Final Year Project Specification

Richard James Howe: 082060589

November 12, 2012

Abstract

This is my Final Year Project specification, it will detail more the technical side of things, it is split into two sections, as how I proceed with my project will depend on if I can get certain sections working before the main event. My final year project is to be a 1980s Style Microcomputer that I will build and program myself using either an FPGA or an ARM based system.

Contents

1 Introduction

I will split this specification into two different sections, while the technologies I will have to deal with will be the same, the way I go around solving these problems will be very different depending on how my prototypes I try to complete before Christmas 2012 go.

I will either be using the `Nexys 3` development board [1] or the `STM32F4 discovery` board [2], the latter being a very cheap ARM based device, which one I chose will greatly affect the route the project goes, but the end product should roughly be the same.

1.1 Specification of Specifications

This specification is more a pointer to other specifications, depending on the deliverables I decide upon, I will have to implement other peoples standards or interface with modules that do implement other peoples standards.

For example, if I use the FPGA, my board would need a way of communicating with the outside world, it would be possible for my to implement my own, or more likely to use a pre-built module written in VHDL, and integrate this into my product.

It would not be cheating to do so, even if all I do is integrate other peoples module into my own project, there still is a lot of work to do, such as writing the software stack to go onto the device, and even working out how all the modules fit together.

If I were to implement my own device, for example a serial port, I could consult the standard, so I am going to link to a list of standards that I will be using in order to complete my project.

The final section of this document contains a small language specification for FORTH, the interpreter I want to be running for my system.

1.2 Decision: FGPA vs STM32F4

I will decide on which route to go down before Christmas, my supervisor for this project, *Marc Eberhard*, [3] and I would prefer if I were to implement my project on an FPGA, however this may be too complicated to do, in order to decide I would need to try to implement a CPU core in VHDL and get it working before Christmas 2012. I would then know if taking this route would be feasible or not.

2 FPGA: Main plan.

As I have said, this would be my main plan, to use an FPGA to make my own computing system, this is not infeasible and has done before, my own design will be unique and customized to the task I want it to do.

While it will not be useful as a general purpose computer, it could be used for pedagogical purposes, both for me and for other people as well.

[1] Nexys 3 Product website: `http://www.digilentinc.com/`

[2] STM32F4 discovery product website: `http://www.st.com/`

[3] Marc Eberhard:`mailto:m.a.eberhard@aston.ac.uk`

The hardware would be very simple, and at minimum would be a Nexys 3 development board, with a serial connector interface. The FGPA contained in the device is a `Xilinx Spartan-6 XC6SLX16` which is enough to fit my project on along with enough internal RAM to run its program.

2.1 Deliverables

The following are a list of the projects deliverables.

- CPU and Waveforms.
- Cross assembler.
- Serial Interface.
- FORTH language interpreter.
- *(Optional)*: Keyboard Interface.
- *(Optional)*: VGA Interface.

The last two being entirely optional, it all depends on how feasible it is to make the last two.

2.2 Languages to use

The following is the languages I wish to use, for a variety of tasks, from hardware description, to automation.

- `VHDL`: Hardware description of CPU core and peripherals.
- `VHDL`: Test Benches for hardware.
- `Python`: Cross Assembler.
- `Assembly`: Used to write the FORTH interpreter in.
- `FORTH`: Ultimate goal of the project, a FORTH language interpreter.

Other miscellaneous languages used will be LaTeXfor writing up the report, GNU Make files for the build system, and Bash shell scripts for automating simple tasks. The Git version control system will be used to keep track of the latest build and documentation.

2.3 Industry and other standards

As I have said I will be linking to other standards, which I will use in my project, some of which will be part of optional modules such as the Keyboard and VGA (video) output.

- J1 Processor Core: Described in a paper [4] and written in Verilog, this will be rewritten in VHDL and modified.

[4] J1: a small Forth CPU Core for FPGAs, James Bowman. `http://www.excamera.com/files/j1.pdf`

- Serial Port: This is a very well known standard, and will form the primary interface between the outside world and my CPU core.
- VGA standard : A well known Video standard (optional) [5]
- PS/2 Keyboard: [6] [7]

2.4 Software to use

I already have the software and build environment set up and ready to be used, I will have to adapt some Makefiles for my purpose, but apart from this, the software environment I will be working on is ready. The project will be developed in a Linux environment (Debian 6.0), using Xilinx Webpack (Latest version), and Digilents drivers. Other software that is relevant to the project is Git, for versioning, GNU make, and finally, GHDL for simulation purposes only.

2.5 J1 Processor

Instead of specify my own instruction set, I am going to use on that is already specified for me, as my device does not have to fit into such a small space, I can utilize much more of the chip if I want to, the original processor had to fit inside a cameras FPGA that had many other tasks to do.

I could experiment with optimizing the core for my specific needs, for example, I could decide if I want to use unsigned or signed arithmetic, I could also add more operations to the core (one bit is unused in the instruction set). I am refraining to say as to what I may add, I would like to see would are the bottlenecks in my design before I start to customize the core. A big boost would probably be having a general purpose multiply instruction, however I may be able to get away with a left shift for multiplication (by 2^n only), and division by 2^n as well (right shift).

Depending on what I want to do, I could add instructions for cyclic shifts, or even more complicated things instructions tailored to my task. This is optional.

2.5.1 Error handling and multitasking (Optional).

I would like to create a robust system, and I would like to do that on many levels, testing as much as possible and eliminating sources of error, in that vein, there could be certain systems in place for checking whether certain things make sense. Bounds checking could be enforced at the hardware level if needs be, making sure certain instructions can not be executing if they operate on data in off limits areas.

Another possibility is to aid multitasking, either with multiple cores, or with banks of registers that can be switch to and from in a few instruction cycles. Both of theses tasks would be entirely optional.

Again, the J1 Processor can be found here `http://www.excamera.com/sphinx/fpga-j1.html` and here `http://www.excamera.com/files/j1.pdf`.

[5] VGA:`http://www.mcamafia.de/pdf/ibm_vgaxga_trm2.pdf`
[6] PS/2:`http://www.computer-engineering.org/ps2protocol/`
[7] PS/2:`http://retired.beyondlogic.org/keyboard/keybrd.htm`

3 ARM : Back up plan.

My back up plan is to use an STM32F4 discovery board, which utilizes a powerful ARM processor and has many peripherals on board. The good news about going down this route is that I already have a working interpreter written in pure ANSI C, my code repository which is available on my Github account is here [8].

The idea would be to port this working interpreter to the STM32F4, interface it with the peripherals on board, and get a working computer system, I am more confident in this approach as I already have the most difficult part of the system up and running.

The same standards apply to the ARM; Serial Port (which is implemented already), VGA and PS/2 Keyboard.

I would also be able to an ADC and DAC, as well as some other peripherals to the devices functionality, which is always a good thing, I want the device to be as functional as possible.

As far as a language specification goes, for this system, my current implementation *is* the standard (which is how some languages actually are specified, such as Perl versions before version six).

The ARM device has more than enough processing power to handle VGA graphics itself, if this is not possible to do by myself (which would be entirely a software issue), then it is possible to buy modules in that could handle this functionality [9].

3.1 Deliverables

- Working Interpreter ported to the STM32F4.
- Serial and other peripherals up and running.
- Software stack for the interpreter.
- *(Optional)*: Keyboard Interface.
- *(Optional)*: VGA Interface.

3.2 Software to use

Like the FPGA option; Git, GNU Make, LaTeX(for documentation) and Shell would all be used, possibly with Python for some serial interfaces. The interpreter would be written in C, as well as all the software for the STM32F4, this needs a special version of GCC (The GNU Compiler Collection), which can be acquire from here[10]. This is a cross compiler that will run on you computer and compile code for the device, which then can be uploaded with ST-Link[11] and debugged with the same software in combination with GDB (the GNU Debugger).

[8] Github Account: howerj `https://github.com/howerj/c-forth`
[9] Serial to VGA:`http://www.hobbytronics.co.uk/serial-vga`
[10] Summon Arm Toolchain: `https://github.com/esden/summon-arm-toolchain`
[11] ST-Link: `http://cgi.cs.indiana.edu/~geobrown/stm32/Main/GDB`

4 Common Goals

There are number of common goals between these project options, the device should be capable of acting as a stand alone one, in terms of processing, there are multiple different options for the interface for the device, ranging from a serial interface, to video and keyboard, to even Ethernet (although the latter is not very likely).

Both should implement a working computing system, where the user can type commands at a prompt. The user should be able to create new programs, with all the usuals that a programming language offers such as arithmetic, Input and Output control, defining new functions, etc., it should be fairly easy to use as well.

4.1 Hardware

The hardware for this project is easy to come by, being very cheap, I either have it already (the STM32F4) or the University has it (the Nexys 3 development board) and I am thinking about buying one for myself. The hardware is not going to be, or should not, an issue for this project. It would be a project in itself to try to make PCBs for this chips due to the high clock frequencies they run at and the types of component packages they use, it is best to use pre-bought solutions.

5 Conclusion

I think this project is achievable in at least one of the two forms proposed above, I would much prefer to go down the FPGA route, but this may not be possible due to complexity issues. My current plan is to develop this project to see if it is possible at all or not.

In the end, I will get a working computing system that the end user could use to perform simple tasks, although it is meant to be primarily used as an educational computer, and not something to perform actual work, it is conceivable that the device could do something useful such as automated data acquisition, using the ADCs on the STM32F4 for example.