARM processors, PLB for Xilinx, or Avalon for Altera devices.

These SoC devices cost more than normal FPGAs, since the processor designs are normally licensed IP, and hence cost more due to licensing. They also need larger sized FPGAs in order to fit in the processor onto the FPGA chip. An SoC can be fit onto an FPGA if it has enough logic units, or a specific SoC FPGA can be purchased. These FPGAs normally come with embedded hard processors. Hard processors are common hardwired processors, incorporated into the same IC package as the FPGA, meaning that the design of the processor cannot be changed. But this allows for a high speed processor to be connected with custom FPGA designs on a single chip.

Examples of SoC FPGAs from different manufacturers are:

- Altera Arria 10 / Cyclone V SE/SX/ST - Dual-core ARM Cortex-A9
- Actel Fusion - ARM Cortex-M3 (soft-core processor)
- Xilinx Zynq-7000 - Dual-core ARM Cortex-A9 - uses Artix-7 / Kintex-7 fabric

SoC designs can also be implemented on normal FPGAs. Implementing a soft-core processor in an FPGA design, such as an SoC implementation, requires a large amount of FPGA resources such as FPGA logic for the processor core itself, along with supporting components. The majority of the space used is memory to store the program and act as run-time memory. As an example, if a mid-range SRAM-based FPGA were used as a memory device, one could consider the Altera Cyclone IV with the following configuration: 8,192 memory bits per block [36]; 3,888 kbits [37]; resulting in 486 kB of RAM.

2.8 Performance

Not all FPGAs are created the same; some are designed to have less logic units, or some underperform due to manufacturing defects. One factor affecting performance is the number of logic elements an FPGA contains. Each FPGA manufacturer produces different FPGAs with varying amounts of logic elements, since those with fewer logic elements are cheaper as the manufacturing yield is higher (more FPGA dies can be placed onto a single silicon wafer). The number of logic elements are not directly comparable between different manufacturers, because of the different designs and technologies involved along with different metrics each manufacturer uses to advertise the performance of their FPGAs.

2.8.1 Clock Speeds

Not all FPGAs from a given batch are the same. When FPGAs are manufactured, they go through a process called binning, where the FPGAs are categorized based on their speed. The FPGAs are given a speed rating, and the better performing chips are priced higher. The rating of the FPGA determines the maximum clock frequency an FPGA can handle and puts a constraint on the design that can be programmed onto the FPGA. The maximum clock speed of a circuit design for an FPGA depends on the timing delays in the circuit. The minimum clock period has to be greater than the timing delays; hence to maximize clock speeds, timing delays need to be minimized.

The two factors that affect the timing of an FPGA are propagation delays, and clock skew. Propagation delays come from the combinational logic that might exist between two registers. If the propagation time is greater than the clock period, then changes at one register might not show up at the next register at the next clock cycle. Clock skew is the delay between the clock signal arriving at one register, and the next. Clock skew is an issue when it is greater than the propagation delay. FPGAs have dedicated clock routing paths to minimize this clock skew.

2.8.2 Operating Temperature

The operating temperature is one parameter that defines the environment an FPGA can be expected to function in. Each model of FPGA is specified an operating temperaturing range by the manufacturer. The operating temperating range of each type of FPGAs is verified after production of the chips to ensure it meets specifications. The temperature ranges based on the intended application: commercial, automotive, or military. Various components of an FPGA might fail at extreme temperatures, such as the programming charge pumps, I/O pins, logic elements, and registers.

2.9 Radiation Concerns

The majority of COTS microcontrollers and FPGAs used are Complementary Metal Oxide Semiconductor (CMOS)-based. Being CMOS-based, they are susceptible to Single-Event Latchups (SELs) and SEUs. An SEU event can cause bits to change, but this memory error can be remedied through error checks and correction. An SEL event causes a short to ground, which can result in permanent damage, and can only be fixed by power cycling the device. These Single-Event Effects (SEEs) are inherent in CMOS devices, and our alternatives are to either use radiation hardened CMOS devices, or use discrete Transistor-Transistor Logic (TTL) logic gates. However, due to cost, board space, and power constraints, COTS CMOS devices are typically used. The two ways to deal with radiation is to either use a radiation tolerant FPGA, or create a design that incorporates SEU mitigation techniques, such as Cyclic Redundancy Check (CRC) or TMR [38] [5].

With a typical SRAM based FPGA, its design (the individual interconnections and LUTs) are stored as bits in SRAM memory. With Flash-based FPGAs, these bits are stored in Flash-based memory, which is less susceptible to SEU effects. One area where SEU effects are of a major concern is the configuration SRAM on SRAM-based FPGAs [39]. The alternative would be to use Flash-based FPGAs instead, since their configuration switches are "intrinsically hard" [40]. Since Flash-based FPGAs are slower, it is typical for SRAM devices to be used in payload applications, where failure can be tolerated and Flash-based or antifuse FPGAs are used in more critical bus systems [38].

2.10 Review of FPGAs

Of the different FPGA manufacturers, only Actel and Xilinx offer space-grade radiation-tolerant devices. Table 2.2 gives a brief overview of the different FPGA series from three different FPGA manufacturers and prices quoted from Digikey.ca [41], in Canadian dollars for January 2014. The units are sorted from low-cost to high-end. A couple of points to note are, the wide price ranges which are dependent on the FPGA die size and therefore yield, and that the second largest FPGA company, Altera, does not have any space qualified FPGAs.

Vendor	Series	Type	Price	Notes
Actel	IGLOO/ProASIC3	Flash	\$3-\$1,668	Radiation tolerant version - RT ProASIC3, 3 million system gates, 350 MHz, 75264 logic tiles. Support for extreme temperatures - IGLOO (1st gen) 185 °C
	Fusion	Flash	\$58-\$1,013	User accessible Non-Volatile Memory (NVM) memory and ADC. Support for extreme temperatures - Fusion (2nd gen) 185 °C, SmartFusion (3rd gen) 200 °C
	AX/SX	Antifuse	\$11-\$14,257	Radiation tolerant version - RTAX, 4 million equivalent system gates, 40320 combinational logic cells
Altera	Cyclone	SRAM	\$12-\$4,782	
	Arria	SRAM	\$203-\$27,242	
	Stratix	SRAM	\$227-\$29,067	Tested and qualified to operate under the military rated temperature ranges (-55 °C to 125 °C) Does not offer a line of space specific, radiation hardened products
Xilinx	Spartan	SRAM	\$7-\$380	
	Artix	SRAM	\$124-\$447	
	Kintex	SRAM	\$152-\$6,645	
	Virtex	SRAM	\$67-\$47,251	Radiation tolerant version - Virtex 4QV/5QV, 130000 logic cells, 450MHz, -55 °C to 125 °C

Table 2.2: FPGA Comparison

2.11 FPGA Test Cases

To demonstrate the feasibility of using FPGAs on nanosatellites, the following are the three test cases that can potentially be used on a CubeSat satellite.

2.11.1 Power Board

First, the feasibility of an FPGA is examined for power management on a nanosatellite power subsystem. The FPGA works alongside the microcontroller, using the ADC of the microcontroller and communicating through an SPI interface. The FPGA is also intended to function independently of the microcontroller in case problems are detected in the FPGA or microcontroller during operations.

2.11.2 GPS Signal Processing

Second, an FPGA is used to implement a basic GPS receiver, interfacing with the RF frontend and providing a computing platform on which to run the GPS signal processing algorithm, along with performing basic calculations to acquire basic signal measurements of satellites in view. The second part is more demanding on the FPGA - implementing a SoC design, along with hardware-based calculations, and to demonstrate the feasibility of using FPGAs by offloading tasks onto the hardware. Such tasks include Fast Fourier Transform (FFT), data acquisition, and satellite signal acquisition.

2.11.3 Camera Interface

Third, is using the FPGA to interface between a camera sensor and microcontroller. The FPGA reads the camera signal, buffers it, and outputs the data lower rate to the microcontroller. This study represents the application of FPGAs in supporting the development of other components of a CubeSat platform.

Chapter 3

FPGA Watchdog Timer for Electrical Power Subsystem

The electrical power subsystem on a satellite powers the rest of the subsystems on the satellite, without power the satellite would not function. CubeSats use COTS components in their design, and the power board is no different. Combined with a COTS microcontroller managing the board, any hardware glitch or software bug in the microcontroller firmware would render the satellite inoperable. This section discusses use an FPGA as a Watchdog Timer (WDT) inorder to reset the microcontroller when needed and prevent glitches in the power system while the microcontroller resets.

3.1 Power Board Design

The electrical subsystem power board was designed for use in a 1U CubeSat, which is by definition, limited in space and mass. For the details on the design of the power board, please refer to Section A. The complete power board is shown in Figure 3.1.

An important component of the power board is the microcontroller, which acts as the brain controlling the other components of the power board, along as acting as an interface, allowing for communication with the OBC subsystem. The FPGA is not a critical component of the power board, and therefore does not have major a impact on the power board in terms of space usage, power consumption, or mass.

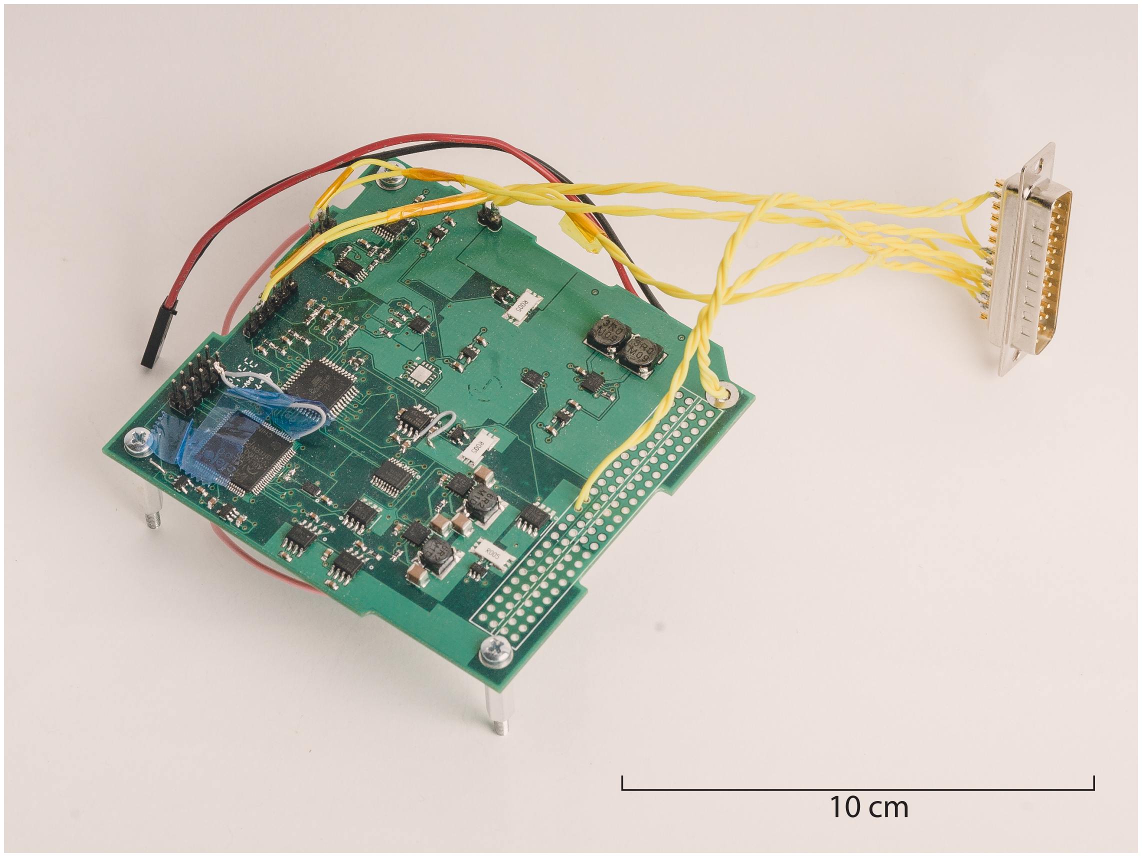


Figure 3.1: Completed prototype power board that was used for simulated space-environment testing

3.2 FPGA

The goal of the FPGA is to have a Flash-based device which is tolerant of radiation upsets, to monitor the microcontroller on the power board. In the event of a radiation upset or software error, the FPGA detects the fault and resets the microcontroller. The microcontroller is used to control the various ICs on the power board, resetting the microcontroller leaves the output pins of control signals in a undefined state. The FPGA is used to sample and hold the values of the control signals as the microcontroller is being reset.

Compared to an SRAM-based FPGA, the Flash-based FPGA used on the power board allows the power board to power on immediately, whereas the SRAM-based FPGA would have need additional time, and circuitry to program the FPGA fabric on start up.

The FPGA was chosen for its low-power consumption, relatively small size and ease of assembly. The FPGA device comes in a variety of different IC packages. A surface-mount based VQ100 package with pins instead of Ball Grid Array (BGA) pads was used. BGA connections are more dense which requires more layers on the Printed Circuit Board (PCB) in order to route all the signals.

The downside to using a Flash-based FPGA is the fewer logic elements available, which limits the possible designs. The FPGA was not able to fit a standard 8051 soft-core processor. In its place, a proprietary IP design called CoreABC was used. The IP has limited features and only allows for basic programs that are written in assembly. The CoreABC IP also does not have the JTAG debug functionality found on the Core8051 IP, which can be used for debugging the software running on the processor.

In terms of software CAD tools, the Actel/Microsemi tools lack debugging capabilities, such as the logic analyzer functionality found in the software suite used by Altera FPGA.

The lack of debugging tools make troubleshooting the hardware design much more time consuming.

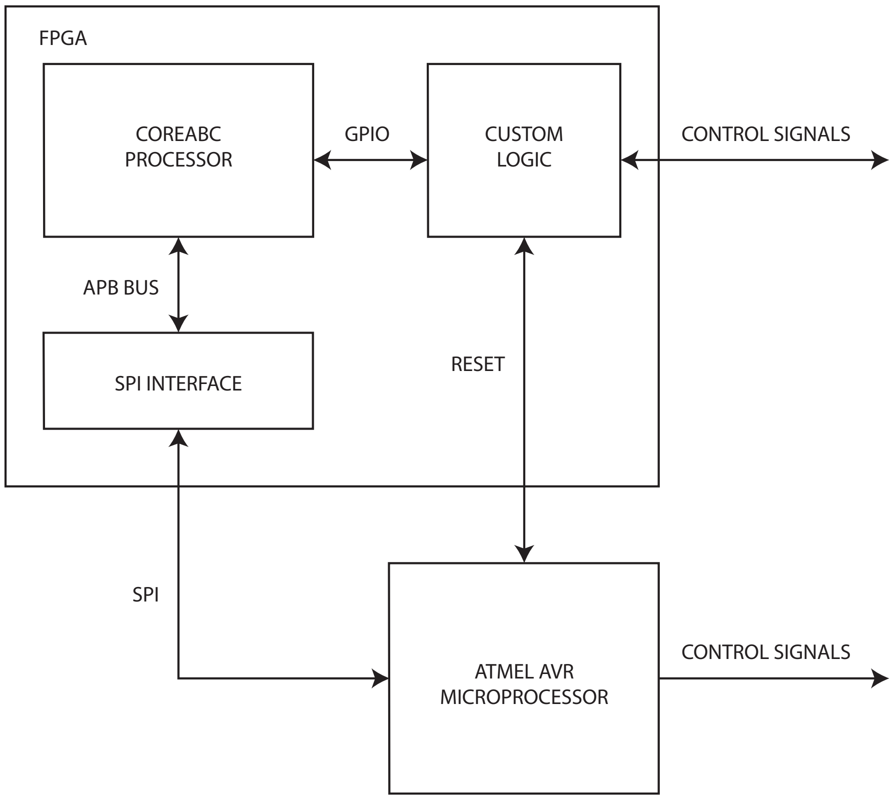


Figure 3.2: Overview of FPGA and microcontroller connections

3.3 Watchdog Timer

The design of the power board is a stepping stone in using FPGAs in CubeSats, therefore the role of the FPGA was simplified. Instead of using the FPGA as a secondary controller along side the microcontroller, the FPGA acts as a WDT with an additional functionality. A watchdog timer monitors the status of the microcontroller, and sends a reset signal in case the microcontroller stops responding. With a traditional WDT, such as the WDT built into the microcontroller, the resetting of the microcontroller would cause the General Purpose Input/Output (GPIO) pins controlling the other ICs to change state, enabling or disabling other components unintentionally. The FPGA-based design solves this problem by monitoring the state of the GPIO pins during normal operation, and during reset, holding the values until the microcontroller finishes booting.

The method that the FPGA uses to monitor the microcontroller is by using a counter that automatically increments itself. Once the timer timeouts after reaching a certain value, it resets the associated circuit. During normal operation, the microcontroller prevents itself from being reset by the watchdog timer by periodically resetting the value stored within the timer.

In the event of a malfunction with the microcontroller, the counter reset signal will stop being received, eventually causing the counter to overflow and the FPGA WDT resetting the microcontroller. During the reset, the control pins of the other circuits would not be driven by the microcontroller, hence the FPGA has to output what the microcontroller would have outputted.

3.4 Pin Monitoring

The I/O pin monitoring module is a custom hardware module, and works by simply sampling and storing the state of each pin during each positive clock edge transition of the system clock into a register. When the WDT detects a fault with the microprocessor, before resetting the processor, the WDT signals the pin monitoring module. When the pin monitoring module receives the signal, it takes out the I/O pins originally controlled by the microcontroller, and outputs the stored, preventing the I/O lines from toggling when the microcontroller resets. The state of the power board is normally stored in the Random

Access Memory (RAM) of the microcontroller, with the state of the I/O pins also stored in the registers of the FPGA. Storing the values into a register also means that when the FPGA and microcontroller both lose power, all information about the state of the power board is lost.

3.5 Serial Peripheral Interface Bus Interface

The SPI is essentially a shift register, where bits that are sent by the master device push out bits contained in the transmit register of the slave device. It allows 8-bits to be sent per transaction between the microcontroller and FPGA. SPI is a master-slave interface, meaning that only the master can initiate a transaction. The microcontroller acts as a master, and the FPGA acts as the slave. The FPGA only knows the status of the microcontroller through the periodic messages the microcontroller sends. When the microcontroller can no longer send a message to the FPGA, the FPGA assumes the microcontroller is stuck and resets the microcontroller.

On the microcontroller, the SPI interface is built into the Atmel AVR device. On the FPGA side, the SPI interface is instantiated by using the CoreSPI IP core from Actel. The microcontroller and FPGA are connected by four wires/traces on the board, which allow for 3 different methods of communication between the two chips: Inter-Integrated Circuit (I^2C), SPI, or using the 4 wires to signal 4 different messages. SPI was used because it offers a trade off in being relatively simple to implement compared to I^2C, and can be used to communicate more data compared to using each wire as a separate signal such as voltage and current data available through the ADC built into the microcontroller.

3.6 Resource Usage

Table 3.6 lists the amount of logic elements and I/O pins used on the FPGA in order to implement the WDT design. The design consists of the custom hardware logic, SPI interface, and CoreABC soft-processor core.

	Used	Total available	Percentage used (%)
Logic Elements	659	6144	10.73
I/O pins	6	68	8.82

Table 3.1: Amount of FPGA resources used to implement the Watchdog Timer design with a CoreABC processor

The particular model of FPGA used, the AGLN250, has 36 kbits of RAM available, which is equivalent to 4kB of RAM. Table 3.6 shows the amount of memory required on the microcontroller inorder to run the firmware used in the power board design. In total, approximately 35k kB of memory is needed on the microcontroller. If the FPGA were to replace the microcontroller, external memory and a processor core supporting external memory would be needed.

Memory Type	Size (bytes)	%
Program	26200	20.0
Data	9563	58.4

Table 3.2: Amount of microcontroller memory used to run the FreeRTOS firmware

3.7 Testing

To verify the functionality of the design, and to verify the hardware works in space, functional and simulated space-environment tests were completed. The tests were completed using the FPGA on the prototype power board.

3.7.1 Verification

Functionality of the FPGA WDT design was verified by testing on the completed power board. Before testing, the microcontroller and FPGA were programmed with the appropriate designs and the board was connected to a power supply, electronic load and oscilloscope.

To control the WDT, software instructions were written for the microcontroller which both enables and resets the counter of the WDT. The reset signal of the microcontroller was monitored on the oscilloscope and the microcontroller was connected to text based terminal. The WDT design was then tested by connecting the power board to a power supply and carrying out the following tasks.

1. After turning on the power board, a welcome message was observed on the text terminal.
2. The power board was left to run for 5 minutes, during which nothing happened - the power board did not reset.
3. The command was given to the microcontroller to send the command over the SPI interface to enable and reset the WDT.
4. Leaving the power board untouched for 5 minutes, the power board continued to respond to inputs and no fluctuations were observed on the reset line of the microcontroller.
5. The command was given to the microcontroller to send a command which stopped

6. Leaving the power board untouched for 10 seconds, the power board was observed to have resetted by examining the oscilloscope and observing the welcome message being displayed on the text terminal.

Functionality of the WDT was verified by monitoring the reset line of the microcontroller upon power up. The first few instructions of the FPGA soft-core processor is to start the watchdog timer, which counts down, times out, and resets the microcontroller. After the watchdog timer times out and resets the microcontroller, it goes back into the idle state. When monitoring the reset line, it should start at 3.3 V, followed by a negative pulse - the line should drop to 0 V for 1s and go back to 3.3 V. The microcontroller uses an active low reset signal, so the negative pulse coming from the watchdog timer resets the microcontroller.

Test of the functionality of the FPGA in maintaining continuity of the power board was done allowing the WDT to reset the microcontroller, and observing the control pins of the various ICs on the board, and monitoring the output voltages and confirming there was no change (that would otherwise be caused by the power regulators/chargers turning off).

The power board was tested in a simulated space environment by using a vacuum chamber. The power board survived the simulated space environment. The development of the FPGA design and microcontroller firmware was done after the test using the tested power board, and no electrical nor physical damage was encountered.

3.7.2 Feasibility

The design as it is, is not feasible to implement on an FPGA. Since the time required and extra steps of laying out the PCB, added separate voltage regulators and oscillator for the

FPGA. The microcontroller has a built in WDT and the resetting of the microcontroller should not happen enough, nor does the boot process of the microcontroller take long enough to warrant the addition of the FPGA.

3.7.3 Components

In order to implement this design, a number of components are needed on the PCB to support the FPGA. These components include:

- 1.2 V Low Dropout (LDO) voltage regulator for the FPGA core.
- 3.3 V LDO voltage regulator for the FPGA I/O pins.
- An oscillator to clock the sequential circuits on the FPGA design.

The programming interface of the FPGA uses a JTAG connection, which is shared with the microcontroller. The JTAG interface allows for one physical connection by daisy chaining the signals between the different devices - the microcontroller and the FPGA.

3.7.4 Future Improvements

The current design is intended as a stepping stone to incorporating FPGAs into traditional CubeSat designs, and therefore, numerous additional improvements can be made. Currently, the microcontroller is the main controller for the board, switching on and off various components, and controlling the amount of power supplied from the board (from the solar panels to charge the batteries and capacitors). Currently, the charge currents are being controlled (for Maximum Power Point Tracking (MPPT)) through the use of digital potentiometers (which store their values in Electrically Erasable Programmable Read-Only Memory (EEPROM) memory). A future design using entirely an FPGA controller would control the current through the use of PWM, with the parameters and values stored in the Flash memory of the FPGA.

A series of FPGAs from Actel called Fusion, have the ability for the user to access the Flash memory of the FPGA. The user accessible Flash memory allows for the FPGA itself to be used as a piece of Flash memory. In comparison, when using a digital potentiometer, the value of the potentiometer is stored in EEPROM memory, which is more susceptible to radiation damage. Using the Fusion FPGA, and implementing the MPPT functionality on the Flash-based FPGA, would allow for instant start-up (and not have to read/write via I^2C the potentiometer value), and to safely store the previous MPPT value on the Flash memory of the FPGA.

The microcontroller has a limited number of GPIO pins - therefore the status and control pins coming from each IC device cannot all be connected to the microcontroller. The FPGA has more pins available, but connecting all of the pins would increase the amount of routing and layers on the PCB. The pins for this particular FPGA are digital (the Actel Fusion FPGAs have some analog pins) so they cannot measure analog voltages.

The FPGA can be made to control discrete Metal Oxide-semiconductor Field-Effect Transistor (MOSFET) transistors. With the addition of an ADC, the FPGA can implement the functionality of a buck-boost converter, battery charger, or super-capacitor charger. The functionality of the various power ICs can be combined into a monolithic chip, saving board space and time routing traces.

Using an FPGA would allow for the design to be adapted to different satellites. The additional independent MPPT controllers can be added for each solar panels simply by instantiating addition MPPT modules within the FPGA design. The design can be expanded to handle more batteries or a higher load by adding additional transistors in parallel along with their corresponding controlling modelling on the FPGA.

Chapter 4

FPGA GPS Baseband Processor

GPS receivers are used on satellites as payloads which can used to determine the position of the satellite in space, or as a scientific instrument. As a scientific instrument, a process called radio occultation can determine properties about the atmosphere as the GPS signals passes through. Using a SDR, the circuit can be changed as newer Global Navigation Satellite System (GNSS) technologies appear.

4.1 Software Defined Radio

An SDR receiver is designed to process data coming from an RF frontend using software instead of hardwired hardware such as ASICs. The reasoning behind using software is that it allows the receiver to be modified or reconfigured to include functionality that was not anticipated when the receiver was originally designed [42].

In the SDR concept, the design consists of three parts: a baseband processor, a data acquisition module, and a processing module. The baseband processor runs the satellite signal detection algorithm in software, and offloads some of the processing onto hardware. The data acquisition module is responsible for receiving data from an RF frontend and storing the data in memory. Finally, the processing module performs satellite acquisition using the previously captured data in order to produce information about the GPS satellite data contained in the signal.

4.2 Why FPGAs for SDR

The FPGA is used as a baseband processor, which processes the digitized signals from the frontend, and attempts to recover useful data such as pseudorange measurements from “visible” satellite signals to send to the main processor. Commercially available civilian GPS processors and receivers are limited in functionality due to International Traffic in Arms Regulations (ITAR) regulations. ITAR regulations for GPS receivers exported from the United States state that the GPS receiver should not be ”designed for producing navigation results above 60,000 feet altitude and at 1,000 knots velocity or greater” [43]. One way to avoid the ITAR regulations and paperwork is to implement a software defined radio version of a GPS receiver.

4.3 Background on GPS Hardware

A GPS receiver captures and decodes signals sent by the numerous GPS satellites that are within view of the receiver at any given time. The GPS satellites transmit their data using a common frequency but use Direct-Sequence Spread Spectrum (DSSS) in order to differentiate the signals coming from the individual satellites. Each GPS satellite uses a Pseudo Random Number (PRN) generator, a type of Linear Feedback Shift Register

(LFSR), but with different taps in order to produce a different sequence of bits - the Coarse Acquisition (CA)-code. The resulting data and CA-codes are modulated on a carrier signal before being transmitted from the satellite. The received carrier signal has a Doppler shift due to the relative motion between the satellite and GPS receiver.

To implement a GPS receiver, multiple channels are used, each consisting of a PRN generator to generate replica CA-codes, a Numerically Controlled Oscillator (NCO) to generate replica carrier signals with Doppler shifts, and multiple correlators to determine which set of CA-code and carrier replicas most closely match a particular GPS signal. The complexity of the GPS receiver is suitable for FPGAs, since the design consists of multiple tasks operating in parallel.

4.4 GPS Baseband Processor Design

A GPS baseband processor handles the data that comes from the RF frontend, decodes it, and outputs it at a lower rate which the microprocessor can handle. In this design, the work is done using software running on a microprocessor, with more computational intensive tasks such as FFTs, off-loaded on to hardware implemented on the FPGA. Figure 4.4 shows the overview of the components for the baseband processor. Different hardware alternatives were investigated by examining their datasheets, while selecting the hardware platform for the design of the software defined radio.

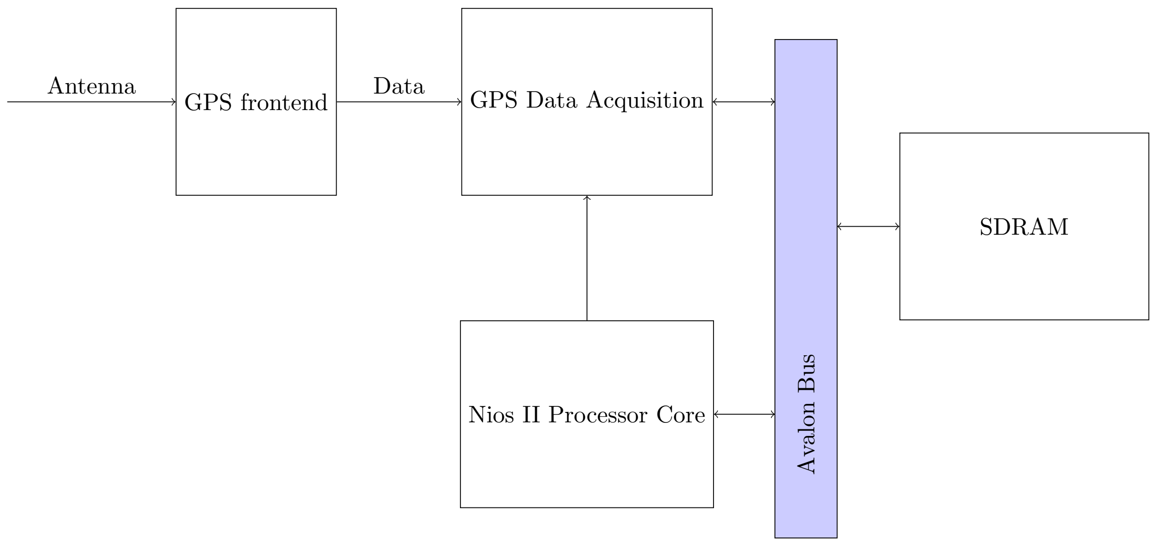


Figure 4.1: Overview of GPS processor design implemented on the FPGA

Ettus Research USRP

The first alternative that was looked at was the Universal Software Radio Peripheral (USRP), which is a line of software defined radios which connect to a desktop computer [18]. They are produced by Ettus Research, which is owned by National Instruments. The USRP incorporates an FPGA and radio frontend together into one unit. They are intended to be connected to a Personal Computer (PC), which runs the SDR software. In order for the software to communicate with the USRP, drivers are needed. The source code for the driver exist for Linux and Windows platforms, allowing it to be built for different processor architectures. The USRP has a built-in FPGA, so the design can be modified to run as a standalone device, without the need of a PC. However, to implement a custom design on the FPGA, a software license for the FPGA design software is must be purchased, which allows support for the particular FPGA used in USRP [44].

USB GPS Receiver Frontend

The second alternative was to use an FPGA development board and Universal Serial Bus (USB) based GPS frontend. [45]. Using a USB frontend would allow for modularity, as different frontends can be swapped and plugged into the USB host port of an FPGA development board. The USB frontends require drivers that connect the hardware and software. However, the drivers supplied are meant for desktop computers, running Windows or Linux, and are complied for x86 architectures. The source code of the driver is not included, which is needed to recompile the drivers specifically for the Nios II processor.

Maxim MAX2769 Evaluation Board

The final alternative that was examined was utilizing an FPGA development board along with the Maxim MAX2769 evaluation board [46]. The evaluation board is intended for evaluating the MAX2769 - Universal GPS Receiver IC in a lab environment, and has the supporting circuitry required to use the IC. The MAX2769 chip performs amplification, mixing, and sampling of the GPS signal. The chip outputs the GPS In-phase (I) and Quadrature (Q) signals as either 1, 2 or 3 bits, selectable through its SPI interface. Using the MAX2769 board solves the problem of interfacing the FPGA and frontend, since it can be directly attached to the GPIO pins on the board, bypassing the need for drivers. The board is intended for use with lab equipment, so it has to be modified to allow for an active GPS antenna, differential voltage signals have to be disabled, and AC coupling capacitors on the clock, and input lines have to be removed in order to use it as a RF frontend for the FPGA board.

4.5 Test Hardware

The chosen hardware was the DE2-115 development board from Terasic, and the MAX2769 evaluation board. The setup can be seen in Figure 4.5. The DE2-115 contains an SRAM-based, Altera Cyclone IV FPGA, along with 128 MB Synchronous Dynamic Random-Access Memory (SDRAM), and 2 MB SRAM.

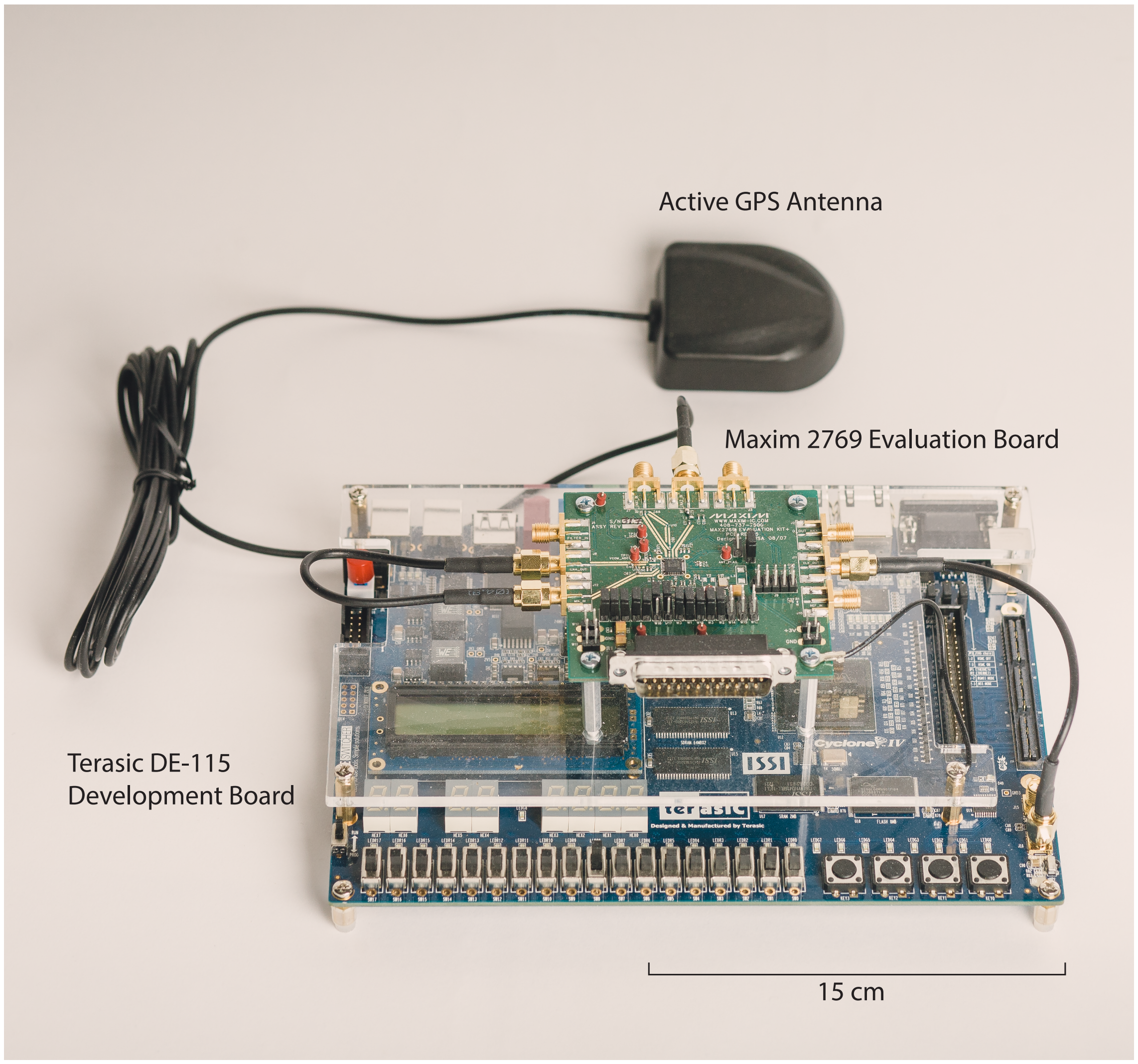


Figure 4.2: Test hardware for GPS processor - Terasic DE2-115 development board with an Altera Cyclone IV FPGA, 128MB SDRAM and 2MB SRAM below, and the MAX2769 evaluation board on top

The Maxim MAX2769 evaluation board contains the Maxim MAX2769 Universal GNSS receiver IC, a 16.384 MHz crystal oscillator, and other components such as capacitors and resistors in order to make the MAX2769 chip work. The configuration registers on the

Maxim IC are configured through an SPI interface and data are transferred through pins from the ADC on the Maxim IC, and clocked by the 16.384 MHz sampling clock of the ADC.

The ADC output and SPI interface of the RF frontend are connected to the GPIO pins on the DE2 board, and the clock used by the ADC of the GPS frontend is fed to the DE2 through a Sub-Miniature version A (SMA) cable, and into the corresponding clock signal input on the DE2 board. To produce a reliable clock signal, the clock signal is passed through a Phase-Locked Loop (PLL), which gives a cleaner signal by recreating a 16.384 MHz clock signal that is synchronized to the incoming ADC clock signal.

4.5.1 Nios II Soft-Core Processor

The design uses a soft-core processor from Altera called Nios II [47]. Initially, an open-source 8051 processor core was used [48], which would make the design independent of Altera specific hardware and software. By using Altera IP cores, implementation was simplified since it allowed using Altera Qsys tool to connect the various other components of the SoC, such as the memory controller, Universal Asynchronous Receiver/Transmitter (UART), SPI controller, etc.

The Nios II IP core from Altera comes in three variations: economy, standard and fast. The economy core is available for free, without a license, and use of the other two requires a paid license. The CAD software from Altera is designed to be used with the Micro-C operating system. While Linux can be made to run on the Nios II processor, it adds additional overhead in terms of code size and complexity compared to Micro-C.

Hardware Multipliers and Dividers

A feature in the standard and fast Nios II processor IP cores are the availability of hardware multipliers and dividers, which allow for use of the dedicated multiplication and division circuits on the FPGA. Without hardware multipliers and dividers, the functionality is implemented in software as a series of shifts and additions. These additional software instructions add significant amount of cycles to the processing time. Figure 4.3 shows the GNU Compiler Collection (GCC) compiler source code for implementing software-based multiplication on the Nios II processor.

```
/* 32-bit SI multiply for Nios II.  */

SItype
__mulsi3 (SItype a, SItype b)
{
  SItype res = 0;
  USItype cnt = a;

  while (cnt)
    {
      if (cnt & 1)
        res += b;
      b <<= 1;
      cnt >>= 1;
    }

  return res;
}
```

Figure 4.3: Compiler source code for Nios II software multiplication [49]

Floating-Point Operations

Basic microprocessors handle numbers as integers, the Arithmetic Logic Unit (ALU) performs the additions, subtraction, multiplications, and division as integers. To handle decimals, either fixed point or floating-point can be used. Floating-point numbers are a way to handle decimal numbers, in a similar manner to scientific notation [50]. As with the hardware multipliers, if the processor core does not contain a Floating-Point Unit (FPU), the floating-point instructions are implemented in software. On the Nios II processor core, hardware floating-point operations can be added by using a feature called "Custom Instructions" [51].

Floating-Point Operations per Second (FLOPS)

The performance of a processor for floating-point operations can be measured in FLoating-point Operations Per Second (FLOPS). FLOPS are calculated using the following formula:

$$\text{FLOPS} = \text{cores} \times \text{clock} \times \frac{\text{FLOPs}}{\text{cycle}} \tag{4.1}$$

To calculate the number of FLOPS the Nios II processor is capable of, numbers from the benchmark results found in the Altera custom instructions guide were used [52]. The benchmark consisted of performing floating-point operations on 1000 pairs of random operands. The results are shown in Table 4.1.

Operation	Total Clock Cycles	Clock Cycles per Operation
Addition	14000	14
Subtraction	14000	14
Multiplication	12000	12
Division	32000	32

Table 4.1: Nios II Floating-Point Benchmark Results

From the benchmark numbers, the best case at 100 MHz was 8.3 MFLOPS for multiplication, and 3.1 MFLOPS for division. In comparison, a 1.8 GHz laptop processor is capable of 28.8 GFLOPS [53].

Millions of Instructions per Second

The performance of the embedded Nios processor was benchmarked to roughly determine its performance. The same test was compiled and ran on a laptop as a comparison. The Dhrystone benchmark simulates processor usage and gives a more better comparison compared to MIPS which can vary based on the type of instruction executed. The Dhrystone benchmark (see Table 4.2) outputs the results as: Dhrystone MIPS.

Compiler Flags	50 MHz Nios II/e	50 MHz Nios II/f	1.8 GHz Intel Core i5
-g,	1.36	23.71	4984.73
-g, -O2	1.32	40.65	12935.27
-O2	1.90	51.74	13156.42

Table 4.2: Dhrystone Benchmark Results

4.5.2 System on Chip

The baseband processor design is based upon the idea of an SoC. It incorporates a processor and various peripherals connected to a bus, allowing the peripherals to communicate directly to the processor, all on one FPGA. Because we are using an Altera FPGA and their soft-core Nios II processors, the blocks are connected using the Avalon bus from Altera, which can be easily configured using the Qsys tool. An overview of the SoC design can be seen in Figure 4.5.2. The Avalon interface defines the clock, reset, and addressing lines of various peripherals on the SoC. Custom hardware blocks that interact with the processor also need to support the Avalon interface.

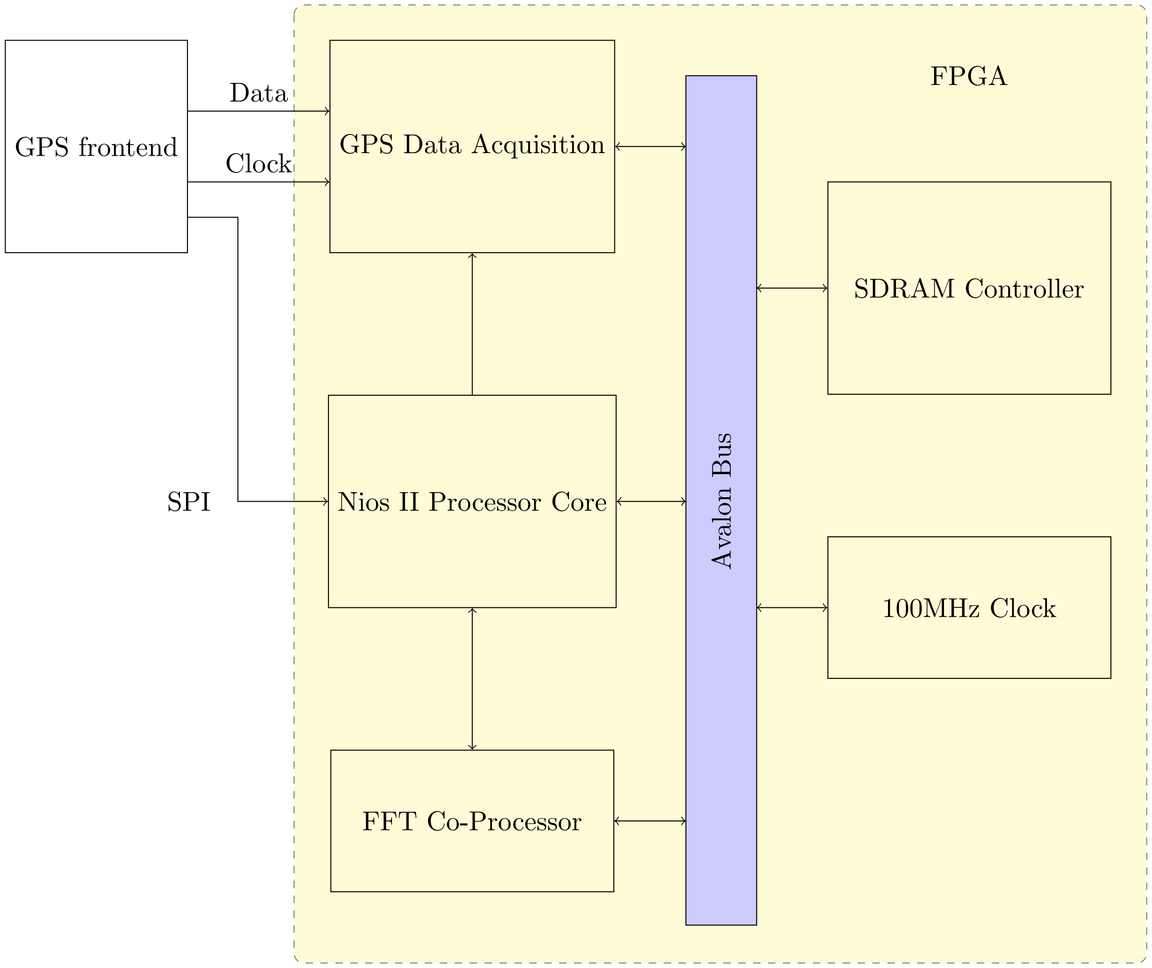


Figure 4.4: Overview of GPS processor design implemented on the FPGA

The hardware FFT block used in this design is an open-source FFT design called “bel_fft” and can be found on SourceForge [54]. It is based on the KissFFT software library implementation, and supports the same Application Programming Interface (API) calls as its software counterpart, which allows existing software using KissFFT to be ported to the Nios II processor and utilization of the hardware-based FFT.

The Avalon bus facilitates communication between the different modules in the Nios II based SoC design. To create a module that talks to another component on the bus, the Avalon interface has to be implemented. The two types of Avalon interfaces used in this design are the Avalon Master, and the Avalon Slave. The slave has its functionality controlled by the master. An example would be a memory controller and a processor core. The memory controller responds to read and write requests from the microcontroller.

4.6 GPS Data Acquisition Block

The design of the data acquisition block is to buffer the incoming data from the RF frontend into memory that will later be read by the processor. The data from the RF frontend comes in at a rate of 16.384 MHz in a 1-bit format. The data needs to be buffered to allow for enough time to process the data, along with formating to convert the 1-bit data into 8-bit signed integers. The rest of this section will provide an indepth explaination of the acquisition module.

4.6.1 Alternatives

The two alternatives to implementing the design of the data acquisition block is to either use software or to use an existing Direct Memory Access (DMA) controller IP. If the design is to be done in software, the processor would need to continuously sample the incoming data and store it in a buffer. The incoming data would be streamed from the RF frontend at a rate of 16.384 Msamples/s. For a 50 MHz processor to sample the data, it would need to complete the software instructions approximately every 3 cycles. It will be shown later that the processor needs at least 6 cycles to complete one instruction - see Section 4.8.1. It is therefore not possible to implement the design on a processor running at 50 MHz. If a 100 MHz clock were to be used, a processor would need to be dedicated to copying the incoming data, as there would not be enough cycles to perform any other task.

The design of the data acquisition module is basically a DMA controller. Altera provides a DMA controller through an IP core [55]. The DMA controller allows for direct access to the memory, meaning the controller itself is capable of reading data from one memory location and writing it to another, without the intervention of a processor. The DMA controller allows for the copying of data to be off-loaded onto hardware. However, since the DMA controller copies data from one memory address to another, the incoming RF frontend needs to be stored in memory. Instead of implementing the design of the data acquisition module to include memory in order to buffer the data for the DMA controller, the design skips the DMA controller and implements its own, without needing addressable memory.

4.6.2 Implementation

The complete implementation has been broken down into three parts: the acquisition module, the Avalon interface, and the software running on the Nios II processor. Figure 4.5 shows the overview of the data acquisition module.

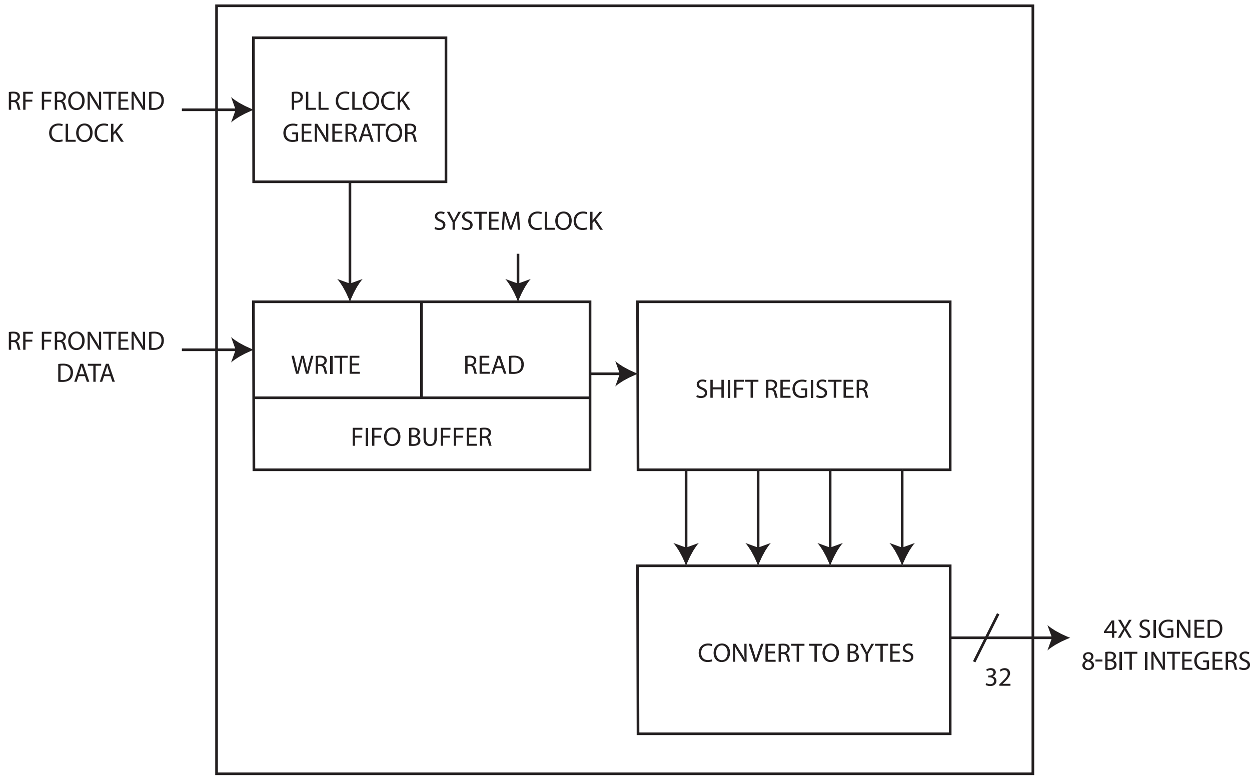


Figure 4.5: Overview of the GPS Data Acquisition module. Data from the frontend is read by a 2-port FIFO, and written from the FIFO into a shift register. Each bit of the shift register is converted to an 8-bit signed integer

4.6.3 Acquisition Module

This block takes the 1-bit I and 1-bit Q signals from the ADC of the RF frontend, buffers it through a small 4-bit First In, First Out (FIFO) buffer, converts the data to signed 8-bit integers and finally stores it in one of two 4ms buffers in SDRAM. The use of 2 buffers in SDRAM allow data to be continuously acquired, while ensuring that the processor has a piece of memory that does not change until the processor has finished processing it. The FIFO buffer is present for metastability purposes, since the system operates off a 100 MHz clock, while the incoming GPS signal is at 16.384 MHz.

Metastability

Because the GPS frontend and FPGA use different clocks, they have different clock domains. The problem with data moving across different clock domains is metastability. Metastability happens when the registers on the FPGA samples the signal while the signal is transitioning, that is the signal is neither a logic 1 or 0. With one clock domain, the signals must be stable during the setup and hold times of the register, but with signals coming from different clocks, it is not certain when the signal changes. To minimize the Mean Time Before Failure (MTBF), a circuit called a Synchronizer is used, which gives the metastable signal extra time to settle to a stable value, by chaining 2 or more flip-flops. Because there are two clocks, a FIFO buffer is used in-between. The FIFO buffer allows the write side (GPS frontend) and read side (FPGA) to have separate read and write clocks. The synchronizer used in the FIFO buffer uses a 3 stage synchronizer to increase the MTBF.

4.6.4 State Machine

The module uses a Moore type state machine, where the output of the state machine depends solely on the current state and not on the inputs. Figure 4.6.4 shows the state machine of the data acquisition module.

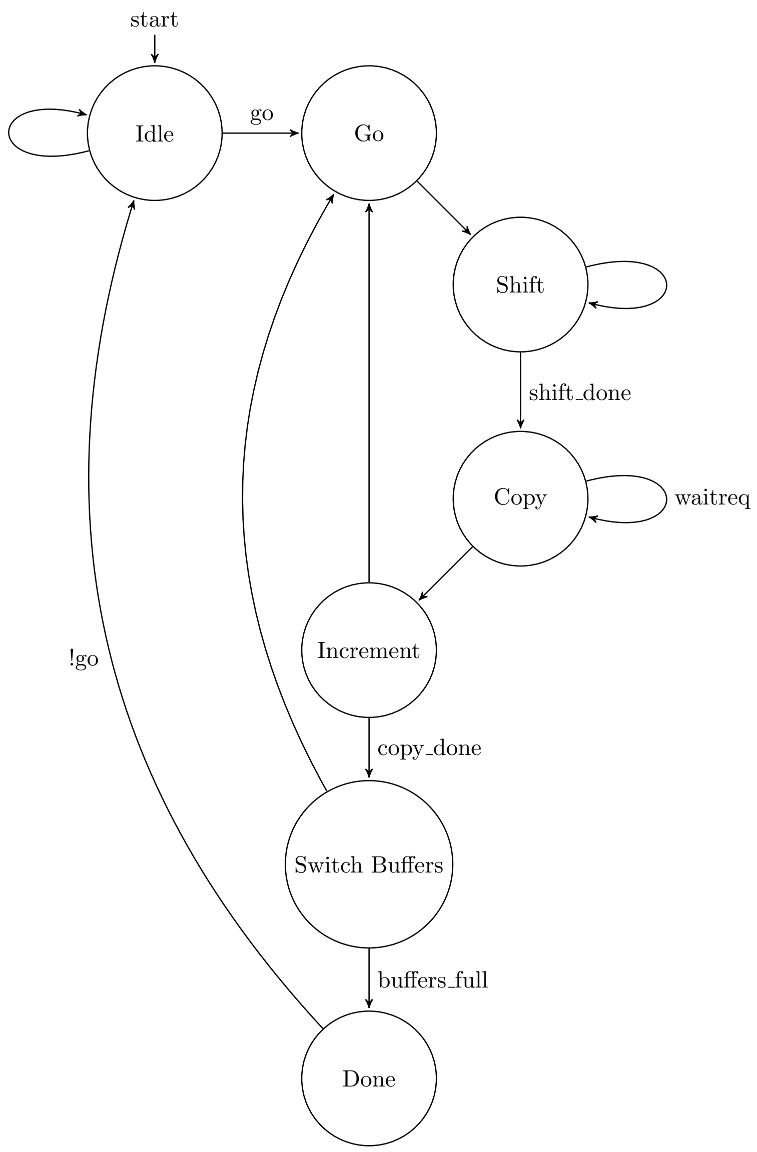


Figure 4.6: Overview of GPS Data Acquisition State Machine

- Shift: Bits are shifted into the shift register from the FIFO buffer.
- Copy: The content of each element of the shift register is converted into an 8-bit signed integer and copied as a memory word (32-bits) to RAM.

- Increment: The memory address is incremented by 4 bytes to make room for the next 32-bits of data to be copied.
- Switch: The memory address jumps to the next block of memory to allow for the processor to read the previous data while new data are copied.
- Done: The module has finished copying 4 ms of GPS signals into memory and returns to the Idle state.

4.6.5 Avalon Interface

The Avalon interface exposes the data acquisition module to the rest of the SoC system through separate interfaces: a master and a slave. The Avalon master allows the module to perform DMA, in which the data acquisition module can write directly to the memory controller without using processor resources. The second interface, the Avalon slave, exposes two registers, a configuration register and one containing the starting external memory address to copy data to. It is through the Avalon slave that the processor controls the data acquisition module. Figure 4.7 shows the overview of the Avalon interface of the data acquisition module.

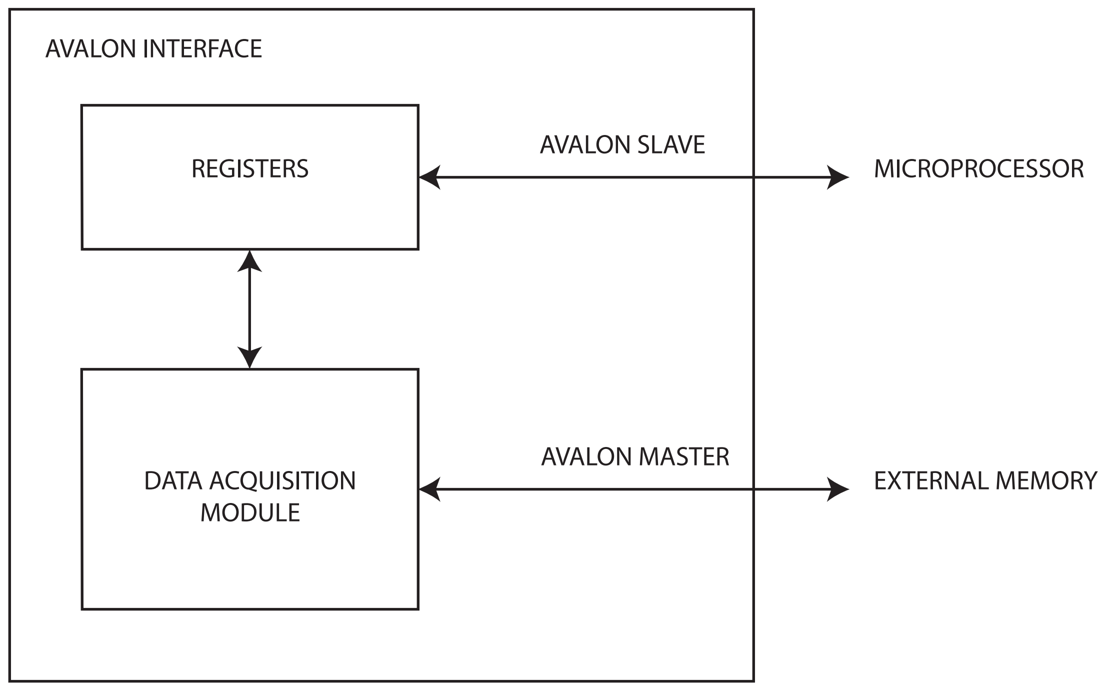


Figure 4.7: GPS Data Acquisition Avalon interface. The Avalon interface is used to allow the data acquisition module to connect to the Avalon bus used by the processor. The processor controls the acquisition module through the registers exposed by the Avalon Slave interface. Data captured by the acquisition module is stored into external SDRAM through the Avalon Master interface.

4.6.6 Software Driver

The software driver is responsible for setting up the data acquisition module through the Avalon interface and also to setup the RF frontend through the SPI interface. The driver must first request a contiguous block of memory through the "malloc" command. Afterwards, the driver provides the Avalon interface with the starting memory address by filling one of the registers on the data acquisition module, with the memory address it had previously requested. Afterwards, the processor can start the data acquisition process by setting a bit on the configuration register of the data acquisition module.

4.7 GPS Satellite Acquisition Block

An alternative approach to satellite acquisition was implemented as a hardware IP. In place of using the FFT, the serial approach is taken, but with multiple blocks operating in parallel, one block per satellite PRN. In place of performing the GPS satellite acquisition using software, a hardware module was written in order to perform the acquisition in real-time and in parallel. The hardware block replaces the FFT Co-processor from the section above (see Section 4.5.2), as the hardware is designed to process the data using the serial approach. Therefore, the FFT module and the corresponding software is not used. The other difference in this design is the use of a slower 50 MHz clock, since the processor does not perform the computation, using a slower clock saves power. Inside each acquisition block, 40 multipliers are used to calculate the power of the 20 different replica carrier signals. Figure 4.7 shows the overview of baseband processor with the satellite acquisition block.

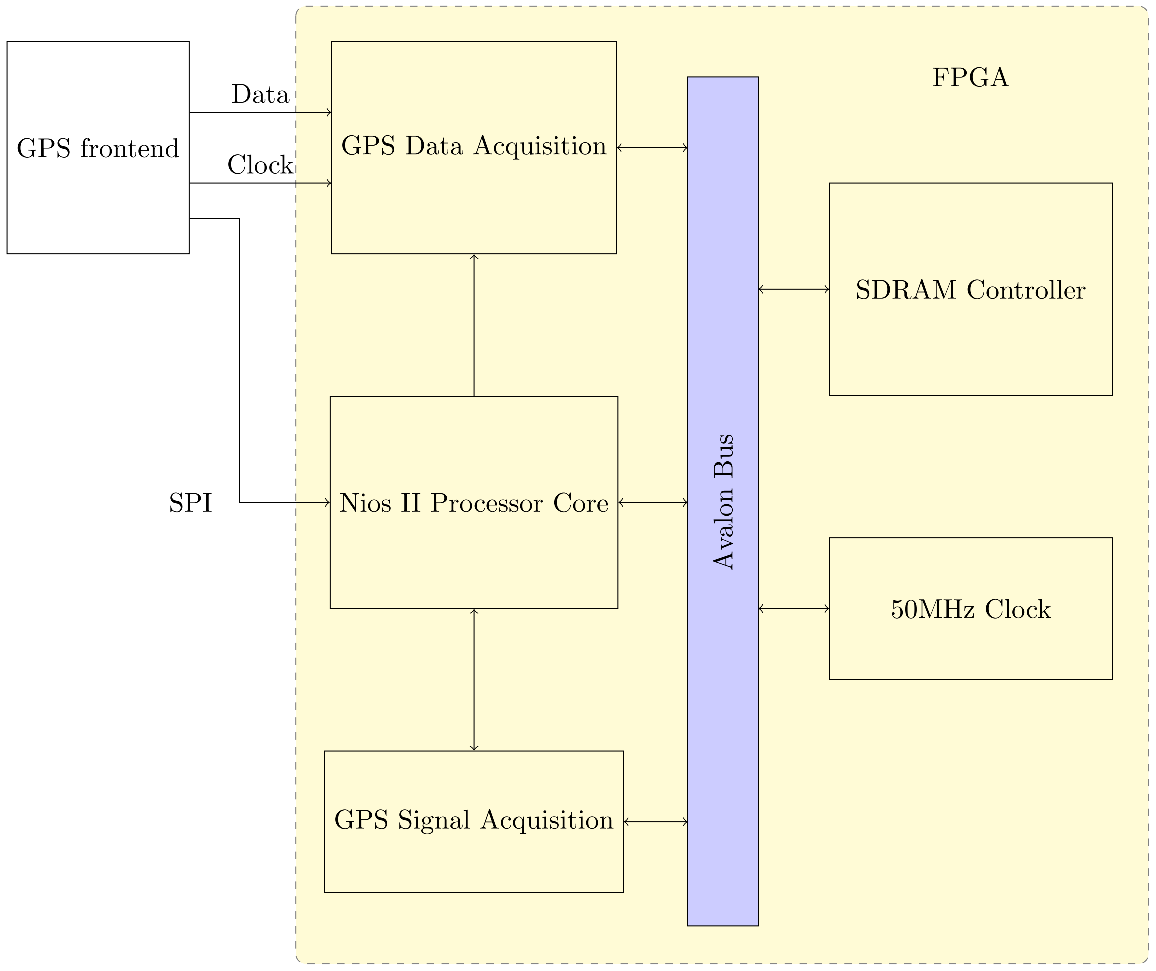


Figure 4.8: Overview of GPS processor design implemented on the FPGA. Notice that this figure differs from that of Figure 4.5.2 by replacing the FFT module with the hardware-based GOS Signal Acquisition module

4.7.1 Design Alternatives

One alternative is to simply acquire a license for an existing IP core, such as CellGuide CGsnap IP Core or IBS Technology HK NOOSHA GPS-BB. The two main problems are the cost of the IP license and the hardware requirements in terms of needing a specific RF frontend and FPGA. Another reason for not using a commercial IP is the goal of creating a platform in which future projects can be implemented. Software is another alternative to implementing a GPS processor design on an FPGA. While a design can be created that runs in real-time on a processor, the clock cycles would need to be much greater than on the FPGA design, and use more power. As will be shown in the following sections, the processor needs many more clock cycles compared to an FPGA in performing a similar task.

4.7.2 Implementation

Figure 4.9 illustrates the overview of the GPS Satellite Acquisition module, . The design has: 20 Replica carrier generators; 32 CA-code generators; and 8 Acquisition channels. This allows the design to search for 20 different possible Doppler shifts and for 8 different PRNs at the same time.

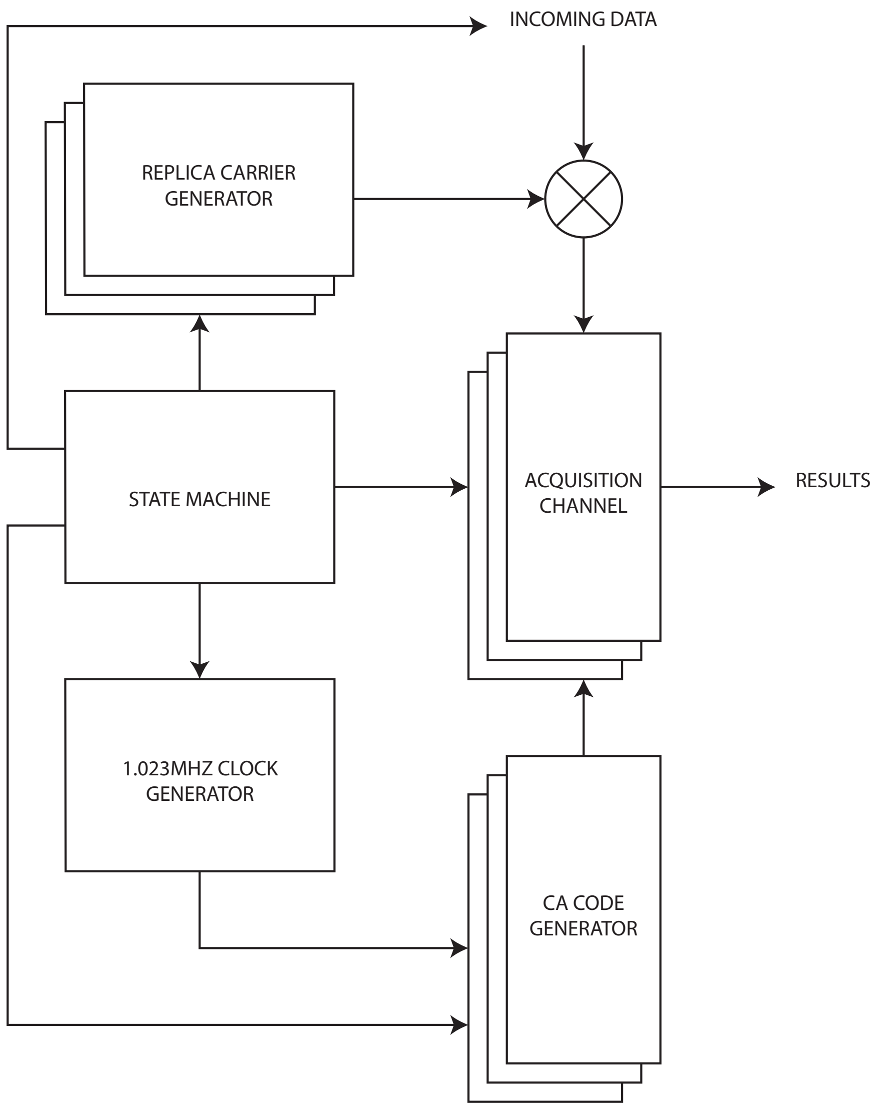


Figure 4.9: GPS Signal Acquisition module overview

4.7.3 Multiplication and Representation of Numbers

The incoming data from the GPS frontend is represented by sign and magnitude. For the 1-bit case, it is simply the sign, which represents either a +1 or -1. This type of signal is called a Non-Return to Zero (NRZ). The signed NRZ integers from the GPS frontend are represented by 1s and 0s in the FPGA. Specifically, a -1 would be represented by a 0 and a 1 would be represented by a 1.

The design simplifies multiplication of the 1-bit signal by using a single negated XOR (XNOR) gate in place of the 18-bit multipliers found on the FPGA. There are a limited number of multipliers found on FPGAs, i.e., 266 multipliers are found on this particular model, along with thousands of logic gates. Using a single gate allows for the design to be easily scaled. The truth tables for an XNOR gate and multiplication are shown in Table 4.3 and Table 4.4 respectively:

x	y	XOR	XNOR
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

Table 4.3: FPGA computation using a single logic gate

x	y	*
-1	-1	1
-1	1	-1
1	-1	-1
1	1	1

Table 4.4: Equivalent computation using signed multiplication

4.7.4 Acquisition Channel

Each acquisition channel is controlled centrally by the state machine. Each module is responsible for the following: XORing the signal I and Q with the CA code from a PRN generator. Integrating the separate I and Q results over a period of 1 ms. Calculating the combined power from the integrated I and Q result Comparing and storing the highest power value seen and corresponding CA code phase. Determining which of the 20 replica carriers and code phase has the best power overall for its PRN.

4.7.5 Replica Carrier Resolution

Since the input data arrives as 1-bit integers, the NCOs produce a 1-bit output. The design was verified using MATLAB by comparing between a 1-bit sinusoidal signal and a 64-bit sinusoidal signal. Both produced the same output after multiplication with a 1-bit signal as shown in Figure 4.10 produced using MATLAB, after cross-correlation with a raw GPS signal.

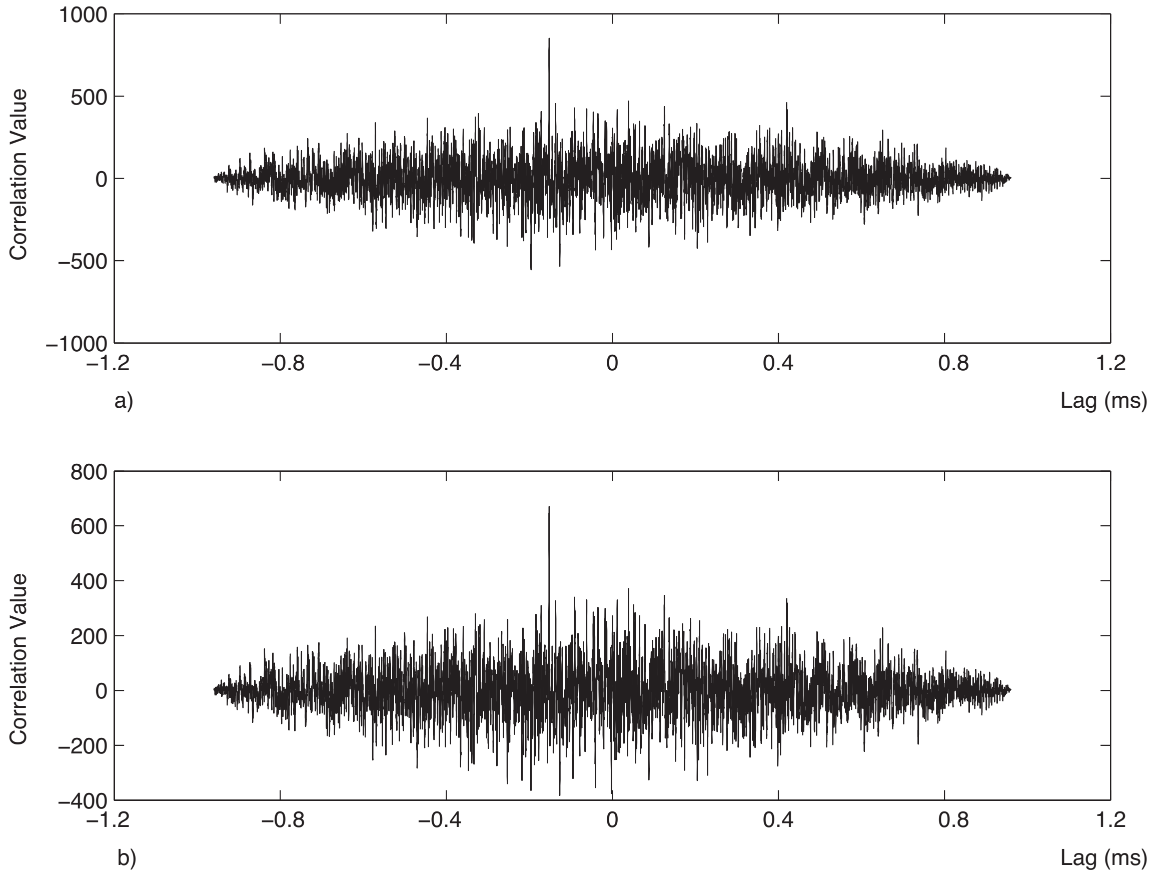


Figure 4.10: Comparison between a) a 1-bit replica carrier signal, and b) a 64-bit replica carrier signal after cross-correlation

4.7.6 Replica Carrier Frequency Resolution

Figure 4.11 shows the effect that the carrier frequency has on being able to acquire the GPS signal. The detected frequency in this case is approximately 2000 Hz. Other carriers signals are generated in increments of 500 Hz, and cross-correlated with the GPS signal. Using carrier frequencies that are with ±500 Hz of the original carrier signal results in a spike indicating the presence of PRN 1 within the signal, but with different code phases.

However, when a replica carrier that is 1000 Hz more than the frequency of the original carrier signal is used, the spike disappears entirely, and we can no longer see the presence of PRN 1, nor its code phase in the signal.

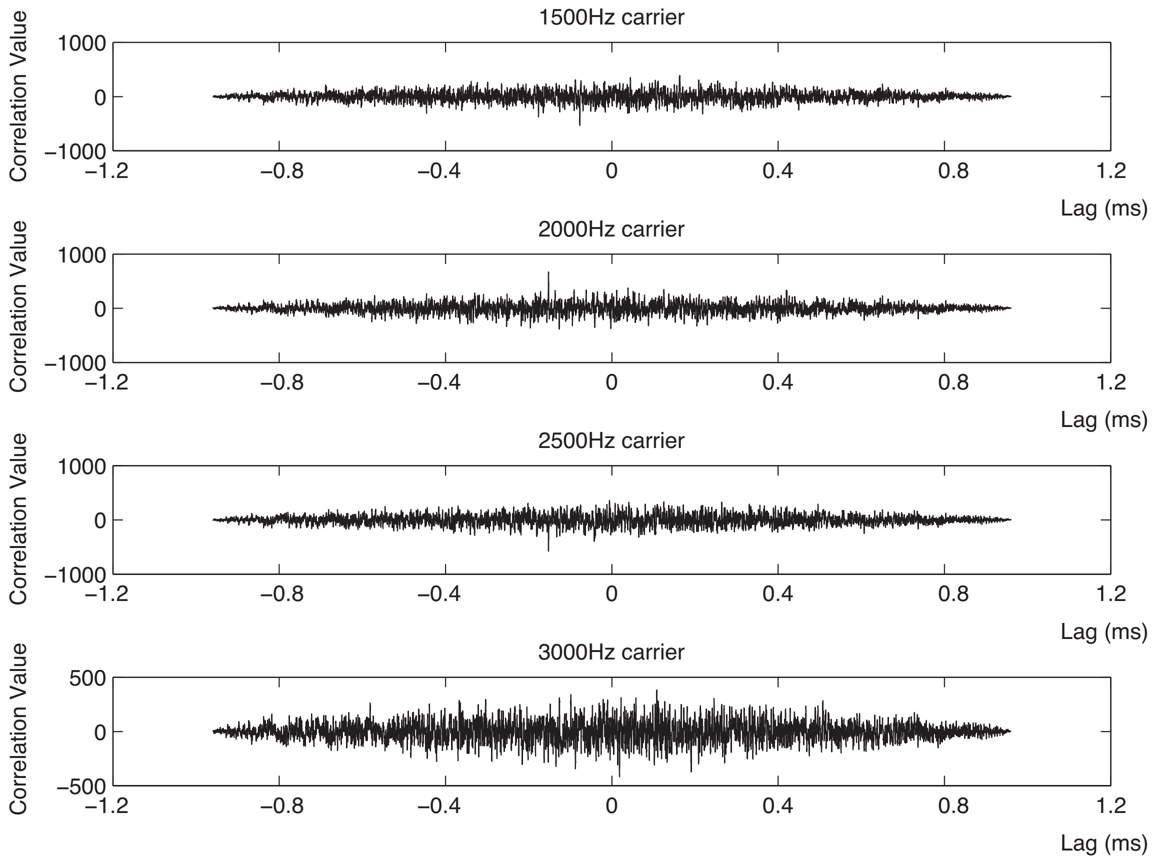


Figure 4.11: Different carrier frequencies and the effect on CA-code correlation

Resolvability

Resolvability refers to the precision in the frequency domain, if the FFT output of one frequency can be distinguished from another. As can be seen from the previous section, the frequency of the carrier signal must be resolvable with an accuracy of at least 500 Hz. The resolvability of the FFT transform depends on the number of data points fed to the FFT function, that is the record length of the signal, which can be represented by Equation 4.2.

$$T = \frac{1}{\Delta f} = \frac{1}{500 \text{ Hz}} = 2 \text{ ms} \tag{4.2}$$

With the GPS signal, there is a trade off between time resolution and frequency resolution: a longer record length gives a more precise frequency determination. However, with a longer time period, the frequency of the carrier signal and the Doppler shift might change during the longer time period. Using the time-domain-based implementation, an arbitrary Δf resolvability can be chosen for arbitrary record lengths. The frequency domain implementation would have to trade-off between time (the record length) and frequency (the resolvability).

Another limitation is in the hardware implementation. One hardware-based FFT core can perform up to a 32768 point FFT [56], which equates to a record length of 2.73 ms when sampling at 12 MHz, giving us a resolvability of 366 Hz. The open-source bel_fft, supports a 256 point FFT. A 256 point FFT at a sampling frequency of 12 MHz means a record length of only 21 ns. This results in a frequency domain resolvability of 46 875 Hz.

4.7.7 Numerically Controlled Oscillators

The design of the hardware module uses a custom NCO to produce the reference 1.023 MHz and replica carrier signals. The oscillator works in a similar manner compared to other

NCOs. A phase accumulator keeps track of the phase of the sinusoidal signal, and different frequencies are derived from different phase increments. Phase increment is computed from Equation 4.3, where N is the number of bits in the phase accumulator. The number of bits in this case is 16, because a 32-bit version of Windows was used, which caused synthesis problems when the phase increment for the 32-bit number was synthesized. The f_{clock} for this design is actually the sampling clock used by the GPS frontend, and not that of the system clock. This means that the design can be clocked faster without modifying the parameters of the NCOs. Because the design produces a 1-bit signal, a lookup table, which is commonly found in NCOs, was not used. In its place, simple AND gates were used to compare the bits of the phase accumulator.

$$\text{Phase increment} = \frac{f \times 2^N}{f_{\text{clock}}} \tag{4.3}$$

4.7.8 Replica Signals

Replica carrier and CA-code signals are used on the receiver to determine the phase shift between the replicas and the actual GPS signal. To create the replica carrier signals, the NCO described previously is used. Both the sine and cosine components of the NCO are used, creating a complex sinusoid. Multiplication between the complex sinusoid and the incoming data, gives us the I and Q components of the signal. In total, 20 separate replica signals are created, each with a 500 Hz difference in frequency. The different frequencies are generated by varying the phase increment of each replica carrier generator NCO.

From the GPS specifications, a 1.023 MHz clock is used to generate the CA-codes on each satellite. To replicate the same CA-code timing, a 1.023 MHz clock was created using the

1-bit NCO from above. The CA-code generator consists of a custom module was created to implement the LFSR, S1 and S2 parameters, along with the 1.023 MHz clock input. In total, 32 copies of the module are made, each with a different S1 and S2 parameter to create the appropriate CA-code for each PRN current in use by GPS satellites.

4.7.9 State Machine

The state machine is what coordinates the other components. It goes through the different states which determines the different control signals going to the different modules. Figure 4.7.9 shows the state machine of the satellite acquisition module.

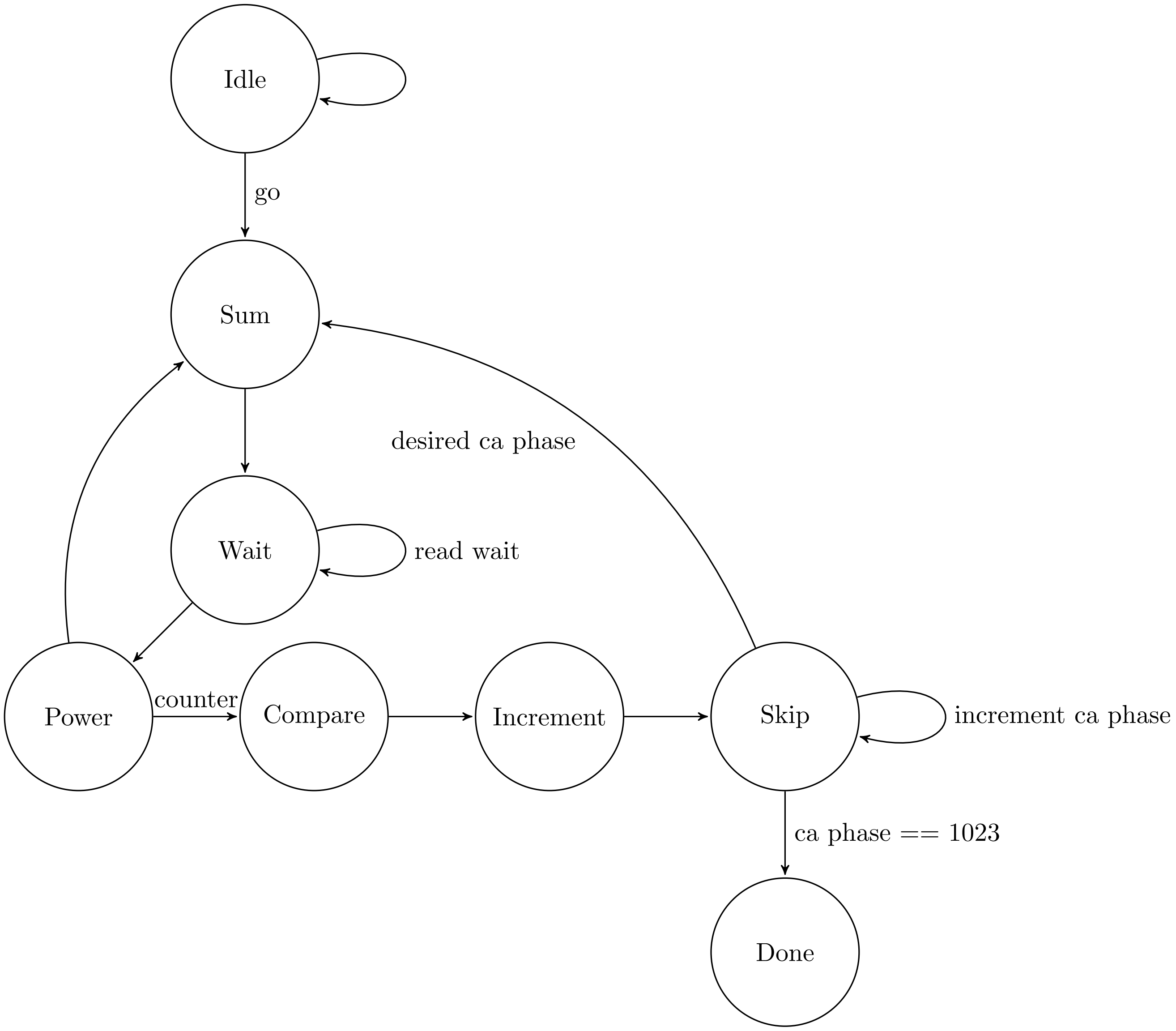


Figure 4.12: Overview of GPS Acquisition State Machine

- Sum: Separately adds the I and Q components into the I and Q running total accumulators.
- Power: Power is calculated by summing the squares of the I and Q components

together. Note that square root was not implemented.

- Compare: The power calculated in the previous step is compared with the previous "best" power, and if the current power is greater, the phase is recorded and the "best" power updated.
- Increment: Resets the counter and the summing registers to prepare for the next set of calculations based on the next code phase value.
- Skip: Goes back to the previous state to increment the code phase of the CA code generator until we arrive at the next code phase we need for our calculations.

4.7.10 Interface

Similar to the Data Acquisition module above, the Satellite Acquisition module has setable registers exposed by the Avalon interface in order to allow communication with the Nios II processor. The Satellite Acquisition module contains four registers for:

- Configuration - used to select the PRN, start / stop the state machine, and the status of the module
- Memory base address - used to set the location of the data used by the module
- Power Level - the calculated power for the selected PRN
- CA Code phase and Doppler shift - the CA code phase and Doppler phase corresponding to the highest power level for the selected PRN

Also similar to the Data Acquisition module, a software driver is used to setup, control and read the values from the module. The driver has to set the base address register to point to the block of memory containing the raw GPS data (from the data acquisition module).

Afterwards, a PRN is selected and the state machine started. When the state machine is finished its calculation for a given bank of PRNs (the module performs calculations on 8 PRNs at a time), the data can be read back through the registers.

4.8 Testing

The design was verified using different methods due to the complexity of the design. The satellite acquisition portion was verified with test benches written on Verilog and using ModelSim to simulate the design. The simulator allows the functionality of circuit to be verified. The components that were tested separately were the 1-bit NCO, and the CA-code generator, as they did not rely on any external components. Afterwards, the module along with the Avalon interface and software component running on the Nios II processor were tested. The functionality of the GPS signal acquisition unit was compared with a software model written in Objective-C. The satellite signal acquisition testing was done by loading a previously recorded GPS signal into both the Nios II processor and the Objective-C code.

4.8.1 Feasibility

An approach to flying a GPS receiver on a CubeSat would be to buy a space-qualified GPS receiver. But the downside is the high price of a space-qualified GPS receiver, e.g., 250,000 USD [57]. Designing the IP is an attempt to create a low-cost alternative. To simplify the design process, instead of building everything from scratch, other IP can be incorporated into the design.

Existing IP Cores

Included in the Altera software is an Avalon DMA Controller, which allows for the FPGA hardware to copy data from one piece of memory to another. However, a custom design was needed, since the design had to handle multiple clock domains and needed to convert the 1-bit data into an 8-bit signed integer. Another IP available from Altera is the NCO MegaCore Function. The IP is available for synthesis as an OpenCore Plus evaluation, but a license is required to generate a programmable design. The license for the IP costs $2495.00 USD [58].

The synthesized NCO MegaCore function was compared with the custom designed 1-bit NCO. The settings for the NCO MegaCore were chosen to closely match the 1-bit NCO as much as possible; 16-bit accumulator with no phase dithering. The minimum selectable signal resolution was 4-bits. From the previous section, it was shown that increasing the resolution of the carrier signal beyond 1-bit does not significantly improve performance. The NCO MegaCore also has different ways of implementing the NCO, each results in different resource usage, see Table 4.5.

	Small ROM	Large ROM	CORDIC	Multiplier-based
Logic Elements	233	36	1168	870
Memory Bits	36	320	0	120
M9K	2	2	0	1
DSP	0	0	0	0

Table 4.5: FPGA resource usage comparison between different alternative NCO implementations

The Small ROM NCO MegaCore implementation was selected, as it had the least combined logic element and memory usage. Table 4.6 shows a comparison on the actual FPGA resource usage when instantiating the NCO MegaCore design, and the custom 1-bit NCO design.

	NCO - Small ROM	Custom NCO
Total logic elements	293	17
Total registers	185	16
Total memory bits	4	0
Total PLLs	0	0

Table 4.6: FPGA resource usage comparison between the smallest alternative implementation and the 1-bit NCO implementation

Theoretical Comparison

This section theoretically compares the number of clock cycles to process 1 bit of data on FPGA state machine versus running software on the Nios II processor. For a processor to do the same work, it would be more efficient to compute the data sequentially, computing calculations for each PRN one after each other, as multi-tasking would require context switching, which increases the overhead. For each instruction in a program, the processor must do the following in order to recognize and execute the instruction: Fetch; Decode; Execute; Memory; Align; and Write-back.

ALU instructions are 6 cycle.

For this comparison, the Nios II/e was chosen, as it has only one pipeline, it makes the comparison simpler.

Below is the software assembly equivalent of the Sum, Wait, and Power cycle part of the state machine:

```
ldw r4, 0(r9)           #load the value into r4
ldw r7, 0(r2)
ldw r8, 0(r3)

xor r5, r4, r7          #xor sampled bit with cos, store in i
nor r5, r5, r5          #NOT - Nios II does not have a NOT
xor r6, r4, r8          #instruction, using nor does the same thing
nor r6, r6, r6
xor r5, r5, r9          #xor with the ca code
xor r6, r6, r9

add r10, r10, r5
add r11, r11, r6

mul r12, r10, r10       #calculate the power
mul r13, r11, r11

add r1414, r12, r13
```

There are a total of 14 instructions, each taking 6 clock cycles to complete. The software version takes 84 clock cycles to compute the same answer the FPGA design takes 3 cycles to calculate, which is 28 times longer.

For the above software to process 8 PRNs, the time would be 672 clock cycles. The FPGA design is able to compute 8 PRNs in parallel, still requiring the same 3 cycle to calculate the result.

Results

The serial-search algorithm was ran on a 1.8 GHz Core i5 laptop and on the FPGA development board with a 50 MHz clock. The time to complete the search is shown in Table 4.7. The time measurements were completed using software running on the processors, recording the start and stop times each time the search algorithm ran. From the measured times, the software implementation running on the 1.8 GHz laptop was 1.3 times slower than the hardware implementation running on the 50 MHz clock. If the clock speeds are normalized, the result is that the software implementation runs approximately 47 times slower than the hardware implementation. The results are not comparable to the theoretical comparison because the Core i5 processor is more advanced than the Nios II processor, supporting pipelining amongst other features.

	1.8 GHz laptop	50 MHz FPGA
Total time (seconds)	9.12	6.86
Average time per PRN	1.14	0.86

Table 4.7: Time to complete a serial-search on a laptop compared to FPGA

4.8.2 Future Improvements

A future improvement would be to further parallelize the design. Instead of multiplying each element one at a time, multiple elements can be multiplied at one, since each computation does not depend on another element. The computation would not be limited by the number of multipliers, but the computation of the replica carrier signals. Therefore, if the replica carrier signals were to be precomputed, then the elements can be simply multiplied and summed in parallel. The computation process is currently controlled by a hardware-based state machine. By implementing parts of the state machine in software, the datapath can be changed dynamically allowing further resources to be freed by reusing multipliers and NCOs.

Chapter 5

FPGA for Camera Interfacing

A CMOS image sensor is a type of payload that can be found on a nanosatellite. An image sensor can used as a star tracker, which is used to take images of stars, and through the relative position of the stars, can be used to precisely determine the orientation of the satellite in space. The imager can be a CMOS camera, which by design uses a dedicated clock to stream pixel data at high speeds over a serial connection. In order for a relatively low-speed processor to handle the data, an interface needs to be created to facilitate the communication.

5.1 Interfacing Components

Tne particular imager used in this case outputs an image where each pixel is 10-bits in size. The camera interface supports both parallel and serial interfaces. Parallel requires more space in terms of the number of connections needed. The serial interface sends data using only two wires, called a differential pair, but it is outputted at a higher rate, in order to transmit the same amount of data as the parallel interface. The baud rate is the number of bits that gets transferred per second. In the case of the camera, the 10 data bits, one start bit and one stop bit, are sent at 26.6 MHz, resulting in a baud rate of approximately 320 Mbit s^{-1}. The FPGA sits between the camera and microprocessor. The functionality of the FPGA is to buffer the data, and re-encode the stream of data as pairs of 8-bits.

5.2 Limitations of Microcontroller UART

Due to the data rate, the microprocessor must use a hardware-based UART in order to capture the data, versus using software interrupts and timers to grab the data through regular I/O pins. The hardware UART on the microcontroller (a 400 MHz, ARM926-based, Atmel AT91SAM9G20) supports a 5-bit to 9 bit asynchronous serial communication. The camera outputs a 10-bit data packet, which makes the microcontroller UART incompatible.

The datasheet for the microcontroller specifies that the maximum baud rate is the Master Clock (MCK) divided by 8 or 16 (since the microcontroller does asynchronous UART by oversampling), or an external clock that must be 4.5 times slower than MCK. Since the camera has a 320 Mbit s^{-1} baud rate, the MCK frequency must be greater than either 1.44 GHz if using an external clock, or 2.56 GHz if using the only the MCK clock.

Another issue with using the microcontroller is the signalling used by the camera compared to conventional UART. The camera uses a logic high (1) for a start bit, and a logic low (0) for a stop bit. The microprocessor UART expects the opposite, a logic low (0) for the start bit, and a logic high (1) for the stop bit. The microcontroller therefore is not able to handle the data coming directly from the camera.

Low Voltage Differential Signal (LVDS) interface uses a clock that is synchronized with the data packet. The camera uses the pixel clock to clock the packets of data sent along its LVDS interface. The camera interface uses an LVDS standard called FPD-Link II, which is an LVDS signal with an embedded clock. There are two types of embedded clock signals: isochronous and anisochronous. The LVDS signal in this case is anisochronous, meaning that start and stop bits are used to imply the clock. The start and stop bits occur in the middle of the period where the pixel clock is high, as shown in Figure 5.1.

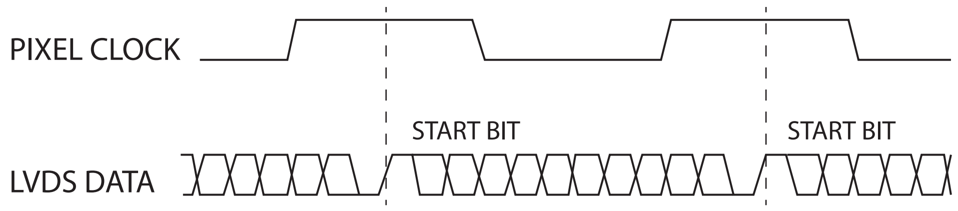


Figure 5.1: LVDS Clock Timing

5.2.1 Alternative Solutions

The problem with using an off the shelf component is that, most COTS LVDS Serializer-Deserializer (SERDES) chips expect an external clock signal, and most bus widths are either 18-bit or 24-bits. The Altera FPGA has an LVDS IP core, but the component expects an external clock signal along with the LVDS data. Possible alternatives include: using a 10-bit LVDS IC that supports an embedded clock; oversampling the 12-bits using an FPGA; or clock recovery and using an LVDS module on an FPGA.

An alternative is to use a dedicated 10-bit LVDS IC that supports an embedded clock. An FPGA is still required as the data are being received faster than the OBC can handle. The FPGA would simply buffer the incoming data onto external memory, and provide a serial output to the microcontroller. The disadvantage of using an external IC is the addition of another chip, and the conversion to parallel results in at least 10 extra traces that must be routed to the FPGA, both of which will require more board space.

Another approach is to use oversampling. Oversampling is sampling the LVDS data on the FPGA at a faster rate, i.e. faster than the 320 Mbit s^{-1} rate at which the data are being received. When oversampling, the goal to detect the transition between the bits, and capture the point that is in between the transitions, when the data are stable, satisfying the setup and hold times of the FPGA.

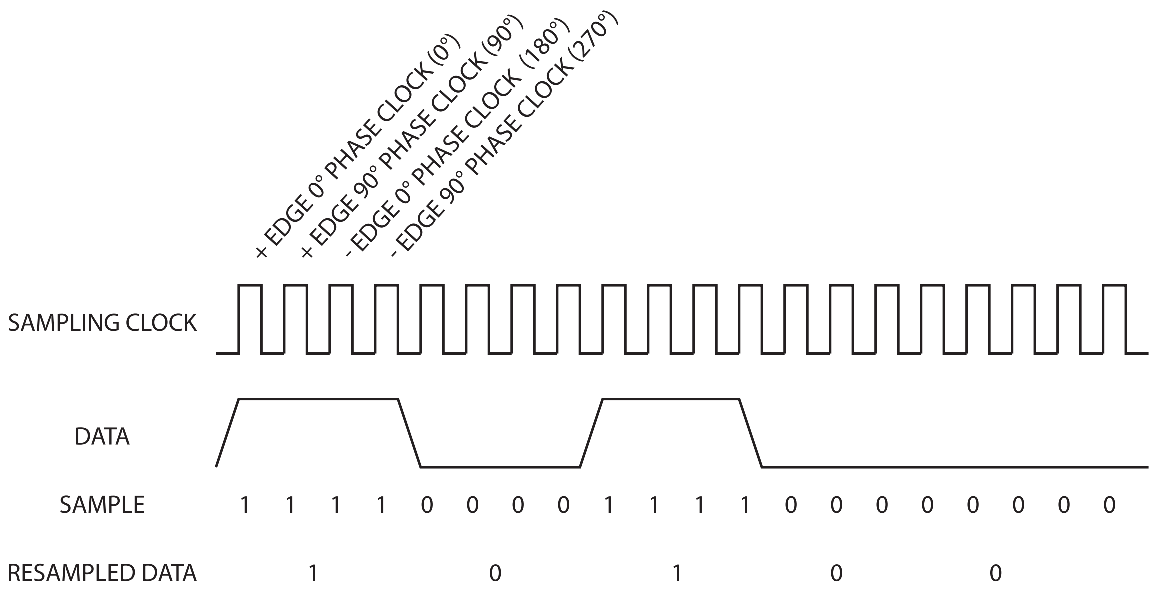

+ EDGE 0° PHASE CLOCK (0°)
+ EDGE 90° PHASE CLOCK (90°)
- EDGE 0° PHASE CLOCK (180°)
- EDGE 90° PHASE CLOCK (270°)
SAMPLING CLOCK
DATA
SAMPLE
1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0
RESAMPLED DATA
1 0 1 0 0

Figure 5.2: Oversampling. Adapted from [59]

To oversample, a deserializer is used (which takes the incoming serial data and outputs it in blocks of 12 bits), and the clock sped that the deserializer operates at is increased, sampling additional bits in order to find the transition, and then determine the value of the bit. The clock for the deserializer module can be set by using a PLL to generate new clock frequencies that are multiples of the incoming data rate. This approach works well for slow data rates, such as 115200 baud or even at 11 Mbit s^{-1}, but the FPGA is unable to generate a clock using the PLL at the lowest oversampling value required, as 3 times oversampling means a clock of 960 MHz. The Altera SRAM FPGA used can only reliably generate approximately 300 MHz clocks. For faster clock speeds, different phases of the clock such 0°, 90°, 180°, 270°, are used to oversample the incoming data.

The final alternative is to create and synchronize a replica clock based off the start bits from the LVDS data stream. The start bits are used to toggle a flip flop, which produces a clock that is half of the actual LVDS clock. A PLL is then used to increase the clock rate in order to have a clock that matches the pixel clock of the camera.

Once the clock is set, it can be used with an LVDS IP module on the FPGA. Afterwards, it is simply a matter of recording the data to external memory and playing it back to the microcontroller over UART.

5.3 Implementation

Each of the alternatives have their own complexities involved. Introducing an additional LVDS IC requires additional space on the PCB and an additional interface to be created to allow for the buffering of data from the LVDS IC. With the clock recovery implementation, a complex clock recovery algorithm needs to be implemented in hardware. Of the three alternatives mentioned above, the oversampling method will be examined as it adds the least amount of complexity.

5.3.1 Oversampling

The first module that is required for oversampling is the PLL, which is used to derive the clocks needed. From the system clock, two separate clocks are generated. Their frequencies are the same as the incoming data rate, but their phase are 90 degrees off from one another. This allows each bit of the incoming data to be sampled four times by the two clocks, i.e., 0°, 90°, 180°, and 270° out of phase. An overview of the modules can be seen in Figure 5.3.

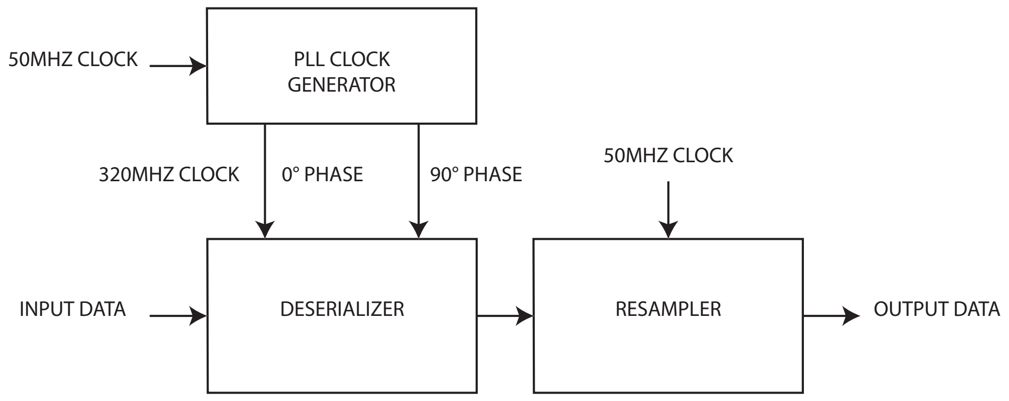


Figure 5.3: Oversampling Modules. Adapted from [59]

The edge of the signal transition is detected using the edge detector circuit shown in Figure 5.4. The edge detector works by comparing a signal with its delayed counterpart, and when an edge is encountered, it causes a pulse to be sent. The pulse determines which of the four clocks to use to sample the data. The deserializer module takes the serial data coming in and outputs it as bits in parallel. The deserialization is done by creating a register to hold the oversampled bits, with 4x oversampling, 48-bits. The 48-bits are then read out in parallel into the next module. The next step is to find the transition from the stop bit to the start bit, which determines the start of the incoming data packet, and also which sample to use in determining the value of the following bits.

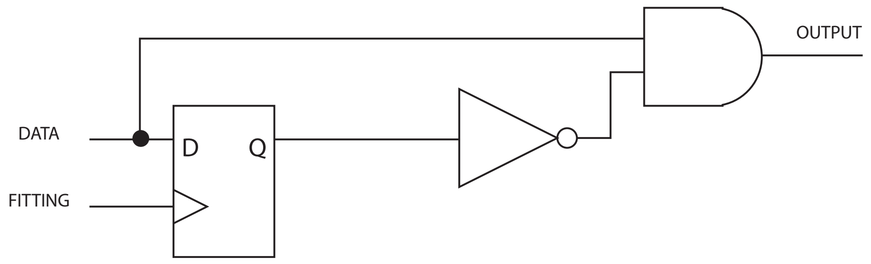


Figure 5.4: Edge Detector Circuit

5.3.2 Verification

To verify the design, it was simulated with ModelSim, and tested on an SRAM-based FPGA development board. The FPGA board was connected to a CMOS camera through a pair of wires for the LVDS signal, and the camera was connected to a USB I^2C interface to configure the camera settings. The deserializer interface was able to detect the start and stop bits of the data from the camera and the data was outputted. The output was verified visually through the use of lights on the development board, each light representing a bit in the output data. When the camera lens was covered, the intensity of the lights decreased, representing an average value of 0. And when the lens was exposed to light, the intensity of the lights on the development board increased, representing an average value of 255, which is the maximum number for an unsigned 8-bit integer.

5.3.3 Feasibility

Implementing an LVDS receiver on an FPGA is feasible. The incoming data at 26.6 MHz is too fast for the microcontroller to handle, and a hardware-based approach needs to be used. If in place of the FPGA-based LVDS module, a separate LVDS receiver IC chip were to be used, the parallel data coming out of the chip is still needs to be stored somewhere, requiring an additional device to manage the transfer of data into memory. Since the design of the FPGA-based camera interface was written using Verilog without using additional libraries, the desgn should work on different types of FPGAs. The design is easily prototyped on an SRAM FPGA due to the features available for debugging. However, for practical applications, a small Flash-based FPGA is ideal, since it uses less power, is smaller, and does not require programming the device on start up.

The design of the camera interface module on the FPGA is similar in function to that of the data acquisition module from the previous chapter. They both show that an FPGA can be used for interfacing with different digital signals. The use of FPGAs for interfacing components can be used in place of dedicated ICs and general purpose processors, allowing for both the capture and processing of data to be contained on a single IC.

Chapter 6

Conclusions and Future Work

This study looked at the feasibility of using an FPGA for different designs to be used on a nanosatellite. Feasibility was looked at in terms of the value the design added and possible alternatives to the design. Also considered was the fact that in a university setting, certain resources might not be available, such as licensed IP cores, or an established work flow. Four designs were examined, a WDT for a power board, data acquisition and processing for a GPS receiver, and finally a serial camera interface.

The first design is a WDT for a power board using a Flash-based FPGA. The WDT is intended to reset the main microcontroller when the microcontroller stops responding. Along with resetting the microcontroller, the FPGA also maintains the state of the I/O pins that are connected to the control pins on the other ICs. The design however was not feasible, as it was ultimately not useful, since it did not improve the power board performance significantly.

The second design consists of two parts for GPS signals: one to capture data, and another to process the data. The GPS Data Acquisition module connects directly to the RF front end with the clock input, and I and Q data. A Nios II processor was added to communicate with the RF front end registers through SPI, and sets up the data acquisition module through the Avalon interface. The second part of the GPS design is the satellite acquisition module. The module works with the data collected from the data acquisition module, and performs work in order to detect the parameters of the different GPS satellites that might be in the data. These parameters include the CA code phase, the Doppler shift, and the power level of each satellite. The design is feasible, although it involved more time and is more complicated than a comparable software implementation. The FPGA design is able to perform much more efficiently compared to the software design.

The final design is a serial camera interface. The design sits between a CMOS imager and a microcontroller, this allows for the slower microcontroller to process the data coming from the much faster CMOS imager.

6.1 Conclusions

Along with the feasibility of implementing each design, the work flow of implementing an FPGA design can also affect the feasibility of either implementing the design on an FPGA or finding an alternative.

6.1.1 FPGA Development

When developing an FPGA design, it is easier to prototype the design using an SRAM-based FPGA instead of a Flash-based one. Using an SRAM-based FPGA typically allows for more features, since the FPGA has more resources available. One useful feature during the design of the FPGA is to be able to probe signals inside the design running on the

FPGA. It is the equivalent of a logic analyzer that one can use to plot the individual signals used in the FPGA design, and examine the state machine. Because an FPGA is hardware, it is not possible to simply print out text onto the screen, or use a debugger to step through the code running on a processor.

6.1.2 In comparison to software development

With software development, there are numerous places to find code to explore and understand the internal workings. Sources such as Software Development Kit (SDK), example codes and open-source software. On the hardware side, designing for FPGA, most designs are released as IP cores, but the actual source code is hidden through licensing. This limits the scope of debugging and testing available, since it is not possible probe deep into the design to determine what parts work, but must instead assume that the IP cores are fully working. There is some hardware code available as open source through "Open Cores", but a) their completion is questionable b) debugging would require reading and understanding their source code in its entirety, and c) they are built to use an open interface, such as the Wishbone interface (not the Avalon interface), so integration into an existing setup is complicated.

As the development begins on the FPGA hardware description, it is necessary to perform synthesis and compile the design. Unfortunately, with the software used for synthesis and compilation, the entire design has to be synthesized from scratch each time, unlike software where only the changes and parts affected gets recompiled. Compilation and synthesis of a large System on Chip design takes much longer compared to simple hardware based designs. Compilation of the GPS SDR design takes approximately 15 minutes each time on a 1.8 GHz laptop. Compared to the simple hardware-based design which takes approximately 1 minute, and software where it takes seconds to compile new changes.

This long synthesis and compilation time affects the approach used for testing. For example, changing the data to be monitored in the in-system logic analyzer program requires full compilation, which involves synthesis, compilation, routing and programming the design onto an FPGA. Due to the time it takes to test each change, methodologies such as Test Driven Development commonly used in software development cannot be applied to FPGA based designs. Other features found in software development, such error codes, tracing and exceptions, are non-existent in hardware development, making hardware designs more complicated to debug. Functional unit testing of FPGA designs are not easily run from the FPGA development software, but are run separately, using a separate program called ModelSim, which simulates the hardware. The lack of testing tool integration in the CAD tools increases the time it takes to test hardware designs, as a separate project file has to be maintained for testing, and software simulations take much longer than real-time to run, especially for a SoC design, which might incorporate hardware and software designs.

One software development practise that can and should be applied is version control. A version control system keeps track of the changes in source code files and allows for figuring out what was changed, when, and by whom. It also allows for collaboration, as different people can work on the same code and merge their changes. Common and free version control systems are; SVN, git or Mercurial. They should be integrated into the hardware description process, even though the software tools used to create FPGA designs do not natively support them.

6.2 Contribution

The goal of this research was to demonstrate the feasibility of using an FPGA on a Nanosatellite through three different designs.

The first design demonstrated a novel use for an FPGA. The second design demonstrated the processing capability of an FPGA. The third design demonstrated the high-speed interface of the FPGA, allowing for a use of a slower clocked microcontroller to save power.

1. A WDT IP for use on an FPGA was developed to monitor and reset an external microcontroller. The design consists of a soft-core processor, SPI interface and custom hardware written in Verilog to monitor the state of the I/O pins of the microcontroller. The complete design ensures that during the reset of the microcontroller, components controlled by the I/O pins do not see glitches in the control lines as the microcontroller goes through its power on procedure. This design demonstrates the use of FPGAs in non-mission critical designs, showing how an FPGA can be used to improve an existing idea.

2. An Avalon bus compatible data acquisition IP core was written in Verilog to capture 1-bit I and Q from an external GPS RF front end board. The design demonstrated the transfer of data across two different clock domains and the ability to present the data to a software running on a processor through the use of shared memory.

3. An Avalon bus compatible GPS satellite signal acquisition IP core was developed using Verilog. The design implements a hardware based satellite acquisition module consisting of 8 channels running in parallel, along with other required components such as 1-bit NCOs, CA code generator. The design was optimized specifically for an FPGA by using an XNOR gate to implement 1-bit multiplication in place of a

dedicated integer multiplier, along with implementing a simplified 1-bit NCO that uses only a counter and logic gates.

4. A serial-search-based GPS satellite signal acquisition method was examined by developing a software implementation of the alogorithm. The software was written using Objective-C and results plotted using CorePlot, which allowed for verification of the data visually by comparing with known data. The verified software allowed for the design of the hardware to be verified using the numerical output of the software implementation.

5. A baseband processor was designed for processing of GPS signals on an FPGA. The processor uses a SoC design to incorporate hardware IPs, e.g., hardware based FFT and the two Avalon peripherals mentioned previous, along with a soft-core processor on a single FPGA chip. This allows for a single FPGA to process data using both hardware and software.

6. A high-speed camera interface was designed for use on an FPGA using Verilog. The design demonstrated the need for FPGAs when interfacing between the high-speed interface of camera to a low-speed microprocessor. The use of the FPGA in capturing the high-speed data allowed for the signals to be analyzed and the camera protocol to be examined.

6.3 Future Work

Further work will need to be done to improve the designs, by incorporating fault tolerant design techniques to reduce the affect radiation might have on the design used on SRAM-based FPGAs. Improving the designs can be done either by investigating the use of radiation tolerant FPGA devices, or by implementing error detection and mitigation techniques such as TMR. The functionality of the individual designs can also be improved, i.e. creating an FPGA based MPPT controller, GPS signal tracking and decoding, or image processing and encoding.

Chapter 7

References

[1] E. Buchen and D. DePasquale, "2014 Nano / Microsatellite Market Assessment," SpaceWorks Enterprises, Inc. (SEI), 2014. [Online]. Available: http://www.sei.aero/eng/papers/uploads/archive/SpaceWorks_Nano_Microsatellite_Market_Assessment_January_2014.pdf

[2] "CubeSat Design Specification Rev. 12," The CubeSat Program, Cal Poly SLO, 2009. [Online]. Available: http://cubesat.org/images/developers/cds_rev12.pdf

[3] H. Heidt, J. Puig-Suari, A. S. Moore, S. Nakasuka, and R. J. Twiggs, "CubeSat: A new Generation of Picosatellite for Education and Industry Low-Cost Space Experimentation," 14 14th Annual/USU Conference on Small Satellites.

[4] I. Andrew E. Kalman, Ph.D. President of Pumpkin, "Hardware and Software Design of an MSP430-based Satellite using an RTOS," Pumpkin, Inc., 2004. [Online]. Available: http://www.cubesatkit.com/docs/press/pumpkin_MSP430_ATC2004.pdf

[5] F. L. Kastensmidt, L. Carro, and R. Reis, *Fault-Tolerance Techniques for SRAM-based FPGAs*. Springer, 2006.

[6] "NanoCom Spacelink Modules," GomSpace ApS, 2014. [Online]. Available: http://gomspace.com/index.php?p=products-u482c

[7] "CMX469A - 1200/2400/4800 Baud FFSK/MSK Modem," Datasheet, Consumer Microcircuits Limited, May 2001.

[8] "FPGA processors keep Mars Rovers moving," Military & Aerospace Electronics, 2005. [Online]. Available: http://www.militaryaerospace.com/articles/2005/01/fpga-processors-keep-mars-rovers-moving.html

[9] D. Ratter, "FPGAs on Mars," *Xcell Journal*, vol. 50, pp. 8–11, 2004. [Online]. Available: http://www.xilinx.com/publications/archives/xcell/Xcell50.pdf

[10] "Actel FPGAs in Mars Rover," Penton Electronics Group, 2007. [Online]. Available: http://electronicdesign.com/energy/actel-fpgas-mars-rover

[11] G. Leopold, "How Mars rover got its 'dream mode'," EE Times, 2012. [Online]. Available: http://www.eetimes.com/document.asp?doc_id=1262350

[12] D. Guzmán, D. E. Rowland, P. Uribe, and T. Nieves, "A Low Power Processor for CubeSat Missions," NASA Goddard Space Flight Center, 2011. [Online]. Available: http://mstl.atl.calpoly.edu/~bklofas/Presentations/DevelopersWorkshop2011/41_Guzman_Processor.pdf

[13] "The LEON2-FT Processor User's Manual," European Space Agency, January 2014. [Online]. Available: https://amstel.estec.esa.int/tecedm/ipcores/leon2ft_2013.2.pdf

[14] D. L. Bekker, T. A. Werne, K. Dontchev, M. Heywood, R. Ramos, B. F. T. O. Wilson, P. J. Pingree, F. Saca, B. Gilchrist, A. Gallimore, and J. Cutler, "A CubeSat Design to

Validate the Virtex-5 FPGA for Spaceborne Image Processing," *Aerospace Conference, 2010 IEEE*, 2010.

[15] "COSMIAC CubeSat Reconfigurable Board," COSMIAC, University of New Mexico, 2010. [Online]. Available: http://www.cosmiac.org/formfactor.html

[16] "Space Plug-and-Play Architecture (SPA) Standard - System Capabilities," standard, American Institute of Aeronautics and Astronautics, February 2011. [Online]. Available: http://aiaa.kavi.com/public/pub_rev/SPA_G-133-10-201X_PR.pdf

[17] S. J. Olivieri, "Modular FPGA-Based Software Defined Radio for CubeSats," Master's thesis, Worcester Polytechnic Institute, May 2011.

[18] "USRP1 Bus Series," Datasheet, Ettus Research / National Instruments, September 2012. [Online]. Available: https://www.ettus.com/content/files/07495_Ettus_USRP1_DS_Flyer_HR.pdf

[19] "GNU Radio," Website. [Online]. Available: http://gnuradio.org/redmine/projects/gnuradio

[20] "CubeSat EPS Board, TI EPS-3," Tiger Innovations. [Online]. Available: http://server2.tigerinnovations.com/homepage/products/TigerInnovations_EPSProductSheet.pdf

[21] "FPGAs," Altera Corporation, 2014. [Online]. Available: http://www.altera.com/products/fpga.html

[22] "Field Programmable Gate Array (FPGA)," Xilinx Corporation, 2014. [Online]. Available: http://www.xilinx.com/training/fpga/fpga-field-programmable-gate-array.htm

[23] "Core8051," Datasheet, Actel Corporation, December 2005. [Online]. Available: http://www.actel.com/ipdocs/Core8051_DS.pdf

[24] "OpenSPARC T1 Microarchitecture Specification," Datasheet, Sun Microsystems, Inc., April 2007. [Online]. Available: http://www.oracle.com/technetwork/systems/opensparc/t1-01-opensparct1-micro-arch-1538959.html

[25] Ian Kuon and Jonathan Rose, *Quantifying and Exploring the Gap Between FPGAs and ASICs.* Springer, 2009.

[26] "Quartus II Introduction Using Verilog Designs," Altera Corporation, January 2011. [Online]. Available: ftp://ftp.altera.com/up/pub/Altera_Material/10.1/Tutorials/Verilog/Quartus_II_Introduction.pdf

[27] "ModelSim - Leading Simulation and Debugging," Website, Mentor Graphics, 2014. [Online]. Available: http://www.mentor.com/products/fpga/model

[28] "Cyclone V FPGAs," Altera Corporation, 2014. [Online]. Available: http://www.altera.com/devices/fpga/cyclone-v-fpgas/cyv-index.jsp

[29] "ProASIC3 Flash Family FPGAs," Microsemi Corporation, 2013. [Online]. Available: http://www.actel.com/documents/PA3_DS.pdf

[30] "Earth Observing 1 (EO-1)," Website, NASA Goddard Space Flight Center, 2014. [Online]. Available: http://earthobservatory.nasa.gov/Features/EO1/eo1_3.php

[31] "Mongoose-V MIPS R3000 Rad-Hard Processor," Website, Synova Inc., 2008. [Online]. Available: http://synova.com/proc/mg5.html

[32] "Fabrication Pricing," Canadian Microelectronics Corporation, 2014. [Online]. Available: http://www.cmc.ca/WhatWeOffer/Make/FabPricing.aspx

[33] "Cyclone II Power Comparison," Altera Corporation, 2014. [Online]. Available: http://www.altera.com/devices/fpga/cyclone2/features/power/cy2-power-compare.html

[34] I. Kuon and J. Rose, "Measuring the Gap Between FPGAs and ASICs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, February 2007.

[35] T. Huffmire, C. Irvine, T. D. Nguyen, T. Levin, R. Kastner, and T. Sherwood, *Handbook of FPGA Design Security.* Springer, 2010.

[36] "Cyclone IV Device Handbook," Altera Corporation, 2014. [Online]. Available: http://www.altera.com/literature/lit-cyclone-iv.jsp

[37] "Cyclone IV FPGA Family Overview," Altera Corporation, 2014. [Online]. Available: http://www.altera.com/devices/fpga/cyclone-iv/overview/cyiv-overview.html

[38] K. Morris, "FPGAs in Space - Programmable Logic in Orbit," *Electronic Engineering Journal*, 2004. [Online]. Available: http://www.eejournal.com/archives/articles/20040803_space

[39] J. Wang, R. Katz, J. Sun, B. Cronquist, J. McCollum, T. Speers, and W. Plants, "SRAM Based Re-programmable FPGA for Space Applications," *IEEE Transactions on Nuclear Science*, vol. 46, no. 6, 1999.

[40] T. Speers, J. J. Wang, B. Cronquist, J. McCollum, H. Tseng, R. Katz, and I. Kleyner, "0.25 m FLASH Memory Based FPGA for Space Applications," Actel Corporation, NASA/Goddard Space Flight Center, Orbital Sciences Corporation. [Online]. Available: http://www.actel.kr/_actel/html/digital.library/q1_2003/PDFs/25umflash_mem_article.pdf

[41] "Embedded - FPGAs (Field Programmable Gate Array)," Digi-Key Corporation, January 2014. [Online]. Available: http://www.digikey.ca/product-search/en/integrated-circuits-ics/embedded-fpgas-field-programmable-gate-array/2556262

[42] E. Grayver, *Implementing Software Defined Radio.* Springer, 2013.

[43] "International Traffic In Arms Regulations PART 121-THE UNITED STATES MUNITIONS LIST," Federation of American Scientists. [Online]. Available: http://www.fas.org/spp/starwars/offdocs/itar/p121.htm#C-XV

[44] "Selecting a USRP Device," Application Note, Ettus Research. [Online]. Available: http://www.ettus.com/content/files/kb/application_note_selecting_a_usrp.pdf

[45] "Stereo," Datasheet, Nottingham Scientific Ltd. [Online]. Available: http://www.nsl.eu.com/datasheets/stereo.pdf

[46] "MAX2769 Evaluation Kit," Datasheet, Maxim Integrated, September 2007. [Online]. Available: http://datasheets.maximintegrated.com/en/ds/MAX2769EVKIT.pdf

[47] "Nios II Processor Reference Handbook," Altera Corporation, February 2014. [Online]. Available: http://www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf

[48] "MC8051 IP Core - Oregano Systems 8-bit Microcontroller IP-Core," Datasheet, Oregano Systems – Design & Consulting GesmbH, June 2013. [Online]. Available: http://www.oreganosystems.at/download/mc8051_overview.pdf

[49] "lib2-mul.c," Source Code, Free Software Foundation, Inc., 2014. [Online]. Available: https://gnu.googlesource.com/gcc/+/trunk/libgcc/config/nios2/lib2-mul.c

[50] "IEEE Standard for Floating-Point Arithmetic," IEEE Standard 754, 2008.

[51] "Hardware Acceleration," Website, Altera Corporation, 2014. [Online]. Available: http://www.altera.com/devices/processor/nios2/benefits/performance/ni2-acceleration.html

[52] "Using Nios II Floating-Point Custom Instructions Tutorial," Altera Corporation, 2010. [Online]. Available: http://www.altera.com/literature/tt/tt_floating_point_custom_instructions.pdf

[53] "Intel® Core i5-3400 Mobile Processor Series," Datasheet, Intel Corporation, July 2013. [Online]. Available: http://download.intel.com/support/processors/corei5/sb/core_i5-3400_m.pdf

[54] F. Storm, "bel_fft - FFT co-processor in Verilog based on the KISS FFT," Website. [Online]. Available: http://sourceforge.net/projects/belfft/

[55] "DMA Controller," Altera Corporation, 2014. [Online]. Available: http://www.altera.com/products/ip/iup/memory/m-eur-dma-cont.html?GSA_pos=5&WT.oss_r=1&WT.oss=dma%20controller

[56] P. Milder, "Spiral Project: DFT/FFT IP Core Generator," Website, Carnegie Mellon University, 2012. [Online]. Available: http://www.spiral.net/hardware/dftgen.html

[57] "SGR-20 - Space GPS Receiver," Website, Surrey Satellite Technology US LLC, 2014. [Online]. Available: http://www.sst-us.com/shop/satellite-subsystems/gps/sgr-20-space-gps-receiver-navigation-and-timing

[58] "IP-NCO - Altera. Order from Arrow Electronics," Website, Arrow Electronics, Inc., 2014. [Online]. Available: http://parts.arrow.com/item/detail/altera/ip-nco#pcRQ

[59] "Serial Digital Interface Reference Design for Cyclone & Stratix Devices," Application Note, Altera Corporation, August 2004. [Online]. Available: http://www.altera.com.cn/literature/an/an356.pdf

[60] "SPV1020 - Interleaved DC-DC boost converter with built-in MPPT algorithm," STMicroelectronics, 2013. [Online]. Available: http://www.st.com/web/catalog/

sense_power/FM142/CL1810/SC1517/PF250769

[61] "Design of Hardware and Software for the Powersupply for AAU Cubesat," Aalborg University, November 2002. [Online]. Available: http://www.space.aau.dk/cubesat/documents/psu/new_psu.pdf

[62] "Clyde space 1u cubesat eps," Clyde Space. [Online]. Available: http://www.clyde-space.com/cubesat_shop/eps/8_1u-cubesat-eps

[63] C. S. Clark and A. L. Mazarias, "Power System Challenges for Small Satellite Missions," Clyde Space Ltd. [Online]. Available: http://www.clyde-space.com/documents/1502

[64] A. Elbrecht, S. Dech, and A. Gottscheber, "1U CubeSat Design for increased Power Generation," SRH University of Applied Sciences Heidelberg, 2011. [Online]. Available: http://www.hochschule-heidelberg.de/fileadmin/srh/heidelberg/pdfs/fakultaeten/engineering/Master_InformationTechnology/projekte/heidelsat/DesignForHeidelSat1U.pdf

Appendices

Appendix A

CubeSat Power Board Design

The EPS board consists of three major components: power converters, chargers, and a controller, which in this case is the microcontroller and FPGA. The connection between these components can be seen in Figure A.1. This particular EPS design uses a single lithium polymer battery cell to store energy. This gives the battery bus a nominal voltage of 3.7 V. The voltage of the battery bus influences the selection of the other ICs.

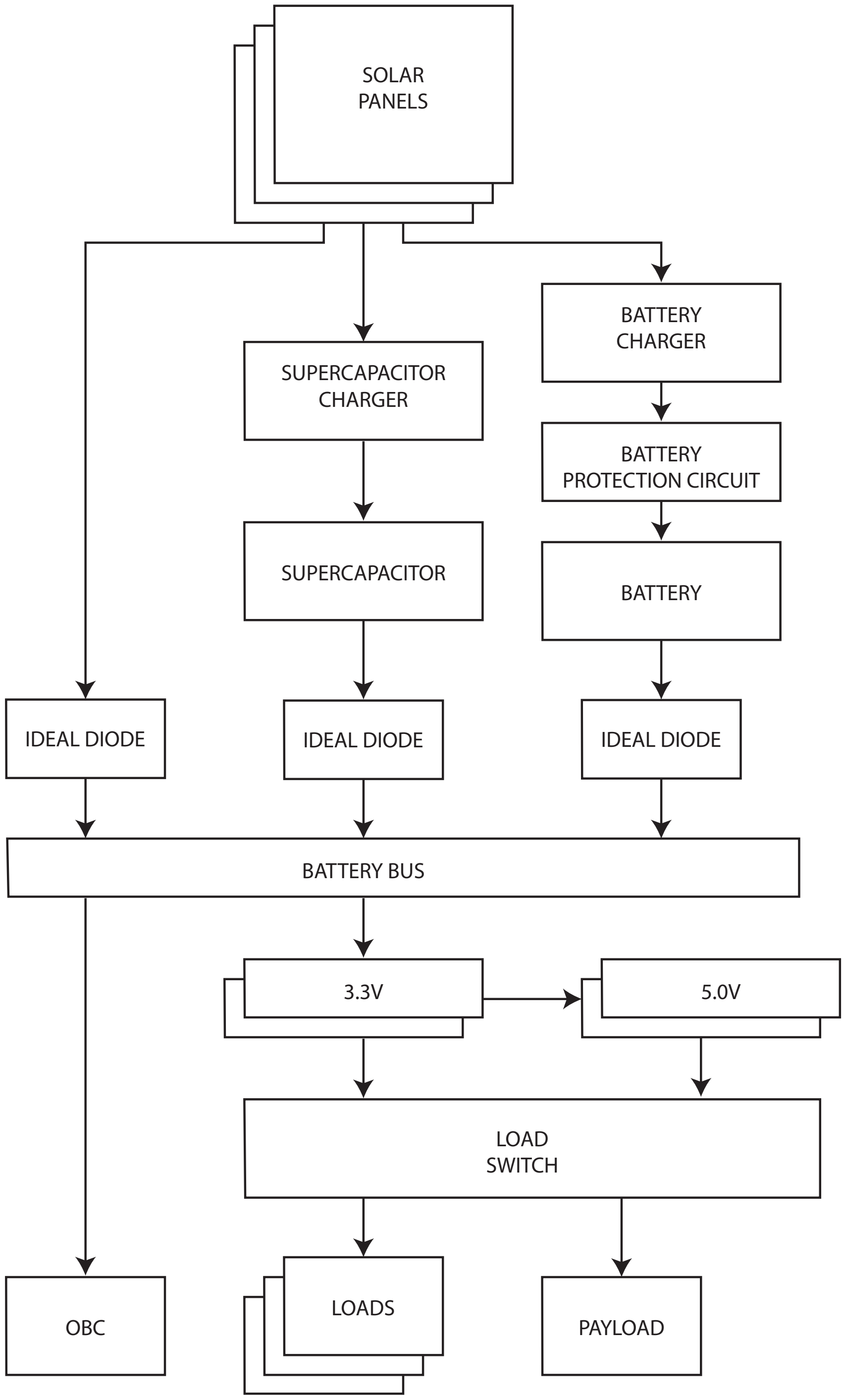


Figure A.1: Overview of power board components and connections

Direct Energy Transfer (DET), and MPPT are two approaches to harnessing the power from the solar panels, and lead to two ways of connecting the solar panels to the power subsystem. With DET, the batteries are connected to the solar panels. DET is the simpler approach, as it does not require additional circuitry. With MPPT, there is either a boost or buck converter in-between the solar panels and batteries. MPPT allows the solar panels to work at the at the voltage and current levels that produces the most amount of power. The design of this particular power board is laid out using the DET topology, but implements MPPT by varying the charging currents of the battery and super capacitor chargers, instead of using a boost or buck converter in between the solar panels and EPS.

There are specialized MPPT ICs available, such as the STMicroelectronics SPV1020 [60]. The design is a boost converter which operates on an input voltage range of 6-40 V. Due to low nominal voltage of the EPS design, using the SPV1020 is not a viable solution.

MPPT has been implemented on other CubeSats before, one example being the AAU CubeSat from Aalborg University in Denmark [61]. Their design uses a boost converter circuit with discrete components to implement the MPPT functionality. The control of the MPPT boost circuit gate driver is done through the OBC and a backup 555 timer IC.

CyldeSpace seems to also use a discrete DC-DC converter circuit, as multiple DC-DC converters ICs cannot be seen based on images of their power subsystem board [62]. Based on the hardware described in their paper [63], appears that they have developed a custom circuit for MPPT.

A paper from the SRH University of Applied Sciences Heidelberg for the HeidelSat power subsystem [64] mentions the use of an unspecified boost converter and digital potentiometer on the feedback pin, to boost the voltage from 4.75 V to 9.9 V.

The the design of the EPS board in this research implements a different approach for MPPT but uses a digital potentiometer, similar to HeidelSat, to control the current drawn to charge the battery and super-capacitor.

There are three different types of switch mode circuit topologies that are used to convert DC power on this EPS board: boost converters, buck converters, and charge pumps. Buck converters are used for 3.3 V, buck/boost converters are used by the super-capacitor charger, and a charge pump provides the 5 V.

The buck and boost converters are able to provide a decent amount of current, such as the 1-2 A available from the buck converters used in this design. The downside is that along with capacitors, they require inductors, which introduce additional mass, increased space requirement and noise. The charge pump is a simple circuit that only requires capacitors in addition to the IC. They do not have the side effects of using inductors. The disadvantage is that they provide much less current, roughly 100-200 mA per charge pump.

The buck converter and charge pump ICs were selected for their ability to work in parallel, e.g., multiple buck converters can be wired in parallel, sharing a common clock reference. The size and mass of the IC were minimized by compromising on the switching frequency. High-speed switching converter ICs are smaller and require smaller capacitor and inductors, with the trade off being the converter efficiency.

The power board features two super-capacitors, which act as a second form of energy storage alongside the battery. The low internal resistance of the super-capacitors in comparison to the battery means that the super-capacitors can provide large amounts of surge current when needed. The low internal resistance also means that the super-capacitors can be charged rapidly, which decreases the amount of time the voltage of the battery bus takes to get to the nominal voltage.

The three sources of power: battery, super-capacitor and solar panels, are connected through ORing diodes, which isolate the sources, preventing one source from directly charging another. Standard diodes have a voltage drop of approximately 0.7 V, meaning the 4.2 V from a fully charged battery will be seen as 3.5 V, reducing the amount of usable power, e.g., the 3.3 V buck converter only works when the bus voltage is greater than 3.3 V. This is solved through the use of ideal diodes, which reduce the voltage drop to less than 100 mV.

The power board uses a four layer PCB in its design. The two outer layers are for signal traces. The two inner traces are the ground plane, and different supply voltages. Having the two inner layers, the ground plane and supply voltages, in close proximity maximizes the capacitance between the supply and ground, increasing the capacitance on the power lines, which is beneficial for filtering noise. The two main components, the microcontroller and FPGA, were selected based on the availability of non-BGA packages, which reduced the routing complexity and number of layers required.

The design of the PCB can be seen in Figure A.2. The design of the power board layout can be divided into 4 quadrants:

- Top-left: This area contains the supercapacitor charger circuit with the supercapacitors occupying the bottom side of the PCB.
- Top-right: This area contains the two 3.3V buck converters and the two 5v charge pumps.
- Bottom-right: This area contains the microcontroller and the FPGA, along with the voltage regulators and oscillator required for the FPGA to function. Also present along the bottom edge are the connections to the microcontroller and FPGA - the JTAG connector and UART connector.

- Bottom-left: This area contains the battery protection circuitry and battery charger circuit.

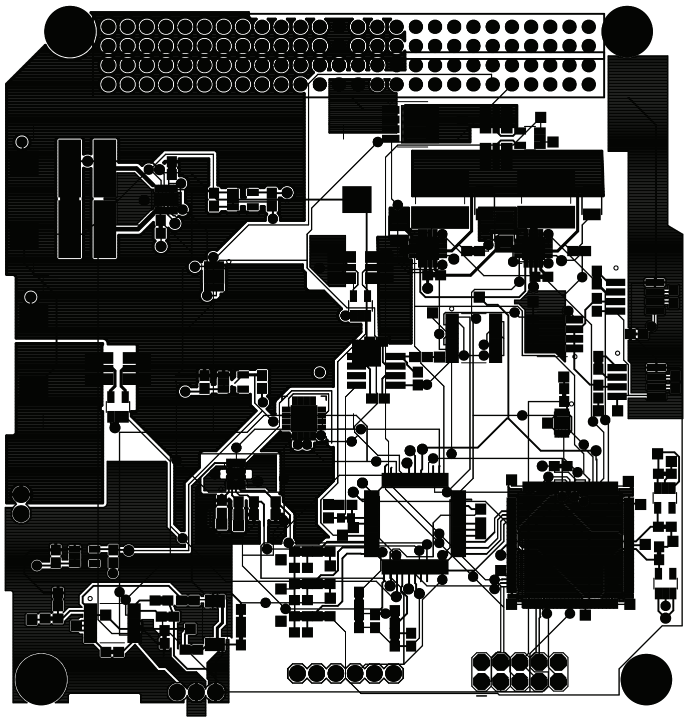


Figure A.2: The PCB layout of the power board

A real-time operating system (RTOS) is used on the Atmel AVR microcontroller. The use of a RTOS allows for execution of multiple tasks simultaneously, e.g., control of various ICs while reading ADC values. The RTOS does however add additional overhead in terms of memory requirements. To handle the overhead, the microprocessor selected contains 128 KiB of memory.

Functional tests of the power board was done using an electronic load. The electronic load allows the current drawn from the power board to be set. In comparison with a load resistor, whose current draw varies as the voltage varies - a constant resistive load, an electronic load can be set to either settable constant voltage load, resistive load, and in our case, a current load.

The power board was tested in a simulated space environment by using a vacuum chamber. The power board survived the simulated space environment. The development of the FPGA design and microcontroller firmware was done after the test using the tested power board, and no electrical nor physical damage was encountered.

Power is currently being wasted as it passes through the voltage dividers used to scale the voltage for the ADC. There are 5 voltage dividers, which are always connected. A way around the leakage currents would be to use the CMOS analog switch to selectively apply power to the voltage dividers - the switch would switch between “open circuit”, bus voltage, battery voltage, and the output voltages. Switching to the “open circuit” would prevent energy from being wasted when a voltage is not being measured.

Currently the control signals to the ideal diodes are not being used. Controlling the state of the ideal diodes would allow the connection from the battery, super capacitors, and input voltage to be individually controlled, preventing the discharge of either the battery or super capacitors when they are not needed.