Towards Layout-Friendly High-Level Synthesis

Jason Cong[1,2,3] Bin Liu[1] Guojie Luo[2,3] Raghu Prabhakar[1]
[1] Computer Science Department, University of California, Los Angeles
[2] Center for Energy-Efficient Computing and Applications (CECA), Peking University
[3] UCLA/PKU Joint Research Institute in Science and Engineering
{cong,bliu,raghu}@cs.ucla.edu gluo@pku.edu.cn

ABSTRACT

There are two prominent problems with technology scaling: increasing design complexity and more challenges with interconnect design, including routability. High-level synthesis has been proposed to solve the complexity problem by raising the abstraction level. In this paper, we share our vision that high-level synthesis can potentially help the routability problem as well. We show that many interconnect problems that occur in layout can be avoided or mitigated by adopting a layout-friendly RTL architecture generated from high-level synthesis. We also evaluate some structural metrics that can be used to estimate the routability impact of design decisions in high-level synthesis. Experimental results have demonstrated correlations between the metrics and the routability of the resulting design.

Categories and Subject Descriptors

B.6.3 [**Logic Design**]: Design Aids—*automatic synthesis, optimization*; G.3 [**Mathematics of Computing**]: Probability and Statistics—*correlation and regression analysis*

General Terms

Design, Experimentation

Keywords

High-Level Synthesis, Routability, Interconnect Estimation

1. INTRODUCTION

Technology scaling has led to the increasing difficulty in resolving interconnect problems. This is due to relatively scarce routing resources and large interconnect delays. Various efforts to solve these problems have reshaped almost every aspect of the IC industry in the past twenty years. Advanced process technologies have offered new materials and more metal layers. 3-D integration provides a further opportunity to reduce the length of interconnects. In the EDA community, early work mainly focused on congestion optimization during placement and routing [8,26,28,30]. As interconnect problems become worse, it is insufficient to repair routability failures only during physical design. With the prevalence of RTL-based design flows, early interconnect estimation and optimization techniques during logic synthesis have been proposed, [24,25,27,31]. On the designer side, optimization of global interconnects is the central task of design planning and architecture definition. New design methodologies and styles have been established to address interconnect challenges [6,9,11,33].

Along with technology scaling, another trend is the rapid increase of complexity in system-on-chip designs. This has encouraged the design community to seek better productivity. Electronic system-level design automation has been widely identified as the next productivity boost for the semiconductor industry, where high-level synthesis (HLS) plays a central role, by enabling the automatic synthesis of high-level, untimed or partially timed specifications (in languages such as C/C++, SystemC, Matlab) to cycle-accurate RTL models. These RTL models can then be accepted by the downstream RTL synthesis flow for implementation.

HLS has been an active research topic for more than 30 years. Early attempts to deploy HLS tools began when RTL-based flows were well adopted. In 1995, Synopsys announced Behavioral Compiler, which accepts behavioral HDL code and connects to downstream flows. Similar tools include Monet from Mentor Graphics and Visual Architect from Cadence. This wave of tools received wide attention, but failed to widely replace RTL design. This is partly ascribed to the use of behavioral HDLs, which are not popular among algorithm and system designers and require steep learning curves. Since 2000, a new generation of HLS tools has been developed in both academia and industry. Unlike their predecessors, many of them use C-based languages for design capture. This makes them more accessible to algorithm and system designers. It also enables hardware and software to be specified in the same language, facilitating software/hardware co-design and co-verification. The use of C-based languages also makes it easy to leverage new techniques in software compilers for parallelization and optimization. As of 2012, notable commercial C-based tools include Cadence C-to-Silicon Compiler, Calypto Catapult C (formerly a product of Mentor Graphics), NEC CyberWorkBench, Synopsys Synphony C (formerly a product of Synfora, and originating from the HP PICO project), and Xilinx AutoESL (originating from the UCLA xPilot project [10]). More detailed surveys on the history and progress of HLS are available from [15,21].

In our experimental HLS system based on xPilot, compiler transformations are first performed on the behavioral specification to obtain an optimized intermediate code represented as a control-data flow graph (CDFG). Operation scheduling then assigns operations in the CDFG to control states. The result of scheduling is a finite-state machine with datapath (FSMD), on which binding is applied to allocate resources in the datapath. After that, an RTL netlist can be generated. The basic flow is illustrated in Figure 1.

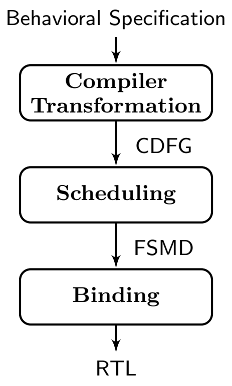

Figure 1: A typical high-level synthesis flow.

Numerous academic research and industry practices have demonstrated the gain in performance, area, and power of implementations obtained by improving the HLS algorithm, or by exploring different options/directives in HLS. This is primarily because decisions at a higher abstraction level often have bigger influences. Thus, we believe that huge opportunity for interconnect optimization exists in HLS. Instead of expanding time and effort to fix problems and making various compromises for a given netlist, the designer could use a HLS tool to generate a *layout-friendly* RTL netlist, which is easier for downstream tools to implement.

In this paper we show the impact of HLS on routability in Section 2. We then describe some structural metrics that can be used in the synthesis engine (i.e., scheduling and binding) in Section 3 and experimentally evaluate their effectiveness in Section 4. Discussions on future directions are presented in Section 5.

2. IMPACT OF HIGH-LEVEL SYNTHESIS

In this section we experimentally study the impact of high-level synthesis on routability. Our study is based on the xPilot HLS tool [10]. Separate studies are performed for the synthesis engine and for compiler transformations.

2.1 Routability Evaluation Flow

In order to evaluate routability, we feed the RTL netlist under evaluation into an implementation flow, which includes the stages of (i) RTL elaboration, (ii) logic synthesis, (iii) placement, and (iv) routing.

For a given netlist, its routability mainly depends on the amount of routing resources on the target platform. Generally, the target platform is either ASIC-style or FPGA-style. The routing resources of an ASIC-style platform are determined by the number of metal layers and the wire pitches. The routing resources of an FPGA-style platform, assuming the architecture model in [4], are determined by the number of tracks between the configurable logic blocks (CLBs).

In this paper we mainly focus on an FPGA-style routability evaluation flow, illustrated in Figure 2. We use Altera Quartus II version 9.1 [1] as an RTL frontend, and then use ABC version 70731 [2] to synthesize and map the netlist into 4-input lookup tables (4-LUTs). The mapped netlist is packed into CLBs by T-VPACK 5.0.2 [3], where each CLB contains ten 4-LUTs and ten flip-flops. We adopt a simplified routing architecture from [4]. The CLBs form a regular array: the space between two neighboring CLB rows or columns is called a channel, and the space between two neighboring CLBs is called a segment. There are multiple routing tracks in the segments, where we assume the span of a track is one CLB, and the number of tracks (channel width) can be different for different segments. The packed netlist is then placed by VPR 5.0.2 [3] with the total bounding-box wire length as the objective. Routing is also done by VPR, which minimizes the maximum channel width using a binary search.

It is obvious that a design will not be routable if the channel width is too small. Thus we consider maximum channel width (CW_max) and average channel width (CW_avg) as indicators for routability. In addition, total wire length (WL_tot) and average wire length (WL_avg) are also used in the evaluation.

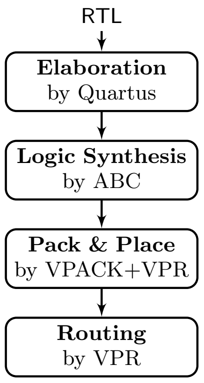

Figure 2: The RTL implementation flow.

2.2 Impact of the Synthesis Engine

To demonstrate how decisions in the synthesis engine impact routability, we generate multiple RTL models from the same CDFG by varying strategies in scheduling and binding. For scheduling, we adjust the optimization objective as well as resource constraints to obtain different FSMDs. The strategies in scheduling are listed in Table 1. Similarly, different objectives and constraints listed in Table 2 are specified for binding to generate different RTL netlists.

By combining each strategy for scheduling with each strategy in binding, we can obtain 60 different strategies in the synthesis engine. In practice, different strategies can sometimes lead to equivalent or identical RTL netlists. We extract several computation-intensive kernels as test cases from DSP applications. These cases generally perform many multiplications and addition/subtractions. For each test case, we measure the routability for all RTL models generated using different synthesis strategies, and report the minimum and maximum values for each metric in Table 3. The results indicate that different RTL models generated from the same CDFG can have drastically different routability.

Table 1: Scheduling strategies.

	objective	constraint
1	ASAP	None
2	ALAP	None
3	MINREG	None
4	ALAP	$M = \lceil 0.25 \times m \rceil$
5	ALAP	$M = \lceil 0.25 \times m \rceil$, $A = \lceil 0.4 \times a \rceil$
6	MINREG	$M = \lceil 0.1 \times m \rceil$, $A = \lceil 0.2 \times a \rceil$

The ASAP/ALAP objective tries to schedule operations as soon/late as possible, subject to optional resource constraints [17]; the MINREG objective tries to minimize the total lifetime of variables in order to reduce registers. M and A are the number of available multipliers and adders in the resource constraints, respectively; m and a are the number of multiplication and addition operations in the CDFG, respectively.

Table 2: Binding strategies.

	objective	constraint
1	total area	None
2	total area	$mux\_input \leq 4$
3	register	$mux\_input \leq 4$
4	multiplier	None
5	multiplier	$mux\_input \leq 4$
6	multiplier and register	None
7	multiplier and register	$mux\_input \leq 4$
8	multiplier and adder	None
9	multiplier and adder	$mux\_input \leq 4$
10	multiplier and adder and register	$mux\_input \leq 4$

Multiplier/adder/register in the objective means minimizing the number of corresponding components. *mux_input* is the maximum number of inputs for each multiplexer; here we try to avoid large multiplexers by setting an upper bound on *mux_input*.

2.3 Impact of Compiler Transformations

In fact, before CDFG is generated for scheduling and binding, the compiler front-end often applies a sequence of transformations on the intermediate representation. Some of the transformations are the same as ones found in conventional compilers, like dead code elimination; yet others are specifically designed for HLS. The performance implications of many transformations are well understood. Here we make a few observations about their impact on routability in the context of HLS.

We note that the following scalar transformations tend to reduce interconnect complexity.

- Expression simplification. For example, constant propagation and strength reduction can replace complex functional units with simpler ones.
- Expression structure optimization. For example, common sub-expression extraction, redundancy elimination and re-association can reduce the number of complex functional units and change the local structure of the datapath.
- Bitwidth optimization. Reducing the bitwidths of operands and results can reduce component sizes and the number of wires. This optimization is particularly useful for HLS.

Table 3: Routability measurements.

design	CW_max	CW_avg	WL_tot	WL_avg
test1	46/86	30/44	36K/166K	9/12
test2	60/98	38/47	61K/251K	11/13
test3	40/70	25/33	25K/103K	7/9
test4	52/86	31/40	47K/196K	10/12
test5	62/122	39/55	94K/562K	12/17

a/b indicates that a is the minimum value and b is the maximum value, among all results generated for the test case.

The following transformations often have more global influences.

- Transformations that change memory organization. In the implementation of xPilot, each array is mapped to a dedicated memory block. Transformations like array partitioning, array mapping, array reshaping can change the number of memory blocks, the number of ports on each memory block, as well as the way memory blocks connect to other components. For example, partitioning an array can increase throughput, but it often leads to more decoding/multiplexing logic and more interconnects [13].
- Loop transformations. Unrolling a loop creates opportunities for code optimization and parallelization; yet it often destroys the regular structure of the data transfers between different loop iterations and thus leads to more components and interconnects.
- Function-level transformations. In xPilot, each function is implemented as a hardware module. When a function is inlined, further optimization may be performed at call sites; on the other hand, when the function calls are optimized differently at different call sites, it becomes difficult for them to share the same set of components and interconnects.

Here we perform a case study on a simple design that multiplies two 8×8 32-bit integer matrices. The matrix multiplication is implemented straightforwardly as a three-level nested loop: the inner loop computes an element in the resulting matrix, the middle loop computes a row, and the outer loop computes the entire result. We generate three solutions.

- Solution 1: no loop unrolling.
- Solution 2: unroll the inner loop completely, and partition the two input matrices into row/column vectors.
- Solution 3: unroll the middle loop and the inner loop completely, partition the two input matrices completely (into scalars), and partition the output matrix into column vectors.

The three solutions are synthesized using default synthesis options and the resulting routability result is reported in Table 4. It is evident from the result that unrolling increases interconnect complexity drastically for this case. Note that a solution with higher interconnect complexity also has higher performance; thus a tradeoff between performance and routability needs to be considered when making decisions on loop unrolling.

Table 4: Routability for matrix multiplication.

solution	CW_max	CW_avg	WL_tot	WL_avg
1	30	18	4543	5
2	44	24	43610	7
3	80	36	223042	10

3. ROUTABILITY ESTIMATION

A frequent problem of optimizing at a higher level is the lack of good estimators: while there are many alternative RTL structures to explore in HLS, it is hard to decide whether one is superior to another. This is particularly true for interconnect optimization, because unlike other metrics (such as latency, throughput, area) which can be estimated to a reasonable accuracy for a given RTL model, the length and density of interconnects can hardly be decided without a layout. While different downstream tools can generate different layouts, the global structure of the block-level netlist plays an important role in deciding what a good layout looks like. The first step toward layout-friendly high-level synthesis is to be able to evaluate the layout-friendliness of an RTL netlist. The trivial approach of running through all downstream steps would give the most accurate result; yet it is often impractical when exploring a large number of alternatives. Most of the existing ways to predict routability fall in one of the following categories.

(1) Incorporate a rough layout in high-level synthesis. There are numerous efforts that combine high-level synthesis with floorplanning to help interconnect estimation and optimization [18, 20, 37, 39, 40]. This is quite a reasonable approach. However, since layout itself is a nontrivial problem, implementation of a stable and fast layout engine itself is a challenge.

(2) Use structural metrics to evaluate interconnect complexity. Such metrics are usually derived from a graph representation of the netlist without performing layout. Widely used structural metrics include the total multiplexer inputs [7, 22, 23, 29], the number of global interconnects [12, 32], the total cut size [24], etc. These metrics are often easier to obtain and are more stable (i.e., not dependent on the layout algorithm), but their accuracy is often a concern.

Given a scheduling/binding solution on a CDFG, the RTL netlist can be constructed. The netlist often consists of components (including functional units, registers, memories, multiplexers, pre-synthesized blocks, etc.) and wires which connect components through ports. A directed graph $G = (V, E)$ can be constructed to represent the netlist, where $V = \{1, 2, \ldots, n\}$ is the set of vertices each representing a component; and $E \subseteq V \times V$ is the set of directed edges each representing a net from the source component to the sink component. Note that an edge is present only when there are data transfers between the two components; if two components are connected in the netlist only because they are both sinks of a net, no edge is be created between the corresponding vertex pair.

The following metrics are considered in our evaluation.

- Total number of datapath nets (#nets).
- Total number of multiplexer inputs (#mux_input).
- The average cut size between components in the netlist (AMC). Here the bitwidth of each net is considered. More details are available in Section 3.1.
- The *spreading score* proposed in [14]. More details are available in Section 3.2.

3.1 A Metric Based on Cut Size

Cut size has been recognized as an indicator of interconnect complexity in a number of important synthesis steps, such as partitioning, clustering, and placement. Intuitively, smaller cut size implies less global connections, and thus better routability.

The cut-size minimization process is explicitly applied in every partitioning-based placer (e.g., Capo [34]). In fact, if we divide the placement region into unit squares, the total cut size of a placed netlist on the edges of these squares is approximately equal to the total wire length. Thus, the recursive cut-size minimization process in partitioning-based placers can be viewed as an approximate wire length minimization process. Moreover, the cut size across a local cut line directly captures the local congestion, and maintains better congestion information than the total wire length metric [35].

Kudva, Sullivan and Dougherty propose the *sum of all-pairs min-cut* (SAPMC) to evaluate the *adhesion* of a gate-level netlist, and use it in logic synthesis [24]. In a netlist represented by graph $G = (V, E)$, the cut size of the an s-t min-cut between two distinct vertices s and t is the minimum number of nets whose removal disconnects s and t. The SAPMC is the sum of min-cut sizes for all pairs of distinct vertices in V. Experimental results reported in [24] show a positive correlation between congestion and SAPMC; in addition, the results indicate that the correlation between congestion and the circuit size (as measured by the number of nodes in the netlist) is even stronger, and thus SAPMC is only used to break ties in optimization. Clearly, SAPMC is also positively correlated with the circuit size, because larger circuits naturally have more distinct node pairs, which lead to larger SAPMC. In an effort to obtain orthogonal metrics, we try to exclude the node count factor from SAPMC, and instead compute the *average min-cut* (AMC), by dividing SAPMC with $n(n-1)/2$, where n is the number of nodes.

3.2 A Metric Based on Graph Embedding

This section describes spreading score, a structural metric proposed in [14]. The metric is based on graph embedding, and can be computed efficiently using convex programming. Here we give a brief review, more details about the metric can be find in [14].

A layout of the netlist can be regarded as an embedding of G in the 2-dimensional Euclidean space $\mathbb{R}^2$. We associated each vertex i with a column vector $p_i = (x_i, y_i)^T$ to represent its position in the embedding. The length of the connection $(i, j) \in E$ can be measured as the distance in $\mathbb{R}^2$, i.e., $\|p_i - p_j\| = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$.

We consider the following optimization problem.

$$\begin{array}{rl} \text{maximize} & \sum_{i=1}^{n} w_i \|p_i\|^2 \\ \text{subject to} & \sum_{i=1}^{n} w_i p_i = 0 \\ & \|p_i - p_j\| \le l_{ij} \quad \forall (i,j) \in E \end{array} \qquad (1)$$

Here $w = (w_1, w_2, \ldots, w_n)^T$ is the non-negative weight vector with w_i being the area of component i; l_{ij} is the maximum allowed length for the wire connecting i and j. The objective function measures how far components are spread from their weighted center of gravity, using a weighted 2-norm of the distance vector. Thus the problem in Equation

1 is to maximize component spreading, under the constraint that the length every connection $(i,j) \in E$ does not exceed l_{ij}.

With proper selection of l_{ij}, we expect that the optimal value of the problem in Equation 1 can be used to evaluate the layout-friendliness of a netlist. This is based on the following observation: if components in a netlist can spread over the chip region without introducing long wires, it will be easy to remove overlaps between components and obtain a layout with small wire length, and thus less congestion. This can be empirically verified using popular hand-designed interconnect topologies. For example, mesh and ring can all spread apart without long interconnects, and they are regarded as scalable layout-friendly topologies; on the other hand, spreading the full crossbar or hypercube on the 2D plane inevitably introduces long interconnects, and these topologies are generally much more expensive in interconnect cost. In our experiments, we set the distance l_{ij} based on estimated sizes of components as $l_{ij} = \sqrt{w_i} + \sqrt{w_j}$.

It is difficult to solve the problem in Equation 1 directly, because maximizing a convex function is generally NP-hard (note that minimizing a convex function is easy). We hereby propose a tractable relaxation.

Consider the graph G with n vertices. We use a $2 \times n$ matrix $P = (p_1, p_2, \ldots, p_n)$ to represent its embedding in $\mathcal{R}^2$, i.e.,

$$P = \begin{pmatrix} x_1 & x_2 & \cdots & x_n \\ y_1 & y_2 & \cdots & y_n \end{pmatrix}. \tag{2}$$

Let $Q = P^T P$. Then Q is a symmetric semidefinite matrix with a rank of at most 2, and

$$Q_{ij} = p_i^T p_j = x_i x_j + y_i y_j. \tag{3}$$

We can use Q as variables in the formulation in Equation 1 without losing any useful information, because p can be reconstructed from Q with Cholesky decomposition.

Using Equation 3, we can rewrite the objective and constraint functions in Equation 1 as follows.

$$\textstyle\sum_{i=1}^n w_i \|p_i\|^2 = \sum_{i=1}^n w_i Q_{ii} = \langle \mathrm{diag}(w), Q \rangle \tag{4}$$

$$\left\| \textstyle\sum_{i=1}^n w_i p_i \right\|^2 = \textstyle\sum_{i=1}^n \sum_{j=1}^n w_i w_j Q_{ij} = \left\langle w w^T, Q \right\rangle \tag{5}$$

$$\|p_i - p_j\|^2 = Q_{ii} + Q_{jj} - 2Q_{ij} = \left\langle K^{ij}, Q \right\rangle \tag{6}$$

Here $\mathrm{diag}(w)$ is the $n \times n$ diagonal matrix with w on its diagonal. e_i is the ith standard basis vector in $\mathcal{R}^n$ and $K^{ij} = (e_i - e_j)(e_i - e_j)^T$. $\langle X, Y \rangle$ is the Frobenius inner product of matrices X and Y, i.e.,

$$\langle X, Y \rangle = \textstyle\sum_i \sum_j X_{ij} Y_{ij} = \mathrm{tr}(X^T Y) = \mathrm{tr}(Y^T X). \tag{7}$$

Then we can rewrite the problem in Equation 1 to use Q as variables, and relax the rank constraint on Q.

$$\begin{array}{rl} \text{maximize} & \langle \mathrm{diag}(w), Q \rangle \\ \text{subject to} & \left\langle w w^T, Q \right\rangle = 0 \\ & \left\langle K^{ij}, Q \right\rangle \le l_{ij}^2 \quad \forall (i,j) \in E \\ & Q \succeq 0 \end{array} \tag{8}$$

This problem is convex. In fact, it is a semidefinite programming (SDP) problem. Like linear programs, SDP problems can be solved optimally in polynomial time, and efficient solvers have been developed in recent years. We will not discuss background on convex programming and SDP here. Interested readers may refer to books and survey papers on these topics [5,38].

The problem in Equation 8 essentially asks for an embedding in $\mathcal{R}^n$ instead of $\mathcal{R}^2$, and thus its optimal value is the lower bound of the problem in Equation 1. In our implementation, we adopt this approximation and divide it by n^2 to normalize it. The result is referred to as spreading score. The expectation is that a larger spreading score implies a more layout-friendly datapath structure.

4. RESULT AND ANALYSIS

Results on the test cases in Table 3 are collected. We extract the pre-layout metrics described in Section 3 after HLS, and try to correlate them with post-layout routability data described in Section 2.1.

Single-variable linear regressions between the pre-layout characteristics and maximum channel width are illustrated in Figure 3. From the results, we can see that spreading score has a strong positive correlation with CW_max for several cases (e.g., test1), but shows a weak correlation on test2; #net shows strong positive correlations with CW_max as well; however, total number of multiplexer inputs performs poorly. Surprisingly, AMC tends to have a negative correlation with CW_max, i.e., larger average cut-size leads to smaller channel width. This seems to indicate that AMC is not a primal factor in deciding congestion in the context of HLS. Similar single-variable regressions can be performed for other post-layout metrics. Figure 4 illustrates the results for average wire length. It can be observed that spreading score has a consistently negative correlation with WL_avg, although the correlation is weak for test4 and test5; other metrics seem weaker.

To examine the usability of these metrics, we perform a two-variable polynomial regression with spreading score and AMC as independent variables, and CW_max as the dependent variable. We randomly select 70% of the data to fit a linear and a quadratic function, respectively, and then use the fitted function to "predict" the CW_max for the remaining 30% of cases. The random selections are performed ten times and we collect the average absolute error and the relative error for all the designs, as listed in Table 5. In addition, we obtain regressions for CW_max with spreading score, AMC and #net as independent variables in Table 7, and we obtain regressions for WL_avg with spreading score and AMC as independent variables in Table 6.

On average, the relative error of CW_max fitted with a linear function is less than 10%, while increasing the order of the fitting function would not help to increase the accuracy. For test2, increasing the order even produces a much less accurate fitted function. The data in Table 7 suggest that including more vaiables in the regression slightly reduces the error when we use a linear or a quadratic function.

Table 5: Errors of the polynomial regression for CW_max using spreading score and AMC.

design	linear		quadratic	
	error	%	error	%
test1	3.3	5%	7.6	11%
test2	5.8	8%	10.1	16%
test3	3.3	7%	4.3	9%
test4	4.4	7%	4.6	7%
test5	7.0	11%	5.7	9%

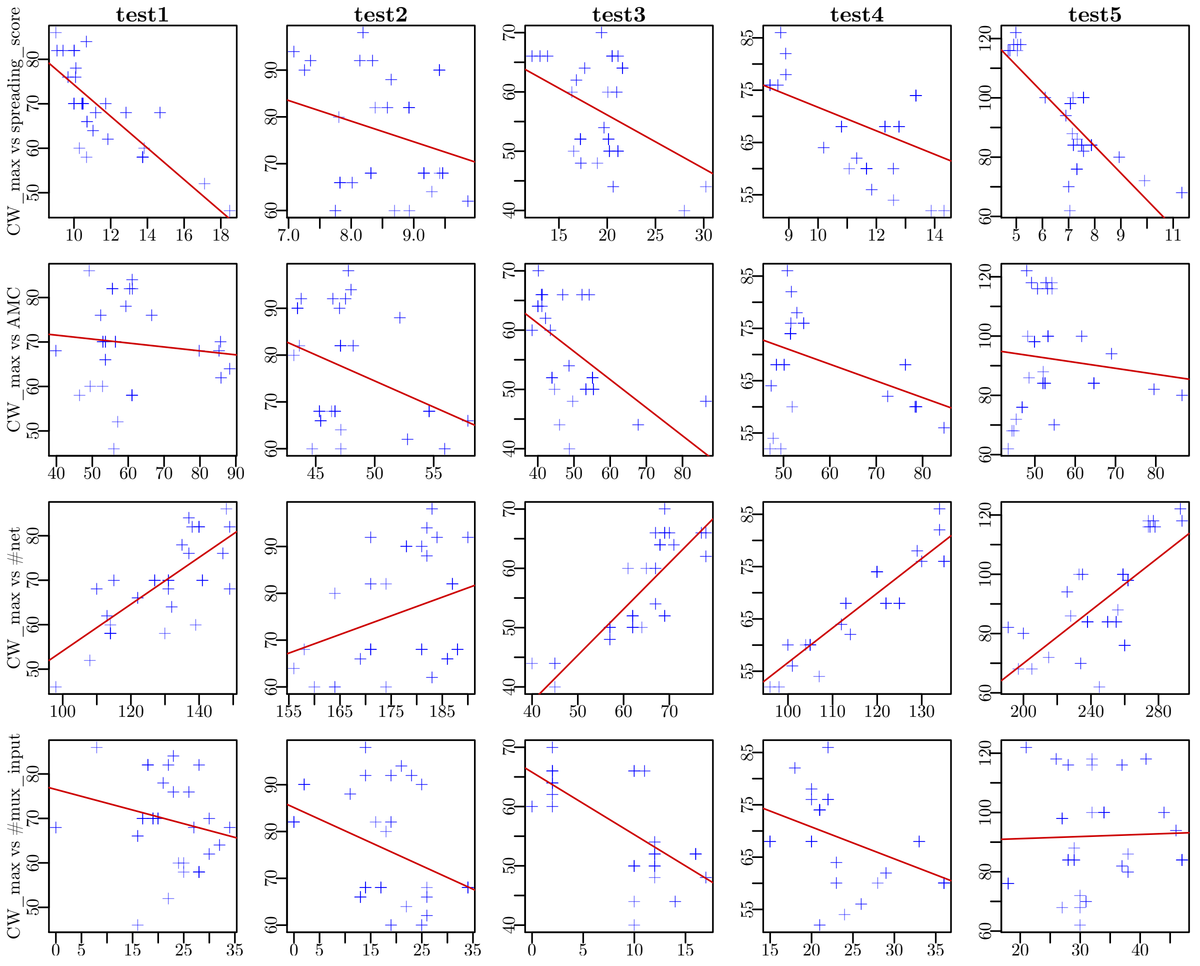

Figure 3: Linear regression of maximum channel width on four high-level metrics.

Table 6: Errors of the polynomial regression for WL_avg using spreading score and AMC.

design	linear		quadratic	
	error	%	error	%
test1	0.35	3%	0.41	4%
test2	0.27	2%	0.22	2%
test3	0.23	3%	0.56	7%
test4	0.43	4%	0.34	3%
test5	0.61	4%	1.1	7%

Table 7: Errors of the polynomial regression for CW_max using spreading score, AMC and #net.

design	linear		quadratic	
	error	%	error	%
test1	3.2	5%	3.7	6%
test2	1.8	2%	1.6	2%
test3	2.2	5%	3.5	8%
test4	3.5	6%	3.4	5%
test5	3.2	4%	2.8	4%

5. CONCLUDING REMARKS AND FUTURE DIRECTIONS

In this paper, we have demonstrated and quantified the opportunities for interconnect optimization in high-level synthesis, and evaluated several metrics in predicting wire length and congestion. This is a preliminary study. We see several directions for fruitful future research.

1. Although some metrics evaluated in this paper, such as spreading score and net number, show reasonably good correlations to results after layout on some designs, none of them can consistently predict routability with a high accuracy at this point. The problem of getting a good high-level routability estimator is still interesting and challenging. Any improvement in this direction can potentially improve the microarchitecture of the RTL design and reduce time and effort in layout. One possibility is to consider a combination of several metrics, together with some modeling of interconnects inside components, for more accurate routability prediction.

2. After obtaining a reasonably accurate metric, another challenge is how to use it to guide the optimization. One possible approach is through iterative refinement by restructuring a section of the HLS solution to gradually improve the routability (under timing constraints). The challenge is to ensure convergence (i.e. not to introduce new hotspots dur-

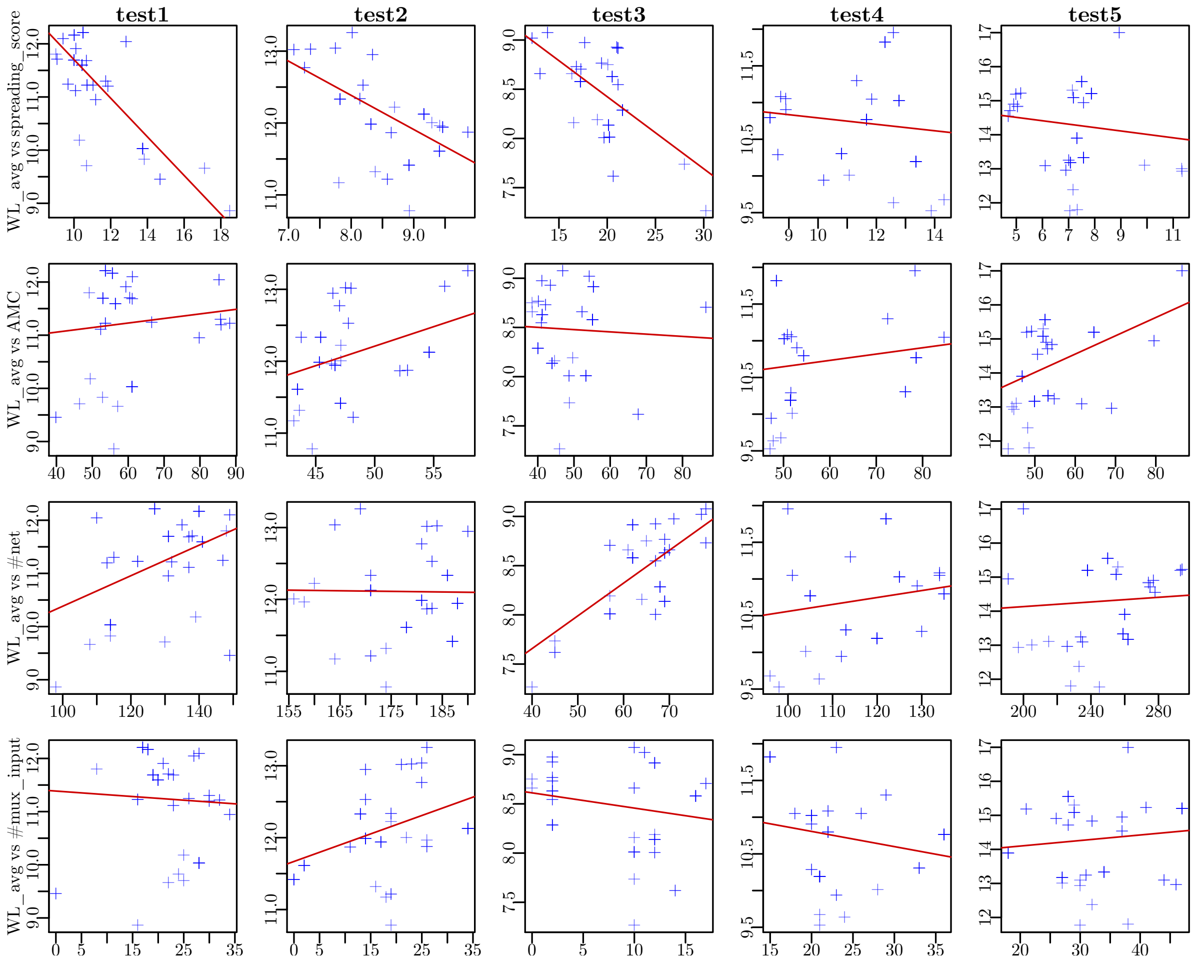

Figure 4: Linear regression of average wire length on four high-level metrics.

ing refinement). We used such an iterative approach to coordinate scheduling and resource binding [16] and achieved encouraging results.

3. As we have demonstrated, proper organization of compiler transformations also has a considerable influence on interconnect optimization. This is even more challenging as the existing metrics will not apply in the absence of RTL netlists. Some recent research efforts from the compiler community use statistical methods, like machine learning, for automated compiler optimization [19, 36]. We are in the process of exploring this direction.

6. ACKNOWLEDGEMENTS

This work was supported in part by the Semiconductor Research Corporation under Contract 2009-TJ-1879 and the UC Discovery program. Guojie Luo would like to thank the Center for Energy-Efficient Computing and Applications at Peking University for providing start-up research fund. The authors thank Janice Martin-Wheeler for her help in editing this paper.

7. REFERENCES

[1] Altera Quartus II. Available: `http://www.altera.com`.

[2] Berkeley Logic Synthesis and Verification Group, ABC: A System for Sequential Synthesis and Verification, Release 70731. Available: `http://www.eecs.berkeley.edu/~alanmi/abc/`.

[3] VPR and T-VPack 5.0.2 . Available: `http://www.eecg.toronto.edu/vpr/`.

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

[5] S. Boyd and L. Vandenberghe. *Convex Optimization.* Cambridge University Press, New York, NY, 2004.

[6] L. Carloni, K. McMillan, and A. Sangiovanni-Vincentelli. Theory of latency-insensitive design. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 20(9):1059–1076, Sept. 2001.

[7] D. Chen and J. Cong. Register binding and port assignment for multiplexer optimization. In *Proc. Asia and South Pacific Design Automation Conf.*, pages 68–73, 2004.

[8] C.-L. E. Cheng. Risa: accurate end efficient placement routability modeling. In *Proc. Int. Conf. on Computer-Aided Design*, pages 690–695, Nov. 1994.

[9] J. Cong. An interconnect-centric design flow for

nanometer technologies. *Proc. IEEE*, 89(4):505–528, 2001.

[10] J. Cong, Y. Fan, G. Han, W. Jiang, and Z. Zhang. Platform-based behavior-level and system-level synthesis. In *Proc. IEEE Int. SOC Conf.*, pages 199–202, 2006.

[11] J. Cong, Y. Fan, G. Han, X. Yang, and Z. Zhang. Architecture and synthesis for on-chip multicycle communication. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 23(4):550–564, April 2004.

[12] J. Cong, Y. Fan, and W. Jiang. Platform-based resource binding using a distributed register-file microarchitecture. In *Proc. Int. Conf. on Computer-Aided Design*, pages 709–715, 2006.

[13] J. Cong, W. Jiang, B. Liu, and Y. Zou. Automatic memory partitioning and scheduling for throughput and power optimization. In *Proc. Int. Conf. on Computer-Aided Design*, pages 697–704, 2009.

[14] J. Cong and B. Liu. A metric for layout-friendly microarchitecture optimization in high-level synthesis. In *Proc. Design Automation Conf.*, 2012. in press.

[15] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang. High-level synthesis for FPGAs: From prototyping to deployment. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 30(4):473–491, Apr. 2011.

[16] J. Cong, B. Liu, and J. Xu. Coordinated resource optimization in behavioral synthesis. In *Proc. Design, Automation and Test in Europe*, pages 1267–1272, 2010.

[17] J. Cong and Z. Zhang. An efficient and versatile scheduling algorithm based on SDC formulation. In *Proc. Design Automation Conf.*, pages 433–438, 2006.

[18] W. Dougherty and D. Thomas. Unifying behavioral synthesis and physical design. In *Proc. Design Automation Conf.*, pages 756–761, 2000.

[19] C. Dubach, T. M. Jones, E. V. Bonilla, G. Fursin, and M. F. P. O'Boyle. Portable compiler optimisation across embedded programs and microarchitectures using machine learning. In *Proc. Int. Symp. on Microarchitecture*, pages 78–88, 2009.

[20] Y.-M. Fang and D. F. Wong. Simultaneous functional-unit binding and floorplanning. In *Proc. Int. Conf. on Computer-Aided Design*, pages 317–321, 1994.

[21] R. Gupta and F. Brewer. *High-Level Synthesis: A Retrospective*, pages 13–28. Springer, 2008.

[22] C.-Y. Huang, Y.-S. Chen, Y.-L. Lin, and Y.-C. Hsu. Data path allocation based on bipartite weighted matching. In *Proc. Design Automation Conf.*, pages 499–504, 1990.

[23] T. Kim and X. Liu. Compatibility path based binding algorithm for interconnect reduction in high level synthesis. In *Proc. Int. Conf. on Computer-Aided Design*, pages 435–441, 2007.

[24] P. Kudva, A. Sullivan, and W. Dougherty. Measurements for structural logic synthesis optimizations. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 22(6):665–674, June 2003.

[25] T. Kutzschebauch and L. Stok. Congestion aware layout driven logic synthesis. In *Proc. Int. Conf. on Computer-Aided Design*, pages 216–223, 2001.

[26] C. Li, M. Xie, C.-K. Koh, J. Cong, and P. Madden. Routability-driven placement and white space allocation. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 26(5):858–871, May 2007.

[27] S. Liu, K.-R. Pan, M. Pedram, and A. Despain. Alleviating routing congestion by combining logic resynthesis and linear placement. In *Proc. European Conf. on Design Automation*, pages 578–582, Feb. 1993.

[28] S. Mayrhofer and U. Lauther. Congestion-driven placement using a new multi-partitioning heuristic. In *Proc. Int. Conf. on Computer-Aided Design*, pages 332–335, Nov. 1990.

[29] M. C. McFarland. Reevaluating the design space for register transfer hardware synthesis. In *Proc. Int. Conf. on Computer-Aided Design*, pages 262–265, 1987.

[30] L. McMurchie and C. Ebeling. PathFinder: a negotiation-based performance-driven router for FPGAs. In *Proc. Int. Symp. on FPGA*, pages 111–117, 1995.

[31] D. Pandini, L. Pileggi, and A. Strojwas. Congestion-aware logic synthesis. In *Proc. Design, Automation and Test in Europe*, pages 664–671, 2002.

[32] B. M. Pangre. Splicer: a heuristic approach to connectivity binding. In *Proc. Design Automation Conf.*, pages 536–541, 1988.

[33] S. Posluszny, N. Aoki, D. Boerstler, P. Coulman, S. Dhong, B. Flachs, P. Hofstee, N. Kojima, O. Kwon, K. Lee, D. Meltzer, K. Nowka, J. Park, J. Peter, J. Silberman, O. Takahashi, and P. Villarrubial. "Timing closure by design," a high frequency microprocessor design methodology. In *Proc. Design Automation Conf.*, pages 712–717, 2000.

[34] J. Roy, S. Adya, D. Papa, and I. Markov. Min-cut floorplacement. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 25(7):1313–1326, July 2006.

[35] P. Saxena, R. S. Shelar, and S. S. Sapatnekar. *Routing Congestion in VLSI Circuits: Estimation and Optimization*. Springer, 2007.

[36] M. Stephenson, S. Amarasinghe, M. Martin, and U.-M. O'Reilly. Meta optimization: improving compiler heuristics with machine learning. In *Proc. ACM SIGPLAN Conf. on Programming Language Design and Implementation*, pages 77–90, 2003.

[37] S. Tarafdar, M. Leeser, and Z. Yin. Integrating floorplanning in data-transfer based high-level synthesis. In *Proc. Int. Conf. on Computer-Aided Design*, pages 412–417, 1998.

[38] L. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM Review*, 38(1):49–95, 1996.

[39] J.-P. Weng and A. C. Parker. 3D scheduling: high-level synthesis with floorplanning. In *Proc. Design Automation Conf.*, pages 668–673, 1991.

[40] M. Xu and F. J. Kurdahi. Layout-driven RTL binding techniques for high-level synthesis using accurate estimators. *ACM Trans. on Design Automation of Electronics Systems*, 2:312–343, Oct. 1997.