

Efficient Multiplier Design Using Adaptive Hold Logic with Montgomery Algorithm

Ramya N¹, Rose Mistica S², Subikma Binusha V³, Prof Savitha G⁴

^{1,2,3}Students, Dept of Electronics and Communication Engineering, Jeppiaar SRR Engineering College, Chennai, Tamil Nadu

⁴Assistant Professor, Dept. of Electronics and Communication Engineering, Jeppiaar SRR Engineering College, Chennai

Abstract - In most of the digital signal processors, multiplier is used as a key component. So, the performance of the system depends on the throughput of the multiplier. Now a days, reliability is an important design concern in advanced technology nodes. Performance of the system is significantly affected by the aging of transistor and the system may fail due to delay problems in long term. The impact of aging getting higher with the scaling of transistor. One of the main cause for aging in transistor is Bias Temperature Instability (BTI). Due to this effect threshold voltage of the transistor increases over time and it reduces the multiplier speed. Over-design approaches can be used to reduce the aging effect, but these may cause power and area inefficiency. Fixed latency designs have high chance of timing violations. So, a multiplier with variable latency is used for reliable operation under BTI effects. An Adaptive Hold Logic (AHL) is used for the proper selection of cycle period and an Error Detection Correction Pulsed Latch (ECPL) is used for the detection of timing errors. In modular arithmetic computation, Montgomery multiplication algorithm is used to perform faster modular multiplication which was introduced by Peter L Montgomery In 1985.

Key Words: Bias Temperature Instability, Razor Flipflop, Error Detection and Correction Pulsed Latch, Adaptive Hold Logic, Montgomery Multiplication Algorithm.

1. INTRODUCTION

Digital multipliers square measure among the foremost vital arithmetic practical units in several applications, like the Fourier remodel, distinct trigonometric function transforms, and digital filtering. The turnout of those applications depends on multipliers, and if the multipliers square measure too slow, the performance of entire circuits are reduced moreover, negative bias temperature instability (NBTI) happens once a pMOS semiconductor is beneath negative bias ($V_{gs} = -V_{dd}$), during this state of affairs, the interaction between inversion layer holes and hydrogen-passivated Si atoms breaks the Si-H bond generated throughout the chemical reaction method, generating H or H₂ molecules. Once these molecules diffuse away, interface traps square measure left. The accumulated interface traps between semiconducting material and therefore the gate chemical

compound interface lead to multiplied threshold voltage (V_{th}), reducing the circuit shift speed. Once the biased voltage is removed, the reverse reaction happens, reducing the NBTI impact.

However, the reverse reaction doesn't eliminate all the interface traps generated throughout the strain section, and V_{th} is multiplied within the future. Hence, it's vital to style a reliable superior number. The corresponding impact on associate nMOS semi-conductor is Positive Bias Temperature Instability (PBTI) that happens once associate nMOS semi-conductor is beneath positive bias. Compared with the NBTI impact, the PBTI impact is way smaller on oxide/polygate transistors, and thus is sometimes un-heeded. However, for high-k/metal-gate nMOS transistors with important charge housing, the PBTI impact will not be unheeded. In fact, it's been shown that the PBTI impact is additional important than the NBTI impact on 32-nm high-k/metal-gate processes. A traditional method to mitigate the aging effect is overdesign including such things as guard-banding and gate oversizing; however, this approach can be very pessimistic and area and power inefficient. To avoid this drawback, several NBTI-aware methodologies are planned. An NBTI-aware technology mapping technique was proposed in to guarantee the performance of the circuit during its life time. In, an NBTI-aware sleep transistor was designed to reduce the aging effects on pMOS sleep-transistors, and the lifetime stability of the power-gated circuits under consideration was improved. Wu and Marculescu planned a joint logic restructuring and pin rearrangement technique, that relies on detection useful symmetries and semiconductor device stacking effects. They additionally planned AN NBTI improvement technique that thought of path sensitization. In and, dynamic voltage scaling and body-biasing techniques were proposed to reduce power or extend circuit life. These techniques, however, need circuit modification or don't offer improvement of specific circuits. Traditional circuits use crucial path delay because the overall circuit clock cycle so as to perform properly. However, the chance that the crucial ways are activated is low.

In most cases, the trail delay is shorter than the crucial path. For these noncritical paths, using the critical path

delay as the overall cycle period will result in significant timing waste. Hence, the variable latency style was planned to cut back the temporal order waste of ancient circuits the variable-latency style divides the circuit into 2 parts: 1) shorter ways and 2) longer ways. Shorter ways will execute properly in one cycle, whereas longer paths need two cycles to execute. When shorter ways are activated oft, the average latency of variable-latency designs is better than that of traditional designs. For example, many variable-latency adders were planned mistreatment the speculation technique with error detection and recovery. For modular arithmetic computation, Montgomery modular multiplication is performed for faster modular multiplication.

2. PRELIMINARIES

2.1 Column-Bypass Multiplier

A column-bypassing multiplier factor is Associate in Nursing improvement on the conventional array multiplier factor (AM). Fig 1 shows a 4x4 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 higher right solfa syllable and therefore the partial product $a_i b_i$. Therefore, the output of the adders in each diagonals is zero, and the output sum bit is simply equal to the third bit, which is the sum output of its higher solfa syllable. Hence, the solfa syllable is changed to feature 2 tri state gates and one electronic device.

The multiplicand bit a_i can be used as the selector of the multiplexer to decide the output of the FA, and a_i can also be used as the selector of the tri state gate to turn off the input path of the FA. If a_i is 0, the inputs of FA are disabled, and the sum bit of the current FA is equal to the submit from its upper FA, thus reducing the power consumption of the multiplier. If a_i is 1, the normal sum result is selected.

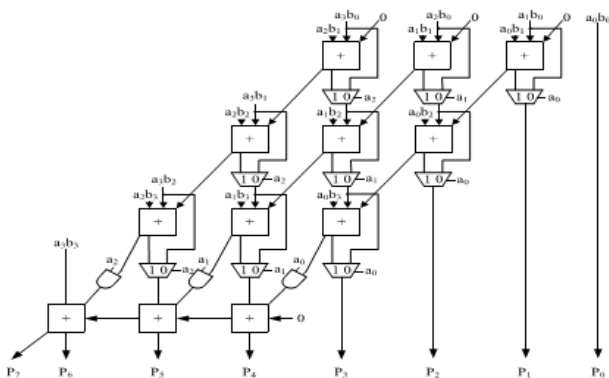


Fig-1: Column Bypass Multiplier

2.2 Row-Bypassing Multiplier:

A low-power row-bypassing number is additionally projected to scale back the activity power of the AM. The operation of the low-power row-bypassing number is comparable to it of the low-power column-bypassing number, however the selector of the multiplexers and also the tri state gates use the multiplier. Fig.2 is a 4 x 4 row-bypassing multiplier. Each input is connected to AN solfa syllable through a tri state gate.

When the inputs are 11112*10012, the two inputs in the first and second rows are 0 for FAs. Because b_1 is 0, the multiplexers in the first row select $a_i b_0$ as the sum bit and select 0 as the carry bit.

The inputs square measure bypassed to FAs within the second rows and the tristate gates shut down the input ways to the FAs. Therefore, no switch activities occur within the first-row FAs; in return power consumption is reduced. Similarly, because b_2 is 0, no switching activities will occur in the second-row FAs. However, the FAs must be active in the third row because the b_3 is not zero

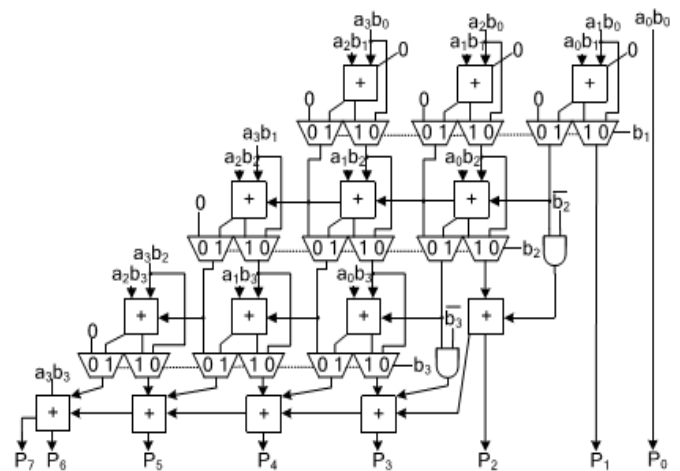


Fig-2: Row Bypass Multiplier

2.3 Variable Latency Design

Variable Latency Unit

Average Case Computation

Average-case computation, as the name suggests, refers to those computations that occur more frequently than others, and also get completed within average delays, considering the delay required by all the computations the circuit performs. Within the synchronous paradigm, two classes of techniques have been proposed for exploiting the average-case computations: variable-latency units, and error detection-correction units. Our work in this chapter focuses

on the design of BTI-resilient circuits using variable latency units (VLUs).

Unlike conventional combinational circuits that complete operations within one clock cycle, VLUs allow the computation of the combinational circuit to be completed in a variable, integer, number of clock cycles. By allowing high-probability operations to complete in a single cycle, but allowing rarer events to use multiple (typically two) cycles, the average cycle time may be shorter than that of the conventional implementation, implying that the circuit throughput for a VLU may be significantly larger. For example, Fig.4 is associate 8-bit variable-latency ripple carry adder (RCA).

A8-A1, B8-B1 are 8-bit inputs, and S8-S1 are Fig. 4. 8-bit RCA with a hold logic circuit. Fig.5 Path delay distribution of AM, column, and row-bypassing multipliers for 65 536 input patterns. 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 amount is about to five, and hold logic is other to inform the system whether or not the adder will complete the operation at intervals a cycle amount. Fig.3 additionally shows the hold logic that's utilized in this circuit.

The operate of the hold logic is $(A4 \text{ XOR } B4)(A5 \text{ XOR } B5)$. If the output of the hold logic is zero, i.e., $A4 = B4$ or $A5 = B5$, either the fourth or the fifth adder will not produce a carryout. Hence, the utmost delay are going to be but one cycle amount.

When the hold logic output is one, this suggests that the input can activate methods longer than five, that the hold logic notifies the system that this operation needs 2 cycles to finish.

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 every input being one is zero. 5, the possibility of $(A4 \text{ XOR } B4)(A5 \text{ XOR } B5)$ being 1 is 0.25.

The average latency for the variable latency style is zero. $75 * 5 + 0.25 * 10 = 6.25$. Compared with the easy fixed-latency RCA, which has an average latency of 8, the variable-latency design can achieve a 28% performance improvement. Fig.4 shows the path delay distribution of a 16×16 AM and for both a traditional column-bypassing and traditional row-bypassing multiplier with 65536 randomly chosen input patterns.

All multipliers execute operations on a set cycle amount.

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, quite ninety eight of the ways 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.

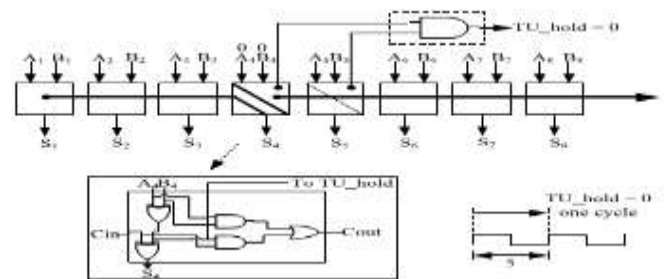


Fig-3: 8-bit RCA with Hold logic circuit

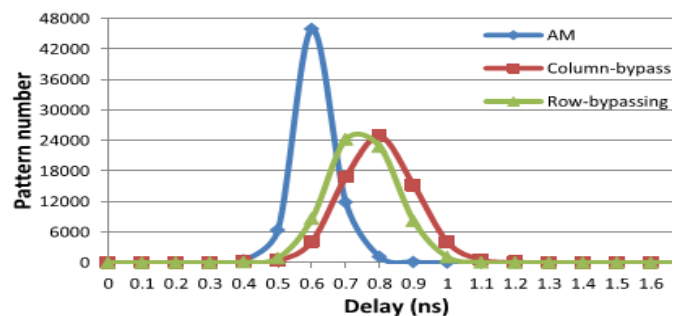


Fig-4: Path Delay Distribution of AM, Column and Row bypassing multipliers for 65536 input patterns

3. AGING AWARE RELIABLE MULTIPLIER

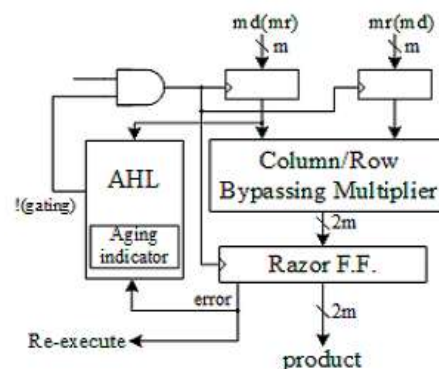


Fig-5: Aging Aware Reliable Multiplier

Fig 5 is an aging-aware multiplier factor design, which incorporates 2 m-bit inputs (m may be a positive number), one 2m-bit output, one column- or row-bypassing multiplier factor, 2m 1-bit Razor flip-flops Associate in Nursing d an AHL circuit. The inputs of row-bypassing multiplier factor square measure the symbols within the parentheses. In the planned design, the column- and row-bypassing multipliers will be examined by the amount of zeros in either the number or multiplier to predict whether or not the operation needs one cycle or two cycles to complete. When input patterns square measure random, the amount of zeros and ones within the multiplier and number follows a traditional distribution. Therefore, mistreatment the amount of zeros or ones because the judgement criteria leads to similar outcomes.

Hence, the 2 aging-aware multipliers will be enforced mistreatment similar design and therefore the distinction between the 2 bypassing multipliers lies within the input signals of the AHL. According to the bypassing choice within the column or row-bypassing multiplier, the input signal of the AHL in the architecture with the column-bypassing multiplier is the multiplicand, whereas that of the row-bypassing multiplier is the multiplier.

4. RAZOR FLIPFLOP

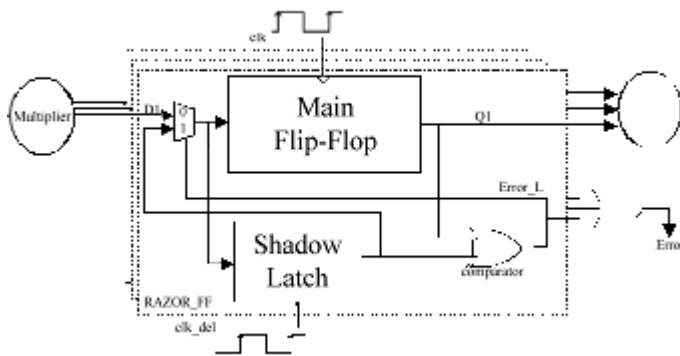


Fig -6:Razor Flipflop

Fig 6. is Razor flip-flops which are often accustomed sight whether or not temporal order violations occur before consecutive input pattern arrives. A 1-bit Razor flip-flop contains a main flip-flop, shadow latch, XOR gate, and mux. The main flip-flop catches the execution result for the mix circuit employing a traditional clock signal, and also the shadow latch catches the execution result employing a delayed clock signal, which is slower than the normal clock signal. If the barred little bit of the shadow latch is completely different from that of the most flip-flop, this suggests the trail delay of this operation exceeds the cycle amount, 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 re-execute the operation and notify

the AHL circuit that an error has occurred. We use Razor flip-flops to sight whether or not Associate in Nursing operation that's thought of to be a one-cycle pattern will extremely end in a very cycle.

If not, the operation is re-executed with two cycles. Although the re-execution may seem costly, the overall cost is low because the re-execution frequency is low. The AHL circuit is the key component in the aging-aware variable-latency multiplier. The AHL circuit contains an aging indicator, two judging blocks, one mux, and one D flip-flop.

The aging indicator indicates whether or not the circuit has suffered vital performance degradation because of the aging result. The aging indicator is enforced in a very straight forward counter that counts the {amount the quantity} of errors over a precise amount of operations and is reset to zero at the tip of those operations.

If the cycle amount is just too short, the column- or row-bypassing multiplier factor isn't ready to complete these operations with success, inflicting temporal order violations. These temporal order violations are going to be caught by the Razor flip-flops, that generate error signals.

If errors happen often 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'll output zero to point the aging result remains not vital, and no actions square measure required. The first decision making block within the AHL circuit can output 1 if the quantity of zeros within the number (multiplier for the row-bypassing multiplier) is larger than n (n is a positive number, which will be discussed in Section IV), and these Cond judging block in the AHL circuit will output 1 if the number of zeros in the multiplicand (multiplier) is larger than n + 1.

They are each utilized to make your mind up whether or not an input pattern needs one or 2 cycles, however only 1 of them are chosen at a time. In the starting, the aging result isn't important, and the aging indicator produces 0, so the first judging block is used. After a amount of your time once the aging result becomes important, the second decision making block is chosen. Compared with the primary decision making block, the second decision making block permits a smaller range of patterns to become one-cycle patterns as a result of it needs a lot of zeros within the number (multiplier).

5. ADAPTIVE HOLD LOGIC

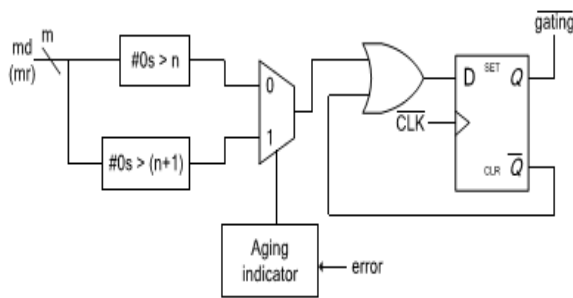


Fig-7: Adaptive Hold Logic

Fig 7 is an Adaptive hold logic when an input pattern arrives, both judging blocks will decide whether the pattern requires one cycle or two cycles to complete and pass both results to the multiplexer. The multiplexer selects one of either result supported the output of the aging indicator.

Then associate degree OR operation is performed between the results of the electronic device, and the Q signal is used to determine the input of the D flip-flop.

When the pattern needs one cycle, the output of the multiplexer is 1. The $\neg(\text{gating})$ signal will become 1, and the and the input flip flops will latch new data in the next cycle. On the other hand, when the output of the multiplexer is 0, which means the input pattern requires two cycles to complete, the OR gate will output 0 to the D flip-flop. Therefore, the $\neg(\text{gating})$ signal are zero to disable the clock signal of the input flip-flops within the next cycle.

Note that solely a cycle of the input flip-flop are disabled as a result of the D flip-flop can latch one within the next cycle. The overall flow of our planned design is as follows: once input patterns arrive, the column- or row-bypassing multiplier, and the AHL circuit execute simultaneously.

According to the number of zeros in the multiplicand (multiplier), the AHL circuit decides if the input patterns require one or two cycles. If the input pattern needs 2 cycles to complete, the AHL will output 0 to disable the clock signal of the flip-flops. Otherwise, the AHL can output one for traditional operations. When the column- or row-bypassing number finishes the operation, the result are passed to the Razor flip-flops. The Razor flip-flops check whether or not there's the trail delay temporal arrangement violation.

If temporal arrangement violations occur, it suggests that the cycle amount isn't long enough for the present operation to complete which the execution results of the multiplier factor is wrong. Thus, the Razor flip-flops can output a slip to tell the system that the present operation must be re dead exploitation 2 cycles to make sure the operation is correct. In this situation, the extra re execution cycles caused by timing violation incurs a penalty to overall average latency.

However, our planned AHL circuit will accurately predict whether or not the input patterns need one or 2 cycles in most cases. Only many input patterns might cause a temporal arrangement variation once the AHL circuit judges incorrectly. In this case, the extra re execution cycles did not produce significant timing degradation.

In summary, our planned multiplier factor style has 3 key options.

First, it is a variable-latency design that minimizes the timing waste of the noncritical paths.

Second, it will give reliable operations even when the aging result happens.

The Razor flip-flops discover the temporal arrangement violations and re execute the operations exploitation 2 cycles.

Finally, our design will regulate the share of one-cycle patterns to attenuate performance degradation thanks to the aging result.

When the circuit is aged, and many errors occur, the AHL circuit uses the second judging block to decide if an input is one cycle or two cycles.

6. MONTGOMERY ALGORITHM

Montgomery multiplication could be a methodology for computing $ab \text{ mod } m$ for positive integers $a, b,$ and m .

1.It reduces execution time on a pc once there are an outsized range of multiplications to be through with constant modulus $m,$ and with a tiny low range of multipliers.

In specific, it's helpful for computing Associate in Nursing $\text{mod } m$ for an outsized worth of $n.$ The number of multiplications modulo m in such a computation is reduced to variety considerably but n by in turn squaring and multiplying in line with the pattern of the bits within the binary expression for n ("binary decomposition"). But it will still be an outsized enough range to be worthy rushing up if potential.

The difficulty is within the reductions modulo $m,$ which are, primarily, division operations, which are costly in execution time. If one defers the modulus operation to the top, then the product can grow to terribly massive numbers, which slows down the multiplications and also the final modulus operation. To use Montgomery multiplication, we tend to should have the multipliers a and b but the modulus $m.$

We introduce another whole number r that should be larger than $m,$ and that we should have $\text{gcd}(r, m) = 1.$

The method, primarily, changes the reduction modulo m to a discount modulo $r.$ sometimes r is chosen to be Associate in

Nursing integral power of two, therefore the reduction modulo r is just a masking operation; that's, retentive the $\lg(r)$ low-order bits of Associate in Nursing intermediate result, and discarding higher order bits. If r could be a power of two, we have a tendency to should have m odd, to satisfy the gcd demand. (Any odd worth from three to $r - 1$ is suitable.)

The method: 1. realize 2 integers $-1 r$ and m' such one. one $- ' = rr -$ millimetre this could be done by the extended gcd algorithmic program. there's a binary extended gcd algorithmic program that will no divisions, and that simplifies considerably once one argument (r) could be a power of two and also the different (m) is odd. This simplified version of the algorithmic program is given below (C perform `xbinGCD`).

2. rework the multipliers to "Montgomery space" by multiplying them by r (a shift left operation if r could be a power of 2) and reducing the merchandise modulo m . That is, mod , mod , and $b r m a r m = =$ These area unit pricey operations, however they're done just one occasion per multiplier factor, and that they aren't done on the intermediate product of a sequence of multiplications.

3. Perform the Montgomery multiplication step. This operates on the remodeled quantities a and b , giving the merchandise of a and b in Montgomery area. That is, the result's $abr \text{ mod } m$. The multiplication tm' isn't too pricey as a result of the $\text{mod } r$ implies that solely the low-order $\lg(r)$ bits of the merchandise want be created. If the calculations area unit performed to some mounted length w bits, with $2, w r =$ then the opposite 2 multiplications area unit of the shape $w \times w \Rightarrow 2w$ bits and also the addition is of the shape $2w + 2w \Rightarrow 2w + \text{one bits}$ (it will overflow). once division by r (a shift), u is of length $w + \text{one bits}$. 4. Do the inverse transformation to convert the result to an ordinary integer: $\text{mod} . ab = \text{city} - 1 m$ allow us to currently derive step three on top of. We would like to reason $u = abr \text{ mod } m$. A 64-bit Implementation. Here we have a tendency to take an in depth consider AN implementation of Montgomery multiplication for arguments up to the computer's word size. For corporeality we have a tendency to take it to be sixty four bits. The modulus m can be as large as $2^{64} - 1$ and a and b can be as large as $m - 1$. We take $2^{64} r =$ this can be a 65-bit variety, however it are often handled while not nice issue.

Step 1: The GCD Operation Below could be a C operate for the binary extended gcd operation, simplified for the case within which its initial argument a could be a power of two and the second argument b is odd. It is a simplification of the rule obtainable on the net.

Step 2: Transform the Multipliers we have a tendency to should reason $a = ar \text{ mod } m$, and equally for b . Because $2^{64} r =$ there's no multiplication to try to. We should kind a 128-bit whole number that consists of a followed by

sixty four 0-bits, and compute the remainder of division of that quantity by m . Some machines have an instruction for that. For different machines, the C operate shown below could also be used. This is the "hardware division" algorithm of Hacker's Delight. Invoke it as follows, wherever the primary 2 arguments represent ar . All variables are 64-bit unsigned integers. `abar = modul64(a, 0, m)`

Step 3: Montgomery Multiplication This step deals with 128-bit integers, however no quite that. The computation $t = ab$ is multiplying 2 64-bit unsigned integers, giving a 128-bit product. Some machines have an instruction for that. For alternative machines, the C operate below could also be used.

Next, the subsequent expression should be evaluated:

$$u = (t + (tm' \text{ mod } r)m)/r.$$

Variable t could be a 128-bit unsigned number, and m' could be a 64-bit unsigned number.

Because of the "mod r ," only the low-order 64 bits of the product tm' is needed.

This means that the high-order half t is neglected, and sixty four \times sixty four \Rightarrow 64-bit multiplication is used.

The subsequent multiplication by m should be sixty four \times sixty four \Rightarrow 128-bit multiplication.

The addition of t should be $128 + 128 \Rightarrow$ 129-bit addition.

This can be through with $128 + 128 \Rightarrow$ 128-bit addition and one by one computing the carry, as shown within the code below (variable `ov`). This sum always ends in 64 0-bits, so the low-order part of the sum is computed only to produce a carry bit. Incidentally, if the low-order halves of the summands were better-known to be each nonzero, then the carry would be one, leading to a simplification.

However, the summand summands are often zero if either a or b is zero. finally (for step 3), we tend to should perform the computation: if $(u \geq m)$ then come $u - m$; else come u . Variable u could be a 65-bit number, in effect, as a result of the overflow mentioned on top of. however the ultimate results of the calculation could be a 64-bit number. If the addition of t overflowed, then actually $u > m$.

Otherwise, u and m could also be compared as 64-bit integers. The subtraction are often a 64-bit operation, as a result of it's glorious that when the subtraction, the sixty fifth little bit of the distinction are zero. A C operate for these computations follows.

Next, the subsequent expression should be evaluated: $u = (t + (tm' \text{ mod } r)m)/r$. Variable t could be a 128-bit unsigned number, and m' could be a 64-bit unsigned number. as a result of the "mod r ," solely the low-order sixty four bits of the merchandise tm' is required. This implies that the high-order half t are often neglected, and sixty four \times sixty four \Rightarrow

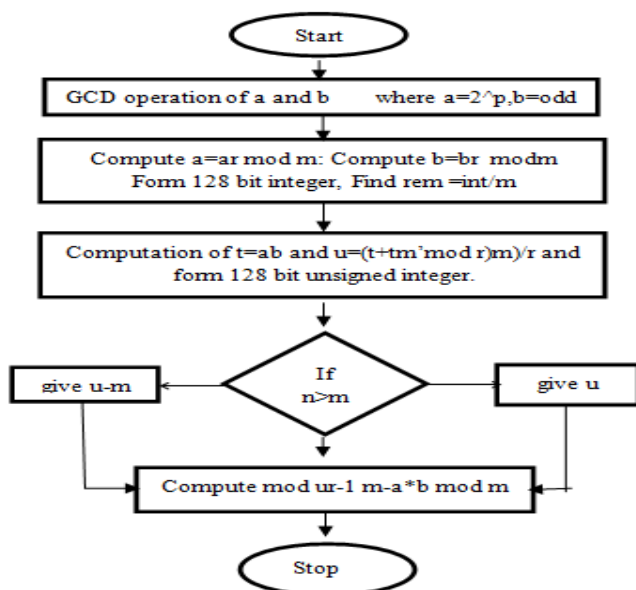
64-bit multiplication are often used. The next multiplication by m should be sixty four \times sixty four \Rightarrow 128-bit multiplication. The addition of t should be $128 + 128 \Rightarrow$ 129-bit addition. This may be through with $128 + 128 \Rightarrow$ 128-bit addition and severally computing the carry, as shown within the code below (variable ov). This add perpetually ends in sixty four 0-bits, therefore the low-order a part of the add is computed solely to supply a carry bit. Incidentally, if the low-order halves of the summands were glorious to be each nonzero, then the carry would be one, leading to a simplification.

However, the summands are often zero if either a or b is zero. finally (for step 3), we tend to should perform the computation: if $(u \geq m)$ then come $u - m$; else come u . Variable u could be a 65-bit number, in effect, as a result of the overflow mentioned on top of. However the ultimate results of the calculation could be a 64-bit number.

If the addition of t overflowed, then actually $u > m$. Otherwise, u and m could also be compared as 64-bit integers. The subtraction are often a 64-bit operation, as a result of it's glorious that when the subtraction, the sixty fifth little bit of the distinction are zero. AC operate for these computation follows.

Step4: The Inverse Transformation we tend to should reason , mod metropolis $-1 m$ that is that the product of a and b modulo m as normal integers. All variables area unit 64-bit unsigned integers. The multiplication should be done mistreatment sixty four \times sixty four \Rightarrow 128-bit multiplication, and also the modulo operation should be done mistreatment 128 / sixty four \Rightarrow 64-bit division (actually remaindering).

64-bit division (actually remaindering).



7. CONCLUSION

This paper proposed an efficient multiplier design with AHL using Montgomery multiplication algorithm. The multiplier is able to adjust the AHL to mitigate the performance degradation because variable latency multipliers have less timing waste, but traditional multipliers need to consider the degradation caused by both BTI effect and electro migration and use the worst case delay as the cyclic period. In this purposed architecture we have shown that, AHL with Montgomery Multiplication Algorithm will decrease the delay and improves the performance compared with previous design.

REFERENCES

1. SaiLakshmy, et.al, "Performance Analysis of Aging-Aware Multiplier Using Various Adders", International Conference on Communication and Signal Processing, April 6-8, 2016, India
2. P.KamilaParveen,et.al."Multiplier Design using MTCMOS with Adaptive Hold Logic" 2016 International Conference on Advanced Communication Control and Computing Technologies (ICACCCT).
3. Y. Cao. (2016). Predictive Technology Model (PTM) and NBTI Model [Online]. Available: <http://www.eas.asu.edu/~ptm>
4. S. Zafaret al., "A comparative study of NBTI and PBTI (charge trapping) in SiO2/HfO2 stacks with FUSI, TiN, Re gates," in Proc. IEEE Symp. VLSI Technol. Dig. Tech. Papers, 2016, pp. 23-25