TORCH: A Design Tool for Routing Channel Segmentation in FPGAs

Mingjie Lin and Abbas El Gamal
Department of Electrical Engineering
Stanford University, CA 94305
{ mingjie, abbas }@stanford.edu

ABSTRACT

A design tool for routing channel segmentation in island-style FPGAs is presented. Given the FPGA architecture parameters and a set of benchmark designs, the tool optimizes routing channel segmentation using the average interconnect power-delay product as a performance metric, which is estimated from placed and routed designs. A simulated-annealing procedure is used, whereby segmentation is incrementally changed in each iteration, the benchmark designs are mapped using VPR, and the performance metric is computed to decide whether to accept or reject the new segmentation. Run time is significantly reduced by using incremental routing in each iteration and parallelizing the metric evaluation. Experimental results using the MCNC benchmark designs demonstrate an average of 22% and 15% reduction in delay and power relative to a baseline segmentation. The results also show that average segment length should decrease with technology scaling. Finally, we demonstrate how the tool can be used to optimize other aspects of programmable routing in an FPGA

Categories and Subject Descriptors

B.7.1 [**Integrated Circuits**]: [Types and Design Styles]

General Terms

Design, Experimentation, Measurement, Performance

Keywords

FPGA, segmentation, routing, architecture, design.

1. INTRODUCTION

Studies have shown that programmable interconnect contributes the majority of FPGA area, delay and power consumption [1, 2]. As such, optimizing programmable interconnect is key to improving FPGA performance. An approach that all FPGAs use to improve programmable interconnect performance is segmentation [3, 4, 5], whereby routing channels comprise different length interconnect segments. The mix of segment lengths used can have a significant impact on interconnect performance. On the one hand, using short segments results in better routability and hence higher logic density and lower power consumption due to low excess net loading, but at the expense of higher net delay due to the high number of switch points that a net needs to go through. Using long segments, on the other hand, can improve delay, but at the expense of a reduction in logic density due to the need for more tracks and increase in power consumption due to the increase in excess net loading. It was further observed in [6] that the utility of long segments decreases with CMOS technology scaling due to the increased wire parasitics relative to that of devices, and hence the average segment length should decrease with scaling. Given these tradeoffs, how should channel segmentation be chosen to optimize FPGA performance?

The problem of segmented routing channel design was first introduced for row-based FPGAs in [7]. Assuming random connection origination point and geometric connection length distribution, the paper showed that it is possible to construct a segmented routing channel that is only a constant factor wider than a custom channel, such that 1-segment routing of all nets is feasible, with high probability. Using similar statistical approaches, Zhu *et al* [8] and Pedram *et al* [9] improved upon the results in [7] and corroborated their results experimentally. A statistical approach is also used in [10] for island-style FPGAs. Empirical distributions for horizontal and vertical net segment left point and length are first determined by performing statistical analysis on net routings in placed designs. Separate horizontal and vertical channel segmentations are then found based on the demand for each segment length. The statistical approaches used in the above studies have several shortcomings: (i) the model for connections is oversimplified and does not accurately model the results produced by the actual placement and routing tools, (ii) delay and power consumption are considered only indirectly, (iii) buffered segments are not considered, (iv) only the channel part of the programmable routing is considered, and (v) the results are technology independent. The approaches in [11, 12] partially addresses the first shortcoming using graph theoretic algorithms. In [12], a bipartite graph matching approach is used to construct a segmented channel for a given set of connections from placed designs. A multi-level matching-based algorithm for general channel segmentation is then described and shown to yield good routability.

An experimental approach to segmentation design that

This work was partially supported by DARPA and SPAWAR System Center (Grant N66001-04-1-8916).

better addresses the aforementioned shortcomings is used in [13, 14]. In [13], Betz *et al* studied segmentation design for island-style FPGAs implemented in 0.35μm CMOS technology node. A set of designs are placed and routed using VPR in 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. They also showed that a channel with a mixture of length 4 and 8 segments can outperform a channel with only length 4 segments. In [14], optimal uniform segmentation is investigated experimentally for nanometer FPGAs with single Vdd and programmable Vdd. They showed that length 3 segment leads to the lowest energy consumption as well as the energy-delay-area product. These studies, however, consider only a very small number of possible segmentations and as such may have arrived at a highly suboptimal segmentation.

In this paper, we describe a tool that we refer to as TORCH, for channel segmentation design in island-style FPGAs. As in [13, 14], an experimental approach is used, but a much larger space of possible segmentations is explored, and as a result TORCH should yield close-to-optimal results. As in [15], the exploration of the architecture design space is facilitated via a simulated annealing procedure. Given the parameters of an FPGA architecture and a representative set of benchmark designs, TORCH can find an optimized segmentation. Because power and delay are the key performance metrics in the design of FPGAs today, TORCH uses interconnect power-delay product averaged over the benchmark designs as a performance metric. Routability is considered only as a constraint and area is indirectly optimized through power and delay. In each iteration, segmentation is incrementally changed, the benchmark designs are mapped into the FPGA with the new segmentation using VPR, the performance metric is updated, and the new segmentation is either accepted or rejected. Performing complete placement and routing of the designs at each iteration, however, would make the run time of this procedure unacceptably high. This problem is addressed using the following key observations:

- Because the change in segmentation in each iteration is small, placements do not need to to be changed in each iteration. Run time can be significantly reduced by only infrequently updating the placements.
- Again because the change in segmentation in each iteration is small, it affects only a small fraction of the routed nets. As such, incremental rip-up and reroute procedure can be performed instead of complete routing.
- The process of rip-up and reroute and performance metric evaluation, which is the most compute intensive part of the procedure, can be performed independently for each design. This means that the procedure can be readily parallelized and run on a computer cluster.

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

In the next section we provide the background and definitions needed to describe TORCH. The tool is described in Section 3. Experimental results using TORCH and the 20 largest MCNC designs are presented in Section 4.

2. PRELIMINARIES

In this section we discuss the FPGA architecture parameters and assumptions needed to describe TORCH. We then show how area, delay, and power used to evaluate the performance metric are estimated.

We choose an island-style FPGA logic fabric [1] as a target architecture for TORCH. The fabric consists of a 2D array of logic blocks (LBs) that can be interconnected via a segmented routing architecture. We assume a Virtex II style LB consisting of four slices, each comprising two 4-input Lookup Tables (LUTs), two flip-flops (FFs), and programming overhead. The programmable routing comprises horizontal and vertical channels with identical segmentation that can be connected to the LB inputs and outputs via *connection boxes* and to each other via *switch boxes*. We assume the MUX-based switch box design described in [16].

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

- The LB array size $N \times N$.
- The switch box width W, flexibility F_s, which is the number of outputs that an input can connect to, and switch point pattern. In our experiments we assume that $F_s = 3$ and subset switch point pattern [17].
- The connection box flexibility F_c, which is the average number of tracks that an LB input or output can be connected to, and the connection pattern. In our experiments we assume that $F_c = 0.5W$.
- A set of segment lengths $\mathcal{L} \subset \{1, 2, \ldots, N\}$.
- 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 equal to the switch box width W. In most of the experiments in Section 4, we only allow the following 4 types of track bundles (see Figure 1):
    1. A Single track bundle consists of one track with length 1 segments.
    2. A Double track bundle consists of 2 tracks with staggered length 2 segments.
    3. 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.
    4. 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.

- A channel *segmentation* s consists of a set of W *track bundles*, where bundle i consists of l_i staggered tracks each with length l_i segments.

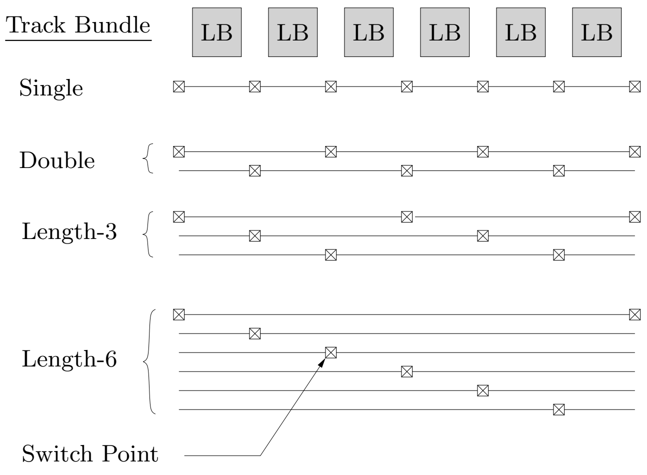


Figure 1: An example of interconnect segmentation.

Area, Delay, and Power Estimation

Evaluating the performance metric of TORCH defined in Section 3 requires estimating the FPGA area, as well as the delay and power for each placed and routed benchmark design in a given technology node. The methodology used to estimate area, delay, and power is the same as that in [6]. We briefly review this methodology here and define interconnect delay and power used in TORCH.

As in [6], to estimate FPGA layout area, we estimate the area of a tile consisting of an LB and a connection and switch boxes. The tile area is estimated by decomposing it into components similar in granularity to standard-cell library elements, estimating the area of each component in λ^2 with a stick diagram and the Magic-8 rules, and adding up the estimated component areas.

To estimate delay and power we use the transistor and metal wire RC models shown in Figure 2 and consider 5 technology nodes: 130nm, 90nm, 65nm, 45nm, and 32nm. The model parameters given in Table 1 are estimated for these technology nodes using the Berkeley Predictive Technology Models and HSPICE [18, 19] .

We define interconnect delay for a placed and routed design as the geometric average of all its pin-to-pin net delays, not including LB delay [1]. As in [6], we first use RC models for the interconnect segments and Elmore delay to optimize the connection and switch box device sizes as well as the number and sizes of the buffers for the Length-3 and Length-6 segments for a given FPGA array size in each technology node. We then use a modified version of the VPR delay calculation function to compute net delays.

We define interconnect power for a placed and routed design as the total equivalent capacitance of all nets. As in [6], this is computed using added code to VPR. Note that frequency and power supply voltage are not included in the definition of power because we only consider power of a design relative to that in a baseline FPGA.

[1]The critical path delay can be readily used instead of the geometric average pin-to-pin net delay.

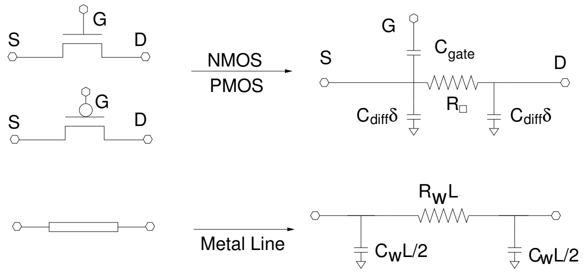


Figure 2: RC circuit model for CMOS transistors and metal wires. C_{gate} is the equivalent transistor gate capacitance (in fF/$\square$), C_{diff} is the transistor diffusion capacitance (in fF/μm), $R_\square$ is the transistor channel resistance (in $\Omega/\square$), C_{w} is the metal wire capacitance (in fF/mm), R_{w} is the metal wire resistance (in Ω/mm), and L_{w} is the length of a metal wire.

	130nm	90nm	65nm	45nm	32nm
V_{dd}(V)	1.3	1.2	1.1	1.0	0.9
L_{eff}(nm)	49	35	24.5	17.5	12.6
C_{gate}(fF/μm)	1.73	1.59	1.32	1.24	1.11
C_{diff}(fF/μm)	1.13	1.09	1.08	1.03	1.01
$R_\square$(kΩ/ $\square$)	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

Table 1: Transistor and metal wire parasitics for five technology nodes.

3. TORCH

The input to TORCH is:

- An initial FPGA architecture $\mathcal{A}$ that uses a *baseline* segmentation. The choice of the baseline segmentation is arbitrary and is needed only to normalize the power and delay for each benchmark design. An example baseline segmentation is given in Section 4.
- Technology node parameters: These include interconnect device sizes and RC parameters for delay and power calculation in VPR (see Table 1).
- A set of m benchmark designs $\mathcal{B}$.

The output of TORCH is an optimized segmentation s^*.

Given the FPGA architecture $\mathcal{A}$ with a segmentation s, VPR generates the routing graph $g(\mathcal{A}, s)$ [17]. 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 baseline 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

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

We are now ready to describe TORCH. The top-level algorithm of TORCH given in Algorithm 1 is based on simulated annealing. First, the benchmark designs are placed and routed using VPR in the FPGA with the baseline segmentation. The delay and power estimates $\{(p_{b0}, d_{b0} : b = 1, 2, \ldots, m\}$ are computed. A random segmentation is chosen and its routing graph is generated. The designs are then routed assuming the random segmentation and an initial value of the performance metric is evaluated. An initial temperature for simulated annealing is set. After a new segmentation is selected, the designs are incrementally rerouted, and the performance metric is re-evaluated. If the metric value is reduced, the segmentation is accepted, otherwise it is accepted with a probability that depends on the increase in the value of the metric and temperature. The process of changing segmentation, 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. To further describe the algorithm of TORCH, we illustrate its structure in Figure 3(a). Our experiments have shown that the most computationally intensive part of TORCH is the subroutine `EvaluateCost()`, which involves placement/routing and computing delay-power for all benchmark circuits. Fortunately, because evaluating each circuit design is independent, this part of the algorithm can be easily parallelized, which can significantly reduce the total running time of TORCH. We illustrate this idea in Figure 3(b).

In the following we describe some of the functions referred to in Algorithm 1 in more detail.

NewSegmentation()

Figure 4 shows how the segmentation is changed in each trial. 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.

ExitCriterion() & InnerLoopCriterion()

`ExitCriterion()` makes sure the freeze_count is less than a predetermined number 100. `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.

IncrementalRoute()

A key ingredient of TORCH is the incremental routing algorithm. This is important because performing complete routing for each trial would be computationally prohibitive. Because the change in segmentation from one trial to the next is very small, on average it affects only 5% of the routed nets. By ripping out the affected nets and rerouting them using the new segmentation, we save significant amount of computing time. `IncrementalRoute()` uses the ripping and rerouting nets part of VPR [20] and is described in Algorithm 2. The definitions of the variables used in Algorithm 2 are as follows:

- RN_k is the set of routing graph nodes associated with track bundle k in all channel.

Algorithm 1 TORCH

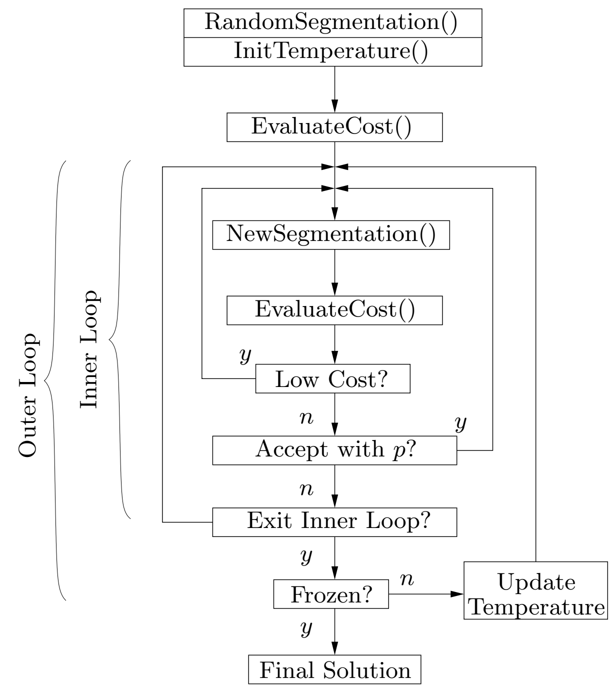
```
1: $s \leftarrow$ RandomSegmentation()
2: $T \leftarrow$ InitialTemperature()
3: $g \leftarrow$ g($\mathcal{A}$, $s$)
4: freeze_count $\leftarrow 0$
5: while ( ExitCriterion() is FALSE) do
6:    changes $\leftarrow 0$
7:    trials $\leftarrow 0$
8:    $c \leftarrow$ EvaluateCost($g, \mathcal{B}$)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

- AN_k is the set of nets affected by the change of track bundle k segmentation.
- A_{ij} is the criticality of the connection from the source of net i to one of its sinks j.
- d_n is the intrinsic delay of routing node n.
- p_n is the present congestion cost of node n.

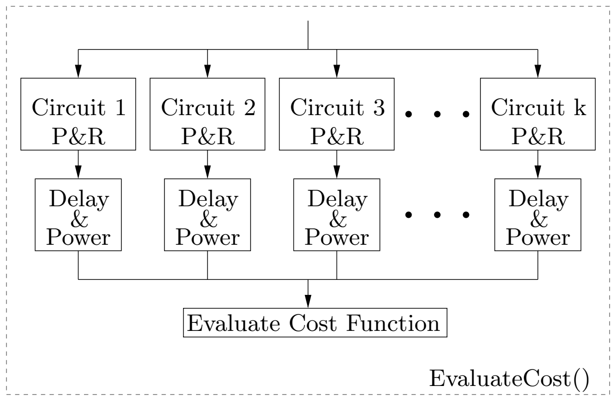
Algorithm 2 first finds RN_i (lines 1 to 6). Next, it constructs AN_i (lines 7 to 12). The nets in AN_i are then routed using routines from VPR [17].

4. EXPERIMENTAL RESULTS

In this section we describe experiments using TORCH and the largest 20 MCNC benchmark designs. We choose a LB array size of 52×52, which accommodates all the designs with minimum utilization of 12%. We use $F_c = 0.5W$ and $F_s = 3$. We ran experiments with uniform all one segmentation to determine the minimum channel width needed for each design. Based on these experiments we set $W = 56$. For baseline segmentation, we assume 18 Single, 16 Double, 10 Length-3, and 12 Length-6 segments. Table 2 lists the pass-transistor and buffer sizes used for each of the five technology nodes. As discussed in Section 1 , to save the running



(a)



(b)

Figure 3: Flow diagram of (a) Algorithm 1 and (b) parallel implantation of the EvaluateCost() function.

time of TORCH, we perform both incremental routing and infrequent placement. Our experimental results have shown that on average, an incremental routing can reduce the running time of a complete rerouting by about 25 times. Additionally, we only perform placement every 20 segmentation updates.

Figures 5 show the improvement in interconnect power and delay for the 20 designs using the TORCH segmentation, relative to the baseline segmentation and the all-one segmentation, for 45nm 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 baseline segmentation are 22% and 15%, respectively. The PC machine used to conduct the experiments has an AMD Athlon(tm) 64×2 Dual Core Processor 5200+ with 2 Gigabytes memory and clock frequency of 2.6 GHz. The typical

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 $i$
38:   end for
39: end while
```

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

Table 2: Pass-transistor and buffer sizes for FPGA interconnects for different technology nodes. b_i: LB input buffer size; b_o: LB output buffer size; x: PT size from LB output to CB; y: PT size from interconnect to LB input; l_N: Number of buffers inserted in a long interconnect; m_l: SP buffer size for interconnect of length l, n_l: Size of inserted buffer in long interconnect.

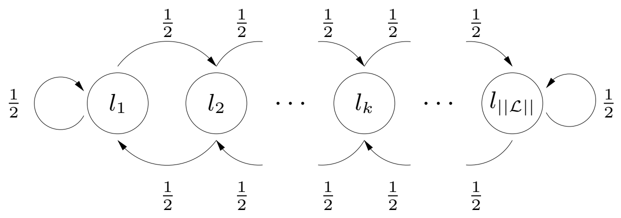


Figure 4: State transition diagram for NewSegmentation() in Algorithm 1.

running time for one run of TORCH is around 29 hours. As illustrated in Figure 3-(b), the run time can be significantly reduced by parallelizing the implementation of the EvaluateCost function. The improvements relative to the all one segmentation are 25% and 8%, respectively. Figure 6 shows the TORCH segmentation, which has an average segment length of 1.91 versus 2.71 for the baseline.

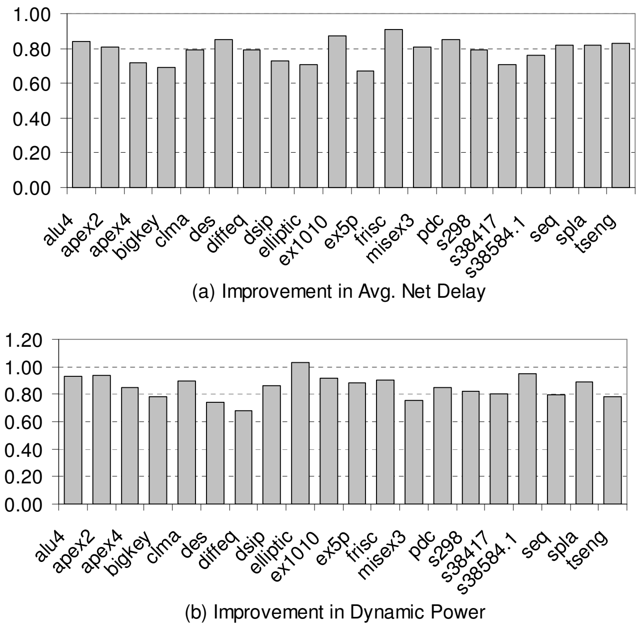


Figure 5: Improvements in power and delay relative to baseline segmentation.

Technology Scaling

In [6], 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. Figure 7 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 130nm to 1.88 at 32nm. These results corroborate the observation in [6].

Switch Box Width

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

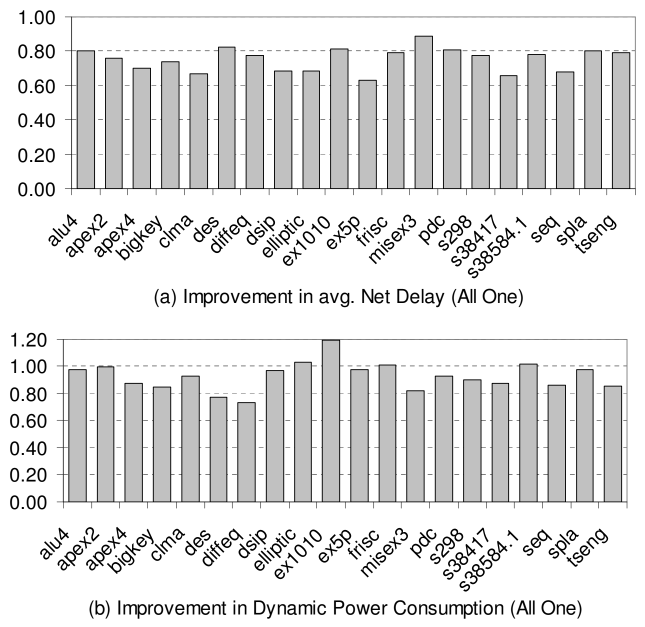


Figure 6: Improvements in power and delay relative to all-one segmentation.

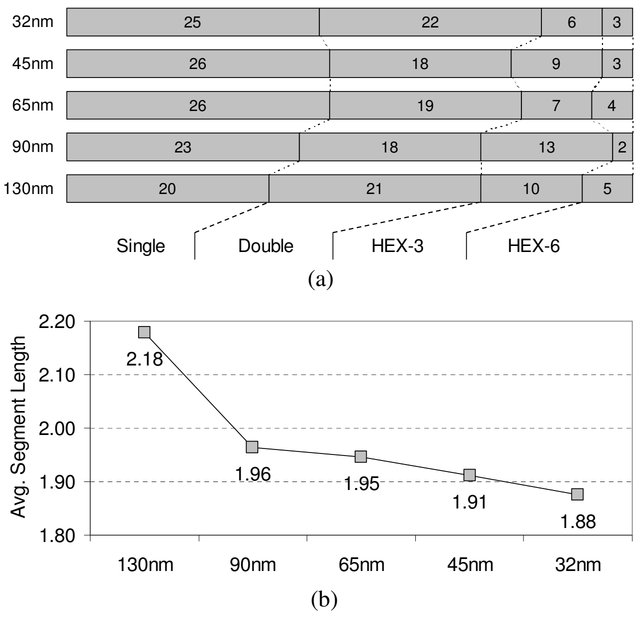


Figure 7: (a) Segmentation results for different technology nodes. (b) Average segment length for different technology nodes.

TORCH can be used to select the switch box/channel width. Figure 8 plots the average interconnect delay and power obtained by TORCH in 45nm technology with $\alpha = \beta = 1$ for different switch box widths together with the corresponding estimates of the FPGA area relative to the FPGA with baseline segmentation. Note that average delay first drops from 0.87 at $W = 40$ to 0.76 at $W = 60$, then remains roughly unchanged. This is because when W is too small, many nets are routed in a highly suboptimal manner resulting in increased delay. As W increases, nets are more optimally routed, which decreases delay. This decrease in delay diminishes as W becomes too large. Power also first decreases as W is increased, but then begins to increase as W 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 $W \approx 60$ is the best choice.

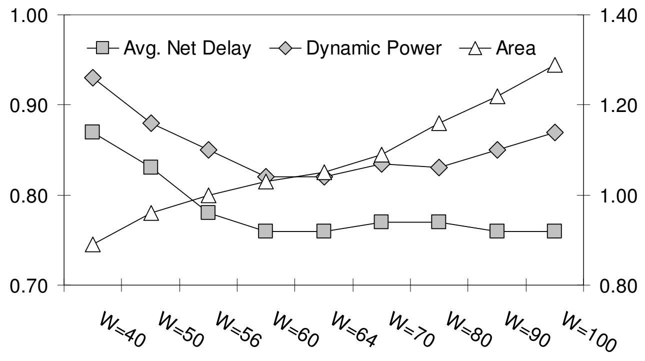


Figure 8: Delay, power, and area improvements for different switch box width.

Power Delay Tradeoff

The performance metric used in TORCH allows for a tradeoff between power and delay. Figure 9 plots the average segment length for different choices of α and β in 45nm technology. As expected, average segment length increases as delay is emphasized more than power. The average power and delay are plotted in Figure 10.

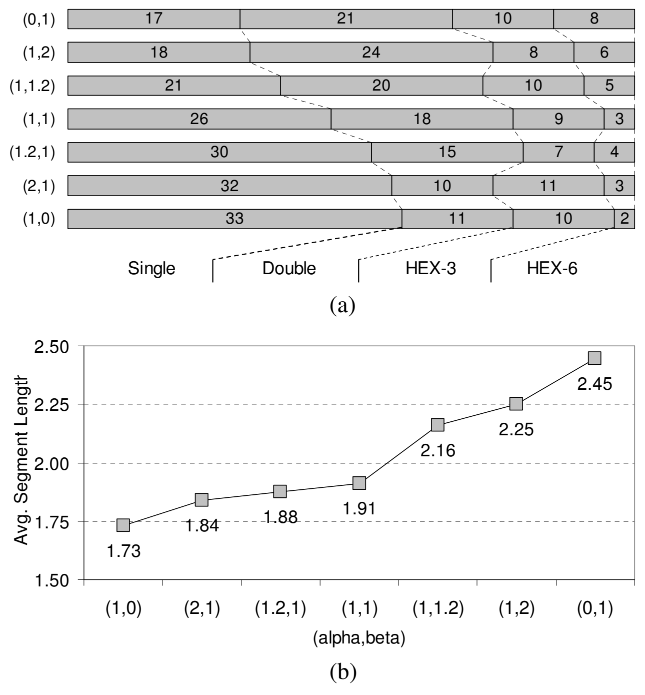


Figure 9: Routing channel segmentation results for different α and β at 45nm technology node.

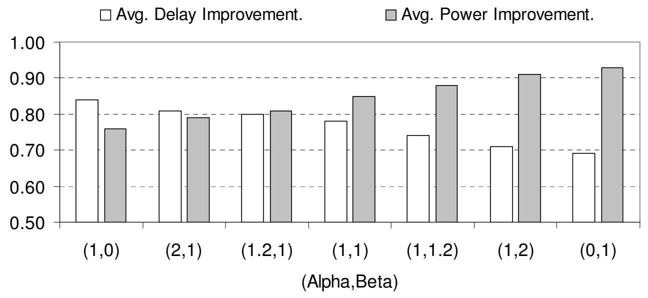


Figure 10: Delay and power improvements for different α and β at 45nm technology.

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 $\{1, 2, \ldots, 8\}$ and ran TORCH with $\alpha = \beta = 1$ and 45nm technology. Figure 11 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 6% and power is improved by 7%. The estimated FPGA area is, however, increased by a factor of 0.12. The run time of the tool is also significantly longer.

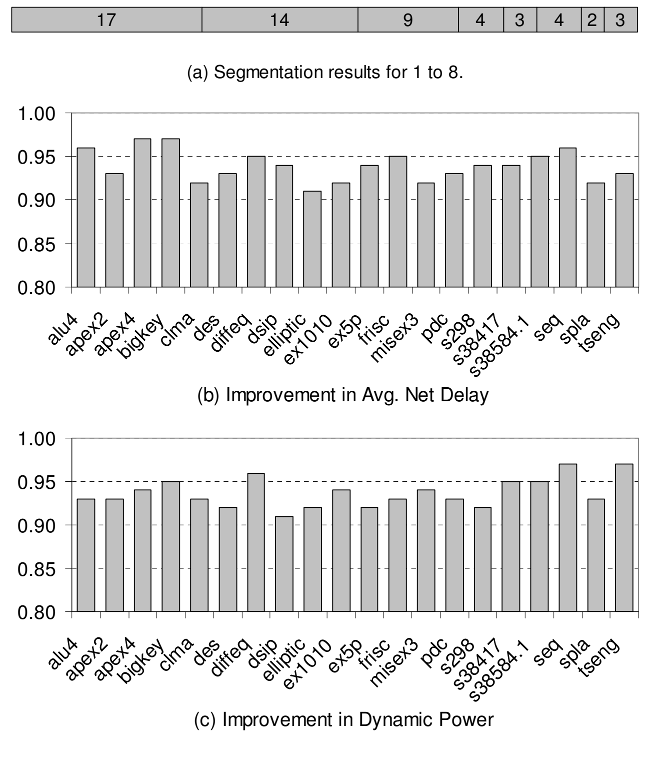


Figure 11: Reductions in delay and power using segment length set $\{1, 2, \ldots, 8\}$ relative to the FPGA with optimized segmentation for segment length set $\{1, 2, 3, 6\}$.

5. CONCLUSION

We described a design tool for routing channel segmentation in island-style FPGAs. Given the FPGA architecture parameters and a set of benchmark designs, TORCH uses a simulated annealing procedure to find an optimized segmentation based on an average delay-power product. In each iteration, segmentation is incrementally changed, the benchmark designs are mapped into the FPGA with the new segmentation using VPR, the performance metric is updated, and the new segmentation is either accepted or rejected. Run time is significantly reduced by performing placements infrequently, performing only incremental routing in each iteration, and parallelizing the metric evaluation. TORCH outputs both an optimized mix of track segment lengths and an optimized *ordering* of the segmented tracks in the channel.

We demonstrated TORCH experimentally and showed that significant improvements in delay and power can be achieved by optimizing segmentation. We used TORCH to validate our observation in [6] that average segment length should decrease with technology scaling. We also showed how TORCH can be used to optimize switch box width. Other routing architecture parameters such as connection and switch box flexibilities and buffer sizes can be similarly optimized.

Although TORCH assumes an island-style FPGA and uses VPR for placement and routing, it can be readily adapted to any FPGA architecture, any placement and routing tool, and any performance metric based on placed and routed benchmark designs.

6. REFERENCES

[1] V. Betz, J. Rose, and A. Marquardt, eds., *Architecture and CAD for Deep-Submicron FPGAs*. Norwell, MA, USA: Kluwer Academic Publishers, 1999.

[2] I. Kuon and J. Rose, “Measuring the gap between FPGAs and ASICs,” in *Proceedings of the 2006 ACM/SIGDA Tenth International Symposium on Field-Programmable Gate Arrays*, pp. 21 – 30, 2006.

[3] Actel, Inc., “Automotive ProASIC3 flash family FPGAs datasheet,” March 2007.

[4] Xilinx, Inc., “Virtex-II Pro / Virtex-II Pro X complete data sheet (all four modules),” March 2007.

[5] D. Lewis, E. Ahmed, G. Baeckler, V. Betz, M. Bourgeault, D. Cashman, D. Galloway, M. Hutton, C. Lane, A. Lee, P. Leventis, S. Marquardt, C. McClintock, K. Padalia, B. Pedersen, G. Powell, B. Ratchev, S. Reddy, J. Schleicher, K. Stevens, R. Yuan, R. Cliff, and J. Rose, “The stratix II logic and routing architecture,” in *Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*, pp. 14 – 20, 2005.

[6] M. Lin, A. El Gamal, Y.-C. Lu, and S. Wong, “Performance benefits of monolithically stacked 3-D FPGA,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, pp. 216–229, Feb. 2007.

[7] A. E. Gamal, J. Greene, and V. Roychowdhury, “Segmented channel routing in nearly as efficient as channel routing (and just as hard),” in *Proceedings of the 1991 University of California/Santa Cruz conference on advanced research in VLSI*, (Cambridge, MA, USA), pp. 192–211, MIT Press, 1991.

[8] K. Zhu and D. F. Wong, “On channel segmentation design for row-based FPGAs,” in *ICCAD ’92: Proceedings of the 1992 IEEE/ACM international conference on computer-aided design*, (Los Alamitos, CA, USA), pp. 26–29, IEEE Computer Society Press, 1992.

[9] M. Pedram, B. Nobandegani, and B. Preas, “Design and analysis of segmented routing channels for row-based FPGAs,” *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 13, pp. 1470–1479, Dec. 1994.

[10] W.-K. Mak and D. F. Wong, “Channel segmentation design for symmetrical FPGAs,” in *Proceedings of the 1997 International Conference on Computer Design*, pp. 496–501, 1996.

[11] E. Bozorgzadeh and M. Sarrafzadeh, “Customized regular channel design in FPGAs,” 1999.

[12] Y.-W. Chang, J.-M. Lin, and M. Wong, “Matching-based algorithm for FPGA channel segmentation design,” *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 20, pp. 784–791, June 2001.

[13] V. Betz and J. Rose, “FPGA routing architecture: segmentation and buffering to optimize speed and density,” in *Proceedings of the 1999 ACM/SIGDA Seventh International Symposium on Field-Programmable Gate Arrays*, pp. 59 – 68, 1999.

[14] R. Tu and B.-X. Shao, “Energy/performance/area tradeoffs in nanometer FPGA segmented routing architecture,” in *Solid-State and Integrated Circuit Technology, 2006. ICSICT ’06. 8th International Conference on*, pp. 1954–1956, 2006.

[15] K. Compton and S. Hauck, “Totem: Custom reconfigurable array generation,” in *Proceedings of the 9 Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pp. 1–9, 2001.

[16] G. Lemieux and D. Lewis, “Circuit design of routing switches,” in *Proceedings of the 2002 ACM/SIGDA Tenth International Symposium on Field-Programmable Gate Arrays*, pp. 19 – 28, 2002.

[17] V. Betz and J. Rose, “VPR: A new packing, placement and routing tool for FPGA research,” in *Proceedings of the 7th International Workshop on Field-Programmable Logic and Applications*, pp. 213 – 222, 1997.

[18] W. Zhao and Y. Cao, “New generation of predictive technology model for sub-45nm design exploration,” in *ISQED ’06: Proceedings of the 7th International Symposium on Quality Electronic Design*, (Washington, DC, USA), pp. 585–590, IEEE Computer Society, 2006.

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

[20] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, 1990.