



UNIVERSITY OF TORONTO
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
ECE496 DESIGN PROJECT

Virtual FPGA Fabrics Implementation of an Overlay FPGA Architecture

Final Report

March 22, 2012

Neil Isaac
n.isaac@utoronto.ca

Keyi Shi
keyi.shi@utoronto.ca

Project ID:

2011017

Supervisor:

Jason Anderson

Administrator:

Ross Gillett

Section:

#7

ECE496Y Final Report Individual Evaluation Form (one sheet per student)

Student	Project ID: 2011017	Project Title: Virtual FPGA Fabrics
	Student Name: Neil Isaac	Supervisor: Jason Anderson
	Section: 7	Administrator: Ross Gillett

Administrator's Evaluation Provide a rating for each section. Circle to indicate problem areas.		Excellent	Good	Adequate	Marginal	Poor/unclear	Comments <i>(also see comments in report)</i>	
Document								
Introduction: clear background, motivation, goals, requirements								
Final Design: system diagram, system and module level descriptions, assessment of strengths and weaknesses								
Testing and Verification: adequate documentation, discussion of results, comparison of results to requirements								
Summary and Conclusions: summary of accomplishments and challenges, success in achieving project goals								
Presentation: clear writing, grammar, organization, use of tables, diagrams, figures								
Project								
Final outcome: success in achieving stated project goals, technical complexity								
Individual effort and contributions								
Administrator's grade (/10):		Section average (/10):		Administrator's signature:				
General Comments:								

Note: Supervisor final evaluations are done online. See Supervisor > Final Evaluation on the course menu at <http://int.ece.utoronto.ca/ece496.xxyy/index.html>, where xxyy are the years (ex 0809 or 0910)

ECE496Y Final Report Individual Evaluation Form (one sheet per student)

Student	Project ID: 2011017	Project Title: Virtual FPGA Fabrics
	Student Name: Keyi Shi	Supervisor: Jason Anderson
	Section: 7	Administrator: Ross Gillett

Administrator's Evaluation		Excellent	Good	Adequate	Marginal	Poor/unclear	Comments (also see comments in report)
Provide a rating for each section. Circle to indicate problem areas.							
Document							
Introduction: clear background, motivation, goals, requirements							
Final Design: system diagram, system and module level descriptions, assessment of strengths and weaknesses							
Testing and Verification: adequate documentation, discussion of results, comparison of results to requirements							
Summary and Conclusions: summary of accomplishments and challenges, success in achieving project goals							
Presentation: clear writing, grammar, organization, use of tables, diagrams, figures							
Project							
Final outcome: success in achieving stated project goals, technical complexity							
Individual effort and contributions							
Administrator's grade (/10):		Section average (/10):		Administrator's signature:			
General Comments:							

Note: Supervisor final evaluations are done online. See Supervisor > Final Evaluation on the course menu at <http://int.ece.utoronto.ca/ece496.xxyy/index.html>, where xxyy are the years (ex 0809 or 0910)

Attribution Table

Section	Neil	Keyi
Executive Summary	ET	RD
Group Highlights	RD	ET
Background and Motivation	RD	ET
Project Goal	RD	ET
Project Requirements	RD	ET
System-level Overview	RD, ET	MR
Module-level Overview	ET, MR	RD
Assessment of Proposed Design	ET	RS, RD, MR
Testing and Verification	ET	RD
Summary	RD	MR, ET
Gantt Chart History	RD, MR	RS, ET
Validation and Acceptance Tests	MR, ET	RD
All	FP, CM	FP

Abbreviation Codes

RS	research and information	“All” row entries	
RD	wrote first draft	FP	final read through
MR	major revision	CM	compiling elements
ET	editing spelling grammar and expression	OR	other (describe OR1, OR2, etc)

Signatures

By signing below, you verify that you have read the attribution table and agree that it accurately reflects your contribution to this document.

Neil Isaac	Date	Keyi Shi	Date
------------	------	----------	------

Executive Summary

Academic studies of Field Programmable Gate Array (FPGA) chip architecture rely on simulations, as commercial FPGA chips contain proprietary designs that make their underlying architecture inaccessible to academic researchers. The goal of this project is to provide a physical platform for researchers to carry out FPGA architecture studies. The finished design is capable of implementing basic Verilog circuits with combinational and sequential logic.

Our design is an implementation of an Overlay FGPA on an existing, commercially available FPGA chip. Using a commercial FPGA as the physical medium for this project makes the design cheaper and more accessible to researchers, as they may have an appropriate FPGA chip already.

We have selected the Xilinx Virtex 5 FPGA as our development platform because we can use its native logic units directly. This will limit the models of FPGA chips that the overlay circuit can be implemented on, but it reduces the design's area overhead and improve its timing characteristics. The features we use on the Virtex 5 are forward-compatible with all current-generation Xilinx FPGA products, allowing the researcher to use a variety of FPGAs.

The validation of the design involves testing a set of unit-test circuits we wrote by placing and routing them with VPR, then transferring them to the FPGA overlay. The circuits can then be tested for correct behavior, confirming that the overlay design can be correctly programmed using VPR output, and that the inputs and outputs to the design are functioning properly.

The current budget for the proposed design was \$14.00 and was covered by the students. The required FPGA development boards and software licenses have been provided by the supervisor.

Acknowledgments

We would like to thank our supervisor, Professor Jason Anderson for providing input and feedback throughout the project.

Group Highlights

In order to build a working *Overlay FPGA*, we produced the following:

- Verilog for the individual FPGA building-blocks: the logic cell, logic block, connection block, and the switch block. (Keyi)
- Verilog to connect the building block components together in a grid and feed the programming signals through them. (Keyi)
- Verilog circuit to interface with the UART to enable serial programming of the Overlay FPGA. (Neil)
- Scripts to run the third-party tools which will process an input circuit (from the user) and produce a valid placement and routing. Some adjustments need to be made so the tools can read each others' outputs. (Neil)
- Software to convert the placement and routing into a programmable bitstream. (Neil)

Our Overlay FPGA circuit meets the functional requirements and objectives set out in our project proposal.

We found that overhead of implementing an FPGA on an FPGA was higher than we expected because some resources we used extensively were limited. Due to this limitation, we reduced our target size constraint for the Overlay FPGA from 3000 logic cells down to 100 logic cells. Although this sounds much lower, it is still large enough to implement working overlay designs.

The size restriction limits the size of circuits our Overlay FPGA can support, so we can no longer accommodate the MCNC benchmark circuits we originally planned to use. We developed our own simple *unit-test* circuits which each test very specific functionality. These new test circuits allowed us to incrementally track down basic issues with our bitstream.

Highlights for Neil Isaac

My main contributions to the project were:

- components of the Overlay FPGA Verilog design,
- compiling and scripting the third-party tool flow, and
- writing the software to produce and program the bitstream.

The basic components of the Overlay FPGA are the *connection block*, *switch block* and the *logic block*. The logic block consists of a variable number of *logic elements* and flip-flops which are the fundamental pieces that allow a user to implement their circuit on the Overlay FPGA . We designed the basic components so they can be replicated and connected on a grid of arbitrary size, and with an arbitrary amount of routing interconnect resources. The level of configurability we support limits the scope of optimizations we could consider, so our highly configurable Overlay FPGA takes more resources on the physical FPGA than a may be strictly necessary.

The third-party software flow consists of *ODIN* for synthesis, *ABC* for technology mapping, *T-VPack* for clustering, and *VPR 5* for placement and routing. My work on the tool flow involved figuring out how to use the individual tools, and producing the appropriate formats of files for each stage. Each of these tools had various bugs or technical issues I needed to work around.

The bitstream generation software I wrote reads the *BLIF* format function tables from ABC, the connectivity netlist from T-VPack, and the placements and routing files from VPR. I needed to correlate the files to find the location of each functional element in the placement file, its inputs from the netlist file, and how the inputs are connected from the routing file. In some cases, this data needs to be inferred because the tools don't fully specify the placements or connections.

Highlights for Keyi Shi

My contributions to the final design were:

- shift multiplexing for all logic tile modules
- component modules for the overlay verilog design
- verilog overlay tile boundary design and optimization

Multiplexers are used extensively in all our verilog modules. Each module has different multiplexing requirements, and one of our design goals was to have a configurable size/routing parameters; as such, the sizes of multiplexers required by our design vary module to module, parameter to parameter. I designed a flexible multiplexer capable of changing its size based on the number of inputs required, and this multiplexer is what drives all our overlay modules.

I focused on the routing elements of the overlay design, building the switch block and connection block modules and ensuring their compatibility with each other and the logic block. The modules were all designed to be flexible, capable of accepting user parameters and changing their sizes and arrangements to fit user needs. The modules also went through bitstream testing to make sure they behaved correctly under all use cases.

The verilog overlay tile combined basic verilog components into a single module, capable of being replicated in a grid formation to create the final overlay. The boundaries of the overlay required special modules to tie the grid together and interface between the overlay's internal and input/output signals. I wrote the boundary modules and debugged/optimized the overlay design, ensuring all signals between tiles are routed correctly, and the boundary signals are gathered into their appropriate input/output pads. I tested the overlay with manual and generated bitstreams to debug both the overlay hardware and the bitstream software.

Contents

1	Introduction (author: N. Isaac)	1
2	Final Design	3
3	Testing and Verification (author: K. Shi)	7
4	Summary (author: N. Isaac)	12
	References	13
	Appendices	14
A	Gantt Chart History	16
B	Validation and Acceptance Tests from Proposal	19

1 Introduction (author: N. Isaac)

This report explains the motivation, implementation and testing, and limitations of our Overlay FPGA circuit.

1.1 Background and Motivation (author: N. Isaac)

Academic researchers who study Field Programmable Gate Array (FPGA) design commonly use variations of an FPGA design architecture, described by Kuon et al[1], which we will refer to as the *Academic FPGA Model*. While FPGA chips are available from a variety of commercial vendors, their inner design is proprietary, making their architectures difficult to study. Furthermore, there is no existing physical implementation¹ of an Academic FPGA Model.

VPR[2] is an open-source placement and routing tool used in FPGA architecture research. It can handle many variations of the Academic FPGA Model. It is not currently possible to implement circuits produced by VPR on a commercial FPGA.²

Computer Aided Design (CAD) researchers who work on placement and routing algorithms for FPGA designs are presently limited to using simulations to evaluate or verify their work. They may be interested in testing circuits on a physical medium because it would be much faster than simulating the circuits.

1.2 Project Goal (author: N. Isaac)

The goal of this project is to design a circuit design based on the Academic FPGA Model. Researchers will be able to use the circuit to study FPGA architecture and CAD algorithms with circuits produced by VPR.

¹Alex Brant is also developing a comparable FPGA overlay platform with Prof. Guy Lemieux at University of British Columbia.

²A technology-mapped input netlist for VPR can be converted to an Altera Quartus VQM netlist file using *nettoqvm*[3], but the placement and routing can not be converted.

1.2.1 Requirements (author: N. Isaac)

Hardware (overlay) requirements:

H1	Compatible with a commercially available FPGA chip
H2	Capable of supporting user-specified number of logic cells and connectivity parameters
H3	Re-programmable over a serial interface after being flashed to the FPGA
H4	Support for inputs to and outputs from test circuits

Software requirements:

S1	Capable of translating VPR placement and route data for a test circuit into a bitstream for the Overlay FPGA
S2	Capable of programming the bitstream onto the Overlay FPGA to implement the actual circuit

General requirements:

G1	Support 6-input logic elements, which allows for comparison with current generation commercial FPGAs
G2	Fit at least 100 overlay logic elements on the Virtex 5 FPGA

2 Final Design

2.1 System overview (author: N. Isaac)

Figure 1 shows the high-level system interactions of the components used in our project. The components we built are shown in solid rectangles.

The finished project consists of three main parts: The overlay FPGA, bitstream generation software, and interfacing of inputs/outputs of the overlay FPGA.

The *Overlay FPGA* is a Verilog HDL circuit implementation of the Academic FPGA model which is constructed as an overlay on a Xilinx FPGA board. The arrangement, size and connectivity of the overlay circuit is controllable via parameters in the source Verilog. The overlay FPGA consists of organized tiles of *logic block*, *connection block*, and *switch block* modules. Together, these modules allow the overlay FPGA to implement different logic circuits.

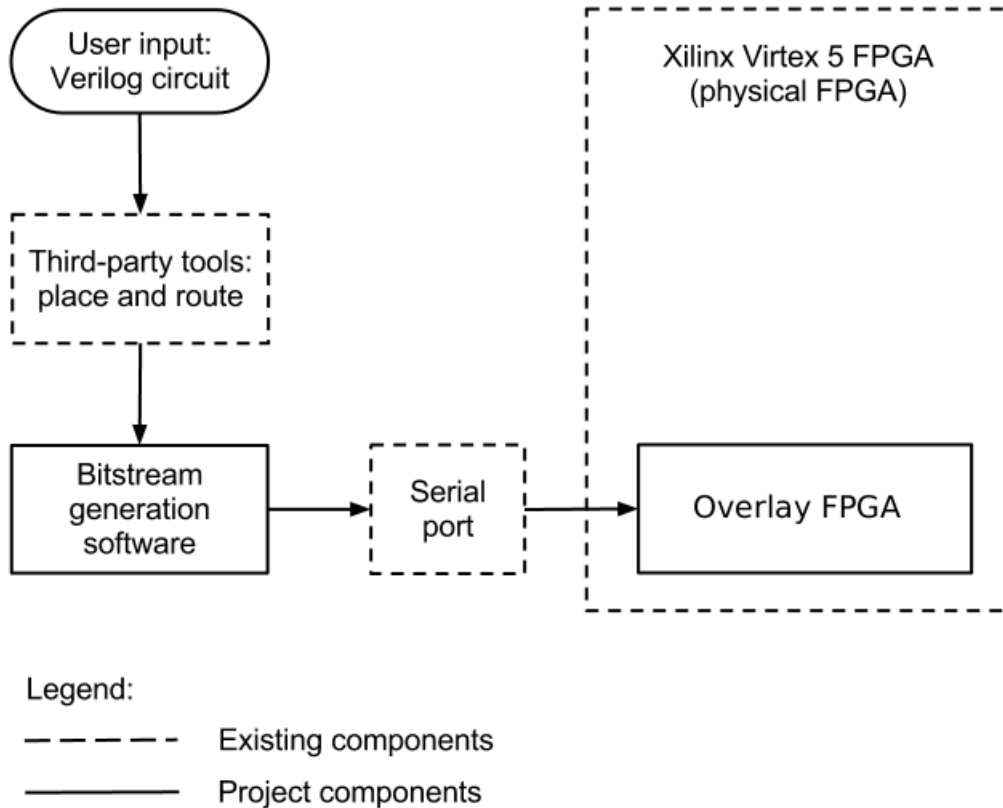


Figure 1: System overview

The Overlay FPGA can be configured to implement user-specified *test circuits* that have been placed and routed by VPR. This is achieved using the *bitstream generation and programming software* that translates the VPR output circuit into a bitstream that the overlay FPGA understands. The bitstream is then injected into the overlay FPGA via a *serial interface*. The FPGA receives and decodes the bitstream into the appropriate test circuits on the overlay.

Finally, the circuits on the overlay FPGA are controlled by connecting devices such as switches and LEDs connected through the physical FPGA.

2.2 Module design (author: K. Shi)

The *Overlay FPGA* is composed of a two-dimensional array of *Logic tiles*. The logic tiles make it easier to build a large overlay, and help keep the internal logic modules organized. Each logic tile consists of one *logic block module*, two *connection block modules*, and one *switch block module*. Figure 2 shows the internal composition of a single logic tile.

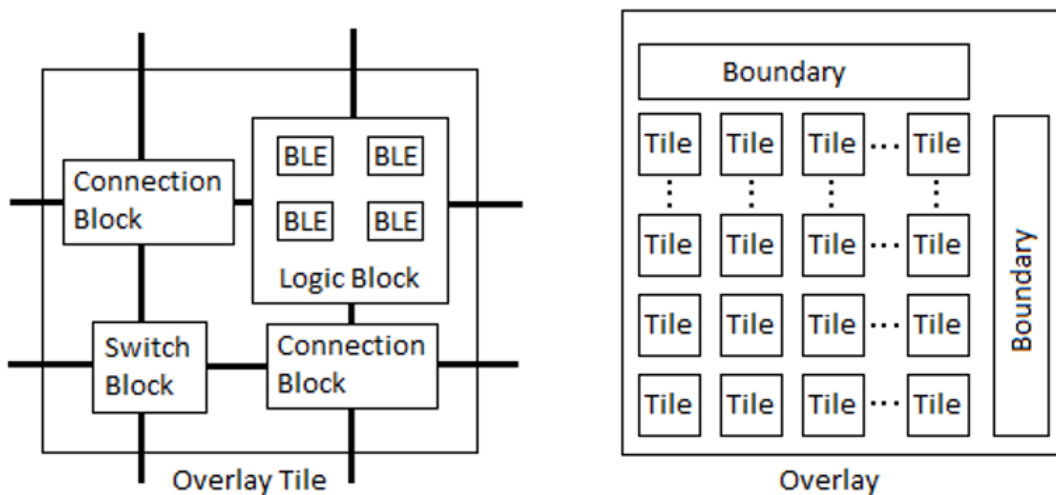


Figure 2: Logic tile and tile arrangement

The *Logic block module* consists of programmable look-up tables that perform all of the logical functionality required by the circuit. A logic block module may be composed of multiple look-up tables, the number of which can be determined by Verilog parameters.

The *Connection block* and *Switch block* modules regulate the routing of signals in the overlay.

Connection blocks connect logic block signals to buses that run throughout the overlay. *Switch blocks* control the routing between buses when they cross each other.

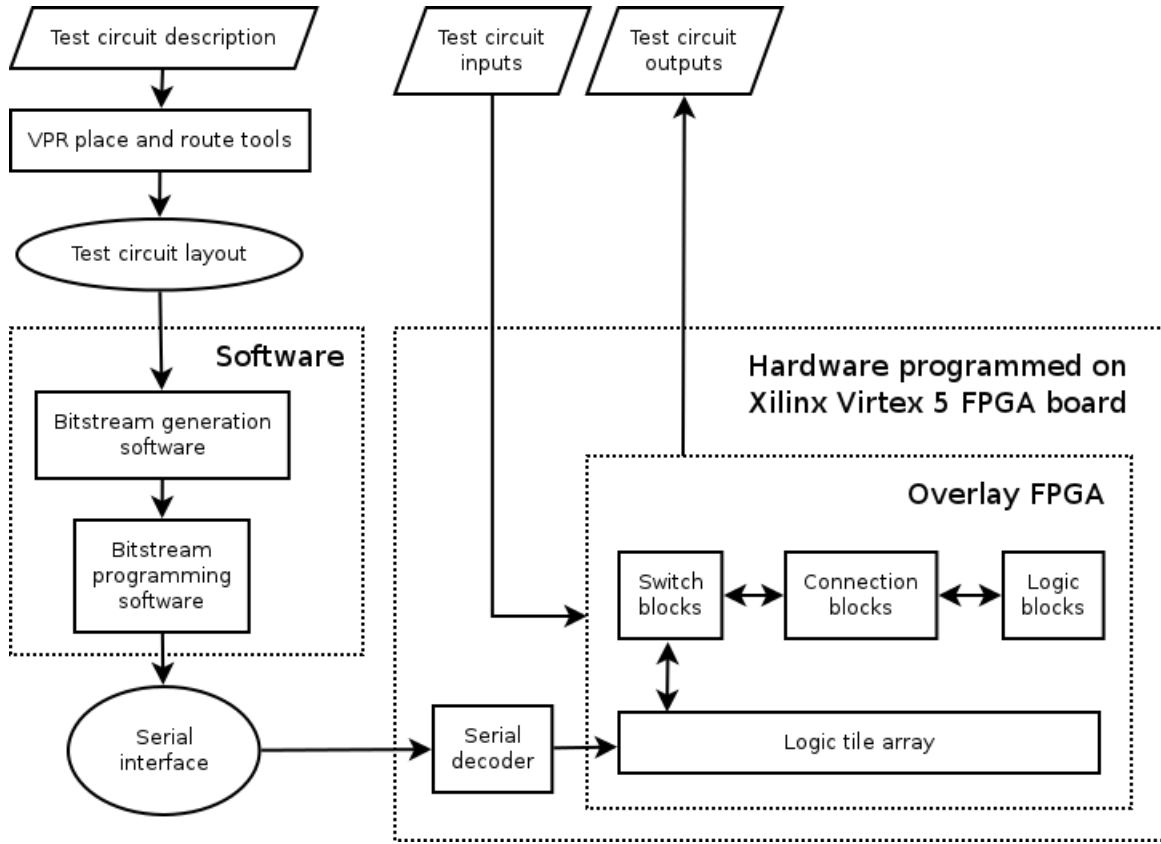


Figure 3: Module overview

Figure 3 shows the relationship between the higher-level modules.

The *Bitstream generation software* is a program that translates VPR output circuits into a bitstream capable of programming the overlay FPGA directly. The program consists of functions that parse the output from VPR and generate the appropriate bitstream from the parsed information.

The *Bitstream programming software* takes the bitstream formed by the bitstream generator and formats it for proper transmission over a serial interface.

The *Serial decoder* is a circuit attached to the overlay FPGA that receives the bitstream sent through the serial interface. It decodes and extracts the bitstream from the serial format, then injects it into the overlay circuit to configure the overlay.

2.3 Assessment of design (author: K. Shi)

The decision to take advantage of the custom 32-bit shift registers in implementing our design entails the following trade-offs versus using only basic Verilog logic:

- The design is more efficient area and timing-wise.
- The data describing the circuit (known as the *bitstream*) which is needed to program the design is larger.
- The maximum size of the design is limited by the amount of 32-bit shift registers available on the FPGA board, as opposed to the amount of flops.
- The design is incompatible with boards that do not have custom 32-bit shift registers.

We have decided that performance efficiency outweighs the negative aspects of the larger bitstream and limitation of implementation platforms. If the design is successful, adaptations can be made in the future to support the implementation of the design on more FPGA boards.


We have also looked into using a Clos network for the logic block crossbar. Currently, the crossbar is implemented using layered multiplexers built from 32-bit shift registers for each lookup table input, and is the most resource-heavy part of the tile. By replacing all the input multiplexers with a large Clos network, it is possible to reduce the number of shift registers consumed by up to 35%. However, the fact that we only have 32-bit shift registers available as building blocks severely restricts the size of the crossbar, and would have removed the flexibility of choosing the number of logic block inputs currently built into the design. Additionally, the algorithm for routing the Clos network is much more complex than the current crossbar design; the signal path to each lookup table input are dependent on each other, and cannot be routed individually. There are existing algorithms for routing Clos networks, but they focus on networks where one input can only route to one output. For our design, we need a network that can route multiple outputs from a single input. We were unable to develop our own algorithm for the Clos network due to time constraints. Because of these reasons, we decided against incorporating Clos networks into our current design. For future improvements, we may return to Clos networks.

3 Testing and Verification (author: K. Shi)

We performed 4 tests on our final design, 1 test to ensure our hardware works correctly, 1 test to verify our software, and 2 tests on the full work flow to confirm that the project is working as a whole.

3.1 Hardware Test (author: K. Shi)

To test that our overlay tiles are working correctly, we built a single tile onto the Virtex 5 FPGA and connected the inputs and outputs of the tile to the switches and LEDs of the FPGA. We then pushed in a manually composed bitstream through a serial connection to program the tile's function and routing. Once the tile was programmed, we verified the behaviour of the tile against what we expected from the bitstream, including the functions in the lookup tables, and the routing of input and output signals. The tile can be re-programmed as many times as we like to test different functions and signal routing.



```
0FFFFFFFFFFFFFFF
1800000000000000
0FFFFFFFFFFFFFFF
0888888888888888
```

Figure 4: Excerpt from a manually written bitstream showing logic element programming. Values represent 65 bit hexadecimal numbers.

Figure 4 shows the values used to program the four 6-input lookup tables and connected flip-flops. The leftmost value indicates whether or not the output of the lookup table is routed through a flop (0=no, 1=yes), and the rest determines the function of the lookup table itself. From the top, the lookup tables are programmed as follows: 6-input OR gate, 6-input AND gate with flop, always 1, 2-input AND gate.

The test showed that our tile was fully functional, behaving as expected in both lookup table functionality and correct routing. The test proved that our project satisfied the following requirements:

- H1: The tile was implemented on a Virtex 5 FPGA chip

- H3: The tile only needed to be built once on the chip, after which it was re-programmable via serial cable without the need to be re-flashed onto the chip.
- H4: The tile had connectivity to the switches and LEDs of the chip
- S2: Our bitstream sending/receiving software operated correctly in order to program the tile
- G1: The tile implemented four 6-input lookup tables

3.2 Software Test (author: K. Shi)

To test our bitstream generation software, we wrote simple circuits in verilog, then used our selected third-party software to synthesize, place and route the circuits according to our architecture. We then took the resultant output and used our software to generate a bitstream capable of programming our overlay circuit. We went through the generated bitstream manually to ensure the logic and routing matched the overlay, and that it was of the appropriate format for the overlay. Because of the size of the bitstream, this test is restricted to simple circuits on a small overlay.

The bitstream passed manual inspection; it was structured correctly and the logic and routing matched the third-party software outputs. The test proved that our project satisfied the following requirements:

- S1: The software was successful in translating VPR placement and route data into a bitstream appropriate for the overlay FPGA

3.3 Full Tool Flow Test (author: K. Shi)

Finally, to test the project as a whole, we combined the hardware and software tests, building test circuits in verilog, passing them through VPR and translating the output into a bitstream, and injecting the bitstream onto a full overlay on the Virtex 5 for testing.

We first tested the project with a simple circuit, where inputs to the overlay were connected directly to outputs. When the first circuit proved successful, we moved on to a more complex circuit, a 4-bit adder. The Verilog for the adder is shown in Figure 5.

```

module adder( in, out);

input  [7:0] in;
output [7:0] out;

assign out = in[7:4] + in[3:0];

endmodule

```

Figure 5: Verilog for adder test circuit

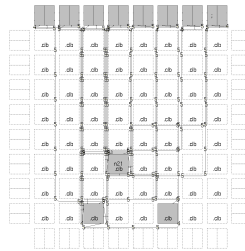


Figure 6: VPR screenshot of 4-bit adder placement and routing

Figure 6 shows a screenshot of VPR after placing and routing the 8-bit adder. The grey *.clb* boxes indicate the logic in use. The grey boxes at the top represent the IOs used for the signals *in* and *out*.

Figure 7 shows the locations in the overlay VPR has placed logic components and input/output pads. This is one of the output files our software utilizes to generate the bitstream.

```

Netlist file: adder.net Architecture file: adder.arch.xml
Array size: 8 x 8 logic blocks

#block name x y subblk block number
#-----
top^in~0 1 9 0 #0
top^in~1 2 9 0 #1
top^in~2 3 9 0 #2
top^in~3 4 9 0 #3
top^in~4 5 9 0 #4
top^in~5 6 9 0 #5
top^in~6 7 9 0 #6
top^in~7 8 9 0 #7
out:top^out~0 1 9 1 #8
out:top^out~1 2 9 1 #9
out:top^out~2 3 9 1 #10
out:top^out~3 4 9 1 #11
out:top^out~4 5 9 1 #12
out:top^out~5 6 9 1 #13
out:top^out~6 7 9 1 #14
out:top^out~7 8 9 1 #15
n21 4 3 0 #16
top^out~0 3 1 0 #17
top^out~5 6 1 0 #18

```

Figure 7: VPR placement result for adder test circuit

Figure 8 shows the routing done by VPR of one net in the overlay. Each line traces the path of the signal as it moves through the overlay, starting at the source of the signal and ending at the sink. This is part of another output file that our software uses to generate the bitstream.

The overlay functioned correctly with the adder, proving that our design meets the following requirements:

- H1: The overlay was implemented on a Virtex 5 FPGA chip.
- H2: The overlay was customized to 8x8 tiles, with a bus width of 4, logic block input per side of 4, and 4 look up tables per logic block.
- H3: The overlay only needed to be built once on the chip, after which it was re-programmable via serial cable without the need to be re-flashed onto the chip.

```

Net 13 (top^out~6)

SOURCE (3,1) Class: 1
OPIN (3,1) Pin: 18
CHANX (3,0) Track: 0
CHANX (4,0) Track: 0
CHANX (5,0) Track: 0
CHANX (6,0) Track: 0
CHANY (6,1) Track: 0
CHANY (6,2) Track: 0
CHANY (6,3) Track: 0
CHANY (6,4) Track: 0
CHANY (6,5) Track: 0
CHANY (6,6) Track: 0
CHANY (6,7) Track: 0
CHANY (6,8) Track: 0
CHANX (7,8) Track: 0
IPIN (7,9) Pad: 3
SINK (7,9) Pad: 3

```

Figure 8: Excerpt from VPR routing result for adder test circuit showing the routing for one output net, connecting a logic block output to drive an output pad

- H4: The overlay had connectivity to the switches and LEDs of the chip.
- S1: Our software generated the overlay bitstream by parsing and translating VPR placement and routing output files.
- S2: Our bitstream sending/receiving software operated correctly in order to program the tile.
- G1: The tile implements four 6-input lookup tables by using 2 multiplexed 32-bit shift registers.
- G2: The 8x8 logic grid contains 256 overlay logic elements (64 logic blocks * 4 logic elements / logic block), which is more than 100.

4 Summary (author: N. Isaac)

We were able to build the overlay circuit and program functional test circuits (such as an adder) onto the Overlay FPGA using our tool flow. This demonstrated the functionality of all parts of our project, including the overlay circuit, bitstream generation and bitstream programming.

We believe that our design is a potentially useful tool for FPGA researchers to test custom architectures or CAD algorithms on real hardware, however its usefulness depends on the research. The current design is a successful proof-of-concept; it may be useful to some researchers, but higher-level research projects may require a lower overhead. Some overhead reduction can be achieved by optimizing the structure of the overlay, by using Clos networks for example. We will outline two possible research ideas that could be implemented using our Overlay FPGA .

Modern FPGAs include a lot of fixed-function blocks (*hard macros*) that implement common functions such as multipliers and memories in order to reduce the overhead of using an FPGA for circuits containing many instances of these elements. A possible extension to our project would be to incorporate *hard macros* available on the Virtex 5 in our overlay circuit.

With our Overlay FPGA design, a researcher could easily instantiate the overlay in an FPGA-based embedded system with a *soft microprocessor* such as Xilinx's *MicroBlaze*. This would enable research on using re-programmable hardware in embedded systems. In this configuration, the Overlay FPGA could be programmed and used directly by the embedded microprocessor over its main bus.

Overall, the project succeeded in what we set out to do. We met all of our final goals and requirements, producing a flexible overlay FPGA design and tool flow that is capable of implementing a user-specified verilog circuit on the overlay.

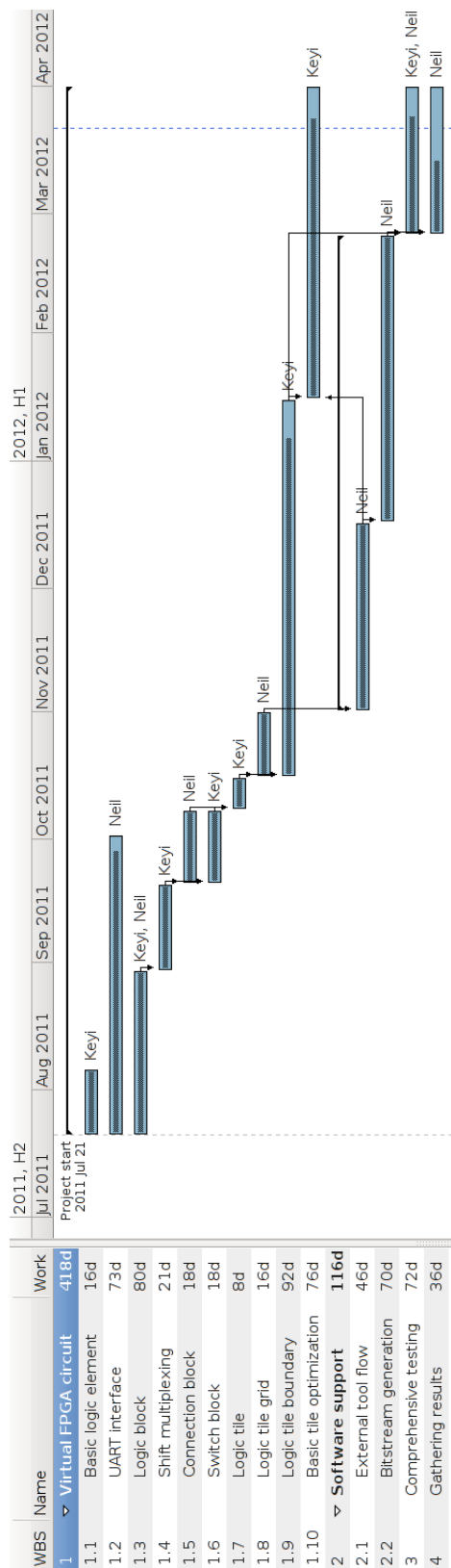
References

- [1] I. Kuon, R. Tessier, and J. Rose, “FPGA architecture: Survey and challenges,” *Foundations and Trends in Electronic Design Automation*, vol. 2, no. 2, pp. 135–253, 2007.
- [2] V. Betz and J. Rose, “VPR: A new packing, placement and routing tool for FPGA research,” in *International Workshop on Field Programmable Logic and Applications*, 1997.
- [3] V. Betz. (2011, Sep.) “A Utility to Convert a VPR netlist to Quartus format”. [Accessed: September 17, 2011]. [Online]. Available: <http://www.eecg.toronto.edu/~vaughn/vpr/download.html>

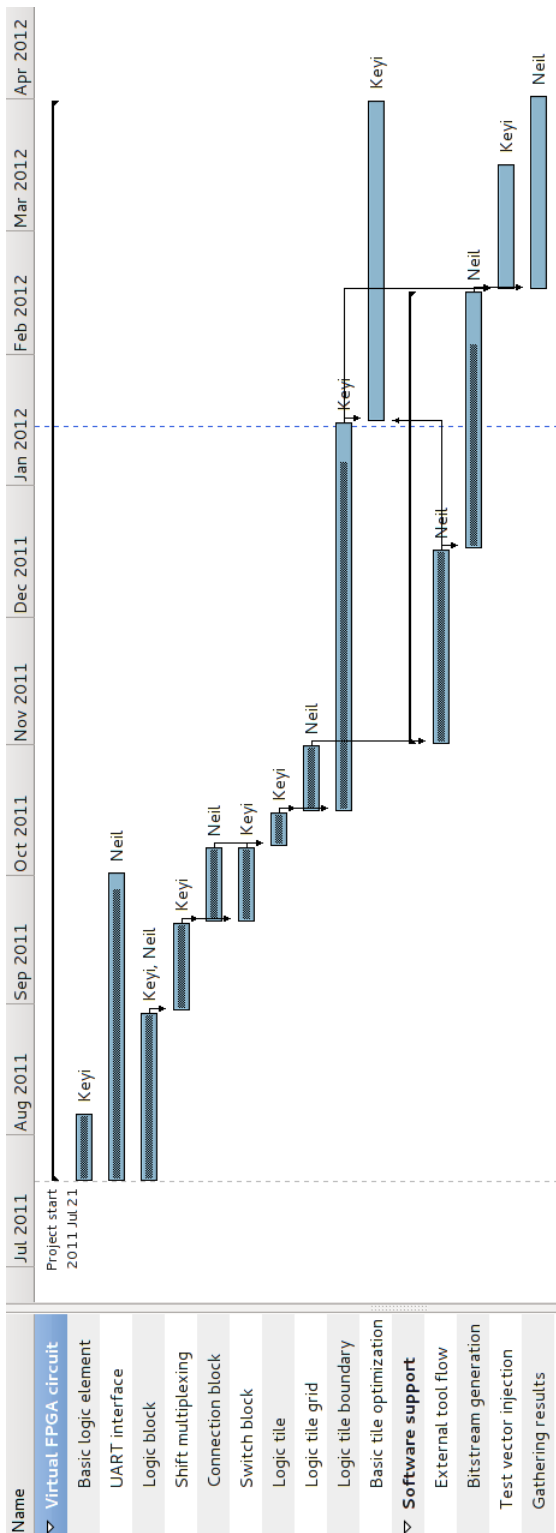
Appendices

A Gantt Chart History

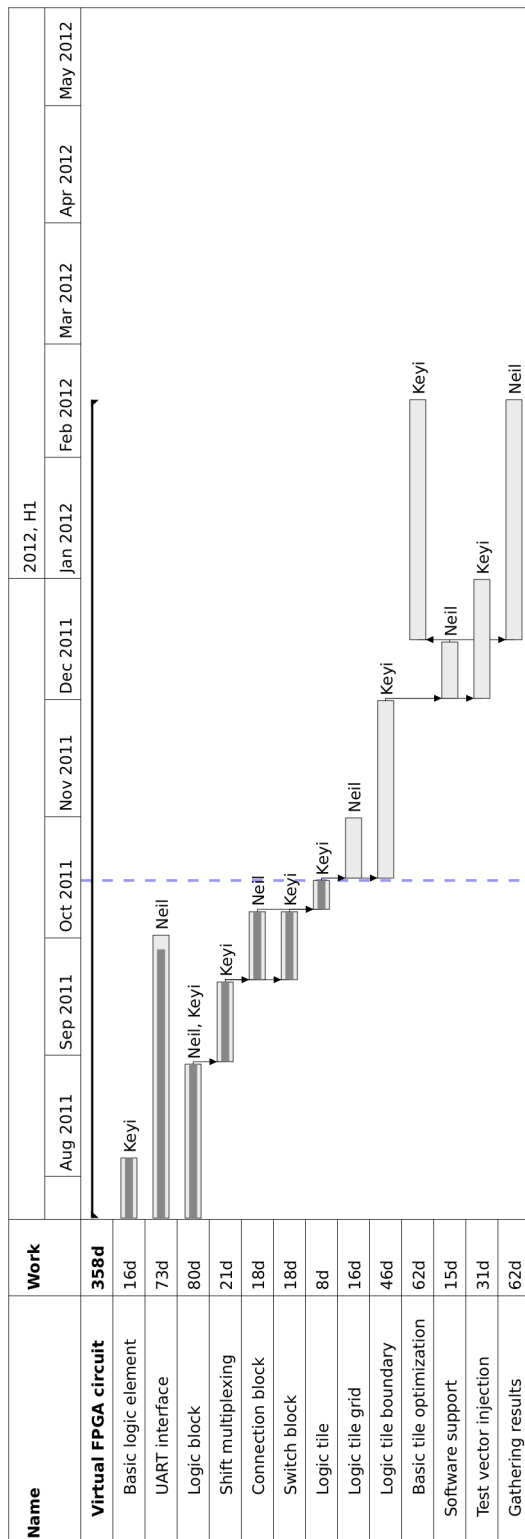
A.1 Final Gantt Chart



A.2 Gantt Chart from Progress Report



A.3 Gantt Chart from Project Proposal



B Validation and Acceptance Tests from Proposal

B.1 Functional validation

To validate the functional requirements, we will:

1. configure the Overlay FGPA and program it onto the physical FPGA,
2. select and use a benchmark circuit commonly used to test VPR,
3. configure VPR to match our architecture and dimensions,
4. place and route the benchmark circuit with VPR,
5. convert the VPR output into a bitstream for the overlay FPGA,
6. load the bitstream onto the overlay FPGA, then
7. test the functionality of the benchmark circuit running on the overlay FPGA.

The exact verification process for inputs and outputs will depend on the benchmark circuit's intended function. For the simple test circuits we will test initially, we will set inputs using hardware switches, and observe outputs on LEDs.

Intermediate outputs can also be verified throughout the development process. We can dump out the bitstream to a text file to verify that it matches the placement and routing in VPR. We can test the hardware component independently by writing a bitstream by hand as well.

B.2 Size and overhead validation

To ensure that the overhead is low enough that the overlay FPGA can fit useful circuits, we will test it using the “*Golden 20*” MCNC benchmark circuits. For each circuit, we will:

1. run synthesis and technology mapping using the ABC synthesis tool,
2. run placement and routing using VPR configured, and
3. confirm that VPR can place and route the benchmark circuit using the number and arrangement of logic blocks that we can fit.