The Design and Implementation of a Context Switching FPGA

Sanders, A Lockheed Martin Company
PTP2-B007
65 River Road
Hudson, NH 03051

Stephen M. Scalera
sscalera@sanders.com
Phone: (603) 885-0679 FAX: (603) 885-7623

Jóse R. Vázquez
jvazquez@ede.sanders.lmco.com
Phone: (603) 885-0746 FAX: (603) 885-7623

Abstract

Dynamic reconfiguration of field programmable gate arrays (FPGAs) has recently emerged as the next step in reconfigurable computing. Sanders, A Lockheed Martin Company, is developing the enabling technology to exploit dynamic reconfiguration. The device being developed is capable of storing four configurations on-chip and switching between them on a clock cycle basis. Configurations can be loaded while other contexts are active. A powerful cross-context data sharing mechanism has been implemented. The current status of this work and future work are described.

1 Introduction

History has seen the methodologies of computing evolve from *fixed* hardware and *fixed* software (ENIAC), to *fixed* hardware and *reconfigurable* software (microprocessors), to *reconfigurable* hardware and *reconfigurable* software (FPGAs). FPGAs have traditionally been utilized in applications that demand the performance of application specific integrated circuits (ASICs) while maintaining the flexibility and rapid design cycle afforded by the use of digital signal processors (DSPs). Although FPGAs are not ideally suited for either requirement, they do offer an excellent compromise. In the recent past, many research efforts have examined the possibility of performance enhancement due to run-time reconfiguration. However, the best of today's commercially available technology requires milliseconds to reconfigure. This reconfiguration time, although acceptable for some applications, such as the *SPEAKeasy* reconfigurable "softradio" developed by Sanders, is an unacceptable delay for most real-time systems. Although partial reconfiguration can reduce the required reconfiguration time, this is believed to be an alternative approach to dynamic reconfiguration. Being able to *completely* reconfigure an FPGA at a rate that far exceeds the necessary persistence of a hardware function, while being able to share data between configuration instantiations is believed to be tomorrow's reconfigurable computing computational model. This model of computation shall be referred to as *context switching reconfigurable computing* and is a natural extension of today's methodology. Arguably, context switching is not unlike the very first mode of computation. In essence, clock-cycle dynamic reconfiguration can be viewed as *fixed* hardware and *fixed* software since the available hardware can be thought of as being virtually infinite – *virtual hardware*.

The context switching reconfigurable computing (CSRC) technology being developed by Sanders extends commercially available field programmable gate array (FPGA) devices to include high speed changes between a number of programmed functions without the need for additional FPGAs. Each configuration, referred to as a *context*, in a CSRC FPGA has the functionality similar to that of many commercially available FPGAs. The context switching can occur at significantly higher speeds than the rate at which current FPGA technology can reconfigure. In addition, unlike commercial FPGAs, where reprogramming destroys any resident data, the CSRC FPGA affords the capability of data sharing between contexts.

The concept of virtual hardware is an obvious benefit of dynamic reconfiguration. If configurations can be swapped in and out of an FPGA upon demand at a real-time system rate, only the necessary hardware need be instantiated at any given time. In this manner, a virtually infinite algorithm cache or an infinite coprocessor can be conceived. In other words, a high level system scheduler can instantiate hardware as needed. In this manner, a reduction in size, weight, and power can be achieved. Additionally, given the CSRC FPGA, if the processing requirements specify a sequential application of

algorithms, the context layers can be set up to share data such that the output of one algorithm is immediately available as the input to the next algorithm upon a context switch. This is not possible with contemporary FPGAs.

A natural extension of the algorithm cache mode of computation is the concept of mission phase reprogrammability. As seen in Figure 1.1, an entire mission can be mapped to a CSRC device. In this case, different contexts can house different algorithmic phases of a mission without requiring that an algorithm be confined to a single context, depicted as layers in Figure 1.1.

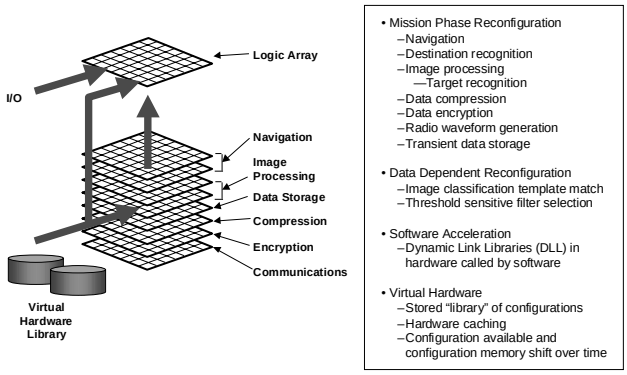

Figure 1.1: Reconfiguration Benefits

Although Figure 1.1 identifies the obvious modes of computation for gaining a performance enhancement, it is believed that the true potential of context switching requires a paradigm shift in algorithm implementation. The capabilities of the CSRC architecture, which extend dynamic reconfiguration to context switching, have the potential to provide improved implementations of signal processing algorithms over those currently available through commercial FPGAs. The inherent ability of CSRC to quickly perform different tasks and share results among different configurations allows one to approach algorithms from a different perspective, enabling mathematical implementations previously inconceivable without context switching.

Although the past few years have seen much interest in context switching reconfigurable computing, it is believed that this paper describes the first design and implementation of such a device. Up until now, all of the substantiated work on this model of computation has been theoretical. The emphasis of this paper is reveal the architecture of the context switchable FPGA being developed by Sanders. It is hoped that this paper will spark new ideas and facilitate algorithmic research that is targeted towards a specific and real architecture. With this achieved, as the world's first context switchable silicon becomes available, members of the adaptive computing systems (ACS) community will be capable of taking full advantage of this new technology. IC development status is advanced in section 3.

2 Architecture

Experience has shown that FPGAs afford the greatest performance benefit when they are used to implement algorithms with deep pipelines. However, pure dataflow algorithms are rare. In fact, generating pipeline control signals, implementing state machines, and interfacing with external RAM or other integrated circuits, are critical, although not typically areas of performance enhancement, to an FPGA's successful system integration. With this in mind, the CSRC device was designed to be a 4 bit DSP dataflow engine that is simultaneously capable of efficiently implementing glue logic. However, since FPGA performance enhancements are oftentimes achieved by implementing the minimum required bitwidth, the CSRC device was developed to allow users to implement scalable pipelines such that the wordwidth can be of any size.

2.1 Data Pipes

The CSRC device is arranged into 16-bit wide data pipes. Each pipe is formed by a plurality of context switching logic arrays (CSLAs) as seen in Figure 2.1. A single CSLA is capable of processing two 16-bit words and outputting a 16-bit result. The result of a CSLA is available as an input to the two adjacent CSLAs in the pipe. Hence, a pipe can naturally be used as a data path. Information can easily flow from one end of the pipe to the other. It is important to point out that in this device data can non-preferentially flow in both directions. This feature has great utility when sharing data among different contexts. For example, one context could process data from left to right, storing it's final result in the right-most set of registers. Note that is quite possible that the final result of a single context is actually an intermediate result of the entire algorithm. Given this situation, an incoming context can pick up where its predecessor context left off by acquiring the intermediate data deposited on the rightmost portion of the pipeline and processing it in a pipeline that flows from right to left. From this simple example, it can be seen that a data path that does not favor data flow in either direction is more efficient for context switching hardware because it alleviates the need to reroute data from its physical origin in one context to its physical input in the subsequent context.

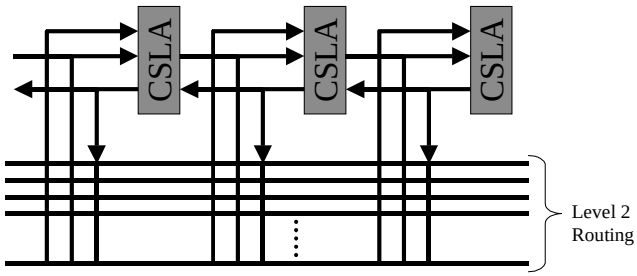

Figure 2.1: 16 Bit Data Pipe Comprised of CSLAs

Level 2 routing can be found alongside the pipe and consists of 16-bit buses. See Figure 2.1. These busses are not segmented and run the entire width of the CSRC device. This type of bussing scheme implies that a signal driven onto level 2 routing is available to any CSLA in the pipe. Additionally, this approach affords the possibility of faster and less complicated programming tools than segmented approaches because the timing is more deterministic. Each CSLA has two 16-bit inputs, each of which is capable of tapping into any of the Level 2 routing busses. Similarly, the CSLA's 16-bit output can drive any of the Level 2 routing busses. Note that Level 2 routing can be utilized as a bus architecture, can be broken down and utilized by individual bits, or can be employed as any combination of these.

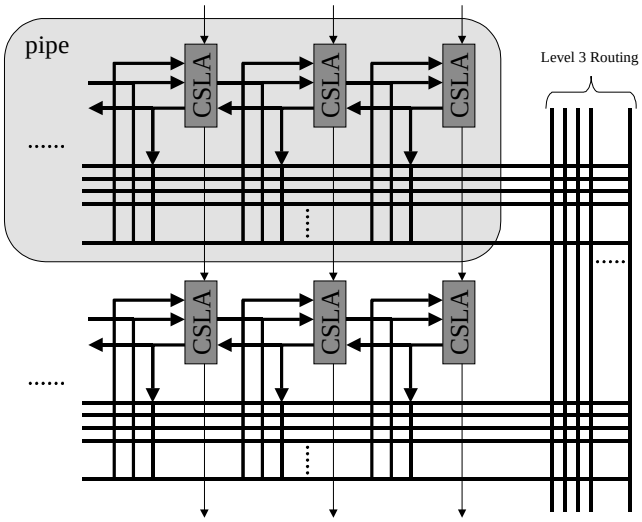

Figure 2.2: Level 3 Routing Bridges Pipes

The CSRC device is formed by stacking up pipes one on top of the other. Corresponding CSLAs on adjacent pipes have dedicated wiring that allows them to pass along their carry bit. This feature allows two adjacent pipes to be bundled together and be used as a single 32-bit wide data path. In actuality, physical 16-bit pipes can be broken down into smaller logical pipes. Although hardware is optimized to break pipes into nibbles, pipes can be n-bits wide.

As seen in Figure 2.2, information driven onto a given pipe's Level 2 routing can be connected to Level 3 routing which in turn makes the data available to any Level 2 routing on the chip. Similar to the Level 2 routing structure, the Level 3 routing is not segmented and spans the device. Note that conceptually the Level 2 and Level 3 routing are perpendicular to each other.

I/O pins on the device are connected to Level 2 and Level 3 routing. All pins physically located on the top and bottom edges of the device connect to Level 3 routing. Pins on the left and right edges can connect to either Level 2 routing or directly into the dedicated routing that normally connects adjacent CSLAs.

2.2 Context Switching Logic Array

A single CSLA is primarily composed of 16 context switching logic cells (CSLCs) and Level 1 routing to interconnect them. Figures 2.3 and 2.4 depict a CSLA and the CSLA as it attaches to the Level 2 routing, respectively. The CSLCs are numbered 0 through 15 and their carry-in and carry-out chains are hardwired appropriately so they can function as a single cohesive unit. Level 1 routing consists of three 16-bit busses. Two of these 16-bit busses are inputs from the Level 2 routing. The third 16-bit bus is hardwired to the outputs of the CSLCs. Level 1 routing was designed with two modes of operation in mind.

2.3 Routing Modes of Operation

As previously mentioned, it is believed that the most beneficial FPGA is capable of exploiting its inherent DSP strengths while simultaneously being capable of implementing the often required glue logic. Hence, the CSRC FPGA has been designed with two modes of operation in mind: (1) Deep pipeline mathematical operations that can be of arbitrary bitwidth & (2) Random logic implementations that encompass control, state machines, and interfacing with external RAM or other integrated circuits. As a direct result, the CSRC FPGA exhibits two types, or modes, of routing.

2.3.1 Bus Routing

The first operational mode of routing is bus routing. The design goal was to provide users with the ability to route entire 16-bit words in and out of CSLAs while maintaining bitwidth order (i.e. the most significant bit (MSB) in the MSB position and the least significant bit (LSB) in the LSB position). Given that the four

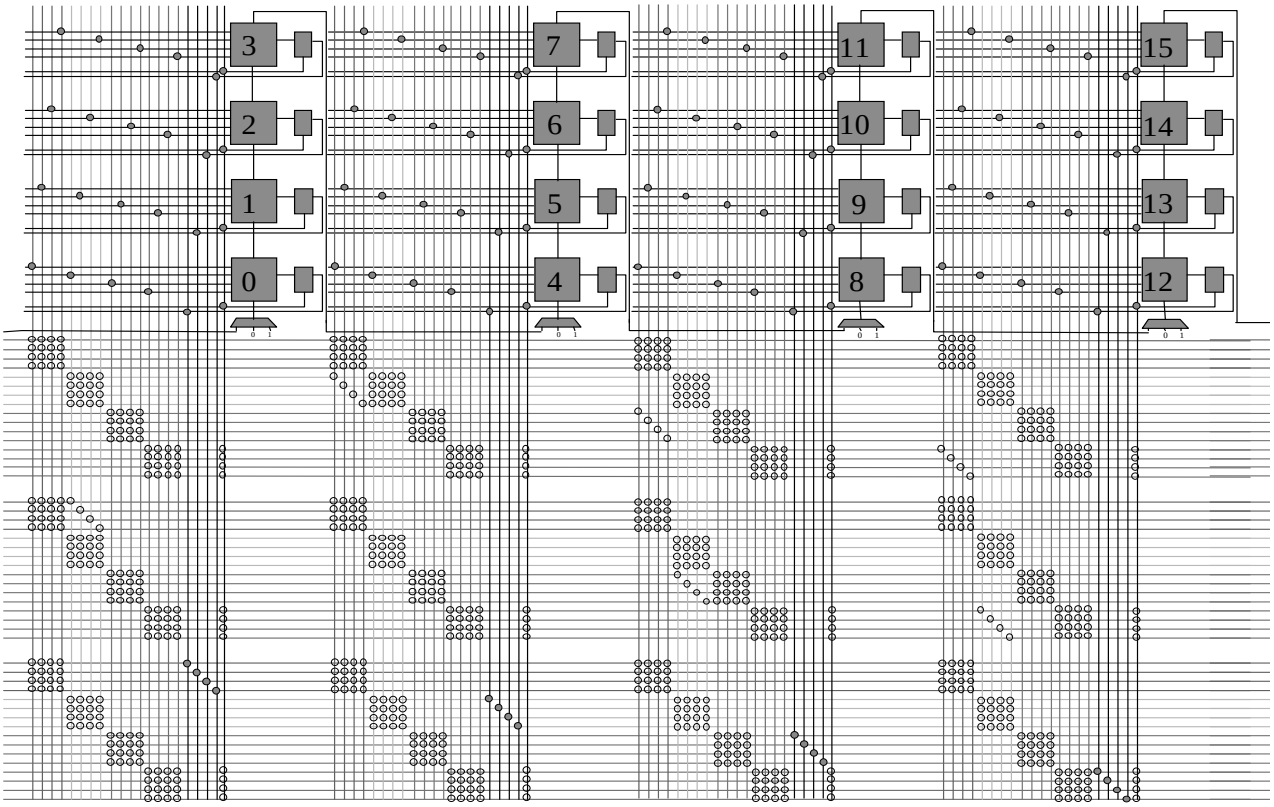

Figure 2.3: Context Switching Logic Array & Level 1 Routing

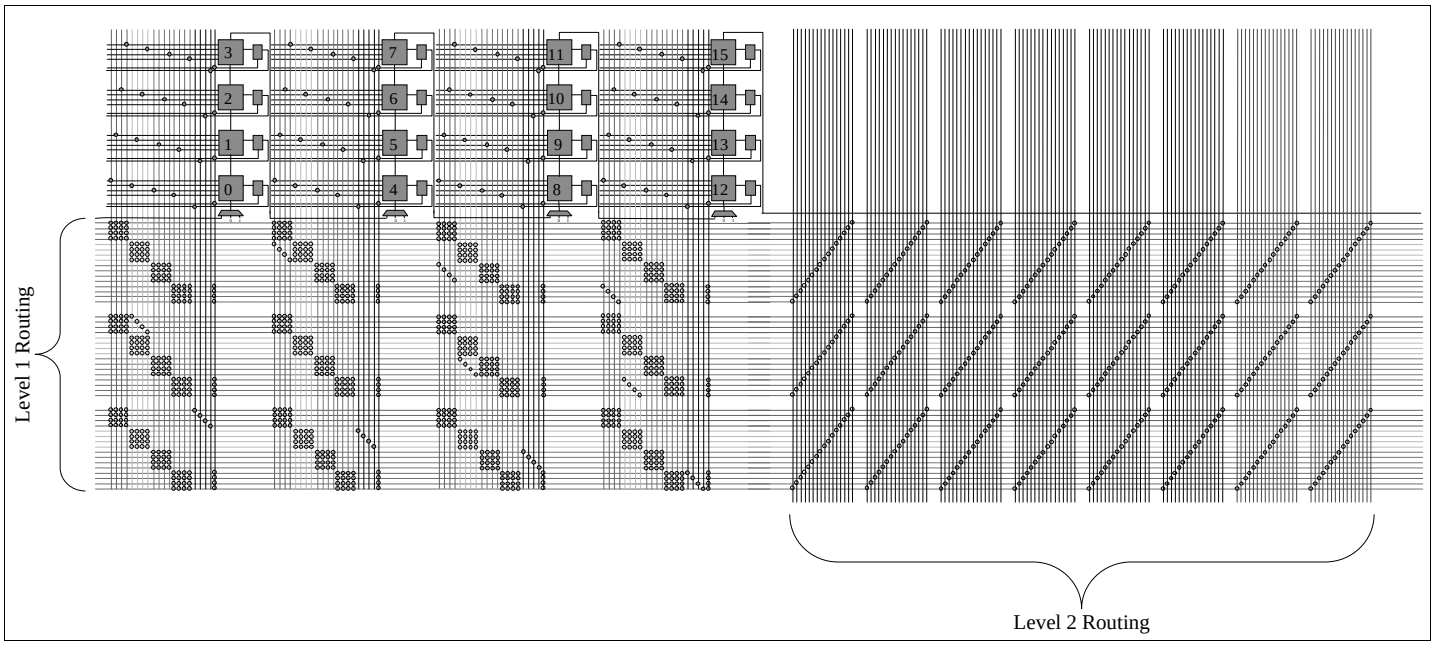

Figure 2.4: Context Switching Logic Array with Level 1 & Level 2 Routing

data inputs to the CSLC are labeled as A, B, C, and D, enough programmable connections are contained in the Level 1 switching matrices to ensure that one of the input buses can be routed into the A inputs of all of the 16 CSLCs. The least significant bit of the bus feeds the A-input of the least significant CSLC and so on. In essence, this bus can be considered the A-input (16-bits wide) for the entire CSLA under this bus routing mode of operation. Note that the second 16-bit bus can be used to feed the B inputs of the CSLCs within a CSLA in a similar fashion. The final bus connection is hardwired to the 16-bit output of the CSLA and attaches to the Level 2 routing. Note that this output is also a direct connect between neighboring CSLAs. As previously described, this non-directional direct connect allows for fast routing between CSLAs within a pipe by alleviating the need for Level 2 routing if the output of a pipe stage is feeding a neighboring CSLA.

2.3.2 Bitwise Routing

The second mode of operation is bitwise routing. No matter how data processing intensive a design might be there is almost always a need for control logic whether it is simple glue logic or more complex state machines. For this reason the bitwise routing mode of operation is necessary. The basic premise is that the output of any given CSLC within a CSLA should have at least one possible path to connect to at least one input of all other CSLCs within the same CSLA. A simple pattern of programmable connections was developed to enable this feature. All the A-inputs of all the CSLCs in a CSLA can tap into the four least significant bits of all three Level 1 routing busses (this includes the output bus to provide a means of local feedback without having to waste Level 2 routing resources). Similarly the B-inputs and the C-inputs tap into the next 4 bit bundles within each level 1 routing bus, and finally the D-inputs tap into the four most significant bits on every bus. As a result, the four least significant CSLCs, which drive the corresponding four-least significant bits of the output bus, are capable of driving any A-input on any CSLC within the same CSLA. For this reason these four CSLCs are known as "A-drivers" under the bitwise routing mode of operation. Similarly, B-drivers refers to CSLCs 4 through 7, C-drivers to CSLCs 8 through 11, and D-drivers to CSLCs 12 through 15. Furthermore, since connections between Level 1 and Level 2 and connections between Level 2 and Level 3 maintain proper bit order (LSBs to LSBs and MSBs to MSBs) any A-driver can drive the A-input of any CSLC anywhere in the chip. For these same reasons, the same functionality applies to the B, C, and D-drivers.

In addition to the four main inputs (A, B, C, & D), each CSLC has a clock enable / tri-state control line. Both of these control lines tap into the four most significant bits of the three Level 1 buses, hence, they are controlled by D-drivers. As seen in Figure 2.5, the clock enable / tri-state control line is a single control line to the CSLC. For this reason, the user can choose to use this control line to control either the clock enable or the tri-state buffer.

2.4 CSLC

The CSLC is the heart of computation for the CSRC device. As seen in Figure 2.5, the CSLC is composed of carry logic, a four input lookup table (CSLUT), a context switching flip-flop (CSFF) and a tri-state buffer. The carry logic unit is capable of generating carry bits for either additions or subtractions. The carry logic chain is connected by dedicated connections. The chain can be connected, disconnected, or fed a logic zero or logic one every four bits. In this manner, the bus routing mode can be utilized to generate a pipeline granularity of four bits. However, in reality, the buswidths can be of an arbitrary bitwidth, *n*. Note that bitwidths with a modulo 4 = *m*, where *m* is greater than zero, will disallow *m* CSLCs from supporting a mathematical pipe that requires the starting of a carry chain.

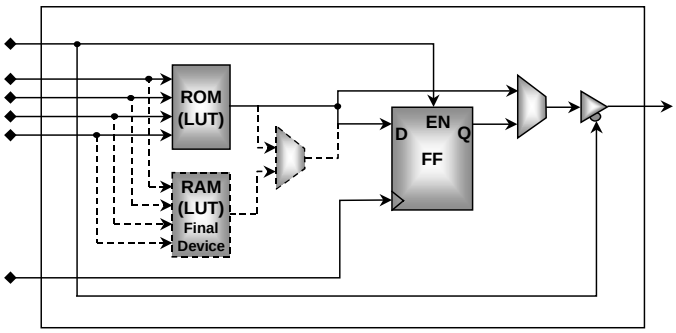

Figure 2.5: Context Switching Logic Cell Architecture

The outputs of the carry logic feed the CSLUT which consists of 16 context switching configuration bits (CSBits) that are multiplexed together. The 4 inputs serve as the select lines therefore implementing a programmable function. Note that the contents of each of the CSLUTs are unique in each context and specified in the configuration bitstream.

The CSBits implement context switching itself. Each CSBit holds a single programming bit for every context. However, only the active context's value drives whatever logic the CSBit is controlling. A detailed description of the context switching functionality afforded by these CSBits is described in section 2.6.

Unlike some commercial FPGAs, the lookup table can not serve as a memory element because the CSLUT is composed of CSBits, not SRAM. Instead a separate context switching RAM (CSRam) provides memory

storage facilities. The CSRam, which is only available in the final CSRC device, implements the *global sharing scheme*. This data sharing scheme is similar to traditional blackboard data sharing. Any data written to a CSRam memory is available to all the CSRam elements that are physically collocated among different contexts. Whatever data value is last written into the active CSRam before deactivation of the current context will be seen by all other collocated CSRams upon the activation of their respective contexts. In fact, one can envision writing to a CSRam in one context and having its contents be used as a LUT in another context. Additionally, it is the CSRam that will allow for large amounts of data passing between contexts to facilitate modes of computation such as moving the algorithm through the data. This mode of computation is advantageous due to the fact that the on/off chip accesses are minimized by loading the data on chip and keeping it on chip until the entire algorithm has been run on the data.

Both the CSRam and the CSLUT will coexist in the final CSRC device and their outputs will be multiplexed together. The select line of this MUX is yet another control line to the CSLC and it is connected to Level 1 routing in the same fashion as the clock enable / tri-state control (driven by D-drivers). The output of this MUX can then be registered or passed directly out of the CSLC as seen in Figure 2.5. In either case, the user has the ability to tri-state the output of the CSLC. Note that if the data is to be registered, it will be done in the context switching flip-flop (CSFF). During regular operation within a single context, the CSFF appears to the users as a normal D-flip-flop (DFF). The DFF connects to the global clock and it is controlled by the clock enable input to the CSLC.

2.5 CSIO

The context switching input/output cell is used to facilitate on/off chip data accesses. As can be seen in Figure 2.6, the CSIO cell is bi-directional, can provide latched or direct outputs, and has a programmable pull-up resistor on the output. In addition, the CSIO cell can tri-state its output. Since on/off chip access time is oftentimes a limiting factor of FPGAs, a programmable drive strength capability has been included to insure maximum performance. The output drive strength can be selected to be 2mA, 6mA, or 12mA. Finally, the flip-flop in the CSIO cell utilizes a different sharing scheme than the flip-flop in the CSLC. Note that since it is believed that sharing data between contexts within a CSIO cell is unlikely to be a key feature, the global sharing scheme is implemented for the CSIO cell DFFs rather than the more complex sharing scheme that is implemented in the CSLC's DFF.

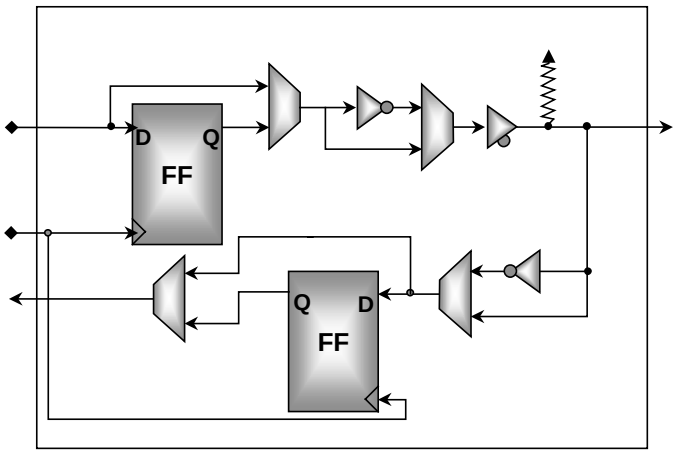

Figure 2.6: Context Switching Input/Output Cell Architecture

2.6 Data Sharing / Context Switching

Research indicates that the major benefits of context switching are afforded by sharing data between contexts and being able to switch between contexts very rapidly. For this reason, a great emphasis has been placed on the development of a device that meets both of these needs. The two sharing schemes that have been designed and implemented are Global Sharing and Private/Public Addressable Sharing (P/PASS). The global sharing scheme is used in the CSIO DFF and in the CSRam while P/PASS is used in the CSLCs by the CSFF. Global Sharing, as previously described, is simply a common memory element between all contexts. Hence, all contexts view these same memory elements and when any context writes to the memory element, the change is seen by all contexts upon their respective activation.

The data sharing scheme used by the CSFF, P/PASS, truly exposes the novelty of the CSFF and is depicted in Figure 2.7. With this type of sharing, each CSFF within each context supported in hardware has a corresponding register. These registers are known as *private* registers since they belong to a particular context and can only be accessed by a specific DFF within the context. Additionally, there is a single active register per CSFF. The active register is what the user actually utilizes during uninterrupted context execution. Upon switching contexts, the outgoing (active) context saves its intermediate values to its private registers. This feature enables many of the capabilities that the NSA would need to develop secure kernels by isolating intermediate data. Additionally, a context can *choose* to write its values to a *public* register (on a Logic Cell by Logic Cell basis) which can be addressed by any and all of the contexts. In this manner, the sharing of data between DFFs within contexts is enabled. The number of public registers available in a P/PASS implementation is independent of the number of contexts supported directly by hardware.

Hence, public registers must be addressed when used. Note that the CSRC device affords two public registers. Upon activation, a context can choose to restore its previous state by reading from the private register or it can opt to load a state from either public register (on a Logic Cell by Logic Cell basis).

P/PASS provides a means to keep secure data isolated while at the same time allowing data to be shared (if so desired) using public registers. This architecture scales to implementations with more contexts than hardware supports, allows sharing data between contexts that do not necessarily follow one another in time, and provides a clean and solid foundation to add features such as interrupt handling and hardware recursion.

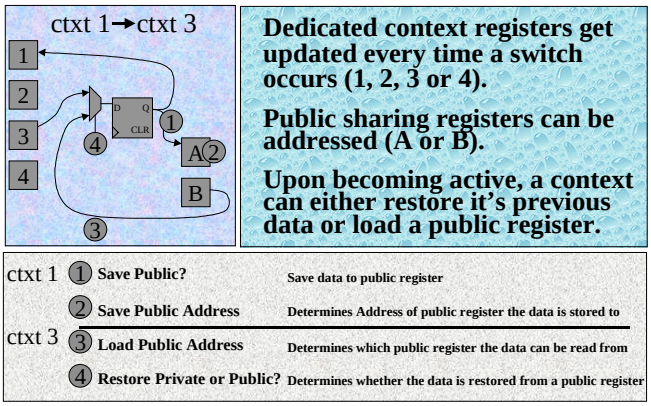

Figure 2.7: Private/Public Addressable Sharing Scheme

Since some modes of computation, such as the virtual coprocessor, require rapid reconfiguration, the CSRC device was designed to be capable of switching contexts on a single clock cycle. A key point to be made is that this single cycle context switching not only includes completely reconfiguring the CSRC device but completely executing all of the data sharing schemes. In fact, the active context can be swapped so rapidly that a context can be processing data on one clock edge, switch to a new configuration (including data sharing) and be processing data in the new configuration on the very next clock edge. SPICE simulations indicate that it is possible to switch contexts in fewer than 5 nanoseconds. A caveat to this rapid context switching is that time will be required to distribute the "switch to" lines throughout the chip. These lines indicate which context the device is supposed to switch to upon receiving the "switch" signal. However, given that the "switch to" lines are stable, the context switch can take place as described above. Note that this delay in switching is merely a latency and can therefore be factored into the logic that initiates a switch. Since the switch can be initiated by the active context or via external stimulus this latency is easily accounted for.

2.7 Programming

The bitstreams for the CSRC FPGAs are downloaded serially. The user is required to specify which context is about to be downloaded and then supply a clock and data. By repeating this process four times, the user can configure all four on-chip configurations. Note that the configuration being downloaded can not be active during configuration download. However, inactive contexts can be downloaded while another context is active and running. Additionally, a bitstream may be downloaded by the active context. In this manner, one can envision the possibility of passing compressed or encrypted bitstreams into the active context so that it may download an inactive context after uncompressing or decrypting the bitstream.

The device will power up, prior to downloading bitstream(s), in a known state possessed by all four configurations. This provides the user with the ability to determine if the device is operational prior to use. The known state, although yet undetermined will either be benign or of some simple functionality providing some level of built-in self-test (BIST).

Software will be available with the device that generates bitstreams for the CSRC devices from VHDL. The user will be required to manually partition the logic among the contexts if the algorithm should span multiple contexts. However, in such a case, the toolkit will be capable of simultaneously placing and routing the logic in each context so that intermediate data passage is optimized. This is believed to be a key element of context switching data sharing because if each of the contexts, which are to share data, are individually and sequentially placed and routed "in a vacuum", the placement restraints imposed on each context would compound and rapidly become prohibitively constraining.

3 Status

Sanders is developing this technology in two phases. The first phase involves the development of a small prototype version of the CSRC technology. This chip will serve both as a concept validation tool and a platform for acquiring empirical data about the performance enhancements afforded by this new technology. The subsequent phase entails the development and fabrication of a larger version of the prototype with a few additional features. Both the prototype and the larger more capable final device are full custom IC designs being designed and fabricated on National Semiconductor's .35µ line. The prototype CSRC device is currently in fabrication while the final device is scheduled to be fabricated in September of 1998.

4 Acknowledgments

This material is based upon work supported by DARPA/ITO under contract number F30602-96-C-0350 and Sanders IR&D.

5 References

[1] A. DeHon, “DPGA-Coupled Microprocessors: Commodity Ics for the 21st Century”, IEEE Workshop on FPGAs for Custom Computing Machines, 1995.

[2] A. DeHon, “Reconfigurable Architectures for General-Purpose Computing”, PhD Dissertation – MIT, 1996.

[3] R. Bittner & P. Athanas, “Wormhole Run-Time Reconfiguration”, ACM/SIGDA International Symposium on Field Programmable Gate Arrays, 1997.

[4] J. Burns, A. Donlin, J. Hogg, S Singh, M. de Wit, “A Dynamic Reconfiguration Run-Time System”, IEEE Symposium on FPGAs for Custom Computing Machines, 1997.

[5] S. Kelem, “Mapping a Real-Time Video Algorithm to a Context-Switched FPGA”, Poster Session, IEEE Symposium on FPGAs for Custom Computing Machines, 1997.

[6] R. Ong, “Programmable Logic Device Which Stores More Than One Configuration and Means for Switching Configurations”, US Patent 5,426,378, 1995.

[7] S. Scalera, J. Murray, & S. Lease, “A Mathematical Benefit Analysis of Context Switching Reconfigurable Computing”, Reconfigurable Architectures Workshop, 1998.

[8] S. Trimberger, “A Time-Multiplexed FPGA”, IEEE Symposium on FPGAs for Custom Computing Machines, 1997.

[9] E. Tau, D. Chen, I. Eslick, J. Brown, A. DeHon, “A First Generation DPGA Implementation’, FPD ’94 – Third Canadian Workshop on Field-Programmable Devices, 1995.

[10] J. Villasenor, B. Schoner, K. Chia, C. Zapata, “Configurable Computing Solutions for Automatic Target Recognition”, IEEE Symposium on FPGAs for Custom Computing Machines, 1996.

[11] M.J. Wirthlin & B.L. Hutchings, “Density Enhancement of a Neural Network Using FPGAs and Run-Time Reconfiguration, IEEE Workshop on FPGAs for Custom Computing Machines, 1994.