A Novel High Throughput Reconfigurable FPGA Architecture

Amit Singh, Luca Macchiarulo, Arindam Mukherjee, Malgorzata Marek-Sadowska
Department of Electrical and Computer Engineering
University of California, Santa Barbara
Santa Barbara, CA 93106
{asingh,luca,arindam,mms}@guitar.ece.ucsb.edu

Abstract

With increased logic density due to the shift towards Deep Submicron technologies (DSM), FPGAs have become a viable option for implementing large designs. However, most commercial FPGAs, due to their general purpose architectural nature, cannot handle designs which require very high throughput. In this paper, we propose a novel high throughput FPGA architecture which tries to combine the high-performance of Application Specific Integrated Circuits (ASICs) and the flexibility afforded by the reconfigurability of FPGAs. This architecture utilizes the concept of 'Wave-Steering' and works best for designs which are highly regular and have almost equal delays along all paths. It has enormous potential in Digital Signal and Image Processing applications since a good portion of these applications are regular in nature. Preliminary results for some commonly used DSP designs are encouraging and yield throughputs in the neighborhood of 770 MHz in 0.5µ CMOS technology.

1 Introduction

The first FPGA was introduced in 1985 and since then, FPGAs have become increasingly popular for their ability to be a low cost solution in a variety of design applications. The advent of DSM technologies has given rise to million gate FPGAs, thereby making them increasingly versatile. However, FPGAs lag far behind Application Specific Integrated Circuits (ASICs) when it comes to the speed of the designs which can be accommodated. Most commercial FPGAs, due to their inherent general purpose architectural nature, cannot handle designs that require very high throughput. With shrinking device size, interconnect delays are increasingly becoming a bottleneck for achieving fast clock speeds on these general purpose architectures.This reduces the attractiveness of using FPGAs for high speed Digital Signal and Image processing applications as well as applications that require very high throughput.

In this paper, we propose an application specific FPGA architecture fabric which targets regular circuits. Our definition of regular circuits encompasses all circuits that have almost equal delays along all paths. This definition includes all designs in which connection between different blocks is constrained to be local (not necessarily next neighbor) and single modules have similar complexity (note that this does not impose the same functionality on different blocks). A range of systolic and arithmetic circuits fall into this category. Our architecture uses a 'Wave-Steered' [13][14] approach to implement circuits in Pass Transistor Logic (PTL) mapped Decision trees. This new proposed architecture could possibly be embedded in a larger general purpose architecture thereby giving both the host and the embedded architectures greater flexibility. Our proposed architecture is attractive in the FPGA domain because in situations where the design specifications change from one implementation of the design to another, custom logic would prove expensive. Our goal is to find a middle ground between the FPGA and the ASIC world by combining the performance of ASICs and the flexibility of FPGAs.

We organize the rest of the paper as follows: Section 2 describes the key architectural ideas on which our proposed architecture is based. They include Binary Decision trees and the concept of 'Wave-Steering' [13][14]. Section 3 describes the Logic Block (LB) architecture. All basic building blocks that make up our LB are discussed in this section. Section 4 describes the routing/interconnect fabric used in our proposed architecture. Section 5 provides experimental results and an analysis of these results. The next section explores future possibilities and enhancements that will increase the attractiveness of this FPGA. This is followed by Conclusions.

2 Binary Decision Trees and 'Wave-Steering'

This section discusses the main motivation behind our architectural approach, including the use of Pass Transistor Logic (PTL) mapped Decision trees in our Logic Blocks and a unique modification to the classical Wave-Pipelining methodology. We call this methodology 'Wave-Steering'[13][14].

2.1 Binary Decision Trees

A Binary Decision Diagram (BDD)[5] is a graph in which each vertex either has exactly two successors (or children), distinguished as high and low child, or no successor (in this case it is called a leaf). If the BDD has no nodes which are simultaneously children of different parents, its graph is a tree. It is possible to use PTL logic to map this tree structure without the risk of sneak paths. A complete balanced tree of height n, where every node has exactly 2 children labelled with the successive variable, and terminated with 2^n leaves, can therefore represent any possible function of up to n variables, and the values of the leaves mimic exactly the truth table of the function. We utilize this mapping methodology to create an architecture that has as its Logic Blocks, PTL mapped Binary Decision trees. In the proposed architecture, there's no

need for any customization inside the tree. The only customization for implementing different designs occurs in the way static RAM memory cells are programmed (i.e. different combinations of '1's and '0's). This means that to implement any 4 variable function for any possible permutation of the inputs, we only need to program these RAM cells uniquely. This is important to get the best possible performance, as it allows the functional part of the cell to work without added logic. This tree based method of realizing functions forms the basis of our cell architecture. We discuss the Logic Block architecture in Section 3.

2.2 Wave-Pipelining and 'Wave-Steering'

The PTL mapped Binary tree based Logic Block architecture uses a modification to the classical Wave-Pipelining concept [8][12] (called 'Wave-Steering' [13][14]) to achieve high throughputs. To understand 'Wave-Steering', it is essential to recognize the difference between a typical conventional combinational circuit and a Wave-Pipelined circuit. In a conventional circuit, current data must propagate to the output latch of the circuit before the next wave of inputs can be pushed in. It is necessary to wait this long because for different inputs, different input-output paths are activated and each path can have different delay. Figures 1.a - 1.f illustrate this point. Each of the triangles represents a multi-level combinational

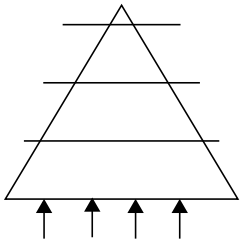


Figure 1.a

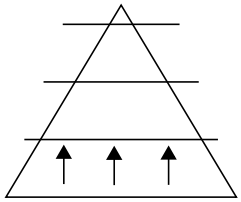


Figure 1.b

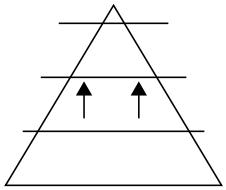


Figure 1.c

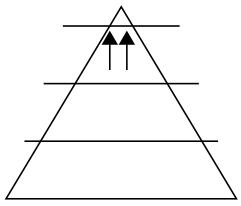


Figure 1.d

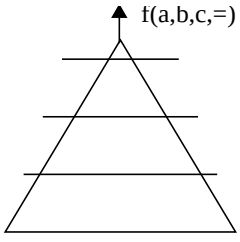


Figure 1.e

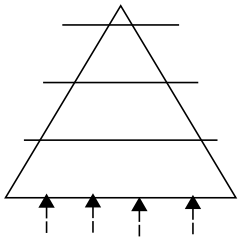


Figure 1.f

Figures 1.a-1.f:Conventional Combinational Circuit Operation

circuit. A set of inputs is pushed in (Figure 1.a) and the wave propagates through successive levels of logic until the output is produced (Figure 1.b - 1.e). Only then (Figure 1.f) can the next set of inputs be pushed in. In this case, the throughput of the circuit equals its latency. However, if we can synthesize a fairly regular circuit such that all paths have almost equal delays, then more than one data wave can exist between two clock cycles. This is true because there is no need for the previous data to be latched into the output flip-flops before pushing in the next set of inputs (in other words, internal node capacitances act as latches for the incoming waves). This is illustrated in the set of Figures 2.a-2.c. Each triangle in the Figures 2.a - 2.c represents a multilevel combinational circuit having all paths of almost equal delays. This makes it possible for a new set of inputs to be pushed in after each clock cycle and the waves continue to propagate upwards. Figure 2.a shows a set of inputs being pushed in. In the next clock cycle (Figure 2.b), this wave is propagated upwards and the next set of inputs (dashed lines) is pushed in. Figure 2.c captures the snap-shot in the third cycle, where the wave has propagated further up, the second wave is present below this wave and the third set of inputs (dotted line) is being pushed in. Hence, different waves can exist in any time snap-shot. This is the underlying principle of Wave-Pipelining.

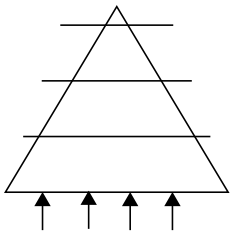


Figure 2.a

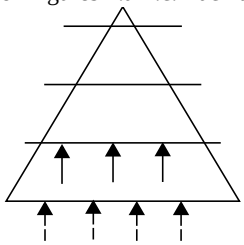


Figure 2.b

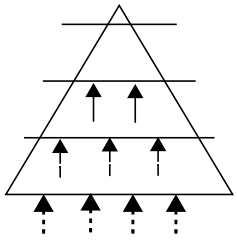


Figure 2.c

Figures 2.a-2.c: A Wave-Pipelined Combinational Circuit

In 'Wave-Steering' multiple data waves corresponding to successive input vectors are made to coexist in a target circuit by skewing the input vectors in time. Although this may resemble a micro-pipelining scheme, it is fundamentally different in the sense that the input application points "spatially follow" the pipelined stages. In a typical pipelined circuit, all the inputs are applied before the first stage of latches, and the outputs are available after the last stage. Fine granularity pipelining of PTL mapped Decision Diagram structures inherently have input application points physically distributed along the stages of the pipeline, where each stage corresponds to a level characterized by a single variable. This requires the input variables corresponding to the same vector to be applied with relative timing skews.

The timing skew between two variables characterizing two successive stages (levels) in such a Wave Steered structure would typically be one stage delay. This skewing is accomplished by a chain of flip-flops and a unique clocking scheme. This will guarantee the operation of the circuit at a given frequency by construction. In the

Wave Steered approach, a regular design is synthesized as a Decision tree. Each level in the tree corresponds to a particular variable in the function (see Figure 3).

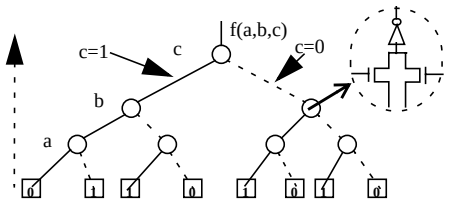


Figure 3: f(a,b,c) evaluated from bottom to top as a Decision Diagram Tree

This tree is evaluated level by level, starting from the leaf nodes. Logic '1' and '0' form the leaf nodes and the output of the node at the topmost level evaluates the function. Each tree node is directly mapped into Pass Transistor Logic (PTL), by replacing tree nodes with a 2:1 multiplexer followed by an inverter. This inverter accounts for voltage degradation between the levels. Alternate Mux levels are called $\phi 1$ and $\phi 2$ levels (Figure 6). In any $\phi 1$ level, the controlling input (or its complement) is logic '1' and data is propagated to the next level during this phase $\phi 1$ of the clock. During phase $\phi 2$ this particular level holds the logic level in the nfet output and inverter gate capacitances, as both the mux's selector lines remain at logic '0'. This provides the electrical isolations between successive data waves. Once the function has been partially evaluated for a particular variable in a level, only the combined information needs to be propagated and the value of that particular variable is no longer needed. In order to clock the circuit faster, we 'Wave Steer' it. To do this, we skew the input vectors in time so that the current input variable selects a path only after the previous variable in the vector had selected the correct path. Using this concept of 'Wave-Steering' [13][14], we propose a new Logic Block Architecture.

3 Logic Block Architecture

In this section, we discuss our Logic Block architecture. At present, our architecture works on circuits without feedback. Figure 4 shows a block diagram of a reconfigurable Logic Block (LB) slice.

A LB slice is a 4-input LUT that can implement any function of upto 4 variables. Each of our Logic Blocks is composed of two such slices, with each slice built of a Pass Transistor Logic (PTL) mapped 2:1 multiplexer Binary Decision tree. Figure 3, for example, shows a function f(a,b,c) of 3 inputs, synthesized as a tree, with each tree node mapped into PTL. In our architecture, each PTL mapped tree has 4 levels, with each level corresponding to an input variable. Input variables are clocked by a 2-level phase clock, with alternate phases (ϕ_1 and ϕ_2) clocking consecutive levels (Figure 6). The ϕ_2 clock is the non-overlapping inverted ϕ_1 clock. Inputs are fed to each tree through a feeder circuit (Figure 5) which is composed of chains of master-slave flip-flops. This feeder circuit is necessary to skew the input vectors in time and achieve the desired 'Wave Steering' effect. There are two types of dynamic flip-flop cells in the feeder circuit, the F1 and F2. The F1-F2 pair will get the data out by the end of ϕ_2 phase while the F2-F1 phase will get the data out by the end of ϕ_1 phase. Each set of flip-flop cells feeds a clocked Nand gate and an inverter that act as a buffer assembly. This clocked Nand-inverter couple is used for driving

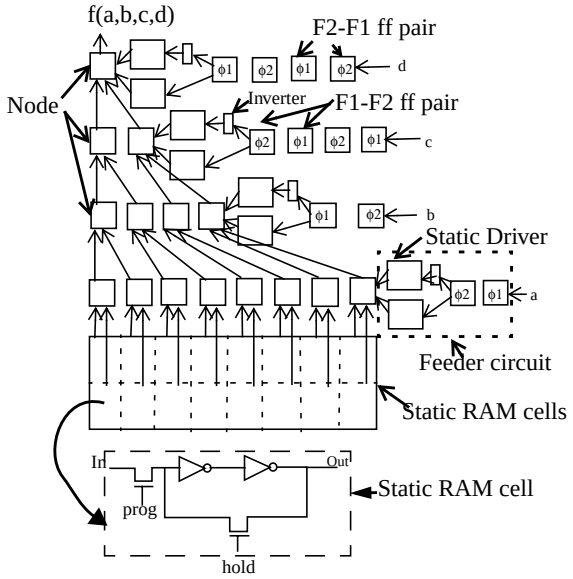
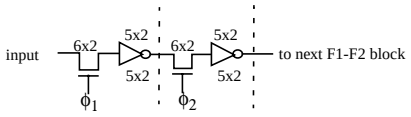
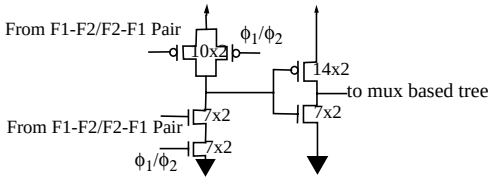


Figure 4: Block-Diagram of a Logic Block Slice



Part of a "Feeder-Circuit" Flip-Flop Chain showing a basic master-slave F1-F2 cell



Driver Assembly (part of Feeder circuit) feeding the Mux Based Decision Diagram Tree

Figure 5 Feeder Circuit

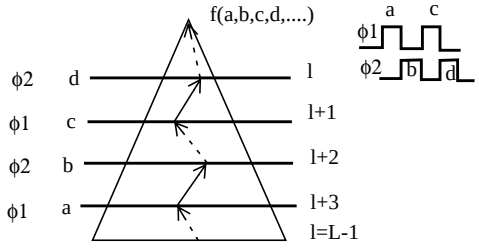


Figure 6: Wave-Steering

the mux tree with the signal values during only one phase of the clock and blocking the data transfer during the alternate phase.

We use static RAM memory cells (see Figure 4) for storing logic '1's and '0's. Each RAM cell is made up of 6 transistors and all 16

RAM cells in a LB slice can be programmed through a scan path in a bit-serial manner. All of our RAM cells consist of minimum width transistors since simulations show that these minimum width transistors are adequate for our purpose.

Figures 7.a through 7.f illustrate Wave-Steering on a PTL mapped tree. Figure 7.a shows a PTL mapped tree for a function of 3 variables f(a,b,c). The dashed arrows show the direction of the inputs (and their complements). The sequence of inputs applied to this tree is shown in Table 1.

Table 1: Sequence of inputs

Variable \ Stage	i	i+1	i+2
a	1	0	1
b	0	1	0
c	1	0	1

Figure 7.a is a snap-shot of the circuit behavior during the first ϕ_1 phase. During this phase, level 'a' is clocked and since 'a' is at Logic '1', the transistors controlled by 'a' are switched 'on' (dark solid arrows). Note that during this phase, level 'c' is also active. However, no data wave has reached level 'c' as yet because of input skewing. Figure 7.b shows what happens during the first ϕ_2 clock phase. In this snap-shot, 'b' is set to logic '0' and the transistors in the 'b' level, selected by b are switched 'on'. The data wave hence, propagates upwards as shown by the dark solid arrows. (The dark solid arrows show the i_{th} data wave) Figure 7.c captures the time-frame during the 2nd ϕ_1 clock phase. Here, a new value is set for variable 'a' and equals logic '0'. This corresponds to the i+1 stage and is illustrated by dark dashed arrows. In this phase, level 'c' is also active but the data wave present here is from the i_{th} stage (shown in figure by solid line and corresponding to c=1). As we can see, 2 data waves exist concurrently in the tree structure. The i_{th} data wave has propagated to level 'c' while the i+1 data wave has been introduced in level 'a'. An important fact to note is that two consecutive levels are clocked in alternate phases of the clock. Figure 7.d shows the data-wave propagation during the 2nd ϕ_2 phase. In this phase only level 'b' is active and according to table 1, b is set to logic '1'. The dashed arrow (corresponding to the i+1 stage) shows the wave propagation.

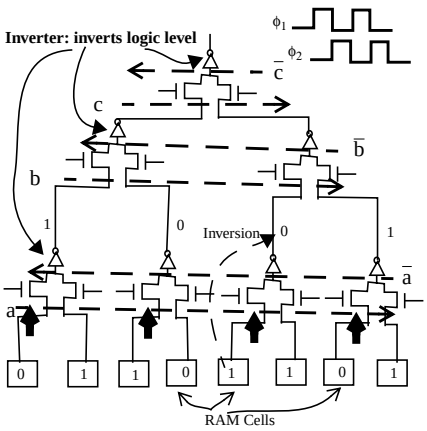


Figure 7.a: Wave-Steering

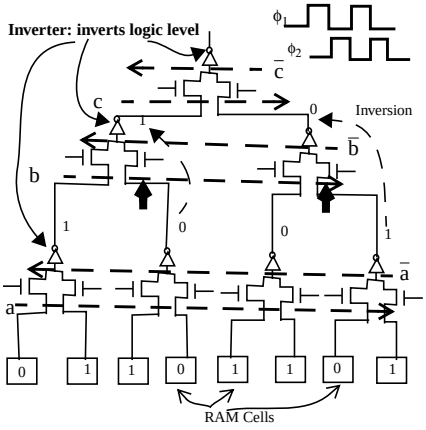


Figure 7.b: Wave-Steering

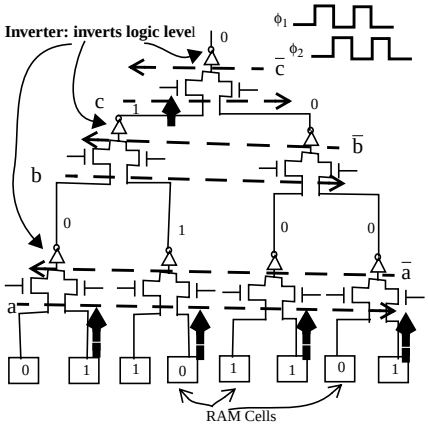


Figure 7.c: Wave-Steering

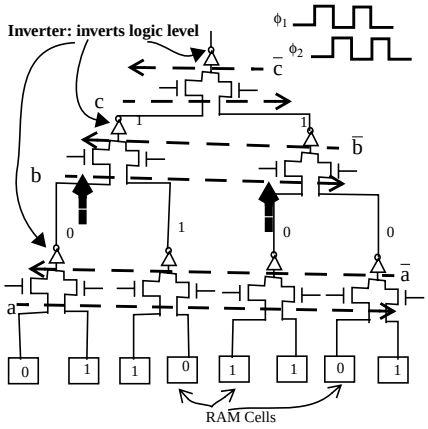


Figure 7.d: Wave-Steering

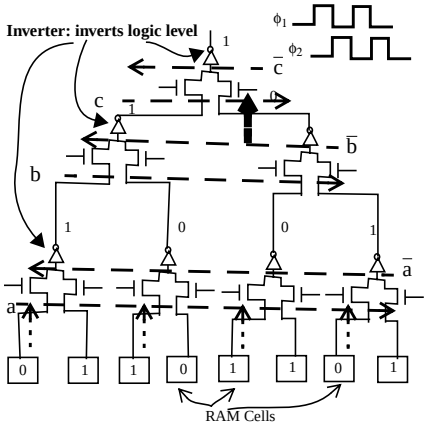


Figure 7.e: Wave-Steering

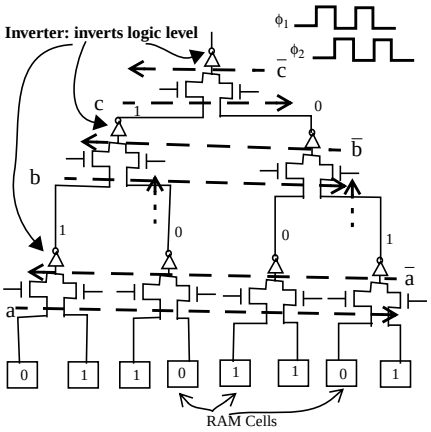


Figure 7.f: Wave-Steering

Figure 7.e captures the time-frame during the 3rd ϕ_1 phase. The i+1 data wave has propagated to level 'c' (shown by dark dashed lines) and the i+2 stage input is being fed in through level 'a'. Note that in stage i+2, 'a' is assigned logic '1', hence all the transistors

in level ‘a’ that are being fed by ‘a’ are switched ‘on’. Finally, Figure 7.f shows the circuit behavior during the ϕ_2 phase. Here the data wave from the i+2 stage has propagated to level ‘b’. Again, note that even though our idea is akin to the classical Wave-Pipelining concept, we use the name “Wave-Steering” to emphasize the difference w.r.t. Wave-Pipelining: in our case, we introduce different data waves in the circuit through the skewing of input vectors.

To build a larger circuit, the design has to be decomposed into 4 input(variable) blocks. At present, we do this decomposition manually. However, any mapping algorithm should work fine for us, as long as we incorporate the additional constraints imposed by our architecture. Since each of our CLBs is made up of 2 “4-input LUTs”, one approach would be to take any 4-input LUT implemented “regular” design on a commercially available FPGA, and implement each LUT function in our LUT, since our LUT can implement any 4-input function by just programming the “1’s” and “0’s” differently. To achieve better optimized circuits, a new decomposition would have to be proposed. This work is currently in progress.

To allow each Logic Block to communicate with its neighbors, we propose an interconnect architecture that suits our goals. It is discussed in the next section.

4 Routing and Interconnect

Before we go into the details of our routing architecture, it is important for us to re-emphasize that our proposed FPGA architecture is specialized in that it targets designs that are regular. Regular designs inherently have local communication and involve iterative communication. With this is mind, we propose dedicated interconnects between different blocks without any crossbar routing switches. Figure 8 presents a global view of how different blocks are connected to each other. Logic blocks have next-neighbor communications, use dedicated interconnects, and can communicate with blocks at most 5 rows/columns away (at a speed of 715 MHz and all 5 tap switches closed). In Figure 8, LB_A can communicate

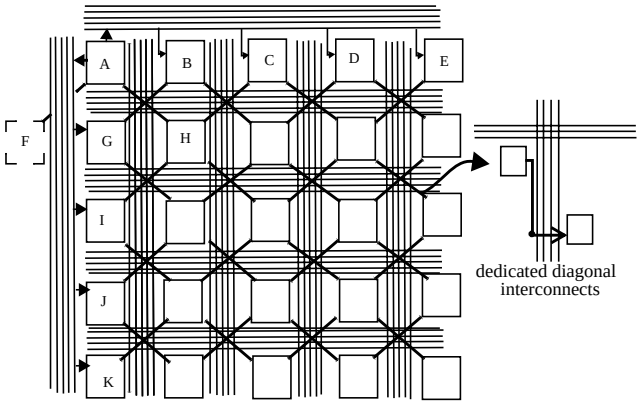


Figure 8: Preliminary Interconnect scheme

with LB_B, LB_C, LB_D, LB_E and LB_H at any point in time. It can also communicate with LBs G, I, J, K and F. Similarly, logic blocks LB_B, LB_C, LB_D, LB_E and LB_H can also communicate to LBs that span upto 5 rows/columns across including their next diagonal neighbors. A particular logic block can place its output on the horizontal/vertical wires running along the array and other neighboring blocks tap this signal. Blocks can also communicate diagonally to their next neighbor blocks. For simulation we model interconnects as RC π chains (each interconnect has 4 such chains). Figure 9 shows a general model for inter-block communi-

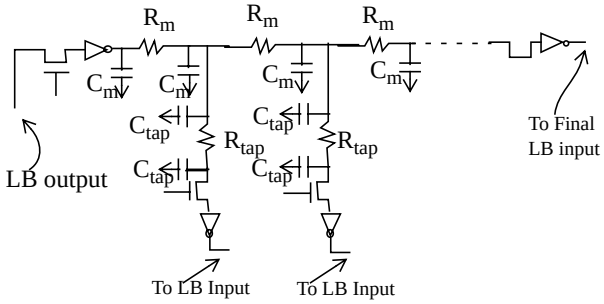


Figure 9: Interconnect Model, shown here with taps

cation interconnect (including taps models).

We performed the following HSPICE simulations (For simplicity, we use an array multiplier as our initial test design):
1) interconnects without any taps
2) interconnects with taps but all tap switches open
3) interconnects with taps any 5 tap switches closed

1) Interconnects without any taps: Such interconnects can span at most 5 Logic Block rows/columns across with the design running at 770 MHz. Since our array multiplier has strict next-neighbor communications, we determine this by increasing the wire length between 2 next-neighbor blocks to span a length of 5 blocks. The interconnect was modeled as in Figure 9, without any taps (i.e, the interconnect is an RC π chain). This simulation is for determining the maximum allowable interconnect length at an operating frequency of 770 MHz (this corresponds to a cycle time of 1.3ns). Note that each Logic Block can individually operate at 833 MHz (1.2ns). Cycle times of less than 1.2 ns do not yield correct results.

2) Interconnects with taps and all switches open: This experiment is similar to the one in 1), except that we have metal 1 taps from the interconnect to Logic Block inputs. The blocks are spanned by the interconnect described in 1). All taps have open switches in this setup. Simulations show that the design (array multiplier) can run at 770MHz with all tap switches open.

3) Interconnects with taps and all 5 tap switches closed: The experimental setup is the same as in 2). The interconnect still spans 5 logic blocks and with all 5 tap switches closed, the design can run at 715 MHz (this corresponds to a cycle time of 1.4ns). This implies that at any point in time, a Logic Block can communicate, at an operating frequency of 715 MHz, with at most 5 neighboring (row or column wise) blocks (including diagonal communication). This interconnect simulation is the most general of the three.

Figure 8 shows the proposed routing model for our architecture. Each LB can communicate with another Logic Block at most 5 rows/columns away (and with all intermediate LBs) while running at 715 MHz. Note that there are no routing switches, and we use only dedicated interconnect. Communication between any 2 LBs is full-duplex.

The proposed architecture does not have a general purpose routing fabric. Here, it means that an arbitrary Logic Block can not communicate with any other Logic Block that spans more than 5 rows/ columns. While this routing scheme may seem too restrictive, we tested this scheme on 3 very regular designs and found them to be easily routable. Note that these 3 designs did not use any Logic Blocks as pure routing blocks. In order to make our routing scheme more robust, we wish to look at applications that demonstrate characteristics of a fair amount of regularity but can also have some portions of the circuit that are irregular. Our goal is to show that these fairly regular circuits can be routed in our modified architecture. To make our architecture more flexible, we propose to pipeline part of the interconnect and make some modifications to the structure of some of our Logic Blocks. Figure 10 shows one

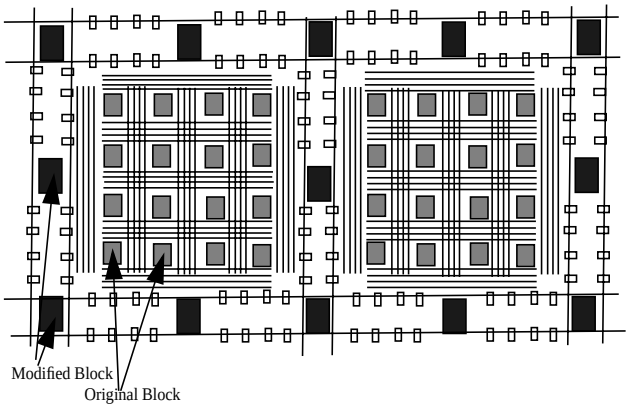


Figure 10: A Hybrid Clustered FPGA layout

setup we are researching. This setup consists of 2 kinds of different Logic Blocks. The external Logic Blocks will only consist of 2 4-level trees with the flip-flops (forming the feeder part of the LB) distributed around them on the interconnect. The internal Logic Blocks have the feeder part of the LBs confined inside the LB as discussed in Section 3. The internal LBs form a cluster of 4x4 and the modified LBs are spread on the periphery of this cluster. The idea for such a setup is that the internal clustered blocks (light shaded) act as logic intensive blocks while the modified logic blocks on the periphery of a 4x4 clustered block act as pseudo-switchboxes which also act as synchronizing points for the different waves coming in. More research as to the feasibility of this modification is currently under way.

Internal LBs in the setup of Figure 10 can also be used for routing purpose. These LBs can be used to bend signals in cases where a horizontal-vertical or vertical-horizontal communication is required. More research into this enhanced routing scheme is currently underway. Preliminary results show that on average, each cluster uses only 10 internal LBs for logic purposes while leaving 6 LBs empty. These 6 LBs can be used for routing within the cluster.

5 Results and Analysis

Layouts of the LBs were done in 0.5µ 2-metal CMOS technology. Each LB occupies 720λ x 920λ. Figure 11 shows a picture of a single LB (made up of 2 slices) laid out in 0.5µ CMOS technology. Assuming a die size of 1.5cm x 1.5cm and an allocated LB area of 1cm x 1cm, our proposed chip can be occupied by approximately 1500 such LBs. To further test the feasibility of this architecture, we note that most DSP related applications are inherently regular, use Multiply and Accumulate (MAC) units intensively and hence would fit naturally in our architecture.

Table 2: Experimental Results

Design	XC4000	Virtex	Ours
Array Multiplier			
Throughput (MHz)	59 MHz	65 MHz	770 MHz+
Area (# Blocks)	40	35	64
Bounding Box (%)	~ 4%	~ 3%	4.25%
4 tap bit-level systolic FIR filter			
Throughput	44.3 MHz	45.5 MHz	770 MHz
Area (# Blocks)	145	139	384
Bounding Box (%)	~ 12%	~ 12%	21%
Auto cross-correlator			
Throughput	(39.5 MHz)# 250MHz*	_____	770 MHz
Area (# Blocks)	160	_____	210
Bounding Box (%)	~10%	_____	12%

* This 250 MHz figure is reported directly from [10] and was in 0.7µm technology for a XC3000 device. This is the fastest reported result for this design (or for any design on a commercial FPGA)

+ While the array multiplier can be simulated to run at 833 MHz, this result reflects the fact that we account for inputs being fed to the multiplier. Interconnects, thus, are not purely next-neighbor. Note that this result also takes into account the presence of taps from these inputs .

Throughput obtained by implementing the design without customization.

We (manually) mapped 3 designs onto our architecture. These include a 4x4 array multiplier, a cross correlator [10] with 10 stages and a 4 tap bit-level systolic FIR filter. The designs were laid out and the extracted SPICE netlists were simulated. Typical voltage/temperature values were used in these simulations. The interconnect model of Figure 8 was used in these simulations and the 3 tested designs were found to be easily routable using this

scheme. Note that these designs are all of a very regular nature and involve next-neighbor communication between blocks. Simulation results for the multiplier, filter and the cross-correlator yield throughputs of 770 MHz (if we include the line-to-line-capacitance in these simulations, the obtained throughput reduces to 625 MHz). The average power dissipation is 7.5mW/block at 770 MHz. Note that local and next neighbor interconnects along with their programming switches are included in these simulation results. For the correlator design, no manual customization/retiming took place in our implementation unlike in [10]. These designs were also implemented in Xilinx's XC4000 series FPGA and their newest FPGA, Virtex for comparison purposes. Table 2 presents the results for these designs on both the architectures. Designs in the XC4000 and Virtex, are in Verilog and are implemented using Synopsys' FPGA Express and Xilinx's M1 place and route tool. The correlator design could not be implemented in the VIRTEX device due to software related problems.

Table 2 shows that at a slight cost in area, our proposed architecture does much better in speed for the designs under test. We acknowledge that we did not perform any retiming when implementing these designs in Xilinx devices. However, parts of these designs were taken directly from the Xilinx CORE generator and modified to meet our comparison needs. Also, note that Xilinx's XC4000 series (as used in our implementation) is in 0.35µ CMOS while VIRTEX is in 0.25µ CMOS. We have designed our LB array in 0.5µ CMOS. Our experimental results are not meant to be a justification of the architecture in terms of empirical data. While we acknowledge that Table 2 is not a true comparison of results, our mission is to demonstrate the enormous potential of our proposed architecture. Even though we have not fabricated our FPGA chip and provide only HSPICE simulation results, it is quite interesting to see that this architecture can achieve, for designs that are regular (most arithmetic circuits), throughputs in the range of 770 MHz (625 MHz if we include the line-to-line capacitances). We note here, that for customized designs, like [10], implemented on Xilinx FPGAs, substantially improved throughput can be achieved. In [10], the author achieves a sampling rate of 250 MHz (in a 0.7µm Xilinx device) for the correlator by choosing this particular clock frequency (250 MHz) and customizing/retiming the design to operate at this frequency. However, in general, most designs, until they are customized and retimed like in [10], are still hard pressed to achieve throughputs exceeding 100 MHz. Thus, our architecture is a substantial improvement in terms of throughput, since a major drawback for most commercially available FPGAs is that they cannot implement designs that require high throughput. Our throughput performance approaches that of ASICs and affords the end user with the added reconfigurability.

6 Discussion and Suggested Improvements

In section 4, we discussed the possibility of pipelining parts of the interconnect to limit long interconnects. But for this kind of pipelining to work, we need to make some modifications to the basic architecture. One such modification was discussed in Section 4 and shown in Figure 10. Instead of confining flip-flops used for skewing input vectors inside the confines of a LB, we propose to eliminate this confinement by spreading around the flip-flops on the interconnect surrounding the Logic Block array area. The benefits of this move are manifold. First, it allows us greater flexibility as we can use these distributed flip-flops to accommodate more diverse designs. The flip-flops can serve well in designs with large fanouts. Current research focusses on testing the feasibility of such a setup as well as testing this setup on different benchmark circuits. We are researching a variety of architectural issues as well:
a) A clock tree design methodology
b) RAM cell programming methodology (we currently program these cells through a scan path in a bit-serial manner)
c) FPGA input-output mechanism (including the number of I/O pins required).
d) Decomposition issues and automation of place and route.
e) Extension to sequential designs.

7 Conclusions

We have proposed a novel FPGA architecture that combines the high performance of ASICs and the flexibility afforded by reconfigurable logic. Our architecture achieves high throughput in the neighborhood of 770 MHz on commonly used DSP designs which include an array multiplier, a bit-level systolic FIR filter and a cross correlator. Current and future work is on refining this architecture to make it more flexible and developing algorithms for efficient decomposition of designs, placement and routing.

Acknowledgement: This work was supported in part by MARCO/ DARPA GSRC, NSF through grants CCR 9811528, MIP 95-29077, and MICRO through Xilinx.

8 References

[1]. V. Bertacco et al, "Decision Diagrams and Pass Transistor Logic Synthesis", Proc. of the ACM/IEEE Int'l Workshop on Logic Synthesis, pp. 1-5, May 1997.
[2] V. Betz, J. Rose, A. Marquardt, "Architecture and CAD for Deep-Submicron FPGAs", Kluwer Academic Publishers, 1999.
[3] E. I. Boemo, S. Lopez-Buedo, J. M. Meneses, "Some Experiments About Wave-Pipelining FPGA's", IEEE Trans. On Very Large Scale Integration Systems, Vol.6, No.2, pp. 232-237, June 1998.
[4] G. Borriello, C. Ebeling, S.A. Hauck, S. Burns, "The Triptych FPGA Architecture", IEEE Transactions on VLSI Systems, Vol.3, No.4, Dec. 1995, pp.491-501.
[5] R.E. Bryant, "Graph-based Algorithms for Boolean functions manipulation", IEEE Trans. Computers, Vol. C-35, pp. 677-691, Aug. 1986.
[6] P. Buch et al, "On Synthesizing Pass Transistor Networks", Proc. of the ACM/IEEE Int'l Workshop on Logic Synthesis, pp. 1-8, May 1997.
[7] P. Buch, A. Narayan, A.R. Newton, A. Sangiovanni-Vincentelli, "Logic Synthesis for Large Pass Transistor Circuits", ICCAD '97, November 1997.
[8] W.P. Burleson, M. Ciesielski, F. Klass, W. Liu, "Wave-Pipelining:A Tutorial and Research Survey", IEEE Trans. on VLSI Systems, Vol.6, No.3, Sep.'98.
[9] L.Cotten, "Maximum Rate Pipelined Systems", Proc. AFIPS Spring Joint Comp. Conf., 1969.

[10] B.V. Herzen, “Signal Processing at 250 MHz Using High-Performance FPGAs”, IEEE Trans. on VLSI Systems, Vol. 6. No.2, June ‘98.
[11] S.Y. Kung, “VLSI Array Processors”, Prentice Hall, 1988.
[12] W.K.C.Lam, R.K.Brayton and A.L.Sangiovanni-Vincentelli, “Valid Clock Frequencies and Their Computation in Wave pipelined Circuits”, IEEE Transactions on CAD of IC and Systems, Vol. 15, No.7, July 1996.
[13] A. Mukherjee, R. Sudhakar, M.Marek-Sadowska, S.I. Long, “Wave Steering in YADDs: A Novel Non-iterative Synthesis and Layout Technique”, Proc. Design Automation Conference ‘99, pp 466-471.
[14] A. Mukherjee, M. Marek-Sadowska, S.I. Long,“Wave Pipelining YADDs- A Feasibility Study”, Proc. IEEE Custom Integrated Circuits Conference, ‘99, pp 559-562.
[15] M. Shamanna, K. Cameron, S.R. Whitaker, “Multiple-input, Multiple-output Pass Transistor Logic”, Int’l Journal Electronics, Vol. 79, No. 1, July 1995.
[16] R.Sudhakar, “YADDA: Layout Synthesis using Pass Transistor Logic”, MS Thesis, UCSB, 1998.
[17] K. Taki, “A Survey for Pass-Transistor Logic Technologies”, ASP-DAC, February 1998.
[18] W. Tsu et al, “HSRA: High Speed, Hierarchical Synchronous Reconfigurable Array”, ACM International Symposium on FPGAs, 1999, pp. 125-134.
[19] K.Yano et al, “A 3.8ns CMOS 16x16b Multiplier using Complementary Pass-Transistor Logic”, IEEE J.Solid-State Circuits, Vol.25, no.2, pp.388-395, April, 1990.
[20] The Programmable Logic Data Book, Xilinx Inc. 1999.
[21] URL: http://www.xilinx.com/products/logicore/lcoredes.htm

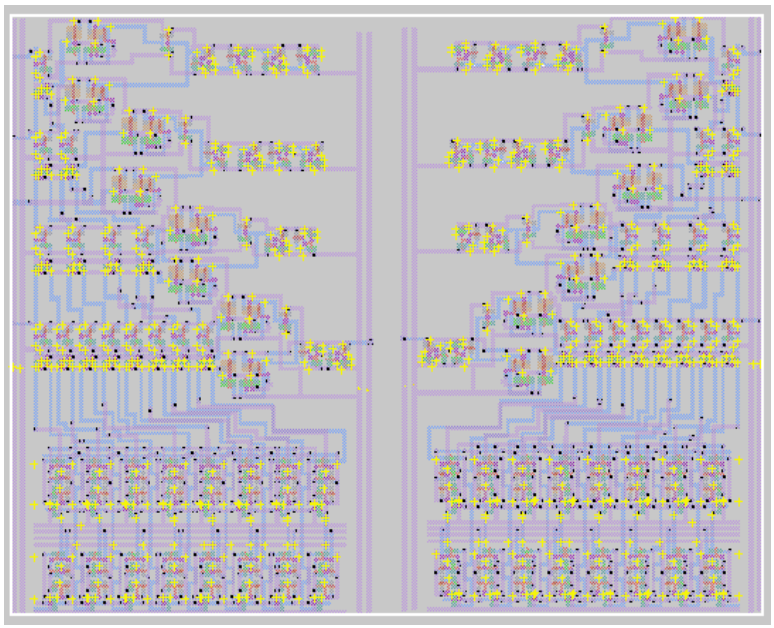


Figure 11: Logic Block in 0.5μ CMOS