John-Philip Taylor

Room 7.03, 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.

Although the course focuses on Altera tools, Xilinx tools are similar. After completing this course, the participant will have enough background to make use of the Xilinx tool-set with minimal effort.

Embedded soft-core processors and SoC systems are not included in this course.

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 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://radarmasters.co.za/.

Lectures: 17 to 21 July 2017

Examination: 4 to 8 September 2017

The course lecture week consists of 5 days, each of which are broken into 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 are expected to complete a project (see section 4.2 for details).

The examination week will consist of project demonstrations and presentations, where the participants showcase their designs.

1.4 Staff

Role	**Name**	**Affiliation**	**eMail**
Convenor:	Prof Daniel O'Hagan	UCT	daniel.ohagan@uct.ac.za
Lecturer:	John-Philip Taylor	UCT	tyljoh010@myuct.ac.za
Tutor:	Stephen Paine	UCT	16stevetp@gmail.com
	Randy Cheng	UCT	pokai0519@gmail.com

1.5 Course Load

Lectures:	5 days of 3.5 hours each	⇒	17.5 hours
Tutorials:	5 days of 3.5 hours each	⇒	17.5 hours
Project:	6 weeks of 26 hours each	⇒	156.0 hours
Demonstration:	Preparation and presentation	⇒	9.0 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 DE10-Lite 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 Octave and Altera Quartus Prime Lite installed.

2 Learning Outcomes

Having successfully completed 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 the Altera tool-set, including Qsys, 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 five days. The general trend is presented below and more detail is provided in subsequent subsections.

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

Details of FPGA internal structure, as well as the Altera tool-set: simulation; JTAG interfaces; on-chip logic analyser; Verilog-based design-entry; compile chain; etc.

Day 2: Using Qsys to quickly develop systems based on library-provided building-blocks, and how to interface to the resulting system from within the HDL-based environment

Simple finite state machines

Assigning pins and setting pin parameters (including pin termination, calibration and drive-strength parameters)

Day 3: Overview of typical building blocks: finite state machines; pipelines; memory-mapped bus structures; streams; queues; etc.

Day 4: Internal timing constraints; external timing constraints; clock domains; clock-domain crossing methods; etc.

Day 5: Arbitration and mutually exclusive access

Advanced architectures: making the most of the available resources and clock-cycles

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 Altera Tools

The Altera 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 Altera compile chain.

3.3.2 JTAG Interfaces

This section includes:

- Loading the bit-stream onto the FPGA
- In-system sources and probes
- In-system memory content editor
- On-chip logic analyser
- Virtual JTAG interface and controlling it by means of TCL scripts

3.3.3 Qsys

A brief introduction to Qsys. Qsys, in this course, is used to implement an SDRAM controller, a JTAG to Avalon-MM bridge, an external Avalon-MM interface to the system and interconnects to connect it all together.

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 Altera 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.

Altera (as well as Xilinx Vivado, Achronix and other FPGA vendors) makes 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 Quartus 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 Advanced Architectures

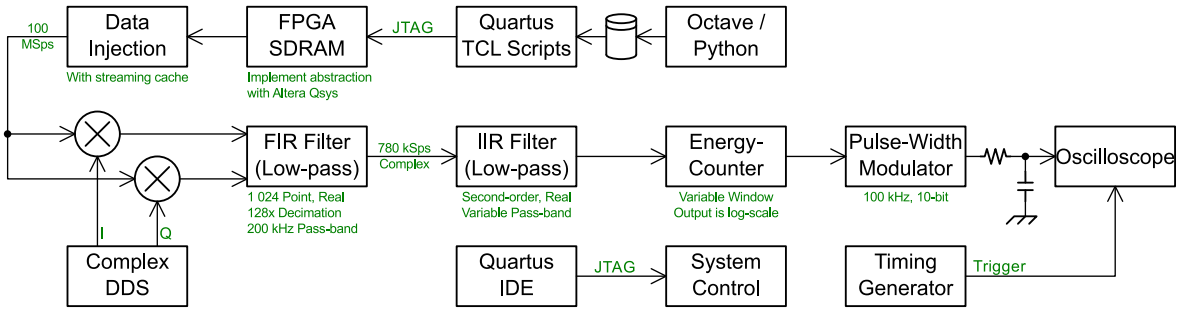
Often the developer must compromise between power usage, resource usage, latency and throughput. This section discusses various means by which the design can use both pipeline an state-machine elements to make the best usage of the available physical resources and available clock-cycles, especially when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

4 Assignments

This course comprises two assignments. The first will take place during the five days of lectures, in the form of tutorials. The second is a larger project, to be implemented by the participants over the course of six weeks.

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 as follows:

Day 1: Introduction to the Altera tools: simulate and implement flashing LEDs by means of a counter; read the buttons and write the LEDs by means of the JTAG interface; make use of the JTAG-based logic analyser; etc.

Day 2: Implement the pulse-width modulation and data injection modules. This involves an introduction to Altera Qsys (to implement the SDRAM controller). Data is injected slowly (at 125 kSps), 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 3: 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 4: Implement the IIR filter, energy counter and various control-related modules. The filter parameters can be modified on-the-fly by means of the JTAG interface.

Day 5: 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

Each module of the tutorial system will essentially follow the same design philosophy. The steps are as follows:

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 six weeks following the lecture week, 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 SDRAM-based 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 this course is based entirely on the project, which is sub-divided as follows:

Part	%
Successful demonstration	20
Presentation (eg. Power-Point; LaTeX Beamer)	20
Quality of the implementation (source-code, etc.)	20
Project report	40