Concurrently Optimizing FPGA Architecture Parameters and Transistor Sizing: Implications for FPGA Design

Alastair M. Smith [1], George A. Constantinides [1], Steven J. E. Wilton [2], Peter Y. K. Cheung [1]

[1] *Dept. of Electrical and Electronic Engineering*
Imperial College London, UK
{alastair.smith, g.constantinides, p.cheung}@imperial.ac.uk

[2] *Dept. of Electrical and Computer Engineering,*
University of British Columbia, Vancouver, Canada
stevew@ece.ubc.ca

***Abstract*—This paper presents a method that combines high-level and low-level architecture parameter exploration. The paper builds on an increasing body of work concerned with modeling reconfigurable architectures, and presents a full area and delay model of an FPGA. The optimization of this model is based on the use of Geometric Programming, and allows high-level architecture parameter selection and transistor sizing to be done concurrently. We use the framework to demonstrate that concurrent optimization of both high and low-level parameters can lead to significantly different architectural conclusions.**

I. Introduction

Recent years have seen considerable evolution in the architecture of FPGAs. Each generation of commercial FPGAs contains new or refined routing, logic, memory and embedded block structures. These architectural enhancements are the result of time consuming and expensive experiments, in which FPGA architects (in both industry and academia) use existing or new CAD tools to map benchmark circuits to the architectures under investigation [1], [2]. Recent work, however, has suggested that this experimental approach can be supplemented by analytical techniques, in which FPGA architectures are modeled by relatively simple equations, and powerful optimization tools are used to "prune" the architecture space, allowing the FPGA architect to investigate a much wider range of architectures than previously possible [3], [4], [5], [6].

One of the advantages of an analytical approach is that values for many architectural parameters can be optimized at once. This is different than an experimental approach, where typically one parameter is swept at a time. In [3], it was shown that several parameters describing the routing structure of an FPGA can be optimized simultaneously by creating analytical equations describing the impact of these parameters on the area of an FPGA, and using a Geometric Programming (GP) framework to determine values for these parameters. This led to an area reduction of 6% compared to architectures obtained using experimental techniques.

In this paper, we take this concept much further by considering the optimization of both architecture and transistor sizing simultaneously. This allows us to optimize both the area and the critical path delay (speed) of the FPGA. In addition, we include the optimization of parameters describing the logic architecture on top of those that describe the routing architecture. We will show that optimizing this much wider range of parameters simultaneously can result in more efficient FPGAs, and that this optimization can be performed much faster than the traditional experimental methods.

Specifically the contributions of this work can be summarized as follows:

- A framework allowing, for the first time, concurrent optimization of both high-level (architectural) and low-level (transistor sizing) parameters.
- Formulation of an area-delay model of FPGA fabrics as a geometric program.
- Demonstration that the concurrent optimization of high- and low-level parameters leads to significantly different architectural conclusions compared to a traditional flow. In particular, that cluster size should increase rather than decrease as delay becomes more important than area, leading to delay improvements of 4% when averaged over a number of benchmark circuits.

II. Related Work

The goal of the work we present is similar to that of [7], in which a tool was presented that optimizes the electrical design of FPGA architectures given a set of high-level architecture parameters. [7] presents a heuristic for optimizing the electrical design (transistor sizing) based on an iterative procedure involving successive placement and routing of benchmarks onto FPGA architectures. The transistor sizing is dependent on the specified architecture. In our work we remove the need for this iterative refinement, building on FPGA modeling techniques and introducing a geometric program to perform

the step of transistor sizing concurrently to a number of architectural parameters.

In [8], a model was developed for the purpose of transistor sizing of interconnect buffers and a heuristic method was developed to optimize the sizing. However, it has been shown in [9] that transistor sizing techniques, including those presented in [8] can be solved efficiently and to optimality through use of geometric programming. Our previous work in [3] made the observation that high-level FPGA models also fit into the GP framework. In this paper, we complete the link between the high-level architectural description and a low-level transistor level description, combining both into a single framework, expressing a complete area and delay model of FPGA fabrics.

A. Geometric Programming

A *geometric program* (GP) is a constrained optimization problem of the following form:

$$\begin{aligned} \text{Minimize :} \quad & f_0(x) \\ \text{Subject to :} \quad & \\ & f_i(x) \leq 1, \text{for } i = 1, 2, ..., m \\ & g_i(x) = 1, \text{for } i = 1, 2, ..., l \end{aligned}$$

where x is a strictly positive n-vector of real values, and the functions f_i and g_i have special mathematical forms, known as *posynomials* and *monomials*, respectively.

A monomial is a function

$$g(x) = cx_1^{a_1}x_2^{a_2}\dots x_n^{a_n}$$

where the coefficient c must be strictly positive. A posynomial is simply a sum of a finite number of monomials. We note that the restriction to strictly positive leading coefficients in a monomial is a serious one and explicitly disallows many problems, including general *polynomial* optimization rather than *posynomial* optimization.

As an example, later in our paper we use the monomial cost function $T_{total}^{z}A_{total}^{1-z}$, where A_{total} and D_{total} represent the variables area and delay. In this case z must be a constant.

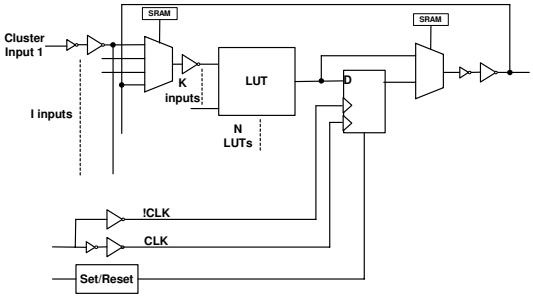
Geometric programming has been used extensively for many circuit design problems. These include transistor sizing, wire sizing and robust design in the presence of statistical variations. We refer the reader to [9] for an extensive review of GP in the context of circuit design. One of the particularly attractive features of GP is that it has excellent tractability. For example, recent methods for solving the circuit sizing problem allow million transistor designs to be optimized in around 40 minutes [10].

III. Architectural Framework

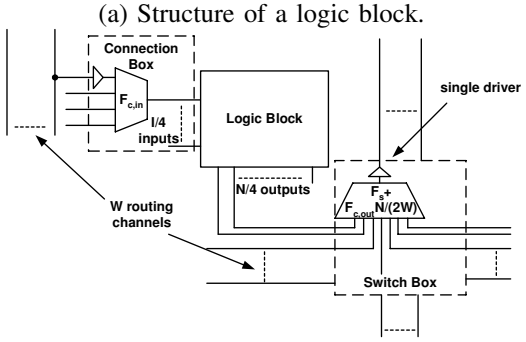
We assume an island-style FPGA in which an array of blocks is connected using tracks organized in horizontal and vertical channels with single-driver routing, as represented by VPR 5.0 [1]. The logic blocks in the architecture consist of K-input lookup tables (LUTs) packed into tightly connected configurable logic blocks (CLBs), each with N LUTs and with I external inputs, as shown in Figure 1(a). A K-input LUT is implemented using a K-level pass transistor multiplexer tree.

TABLE I
Model Parameters

Architectural Parameters:	
K	Number of inputs per lookup table
N	Number of lookup tables per logic block
I	Number of inputs per logic block
$F_{c,in}$	Number of tracks that connect to each logic input pin
$F_{c,out}$	Number of tracks each logic block can connect to
F_s	Number of track end-points that connect to each track driver
Circuit Parameters:	
p	Rent parameter of a given circuit
n_2	Number of 2-LUTs in a given circuit
d_2	depth of circuit netlist in number of 2-LUTs



(a) Structure of a logic block.



(b) Routing Fabric of an FPGA.

Fig. 1. Detailed view of FPGA architecture.

The routing architecture, shown in Figure 1(b), consists of connection boxes and switch boxes. The routing blocks in the architecture can be described by three parameters: the number of tracks that can connect to each logic block input, $F_{c,in}$; the number of tracks that each logic block output can connect to, $F_{c,out}$; and the number of track end-points that connect to each channel driver, F_s.

All multiplexers in the FPGA except for the LUT multiplexer are implemented using a two-stage pass transistor structure. This type of multiplexer, including a transistor-level implementation is shown in Figure 2. In each case, the multiplexers are used to configure signal routing paths around the device, and thus are connected to SRAM configuration memory.

IV. Delay Model

GP has previously been shown to be capable of optimizing transistor sizing for delay [11]. We employ this type of delay optimization technique here to model the combination of

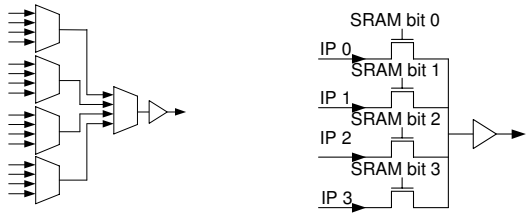


Fig. 2. Multiplexing schemes in VPR 5.0: (a) a two-level multiplexing scheme for a 16:1 multiplexer, (b) a pass transistor-based 4:1 multiplexer.

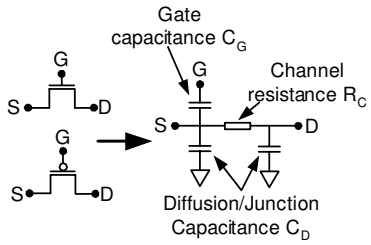


Fig. 3. RC delay model for a MOSFET.

CMOS and pass transistor structures present in FPGA devices. Consider the delay of the critical signal path through a circuit implemented on an FPGA. The critical path will typically pass through a number of LUTs, CLBs, switch boxes, connection boxes and CLB feedback paths. We model the critical path delay as a weighted sum of these; the value of each weight depends on the expected number of each of these components along the path. We use the formula in (1), where each term is as described below. D_k represents the depth of the netlist when implemented in K-input LUTs: we assume the critical path is through the deepest part of the netlist. D_c represents the number of CLBs through which the critical path travels. $D_i = D_k - D_c$ represents the number of internal feedback connections through which the critical path traverses. Finally, D_r represents the number of switchboxes through which each external connection on the critical path propagates.

$$\begin{aligned} \mathrm{T_{total}} = & T_{reg\ to\ OMUX} + D_i T_{LUT\ F/B\ path} \\ & + (D_k - 1) T_{LUT\ delay} + D_c T_{O/P\ CB\ delay} \\ & + D_c D_r T_{SB\ delay} + D_c T_{I/P\ CB\ delay} \\ & + D_c T_{input\ MUX\ delay} + T_{LUT\ to\ reg\ delay} \end{aligned} \quad (1)$$

Each of the scaling factors can be determined through use of analytically derived mathematical expressions. We employ the methods from [12] to estimate the netlist depths D_k, D_c and D_i. The average wirelength is assumed to determine the number of switchboxes through which each external CLB connection travels (D_r). We note that the terms involving N and K cannot easily be expressed in a form amenable to GP, hence in our experimentation it is necessary to perform a sweep over these two logic parameter values.

Each transistor in the circuit can be represented as an RC network as in Figure 3. The resistance and capacitances can be derived from SPICE models of MOSFET devices. In this work, we derive the values using 65nm predictive technology models [13]. We use a standard model for the resistance and capacitance values within a MOSFET [9]: each resistance value for a transistor in the architecture takes the form (2) and each capacitance takes the form (3) or (4). R_C represents the channel resistance of a transistor, and C_G and C_D represent the gate and diffusion capacitances respectively. In each of these equations S_i refers to the width of the transistor assuming all transistors have minimum length. The nominal values R_{nom} and $C_{nom,x}$ are dependent on the type of transistor (nMOS/pMOS), the process technology and in the case of the capacitance, whether it is the nominal gate or diffusion capacitance.

$$R_C = \frac{R_{nom}}{S_i} \quad (2)$$

$$C_G = C_{nom,G} S_i \quad (3)$$

$$C_D = C_{nom,D} S_i \quad (4)$$

To evaluate the delays through each of the paths in (1), the Elmore delay model is employed [14]. The Elmore model is used to represent delay in networks of RC trees and has previously been shown to model delay in FPGA routing pass transistor networks [8]. As shown in (5), Elmore delay is calculated through the evaluation of the sum of each segment delay from the signal source to its sink, where the delay of each segment is the sum of the resistance along the path multiplied by the capacitance of that segment.

$$T_{elmore} = \sum_{paths\ i\ source\ to\ sink} C_i R_{source\ to\ C_i} \quad (5)$$

In order to derive an expression for the total delay, we must break down each of the delay terms T_x in (1) into its constituent paths. Each path starts from VDD or GND and terminates at a transistor gate input. This leads to a number of paths, given in Figure 4(a-j).

The delay $T_{reg\ to\ OMUX}$ represents the delay from the register producing the critical path signal through the MUX selecting whether the LUT output is registered or not, and through the two-level buffer. This is given in Figure 4(c).

The delay $T_{LUT\ F/B\ path}$ represents the delay from the BLE output buffer through the pass transistor-based MUX on the LUT input to its buffer, as shown in Figure 4(b). The Elmore delay path in this case terminates at the LUT driver input gate.

The delay $T_{LUT\ delay}$ represents the delay from the LUT driver through all levels of the multiplexer implementing the LUT, the 2:1 select MUX and to the LUT output driver. The delay is thus the sum of the paths these paths, given in Figure 4(h), and Figure 4(e) respectively.

The delay $T_{O/P\ CB\ delay}$ represents the delay from the BLE output buffer, through the switchbox multiplexer to its first inverting buffer, as represented by Figure 4(b), where in

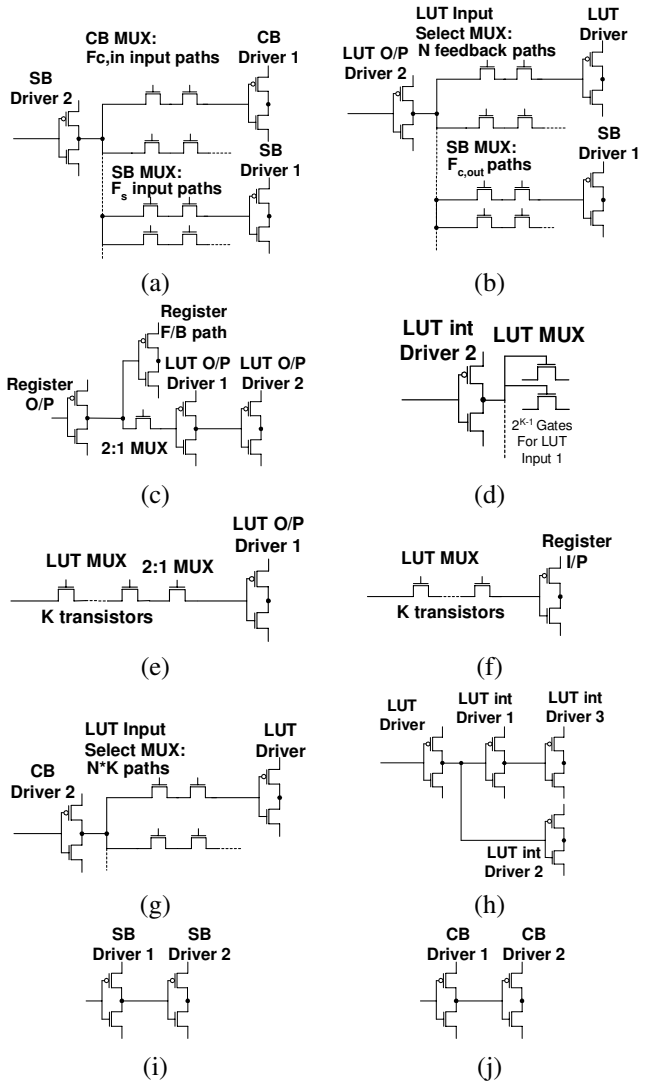


Fig. 4. RC delay models for circuit path.

this case the Elmore delay is through the path to the switchbox driver.

$T_{SB\ delay}$ represents the routing path signal between switchboxes. This is represented by the sum of the driver delay and delay through the two-level switchbox multiplexer, as in Figure 4(a) and (i).

The delay $T_{I/P\ CB\ delay}$ represents the path through the switchbox to the connection box where the routed signal is consumed. This is the sum of the initial driver delay, the delay through the connection box multiplexer and through the two inverting drivers in the connection box. These are shown in Figure 4(i), (a) and (j) respectively.

$T_{input\ MUX\ delay}$ is the delay from the connection box output driver through the LUT input select multiplexer to the LUT input driver.

Finally, $T_{LUT\ to\ reg\ delay}$ is the delay through the multiplexer implementing the LUT to the register input where the critical path terminates, as in Figure 4(f).

Since there are two paths to consider for each driving gate: the charge and discharge path from the driving gate, the terms representing delay are given as inequalities. Due to space limitations it is not possible to list all of the Elmore delay equations. However, as an illustrative example, the inequality for the charge path through the nMOS transistor which represents $T_{O/P\ CB\ delay}$ is given by (6), represented in Fig. 4(b).

$$\begin{aligned} T_{O/P\ CB\ delay} \geq & R_{C,n,LOdrv2}(C_{D,n,LOdrv2} \\ & + NKC_{D,mux} + F_{c,out}C_{D,SB\_mux}) \\ & + (R_{C,LOdrv2} + R_{C,SB\_mux})2C_{D,SB\_mux} \\ & + (R_{C,LOdrv2} + 2R_{C,SB\_mux}) \\ & \times (C_{D,SB\_mux} + C_{G,SB\_driver}) \end{aligned} \quad (6)$$

V. Area Model

An area model of only the FPGA routing fabric was first presented in [3], and is summarized in Section V-B. In this work we extend the model in order to consider variable buffer sizing and include both the routing and logic architecture. The area model is based on the minimum width transistor sizing model employed in [2].

In our experiments in Section VII, we apply the model on a per benchmark basis. The size of the FPGA is assumed to be the smallest square that fits the benchmark circuit. Thus, the grid size $N_c = \lceil \sqrt{n_c} \rceil^2$, where n_c is the number of CLBs in the benchmark circuit and can be estimated using the formula in [4]. N_c is used to evaluate how much logic area is consumed.

The total area of an FPGA, A_{total}, can be represented as the sum of the routing area A_r and logic area A_l, as in (7). The routing resources surround the logic resources, hence the width of the grid of routing resources is one larger than the logic grid. Since I/O pins are on the edge of the grid, the routing resources at the edge of the grid are different to those in the center. $N_{s,e}$ and $N_{s,m}$ represent the number of switch boxes at the edge and middle of the FPGA respectively. We assume that the transistor sizing in both switch boxes and connection boxes to be the same across the grid. However, the sizes of the multiplexers vary according to the required number of inputs.

$$A_{total} = A_l + A_r \quad (7)$$

A. Logic Block Area

The area of the logic block is the sum of the area devoted to the following: the LUT, 2:1 multiplexer, register and output buffer combination; the LUT input select multiplexer; the set/reset logic; and the clock buffer. We assume the set/reset logic and clock buffer sizing to be constant regardless of the logic block architecture; we take the values from [2]. Similarly, the size of the register on the LUT output is assumed to be constant.

The LUT area is composed of the SRAM cells, the pass-transistor multiplexer cells and the internal drivers. The size of each pass transistor in the multiplexer ($S_{n,LM}$) is assumed to be the equivalent. Similarly, each input buffer scheme is assumed to have the same size transistors. The sum of these

buffer areas is given by B_{li}. This leads to (8) as an expression for the area consumed by a K-input LUT, where S_{SR} is the size of an SRAM cell and B_{li} is the size of the buffer driving each LUT input.

$$A_{lut} = \quad 2^K S_{SR} + K B_{li} + (2^{K+1} - 2) S_{n,LM} \tag{8}$$

The 2:1 multiplexer consists of a one level pass transistor multiplexer. This consumes area A_{21mux}, given by (9), where $S_{n,21mux}$ represents the size of each of the two pass transistors implementing the 2:1 multiplexer, and S_{SR} is the one bit configuration memory required.

$$A_{21mux} = S_{SR} + 2S_{n,21mux} \tag{9}$$

The output buffer combination is the sum of the transistor areas for the two inverters implementing the driver. The combination of these inverters consumes area:

$$B_{lo} = S_{n,LOdrv1} + S_{p,LOdrv1} + S_{n,LOdrv2} + S_{p,LOdrv2} \tag{10}$$

where the area of $S_{n,LOdrv*}$ represents the size of each nMOS transistor in the CMOS inverter and $S_{p,LOdrv*}$ represents the size of the pMOS transistor.

The LUT input select multiplexer is implemented with the two-level multiplexing scheme. An example of a 16:1 multiplexer using this scheme is shown in Figure 2(a). In such a scheme, the first and second level of multiplexers are balanced such that each stage has approximately the same size of multiplexer. Multiplexers can be efficiently implemented using pass transistors, as shown in Figure 2(b). This structure is taken from VPR 5.0, in which one-hot encoding of configuration bits is used. These observations lead to the expression for multiplexer area given in (11); S_n represents the size of the pass transistors and E is the number of inputs. An approximation for this area is given in (12).

Each of the input select multiplexers is fully connected; every input from the connection box and every output feedback path can reach any LUT input. This leads to the expression in (13) for the area devoted to each of these multiplexers, where $S_{n,ISmux}$ represents the size of the pass transistors implementing the input select multiplexer. Since there are $I+N$ inputs to the multiplexer, $E_{IS,tree}$ in (14) represents the number of pass transistors in the multiplexer tree and $E_{IS,tree}$ in (15) represents the number of SRAM bits.

Combining the above expressions for the constituent parts of the logic block leads to the expression in (16). The total area of the FPGA logic fabric is as in (17).

$$A_{mux} = S_n \left(E + \lfloor \sqrt{E} \rfloor \right) + S_{SR} \left(\left\lceil \frac{E}{\lfloor \sqrt{E} \rfloor} \right\rceil + \lfloor \sqrt{E} \rfloor \right) \tag{11}$$

$$\approx S_n \left(E + \sqrt{E} \right) + 2 S_{SR} \sqrt{E} \tag{12}$$

$$A_{ISmux} = E_{IS,tree} S_{n,ISmux} + E_{IS,RAM} S_{SR} \tag{13}$$

$$E_{IS,tree} = N + I + \lfloor \sqrt{N+I} \rfloor \tag{14}$$

$$E_{IS,RAM} = \lceil \sqrt{N+I} \rceil + \lfloor \sqrt{N+I} \rfloor \tag{15}$$

$$A_{LB} = N A_{LUT} + N A_{reg} + N A_{21mux} + K N A_{ISmux} + N B_{lo} + A_{clkB} + A_{rst} \tag{16}$$

$$A_l = N_c A_{LB} \tag{17}$$

B. Routing Area

We consider the amount of silicon area devoted to the routing fabric to consist of all switch box and connection box multiplexers, in addition to their output buffers and configuration memories. Routing area is thus dependent on the size of the grid of logic cells, the channel width, the transistor sizing and the size of the multiplexers used to connect signals to and from logic blocks and I/O pins.

An estimation of the multiplexer sizes in the switch and connection boxes is based on the observation that the expression for the area of a two level multiplexer in (11) can be approximated as in (12). The sizes of these multiplexers are dependent on the channel width of the device.

The model developed in [5] is used to estimate channel width. This model is shown in (18) for architectures with wires that span one logic block, where the nominal minimum channel width W_{min} is described by (19), and β, α_{in}, α_{out} and p_f are empirically derived constants. In (19), λ represents the average number of inputs used on each logic block and $\bar{R}$ represents the average point-to-point wirelength.

$$W = W_{min} + \frac{1}{\beta} \left(\frac{W_{min}}{F_s} \right) \left(\frac{W_{min}}{F_{c,in}} \right)^{\alpha_{in}} \left(\frac{W_{min}}{F_{c,out}} \right)^{\alpha_{out}} \tag{18}$$

$$W_{min} = p_f \frac{\lambda \bar{R}}{2} \tag{19}$$

The methods described by [6] are used to calculate the value of point-to-point wirelength for different logic parameters.

In the routing fabric there are two types of multiplexers: switch box multiplexers and connection box multiplexers. Using the approximation (12) leads to the expression of multiplexer area for the connection box given in (20), and (21) for the approximation of switchbox area. $F'_{c,out} = \frac{F_{c,out}}{W}$ represents the proportion of routing tracks that each logic block output connects to, and $F'_{c,in} = \frac{F_{c,in}}{W}$ represents the number of tracks that can connect to each logic block input. Combining these models leads to (22) for the routing area. Further details are available in [3].

$$S_{cb} = S_{n,cb} \left(W F'_{c,in} + \sqrt{W F'_{c,in}} \right) + 2 S_{SR} \sqrt{W F'_{c,in}} \tag{20}$$

$$S_{sb,m} = S_{n,sb} \left(\frac{N}{2} F'_{c,out} + F_s + \sqrt{\frac{N}{2} F'_{c,out} + F_s} \right) + 2 S_{SR} \sqrt{\frac{N}{2} F'_{c,out} + F_s} \tag{21}$$

$$A_r = I N_c \left(S_{cb} + B_{cb} \right) + 4 I_{io} \sqrt{N_c} \left(S_{cb,io} + B_{cb,io} \right) + 2 N_{s,m} W \left(S_{sb,m} + B_{sb,m} \right) + 6 N_{s,e} W \left(S_{sb,e} + B_{sb,e} \right) \tag{22}$$

$$2^K S_{SR} A_{lut}^{-1} + K B_{li} A_{lut}^{-1} + (2^{K+1} - 2) S_{n,LM} A_{lut}^{-1} \leq 1 \quad (26)$$
$$S_{SR} A_{21mux}^{-1} + 2 S_{n,21mux} S_{SR} A_{21mux}^{-1} \leq 1 \quad (27)$$
$$E_{IS,tree} S_{n,ISmux} A_{ISmux}^{-1} + E_{IS,RAM} S_{SR} A_{ISmux}^{-1} \leq 1 \quad (28)$$
$$N A_{LUT} A_{LB}^{-1} + N A_{reg} A_{LB}^{-1} + N A_{21mux} A_{LB}^{-1} + K N A_{ISmux} A_{LB}^{-1} + N B_{lo} A_{LB}^{-1} + A_{clkB} A_{LB}^{-1} + A_{rst} A_{LB}^{-1} \leq 1 \quad (29)$$
$$N_c A_{LB} A_l^{-1} = 1 \quad (30)$$
$$A_{total}^{-1} A_l + A_{total}^{-1} A_r \leq 1 \quad (31)$$

Fig. 5. Geometric program: logic area constraints.

VI. Geometric Programming formulation

In this section, we show that our model can be expressed in a form amenable to GP. It is necessary to express the model as monomial terms with equality to one, or posynomial terms less than or equal to one. We begin by considering the cost function, which takes the form of a monomial (23). By varying the exponent weight z it is possible to target, for example, delay only by setting $z = 1$, or an equal weighting by setting $z = 0.5$. This value must be a constant for each run of the GP.

$$\min : T_{total}^{z} A_{total}^{1-z} \quad (23)$$

The model presented in Section V is not in a form that is amenable to GP. The GP representation of the routing architecture model was presented in [3], thus we focus here on presenting the logic area constraints in the correct form. The area constraints in a standard form GP representation are given by (26)-(30). As an example of how the model maps to constraints, the area sum of logic and routing in (7) maps directly to the inequality constraint in (31).

The area of each buffer in the FPGA is the sum of its nMOS and pMOS transistors for each inverting stage, for example B_{lo} in (10). The transformation into posynomial form for buffer area is given by (24) and (25), where B_x represents the area of the buffer x.

$$B_x = \sum_{\text{all inverters in } x} S_n + S_p \quad (24)$$
$$\Longrightarrow \sum_{\text{all inverters in } x} S_n B_x^{-1} + S_p B_x^{-1} \leq 1 \quad (25)$$

The mapping of delay constraints is relatively straightforward, as they take posynomial form. An example of how delay constraints are represented in a GP are given in (32)-(41). The example shows the delay and related constraints between the inverters used in the switch box buffer. (32) and (33) represent the charge/discharge path of the first inverter through pMOS and nMOS transistors respectively, where $T_{SB\ inv1\ inv2}$ is the variable representing the delay. C_{G,SB_inv2} is the load capacitance of the second inverter gate, and is the sum of the two transistor gates used to make up the inverter - the sum is expressed in (34). The capacitance and resistance values required are given by (35)-(40), where $C_{nom,*}$ and $R_{nom,*}$ represent the nominal values for a minimum gate length transistor in a given technology. (41) is used to ensure the transistor size does not violate the smallest feature size possible in the process technology, where S_{TECH} is the constant representing the minimum feature size. This final constraint must be applied to all transistors in the GP.

$$T_{SB\ inv1\ inv2}^{-1} R_{C,n,SB\_inv1} C_{D,n,SB\_inv1} + T_{SB\ inv1\ inv2}^{-1} R_{C,n,SB\_inv1} C_{G,SB\_inv2} \leq 1 \quad (32)$$
$$T_{SB\ inv1\ inv2}^{-1} R_{C,p,SB\_inv1} C_{D,n,SB\_inv1} + T_{SB\ inv1\ inv2}^{-1} R_{C,n,SB\_inv1} C_{G,SB\_inv2} \leq 1 \quad (33)$$
$$C_{G,SB\_inv2}^{-1} C_{G,p,SB\_inv2} + C_{G,SB\_inv2}^{-1} C_{G,n,SB\_inv2} \leq 1 \quad (34)$$
$$R_{nom,nMOS} R_{C,n,SB\_inv1}^{-1} S_{n,SB\_inv1}^{-1} \leq 1 \quad (35)$$
$$R_{nom,pMOS} R_{C,p,SB\_inv1}^{-1} S_{p,SB\_inv1}^{-1} \leq 1 \quad (36)$$
$$C_{nom,D,nMOS} C_{D,n,SB\_inv1}^{-1} S_{n,SB\_inv1} \leq 1 \quad (37)$$
$$C_{nom,D,pMOS} C_{D,p,SB\_inv1}^{-1} S_{p,SB\_inv1} \leq 1 \quad (38)$$
$$C_{nom,G,nMOS} C_{G,n,SB\_inv2}^{-1} S_{n,SB\_inv2} \leq 1 \quad (39)$$
$$C_{nom,G,pMOS} C_{G,p,SB\_inv2}^{-1} S_{p,SB\_inv2} \leq 1 \quad (40)$$
$$S_{TECH} S_{n,SB\_inv1}^{-1} \leq 1 \quad (41)$$

Fig. 6. Geometric program: an example of delay constraints.

There are approximately 150 transistors in the design that need to be optimized. This equates to 150 transistor widths and 450 variables to represent the resistance and capacitances. In addition to the high-level parameters, the Geometric program has in excess of 600 variables. The Geometric program takes approximately 1 minute to run on a Quad Core Pentium 2.6GHz running windows XP. This is for each value of N and K: thus far it has not been possible to express constraints involving these two variables in posynomial form.

VII. Results

To demonstrate the power of the Geometric Programming framework developed, we performed experimentation with our tool. The tool has been developed within the CVX framework for Matlab [15]. Our tool allows the high and low-level architecture parameters to be evaluated within the same framework. This has previously not been possible, and past studies have employed a two-stage approach in which architecture parameters are fixed before transistor sizing. To demonstrate the impact of optimizing the high-level and low-level parameters concurrently, we modeled three different flows using our framework. Each experiment was performed on the 20 largest MCNC circuits and the geometric mean was taken.

Our tool requires that the logic parameters N and K are fixed for each run of the optimization tool. In order to find the optimal set of all parameters, it is necessary to sweep across a range of values of interest. Each run of our tool reports the

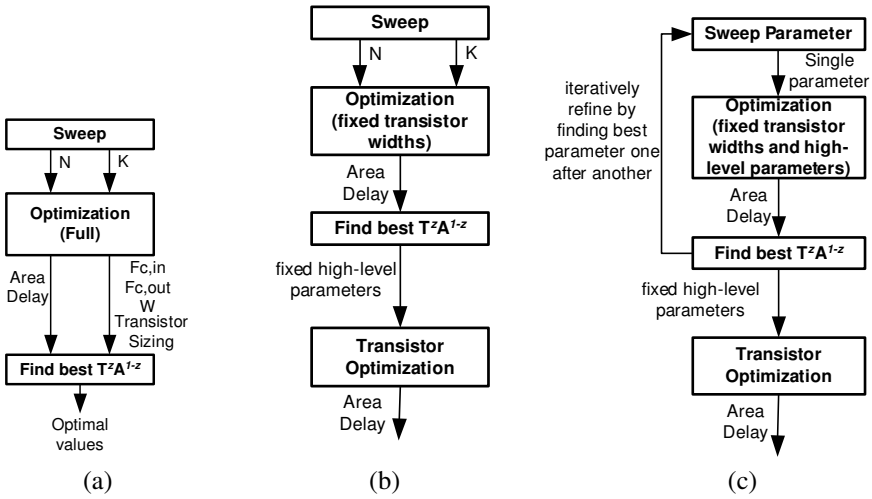


Fig. 7. The three flows used in our experimentation.

transistor sizing, switch box and connection flexibility as well as the value of the objective function and the total area and critical path delay. The best architecture is selected as the one which gives the best value of the objective function. This is the first experimental flow and is shown in Figure 7(a).

The second experiment also employs our framework, but models a two stage approach to optimization. The flow is shown in Figure 7(b). In the first stage, the GP optimization procedure is performed, but all transistors are fixed to be minimum width - this is done through the introduction of equality constraints in the GP. A full sweep across all combinations of N and K is performed to find the optimal architecture parameters under the minimum width condition. The values of $F_{c,out}$ and $F_{c,in}$ are determined from the results of the GP otimization. The objective function is reported for each N and K and the architecture parameters are selected for the device for the best value of the objective. After selecting the high-level parameters, the optimal parameters N, K, $F_{c,out}$ and $F_{c,in}$ chosen in the first stage are then fed into a new GP as constants, and the constraints on transistor widths are removed. The final delay, area and objective function are then reported and the transistor widths are determined.

The final experiment is a naïve approach in which each of K, N, $F_{c,out}$ and $F_{c,in}$ is chosen successively. The flow of this experiment is shown in Figure 7(c). We sweep across $K = 2 - 7$, determining the best LUT size for an abitrarily chosen value of N, $F_{c,out}$ and $F_{c,in}$. K is then fixed and the CLB size N is chosen from $N = 2-12$. The routing flexibility parameters are chosen similarly from a sweep of ten different values. Finally, the transistor sizing is determined given the best values found during the sweep procedure.

The exploration was performed on 20 MCNC circuits. During this exploration the exponent parameter z was varied to see how each approach performs depending on how the cost function is weighted. Figure 8(a) shows the geometric mean area in minimum width transistors when varying the exponent z in the objective function $T^z A^{1-z}$, and Figure 8(b) shows the critical path delay of each architecture. The single stage approach improves the delay when the cost function is weighted towards delay as an objective. The difference between this single stage approach and the dual stage approach (in which the transistor sizing is considered after high-level parameter selection) is around 4%. For the multistage heuristic, the difference from optimal is around 10% in the worst case. However, this gain in delay is not without a penalty - that of area, the cost of which is around 15% when compared or the dual-stage experiment and 23% for the multi-stage heuristic in the case where delay is most important.

In the case where the cost function is weighted towards area, the single stage optimization and two stage optimization give very similar results in terms of area. The difference in delay is similarly small. Unsurprisingly, the multi-stage heuristic performs worse for both metrics in this region - around 1% in area and 6% in delay compared to the best architecture.

These results are interesting from a design perspective: high-level architectural decisions can be made independently of transistor sizing with only a small effect on the metrics of interest. The reason for these two approaches being close together is that in the two-stage approach, every transistor is assumed to be minimum size. This assumption turns out to be close to optimal, as the buffer sizes are the most critical to delay and are the only transistors differing from minimum width in the resulting optimized architectures. This also explains why the single stage approach is better when delay takes on increased significance in the objective function.

Figure 8(c) shows how the LUT size K and cluster size N vary as the cost function is varied. The optimal values for each benchmark have been averaged. These results show that the single-stage and two-stage approaches head in different directions as delay takes more importance. The reason for this is that increasing the cluster size adds many capacitive loads on feedback paths (between blocks mapped into the same CLB). In the two-stage approach the drivers of these paths are not being sized simultaneously and therefore the cost of routing

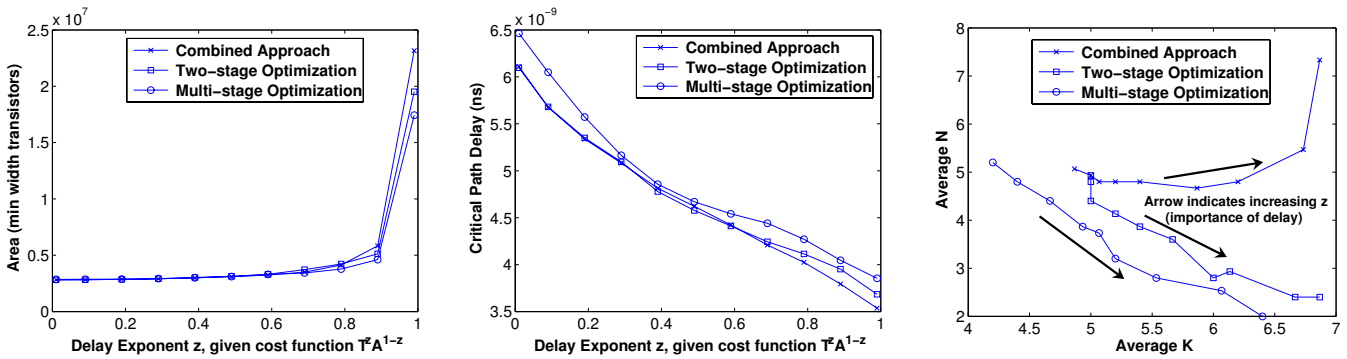


Fig. 8. Comparison of combined optimization, dual-stage and multi-stage optimization procedures: (a) gives the mean area of each approach, (b) shows the critical path delay and (c) shows what happens to N and K as the exponent is varied.

within a cluster is worse than if you had the flexibility to simultaneously size the transistors. This leads to a sub-optimal choice of N and K.

The geometric programming approach has a significant advantage in terms of run-time. The geometric program takes approximately 45 seconds to solve and includes optimization of the routing flexibility parameters and the channel width. This is a significant improvement over previous work: 12 hours was reported in [7] for a single set of high-level architecture parameters. However, we note that [7] is likely to be more accurate due to the use of the SPICE simulator, the inclusion of wire delay and the feedback loop that involves full placement and routing of designs using VPR.

VIII. Conclusion

This paper has presented a novel method for allowing high-level and low-level architecture parameters to be optimized simultaneously. We have shown that it is theoretically possible to gain a performance benefit by performing high-level and low-level architecture parameter optimization concurrently. The use of GP is particularly advantageous for fast, early stage exploration of configurable architectures. This is largely made possible due to the regular nature of FPGAs; the number of transistor widths that need to be optimized is relatively small allowing the optimization to be performed in a reasonable amount of time. To encourage the use of such techniques in FPGA design, our models are available at *http://cas.ee.ic.ac.uk/people/as999/GP*. In future we intend to use SPICE in order to extract accurate delay information and compare our architectures against [7]. The addition of wire delay to the model will also be included in future work in order to improve the accuracy of our modeling approach.

Acknowledgement

This work has been funded by the EPSRC under grant numbers EP/C549481/1 and EP/E00024X/1. The authors would also like to acknowledge Michael Grant and Professor Stephen Boyd for making the CVX convex programming toolbox used in this work publicly available.

References

[1] J. Luu, I. Kuon, P. Jamieson, T. Campbell, A. Ye, M. Fang, and J. Rose, "Vpr 5.0: FPGA CAD and architecture exploration tools with single-driver routing, heterogeneity and process scaling," in *Int'l Symp. on Field-Programmable Gate Arrays*, Feb. 2009, pp. 133–142.

[2] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-submicron FPGAs*. Kluwer Academic Publishers, 1999.

[3] A. M. Smith, G. A. Constantinides, and P. Y. K. Cheung, "Area estimation and optimisation of FPGA routing fabrics," in *Int'l Conf. on Field-Programmable Logic and Applications*, Sep. 2009.

[4] A. Lam, S. J. Wilton, P. Leong, and W. Luk, "An analytical model describing the relationships between logic architecture and FPGA density," in *Int'l Conf. on Field-Programmable Logic and Applications*, Sep. 2008.

[5] W. Fang and J. Rose, "Modeling FPGA routing demand in early-stage architecture development," in *Int'l Symp. on Field-Programmable Gate Arrays*, Feb. 2008, pp. 139–148.

[6] A. M. Smith, J. Das, and S. J. E. Wilton, "Wirelength modeling for homogeneous and heterogeneous FPGA architectural development," in *Int'l Symp. on Field-Programmable Gate Arrays*, Feb. 2009, pp. 181–190.

[7] I. Kuon and J. Rose, "Area and delay trade-offs in the circuit and architecture design of fpgas," in *Int'l Symp. on Field-Programmable Gate Arrays*, Feb. 2008, pp. 149–158.

[8] M. Lin, A. E. Gamal, Y.-C. Lu, and S. Wong, "Performance benefits of monolithically stacked 3-D FPGA," *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 216–229, Feb. 2007.

[9] S. P. Boyd, S.-J. Kim, D. D. Patil, and M. A. Horowitz, "Digital circuit optimization via geometric programming," *Operations Research*, vol. 53, no. 6, pp. 899–932, Nov-Dec 2008.

[10] S. Joshi and S. Boyd, "An efficient method for large-scale gate sizing," *IEEE Trans. on Circuits and Systems I: Regular Papers*, vol. 55, no. 9, pp. 2760–2773, Oct. 2008.

[11] S.-J. Kim, S. P. Boyd, S. Yun, D. D. Patil, and M. A. Horowitz, "A heuristic for optimizing stochastic activity networks with applications to statistical digital circuit sizing," *Optim Eng*, vol. 8, no. 4, pp. 397–430, Dec 2007.

[12] J. Das, S. J. Wilton, P. Leong, and W. Luk, "An analytical model describing the relationships between logic architecture and FPGA density," in *Int'l Conf. on Field-Programmable Logic and Applications*, Sep. 2009.

[13] W. Zhao and Y. Cao, "New generation of predictive technology model for sub-45nm design exploration," in *Int'l Symposium on Quality Electronic Design, 2006. ISQED '06.*, March 2006, pp. 6 pp.–590.

[14] W. C. Elmore, "The transient analysis of damped linear networks with particular regard to wideband amplifiers," *J. Appl. Phys.*, vol. 19, no. 1, pp. 55–63, Jan. 1948.

[15] M. Grant and S. Boyd, "CVX: Matlab software for disciplined convex programming (web page and software)," Feb. 2009, http://stanford.edu/~boyd/cvx.