DesignCon 2003

System-on-Chip and ASIC Design Conference

A Programmable System with Quick Reconfiguration

Roozbeh Jafari
Henry Fan
Majid Sarrafzadeh

Computer Science Department
University of California, Los Angeles

Abstract

This paper presents a Micro-Sequencer based reconfigurable system. We introduce the concept of Quick Reconfiguration and compare it with existing "full reconfiguration" schemes. In a typical scenario, reconfiguring a system from one application to another does not require a "full" reconfiguration. Instead we can exploit similarities among various applications to save on reconfiguration time. This task can be accomplished via micro-programming of a micro-sequencer architecture. We show that a significant speed up is gained by reconfiguring a system on a set of image processing benchmarks, which takes hundreds of μseconds versus hours in traditional FPGA reconfigurations. Furthermore, by using this architecture, it has been shown that other parameters of system, for instance the size of data bus, can be easily modified to customize the system for a specific application. This results in significant improvements in power consumption, speed and silicon area. Experimental results show that power consumption and silicon area are reduced by 72% and 77% respectively by using a customized 8-bit data bus versus 64-bit data bus while the speed is improved by 157%.

Roozbeh Jafari

Roozbeh Jafari received his B.S. in Electrical Engineering in 2000 from the Sharif University of Technology, Tehran, Iran. He joined the State University of New York at Buffalo,NY in 2000 and received his M.S. in Electrical Engineering in 2001. His main research focus was VLSI Testing and Verification. He also worked at IBM, Endicott, NY on development of the IBM TestBench tool designed for VLSI Testing. He is currently pursuing his Ph.D. degree in Computer Science at the University of California, Los Angeles, CA. He is a member of the ER group working under the supervision of Professor M. Sarrafzadeh.

Henry Fan

Henry Fan received his B.S. in 1998 from University of California at Berkeley. In 2001, he joined the ER group in UCLA. He worked under the supervision of Professor Majid Sarrafzadeh and received his M.S. in 2002. He research topics were mainly focused on reconfigurable devices and high-level synthesis.

Majid Sarrafzadeh

Majid Sarrafzadeh (http://www.cs.ucla.edu/~majid) received his B.S., M.S. and Ph.D. in 1982, 1984, and 1987 respectively from the University of Illinois at Urbana-Champaign in Electrical and Computer Engineering. He joined Northwestern University as an Assistant Professor in 1987. In 2000, he joined the Computer Science Department at University of California at Los Angeles (UCLA). His recent research interests lie in the area of Embedded and Reconfigurable Computing, VLSI CAD, and design and analysis of algorithms. Dr. Sarrafzadeh is a Fellow of IEEE for his contribution to "Theory and Practice of VLSI Design". He received an NSF Engineering Initiation award, two distinguished paper awards in ICCAD, and the best paper award in DAC. He has served on the technical program committee of numerous conferences in the field.

Professor Sarrafzadeh has published approximately 250 papers, is a co-editor of the book "Algorithmic Aspects of VLSI Layout" (1994 by World Scientific), and co-author of the book "An Introduction to VLSI Physical Design" (1996 by McGraw Hill). Dr. Sarrafzadeh is on the editorial board of the VLSI Design Journal, an Associate Editor of ACM Transaction on Design Automation (TODAES) and an Associate Editor of IEEE Transactions on Computer-Aided Design (TCAD).

1. Introduction

There exists two methods to execute algorithms on hardware. One is to use hard-wired technology such as Application Specific Integrated Circuit (ASIC) or a group of components to perform an algorithm in hardware. ASICs are designed to perform a given computation and thus they are very fast with specific applications using the exact designed computational units. However, they cannot be altered after the design and hence algorithms which require new operations (computational units) cannot be performed as fast as others. The other method is to use software-programmed microprocessors. Processors execute a set of instructions. This makes a system flexible and by changing the software, functionality of system alters. However, the downside of this flexibility results in performance decrease. To execute instructions, the tasks of fetching, decoding and execution have to be performed which creates in a high execution overhead.

A programmable system fills the gap between hardware and software. It achieves a better performance than software while providing a more flexible solution than hardware implementation. Programmable devices including Field Programmable Gate Array (FPGA) contain an array of programmable computational units which can be programmed through the configuration bits. This gives us the flexibility of having dedicated hardware to perform specific computational units and meantime it can be designed with parallelism capability.

Reconfigurable systems provide the flexibility and reuse of hardware for multiple applications. Reconfigurable hardware can be used to execute designs, which 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.

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 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 up is called Configuration Prefetching [1]. This method tries to overlap the computation with reconfiguration of the hardware. Therefore,

to maximize this overlap, it seeks a way to minimize the chance that reconfiguration is prefetched falsely.
Configuration Compression [2][3][4][5] is another approach to minimize the reconfiguration time. In this method, the configuration delay is reduced by compressing the data transferred from host computer to the programmable system.
A major portion of delay is due to the distance between host computer and programmable device. Reconfiguration can be accelerated by using a fast memory (configuration cache) near reconfigurable array. This method is called Configuration Cashing [6][7].
This paper presents a novel approach for quick reconfiguration via a micro-sequencer. In this method (Micro-Sequencer based Quick Reconfiguration – MSQR), the reconfiguration is performed by altering the algorithm loaded onto the memory of proposed architecture. Also new instructions can be generated by modifying the microcodes stored in the control unit. Further details on the architecture of micro-sequencers will be given in Section 3. Furthermore, using micro-coded architecture in the control unit of the micro-sequencer facilitates the parallelism and adding 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.
In Section 2, our novel method for reconfiguration is introduced. Section 3 illustrates the micro-sequencer architecture proposed for reconfiguration. Section 4 proposes a heuristic for quick reconfiguration which increases the overall performance of the system by providing flexibility in design and size of data bus/address bus. Moreover, the experimental results are described in Section 4 and finally conclusion is discussed in Section 5.

2. A NOVEL METHOD FOR RECONFIGURATION

The traditional computer consists of a central processing unit (CPU) and a main memory. CPU is further divided into a control unit, a datapath and a memory. The structure of the memory and data path unit is regular and well-organized while the control unit structure is irregular and global. In this paper, we explore the regularity of the datapath for certain algorithms. We will suggest that such regularity will help us to design a control unit by which reconfiguration can be easily done.
As we mentioned in Section 1, efficient reconfiguration of an FPGA is a critical issue because of the time overhead of reconfiguration. Sometimes in order to reconfigure a system from an 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 our micro-sequencer approach, when a system is to be reconfigured, only the new algorithm has to be loaded onto the memory micro-sequencer. Further, in case new instructions are required for the new algorithm, the control store of control unit which has memory structure is updated. This method is called Micro-Sequencer Based Quick Reconfiguration (MSQR). Since this method does not require physical reconfiguration, a significant speed up is gained in the process of reconfiguration versus full reconfiguration.

2.1 MSQR IN IMAGE PROCESSING ALGORITHMS

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, hence, it has to be verified if image processing algorithms alter substantially during the reconfiguration process.
Most of 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 feature tracking algorithm, the number of targets, their position and their distance to camera can change the algorithm (or its parameters) to increase the efficiency of tracking motions. Therefore, 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 run time reconfiguration. Both cannot be achieved by mapping this type of algorithms onto traditional fixed software or hardware platforms.
Consequently, first it has to be verified that image processing algorithm are similar in terms of computational behavior i.e. they use similar computational units. A brief study on various MATLAB image processing functions verifies this.

Function	(+)	(*)	(/)
area3D	33%	67%	0%
PSNR	100%	0%	0%
findCircles	78%	11%	11%
rgb2ind	41%	45%	14%
Opthr	50%	20%	30%

Table 2.1 Instruction Breakdown for some image processing functions

As shown above, image processing algorithms mostly use the same type of computational units. As a result, they are good candidates for MSQR. Also significant speed up can be gained by implementing parallel computation due to the nature of these algorithms.

3. MICRO-SEQUENCER ARCHITECTURE

Micro-sequencer makes use of microcode architecture for design of the control unit. In this approach, the relation between inputs and outputs are 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 concept of microcoded control units originated early in the history of computing. Maurice Wilkes proposed the concept in 1951, as a way of simplifying the control logic of a computer. Although Wilkes did construct a machine, the EDSAC2, with a

microcoded control unit, it was not until the early 1960s that IBM made the concept popular with the entirely programmed 360 line of computers [8].
The microcode control unit itself is a small stored program computer. It has a micro PC, 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 [9][10].
Figure 3.1 shows a block diagram of a typical design of a microcoded control unit.

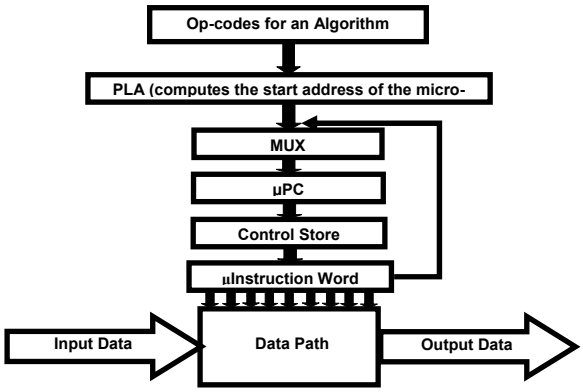

Table 3.1 Block Diagram of Microcoded Control Unit

The µPC 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 µIR, the microinstruction registers. Control store consists of microinstructions which control the datapath directly. The format of microinstructions will be presented in Section 3.2.

3.1. IS MICRO-SEQUENCER A GOOD SOLUTION?

The goal of this study is to perform quick reconfiguration in image processing applications. In section 2.1, it was shown that image processing algorithms have similar computational behavior. Consequently, it can be inferred that the capabilities of 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 gains a significant speed up compared to traditional physical reconfiguration. Sometimes, due to the constraints of new algorithms, the order of utilization of components in datapath may have to be altered ,or further, instantiation of a new computational unit may be ineviTable. In this case, the micro-codes in control unit can be easily modified to satisfy the new requirements. Meantime, a new computational unit is added to the datapath. This achieves higher performance because of utilizing parallelism in computational intensive algorithms. Therefore, MSQR is effective both in terms of providing flexibility and performance.

3.2 MICROINSTRUCTION FORMAT

Microinstructions are an important portion of the controller and have to be defined the way 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 datapath. In the vertical microcode, however, the control of similar components in datapath is compacted in a group of bits in microinstructions. This requires a local decoder to generate all control signals usable for components in datapath. The advantage of the horizontal microinstructions is its lower access time to reach the designated component in datapath. However, as the size of the datapath grows, the number of control bits increases resulting in larger Very Large Instruction Word (VLIW) as microinstructions [11][12][13].

Table 3.2 shows the microinstruction format for our architecture. A few more bits are reserved for newly inserted components in the datapath.

μIR 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 3.2 Horizontal Microcode Format

3.3 ASSEMBLER AND OPCODE STRUCTURE

The instruction set format of our design is very similar to RISC instruction set. It has a fixed 32-bit instruction width. The detail of the instruction format is shown in Figure 3.3. 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.

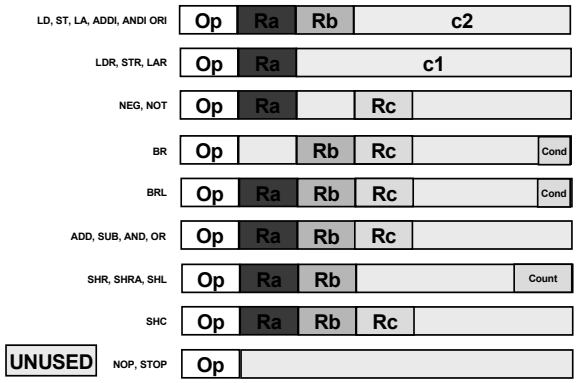

Figure 3.3 Instruction Set Format

In order to support the template generation, we suggest another instruction format. The format has four register index operands. The op-code of this instruction is specified by the template generator. In order to support the quick reconfiguration, our micro-sequencer machine needs to support the new instruction set whenever there is a new hardware implementation. Figure 3.4 shows the fields of the new 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.

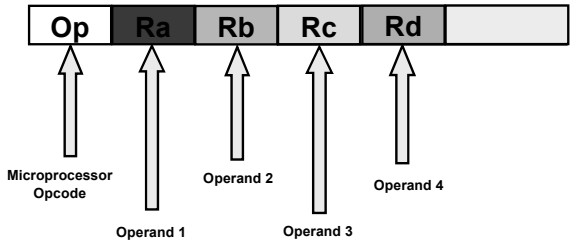

Figure 3.4 New Instruction Set Format

Also we implemented an assembler for our proposed instruction set. The syntax is chosen such that parsing will be minimized. Figure 3.5 shows some common RISC assembly instructions and their translation into our assembler format.

ld 4 1 0 128 = ld r4, [128+r1]

ld 5 1 0 124 = ld r5, [124+r1]

add 3 4 5 0 = add r3, r4, r5

st 3 1 0 128 = st r3, [128+r1]

Figure 3.5 Syntax of the Assembler

4. EXPERIMENTAL RESULTS

As described before, we have implemented a micro-sequencer in VHDL. The code is written structured and generic so that size of data bus/address bus can be easily modified [14].

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 WILDSTAR™ /PCI board. The results are shown in Tables 4.1, 4.2 and 4.3.

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

Table 4.1 Traditional Reconfiguration Time Breakdown

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

Table 4.2 Quick Reconfiguration Time

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

Table 4.3 Reconfiguration Time vs. Quick Reconfiguration Time

As shown in Table 4.3, it is apparent that a significant speed up is gained by using our novel approach of MSQR.
To show the flexibility of our proposed architecture, a background subtraction algorithm which requires 8 bit data bus has been implemented on various micro-sequencers with 8, 16, 32, 64 and 128 bit data buses. The goal is to measure power, area and longest path delay for all above variations. This experiment has been done using Synopsys® Power Compiler. The measured power, area and data arrival times are shown in Figures 4.4, 4.5 and 4.6.

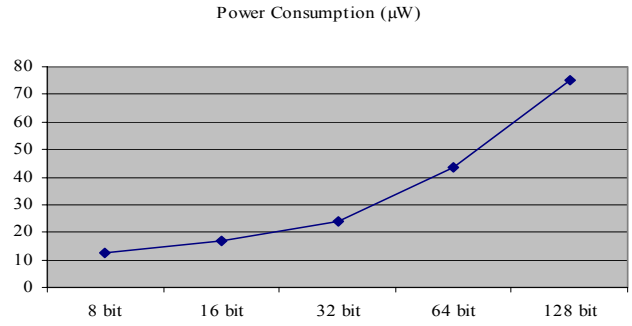

Figure 4.4 Power Consumption

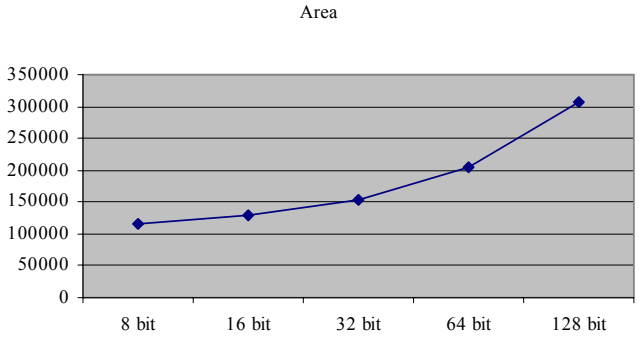

Figure 4.5 Area

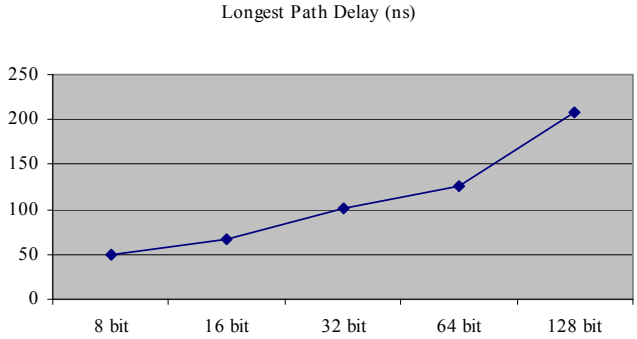

Figure 4.6 Longest Path Delay

5. CONCLUSION

A micro-sequencer based architecture enhances the efficiency of reconfiguration on FPGA. The experimental results prove that the time needed to reconfigure the control unit of a system is far less than the time taken to physically reconfigure the whole system. Besides the reconfiguration time, the micro-sequencer based system provides flexibility for data bus/ address bus design. We demonstrated this point by modifying the size of 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 considerable speed up in reconfiguration process versus the traditional physical reconfiguration.

6. REFERENCES

[1] S. Hauck, "Configuration prefetch for single context reconfigurable coprocessors", ACM/SIGDA International Symposium on FPGAs,65–74, 1998a.

[2] S. Hauck, "The roles of FPGAs in reprogrammable systems", Proc. IEEE 86, 4, 615–638, 1998b.

[3] S. Hauck, W. D. Wilson, "Runlength compression techniques for FPGA configurations", Dept. of ECE Technical Report, Northwestern Univ., 1999

[4] Z. Li, S. Hauck, "Don't care discovery for FPGA configuration compression", ACM/SIGDA International Symposium on FPGAs, 91–98, 1999.

[5] A. Dandalis, V.K. Prasanna, "Configuration compression for FPGA-based embedded systems", ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 173–182, 2001.

[6] D. Deshpande, A. K. Somani., A. Tyagi, "Configuration caching vs data caching for striped FPGAs", ACM/SIGDA International, Symposium on FPGAs, 206–214, 1999.

[7] Z. Li, K. Compton, S. Hauck, "Configuration caching for FPGAs", IEEE Symposium on Field-Programmable Custom Computing Machines, 22–36, 2000.

[8] V.P. Heuring, H.F. Jordan, "Computer Systems Design and Architecture", ISBN: 0-8053-4330-X, Addison Wesley Longman, 1997

[9] C.P.Tsang, S.E.Smith, "Designing a Microcode Synthesis System", IEEE Region 10 Conference on Computer, Sep. 1990, Hong Kong

[10] C. Iseli, E. Sanchez, "A high-level Microprogrammed Processor", IEEE, 1990

[11] T. Phillips, J.B. Michael, Z. Abuhamdeh, "MicroCode Generation for the Control of a Massively Computer'', IEEE, 1998

[12] R. Rauscher, "A new approach in Microcode Optimization", IEEE, 1994

[13] M. Benmimmed, A. Rahmoune, "Automatic Generation of Reprogrammable Microcoded controllers within a high-level synthesis environment", IEE Proc-Comput. Digit. Tech., Vol 145, No. 3, May 1998

[14] B.W. Bomar, "Implementation of Microprogrammed Control in FPGAs", IEEE Transaction on Ind. Elecetronics, Vol. 94, No. 2, April 2002