John-Philip Taylor

Department of Electrical Engineering,
Menzies Building, University of Cape Town
Cape Town, South Africa 7701
Tel: +27 82 354 6741
eMail: tyljoh010@myuct.ac.za
Internet: http://www.uct.ac.za

FPGA Development for Radar, Radio-Astronomy and Communications

Contents

1 General Information

1.1 Course Description

This course presents the principles and techniques fundamental to low-level FPGA firmware development. It is biased towards digital signal processing typically found in Radar, Radio-astronomy and Communication systems.

Due to the global chip shortage, the availability of Altera and Xilinx based development kits are limited. As a consequence, this course will use the LatticeXP2 Brevia2 development kit and associated development environment. Altera and Xilinx tools are similar, so, after completing this course the participant will have enough background to make use of the Altera or Xilinx tool-set with minimal effort.

Embedded soft-core processors and SoC systems are not included in this course. Furthermore, this course is aimed at low level development: high level synthesis is not covered.

1.2 Desirable Experience

This course assumes that participants have:

- A basic conceptual understanding of digital systems and architectures, such as:
  - Memory hierarchies and cache systems
  - Computational pipelines and streaming processors
  - Finite state machines
- A conceptual understanding of digital signal processing, such as:
  - Laplace and Z transforms
  - Fourier transforms and the FFT
  - FIR filters and other correlation-based functions
  - IIR filters and other Z transform based difference equations
  - The effects of RF mixing, windowing, etc.
- Experience in using scientific tools, such as Matlab, Octave or Python. In particular:
  - Generating figures from data
  - DSP functions (windows, FFTs, vectorised arithmetic, etc.)
  - Reading and writing binary, CSV and ASCII-based files
- Programming experience, in any language (although C / C++ experience will prove beneficial)

1.3 Course Format and Dates

The preliminary dates for the course are given below. For final dates, refer to http://www.radarmasters.uct.ac.za/.

Lectures (EEE5117Z): 27 to 29 April 2022
Lectures (EEE5118Z): 2 to 4 May 2022
Examination: 26 July 2022

The course is split over two parts. EEE5117Z (FPGA Basics) presents the FPGA internal architecture and the basics required to program FPGAs. EEE5118Z (Advanced FPGA Technologies) presents system level design concepts, standard building blocks such as finite state machines and pipelines, and more advanced debugging techniques.

Each part is presented over 3 days. Each day consists of morning lectures from 09:00 to 13:00 (with a half-hour tea-break at 11:00) and an afternoon tutorial / practical from 14:00 to 17:30 (see sections 3 and 4.1 for details).

The afternoon sessions are semi-formal, in the sense that the lecturer will drive the activities and be available for questions, but the participants can structure the time as they deem appropriate.

During the time between the lecture week and the examination week, participants of the more advanced EEE5118Z part are expected to complete a project (see section 4.2 for details). Evaluation is based on a short project report and associated source code. EEE5117Z is evaluated by means of a written examination.

1.4 Staff

Role	**Name**	**Affiliation**	**eMail**
Convenor:	Dr Stephen Paine	UCT	stephen.paine@uct.ac.za
Lecturer:	John-Philip Taylor	UCT	tyljoh010@myuct.ac.za

1.5 Course Load

EEE5117Z Lectures	3 days of 3.5 hours each	⇒	10.5 hours
EEE5117Z Tutorials	3 days of 3.5 hours each	⇒	10.5 hours
EEE5117Z Examination	Preparation	⇒	76.0 hours
EEE5117Z Examination	Writing	⇒	3.0 hours
EEE5118Z Lectures	3 days of 3.5 hours each	⇒	10.5 hours
EEE5118Z Tutorials	3 days of 3.5 hours each	⇒	10.5 hours
EEE5118Z Project	4 weeks of 16 hours each	⇒	64.0 hours
EEE5118Z Report	Preparation	⇒	15.0 hours
Total		⇒	200.0 hours

1.6 Available Hardware

For the duration of the course, participants are provided with a laboratory station that includes an oscilloscope, signal generator, bench power supply and computer.

The course fee further includes a LatticeXP2 Brevia2 development kit. The participant keeps the board after the course.

Even though participants are encouraged to work on their own computer / laptop, the laboratory computers will have Python and Lattice Diamond installed.

2 Learning Outcomes

Having successfully completed both parts of this course, participants should be able to:

2.1 Knowledge Base

- Understand the underlying physical architecture of FPGAs;
- Understand the concept of timing constraints, clock domains and other timing-related issues;
- Use an FPGA tool-set, including JTAG debugging and the general Verilog-based compilation process;

2.2 Engineering Ability

- Design FPGA firmware systems on a high level;
- Design FPGA firmware blocks on a low level (i.e. RTL representations of finite state machines and pipelines);

2.3 Practical Skills

- Implement FPGA firmware systems;
- Debug an FPGA firmware implementation;
- Analise timing closure issues and solve the problem such that the final design meets all timing requirements.

3 Lecture Programme

Lectures are split over six days: three days for each of the two parts. The general trend is presented below and more detail is provided in subsequent subsections. This schedule is preliminary and subject to change.

Day 1: Comparisons between CPU, GPU and FPGA based high-performance computing.

Details of FPGA internal structure, as well as the tool-set: simulation; Verilog-based design-entry; pin-assignment; compile chain; etc.

Day 2: Library-provided building-blocks, and how to interface to them from within the HDL-based environment

Finite state machines;

JTAG interfaces; on-chip logic analyser;

Setting pin parameters (including pin termination, calibration and drive-strength parameters)

Day 3: Memory-mapped bus structures

Processing pipelines

Introduction to timing constraints and clock domains

Register-based FPGA control over UART

Data injection and pulse-width modulation

Day 4: Clock-domain crossing methods

Multi-cycle and external timing constraints

Simulation-based verification of DSP modules

Day 5: Arbitration and mutually exclusive access

Flow control and credit-based streams

Caching systems

Data injection based testing

Day 6: FPGA-based DSP, especially numerically controlled oscillators and digital filters.

Building a DSP-chain

3.1 CPU vs GPU vs FPGA

The pros and cons of each platform is presented, with examples to illustrate the strengths and weaknesses.

This section also includes a brief overview of the programming model of each platform.

3.2 Field Programmable Gate Arrays

The internal structure of FPGAs, with particular attention given to the detail of logical elements, registers, RAM blocks and DSP elements.

This section further includes a brief introduction to hardened IP blocks, such as SerDes I/O, PCIe controllers, SDRAM interfaces and embedded processors.

3.3 Development Tools

FPGA development tools have many aspects. This course will include:

3.3.1 Hardware Description Language

The Verilog (and System Verilog) HDL, and how to use it with the compile chain.

3.3.2 JTAG Interfaces

This section includes:

- Loading the bit-stream onto the FPGA
- On-chip logic analyser
- Other JTAG-based debugging, as applicable

3.4 Finite State Machines

Traditional finite state machines are implemented by means of a large case statement, but there are also other ways. This section details how RTL (register transfer level) code relates to what is actually happening at the register level, with particular attention to detail such as reset schemes and clock-enable signals, as well as methods by which to implement a finite state machine.

3.5 Simulation and Test-benches

This section details how to write test-benches, including injecting data from files, and logging results to files. The Modelsim simulation tool will be used to simulate these test-benches.

3.6 Pipelines

Simple pipeline structures are presented, where the pipeline itself is simply a combinational circuit with registers inserted into the calculation path.

This concept is then expanded to include more complicated pipelines, where the incoming data rate is lower than the clock frequency, thereby enabling resource sharing between the stages, among other details.

3.7 Memory-mapped Bus Structures

The concept of a memory-mapped bus is presented, where the registers of various modules are mapped to the address-space of the bus. The bus is then controlled by a master. This section details how to implement such structures.

3.8 Streams and Queues

This section includes streaming processors, and how to interface between them by means of streams, queues and packets. It further explains how to synchronise various stages of the processor in the case where the different stages have unknown, or variable, latency and throughput.

An introduction to packet-based processing, and how to incorporate this into a streaming processor, is also provided.

3.9 Clock Domains

This section covers the various means by which to generate clocks, as well as the pro's and con's of each. It further explains the need to separate clock domains, and how to go about crossing data (or control signals) from one domain to the other. Methods included in this course are:

- Register-chain
- Crossing counter data by means of Gray coding
- FIFO queues
- Hand-shaking

3.10 Timing Constraints

The concept of timing requirements is explained by means of various scenarios, detailing the effects of setup and hold requirements, as well as clock skew and propagation delays.

Most FPGA vendors make use of the Synopsis Design Constraints standard in order to specify timing requirements. An introduction to the standard is provided, with particular attention given to:

- Clock definitions
- Clock domain definitions
- Internal timing requirements (including multi-cycle)
- External timing requirements, including:
  - False paths (used for asynchronous I/O pins)
  - Input delays
  - Output delays

An introduction to the Timing Analyser is provided.

3.11 Arbitration and Mutually Exclusive Access

The difference between arbitration and mutual exclusion is highlighted, as well as how to implement both. Priority-based as well as round-robin based techniques are discussed.

3.12 FPGA-Based DSP

This section presents two of the more common DSP elements used in FPGA-based DSP: the numerically controlled oscillator (used for digital downconversion) and the FIR filter.

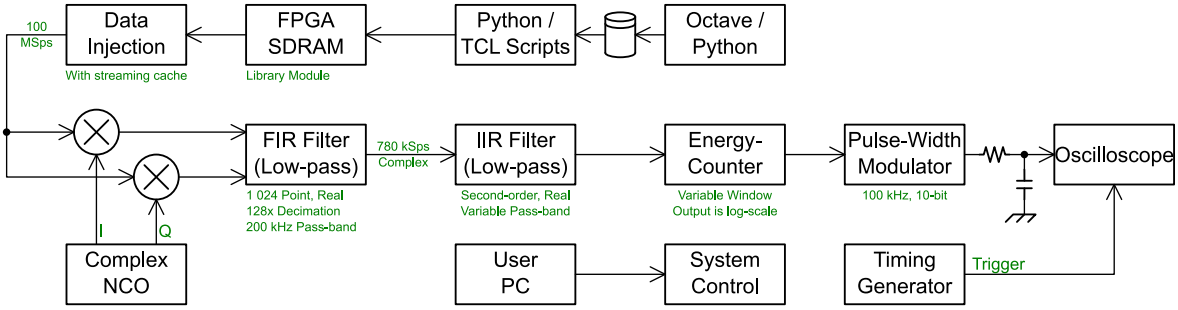
Often the developer must compromise between power usage, resource usage, latency and throughput. This section further discusses various means by which the design can use both pipeline and state-machine elements to make the best usage of the available physical resources and available clock-cycles, especially when the clock is faster than the data-rate.

4 Assignments

This course comprises two assignments. The first will take place during the six days of lectures, in the form of tutorials. The second is a larger project, to be implemented by the participants after the course, with evaluation during the examination week.

4.1 Tutorials

The tutorials aim to teach the practical aspects of firmware-design by means of a small project. The participants are expected to implement a digital spectrum analyser. The preliminary system block diagram is presented below, but is subject to change before commencement of the course.



4.1.1 Schedule

The intended tutorial schedule is presented below. This schedule is preliminary and subject to change.

Day 1: Introduction to the development tools: simulate and implement flashing LEDs by means of a counter; etc.

Day 2: Implement and simulate a UART module.

This practical further includes a guide on using the JTAG-based logic analyser.

Day 3: Implement a memory-mapped register control interface, which is controlled from the development PC.

Implement the pulse-width modulation and data injection modules. Data is injected slowly, so that the streaming cache is not required.

The injected data can potentially be a piece of music, so the participants are encouraged to bring earphones, which can be driven directly by the FPGA.

Day 4: Implement the streaming cache and complex mixer. The FIR and IIR filter combination is implemented as simple decimation-by-128 at this point (no filtering).

Day 5: Implement the IIR filter, energy counter and various control-related modules. The filter parameters can be modified on-the-fly by means of the control interface.

Day 6: Implement the FIR filter. This is the most challenging part of the project, so participants are encouraged to continue implementation after the course, in case they cannot get it to work on the day.

4.1.2 Design Philosophy

The later modules of the tutorial system will follow the following design philosophy:

1. Pen-and-paper design
2. Matlab / Octave / Python algorithm simulation
3. Verilog HDL implementation
4. Simulation and verification
5. Integration into the larger system

4.2 Project

During the weeks following the lectures, EEE5118Z course participants will be expected to implement some DSP-related system. Each participant will implement a different

project. A list of projects will be provided, from which the participants can choose which one they would like to implement.

Interesting projects are often too large to implement in the time provided, so some of the projects will be broken up into sub-systems, each thereby forming a project on its own. By means of data-injection and logging, each of these sub-systems can be implemented independently. The rest of the larger system can be emulated with Matlab / Octave / Python.

A preliminary list of projects are presented below:

- FMCW RADAR with multiple input channels, Doppler processing and angle extraction. This can be broken up into three sub-systems: before the corner-turn, after the corner-turn and angle-extraction.
- Chirped pulse RADAR with matched filter receiver and Doppler processing. This can be broken up into two sub-systems: before the corner-turn and after the corner-turn.
- FSK-based, bi-phase-coded communication channel. Both the transmitter and the receiver can be implemented by a single developer.
- 16-QAM communication channel. This can be broken up into three sub-systems: transmitter, raw receiver (clock-recovery and channel estimation) and decoder.
- Radio-astronomy receiver, the details of which are undecided at this point.

5 Course Assessment and Examination

The assessment of the first part of this course (i.e. EEE5117Z) is based entirely on a written examination, which covers the content of the first 3 days of lectures.

The assessment of the second part of this course (i.e. EEE5118Z) is based entirely on the project, which is sub-divided as follows:

Part	%
Quality of the implementation (source-code, etc.)	40
Project report	60