[10] S. K. Mathew *et al.*, "Sub-500-ps 64-b ALU's in 0.18 $\mu$m SOI/bulk CMOS: design and scaling trends," *IEEE J. Solid-State Circuits*, vol. 36, no. 11, pp. 1636–1646, Nov. 2001.
[11] S. Naffziger, "A sub-nanosecond 0.5 $\mu$m 64-b adder design," in *Proc. IEEE Int. Solid-State Circuits Conf. Dig. Technical Papers*, Feb. 1996, pp. 362–363.
[12] J. Park *et al.*, "470 ps 64-bit parallel binary adder," in *Proc. Symp. VLSI Circuits Dig. Tech. Papers*, pp. 192–193.
[13] B. Davari, R. H. Dennard, and G. G. Shahidi, "CMOS scaling for high performance and low power—The next ten years," *Proc. IEEE*, vol. 83, no. 4, pp. 595–606, Apr. 1995.
[14] S. K. Mathew *et al.*, "A 4 GHz 130 nm address generation unit with 32-bit sparse-tree adder core," *IEEE J. Solid-State Circuits*, vol. 38, no. 5, pp. 689–695, May 2003.
[15] V. G. Oklobdzija, B. R. Zeydel, H. Q. Dao, S. Mathew, and R. Krishnamurthy, "Energy-delay estimation technique for high-performance microprocessor VLSI adders," in *Proc. 16th Int. Symp. Computer Arithmetic*, Santiago de Compostela, Spain, Jun. 2003, pp. 272–279.
[16] V. G. Oklobdzija, *High-Performance System Design: Circuits and Logic*. Piscataway, NJ: IEEE Press, 1999.
[17] HSPICE User's Guide. [Online] Available: www.synopsys.com

# A Novel FPGA Architecture Supporting Wide, Shallow Memories

Steven W. Oldridge and Steven J. E. Wilton

***Abstract*—This paper investigates an architecture designed to implement wide, shallow memories on a field programmable gate array (FPGA). In the proposed architecture, existing configuration memory normally used to control the connectivity pattern of the FPGA is made user accessible. Typically, not all the switch blocks in an FPGA are used to transport signals. By adding only a modest amount of circuitry, the configuration memory in these unused switch blocks (or unused paths within used switch blocks) can be used to implement wide, shallow buffers and other similar memory structures. The size of FPGA required to implement a benchmark circuit that makes use of the wide, shallow memories, is 20% smaller than a standard memory architecture. In addition, the benchmark circuit is on average 40% faster using the proposed architecture.**

***Index Terms*—Embedded memory, field programmable gate arrays (FPGAs).**

## I. Introduction

In the past ten years, the capacity of field programmable gate arrays (FPGAs) has grown to such an extent that they are now being used to implement entire systems, and represent an alternative to system-on-chip (SoC) and application specific integrated circuit (ASIC) development. Logic and memory resources on the FPGA have reached levels such that they are capable of providing a complete, one-chip solution. Input/output (I/O) capabilities have also followed this trend, most notably with the introduction of high-speed I/O (such as RapidIO, POS-PHY Level 4, or UTOPIA IV) [1], [2] that allow gigahertz range signals to access an FPGA running at a lower frequency. High-speed I/O has also allowed users to implement high bandwidth circuits on an FPGA. The high-speed interface is accomplished by time demultiplexing an incoming signal to create a wide set of signals on the FPGA. In many applications, these signals must be buffered in wide, shallow memories as they are processed throughout the FPGA. These memories may be over 100 bits in width, but only a handful (8–32) words deep. As I/O bandwidth on FPGAs increases, and more circuits begin to take advantage of these high speed I/Os, the need for efficient wide data blocks, including wide, shallow memories, will grow.

Traditionally, vendors support the memory requirements of user circuits by embedding large random access memory (RAM) blocks [1], [3]–[6]. The RAM blocks provide a very dense method of implementing storage, but they are not well suited to wide, shallow memories. The aspect-ratio of most block memories allow a maximum width of 16–32 bits, meaning wide memories must be constructed by cascading several blocks. In addition, the depth of the memories at that ratio are deeper than those needed by simple buffering applications.

Another method for supporting on-chip memory is employed by Xilinx and Lattice Semiconductor (in the ORCA product line, formerly produced by Lucent) [4], [5]. In their devices, the 16-bit lookup-tables within each logic element can be configured as a RAM. Since each logic block can be configured as RAM individually, the memories can be placed close to the attached logic, and several logic blocks, each configured as a RAM, can be combined to implement wider structures. Unfortunately, the area overhead required to glue these smaller memories together into a large memory is significant. As an example, a 32 $\times$ 128 bit memory would require 256 look-up tables (LUTs) to implement the storage, and another 128 LUTs to implement the data output multiplexors.

In this paper, we present an alternate implementation of on-chip memory, designed specifically to support wide buffering applications. In our architecture, we reuse the configuration memory within each switch block, providing the user with access to this memory through additional circuitry. Normally, this memory is used to store the connection patterns required to implement the user circuit. Typically, however, not all the switches in all switch blocks in an FPGA are used to transport signals. By adding only a modest amount of circuitry, the configuration memory in these unused switch blocks can be used to implement wide, shallow buffers and other similar memory structures.

Utilizing unused switch blocks in this way has a number of advantages. The amount of storage within each switch block is significant; an FPGA with 128 tracks/channel contains 1088 configuration bits within each switch block. As we will show, the structure of the switch blocks leads to a natural implementation of wide, shallow memories. In addition, since the switch blocks are evenly distributed across the FPGA, user memories can be instantiated close to the logic that uses them, reducing routing delay. In order to realize these advantages, however, it is important that the additional circuitry does not slow the switch block unacceptably, or result in a significant area overhead.

The use of switch block memory as user storage has been described in [20]. That architecture provided 16 $\times$ 8 bit memories within each switch block. A significant difference between this and our architecture is that we provide 128-bit wide memories rather than 8-bit wide memories. This very wide data width is one of the key features of our architecture, and is important when implementing the wide shallow memories considered in this paper.

## II. Baseline Programmable Connection

In this paper, we focus on island-style FPGAs [11]. Each horizontal and vertical channel consists of 128 parallel tracks, each track spanning four clusters (although the concept can be applied to FPGAs of any channel width and segment length, including FPGAs containing more than one segment length). We have concentrated on FPGAs with length-four segments, to be consistent with the architectures in [11]. At the intersection of each horizontal and vertical channel is a Wilton

Manuscript received May 22, 2002; revised December 30, 2004.

The authors are with the University of British Columbia, Vancouver, BC V6T 1Z4, Canada (e-mail: steveo@ece.ubc.ca; stevew@ece.ubc.ca).

Digital Object Identifier 10.1109/TVLSI.2005.848817

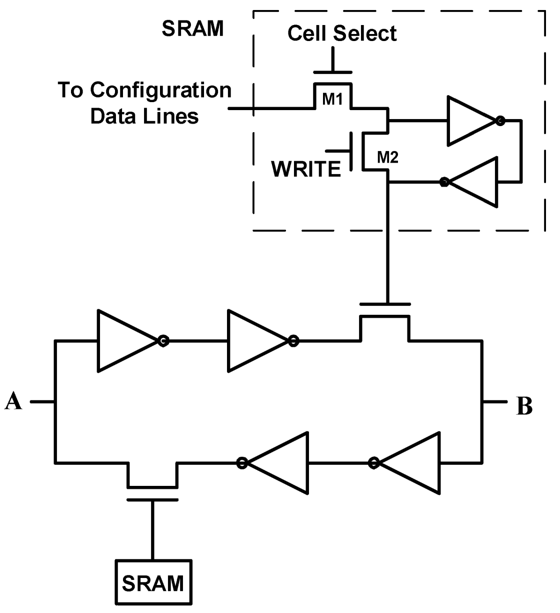

Fig. 1. Baseline bidirectional programmable connection.

switch block [12]. We assume that the tracks within each channel are staggered so that one quarter of the tracks start/terminate at each switch block. As shown in [11], the tracks that terminate at a switch block can be connected to one track in each of the other three incident channels, while the tracks that pass through a switch block can be connected to two tracks in each of the two perpendicular incident channels.

Each programmable connection within the switch block is a bidirectional repowering switch, containing buffers and two configuration bits, as shown in Fig. 1.

## III. Enhanced Architecture

### A. Memory Organization and Access Modes

In the enhanced architecture, each switch block can be optionally used as a memory with eight words of up to 124 bits each. Rather than adding new memory bits to the switch block (as in [13]), our approach is to provide circuitry to allow the user to write and read the configuration memory corresponding to all diagonal connections within the switch block. Assuming 128 tracks per channel, there are 512 diagonal programmable connections within each switch block. We group these programmable connections into 128 groups of four connections each; each group implements a single bit position of the memory. Since each connection contains two configuration bits, each group contains 8 bits of storage. By providing appropriate select circuitry, any one of these 8 bits can be read or written to; thus, the switch block acts as a 8 $\times$ 128 bit memory. As described in [7], to allow for address and control lines, our architecture can only use up to 124 of the 128 groups, thus, the maximum memory size we can support in a single switch block is 8 $\times$ 124.

### B. Enhanced Programmable Bidirectional Connection

The key to our architecture is an enhanced programmable connection (EPC) element. As Fig. 2 shows, the EPC provides access to its two configuration memory bits by linking each configuration memory cell to a horizontal or vertical track through pass transistor circuitry. Each of the two configuration bits in the EPC are connected to an intermediate staging point through pass transistors M3 and M4. These transistors are controlled by lines CA and CB, the values of which are generated by the address control circuitry, as described below. All four connections within a group share the same staging point. From this intermediate

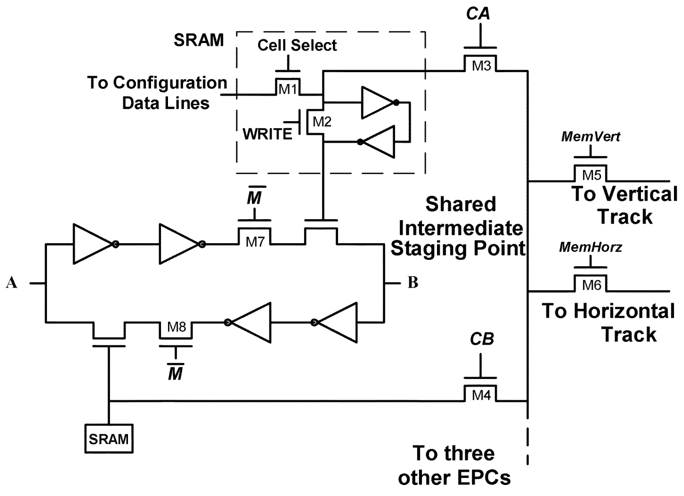

Fig. 2. Enhanced prgrammable connection (EPC).

staging point, pass transistors M5 and M6 connect to a vertical and a horizontal track respectively, with these transistors controlled by the MemVert or MemHorz (discussed below) which allow access to the memory from either the vertical or horizontal direction. The WRITE signal in Fig. 2 is used to indicate whether the memory cell should be written or read; it is sourced by the user circuit residing on the FPGA.

Additional pass transistors M7 and M8 are used to isolate the two sides of the programmable connection when the switch block is used as a memory. When not used as a memory, transistors M7 and M8 are closed, allowing the circuit to operate as a normal programmable connection.

### C. Memory Addressing and Control

The eight-word memory requires three address bits, as well as a read/write control signal. Rather than tying these inputs to four specific incident tracks, the address and control inputs are chosen from the entire set of incident tracks in either the horizontal or vertical channel, as shown in Fig. 3. Each address and control line can be selected from one quarter of the incident tracks; those tracks chosen to supply address and control information are, of course, not used for data. The input lines to the control circuitry are shared between the horizontal and vertical incident tracks using four two input multiplexors (only shown for line A2 in Fig. 3). The address and control signals are fed to a 3-to-8 decoder; the eight outputs of the decoder are used to drive the CA and CB lines of one of the four programmable connections in each group.

In addition to this circuitry, two configuration bits are needed per switch block. The first configuration bit is used to indicate whether the switch block is acting as a memory or as a routing switch. This bit is connected through an inverter to M7 and M8 of the EPCs. The second configuration bit indicates whether the data inputs are connected to the horizontal or vertical channel (the data outputs are connected to the other), and is used both to control the previously mentioned multiplexors, and with the read/write signal to produce the MemVert and MemHorz signals. These signals control the connection between the intermediate staging point of Fig. 2 and the channels incident to the switch block. If the switch block is used as a memory, the intermediate staging point is connected to either the horizontal or vertical channels (depending on whether a read or write is occurring) but never both simultaneously. When not used as a memory, the intermediate switching point is isolated.

Finally, transistor M2, which directly enables the writing of the memory cell, must share its control with both the configuration circuitry and the read/write line. This allows the bit to be written during configuration as well as when it is written to by the user.

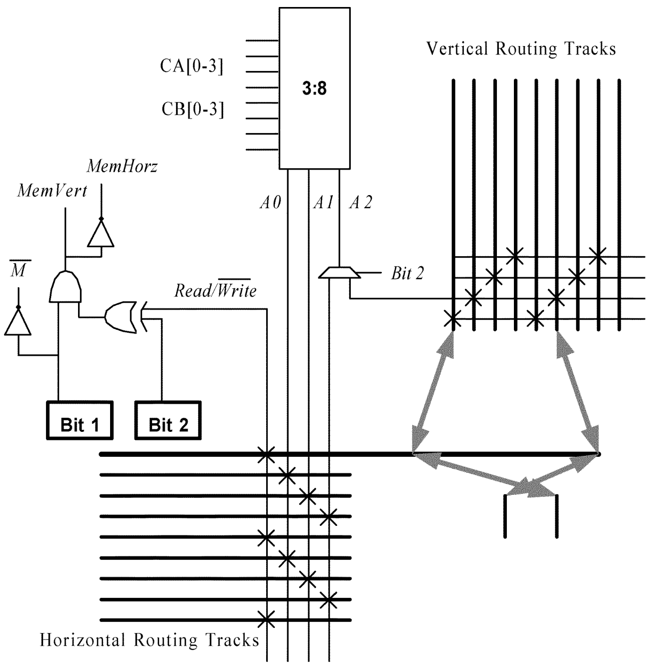

Fig. 3. Memory addressing and control circuit.

## IV. Speed and Area Overhead

In this section, we describe the area and speed overhead inherent in our new architecture.

Compared to the baseline architecture, the routing path through a switch in our architecture is 11.6% slower than that in the baseline architecture. This slowdown is mainly due to the parasitic capacitances of the added transistors. The corresponding memory access time was 2.5 ns (based on HSPICE measurements), not including routing into and out of the memory. This corresponds to an operating frequency of 400 MHz, faster than the 4-K and MegaRAM blocks from the Stratix Architecture [3].

The average speed degradation, including both logic and routing delay, was found to be 5.2%. Measurements were made by comparing the critical paths of a set of sixteen circuits, ranging in size from 641 4-LUTs to 3539 4-LUTs, after place and route, using the methodology in [11]. A 0.18-$\mu$m CMOS technology was assumed throughout.

In order to quantify the overhead, we have used a detailed area model. Assuming typical FPGA parameters, have estimated a 52% increase in the size of a single switch box, and a 36% overhead in an FPGA containing these switches.

## V. Implement Storage in the Enhanced Architecture

In this section, we evaluate the ability of our architecture to implement user circuits that contain wide, shallow memories. In order to evaluate the proposed architecture, a benchmark suite with circuits that make use of wide, shallow memories is needed. Unfortunately we do not have access to such circuits, forcing us to create our own benchmark. Time and technology constraints meant that rather than a suite of representative circuits, only a single benchmark circuit, a buffered 2 × 2 crossbar switch [19], could be created. This circuit allows an initial evaluation of the switch block memory architecture, but does not allow for a robust evaluation of the average effect on real user circuits. The creation of a more complete benchmark suite to fully evaluate the proposed architecture is left as future work.

The circuit chosen to test the switch block memory architecture is a crossbar switch [16], [17]. Signals enter the circuit at one of two inputs, and are sent to one of two outputs based on an address tag that is a part of the incoming data. Because data destined for the same output could arrive at both of the inputs simultaneously, the data is buffered in order to prevent data loss. The crossbar is fully buffered, meaning that every input-output combination has its own buffer. For the 2 × 2 crossbar, this means that four memories are required to buffer the incoming data. More generally, for an $N \times M$ crossbar, $N * M$ memories are required.

Our intent in using this circuit is to stress the limits of the proposed architecture, so a very wide input data width of 120 bits has been chosen.

To measure the improvement in density obtained by the proposed architecture, the presynthesized test circuit was mapped onto FPGAs with each of the three different memory types: LUT-based, memory block, and switch block. In order to meaningfully compare the three architectures a common measure of area is required. The next section details the methodology used in measuring and comparing these architectures.

TABLE I
Number of Transistors for Each Component

| | Minimum Width Transistors |
|---|---|
| 10 x 4 LUT Logic Block | 7830 |
| Connection Block | 1840 |
| Switch Block | 187 / track |
| Switch Block (EPC) | 255 / track |

### A. Area Methodology

The area of an FPGA can be broken into four components: logic block area, connection block area, switch block area, and memory area. Table I shows the number of minimum-width transistors for each component that makes up a tile's area, assuming the architecture of Figs. 2 and 3. These values were calculated using the area model from [11]. The area of switch blocks are expressed as the number of minimum-width transistors per track, because the required channel width of the FPGA varies significantly between memory architectures.

Expressing the area of an FPGA as the total number of minimum width transistors required to implement the test circuit is useful; however, the size of the FPGA (in columns and rows) presents a more intuitive metric. In order to be able to accurately compare architectures based on FPGA size, the tile for that particular architecture must be normalized to a base tile area. For the purposes of this paper, the base tile has a channel width of 128 tracks, and uses the architecture described in Fig. 1. Using Table I, the number of minimum-width transistors in the base tile is 35 446.

### B. LUT-Based Memory

In the LUT-based memory architecture, no additional area is needed to account for the memory, since all memory is packed into logic blocks. No block memory resources were included within the FPGA, forcing the four user memories to be mapped directly to the 16-bit LUTs. The smallest size FPGA that can implement this circuit consists of a 49 × 49 set of logic and routing tiles for the FPGA. Because user logic and memory on this FPGA are less densely packed than the FPGA architectures containing block or switch block memories, the routing resources required are not as significant; the FPGA only requires 25 tracks per channel as opposed to 128 tracks per channel required by the other architectures. This means that the LUT-based tiles area is only 14 345 minimum-width transistors. Normalizing leaves us with a 19.8 × 19.8 base tile-sized FPGA.

### C. Block Memory

In the case of block memories, the test circuit was mapped to an architecture with 4-K block memories similar to those found in Altera's

TABLE II
AREA RESULTS FOR THE BENCHMARK CIRCUIT

| | Number minimum-width transistors | Minimum Size (in base tiles) | Ratio |
|---|---|---|---|
| LUT-Based | 34.4M | 19.6 x 19.6 | 11.4 |
| Block Memory | 3.69M | 10.2 x 10.2 | 1.22 |
| Switch Block | 3.02M | 9.3 x 9.3 | 1 |

Stratix Architecture [3]. The size of a configurable 4-K memory block and the associated routing is estimated to be 2.5 times the size of a tile containing a logic block, or 88 615 minimum-width transistors. The nonmemory portions of the circuit can be implemented on an 8 $\times$ 8 base tile-sized FPGA with a channel width of 128. In order to find the minimum-sized FPGA required to implement the test circuit we need to determine how many memory blocks are required to implement the memory portion. The aspect ratio of the memory blocks are chosen to meet the depth requirements of the user memory. With an aspect ratio of 128 $\times$ 32 for the 4-K blocks, the minimum allowable depth, four memories are needed to implement each of the 120 bit wide memories found in the test circuit. Converting the area of the 16 memories required (1.42 M minimum-width transistors) into base tile sizes and adding them to the 8 $\times$ 8 FPGA required to implement the nonmemory portion of the circuit results in a square 10.2 $\times$ 10.2 base tile-sized FPGA.

### D. Switch Block Memory

For the switch block memory architecture, we used custom place and route tools to implement the circuit on an 8 $\times$ 8 FPGA architecture with a channel width of 128 [19]. The memories are mapped to four switch blocks and thus take up no extra tiles. However, the extra circuitry to enable switch block memories adds transistors to the switch block component of the tile. The area of a switch block memory tile is 47 191 minimum-width transistors. Translated into logic and routing tiles of the base size, the size of the FPGA required is 9.3 $\times$ 9.3 base tiles.

### E. Comparing Memory Architectures

Table II lists the FPGA size, as well as the total number of transistors, required to implement our benchmark circuit for the three memory architectures, not accounting for the routing into and out of each memory structure. Even though the switch block memory architecture has significant area overhead, it still provides a denser implementation of wide, shallow memories than the other memory architectures. Even with only four switch block memories in use out of a possible 64, the architecture is still 22% smaller than a block memory-based solution, and 11.4 times smaller than a LUT-based solution. In addition, adding more memories to the FPGA does not increase the area overhead; a circuit with more memories (and the same amount of logic) could be implemented on an FPGA of the same size, increasing the density even more.

These numbers do not take into account the routing required into and out of each memory. Adding more memories may increase the routing congestion around these memories; this would reduce the improvements shown.

## VI. Timing Results

Using only a single benchmark circuit, it is hard to accurately measure the overall timing benefits of the proposed architecture on the different types of circuits that could potentially take advantage of switch block memories. Still, conclusions about the impact that the proposed architecture has on circuits that are similar to the benchmark can be examined. The nondeterministic nature of the placement and routing algorithms can cause significant differences in the resulting critical paths, potentially making conclusions drawn from a single placement and routing run error prone. To overcome this, every unique place and route attempt is iterated ten times, each time with a different initial random seed. The average critical path of the successful iterations, is used when making a comparison between attempts.

### A. LUT-Based Memory

The LUT-based implementation of the circuit was placed and routed on appropriately-sized FPGA with 25 tracks per channel (other channel widths were attempted, and the results were similar) using timing-driven place and route tools. An average critical path of 10.9 ns was obtained.

### B. Block Memory

Unfortunately, a version of VPR capable of the placement and routing of block memories has not been developed. This prevents us from properly gathering timing results regarding block memory-based architectures and makes a direct comparison of block memory-based architectures to switch block memory-based architectures impossible. However, the access time of switch block memories allows them to operate at speeds of up to 400 MHz. Comparatively, the Stratix Architecture's memory, which was constructed using a 0.13-$\mu$m process, operates at 312 MHz for the 4-K blocks and 300 MHz for the MegaRAM blocks [3]. This suggests that the benchmark circuit could be up to 30% faster on an architecture containing switch block memories, depending on the input and output path.

### C. Switch Block Memory

Using our new architecture, a critical path delay of 7.8 ns for a channel width of 140 was measured. This is 40% faster than the circuit implemented using LUT-based memories.

## VII. Conclusion

Logic and memory resources that efficiently implement specific functionality commonly used in system-sized designs, such as multipliers or large memories, are already a part of modern FPGAs. As high-speed I/O becomes increasingly faster and FPGAs are expected to meet higher bandwidth requirements, the width of datapaths in certain classes of user circuits are likely to increase. To meet the needs of these circuits, logic and memory resources that efficiently implement wide datapaths will be necessary additions to future FPGA architectures. In this paper, we have presented an architecture that uses unused configuration bits within an FPGA as user storage.

## References

[1] Xilinx, Inc., "Datasheet: Virtex II pro platform FPGAs: Functional description (DS083-2)", Jan. 2002.
[2] Altera Corporation, "Application note: using high-speed differential I/O in stratix devices", Mar. 2002.
[3] Altera Corporation, "Datasheet: stratix programmable logic family device", Feb. 2002.
[4] Xilinx, Inc., "Datasheet: Virtex-E 1.8 V field programmable gate arrays", Sep. 2000.
[5] Lattice Semiconductor, "Datasheet: ORCA series 4 field-programmable gate arrays", Aug. 2000.
[6] Altera, "Datasheet: APEX 20 K programmable logic device family", Mar. 2000.
[7] S. Oldridge and S. Wilton, "A novel FPGA architecture supporting wide shallow memories," in *Proc. IEEE Custom Integrated Circuits Conf.*, May 2001, pp. 75–78.
[8] V. Betz and J. Rose, "VPR: a new packing, placement, and routing tool for FPGA research," in *Proc. Int. Workshop Field-Programmable Logic and Applications*, 1997, pp. 213–222.
[9] *HSPICE User's Manual*, Meta-Software, Feb. 1996.

[10] Xilinx, Inc., "The programmable logic data book", 1994.

[11] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*. Norwell, MA: Kluwer, Feb. 1999.

[12] S. J. E. Wilton, "Architecture and algorithms for field programmable gate arrays with embedded memory," Ph.D. dissertation, Dept. Elect. Comp. Eng., Univ. Toronto, Toronto, ON, Canada, 1997.

[13] W. Tsu, K. Macy, and A. Joshi *et al.*, "HSRA: high-speed, hierarchical synchronous reconfigurable array," in *Proc. Int. Symp. FPGAs*, Feb. 1999, pp. 125–134.

[14] S. Kirkpatrick, C. Gelatt, and M. Vecchi, "Optimization by simulated annealing," *Science*, pp. 671–680, May 1983.

[15] E. Dijkstra, "A note on two problems in connection with graphs," *Numer. Math.*, vol. 1, pp. 269–271, 1959.

[16] H. Ahmadi and W. Denzel, "A survey of modern high-performance switching techniques," *IEEE J. Sel. Areas Commun.*, vol. 7, no. 7, pp. 1091–1103, Sep. 1989.

[17] H. Eggers, P. Lysaght, H. Dick, and G. McGregor, "Fast reconfigurable crossbar switching in FPGAs," in *Proc. 6th Int. Workshop on Field-Programmable Logic and Applications*, 1996, pp. 297–306.

[18] S. J. E. Wilton and R. Saleh, "Progammable logic IP cores in SoC design: opportunities and challenges," in *Proc. IEEE Custom Integrated Circuits Conf.*, San Diego, CA, May 2001, pp. 63–66.

[19] S. Oldridge, "A novel FPGA architecture supporting wide, shallow memories," M.A.Sc. thesis, Dept. Elect. Comp. Eng, Univ. British Columbia, Vancouver, BC, Canada, Apr. 2002.

[20] A. Marshall, T. Sansfield, I. Kostamov, J. Vuillemin, and B. Hutchings, "A reconfigurable arithmetic array for multimedia applications," in *Proc. 1999 ACM/SIGDA 7th Int. Symp. Field Programmable Gate Arrays*, Feb. 1999, pp. 135–143.

# Simultaneous $V_t$ Selection and Assignment for Leakage Optimization

Vishal Khandelwal, Azadeh Davoodi, and Ankur Srivastava

***Abstract*—This paper presents a novel approach for leakage optimization through simultaneous $V_t$ selection and assignment. $V_t$ selection implies deciding the right value for $V_t$ and assignment implies deciding which gates should be assigned a particular threshold voltage. We also include the effect of variability in threshold voltage on delay and leakage due to fabrication process variations in our formulations and present a scheme that lets the designer control the leakage and delay variability in his design. The proposed algorithm is a general mathematical formulation that has been shown to trivially extend to multiple threshold voltages.**

***Index Terms*—Low power, multiple-threshold voltage, variability.**

## I. Introduction

Technology scaling has lead to exponential increase in leakage currents. Leakage has emerged as a potent problem and is a significant fraction of the total power consumption in chips. Recently, a lot of work has been presented in leakage optimization techniques, the most popular being the use of multiple threshold voltages on the same chip. The MTCMOS scheme [4] and using high threshold gates on noncritical paths [8], [6] being a few of them. No existing work presents a general framework for using multiple threshold voltages in the design and focus mainly on dualthreshold schemes [3].

Manuscript received May 27, 2003; revised December 6, 2003 and September 20, 2004.

The authors are with the University of Maryland, College Park, MD 20742 USA (e-mail: vishalk@eng.umd.edu; azade@eng.umd.edu; ankurs@eng.umd.edu).

Digital Object Identifier 10.1109/TVLSI.2005.844304

In this work, we present a general framework for assigning multiple threshold voltages to gates in the circuit where the number of threshold voltages, their values and their gate assignments can be decided by the designer. The scheme uses the trade-off between delay and leakage to simultaneously select threshold voltage values and assign them to gates. Additionally, we explore the effect of fabrication variability on threshold voltage which in turn imposes a variability on delay and leakage of the circuit. We present a variability driven threshold selection and assignment scheme that lets the designer control the variability in leakage and delay in his design while optimizing the nominal leakage under nominal delay constraints.

The rest of the paper is organized as follows. Section III contains our complete mathematical and algorithmic formulation for simultaneous threshold selection and assignment. Section IV presents our variability driven threshold selection and assignment formulation. Section V contains the experimental results and Section VI contains the conclusion.

## II. Simultaneous Selection and Assignment

The leakage current of a MOS transistor according to the BSIM model [1] can be approximated as follows:

$$I_{\text{leak}} = Ae^{q(V_{\text{gs}}-V_t)/nkT}(1-e^{-qV_{\text{ds}}/kT}) \tag{1}$$

where $A = \mu o C_{\text{ox}}(W_{\text{eff}}/L_{\text{eff}})(kT/q)^2 e^{1.8}$ and $C_{\text{ox}}$ is the gate oxide capacitance per unit area and $V_t$ is the threshold voltage. It can be seen that leakage current is exponentially dependent on threshold. On the other hand, delay of a MOS transistor follows the following approximate relation w.r.t. $V_t$ [8], [1], [7]:

$$t_d = 2C_{\text{load}}V_{\text{dd}}/(\beta)(V_{\text{dd}}-V_t)^{\alpha} \tag{2}$$

where $\alpha$ is around 1.3 for short channel and 2 for long channel devices.

In this work we address the following problems:

1) threshold selection: what should be the values of the threshold voltages;
2) threshold assignment: which gates should be assigned which selected threshold voltage.

### A. Polynomial Time Linear Programming Approach

The dependence of delay and leakage on threshold follows a convex curve. This convexity property can be exploited to generate a linear program under the piecewise-linear assumption. A similar approach was presented in [5] for the slack distribution in the placement problem.

For every gate in the circuit, we have a delay versus threshold voltage curve for each input pin–gate output [from (2)] and a corresponding leakage versus threshold curve as well. We assume each of these curves to be approximated by $m$ lines in a piecewise linear fashion. Let $z_g$ denote the leakage of a gate, let $a_g$ be the arrival time of a gate and $t_{i,g}$ denote the delay from pin $i$ to the output of $g$. Let the $i$th line representing the delay of $j$th pin for gate $g$ have the following parameters $y_i^{g,d,j}$, $c_i^{g,d,j}$. Here $y_i^{g,d,j}$ signifies the slope of the line and $c_i^{g,d,j}$ signifies the constant. Let the $i$th line representing the leakage curve for gate $g$ have the following parameters $y_i^{g,l}$, $c_i^{g,l}$. Let the arrival time at a primary input gate $g$ be given as $A_g$ and the required time at the primary output gate $g$ be given by $T_{\text{nom},g}$

$$\text{Minimize} \sum_{\forall g} z_g \tag{3}$$

$$z_g \geq y_1^{g,l}V_t(g) + c_1^{g,l} \tag{4}$$

$$-- \tag{5}$$

$$z_g \geq y_m^{g,l}V_t(g) + c_m^{g,l} \tag{6}$$

$$t_{i,g} \geq y_1^{g,d,i}V_t(g) + c_1^{g,d,i} \quad \forall \text{ inp pin } i \text{ of } g \tag{7}$$