International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

COPY RIGHT

IJIEMR Transactions, online available on 23rd February 2018. Link :
http://www.ijiemr.org/downloads.php?vol=Volume-7&issue=ISSUE-2

Title: Multiple Clock gating Schemes for Fused-MAC for Floating-Point Units.

Volume 07, Issue 02, Page No: 635 – 646.

Paper Authors

***RAYADU LAVANYA, S.DEVA KARUN.**

* Dept of ECE, Kakinada Institute of Engineering & Technology.

USE THIS BARCODE TO ACCESS YOUR ONLINE PAPER

To Secure Your Paper As Per UGC Guidelines We Are Providing A Electronic Bar Code

International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

MULTIPLE CLOCK GATING SCHEMES FOR FUSED-MAC FOR FLOATING-POINT UNITS

***RAYADU LAVANYA, **S.DEVA KARUN**
*PG Scholar, Vlsd, Dept of ECE, Kakinada Institute of Engineering & Technology For Women, Korangi, A.P.
**Assistant Professor, Dept of ECE, Kakinada Institute of Engineering & Technology For Women , Korangi, A.P.

ABSTRACT:
The paper introduces fine-grain clock gating schemesfor fused MAC-type floating-point units (FPU). Theclock gating is based on instruction type, precision andoperand values. The presented schemes focus on reducingthe power at peak performance, where each FPU stage isused in nearly every cycle and conventional schemes havelittle impact on the power consumption. Depending on theinstruction mix, the schemes allow to turn off 18% to 74%of the register bits. Even for the worst case instruction 18%to 37% of the FPU are shut down depending on the dataPatterns.

OVERVIEW:

Considering, the new floating-point standard IEEE 754-2008 [8] fused multiply-add (FMA) A·C+B is introduced as mandatory operation. The product is computed at full precision; rounding only gets applied when adding together product and addend. The first FMA-type floating-point unit (FPU) was introduced in 1990 [11] and since then many designs have been described in the literature [6, 9, 10, 12, 15]. The main focus of all those design was to make the FPUs faster, but very little has been said about how to make such an FPU power-efficient. In the last decade, the power consumption and the effort for cooling the processors and computer systems have become a major issue. In the embedded market and game console market, designers are fighting for every milli-Watt [16], and in the server business a big focus is put on green IT [7]. Even supercomputers are not just ranked by their FPU performance; the top-500 lists now also takes the power efficiency into account [5, 17]. The most common way for saving power is to shut-down pieces of the hardware when they are not used. An effective approach for a pipelined design is to clock gate register stages that are idle [16].

This paper describes how this mechanism can be applied to an FMA-type FPU, and that it is possible to shut down parts of the FPU even when the system is running at peak FPU performance. After an overview of the structure of an MAC design and its importance (Section 1.1) and in section 1.2 we provide the basic block diagram of FMAC based FPU.

Now we introduce the concept of clock gating (Section 1.2), we show how the standard clock gating schemes can be applied to such an FPU and which aspects need to be considered. We

International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

then introduce new clock gating schemes into FMA-type FPUs, as used in recent products. Those schemes are instruction based, precision based, and data based clock gating; Sections 2, 3 and 4 describe them in detail. For each of the schemes it is shown what percentage of the FPU can be shut down.

LITERATURE SURVEY:

1.1 Motivation

Now days, the demand for the high speed mobile wireless communications is rapidly growing. The Multiplier-Accumulator (MAC) operation is the key operation not only in DSP applications but also in multimedia information processing and various other applications. As mentioned above, MAC unit consist of multiplier, adder and register/accumulator.

In this project, we used 64 bit modified Wallace multiplier. The MAC inputs are obtained from the memory location and given to the multiplier block. This will be useful in 64 bit digital signal processor. The input which is being fed from the memory location is 64 bit.

1.11.1 Objective of project

A design of high performance 64 bit Multiplier-and-Accumulator (MAC) is implemented in this paper. MAC unit performs important operation in many of the digital signal processing (DSP) applications. The multiplier is designed using modified Wallace multiplier and the adder is done with carry save adder. The total design is coded with Verilog-HDL and the synthesis is done using Cadence RTL complier using typical libraries of TSMC O.18um technology. The total MAC unit operates at 217 MHz The total power dissipation is 177.732 mW.

1.11.2 Block Diagram

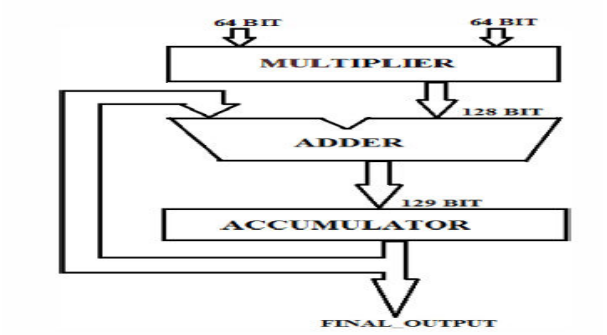


Fig 1.1Block diagram of MAC

1.12 Mac operation:

The Multiplier-Accumulator (MAC) operation is the key operation not only in DSP applications but also in multimedia information processing and various other applications. As mentioned above, MAC unit consist of multiplier, adder and register/accumulator. In this paper, we used 64 bit modified Wallace multiplier. The MAC inputs are obtained from the memory location and given to the multiplier block. This will be useful in 64 bit digital signal processor. The input which is being fed from the memory location is 64 bit. When the input is given to the multiplier it starts computing value for the given 64 bit input and hence the output will be 128 bits. The multiplier output is given as the input to carry save adder which performs addition.

The function of the MAC unit is given by the following equation

$F = I\ PiQi$

International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

The output of carry save adder is 129 bit i.e. one bit is for the carry (128bits+1 bit). Then, the output is given to the accumulator register. The accumulator register used in this design is Parallel-In Parallel-Out (PIPO). Since the bits are huge and also carry save adder produces all the output values in parallel, PIPO register is used where the input bits are taken in parallel and output is taken in parallel. The output of the accumulator register is taken out or fed back as one of the input to the carry save adder. As shown in fig 1.1 the basic architecture of MAC unit.

1.2 FMA Type Floating-point Unit BASED ON MAC DESIGN

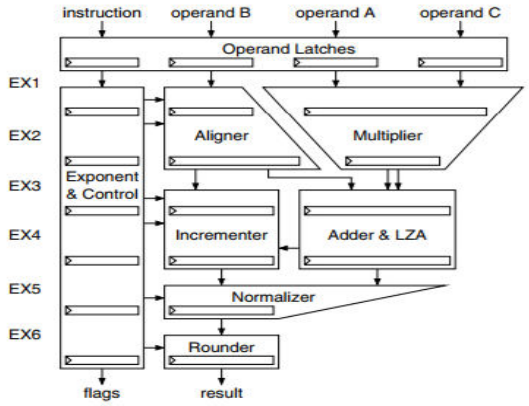
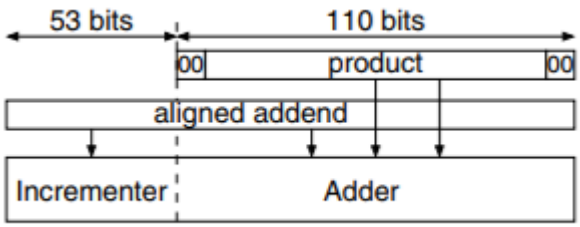


Figure 1: Representing the FPU

Figure 1 illustrates the basic structure of a state-of-theart, 6-cycle FMA-type FPU. The aligner, multiplier, adder, normalizer and rounder mainly operate on the mantissa of the operands. The exponent and sign information is processed in the exponent dataflow, which also holds the FPU control. The operand registers hold the operands; they also include logic for pre-processing the operands, such as unpacking the operands into sign, exponent and mantissa. The multiplier computes the partial products for A·C and compresses them into two product vectors. In parallel, the aligner aligns the mantissa of the addend to that of the product; this requires very wide shifts. The adder then computes the sum or absolute difference of the two product vectors and of the aligned addend. It also determines the number of leading zeros in the adder result using leading-zero anticipator logic (LZA). The normalizer then shifts out the leading zeros and the rounder rounds the intermediate result to the required precision. As described in [14], it suffices to use an aligned addendand intermediate results which are 3 times as wide as the precision of the operands plus a few extra bits. For double precision operands, the product padded with two bits at either side for rounding is 110 bits wide, and the aligned addend with its 163 bits sticks out 53 bits to the left of the product (Figure 2). In order to save hardware, an adder is used for the trailing 110 bits and an incremented for the leading 53 bits. Both include re-complement logic for subtraction. The leading-zero-anticipator is only needed for the trailing 110 bits. The position of a leading one in the incremented part can be derived from the aligner shift amount.



International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

Figure 2: Representing Adder split for a double-precision dataflow

Apart from FMA-type operations which include A·C+B and derivate like A·C-B and -A·C+B, FMA-type FPUs support various other floating-point instruction types, such as add, multiply, converts between integer and floating-point formats, compare operations, minimum and maximum function, and moves with potential sign manipulation. It also provides support for divide and square root. In some implementations, the FPU is also used for integer multiply and multiply-add operations.

In order to keep the FPU design simple and small, all these instructions are mapped onto the FMA dataflow and are executed as FMA with some corrections. Multiply A·C, for example, can be executed as A·C+0, and a subtract A-B can be executed as A·1-B. For the converts, the product exponent is forced to a special value and a correction is applied to the least-significant input bits of the adder. Estimate instructions need special hardware such as tables and reuse only small parts of the FMA pipeline. Divide and square root operations can be implemented as a series of estimates and FMA operations.

Multiple floating-point precisions are supported using an internal data format which is at least as wide as the largest supported precision. Input data are unpacked into the internal format; the result is rounded and packed into the desired result format. The packing and unpacking is independent of the executed instruction type. Thus, converts between different floating-point precisions can be treated as normalizing moves.

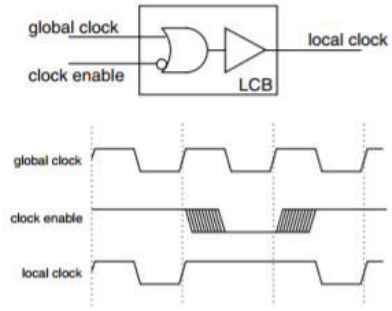


Figure 3:LCB with clock gating support (conceptual)

For each circuit of a design, power simulation tools can measure the switching power as a function of data switching factor on its data inputs (SF) and clock activity. Table 1 lists the switching power data for the 2-cycle aligner circuit of the presented FPU design; the estimated leakage power for the aligner contributes an additional 12 mW.

For this aligner circuit, there is a switching power reduction by more than a factor of 300 between worst and best case. Within each row and column an order of magnitude can be gained. Even if there is no switching at the inputs (SF=0), clock gating can reduce the switching power by about a factor 30. With high clock activity, the reduced switching factor still contributes to a substantial power saving.

Assuming peak performance, i.e., every stage of the FPU is used in every cycle and an optimistic switching factor of 30%, the switching power is five times larger than the leakage power. This indicates that adding additional logic can be a net power reduction if it enables to significantly increase the clock

gating. The sum of all registers controlled by the same clock gating function is called clock domain. Each LCB only accepts a single clock gate signal. In other words, all register bits connected to the same LCB are in the same clock domain. Hence, increasing the number of clock domains leads to an increasing number of LCBs. Since LCBs use a significant amount of power, the power improvement by splitting a clock domain has to exceed the penalty introduced by the additional LCB. For the state-of-the-art CMOS SOI technology that is used here, a clock domain should contain at least 8 to 10 register bits.

Timing puts another constraint to clock gating. As shown in Figure 3, the clock enable signals for the LCB have to be stable before the clock signal drops, to avoid glitches on the local clock net. The circuits computing the clock gating signals therefore have to be kept simple or use signals which are precomputed in the previous cycle.

The power consumption depends not only on the implemented clock gating scheme, but also on the data switching, chip technology and register type. For the sake of simplicity, in this paper we use the number of clocked register bits as a measure for switching power.

1.3. Clock gating Schemes

Clock gating schemes used in previous designs include unit based clock gating [4] and stage based clock gating [1]. Unit based clock gating turns off the functional units that do not execute any instruction and stage based clock gating turns off the pipeline stages that do not hold a valid instruction. Unit based clock gating [4] is targeted at functional units consisting of several sub-units like integer units consisting of adder, shifter, and logic unit. In an FMA-type FPU, all instructions use the same basic data path. Hence, with such a coarse-grain clock gating scheme, the whole FPU is either turned on or off and FPU switching power is only saved when no FPU instruction is in flight. Stage based clock gating [1] is a refinement of the previous scheme. Rather than whole units it activates each pipeline stage separately; only stages with valid instruction are active. This can also be applied to an FMA-type FPU, and then saves power for idle FPU cycles. This paper focuses on reducing the power of heavily used FMA-type FPUs, where each FPU stage is used in nearly every cycle. In this scenario even stage based clock gating has little impact on the FPU power consumption. We therefore introduce three new clock gating schemes that are implemented in addition to stage based clock gating. Instruction based clock gating [2] extends the idea of unit based clock gating. For this, we partition the FPU data path into blocks that can be turned off independently. The borders of the blocks are chosen carefully such that no side effects are introduced. The details of instruction based clock gating are discussed in Section 2.

In addition to the instruction type, the precision of the inputs and the outputs of an instruction can be used to reduce power. In a design that supports single and double precision, some blocks, e.g., parts of the multiplier, are not needed to compute single-precision results. Turning off blocks based on the precision of inputs or outputs is called precision based clock gating (Section 3). Section 4 describes data

International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

based clock gating [2]. In addition to the instruction type, this clock gating scheme also looks at the input data to predict which parts of the unit will not have any impact on the result and can be turned off.

EXIISTING TECHNIQUES:

2. Instruction Based

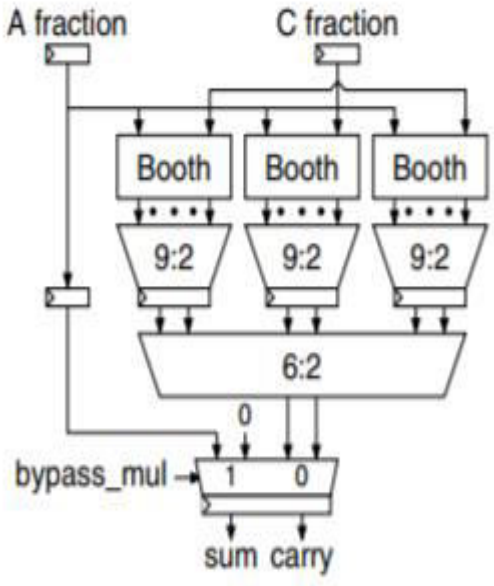
Clock gating for instruction based clock gating, the FPU is carefully partitioned into blocks that can be turned off independently. Some clock gating opportunities only arise if additional logic is introduced. In these cases the cost of the new logic has to be balanced with the power reductions gained by clock gating.

2.1. Multiplier Bypass

In high-frequency FPU designs, the multiplier is divided in two or more pipeline stages. That requires a lot of staging registers to store intermediate partial products and partial sums. In the presented 6-cycle FPU, the intermediate multiplier registers plus the operand register for multiplicand C account for over 25% of all register bits. Add-type instructions, for example, only use the multiplier to pass the A operand to the adder inputs by computing A·1. This wastes a lot of power. In order to turn off the intermediate registers of the multiplier, a multiplier bypass [3] is added to the FPU pipeline (Figure 4). This bypass can be used to either pass the fraction of A to the multiplier result or force the multiplier result to zero while the multiplier itself is clock gated. Since add-type instructions are not rare, the opportunity to disable about 25% of the registers bits leads to a considerable net power saving.

2.2. Fixed-Point Result

Bypass the instructions returning integer results, like float-to integer converts and integer multiply-add, require a significantly different post-processing of the intermediate adder result compared to instructions with floating-point results. Instead of normalization and rounding, the integer results require saturation. For timing reasons it is advantageous to create separate data paths for the two result types and merge the results in the final packing step (Figure 5). Any given instruction uses only one of the data paths and clock gatesthe other. In addition, the LZA which controls the normalizer shift amount can be clock gated for fixed-point results. Normalizer and LZA account for 6% of the registers and the fixed-point result logic for 2%.



International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

2.3. Partial Adder Bypass

The logic used for the addition of product and addend is split into an adder and an incremented circuit (see figure 1). Some instruction types need only one of these circuits. Move-type instructions, for example, need no addition at all; they can use the incremented to pass the operand down to the next pipeline stages. On the other hand, instructions using the fixed-point result bypass, only need the output of the adder; the incremented part is not needed. In order to save power it is therefore advantageous to activate the adder (20% of the total register bits) and the incremented (about 5% of the total register bits) independently. The only interaction between these circuits is the carry bit from the adder to the incremented. Hence, the incremented can be turned off without impacting the adder. In order to turn off the adder and use only the incremented, the carry needs to be forced to a desired value. Also, instructions with floating-point results must ensure that the normalizer does not read data from a disabled adder or incremented.

PROPOSED MODEL:

3. Precision Based Clock gating

The presented FPU pipeline supports double-precision (DP) and single-precision (SP) floating-point operations. Internally, all inputs are extended to a format with sign, 13- bit exponent, integer bit and 52-bit fraction. If the result of an instruction is an SP number the least significant bits of the adder and normalizer result are only used for the sticky bit computation. The sticky bit of the lower adder half is computed by the LZA. The sticky bit of the lower normalizer bits is pre-computed during the normalization for timing reasons. Hence, in case of an SP result, the lower half of the adder and normalizer result must not be computed. The latches only used to compute the lower result half's can be clock gated. Note that this does not include the latches needed for the carry network of the adder. When converting SP inputs to the internal format, the least significant bits of the operands are set to zeros. Hence, the 9:2 reduction tree of the multiplier that uses the least significant bits of the C operand computes $0 \cdot A = 0$. Instead of computing this with the reduction tree, the output of this tree is forced to zero and the intermediate registers are turned off. Since the lower half of the multiplier result is zero for SP inputs, it is possible to compute the sticky bit of the lower half of the adder result already in the aligner. This would allow to clock gate the lower part of the aligner output to the adder as well as the lower part of the LZA (which account for approximately 3.4% of the latches). This optimization would need modifications to the aligner sticky logic and the adder carry tree and was not implemented in the design. Table 4 lists the clock domains that need to be activated for floating-point multiply-add instructions with different precision. The clock domains MR, AD and NR are divided into subdomains HI and LO, where the domain HI contains the registers that are needed by any precision and LO contains the registers that are only needed for double-precision operations. The table indicates that precision based clock gating reduces the number of clocked register bits for SP instructions by up to 11.9%. The precision based clock gating in the multiplier canalso be

International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

used for fixed-point multiply-add instructions. Assuming that the fixed-point inputs are at most 32 bits wide, the multiplication of two inputs can be done without using the third 9:2 reduction tree. This reduces the clock activity of fixed-point multiply-add instructions by 7.9%.

4. Data Based Clock gating

In a double-precision, FMA-type FPU, the mantissa of the intermediate data is up to 163 bits wide. For floating-point instructions like multiply-add, multiply, and add, it depends on the operand values whether the full 163-bit wide intermediate data are needed or whether the operation can do with portions of the data vectors. This section details how to detect these data dependent cases early in the pipeline and how this can be used to reduce the number of clocked registers.

4.1. Special Inputs

If one of the inputs of an arithmetic operation is a Nan or infinity, the result of the operation is computed using logic in the incremented and the Nan forwarding logic in the NN domain. The normalizer is forced to select the output of the incrementer. If none of the operands is a Nan or infinity, the NN domain, accounting for 3.6% of the register bits, can be clock gated. The logic in the NN domain is not needed if at least one operand is infinity but no operand is a Nan. However, differentiating infinities from Nans requires a zero check on the fraction, whereas detecting Nan or infinity just requires an all-one check of the exponent. Turning off the NN domain for infinity-only operands would therefore increase the timing pressure on the activate signal for the NN domain. Since this case is assumed to be rare, it is not considered worthwhile; Nan and infinity cases are treated alike. For these cases, adder and leading-zero-anticipator can be gated; that are the domains AD and LZ which account for about 24% of the register bits. A multiply-add instruction where either the A or C operand is zero is treated like a move and the corresponding instruction based clock gating is applied. Otherwise, if the B operand is zero, the instruction is treated as a multiply. Similar optimizations apply for the multiply and add instruction types. Detection of zero operands requires nearly the whole first cycle, unless this information is already stored in the register file. The presented FPU has no special information in the register file. Thus, the zero operand information is too late for controlling the gating of the first cycle aligner and multiplier domains AL and MR. In case of zero product, adder and LZA are turned off (24% register bits), whereas for zero addend the incremented domain is gated (5% register bits).

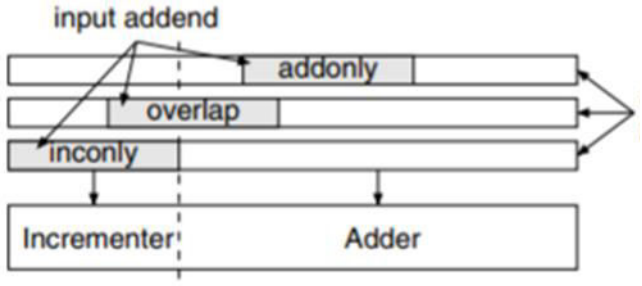
4.2. Operand Alignment

If all inputs are finite, non-zero numbers, the aligner shifts the addend fraction based on the exponent difference of addend and product. The aligned addend, which is the result of this alignment shift, is partitioned into an incremented part which is sent to the incremented and an adder part which is sent to the adder (Figure 2). The width of the aligned addend is limited to 163 bits by special handling of the big shift amounts. If the addend is shifted out on the right, all bits shifted out are collected in an aligner sticky bit. If the addend is shifted out on the left, the shift amount is

International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

ignored and the addend is forced into the incremented part of the aligned addend [14]. Based on the position of the input addend within the aligned addend, we distinguish three cases: in only, add only, and overlap (Figure 7). The case where the whole addend is placed in the incremented part of the aligned addend is called in only. Add only denotes the case where the input addend is fully contained in the adder part and the aligner sticky bit. The remaining case is called overlap. The case information is available in the middle of the second cycle, just in time for controlling the clock gating of the incremented and adder inputs.

In the addonly case, the incrementer part of the aligned addend consists of all zeros. Thus, the incrementer part of the addend does not contribute to the sum or absolute difference of product and aligned addend. The normalizer only needs to consider the adder output and normalize it based on the information provided by the leading-zero-anticipator. For this case, the incrementer domain IN can be turned off.

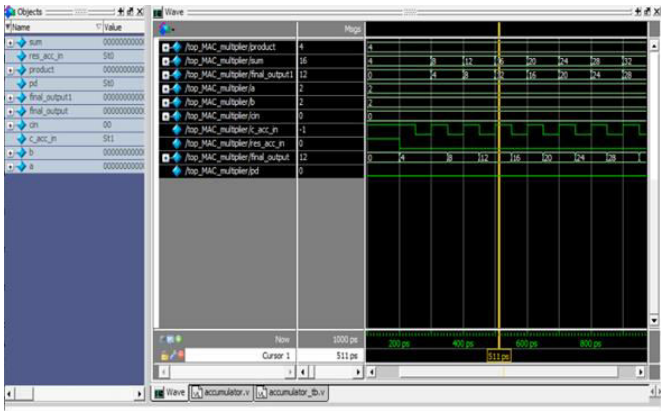


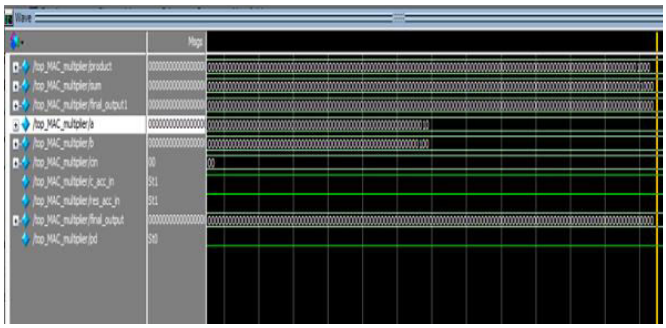
In the in only case, the addend is much larger than the product. The final result is either addend B or B ± up, depending on the effective operation, on the rounding mode, and whether the product is non-zero. The adder part is only needed to provide the carry to the incremented and to compute the sticky information for rounding. Since the adder part of the aligned addend consists of all zeros, the sticky bit is one in case of a non-zero product, i.e., if neither A nor C is zero. On the other hand, a carry from the adder to the incremented is only produced in case of an effective subtraction with a zero product. Hence, both signals can be derived directly from the operands without using the adder or multiplier result. Thus, in the in only case, adder and LZA can be turned off. In the overlap case, both the incremented and the adder are needed. However, since the leading one of the result is in the incremented part, the normalizer shift amount is derived from the aligner shift amount; the leading zero count of the adder result is not needed. The normalizer shift amount is at most 53, and therefore the lower half of the adder only contributes to the sticky bit (similar to SP results described in section 3). The sticky bit information can be provided by the lower half of the leading-zero-anticipator logic. Thus, the lower half of the adder result computation (excluding the carry tree which impacts the upper half) and the upper half of the LZA can be turned off.

Table 5 summarizes the impact of data based clock gating for a double-precision multiply-add instruction. Both the adder domain AD and the leading zero anticipator domain LZ are split up in an upper half HI and a lower half LO. In the overlap case, data based clock gating reduces the number of clocked registers by 8.3%. In the in only case, data based clock gating reduces the number of clocked registers even by 27.2%.

International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

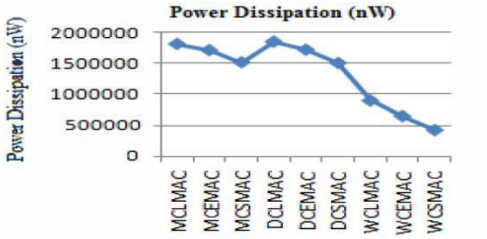
SIMULATION RESULTS:





1 Benefits

- The MAC unit operates completely independent of the CPU
- It can process data separately and thereby reduce CPU load.
- The application like optical communication systems which is based on DSP , require extremely fast processing of huge amount of digital data



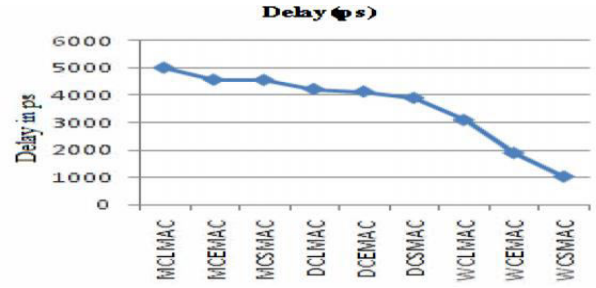


Fig.(a) power dissipation

Fig.(b) delay

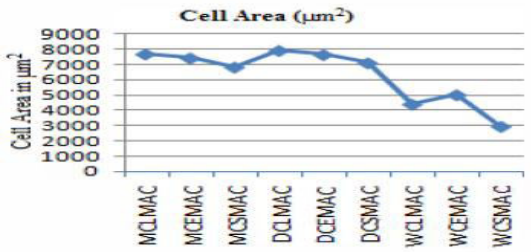


Fig. (c) cell area

CONCLUSION:

Traditional clock gating approaches reduce FPU power consumption if no instructions are executed, or at best, reduce the power consumption for the idle cycles between subsequent instructions. In numerical applications with highly optimized floating-point routines these traditional clock gating schemes are not efficient for the FPU. We have developed new clock gating schemes that address exactly this scenario, i.e., they save power even if the FPU executes an instruction every cycle. The schemes clock gate parts of the FPU based on instruction type, precision, and operand values. Table 6 lists for every instruction type and precision the minimum and maximum percentage of register bits that are enabled if the three clockgating schemes are

International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

applied. The percentage of active register bits is a good indication for the switching power.

	SP		DP	
	min	max	min	max
fma	54.3	73.2	63.2	82.1
mul	52.1	63.7	61.0	75.6
int-muladd	68.8			
add	38.6	57.5	39.6	58.5
round-to-int	35.3	54.2	36.3	55.2
int-to-float	51.1		52.1	
float-to-int	30.3	50.5	30.3	50.5
estimate	34.2	36.3	34.2	36.3
min/max	38.1		39.1	
move	32.9		33.9	
compare	25.7			

The table indicates that executing a workload in single precision (SP) instead of double-precision (DP) reduces the clock activity by up to 9%. It further shows that for compare operations 74% of the register bits can be disabled. Floating-point additions can clock gate at least 41% of the register bits and floating-point multiply instructions at least 24%. Even double-precision floating-point multiply-adds, which are the most energy consuming instructions, can shut off between 18% and 27% of the register bits. For these instructions, between 8% and 17% are disabled by data based clock gating; the remaining 10% are disabled by instruction based clock gating. Note that for area and power efficient designs, it is better to avoid extra hardware rather than to clock gate it. The goal is therefore to map all instructions efficiently on the FMA dataflow. The fact that only 10% of all register bits are never used for DP FMA operations is an indicator for the efficiency of the overall FPU design. In our case, extra hardware was spent for the estimate dataflow, fixed-point saturation logic, and the multiplier bypass. The latter was introduced since it allows add-type instructions to disable 25% of the FPU. Even in such an optimized FPU design, our clock gating schemes significantly reduce the switching power.

REFERENCES

[1] C. M. Abernathy, G. Gervais, and R. Hilgendorf. Method and apparatus for dynamic power management in an execution unit using pipeline wave flow control. United States Patent 7,137,013, November 2006.

[2] S. H. Dhong, S. M. Mueller, and H.-J. Oh. Power saving in FPU with gated power based on opcodes and data. United States Patent 7,137,021, November 2006.

[3] S. H. Dhong, S. M. Mueller, H.-J. Oh, and K. D. Tran. Power saving in a floating point unit using a multiplier and aligner bypass. United States Patent 7,058,830, June 2006. [4] M. A. Filippo. Clock control of functional units in an integrated circuit based on monitoring unit signals to predict inactivity. United States Patent 6,983,389, January 2006.

[5] Green500.org. The green500 list, June 2008. http://www.green500.org/lists.php. [6] T. N. Hicks, R. E. Fry, and P. E. Harvey. Power2 floatingpoint unit: architecture and implementation. IBM Journal of Research and Development, 38(5):525–536, 1994.

[7] IBM. Green IT, 2008. http://www.ibm.com/ibm/green/index.shtml.

[8] IEEE Task P754. IEEE 754-2008, Standard for FloatingPoint Arithmetic. Aug. 2008.

[9] R. M. Jessani and C. H. Olson. The floating-point unit of the PowerPC 603e microprocessor. IBM Journal of Research and Development, 40(5):559–566, 1996.

[10] T. Lang and J. D. Bruguera. Floating-point fused multiplyadd with reduced latency. Computer Design, International Conference on, 0:145, 2002.

[11] R. K. Montoye, E. Hokenek, and S. L. Runyon. Design of the IBM RISC System/6000 floating-point execution unit. IBM Journal of Research and Development, 34(1):59–70, 1990.

[12] S. M. Mueller et al. The vector floating-point unit in a synergistic processor element of a cell processor. In ARITH '05: Proceedings of the 17th IEEE Symposium on Computer Arithmetic, pages 59–67, Washington, DC, USA, 2005. IEEE Computer Society.