Space optimised FPGA-based side-microprocessor.

PRCO304 - Project Initiation Document

Ben Lancaster
February 1, 2018

Table of Contents

1 Introduction **2**

2 Business Case **2**

3 Project Objectives **3**
3.1 Core Deliverables 3

4 Initial Scope **3**
4.1 Core Deliverables 3
4.2 Extended Deliverables 3

5 Resources and Dependencies **4**

6 Method of Approach **4**

7 Initial Project Plan **5**
7.1 Project time-line breakdown 5
7.2 Control Plan 5

8 Initial Risk Assessment **6**

9 Quality Plan **7**

10 Legal, Social, and Ethical Considerations **7**

11 References **8**

1 Introduction

Field-Programmable Gate Array (FPGA) devices are an incredibly powerful and versatile solution to many electronics applications including digital signal processing and high-speed test and measurement tools. I will use this project opportunity to learn more about FPGA development and CPU architecture and apply knowledge learnt to create a solution to the need of a side-microprocessor in many FPGA-based applications.

Modern computing and electronics equipment, like function generators, oscilloscopes, and spectrum analysers, use FPGAs to implement their compute intensive logic. These FPGAs are often accompanied by a small, low-cost, microprocessor to supervise and provide interfaces to external peripherals.

The aim of this project is to implement this side-microprocessor into the FPGA to save on BOM costs, PCB space, and power costs, which contribute to higher development and product costs. While savings can be made by the lack of side microprocessor, the product may need a larger FPGA to accommodate the embedded microprocessor. The project will produce a small, soft-core, CPU design and compiler.

Although there is no direct client in this project, I believe this project will produce an attractive product for FPGA-based product designers wishing to employ an embedded processor solution.

2 Business Case

I will target my interest in FPGA development and apply my learning of such in tackling the issues resulting from the use of a side-microprocessor in FPGA based applications.

The requirement of a side-microprocessor to control and provide external interfaces to FPGA-based applications carries a significant demand in both development and projects costs. Firstly, the inclusion of a external microprocessor in a project design will require more PCB space and design considerations, adding to the development time and costs of the project. The external microprocessor may also require a licensed compiler to compile and load the code onto the microprocessor, adding to the cost of the project. In addition, the microprocessor's on-chip memory may not be large enough to store the compiled code and an external flash memory chip may also be required.

Moving to an integrated microprocessor on the FPGA brings many significant advantages: reduction of required PCB space and development time, lower BOM (bill of materials) cost, and better in-field updating.

Releasing updates to embedded projects is a challenging problem. With the integrated solution, FPGA bitstreams and the soft-microprocessor code can be bundled together, making it much easier to update products in the field without sending an engineer to the location or providing complicated instructions which require specific equipment (e.g. in-circuit debuggers).

3 Project Objectives

The outcome of the project will be to design a small, portable, FPGA-based, CPU core that electronic Product Designers can choose as an alternative to a physical side-microprocessor to embed into their product.

3.1 Core Deliverables

1. (Core deliverable) To improve my knowledge and experience of FPGA development, CPU architecture, and low-level programming.
2. (Core deliverable) To build a working and operational soft-core CPU core capable of performing simple tasks.
3. (Core deliverable) To provide product designers with an affordable alternative to a side-microprocessor in their FPGA-based products.
4. (Core deliverable) To provide a technical documentation and specification for the embedded core.
5. (Sub deliverable) To provide embedded products a convenient solution to in-field updating.
6. (Sub deliverable) To provide easy interfacing between the FPGA design and the embedded core.

4 Initial Scope

4.1 Core Deliverables

These deliverables are the base requirement for the project to be released in a functional and worthwhile state.

1. (Core deliverable) A small, portable, instantiate-able, FPGA-based CPU core.
2. (Core deliverable) A C-like programming interface. A compiler taking input of a C-like grammar and outputting executable machine code for the embedded core. The machine code can be embedded into the FPGA bitstream and loaded onto the FPGA to run. Time estimate: 1 month.
3. (Core deliverable) A 16-bit RISC instruction set architecture (ISA). The core (1) will decode and execute instructions encoded in this format. The compiler (2) will output machine code in this format. The ISA will support: fixed length instructions; 12-bit immediate values; primitive arithmetic instructions (ADD, SUB, MUL, etc.); GPIO read and write instructions; RAM stack operators (PUSH, POP). A custom ISA will be designed and implemented (see section 10).

4.2 Extended Deliverables

These deliverables may not be achievable in the time frame specific in section 7. These deliverables may require extra time to develop, require more experience and skill to develop, or require resources currently unattainable.

1. GCC/LLVM/8CC compiler backend for C programming.
2. Wishbone interface for easier modularity and inter-module communication.
3. Multi-core design with Wishbone (2).
4. Single-step debugging interface (with JTAG?).
5. Configurable build options (register/bus widths, optimisations/pipelining, user/privileged user mode to support modern operating systems).
6. Memory management modules to provide protected and virtual memory lookup tables.

5 Resources and Dependencies

For the first half of the development cycle, the core can be developed and verified using the Verilog simulator and test suite, Verilator, and VHDL and Verilog simulator, iSim.

The second half of development will require deploying and debugging on real hardware. This will require an FPGA development kit. To better emulate customer products, the development kit should feature common components such as LEDs, GPIO, USB interface, flash-based storage and memory, and optionally an analogue audio output port.

The low-middle range of FPGA devices I am targeting is the popular and affordable yet feature rich Spartan-6 and Artix-7 FPGAs. From my placement, I have gained experience in Xilinx FPGAs and so will be targeting them for this project to reduce risk and development time.

The following FPGA development kits are suitable for this project:

1. MiniSpartan6+ - Scarab Hardware - £79 (already owned) (*MiniSpartan6+*, 2014). The MiniSpartan6+ features a Spartan-6 XC6SLX9 FPGA, 8 LEDs, 2 digital and analogue headers, FT2232 FTDI USB to JTAG, 64Mb SPI flash memory, 32MB SDRAM, an audio output jack, and a MicroSD socket.

2. Arty Artix-7 FPGA Development Board - Digilent - £100 (*Arty Artix-7 FPGA Development Board*, 2015). The Arty development board features a larger Artix-35T FPGA with over 20x the number of logic cells and block memory compared to the LX9 in the MiniSpartan6+. The board components include 256MB DDR3 RAM, 16MBx4 SPI flash memory, USB-JTAG, 8 LEDs (4 of which are RGB), 4 switches, 4 buttons, and multiple Pmod connectors.

   The greater number of IO options and larger FPGA make the Arty board better suited to emulating real customer products.

6 Method of Approach

Development of the core and compiler will be done in separate stages of the project (see section 7). The two deliverables will be split into 2 sub-projects. Both sub-projects will employ the Agile development process, using Agile's sprints to split up tasks into sub-tasks and Agile's scrums to discuss progress, features, and changes.

Technologies used will be:

1. Verilog - A hardware description language used to code the internal FPGA design.

2. C - A low-level programming language to develop the compiler and assembler.

3. Verilator - A C++ Verilog simulator and unit testing framework for verifying the FPGA design. Unit tests will be written for each component of the core: register set, decoder, arithmetic logic unit (ALU), and IO. This will aid the sprint approach by ensuring that requirements implied by the unit tests do not break over development iterations.

4. iSim - A Verilog and VHDL Simulator. This will be used to visualize the timings of internal signals within the FPGA components such as the decoder and ALU.

7 Initial Project Plan

7.1 Project time-line breakdown

The project will be split into 4 parts: 1) Project information gathering and requirement generation; 2) Active development sprints; 3) Test and verification; 4) Final report and clean up.

The following table breaks down the 4 parts into sub-tasks and provides their descriptions and estimated start and end times.

Stage	Start Date*	End Date*	Project Deliverables
1.0. Project Initiation		02 Feb	Process Initiation Document
1.1. Research and requirement gathering	02 Feb	09 Feb	Existing soft-core processor designs, constraints, features, implementation.
1.2. Core high level design	10 Feb	17 Feb	Soft-core CPU architecture; Register definitions; Bus widths; Initial ISA instruction table.
2.1. Core development sprints	18 Feb	10 Mar	Iterative soft-core development sprints
2.1.1. Core testing and verification	11 Mar	15 Mar	Any tasks required to meet design constraints.
2.2. Compiler development sprints	15 Mar	31 Mar	Iterative compiler development sprints
2.2.1. Compiler testing and verification	10 Apr	14 Apr	Any tasks required for compiler to produce correct code generation.
3.1. Real hardware deployment	15 Apr	19 Apr	Deployment of Verilog code to a real FPGA device.
3.2. Final verification	20 Apr	24 Apr	Verification for FPGA design and compiler.
4.1. Complete final report	25 Apr	4 May	PRCO304 Report.

Table 1: Initial Project Plan time breakdown
*Expected time.
Shaded stages are time varying periods for bug fixing.

7.2 Control Plan

Management of the project will be done using the PRINCE2 technique.

The project initiation document (this) describes high-level requirements, objectives, and business cases.

Weekly highlight reports and meetings will be held to ensure task proficiency and to identify any challenges that need attention.

Project risks and challenges are identified in section 8 along with proposed solutions for their occurrence.

8 Initial Risk Assessment

The following section outlines potential projects risks their suitable management strategy.

1. Real hardware synthesis.

   A challenge involved in the development of FPGA, CPLD, and other programmable logic devices, is the realization of the HDL code on real hardware. This can result in different behaviour of the real implementation to the simulated design - a major (and expensive) problem. This issue is caused by not meeting physical constraints required by the FPGA. These include timing, space, and power constraints.

   To help reduce this issue, I will utilise the ISE Design Suite's constraint validator tool. Before deploying to real hardware, the design must meet the constraints I declare that enable it to run correctly on real hardware. I can use these constraints to identify how much space, time, and power, I have left to implement features.

2. HDL programming.

   HDL (Hardware Description Language) is a text based language used to describe hardware components and their inter-connections. Verilog, a HDL language closer to C than VHDL, is what my FPGA core will be programmed in. This language is taught very little of in the Computer Science course and will require external learning resources so I can use it effectively.

   My placement, telecommunications signal generator company, Spirent Communications, heavily utilise FPGA devices in their products, in which I gained valuable knowledge on the FPGA development life cycle and deployment. To improve my knowledge of the tools required (ISE Design Suite) gained from my placement experience, I shall learn from HDL programming books such as HDL Programming Fundamentals: VHDL and Verilog (Botros, 2006).

3. Compiler development time.

   A compiler will be required to provide an easy method of running user code on the FPGA core. The compiler is a lesser deliverable but will take considerable to time implement.

   If time is short, the compiler may only convert and assemble an assembly-like language with simple features (goto statements, stack management i.e. stack frames). If time is available, a better grammar can be developed with common language features such as if statements, scope blocks, and variables.

   The possibility also exists of using an existing compiler, such as GCC, LLVM, or 8CC, and creating a custom back-end for the FPGA core's architecture. My already brief experience with these compilers with their poor documentation means it may be quicker to build a compiler from scratch than create a custom back-end. A short period of time will be a given to allow exploration of compilers as it may allow using more language features (ANSI C) instead of a small subset. This will allow for a more complex demo of the FPGA core.

4. Schedule overrun.

   This is a complex project will multiple sub-projects (core & compiler). Ensuring the large number of features will require a tight development schedule which is prone to over-running.

   I can identify and account for this by having weekly progress updates that will be scheduled with the project supervisor outlying feature progress and challenges. If the schedule slips largely due to an unforeseen problem or unreasonable requirement, this shall be brought up in the following meeting and a solution will be agreed upon, be it modifying deliverable or allowing extra time for the feature.

5. Technology failure.

   To overcome the risks of data loss all code and resources will be stored in local and remote Git repositories. In the event of the FPGA development kit failing, be it a component on the board or the FPGA itself, either: (a) a demo of the FPGA core not showing features of the failed component;

or (b) a simulated design that meets constraints imposed by the physical FPGA will be provided and demonstrated in a simulator.

9 Quality Plan

The following quality strategies will be employed to achieve a successful project and product.

Quality Check (QC)	Strategy
QC1. Requirement reality	Requirements will be checked during the weekly highlight reports to verify that when requirements begin to be implemented they are realistic and achievable within the time frame specified in section 7.
QC2. Soft-core design validation	While continuous testing and verification will be performed on the core (unit test, FPGA constraints reached), a variable period of time (stage 2.1.1) will be allocated after the development sprints to fix bugs and unexpected behaviour, and polish the final design.
QC3. Compiler validation	As with QC2, unit tests and continuous integration tests will be performed for each code change to validate that changes do not produce bad code generation. A time varying period (stage 2.2.1) is also allocated to fix and polish the compiler.
QC4. Real hardware performance	Electronic test equipment, such oscilloscopes and digital logic analysers, will be used to verify the correct behaviour of the code on real hardware. Initial risk (1) states that there is a risk of the FPGA deployed core will behave differently to the simulation.

Table 2: Initial Quality Plan.

10 Legal, Social, and Ethical Considerations

Legal considerations need to be taken into account due to already existing commercial soft-processor designs. Existing soft-processor designs include the ARM family of soft-cores (*ARM in the World of FPGA-Based Prototyping*, 2016) and Xilinx's MicroBlaze soft-core (*MicroBlaze Processor Reference Guide*, 2017). Emulating another soft-core processor's architecture may result in legal challenges even if I do not distribute the final product. As this is a learning project, instead of emulating another architecture, I will design my own architecture from the ground up to learn first-hand the design considerations, implementation, and verification of CPU designs.

Social and ethical considerations are not applicable for this project.

11 References

ARM in the World of FPGA-Based Prototyping (2016).
URL: *https://community.arm.com/processors/b/blog/posts/arm-in-the-world-of-fpga-based-prototyping*

Arty Artix-7 FPGA Development Board (2015).
URL: *https://uk.rs-online.com/web/p/programmable-logic-development-kits/1346478/*

Botros, N. (2006). *HDL Programming Fundamentals: VHDL and Verilog*, Da Vinci Engineering Press.

MicroBlaze Processor Reference Guide (2017).

MiniSpartan6+ (2014).
URL: *https://www.scarabhardware.com/minispartan6/*