COFFE: Fully-Automated Transistor Sizing for FPGAs

Charles Chiasson and Vaughn Betz

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

***Abstract*—In this paper, we present COFFE (*Circuit Optimization For FPGA Exploration*), a new fully-automated transistor sizing tool for FPGAs. Automated transistor-level CAD tools are an important part of the architecture exploration flow because they provide accurate area and delay estimates of low-level FPGA circuitry, which must be obtained for each architecture. We show that modeling transistors as linear resistances and capacitances as has been done in previous FPGA transistor sizing tools is highly inaccurate for fine-grained transistor-level design in advanced process nodes. Therefore, COFFE's transistor sizing algorithm maintains circuit non-linearities by relying exclusively on HSPICE simulations to measure delay. Area is estimated with a transistor size-based model that incorporates a number of improvements to enhance its accuracy in advanced process technologies versus prior methods. In addition to more accurate area and delay estimation, COFFE considers more layout effects than prior published work by automatically accounting for transistor and wire loads, which are computed based on architectural parameters and layout area. This new FPGA transistor sizing tool requires only several hours to produce high-quality transistor sizing results for an entire FPGA tile; a task that would normally take months of manual effort. We demonstrate COFFE's utility in FPGA architecture studies by investigating an important new architectural question at the logic-to-routing interface.**

I. Introduction

When developing a new chip, FPGA architects are faced with two main tasks: choosing an architecture for their FPGA and performing the transistor-level design of that architecture. Choosing an architecture is typically accomplished with architecture exploration tools such as VPR [1]. By implementing benchmark circuits on a proposed FPGA, these tools allow architects to evaluate the area, delay and power impact of various architectural choices. Based on their observations, architects can then select an FPGA architecture that meets their design goals and constraints.

Transistor-level design consists of selecting circuit topologies for the various subcircuits that implement the chosen architecture as well as sizing the transistors of those subcircuits. Transistor-level design is an essential precursor to the evaluation of an architecture because it provides accurate area, delay and power estimates of the underlying FPGA circuitry; these estimates are required inputs to the architecture exploration tools. Transistor sizing also provides an additional opportunity to tune the area, delay and power of an FPGA. Therefore, developing a new FPGA is an iterative process that involves performing the transistor-level design of various architectures before evaluating them through synthesis, placement and routing experiments. This interdependence between architecture exploration and transistor-level design necessitates automated design tools if high-quality results are to be obtained in reasonable amounts of time. In this paper, we describe COFFE (*Circuit Optimization For FPGA Exploration*), a fully-automated transistor sizing tool for FPGAs that enables the design flow detailed above by providing area, delay and power estimates of properly sized FPGA circuitry. COFFE also enables design exploration of FPGA circuitry, of which we give an example in Section VIII.

Transistor sizing for custom circuits is a well-studied problem that consists of improving a circuit's performance by increasing the sizes of its transistors. This optimization problem is usually formulated in one of three ways: 1) minimize some function of area and delay, 2) minimize area subject to a delay constraint or 3) minimize delay subject to an area constraint. In [2], it was shown that modeling transistors as linear resistances and capacitances and calculating the delay of the resulting RC circuits with the Elmore [3] or the Penfield-Rubinstein [4] delay model allows the transistor sizing problem to be formulated as a convex optimization problem, which guarantees that any local minimum is the global minimum. With this useful property, [2] develops TILOS, a transistor sizing tool for custom circuits based on a heuristic method that iteratively identifies a circuit's critical path and increases transistor sizes on that path until all timing constraints are met. Despite the convexity of the problem, TILOS's heuristic is such that it can terminate with a sub-optimal solution. Algorithms guaranteeing the optimal solution have subsequently been proposed [5]–[7] but these algorithms, along with TILOS, all suffer from their reliance on linear device models and the Elmore delay, which have long been known to be inaccurate [8], [9]. To enhance accuracy, at the cost of increased computational complexity, some transistor sizing algorithms have turned towards time-domain simulation to obtain delay estimates [10], [11].

The programmability of FPGAs adds unique features to the transistor sizing problem that Kuon and Rose tackle with an FPGA-specific transistor sizing tool [12]. Their two-phased algorithm consists of an exploratory phase that utilizes linear device models and a TILOS-like transistor sizing heuristic followed by an HSPICE-based fine-tuning phase that adjusts the transistor sizes to account for the inaccuracies of linear models. In [13], Smith et al. present a method that enables the rapid and concurrent optimization of high-level architecture parameters and transistor sizes for FPGAs through the use of analytic architecture models, linear device models and a convex optimization-based transistor sizing algorithm. They show that this concurrent optimization can have a significant impact on architectural conclusions versus separate optimization. COFFE differs from both [12] and [13] because it completely avoids the use of linear models and makes other modeling improvements which are necessary for FPGAs in advanced process nodes. Specifically, our contributions include:

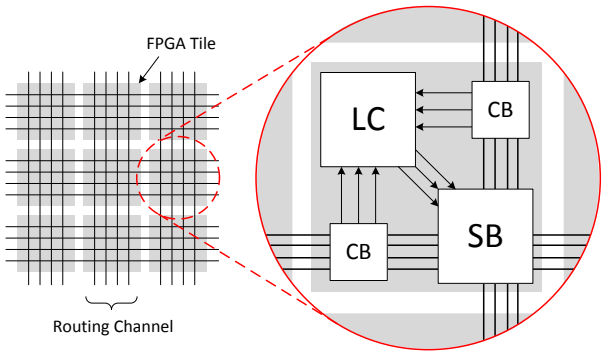


Fig. 1: Tile-based FPGA.

- An FPGA transistor sizing tool[1] that sizes the transistors of an entire FPGA tile by intelligently searching the design space while modeling all circuit non-linearity. We show in Section III that this full non-linear modeling is key.
- New and more accurate area and wire load models.
- An analysis of the effect of wire loading at the interface between logic clusters and routing channels.

II. Problem Formulation and Design Flow

A. FPGA Architecture

An FPGA is composed of an array of *tiles* interconnected by programmable routing channels (Fig. 1). Each tile consists of a logic cluster (LC), a connection block (CB) and a switch block (SB). The logic cluster implements a small amount of logic and local routing while the connection block and switch block provide connectivity between the logic cluster and the routing channels. The switch block also provides connectivity between wires within the routing channels.

Fig. 2 shows the tile architecture that COFFE supports in its designs and Table I lists the architecture parameters that COFFE expects as inputs. Parameters listed in the top portion of Table I are commonly used in FPGA research [1], [14]. Parameters listed in the bottom portion are new and help COFFE describe a more flexible basic logic element (BLE) than the commonly used academic BLE [1]. The COFFE BLE still consists of a K-input lookup table (LUT) and flip-flop (FF) but has more flexible ways to use the LUT and FF simultaneously. The R_{fb} parameter allows optionally including a register feedback multiplexer (MUX-A in Fig. 2) on a LUT input. Each input has an R_{fb} parameter. The FF can be designed to accept its input either directly from the LUT output or from either the LUT output or a BLE input (through MUX-B) with the R_{sel} parameter. COFFE's BLEs support variable numbers of local feedback and general routing outputs which are specified by the O_{fb} and O_r parameters respectively. All BLE outputs can be driven by either the LUT output or the FF output (MUX-C and MUX-D). These extra 2:1 MUXes can potentially help improve density and speed and are similar to the ones used in Stratix [15]. COFFE currently only supports one wire segment length (L) and uses directional, single-driver routing wires [16].

B. Circuit Topologies

The architecture described in the previous section consists entirely of LUTs, MUXes and FFs. COFFE assumes a two-level MUX topology as shown in Fig. 3a for all its MUXes except for the 2:1 MUXes inside the BLE, which are implemented with a single MUXing level as shown in Fig. 3b. A two-level topology is assumed because it is used in commercial architectures [17] and has been shown to have the best area-delay product in [18]. LUTs are implemented with a fully encoded MUX tree topology. Fig. 4 shows the topology for a 6-LUT where internal buffering is included to minimize the quadratically increasing delay of chains of pass-transistors.

[1]Available at: http://www.eecg.utoronto.ca/~vaughn/software.html.

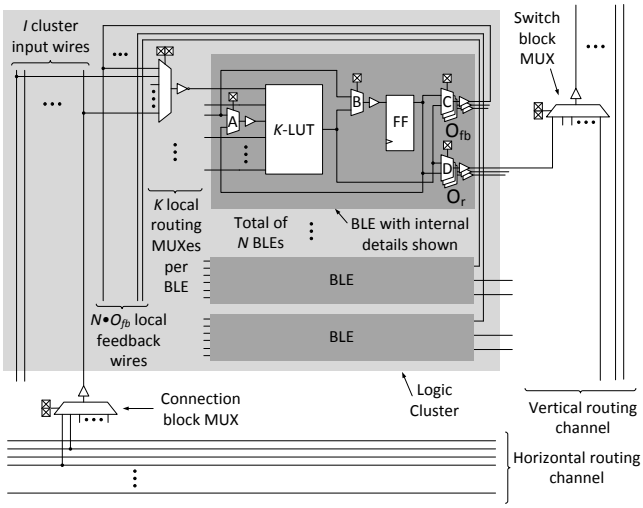


Fig. 2: COFFE's supported tile architecture.

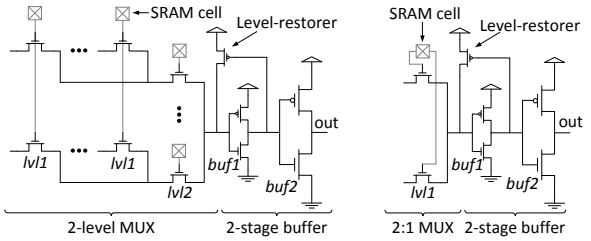
TABLE I: COFFE's expected input architecture parameters.

Parameter	Description
K	LUT size
N	Cluster size
I	Number of cluster inputs
Fc_{in}	Cluster input connection flexibility
Fc_{out}	Cluster output connection flexibility
W	Routing channel width
L	Wire segment length
F_S	Switch block flexibility
Fc_{local}	Local interconnect to BLE input connection flexibility [14]
R_{fb}	Register feedback per LUT input (on/off)
R_{sel}	Register input select (LUT only/LUT and BLE input)
O_{fb}	Number of local feedback outputs per BLE
O_r	Number of general routing outputs per BLE

C. Transistor Sizing for FPGAs

As described in Section II-A, an FPGA consists of an array of tiles. Since these tiles are all identical, transistor-level design only needs to be performed for one of them. This design can then be replicated to obtain a complete FPGA. Similar design space reductions can be found within a tile. For example, a switch block can include over 100 logically equivalent multiplexers whose transistor-level design should be kept identical. Consequently, only ~80 unique transistors need to be sized when designing an FPGA despite there being millions of transistors on the chip, which is in contrast to transistor sizing for custom circuits where the whole chip must be considered. This reduced design space makes HSPICE-based optimization practical for FPGAs, but as we show in Section VI, we must still search this space intelligently with COFFE to keep runtime reasonable.

There is another aspect of transistor sizing that is very different for FPGAs. Because they are programmable, FPGAs have application dependent critical paths which implies that



(a) (b)

Fig. 3: a) Two-level MUX topology. b) 2:1 MUX topology.

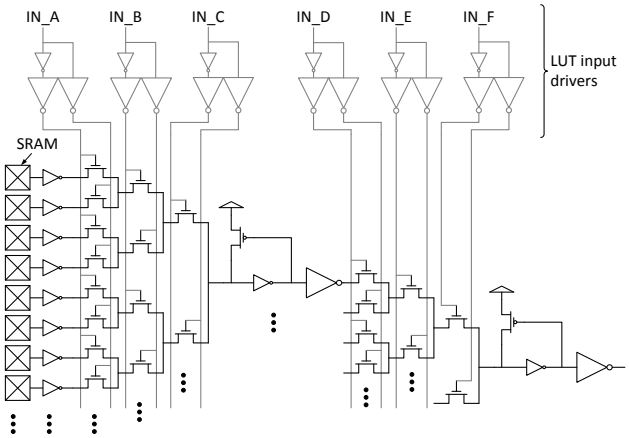


Fig. 4: Fully-encoded MUX tree 6-LUT topology with internal buffering (partial view).

at design time, there is no clear critical path to optimize for delay. To deal with this issue, [12] optimizes a *representative* path that contains one of each type of FPGA subcircuit (LUTs, MUXes, etc.). Delay is taken as a weighted sum of the delay of each subcircuit and the weighting scheme is chosen based on the frequency with which each subcircuit was found on the critical paths of placed and routed benchmark circuits. In [19], the lack of a critical path is confronted by simply optimizing each subcircuit individually. As we will describe in more detail in Section VI, COFFE can be configured to use either of these two approaches.

D. Design Flow

Fig. 5 shows the FPGA design flow we wish to enable. COFFE is used to perform transistor-level optimization for some architecture of interest thus producing accurate area and delay estimates for the subcircuits of this architecture. These estimates are used by VPR to evaluate the architecture through place and route experiments. Based on the results of the assessment, the architecture parameters are adjusted and sent back to COFFE to begin a new iteration of optimization and evaluation.

COFFE's transistor-level optimization makes area and performance tradeoffs through transistor sizing. Like [12], COFFE's optimization objective is of the form $Area^b Delay^c$ thus allowing for different area and performance tradeoffs by varying b and c. Creating a complete layout is the most accurate way to obtain the area and delay measurements needed during transistor sizing. However, for the iterative design flow of Fig. 5, this approach is impractical as layout is a very time consuming task. Instead, COFFE estimates

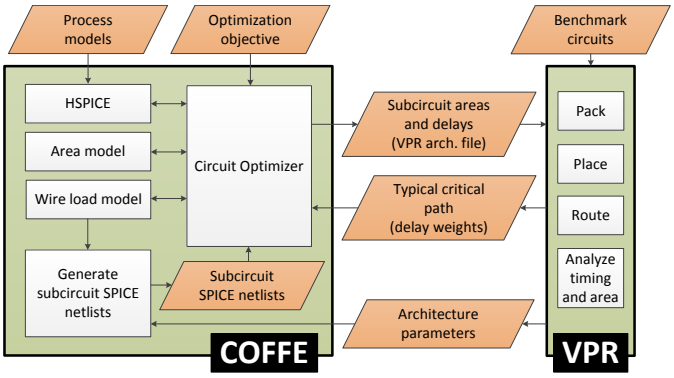


Fig. 5: FPGA design flow.

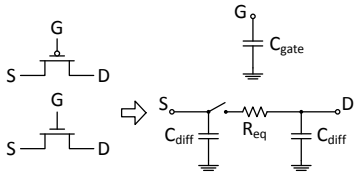


Fig. 6: A switch-level model.

area with the predictive model described in Section IV and measures delay with HSPICE simulations. Although previous FPGA transistor sizing tools have used linearized models of transistors to measure delay during certain phases of the optimization, we show in Section III that such models are highly inaccurate for the fine-grained transistor-level design we wish to undertake.

COFFE automatically generates the SPICE netlists required for delay measurement based on the input architecture (Table I) and the circuit topologies described in Section II-B. To obtain meaningful delays, COFFE is careful to ensure that these netlists include realistic transistor and wire loads. Transistor loads are easy to determine based on architecture parameters and circuit topologies. Wire loads, on the other hand, are layout dependent making them more difficult to determine since the exact layout is not known. COFFE estimates wire loads with the model described in Section V.

III. The Problem with Switch-Level Modeling

We define switch-level modeling as the characterization of complex, non-linear MOSFET transistors into a set of linear resistances and capacitances (Fig. 6). Although they are less accurate at modeling transistor behavior than circuit simulators like SPICE, switch-level models are often used for delay estimation [2], [8], [12], [13] because the delay of the equivalent RC circuits can be computed with the Elmore [3] or the Penfield-Rubinstein [4] delay models which are much quicker than the time-domain simulations required to measure delay with SPICE. In addition, [2] showed that when transistors are treated as linear resistances and capacitances, the transistor sizing problem can be formulated as a convex one thus guaranteeing that a local minimum is always the global minimum.

The resistive and capacitive behavior of a transistor is influenced by a variety of factors such as its operating-point, its size and the shape of the input waveform. Therefore, switch-level models are most accurate when estimating delay for circuits that exhibit a high degree of regularity (e.g. a circuit composed of a few basic gates with a limited number of

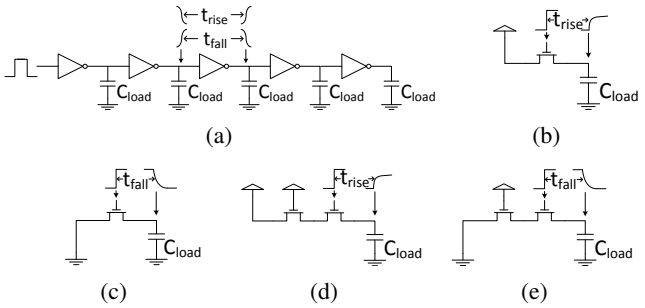


Fig. 7: Circuits used to measure transistor resistance.

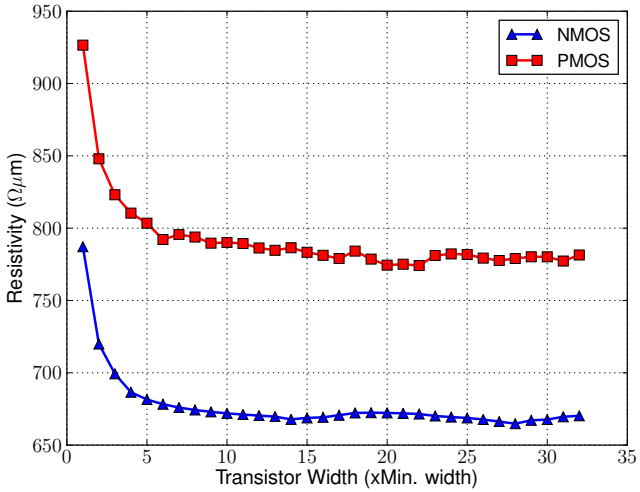


Fig. 8: Inverter NMOS and PMOS resistivity vs. transistor width.

P/N ratios) because many transistors will experience similar operating conditions. Different resistance and capacitance values (R_{eq}, C_{gate} and C_{diff}) can be used for each group of transistors experiencing similar operating conditions to construct a reasonably accurate switch-level model.

FPGA circuit design consists of custom, fine-grained transistor-level design which can lead to a large variation in transistor operating conditions. Using PTM 22nm HP device models [20] and HSPICE, we experimented with switch-level modeling for some of the circuit topologies commonly used in FPGAs. In the following sections, we highlight some of the reasons why we found that switch-level models were not suitable for our purposes.

A. Non-Linearity of Transistor Resistance and Capacitance

We use a chain of five loaded inverters (Fig. 7a) to find the equivalent switching resistance for the NMOS and PMOS of an inverter. Using a large C_{load} to minimize the effects of transistor capacitances, we simulate this circuit with HSPICE for several transistor widths and measure the rise and fall times of the third inverter in the chain (to avoid end-effects). The rise time, t_{rise}, is measured as the time it takes for the inverter output to rise from $0V$ to $V_{DD}/2$ and the fall time, t_{fall}, the time it takes for the output to fall from V_{DD} to $V_{DD}/2$. Delay measurement in both cases starts when the input of the inverter is at $V_{DD}/2$. With the rise and fall times, NMOS and PMOS switching resistances can be computed as $R_N = t_{fall}/C_{load}$ and $R_P = t_{rise}/C_{load}$. As shown in Fig. 8, our experiments show that transistor resistance varies with transistor width, particularly for smaller transistors. We found the same to be true for transistor capacitance. This implies that transistor resistance and capacitance are non-linear functions of transistor width and as a result, an accurate switch-level model would require a table of pre-computed resistances and capacitances for many different transistor widths.

TABLE II: Resistance of a $4\times$ minimum-width NMOS transistor for different circuit topologies (Fig. 7) and switching-thresholds.

Circuit Topology	Transition Type	Switching-Threshold	Resistance ($k\Omega$)
Chain of 5 inverters	fall	$V_{DD}/2$	3.8
Single pass-tran.	fall	$V_{DD}/2$	1.9
Single pass-tran.	rise	$V_{DD}/2$	13.7
Single pass-tran.	fall	$V_{DD}/3$	2.7
Single pass-tran.	rise	$V_{DD}/3$	2.2
2 series pass-tran.	fall	$V_{DD}/3$	2.8
2 series pass-tran.	rise	$V_{DD}/3$	3.3

B. Topology Dependence of Transistor Resistance

The switching resistance of an NMOS pass-transistor, a key building block for FPGA circuitry, is different than that of an NMOS in an inverter. Furthermore, the resistance of a pass-transistor is different during rising and falling transitions due to the NMOS's inability to propagate a full rising transition. Using HSPICE simulations, we measure the resistances of an NMOS pass-transistor by charging and discharging a large capacitor through a single pass-transistor (Figures 7b and 7c). Again, t_{rise} and t_{fall} are measured from $V_{DD}/2$.

In Table II, we compare the rising and falling resistance of a pass-transistor to the resistance of an NMOS in an inverter for a $4\times$ minimum-width transistor. We can clearly see that the resistance of the NMOS in the inverter (3.8k) is different from the rising (13.7k) and falling (1.9k) resistances of a pass-transistor. The very large rising resistance is caused by the pass-transistor's degraded output voltage. It is possible to achieve more balanced rising and falling resistances by measuring t_{rise} and t_{fall} at $V_{DD}/3$ instead of $V_{DD}/2$ which in terms of circuit design, corresponds to lowering the switching-thresholds of downstream inverters by skewing their P/N ratios. As shown in Table II, at $V_{DD}/3$ the rising and falling resistances of a pass-transistor are 2.2k and 2.7k respectively. Table II also shows that the resistance of an NMOS in a chain of 2 series connected pass-transistors (Figures 7d and 7e) is different from both the single pass-transistor and the inverter.

The results of Table II demonstrate that the custom pass-transistor based topologies of FPGA circuitry do not lend themselves well to switch-level modeling. Not only does resistance depend on circuit topology, it also depends on the switching-threshold of downstream inverters and on transistor dimensions (Section III-A). The complexity of a switch-level model sufficiently accurate for the type of fine-grained transistor level design we wish to undertake is impractical so we rely solely on circuit simulation to estimate delay.

IV. Area Modeling

The most accurate way to determine the area of an FPGA is to create a complete layout, as FPGAs are generally transistor area limited [1]. However, as described in Section II-D, designing an FPGA is an iterative process, making layout of each iteration impractical. A fast to compute but accurate estimate of transistor layout area is needed. The minimum-width transistor area model of [1] is such an area estimation

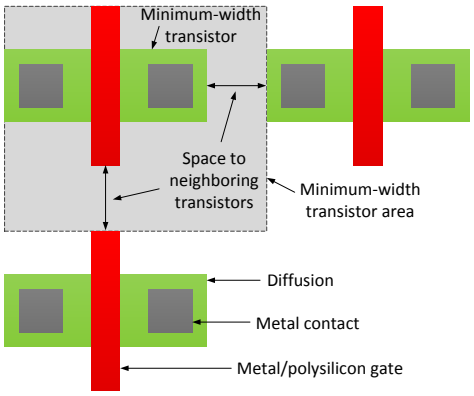


Fig. 9: Minimum-width transistor area model.

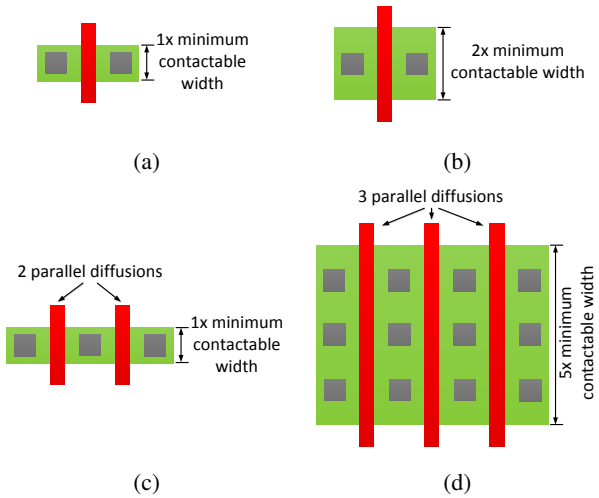


Fig. 10: (a) A minimum drive-strength transistor. (b) A 2× minimum drive-strength transistor obtained by diffusion widening. (c) A 2× minimum drive-strength transistor obtained by parallel diffusion regions. (d) A 15× minimum drive-strength transistor with square layout. Note: Although not shown in the figure for simplicity, parallel diffusions must be connected together.

technique frequently used in FPGA research [1], [12]. In this model, layout area is expressed in units of *minimum-width transistor areas*. A *minimum-width transistor* is defined as the smallest possible contactable transistor for a specific process technology and one *minimum-width transistor area* is the area of this transistor plus the spacing to neighboring transistors as shown in Fig. 9.

A transistor's drive-strength can be increased by either widening its diffusion region (Fig. 10b) or by adding parallel diffusion regions (Fig. 10c). The widely-used area model of [1] estimates the layout area of a transistor with drive-strength x, in units of minimum-width transistor areas, with (1) and calculates the area of an FPGA subcircuit by simply summing the areas of all the transistors in that subcircuit.

$$Area(x) = 0.5 + 0.5x \tag{1}$$

However, Fig. 11 shows that (1) over-predicts transistor area by as much as 143% compared to our manual layouts with TSMC's 65nm layout rules (which were the most advanced layout rules to which we had access). In [12], the constants in (1) were adjusted to match more advanced process rules but its area estimates for our 65nm layouts are still inaccurate (Fig. 11). Therefore, COFFE uses a new version of the minimum-width transistor area model whose accuracy is improved in two ways. First, we assume reasonably square layouts. To obtain (1), [1] averages the layout areas that result from either widening the diffusion region or adding parallel diffusion regions to increase drive-strength. For large transistors, however, both approaches yield layouts with very high aspect-ratios. We found that smaller area can be obtained by keeping a reasonably square transistor layout, which is accomplished by combining both diffusion widening and parallel diffusion regions to increase a transistor's drive-strength as in Fig. 10d. Therefore, our manual layouts in Fig. 11 use square layouts.

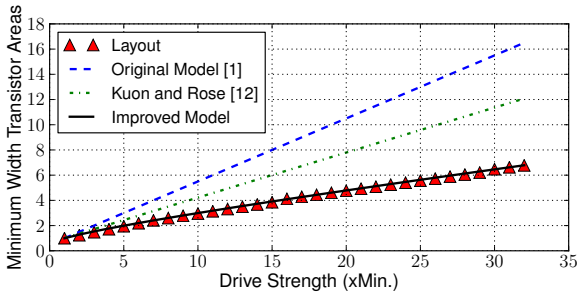


Fig. 11: Transistor area prediction accuracy of original (1) and improved (2) area models against TSMC 65nm layouts.

Second, we develop a new transistor area equation tailored towards more advanced process technologies by using a least-square fit of our 65nm layout areas versus drive-strengths to obtain area as a function of drive-strength.

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

Fig. 11 shows that (2) predicts transistor area with much more accuracy than prior models. We make two further enhancements to the model to better estimate the layout density of different structures. The area model described thus far does not account for the fact that in a design with both NMOS and PMOS transistors, extra spacing is required for N-wells. It would be pessimistic to assume that each PMOS transistor is in a separate well as the amount of N-well spacing required can be reduced by placing multiple PMOS transistors in the same well. Although it is difficult to predict how much well sharing is possible in a given layout, our sample layouts suggest that well sharing can reduce the *per-transistor* well spacing required by approximately 75%. With this estimate, we derive the following equation to calculate the area of transistors requiring N-well spacing.

$$Area(x) = 0.518 + 0.127x + 0.428\sqrt{x} \tag{3}$$

COFFE calculates the area of NMOS pass-transistors with (2) and the area of CMOS transistors (e.g. inverters) with (3). We find that accounting for N-well spacing increases our tile area estimates by ~2% for a pass-transistor based FPGA.

Finally, despite the fact that 6 small transistors are required per SRAM cell, COFFE uses an area of 4 minimum-width transistors because a denser, more optimized layout is typical for such a frequently used cell.

V. Wire Load Modeling

Past FPGA transistor sizing efforts have often only accounted for the loading effects of long wires such as the

routing wires or the cluster local interconnect wires. In reality, an FPGA contains much more metal wiring. Ignoring this extra metal is increasingly problematic as the impact of wires is becoming ever more important with shrinking feature sizes [21]. Accordingly, COFFE models all wire loading even including the relatively short metal connecting two transistors inside a multiplexer.

COFFE estimates wire lengths based on area estimates obtained with the model of Section IV along with the following set of general layout assumptions. The layout of a sub-block (e.g. a MUX, a BLE, a logic cluster, etc.) is assumed to be square such that its width is equal to its height. The length of a wire that broadcasts a signal across a sub-block is equal to the width (which equals the height) of that sub-block. The length of a point-to-point wire between two sub-blocks is equal to $1/4$ the sum of the width of both sub-blocks. For example, cluster local interconnect wires are broadcast wires so they span the height of a logic cluster. Wires that connect two inverters together inside a buffer are point-to-point wires; they span $1/4$ the width of each inverter.

The resistance and capacitance of a wire are obtained from its length estimate as well as its metal layer. COFFE implements most wires in the lowest metal layer, with the exception of routing wires, which are placed in a higher metal layer as they benefit from its lower resistance. With the resistance and capacitance values of a wire, COFFE includes its equivalent π-model in the generated SPICE netlists.

VI. Transistor Sizing Algorithm

When transistors are treated as linear resistances and capacitances, the transistor sizing problem can be formulated as a convex optimization problem [2]. Such a formulation has the highly useful property that there is only one minimum: the global minimum. Past transistor sizing algorithms have exploited this fact by either making a series of local optimizations in hopes of eventually reaching the global minimum [2], [12] or by making use of mathematical programming techniques [5]–[7], [13]. In Section III, we showed that it is very difficult to obtain linear models of transistors that are sufficiently accurate for the fine-grained transistor-level design of FPGA circuitry in advanced process nodes. Instead, we chose to use HSPICE simulations to measure delay, which produces more accurate delay estimates, but also makes the shape of the optimization space more ambiguous. Therefore, COFFE takes a more exhaustive approach and searches for a minimal cost solution by simulating many possible transistor sizing combinations over a range of transistor sizes. Exhaustively searching the entire optimization space in this way would lead to prohibitively long runtimes because there are ~80 unique transistors to size in one FPGA tile and sweeping each transistor over ~10 sizes would require 10^{80} HSPICE simulations. COFFE uses two techniques to confront this problem: *divide-and-conquer* and *inverter rise-fall balancing*.

A. Divide-and-Conquer

COFFE reduces the transistor sizing combinations to examine by sizing loosely coupled subcircuits individually. This divide-and-conquer approach reduces the search space but requires iteration to account for changes in loading. More specifically, since subcircuits are usually loaded by other subcircuits, changing the transistor sizes of one subcircuit will change the load on another. Because of this, COFFE performs multiple *FPGA sizing iterations* in which it sizes each subcircuit once with the loading coming from the last sizing of the other subcircuits. FPGA sizing iterations are performed until no reduction in cost is achieved (implying loading has stabilized) or until a maximum number of iterations have been completed. In our experience, COFFE finds a transistor sizing solution after 2-4 iterations.

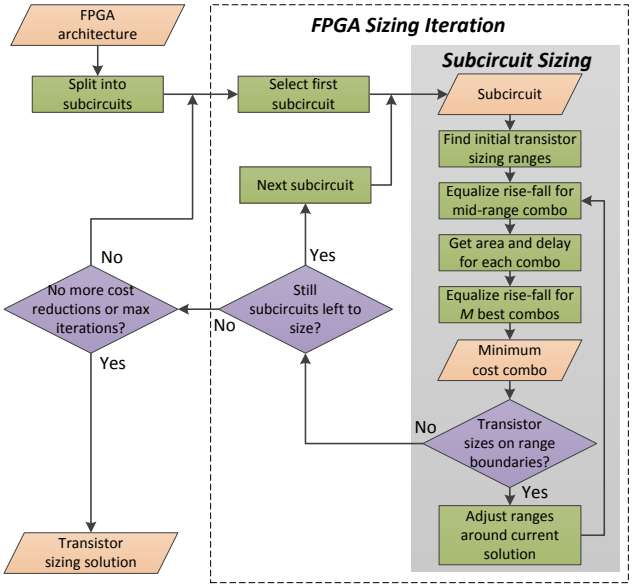


Fig. 12: COFFE's transistor sizing algorithm.

Sizing a subcircuit proceeds as follows. Based on the current sizes of transistors in the subcircuit, COFFE selects initial transistor sizing ranges that place the current sizes near the center. Then, for each sizing combination within these ranges, the area of the subcircuit is calculated with the model of Section IV, wire loading is determined with the model of Section V and delay is measured with HSPICE. COFFE can be configured to choose transistor sizes that minimize either the global cost or the local cost. The global cost is some product of total tile area and *representative* path delay (as in [12]) while the local cost is a product of this particular subcircuit's area and delay (as in [19]). Once the best cost sizing combination has been selected, COFFE checks if the solution is on the boundaries of the initial sizing ranges. If it is, we may not have explored a large enough size range, so the sizing ranges are adjusted around the current solution and the process is repeated until a solution that is contained entirely within the ranges is found. Fig. 12 shows the flow of COFFE's transistor sizing algorithm.

B. Inverter Rise-Fall Balancing

COFFE further reduces the number of transistor sizing combinations evaluated by using pre-determined P/N ratios to size the NMOS and PMOS transistors of inverters as a unit instead of as individual transistors. As shown in Fig. 12, the initial P/N ratios of inverters in a subcircuit are obtained by equalizing their rise and fall times for a mid-range transistor sizing combination. These P/N ratios are used to calculate area and measure delay for all transistor sizing combinations. Since the rise and fall times will not remain perfectly balanced as we evaluate different transistor sizing combinations, COFFE

TABLE III: Architecture parameters used for wire load experiments.

Parameter	Value	Parameter	Value
K	6	F_S	3
N	10	Fc_{local}	0.5
I	40	R_{fb}	"on" for LUT-input C
Fc_{in}	0.2		"off" for all other LUT inputs
Fc_{out}	0.025	R_{sel}	LUT & BLE input
W	320	O_{fb}	1
L	4	O_r	2

uses the *average* of rise and fall times *in this phase* because we will later balance the rise and fall time and, for small perturbations, this re-balancing makes the *worst* of the rise and fall delays close to this *average*. COFFE re-balances the rise and fall times on a user-specified M number of top-ranked transistor sizing combinations before selecting its final best transistor sizing solution as this re-balancing may re-order the final ranking. Thus, COFFE's final transistor sizing solution always has balanced inverter rise and fall times and we use the *maximum* of the rise and fall delays as the *final delay*.

With *divide-and-conquer* and *inverter rise-fall balancing*, we reduce the number of transistor sizing combinations to examine from ~10^{80} to the much more tractable number of ~$3 \times 12 \times 10^4$. That is, for ~12 subcircuits containing ~4 sizeable items (transistors or inverters), we try ~10 possible sizes per sizeable item. This is done ~3 times to account for changes in loading (i.e. an FPGA sizing iteration). Total runtime is ~4h for $M = 1$ or ~10h for $M = 5$ on a single Intel Xeon E5-1620 3.6GHz processor core. This runtime is of the same rough magnitude as [12], however we cannot make detailed quality comparisons as the CAD tool of [12] is not available.

VII. Impact of Improved Wire Load Modeling

To examine the impact of improved wire load modeling on the area and delay of an FPGA, we use COFFE to perform transistor sizing under different wire loading scenarios. The architecture parameters used for these experiments are shown in Table III and were selected based on [19]. We use PTM 22nm HP predictive SPICE models [20] and we extract wire resistance and capacitance per unit length from ITRS 2011 [22]. Pass-transistor gate voltages are boosted 200mV above the nominal V_{DD} of $0.8V$ as this was shown to be a good choice in [19]. Finally, we set COFFE's optimization objective to minimize the product of tile area and *representative* path delay and we re-balance the rise and fall times of the 5 top-ranked transistor sizing combinations ($M = 5$).

We begin by sizing transistors without including the effects of any wires. The resulting tile area and representative path delay are shown in the first row of Table IV. Then, we gradually add groups of wires to our FPGA, re-sizing its transistors after every addition. As shown in Table IV, each time we add wires, we observe an increase in delay as well as an increase in tile area because COFFE chooses larger transistor sizes in an effort to cope with the extra wire loading.

Table IV clearly shows that it is important to account for the effects of more than just the routing wires. In fact, 24% of the delay comes from two groups of wires that have often been overlooked in prior academic work: logic-to-routing wires and smaller wires like those inside MUXes and LUTs (which are included in the *All wires* row of Table IV). The logic-to-routing wires are those that connect specific routing tracks

TABLE IV: Impact of wire loading.

Wire load	Tile area (μm^2)	Delay (ps)
No wires	836	58
Routing only	899	79
Routing & cluster local interconnect	905	85
Routing, local interc. & logic-to-routing[a]	919	98
All wires	938	112

[a]We use an input track-access span of 0.5 and an output track-access span of 0.25 for logic-to-routing wires in this section. See Section VIII.

to cluster inputs (through connection block MUXes) as well as cluster outputs to specific routing tracks (through switch block MUXes) and they can span a significant fraction of a tile. We study the impact of the lengths of these wires in more detail in Section VIII.

VIII. Architecture Study: Track-Access Locality

In the previous section, we showed that wire loading at the logic-to-routing interface has a considerable impact on delay. Prior academic work has implicitly assumed that logic cluster pins can access *all* the routing tracks in an adjacent channel but has not considered the large logic-to-routing wire loading that this creates. It is possible to reduce this wire load by imposing limits on the lengths of logic-to-routing wires. We refer to this concept as *track-access locality* and we define *track-access span* as the portion of a routing channel that can be accessed by a logic cluster input or output. A large span implies little locality and vice-versa. Fig. 13 illustrates this concept for logic cluster outputs. In the figure, *output A* can only reach half of the routing tracks in a channel (the 50% physically close to it) while *output B* can reach all of them. *Output A* has a track-access span of 1/2; *output B* has a track-access span of 1. Clearly, *output B* has twice as much wire load as *output A*. Thus, *output A* is faster than *output B*. Fig. 14 illustrates the same concept as it applies to logic cluster inputs. The wire loading associated with cluster inputs comes from the wires required to connect routing tracks to the connection block multiplexers. This wire loading is seen by the routing wire drivers and will tend to slow down the general routing tracks. Note that, as shown in Fig. 14, COFFE does not include the *track buffers* that have often been used in academic work [1] because they are difficult to lay out and are not used in modern commercial architectures.

We use COFFE to size the transistors of the FPGA architecture described in Table III for different degrees of track-access locality. Table V shows the effect of cluster output locality on tile area and *representative* path delay while Table VI shows results for cluster input locality. The results suggest that reducing the input track-access span can lead to a large reduction in loading (~17% delay reduction for a span of 0.25). The effect is lesser for cluster outputs but we still observe a small reduction in overall area-delay product. Although track-access locality seems beneficial from a delay perspective, it could have a negative impact on routability since increasing locality could reduce the interconnect flexibility. It follows that the ideal track-access span will likely also depend on the values of Fc_{in} and Fc_{out}. For example, for our $Fc_{in} = 0.2$ and $Fc_{out} = 0.025$ architecture, cluster outputs may be better suited for high locality given the fact that they connect to relatively few routing multiplexers due to a low Fc_{out} value.

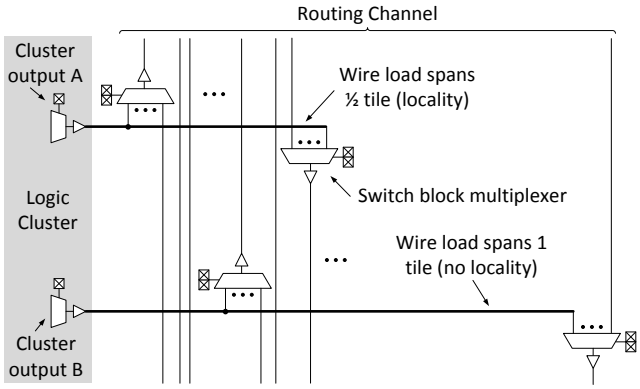


Fig. 13: Cluster output wire load for different locality.

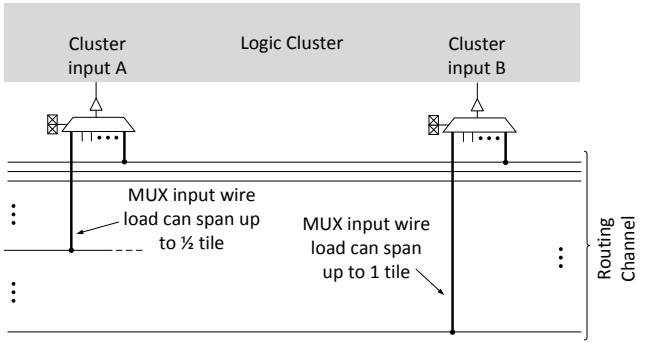


Fig. 14: Cluster input wire load for different locality.

A detailed analysis of these tradeoffs was not performed in this work but merits future research. When used with an architecture exploration tool such as VPR, COFFE enables a thorough evaluation of such architectural issues which combine changes in connectivity, loading and transistor sizing.

IX. Conclusion

We presented COFFE, a new fully automated transistor sizing tool for FPGAs. We showed that for fine-grained transistor-level design in advanced process nodes, modeling transistors as linear resistances and capacitances as in previous FPGA transistor sizing tools is highly inaccurate. For that reason, COFFE maintains all circuit non-linearities by relying exclusively on HSPICE simulations to measure delay. COFFE estimates area with a version of the minimum-width transistor area model to which we've made a number of improvements to enhance its accuracy in advanced process nodes. We showed that only accounting for the loading effects of long wires as has often been done in prior work can lead to delay under-predictions of 24%. To ensure realistic transistor sizing, COFFE automatically models *all* transistor and wire loads. These models have an important architectural impact: they favor larger transistors in FPGA LUTs and MUXes.

COFFE can size the transistors of an entire FPGA tile in ~10 hours, which is a task that would normally take months of manual effort. We illustrate COFFE's use by investigating a new architectural question concerning the wire loading at the interface between routing channels and logic clusters. We find that, at a possible cost in routability, restricting the portion of a routing channel that can be accessed by a logic cluster input can reduce delay by up to 17%.

TABLE V: Effect of cluster output track-access locality on area and delay. Input track-access span is set to 0.5.

Cluster Output Track-Access Span	Tile Area (μm^2)	Delay (ps)	Area-Delay Product
1.00	959	113	1.08
0.75	934	115	1.07
0.50	930	114	1.06
0.25	938	112	1.05

TABLE VI: Effect of cluster input track-access locality on area and delay. Output track-access span is set to 0.25.

Cluster Input Track-Access Span	Tile Area (μm^2)	Delay (ps)	Area-Delay Product
1.00	952	127	1.21
0.75	953	117	1.11
0.50	938	112	1.05
0.25	955	105	1.00

Acknowledgement

The authors thank David Lewis for insightful discussions, NSERC and Altera for funding and CMC for CAD tools.

References

[1] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer, 1999.
[2] J. P. Fishburn and A. E. Dunlop, "TILOS: A Posynomial Programming Approach to Transistor Sizing," in *ICCAD 1985*, pp. 326–328.
[3] W. C. Elmore, "The Transient Response of Damped Linear Networks with Particular Regard to Wideband Amplifiers," *Journal of Applied Physics*, pp. 55–63, 1948.
[4] J. Rubinstein, P. Penfield and M. Horowitz, "Signal Delay in RC Tree Networks," *TCAD*, pp. 202–211, July 1983.
[5] S. Sapatnekar et al., "An Exact Solution to the Transistor Sizing Problem for CMOS Circuits Using Convex Optimization," *TCAD*, pp. 1621–1634, 1993.
[6] C.-P. Chen, C. Chu, and D. Wong, "Fast and Exact Simultaneous Gate and Wire Sizing by Lagrangian Relaxation," *TCAD*, pp. 1014–1025, 1999.
[7] V. Sundararajan, S. Sapatnekar, and K. Parhi, "Fast and Exact Transistor Sizing Based on Iterative Relaxation," *TCAD*, pp. 568–581, 2002.
[8] J. K. Ousterhout, "Switch-Level Delay Models for Digital MOS VLSI," in *DAC 1984*, pp. 542–548.
[9] K. Kasamsetty, M. Ketkar, and S. Sapatnekar, "A New Class of Convex Functions for Delay Modeling and its Application to the Transistor Sizing Problem," *TCAD*, pp. 779–788, 2000.
[10] Conn, A. R. et al., "JiffyTune: Circuit Optimization Using Time-Domain Sensitivities," *TCAD*, pp. 1292–1309, 1998.
[11] ——, "Gradient-Based Optimization of Custom Circuits Using a Static-Timing Formulation," in *DAC 1999*, pp. 452–459.
[12] I. Kuon and J. Rose, "Exploring Area and Delay Tradeoffs in FPGAs With Architecture and Automated Transistor Design," *TVLSI*, pp. 71–84, 2011.
[13] A. Smith, G. Constantinides, and P. Y. K. Cheung, "FPGA Architecture Optimization Using Geometric Programming," *TCAD*, pp. 1163–1176, 2010.
[14] G. Lemieux and D. Lewis, "Using Sparse Crossbars within LUT Clusters," in *FPGA 2001*, pp. 59–68.
[15] D. Lewis et al., "The Stratix™ Routing and Logic Architecture," in *FPGA 2003*, pp. 12–20.
[16] G. Lemieux et al., "Directional and Single-Driver Wires in FPGA Interconnect," in *FPT 2004*, pp. 41–48.
[17] D. Lewis et al., "The Stratix II™ Logic and Routing Architecture," in *FPGA 2005*, pp. 14–20.
[18] C. Chen et al., "Efficient FPGAs using Nanoelectromechanical Relays," in *FPGA 2010*, pp. 273–282.
[19] C. Chiasson and V. Betz, "Should FPGAs Abandon the Pass-Gate?" in *FPL 2013*.
[20] Predictive Technology Model (PTM), *http://ptm.asu.edu/*, 2012.
[21] R. Ho, K. Mai, and M. Horowitz, "The Future of Wires," *Proceedings of the IEEE*, pp. 490–504, 2001.
[22] ITRS, *Interconnect Chapter*, 2011.