INTERNATIONAL JOURNAL OF RESEARCH
In Advanced Engineering Technologies

Volume 6, Issue 1 JAN 2017

Implementation of High Throughput Aging-Aware Reliable Multiplier with variable latency

Rutuja Deshpande[1] **Keerthi Srilekha**[2]
rutuja.deshpande3@gmail.com[1] srilekha402@gmail.com[2]
[1]PG Scholar, Dept of ECE, Sudheer Reddy College of Engineering & Technology for Women, Dichpally, Nizamabad, Telangana.
[2]Assistant Professor, Dept of ECE, Sudheer Reddy College of Engineering & Technology for Women, Dichpally, Nizamabad, Telangana.

Abstract: With the rapid progress in semiconductor technology and the shrinking of device geometries, the resulting processors are increasingly becoming prone to effects like aging and soft errors. As a processor ages, its electrical characteristics degrade, i.e., the switching times of its transistors increase. Hence, the processor cannot continue error-free operation at the same clock frequency and/or voltage for which it was originally designed. In order to mitigate such effects, recent research proposes to equip processors with special circuitry that automatically adapts its clock frequency in response to changes in its circuit-level timing properties (arising from changes in its electrical characteristics). From the point of view of tasks running on these processors, such autonomic frequency scaling (AFS) processors become slower as they gradually age. This leads to additional execution delay for tasks, which needs to be analyzed carefully, particularly in the context of hard realtime or safety-critical systems. Hence, for real-time systems based on AFS processors, the associated schedulability analysis should be aging-aware which a relatively unexplored topic is so far. In this paper we propose a schedulability analysis framework that accounts such aging-induced degradation and changes in timing properties of the processor, when designing hard real-time systems. In particular, we address the schedulability and task mapping problem by taking a lifetime constraint of the system into account. In other words, the system should be designed to be fully operational (i.e., meet all deadlines) till a given minimum period of time (i.e., its lifetime). The proposed framework is based on an aging model of the processor which we discuss in detail. In addition to studying the effects of aging on the schedulability of real-time tasks, we also discuss its impact on task mapping and resource dimensioning.

Keywords—Adaptive hold logic (AHL), negative bias temperature instability (NBTI), positive bias temperature instability (PBTI), reliable multiplier, variable latency.

I. Introduction

As device geometries continue to shrink with the advances in fabrication technology, the resulting processors are increasingly becoming susceptible to effects like aging. As a processor ages, the switching times of its constituent transistors increase because of which the processor is no longer able to sustain the clock frequency it was originally designed for. Hardware solutions to mitigate such effects consist of equipping processors with on-chip monitors or sensors [8], [5] that measure the timing margin available to circuits on the chip, or variations in their timing behavior arising from changes in their electrical characteristics. The output from such monitors is coupled with the clock generation circuit to adjust the clock frequency in response to changes in the signal propagation delay due to effects like aging. Such techniques have already been used in IBM's POWER7 architecture in order to automatically adjust the processor's clock frequency and voltage level, with the aim of saving energy. The same technique is also applicable to cope with the effects of aging.

In such autonomic frequency scaling (AFS) processors, the execution times of tasks increase over time as the processor ages. For real-time or safety-critical applications running on such AFS processors, a natural question is: how to perform timing or schedulability analysis? Given a life-time constraint, i.e., the duration of time during which all deadlines have to be met, one obvious solution would be to perform schedulability analysis with task execution times at the end of this duration (i.e., using the aged execution times). However, for cost sensitive domains such a naıve approach might be overly pessimistic and lead to resource over dimensioning.

As a concrete example, let us consider the automotive domain. Today, high-end cars have 50-100 electronic control units (ECUs) or processors to run a variety of hard real-time, safety-critical control applications. Currently, these ECUs are often low-cost processors or microcontrollers that run at 200-500 MHz. However, in order to reduce cost, weight and cabling requirements, there is an increasing effort on ECU consolidation, i.e., integrating multiple functionalities on fewer more powerful ECUs. In the near future, we will see ECU architectures evolve into powerful, multi-core processors that are currently found in the high-performance computing domain. As soon as that happens, such ECUs will be faced with the effects of aging, soft errors, etc., especially because they will be exposed to a wide variety of operating environments and temperatures, depending on where the car is deployed. This will be coupled with the fact that additional processor cooling mechanisms – as found in the mainstream computing domain – will probably be absent. Given that it is common for automotive OEMs to provide guarantees in the range of 15 years, it has to be ensured that tasks running on aging-aware autonomic frequency scaling or AFS ECUs continue to meet all deadlines over this entire time period. In this paper we propose an appropriate processor aging-aware schedulability analysis for such scenarios.

Processors with autonomic frequency scaling: In general, aging effects can be divided into two categories: degradation and destructive effects. Degradation effects may be classified into Negative Bias Temperature Instability (NBTI) and HotCarrier Injection (HCI). These cause a shift in a device's electrical characteristics, such as threshold voltage or on-current, which leads to prolonged switching times of transistors. On the other hand, destructive effects like Time-Dependent Dielectric Breakdown (TDDB) or Electro Migration (EM) cause the destruction of device parts and irreparable damage. In this paper, we are concerned with degradation effects, i.e., those that do not damage a processor but rather affect its timing behavior.

As a processor ages, its transistors gradually need more time for switching (due to degradation effects). Hence, as mentioned before, the processor cannot sustain the clock frequency it was originally designed for. While very large safety margins or timing guardbands can be used to mitigate such effects, they lead to poor resource utilization and hence more expensive processors, which are not acceptable in cost-sensitive domains like automotive architectures. Autonomic frequency scaling (AFS) processors use critical path monitors (CPM) which allow measuring the maximum delay degradation at runtime (see Fig. 1). Depending on the implementation, an AFS processor may become slower as it ages, in order to keep the processor operational. To reliably design real-time systems based on AFS processors, it is necessary to analyze the system's aging behavior and develop timing or schedulability analysis techniques that are aging-aware.

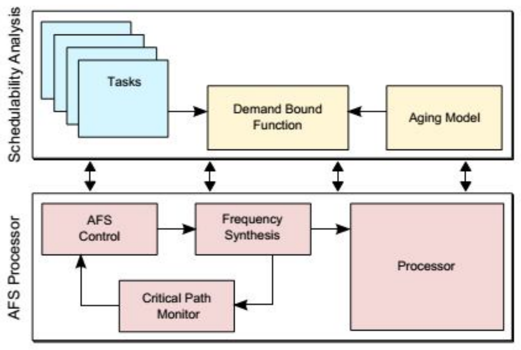


Fig. 1. Schematic view of an AFS processor

Our contributions: As discussed above, an AFS processor automatically reduces its clock frequency to account for additional circuit-level signal propagation delay incurred due to aging. This way, although the processor becomes slower, it avoids the occurrence of aging-dependent functional/computation errors. However, in the context of realtime systems, we need to analyze what the maximum slowdown is, to be able to guarantee deadlines all along the system's lifetime. For this purpose, we make use of the aging model from [14] to predict the processor's worst-case aging behavior within the desired lifetime tlif e. This model returns the maximum aging-dependent delay Dmax as a function of the time under stress tstress (i.e., the time in which the processor is busy executing some workload).

Now, to account for aging in schedulability analysis, we can use our aging-model to obtain Dmax considering a tstress equal to tlif e and compute the speed of the aged AFS processor after tlif e of continuous use. Clearly, if the aged AFS processor can guarantee all deadlines, then the system is schedulable along its whole lifetime. However, even considering aging, the processor utilization is normally below 100% and, in most cases, tstress is much less than tlif e (i.e., the processor is not being constantly used during its lifetime, but rather it has some idle intervals). As a result, this first na¨ ıve approach leads to a pessimistic slow-down estimation and, hence, a more expensive design.

II. Literature Survey:

The aging behavior of semiconductors is a major topic of research within the processor architecture community, but it has so far received relatively less attention from the software community. Notable exceptions to this are [7], [23], [16], [17]. There exists a large body of work that addresses aging effects at the hardware level. For example, Wu and Marculescu presented optimization techniques that allow synthesizing digital circuits that are less prone to aging degradation [23]. In [24], different process variations at a chip level are studied and characterized. In [16], [17], software techniques are presented to cope with problems that are posed by unreliable hardware. These include devising reliability-aware instruction sets, coupled with appropriate instruction scheduling using reliability aware compilation techniques.

Recently, in [7], Huang et al. presented an allocation framework based on a heuristic that aims at maximizing the lifetime of an SoC (System-on-Chip). Our work here follows this line of research and also deals with the allocation issues that arise in aging devices. However, in contrast to recent research efforts, the approach presented here deals with the schedulability of real-time tasks and extends the state-ofthe-art by considering hardware-dependent behavior in the schedulability and mapping problem, viz., aging effects.

III.PRELIMINARIES

Row-Bypassing Multiplier

A low-power row-bypassing multiplier is also proposed to reduce the activity power of the AM. The operation of the low-power row-bypassing multiplier is similar to that of the low-power column-bypassing multiplier, but the selector of the multiplexers and the tri state gates use the multiplication.

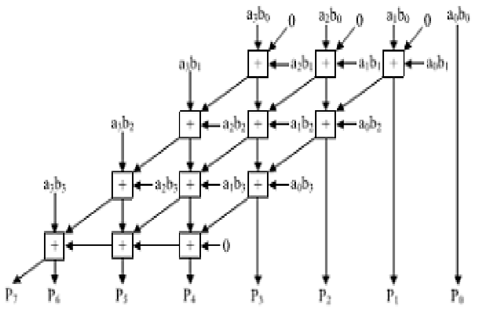


Fig 2 .4 × 4 normal AM.

Fig. 3 is a 4 × 4 row-bypassing multiplier. Each input is connected to an FA through a tristate gate. When the inputs are 11112 * 10012, the two inputs in the first and second rows are 0 for FAs. Because b1 is 0, the multiplexers in the first row select aib0 as the sum bit and select 0 as the carry bit. The inputs are bypassed to FAs in the second rows, and the tristate gates turn off the input paths to the FAs. Therefore, no switching activities occur in the first-row FAs; in return, power consumption is reduced. Similarly, because b2 is 0, no switching activities will occur in the second-row FAs. However, the FAs must be active in the

Volume 6, Issue 1 JAN 2017

third row because the b3 is not zero.

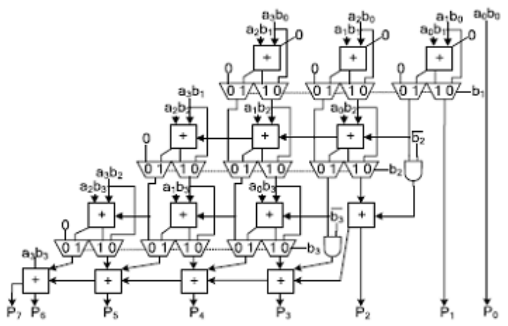


Fig 3. 4X4 Row by passing multiplier

selector of the multiplexer to decide the output of the FA, and ai can also be used as the selector of the tristate gate to turn off the input path of the FA. If ai is 0, the inputs of FA are disabled, and the sum bit of the current FA is equal to the sum bit from its upper FA, thus reducing the power consumption of the multiplier. If is 1, the normal sum result is selected. More details for the column-bypassing multiplier can be found.

Column-Bypassing Multiplier

A column-bypassing multiplier is an improvement on the normal array multiplier (AM). The AM is a fast parallel AM and is shown in Fig. 1. The multiplier array consists of (n−1) rows of carry save adder (CSA), in which each row contains (n − 1) full adder (FA) cells. Each FA in the CSA array has two outputs: 1) the sum bit goes down and 2) the carry bit goes to the lower left FA. The last row is a ripple adder for carry propagation.

The FAs in the AM are always active regardless of input states. A low-power column-bypassing multiplier design is proposed in which the FA operations are disabled if the corresponding bit in the multiplicand is 0. Fig. 4 shows a 4×4 column-bypassing multiplier. Supposing the inputs are 10102 * 11112, it can be seen that for the FAs in the first and third diagonals, two of the three input bits are 0: the carry bit from its upper right FA and the partial product aibi . Therefore, the output of the adders in both diagonals is 0, and the output sum bit is simply equal to the third bit, which is the sum output of its upper FA.

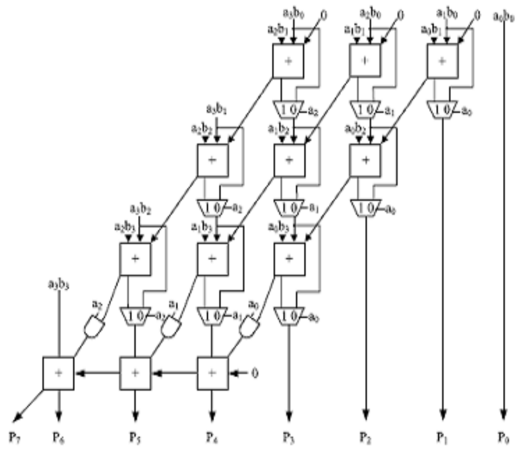


Fig. 4. 4 × 4 column-bypassing multiplier

Variable-Latency Design

In ripple carry adders, the carry propagation time is the major speed limiting factor as seen in the previous lesson.

Most other arithmetic operations, e.g. multiplication and division are implemented using several add/subtract steps. Thus, improving the speed of addition will improve the speed of all other arithmetic operations.

Accordingly, reducing the carry propagation delay of adders is of great importance. Different logic design approaches have been employed to overcome the carry propagation problem.

One widely used approach employs the principle of carry look-a head solves this problem by calculating the carry signals in advance, based on the input signals.

This type of adder circuit is called as carry look-ahead adder (CLA adder).It is based on the fact that a carry signal will be generated in two cases:

(1) When both bits Ai and Biare 1, or

(2) When one of the two bits is 1 and the carry-in (carry of the previous stage) is 1.

The Boolean expression of the carry outputs of various stages can be written as follows:

$$C_1 = G_0 + P_0C_0$$
$$C_2 = G_1 + P_1C_1 = G_1 + P_1(G_0 + P_0C_0)$$
$$= G_1 + P_1G_0 + P_1P_0C_0$$
$$C_3 = G_2 + P_2C_2 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$$
$$C_4 = G_3 + P_3C_3$$
$$= G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0$$

In general, the ith. carry output is expressed in the form Ci = Fi(P's, G's , C0).

In other words, each carry signal is expressed as a direct SOP function of C0 rather than its preceding carry signal.

Since the Boolean expression for each output carry is expressed in SOP form, it can be implemented in two-level circuits.

The 2-level implementation of the carry signals has a propagation delay of 2 gates, i.e., 2τ.

The 4-bit carry look-ahead (CLA) adder consists of 3 levels of logic:

First

First level: Generates all the P & G signals. Four sets of P & G logic (each consists of an XOR gate and an AND gate).

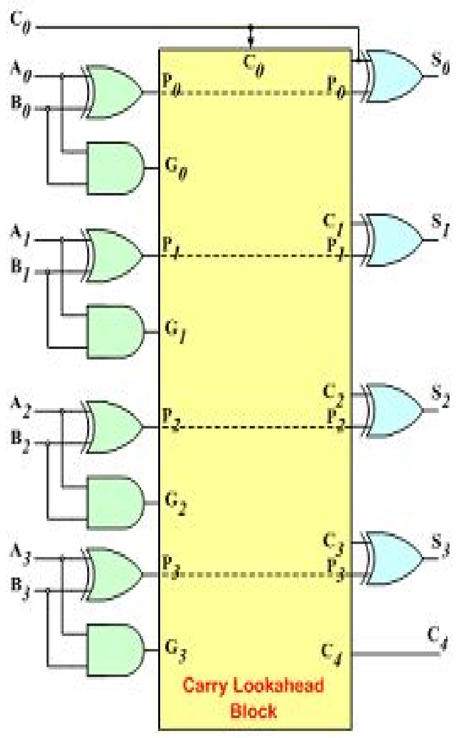
Output signals of this level (P's & G's) will be valid after 1τ. Second level: The Carry Look-Ahead (CLA) logic block which consists of four 2-level implementation logic circuits. It generates the carry signals (C1, C2, C3, and C4) as defined by the above expressions. Output signals of this level (C1, C2, C3, and C4) will be valid after 3τ.

Third level: Four XOR gates which generate the sum signals (Si) (Si= Pi⊕ Ci). Output signals of this level (S0, S1, S2, and S3) will be valid after 4τ.

Thus, the 4 Sum signals (S0, S1, S2 & S3) will all be valid after a total delay of 4τ compared to a delay of (2n+1) □τ for Ripple Carry adders.

For a 4-bit adder (n = 4), the Ripple Carry adder delay is 9τ.

The disadvantage of the CLA adders is that the carry expressions (and hence logic) become quite complex for more than 4 bits.



The basic concept is to execute a shorter path using a shorter cycle and longer path using two cycles. Since most paths execute in a cycle period that is much smaller than the critical path delay, the variable-latency design has smaller average latency.

For example, Fig. 4 is an 8-bit variable-latency ripple carry adder (RCA). A8–A1, B8–B1 are 8-bit inputs, and S8–S1 are the outputs. Supposing the delay for each FA is one, and the maximum delay for the adder is 8.

Through simulation, it can be determined that the possibility of the carry propagation delay being longer than 5 is low. Hence, the cycle period is set to 5, and hold logic is added to notify the system whether the adder can complete the operation within a cycle period.

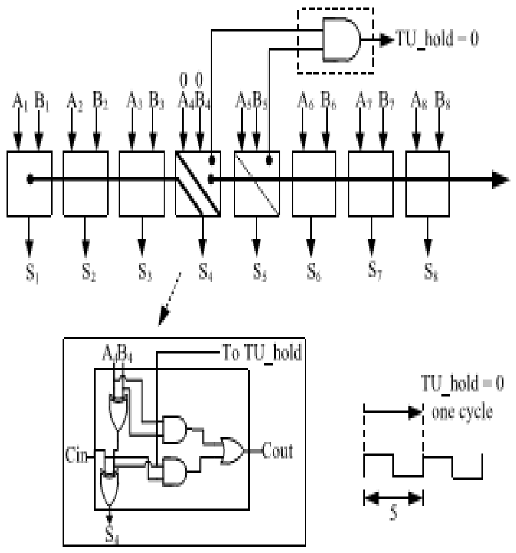

TU_hold = 0
0 0
A_1 B_1 A_2 B_2 A_3 B_3 A_4B_4 A_5B_5 A_6 B_6 A_7 B_7 A_8 B_8
S_1 S_2 S_3 S_4 S_5 S_6 S_7 S_8
A_4B_4
To TU_hold
Cin
Cout
S_4
TU_hold = 0
one cycle
5

Fig. 5. 8-bit RCA with a hold logic circuit.

Fig. 4 also shows the hold logic that is used in this circuit. The function of the hold logic is (A4 XORB4)(A5 XORB5). If the output of the hold logic is 0, i.e., A4=B4orA5=B5, either the fourth or the fifth adder will not produce a carryout. Hence, the maximum delay will be less than one cycle period. When the hold logic output is 1, this means that the input can activate paths longer than 5, so the hold logic notifies the system that the current operation requires two cycles to complete. Two cycles are sufficient for the longest path to complete (5 * 2 is larger than 8).

The performance improvement of the variable-latency design can be calculated as follows: if the possibility of each input being 1 is 0.5, the possibility of (A4 XORB4) (A5 XORB5) being 1 is 0.25. The average latency for the variable-latency design is $0.75*5+0.25*10=6.25$. Compared with the simple fixed-latency RCA, which has an average latency of 8, the variable-latency design can achieve a 28% performance improvement.

Fig. 5 shows the path delay distribution of a 16×16 AM and for both a traditional column-bypassing and traditional row-bypassing multiplier with 65 536 randomly chosen input patterns. All multipliers execute operations on a fixed cycle period. The maximum path delay is 1.32 ns for the AM, 1.88 ns for the column-bypassing multiplier, and 1.82 ns for the row-bypassing multiplier. It can be seen that for the AM, more than 98% of the paths have a delay of<0.7 ns. Moreover, more than 93% and 98% of the paths in the FLCB and row-bypassing multipliers present a delay of <0.9 ns, respectively. Hence, using the maximum path delay for all paths will cause significant timing waste for shorter paths, and redesigning the multiplier with variable latency can improve.their performance Another key observation is that the path delay for an operation is strongly tied to the number of zeros in the multiplicands in the column-bypassing multiplier. Fig. 6 shows the delay distribution of the 16×16 column-bypassing multiplier under three different numbers of zeros in the multiplicands: 1) 6; 2) 8; and 3) 10. Three thousand randomly selected patterns are used in each experiment. It can be seen as the number of zeros in the multiplicands increases, delay distributionis left shifted, and average delay is reduced. The reason for this is the multiplicand is used as the select line for column-bypassing multipliers, and if more zeros exist in the ,multiplicand, more FAs will be skipped, and the sum bit from the upper FA is passed to the lower FA, reducing the path delay. Note that similar experiments are also done for row-bypassing multipliers. However, because the results are similar, they are not shown to avoid duplications.

For a row-bypassing multiplier, the multiplicators are used to determine whether a pattern needs one cycle or two cycles to complete an operation because the multiplicator is used as the select line. This makes the column-bypassing multiplicand and row-bypassing multiplier excellent candidates for the variable latency design since we can simply examine the number of zeros in the multiplicand or multiplicator to predict whether the operation requires one cycle or two cycles to complete.

IV.Proposed Aging-Aware Multiplier

Proposed Architecture

Fig. 5 shows our proposed aging-aware multiplier architecture, which includes two m-bit inputs (mis a positive number), one 2m-bit output, one column- or row-bypassing multiplier, 2m1-bit Razor flip-flops [27], and an AHL circuit.

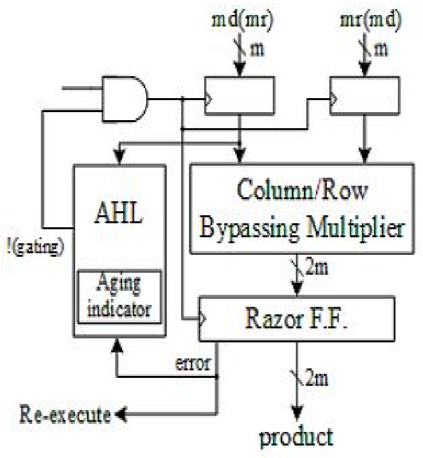


Fig. 6. Proposed architecture (md means multiplicand; mr means multiplicator).

Hence, the two aging-aware multipliers can be implemented using similar architecture, and the difference between the two bypassing multipliers lies in the input signals of the AHL. According to the bypassing selection in the columnor row-bypassing multiplier, the input signal of the AHL in the architecture with the column-bypassing multiplier is the multiplicand, whereas that ofthe row-bypassing multiplier is the multiplicator. Razor flip-flops can be used to detect Fig. 6. Razor flip flops. whether timing violations occur before the next input pattern arrives.

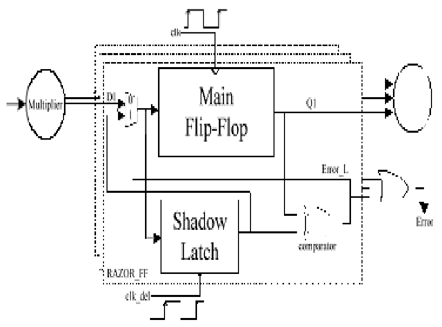


Fig.76. Razor flip flops

Fig. 6 shows the details of Razor flip-flops. A 1-bit Razor flip-flop contains a main flip-flop, shadow latch,XORgate, and mux. The main flip-flop catches the execution result for the combination circuit using a normal clock signal, and the shadow latch catches the execution result using a delayed clock signal, which is slower than the normal clock signal. If the latched bit of the shadow latch is different from that of the main flip-flop, this means the path delay of the current operation exceeds the cycle period, and the main flip-flop catches an incorrect result. If errors occur, the Razor flip-flop will set the error signal to 1 to notify the system to reexecute the operation and notify the AHL circuit that an error has occurred. We use Razor flip-flops to detect whether an operation that is considered to be a one-cycle pattern can really finish in a cycle. If not, the operation is reexecuted with two cycles. Although the reexecution may seem costly, the overall cost is low because the reexecution frequency is low.

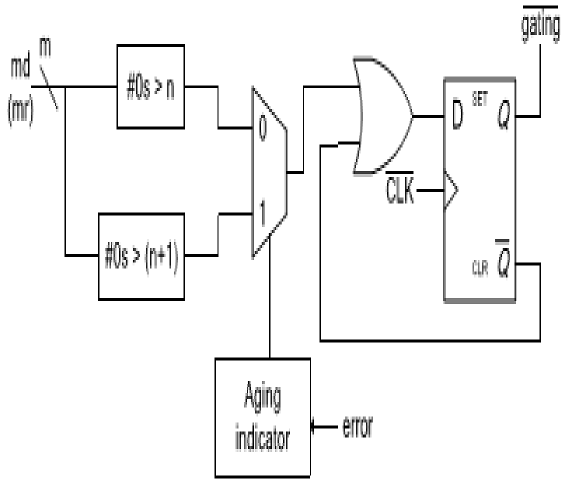
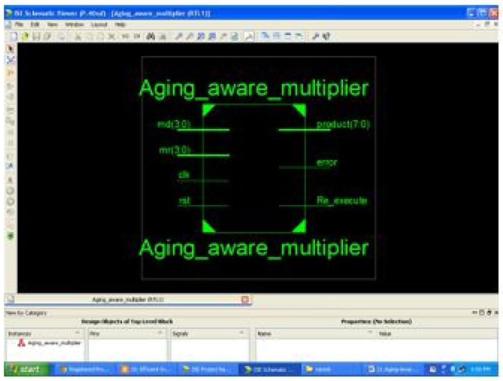


Fig. 8. Diagram of AHL (md means multiplicand; mr means multiplicator).

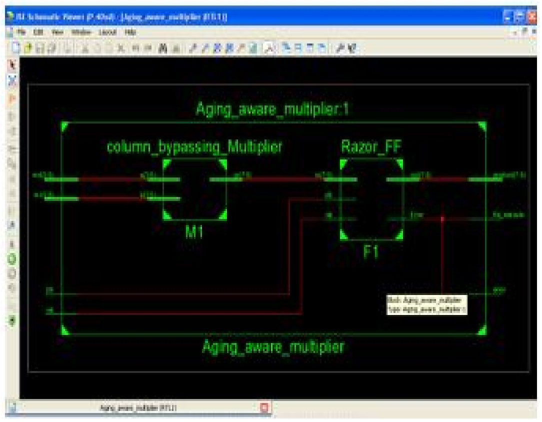
The AHL circuit is the key component in the aging-ware variable-latency multiplier. Fig. 12 shows the details of the AHL circuit. The AHL circuit contains an aging indicator, two judging blocks, one mux, and one D flip-flop. The aging indicator indicates whether the circuit has suffered significant performance degradation due to he aging effect. The aging indicator is implemented in a simple counter that counts the number of errors over a certain amount of operations and is reset to zero at the end of those operations. If the cycle period is too short, the column- or row-bypassing multiplier is not able to complete these operations successfully, causing timing violations. These timing violations will be caught by the Razor flip-flops, which generate error signals. If errors happen frequently and exceed a predefined threshold, it means the circuit has suffered significant timing degradation due to the aging effect, and the aging indicator will output signal 1; otherwise, it will output 0 to indicate the aging effect is still not significant, and no actions are needed.

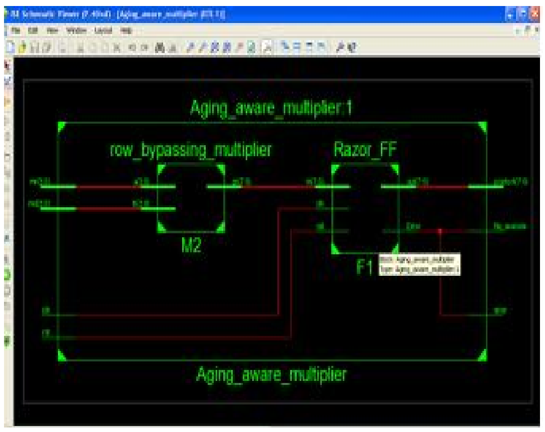
V.SIMULATION RESULTS

Block diagram

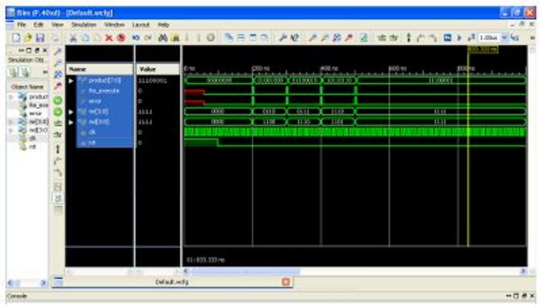


RTL Schematic diagram





Simulation output waveform



Extension Work:

This paper can be extended by Vedic Multiplier ,so that we get less delay and lower area can be achieved while comparing to other multiplier techniques.

Vedic mathematics was reconstructed from the ancient Indian scriptures (Vedas) by Swami Bharati Krishna Tirthaji Maharaja (1884-1960) after his eight years of research on Vedas. Vedic mathematics is mainly based on sixteen principles or word-formulae which are termed as sutras. This is a very interesting field and presents some effective algorithms which can be applied to various branches of engineering such as computing and digital signal processing. Integrating multiplication with Vedic Mathematics techniques would result in the saving of computational time. Thus, integrating Vedic mathematics for the multiplier design will enhance the speed of multiplication operation. This type of multiplier operates much faster than an array multiplier for longer operands because its computation time is proportional to the logarithm of the word length of operands. This gives us method for hierarchical multiplier design. So the design complexity gets reduced for inputs of large number of bits and modularity gets increased. Urdhva tiryakbhyam, Nikhilam and Anurupye sutras are such algorithms which can reduce the delay, power and hardware requirements for multiplication of numbers.

FPGA implementation of this multiplier shows that hardware realization of the Vedic mathematics algorithms is easily possible. The high speed multiplier algorithm exhibits improved efficiency in terms of speed.

VI.CONCLUSION

This paper proposed an aging-aware variable-latency multiplier design with the AHL. The multiplier is able to adjust the AHL to mitigate performance degradation due to increased delay. The experimental results show that our proposed architecture with 8x8 multiplication with CSA as last stage instead of Normal RCA adder it will decrease the delay and improve the performance compared with previous designs.

REFERENCES

[1] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. Software Engineering Journal, 8(5):284–292, 1993.
[2] E. Bini and G. Buttazzo. Biasing effects in schedulability measures. In Euromicro Conference on Real-Time Systems, June 2004.
[3] E. Bini and G. Buttazzo. Measuring the performance of schedulability tests. Real-Time Systems, 30(1-2):129–154, 2005.
[4] K. Bowman, J. Tschanz, C. Wilkerson, S.-L. Lu, T. Karnik, V. De, and S. Borkar. Circuit techniques for dynamic variation tolerance. In Design Automation Conference (DAC), July 2009.
[5] A. Drake, R. Senger, H. Deogun, G. Carpenter, S. Ghiasi, T. Nguyen, N. James, M. Floyd, and V. Pokala. A distributed critical-path timing monitor for a 65nm high-performance microprocessor. In International Solid-State Circuits Conference (ISSCC), Feb. 2007.
[6] M. Floyd, M. Allen-Ware, K. Rajamani, B. Brock, C. Lefurgy, A. Drake, L. Pesantez, T. Gloekler, J. Tierno, P. Bose, and A. Buyuktosunoglu. Introducing the adaptive energy management features of the POWER7 chip. IEEE Micro, 31(2):67–75, 2011.
[7] L. Huang, F. Yuan, and Q. Xu. Lifetime reliability-aware task allocation and scheduling for MPSoC platforms. In Design, Automation, and Test in Europe (DATE), Apr. 2009.
[8] J. Keane, T.-H. Kim, X. Wang, and C. Kim. On-chip reliability monitors for measuring circuit degradation. Microelectronics Reliability, 50(8):1039–1053, 2010.
[9] C. Lefurgy, A. Drake, M. Floyd, M. Allen-Ware, B. Brock, J. Tierno, and J. Carter. Active management of timing guardband to save energy in POWER7. In International Symposium on Microarchitecture (MICRO), Dec. 2011.
[10] J. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In Real-Time Systems Symposium (RTSS), Dec. 1990.
[11] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: exact characterization and average case behavior. In RealTime Systems Symposium (RTSS), Dec. 1989.
[12] J. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. Performance Evaluation, 2(4):237–250, 1982.
[13] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in hard real-time environments. Journal of the Association for Computing Machinery, 20(1):40–61, 1973.
[14] D. Lorenz, M. Barke, and U. Schlichtmann. Aging analysis at gate and macro cell level. In International Conference on Computer-Aided Design (ICCAD), Nov. 2010.
[15] J. Park and J. Abraham. A fast, accurate and simple critical path monitor for improving energy-delay product in DVS systems. In International Symposium on Low Power Electronics and Design (ISLPED), Aug. 2011.
[16] S. Rehman, M. Shafique, F. Kriebel, and J. Henkel. Reliable software for unreliable hardware: embedded code generation aiming at reliability. In International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), Oct. 2011.
[17] S. Rehman, M. Shafique, F. Kriebel, and J. Henkel. RAISE: Reliabilityaware instruction scheduling for unreliable hardware. In Asia and South Pacific Design Automation Conference (ASP-DAC), Jan. 2012.
[18] S. S. Sapatnekar. Static timing analysis. In L. Scheffer, L. Lavagno, and G. Martin, editors, EDA for IC implementation, circuit design, and process technology. Taylor and Francis, 2006.
[19] V. Stojanovic, D. Markovic, B. Nikolic, M. Horowitz, and R. Brodersen. Energy-delay tradeoffs in combinational logic using gate sizing and supply voltage optimization. In European Solid-State Circuits Conference (ESSCIRC), Sept. 2002.
[20] J. Tschanz, N.-S. Kim, S. Dighe, J. Howard, G. Ruhl, S. Vanga, S. Narendra, Y.

Hoskote, H. Wilson, C. Lam, M. Shuman, C. Tokunaga, D. Somasekhar, S. Tang, D. Finan, T. Karnik, N. Borkar, N. Kurd, and V. De. Adaptive frequency and biasing techniques for tolerance to dynamic temperature-voltage variations and aging. In International Solid-State Circuits Conference (ISSCC), Feb. 2007.
[21] C. Visweswariah, K. Ravindran, K. Kalafala, S. Walker, S. Narayan, D. Beece, J. Piaget, N. Venkateswaran, and J. Hemmet. First-order incremental block-based statistical timing analysis. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 25(10), 2006.
[22] Y. Wang, X. Chen, W. Wang, V. Balakrishnan, Y. Cao, Y. Xie, and H. Yang. On the efficacy of input vector control to mitigate NBTI effects and leakage power. In International Symposium on Quality Electronic Design (ISQED), Mar. 2009.

BIOGRAPHIES

Keerthi Srilekha Currently working as Assistant Professor Department of Electronics and Communication Engineering in Sudheer Reddy College of Engineering & Technology for Women, Dichpally, Nizamabad, Telangana. Her current research interest includes VLSI Design.

Rutuja Deshpande is currently a PG scholar of VLSI in ECE Department. She received B.TECH degree from JNTU. Her current research interest includes Analysis &VLSI System Design.