Power-conscious High Level Synthesis Using Loop Folding

Daehong Kim Kiyoung Choi
School of Electrical Engineering
Seoul National University, Seoul, Korea, 151-742
E-mail: daehong@poppy.snu.ac.kr

Abstract

In this paper, a transformation technique, called power-conscious loop folding is proposed for high level synthesis of a low power system. Our work is focused on reducing the power consumed by functional units through the decrease of switching activity in a data path dominated circuit containing loops. The transformation algorithm has been implemented and integrated into a high level synthesis system for experiments. In our experiments, we could achieve power reduction of up to 50% for circuits dominated by functional units.

1. Introduction

Until 1980's, one of the most important factors that determined the quality of a system was the speed and so much effort had been made to increase the speed at a minimal cost or silicon area. But, in 1990's, as the portable system market grows rapidly and reliability problems due to high power dissipation are becoming an issue for systems operating in high clock frequency, low power design is becoming more and more important and is now one of the major concerns in system design.

There have been many researches for low power design at low levels of abstraction and many effective techniques have been proposed [1] [2] [3]. However, if we consider low power design at higher levels of abstraction, we can obtain much more effective power reduction. At higher levels of abstraction, we can apply various transformation techniques to system design with wider view and obtain much more power reduction with less cost and effort. In this paper, we focus on transformation techniques for high level synthesis of low power systems.

There have been quite a few researches in high level low power design. Chandrakasan utilized various transformations to minimize the power in application specific data-intensive circuits [10] [13]. His approach is to introduce more concurrency in a circuit to speed it up relatively and then to induce low power by reducing the voltage down to the minimum without violating the original speed constraints. To reduce the supply voltage ultimately, he used transformation techniques such as loop unrolling, retiming, and pipelining. Loop unrolling is used to increase concurrency and retiming and pipelining are used to reduce the length of critical paths. Although capacitance increases linearly due to the growth of parallelism, the total power dissipation decreases because of the quadratic power reduction effect of the lowered supply voltage. Most of the transformations used are basically the same as the conventional high level transformations, but different cost functions are used to evaluate the results obtained through such transformations.

In [12], scheduling and resource binding algorithms for low power data path design are proposed. They minimize the number of transitions on the signals feeding functional units(adders, multipliers, etc.) and registers, which effectively minimize the switched capacitance. This is achieved by scheduling the candidate nodes in control steps as close as possible and binding them to the same resource. The candidate nodes are selected such that there is no change of values in the input operands among consecutive operations of the same functional unit. In addition, in [14], high level transformation techniques such as loop interchange and operand reordering are proposed to reduce the activity of functional units.

As a basic control-flow element, loop has been one of the major targets of various transformations for optimization of the throughput and the resource utilization. In [5] and [11], a technique called loop folding, is presented to obtain a significant improvement in the utilization of parallel hardware and the throughput. In this paper, we divert the traditional concept of loop folding to the low power design. Our work is focused on reducing the power consumption due to the switching. We use the loop folding technique to minimize the number of transitions in the input operands and thereby to reduce the power consumption. The transformation algorithm is incorporated into HYPER [9], a high level synthesis program, so that we can obtain synthesized hardware from a Silage specification. For the evaluation of our algorithm, we measure the power reduction using SPA [8], a power analysis program, mutatis mutandis.

This paper is organized as follows. Section 2 describes the basic concept about operand sharing, power-conscious loop folding, and their effects on the power consumption. In Section 3, the transformation algorithm is described. The experimental results are presented In Section 4. Finally, conclusions and future work are given in Section 5.

2. Basic Concept

In this section we examine the effect of operand sharing on power consumption. Then we describe how the power-conscious loop folding reduces power consumption through the operand sharing.

2.1 Operand Sharing

It is the power consumed in data paths that accounts for a large fraction of overall power budget in a data path dominated

34th Design Automation Conference®

DAC 97, Anaheim, California

application specific circuit such as DSP. For this reason, various techniques that reduce the switching activity in functional units by minimizing the change in the input operands have been proposed. Operand sharing described here is one of the techniques. Generally, the power consumption (switching power) is dependent on the correlation of the input operands. The switching power decreases if the correlation between two consecutive set of input operands to the same functional unit is high. However, an accurate computation of the correlation is very time-consuming because it requires an exhaustive simulation.

Typically high level synthesis system takes CDFG(Control Data Flow Graph) as the intermediate form, where nodes represent operations that are to be bound to functional units and edges represent control or data flows. During the synthesis, multiple operation nodes can be implemented by one functional unit through hardware sharing. In this case, the switching activity of the functional unit may increase because the input operands change between the executions of the two operations. Operand sharing technique binds one identical functional unit to more than two operation nodes that have at least one common input operand. It achieves switching power reduction by maximizing the temporal correlation of input signals to a functional unit. Assume that P1 is the average power consumption of a binary functional unit when only one operand changes and P2 is the average consumption when both operands change simultaneously and α is the ratio P1/P2. If one functional unit is shared by n operation nodes having one common input operand, the power reduction is given by

$$power\ reduction = 1 - \frac{power\ consumed\ after\ operand\ sharing}{power\ consumed\ before\ operand\ sharing}$$

$$= 1 - \frac{P_2 + (n-1)P_1}{nP_2}$$

$$= (n-1)(1-\alpha)/n$$

Considering that typical value of α is about 0.65 for a 12bit multiplier and 0.75 for a 12 bit adder [14], power reductions of 26% and 18% are obtained for n = 4 respectively.

The scheduling algorithm proposed by Musoll utilizes this operand sharing technique to obtain power reduction from 5% to 8% over conventional scheduling algorithms. The limitation of the above techniques rises from the fact that for most designs including DSP applications, it is not easy to find common input operands. This paper presents a novel loop transformation called power-conscious loop folding which finds common input operands hidden in a loop. This transformation has a significant power-reducing effect on DSP applications such as filters.

2.2 Loop Folding

The loop folding technique proposed here is somewhat different from the conventional one, although the basic process of folding loop iterations is the same.

2.2.1 Conventional Loop Folding

Loop folding is a transformation technique which reduces the execution time of a loop or improves the utilization of the resource by introducing partial overlaps between the execution times of successive loop iterations in the original description. Figure 1 is an example of loop folding that reduces the execution time of a loop. Assume that one adder and one multiplier are allocated. If an adder node in time step 4 is moved to the next loop iteration, the total latency of the loop body is reduced from 4 to 3.

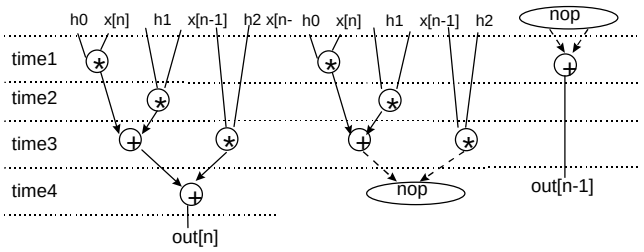

Figure 1. Before and after the conventional loop folding.

2.2.2 Power-Conscious Loop Folding

While conventional loop folding aims at reducing the execution delay of a loop or obtaining high utilization of the resource, the proposed loop folding aims at reducing the switching activity (hence the power consumption) of a functional unit by minimizing changes in input operands to the functional unit. The effect of the proposed technique is quite significant for DSP applications such as filters. The reason is found in the observation that for most DSP applications, the following form of equations is frequently used in their behavioral specifications.

$$y[n] = \sum_i h_i x[n-i]$$

As shown in the above equation, the sum of products of a constant value and a delayed signal value determines the output value. The equation implicitly represents a loop. Assume that one multiplier is shared by the multiplication operations required in the equation. The switching activity of the multiplier is determined by the changes of values of the two input operands occurring between each pair of consecutive executions. Note that, without any transformation, both input operands change their values every iteration. However, if we fold two consecutive iterations in such a way that $h_i x[n-i]$ for y[n] and $h_{(i+1)}x[(n+1)-(i+1)]$ for y[n+1] are computed consecutively, we can save power because one input to the multiplier does not change. In this case, we need some additional codes for start-up and clean-up, but their effect is negligible provided that a large number of loop iterations are assumed.

The example in Figure 2 illustrates step by step power-conscious loop folding described above. The example represents the operation of a 4^{th} order FIR filter. As shown in the figure, the number of loop iterations is decreased as the folding steps proceed and the start-up and clean-up codes are modified accordingly. If the number of delay terms is n (4 in our example), the maximum number of folding steps is n-1 (3 in our example). Every step, registers which save the results of multiplication are produced and additional codes for start-up and clean-up are inserted. The total number of registers required increases because the life times of the newly produced registers are relatively long. They can hardly be shared by multiple variables. In our example, total overhead of about 6 registers is necessary. But, in a typical data path, the fraction of the register overhead in area and power is small because relatively large functional units such as multipliers dominate the area and power consumption. Refer to section 5 for experimental results. In this paper, we focus on power reduction in multipliers only.

Assume that Pmul1 is the average power consumed by a multiplier when only one operand changes and Pmul2 is the average power consumed when both operands change simultaneously and α is the ratio Pmul1/Pmul2. In our example, if only one multiplier is shared by the 4 multiplication operations, the power consumption before the loop folding is 4·Pmul2 and that after the folding is (Pmul2 + 3·Pmul1), resulting in power reduction of 75(1-α)% as explained above. Power reduction of about 26% can be obtained for a typical value of α (0.65 for a 12 bit multiplier). Because the power consumed by multiplier accounts for a large fraction of the total power budget, the effect of overall power reduction corresponding to reduction of multiplier can be taken.

Generally if there are (n+1) delay terms and n folding steps are performed, we obtain power reduction of (n-1)(1-α)/n and overhead of n(n-1)/2 registers.

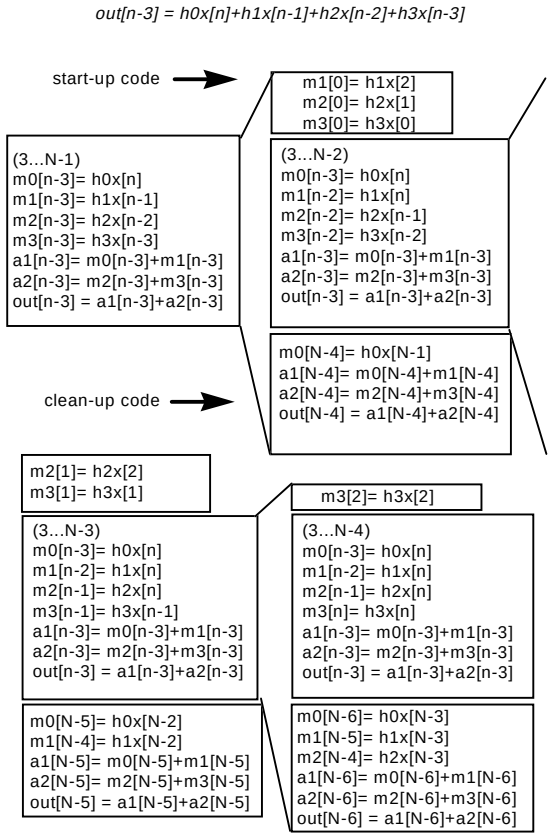

Figure 2. Steps for power-conscious loop folding.

3. Algorithm for Power-Conscious Loop Folding

Before we describe the algorithm, let's define terms: base index and base operand. The base index for a loop is the largest index used for input operands in the loop body and the base operands are the corresponding input operands.

Power-conscious loop folding consists of the following three steps.

1. Increase the index (decrease the delay) by one for all the input operands except for the base operands.
2. Increase by one the index of the result of each multiplication whose operand has an index increased at step 1. This process generates a new variable(register) which will be used in the following iterations.
3. Generate start-up and clean-up codes

Above steps are repeated until the indices of all input operands are the same as the base index.

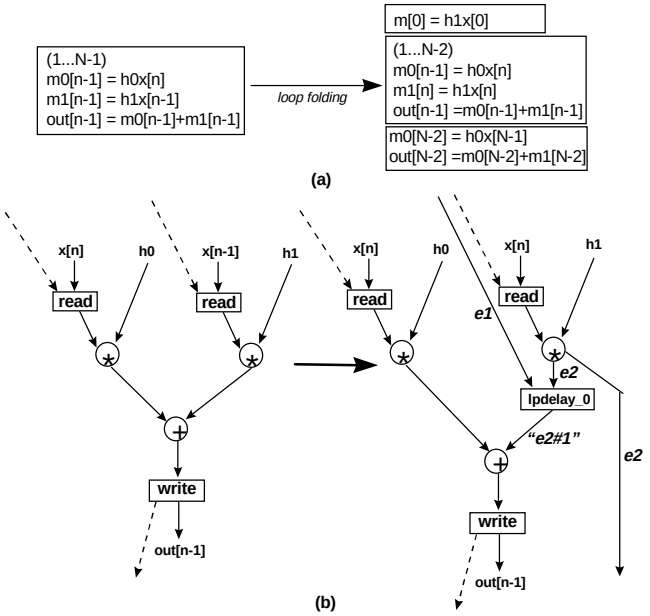

Figure 3. CDFG before and after power-conscious loop folding.

Figure 3 shows a simpler example.

Figure 3(b) shows two CDFGs: one before the power-conscious loop folding and one after the folding. The nodes represent functional operations, read operations, write operations, and delays and the edges show the dependency(the dashed lines and solid lines indicate control and data dependency respectively). After one iteration of the loop folding, vertex lpdelay_0 and edges e1,e2, and "e2#1" are inserted as shown in the CDFG on the right hand side of Figure 3(b). Edge e1 indicates the data dependency from the CDFG for the start-up code and edge e2 indicates the data dependency to the CDFG for the clean-up code as well as the data dependency to the delay node. Vertex lpdelay_0 is a queue that saves the multiplication results which will be used in the following iterations. Edge "e2#1" indicates the data dependency between the output from the multiplier and the input to the adder across one iteration. An edge "e#n" indicates data dependency across n iterations.

Each of the three steps described above corresponds to each of the following steps applied to the CDFG.

1. Replace x[n-k] by x[n-k+1] for all the input operands other than base operands.
2. Insert a delay node and edges between the multiplication node having an input operand modified at step 1 and the addition node unless there exists a delay node. Name the output edge of the delay node "e#1". If there already exists a delay node with an output edge named "e#n", just replace the name by "e#(n+1)"

3. Generate two additional CDFGs -one for the start-up code and the other for the clean-up code- and insert edges to represent the dependency among the three CDFGs.

The CDFG described here is a hierarchical graph, which is suitable for representing control constructs such as loops, conditional branches, etc. In the CDFG, a loop is represented by a node at the right upper level of hierarchy.

Figure 4 shows the pseudo-code for the transformation algorithm.

Routine ***PCLP()*** searches the CDFG for loop nodes from the top level and then searches each loop for nodes that are inputs to multiplication nodes. Then it determines the base index and base operands and starts repeating the folding operation by calling routine ***Folding(),*** which modifies the CDFG by increasing indices of input operands, inserting delay nodes and edges, and creating graph nodes and edges for the start-up and clean-up codes.

```
PCLP() {
    Search graph nodes for loop
    For each  loop {
        Search for Input nodes to multiplication
        Find base index and base operands
        Repeat {
            Call Folding() for one step of folding
        } until (all the input nodes have the same
            index as the base operands)
    }
}
Folding(){
        For all the non-base input nodes {
                Increase index by one
                If (any delay node between
                 multiplication node and adder node) {
                        Increase the value in the name of the
                         output edge of the delay node by one
                 } else {
                        Create the delay node and edges
                }
                Create nodes and edges for start-up and
                 clean-up codes
        }
}
```

Figure 4. Pseudo-code for transformation algorithm.

4. Experimental Results

The transformation algorithm was incorporated into HYPER. HYPER takes a Silage specification and converts it to the flowgraph database format (.afl file). We applied the proposed transformation algorithm to the .afl files and generated SDL files using scheduling and hardware mapping routines of HYPER. In this section, we estimated the power consumed by the functional units in the circuits and the total power consumed by the whole circuits. We compare the estimation results for circuits synthesized with and without loop folding. For the estimation of power, we utilized SPA. Because SPA does not support control constructs such as loops, we estimated the power consumption of a loop for one iteration. In all the circuits, only one multiplier was allocated. Table 1 shows the comparison results obtained for 4 circuits: 11^{th} order fir filter, wavelet filter, noise canceller, and volterra filter.

The third column compares the switched capacitance values estimated for functional units and for the entire circuit. The fourth column represents the power consumed in one iteration of the loop respectively. In the fifth column, we give the ratio of the power consumed at the functional unit to that consumed at an entire circuit. This ratio indicates the fraction of the power consumed by functional units in the total power budget of the circuit and hence shows how functional units dominate the rest of the circuit in power's point of view. The last column of the table shows the estimated power reductions achieved by the loop folding transformation technique in functional units and in the whole circuit. The effect of the power reduction in functional units is compensated by the power consumed by the newly added multiplexers and registers. Moreover, the effect is somewhat hidden by the power consumed in the control units. That is why the number for the entire circuit is smaller than that for the functional units.

Table 1. Comparison of power consumption.

	Fold-ing	switched capacitance (pF)		power (mW)		$\frac{pow_{func}}{pow_{tot}}$	power reduction (%)	
		func	total	func	total		func	total
1	with-out	2119	3147	44.1	65.6	67.3	58.7	51.2
	with	800	1407	18.2	32.0	56.9		
2	with-out	683	1632	11.4	27.2	41.8	20.2	11.4
	with	544	1446	9.1	24.1	37.6		
3	with-out	1675	3767	10.7	24.1	44.5	-7.5	-15
	with	1523	3651	11.5	27.7	41.7		
4	with-out	19	327	0.3	4.5	5.7	0	-30
	with	16	312	0.3	5.2	5.2		

1. 11^{th} order fir filter 2. Wavelet filter
3. Noise canceller 4. Volterra filter

For the *11^{th} order fir filter*, we obtained a power reduction greater than we expected. Such an unusual reduction is related to the symmetry of constants, fed to one of the two inputs of the multiplier. Although the original description of the filter has 11 multiplications in one iteration of the loop body, the transformed one has only 6 effective multiplications as shown in Figure 5. So the multiplier consumes only half of the power that is consumed in usual case where there is no symmetry of constants. For this reason, the symmetry of constants amplifies the effect of power-conscious loop folding. Of course, this effect is cancelled if the original description does summations such as (In[n-10]+In[n]), (In[n-9]+In[n-1]), etc. first and then does multiplications with a0, a1, ..., a5 to obtain the final result.

For the *adaptive noise cancellation using LMS(Least Mean Square) algorithm* and *volterra filter*, we observe that both of power consumed by the functional units and the entire circuit increase contrary to the decrease in the switched capacitance shown in the third column of Table 1. It is because the latency is reduced due to the loop folding as shown in Table 2. The reduction of latency is another merit of our method. For a fair comparison, we estimated the energy reduction both in functional units and in the entire circuit as shown in Table 2. The table also shows the estimation of area obtained with and without folding. We allocated the same number of functional units. In spite of the increase in power, we observe that the energy is reduced even for

the noise canceller and volterra filter. However, the amount of reduction for these two circuits is still small compared to the other circuits. The reason is as follows. In the *noise canceller*, the number of products of a constant and a delayed signal is small compared to the total number of products. In the *volterra filter*, the power consumed by functional units does not dominate the total power of the circuit. In particular, in the *volterra filter* the ratio on the fifth column of Table 1 is so small that the energy reduction of 20% in the functional units results in the reduction of just 4.9% in the entire circuit. So power-conscious transformation is not very suitable for these kinds of applications. As can be observed, however, it is possible to achieve a power reduction of up to 50% in circuits whose power is dominated by that of functional units.

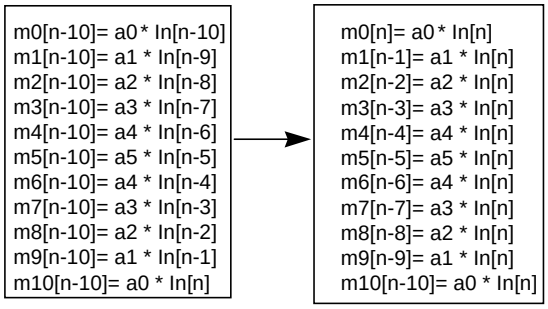

Figure 5. Symmetry of constants.

Table 2. Comparison of area, latency, and energy.

	Fold-ing	area (mm^2)	latency (clock cycles)	energy (nJ)		energy reduction (%)	
				func	total	func	total
1	with-out	8.98	12	53.0	78.7	62.2	55.2
	with	9.63	11	20.0	35.2		
2	with-out	3.45	15	17.1	40.8	20.5	11.3
	with	4.67	15	13.6	36.2		
3	with-out	3.53	39	41.9	94.2	9.1	3.1
	with	3.78	33	38.1	91.3		
4	with-out	1.00	18	0.5	8.2	20	4.9
	with	1.08	15	0.4	7.8		

The increase in the estimated area, given in the first column of Table 2, describes the overhead mainly due to registers. The amount of overhead is proportional to the number of folding steps.

5. Conclusions and Future Work

In this paper, we discussed a high level transformation technique, called *power-conscious loop folding*. This transformation allows us to obtain a significant power reduction in DSP applications such as filters. Such a reduction is based on reducing the switching activity of functional units by minimizing changes in input operands to the functional units. With the loop folding transformation, a significant power reduction is achieved in circuits whose power consumption is dominated by that of functional units, even though we have some additional code overhead for start-up and clean-up and some additional register overhead.

The transformation technique has been tested with some DSP applications. The results show that it is possible to obtain a power reduction of up to 50% for circuits such as fir filters But, this technique is not so effective in circuits whose power is dominated by that of control units.

In this paper, the impact of the correlation between constants fed to the multiplier when we apply the power conscious loop folding has not been addressed. We are currently trying to further reduce power consumption through scheduling using the correlation between constant operands. We are also planning to apply the proposed technique to the generation of DSP application software running on DSP processors to achieve system level power reduction.

References

[1] V. Tiwari, P. Ashar, and A. Malik, “Technology mapping for low power,” In Proc. of Design Automation Conf., pp. 74-79, 1993.

[2] J. Monteiro, S. Devades, and A. Ghosh, “Retiming sequential circuits for low power,” In Proc. of the IEEE Int. Conf. on Computer-Aided Design, pp. 398-402.

[3] C. Tsui, M. Pedram, and A. Despain, “Technology decomposition and mapping targeting low power dissipation,” In Proc. of Design Automation Conf., pp. 68-73, 1993.

[4] A. P. Chandrakasan and R. W. Brodersen, “Minimizing power consumption in digital CMOS circuits,” Proc. of the IEEE, vol. 83, pp. 498-523, Apr. 1995.

[5] C. T. Hwang, Y. C. Hsu, and Y. L. Lin, “Scheduling for functional pipelining and loop winding,” In Proc. of Design Automation Conf., pp. 764-769, 1991.

[6] P. N. Hilfinger, Silage reference manual, 1993.

[7] A. Raghunathan and N. K. Jha, “An iterative improvement algorithm for low power data path synthesis,” In Proc. of the IEEE Int. Conf. on Computer-Aided Design, pp. 597-602, 1995.

[8] P. E. Landman and J. M. Rabaey, “Architectural power analysis: the dual bit type method,” IEEE Trans. on VLSI Systems, vol. 3, pp. 173-187, June 1995.

[9] C. Chu, et al., “HYPER: An interactive synthesis environment for high performance real time applications,” In Proc. of the IEEE Int. Conf. on Computer Design , Nov. 1989.

[10] A. P. Chandrakasan, M. Potkonjak, R. Mehra, J. M. Rabaey, and R. W. Brodersen, “Optimizing power using transformations,” IEEE Trans. on Computer-Aided Design, vol. 14, pp. 12-30, Jan. 1995.

[11] G. Goossens, J. Vandewalle, and H. D. Man, “Loop optimization in register-transfer scheduling for DSP-systems,” In Proc. of Design Automation Conf., pp. 826-831, 1989.

[12] E. Musoll and J. Cortadella, “Scheduling and resource binding for low power,” In Proc. of Int. Symp. on System Synthesis, pp.104-109, 1995.

[13] A.P. Chandrakasan, M. Potkonjak, J. M. Rabaey, and R. W. Brodersen, “HYPER-LP: A system for power minimization using architectural transformations,” IEEE Trans. on Computer-Aided Design, pp. 300-303, Nov. 1992.

[14] E. Musoll and J. Cortadella, “High-level synthesis technique for reducing the activity of functional units,” in Proc. of Int. Symp. on Low Power Design, pp. 99-104, 1995.