Low-Power High-Level Synthesis for FPGA Architectures

Deming Chen, Jason Cong, Yiping Fan
Computer Science Department
University of California, Los Angeles
{demingc, cong, fanyp}@cs.ucla.edu

ABSTRACT

This paper addresses two aspects of low-power design for FPGA circuits. First, we present an RT-level power estimator for FPGAs with consideration of wire length. The power estimator closely reflects both dynamic and static power contributed by various FPGA components in 0.1um technology. The power estimation error is 16.2% on average. Second, we present a low power high level synthesis system, named LOPASS, for FPGA designs. It includes two algorithms for power consumption reduction: (i) a simulated annealing engine that carries out resource selection, function unit binding, scheduling, register binding, and data path generation simultaneously to effectively reduce power; (ii) an enhanced weighted bipartite matching algorithm that is able to reduce the total amount of MUX ports by 22.7%. Experimental results show that LOPASS is able to reduce power consumption by 35.8% compared to the results of Synopsys' Behavioral Compiler.

Categories and Subject Descriptors
B.5.2 [**Register-Transfer-Level Implementation**]: Design Aids – Optimization.

General Terms
Algorithms, Measurement, Performance, Design.

Keywords

RT-level power estimation, Data path optimization, FPGA power reduction.

1. INTRODUCTION

Power optimization has attracted increased attention due to the rapid growth of personal wireless communications, battery-powered devices and portable digital applications. Compared to ASIC chips, FPGA chips are generally perceived as not power efficient because they use a larger amount of transistors to provide programmability. Large power consumption of FPGA chips becomes a constraining factor for FPGA designs to enter main-stream low-power applications. Our goal is to reduce the power consumption without sacrificing much performance or incurring a larger chip area so that we can expand the territories of the FPGA applications effectively.

There have been extensive studies on power optimization in high-level synthesis for ASIC designs [1,2,3,4,5]. However, there is little work on high-level synthesis research specifically targeting the low power FPGA designs. Most of previous high-level synthesis research for FPGAs is not on power reduction. Works in [6,7] presented algorithms for dynamically reconfigurable FPGAs. In [8], a layout-driven high-level synthesis approach was presented to reduce the gap between predicted metrics during RTL synthesis and the actual data after implementation of the FPGA. High-level synthesis for a Multi-FPGA system was done in [9]. The only work we found for low-power high-level synthesis on FPGAs was [10]. A design technique was presented that used pre-computed tables to characterize the RTL and IP components for power estimation. It showed that a low power design could be achieved through this design methodology. However, the model presented was quite simplistic and didn't consider the power consumption of the steering logic, such as the MUX (multiplexer).

As multi-million-gate FPGAs become a reality, increasing design complexity and the need to reduce the design time require early design decisions, especially for the FPGA customers because they care more about time-to-market. As a result, we need to estimate the power consumption at a high level of abstraction, before the low level details of the circuit have been finalized. An accurate RT-level power estimator will provide invaluable directions for effective power reduction.

A recent study [11] indicates that power consumption of interconnects is a dominant source in deep sub-micron (0.1um) FPGAs (more than 60% of the total power). Consequently, power estimation in high-level synthesis must consider total wire capacitance. In this work, we first explore the accuracy of applying *Rent*'s rule for wire length estimation during high-level synthesis for FPGA architectures. Secondly, due to the importance of switching activity for power estimation, we adopt a fast switching activity calculation algorithm [12]. Thirdly, we build a simulated annealing engine that uses estimated power as its cost function during the annealing process and carries out resource selection, function unit binding, scheduling, register binding, and data path generation simultaneously. Finally, we apply a MUX optimization algorithm to further reduce the power consumption of the design. The examples used in this study are data-dominated behavioral descriptions with predominantly arithmetic operations that are commonly encountered in signal and image processing applications. The rest of the paper is organized as follows. In Section 2, we show the architecture and power evaluation flow for the FPGA. Section 3 presents our RT-level power estimator. Section 4 first shows the functional unit library we build, and then it presents our simulated annealing algorithm and MUX optimization algorithm for power reduction. Section 5 presents the experimental data and Section 6 concludes this paper.

2. ARCHITECTURE MODELING AND POWER EVALUATION FRAMEWORK

In this section, we will first briefly introduce the targeted FPGA architecture and then introduce the power evaluation framework.

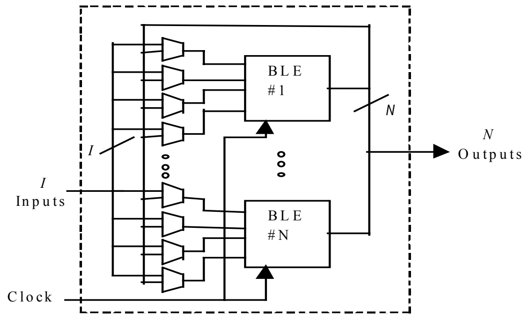
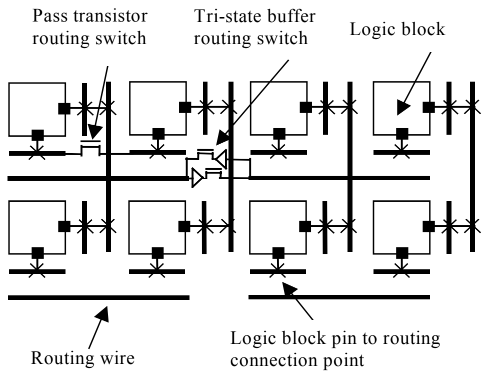


Figure 1: Configurable Logic Block

2.1 Candidate Architectures

FPGA architecture is mainly defined by its logic block architecture and routing architecture. The basic building logic cell is called the basic logic element (BLE) that consists of one K-input lookup table (K-LUT) and one flip-flop. A group of BLEs can form a cluster, or a so-called configurable logic block (CLB), as shown in Figure 1. The number of BLEs (*N* in the figure) is referred as the size of the logic block.



Pass transistor routing switch
Tri-state buffer routing switch
Logic block
Routing wire
Logic block pin to routing connection point

Figure 2: An Island Style FPGA Routing Architecture

We will examine *island-style* FPGA routing architectures. A simplified view of such a routing architecture is shown in Figure 2 [13]. In Figure 2, for example, half the routing tracks consist of length one wire segments (span one logic block), while the other half consist of length two wire segments. Some of the programmable routing switches are pass transistors, while others are tri-state buffers. There are also switches (connection boxes) to connect the wire segments to the inputs and outputs of each logic block.

By varying logic blocks and routing structures, one can easily create many different FPGA architectures. In this work, we will use logic block size *N* as 4 and LUT input size *K* as 4. All the wire segments are length one segments, and all the routing switches are tri-state buffers. This architecture is similar as the one used in [14]. We believe our results hold for similar architectures with different logic or routing parameters.

2.2 Power Evaluation Framework

In order to achieve accurate quantitative analysis of the effects of different FPGA architectural parameters as well as novel power minimization techniques, we need a flexible power evaluation framework. Such a framework was recently developed, named *fpgaEva_LP* [11]. It takes logic block architecture and routing architecture descriptions, as well as the process technology as inputs, goes through synthesis, mapping, placement, routing, delay/capacitance extraction, and analysis/estimation steps to provide quantitative evaluation of area, performance, and power of the proposed architecture on the given benchmark examples. *fpgaEva_LP* is used in this work to evaluate the efficiency of our high-level power optimization tool.

3. RT-LEVEL POWER ESTIMATION

3.1 Wire Length Estimation

Wire length estimation before layout has been one of the most important applications of *Rent*'s rule. Rent's rule was first introduced by E. F. Rent of IBM, who published an internal memoranda for log plots of "number of pins" vs. "number of circuits" in a logic design in 1960. Such plots tend to form straight lines in a log-log scale and follow the relationship

$$T = kN^{P}$$

where *T* is the number of external pins of a logic network; *N* is number of gates contained in the network; *k* is the average number of pins per gate in the network, and *p* is the Rent's parameter. A series of works followed starting with Landman and Russo in 1971 [15]. The classical work [16, 17] gives good estimates for post-layout interconnect wire length. More recent work improves the estimation by considering occupying probability [18] or recursively applying Rent's rule throughout an

Region I: $1 \le l < \sqrt{N}$

$$i(l) = \frac{\alpha k}{2}\Gamma\left(\frac{l^3}{3} - 2\sqrt{N}l^2 + 2Nl\right)l^{2p-4}$$

Region II: $\sqrt{N} \le l < 2\sqrt{N}$

$$i(l) = \frac{\alpha k}{6}\Gamma(2\sqrt{N} - l)^3 l^{2p-4}$$

$$where \quad \alpha = \frac{f.o.}{f.o. + 1}$$

$$such\ that \quad I(a < l < b) = \int_a^b i(l)dl$$

Figure 3: Interconnect Density Function

entire monolithic system [19]. In [19], it offers a complete description of local, semi-global, and global wires for targeted microprocessor architectures. It models the architecture as homogeneous arrays of gates evenly distributed in a square die. This architecture model closely reflects the characteristics of an island-style FPGA architecture, where we can treat each logic block as a gate (Figure 2). Therefore, we apply the interconnect density function derived in [19]. In Figure 3, $I(a<l<b)$ gives the total number of interconnects between length $l = a$ and $l = b$ (l in units of logic block pitches). N is the number of logic blocks in the design, p is the Rent's exponent, α is the fraction of the on-chip terminals that are sink terminals, *f.o.* is the average fanout, and Γ represents a constant calculated through N and p [19]. We use the Rent's exponent extracted from [14] because they explore similar FPGA architecture, and the placement and routing flow is quite similar as well. This is important because p is an empirical constant that closely relates to architecture and design flow.

3.2 Switching Activity Estimation

We implement an efficient switching activity calculator using *CDFG* (control data flow graph) simulation, extending the idea from [12] that performs simulation just once at the beginning and computes switching activities for any legal binding afterwards without repeating simulations.

For a functional unit, $TC_{in}(O, O')$, called the *toggle count* from operation O to operation O', represents the input transitions when the functional unit switches the execution from O to O'.

After binding and scheduling, every node (operation) of the CDFG is bound to a functional unit and scheduled to a certain control step. In other words, a bound functional unit will execute a set of operations in a certain order. For functional unit FU, let $(O_1 \rightarrow O_2 \ldots \rightarrow O_N)$ be the *operation set* in the execution order. Let $(IV_1 \rightarrow IV_2 \ldots \rightarrow IV_K)$ be a set of input vectors for the CDFG.

$TC_{in}(O_i, O_{i+1})$ and $TC_{in}(O_N, O_1)$ are defined as follows:

$$TC_{in}(O_i, O_{i+1}) = \sum_{j=1}^{K} D_H(IN_i^{\,j}, IN_{i+1}^{\,j}) \qquad (1)$$

$$TC_{in}(O_N, O_1) = \sum_{j=1}^{K-1} D_H(IN_N^{\,j}, IN_1^{\,j+1}) \qquad (2)$$

where $1 \leq i < N$, and $D_H(X, Y)$ represents the Hamming Distance between bit vectors X and Y, and IN_i^j is the input vector on the FU when executing O_i with the input vector IV_j.

The *transition probability* of the inputs of FU is defined as

$$TP_{in} = \frac{\sum_{i=1}^{N-1} TC_{in}(O_i, O_{i+1}) + TC_{in}(O_N, O_1)}{Bit\_width \times (N \times K - 1)},$$

where *Bit_width* is the input vector width of FU.

In [12], a matrix of TC_{in} is constructed after scheduling but before binding, and is used for looking up when calculating the TP_{in} after every binding solution. Two operations are compatible if they can be bound to the same functional unit. For two compatible operations O_i and O_j, there will be two entries $[O_i, O_j]$ and $[O_j, O_i]$ in the pre-calculated matrix. Suppose O_i is scheduled before O_j, the value of $[O_i, O_j]$ is from equation (2) and the value of $[O_j, O_i]$ is from (3). After binding, the *operation set* is known for every functional unit. According to the execution order of the *operation set*, every TC_{in} value is looked up in the matrix, and the input *transition probability* can be calculated based on the above equation. The scheduling cannot be changed after the TC_{in} matrix is constructed in [12].

To make the switching activity estimation more flexible, we extend the TC_{in} matrix to support every possible scheduling and binding. That is, for every two compatible operations O_i and O_j, we pre-calculate the TC_{in} values for scheduling order $(O_i \rightarrow O_j)$ and $(O_j \rightarrow O_i)$ using both equation (1) and (2), so there will be two values for each scheduling order of O_i and O_j. As such, regardless how O_i and O_j are scheduled and bound, we can still find the entries in the matrix when calculating the TP_{in}.

For the *transition probability* of the outputs of FU, we use the same method. The total switching activity of the CDFG is the weighted sum of the input and output transition probabilities of each used functional unit.

3.3 RT-level Power Model

We consider both *dynamic* and *static* power for various FPGA components. FPGA contains buffer-shielded LUT cells with fixed capacitance load and routing wires of unpredictable capacitances. We can use pre-characterization-based macro-modeling to capture the average switching power per access of the LUT and register. As for interconnects, switch level calculation can be used. This mixed-level FPGA power model is also used in [11]. A gate-level power estimator is presented in [11], where power-macro-modeling of individual LUT and registers are carried out using SPICE simulation for 0.1um technology, and the interconnect delay and capacitance are extracted after layout to calculate interconnect power consumption.

Our RT-level power model can be summarized in equations (3) and (4). In equation (3), S is the estimated switching activity. The dynamic power is contributed from P_{LUT} (macro-modeling power summing over all the LUTs), P_{REG} (macro-modeling power summing over all the registers), P_{LW} (power of local wires within the CLB estimated through CLB size), and P_{GW} (power of global routing wires estimated by the method explained in Section 3.1). P_{LW} and P_{GW} are calculated through $0.5 f \cdot V_{dd}^{\,2} \cdot C_{Wire}$. In equation (4), the static power of all the idle LUTs and local and global buffers are counted in. The total power is the sum of $P_{Dynamic}$ and P_{Static}.

$$P_{Dynamic} = S(P_{LUT} + P_{REG} + P_{LW} + P_{GW}) \qquad (3)$$

$$P_{Static} = P_{Idle\_LUT} + P_{Static\_LB} + P_{Static\_GB} \qquad (4)$$

4. POWER OPTIMIZATION

In this section, we will first introduce our RT-level library characterization, and then we present a simulated annealing procedure and a MUX optimization algorithm for power reduction.

4.1 Library Characterization

Synopsys offers collections of reusable parameterized Intellectual Property (IP) blocks that are integrated into their synthesis products. The DesignWare-Basic and DesignWare-Foundation libraries contain multipliers, multiplier accumulators, adders and FIR components. These IP blocks are available for Synopsys' FPGA compiler. Since we assume that the FPGA architecture can take advantage of these soft IP blocks during their design process, we will provide different resources implementing the same type of operation in this work. These resources will have different area, delay and power characteristics. It is up to the high-level synthesis procedure to select various resources to serve different objectives. Under this assumption, we select adders, multipliers, comparators and other FU (functional unit) components with different implementations and characterize their area, delay and power respectively. Figure 4 shows the flow for the characterization. Table 1 shows some of the characterization data. Area in terms of number of CLBs required to map the FU, critical path delay after layout, and power value are reported. The average number of pins per CLB and the average fanout number of the FUs are also recorded because they are used in the calculations of the wire distributions (Section 3.1). The power values shown in Table 1 are just for reference and are not used in our power estimator because they only represent atomic power values. Our RT-level power model considers detailed power characterization for both logic elements used by the entire design (including the LUTs mapped by both operational nodes and steering logic such as MUXes) and the estimated interconnect usage.

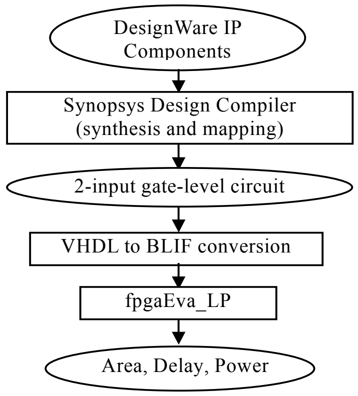


Figure 4: FU Characterization Flow

4.2 Simultaneous Binding and Scheduling for Power Minimization

Before we show our algorithm, we will examine some of the FPGA's unique features that will help us gain some insights for forming an efficient algorithm:

(1) FPGA offers an abundance of distributed registers.

(2) It has no efficient support for wide MUXes (Table 1).

(3) Smaller numbers of functional units and/or registers may not correspond to a smaller area or power.

These properties will influence register binding and steering logic allocation, i.e., MUX generation, during high-level synthesis. Particularly, since FPGA is not efficient in implementing wide input MUXes due to limited routing resources, smaller numbers of functional units allocated but larger number of wide-input MUXes incurred may lead to an unfavorable solution. This requires an algorithm to explore a large solution space considering multiple constraining parameters for FU and register binding, MUX generation, and scheduling.

FU	Implementation	Area (clb)	Delay (ns)	Power (w)
add24bit_bk	Brent-Kung	42	6.09	0.016
add24bit_cla	Carry look-ahead	26	11.78	0.010
ash24bit	Arithmetic shifter	67	5.33	0.023
cmp24bit	Comparator	14	4.17	0.003
mul18bit_nbw	Non-Booth-recoded	288	14.55	0.246
mul18bit_wall	Booth-recoded Wallace	280	14.78	0.308
Mul18bit_wall_s2	Wallace tree 2 stage	286	11.43	0.164
Mul18bit_wall_s4	Wallace tree 4 stage	262	7.14	0.114
mux24bit_2to1	Synopsys synthesis	6	0.57	0.002
mux24bit_4to1	Synopsys synthesis	18	2.34	0.005
mux24bit_8to1	Synopsys synthesis	66	4.60	0.023
mux24bit_16to1	Synopsys synthesis	135	6.91	0.083
mux24bit_32to1	Synopsys synthesis	276	10.93	0.240

Table 1: Function Unit Characterization Data

The simulated annealing algorithm has been proved efficient for high-level synthesis to tackle intractable problems [7,9,20], and is adopted in this work. Our simulated annealing engine starts with an initial FU binding generated by a *force-directed* algorithm. It then performs five types of moves to gradually reduce the overall cost. The cost is the total power consumption calculated by our RT-level power estimator. The moves are randomly picked and the targeted FU binding(s) for each move is randomly picked as well. The moves are as follows:

• *Reselect*: selects another FU of the same functionality but different implementation for a binding.

• *Swap*: swaps two bindings of the same functionality but different implementations.

• *Merge*: merges two bindings into one, i.e., the operations bound to the two FUs are combined into one FU.

• *Split*: splits one binding into two. Reverse of *Merge*.

• *Mix:* selects two bindings, *merge* them, sort the merged operations according to their *slack*, and then *split* the operations.

Each of these moves has its own attributes. For example, *Reselect* may pick a smaller FU (possibly larger delay) for operations that are not on critical path (*slack* > 0) of the CDFG without violating latency constraint, and *Mix* may lead to rebinding the operations that have larger slacks into a pipe-lined function unit such as *Mul18bit_wall_s4*. *Split* will be disabled when the temperature is low so the binding solution will not be dramatically changed.

After each move, a *list scheduling* is called to verify the total latency. Then, the *left edge* algorithm is used for register binding followed by MUX generation. The total amount of CLBs is estimated through the FU and MUX characterization library, and the routing wires are estimated as shown in Section 3.1. Finally, the cost is calculated for the current binding and scheduling solution. The annealing process exits when the percentage of accepted moves are low enough.

4.3 MUX Optimization

Since wide-input MUX is very expansive for FPGAs in terms of area, delay and power, an efficient MUX reduction algorithm is required to reduce steering logic expanses. Pangrle showed that connectivity reduction with a fixed unit binding is an NP-Complete problem [21]. Register binding has a great impact on the MUX cost in the final data path, especially when scheduling and functional unit binding are fixed. A register allocation algorithm based on weighted bipartite matching was proposed in [22] trying to optimize the MUX cost before functional unit binding. We design a new cost function so the register binding can be carried out after the functional unit binding and reduce the total amount of MUX ports directly. Meanwhile, we allow the register number to be relaxed by a small percentage, which will introduce more flexibility to reduce MUX cost.

First, the algorithm calls the left edge algorithm to get the minimum number of registers required. We then relax the register number by a certain ratio. After that, we get a register set *R*.

The variables will be assigned to *R* iteratively. In an iteration, according to the ascending order of the left edges of the variables, we select a mutually incompatible set of unassigned variables V_{IC}, where $|V_{IC}| = |R|$ (We may also relax the size of V_{IC} to include more variables in order to catch a more global picture). We then construct a weighted bipartite graph $G = (V_{IC} \cup R, E)$, where $E = \{(v, r) \mid v \in V_{IC}$ and $r \in R$ such that v is compatible with the variables allocated in $r\}$. Each edge will be attached a weight, which will be discussed later. After solving the minimum weight bipartite matching, we allocate the variables to *R* according to the matching. The process is repeated until all the variables are allocated.

The weight of an edge (v, r) in G is

$$w(v,r) = \alpha_1 \cdot x_1(v,r) + \alpha_2 \cdot x_2(v,r) + \beta \cdot y(v,r)$$

A MUX is introduced before a register *r* when more than one functional units produce results and store them into this register, as shown in Figure 5 (a). We use $MUX_R(r)$ to represent this MUX. A MUX is introduced before a port *p* of a functional unit when more than one registers feeding data to this port, as shown in Figure 5 (b). $MUX_P(p)$ is used to represent this MUX.

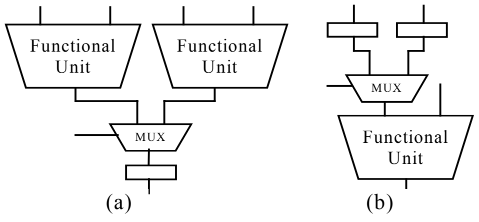


Figure 5: (a) MUX Introduced Before a Register; (b) MUX Introduced Before a Port.

In the weight function, $x_1(v, r)$ is the size of $MUX_R(r)$ if v is assigned to r. This item tries to reduce the maximal MUX width. $x_2(v, r)$ represents the increase of the width of $MUX_R(r)$ if v is assigned to r. That is, $x_2(v, r) = 0$ if the functional unit producing v already drove register r before this register binding iteration. Otherwise, $x_2(v, r) = 1$. $y(v, r)$ is the sum of $MUX_P(p)$ for every port *p* of every functional unit if *v* is assigned to *r*. Terms x_2 and *y* are to control the total width of MUXes.

5. EXPERIMENTAL RESULTS

Our LOw Power Architectural Synthesis System (LOPASS) consists of the simultaneous binding and scheduling followed by MUX optimization. We will show our MUX optimization results separately in Section 5.1 before we show the power reduction results in Section 5.2. Our benchmarks include several different DCT algorithms, such as PR, WANG, and DIR, and two DSP programs MCM and HONDA. These benchmarks are from [23].

	Left-edge		LOPASS		Comparison	
Bench-marks	Reg No.	Mux Port	Reg No.	Mux Port	Reg No.	Mux Port
dir	44	320	48	237	9.1%	-25.9%
honda	34	214	37	154	8.8%	-28.0%
mcm	54	173	59	146	9.3%	-15.6%
pr	31	82	34	68	9.7%	-17.1%
wang	29	101	32	74	10.3%	-26.7%
Ave.					**9.4%**	**-22.7%**

Table 2: MUX Reduction Results of LOPASS

5.1 MUX Reduction Results

Table 2 shows that our MUX optimization algorithm reduces total MUX ports by 22.7% on average with register number increased by 3 to 5 compared to the *left edge*-based register binding algorithm. Since an FPGA contains a rich amount of registers on the chip, we believe this increase is trivial in practice. On the other hand, the amount of MUX ports reduced is significant. We also tried no register number relaxation, the result is 6.3% worse on MUX port reduction than that with relaxation.

	Estimated		Actual		Estimation Error	
Bench-marks	Wire Length	Power (w)	Wire Length	Power (w)	Wire Length	Power
dir	158187	1.79	188891	1.68	-16.3%	6.0%
honda	119140	1.33	100804	1.04	18.2%	27.5%
mcm	76569	0.88	97292	1.08	-21.3%	-18.8%
pr	39617	0.47	43662	0.58	-9.3%	-18.8%
wang	39978	0.46	41167	0.52	-2.9%	-10.1%
Ave.					**13.6%**	**16.2%**

Table 3. Wire Length and Power Estimation

5.2 Power Reduction Results

The experimental flow is similar to that of Figure 4. The RT-level design generated from LOPASS will go through Synopsys' Design Compiler for synthesis and mapping. After VHDL-BLIF conversion, *fpgaEva_LP* reports area, delay and power data. Table 3 shows how our wire length and power estimation work. Wire length is just 13.6% away from reality. This indicates that

		S-BC[1]				LOPASS			
Bench marks	Node No.	Adder	Multi-plier	Cycle	Reg No.	Adder	Multi-plier	Cycle	Reg No.
dir	152	9	16	32	75	5	7	32	55
honda	101	9	14	29	55	3	6	28	40
mcm	98	23	6	36	118	4	3	35	59
pr	46	13	8	24	33	2	2	23	34
wang	52	5	8	29	29	2	2	28	30

Table 4. Binding and Scheduling Comparison

[1] S-BC usually uses multipliers of different sizes for constant handling and timing optimization. Although S-BC uses more multipliers than LOPASS, the sizes of their multipliers can be smaller than those used in LOPASS. LOPASS only uses multipliers of the same size. We set high effort option for S-BC.

	S-BC			LOPASS			Comparison		
Benchmarks	LUT No.	Delay (ns)	Power (w)	LUT No.	Delay (ns)	Power (w)	LUT No.	Delay	Power
dir	18658	54.7	2.55	11281	53.5	1.68	-39.5%	-2.2%	-34.0%
honda	16426	43.4	1.85	7584	40	1.04	-53.8%	-7.8%	-43.8%
mcm	15991	46.8	1.97	7396	50.8	1.08	-53.7%	8.5%	-44.8%
pr	7663	30.2	0.72	3873	34.4	0.58	-49.5%	13.9%	-19.1%
wang	9057	35.7	0.83	3925	35.4	0.52	-56.7%	-0.8%	-37.4%
Ave.							**-50.6%**	**2.3%**	**-35.8%**

Table 5: LUT Number, Delay and Power Comparison

Rent's rule-based estimation method is effective to estimate wire length for FPGA designs before layout information is available. Our RT-level power estimation also works well with a 16.2% average error.

Our simulated annealing engine can either pick the moves that fulfill the latency requirement set by the user or allow a certain percentage of latency relaxation to trade-off latency with power. Table 4 shows the results when we control the latency within the value generated by Synopsys' Behavioral Compiler (S-BC). *Node No.* column shows the number of the operational nodes of the benchmarks. *Cycle* columns show the control steps scheduled, and the *adder* and *multiplier* columns show the binding information for both S-BC and LOPASS.

Table 5 shows the area, delay and power comparison results. Area is the number of the LUTs used in the design. On average, our solution reduces required LUTs by half to realize the design on an FPGA and improves power by 35.8% compared to S-BC. There is a small delay overhead (2.3%).

6. CONCLUSION AND FUTURE WORK

We have presented an RT-level power estimator for FPGAs with consideration of wire length. We showed that our wire length estimation error is 13.6% on average. Our RT-level power estimator controls estimation error as 16.2% on average. We also presented two algorithms to reduce power consumption. We first built a simulated annealing engine that carried out resource selection, function unit binding, scheduling, register binding, and data path generation simultaneously to effectively reduce power. We then designed an enhanced weighted bipartite matching algorithm and reduced the total amount of MUX ports by 22.7% on average. Experimental results showed that we were able to reduce power consumption by 35.8% after placement and routing on average. In the future, we plan to investigate the trade-off behavior between latency and power.

7. ACKNOWLEDGMENTS

This work is partially supported by the NSF Grant CCR-0096383 and Altera Corporation under the California MICRO program.

8. REFERENCES

[1] A. Raghunathan and N.K. Jha, "Behavioral synthesis for low-power," International Conference on Computer Design, Oct 1994.

[2] P. Kollig and B.M. Al-Hashimi, "A new approach to simultaneous scheduling, allocation and binding in high level synthesis," IEE Electronics Letters, vol. 33, Aug 1997.

[3] A.P. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey and R.W. Brodersen, "Optimizing power using transformations," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 1, pp. 12-31, Jan. 1995.

[4] A. Raghunathan and N.K. Jha, "SCALP: An iterative improvement-based low-power data path synthesis system," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16 11, Nov. 1997, pp 1260-1277.

[5] M. Ercegovac, D. Kirovski and M. Potkonjak, "Low-power behavioral synthesis optimization using multiple precision arithmetic," Proc. 37th Design Automation Conference*,* 1999.

[6] M. Vasilko and D. Ait-Boudaoud, "Scheduling for dynamically reconfigurable FPGAs," Proc. of International workshop on logic and architecture synthesis, 1995.

[7] J. C. Alves and J. S. Matos, "A simulated annealing approach for high-level synthesis with reconfigurable functional units," Proc. 38th Midwest Symposium on Circuits and Systems, 1996.

[8] M. Xu and F. J. Kurdahi, "Layout-driven high level synthesis for FPGA based architectures," Proc. IEEE Symposium on FPGAs for Custom Computing Machines, 1998.

[9] A. A. Duncan, D. C. Hendry and P. Gray, "An overview of the COBRA-ABS high level synthesis system for multi-FPGA systems," Proc. IEEE Symposium on FPGAs for Custom Computing Machines, 1998.

[10] F. G. Wolff, M. J. Knieser, D. J. Weyer and C. A. Papachristou, "High-level low power FPGA design methodology," IEEE National Aerospace Conference, 2000.

[11] F. Li, D. Chen, L. He and J. Cong, "Architecture evaluation for power-efficient FPGAs," ACM International Symposium on FPGA, February 2003.

[12] A. Bogliolo, L. Benini, B. Riccó and G. De Micheli, "Efficient switching activity computation during high-level synthesis of control-dominated designs," Proceedings 1999 International Symposium on Low Power Electronics and Design, pages 127-132, August 16-17, 1999.

[13] V. Betz and J. Rose, "FPGA routing architecture: segmentation and buffering to optimize speed and density," ACM International Symposium on FPGA, February 1999.

[14] A. Singh and M. Marek-Sadowska, "Efficient circuit clustering for area and power reduction in FPGAs," ACM FPGA, February 24-26, 2002.

[15] B. Landman and R. Russo, "On a pin versus block relationship for partitions of logic graphs," *IEEE Transactions on Computer*s, c-20:1469–1479, 1971.

[16] W. E. Donath, "Placement and average interconnection lengths of computer logic," *IEEE Transactions on Circuits and System*s, 26(4):272–277, April 1979.

[17] M. Feuer, "Connectivity of random logic," *IEEE Transactions on Computer*s, C-31(1):29–33, Jan 1982.

[18] D. Stroobandt and J. V. Campenhout, "Accurate interconnection length estimations for predictions early in the design cycle," *VLSI Design, Special Issue on Physical Design in Deep Submicro*n, 10(1):1–20, 1999.

[19] J.A. Davis, V.K. De and J. Meindl, "A stochastic wire-length distribution for gigascale integration (GSI) –Part I: Derivation and validation," *IEEE Trans. on Electron Device*s, 45(3):580–589, Mar. 1998.

[20] A. Dasgupta and R. Karri, "Simultaneous scheduling and binding for power minimization during microarchitecture synthesis," Proc. 1995 International Symposium on Low Power Design, April 23-26, 1995.

[21] B.M. Pangrle, "On the complexity of connectivity binding," *IEEE Transactions on Computer-Aided Design*, Vol. 10. No. 11, 1991.

[22] C.Y. Huang, Y.S. Chen, Y.L. Lin and Y.C. Hsu, "Data path allocation based on bipartite weighted matching," 27th ACM/IEEE Design Automation Conference, pp.499-504, June 24-27, 1990.

[23] M. B. Srivastava and M. Potkonjak, "Optimum and heuristic transformation techniques for simultaneous optimization of latency and throughput," *IEEE Trans. on VLSI Systems*, vol.3 (1), pp.2-19, Mar. 1995.