On The
# Softer Side
Configurable Digital Signal Processing

# Micro-Sequencer Approach Speeds Reconfiguration

Not all reconfigurable computing schemes were created equal. A micro-sequencer architecture offers faster performance and greater flexibility.

Roozbeh Jafari, Graduate Student
Henry Fan, Graduate Student
Majid Sarrafzadeh, Professor
UCLA Computer Science Department

Key military applications such as image processing, sonar/radar and SIGINT can enjoy significant performance gains by using reconfigurable computing. Therefore, it's perhaps surprising that the technology has been so slow to gain broad acceptance. Among the barriers is a lack of clear understanding about its real performance differences compared to traditional compute architectures.

Work done at ULCA's Computer Science department paints a clearer picture and proposes a method to enhance the performance of reconfigurable computing. Researchers there devised the concept of a micro-sequencer-based reconfigurable system, and introduced with it the concept of Quick Reconfiguration. The approach exploits similarities among various applications to save on reconfiguration time.

Using the micro-sequencer architecture, the group demonstrated that a significant increase in speed is gained by reconfiguring a system on a set of image-processing benchmarks. The benchmarks took merely hundreds of microseconds versus the hours needed in traditional FPGA reconfigurations. Using this architecture also makes the parameters of a system, for instance the size of data bus, easy to modify in order to customize the system for a specific application. The results, detailed later, show that power consumption and silicon area are reduced by 72% and 77% respectively, by using a customized 8-bit data bus versus a 64-bit data bus, while the speed is improved by 157%.

## FPGAs Make it Possible

Before getting into the details of the micro-sequencer scheme, it's helpful to examine the underlying hardware that makes reconfigurable computing possible. Programmable system capability forms the heart of reconfigurable computing. Programmable devices including FPGAs contain an array of programmable computational units that can be programmed through the configuration bits. This gives us the flexibility of having dedicated hardware to perform specific computational units combined with a parallelism capability.

Reconfigurable systems provide the flexibility and reuse of hardware for multiple applications. Reconfigurable hardware can be used to execute designs that are larger than the available hardware resources. In such cases, a part of a large application is executed on the hardware. By reusing the reconfigurable hardware, the remaining tasks of the application can be loaded and executed on the hardware at runtime. This is known as runtime reconfiguration. Another issue that necessitates the integration of reconfiguration in a hardware platform is that some applications require reconfiguration in different abstraction levels of the system. For example, some applications require different variations of an algorithm to execute their task. A non-flexible hardware realization for such applications has to fit all required algorithm variations on the die. This, if possible, makes the design and fabrication processes more complicated and expensive.

## Dealing with Delays

A major drawback of using runtime reconfiguration is the significant delay of reprogramming the hardware. The total runtime of an application includes the actual execution delay of each task on the hardware along with the total time spent for hardware reconfiguration between computations. The latter might dominate the total runtime, especially for classes of applications with a small amount of computation between two consecutive reconfigurations. Hardware reconfiguration often takes hundreds of milliseconds or longer based on the size of the application. To reduce the reconfiguration overhead, some previous works have used different approaches.

In many applications, only a small portion of the design changes at a time and the entire hardware does not have to be reconfigured. This has led the industry to add the capability of Partial Reconfiguration to some of their recent products. FPGAs are examples of such reconfigurable hardware, and some of the recent FPGA devices have the capability of partial runtime reconfiguration.

Another method used to gain speed is called configuration prefetching. This method tries to overlap the computation with the reconfiguration of the hardware.

Therefore, to maximize this overlap, it seeks a way to minimize the chance that reconfiguration is prefetched falsely.

Configuration compression is another approach to minimize the reconfiguration time. In this method, the configuration delay is reduced by compressing the data transferred from the host computer to the programmable system. A major portion of delay is due to the distance between the host computer and the programmable device. Reconfiguration can be accelerated by using a fast memory (configuration cache) near the reconfigurable array. This method is called configuration caching.

## Speedy Micro-Sequencing

In Micro-Sequencer-based Quick Reconfiguration (MSQR), the reconfiguration is performed by altering the algorithm loaded onto the memory of the proposed architecture. Also new instructions can be generated by modifying the microcodes stored in the control unit. Using microcoded architecture in the control unit of the micro-sequencer facilitates the parallelism and adds new computational units to the datapath. In this case, the control unit needs the minimum modification since it is highly regular compared to sequential-machine-based controllers.

Efficient reconfiguration of an FPGA is a critical issue because of the time overhead involved. Sometimes in order to reconfigure a system from one algorithm to another, the processes of synthesis, placement and routing have to be performed, which are highly expensive in terms of CPU time and may take hours to complete. In the micro-sequencer approach, when a system is to be reconfigured, only the new algorithm has to be loaded onto the memory micro-sequencer. When new instructions are required for the new algorithm, the memory structure is updated. This is the Micro-Sequencer-based Quick Reconfiguration method. Because this method does not require physical reconfiguration, a significant speed increase is gained in the process.

## MSQR in Image Processing

MSQR is applicable to those applications where the types of computations do not vary substantially from one task to another. The motivation of this research is to enhance the reconfiguration process for image-processing algorithms. It's important, therefore, to verify if image-processing algorithms are altered substantially during the reconfiguration process.

Most image-processing algorithms are computationally intensive and should be executed on hardware resources to allow real-time processing. Moreover, these algorithms change in nature and parameters based on information available from targets. For instance, in a feature-tracking algorithm, the number of targets, their position and their distance to the camera can change the algorithm (or its parameters) to increase the efficiency of tracking motions. As a result, image-processing algorithms are proper candidates for mapping onto reconfigurable resources. This not only provides fast running time, but also allows dynamic modification of the algorithm through runtime reconfiguration. Both cannot be achieved by mapping this type of algorithm onto traditional fixed software or hardware platforms.

The micro-sequencer architecture makes use of microcode architecture for the design of the control unit. In this approach, the relation between inputs and outputs is treated as a memory system. Control signals are stored as words in a microcoded memory. At each clock tick during instruction execution, the appropriate (micro) control word is fetched from microprogram memory to supply the control signals.

The microcode control unit itself is a small stored program computer. It has a micro program counter, a microprogram memory and a microinstruction word, which contains the control signals and sequencing information. The action of the microcode control unit is exactly like that of a general-purpose computer: fetch a microinstruction, execute it by applying the control signals in the control word to the computer's datapath, determine the address of the next microinstruction, and fetch the next instruction. Figure 1 shows a block diagram of a typical design of a microcoded control unit.

The microprogram counter contains the address of the next microinstruction to be fetched from the control store, a fast local memory that contains the control words. The control word is copied into the microinstruction registers. The control store consists of microinstructions that control the datapath directly.

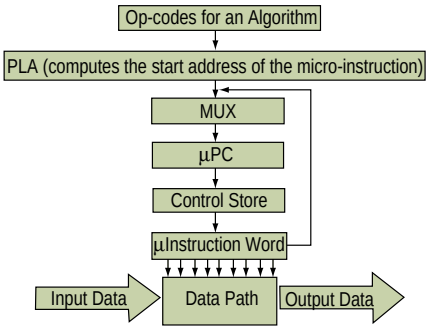


Figure 1

The block diagram depicts a typical design of a microcoded control unit. The microprogram counter contains the address of the next microinstruction to be fetched from the control store, a fast local memory that contains the control words. The control word is copied into the microinstruction registers.

Because image-processing algorithms tend to have similar computational behavior, it can be inferred that the capabilities of the datapath satisfies the new algorithm and it does not have to be modified. However, since the algorithm changes, the opcode part of the micro-sequencer, which contains the instructions for a specific algorithm, has to be updated. This process can be simply done by writing the new algorithm (the new instructions) onto the memory. This approach is significantly faster than traditional physical reconfiguration.

Sometimes, due to the constraints of new algorithms, the order of utilization of components in the datapath may have to be altered, or further, instantiation of a new computational unit may be inevitable. In this case, the microcodes in the control unit can be easily modified to satisfy the new requirements. Meanwhile, a new computational unit is added to the datapath. This achieves higher performance because of utilizing parallelism in computationally intensive algorithms. MSQR is therefore effective both in terms

Horizontal Microcode Format

| Micro-instruction register Index | Signal Name |
|---|---|
| 00 | ENDbit |
| 01 | PCena |
| 02 | Cout |
| 03 | MDout |
| 04 | Rout |
| 05 | MAin |
| 06 | MDin |
| 07 | Cin |
| 08 | PCin |
| 09 | IRin |
| 10 | Ain |
| 11 | Rin |
| 12 | INC4 |
| 13 | RD |
| 14 | WR |
| 15 | ADD |
| 16 | GRa |
| 17 | GRb |
| 18 | GRc |

Table 1

Shown here is the microinstruction format for the MSQR architecture. A few more bits are reserved for newly inserted components in the datapath.

of providing flexibility and performance.

## Microinstruction Format is Key

Microinstructions are an important part of the controller and have to be defined so that the maximum flexibility is gained for reconfiguration. There are two types of microinstructions: horizontal and vertical. In the horizontal microcode, each bit represents a control signal for a component in the datapath. In the vertical microcode, however, the control of similar components in the datapath is compacted in a group of bits in microinstructions.

This requires a local decoder to generate all control signals usable for components in the datapath. The advantage of the horizontal microinstructions is its lower access time to reach the designated component in the datapath. However, as the size of the datapath grows, the number of control bits increases resulting in larger microinstructions. Table 1 shows the microinstruction format for the MSQR architecture. A few more bits are reserved for newly inserted components in the datapath. The instruction set format of the MSQR design is very similar to a RISC instruction set. It has a fixed 32-bit instruction width. The details of the instruction format are shown in Figure 2.

In order to support the template generation, a better format is one with four register index operands. The op-code of this instruction is specified by the template generator. In order to support the quick reconfiguration, the micro-sequencer machine needs to support the new instruction set whenever there is a new hardware implementation. Figure 3 shows the fields of the new instruction format. The UCLA team also implemented an assembler for its proposed instruction set. The syntax was chosen such that parsing is minimized.

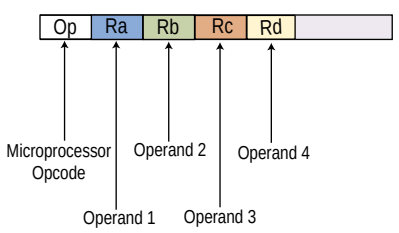


Figure 3

Shown here are the fields of the MSQR instruction format. The first field is the op-code of the instruction. The second to the fifth fields are the indices of the operand registers.

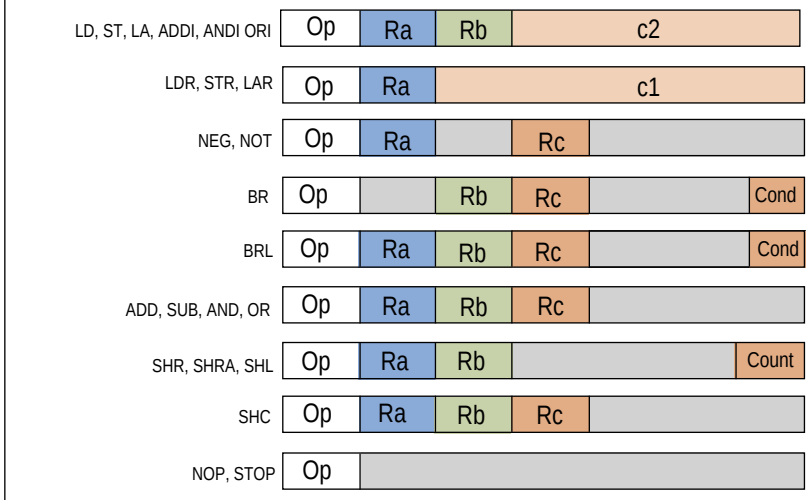


Figure 2

Similar to a RISC instruction set, the instruction set format of the MSQR design has a fixed 32-bit instruction width. The first field is the operation code (op-code) by which the type of operation is determined. The second, third and forth fields are the register indices. Some instruction, for example, ADD, SUB and BRL have three indices, but some only have one.

## Experimental Results

The UCLA team implemented the micro-sequencer in VHDL. The code was written structured and generic so that the size of the data bus/address bus can be easily modified. The reconfiguration times of two image-processing algorithms were measured: once using the traditional physical reconfiguration and once using the novel method of MSQR on a Wildstar/PCI board. The results are shown in Tables 2a, 2b and 2c.

To show the flexibility of the proposed architecture, a background subtraction algorithm, which requires an 8-bit data bus, was implemented on various micro-sequencers with 8, 16, 32, 64 and 128-bit data buses. The goal was to measure power, area and longest path delay for all above variations. This experiment was done using Synopsys' Power Compiler.

Traditional Reconfiguration Time Breakdown

| Algorithm Process | Feature Selection | Background Subtraction |
|---|---|---|
| Synthesis (sec) | 176 | 54 |
| Placement and Routing (sec) | 1514 | 270 |
| Programming FPGA (msec) | 621 | 220 |

Table 2a

Quick Reconfiguration Time

| Algorithm | Feature Selection | Background Subtraction |
|---|---|---|
| # of Instructions | 387 | 131 |
| MSQR Time (µsec) | 42 | 30 |

Table 2b

Reconfiguration Time vs. Quick Reconfiguration Time

| Algorithm | Feature Selection | Background Subtraction |
|---|---|---|
| Traditional Reconfiguration Time (sec) | 1690.621 | 322.220 |
| MSQR Time (µsec) | 42 | 30 |

Table 2c

Table 2

The UCLA team measured reconfiguration times of two image-processing algorithms: once using the traditional physical reconfiguration and once using the MSQR method on a Wildstar/PCI board. The tables show clear speed advantages for MSQR.

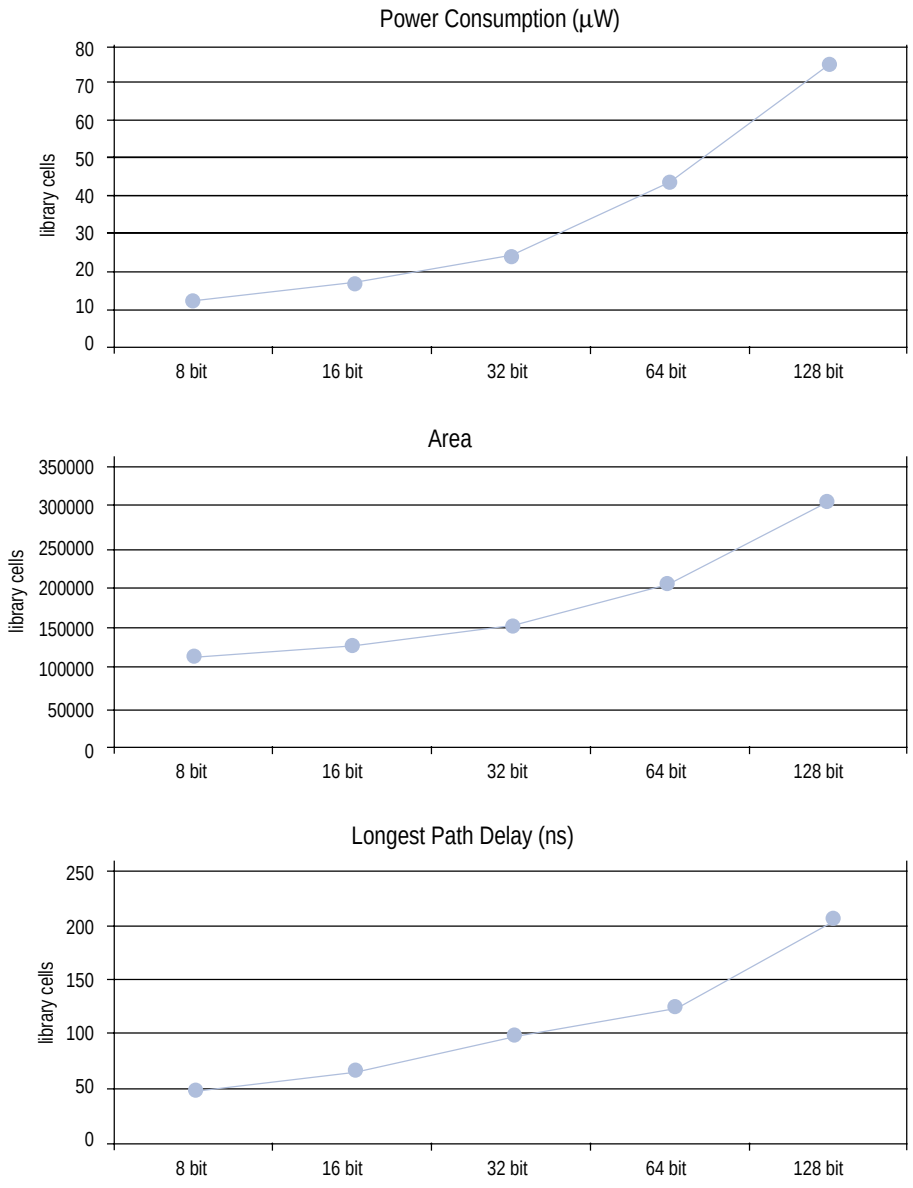


Figure 4

In order to illustrate the flexibility of the MSQR architecture, the UCLA team ran a background subtraction algorithm, which requires 8-bit data bus, implemented on various micro-sequencers with 8, 16, 32, 64 and 128-bit data buses. They measured power, area and longest path delay for all above variations.

The measured power, area and data arrival times are shown in Figures 4a, 4b and 4c.

A micro-sequencer-based architecture enhances the efficiency of reconfiguration on an FPGA. The experimental results prove that the time needed to reconfigure the control unit of a system is far less than the time it takes to physically reconfigure the whole system. Besides the reconfiguration time, the micro-sequencer-based system provides flexibility for data bus/ address bus design.

The UCLA team demonstrated this point by modifying the size of the data bus for a specific application. Such features can be exploited when the algorithms are unknown at the stage of micro-sequencer design. A significant improvement in power consumption, speed and silicon area is gained by providing this flexible feature. Moreover, MSQR provides an efficient way for reconfiguration with a considerable increase in speed in the reconfiguration process versus the traditional physical reconfiguration. ■■

UCLA Computer Science Dept.
Los Angeles, CA.
(310) 825-3886.
[www.cs.ucla.edu].