John-Philip Taylor

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

FPGA Development for Radar, Radio-Astronomy and Communications

Contents

1 General Information **2**

2 Learning Outcomes **4**

3 Lecture Programme **5**

4 Assignments **11**

5 Course Assessment and Examination **14**

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. 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: 9 to 13 September 2024

Examination: 4 to 8 November 2024

The course lecture week consists fo 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). Evaluation is based on a short project report and associated source code.

Course notes and resources are hosted on GitHub:
https://github.com/jpt13653903/UCT-FPGA-Course-2024

1.4 Staff

Role	**Name**	**Affiliation**	**eMail**
Convenor:	Dr Stephen Paine	UCT	stephen.paine@uct.ac.za
Lecturer:	Dr John-Philip Taylor	RRS	john-philipt@reutech.co.za
Tutor:	William Bourn	UCT	BRNWIL012@myuct.ac.za

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
Report	Preparation	⇒	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, Python 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 Platform Designer, 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;
- Analyse 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. This schedule is preliminary and subject to change.

Day 1 (a): Introduction:

- Comparisons between CPU, GPU and FPGA based high-performance computing
- What is an FPGA and how does it work inside?
- What resources are available in the Max 10 FPGA?
- Typical design cycle and introduction to the tools: simulation, Verilog-based design entry, pin-assignment, compile chain, etc.
- Introduction to JTAG and debugging tools

Day 1 (b): Verilog detail:

- General code structure
- Combinational vs. clocked logic
- Reset strategy, and why a local reset is a good idea
- Semantics of a process: concept of "everything happens at the same time"
- Blocking vs. non-blocking statements
- Inferred latches
- Verilog operators
- Altera library

Day 2: Finite state machines, simulation and timing

- Traditional Mealy (combinational outputs) and Moore (registered outputs) machines
- SPI example to show why it is useful to deviate from the traditional formalised implementation style
- Basic simulation
- Synthesisable vs. non-synthesisable code
- Internal timing, and what influences it (routing delays, clock skew, large combinational circuitry, etc.)
- External timing, and the Synopsis Design Constraints industry standard
- How the IO standard can influence timing
- Using the PLL to gain timing closure on external devices (phase-shifted clocks)
- Controlling the circuit using a virtual JTAG interface
- Introduction to Platform Designer (formerly QSys)

Day 3: Bus architectures, clock details and pipelines

- Memory-mapped bus architecture basics
- Registers
- Clock generation techniques
- Clock domains, and how to cross data between them
- Simple pipelines
- Long processing chains, and the Avalon-ST data stream standard

Day 4: Arbitration and caching systems

- Injection and log based simulation
- Automated test-benches
- Mutual exclusion and arbitration
- Caching systems and cache coherency
- Running a pre-fetch read cache

Day 5: Digital signal processing

- Numerically controlled oscillator
- Pulse-width modulation
- IIR filter
- FIR filter
- Power spectrum meter

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 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
- In-system sources and probes
- In-system memory content editor
- On-chip logic analyser
- Virtual JTAG interface and using it to control the device

3.4 Platform Designer

A brief introduction to Platform Designer (formerly Qsys). Platform Designer, 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.

3.5 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.

3.6 Simulation and Test-benches

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

3.7 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.

3.8 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.

3.9 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.

3.10 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

3.11 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.

3.12 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.

3.13 FPGA-Based DSP

This section presents some common DSP elements used in FPGA-based DSP, such as the numerically controlled oscillator (used for digital downconversion), as well as IIR and FIR filters.

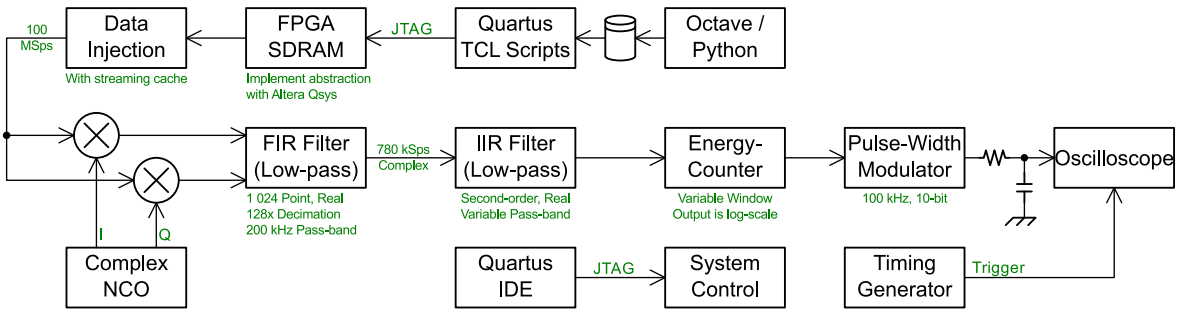
Often the developer must compromise between power usage, resource usage, latency and throughput. This section further discusses various means by which the design can use both pipeline and state-machine elements to make the best usage of the available physical resources and available clock-cycles, especially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

4 Assignments

This course comprises two assignments. The first will take place during the five 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 plan is presented below. It is challenging to complete within the lecture week, so participants are encouraged to work at their own pace and continue implementation after the course.

1. Introduction to Altera Quartus by means of a simple switches-to-LEDs circuit.
2. Introduction to the Sources and Probes feature, and using it to read the switches and control the LEDs.
3. Use Questa to simulate a finite state machine. The machine source code is provided in the lectures.
4. Timing constraints and analysis using the SDC file and Timing Analyzer tool.
5. Hook up a provided SDRAM controller and test using Signal Tap Logic Analyzer.
6. Use Platform Designer to create an Avalon bus architecture, controlled by using a JTAG to Avalon Master Bridge. Python is then used (in conjunction with the Altera System Console) to control the circuit over the JTAG bridge.
7. Implement pulse-width modulation and data injection modules. 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.
8. 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).
9. 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.
10. Implement the FIR filter and finalise the project.

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 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.

The students are reminded that they are not designing a real-world system, and are encouraged to simplify the project for easy implementation and fitting onto the relatively small FPGA provided. At the same time, the project must show FPGA competence, so it should include at least a DSP chain and control scheme.

Typically, a design will inject data from SDRAM into the DSP-chain, and then store the result in the same SDRAM, which is then read and analysed by the PC. Control will typically be done by means of a JTAG-based interface to the PC.

The students then submit a report that needs to:

- Describe the design overview by means of a block diagram
- Provide minimal detail of the various modules that compose the system
- Provide practical results: performance figures, etc.
- Show a functional FPGA-based DSP chain (using photos, oscilloscope screenshots, etc. as evidence)
- Provide a link to the source code on an accessible Git server (typically GitHub)

5 Course Assessment and Examination

The assessment of this course is based entirely on the project, which is sub-divided as follows:

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

5.1 Marking Rubric

The 40% / 60% breakdown is further subdivided as follows:

- 20% – Source code quality
- 20% – "The project works"
- 20% – The project overview in the report, i.e. the "overview" section
- 20% – The project design in the report, i.e. the "detail design" section
- 20% – The implementation review in the report, i.e. the "results" section

A mark out of 5 is assigned for each "20%" section above, as follows:

0: Did not hand in (strictly according to the deadline on the Amathuba assignment); or missing from the report.

1: Minimal effort

2: Reasonable attempt, but I cannot easily understand what you're doing

3: Good attempt, and reasonably well explained, but incorrect in small details

4: Everything is clear, concise and correct

5: Extraordinary – went the extra mile

5.2 Example Report Layout

An example layout for a "went the extra mile" report would be:

1. Title Page (1 page), including project title, name, student number and date
2. Overview (2 pages)
   - Briefly introduces the system concept, and what it's supposed to be doing.
   - Presents an overview block diagram – nothing fancy, and no more than 5 blocks.
   - Briefly introduces each block, explaining its purpose in the design.
3. Detail Design (about 5 pages). For each of the blocks in the overview:
   - Explain the purpose of each block on these detail block diagrams
   - Explain any major design decisions (such as the length of a FIFO, or the number of bits in the data word of a stream)
   - Explain the external interfaces: how does this sub-system communicate to the other sub-systems?
   - Explain how the system is controlled, and how the user interacts with it
   - Explain how the system guarantees real-time operation
   - Show a tree diagram of the firmware module hierarchy
4. Results (2 pages), which answers questions such as:
   - How much resources does the design use?
   - How was the system tested?
   - How was test data generated?
   - How do the tests relate to the intended real-world system?
   - How does the measured data compare to expectations?
5. Conclusions (1 page), including a brief summary.