Temporal Partitioning for Partially-Reconfigurable-Field-Programmable Gate

John Spillane
Georgia Institute of Technology
Department of Electrical and Computer Engineering
Atlanta, GA 30332
spillane@eecom.gatech.edu

Henry Owen
Georgia Institute of Technology
Department of Electrical and Computer Engineering
Atlanta, GA 30332
howen@eecom.gatech.edu

Abstract

The recent introduction of partially-reconfigurable-field-programmable gate arrays (PRFPGAs) has led to the need for new algorithms suited for use with these devices. Although algorithms developed for use with field-programmable gate arrays can be applied to PRFPGAs, these algorithms do not take advantage of features available in these new devices.

This paper examines the applicability of PRFPGAs in hardware emulation systems. A partitioning algorithm known as temporal partitioning is introduced for use with PRFPGA-based hardware emulation systems.

1.0 Introduction

Partitioning for field-programmable gate arrays (FPGAs) refers to the process of splitting a large circuit into smaller sub-circuits. The goal of partitioning is the creation of sub-circuits suitable for implementation in a single FPGA. These single FPGAs are interconnected in a way that allows the sub-circuits to communicate all necessary signals. The communicating sub-circuits are then known to implement the original circuit (which is assumed to not fit within a single FPGA).

Although standard partitioning algorithms can be applied to interconnected partially-reconfigurable-field-programmable gate arrays (PRFPGAs), these algorithms do not exploit the partial reconfigurability of the target devices. In this paper, we introduce a partitioning technique known as temporal partitioning which allows a single PRFPGA to perform functions previously requiring an interconnected group of FPGAs.

2.0 Temporal Partitioning

The minimum reconfiguration increment of PRFPGAs allows large circuits to be implemented by multiple reconfigurations of a single device. Accomplishing the same task with an FPGA requires an external memory coupled with on-chip control logic.

Figure 1 shows an example circuit graph represented by large groups of interconnected gates. In this example, each edge is labeled with the number of signals communicated from node to node and each node is labeled with an identifier and number of contained gates. The total number of gates represented in this graph is 19,000. The target PRFGPA device for this example has a static capacity of 16,000 gates. While multiple groups may fit within the target PRFPGA, the entire circuit has a gate count larger than the static gate capacity of the device.

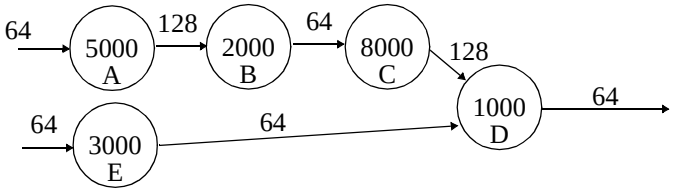

Figure 1: Example Circuit Graph

With multiple reconfigurations, a single PRFPGA can implement this large circuit. Figure 2 shows one possible schedule for implementation of the example circuit.In this figure, one PRFPGA is assigned three different configurations: C_0, C_1, and C_2. Communication between groups is handled by on-chip resources named signal storage points (SSPs). In the XC6200 series, SSPs can be implemented using available configurable logic block (CLB) registers. Groups, or partitions, of the circuit store their outputs in SSPs. The section of the device containing the SSPs is held while the section containing used circuitry is reconfigured. The newly configured circuit must be properly connected to the existing SSPs to provide the partition to partition interconnection.

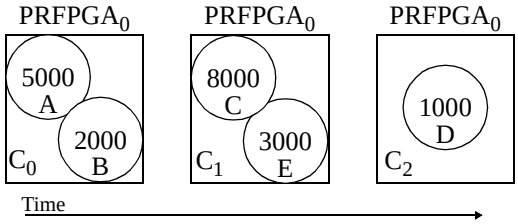

Figure 2: Example Implementation Schedule

We refer to the process of clustering and then scheduling a design in this manner as temporal partitioning. The next sections detail our research into this new methodology.

2.1 Clustering

Clustering is the first step in temporal partitioning. The objective of the clustering algorithm is to group nodes such that they may be subsequently scheduled. The cones partitioning algorithm provides the basis for our clustering stage. Clusters are formed as follows:

* find external outputs

* from each external output: recurse into the network and mark nodes with a corresponding label

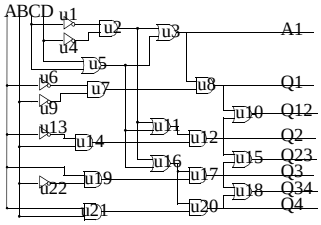

Figure 3: Example Digital Circuit

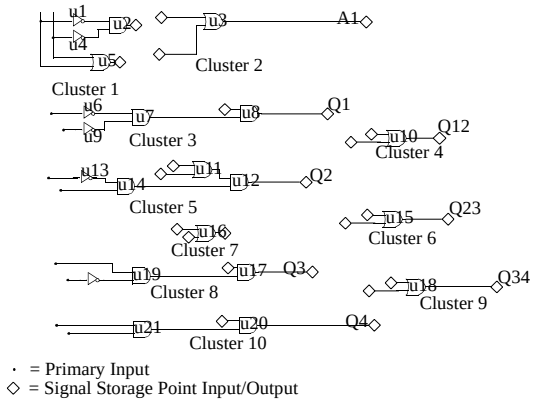

Figure 4: Example Digital Circuit - Clustered

*group nodes with equivalent label lists into clusters
*insert SSPs between clusters for signal retention during reconfiguration

These are similar to the first few steps in cone partitioning[1]. Currently our algorithm is being tested on feed-forward asynchronous digital networks. With this constraint, synchronous elements are not dealt with explicitly. Figure 3 shows an example digital circuit. Figure 4 shows the result of clustering on the example circuit.

2.2 Scheduling

Once the circuit is divided into clusters, the scheduling algorithm orders the clusters for implementation in the PRFPGA. We have investigated the performance of four variations of scheduling based on two parameters. The first parameter is the type of SSP implementation. SSPs can be implemented as queues or registers. The second parameter is the targeted reconfiguration level of the chip. We define the CLB array of the device as an *M*x*M* matrix of CLBs. The targeted reconfiguration levels are: M^2, $M^2/2$, and *M*. Due to sharing requirements, the queue SSP implementation was examined only in combination with the M^2 level of reconfiguration.

2.3 System Gate-Capacity

One measure of hardware emulation performance is the gate capacity of the system. FPGA-based emulation systems scale system gate capacity by adding more FPGAs to the system. For extremely large designs, multiple systems are interconnected to meet capacity requirements[2].

A PRFPGA-based emulation system has unlimited gate capacity. The performance limiting factor is peak SSP usage. When all SSPs on the device are used, communication with the host processor accommodates nearly infinite SSP storage. Along with increased capacity comes reduced performance. Note that a PRFPGA-based system is able to trade emulation speed for gate capacity. In a typical FPGA-based system, if the gate capacity is exceeded the designer must expand the system.

2.4 System Emulation Speed

Our performance analysis of temporal partitioning focuses on the time taken to compute a result for a given circuit input. Figure 5 shows the system timing.

All times will be presented as $t_z(x)$ where z identifies the time interval and x represents a particular combination of SSP implementation and targeted reconfiguration level. The time $t_1(x)$ represents initial configuration time for the system. This

involves one configuration at the targeted level. The time $t_2(x)$ represents the amount of time between the presentation of a test vector and the production of the result from that vector. This involves multiple reconfigurations of the PRFPGA. The time $t_3(x)$ represents the amount of time between the production of the result from one test vector and the presentation of the next test vector. This involves one configuration at the targeted level.

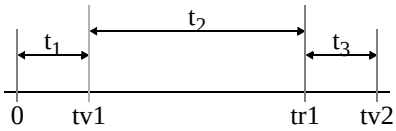

Figure 5: System Timing

Possible combinations for SSP implementations and targeted reconfiguration levels are shown along with their designated abbreviation in Table 1.

SSP Implementation	Reconfiguration Level	Abbreviation
Queue	M^2	Q-M^2
Register	M^2	R-M^2
Register	$M^2/2$	R-$M^2/2$
Register	M	R-M

Table 1: Abbreviations For SSP/Reconfiguration

In the following sections, we focus on the $t_2(x)$ component of the system timing. This delay component involves multiple reconfigurations of the PRFPGA and is the dominate term in system delay.

2.4.1 Performance Of R-M^2

We first examine scheduling based on reconfiguring the entire device while using registers to implement SSPs. Although this level of reconfiguration does not take full advantage of the PRFPGA, it provides a starting point for comparison with other levels of reconfiguration.

The time $t_2(R\text{-}M^2)$ is represented as the sum across all n configurations required to implement the design of the reconfiguration time, ρ, and the emulation time, η, for each configuration. The worst-case equation for $t_2(R\text{-}M^2)$ is shown below:

$$t_2(R-M^2) = \sum_{i=1}^{n_1} \rho(M^2)_i + \eta(M^2)_i$$

We note that for the XC6200, reconfiguration times for the entire device are on the order of tens of milliseconds. Although there is no real bound on the worst-case emulation time, we assume it to be on the order of tens of microseconds.

2.4.2 Performance Of Q-M^2

When reconfiguring the entire device, one method of improving performance is to present multiple test vectors at once. This is accomplished by implementing the SSP in a queue. Queue-based SSPs allow multiple test vectors to be processed by each configuration. The worst-case average for t_2(Q-M^2), with a queue depth of D, is shown below.

$$\overline{t_2(Q-M^2)} = \frac{1}{D}\sum_{i=1}^{''_1} \rho(M^2)_i + \eta(M^2)_i$$

Implementation of queue-based SSPs within the XC6200 series PRFPGA is difficult since the CLBs cannot be used as local memory. Since the area required to implement queue-based SSPs is large in the XC6200, a new PRFPGA architecture is required before this strategy is viable.

2.4.3 Performance Of R-$M^2/2$

Configuring the entire device does not take full advantage of the PRFPGA. The

$$t_2\left(R-\frac{M^2}{2}\right) = \sum_{i=1}^{n} \rho\left(\frac{M^2}{2}\right)_i + \eta\left(\frac{M^2}{2}\right)_i$$

equation for the worst-case time for t_2(R-M^2/2) is shown above. We will show that this reconfiguration level makes it possible to reduce the contribution of the emulation delay term. The reduction of this contribution is possible under the following constraint:

$$\rho\left(\frac{M^2}{2}\right)_i > \eta\left(\frac{M^2}{2}\right)_{i-1}$$

Given this constraint, we see that as a following section is being configured the previous section is processing data. Further, the previous section has completed all processing before the following section is completely configured. It follows that the t_2(R-M^2/2) reduces to:

$$t_2\left(R-\frac{M^2}{2}\right) = \eta\left(\frac{M^2}{2}\right)_{n_2} + \sum_{i=1}^{n_2} \rho\left(\frac{M^2}{2}\right)_i$$

Comparing this result with that found for t_2(R-M^2), we find that we have significantly reduced the contribution of the emulation delay term. Given the current orders of magnitude difference between emulation delay and reconfiguration time, this is currently not a significant result. As reconfiguration times approach emulation times this level of reconfiguration is more appropriate. The total number of configurations required, n_2, is approximately double that required for t_2(R-M^2). Assuming the reconfiguration time scales with reconfiguration level, the total time spent configuring the device is unchanged.

2.4.4 Performance Of R-M

With the dominance of configuration time, even the near elimination of the emulation time factor has little impact on system performance. One method of reducing configuration time is to reduce the amount of circuitry configured. This reduction

results in a likewise increase in the number of configurations required, again the total time spent in configuration is unchanged. One possibility for the reduction of configuration time is through the use of specialized addressing techniques available with the XC6200. Through a method known as wildcarding, multiple CLBs are configured simultaneously. Each of these CLBs is identically configured. Since each XC6200 series CLB is likely to handle one bit of an operation, this technique can reduce required configuration times for circuits involving operations on multi-bit signals. The worst-case equation for t_2(R-M) is shown below.

$$t_2(R-M) = \sum_{i=1}^{n} \rho(M)_i + \eta(M)_i$$

If we again place the constraint:

$$\rho(R-M)_i > \eta(R-M)_{i-1}$$

The equation for t2(R-M) reduces to:.

$$t_2(R-M) = \eta(M)_{n_3} + \sum_{i=1}^{n_3} \rho(M)_i$$

2.4.5 Best-Case Performance

All t_2(x) equations given have been worst-case equations. These equations assume that every configuration must be downloaded and executed before a result is complete. In fact, the entire circuit may not be required for the computation of a particular output[3].

3.0 Conclusions and Future Work

We have shown that it is possible to create a hardware emulation system from a single PRFPGA. This is possible through the use of temporal partitioning algorithms currently in development. Future work includes further system characterization through the use of large, computer generated circuits as well as trimming of unnecessary configurations.

4.0 References

[1] D. Brasen, J.P. Hiol, G. Saucier, “Finding Best Cones From Random Clusters For FPGA Package Partitioning,” IFIP International Conference on Very Large Scale Integration, Aug 1995, pp. 799-804.

[2] Gateley, et. al., “UltraSPARC^{tm}-I Emulation,” 32nd ACM/IEEE Design Automation Conference, Jun 1995, pp. 13-18.

[3] S. Smith, M. Mercer, B. Brock, “Demand Driven Simulation: BACKSIM,” 24th ACM/IEEE Design Automation Conference, Jun 1987, pp. 181-187.