1 Introduction and Motivation

Circuit design can be separated into two major portions, datapath and control unit. Datapath’s size and the complexity of its control unit holds a certain relationship to one another. One can says that the bigger the datapath size, the easier the control unit becomes. A basic example of this is when comparing a K-bit parallel adder versus a K-bit sequential adder. As you can imagine, parallel adder requires no control unit at all as everything is combinational. On the other hand, sequential adder requires some rudimentary control logic such as the counter to indicate the end of operation.

Indeed, datapath’s size and control unit’s complexity are inversely proportional to each other. As lower area generally translates to lower cost of production and better efficiency in terms of power, hardware designers almost always face the problem of designing complex control unit. This problem is especially true in the design of low-area cryptographic devices such as a hash function. Simple operations such as permutation can become very complex when operates in a very low area environment. This can result in a very large control unit that definitely not desirable for low-area design.

The process to manually develop a controller tailored toward area critical design is very tedious and plague with errors. A simple bug in the compaction or minimization process can destroy several days, or weeks, worth of work. Furthermore, the final controller can vary from person to person even though they
are designing for the same datapath. This is due to the varying degree of the
designer’s perspective, area reduction approach, hardware design and language
proficiency.

All the aforementioned problems can be easily solve if a specialized tool exists
to automated and standardize the minimization process. This project aims to
create such a tool to address the problems to speed-up the time needed for
controller’s implementation and minimization.

2 Design Concept

The tool accepts a spreadsheet containing microcoded instructions. Then,
a user can manually select a technique or a set of techniques to be applied to
the microinstructions. Alternatively, the tool should be able to read a script
provided by a user. Finally, an optimized output of the instructions should be
able to automatically convert the minimized controller to HDL codes.

3 Problems

Foreseeable problems are as followed:

– A spreadsheet format that is flexible and can be easily read by the tool as
  well as a user can be hard to develop and standardized.
– Implementing various compaction and minimization techniques can become
  a problem as they have to be a generic solution.
– Create an object structure that can be easily manipulated by the tool is a
  challenge.
– Tailored the compaction techniques toward specific hardware platform such
  as 4-input LUT FPGA, 6-input LUT FPGA, and ASIC is difficult.
4 Project Timeline

- 03/31 - 04/04: Literature Reviews
- 04/05 - 04/12: Core program development
- 04/13 - 04/25: Implement various compaction techniques
- 04/26 - 05/03: Code generator
- 05/04 - 05/10: Fine-tuned / Project Reports

5 Literature Search

- Microcode Compaction and Minimization Techniques
  - [7] - Local Microcode Compaction Techniques
  - [8] - Minimisation of control store width in digital systems
  - [12] - Novel state minimization and state assignment in finite state machine design for low-power portable devices
  - [10] - A new approach in microcode optimization
  - [1] - A development environment for horizontal microcode
  - [3] - Clustering Based Microcode Compression
- Implementation and Experiments on Compaction techniques
  - [13] - Designing a microcode synthesis system
  - [4] - Local Microcode Compaction for Horizontal Machines
- Miscellaneous
  - [9] - Compaction techniques for pipelined architectures (Realistic Scheduling)
  - [6] - State reduction on incompletely specified FSM
  - [2] - Genetic algorithms for microcode compaction
6 Tentative Final Table of Contents

1. Introduction and Motivation
2. Previous Work
3. Tool’s Framework
   - Input Format Descriptions
   - Data structure
   - Output Format Descriptions
4. Implementation methodology
   - Description of compaction and minimization techniques used in the tool
   - Specific platform implementation style
   - Code generator description
5. Case study
   - Compact Skein
   - Additional case study
6. Comparison and Summary
7. Future Improvement
8. References
References


