Leakage Power Driven Behavioral Synthesis Of Pipelined Datapaths

Ranganath Gopalan, Chandramouli Gopalakrishnan* and Srinivas Katkoori
Department of Computer Science and Engineering
University of South Florida, Tampa, FL-33620
Email: {rgopalan, cgopalak, katkoori}@csee.usf.edu

Abstract

We present a scheduling, allocation and binding methodology that employs MTCMOS as the standby leakage reduction mechanism. We use the simulated annealing metaheuristic for optimizing leakage power. The cost functions for our approach are obtained after extensive characterization trials taking into account, the run-time characteristics of the MTCMOS approach. Our approach makes use of two cost factors: leakage cost, for optimizing the number of MTCMOS instances, and settling cost, for the minimization of their active-to-standby transitions. We enhance throughput and performance of the datapaths by synthesizing them as functionally pipelined systems before performing our optimizations. Using fully pre-characterized leakage libraries for RT-level simulation, we obtain an average leakage power reduction of 36.2%, and an average area overhead of 6.2%. However with a small increase in schedule latency we obtain an average reduction of around 3.95%-4.6% in the total area.

1 Introduction & Related Work

Leakage power is increasingly becoming the dominant component of overall power dissipation when compared to dynamic power. For wireless and portable domains, it is especially necessary to have systems that have significantly low leakage power dissipation. Rapid depletion of battery power and damages to circuitry over a long term, can be caused if these problems are overlooked. As today's systems become faster and smaller in size, the leakage power dissipation tends to increase exponentially [13]. However, many logical and physical-level techniques have been developed for reducing the sub-threshold leakage problem [7, 8, 14, 15, 16].

In our work, we aim to optimize leakage power during the process of behavioral synthesis. We make use of the MTCMOS design approach as the physical-level leakage reduction technique. The MTCMOS approach which is illustrated in Figure 1, has a significant drawback of increased area overhead and delay penalty. However, in our approach we perform scheduling, allocation and binding of the data-flow graph (DFG) operations in such a way as to incur minimal area penalty due to MTCMOS. Further, we enhance performance of the datapaths by synthesizing them as functionally pipelined datapaths. Work on leakage power optimization during behavioral synthesis has been done in [2, 6]. These approaches have however targetted traditional flow-based behavioral synthesis. Our work is one of the first approaches which considers the synthesis of functionally pipelined leakage-optimized datapaths. The contributions of our work are as follows:

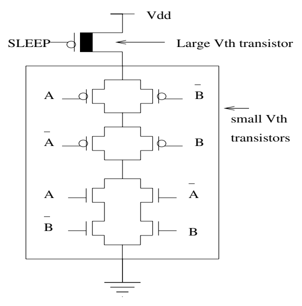

Figure 1. MTCMOS implementation

1. A framework for the behavioral synthesis of pipelined datapaths with low leakage power.
2. A scheduling, allocation, and binding algorithm employing cost-functions which are extensively characterized to model the run-time characteristics of the MTCMOS approach, and thus aims towards optimizing the potential area overhead of MTCMOS.

The rest of this paper is organized as follows: section 2 gives an overview of pipeline synthesis, section 3 gives the details of our proposed approach, experimental results are given in section 4, and in section 5 we conclude the paper.

*C. Gopalakrishnan contributed to this work while he was at University of South Florida. He is currently with Cadence Design Systems (India)

2 Pipeline synthesis

Pipeline synthesis has been explored in [3, 5, 9, 10]. The issue of dynamic power optimization during pipeline synthesis has been dealt with in [4]. A functionally pipelined datapath is segmented into N linear stages, where each stage contains the mapped hardware resources to execute the relevant operations of that stage. The number of stages is dependent on the latency λ of the design and the data-introduction interval δ_0 (also known as pipeline latency) as given in eqn. 1.

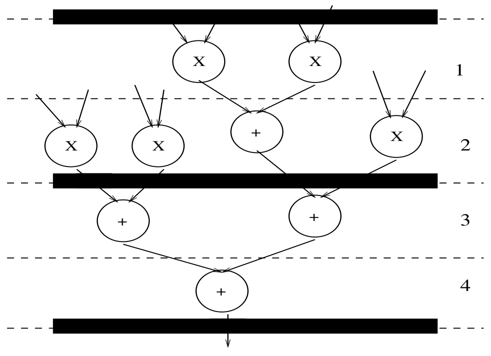

Figure 2. Pipeline with data initiation rate = 2

$$N = \lceil \frac{schedule\ length}{data\ introduction\ interval} \rceil = \lceil \frac{\lambda}{\delta_0} \rceil \qquad (1)$$

The data introduction interval is a global constraint representing the time interval between two consecutive input vectors. Figure 2 depicts pipelining during high level synthesis. Consider a data-flow graph $G(V, E)$ where V denotes the set of operations, and E denotes the set of edges, containing k resource types such as adder, multiplier, comparator etc. For each resource type k_i, given a fixed $\delta_0 \in \{1,2,3,...\lambda\}$, the operations that execute at timesteps $i.\delta_0 + l$ (where $i \in \{1,2,...N\}$ and $l \in \{1,2,...\delta_0 - 1\}$) occur concurrently, and cannot share resources amongst one another. The pseudocode of this pipelining procedure is shown in Figure 3.

Here V_n can be interchanged between V and E for pipelining both operations and edges. T_v represents the timestamp of the operation, obtained after the scheduling phase. For non-concurrent operations, $S(i, j, 1)$ stores an edge between i and j in the compatibility set S. This compatibility set is then partitioned into minimal number of cliques using a clique-partitioner. The controller for pipelined datapaths has only δ_0 states, though the number of control signals per state is more than in the case of regular flow-based high level synthesis.

```
Pipeline allocation: $PAlloc[G(V_n, T_V, \delta_0)]$
1 for $i$ in $\{v_0, v_1, v_2, ...., v_n\}$ of $V$
2     for $j$ in $\{v_0, v_1, v_2, ...., v_n\}$ of $V$
3         if $(v_i \neq v_j)$
4             if $(T_{v_i} \bmod \delta_0) \neq (T_{v_j} \bmod \delta_0)$
5                 $S(i, j, 1)$
6             else
7                 $S(i, j, 0)$
8         else
9             $S(i, j, 1)$
10 Clique_Partition$\{S\}$
```

Figure 3. Procedure for generic pipeline allocation

3 Proposed Approach

For our functional unit optimization approach, we make use of simulated annealing for performing a leakage-driven scheduling and binding. Simulated annealing is a meta-heuristic that mimics the annealing process of metals and has been widely incorporated for solving high level synthesis problems [11, 12]. We note that when $\delta_0 = 1$, all components of the datapath are active at all times. Due to the absence of idle-states in this case, any form of standby leakage reduction becomes difficult. Hence in all our approaches, we assume that δ_0 is always greater than 1, so as to impose a finite allocation and binding freedom to the operations and registers.

3.1 Functional Unit Optimization

The pseudo-code of the SA-based optimization procedure is shown in Figure 6. We constrain the moves that the operations can make, to be strictly within their mobility range. An ASAP schedule of a DFG is taken as the initial solution. The operations are then bound in the order of their timesteps and their data-initiation rates, to a matrix known as a binding matrix. The binding matrix is a two-dimensional ordering of operations and idle-states. The rows of the matrix represent timesteps, while the columns represent bound instances. A typical binding matrix containing numbered operations which is obtained after an initial ASAP schedule on an EWF filter is shown in Figure 4.

For making improvements in the schedule, we consider two kinds of moves:

1. An operation can move either “up” or “down” depending on its mobility range. For example, if an operation has a mobility of 3, it can make 3 moves (which may be a 1-step, 2-step, or a 3-step move and so on).

Proceedings of the IEEE Computer Society Annual Symposium on VLSI
New Frontiers in VLSI Design
0-7695-2365-X/05 $20.00 © 2005 IEEE

2. Operations can move within the binding matrix, after a schedule-move is performed. For example, an operation with no mobility can move to a binding instance that needs at least one more operation to be fully bound. This allows for low-mobility operations to quickly become permanently bound.

	1	2	3	4	5	6	7	8	9	10	11	12	13
T1	1	5	9	11	18	19	30	31	17	29	2	20	10
T2	3	22	23	32	33	13							
T3	4	7	8	15	16	25	28						

Multiplier

	1	2	3	4	5
T1					
T2	6	12	14	21	24
T3	26	27			

Adder

Figure 4. Initial Binding matrix

For each operation-type, a seperate binding matrix is created. For example, the EWF filter we use in our example has 2 operation-types: multiplication and addition. Hence, there are two binding matrices. There will be seperate binding matrices for different bit-width operations of the same operation-type. Once the binding matrix is created and filled with operations, it is then cost-evaluated by observing its activity-profile.

```
$COST[G(V,E)]$
1 for all instances $r \in \{1,2,3...r_n\}$ of resource type
  $k_i \in \{k_1, k_2, k_3...k_n\}$
2     Determine Active-Idle combination $Comb_r$
4     Match $Comb_r$ with Leakage_Cost table
5     cost $C(Comb_r) \leftarrow L_c \times (1 - W_L)$
6     $cost \leftarrow cost + C(Comb_r)$
9     Match $Comb_r$ with Settling_Cost table
10    cost $S(Comb_r) \leftarrow S_c \times (1 - W_S)$
11    $cost \leftarrow cost + S(Comb_r)$
12 endfor
13 return cost
```

Figure 5. Cost Evaluation procedure

Each instance of the binding matrix has its own activity profile, which is related to its combinational ordering of operations and idle-states. For a data-initiation rate of n, there will be 2^n different activity combinations for an instance. At every iteration, when the binding matrix is created, each instance is resolved for its activity combination. This gives an active-idle combination of length δ_0 which is referred to as $Comb_r$. The cost of this combination is determined from two tables namely Leakage_Cost table and Settling_Cost table. The cost-evaluation procedure is described in Figure 5.

The Leakage_Cost and Settling_Cost tables are determined through an iterative process. The Leakage_Cost table attempts to minimize the area-overhead due to MTCMOS by employing minimal number of MTCMOS instances. It also aims at minimizing the overall leakage dissipation of the datapath. The Settling_Cost table tries to reduce the number of transitions between ‘sleep’ and ‘wake-up’ that a functional instance goes through. To ensure optimal leakage dissipation, operations will be mapped to a functional instance in such a manner that there will either be no idle-states or only contiguous idle-states in the instance.

```
SA_Optim[$G(V,E)$]
1 $ASAP_G \leftarrow ASAP$ scheduling performed on $G(V,E)$
2 $ALAP_G \leftarrow ALAP$ scheduling performed on $G(V,E)$
3 $\mu(G) \leftarrow ALAP_G$ - $ASAP_G$
4 Select starting solution $x_{start} \leftarrow ASAP_G$
5 $B_{x_{start}} \leftarrow allocate\_bind(x_{start})$
6 Initial Cost $\leftarrow cost(B_{x_{start}})$
7 Current solution $S \leftarrow B_{x_{start}}$
8 Initial temperature $T \leftarrow T_0$
9 while cost is changing
10   $I \leftarrow$ number of iterations
11   while $I > 0$
12     new_solution $S' \leftarrow generate\_neighbor(S)$
13     $\Delta C \leftarrow$ cost($B_S$) - cost($B_{S'}$)
14     if ($\Delta C < 0$) or ($random(0,1) < e^{\frac{\Delta C}{T}}$)
15       $S \leftarrow S'$
16     else
17       $S \leftarrow Best\_found\_so\_far$
18     endif
19     $I \leftarrow I - 1$
20   endwhile
21   $T \leftarrow T * cooling\_rate$
22 endwhile
23 return $Best\_found\_so\_far$
```

Figure 6. Simulated annealing procedure for FU leakage optimization

For the cost function characterization and simulation of the RTL datapaths, we make use of the Fast Architectural Simulator for Leakage power (FASL) [1]. This simulator consists of a pre-characterized RT-level leakage library. For the purpose of our experiments, we use the idea of *threshold time* which we denote as the time after which the leakage profile of a cell (or gate) settles down to a steady-state value, after the change in the cell-inputs.

To obtain these cost tables, we used a sample DFG such as an FIR filter for our analysis. We synthesized the RTLs in such a way as to force the binding of the operations to match with our desired activity combinations. The RTL is then simulated using FASL, and the leakage power profile of the functional instance under observation, is extracted using a script. We average this profile for several repetitions, and this average is represented by the Leakage_Cost L_C. The Settling_Cost S_C is determined in the same way however we do not consider the leakage dissipation of the instance beyond the *threshold time*. This is so as to prioritize combinations with less transitions over those with more transitions.

Comb.	L_c	W_l	S_c	W_s
1	2.6628	0.73	1.1289	0.69
2	2.7359	0.66	1.8178	0.56
3	2.7528	0.64	1.8256	0.53
4	2.7715	0.62	1.4687	0.46
5	2.7559	0.54	1.4687	0.46
6	2.8100	0.33	1.4812	0.34
7	2.7731	0.52	1.4799	0.45
8	2.3842	0.78	1.2112	0.78

Table 1. Leakage and settling constants table for 16-bit adder activity combinations (δ_0=3)

The weightages W_l and W_s are determined by iterative comparisons between the leakage costs and the settling costs. Initially we normalize all the weights in the tables to $1/2^{\delta_0}$, before beginning our iterations. Thus if δ_0 is 3, there will be 8 combinations, and the initial weights will be 0.125. We then perform the RTL synthesis and simulations for two combinations at a time. The average leakage powers for both the combinations are compared. If combination A has a better average leakage power than combination B, combination A's weightage increases by 0.5 and B's weightage will remain the same since its a starting comparison. However after this, A's weightage will only increase or decrease by (1/8)*0.5 for future comparisons. If A has a higher leakage dissipation than another combination (say C), then its weightage reduces by (1/8)*0.5, while C's will increase by 0.5. This comparison process continues until all the combinations are compared against each other, and this is repeated for several iterations. Table 1 was obtained after extensive characterization on a 16-bit adder. This table contains both the Leakage_Cost and the Settling_Cost.

3.2 Register optimization

The lifetime information of the edges is resolved into a compatibility set using the procedure shown in Figure 3. This compatibility set is then partitioned into a minimum number of maximal-sized cliques using a modified version of the clique-partitioning algorithm presented in [2]. The algorithm aims at maximizing the idle-times of the registers and minimizing the number of transitions between active and standby states.

4 Experimental Results

We synthesized pipelined datapaths for five linear DSP benchmarks. The RTL simulations were performed using the FASL architectural simulation library. The run-times for FASL were considerably short, being typically the simulation run-times of the Cadence NCLaunch VHDL simulator, and the accuracies were within 2% of that of HSpice. The simulations were run on a Sun Ultra Sparc II machine with a Dual-200Mhz CPU and 256MB memory.

We compare our approach to an unoptimized pipeline datapath synthesized using force-directed scheduling, and bound using regular clique-partitioning. In tables 2, 3, 4, 5 and 6 we report the leakage power reductions provided by our approach for various values of data initiation rates (δ_0). In these tables, we also report the average area overhead incurred by our approach for the initial case.

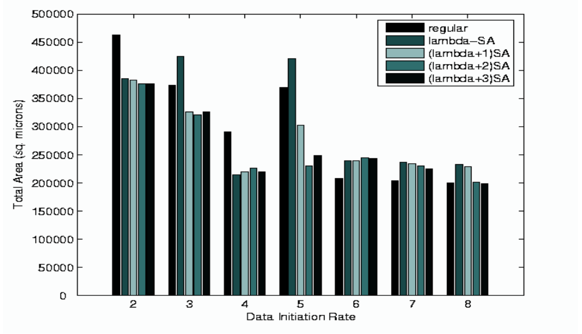

Figure 7. Total area consumption of datapath (AR filter)

For each simulation run, we simulated using 2000 random test vectors, to observe the leakage reduction. The current FASL library has simulation support for the 100nm Berkeley Predictive Technology Models, and extensive characterization has been performed for this generation. The clock period was kept at 50ns to fully observe the effects of the MTCMOS sleep transistors.

From the tables, it can be seen that our algorithm provides good leakage gains ranging from 5% to about 71%. We obtain an overall average leakage power reduction of around 36.2% with our SA-based approach over an unoptimized pipeline synthesized using FDS. We also obtain an average area overhead of 6.3% with area-reductions in a few cases, and overhead in other cases. We attribute the

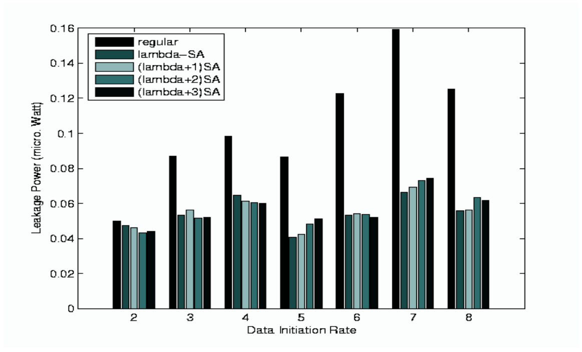

Figure 8. Leakage power profile comparison (AR filter)

area overheads incurred by our approach to the lack of large mobilities present in the benchmark designs we considered. We found that our SA-based approach was able to optimize area considerably when the latency of the schedule was increased by a small number. We illustrate this case using the AR filter example in Figures 7 and 8. For all cases, we obtained an area optimization of around 3.95%-4.64%. This proves that as we increase the latency, the area optimization provided by our approach increases. However, a small reduction in leakage power optimization is seen for a few cases when the latency is increased.

5 Conclusion

In this work, we have proposed a leakage reduction methodology for datapaths that are functionally pipelined in nature. Our algorithm intelligently determines a schedule and a binding that has low area and reduced leakage power. Although we have shown the working of our algorithm for pipelined datapath synthesis, our algorithm can be easily extended to regular high level synthesis. Our approach is beneficial towards both portable and desktop applications, as it provides advantages of low-leakage, high performance and low area-overhead datapaths.

References

[1] C. Gopalakrishnan and S. Katkoori. "An architectural leakage power simulator for VHDL structural descriptions". In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, pages 211–212, Feb 2003.

[2] C. Gopalakrishnan and S. Katkoori. "Resource allocation and binding approach for low leakage power". In *Proceedings of 16th International Conference on VLSI Design*, pages 297–302, 2003.

[3] C. Hwang, Y. Hsu, and Y. Lin. "PLS: A scheduler for pipeline synthesis". *IEEE Transactions on Computer-Aided Design of Circuits and Systems*, 12(9):1279–1286, Sep 1993.

[4] D. Kim, D. Shin and K. Choi. "Low power pipelining of Linear Systems: A common operand centric approach". In *International Symposium on Low Power Electronics and Design*, pages 225–230, 2001.

[5] H. Jun and S. Hwang. "Design of a pipelined datapath synthesis system for digital signal processing". *IEEE Transactions on Very Large Scale Integration Systems*, 2(3):292–303, Sep 1994.

[6] K. S. Khouri and N. K. Jha. "Leakage power analysis and reduction during behavioral synthesis". *IEEE Transactions on Very Large Scale Integrated Systems*, 10(6):876–885, Dec 2002.

[7] M. Powell, S. Yuang, B. Falsafi, K. Roy, T.N. Vijaykumar. "Gated-Vdd: a circuit technique to reduce leakage in deep sub-micron cache memories". In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 90–95, July 2000.

[8] N. Hanchate and N. Ranganathan. "LECTOR: A Technique for Leakage Reduction in CMOS circuits". *IEEE Transactions on Very Large Scale Integrated Systems*, 12(2):196–205, Feb 2004.

[9] N. Park and A. C. Parker. "Sehwa: A software package for synthesis of pipelines from behavioral specifications". *IEEE Transactions on Very Large Scale Integrated Systems*, 7(3):356–370, Mar 1988.

[10] P. G. Paulin and J. P. Knight. "Force-directed scheduling for the behavioral synthesis of ASICs". *IEEE Transactions on Computer-Aided Design of Circuits and Systems*, 8(6):661–679, June 1989.

[11] P. Prabhakaran, P. Bannerjee, J. Crenshaw and M. Sarrafzadeh. "Simultaneous scheduling, allocation and floorplanning for interconnect power optimization". In *Proceedings of the 12th International Conference on VLSI design*, pages 423–427, Jan 1999.

[12] S. Devadas and A. R. Newton. "Algorithms for hardware allocation in data path synthesis". *IEEE Transactions on Computer-Aided Design of Circuits and Systems*, 8(7):768–781, July 1989.

[13] S.Borkar. "Design challenges of technology scaling". *IEEE Micro*, 19(4):23–29, Aug 1999.

[14] V. Sundararajan and K. K. Parhi. "Low power synthesis of dual threshold voltage CMOS VLSI circuits". In *Proceedings of the International Symposium of Low Power Electronics and Design*, pages 139–144, Aug 1999.

[15] Y. Ye, S.Borkar and V. De. "A new technique for standby leakage reduction in high performance circuits". In *Digest of Tech. Papers Symposium VLSI Circuits*, pages 40–41, June 1998.

[16] Z. Chen, M. Johnson, L. Wei and K. Roy. "Estimation of standby leakage power in CMOS circuits considering accurate modelling of transistor stacks". In *Proceedings of Low Power Electronics and Design*, pages 239–244, Aug 1998.

δ_0	Regular		SA-based		Leakage	Area
	Leakage (μW)	Area (μ^2)	Leakage (μW)	Area (μ^2)	Reduction(%)	Overhead(%)
2	0.01941	140727.67	0.01364	151928.23	29.72	7.95
3	0.01841	138768.96	0.013784	111322.49	25.12	-19.77
4	0.02407	96923.78	0.01794	110040.66	25.46	13.53
5	0.01489	94654.53	0.01108	106581.12	25.58	12.61

Table 2. FIR filter analysis (Operations: 5*, 4+)

δ_0	Regular		SA-based		Leakage	Area
	Leakage (μW)	Area (μ^2)	Leakage (μW)	Area (μ^2)	Reduction(%)	Overhead(%)
2	0.03781	254996.51	0.03245	272959.93	14.17	7.04
3	0.05430	240736.48	0.03026	227423.48	44.27	-5.53
4	0.06793	233283.65	0.02702	283322.18	60.22	21.44
5	0.06583	187234.59	0.04085	173279.12	37.94	-7.45
6	0.09305	147085.09	0.04134	172924.48	55.57	17.56
7	0.10307	180952.28	0.04249	211773.85	58.77	17.03
8	0.09553	140014.73	0.04200	163622.51	56.03	16.86
9	0.08028	137864.93	0.05388	163764.76	32.88	18.78
10	0.12168	138199.39	0.05586	165383.15	54.07	19.66
11	0.20879	137984.42	0.05995	164570.98	71.28	19.26
12	0.20763	137864.90	0.05853	160621.64	71.81	16.50

Table 3. EWF filter analysis (Operations: 7*, 26+)

δ_0	Regular		SA-based		Leakage	Area
	Leakage (μW)	Area (μ^2)	Leakage (μW)	Area (μ^2)	Reduction(%)	Overhead(%)
2	0.01689	137264.06	0.01390	149942.93	17.70	9.23
3	0.02756	136069.75	0.02141	154207.84	22.31	13.33
4	0.03563	134015.48	0.02609	152118.56	26.77	13.54

Table 4. IIR filter analysis (Operations: 5*, 4+)

δ_0	Regular		SA-based		Leakage	Area
	Leakage (μW)	Area (μ^2)	Leakage (μW)	Area (μ^2)	Reduction(%)	Overhead(%)
2	0.05005	462236.62	0.04744	385632.81	5.21	-16.57
3	0.08714	373625.06	0.05334	424518.06	38.78	13.62
4	0.09851	290627.65	0.06460	213779.62	34.42	-26.44
5	0.08660	368872.00	0.04060	420956.72	53.11	14.12
6	0.12258	207223.70	0.05315	238991.09	56.64	15.33
7	0.15931	203903.62	0.06653	236505.41	58.23	15.98
8	0.12515	199795.06	0.05584	232221.79	55.39	16.23

Table 5. AR filter analysis (Operations: 16*, 12+)

δ_0	Regular		SA-based		Leakage	Area
	Leakage (μW)	Area (μ^2)	Leakage (μW)	Area (μ^2)	Reduction(%)	Overhead(%)
2	0.07016	453783.03	0.06115	417946.59	12.83	-7.89
3	0.09974	403099.18	0.08181	339320.12	17.96	-15.82
4	0.12847	278065.46	0.07344	245987.60	42.83	-11.53
5	0.06097	274339.40	0.03505	242233.36	42.51	-11.70
6	0.07164	267173.50	0.03442	300249.57	51.95	12.38

Table 6. FFT filter analysis (Operations: 16*, 25+, 7-)