SHOULD FPGAS ABANDON THE PASS-GATE?

Charles Chiasson and Vaughn Betz

Department of Electrical and Computer Engineering
University of Toronto, Toronto, ON, Canada
{charlesc,vaughn}@eecg.utoronto.ca

ABSTRACT

Pass-transistors have been the key building block for field-programmable gate array (FPGA) circuitry for many years due to the very small switch they enable. However, pass-transistor performance and reliability have been degrading with technology scaling. Transmission gates are an alternative to pass-transistors; while larger, they are more robust. We develop a new FPGA circuit optimization flow and use it to investigate the area, delay and power impact of building FPGAs out of transmission gates instead of pass-transistors in a 22nm process. Our results show that transmission gate FPGAs are 15% larger than pass-transistor FPGAs but are 10-25% faster depending on the allowable level of "gate boosting". Without gate boosting, transmission gate FPGAs are the better option with 14% lower area-delay product. If 200mV of gate boosting is possible however, pass-transistor FPGAs remain the slightly better choice with a 2% better area-delay product. We also show that transmission gates with a separate power supply for their gate terminal enable a low-voltage FPGA with 50% less power and good delay.

1. INTRODUCTION

The reconfigurability of field-programmable gate arrays (FPGAs) is achieved through a combination of look-up tables (LUTs) and multiplexers (MUXes) whose construction relies heavily on the use of transistor-based switches. Commercial FPGAs and almost all academic FPGA studies use NMOS pass-transistors as the basic switching element (see Figure 3a) because each switch requires only one transistor, minimizing area. However, NMOS pass-transistors have an important disadvantage: they are incapable of passing a full logic-high voltage. That is, their output voltage saturates at approximately $V_G - V_{Th}$ where V_G is the gate voltage and V_{Th} is the threshold voltage of the transistor. Static power dissipation in downstream inverters caused by this reduced voltage swing has long been a cause for concern for pass-transistor based circuits [1]. To mitigate this problem, gate boosting (applying a voltage larger than the supply voltage (V_{DD}) on the pass-transistor gate) and PMOS level-restorers have been used to help pull pass-transistor output voltages up to V_{DD}.

As technology scales, V_{DD} drops more rapidly than V_{Th} to control power; this results in an increasingly degraded pass-transistor output voltage. For a 22nm process with a V_{DD} of 0.8V for example, the output of a non-gate boosted pass-transistor switches only between 0V and 0.55V. In addition, the waveform slew rate rising above 0.45V is very slow. Consequently, the inverter sensing this signal (whose input can remain near $V_{DD}/2$ for some time) can experience a high short-circuit current and a slow switching speed. Furthermore, recent work has shown that pass-transistor based FPGAs are very sensitive to aging induced by *positive bias temperature instability* which has become larger with the new high-k gate dielectrics [2, 3].

To increase the pass-transistor output voltage, one can apply larger amounts of gate boosting, but this poses a reliability risk as larger V_{GS} values accelerate device aging. Furthermore, the latest high-k gate processes do not offer a "mid-oxide" thickness transistor; such transistors were available in 90nm through 40nm conventional oxide processes to give a reduced gate leakage transistor option to designers [4]. These mid-oxide thickness transistors were excellent pass-gates as their thicker oxide allowed a high level of gate boosting without compromising reliability. With PMOS level-restorers the issue is one of robustness. A V_{Th} that is a larger fraction of V_{DD} means it takes longer for level-restorers to turn on (which increases short-circuit currents) or, in the extreme case, they might not turn on at all. Reliability concerns, a higher susceptibility to device aging, performance degradation and increasing short-circuit power dissipation make the pass-transistor an increasingly less desirable switch.

Instead of pass-transistors, FPGAs could use CMOS transmission gates as the basic switching element [5, 6] (see Figure 3b). While larger, transmission gates are capable of passing a full rail-to-rail voltage swing, making them more robust than pass-transistors at low V_{DD}. Hence, it is unclear where in the area-delay optimization space a fully transmission gate based FPGA would fall in relation to a fully pass-transistor based FPGA. In this work, we locate them both in advanced process technology (with PTM 22nm HP models [7]) by designing each type of FPGA from scratch, complete with architectural design, circuit design and detailed transistor sizing. We also experiment with gate boosting both switch types.

To ensure our comparison is accurate, we select state of

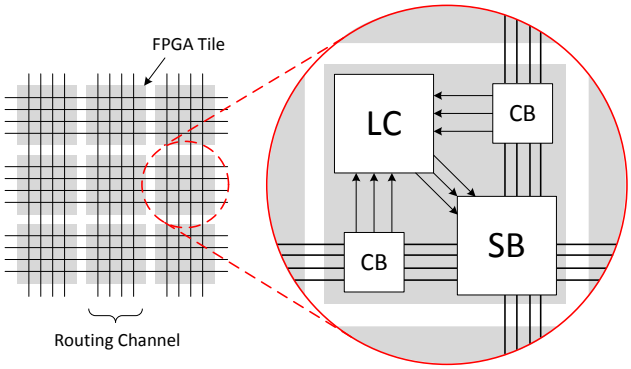


Fig. 1: Tile-based FPGA.

the art topologies for the various subcircuits that make up the FPGA (LUTs, MUXes, etc.) which we then optimize for minimal area-delay product using a custom transistor sizing tool that employs new, more accurate, area and wire load modeling. Our contributions include:

- A comparison between pass-transistor and transmission gate FPGAs for various levels of gate boosting.
- A new methodology for FPGA circuit design including more accurate area and wire load models.
- Detailed circuit designs and VPR architecture files[1] that reflect the complexity of current commercial FPGAs; interestingly, these lead to tile area and critical path delay breakdowns that differ from oft-quoted maxims.

The remainder of this paper is organized as follows. Section 2 describes the chosen FPGA architecture. Section 3 gives details on our circuit designs. Our methodology is presented in Section 4 and results are given in Section 5. Section 6 concludes the paper.

2. FPGA ARCHITECTURE

An FPGA consists of an array of tiles that can each implement a small amount of logic and routing. Horizontal and vertical routing channels run on top of the tiles and allow them to be stitched together to perform larger functions. Figure 1 illustrates FPGA tile architecture at a high-level. A logic cluster (LC) supplies the tile's logic functionality. Connection blocks (CBs) provide connectivity between LC inputs and routing channels. A switch block (SB) connects LC outputs to routing channels and provides connectivity between wires within the routing channels. One replicates this basic tile to obtain a complete FPGA. Although Figure 1 shows logic and switching functions as distinct sub-blocks, we assume an interleaved layout in our area, loading and delay estimates.

Figure 2 shows our logic architecture. Each logic cluster contains $N = 10$ basic logic elements (BLEs) and each BLE contains a 6-input LUT ($K = 6$) as these parameters have been shown to produce FPGAs with good area-delay

[1]Available for download at: `http://www.eecg.utoronto.ca/~vaughn/downloads/FPGA_architecture.html`.

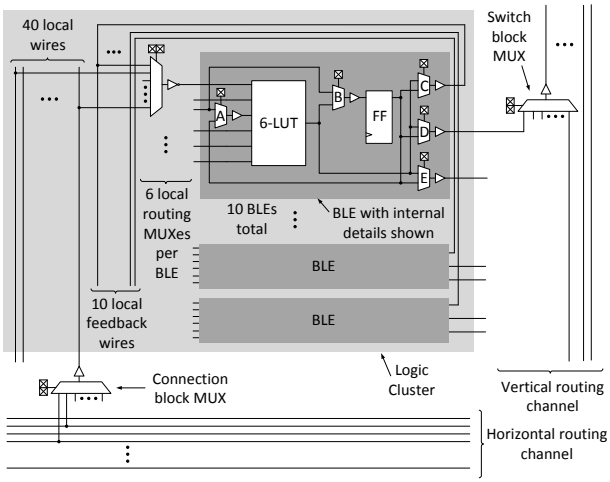


Fig. 2: Logic cluster architecture.

product [8] and are close to the values used in current commercial FPGAs (Virtex 7: K=6, N=8 and Stratix V: K=6, N=10). The BLEs of modern commercial FPGAs [9, 10] contain many more features than the commonly used academic BLE which consists of a K-input LUT and a FF with a very limited ability to use both LUT and FF together [1]. To design a more realistic FPGA where the LUT and FF can be used in concert in many more ways, we add additional 2-input MUXes to our design which can potentially improve density and speed. These MUXes are labeled A to E in Figure 2 and are similar to those used in Stratix [11].

Local routing MUXes select the BLE inputs from the cluster's local interconnect. These MUXes are sparsely populated (at 50%) as this was shown to be a good choice in [12]. The local interconnect consists of 10 local feedback wires from the BLEs and 40 cluster input wires. The number of cluster inputs is set to 40 based on the relationship $I = K(N+1)/2$ given in [8] plus a few extra cluster inputs required by the sparsely populated local interconnect [12].

The wires in the routing channels are directional, single-driver wires which means they can only be driven from one end [13]. All routing wires span 4 tiles ($L = 4$). To obtain a practical tile layout, the number of wires in a routing channel should be a multiple of $2L$ [13]. The routing channel width is set to $W = 320$ by adding 30% more routing tracks to the minimum channel width required to route our biggest benchmark circuit. As is common in FPGA research, each incoming wire can connect to 3 routing multiplexer inputs in a switch block ($Fs = 3$).

Cluster input flexibility, Fc_{in}, is set to $0.2W$ based on results from [1, 12] for similar N and K. Since the architecture described thus far is fairly different from prior work in terms of logic cluster outputs (e.g. two outputs per BLE and single-driver routing wires), Fc_{out} is determined experimentally. In Section 5.1, we show that for this architecture, an $Fc_{out} = 0.025W$ produces an FPGA with the best area-delay product.

Table 1: FPGA subcircuit count per tile.

Subcircuit	Size	Count
Local routing MUXes	25:1	60
Connection block MUXes	64:1	40
Switch block MUXes	10:1	160
BLEs	–	10

We use a *two-sided* architecture which means LC inputs and outputs can only access the two routing channels (one vertical and one horizontal) which run over top of the tile, as shown in Figure 1. Four-sided architectures (capable of accessing 2 vertical and 2 horizontal channels) have often been assumed in prior work but are less realistic since such architectures are difficult to lay out. VPR experiments show that using the more realistic two-sided architecture results in a 3-4% critical path delay increase and 8-9% routed wire length increase over a four-sided architecture. Table 1 details the subcircuits per tile for this architecture.

3. CIRCUIT DESIGN

The FPGA architecture described in the previous section consists entirely of MUXes, LUTs and FFs. Our topology choices for each are detailed below.

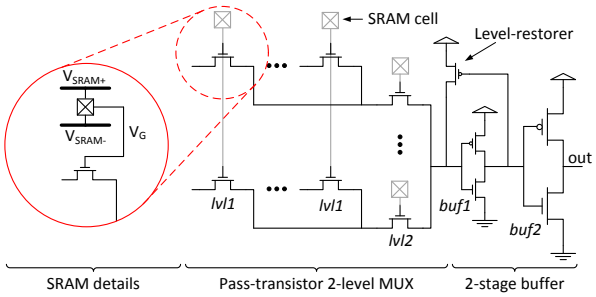
3.1. Multiplexers

A multiplexer can be implemented in several different topologies, each of which possesses a different area-delay tradeoff [6]. All our MUXes are implemented as two-level multiplexers because they have been shown to give the best area-delay product [14] and are used in commercial architectures [15]. An exception to this are the 2:1 MUXes inside the BLE. They are implemented using a single MUXing level and a shared SRAM bit. The output of each MUX is driven by a two-stage buffer enabling it to drive a frequently large downstream capacitance. Figure 3 shows a pass-transistor implementation and a transmission gate implementation of a generic two-level MUX with two-stage buffer. Note that in the pass-transistor implementation, a level-restoring PMOS transistor must be included to pull the degraded output of the MUX up to V_{DD}.

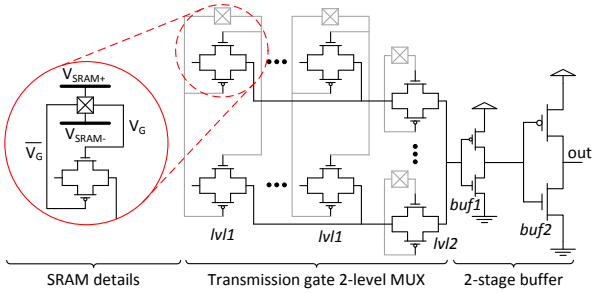
An important parameter in the design of two-level MUXes is the size of each level. If S_1 and S_2 are the sizes of the first and second levels respectively, any combination of S_1 and S_2 such that $S_1 \times S_2 = MUX_{size}$ is a possible MUX topology. Since SRAM cells occupy 35-40% of tile area (as shown in Section 5.5), we choose a MUX topology that minimizes the number of SRAM cells required by having $S_1 \approx S_2$.

3.2. Lookup-Tables

Lookup tables are generally implemented as fully encoded MUX trees where each level of the MUX tree is controlled by a LUT input. Our 6-LUT is implemented in this fashion but we insert a two-stage buffer after 3 levels to minimize



(a)



(b)

Fig. 3: A generic two-level MUX with two-stage buffer implemented with a) pass-transistors and b) transmission gates.

the quadratically increasing delay associated with chains of pass-transistors. We experimented with different inverter locations within the pass-transistor tree and found this to be the best choice. Figure 4 shows a portion of a pass-transistor based 6-LUT. The LUT contains 64 SRAM cells, a 64-input fully encoded MUX tree, 8 internal buffers, an output buffer and 6 distinctly sized input drivers. We also include an isolation buffer between the SRAM cells and the MUX to improve both speed and robustness. In our transmission gate FPGAs, pass-transistors are replaced with transmission gates and the level-restoring transistors are removed.

3.3. Flip-Flops

As we will show in Section 5.5, the impact of the flip-flops on critical path delay and tile area is relatively small. Consequently, we did not explore different FF implementations. We use a static transmission gate based master-slave register similar to the one used in [1].

3.4. Gate Boosting

Commercial FPGAs have often used a voltage greater than V_{DD} on the gates of pass-transistors (gate-boosting). The more V_G is boosted above V_{DD}, the faster a pass-transistor circuit will become due to faster and larger swinging pass-transistor outputs. A thorough comparison of pass-transistor and transmission gate FPGAs should include an analysis of the effect of gate boosting both switch types. Gate boosting a MUX is achieved by connecting SRAM cells to separate power and/or ground rails (V_{SRAM+} and V_{SRAM-} in Figure 3). Setting V_{SRAM+} above V_{DD} will effectively apply

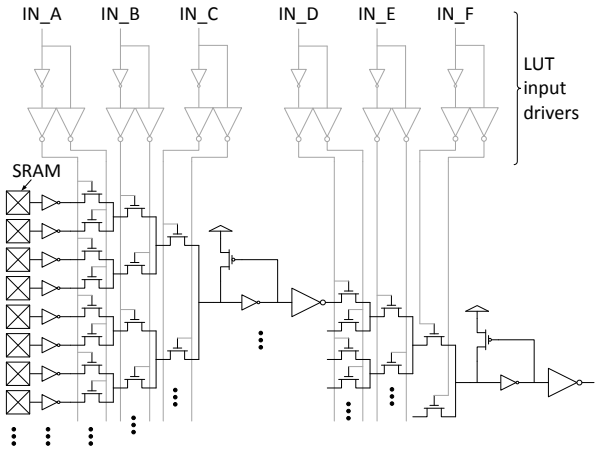


Fig. 4: Fully encoded MUX tree 6-LUT with internal re-buffering (partial view)

a higher voltage to the gates of transistors inside the multiplexer (provided the cell contains a logic-high value). In addition to increasing V_{SRAM+}, transmission gate FPGAs can set V_{SRAM-} below 0V to improve PMOS transistor performance. Since SRAM cells only switch at configuration time, gate boosting does not increase dynamic power consumption and high-V_{Th}, low-leakage transistors can be used in the SRAM cells to minimize static power consumption (their speed is not important). Through HSPICE simulation, we found that boosting the voltage by 200mV on an SRAM cell built from PTM 22nm low-power transistors increased its static leakage by 3.6× However, the SRAM contribution to the chip-wide static power consumption remained below 1mW. We do not gate boost LUTs since it is less straight forward to do so and would come at a cost of increased power consumption.

Too much gate boosting will cause faster aging by accelerating *time-dependent dielectric breakdown* and *bias-temperature instability* or could even destroy the transistor. Since it is unclear exactly how much gate boosting is safe for a 22nm process, we sweep the gate voltage over three values (V_{DD}, $V_{DD} + 0.1V$ and $V_{DD} + 0.2V$) thus providing a general indication of the effect of gate boosting from which a safe gate boosting level can be chosen. Consequently, we design six different FPGAs representing three levels of gate boosting for both pass-transistor and transmission gate switches. All six FPGAs have identical architectural parameters (W, N, K, etc.) but differ in circuit design. Throughout this paper, we refer to these FPGAs as *implementations*.

4. METHODOLOGY

To obtain a fair comparison, we optimize the transistor sizing of each of the six FPGA implementations to minimize area-delay product. Once all implementations have been optimized, tile area, critical path delay and power are measured and compared. Figure 5 shows the CAD flow used for each FPGA implementation.

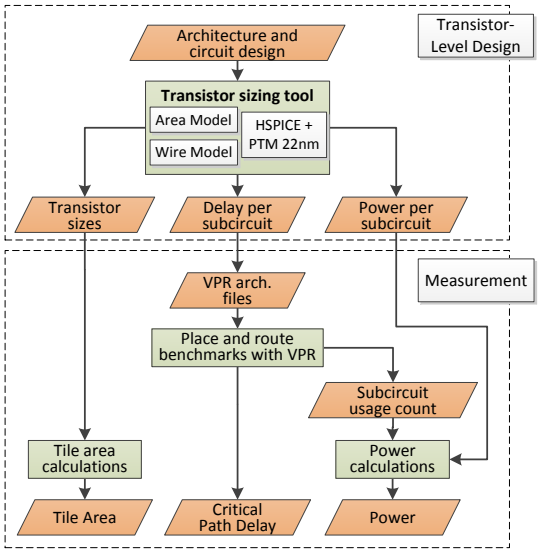


Fig. 5: CAD flow for each FPGA implementation.

4.1. Transistor-Level Design Methodology

The most accurate transistor-level design methodology involves creating a complete layout from which to extract area and delay; a process that is much too time consuming for multiple designs. We instead estimate layout area and layout-dependent wire loading with predictive models detailed below. Even with these estimates, the design space is much too large for manual exploration as there can easily be thousands of different transistor sizing combinations in a single FPGA implementation. To facilitate the transistor-level optimization process, we developed a semi-automated transistor sizing tool that finds the transistor sizing combination that yields a target area-delay objective.

4.1.1. Area Modeling

We model area via an updated version of the minimum-width transistor model of [1] which estimates the area of a transistor as a function of its relative drive-strength, x:

$$Area(x) = 0.5 + 0.5x \quad (1)$$

We find that for more advanced process technology, (1) significantly over-predicts area, particularly for large drive-strengths, with over-predictions of 21-143% for drive-strengths ranging from 2-32x minimum drive-strength. Since we do not have access to layout rules for a 22nm process, we scale TSMC's 65nm layout rules to 22nm and use a least-square fit of area versus drive-strength to obtain area as a function of drive-strength.

$$Area(x) = 0.447 + 0.128x + 0.391\sqrt{x} \quad (2)$$

The area of an FPGA subcircuit is obtained by summing the areas of all the transistors in that subcircuit. Despite the fact that 6 small transistors are required per SRAM cell, an area of 4 minimum-width transistors is used because a

denser, more optimized layout is assumed for such a frequently used cell.

For our transmission gate FPGAs, we assume that the extra PMOS transistors can be placed in existing N-wells. If this is not possible and additional wells are required, our sample layouts suggest that transmission gate FPGA area would increase by no more than 7%, which would not significantly change our overall conclusions.

4.1.2. Wire Load Modeling

To get realistic transistor sizes, it is important to include the effects of all transistor and wire loading. Transistor loads are relatively easy to determine based on architectural parameters and circuit topologies. Wire loads, on the other hand, are length-dependent making them more difficult to determine since the exact layout is not known. We estimate wire lengths based on the area estimates of (2) along with a set of general layout assumptions. For example, local interconnect wires (see Figure 2) are assumed to span the height of a logic cluster. The logic cluster's layout is assumed to be square and its area is obtained from our area model. Since the effects of wire loading are becoming more important in advanced process technology, we model all wire loading as far down as the metal connecting two transistors inside a multiplexer. All wire loads are automatically accounted for by our transistor sizing tool. Wire resistance and capacitance per unit length are extracted from *ITRS 2011* [16]. All wires are implemented in ITRS's intermediate layer (minimum width and spacing) except for general routing wires which are implemented in the semi-global layer (2x minimum width and spacing).

4.1.3. Transistor Sizing Tool

Our transistor sizing tool solves the same problem as Kuon and Rose's automated transistor sizing tool [17] but we take a different approach. While [17] sizes an entire FPGA tile at once by optimizing a representative critical path that contains at least one of each type of FPGA subcircuit (LUTs, MUXes, etc.), we size each subcircuit individually. This difference stems from our different delay measurement tactics. Optimizing a representative critical path presents a huge design space which [17] confronts with a two-phase algorithm consisting of an exploratory phase that utilizes linear device models to keep CPU times reasonable followed by an HSPICE-based fine-tuning phase that adjusts the transistor sizes to account for the inaccuracies of linear models. We found linear device models to be highly inaccurate at 22nm, so our tool relies exclusively on HSPICE simulations to measure delay. Area is calculated with the model of Section 4.1.1. Exhaustively simulating large quantities of transistor sizing combinations quickly reaches prohibitively long runtimes. We tackle this problem in two ways.

First, transistor sizing is performed on subcircuits rather than larger structures (e.g. a tile). This divide-and-conquer approach produces smaller search spaces but requires iteration to account for changing transistor loads. That is, subcircuits are usually loaded by other subcircuits and changing the transistor sizes of one subcircuit changes the load on another. In our experience, transistor sizes usually stabilize after 2-4 iterations.

Second, we size the NMOS and PMOS of transmission gates and inverters as a unit. More specifically, instead of sizing the NMOS and PMOS of a transmission gate independently, the tool forces them to be of equal size and changes them both simultaneously. Similarly, the NMOS and PMOS of an inverter are sized concurrently based on some P/N ratio. The initial P/N ratio is determined by equalizing the inverter rise and fall times for a mid-range transistor sizing combination of the subcircuit. Once the best area-delay sizing is found, the P/N ratios of all inverters are re-optimized in a final step to balance rise and fall times.

4.2. Area, Delay and Power Measurement Methodology

Tile area is obtained by first calculating the area of each FPGA subcircuit using our area model and the final transistor sizes obtained from the transistor sizing tool. Then, the subcircuit areas are multiplied by the number of subcircuits in a tile (Table 1) and summed to obtain total area.

A VPR architecture file is created for each of the six FPGA implementations. Critical path delay is measured experimentally with VPR by placing and routing MCNC [18] and VTR [19] benchmarks on each FPGA for five different placement seeds.

Dynamic power is obtained for each FPGA subcircuit by using HSPICE to measure the average current required to propagate a rising and a falling transition through the subcircuit and then multiplying it by V_{DD}. To compute relative total power, we multiply the power-per-subcircuit numbers by the average number of times each subcircuit is used in VPR placed and routed benchmarks. Since we are only interested in a relative power comparison between our six FPGA implementations, we do not need to perform a functional simulation to obtain toggle activities as we expect them to be the same across implementations except for very slight glitch changes due small variations in timing.

5. RESULTS

5.1. Choosing Fc_{out}

Previous work has shown that $Fc_{out} = W/N$ is an appropriate cluster output pin flexibility [1]. However, our cluster output architecture differs from that of [1] (e.g. two outputs per BLE and single-driver routing wires). Therefore, we reinvestigate cluster output pin flexibility. The area tradeoffs are as follows. Smaller Fc_{out} values lead to smaller switch block MUXes as there are fewer connections from the cluster outputs to routing wires. However, larger channels are needed due to poorer routability, leading to a larger number of switch block MUXes. The delay tradeoffs are similar.

Table 2: Area and delay for different Fc_{out} values.

Fc_{out}	W	Tile Area (μm^2)	Crit. Path Delay (ns)	Area-Delay Product
0.250	288	936	7.96	7.4
0.100	296	891	7.83	7.0
0.025	320	873	7.84	6.8

Smaller values of Fc_{out} reduce loading and lead to faster cluster outputs but might lead to circuitous routing.

We use VPR to place and route the MCNC benchmarks on three architectures with different values of Fc_{out}. The channel width for each architecture is chosen such that all architectures are equally routable (same W/W_{min} where W_{min} is the average minimum channel width required to successfully route the benchmarks) despite their differing Fc_{out} values. Tile area and critical path delay for each architecture is shown in Table 2. Based on these results, we set $Fc_{out} = 0.025W$ as it gives the best area-delay product for our $N = 10$, $K = 6$ and $Fc_{in} = 0.2W$ architecture. Since single-driver routing reduces the portion of a routing channel that can be accessed by logic cluster outputs to W/L, it seems intuitive that Fc_{out} should be lower than it is for architectures with tri-state driver routing [1] where the whole channel is accessible.

5.2. Gate-Boosting Transmission Gates

A transmission gate can be gate boosted by applying a voltage larger than V_{DD} on the gate of the NMOS transistor, by applying a voltage smaller than 0V on the gate of the PMOS transistor or by a mixture of both. To choose a gate boosting strategy, we experiment with different levels of gate boosting on our completely optimized, non-gate boosted, transmission gate FPGA design.

Figure 6 shows the delay reductions observed in the switch block MUXes; results for other MUXes follow the same trend. Gate boosting only the NMOS transistor (leftmost bar graph) results in almost twice the delay reduction that is obtained when only the PMOS transistor is gate boosted and results in nearly the same amount of delay reduction obtained when both transistors are gate boosted. Therefore, we choose to only gate boost the NMOS transistors of transmission gates since the additional delay reduction achieved by also gate boosting the PMOS transistors probably does not merit the creation of a new supply plane. As well, some transistors in the configuration SRAMs will be subjected to a voltage difference of $V_{SRAM+} - V_{SRAM-}$. Hence, simultaneously gate boosting both NMOS and PMOS transistors by some voltage increases the reliability risk versus gate boosting only the NMOS transistors by that voltage. Bars of the same color in Figure 6 have the same stress on the SRAM cells.

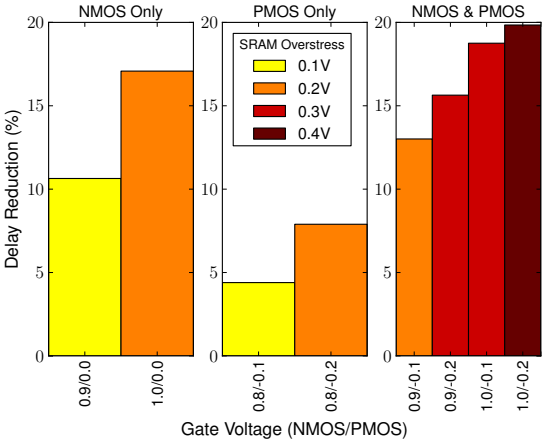


Fig. 6: Effect of different gate boosting strategies on transmission gate switch block multiplexer delay ($V_{DD} = 0.8V$).

Table 3: FPGA tile area.

V_G	PT (μm^2)	TG (μm^2)	TG/PT
V_{DD}	875	1006	15.0%
$V_{DD} + 0.1V$	873	1010	15.7%
$V_{DD} + 0.2V$	887	1015	14.5%

5.3. Pass-Transistor Vs. Transmission Gate FPGAs

Table 3 shows the tile area for pass-transistor (PT) and transmission gate (TG) FPGAs with different levels of gate boosting ($V_{DD} = 0.8V$ in this section). The results indicate that transmission gate FPGAs are approximately 15% larger than pass-transistor FPGAs. Gate boosting does not significantly affect tile area. In general, we noticed that as the level of gate boosting is increased on pass-transistor FPGAs, our transistor sizing tool tends to reduce pass-transistor sizes but increases buffer sizes resulting in an FPGA that has similar tile area but reduced delay. Due to their larger area, our transistor sizing tool almost always choses minimum sized transmission gates. The buffers in transmission gate FPGAs are larger than those of pass-transistor FPGAs due to more transistor and wire loading. The P/N ratios of buffers are also different for different levels of gate boosting as the signal swings at the buffer inputs are changing. Table 4 shows the transistor sizes for a switch block MUX in units of minimum *contactable* transistor width (45nm in this process).

Table 5 shows average critical path delay for all 6 FPGA designs for the VTR benchmark set (MCNC benchmarks yielded similar results). The results show that, with no gate boosting, transmission gate FPGAs are 25% faster than pass-transistor FPGAs. As the level of gate boosting is increased, the delay gap is reduced but transmission gate FPGAs remain faster. The higher speed with transmission gates is due to the increased voltage swing and the fact that we now have two switch transistors in parallel, providing lower resistance. The resistance of transmission gates is further reduced in advanced processes because highly strained

Table 4: Switch block multiplexer transistor sizes for PT and TG implementations for different levels of gate boosting (see Figure 3 for transistor labels). Note that with the exception of P/N ratios, the transistor sizing tool uses integer granularity.

$Type, V_G$	lvl1		lvl2		buf1	buf2
	P	N	P	N	P/N	P/N
$PT, V_{DD} + 0.0$	-	3	-	3	3/3.2	20.7/11
$PT, V_{DD} + 0.1$	-	2	-	2	7.0/3	31.6/12
$PT, V_{DD} + 0.2$	-	2	-	2	12.6/3	37.5/14
$TG, V_{DD} + 0.0$	1	1	1	1	3/4.3	35.7/21
$TG, V_{DD} + 0.1$	1	1	1	1	4/4.3	39.9/19
$TG, V_{DD} + 0.2$	1	1	1	1	5.6/4	44.9/19

Table 5: Critical path delay (VTR benchmarks).

V_G	PT (ns)	TG (ns)	TG/PT
V_{DD}	23.3	17.4	-25.4%
$V_{DD} + 0.1V$	18.9	15.8	-16.3%
$V_{DD} + 0.2V$	16.2	14.4	-10.7%

silicon has narrowed the gap between PMOS and NMOS mobility.

The area-delay product for each FPGA design is given in Table 6. With no gate boosting, transmission gate FPGAs have an area-delay product that is 14% lower than pass-transistor FPGAs. However, given the right amount of gate boosting (in this case somewhere between +0.1V and +0.2V), pass-transistor FPGAs eventually become more efficient than transmission gate FPGAs.

Table 7 shows dynamic power, normalized to the non-gate boosted pass-transistor FPGA implementation. Transmission gate FPGAs consume slightly more power than pass-transistor FPGAs. This is likely due to their larger tile area. The small decrease in power consumption experienced by pass-transistor FPGAs with 0.1V of gate boosting is due to reduced short-circuit current. With 0.2V of gate boosting however, the gains from reduced short-circuit current are lost due to the power increase from higher voltage swings in the internals of the pass-transistor MUXes.

5.4. Decoupling V_{DD} and V_G for Low-Power FPGAs

An FPGA that employs adaptive voltage scaling can trade delay for power by using an operating V_{DD} that is lower than its nominal supply voltage (V_{DDn}). To reduce the delay penalty without adversely affecting power, the resulting low-power FPGA can mimic the concept of gate boosting by lowering V_{DD} but not V_G. What is particularly interesting about decoupling V_{DD} and V_G in this way is the fact that, as long as $V_G <= V_{DDn}$, "gate boosting" low-power FPGAs does not pose a reliability risk as it does for FPGAs running at V_{DDn} where any amount of gate boosting results in $V_G > V_{DDn}$.

We explore the idea of adaptive voltage scaling with decoupled V_{DD} and V_G on our non-gate boosted pass-transistor and transmission gate FPGA implementations (that have been fully optimized for $V_{DD} = 0.8V$) by experimenting with two low-power FPGA schemes. In the first, V_{DD} and V_G are kept equal and are both lowered below 0.8V to produce a low-power mode. In the second, V_G is maintained at 0.8V and only V_{DD} is lowered, resulting in a "gate boosted" low-power mode. Figure 7 shows critical path delay and dynamic power (normalized to PT, $V_{DD} = V_G = 0.8V$) for both schemes. The results show that lowering V_{DD} and V_G to 0.6V results in a 2× power reduction for both pass-transistor and transmission gate FPGAs but a 6× and 2.5× increase in delay respectively. However, if we maintain V_G at 0.8V when V_{DD} is lowered to 0.6V, pass-transistor and transmission gate FPGA delays improve by 65% and 18% respectively at no additional power cost. Clearly pass-transistor FPGAs are a very poor choice for low-power if gate voltages are not maintained at V_{DDn}.

Figure 8 shows that decoupling V_{DD} and V_G for low-power FPGAs is very beneficial. If we maintain V_G at 0.8V, the V_{DD} yielding minimal power-delay product shifts from 0.8V to 0.7V where we experience a 25% power reduction. In addition, the results indicate that transmission gate FPGAs always achieve lower power-delay product than pass-transistor FPGAs in the low-power regime with a 26% advantage at 0.6V.

Table 6: Area-delay product (VTR benchmarks).

V_G	PT	TG	TG/PT
V_{DD}	20.4	17.5	-14.2%
$V_{DD} + 0.1V$	16.5	16.0	-3.1%
$V_{DD} + 0.2V$	14.3	14.7	2.2%

Table 7: Relative power (VTR benchmarks).

V_G	PT	TG	TG/PT
V_{DD}	1.00	1.04	3.8%
$V_{DD} + 0.1V$	0.99	1.05	6.4%
$V_{DD} + 0.2V$	1.02	1.06	4.4%

5.5. Area and Delay Breakdown

Figure 9a shows the area contributions of different FPGA subcircuits averaged over our 6 FPGA implementations. Approximately 26% of the area is devoted to BLEs (LUT + FF) leaving 74% of the area to routing. This number is lower than the 90% routing area commonly quoted in academic work (e.g. [20]), but is higher than the commercial Stratix V architecture where routing area is said to account for only 50% of tile area [9]. This discrepancy could be due to our architecture having fewer features than commercial architectures (e.g adders, more complex FFs, LUTRAM, etc.). SRAM cells cover 40% of tile area for pass-transistor FPGAs and 35% of tile area for transmission gate FPGAs.

The critical path contributions are shown in Figure 9b. Approximately 24.5% of the critical path delay comes from

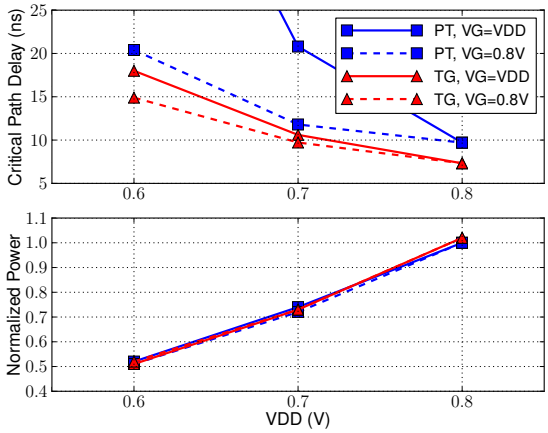


Fig. 7: Critical path delay (top) and dynamic power (bottom) for PT and TG FPGAs for different V_{DD} and V_G voltages.

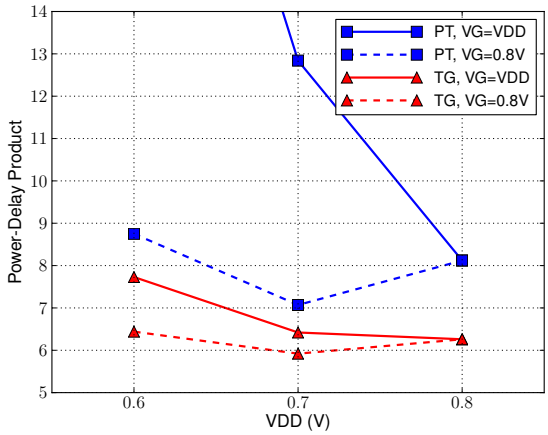


Fig. 8: Power-delay product for PT and TG FPGAs for different V_{DD} and V_G voltages.

the BLEs, 73% comes from the routing and 2.5% comes from hard multipliers and block memory (where we use Stratix IV-like delay values).

6. CONCLUSION

We develop a new methodology for designing FPGA circuitry and use it to compare pass-transistor and transmission gate FPGAs in 22nm process technology. Transmission gate FPGAs consume 15% more area than pass-transistor FPGAs but are 25%, 16% and 10% faster for 0V, 0.1V, and 0.2V of gate boosting respectively. In terms of area-delay product, transmission gate FPGAs are 14% better than pass-transistor FPGAs without gate boosting but 2% worse with 0.2V of gate boosting. Clearly, if gate boosting is not permitted, building FPGAs out of transmission gates is the better choice. However, given enough gate boosting, pass-transistor FPGAs are still more efficient. Even if 0.2V of gate boosting is safe, however, a case can be made for transmission gate FPGAs due to the reliability concerns associated with pass-transistors in advanced process technology as they incur only a 2% area-delay product and 5% power increase. If low-V_{DD} operation is desired, transmission gate FPGAs that maintain V_G at the nominal supply voltage yield the best power-delay product.

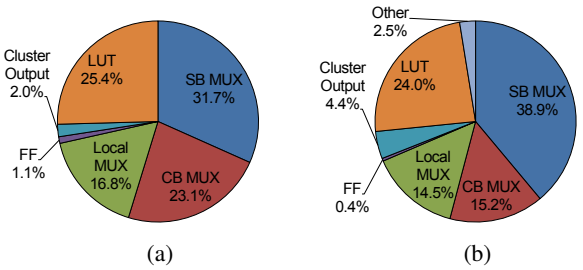


Fig. 9: Tile area (a) and critical path delay (b) breakdown.

ACKNOWLEDGMENTS

The authors would like to thank David Lewis for insightful discussions, NSERC and Altera Corporation for funding this research and CMC for providing CAD tools.

REFERENCES

[1] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer, 1999.

[2] S. Kiamehr, A. Amouri, and M. Tahoori, "Investigation of NBTI and PBTI Induced Aging in Different LUT Implementations," in *FPT 2011*, pp. 1–8.

[3] A. Amouri, S. Kiamehr, and M. Tahoori, "Investigation of Aging Effects in Different Implementations and Structures of Programmable Routing Resources of FPGAs," in *FPT 2012*, pp. 215 –219.

[4] A. Telikepalli, "Power vs. Performance: The 90 nm Inflection Point," *Xilinx White Paper*, vol. 223, 2006.

[5] T. Pi and P. J. Crotty, "FPGA Lookup Table with Transmission Gate Structure for Reliable Low-Voltage Operation," U.S. Patent 6 667 635, Dec. 23, 2003.

[6] E. Lee, G. Lemieux, and S. Mirabbasi, "Interconnect Driver Design for Long Wires in Field-Programmable Gate Arrays," *Journal of Signal Processing Systems*, pp. 57–76, 2008.

[7] Predictive Technology Model (PTM), *http://ptm.asu.edu/*, 2012.

[8] E. Ahmed and J. Rose, "The Effect of LUT and Cluster Size on Deep-Submicron FPGA Performance and Density," *TVLSI*, pp. 288–298, March 2004.

[9] D. Lewis et al., "Architectural Enhancements in Stratix V™," in *FPGA 2013*, pp. 147–156.

[10] Xilinx Inc., "7 Series FPGAs Overview," *Data Sheet*, 2012.

[11] D. Lewis et al., "The Stratix™ Routing and Logic Architecture," in *FPGA 2003*, pp. 12–20.

[12] G. Lemieux and D. Lewis, "Using Sparse Crossbars within LUT Clusters," in *FPGA 2001*, pp. 59–68.

[13] G. Lemieux et al., "Directional and Single-Driver Wires in FPGA Interconnect," in *FPT 2004*, pp. 41–48.

[14] C. Chen et al., "Efficient FPGAs using Nanoelectromechanical Relays," in *FPGA 2010*, pp. 273–282.

[15] D. Lewis et al., "The Stratix II™ Logic and Routing Architecture," in *FPGA 2005*, pp. 14–20.

[16] ITRS, *Interconnect Chapter*, 2011.

[17] I. Kuon and J. Rose, "Automated Transistor Sizing for FPGA Architecture Exploration," in *DAC 2008*, pp. 792–795.

[18] S. Yang, "Logic Synthesis and Optimization Benchmarks, Version 3.0," in *Tech. Report*. MCNC, 1991.

[19] J. Rose et al., "The VTR Project: Architecture and CAD for FPGAs from Verilog to Routing," in *FPGA 2012*, pp. 77–86.

[20] G. Lemieux and D. Lewis, *Design of Interconnection Networks for Programmable Logic*. Kluwer, 2004.