

Review of:

Self Modifying Circuitry

- Paper by:
 - Adam Donlin (Edinburgh)
- Published in:
 - FPL 1998
- Original copy on-line as:
 - <http://www.arl.wustl.edu/~lockwood/class/cs6812/adam-fpl98.pdf>
- Survey by:
 - Dave Lim

Style of the Paper

- Limitations of existing virtual circuitry
- Introduce Flexible URISC
 - URISC
 - Flexible URISC
- Why use self-modifying circuits (move to introduction)
- Implementation
- Performance results
- Programming Environment

Introduction

- Why use self-modifying circuit?
 - FPGA device densities still limited
 - Configuration penalties remain high
- What is Virtual Circuit?
 - A means for circuit to modify itself
 - Bunch of bit files for different modules (Swappable Logic Units or SLU) kept in RAM, circuit decides which one is needed, pulls it out and programs FPGA with it

Introduction (Continued)

- Typical virtual circuitry system
 - FPGA couple to a host processor via a host peripheral bus
 - FPGA works exclusively as a slave processor
 - FPGA is managed by a software component running on the host processor
- Limitations:
 1. Technical bandwidth limitation imposed by the host peripheral bus
 2. Penalties incurred in traversing the hardware/software boundary
- Flexible URISC (Ultimate RISC) addresses those limitations

Ultimate Reduce Instruction Set Computer

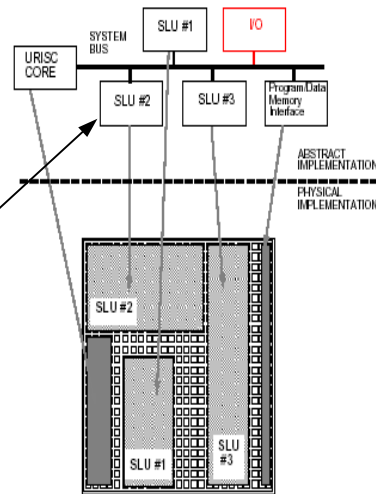
- One instruction:
 - Move memory to memory
- Computation is achieved by migrating devices onto the system bus, then mapping those devices into memory space of the URISC processor core
 - e.g. the ALU is not in URISC core, but on a system bus, and I/O registers of ALU is memory mapped into the memory space of URISC core

URISC (continued)

- Advantage
 - Very little resource requirements for the control and data paths
- Disadvantage
 - Not very *flexible*

Flexible URISC

- Flexible URISC allows more Logic Units to be available to user than there is room on the die by swapping in Logic Units that are needed from RAM, hence Swappable Logic Units
- Uses FastMap™ of Xilinx XC6200 series



Flexible URISC (continued)

- Things of note:
 - There is not explicit system bus in Flexible URISC – addresses limitation one
 - Flexible URISC core does not do any processing, it acts more as a “communication agent”, transferring operands between SLUs and memory
 - Allows placement of SLUs on virtually any unoccupied space, can be irregularly shaped

Implementation

- Implement Prototype of Flexible URISC core, system specifications:
 - XC6200
 - Dedicated SRAM with PCI bus interface logic
 - In addition to SLU I/O registers and SRAM, also memory map configuration memory of FPGA into Flexible URISC core
 - To reconfigure simply move bits from one memory space (SRAM) to the different memory space (configuration memory), makes reprogramming simple – addresses limitation two
 - 16 stage machine

Analysis and Results

- Test program : repeatedly move a constant from position 0 to position n
- Note that bottleneck moved from host peripheral bus to the bus between Flexible URISC and SRAM
- Right now, processor is unpipelined, only issues 4 memory transactions / 16 cycles

Analysis and Results

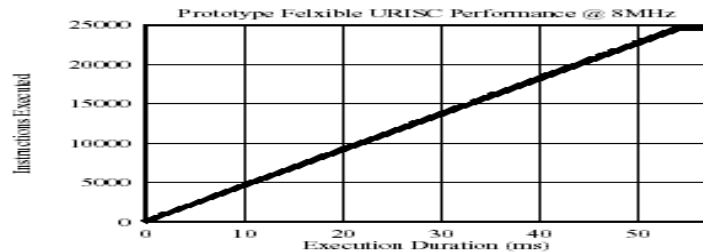


Figure 4: Performance Results of the Prototype Implementation

Clock Speed	Instns per Sec	Mem Cycles per Sec	Bandwidth (MB/s)
8 MHz	500,000	2,000,000	8
16 MHz	1,000,000	4,000,000	16
32 MHz	2,000,000	8,000,000	32

Programming Environment

- Programming model
 - Very cumbersome and error if user had to program solely using a single move instruction and absolute addressing mode
 - Suggested alternative : functional programming language e.g. `sum [1..10]`

Programming Environment (continued)

- **Static Environment**
 - One program: interleave configuration and application commands
- **Dynamic Environment**
 - Two programs
 - System program – service system calls
 - User program – make system calls

Future Work

- Develop programming environment
- Look into other applications for Flexible URISC
 - Active networking