Bus Optimization for Low-Power Data Path Synthesis Based on Network Flow Method

Sungpack Hong and Taewhan Kim
Department of Electrical Engineering & Computer Science
and Advanced Information Technology Research Center (AITrc)
KAIST, Taejon, 305-701 KOREA

Abstract— Sub-micron feature sizes have resulted in a considerable portion of power to be dissipated on the buses, causing an increased attention on savings for power at the behavioral level and RT level of design. This paper addresses the problem of minimizing power dissipated in switching of the buses in data path synthesis. Unlike the previous approaches in which minimization of the power consumed in buses has not been considered until operation scheduling is completed, our approach *integrates the bus binding problem into scheduling* to exploit the impact of scheduling on reduction of power dissipated on the buses more fully and effectively. We accomplish this by *formulating the problem into a flow problem in a network,* and *devising an efficient algorithm which iteratively finds maximum flow of minimum cost solutions* in the network. Experimental results on a number of benchmark problems show that given resource and global timing constraints our designs are 22% power-efficient over the designs produced by a random-move based solution, and 18% power-efficient over the designs by a clock-step based optimal solution.

1 Introduction

The advent of portable digital devices such as laptop personal computers has made low power CMOS circuit design an increasingly important research area. For example, laptop computers have limited battery life, and so the circuitry in the computer must be designed to dissipate as little power as possible without sacrificing performance in terms of speed. It has been shown [1] that a dominant portion of power dissipation in digital CMOS circuits is due to the dynamic power, expressed below:

$$\sum_{i=1}^{N} \frac{1}{2} C_i V_{dd}^2 f_i \qquad (1)$$

where C_i is the output capacitance of the i-th gate, V_{dd} is the supply voltage, f_i is the frequency of transitions at node i, and N is the total number of gates on the chip. Note that as pointed out in [2] C_i values for long buses are often significantly higher than the C_i values for gates. Constraints on limited silicon area force signals from module components to be multiplexed onto a single bus line. Multiplexing different signals onto the highly capacitive buses leads to increase switching activity, causing the increase of power dissipation (about 40% of the on-chip power [3, 4]). Consequently, in bus-based data path synthesis it is very important to reduce power consumption by minimizing the factor $\sum C_i f_i$. This motivated us to investigate methods to reduce power dissipation on the highly capacitive buses at an early stage of design process.

Most of high-level synthesis systems perform scheduling of the control and data flow graph (CDFG) before allocation of the functional units and registers since this approach provides timing information for allocation and binding tasks. Further, for a bus-based RTL design bus allocation and binding are generally performed after scheduling. Chang and Pedram [5] proposed a technique for reducing power consumption during the register allocation and binding. They [6] also proposed a technique of reducing power consumption during the binding of functional units. The problem is formulated as max-cost multi-commodity flow problem and solve it optimally. Since the multi-commodity flow problem is NP-hard, they restricted the domain of the functional unit binding problem to functionally pipelined designs with a short latency. Raghunathan and Jha [7] have used an iterative improvement technique for scheduling and module allocation based on switched capacitance matrices. However, all of the above approaches assumed a point-point RTL architecture, and have not taken into account the power dissipation on interconnections.

There are many researches which have addressed the problem of minimizing the switching activity on buses. Panda and Dutt [8] have tried to reduce power in memory intensive applications by minimizing transitions on the (off-chip) memory address buses. They reduced the activity on the memory address buses by analyzing the access patterns of behavioral arrays in the specification and organizing the arrays in memory. Various bus encoding schemes (e.g., [2, 9] have been proposed to decrease the number of transitions at input/output (I/O) (off-chip) bus transitions. Dasgupta and Karri [10, 11] have proposed algorithms for scheduling and binding in order to minimize (on-chip) data bus transitions. The algorithm was based on a simulated annealing process.

The bus optimization approach presented in this paper is intended to overcome some of the limitations of the previous approaches. The key features are: (1) In previous approaches, bus binding is performed at a later stage of data path synthesis, mainly after scheduling. This would result in a less flexibility in optimizing bus switching activity. Since the amount of bus activity is an important cost measure of power dissipation, *our algorithm performs scheduling and bus binding simultaneously so that the effects of scheduling on bus activity are exploited more fully and effectively;* (2) Contrary to the previous integrated scheduling and biniding approaches in which estimation of the amount of switching activity on buses is calculated based on simple intuitions or heuristics, *our algorithm calculates the bus activity near-optimally*[1] *in polynomial time by solving a network flow problem.* In addition, to speed up the calculation of bus activity for a local change of scheduling, we *devise a mechanism of evaluating the network flow in a partial and very limited way* while producing an almost optimal computation.

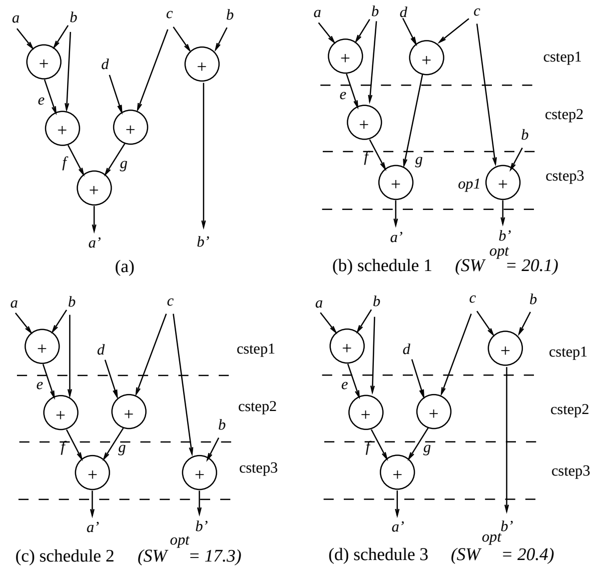
2 Preliminaries

The total power dissipated on a bus is proportional to the switching activity on the bus [12]. Further, the switching activity is an indicator of signal transitions on the bit lines of the bus. Consequently, minimizing the number of signal transitions on a bus is equivalent to reducing the total power dissipated on the bus. The signal switching activity on each bit line of a bus is changing according to not only the data transfers implemented on the bus but also the sequence of the data transfers. Note that scheduling determines the set of data transfers which are to be executed at each clock cycle. However, it does not tell which bus a data transfer will use at that clock time. Bus binding assigns the scheduled data transfers to the buses, generating a complete sequence of data transfers to be implemented on each bus.

We use a probabilistic model to measure the switching activity on a bus. Let $SW^k(x,y)$ be the expected number of bit lines of bus k that toggle when data transfers x and y are successively implemented on the bus. Then, the problem we want to solve is to schedule operations (thus, schedule data transfers) and bind the data transfers to buses in a way to minimize the quantity of

$$SW = \sum_{\forall k \ of \ buses} \quad \sum_{\forall (x \to y) \ transitions \ on \ k} SW^k(x,y) \qquad (2)$$

The signal values a data transfer carries during repeated execution of a CDFG could be different. Let $SW(x,y)$ be the average number of bit transitions (i.e., Hamming distance) when data transfers x and y are successively implemented on a bus. The value of $SW(x,y)$ for every pair of data transfers in the (unscheduled) CDFG can be obtained by repeated simulation of the CDFG; For each set of typical values of primary input signals in the CDFG, the signal values of all data transfers in the CDFG are calculated by simulating the CDFG. From the signal values of the data transfers, the hamming distance for every ordered pair of data transfers is calculated. This process repeats for a sufficiently large number of times. Then, the value of $SW(x,y)$ becomes the average of the hamming distances between x and y. Figure 1(a) shows an unscheduled CDFG where variable a' and b' are cyclic variables, and becomes variables a and b in the next iteration instance of the loop, respectively. We assume the bit-width of each variable is 8. Table 1 shows the $SW(x,y)$ values for the data transfers in Figure 1(a). For example, $SW(a,b)$ = 3.9 indicates that on average 3.9 bit lines out of 8 toggle when a bus carries signals a and b successively.



a
b
c
d
e
f
g
a'
b'
op1
cstep1
cstep2
cstep3
(a)
(b) schedule 1 (SW opt = 20.1)
(c) schedule 2 (SW opt = 17.3)
(d) schedule 3 (SW opt = 20.4)

Figure 1: (a) A CDFG with 8-bit data size. (b)-(d) Possible schedules for (a).

	a	b	c	d	e	f	g	a'	b'
a	0.0	3.9	4.0	3.9	3.9	3.9	2.8	3.1	2.5
b	3.9	0.0	5.7	4.1	3.7	3.9	2.9	3.0	3.1
c	4.0	5.7	0.0	3.1	2.9	2.2	3.1	3.8	4.1
d	3.9	4.1	3.1	0.0	3.9	3.3	2.8	3.1	2.6
e	3.9	3.7	2.9	3.9	0.0	3.8	3.1	2.8	4.0
f	3.9	3.9	2.2	3.3	3.8	0.0	3.1	3.0	3.9
g	2.8	5.7	3.1	2.8	3.1	3.1	0.0	3.2	4.0
a'	3.1	3.0	3.8	3.1	2.8	3.0	3.2	0.0	2.6
b'	2.5	3.1	4.1	2.6	4.0	3.9	4.0	2.6	0.0

Table 1: $SW(x,y)$ values for the CDFG in Figure 1(a).

Figures 1(b)-(d) show three possible schedules of the CDFG in Figure 1(a) when the global timing is 3 clock steps and two adders are available. We assume that every operation takes one clock time to execute.

Figure 2(a) shows the set of data transfers which are to

[1]Note that the optimality is in terms of *average* bus activities. Thus, computing optimally does not necessarily mean computing accurately.

be executed at each clock step. For example, *schedule 1* performs data transfers a, b, c and d at clock step 1, b and e at clock step 2, and b, c, f and g at clock step 3. The dotted backward arrow represents execution of the next iteration instance of *schedule 1.* From the schedule, it is found that at least four buses are needed to implement the data transfers. Figure 2(b) shows an example of assigning the data transfers scheduled by *schedule 1* to four buses where bus 1 carries a at clock step 1, a at clock step 2, g at clock step 3, and then a at clock step 1 of the next iteration instance (i.e., a' at the current instance) and so on. Consequently, SW value on bus 1, SW^1 = $SW^1(a,a)$ + $SW^1(a,g)$ + $SW^1(g,a')$ = 0 + 2.8 + 3.2 = 6.0. In this way we can compute SW^2 = $SW^2(b,b)$ + + $SW^2(b,b)$+ $SW^2(b,b')$ = 0 + 0 + 3.1 = 3.1, SW^3 = 0, and SW^4 = 11.0. Thus, SW = 6.0 + 3.1 + 0 + 11.0 = 20.1

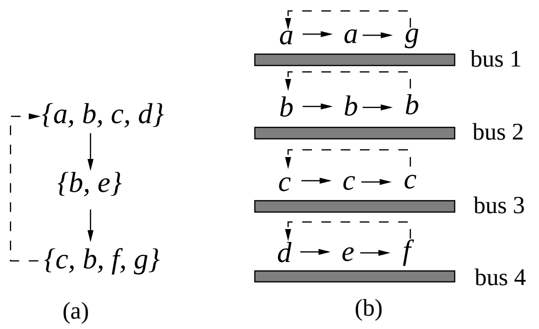


Figure 2: (a) The schedule of data transfers by *schedule 1.* (b) An example of bus binding for (a).

For a given schedule we calculate the value of its optimum SW (called SW^{opt}) by using an exhaustive search for bus binding. The values of SW^{opt} are shown in Figures 1(b)-(d). The big difference between the SW^{opt} values of the schedules strongly suggests that bus optimization for low power must take into account during scheduling. In fact, our algorithm explores every possible schedule which essentially leads to find an optimal or close-to-optimal bus binding. This is accomplished by formulating the problem as a network flow problem for each instance of schedule, and evaluating it accurately, but efficiently.

3 Bus Optimization for Low-Power

3.1 An Overview

Our algorithm of bus optimization for low-power consists of two phases: (1) *an initial phase* which produces an initial solution of scheduling and bus binding, and (2) *a refinement phase* which iteratively improves the initial solution by rescheduling and rebinding. Given global timing and resource constraints, the initial phase of our algorithm produces a schedule for the CDFG by using a conventional scheduling algorithm. Bus binding for the data transfers in the schedule is performed by employing a network flow method which minimizes the cost of SW in Eq. (2). For every possible *local* move of operations for reschedule the network corresponding to the initial schedule is partially updated to reflect the reschedule. Then, we apply *the minimum cost augmentation method* [13] to a *local* section of the network, and determine the amount of the change between SW values before and after the rescheduling.

```
Bus_Opt: Algorithm for Bus Optimization:
• Set $SW(x,y)$s by simulating CDFG;
• Schedule CDFG using a conventional scheduling algorithm;
• Derive a network from the scheduled CDFG; (Sec. 3.2)
• Get an initial bus binding; (Sec. 3.3)
• Set $min\_cost$ to $SW$ of the initial binding;
• Set $best\_sch\_bind$ to the initial schedule and bind;
repeat
    • Set $curr\_cost$ to $min\_cost$;
    • Set $curr\_sch\_bind$ to $best\_sch\_bind$;
    while (there is a "movable" operations)
        • Compute $\Delta SW$ optimally for each move; (Sec. 3.4)
        • Reschedule by the move with the smallest $\Delta SW$;
        if (current $SW > min\_cost$)
            • Set $min\_cost$ to the current $SW$ cost;
            • Set $best\_sch\_bind$ to the current reschedule and bind;
        endif
        • Lock the operation at the move;
    endwhile
until ($best\_schedule\_bind$ is unchanged)
• Return $best\_schedule\_bind$;
```

Figure 3: The proposed algorithm for simultaneous scheduling and bus binding.

The flow of our algorithm is described in Figure 3. An operation which is scheduled at clock step i is called "movable" (i.e., reschedulable) to another clock step j if scheduling the operation at j does not violate the timing and resource constraints. For every movable operation, the amount of increase/decrease of SW is computed (Sec. 3.4) in the *while-loop.* Among the operations, the operation with the largest decrease of SW is selected, and rescheduled to the corresponding clock step. Once the operation is rescheduled, it is locked at that clock step during the rest of the execution of *while-loop.*

3.2 The Network Flow Formulation

Let $OP(i)$ and $DT(i)$ denote the sets of operations and data transfers which are to be performed at clock step i, respectively. A network $G = (N, A)$ is a directed graph with a set of nodes N and a set of arcs (directed edges) A. Note that our formulation is structurally similar to that used in [6]. However, the main feature of our work is how to utilize the formulation efficiently in the context of rescheduling and rebinding. Let us first consider to model *intra tran-*

sitions of data transfers[2] in G. For n total data transfers in the CDFG, N has $2n$ nodes, two for each data transfer, and additional two nodes s and r where s is called the source and r called the sink of the network. The nodes other than the source and sink are arranged vertically according to the clock steps at which the corresponding data transfers are to be executed. The network inside of the box on the left-side in Figure 4 shows the G (except s and r) for the intra-transitions of the scheduled CDFG in Figure 1(b).

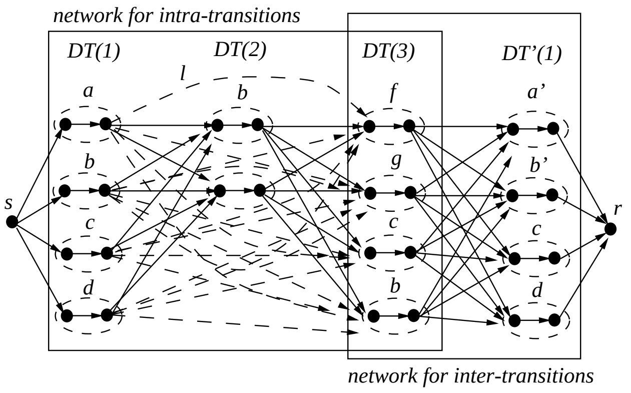


Figure 4: Network model for data transfers in Figure 1(b)

The two nodes corresponding to a data transfer are grouped in a dotted circle as shown in Figure 4, where the arc between them with capacity 1 ensures that at most one data transfer is bound to a bus at a clock step. Node s is connected to every nodes at the first column of data transfers in G and node r is to every node at the last column. The arcs between nodes in different columns in G are classified into two types:

1. short arcs: are the ones with solid lines in Figure 4. These are the arcs from a node in a column of G to every node in the next column. The arc from node x in column i to node y in column $i+1$ represents the change of signal values in a bus when data transfers x and y are performed successively.

2. long arcs: are the ones with dotted lines in Figure 4. The total number of buses to be used is bounded by the maximum number of data transfers that should be performed concurrently. This implies that at the clock steps when the maximum (concurrent) data transfers are to be executed all buses shall be used while at the other clock steps some of the buses shall be idle. The long arcs in the network ensure such a bus utilization. For example, in Figure 4, selecting arc l in a solution tells that a bus implements data transfer a at clock step 1, idles at clock step 2 and f at clock step 3.

The capacity is 1 for every arc in A. The cost of each arc incident on s and r is 0. Let SW^{max} and SW^{min} be the maximum and minimum among the values of all $SW(i,j)$s, respectively. Cost $C(x,y)$ to be assigned to the arc from a node of data transfer x at column $t1$ to a node of data transfer y at column $t2$ is defined as

$$C(x,y) = SW(x,y) - (2 \cdot SW^{max} - SW^{min}) \qquad (3)$$

The term $2 \cdot SW^{max} - SW^{min}$ ensures a maximum flow of minimum cost solution in G to cover every node. In other words, for two different flows, $x \to y \to z$ and $x \to z$, our network flow formulation will select $x \to y \to z$ since its flow cost is always less than the cost of the other. This constraint is proven simply by showing the inequality relation $C(x,y) + C(y,z) \leq C(x,z)$.[3] Further, from the fact that every feasible maximum flow of minimum cost solution which satisfies the triangular inequality relation has the same total number of transition flows (i.e., arcs), a maximum flow of minimum total cost of Eq. (3) solution also becomes a maximum flow of minimum SW of Eq. (2).

To take into account *inter-transitions* of data transfers, the network is then extended to include an additional column of nodes as shown in the box on the right-side of Figure 4. The data transfers of the nodes are the same as those in the data transfers of the first column except replacing the nodes of cyclic data transfers with new nodes.

3.3 The Initial Bus Binding

Given the network flow model constructed from an initial schedule, we apply the *minimum cost augmentation method* [13] to generate an initial binding of the data transfers. Each flow path of the solution corresponds to the sequence of data transfers to be implemented in a bus. Figure 5 shows the set of flow paths with a minimum SW. Since the CDFG is executed repeatedly, there might be some invalid flow paths. In Figure 5 flow paths $a \to e \to g \to d$ and $d \to f \to a'$ are invalid since the last data transfers of the paths (which is also the first data transfers in the next iteration of the CDFG execution) must be identical with the first data transfers of the paths.

We resolve the conflict of flow paths by locally changing the path. Figure 5(b) shows two conflicting flow paths of Figure 5(a). Let *permute_cost(i)* is defined to be the increase of SW cost when the two flows between columns i and i+1 of the network in the flow paths are switched. For example, *permute_cost(1)* = 3.9 means that the new two

[2]An *intra-transition* refers to a consecutive execution of two data transfers in the same iteration instance of the CDFG. Otherwise, the execution is called an *inter-transition*

[3]Suppose data transfers x, y and z are the nodes at columns $t1$, $t2$ and $t3$ ($t1 < t1 < t3$) in G, respectively. Then, $C(x,y)$ + $C(y,z)$ - $C(x,z)$ = $SW(x,y)$ - $(2 \cdot SW^{max} - SW^{min})$ + $SW(y,z)$ - $SW(x,z)$ = -$(SW^{max} - SW(x,y))$ - $(SW^{max} - SW(y,z))$ - $(SW(x,z) - SW^{min}) \leq 0$.

flow paths $a \to d \to f \to a'$ and $d \to e \to g \to d$ have 0.2 more switching cost than original paths $a \to e \to g \to d$ and $d \to d \to f \to a'$ have. For every pair of invalid flow paths in which the first data transfer of a path and the last data transfer of the other path are identical, we perform such flow switches, and calculate the values of *permute_cost(i)*. Then, we select the pair with the least increase of *permute_cost(i)*, and switch the corresponding flows. In Figure 5(b), the second flows of the paths are switched, resulting in new valid flow paths $a \to e \to f \to a'$ and $d \to d \to g \to d$.

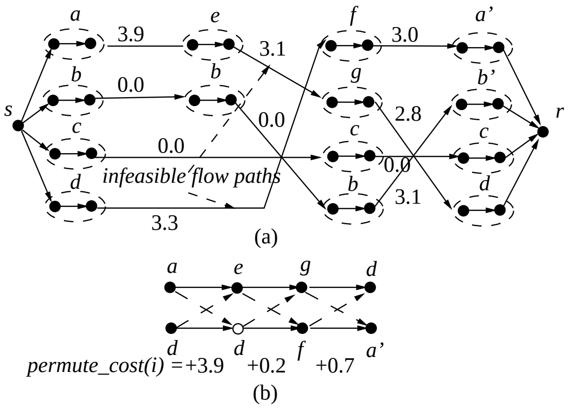


Figure 5: (a) Network flow solution for the example in Figure 4. (b) Resolving the flow conflict.

The flow of the initial binding is summarized below:

```
Init_Bind: Algorithm for Initial Bus Binding:
• Construct network $G$ for a schedule;
• Apply the network flow to $G$;
• Identify the invalid flow paths;
while (there is an invalid flow path)
    • For pairs of invalid flow paths,
      computes permute_costs;
    • Adjust the flow path with the least permute_costs;
endwhile
```

3.4 The Rescheduling and Rebinding

The key of our algorithm is to estimate the amount of increase/decrease of total SW accurately and efficiently when an operation is rescheduled from clock step i to clock step j. Since the operations to be executed at clock steps i and j are updated, the data transfers to be executed at i and j are also updated. This forces to restructure network G, possibly invalidating some flows of the current flow path solution. To maintain a feasible solution, it may be required to apply the initial binding algorithm in Sec. 3.3. again to the updated network. However, this is definitely very expensive for the case of many trials of rescheduling in the *while-loop* of Figure 3. However, if we restrict the distance between i and j to be relatively a small number of clock steps, we can apply the network flow in a local way while producing almost the same result as that of the application of the initial binding algorithm. Specifically, the scope of network to be considered for updating the flows is the nodes in between columns i and j and the arcs adjacent to the nodes.

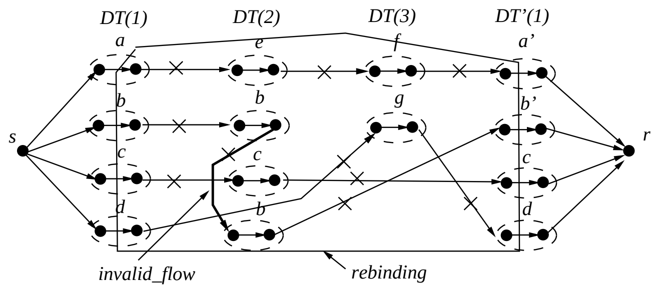


Figure 6: An example of local rebinding

To clarify our idea, let us consider the example in Figure 1(a) (its initial binding is shown in Figure 5) in which $op1$ is rescheduled from cstep 3 to cstep 2. This leads to the network shown in Figure 6. Consequently, the flow with heavy line at column 2 is invalid. Since the nodes in columns 2 and 3 are updated, we invalidate (i.e., marked with $\times$ in the figure) the flows adjacent the nodes, and reevaluate the flows by applying network flow locally to determine a new binding for the reschedule. As indicated in our experimentations (Table 4), this local adjustment of flows due to reschedules significantly improves the run time of our algorithm while maintaining almost the same results as those produced by the application of the full network flow. The following summarizes the flow of our rebinding algorithm:

```
Bus_Rebind: Algorithm for Bus Rebinding:
• Update network $G$ for the reschedule;
• Identify the columns whose nodes are updated;
• Invalidate the flows inside of the columns;
• Invalidate the flows adjacent to the columns;
• Extract a network covering the invalidated flows;
• Apply the network flow to the extracted network;
```

4 Experimental Results

Our algorithm *Bus_Opt* was implemented in C++ and tested on a set of high-level synthesis benchmark examples. Table 2 shows the comparison of the bus switching activities, measured in terms of the quantity of SW in Eq. (2), for the designs produced by the *random-move* based method proposed by [11], the designs produced by the *greedy* one which solves the bus binding problem at each clock step optimally, and the designs produced by our algorithm. DIFF is the differential equation solver and DIFF.2 is the design produced by unrolling DIFF twice. EWF is the 5-th order elliptic filter design, KALMAN is the state vector computation part of the kalman filter design, and COMPLX is the arithmetic part of complex number calculation. The comparisons show that our algorithm was able to use overall 22% and 18% less bus switching activity than those by the *random-move* and *greedy* methods, respectively.

design (#bus)	*random-move* [11]	*greedy*	*Bus_Opt* (% red. [11]/greedy)
DIFF(4)	22.48	26.58	16.40 (27.1/28.1)
DIFF.2(4)	37.44	43.04	33.28 (11.1./21.6)
EWF(6)	17.44	11.44	8.48 (51.3/25.8)
KALMAN(4)	21.12	22.08	18.08 (14.4/16.9)
IDCT(6)	74.00	70.24	65.12 (12.0/6.3)
COMPLX(4)	9.92	8.88	8.32 (16.1/6.3)
Average			(22.0/17.7)

Table 2: Results of bus switching activity using high-level synthesis benchmarks

Table 3 summarizes bus binding results (together with run times) produced by *Init_Bind* and *Bus_Rebind* of our algorithm. Note that the quality of results produced by *Init_Bind* which uses network flow method over the entire network of the CDFG is comparable to that by the *greedy* one which performs bus binding optimally at each cycle step. Further, the results generated by *Bus_Rebind* which refines bus binding by rescheduling shows that bus switching is consistently decreased from the initial binding. This strongly suggests that our *Bus_Rebind* algorithm can be used effectively to improve the bus binding in data paths that are synthesized manually or by any of conventional scheduling and allocation systems.

Table 4 shows the comparisons of results produced by our rebinding algorithm which performs in a partial way in the network and by the full execution of network flow for rebinding to show how much our incremental refinement of bus binding is efficient. The comparisons indicate that our refinement solutions are almost the same as that of the full executions of network flow. Nevertheless, the execution time is significantly less.

5 Conclusions

In this paper, we presented an effective integrated bus optimization approach for low-power which solves scheduling and bus binding problem simultaneously. The algorithm is based on an efficient application of network flow method. In terms of both of bus switching activity and run time, our algorithm produced excellent results: The former one is due to *a tight incorporation of scheduling in the bus optimization*, and the latter is due to *an accurate incremental evaluations of the effect of scheduling changes* in the network flow formulation. Experimental results on a number of benchmark problems show that our algorithm was able to produce designs with 22% and 18% power-efficient than the designs by a random-move based solution and by a greedy (cycle-step by cycle-step optimal) solution.

design (#bus)	*Init_Bind* (run_time)	*Bus_Rebind* (run_time)	% reduction
DIFF(4)	26.56 (17 s)	16.40 (71 s)	38.3
DIFF.2(4)	43.04 (3 m)	33.28 (3 m)	23.0
EWF(6)	11.44 (20 m)	8.48 (30 m)	21.7
KALMAN(4)	22.08 (4 s)	18.08 (15 s)	4.34
IDCT(6)	70.24 (10 m)	65.12 (24 m)	3.53
COMPLX(4)	9.76 (1 s)	8.32 (32 s)	13.9
Average			16.9

Table 3: Comparisons between the initial binding and rebinding of our algorithm.

design (#bus)	*Bus_Rebind* (run_time)	*Full_Extended_Rebind* (run_time)
DIFF(4)	16.40 (71 s)	11.92 (6 m)
DIFF.2(4)	33.28 (3 m)	34.44 (7 m)
EWF(6)	8.48 (30 m)	8.96 (1.5 hr)
KALMAN(4)	18.08 (15 s)	19.28 (27 s)
IDCT(6)	65.12 (25 m)	65.10 (3 hr)
COMPLX(4)	8.32 (32 s)	8.32 (40 s)
Average	24.92	24.67

Table 4: Comparisons between our (local) rebinding and the full rebinding.

Acknowledgment: This work was supported by the Korea Science and Engineering Foundation (KOSEF) through the Advanced Information Technology Research Center (AITrc).

References

[1] A. P. Chandrakasan and R. W. Broderson, *Low Power Digital CMOS Design*, Kluwer Academic Publishers, 1995.

[2] M. R. Stan and W. P. Burleson, “Bus-Invert Coding for Low Power I/O,” *IEEE Trans. VLSI*, Vol.3, No.1, 1995.

[3] H. B. Bakoglu, “Circuits, Interconnection and Packaging for VLSI,” Addisson Wesley, 1990.

[4] C. Svenson and D. Liu, “A Power Estimation Tool and Prospects of Power Savings in CMOS VLSI Chips,” *ISLPED*, 1994.

[5] J.-M. Chang and M. Pedram, “Register Allocation and Binding for Low Power,” *DAC*, 1995.

[6] J.-M. Chang and M. Pedram, “Module Assignment for Low Power,” *EDAC*, 1996.

[7] A. Raghunathan and N. K. Jha, “SCALP: An Iterative Improvement Based Low-Power Data Path Synthesis System,” *IEEE Trans. CAD*, Vol.16, No.11, 1997.

[8] P. R. Panda and N. D. Dutt, “Low-Power Memory Mapping Through Reducing Address Bus Activity,” *IEEE Trans. VLSI*, Vol.7, No.3, 1999.

[9] S. Ramprasad, N. R. Shanbhag, and I. Hajj, “A Coding Framework for Low-Power Address and Data Busses,” *IEEE Trans. VLSI*, Vol.7, No.2, 1999.

[10] A. Dasgupta and R. Karri, “Simultaneous Scheduling and Binding for Power Minimization During Microarchitecture Synthesis,” *ISLPED*, 1995.

[11] A. Dasgupta and R. Karri, “High-Reliability, Low-Energy Microarchitecture Synthesis,” *IEEE Trans. CAD*, Vol.17, No.12, 1998.

[12] F. N. Najm, “Transition Density, A Stochastic Measure of Activity in Digital Circuits,” *DAC*, 1991.

[13] R. E. Tarjan, *Data Structures and Network Algorithms,* Society for Industrial and Applied Mathematics, 1983.