Power Minimization of Functional Units by Partially Guarded Computation

Junghwan Choi, Jinhwan Jeon, and Kiyoung Choi
School of Electrical Engineering, Seoul National University
Seoul 151-742, Korea
{exotic,jeonjinh,kchoi}@poppy.snu.ac.kr

Abstract
This paper deals with power minimization problem for data-dominated applications based on a novel concept called partially guarded computation. We divide a functional unit into two parts – MSP (Most Significant Part) and LSP (Least Significant Part) - and allow the functional unit to perform only the LSP computation if the range of output data can be covered by LSP. We dynamically disable MSP computation to remove unnecessary transitions thereby reducing power consumption. We also propose a systematic approach for determining optimal location of the boundary between the two parts during high-level synthesis. Experimental results show about 10~44% power reduction with about 30~36% area overhead and less than 3% delay overhead in functional units.

Keywords
Low Power, Partially Guarded Computation

1. Introduction
Recently, electronics systems market has proliferated rapidly toward portable computing and communication systems thereby increasing demands for considering low power during VLSI design [1, 2]. From the viewpoint of long battery life and high reliability, power dissipation has become one of the major objectives during synthesis procedure. In CMOS circuits, most of the power dissipation is caused by charging and discharging load capacitance of gates. Therefore, it is crucial to minimize the number of signal transitions in circuits for low power design.

In high-level synthesis domain, there have been quite a few studies devoted to minimize transitions in functional units, registers, multiplexers, and buses [3 - 11]. Many of them focus on minimizing transition activity in functional units because they are the main source of power dissipation in data dominated applications [3 - 8]. The most effective method to reduce the number of transitions in functional units is increasing the correlation of input data. Therefore, many of the previous work focus on increasing input data correlation by changing operation binding [3, 8], loop pipelining [7], loop interchange, operand reordering, operand sharing, unrolling [5], and guarded evaluation [11].

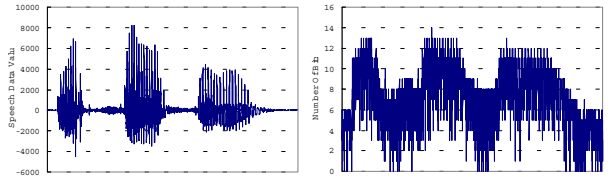
ISLPED '00, Rapallo, Italy.

In this paper, we propose yet another technique which we call partially guarded computation. The technique disables a part of a functional unit based on dynamic range of input operands. We divide a functional unit into two parts – MSP (Most Significant Part) and LSP (Least Significant Part) - and allow only the LSP computation when the range of input operands is covered by the range of the LSP. For the division of a functional unit, we propose a systematic method that finds the location of boundary that maximizes power reduction. We also propose an effective operation and operand binding algorithm for high level synthesis in order to maximize the effect of the proposed technique.

This paper is organized as follows. In section II, we present the basic concept of partially guarded computation and explain the application to adders and multipliers. Section III proposes an algorithm for dividing a functional unit into two parts. In section IV, we propose an effective operation binding algorithm for maximizing power reduction by our partially guarded computation technique. We show experimental results in section V and conclude our work in section VI.

2. Partially Guarded Computation

2.1 Basic Concept
In designing signal processing applications, we determine the word lengths of functional units based on dynamic range of input data such that maximum range of the data does not exceed the word length of functional units. However, real data is generally limited to small range in most cases, and the case of maximum range rarely occurs. Figure 1 shows our motivational example which is composed of a short segment of a speech data and associated range information. As shown in Fig. 1 (b), the maximum range of the speech data is 14 bits. However, about 60 % of the data do not have range larger than 8 bits. The extra 6 bits belong to sign extension region of the data. Generally, we don't need to perform expensive computation for the sign extension region because we can compute sign bit by looking at only the LSB side. If we assume that the speech data is used as one input of an adder and dynamic range of the other input does not exceed 8 bits at all, we don't need to perform computation for the sign extension bits during 60% of total execution cycles. By performing only 8 bit addition for such cases, we can reduce unnecessary transitions in the sign extension part of the adder thereby reducing power consumption. Disabling the computation of the sign extension part can be achieved by preventing propagation of input data transition to the sign extension part of a functional unit.



(a) (b)

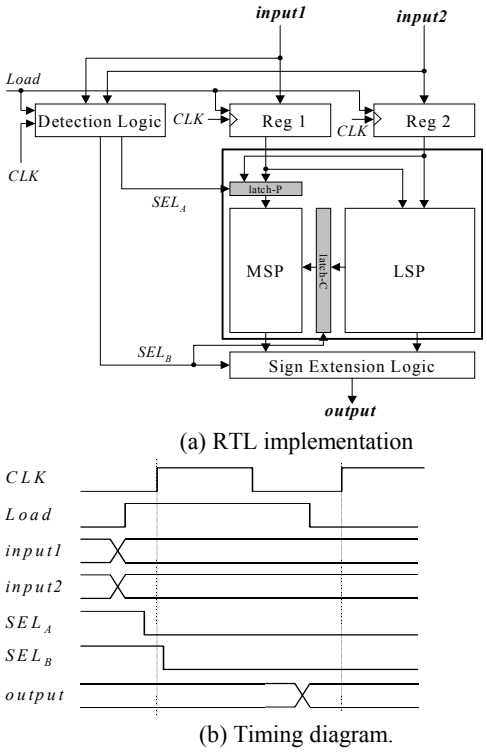
Fig. 1. Example speech data and associated range.

Figure 2 illustrates an RTL implementation of our partially guarded computation and timing diagram for the circuit operation. We assume that input registers are reserved for each functional unit, where the registers play the role of guard latches for guarded evaluation technique [11]. We divide the output data of a functional unit into two parts - MSP and LSP – such that the range of the output data does not exceed the maximum range of LSP in as many cases as possible while keeping large range of MSP. The bit lengths (or maximum ranges) of MSP and LSP are denoted as N^M and N^L, respectively. Corresponding to the division of the output data, the functional unit is also divided into two parts: MSP and LSP. The functional unit performs computation in only the LSP when the detection logic signals that the range of the output data will not exceed N^L. Otherwise, the functional unit performs computation in both MSP and LSP. To disable the computation in MSP, we use guarded evaluation technique by inserting guard latches to the inputs of the MSP. The inputs are composed of the MSB side of primary inputs and carry inputs propagated from the LSP. When the functional unit performs only LSP computation, sign extension logic produces correct output data by extending sign bit from the output data of LSP.

The output signal of detection logic, which is denoted as SEL_A, is connected to the enable signal of guard latches. The inputs of the detection logic are not connected to the outputs of input registers but to the inputs of input registers to have SEL_A asserted before the next input data is loaded into the guard latches, which is crucial for correct guarding. However, SEL_B, which is connected to the sign extension logic, is asserted after the rising edge of clock. This is to guarantee the output of the functional unit to be loaded correctly into the input register of other functional units. The detection logic asserts zero when both input1 and input2 generate output data ranges not exceeding N^L. The detection logic can be implemented simply by using N^M input AND gates or N^M input NOR gates[1] and D-latches.

To reduce unnecessary power dissipation in detection logic, we implement the circuit such that it is enabled only when the register load signal is enabled. When the detection logic is disabled, it holds the previous value of SEL_A by using internal latch. Since it takes time for the input1 and input2 to be stable, there may occur glitches on SEL_A. There are two types of glitches ($1 \rightarrow 0 \rightarrow 1$ glitch and $0 \rightarrow 1 \rightarrow 0$ glitch) according to the initial state of SEL_A. The first type glitch ($1 \rightarrow 0 \rightarrow 1$) does not induce much power dissipation because $1 \rightarrow 0$ and $0 \rightarrow 1$ transitions in SEL_A generate no transitions in the already enabled MSP. However, the second type ($0 \rightarrow 1 \rightarrow 0$) causes unnecessary transitions in the previously disabled MSP. Therefore, we remove the second type glitch by delaying SEL_A until *CLK*'s rising edge when the value of SEL_A is 1. In this case, SEL_A may be asserted after input data is loaded. However, it does not break our guarding scheme because SEL_A needs to be asserted before *CLK* only when SEL_A changes from 1 to 0.

[1] We need to check leading successive one's and zero's for both input1 and input2.



(a) RTL implementation

(b) Timing diagram.

Fig. 2. RTL implementation of partially guarded computation circuitry and timing diagram.

2.2 Implementation of Functional Units

Figure 3 illustrates the implementation of partially guarded circuitry for ripple carry adder. Since there may occur overflow when we add two numbers, we set the input bit length of LSP as N^L-1. Sign extension logic can be simply implemented by using multiplexers.

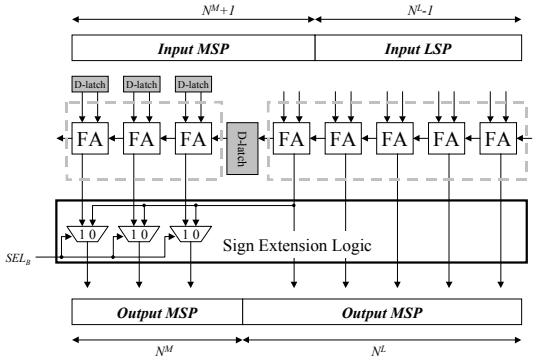


Fig. 3. Partially guarded circuitry for ripple carry adder.

Figure 4 illustrates the implementation of partially guarded circuitry for signed array multiplier. We insert the guard latches between the MSP and LSP of the full adder array. Let N_1^S and N_2^S be the bit length of sign extension region for input1 and input2, respectively. The bit length of the sign extension region of the output is computed as $N_1^S+N_2^S$. If it exceeds the range of

MSP, then we disable the latches and the MSP and activate only the LSP. However, dynamically extracting the values of N_1^S and N_2^S from the input patterns and computing the values $N_1^S+N_2^S$ to see if it exceeds the range of the MSP require too much computation resulting in high circuit overhead and power consumption. To avoid such problem we disable the MSP only when N_1^S and N_2^S independently exceed their own pre-determined bounds N_1^M and N_2^M, respectively. There can be one or more combinations of bounds N_1^M and N_2^M such that $N_1^M+N_2^M=N^M$. In the next section, we present an algorithm that determines the values of N_1^M and N_2^M from the statistics of the inputs such that the resulting power reduction is maximized. Once the values are determined, the guard latches are inserted between ($N^M=N_1^M+N_2^M$) th bit position and (N^M+1) th bit position from the MSB. The sign extension logic is implemented in the same way as ripple carry adder. We use two types of multiplier FA' and FA, where FA' is a slightly modified version of FA for correct computation of sign bit [15].

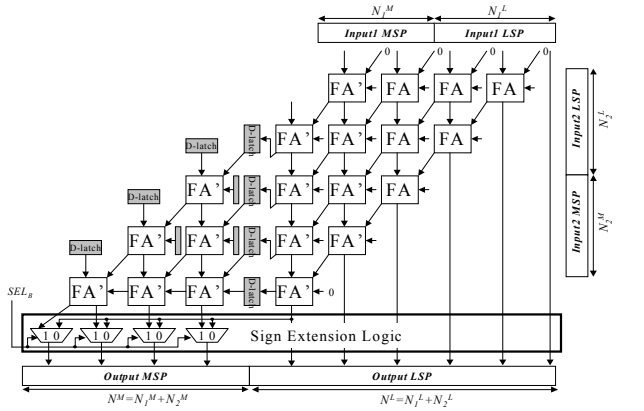


Fig. 4. Partially guarded circuitry for signed array multiplier.

Due to the overhead of the augmented circuitry, we do not expect much power reduction in the ripple carry adder. However, in case of the array multiplier, we can obtain much power reduction because it contains much more computing elements than the adder.

3. Boundary Positioning Algorithm

3.1 Problem Formulation

We denote the two inputs of a functional unit as in_1 and in_2, respectively and assume they have the same bit length which is denoted as N. The j th bit of in_i is denoted as in_i. The MSB and the LSB of in_i correspond to in_i^0 and in_i^{N-1}, respectively. We denote the location of the boundary between the MSP and LSP of input data in_i as D_i. D_i has a value between 0 and N-1. If $j < D_i$, in_i^j belongs to the MSP. Otherwise, it belongs to the LSP. Note that, in case of multiplier, there can be one or more combinations of D_1 and D_2 satisfying $D_1 + D_2 = N^M$. We formulate the boundary positioning problem as follows:

Given streams of input data to a functional unit, determine the set of boundary location D_1's such that power reduction by partially guarded computation is maximized.

We obtain the input data streams to the functional unit by performing behavioral simulation with the simulation vectors given by the user

3.2 Algorithm

In the case of multiplier, the bit length of output MSP (N^M) is computed as $D_1 + D_2$. Therefore, there can be more than one combinations of boundary points D_1 and D_2 for the same value of N^M. In this section, we consider the case where only single combination of D_1 and D_2 is selected. Extended problem for finding multiple combinations of D_1 and D_2 will be treated in the next section. We solve the single boundary positioning problem by selecting an optimal combination of D_1 and D_2 which gives the largest power reduction among all possible combinations. We denote the power cost per single operation of the MSP of a functional unit F_t and the power cost per single operation of detection logic as $P^1_{t,M}(D_1, D_2)$ and $P_{DET}(D_1, D_2)$, respectively. We denote total power reduction in the MSP of F_t as $P_{t,M}(D_1, D_2)$ which is computed by accumulating $P^1_{t,M}(D_1, D_2)$ whenever the MSP is disabled. The total power reduction in F_t, which is denoted as $P_t(D_1, D_2)$, is computed as

$$P_t(D_1, D_2) = P_{t,M}(D_1, D_2) - \beta \cdot I \cdot P_{DET}(D_1, D_2) \quad (1)$$

where β and I represent weighting factor and the size of input data stream, respectively. The value of β is determined experimentally. We compute $P_t(D_1, D_2)$ for all combinations of D_1 and D_2. In the case of adder, D_1 and D_2 must have the same value because the range of the output MSP is determined by the minimum of D_1 and D_2. Therefore, there are only N combinations. However, in the case of multiplier, D_1 and D_2 may have different values because the range of the output MSP is computed as the sum of D_1 and D_2. Therefore, there are $N \times N$ combinations of boundary points. We define the power cost of the MSP as weighted number of full adder cells in the part.

The power cost $P^M_{ADD}(D_1, D_2)$ of ripple carry adder is defined as

$$P^1_{ADD,M}(D_1, D_2) = \alpha_{ADD} \cdot D_1 = \alpha_{ADD} \cdot D_2 \quad (2)$$

where α_{ADD} is a weighting factor to reflect the effect of sign bit transition[2] whose value is obtained experimentally. The power cost of the detection logic is computed as

$$P_{DET}(D_1, D_2) = D_1 + D_2 \quad (3)$$

The power cost $P^1_{MULT,M}(D_1, D_2)$ of array multiplier is defined as

$$P^1_{MULT,M}(D_1, D_2) = \alpha_{MULT} \cdot (D_1+D_2-1) \cdot (D_1+D_2)/2 \quad \text{if } (D_1+D_2) \le N$$

$$P^1_{MULT,M}(D_1, D_2) = \alpha_{MULT} \cdot (N \cdot (N-1) - (2N-D_1-D_2) \cdot (2N-D_1-D_2-1)/2) \quad \text{otherwise} \quad (4)$$

```
1.  PositionBoundary
2.  INP: input data stream;
3.  begin
4.      Initialize P_t(D_1, D_2) and P_t,M(D_1, D_2) to 0;
5.      for each input data from INP loop
6.          for D_1 =0 to N-1 loop
7.              for D_2 =0 to N-1 loop
8.                  if both D_1 and D_2 are sign bits then
9.                      P_t(D_1, D_2) += P^I_t,M(D_1, D_2);
10.                 end if;
11.                 P_t(D_1, D_2)  -= β·P_DET(D_1, D_2);
12.             end for;
13.         end for;
14.     end for;
15.     Sel_D_1 = Sel_D_2 = 0;
16.     for D_1 =0 to N-1 loop
17.         for D_2 =0 to N-1 loop
18.             if P_t(D_1, D_2) > P_t(Sel_D_1, Sel_D_2) then
19.                 Sel_D_1 =D_1 ;        Sel_D_2 = D_2 ;
20.             end if;
21.         end for;
22.     end for;
23. end;
```

Fig. 5. Pseudo code of boundary positioning algorithm.

Figure 5 shows the pseudo code of our boundary positioning algorithm. In the first loop including the loops nested inside, we compute $P_t(D_1, D_2)$ for all combinations of D_1 and D_2. We accumulate $P^I_{t,M}(D_1, D_2)$ to $P_t(D_1, D_2)$ when both D_1 and D_2 belong to the sign extension regions of the current input data whereas $\beta \cdot P_{DET}(D_1, D_2)$ is subtracted from $P_t(D_1, D_2)$ for every input data. After we obtain $P_t(D_1, D_2)$ in the first loop, we select D_1 and D_2 which induce the largest value of $P_t(D_1, D_2)$ in the second loop. The complexity of the algorithm is $O(N^2 \cdot I)$, where I is the number data in the input data streams.

3.3 Extension to Multiple Boundary Combinations

If we check multiple combinations of boundary points, we can increase the duration in which MSP of the multiplier is disabled. Though checking multiple boundary points helps reducing MSP power, it increases detection logic overhead. However, overhead in detection logic is not linearly proportional to the number of boundary points because there are lots of common components among detection logics with different boundary points.

The problem of determining multiple boundary points is formulated as selecting n_D combinations of $D_{1,i}$ and $D_{2,i}$ such that total power reduction is maximized and $D_{1,i}+D_{2,i}=N^M$ for i=0, ... , n_D-1. We adopt the same scheme for positioning multiple boundary points as proposed in the previous section. We exhaustively evaluate all the candidate solutions using the cost function $P_{MULT}(D_{1,i}, D_{2,i})$ and select the best one. If we assume n_D is given by user, the complexity of the multiple boundary positioning algorithm can be computed as $M \cdot N \cdot \binom{N}{n_D}$, where the worst case complexity is $O(M \cdot N^{I+N/2})$ which is almost intractable. However, from the experimental results, we find that n_D=3 is sufficient in most cases because we cannot obtain significant power improvement even if we increase the value of n_D larger than 3. The graph in Fig. 6 shows the effect of the number of boundary points on power improvement. Note that the curves begin to saturate from the point where n_D =3.

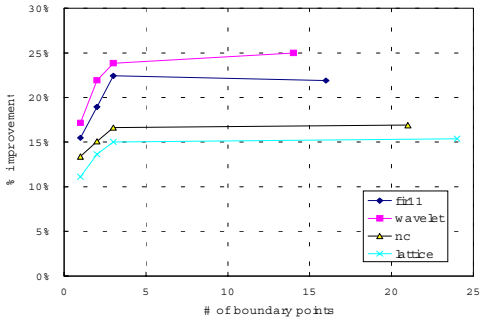


Fig. 6. Effect of multiple boundary points on power improvement.

4. Operation Binding Algorithm

Since partially guarded computation technique reduces power consumption based on dynamic range of the input data to a functional unit, obtainable power improvement strongly depends on the status of operation and operand binding. In general, increased input correlation reduces power consumption in a functional unit. However, it does not always hold when the functional unit supports partially guarded computation. For example, let us compare two input data sequences: sequence A (111101 → 101110) and sequence B (111101 → 000010). From the viewpoint of input data correlation, sequence A is superior to B. However, if we use partially guarded computation technique, sequence B can result in lower power design than sequence A. Note that the boundary point of sequence A is located at the 4th bit from the MSB, whereas that of sequence B is located at the 1st bit. Thus, we need to devise a new operation and operand binding algorithm that can consider tradeoff between data correlation and dynamic range.

For this purpose, we propose a greedy algorithm that incrementally binds an operation to a functional unit while evaluating the power cost of the current design. After all the operations are scheduled, we perform operation binding such that power cost is minimized. We compute the power cost by multiplying power reduction ratio to the estimated power of the current design obtained using DBT model [16]. The power reduction ratio can be computed using equation (1) ~ (4). We perform operation binding for each control step starting from initial control step. For each control step, we compute power costs of all the feasible bindings for each operation scheduled at current control step. Among the feasible bindings we select the one with minimum power cost. We repeat above procedure until all the

[2] We give more weight when there is signal transition in sign bit.

operations at current control step are bound. When we compute power costs of all the feasible bindings, we consider swapping of input operands and select a solution with minimum cost.

5. Experimental Results

For automatic generation of the layout, we designed partially guarded circuitry for Lager IV layout synthesis tool [12]. We implemented our boundary positioning algorithm and operation binding algorithm using C++ under UNIX environment. To measure power consumption, we developed a power estimation tool based on DBT model. We verified the reliability of the estimator by comparing the estimated power with the power obtained by IRSIM [13], a switch level simulator, running on the simulation file extracted from the layout generated by Lager IV. We used 1.2 micron technology for the generation of layouts. We tested our algorithm using well know data dominated circuit examples from HYPER [14]. We first performed behavioral simulation by using simulation vectors given by user. We used speech data, music data, and random data as the simulation vectors. After we performed scheduling and binding, we determined two parts – MSP and LSP – by using the proposed algorithm. We set the parameter value β to 0.4 for computing power cost. In our experiment, we applied the partially guarded computation technique only to multipliers, because it takes more than 50% of the total power consumption in most cases. Due to very long IRSIM simulation time, we do not simulate the complete circuit. Instead, we picked up functional units and related units such as register, detection logic, and sign extension logic from the complete circuit, and performed simulation for the selected sub-circuit using the already obtained input data stream.

Table 1. Comparison of layout area for multiplier

	Resource	*N*	w/o PGC		w/ PGC		Overhead	
			Area (mm^2)	Delay (ns)	Area (mm^2)	Delay (ns)	Area (%)	Delay (%)
fir11	+1 *3	16	6.58	94.3	8.95	96.6	**36**	**2.4**
	+10, *11	16	6.58	94.3	8.95	96.6	**36**	**2.4**
wavelet	+2 *4	16	6.58	80.6	8.95	82.8	**36**	**2.7**
nc	+2 *4	25	15.65	140.8	20.35	143.2	**30**	**1.7**

Table 2. Comparison of estimator with IRSIM.

		Estimator			IRSIM			error in red. (%)
	Resource	w/o PGC (nJ)	w/ PGC (nJ)	red. (%)	w/o PGC (nJ)	w/ PGC (nJ)	red (%).	
fir11	+1 *3	58.49	49.28	**15.7**	62.97	53.22	**15.5**	***-1.1***
	+10 *11	29.02	20.80	**28.3**	26.09	18.72	**28.3**	***0***
wavelet	+2 *4	91.62	76.19	**16.9**	93.77	77.69	**17.2**	***+1.8***
nc	+2 *4	280.4	251.2	**10.4**	270.0	240.1	**11.1**	***+6.3***

Table 1 shows the layout area and delay information of the sub-circuit for each synthesis example. In the table, PGC stands for our Partially Guarded Computation technique. The layout area overhead by partially guarded circuitry is about 30-36%. The area overhead is mainly caused by two dimensional tiling structure of array multiplier supported by LAGER IV. Such generic structure is assumed for the automatic generation of layout. When we enlarged several leaf cells for the modification of the multiplier, we had to enlarge other cells because they must have the same height and width. If we carefully restructure the multiplier such that the modified leaf cells do not affect the size of other cells, we can further reduce the overhead caused by the partially guarded computation circuitry. Moreover, such area overhead in functional units takes relatively small portion as the complexity of the design increases. The worst case delay overhead by our additional circuitry is below 3%. The overhead is caused by additional input latches and sign extension logic.

Table 2 compares the results by our power estimator with those by IRSIM. We performed simulation using 128 samples of speech data. The table shows our estimator is reliable in evaluating the effect of PGC because it has up to 10% error in estimating power and up to 6.3% error in estimating improvement by PGC.

Table 3 summarizes power consumption and resource allocation information for the examples. The sixth column in the table indicates the results when we applied multiple boundary points (n_D=2) and the proposed operation binding algorithm. The value within parentheses represents the power consumption when multiple boundary points are selected. Before we applied our technique, we minimized power consumption in functional units by using the power conscious scheduling and binding algorithm proposed in [8]. For fair comparison, we implemented a multiplier such that it does not produce output bits which will not be used. For example, if we use only $N' < 2N$ output bits among $2N$ output bits of a $N \times N$ multiplier, the cells for computing upper $2N - N'$ bits are replaced by NOP cells which do not consume any power at all. Moreover, we set the value of N as the maximum bit length of input operands without allowing any margin to the design.

As simulation vectors, we used three different sets of data: speech data with large dynamic range (*speech1*), speech data with small dynamic range (*speech2*), and normally distributed random data (*normrand*). Each set of data is composed of 128 samples. The maximum bit length of *speech1* and *normrand* is 14, whereas that of *speech2* is 12. We first determined the boundary location by using only the *speech1* data set. Then we apply other data sets such as *speech2* and *normrand*, which have different characteristics, to the examples in order to show that our technique is still effective when the characteristics of input data is changed. The table shows that we can still reduce power consumption under different input data set.

For the *fir11* (11^{th} order FIR filter) example, we measured power consumption for two different designs: shared design and fully parallel (or unshared) design. The shared design uses three multipliers, where they are shared among operations in different control steps. The fully parallel design uses eleven multipliers which are not shared at all. For the shared design, we reduced power consumption about 31 to 37%, whereas we can reduce power consumption by 36 to 44% for the fully parallel design. In general, once we have a fully parallel design, we cannot obtain further power reduction by allocating more functional units or changing input data correlation [9]. However, our technique can still reduce power consumption for the fully parallel design. Moreover, our technique can reduce substantial power

consumption even though we already minimized power using the method in [8]. From the results for *speech1* and *speech2* on the designs with same word length, we expect about 5~10% more power reduction if we increase N by one or two in order to give margin to the design. Note that the dynamic range of *speech2* is two bits less than that of *speech1*. In our experiment, we optimized word length such that the dynamic range of the input exactly matches that of the word length. In practical situation, such case rarely occurs and most of data can be represented by much smaller number of bits. Therefore, we expect much more power reduction in practical applications.

Table 3. Comparison of power consumption in multipliers

	Resource	Input Data Set	w/o PGC	w/ PGC	w/ BIND & PGC (n_D=2)	Reduction
			nJ	*nJ*	*nJ*	%
fir11	+1 *3 (shared)	*speech1*	54.2	44.6	39.1 (37.4)	**31.1**
		speech2	55.0	43.0	37.7 (34.6)	**37.1**
		normrand	54.7	43.4	37.4 (37.4)	**31.8**
	+10, *11 (fully parallel)	*speech1*	26.1	16.8	N/A	**35.5**
		speech2	26.4	14.8	N/A	**43.9**
		normrand	25.8	15.3	N/A	**40.5**
Wavelet	+2 *4	*speech1*	82.8	76.2	75.6 (71.5)	**13.6**
		speech2	83.4	69.9	67.0 (55.9)	**20.4**
		normrand	83.7	77.5	74.5 (69.1)	**17.4**
nc	+2 *4	*speech1*	255.3	233.6	233.6 (230.2)	**9.8**
		speech2	263.7	229.1	229.1 (235.1)	**13.1**
lattice	+1 *2	*speech1*	46.4	39.5	39.5 (37.9)	**18.3**
		speech2	38.1	31.0	31.0 (24.0)	**34.0**
		normrand	47.7	42.4	42.4 (40.7)	**11.6**
Average						**23.87**

6. Conclusion

In this paper, we proposed a partially guarded computation (PGC) technique which disables a part of a functional unit according to the dynamic range of input data. The technique first divides the input data into two parts – MSP and LSP – and accordingly divides the functional unit into two parts. We allow only computation of LSP of the functional unit when the range of any input data does not exceed the maximum range of the LSP. We also propose an algorithm which systematically determines the boundary between the two parts and an effective operation binding algorithm for maximizing power reduction effect by PGC. By using the proposed technique we reduced power consumption in an array multiplier by about 10 to 44%. Our method can effectively reduce power consumption even after we minimize power by using high-level power minimization technique

7. References

[1] A. P. Chandrakasan, S. Sheng, and R. Brodersen, "Low Power CMOS Digital Design," IEEE Trans. on Solid-State Circuits., vol. 27, No. 4, April, pp. 473-483, 1992.

[2] C. Tsui, M. Pedram, and A. Despain, "Technology Decomposition and Mapping Targeting Low Power Dissipation," Proceedings of Design Automation Conference, pp. 68-73, 1993.

[3] A. Raghunathan and N. K. Jha, "Behavioral synthesis for low power," Proceedings of International Conference on Computer Design, pp. 318-322, Oct. 1994.

[4] A. Raghunathan, S. Dey, N. K. Jha, "Controller re-specification to minimize switching activity in controller/data path circuits," Proceedings of International Symposium on Low Power Electronics and Design, pp. 301-304, Aug. 1996.

[5] E. Musoll and J. Cortadella, "High-level synthesis techniques for reducing the activity of functional units," Proceedings of International Symposium on Low Power Design, pp. 99-104, Nov. 1995.

[6] L. Benini, P. Vuillod, G. D. Micheli, and C. Coelho, "Synthesis of low power selectively-clock systems from high-level specification," Proceedings of International Symposium on System Synthesis, pp. 57-63, Nov. 1996.

[7] D. Kim and K. Choi, "Power conscious high level synthesis using loop folding," Proceedings of Design Automation Conference, pp. 441-445, 1997.

[8] D. Shin and K. Choi, "Lower power high level synthesis by increasing data correlation," Proceedings of International Symposium on Low Power Electronics and Design, Aug. pp. 441- 445, Aug. 1997.

[9] R. Mehra, L. M. Guerra, and J. Rabaey, "Low-power architectural synthesis and the impact of exploiting locality," Journal of VLSI Signal Processing, 1996.

[10] A. Dasgupta and R. Karri, "Simultaneous scheduling and binding for power minimization during microarchitecture synthesis," Proceedings of International Symposium on Low Power Design, 1995.

[11] V. Tiwari, S. Malik, and P. Ashar, "Guarded Evaluation: Pushing Power Management to Logic Synthesis/Design," IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems, vol. 17, no. 10, pp.1051-1060, Oct. 1998.

[12] Brodersen, et al, "An Integrated CAD System for Algorithm-Specific IC Design," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 10, No. 4, pp. 447-463, April 1991.

[13] A. Salz and M. Horowitz, "IRSIM: An Incremental MOS Switch-Level Simulator," Proceedings of Design Automation Conference, pp. 173-178, 1989.

[14] J. Rabaey, C. Chu, P. Hoang, and M. Potkonjak, "Fast prototyping of datapath-intensive architectures," IEEE Design and Test of Computers, pp. 40-51, June 1991.

[15] I. Koren, Computer Arithmetic Algorithms, Prentice-Hall International, p. 116, 1993.

[16] P. Landman and J. M. Rabaey, "Black-box capacitance models for architectural power analysis," Proceedings of International Symposium on Low Power Degisn, pp. 165-170, Apr. 1994.