Low-Power Process-Variation Tolerant Arithmetic Units Using Input-Based Elastic Clocking

Debabrata Mohapatra, Georgios Karakonstantis and Kaushik Roy
ECE School, Purdue University
1285 EE Building, 465 Northwestern Avenue
West Lafayette, IN 47907, USA
+1-765-49{43372, 43372, 42361}
{dmohapat, gkarakon}@purdue.edu and kaushik@ecn.purdue.edu

ABSTRACT

In this paper we propose a design methodology for low-power, high-performance, process-variation tolerant architecture for arithmetic units. The novelty of our approach lies in the fact that possible delay failures due to process variations and/or voltage scaling are predicted in advance and addressed by employing an elastic clocking technique. The prediction mechanism exploits the dependence of delay of arithmetic units upon input data patterns and identifies specific inputs that activate the critical path. Under iso-yield conditions, the proposed design operates at a lower scaled down Vdd without any performance degradation, while it ensures a superlative yield under a design style employing nominal supply and transistor threshold voltage. Simulation results show power savings of upto 29%, energy per computation savings of upto 25.5% and yield enhancement of upto 11.1% compared to the conventional adders and multipliers implemented in the 70nm BPTM technology. We incorporated the proposed modules in the execution unit of a five stage DLX pipeline to measure performance using SPEC2000 benchmarks [9]. Maximum area and throughput penalty obtained were 10% and 3% respectively.

Categories and Subject Descriptors

B.7.1 [Integrated Circuits]: Types and Design Styles – *VLSI (very large scale integration)*

General Terms

Design, Reliability

Keywords

Low power, process tolerant, elastic clocking.

1. INTRODUCTION

As CMOS technology continues to scale aggressively into the sub-100nm regime, design in the presence of unreliable components has become exceedingly challenging. Due to increased levels of process variation in scaled technologies, delay failures are becoming increasingly frequent upon supply voltage scaling [1]. Combinational logic blocks which are designed to meet the target frequency of operation fail, because the critical path delay of the circuit exceeds the nominal delay target. Achieving lower power dissipation merely by Vdd scaling is no longer easy to achieve under severe parameter variations. Hence there is a growing need for designing not only low power but also robust, delay failure resilient logic units.

Though extensive research has been done in the domain of low power implementation of arithmetic units [2, 3], yet there is a growing need to address the issue of low power and process tolerant design jointly. Traditional approaches such as [4] suggest operation at a conservative target frequency or scaling up of Vdd in order to meet the target yield. The concept of using variable latency units to increase the average throughput of combinational logic blocks has been proposed in [5]. However, the primary application of [5] is an improvement in performance and it does not deal with low power and process tolerant design issues. Recent research such as [6] has focused on issues related to low power and process tolerance jointly, from a CAD perspective, by partitioning random logic. However to the best of our knowledge, no significant research has been conducted that concurrently targets these two issues for arithmetic units.

In this work, the novelty of our proposed architecture lies in the achievement of robustness in design, while operating in reduced energy per computation (EPC) mode. The central idea of the paper is that, under voltage scaling, depending upon input data patterns, we adaptively change the number of clock cycles required for computation. Hereafter, we call this feature of our design as *elastic clocking*. We have observed that a) certain input patterns take more time to be computed than others and b) typically the probability of occurrence of such patterns is rare. We take advantage of this fact by predicting in advance the input patterns that will activate the critical path as explained in later sections. By a novel architectural method explained in the next section we make the activation probability of long latency paths small. This enables us to operate at a fixed lower scaled down supply voltage resulting in EPC improvement at the same frequency of operation. The proposed technique has the following advantages: a) *reduced energy per computation* (by supply voltage scaling) under iso-yield conditions, while incurring negligible area overhead and throughput penalty, b) *higher yield* (due to the reclamation of chips which earlier failed to meet the target delay in the conventional case), and c) *same frequency of operation* (instead of operating at a lower frequency to attain a higher yield).

The rest of the paper is organized as follows. Section 2 describes the generic architectural framework for the proposed design. Section 3 gives a detailed description of the implementation of our scheme considering different adders and multipliers. Simulation results are given in section 4 while section 5 concludes the paper.

ISLPED'07, August 27–29, 2007, Portland, Oregon, USA.

2. PRELIMINARY ANALYSIS

2.1 The Proposed Architectural Framework

Let us consider a combinational logic with *n* inputs X_1, X_2, $X_3 \ldots X_n$. We choose a function ***f*** which partitions the input set **V** into two subsets $\mathbf{V_1}$ and $\mathbf{V_2}$. The input set **V** is a collection of *n* dimension vectors, each of which is a collection of *n* possible inputs to the logic block. The subset $\mathbf{V_1}$ consists of input vectors that activate long latency paths, while those in $\mathbf{V_2}$ activate short latency paths. This separation of the set of input vectors **V** is based on the idea that delay of a combinational logic unit is dependent upon input patterns. Let us consider a simple example of addition of two *n* bit numbers to illustrate the concept mentioned above. Let A=0000……0001 and B=1111……1111, then the carry generated in the first bit position is propagated all the way to the final bit position. Thus, there is a set of inputs that activate the worst case delay of the adder block. Let us modify the inputs to A=1111....0….1111 and B=0000……0001, where the chain of 1's in A is broken by insertion of a 0 in the $(n/2)^{th}$ bit position. For this case there is no carry propagation across the middle bit and the effective computation time is maximum of the two delays, one from the 0 to $(n/2)^{th}$ bit and the other from $(n/2)^{th}$ to *n*-1 bit. Let us define a partition function as ***f***, which evaluates a value of 1 if the inputs give rise to a long latency operation and 0 otherwise. For short latency operation, the supply voltage can be low while for longer latency operation, clock can be stretched at the same lower supply voltage. However, one has to ensure that the probability of short latency operation is large so that the performance penalty is low. Let:

$$f(X) = \begin{cases} 1, & X \in V_1 \\ 0, & X \in V_2 \end{cases} \qquad (1)$$

If ***P(S)*** is the probability of activation of short latency and ***P(L)*** is the probability of activation of long latency paths, then we would like to minimize ***P(L)*** for increasing power savings with minimum performance penalty as explained later. This can be achieved by a judicious choice of the partitioning function ***f***. However, we note that the complexity of ***f*** grows exponentially as the number of inputs increases. Thus, our objective is not only to choose a function ***f*** that minimizes ***P***(***L***) but also minimizes complexity (for minimizing area and power overheads). To concurrently minimize complexity and ***P***(***L***), we choose a partitioning function ***f'*** of reduced complexity. The number of inputs to ***f'*** is *k* where *k*≤*n*. Let **Z=*f'* (X')** where **X'** = (X_i', X_{i+1}', X_{i+2}' ……. X_{i+k-1}') is the vector of *k* inputs chosen out of *n* inputs.

$$f'(X') = \begin{cases} 1, & X' \in V_1 \\ 0, & X' \in V_2 \end{cases} \qquad (2)$$

The relationship between ***f'*** (**X'**) and ***f***(**X**) is that the event {***f***(**X**) = 1} implies {***f'***(**X'**) = 1} but the converse may not be true. Since ***f'*** is expanded form of ***f, P(L)*** using ***f'*** is smaller than the ***P(L)*** using ***f***. However, we must ensure that ***P (L)*** using ***f'*** is sufficiently close to 0 so that throughput penalty due to elastic clocking is not excessive.

There are two issues regarding the choice of a partitioning function: a) choice of ***f'*** that is of very low-complexity while achieving *P(L)* sufficiently close to 0, and b) selection of *k* inputs from the set of *n* inputs to the logic block. Choice of a reasonable ***f'*** is a non trivial task for complex circuits. However, for simple arithmetic units like adders and multipliers we choose ***f'*** to be a function of the propagate signal of a full adder block. It is because the propagate signal determines if the input carry

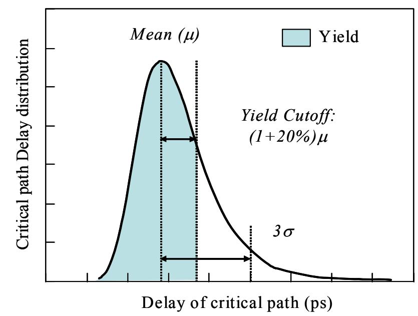

Figure 1. Critical path delay distribution under process variation

will be propagated from, $(n/2)^{th}$ bit to the n^{th} bit or not. This fundamentally determines whether the blocks before and after the $(n/2)^{th}$ block can be computed in parallel. Returning to the other issue related to ***f',*** the two criteria for selection of the *k* inputs are: 1) splitting of the critical path should result in two shorter paths which would enable us to take greater advantage of Vdd scaling (due to the large slack between the critical and maximum of the two shorter paths), and 2) ***P(L)*** should be sufficiently close to 0 so that the penalty due to elastic clocking is minimized. Once we have chosen ***f'*** and **X'**, we can predict in advance the nature of latency. Upon detection of a long latency operation (which is the uncommon case due to proper choice of ***f'***), *the clock is stretched to allow two cycles for the operation to complete*. During short latency operation we evaluate the computation in one cycle. In this manner, the design achieves robustness to failures due to process variations.

2.2 Relative Comparison Using a Yield Model

In this work, we have modeled the process variation (L, T_{ox}, W etc.), as a lumped variation in the threshold voltage. We define the delay target of a design to be 1.20T (say) where T is the nominal path delay (shown in figure 1). A conventional design will lead to failure of circuits which have path delay greater than 1.20 times the nominal path delay. Reclamation of chips that fall beyond the yield cutoff is possible in our design. This is because the critical path will be activated for certain input patterns which we predict in advance and evaluate in 2 clock cycles to ensure no delay failures (i.e. have yield approximately equal to 100%). It should be noted that conventional techniques for achieving high yield such as scaling up of supply voltage or aggressive sizing up of transistors typically increase the power consumption. However, in our proposed approach, besides superior yield we gain improvement in EPC, while incurring a negligible degradation in performance. Let us define two terms T_{short} and T_{long} which are the short and long latency delays respectively. Since there is a slack between T_{short} and T_{clk}, we scale down Vdd to attain energy per computation savings. However scaling of supply voltage should take care of two facts: a) short paths maintain a specified yield under one clock cycle operation ($T_{short} < T_{clk}$), and b) long paths maintain 100% yield operating in a two cycle mode ($T_{long} < 2T_{clk}$). Assuming the probability of occurrence of long latency operations to be ***p***, the scaled down supply voltage to be *Vddlow* and nominal supply voltage Vdd, e*quation 3* gives the EPC ratio of our proposed approach over conventional schemes. The factor of 2 in *equation 3* is due to the fact that in our elastic clocking scheme we allow two clock cycles for a long latency operation. It shows that we can improve EPC only when *p* is close to 0.

$$\frac{E_{proposed}}{E_{conventional}} = \frac{(1-p)*Vddlow^2 + p*Vddlow^2*2}{Vdd^2} \qquad (3)$$

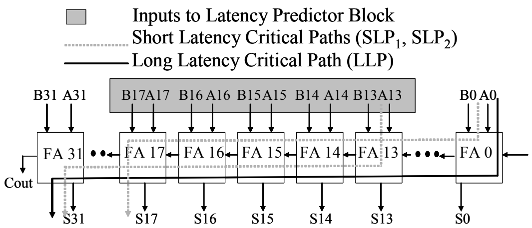

Figure 2. Splitting of critical path using partitioning function

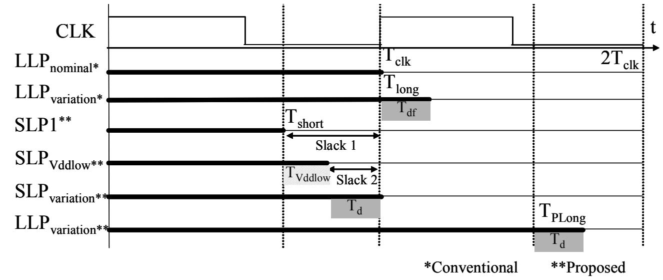

Figure 3. Timing diagram

3. DESIGN METHODOLOGY

3.1 Analysis & Case Study

Dependency of subsequent stages upon previous stages is a major bottleneck in the performance of any system. While considering arithmetic units such as adders and multipliers this bottleneck manifests itself in the form of carry propagation. Subsequent stages have to wait for the correct carry, before they can generate a valid output. The universal technique employed for isolating any kind of dependency is via prediction. By predicting whether there will be carry propagation across blocks or not, the two paths can be computed in parallel. In order to illustrate our idea, we describe a 32 bit ripple carry adder (RCA). We choose a partitioning function ***f'*** given by *equation 4*, where ⊕ denotes bitwise XOR operation and **X'** given by the set $\{A_{15}, B_{15}\}$.

$$f'(X') = (A_{15} \oplus B_{15}) \quad (4)$$

The choice of ***f'*** and **X'** is determined by the following two criteria: 1) ***f'*** should be a function of low complexity logic that partitions the input set **V** into long and short latency sets $\mathbf{V_1}$ and $\mathbf{V_2}$ respectively. 2) **X'** should be such that: a) splitting of the critical path should result in 2 shorter paths enabling us to take maximum advantage of Vdd scaling and b) ***P(L)*** should be sufficiently close to 0 so that the penalty due to elastic clocking is minimized. The probability of ***P(L)*** is the probability of the event {***f'***(**X'**) =1}. From *equation 4* we get:

$$P(L) = P(\{f'(X') = 1\}) = 2\,(1-p')\,p' \quad (5)$$

The signal probability of input is denoted by ***p'***. Assuming signal probability to be 0.5 we have ***P(L)*** = 0.5 which implies a 50% probability of a long latency operation. It does not satisfy the second criteria which states that ***P(L)*** should be close to 0. We circumvent this problem by choosing the set **X'** to include more inputs and modify the partitioning function ***f'***. The new **X'** is given by the set $\{A_{13}, B_{13}, A_{14}, B_{14}, A_{15}, B_{15}, A_{16}, B_{16}, A_{17}, B_{17}\}$ and ***f'*** given by *equation 6* where • denotes bitwise AND.

$$f'(X')=(A_{17}\oplus B_{17})\bullet(A_{16}\oplus B_{16})\bullet(A_{15}\oplus B_{15})\bullet(A_{14}\oplus B_{14})\bullet(A_{13}\oplus B_{13}) \quad (6)$$

With the modified ***f'*** and **X'**, ***P(L)*** is now equivalent to the probability of carry propagation across the 17^{th} bit of the adder as shown in figure 2. For this particular choice of ***f'*** and **X'**, we define an add operation to be long latency type only when there is carry propagation across the 17^{th} bit of the adder. In order to determine the latency type of the add operation, we consider five full adder blocks FA13 to FA17. If the output carry of FA17 depends on the input carry of FA13 then we classify the operation as long latency type, with probability ***P(L)***. This is because the probability of carry propagation across one bit adder block is given by $2\,(1-p')\,p'$ and from the independence assumption of inputs, carry propagation probability across five one bit adder blocks is given by $[2\,(1-p')\,p']^5$. Hence ***P(L)*** for ***p'*** equal to 0.5 is 0.0312. Since the value of ***P(L)*** is small (approximately 3%), the chosen ***f'*** and **X'** are acceptable. Greater is the number of inputs (k) to the function f', smaller is the probability of long latency operation. Table 1 gives the probability of long latency operation as a function of number of inputs in set **X'** used in the latency prediction process. The design tradeoff between the number of bits used as input in the latency prediction process and area and power overhead due ***f'*** implementation is discussed in section 4.

Table 1. Short Latency probability

X'	{A[16:14] B[16:14]}	{A[17:14] B[17:14]}	{A[17:13] B[17:13]}	{A[18:13] B[18:13]}
P (L)	0.1250	0.0625	0.0312	0.0156

In figure 2, we observe that the longest critical path is split into two short latency paths. Let us denote the delay of carry propagation path from FA0 to FA17 as SLP_1 (short latency path 1) and that of path from FA13 to FA31 as SLP_2. The worst case carry propagation path is from FA0 to FA31 and is denoted by LLP (long latency path). We define the time taken for a short latency operation T_{short} to be the maximum of the two delays SLP_1 and SLP_2. Under process variations, a conventional design may have LLP delay exceeding T_{clk} by T_{df} which will lead to a delay failure. The relationship between SLP_1, SLP_2 and T_{clk} is given by *equation 7.* The timing diagram, assuming that T_{short} is equal to SLP_1 is given in figure 3.

$$T_{short} = \max(SLP_1, SLP_2) < T_{long} < 2T_{clk} \quad (7)$$

Due to the slack available between T_{short} and T_{clk} (slack 1) as shown in figure 3, we can scale down Vdd to *Vddlow* such that T_{short} increases by an amount T_{Vddlow}. This enables us to achieve lower power (reflected as lower EPC). Note that we allow timing slack to be present (slack 2, figure 3) between T_{short} and T_{clk} to make sure that the short latency paths do not exceed the one cycle bound under variations. The LLP delay considering Vdd scaling is T_{Plong} while T_d is the increment in path delay due to process variations. The LLP delay shown in figure 3 under variations is within the two clock cycle bound given by *equation 7*. The presence of timing slack in T_{long} ensures that there is no delay failure under process variations.

3.2 Adder Architecture

In this section we apply our design methodology to a 32 bit modified cascaded carry select adder (CCSA) besides the 32 bit RCA, mentioned in section 3.1. For any adder architecture we consider, our primary goal is to be able to split the critical path into shorter paths and then use the prediction mechanism on input data pattern to perform elastic clocking. We do not use the conventional square root adder because each subsequent stage has more number of bits than the previous stage, due to which the resulting short latency path upon splitting, would be comparable to the critical path delay. Hence, we use a cascaded form of carry select adder proposed in [2]. For example a 32 bit conventional square root adder implementation will have {2, 2, 3, 3, 4, 5, 6, 7} bits in stage 1 to 8 respectively (each stage implemented as RCA). Even if we split the critical path based on carry propagation across the 4 bits in stage 5, the resulting short

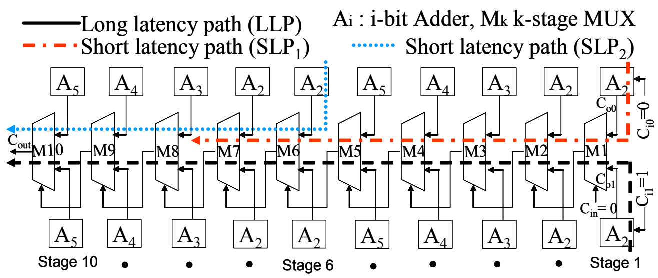

Figure 4. 32 bit cascaded carry select adder

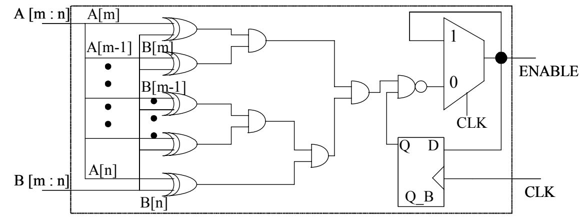

Figure 6. Latency predictor block implementation

latency path may still be comparable to the critical path due to the presence of higher number of bits in the subsequent stages. The propagation of carry across the 5 bits in stage 6, followed by carry propagation through the multiplexers in stage 6 and 7 upto the sum generation block, might be of the same delay as the critical path. Hence, we need to have a modified adder structure where the critical path after being partitioned gives rise to shorter paths. This is achieved by breaking the 32 bit adder into two 16 bit adders (shown in figure 4), each of which is implemented as a square root adder. The 32 bit CCSA implementation will have {2, 2, 3, 4, 5, 2, 2, 3, 4, 5} bits in stage 1 through 10, respectively. By predicting, whether there is carry propagation across the adders in stages 6 and 7, we can split the carry propagation path from stage 1 to stage 10 into two short latency paths: one from stage 1 to stage 7 and the other from stage 6 to stage 10. Figure 5 shows the general architecture of an adder (with single critical path) in the purview of our elastic clocking scheme. The latency predictor block (LPB) is the hardware implementation of the partitioning function which predicts the activation of critical path. We choose certain input operand bits, given by **X'**, as inputs to the LPB which determines the nature of operational latency and accordingly generates an *enable* signal. An *enable* signal of value 1 implies a short latency operation, which allows the output to be written into the output register in the next clock cycle. Otherwise it implies a long latency operation which stretches the clock for an extra period. The result of addition is written into the output register after one clock cycle delay.

Figure 6 shows the implementation details of the LPB which implements the function ***f'***(**X'**) given by *equation 6*. The D flip flop shown is necessary because the LPB needs to remember the nature of input latency in the previous clock cycle in order to generate the correct value of *enable* in the current cycle. One of the aspects of the LPB that needs special mention is the use of a negative D latch along with a positive edge triggered D flip flop. The circuit (shown in figure 6) allows computation of the enable signal before the next set of inputs arrive (at the next rising edge of the clock), an aspect which is extremely critical for the success of our design methodology. Its importance will be explained in greater detail in section 3.3. The value of *enable* latched in the negative clock cycle is used to

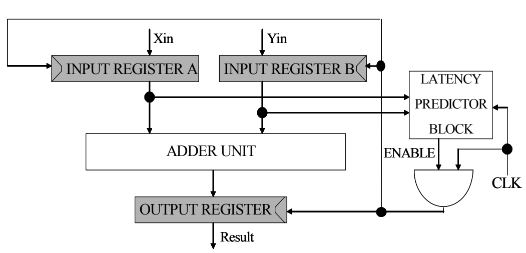

Figure 5. Generic adder architecture with elastic clocking

determine whether the output register will be written at the next rising edge of the clock or be delayed by one clock cycle to implement elastic clocking. Disabling of the write operation to the input and output registers is achieved by clock gating as shown in figure 5. While considering the predictor block we make the important assumption that the latency detection circuitry is designed to be process tolerant by proper sizing of the transistors.

3.3 Multiplier Architecture

We have also applied our technique to two classes of multipliers namely the carry save (CSM) and the Wallace tree multipliers (WTM). In figure 7, we show an *N* x *N* CSM and its critical paths. The methodology involved in splitting of the critical path in case of the multipliers is slightly different than that of adders. For the adders considered in section 3.2, the primary input bits in the set **X'** were the inputs to the LPB. However, finding a low complexity ***f'*** and set **X'** (consisting of *k* out of *n* primary inputs), which can predict in advance the long or short latency operation of the multiplier, is extremely difficult due to the large overhead associated with hardware implementation of ***f'***. Hence, we relax the constraint that **X'** should be a set of *k* inputs chosen from the primary inputs vector set. Instead, we allow **X'** to be the intermediate stage outputs of the multiplier. In the two types of multipliers that we considered, CSM and WTM, the final stage of multiplication consists of a vector merging adder (VMA). In both the multiplier architectures, we consider that the VMA is implemented as a CCSA. *We split the critical path in the multiplier by using the internal bits, which are inputs to the VMA*. The resulting short latency paths (SLP_1, SLP_2) are shown in figure 7.

Multipliers have many paths of similar delays and under variations any one of these paths may become critical. Hence we make an assumption that all potential critical paths in presence of variability have in common the carry propagation path of the VMA. The critical path delay is equal to the sum of path delays through the non-VMA part of the circuit and the delay through the VMA. Hence, by breaking the critical path in the adder stage, we take care of paths that have probability of becoming critical under variations. Since we use VMA inputs as our inputs

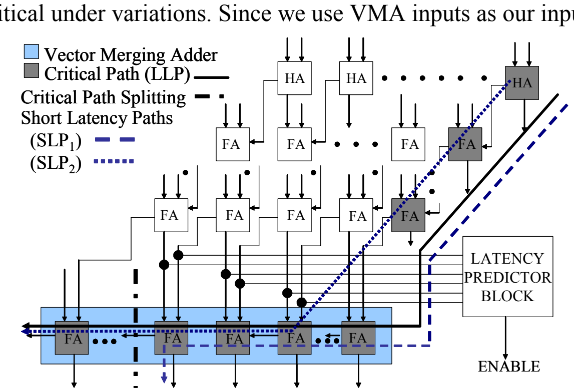

Figure 7. Critical path in a *N* x *N CSM* multiplier

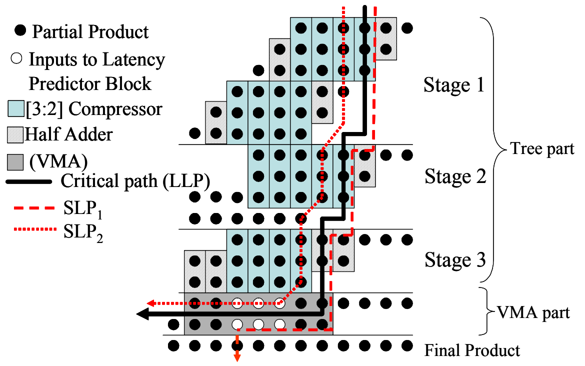

Figure 8. Critical path in a *6 x 6* Wallace tree multiplier

to the LPB, there is a probability that sufficient time may not be available for the *enable* to be computed by the time the falling edge of the clock arrives. We circumvent this problem by using a negative D latch along with the positive edge triggered D flip flop (instead of a negative D flip-flop) as shown in figure 6.

Our technique is also applied to WTM, shown in figure 8, which consists of a tree part and a VMA part. The tree part consists of stages, each of which contributes an adder delay to the critical path. The critical path delay is the sum of the number of stages in the tree and the delay through the VMA. For instance, in a 16x16 WTM (using 3:2 compressors), there are 6 stages in the tree part and a 27 bit VMA. Hence, the critical path delay is the sum of 6 adder delay and the worst case carry propagation delay of VMA. We note that there is a great potential to be exploited in case of WTM because of the relatively large size of the VMA. This gives us greater scope for Vdd scaling, resulting in power savings. In figure 8, a *6 x 6* WTM, showing the tree part and the VMA part is illustrated. In a manner similar to the splitting of critical paths in CSM, the critical path in WTM is cut into SLP_1 and SLP_2 as shown in figure 8. The slack between LLP and the maximum of SLP_1 and SLP_2 can be used for Vdd scaling as explained in section 3.1.

4. SIMULATION RESULTS

In this section we compare adders (12, 16, 32 bit RCA and CCSA) and multipliers (12, 16 bit CSM and 8, 12, 16 bit WTM) implemented in the conventional and our proposed design. All the arithmetic units mentioned above were implemented in 70nm BPTM technology [7]. The metrics used for comparison were parametric yield improvement, power dissipation, EPC, area overhead and throughput penalty. We used VHDL to design the adders and multipliers. The VHDL code was synthesized using Synopsis Design Compiler [8]. In order to obtain the parametric yield in presence of process variations, we ran Monte Carlo simulations in Hspice, assuming a Gaussian V_{th} variation distribution of zero mean and standard deviation of 40 mV. The power dissipation results were obtained by simulating 10000 random input vectors in NanoSim.

Table 2. Yield % of different arithmetic units at nominal Vdd (1V)

Arithmetic Units	RCA (32bits)	CCSA (32bits)	WTM (16bits)	CSM (16bits)
Conventional	92 %	90 %	96 %	93 %
Proposed	100 %	100 %	100 %	100 %
% Improvement	8.7%	11.1%	4.2%	7.5%

Table 3. %Yield and EPC savings under Vdd scaling (iso-yield cond.)

Arithmetic Units (16bit)	Vdd (V)	%Yield	%EPC savings
Proposed CSM	0.9	93	16.5
Proposed WTM	0.85	96	25.5

Assuming the critical path delay of a combinational logic block in the absence of process variations to be T_{crit}, we designate a simulation run to be a failure if the critical path delay of the combinational unit exceeded $1.2T_{crit}$. With this criterion as the delay failure metric, the yield of different arithmetic units was calculated in table 2. In all the arithmetic units considered, the yield of our proposed design was found to be 100% under a nominal Vdd of 1V. This can be attributed to the fact that the short latency paths under variations do not exceed the one clock cycle bound and the long latency path, if activated, is evaluated by elastic clocking scheme. We can clearly observe that our proposed approach achieves varying degree of yield improvement (4.2% to 11.1%) for different arithmetic units compared to the conventional implementation.

In our simulations we consider two iso-yield conditions: 1) when the proposed design is operated at 1V and 2) when conventional design is operated at 1V. In order to have an iso-yield of approximately 100%, when the proposed design is operating at 1V, the conventional design had to be operated at 1.1V which gives *24%* and *29%* power savings for CSM and WTM, respectively as shown in figure 9a and 9b. However, for the conventional design operating at 1V, if a certain yield target is desired (CSM Yield=93%, WTM Yield=96%), then the proposed design can operate at a lower Vdd. Table 3 shows the percentage EPC savings obtained by using *equation 3*. Figures 10 and 11 show the power dissipation under iso-yield conditions for the conventional design operating at 1V and the proposed architecture at a scaled down Vdd (to meet the same yield target). An interesting observation is that the percentage power savings increases with an increase in the number of bits in the adders and multipliers. This is due to the fact that, with an increase in the length of the critical path, there is more slack to be exploited, in terms of Vdd scaling, after splitting the critical path. For example, in figure 11c, to have iso-yield of 96%, the proposed 16 bit WTM can be operated at a scaled down Vdd of 0.85V. The area overhead in the adders and multipliers decreases with an increase in the number of bits which is expected as the LPB circuitry remains unchanged (10 bits input to LPB) while the area of original circuit increase as the number

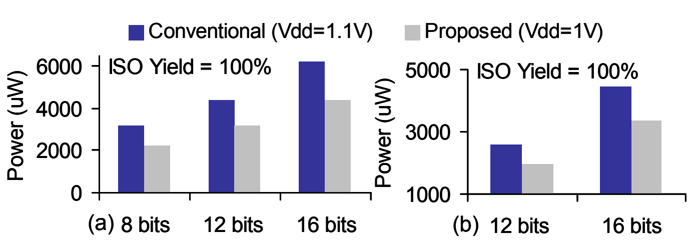

Figure 9. Power consumption under ISO yield conditions for (a) CSM and (b) WTM

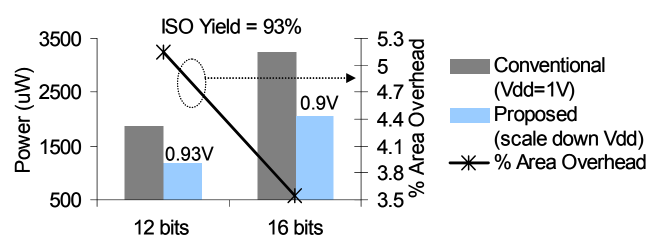

Figure 10. Power consumption under ISO Yield conditions and area overhead for CSM

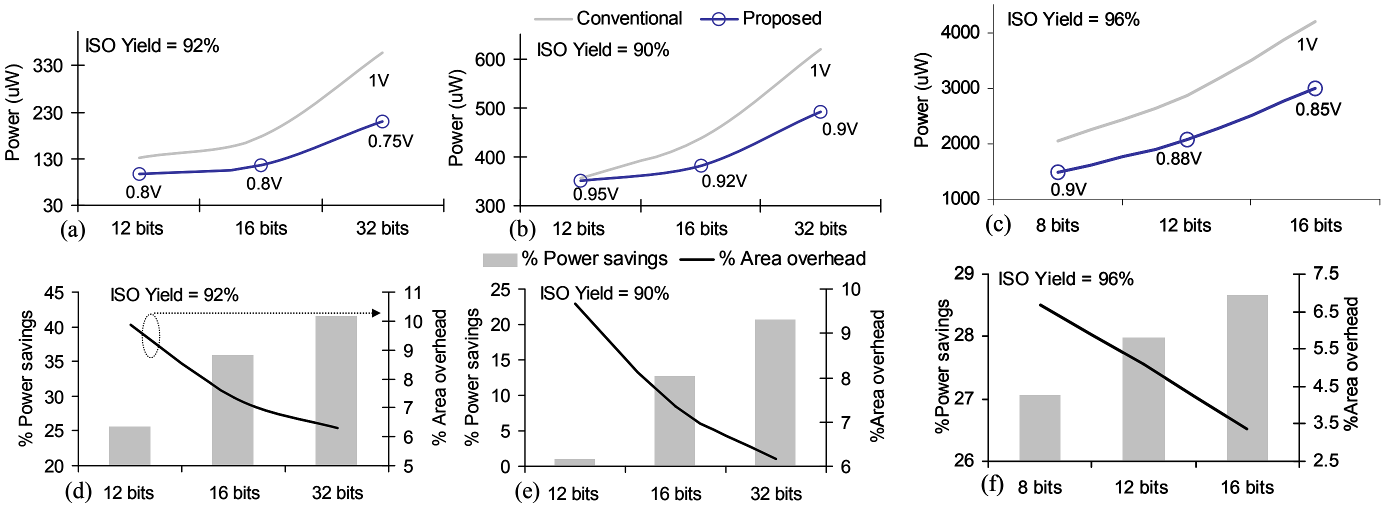

Figure 11. Power consumption under ISO Yield conditions for (a) ripple carry adder (RCA) , (b) cascaded carry select adder (CCSA) and (c) Wallace multiplier (WTM), % Power savings and area overhead under ISO Yield conditions for (d) RCA, (e) CCSA, (f) WTM

of bits increases. An area overhead of approximately 5-10% is obtained in case of the arithmetic units. In figure 12, the tradeoff between throughput penalty and area overhead for a 16 bit WTM is shown. From the graph we observe that throughput penalty decreases with increase in number of input bits to the LPB. In order to determine the actual throughput penalty from a system level perspective, we incorporated the proposed arithmetic units in a five stage DLX pipeline. The throughput penalty was assessed by running SPEC2K [9] benchmarks in Simple Scalar [10] simulator. The results given in figure 13 show on an average a 3.03% throughput penalty by using 10 bits as input to the LPB.

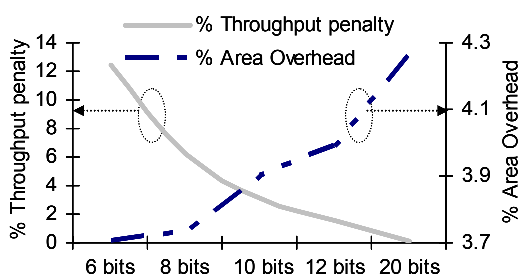

Figure 12. Performance versus area overhead with # of inputs to LPB (WTM 16 bits)

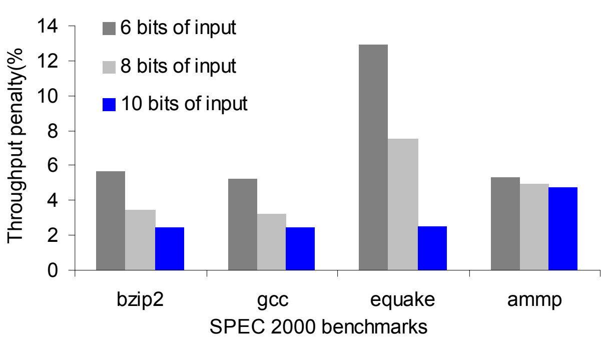

Figure 13. Throughput penalty Vs. Number of inputs to LPB

5. CONCLUSION

In this paper we proposed a new design methodology for process variation tolerant, low power arithmetic units (adders and multipliers). The design technique improves the yield of arithmetic units by reclaiming chips which would otherwise fail due to Vdd-scaling and/or process variations. We use a novel elastic clocking scheme to work around possible failures. Simulation results show a significant improvement in the yield while achieving reasonable energy per computation savings. Compared to traditional techniques our methodology tackles two issues: higher yield in the face of process variations and lower power consumption by Vdd-scaling, in a unified manner. Another attractive feature of our design is the flexibility it offers. We can either choose to have yield as the primary design metric in which case we sacrifice little power saving under scaled Vdd or decide to have a certain yield and obtain additional advantage of reduced energy per operation.

6. ACKNOWLEDGEMENTS

The authors acknowledge the support of the Gigascale Systems Research Focus Center, one of five research centers funded under the Focus Center Research Program, a Semiconductor Research Corporation program.

7. REFERENCES

[1] S. Borkar et. al., “Design and reliability challenges in nanometer technologies”, *DAC*, 2004.
[2] Y. Chen, et. al, “Cascaded carry-select adder (CCSA): a new structure for low-power CSA design”, *ISLPED*, 2005.
[3] H. Suzuki, et. al, “Low Power Adder with. Adaptive Supply Voltage”, *ICCD*, pp. 103-106, October 2003.
[4] J. Rabaey, “*Digital Integrated Circuits: A Design Perspective”*, Prentice Hill, Second Edition, 2003.
[5] L. Benini, et. al, “Telescopic units: Increasing the average throughput of pipelined designs by adaptive latency control”, *DAC*, pp. 22-27, June 1997.
[6] S. Ghosh, et. al, “A New Paradigm for Low-power, Variation-Tolerant Circuit Synthesis Using Critical Path Isolation”, *ICCAD*, 2006.
[7] BPTM 70nm: Berkeley predictive technology model.
[8] Synopsys Design Compiler, www.synopsys.com
[9] SPEC 2000 Benchmarks, www.spec.org
[10] Simplescalar Tool Set, www.simplescalar.com