**Carnegie Mellon University**
**School of Computer Science**
**Reconfigurable Computing Seminar**

15-828/18-847, Spring 1998 Lab 2 February 4

Due: Friday 2/21/98 11:59PM

# 1 Overview

The purpose of this lab is to gain experience implementing an algorithm on a reconfigurable computing device. For this lab you will target the PipeRench architecture and optionally a Xilinx XC40xx device. You will map and test your algorithm to PipeRench using the assembler, dataflow compiler, and simulator that have been developed for PipeRench.

You should first implement the algorithm in C, using the C implementation as a reference specification. Them use DIC as an aid in determining the structure and memory requirements of your design. This tool will also allow high level simulation to debug the general implementation you have chosen. Finally, you should implement the design in CVHASM. Once implemented you can test the implementation using CVHSIM. For extra credit, you can map the design to the Xilinx architecture.

You should hand in a report, in HTML, that describes the problem, the implementation approach, your final design, and simulation results. While the time for the lab does not permit much exploration of the design space, we do expect that you will evaluate the tradeoffs that would be possible if you had more time. The report should also evaluate the tools you used.

# 2 Approach

You should pick a kernel to implement by 2/6/97. There is a sign-up sheet outside of Seth's office, 7122 Wean Hall. If you decide to pick a kernel not listed in Section 5, you should speak with Herman or Seth about it before Friday. By Monday you should turn in (By running `handin -lab 2 proposal.html`. Use the template[1] we provide.) a proposal which describes the kernel and your initial ideas in how to map it to PipeRench.

You should next implement the kernel in C. There are no particular guidelines for your C implementation, it is mainly to provide you with a reference specification for debugging and guidance as you implement the reconfigurable design.

[1] proposal2.html

Your next step should be to describe the algorithm to `dic`, the dataflow intermediate compiler. `dic` allows the algorithm to be described in an architecture neutral manner. `dic` will schedule your algorithm onto an abstract pipeline reconfigurable architecture. Its output of interest is a C program which can simulate the design and shows the structure of the algorithm as it would be mapped onto a pipelined reconfigurable architecture. The `dic` programming guide[2] will be published on the web before 2/10/98.

With the output of `dic` to guide you, you should map the algorithm to PipeRench using CVHASM. Your goal should be to minimize the width*height product. Some tricky things to keep in mind:

- stripe state and store/restore.
- pipelining adds across stripes requires routing the carry out signal from one stripe to the next which require 1 PE in each stripe.
- 

Test out your design using CVHSIM. You can use `simwave` to debug the design–painful, but complete.

# 3 What To Hand In

- Project proposal (Due on 2/6).
- Code for your kernel in C, DIC, and CVHASM.
- The necessary .data files to run the simulation.
  - memory.data: the input memory file
  - memconfig.data: the configuration of the memory controllers.
- A write-up of the lab which describes your design and implementation. This should be in HTML and be about 1000 words with some figures. We recommend the following structure for the report.
  - Problem statement in a paragraph.
  - Implementation overview in a couple of paragraphs. References to papers and or figures should be included here as well.
  - Describe what is interesting about your design in a few paragraphs. This should include neat tricks, really hard things, architectural changes that would make life easier, etc.
  - Results in a paragraph. This should include things like maximum number of registers used, stripes, width of largest stripe, avg width, etc.

[2] dic.html

- Finally, other approaches that might be profitable.

If you implement your design on a Xilinx, hand in the verilog/vhdl/schematics, etc. Also describe the Xilinx design and compare it to the PipeRench design. Did you use features on the Xilinx that were not available on PipeRench (feedback paths, local memory, etc.)

# 4 Presentations

We will pick a few of the projects to be presented in class. We will notify the lucky winners shortly after the lab is turned in.

# 5 Kernels

Here are some kernels that we think are of an appropriate complexity for this lab. Many of these kernels are parameterized in some fashion. While not required, think about how you would generate a class of implementations depending on the parameter. For instance, the parameter for nqueens is the board size, for the systolic priority queue, the size of the queue, etc.

- **Floating Point Addition/Subtraction**
  Implement a floating point adder/subtracter for a 16-bit floating point number [6]. The breakdown of the 16-bits is: 1 sign bit, 6 exponent bits, and 9 mantissa bits.
- **Floating Point Multiply**
  Implement a floating point multiplier for a 16-bit floating point number [6]. The breakdown of the 16-bits is: 1 sign bit, 6 exponent bits, and 9 mantissa bits.
- **Systolic Priority Queue**
  Implement a priority queue on PipeRench. This structure should handle delete_min and insert operations [2]. It can have a constant size, but should detect overflow. Essentially, it keeps $k$ items sorted from min to max.
- **IDEA encryption**
  Implement one round of IDEA encryption [4]. In addition to a general solution, you should create a solution which has a non-trivial key compiled into the design.
- **Nqueens Evaluator**
  Implement a pipeline that can evaluate an nqueen position every cycle. It should return whether or not a position is valid or not. This can then be embedded in a position generator. Final implementation should handle a board at least 8x8.
- **Huffman Decoding**
  Generate a Huffman trie for ASCII text compression (not in hardware), and implement the encoder and decoder in CVHASM. Compress and decompress this PostScript document. A good reference for Huffman coding is [5].

- **Integer Square Root**
  Implement 24 bit integer square root. An FPGA-oriented pipelineable solution is presented in [3].

- **DNA Match**
  Implement the unidirectional matcher from the Hoang paper [1]. Implement both a general matcher and one which has a reasonable source string compiled in.

- **64x64 Multiplier**
  Implement a 64-bit by 64-bit integer multiplier on a stripe of no more than 10 4-bit PEs.

- **Vector-Matrix Multiply**
  Implement a vector matrix multiply on PipeRench. You wide latitude in how you do this because it requires not only work in CVHASM, but also in the data controller. This is really only for someone very familiar with Verilog/

# 6 Bugs

Remember, the tools are new and buggy. It will help everyone if you post questions, bugs, comments, hints, etc. to the class news group[3].

# References

[1] Dzung T. Hoang. *Splash-2: FPGAs in a custom computing machine*, chapter Searching genetic databases on Splash 2, pages 97–109. IEEE, 1996.

[2] Charles Leiserson. Systolic priority queues. Technical Report CMU-CS-79-115, Carnegie-Mellon University, Pittsbugh, Pennsylvania 15213, April 1979., Pittsbugh, Pennsylvania 15213, April 1979.

[3] Y. Li and W. Chu. Implementation of single precision floating point square root on FPGAs. In *Proceedings FPGAs for Custom Computing Machines*, April 1997.

[4] Bruce Schneier. *Applied Cryptography*, chapter 13, pages 321–322,638–641. ?, 1996.

[5] Robert Sedgewick. *Algorithms in C*, chapter 22. ?, 1999.

[6] N. Shirazi, A. Walters, and P. Athanas. Quantitative analysis of floating point arithmetic on FPGA based custom computing machines. In *Proceedings FPGAs for Custom Computing Machines*, April 1995.

[3] news:cmu.cs.class.cs828.discuss