Exploring FPGA Routing Architecture Stochastically

Mingjie Lin, *Member, IEEE,* John Wawrzynek, *Member, IEEE,* and Abbas El Gamal, *Fellow, IEEE*

***Abstract*—This paper proposes a systematic strategy to efficiently explore the design space of field-programmable gate array (FPGA) routing architectures. The key idea is to use stochastic methods to quickly locate near-optimal solutions in designing FPGA routing architectures without exhaustively enumerating all design points. The main objective of this paper is not as much about the specific numerical results obtained, as it is to show the applicability and effectiveness of the proposed optimization approach. To demonstrate the utility of the proposed stochastic approach, we developed the tool for optimizing routing architecture (TORCH) software based on the versatile place and route tool [1]. Given FPGA architecture parameters and a set of benchmark designs, TORCH simultaneously optimizes the routing channel segmentation and switch box patterns using the performance metric of average interconnect power-delay product estimated from placed and routed benchmark designs. Special techniques—such as incremental routing, infrequent placement, multi-modal move selection, and parallelized metric evaluation—are developed to reduce the overall run time and improve the quality of results. Our experimental results have shown that the stochastic design strategy is quite effective in co-optimizing both routing channel segmentation and switch patterns. With the optimized routing architecture, relative to the performance of our chosen architecture baseline, TORCH can achieve average improvements of 24% and 15% in delay and power consumption for the 20 largest Microelectronics Center of North Carolina benchmark designs, and 27% and 21% for the eight benchmark designs synthesized with the Altera Quartus II University Interface Program tool. Additionally, we found that the average segment length in an FPGA routing channel should decrease with technology scaling. Finally, we demonstrate the versatility of TORCH by illustrating how TORCH can be used to optimize other aspects of the routing architecture in an FPGA.**

***Index Terms*—Design exploration, FPGA, routing architecture, stochastic.**

I. Introduction

STUDIES HAVE shown that programmable routing structures contribute the majority of field-programmable gate array's (FPGA) silicon area, delay, and power consumption [2], [3]. As such, optimizing routing architecture is the key to improving FPGAs' overall performance. However, as the routing architecture of modern FPGAs becomes increasingly complicated, the design space grows so large that exhaustive exploration becomes computationally infeasible. Therefore, most FPGA architecture studies either take an analytical approach with simplifying assumptions [4], [5] or choose a limited design space over which to perform empirical studies [6], [7]. Although both these approaches provide important design insights and valuable empirical results, exploring a broader design space can conceivably produce even better results. This paper proposes a strategy based on stochastic methods to quickly locate a near-optimal design solution without exhaustively searching the entire design space of FPGA routing architectures.

In principle, our proposed stochastic method can be applied to optimize many aspects of the FPGA routing architecture. This paper, however, focuses on only two: *routing channel segmentation* and *switch box patterning* (see Fig. 1), both of which are essential to the overall FPGA performance [8], [9]. In the rest of this section, we first survey prior research on optimizing routing channel segmentation and crossbar switch patterns in FPGAs, then highlight several distinctions of this paper, and finally summarize our contributions.

A. Routing Channel Segmentation

All FPGAs use routing channel segmentation, whereby each routing track is divided into wire segments with different length. Studies have shown that the mix of segment lengths used in different routing tracks can have a significant impact on interconnect performance [2], [10], and hence the overall FPGA performance. Conceptually, using shorter segments results in better routability and lower excess net loading, and hence higher logic density and lower power consumption. However, nets routed with only short segments pass through more switch points, which typically results in higher delays. On the other hand, longer segments can improve delay, but at the expense of lower logic density and higher power consumption due to the need for more routing tracks and higher capacitive loading. It was further observed in [11] that the utility of long segments decreases with complementary metal-oxide-semiconductor (CMOS) technology scaling due to the increase in wire parasitics relative to device parasitics; and hence the average segment length should decrease with scaling. Given these tradeoffs and observations, how should

Manuscript received September 17, 2008; revised December 28, 2009 and May 5, 2010; accepted May 6, 2010. Date of current version September 22, 2010. This work was supported in part by the DARPA and SPAWAR System Center, under Grant N66001-04-1-8916. This paper was recommended by Associate Editor K. Bazargan.

M. Lin is with the Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA 94720 USA (e-mail: mingjie.lin@gmail.com).

J. Wawrzynek is with the University of California, Berkeley, CA 94720 USA, and also with the Lawrence Berkeley National Laboratory, NERSC Division, Berkeley, CA 94720 USA (e-mail: johnw@eecs.berkeley.edu).

A. El Gamal is with the School of Engineering, Stanford Faculty, Stanford, CA 94305 USA (e-mail: abbas@stanford.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2010.2061530

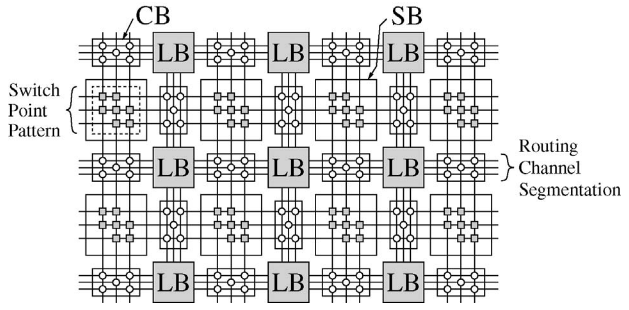


Fig. 1. Depiction of a island-style FPGA architecture. LB: logic block, SB: switch box, CB: connection box.

routing channel segmentation be chosen to optimize the overall FPGA performance?

The design of segmented routing channel was first discussed in [12] for row-based FPGAs. Assuming both random origination points and geometrically distributed connection lengths, El Gamal *et al.* showed how to construct a segmented routing channel, while only being a constant factor wider than a custom channel, achieves high routability. Using similar statistical approaches, Zhu *et al.* [13] and Pedram *et al.* [14] improved upon the results in [12] and corroborated the results experimentally. In [15], this statistical approach was extended to island-style FPGAs, where empirical distributions for horizontal and vertical net segment lengths were first determined by statistically analyzing placed and routed designs, and then separate horizontal and vertical channel segmentations were found according to the demand for each segment length. While studies such as above provide important insights into the design of routing channel segmentation, they have several shortcomings: 1) the connection models used do not accurately reflect the results produced by the actual placement and routing tools; 2) delay and power consumption are considered only indirectly; 3) buffered segments are not considered; 4) only the channel part of the programmable routing is considered; and 5) the results are technology independent.

To address these weaknesses, both analytical and empirical methods were attempted. Studies [16] and [17] used a bipartite graph matching approach and a multi-level matching-based algorithm to construct a segmented channel for a given set of connections from placed designs. More recently, several studies attempted to experimentally optimize routing channel segmentation. For example, in [6], Betz *et al.* studied segmentation design for island-style FPGAs implemented in $0.35\,\mu$m CMOS. Having placed and routed a set of designs with VPR on FPGAs with various segmentations, they showed that among channels of equal length segments, a channel with only length-4 segments achieves the lowest routing area and critical path delay. Moreover, a routing channel with a mixture of length-4 and 8 segments can outperform a channel with only length-4 segments. Similarly in [7], optimal uniform segmentation was investigated experimentally for nanometer FPGAs with single Vdd and programmable Vdd. Their results have shown that using length-3 segments leads to the lowest energy consumption as well as energy-delay-area product. All these studies, however, considered only a small subset of possible segmentations and therefore may have derived the results far from the optimum. In contrast, our approach, as discussed in Sections III and IV: 1) uses the connection models produced by the actual placement and routing tools; 2) considers delay and power consumption computed directly by placing and routing the chosen benchmark designs; 3) takes segment buffering into consideration by modeling the interconnect at the gate level; 4) simulates the total routing channel in computing delay, area, and power consumption; and 5) incorporates the impact of device technology in circuit simulation.

B. Switch Box Pattern Design

In modern FPGAs, switch modules (such as switch boxes and connection boxes) form the connections between wire segments and logic blocks, hence directly affect the routability and area efficiency of an FPGA [2]. In particular, switches in switch boxes allow intersected wire segments to inter-connect horizontal and vertical routing channels. Commonly, a switch box is constructed with crossbars and its design quality is measured by its capability of successfully routing selected target designs with fixed routing area [2]. In practice, although full crossbar provides high routability, a sparse crossbar is often selected because it can have significantly fewer switching points, and therefore less area consumption. Naturally, in designing switch boxes, it is desirable to achieve the required routing capacity using the minimum number of switches, i.e., finding the optimal switch pattern.

The switch pattern inside a switch block has long been a subject of intense research. For example, [18] discovered that 3 or 4 are optimal choices for the flexibility of a switch block (F_s) when considering both routability and area efficiency, which is later confirmed in [2] and [19]. Partially due to the complexity of designing a optimal switch pattern, several theoretical studies have been conducted and numerous "perfect" switch patterns with guaranteed-capacity were proposed, two of which are depicted in Fig. 2(a) and (b) [20], [21] and both use sparse switch pattern. More recently, the universal switch box (USB) [22]–[24] was proposed as a specification for routing capacity, i.e., routable for every set of 2-pin net routing requirements. In these studies, the first theoretically optimal (4, w)-USB which has flexibility $3w$ and $6w$ switches, where w is the width of crossbar switch, was designed. Although all these theoretic studies provide valuable insights to the routability of various crossbar switch patterns, their results are not very applicable in real-world FPGA architecture design mainly because, in practice, the design of switch pattern needs to be considered together with device technology, CAD software, and specific circuit design. As a result, finding "optimal" crossbar switch patterns in modern FPGAs is mostly done by empirical methods [10].

This paper treats designing the switch pattern in a crossbar as merely one aspect of optimizing the overall FPGA architecture. Our approach is empirical and conceptually similar to the method proposed in [11, p. 53] but with several distinctions: 1) while the study in [10] concentrated on finding switch pattern that maximizes routability, our objective is to achieve the overall optimal performance defined as the delay-power product for a given set of benchmark designs; 2) the

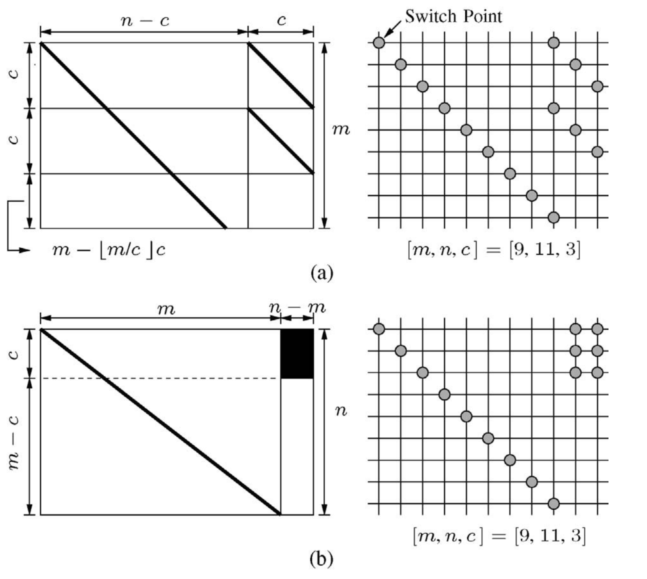


Fig. 2. (a) Orug-Huan guaranteed-capacity sparse crossbar switch. (b) Azegami guaranteed-capacity sparse crossbar switch.

optimization of switch patterns in our paper is integrated with the design of optimal routing channel segmentation, therefore allows a more holistic view; and 3) the optimization algorithm used in [10] is a greedy heuristic, while as in [25], a simulated annealing procedure is used in our paper to facilitate the exploration of the architecture design space.

C. Contributions

This paper advocates for a stochastic approaches in exploring the routing architecture of FPGAs. To manifest such an idea, we develop the tool for optimizing routing architecture (TORCH), a software tool that co-optimizes both routing channel segmentation and crossbar switch patterns in island-style FPGAs. Though it uses an experimental approach as in [6] and [7], TORCH explores a much larger design space of possible segmentations and switch patterns, hence can potentially yield more optimal results. Part of this paper was published in [26]. However, this paper provides: 1) a new design strategy to co-optimize both routing channel segmentation and switch patterns by introducing a novel multi-modal move selection scheme; 2) a more thorough description of optimizing routing channel segmentation; and 3) additional experimental results for larger and more realistic designs generated by the Quartus II University Interface Program (QUIP) package [27].

Given the parameters of an FPGA architecture and a representative set of benchmark designs, TORCH can find a set of optimized segmentation and crossbar switch pattern. Because power and delay are the key performance metrics in the design of FPGAs today, TORCH uses interconnect power-delay product averaged over a number of benchmark designs as a performance metric (also referred to as the objective function in the literature). Routability is considered only as a constraint and area is indirectly optimized through power and delay. Overall, the optimization of TORCH is governed by an iterative procedure based on simulated annealing. Each iteration comprises: 1) adaptively and incrementally changing the FPGA segmentation or crossbar switch pattern; 2) mapping the benchmark designs to the modified FPGA using VPR; 3) updating the performance metric; and 4) either accepting or rejecting the new segmentation or the new crossbar switch pattern. Our experiments have shown that performing complete placement and routing of the designs at each iteration, however, would make the total run time unacceptably high. One contribution of this paper is to address the long run-time problem of TORCH by developing the following techniques.

1) During each iteration of TORCH, because the change in either segmentation or crossbar switch pattern is relatively small, complete placements from scratch are not necessary. In fact, total run time can be significantly reduced by infrequently performing placement.
2) For the similar reason, the changes of routing channel during each iteration only affects a small fraction of all routed nets, hence complete routing from scratch are not necessary. Again, total run time can be significantly improved by incrementally ripping-up and rerouting only affected nets.
3) To accelerate the simulated annealing procedure, our paper adopts a multi-modal move selection scheme based on Gibbs sampling, i.e., each random move chooses among several move types dynamically instead of being limited to a single move type. Our test results show that this adaptation strategy can reduce run time by about 14% on average for the Microelectronics Center of North Carolina (MCNC) benchmark circuits.
4) The process of ripping-up, rerouting, and performance metric evaluation—the most computationally intensive part of the procedure—can be performed independently for each benchmark design. This means that the procedure can be readily parallelized and run on a computer cluster as discussed in Section VI. This technique, although trivial, can be quite useful in practice.

TORCH outputs not only an optimized mix of track segment lengths as in previous work [6] and [7], but also an optimized *ordering* of the segmented tracks in the channel. Possibly due to the sparse connectivity of the connection and switch box designs, our experiments have shown that track ordering can have a non-negligible effect on routability, and hence delay and power consumption. We performed experiments that show around a 5% variation in power and delay due to track reordering.

Although TORCH allows switch boxes to have different switch patterns, in this paper, we kept all switch boxes identical to permit a repeated-tile style FPGA chip layout.

In what follows, Section II provides the background and definitions needed for describing TORCH, followed by Section III that explains its algorithm and implementation in detail. Experimental results using TORCH and the 20 largest MCNC designs are analyzed in Section IV. Finally, parallel implementation of TORCH and experimental results using larger designs are presented in Section VI.

II. Preliminaries

We choose an island-style FPGA logic fabric [2] as the target architecture for TORCH, which consists of a 2-D array

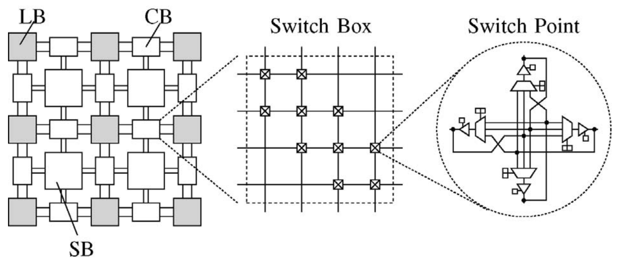


Fig. 3. Baseline FPGA architecture.

of logic blocks (LBs) interconnected via a segmented routing architecture [28] (see Fig. 3). To a large extent, our paper assumes a simplified version of a Virtex II style logic block comprising four logic slices, each of which contains two 4-input lookup tables (LUTs), two flip-flops (FFs), and programming overhead [28]. Our choice of cluster and LUT sizes follows the results of [29], where under a fixed routing architecture (using wire segment length-4 and 50% pass transistors and 50% tri-state buffers in routing switches), an island-style FPGA with the cluster size of 8 and LUT size of 4 was shown to consume the lowest energy. The routing fabric comprises horizontal and vertical segmented routing channels. In all delay and power comparisons that follow, we set the number of routing tracks to be 15% higher than the minimum required to avoid excessive routing congestion as in [2], [6]–[31]. The design of the connection box (CB) and switch box (SB) follows [2] and the design of the switch point is MUX-based as described in [32] (see Fig. 3). The channel segmentation for the baseline FPGA comprises sets of staggered single, double, length-3, and length-6 segmented tracks, each of which consists of two unidirectional metal wires that can be connected via CBs only to the inputs and outputs of the first and last LBs it spans. Segments can be connected to each other via SBs.

For the purpose of clearly defining the input to TORCH, we define an *FPGA architecture* $\mathcal{A}$ by the following.

1) The LB array size $N \times N$.
2) The switch box width W, flexibility F_s; the number of outputs that an input can connect to, and switch point pattern $\mathcal{P}_{\text{SB}}$. In our experiments, we assume the total number of switch points is $F_s \times W$ and the initial switch pattern $\mathcal{P}^0_{\text{SB}}$ to be a subset switch point pattern [1].
3) The connection box flexibility F_c; the average number of tracks that an LB input or output can be connected to, and the connection pattern $\mathcal{P}_{\text{CB}}$.
4) A set of segment lengths $\mathcal{L} \subset \{1, 2, \ldots, N\}$.
5) A channel consisting of *track bundles*. Each bundle consists of l staggered and uniformly segmented tracks. The purpose of staggering is to provide uniform connectivity to all LBs. Note that this track structure makes the number of track bundles in a channel (W_{tb}) equal to the switch box width (W). In most of the experiments in Section IV, we only allow the following four types of track bundles (see Fig. 4).
   a) A single track bundle consists of one track with length-1 segments.
   b) A double track bundle consists of 2 tracks with staggered length-2 segments.

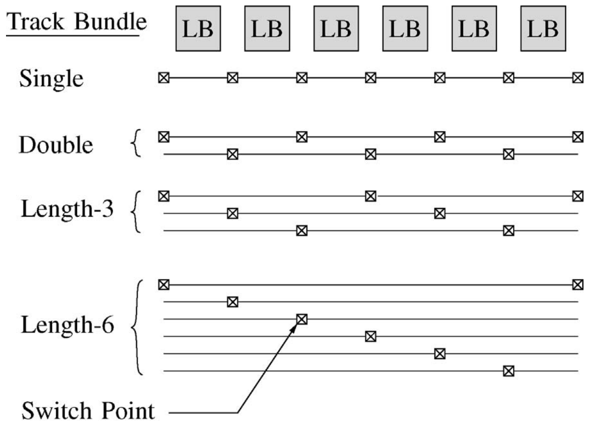


Fig. 4. Example of interconnect segmentation.

   c) A length-3 track bundle consists of 3 tracks with staggered length-3 segments. Each length-3 track can connect only to its leftmost and rightmost LB inputs and outputs.
   d) A length-6 track bundle consists of 6 tracks with staggered length-6 segments. Each length-6 track can connect only to its leftmost and rightmost LB inputs and outputs.

   Note that each longer segments (length-3, length-6) may include switch-transistors and buffers to optimize its delay.
6) A channel *segmentation s* consists of a set of W_{tb} *track bundles*, where bundle i consists of l_i staggered tracks.

A. Area, Delay, and Power Estimation

In TORCH, evaluating the performance metric as defined in (1) (see Section III) requires estimating the FPGA's routing area, delay, and power for each placed and routed benchmark design at a given technology node. In this paper, we use the same estimation methodology detailed in [11]. Specifically, to estimate the FPGA routing area, TORCH decomposes an FPGA tile consisting of an LB, a connection box, and a switch box into components with the granularity similar to standard-cell library elements, estimates the area of each component in λ^2 from a stick diagram and the Magic-8 rules, and finally sums the areas of all component to obtain the tile area. Note in TORCH, a more accurate area model can be employed if more realistic circuit layout data are available. To estimate delay and power, we use the transistor and metal wire RC models shown in Fig. 5. This paper considers five technology nodes: 130 nm, 90 nm, 65 nm, 45 nm, and 32 nm. Table I presents the model parameters estimated for these technology nodes using the Berkeley Predictive Technology Models and HSPICE [33], [34].

The interconnect delay for a placed and routed design can be defined either as the geometric average of all its pin-to-pin net delays, or as the critical path delay. In fact, our numerical results have shown that the choice of delay type can significantly skew the final results. Because it is generally believed that critical-path delay better matches the user's perception of FPGA delay performance,

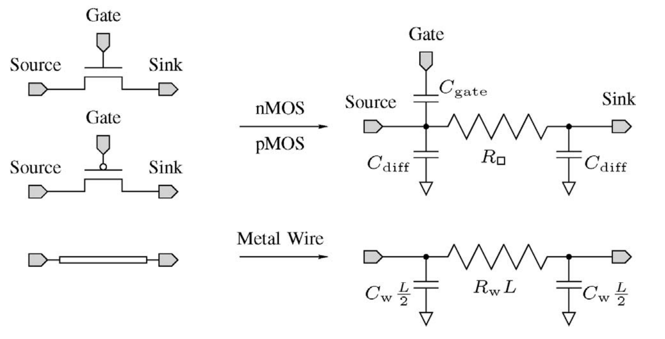


Fig. 5. RC circuit model for CMOS transistors and metal wires. $C_{\rm gate}$ is the equivalent transistor gate capacitance (in fF/□), $C_{\rm diff}$ is the transistor diffusion capacitance (in fF/μm), $R_\square$ is the transistor channel resistance (in Ω/□), C_w is the metal wire capacitance (in fF/mm), R_w is the metal wire resistance (in Ω/mm), and L_w is the length of a metal wire.

we use critical-path delay as the delay performance metric in this paper.

As in [11], we first use Elmore RC models to size all the devices in a switch box and determine the number and sizes of buffers for the length-3 and length-6 segments at each technology node. Then using a modified version of the VPR delay calculation function, net delays are computed.

Our power estimation method is largely based on the mixed-level power model proposed in [5], which consists of both dynamic and static components. For dynamic power, we consider two main components: switching power and short-circuit power. We define the total switching power for a placed and routed design as the total equivalent capacitance of all nets, computed with an extension to VPR as in [11]. Short-circuit power is another type of dynamic power that occurs at a signal transition and is generally hard to estimate. TORCH assumes the ratio between short-circuit power and switching power to be constant and uses SPICE simulation to determine this ratio. Prior to the TORCH run, we simulate interconnect buffers with different sizes and load capacitances to obtain the dynamic power per output signal transition. Estimating static power is even more challenging. TORCH ignores the reverse-biased leakage power for the lack of accessible device model and only considers the sub-threshold leakage power. As in [11], we use SPICE simulation to obtain the average leakage power, assuming all the input vectors have the same probability of occurrence and either Vdd or GND as the input signals in the simulation. To simplify the TORCH estimation, all possible input vectors are mapped into eight typical vectors and SPICE simulation is performed only for these eight typical vectors to build macromodels.

III. TORCH

The input to TORCH is as follows.

1) An initial FPGA architecture $\mathcal{A}$, which includes the dimension N, the logic design of each logic block, the number of LUTs in each logic block, and so on.
2) The switch box width $W(=W_{\rm tb})$, flexibility F_s, the connection box flexibility F_c, and the connection pattern $\mathcal{P}_{\rm CB}$.

TABLE I
TRANSISTOR AND METAL WIRE PARASITICS FOR FIVE TECHNOLOGY NODES

	130 nm	90 nm	65 nm	45 nm	32 nm
$V_{\rm dd}$ (V)	1.3	1.2	1.1	1.0	0.9
$L_{\rm eff}$ (nm)	49	35	24.5	17.5	12.6
$C_{\rm gate}$ (fF/μm)	1.73	1.59	1.32	1.24	1.11
$C_{\rm diff}$ (fF/μm)	1.13	1.09	1.08	1.03	1.01
$R_\square$ (kΩ/ □)	32.61	22.70	18.68	16.76	15.88
R_w (Ω/mm)	174	244	448	1527	2444
L_w (nH/mm)	1.68	1.71	1.76	1.89	1.93
C_w (fF/mm)	210	212	177	157	168

3) A *baseline* segmentation s_0. The choice of the baseline segmentation is arbitrary and is needed only to normalize the power and delay for each benchmark design. In the rest of this paper, we refer to this baseline segmentation as the Virtex-like segmentation. An example of such Virtex-like segmentation is given in Section IV.
4) A *baseline* cross switch pattern $\mathcal{P}_{\rm SB}^0$. The initial switch pattern is assumed to be subset switch point pattern [1].
5) Technology node parameters. These include interconnect device sizes and RC parameters for delay and power calculation in VPR (see Table I).
6) A set of m benchmark designs $\mathcal{B}$.

The output of TORCH is an optimized segmentation s^* and an optimized crossbar switch pattern $\mathcal{P}_{\rm SB}^*$.

Given the FPGA architecture $\mathcal{A}$ with a segmentation s and a crossbar switch pattern x, VPR generates the routing graph $g(\mathcal{A}, s, x)$ [1]. Given g, the technology node parameters, and $\mathcal{B}$, the performance metric of TORCH is defined as follows. Let $(p_{b,0}, d_{b,0})$ be the power-delay pair for design b mapped to the FPGA with the Virtex-like segmentation, and $(p_{b,s}, d_{b,s})$ be the power-delay pair for design b using segmentation s. For power and delay exponents $\alpha, \beta \geq 0$, the performance metric is

$$c = \frac{1}{m}\sum_{b=1}^{m}\left(\frac{p_{b,s}}{p_{b,0}}\right)^{\alpha} \cdot \left(\frac{d_{b,s}}{d_{b,0}}\right)^{\beta}. \tag{1}$$

We are now ready to describe TORCH in detail. A flow diagram of TORCH, based on simulated annealing, is given in Fig. 6 and further described in Algorithm 1. First, the benchmark designs are placed and routed using VPR in the FPGA with the Virtex-like segmentation and the initial switch box pattern. The delay and power estimates $\{(p_{b0}, d_{b0} : b = 1, 2, \ldots, m\}$ are computed as the initial value of the performance metric. After an initial temperature for simulated annealing is set, a random perturbation to segmentation or switch-box pattern is chosen and its routing graph is generated. The designs are then incrementally placed and routed assuming the newly generated routing channel and the performance metric is re-evaluated. If the metric value is reduced, the new segmentation or crossbar switch pattern is accepted, otherwise it is accepted with a probability that depends on the increase

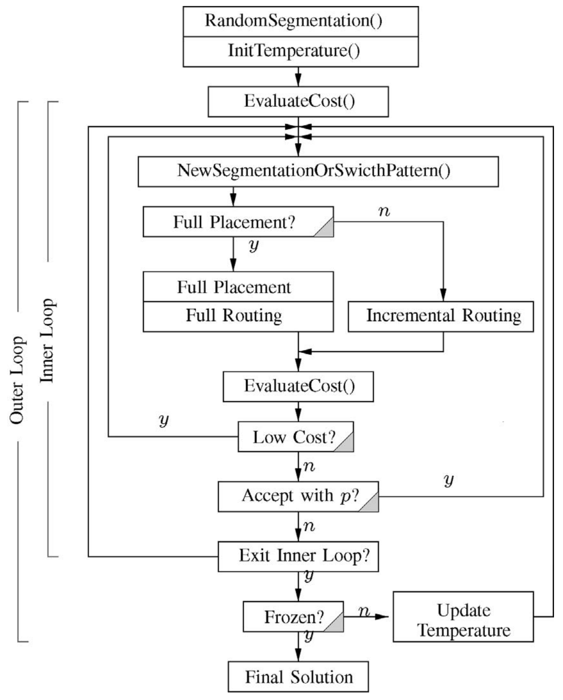


Fig. 6. Flow diagram of Algorithm 1. Box with gray triangle denotes control block.

Algorithm 1 TORCH

```
1: $s \leftarrow$ RandomSegmentation()
2: $T \leftarrow$ InitialTemperature()
3: $g \leftarrow g(\mathcal{A}, s)$
4: freeze_count $\leftarrow 0$
5: while ( ExitCriterion() is FALSE) do
6:    changes $\leftarrow 0$
7:    trials $\leftarrow 0$
8:    $c \leftarrow$ EvaluateCost($g, \mathcal{B}$)
9:    while ( InnerLoopCriterion() is FALSE) do
10:      trials $\leftarrow$ trials + 1
11:      $s_{\text{new}} \leftarrow$ NewSegmentationOrSwicthPattern($s$)
12:      if FullPlacementCriterion() is TRUE then
13:         perform full placement & routing
14:      else
15:         IncrementalRoute($g(\mathcal{A}, S_{\text{new}}), \mathcal{B}$)
16:      end if
17:      $\Delta c \leftarrow$ EvaluateCost($g(\mathcal{A}, S_{\text{new}})$) - $c$
18:      if $\Delta c < 0$ /*downhill move*/ then
19:         changes $\leftarrow$ changes + 1
20:         $s \leftarrow s_{\text{new}}$
21:         $g \leftarrow g(\mathcal{A}, S)$
22:         $c^* \leftarrow$ EvaluateCost($g(\mathcal{A}, S_{\text{new}})$)
23:      end if
24:      if $\Delta c > 0$ /*uphill move*/ then
25:         $r \leftarrow$ Random(0, 1)
26:         if $r < e^{-\frac{\Delta c}{T}}$ then
27:            $s \leftarrow s_{\text{new}}$
28:            $g \leftarrow g(\mathcal{A}, S)$
29:         end if
30:      end if
31:   end while
32:   $T \leftarrow$ UpdateTemperature()
33:   if $c^*$ changes then
34:      freeze_count $\leftarrow 0$
35:   end if
36:   if $\frac{changes}{trials} < 0.01$ then
37:      freeze_count $\leftarrow$ freeze_count + 1
38:   end if
39: end while
```

in the value of the metric and temperature. The process of changing segmentation or crossbar switch pattern, computing its performance metric, and accepting or rejecting it is repeated until `InnerLoopCriterion` is false. After exiting the inner loop, temperature is reduced and the process is repeated until the `ExitCriterion` becomes false. TORCH then outputs the final segmentation and the final crossbar switch pattern. Not surprisingly, the most computationally intensive part of TORCH is the subroutine `EvaluateCost()`, which involves performing placement/routing and computing delay and power for all benchmark designs. Fortunately, because these evaluations are completely independent for different designs, the subroutine of this nature is often called "embarrassingly parallel" and therefore can be readily parallelized, which significantly reduces the total run time as illustrated in Section VI. Although this parallelizing technique may be trivial, it is quite useful in design practice. The following describes in detail the key functions of Algorithm 1.

A. NewSegmentationOrSwicthPattern()

When generating candidate moves in simulated annealing, it is essential that after a few iterations of the algorithm, the current state should have much lower energy than a random state in order to ensure the optimization's efficiency. Therefore, as a general rule, one should skew the move generating process toward candidate moves where the energy of the destination state is likely to be similar to that of the current state. Unfortunately, this heuristic—the main principle of the Metropolis-Hastings algorithm—tends to exclude "very good" candidate moves as well as "very bad" ones; however, the latter are usually much more common than the former, so the heuristic is generally quite effective [35]. Intuitively, at high annealing temperature, distant moves are more likely to improve the solution. While at low temperature, close-by moves are expected to be far more effective. Obviously, during the process of annealing, depending on the temperature and the current landscape of solution space, certain move types tend to be noticeably more effective than others. The challenge is of course to select the most effective move types without prior knowledge of the global solution and the overall solution space.

In the following, we develop a heuristic technique to skew the generated random moves toward more effective ones. We call this method *multi-modal move selection* to reflect the fact that during each iteration, instead of limiting to only one move type, we stochastically choose one from a candidate pool of several choices. This technique is especially important when co-optimizing several aspects of FPGA routing architecture simultaneously. Specifically in TORCH, the move is chosen according to the success rate of all candidate move types. In this paper, we consider two move types: new segmentation and new switch pattern, while the original VPR, to our knowledge, considers only single move type. More importantly, we propose a robust adaptive strategy to automatically select the "best" move type throughout the annealing procedure based on Gibbs sampling [36]. By "best" we mean the chosen move type will yield the best results on average in a probabilistic sense.

Our proposed *multi-modal move selection*, although fairly intuitive, has its roots in the theory of *Gibbs sampling*—a special case of Metropolis-Hastings algorithm. The key observation of Gibbs sampling is that given a multivariate distribution it is simpler to sample from a conditional distribution than to marginalize by integrating over a joint distribution, which is often not known explicitly, whereas the conditional distribution of each variable is known. The Gibbs sampling algorithm generates an instance from the distribution of each variable in turn, conditional on the current values of the other variables. In this paper, during the process of simulated annealing, in order to obey Metropolis-Hastings criterion (the underlying law to generate acceptance probability in simulated annealing), the correct choices of various move types have to obey some implicit probabilistic distribution. In other words, when having multiple choices of move types, according to Gibbs sampling theory, the only intuitive/applicable way is to individually determine the acceptance probability for one particular move while fixing other moves. The theoretic framework of Gibbs sampling may seem to be overkill in our case, but will prove to be invaluable when constructing more sophisticated *multi-modal move selection* than ours.

We now describe the implementation details of *multi-modal move selection*. Let $m_i \in M$, $i = 0, 1, \cdots, n-1$ be one of n possible move types. Initially, we draw a move from M with equal probability $\frac{1}{n}$ among all move types. As the annealing process progresses, we keep track of the acceptance rate a_i of each move type m_i. During each iteration, each move is draw according to the following probability:

$$p(m_i) = \frac{a_i}{\sum_{i=0}^{n-1} a_i}.$$

In this paper, n equals 2 and possible moves consists of new segmentation and new cross switch pattern. Fig. 7 shows how the segmentation is changed in each trial. Note that this is often referred to as the candidate generator procedure. One track bundle is selected at random and its segment length is chosen according to the state diagram in the figure. Except for the shortest and longest segments, the track bundle segment length is increased or decreased to one of the two nearest segment lengths with equal probability.

Optimizing switch patterns in an FPGA switch box has long been a topic of extensive research. For example, the routability of common switch block styles, such as the disjoint switch blocks shown in Fig. 2, was validated through extensive experimentation. Our method to optimize switch patterns in TORCH is similar to the one discussed in [10], but with several differences. First, although both TORCH and [10] perform iterative optimizations for switch patterns, [10] uses greedy scheme and TORCH adopts an annealing-based approach. More importantly, in TORCH, the process of optimizing switch-box patterns is intertwined with optimizing routing channel segmentation, while in [10], the optimization of switch patterns is considered independently. Additionally, Hamming distance was used as the cost function in [10], while TORCH uses the normalized delay-power product over 20 largest MCNC benchmark circuits. Finally, while in [10], swap candidates are determined by the random selections of two input wires, in TORCH, each random swap for optimizing switch pattern is generated by stochastically moving a switch point. Specifically, a new crossbar switch pattern is generated with random swapping. Assuming the position of each switch point is denoted by (i, j), where $i, j = 0, 1, \cdots, N-1$, and all $N \times N$ possible positions of switch points is partitioned into two sets $\mathbf{S_0}$ and $\mathbf{S_1}$ consisting of unoccupied and occupied crossbar switch points, respectively. Each candidate move is generated by swapping two positions randomly picked within $\mathbf{S_0}$ and $\mathbf{S_1}$. Our early attempts have shown that unconstrained random swaps do not perform satisfactorily and too many rejected moves. Instead, we developed a constrained strategy for move generation where, during each iteration, we first randomly pick a switch point and then swap it with its closest empty spot in the switch box. Should several candidate positions exist, one is chosen at random. Fig. 8(a) and (c) depicts a $N = 6$ and $W_{\text{tb}} = 1$ crossbar switch before and after the random swapping of a switch point. Furthermore, Fig. 8(b) and (d) illustrates their routing graphs accordingly.

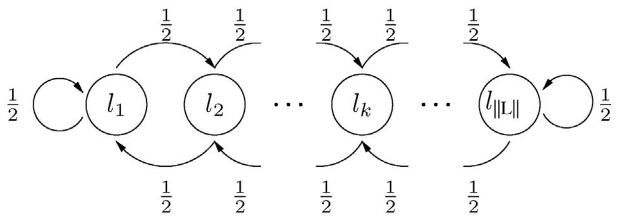


Fig. 7. State transition diagram for generating new segmentation in Algorithm 1.

B. ExitCriterion() & InnerLoopCriterion()

`ExitCriterion()` makes sure the freeze_count is less than 50 000. `InnerLoopCriterion()` is true if `trials` < `TRIALS` and `changes` < `CHANGES`. Both constants TRIALS and CHANGES are related to the problem size. We set TRIALS and CHANGES equal to $10W$ and $0.01W$, respectively, where W is the SB width.

C. IncrementalRoute()

Incremental routing does not affect TORCH's correctness but significantly impacts its run time. This is because performing complete routing for each trial in TORCH is often computationally prohibitive. Incremental routing is possible because the change in either segmentation or switch pattern

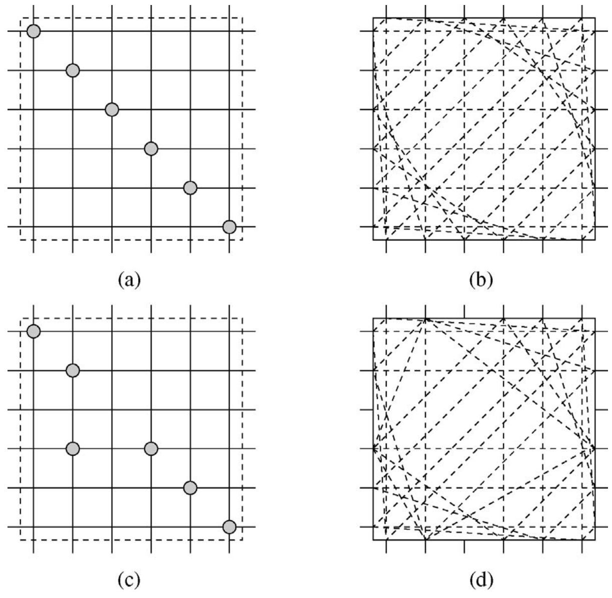


Fig. 8. Diagrams of switch pattern and routing graph changes. (a) and (b) Before a random move. (c) and (d) After a random move.

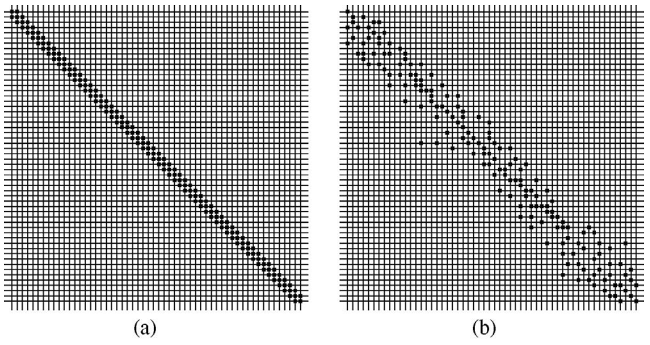


Fig. 9. Switch patterns for a $W_{\mathrm{tb}} = 56$, $F_s = 3$ switch box at 45 nm (a) before and (b) after the TORCH optimization.

from one trial to the next is relatively small and typically affects only a small fraction of the routed nets. By only ripping out the affected nets and rerouting them using the new segmentation or the new switch pattern, we save a significant amount of computing time.

`IncrementalRoute()` uses the ripping and rerouting nets part of VPR [2] and is described in Algorithm 2. The definitions of the variables used in Algorithm 2 are as follows.

1) RN_k is the set of routing graph nodes associated with track bundle k in all channel.
2) AN_k is the set of nets affected by the change of track bundle k segmentation.
3) A_{ij} is the criticality of the connection from the source of net i to one of its sinks j, which is computed according to (4.13) in [31, p. 83]. To save the total run time, A_{ij} is only updated after each full routing by analyzing timing of the entire circuit.
4) d_n is the intrinsic delay of routing node n.
5) p_n is the present congestion cost of node n, which is updated whenever any net is ripped and rerouted. We use the same formula as in [31, p. 78, (4.4)] to compute p_n. However, specific to TORCH, we start with the penalty ratio $p_n = 0.5$, and double its value only after each time of re-routing all nets in AN_k.

Algorithm 2 Incremental Routing Algorithm

```
1: $RN_k \leftarrow \emptyset$
2: **for all** nodes $n$ in the whole routing graph **do**
3:    **if** node $n$ is in the routing track $k$ **then**
4:       $RN_k \leftarrow RN_k \cup$ node $n$
5:    **end if**
6: **end for**
7: affected nets $AN_k \leftarrow \emptyset$
8: **for all** nets $p$ in the previously routed circuit **do**
9:    **if** net $p$ contains a routing node that belongs to $RN_k$ **then**
10:      $AN_k \leftarrow AN_k \cup$ net $p$
11:   **end if**
12: **end for**
13: **for all** net $i$ in $AN_k$ **do**
14:    $A_{ij} \leftarrow 1$ for each sink $j$
15: **end for**
16: **while** shared routing nodes exist **do**
17:    **for all** nets $i$ in $AN_k$ **do**
18:       rip up routing tree $RT_i$
19:       initialize the queue $PQ$
20:       **for all** sinks $t_{ij}$ **do**
21:          enqueue each node $n$ in $RT_i$ at costs $A_{ij}d_n$ to $PQ$
22:          **while** $t_{ij}$ is not found **do**
23:             dequeue node $m$ with the lowest cost from $PQ$
24:             **for all** fanout node $n$ of $m$ **do**
25:                **if** node $n$ is unseen **then**
26:                   mark node $n$ as seen
27:                   enqueue $n$ to $PQ$ with the cost of $A_{ij}d_n + (1 - A_{ij})d_n p_n$
28:                **end if**
29:             **end for**
30:             **for all** node $n$ in the routed path $t_{ij}$ to $s_j$ **do**
31:                update the cost of node $n$
32:                add $n$ to $RT_i$
33:             **end for**
34:          **end while**
35:       **end for**
36:       mark all nodes in $PQ$ as unseen
37:       update $A_{ij}$ for net $i$ and set it to the value computed from the last timing analysis
38:    **end for**
39: **end while**
```

Algorithm 2 first finds RN_i (lines 1 to 6). Next, it constructs AN_i (lines 7 to 12). The nets in AN_i are then routed using routines from VPR [1].

IV. Experimental Results

This section describes our experiments using TORCH and the largest 20 MCNC benchmark designs.

TABLE II
BASELINE FPGA PARAMETER VALUES

Tile Width L	Array Size $N \times N$	LB Buffer Size, b	LB Inputs K_i
4100λ	64×64	4	32

LB Outputs K_o	SB Width W	Density d	CB Flexibility F_c
8	56	3	0.5

Segment Type	Single	Double	Length-3	Length-6
Number of tracks	25	28	24	54

TABLE III
PASS-TRANSISTOR AND BUFFER SIZES FOR FPGA INTERCONNECTS FOR DIFFERENT TECHNOLOGY NODES

	CB		Single	Double	Length-3	Length-6
Tech.	b_i, b_o	x, y	m_l		m_l, l_N, n_N	
32 nm	6, 6	8, 7	10	12	12, 1, 8	14, 2, 11
45 nm	5, 5	7.7	9	10	11, 1, 8	14, 2, 10
65 nm	4, 4	6, 6	8	9	11, 1, 7	13, 2, 9
90 nm	4, 5	6, 5	8	9	10, 1, 7	12, 2, 8
130 nm	4, 4	5, 4	6	8	9, 1, 6	11, 1, 7

A. Experimental Setup

Table II lists the key architecture parameter values for the baseline FPGA used in the performance comparisons. The channel width $W = 25+28/4+24/3+54/6 = 56$ is determined by running experiments with all segment lengths at one. Table III lists the pass-transistor and buffer sizes used for each of the five technology nodes. In this table, all transistor sizes are in the unit of minimum transistor width corresponding to their technology nodes. Specifically, b_i denotes LB input buffer size, b_o denotes LB output buffer size, x denotes pass transistor (PT) size from LB output to CB, and y denotes PT size from interconnect to LB input. Additionally, l_N is the number of buffers inserted in a long interconnect, m_l is the switch point (SP) buffer size for interconnect of length l, and n_N is the size of inserted buffer in long interconnect.

It is well-known that selecting the proper switch size is an important step in creating a low-delay, area-effective interconnect in an FPGA. In order to obtain proper sizes for switches in each interconnect, we simulated the end-to-end delay of a buffer driving one to eight wires connected in series using pass transistors, which represents a wide range from only-buffered wires to primarily pass-transistor connected wires. For each interconnect length, we then choose the best pass-transistor size that produces the best delay-area values. This methodology is similar to the one outlined in [32].

Simulated annealing parameters such as starting temperature and cooling rate can have a significant impact on the quality of results. The assumed values of these parameters, given in Table IV, are chosen based on experiments. Further validation of these choices is provided in Section VI.

TABLE IV
PARAMETER VALUES OF SIMULATED ANNEALING COOLING SCHEDULE

Parameter	Value
Starting temperature T	100
Reducing rate δ	0.95
Moves at each temperature M	$200 \times$ routing channel width

As discussed earlier, TORCH run time can be improved by using incremental routing and infrequent placement. To quantify the improvements in run time and the effect of using these techniques on the quality of results, we ran TORCH with the 20 largest MCNC benchmarks, 45 nm technology, and $\alpha = \beta = 1$, i.e., equal weight on delay and power in the performance metric, for three scenarios.

1) Using full placement and routing at each iteration—each time when the routing channel segmentation or switch point pattern changes, the placement and routing for each benchmark circuit will be rerun from scratch.
2) Using full placement but only incremental routing at each iteration—each time when the routing channel segmentation or switch point pattern changes, the placement will be rerun from scratch while the re-routing will only be performed on affected nets for each benchmark circuit.
3) Using placement once every 20 iterations and incremental routing—when the routing channel segmentation or switch point pattern changes, the re-placement is only performed every 20 iterations while re-routing for each benchmark circuit will only be performed on affected nets for each benchmark circuit. Note, if any benchmark circuit becomes unroutable due to infrequent placement and/or incremental routing, the corresponded routing channel segmentation or switch point pattern change will be discarded.

The experiments were performed on a PC with 2.6 GHz AMD Athlon(tm) 64×2 Dual Core Processor 5200+ with 2 GB memory. Table V summarizes run times and final normalized average delay and power values relative to using the Virtex-like segmentation for the three scenarios. Note that incremental routing reduces the run time by about seven times. Combined with infrequent placement, the improvement in run time is about 30 times. As can be seen in the table, these improvements in run time are achieved with little degradation in the quality of results. As we discuss in Section VI, additional speedup with no further degradation in the quality of results can be obtained by parallelizing the `EvaluateCost` function.

B. Optimization Results

Fig. 10(a)–(c) shows the improvements in interconnect power and delay using incremental routing and infrequent placement for the 20 benchmark designs. The average reduction in delay and power relative to the Virtex-like segmentation are between 9% and 32% and between −2% and 33%, respectively. The average improvements relative to the all length

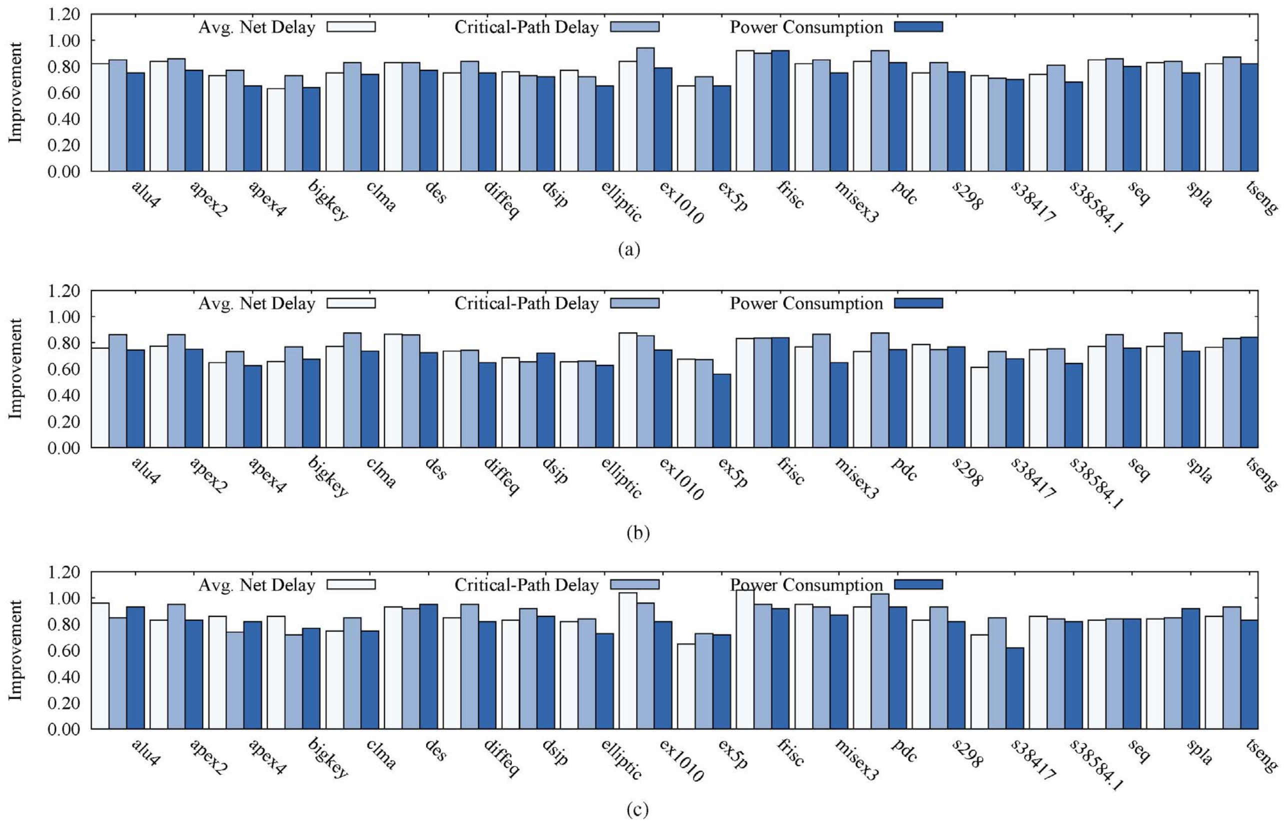


Fig. 10. Improvements in power and delay optimizing only routing channel segmentation. (a) Performance improvements relative to Virtex-like segmentation. (b) Performance improvements relative to all-one segmentation. (c) Performance improvements relative to 83% length-4 and 17% length-8 segmentation [6].

TABLE V

Run Time and Final Optimization Results for Full Placement and Routing (FP+FR), Full Placement and Incremental Routing (FP+IR), and Infrequent Placement and Incremental Routing (IP+IR)

	FP+FR	FP+IR	IP+IR
Run time (s)	152 362	23 231	5634
Normalized delay	0.72	0.76	0.77
Normalized power	0.83	0.85	0.86

one segmentation are 25% and 8%, respectively. Because critical-path design is an important performance metric for FPGA implementation, we also added the critical-path-delay and power product as an additional optimization objective. Our new results have shown that this new objective function results in quite similar routing channel segmentation.

Fig. 10(a) and (b) shows the improvement in interconnect power and delay for the 20 designs using the TORCH segmentation, relative to the Virtex-like segmentation and the all one segmentation, for 45 nm technology and $\alpha = \beta = 1$, i.e., equal weight on delay and power in the performance metric. The average reduction in delay and power relative to the Virtex-like segmentation are 24% and 15%, respectively. The improvements relative to the all one segmentation are 27% and 9%, respectively. Comparing Fig. 10(a) with (b), our results suggest that shorter segmentation tends to result in better power performance but worse delay performance (note: larger improvements due to TORCH optimization indicates the original design is worse.). To further evaluate our method, we compare our optimized segmentation with the results in [6]. As shown in Fig. 10(c), our segmentation has 10% and 13% better average delay and power consumption over the reported segmentation in [6] (83% length-4 and 17% length-8 segmentation).

To illustrate the effectiveness of co-optimizing both routing channel segmentation and switch point pattern relative to Virtex-like segmentation, we first use TORCH to co-optimize both, then use the resulting FPGA architecture to place and route all 20 MCNC designs, and finally compute its improvements relative to Virtex-like segmentation. Comparing the results in Fig. 11 with that of Fig. 10(a), we observe that co-optimization results in about 5% in average net delay, 7% in critical-path delay, and 6% in power consumption on average.

Finally, to test whether or not TORCH explores the architecture space well, we run TORCH ten times, each time started with a randomly generated routing channel segmentation and switch point pattern. More specifically, for routing channel segmentation, we randomly picked the ratio of each particular segment length; for switch point pattern, we started with the diagonal pattern and then randomly performed 20 swaps. Our results have shown that the resulted FPGA architecture from each run does differ, but the final performance(delay and power) of these runs are within 3% in difference. More

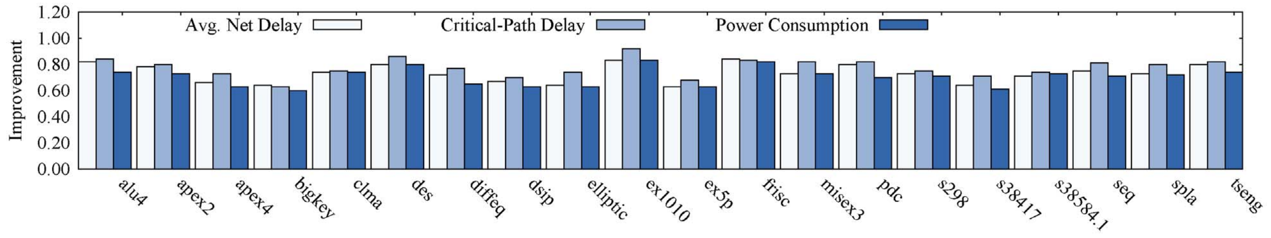
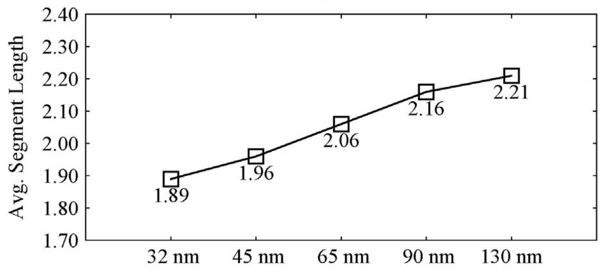


Fig. 11. Improvements in power and delay optimizing both routing channel segmentation and switch pattern relative to Virtex-like segmentation.

	Single	Double	Length-3	Length-6
32nm	26	22	4	4
45nm	24	22	6	4
65nm	22	25	6	5
90nm	20	19	13	4
130nm	16	24	12	4

(a)



(b)

Fig. 12. (a) Segmentation results for different technology nodes. (b) Average segment length for different technology nodes.

interestingly, the routing channel segmentation resulted from these runs typically differ in their exact arrangement but almost the same in terms of the ratio of different segment length.

C. Technology Scaling

In [11], we observed that average segment length should decrease with technology scaling. This decrease is expected because of the significant increase in wire parasitics relative to transistor parasitics with technology scaling. We use TORCH to study how segmentation should change with technology scaling more systematically. Fig. 12 shows the segmentations produced by TORCH for the five technology nodes and their average segment lengths. Note that average segment length is reduced from 2.21 at 130 nm to 1.89 at 32 nm. These results corroborate the observation in [11].

V. Alternative Ways of Using TORCH

TORCH can be used to optimize other interconnect architecture parameters, such as switch box width and switch and connection box flexibilities.

A. Switch Box Width

Here we demonstrate how TORCH can be used to select the switch box/channel width. Fig. 13 plots the average interconnect delay and power obtained by TORCH in 45 nm technology with $\alpha = \beta = 1$ for different switch box widths

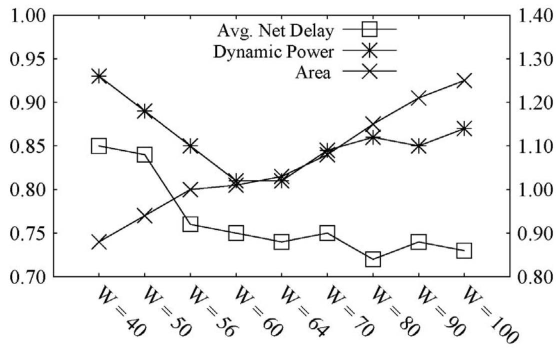


Fig. 13. Delay, power, and area improvements vs. the baseline architecture for different switch box width.

together with the corresponding estimates of the FPGA area relative to the FPGA with Virtex-like segmentation. Note that average delay first drops from 0.87 at $W_{\text{tb}} = 40$ to 0.76 at $W_{\text{tb}} = 60$, then remains roughly unchanged. This is because when W_{tb} is too small, many nets are routed in a highly suboptimal manner resulting in increased delay. As W_{tb} increases, nets are more optimally routed, which decreases delay. This decrease in delay diminishes as W_{tb} becomes too large. Power also first decreases as W_{tb} is increased, but then begins to increase as W_{tb} becomes too large because of the increase in parasitic loading due to the increase in area and number of tracks. To optimize power and delay, the graph suggests that $W_{\text{tb}} \approx 60$ is the best choice.

B. Power Delay Tradeoff

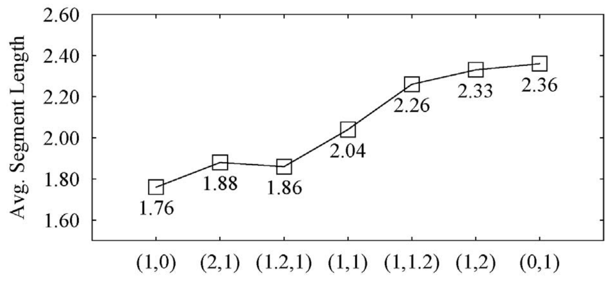
The performance metric used in TORCH allows for a tradeoff between power and delay. Fig. 14 plots the average segment length for different choices of α and β in 45 nm technology. As expected, average segment length increases as delay is emphasized more than power. The average power and delay are plotted in Fig. 15.

C. Set of Segment Lengths

In the previous results we limited the set of allowable segment length to $\{1, 2, 3, 6\}$. Is there a benefit from using more segment types? To explore this question we increased the size of the set of allowable segment lengths to $\{1, 2, 3, 4, 5, 6, 7, 8\}$ and ran TORCH with $\alpha = \beta = 1$ and 45 nm technology. Fig. 16 shows the resulting segmentation and the reduction in delay and power relative to the FPGA with optimized segmentation assuming segment length set $\{1, 2, 3, 6\}$. Note that on average delay is improved by 7% and power is improved by 6%. The

	Single	Double	Length-3	Length-6
(0,1)	17	23	9	7
(1,2)	20	21	7	8
(1,1.2)	22	18	9	7
(1,1)	25	19	7	5
(1.2,1)	28	17	8	3
(2,1)	30	12	11	3
(1,0)	34	10	9	3

(a)



(b)

Fig. 14. Routing channel segmentation results for different α and β at 45 nm technology node. (a) Segmentation results. (b) Average segment length.

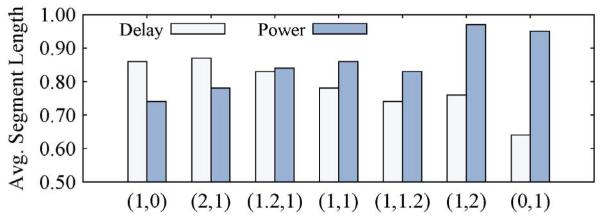


Fig. 15. Delay and power improvements vs. the baseline architecture for different α and β at 45 nm technology.

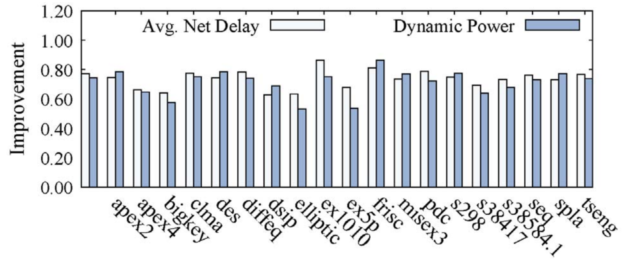


Fig. 16. Reductions in delay and power using segment length set $\{1, 2, \ldots, 8\}$ relative to the FPGA with optimized segmentation for segment length set $\{1, 2, 3, 6\}$.

estimated FPGA area is, however, increased by a factor of 0.14. The run time of the tool is also significantly longer.

VI. Parallelizing TORCH

In the previous section, we presented results using TORCH with small benchmark designs. In spite of performing infrequent placements and incremental routing, the run time is still too long for TORCH to be usable on a large number of large, state-of-the-art FPGA designs. This problem can be addressed by parallelizing the `EvaluateCost()` subroutine of TORCH, which is the most computationally intensive part. Because this subroutine is executed independently for each design, parallelization is quite straightforward and results

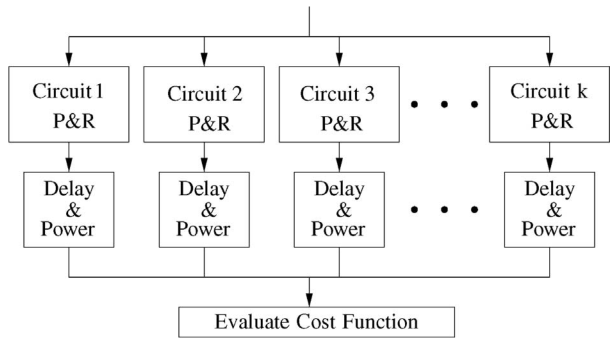


Fig. 17. Flow diagram of parallel implantation of the `EvaluateCost` function in TORCH.

in a speed up on the order of the number of benchmark designs used (see Fig. 17). In this section we report on experiments with parallel implementation of TORCH on the Stanford computer cluster Bio-X^2, which consists of 276 Dell PowerEdge 1950 compute nodes each with dual-socket quad-core processors and 16 GB of memory. Our experiment only uses a small number of these nodes.

We first used the parallel implementation of TORCH with the 20 largest MCNC benchmark designs and the same parameters and conditions given in Section IV. The run time of the parallel implementation was 14.63 times faster than the serial implementation using only one node.

We then used this parallel implementation with the 20 largest MCNC designs to validate our choice of the simulated annealing parameters used in the previous section. We increased the number of moves at each annealing temperature (denoted by Iter) by a factor of 10 to $2000 \times W_{\text{tb}}$. We found the difference in the final metric value between Iter = 200 and Iter = 2000 to be about 3%.

We compared two different annealing schedules, the linear schedule used in the experiments of Section IV and the adaptive cooling schedule in [37], which automatically adjusts the temperature at each step based on the energy difference between the two states. We found the difference between the final metric values using these two cooling schedules to be within 7%.

To test the effectiveness of the TORCH tool on large benchmark designs, we used the parallelized TORCH with eight designs synthesized with the Altera QUIP tool. These designs are three to four times larger than the largest MCNC benchmark. We choose a 100×100 array size and the SB width of 72 to accommodate the larger design size. For the Virtex-like segmentation, we assume 28 single, 26 double, 10 length-3, and 8 length-6 track bundles. The rest of the FPGA architecture parameters are the same as in Table II. We performed experiments with uniform all one segmentation to determine the minimum channel width needed for each design. The experiments used eight nodes of the computer cluster and run time of 267 min. The speedup over a single node implementation is about 9.5.

Fig. 18 shows the improvement in interconnect power and delay for the eight designs using the TORCH segmentation, relative to the Virtex-like segmentation for 45 nm technology and $\alpha = \beta = 1$, i.e., equal weight on delay and power in

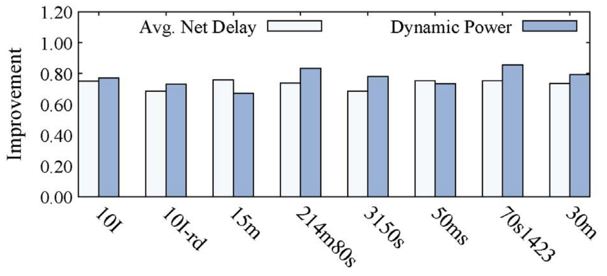
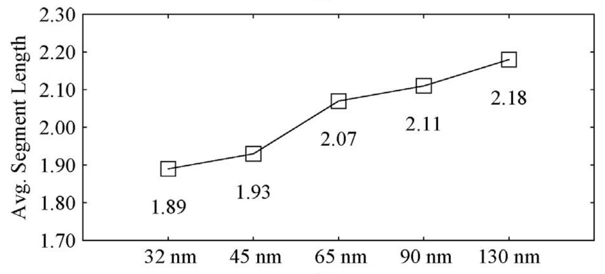


Fig. 18. Improvements in power and delay relative to Virtex-like segmentation for eight larger designs.

	Single	Double	Length-3	Length-6
32nm	35	22	11	4
45nm	36	17	15	4
65nm	33	19	14	6
90nm	32	15	20	5
130nm	29	19	18	6

(a)



(b)

Fig. 19. (a) Segmentation results for different technology nodes. (b) Average segment length for different technology nodes. (for large QUIP designs).

the performance metric. The average reduction in delay and power relative to the Virtex-like segmentation are 27% and 23%, respectively, and its segmentation is shown in Fig. 19.

As discussed in [11], the average segment length in FPGA routing channel tends to decrease with technology scaling due to the significant increase in wire parasitics relative to transistor parasitics with technology scaling. Our results in Sections IV confirm this observation. To further investigate this, we used TORCH to study how segmentation should change with technology scaling for larger designs. Fig. 19 shows the segmentations produced by TORCH for the five technology nodes and their average segment lengths. Note that average segment length is reduced from 2.18 at 130 nm to 1.88 at 32 nm.

The segmentation results in Fig. 19 is quite different from that in Fig. 12, which clearly shows the importance of choosing suitable benchmark suite when designing FPGA routing architecture. Moreover, as shown in both figures, the choice of technology nodes has a significant impact on the final results. All these observations enforce our belief that all contributing factors of FPGA architecture design should be considered in a systematic way in order to achieve high performance in an FPGA, where our proposed stochastic approach can potentially be a good solution in exploring large design space.

VII. Conclusion

As device technology scales to tens of nanometers, a single FPGA chip will soon contain billions of transistors, which makes optimally designing FPGA increasingly difficult. To meet such a challenge, we believe a systematic strategy is needed. This paper studies the idea of using a stochastic method to explore FPGA routing architecture. Specifically, TORCH uses simulated annealing to quickly locate near-optimal solutions in designing FPGA channel segmentation and switch patterning for a particular island-style FPGA architecture without exhaustively enumerating all design points. TORCH, in principle, can be readily adapted to any FPGA architecture, any placement and routing tool, and any performance metric based on placed and routed benchmark designs. Its effectiveness, however, may vary and special techniques may be needed to reduce the overall run time.

References

[1] V. Betz and J. Rose, "VPR: A new packing, placement and routing tool for FPGA research," in *Proc. 7th Int. Workshop Field-Programmable Logic Applications*, 1997, pp. 213–222.

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

[3] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," in *Proc. ACM/SIGDA 10th Int. Symp. Field-Programmable Gate Arrays*, 2006, pp. 21–30.

[4] H.-Y. Wong, L. Cheng, Y. Lin, and L. He, "FPGA device and architecture evaluation considering process variations," in *Proc. IEEE/ACM ICCAD*, 2005, pp. 19–24.

[5] F. Li, D. Chen, L. He, and J. Cong, "Architecture evaluation for power-efficient FPGAs," in *Proc. ACM/SIGDA 11th Int. Symp. FPGA*, 2003, pp. 175–184.

[6] V. Betz and J. Rose, "FPGA routing architecture: Segmentation and buffering to optimize speed and density," in *Proc. ACM/SIGDA 7th Int. Symp. FPGA*, 1999, pp. 59–68.

[7] R. Tu and B.-X. Shao, "Energy/performance/area tradeoffs in nanometer FPGA segmented routing architecture," in *Proc. 8th ICSICT*, 2006, pp. 1954–1956.

[8] *Automotive ProASIC3 Flash Family FPGAs Datasheet*, Actel, Inc., Mar. 2007.

[9] *Virtex-II Pro/Virtex-II Pro X Complete Data Sheet (All Four Modules)*, Xilinx, Inc., Mar. 2007.

[10] G. Lemieux and D. Lewis, *Design of Interconnection Networks for Programmable Logic*. Norwell, MA: Kluwer, 2004.

[11] M. Lin, A. El Gamal, Y.-C. Lu, and S. Wong, "Performance benefits of monolithically stacked 3-D FPGA," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 26, no. 2, pp. 216–229, Feb. 2007.

[12] A. El Gamal, J. Greene, and V. Roychowdhury, "Segmented channel routing in nearly as efficient as channel routing (and just as hard)," in *Proc. Univ. California/Santa Cruz Conf. Adv. Res. VLSI*, 1991, pp. 192–211.

[13] K. Zhu and D. F. Wong, "On channel segmentation design for row-based FPGAs," in *Proc. IEEE/ACM ICCAD*, 1992, pp. 26–29.

[14] M. Pedram, B. Nobandegani, and B. Preas, "Design and analysis of segmented routing channels for row-based FPGAs," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 13, no. 12, pp. 1470–1479, Dec. 1994.

[15] W.-K. Mak and D. F. Wong, "Channel segmentation design for symmetrical FPGAs," in *Proc. 1997 Int. Conf. Comput. Design*, 1997, pp. 496–501.

[16] E. Bozorgzadeh and M. Sarrafzadeh, "Customized regular channel design in FPGAs," in *Proc. ACM/SIGDA 7th Int. Symp. FPGA*, Aug. 2003, p. 240.

[17] Y.-W. Chang, J.-M. Lin, and M. Wong, "Matching-based algorithm for FPGA channel segmentation design," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 20, no. 6, pp. 784–791, Jun. 2001.

[18] S. Brown, R. Francis, J. Rose, and Z. Vranesic, Eds., *Field Programmable Gate Arrays*. Norwell, MA: Kluwer, 1992.

[19] J. Rose and S. Brown, "Flexibility of interconnection structures for field-programmable gate arrays," *IEEE J. Solid-State Circuits*, vol. 26, no. 3, pp. 277–282, Mar. 1991.

[20] A. Yavuz Oruc and H. Huang, "Crosspoint complexity of sparse crossbar concentrators," *IEEE Trans. Inf. Theory*, vol. 42, no. 5, pp. 1466–1471, Sep. 1996.

[21] K. Azegami, "Integrated circuit device with programmable junctions and method of designing such integrated circuit devices," U.S. Patent 6 323 678, 2001.

[22] H. Fan, J. Liu, and Y.-L. Wu, "General models and a reduction design technique for FPGA switch box designs," *IEEE Trans. Comput.*, vol. 52, no. 1, pp. 21–30, Jan. 2003.

[23] H. Fan, Y.-L. Wu, C.-C. Cheung, and J. Liu, "On optimal irregular switch box designs," in *Field Programmable Logic and Application*. Berlin, Germany: Springer, 2004, pp. 189–199.

[24] H. Fan, Y.-L. Wu, and C. L. Zhou, "Augmented disjoint switch boxes for FPGAs," in *Proc. WISICT*, 2005, pp. 129–134.

[25] K. Compton and S. Hauck, "Totem: Custom reconfigurable array generation," in *Proc. 9th Annu. IEEE Symp. Field-Programmable Custom Comput. Mach.*, 2001, pp. 1–9.

[26] M. Lin and A. El Gamal, "TORCH: A design tool for routing channel segmentation in FPGAs," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, 2008, pp. 131–138.

[27] Altera, Inc. (2003). *Quartus II University Interface Program* [Online]. Available: http://www.altera.oom/education/univ/research/unv-quip.html

[28] *Virtex-II Cpmplete Datasheet (All Four Modules)*, Xilinx, Inc., Nov. 2007.

[29] F. Li, Y. Lin, L. He, D. Chen, and J. Cong, "Power modeling and characteristics of field programmable gate arrays," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 24, no. 11, pp. 1712–1724, Nov. 2005.

[30] V. Betz and J. Rose, "Directional bias and non-uniformity in FPGA global routing architectures," in *Proc. IEEE/ACM ICCAD Dig. Tech. Papers*, Nov. 1996, pp. 652–659.

[31] V. Betz and J. Rose, "Effect of the prefabricated routing track distribution on FPGA area-efficiency," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 6, no. 3, pp. 445–456, Sep. 1998.

[32] G. Lemieux and D. Lewis, "Circuit design of routing switches," in *Proc. ACM/SIGDA 10th Int. Symp. FPGA*, 2002, pp. 19–28.

[33] W. Zhao and Y. Cao, "New generation of predictive technology model for sub-45 nm design exploration," in *Proc. 7th ISQED*, 2006, pp. 585–590.

[34] Y. Cao, T. Sato, D. Sylvester, M. Orshansky, and C. Hu, "New paradigm of predictive MOSFET and interconnect modeling for early circuit design," in *Proc. AIEEE CICC*, Jun. 2000, pp. 201–204.

[35] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, May 1983.

[36] G. Casella and E. I. George, "Explaining the Gibbs sampler," *Am. Statist.*, vol. 46, no. 3, pp. 167–174, 1992.

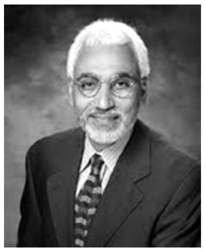
[37] J. de Vicente, J. Lancharesb, and R. Hermidab, "Placement by thermodynamic simulated annealing," *Phys. Lett. A*, vol. 317, nos. 5–6, pp. 415–423, 2003.

Mingjie Lin (M'08) received the B.S. degree in engineering from Xi'an Jiaotong University, Xi'an, China, in 1996, the M.S. degree in mechanical engineering and the M.S. degree in electrical engineering from Clemson University, Clemson, SC, in 2001, and the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, in 2008.

From 2008 to 2009, he was with an FPGA startup Tabula, Inc., Santa Clara, CA, as a Senior Engineer. Missing the atmosphere of academic research, he returned to academia at the beginning of 2009. Since then, he has been a Post-Doctoral Scholar with the Department of Electrical Engineering and Computer Science, University of California, Berkeley. His previous research interests include very large scale integration reconfigurable array architecture, bio-inspired/neuromorphic arrays, and monolithically stacked 3D-IC. His current research interests include exploring novel ways to construct scalable computing machine with high performance and low power consumption. To this end, his research activities spanned across computer architecture/compiler, reconfigurable computing, integrated circuit, and system design.

John Wawrzynek (M'85) received the B.S. degree in electrical engineering from the State University of New York at Buffalo, Buffalo, the M.S., Ph.D. degrees in computer science from the California Institute of Technology, Pasadena, and the M.S. degree in electrical engineering from the University of Illinois at Urbana-Champaign, Urbana.

He is currently a Professor of Electrical Engineering and Computer Sciences with the University of California, Berkeley. He is a Faculty Scientist with the Lawrence Berkeley National Laboratory, NERSC Division, Berkeley, CA, and is a Co-Director with the Berkeley Wireless Research Center, Berkeley. He currently teaches courses in digital design, computer architecture, very-large-scale integration system design, and reconfigurable computing. He was a Co-Founder of Andes Networks, Inc., Mountain View, CA, a company specializing in the design and manufacturing of accelerators for network security, and of BEECube, Inc., Fremont, CA, a company specializing in reconfigurable computing systems. His current research interests include reconfigurable computing and computer architecture.



Abbas El Gamal (F'00) received the B.S. (Honors) degree in electrical engineering from Cairo University, Giza, Egypt, in 1972, and the M.S. degree in statistics and the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, in 1977 and 1978, respectively.

From 1978 to 1980, he was an Assistant Professor of Electrical Engineering with the University of Southern California, Los Angeles. Since 1981, he has been with the Stanford Faculty, where he is currently the Hitachi America Professor with the School of Engineering. From 1997 to 2002, he was the Principal Investigator for the Stanford Programmable Digital Camera Program. From 2004 to 2009, he was the Director of the Information Systems Laboratory. His research contributions have spanned several areas, including network information theory, field programmable gate array, and imaging sensors. He has authored or co-authored over 200 papers and holds over 30 patents in these areas. He helped co-found several semiconductor and electronic design automation (EDA) companies, and has served on the board of directors and advisory boards of several other semiconductor, EDA, and biotech companies.