University of Toronto
Department of Electrical and Computer Engineering
ECE496 Design Project

Virtual FPGA fabrics
Implementation of a Virtual FPGA Architecture

Individual Progress Report

January 17, 2012

Neil Isaac n.isaac@utoronto.ca

Project ID:	2011017
Supervisor:	Jason Anderson
Administrator:	Ross Gillett
Section:	#7

ECE496Y Individual Progress Report Evaluation Form

Student				
	Project ID: 2011017	**Project Title:** Virtual FPGA Fabrics		
	Student Name: Neil Isaac		**Supervisor:** Jason Anderson	
	Section # 7	**Administrator:** Ross Gillett		
	Contact hours per month with supervisor :	4	**Optimal # contact hours per month:**	4

Administrator's Evaluation (Provide a rating for each section)	Excellent	Good	Adequate	Marginal	Poor/Missing	**Comments** *(also see comments in report)*
Progress: execution of work plan, significance of reported work Problems(circle): *claims not justified, status of tasks unclear*						
Method: Sound engineering practices reflected in actions, decisions, and testing. Changes reflect good project management. *Problems(circle): actions, decisions, testing*						
Reported Results: Quality and completeness of documentation *Problems(circle): inadequate documentation or analysis, inadequate test results, completion of task not verifiable*						
Overall Quality of Document						
Presentation (circle problem areas) *Grammar, spelling, clarity, writing style, use of figures & tables*						
Content (circle problem areas) *Organization, substance, references, Gantt chart incomplete*						
Other problems (circle): *late submission, other (specify)*						

Administrator's grade (/10):	**Section average (/10):**	**Administrator's signature:**

Note to supervisors: Your evaluation was done online. There is no need to return anything to the administrator.

Executive Summary

Our project is progressing well and we are optimistic that we can meet our deadlines. We initially planned optimistic deadline for long-term tasks, but left a couple of months of slack at the end of the project. Now that we have begun those tasks, we have adjusted the gantt chart to reflect more realistic expectations for some of those tasks. The slack time is now reflected within the time allocations for the tasks themselves.

My main contributions to the project are:

- components of the Virtual FPGA verilog design,
- the third-partly tool flow to produce placement and routing data,
- software to convert the placement and routing into a programmable bitstream, and
- I will work on improving the quality of this software and support future improvements to the verilog design.

We have completed the basic components of the design, and the components all work individually, but need to do extensive debugging work to get it all working together. Once we complete this debugging, we have a number of improvements planned to improve the quality of the virtual FPGA (in terms of usability to researchers, as well as its efficiency.) These improvements are not necessary for a proof-of-concept, or for a fully functional demo.

We are confident that we can complete these remaining tasks before the design fair.

1 Individual Progress and Contributions

This section lists the four major tasks I have been responsible for. The respective tasks are discussed in more detail in section 2, with emphasis on my own contributions. The complete gantt chart, showing all of our group's tasks is included in Appendix A.

Task title	**Category**	**Status**	**Original date**	**New date**
1. Virtual FPGA Verilog code	hardware	debugging	Aug-Dec 15	Aug-Jan 15
2. External tool flow	tools	completed	Dec 1-Dec 15	Dec 1-Dec 15
3. Bitstream generation software	software	debugging	Dec 1-Dec 15	Dec 1-Feb 15
4. Stress-testing and improvement	testing	not started	Dec 15-Apr 1	Jan 15-Apr 1

2 Information on Individual Milestones

1. Virtual FPGA Verilog code

Category: hardware

Original date: August - December 15

New date: August - January 15

Responsibility: Neil, Keyi

The components I wrote were the UART, logic block, connection block, and tile grid.

Status: debugging; future improvements are planned

- Verilog code for basic virtual FPGA is complete.
- Debugging work is required to get it to work in new configuration.
- Future improvements will aim to to reduce the size of the circuit.

Actions

- I wrote the UART, logic block, and connection block modules.
- I worked on integrating them into the tile grid of the virtual FPGA design.
- Serial data transmission over the UART seems is prone to data corruption with some of the hardware we have.

Decisions

- The connectivity architecture of our modules was decided by taking the simplest approach supported by the placement and routing tool *VPR*.
- We implemented all the memory elements of the virtual FPGA circuit (including routing switches and logic tables) using native logic elements on the Virtex 5 board, configured as shift registers, as we described in the proposal document.
- To address the UART corruption issues, I added a parity bit to each byte sent to program the virtual FPGA in the bitstream. If a parity bit doesn't match the byte, it can try to resend the byte. This was not good enough for large bitstreams, because sometimes 2 bits with flip within the byte, so the parity bit will still be 'correct', but the byte isn't valid. To address this, I changed the hardware to send every byte received by the board back to the computer to verify it. If bytes are returned and are incorrect, it can't recover, but the user can try sending the bitstream again. If this remains a serious issue, I can adapt the programming protocol to allow it to replace the previous byte if it doesn't match when it is sent back to the computer.

Testing, verification and results

- To test the virtual FPGA at this stage, we created the bitstream (programming) files manually to implement small test circuits. Functionality was tested using switches and LEDs.

2. External tool flow

Category: tools

Original date: December 1 - December 15

New date: December 1 - December 15

Responsibility: Neil

Status: completed

Third-party tools will be used to produce the placement and routing for our input circuits. These tools are: *odin* (synthesis), *abc* (technology mapping), *t-vpack* (clustering) and *VPR* (placement and routing).

Actions

- Wrote a test circuit, shown in Appendix B.
- Verified that the tools will run using each others' outputs.
- A bug in *abc* results in it deleting clock information in circuit descriptions.

Decisions

The bug in *abc* can be worked around, subject to several limitations. For the time being, I avoided it by writing an *awk* script to process the output from *abc* to add the clock information back in. Implicitly in this script, I assume that there is only one clock in the design, and it is always called "clk", which are both limiting assumptions. Ideally in the future, we will find a patched version of *abc*, or will write more complex software to read the input file and interpolate less limited clock information. The temporary *awk* script is shown below:

```
{    if ($1 == ".latch") { print $1, $2, $3, "re", "top^clk", $4; }
     else { print $0; }    }
```

Testing, verification and results

- Checked the 4 output files manually for correctness and to verify that they provide adequate information to reproduce the circuit's function, placement, and routing on our virtual FPGA.
- A screenshot of *VPR* showing the placement and routing is shown in Appendix C.

3. Bitstream generation software

Category: software

Original date: December 1- December 15

New date: December 1 - February 15

Responsibility: Neil

Status: debugging

- Code is feature-complete for prototype design.
- Working Verilog FPGA circuit is required to verify correctness.
- Changes will be required to reflect future changes to hardware design.

Actions

- The four inputs required to generate the bitstream are: the logic functions (in BLIF format) from *odin* and *abc*, the netlist from *t-vpack*, and the placement and routing files from *VPR*.
- All of the information needed is in these files, but it needs to be inferred by cross-referencing between the files. This was more complicated than expected, so it took longer then initially planned.
- Once I built a software model containing the data required, I wrote functions to loop over it and generate the bit patterns that will ultimately be programmed.
- Once all the bit patterns are ready, they are combined into bytes before being programmed to the virtual FPGA over a serial connection.

Decisions

- I wrote this software in Python 2.x because it is not performance critical, and the high-level abstractions available in Python speed up the development process.
- One technical issue that came up is that some of the bit patterns are not evenly divisible into 8-bit bytes, so all of the patterns have to be compiled before they can be packed into bytes. For example, the patterns `100110` and `1100001101` should be combined into the bytes `10011011 00001101`, not `00100110 00000011 00001101` (by padding each individual pattern.)
- Since the bitstream is programmed serially, the first hardware in the programming chain corresponds to the last bits in the bitstream. This means that the bitstream is in reverse order of the intuitive numerical indices of the hardware. This has led to a lot of trivial mistakes in the programming that I fixed gradually.

Testing, verification and results

- I am testing basic functionality using the simple circuit in Appendix B.
- This program outputs a text file containing each pattern in the bitstream which I can examine by hand, so I can test the software without the virtual FPGA circuit.
- Extensive testing is covered by task #4.

4. Stress-testing and improvement
Category: testing, hardware
Original date: December 15 - April
New date: January 15 - April

Responsibility: Neil, Keyi
I will focus on verification and improvements in the tool flow and bitstream software.

Status: not started
We will begin extensive verification and improvement once the prototype works.

Actions n/a

Decisions n/a

Testing, verification and results

- Make sure clocked logic works properly.
- Make sure signal routing is always correct.
- Make sure designs with hierarchical modules can be compiled.
- Develop a way for the user to assign their inputs and outputs to specific pads (the peripheral input/output blocks) on the virtual FPGA.

3 Progress Assessment

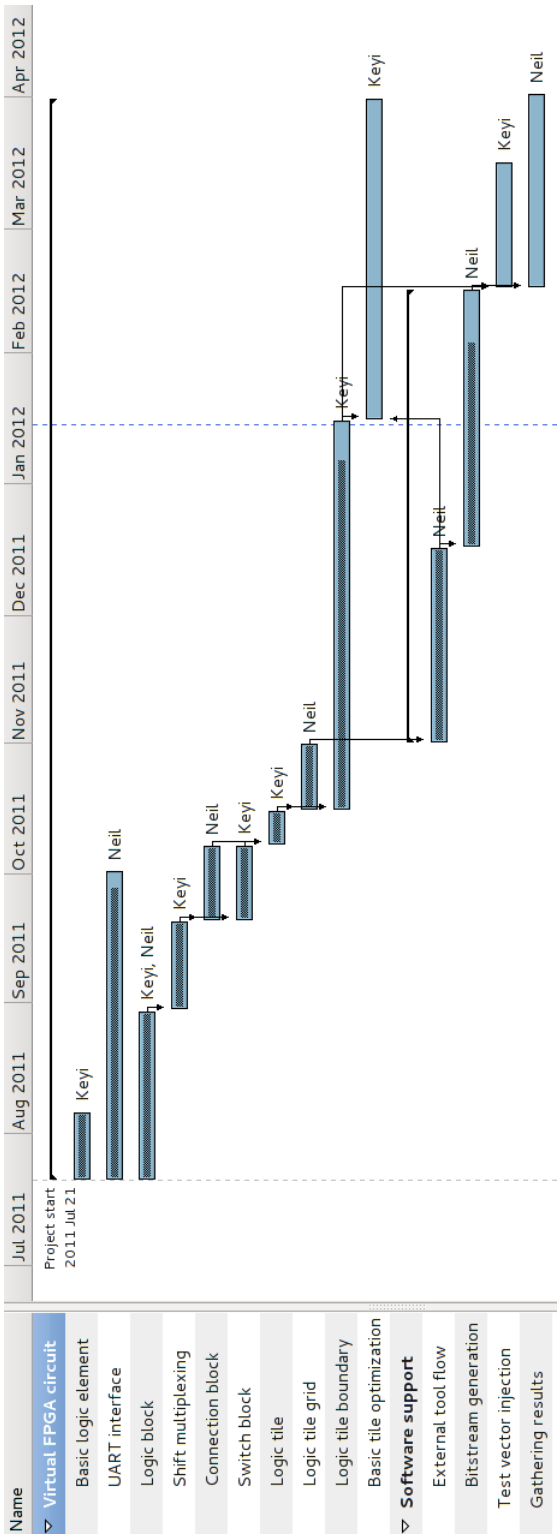
Our project is progressing well and we are optimistic that we can meet our deadlines. Note that we initially planned optimistic deadline for long-term tasks, but left a couple of months of slack at the end of the project. Now that we have begun those tasks, we have adjusted the gantt chart to reflect more realistic expectations for some of those tasks. The slack time is now reflected within the time allocations for the tasks themselves.

We have completed the basic components of the design, and the components all work individually, but need to do extensive debugging work to get it all working together. Once we complete this debugging, we have a number of improvements planned to improve the quality of the virtual FPGA (in terms of usability to researchers, as well as its efficiency.) These improvements are not necessary for a proof-of-concept, or for a fully functional demo.

We are confident that we can complete these remaining tasks before the design fair.

Appendices

A Gantt Chart



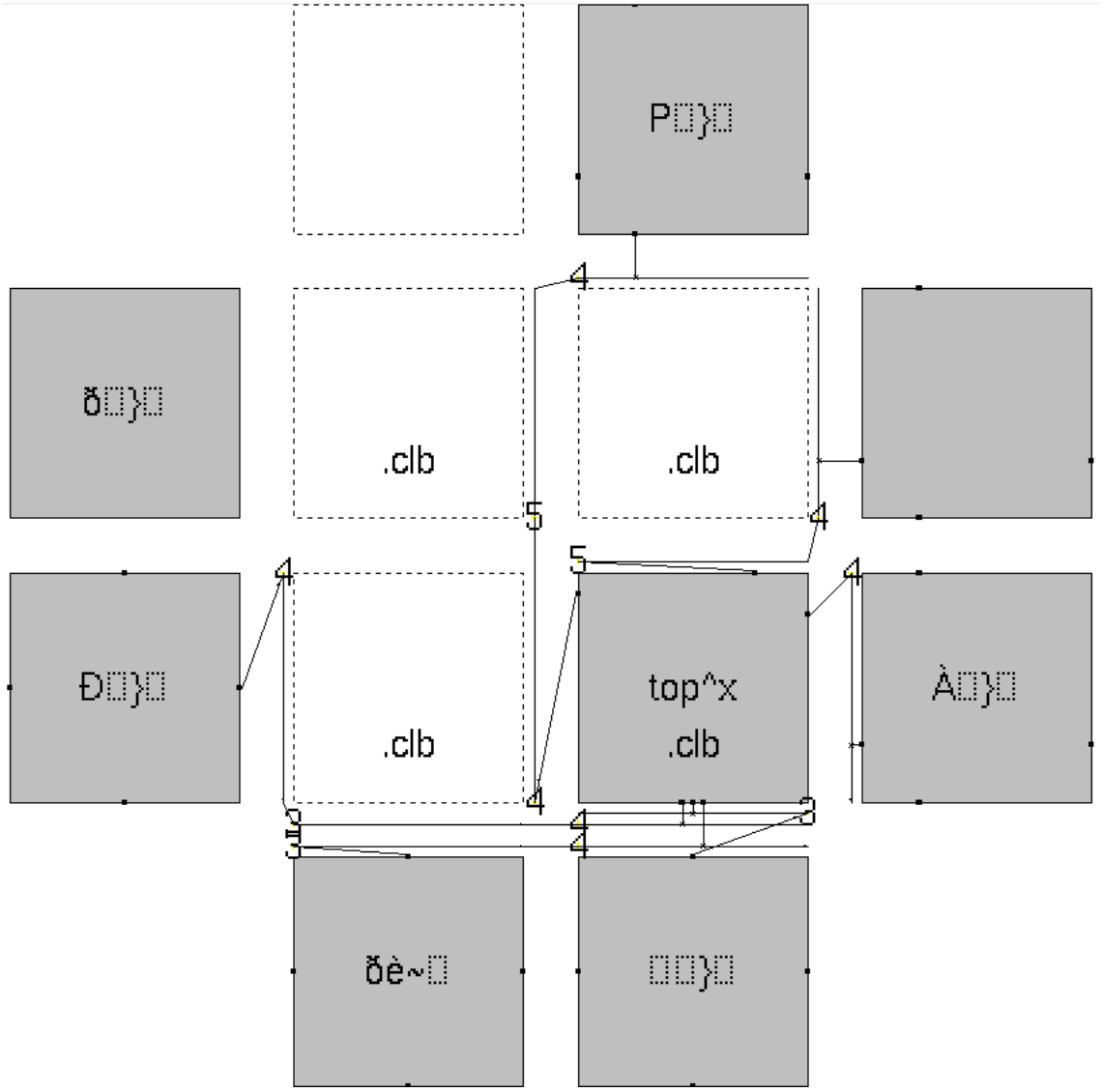
B Test Verilog file

```
module test(
    clk,
    a, b, c,
    x, y, z
);

    input clk;
    input a, b, c;
    output x, y, z;

    assign x = a & b;
    assign y = a | b;

    always @ (posedge clk)
        z <= c;

endmodule
```

This circuit has 3 outputs, x, y and z. It contains an AND gate, an OR gate, and a flopped path. The AND and OR gates allow me to test the tool flow and my bitstream generation software for logic functions that are mostly 0s (AND) and mostly 1s (OR), which is important in the BLIF format, as they are represented in the form that reduces the amount of function lines in the BLIF file. The flopped path lets me test the way I need to handle internal signals (before and after the flop) and how latches are represented in the BLIF and netlist files.

The result of running placement and routing in *VPR* for a the circuit derived from this verilog is shown in Appendix C.

C VPR Placement and Routing Result



This is a screenshot from the VPR tool after placement and routing were completed for the test circuit shown in Appendix B. Placed blocks are shown with a dark grey background and routing tracks are shown as black lines.

Note that there is a VPR issue in rendering the fonts for the IO pad labels.