A High-Performance Unicast Configuration Scheme for an H-Tree Based Reconfigurable Hardware

Andy Widjaja and José G. Delgado-Frias
School of Electrical Engineering and Computer Science
Washington State University
Pullman, WA 99164-2752, USA

***Abstract*—Reconfigurable computing has attracted considerable attention recently because of the potential to deliver the performance of application-specific hardware along with the flexibility of general-purpose computers. Many reconfigurable architectures have been proposed in the last few years, however, few discussions have been conducted on the specifics of the configuration scheme itself. This paper describes an efficient configuration scheme for a reconfigurable DSP hardware that utilizes a hierarchical interconnection network to link clusters of logic blocks, or cells, to map the desired circuits. The scheme makes use of the hardware's existing structure and interconnects to communicate configuration data and keeps data path controls simple. The result is a speedy configuration scheme that requires minimal additional control wires and hardware.**

I. INTRODUCTION

Application specific architectures have long been used to achieve higher performance than general-purpose processors. However, their circuits cannot be altered after fabrication. They require a redesign and chip re-fabrication if any part of the circuit needs modification. This inflexibility and the high design cost make them unattractive for a wide-spread application like digital signal processing. Reconfigurable computing has the potential to achieve most of the performance of tailored architectures while maintaining the flexibility of general purpose processors. This is done through configurable logic blocks that are connected using a set of routing resources that are also programmable. The design of the logic blocks within the reconfigurable hardware varies from system to system. Some of which are described in [1]. For the purpose of this paper, the targeted system is a two-level reconfigurable architecture optimized for DSP described in [2, 3]. The routing between the logic blocks is also of great importance. To configure the routing, typically a passgate structure is used. A programmable bit turns on a routing connection when configured with a true value, allowing signal to flow from one wire to another. It disconnects the wires when configured to a false value. In a large array of logic blocks, the routing and its programmable bits usually contribute to a significant area of the reconfigurable hardware. Furthermore, the amount of routing required does not grow linearly with the amount of logic present. Larger devices require even more routing resources per logic block than smaller ones [6]. DeHon showed in [7] that the most area efficient designs will be those that optimize their use of the routing resources rather than the logic resources. The contribution of this paper is a configuration scheme that maximizes speed and the use of existing routing resources in a reconfigurable architecture. The targeted system uses a hierarchical routing network that is described in detail in [8].

In previous research, we proposed a configuration scheme for the two-level reconfigurable architecture [9]. The scheme uses an H-tree communication network present within the reconfigurable architecture. It provides fast configurations, but at the same time requires a large and complex *broadcast switch* design. The scheme reduces the number of clock cycles for configuring the cells by almost half. However, the number of configuration cycles required for the H-tree global communication switches remained high due to the highly irregular data these switches require. As the reconfiguration of the global switches is expected to occur more frequently than that of the cells, the speed advantage in configuring the cells alone may become less significant in the long run. In addition to that, the high number of cross-points in the broadcast switch that need to be managed greatly increased its design complexity and space requirement.

In this paper, we propose a simpler configuration scheme that requires almost no additional hardware other than those that are already present in the existing architecture [3, 8]. The scheme continues to make use of the H-tree global communication network as the main channel of transmitting configuration data. However, instead of having a differentiated and complex broadcast switch, all the global switches have the same design. Fig. 1 illustrates a typical setup in a global H-tree switch. Each level of the H-tree contains eight busses, four in each direction. The number of lines per bus doubles at each level from 4 bits to 8, 16, 32 and eventually levels off at 64. As such, the number of cross-points in the global switches remains the same throughout the levels of the H-tree.

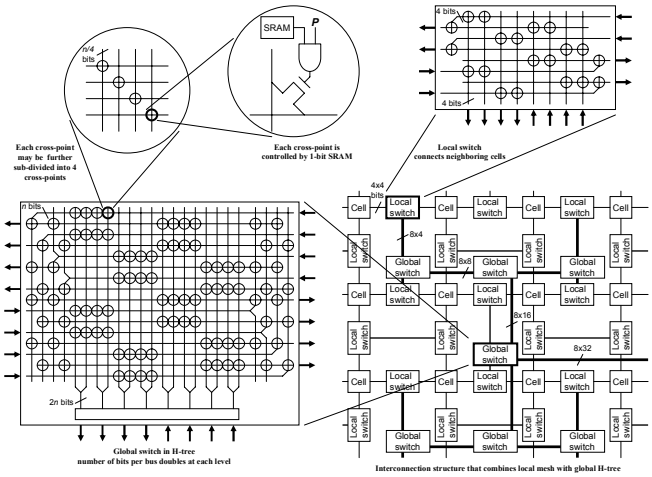


Figure 1. Global and local interconnect switches in cell array.

II. CONFIGURATION SCHEME

The targeted system is a medium grain, two-level architecture described in [2, 3] and its H-tree interconnecting structure is described in [8]. Though targeted at this system, the configuration scheme can, however, be easily modified to support any reconfigurable system with a hierarchical interconnecting structure. As configuration data will only be communicated in a top-down manner, only the downstream bandwidths (i.e. half the total bandwidths) are utilized for configuration purposes. Configuration data will be sent in words of 4x64 (or 256) bits wide each clock cycle. As such, each word is sufficient to provide data for a sub-tree of 32 cells. Naturally, all the global switches above the cells will have to be connected in a *default* connection which ensures that switches above the cells pass the configuration data all the way through to the targeted components. The *default* will be detailed in the next section.

A. Configuring by layers

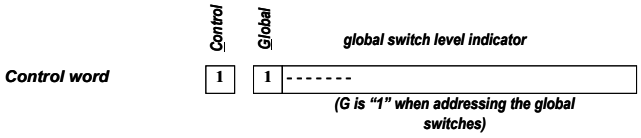
The reconfigurable architecture is made of several components that require configuration, namely (1) reconfigurable elements or cells, (2) local interconnect switches, and (3) global interconnect switches. In addition, each cell can be further divided into two reconfigurable components; the cell processing core and the cell input/output switches. From Fig. 1, it can be observed that half of the local interconnect switches form the lowest layer in the hierarchy. As configuration data is communicated in a top-down manner, components at the lowest layer will naturally have to be configured first. Configuration is executed in layers, moving up along the hierarchy. After its configuration, each layer will be *closed off* to further configuration signals while the system configures components of the next higher layer.

B. Global control signals

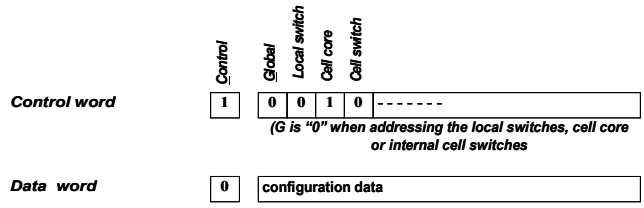
To perform configuration of the hardware components in hierarchical layers, a mechanism is needed to tell the targeted components to receive/store the incoming data, and the other components above them to pass the data through. This also means that each component recognizes if the next data word that is arriving is meant to be stored in its SRAM's or to be rerouted to the next layer of components. To facilitate this, control signals (or a control word) need to be sent concurrently or prior to the configuration data. The purpose of this control word is to signal the targeted layer of components to be ready to store the subsequent data words. To minimize the number of wires in the architecture, we aim to achieve the necessary data control with minimal number of global signals. There are, however, two global signals that this scheme requires. The first, and more obvious, is a 1-bit signal known as *Control* (from here onwards designated as *C*). The signal *C* indicates whether the busses are carrying *control words* or *data words*. A control word is carried in the data bus when *C* is exerted with a true value otherwise a data word (or configuration data) is carried in the bus. The second global signal is known as *Programming* mode signal (designated as *P*). *P* is responsible for the *default* global switch connections mentioned earlier in this section. When *P* is exerted, the switches will revert to their *default* crossbar settings. As the routing system is highly pipelined, the global signals will also be channeled downstream in a pipelined manner along with the corresponding data bus.

C. Configuration word

All configuration data will always be preceded by a control word. The *control word* comes in two flavors. The first is used to communicate with the global switches. This is done by exerting the *Global* bit to a true value:



The second is used to communicate with the local switches and cell components. This is done by exerting the *Global* bit to a false value. When communicating with the local components, the three subsequent bits after the *Global* bit designate the local switch, cell processing core and cell internal I/O switches respectively:



Configuration data is sent in data words following the control word. If the configuration bits required for a particular component exceed the bus bandwidth, multiple data words may be sent before the next control word. The configuration sequence for a particular component (or a layer of components) is terminated by a control word that closes the data gates leading to that component. A terminating control word can also carry information for the next layer of components to be opened for configuration.

III. PROGRAMMABLE ARCHITECTURE

A. The programmable bit

We begin by looking for an appropriate SRAM design to store the programmable routing bits. We chose the SRAM as our main storage device mainly for its low power consumption. Fig. 2 shows the two designs that we narrowed down to. Both designs use a total of 6 transistors. Fig. 2(a) depicts a more typical SRAM with a transmission gate feedback to assist in getting both strong '1' and strong '0' at the input to the double inverters. Fig. 2(b) illustrates a modification to the SRAM where the second inverter's rail voltages are cut off during a write session. As a result, we found (b) to have 10% lower power consumption than (a) during our simulations. Investigation into lower power SRAM designs for the purpose of our reconfigurable system is currently under way, and will be part of a separate paper.

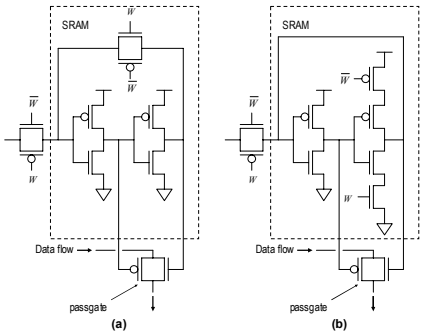


Figure 2. RAM designs.

B. *Programming the switch.*

One of the methods to reduce configuration time is through compression of the configuration data. Lower number of configuration bits means less configuration cycles. The number of programmable bits per switch depends on the switch function itself and its size. The global interconnect switches, for example, have 96 cross-points each, which translate to 96 programmable bits. The local switches have 20 bits each, and the cell internal I/O switches have 64 bits each. The arrangements of the cross-points for the global and local switches are depicted in Fig. 1. The cell internal I/O switches are made up of the full 8x8 crossbars.

Fig. 3 illustrates the use of row and column decoders on an 8x8 programmable bits of the crossbar for the cell internal I/O switch. Only three bits per decoder are required for eight rows or columns. So each clock cycle, a 6-bit data word can be used to configure an output row. However, at the cell layer of the H-tree network, each cell receives an 8-bit input data bus. Therefore, an extra bit is used for the row decoder to select all rows at once for writing. The other extra bit is used for the column decoder to write a '0' (false value) to all the SRAM's in a row. This is useful when we need to deactivate all the connections in the switch, which now can be done in merely one clock cycle with the extra bits. Turning off all connections in a switch is useful for setting up the *default* connections as shown in Fig. 4.

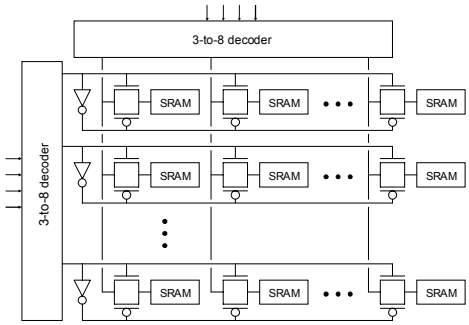


Figure 3. Decoding the configuration data.

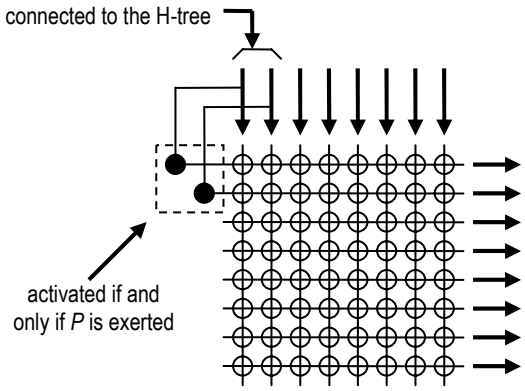


Figure 4. An 8x8 crossbar with added crosspoints for *default* connections.

C. *Partial reconfiguration*

In some cases, configurations do not occupy the entire reconfigurable array. Yet in other instances, only a part of the configuration requires modifications. In these situations, a partial reconfiguration of the array will suffice, and will cost less downtime as opposed to a full reconfiguration. The hierarchical nature of our system allow for configurations in clusters of cells or switches. When necessary, the tree structure also allows for configuring interconnection switches that are at the higher levels without disturbing the lower level switches and logic blocks. We found this useful in large applications like the Fast Fourier Transform (FFT) which is often executed in a DSP operation.

IV. PERFORMANCE ESTIMATION

There is not a straightforward method of comparison between the performance of different reconfigurable architectures. Despite some works that have been done, like Dehon's in [10] and the *remanence* in [11], a concrete metric for comparison remains elusive. We thus do comparison based on two simple, and yet much sought after factors, in configuration/reconfiguration, i.e. speed and design complexity.

A. *Number of configuration bits*

To analyze the number of clock cycles required for a full configuration, we first tally up the number of configuration bits in a 32x32 array of reconfigurable cells. Table I shows how the cell cores make up a majority of the configuration bits required in a full configuration. Furthermore, the cores are found in one of the lowest levels in the H-tree network. Reconfiguring the cores would mean reconfiguration of all the switches above them. As such, it motivates us to perform minimal reconfiguration of the cell cores after the initial full configuration. Partial reconfiguration will play an important role in programming portions of the array while leaving reusable circuits unchanged for the subsequent operations.

TABLE I. NUMBER OF CONFIGURATION BITS IN A 32X32 CELL ARRAY

Component	Quantity	Bits per component	Configuration bits
Cell cores	1,024	512	524,288
Internal switches	1,024	128	131,072
Local switches	1,984	20	39,680
Global switches	511	96	49,056
		Total	744,096

B. *Configuration time*

In evaluating the length of configuration time, we divide the analysis into two scenarios. The first is the case where the configuration data is not cached within the system. The second assumes a cached memory embedded near the reconfigurable array. Let us first evaluate the first case. Assuming an 8-bit connection to outside the array, the system loads 8-bit configuration words into the global interconnection lines. One configuration word is transmitted every clock cycle. In a full configuration, it takes 64 cycles to fill the 64x8 bit memory in the processing cell core. The two internal cell I/O switches each takes 8 cycles to program. An additional 3 cycles are needed for the control words. In total, it takes 64 + 8 + 8 +3 which equals to 83 cycles to program a cell. The estimation of total configuration cycles for a 32x32 array is summarized in Table II.

The most basic Xilinx Virtex-II device contains 338,976 bits of configuration that can be programmed at 50 MHz in their express mode (i.e. 8-bit loading per clock cycle) [12]. A high-level version has 26,194,208 configuration bits and can be programmed at 200 MHz. Table II shows how the configuration cycles of the proposed system compares with those of the Xilinx Virtex-II series and the more matured but popular XC4000 series. The Xilinx devices are

programmable at 50-200 MHz. In our simulations, the proposed scheme was able to achieve a configuration speed of 1 GHz with a modest 0.18-µm technology. The proposed scheme compares favorably against the systems currently available in the market.

TABLE II. COMPARISON TO OTHER RECONFIGURABLE SYSTEMS

Device		Config. Bits	Config. cycles	Est. clock	Config. time
Unicast		744,096	99,571	1GHz	99.6µs
Xilinx Virtex-II	XC2V40	338,976	42,372	50MHz	874.4µs
	XC2V2000	6,812,960	851,620	100MHz	8,516.2µs
	XC2V8000	26,194,208	3,274,276	200MHz	16,373.4µs
Xilinx XC4000	XC4013XLA	393,632	49,204	50MHz	984.1µs
	XC4062XLA	1,433,864	179,233	50MHz	3,584.7µs

In the second scenario, we assume the presence of cached memory within the reconfigurable system. The proposed scheme will be able to make use of the entire 256-bit data bus going into the global interconnection network. As such we were able to load configuration data to multiple cells (or other components) that are on the same level in the H-tree at the same time. Specifically, the 256-bit bus is able to feed data into 32 processing cores (8-bit per core) every clock cycle. The proposed unicast scheme is compared to the previous broadcast based scheme [9]. Table III summarizes the configuration cycles and the number of additionally memory required to support the respective configuration schemes. In the worst case, the unicast scheme reduces the number of configuration cycles to 3,264 for a full configuration. In a similar situation, our previous scheme, using broadcast switches, requires only 912 clock cycles. However, it was achieved at the cost of more memory and complex data path controls within the broadcast switches.

TABLE III. 256-BIT LOADING FROM INTERNAL CACHED MEMORY

	Scheme	Additional mem. for configuration	Config. cycles
Full configuration	Unicast	12,152 bits	3,264
	Broadcast	44,920 bits	912
Partial configuration (only the global switches)	Unicast	n/a	352
	Broadcast	n/a	468

Additionally, when only a partial configuration is required, the unicast scheme performs better than the broadcast scheme. We assume a partial configuration where only the global switches are reconfigured while the rest of the array remains unchanged. The unicast scheme reconfigures the global interconnects in 352 clock cycles, whereas 468 cycles are required by the broadcast scheme. The difference becomes larger if even fewer of the global switches require reconfiguration. This is understandable because the previous scheme imposes a fixed overhead in first configuring the *broadcast switches* before using them to configure the array components. In applications where runtime reconfiguration does not occur frequently, the broadcast scheme offers shorter configuration time. However, in applications where mapped circuits are often reused, the unicast scheme quickly pays off its initial full configuration cost with regular partial reconfigurations, and reduces the overall reconfiguration times.

V. CONCLUDING REMARKS

In this paper, an efficient H-tree based configuration scheme for reconfigurable hardware was presented. The scheme utilizes the existing global interconnection network for communication of configuration data in a top-down manner. Configuration was performed in layers, starting at the lowest level of the H-tree and moving up from there. Data paths were managed by differentiating *control words* from *data words* as they make use of the same communication wires. Transistor level simulations indicate that configuration can be performed at a clock frequency of 1 GHz using a modest 0.18-µm technology.

While we explored low power SRAM briefly in this paper, we believe there are other components that can yield more power savings. Switches, decoders, and wiring routes will be scrutinized for lower power requirements in future development. Another direction for this research is to merge the *broadcast* and *unicast* schemes to form a hybrid configuration scheme. The broadcast scheme will be used to configure the cell core, cell internal switches and local mesh switches as it achieves the highest performance in these components. The unicast scheme will be employed solely for the global interconnect switches. This approach will greatly increase the total configuration speed as we capitalize on the strengths of both schemes. However, it is also expected to significantly increase the complexity and size of the broadcast switches as they must be designed to alternate between the two schemes. The tradeoffs of such a hybrid scheme will be explored.

REFERENCES

[1] K. Compton and S. Hauck, "Reconfigurable Computing: a survey of systems and software," ACM Computing Surveys, vol. 34, no. 2, Jun 2002, pp. 171-210.

[2] J. G Delgado-Frias, M. Myjak, F. Anderson, and D. Blum, "A medium-grain reconfigurable cell array for DSP application," in Proc. 3rd iasted International Conference on Circuits, Signals, and Systems, Cancun, Mexico, May 2003, pp. 231-236.

[3] M. J. Myjak and J. G. Delgado-Frias, "A two-level reconfigurable architecture for digital signal processing," in Proc. 2003 International Conference on VLSI, Las Vegas, NV, Jun 2003, pp. 21-27.

[4] C. Ebeling, D. Cronquist, and P. Franklin, "RaPiD – Reconfigurable Pipelined Datapath. A Configurable Computing Architecture for Computer-Intensive Applications," Dept. of Computer Science and Engineering, University of Washington, Nov 1996.

[5] P. Chow, S. O. Seo, J. Rose, K. Chung, G. Páez-Monzón, I. Rahardja, "The design of an SRAM-based Field-Programmable Gate Array – Part I: Architecture," IEEE Trans. VLSI Syst. 7, 2, 1999, pp.191-197.

[6] S. Trimberger, K. Duong, and B. Conn, "Architecture issues and solutions for a high-capacity FPGA," ACM/SIGDA International Symposium on FPGAs, 1997, pp. 3-9.

[7] A. Dehon, "Balancing interconnect and computation in a reconfigurable computing array (or, why you don't really want 100% LUT utilization)," ACM/SIGDA International Symposium on FPGAs, 1999, pp. 69-78.

[8] M. J. Myjak, F. L. Anderson, and J. G. Delgado-Frias, "H-Tree Interconnection Structure for Reconfigurable DSP Hardware," in Proc. 2004 International Conference VLSI, Las Vegas, NV, Jun 2004.

[9] A. Widjaja and J. Delgado-Frias, "An H-tree based configuration scheme for reconfigurable DSP hardware," in Proc. 2004 International Conference on VLSI, Las Vegas, NV, Jun 2004.

[10] A. Dehon, "Comparing Computing Machines," Configurable Computing: Technology and Applications, Proc. SPIE 3526, Nov 1998, pp. 2-3.

[11] P. Benoit, G. Sassatelli, L. Torres, D. Demigny, M. Robert and G. Cambon, "Metrics for reconfigurable architectures characterization: Remanence and Scalability," in Proc. International Parallel and Distributed Processing Symposium (IPDPS'03), 2003.

[12] Xilinx Product Specifications, "FPGA Devices Families," http://www.xilinx.com/xlnx/xweb/xil_publications_index.jsp