2009 19th IEEE International Symposium on Computer Arithmetic

Advanced Clockgating Schemes for Fused-Multiply-Add-Type Floating-Point Units

Jochen Preiss, Maarten Boersma, Silvia Melitta Mueller
IBM Deutschland Research and Development GmbH
Schoenaicher Strasse 220, 71032 Boeblingen, Germany
{preiss, mboersma, smm}@de.ibm.com

Abstract

The paper introduces fine-grain clockgating schemes for fused multiply-add-type floating-point units (FPU). The clockgating is based on instruction type, precision and operand values. The presented schemes focus on reducing the power at peak performance, where each FPU stage is used in nearly every cycle and conventional schemes have little impact on the power comsumption. Depending on the instruction 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 data patterns.

Keywords: clockgating, power reduction, fused multiply-add, floating-point hardware, IEEE 754 Standard

1. Introduction

In 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 clockgate 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 FMA-type FPU (Section 1.1) and introducing the concept of clockgating (Section 1.2), we show how the standard clockgating schemes can be applied to such an FPU and which aspects need to be considered. We then introduce new clockgating schemes into FMA-type FPUs, as used in recent products. Those schemes are instruction based, precision based, and data based clockgating; 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.

1.1. FMA Type Floating-point Unit

Figure 1 illustrates the basic structure of a state-of-the-art, 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 addend

1063-6889/09 $25.00 © 2009 IEEE
DOI 10.1109/ARITH.2009.17

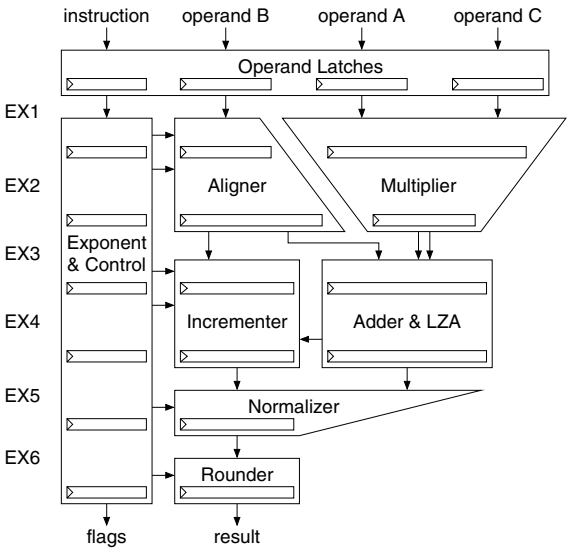

Figure 1. Basic floating-point pipeline

and 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 incrementer for the leading 53 bits. Both include recomplement logic for subtraction. The leading-zero-anticipator is only needed for the trailing 110 bits. The position of a leading one in the incrementer part can be derived from the aligner shift amount.

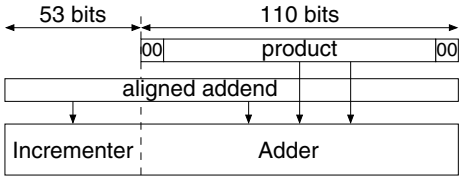

Figure 2. Adder split for a double-precision dataflow

Apart from FMA-type operations which include A·C+B and derivates 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.

1.2. Clockgating Concept

To reduce the switching power of the FPU, the number of transtitions needs to be reduced. An extensive description of techniques to avoid unnecessary transitions, concentrating on glitch reduction, is given in [13]. For a highly pipelined design like the FPU presented in this paper, the registers between the logic stages prevent glitches to propagate into the next cycle and limit total glitch power. This paper uses clockgating in order to reduce the number of transitions and glitches.

As opposed to the approach described in [18] where each flipflop can be clockgated individually, the FPU described in this paper uses registers that consists of multiple flipflops and a local clock buffer (LCB). The LCB has a local clock output, shared by all connected flipflops. Disabling a register can be done by gating the clock signal in the LCB, as described in [16]. A conceptual view of the LCB, together with a functional timing diagram, is given in Figure 3. It is assumed here that the registers are triggered by the rising clock edge.

When the clock enable signal is 1, the global clock is propagated into the local clock net; when the enable signal is 0, the local clock stops switching. It is important that the clock enable signal is stable when the global clock signal is low, to avoid glitches in the local clock net.

Clockgating reduces the power in three ways. First, reducing the clock activity will save power since the switching of the global clock net does not propagate into the local clock nets. Second, the local clock is quiet and the register content does not change; that saves the switching power of the register bits. Third, since the content of the register is unchanged, there is no switching at the outputs of the register. Hence, the switching factor in the subsequent pipeline stage will be zero.

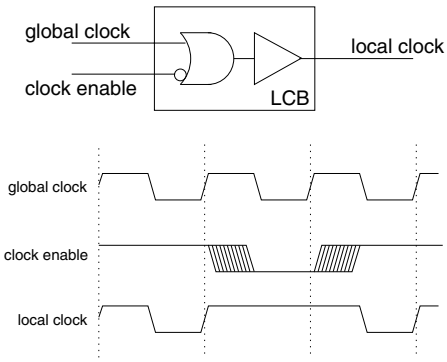

Figure 3. LCB with clockgating support (conceptual)

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.

	Clock Activity				
SF	0%	25%	50%	75%	100%
0 %	0.24	2.14	4.03	5.92	7.81
10 %	1.67	7.54	13.41	19.28	25.15
20 %	3.10	12.95	22.80	32.65	42.50
30 %	4.53	18.35	32.18	46.01	59.84
40 %	5.95	23.76	41.57	59.38	77.19
50 %	7.38	29.17	50.95	72.74	94.53

Table 1. Switching power (mW) as function of clock activity and switching factor for an FPU component

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), clockgating 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 clockgating.

The sum of all registers controlled by the same clockgating function is called clock domain. Each LCB only accepts a single clockgate 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 clockgating. 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 clockgating 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 clockgating 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. Clockgating Schemes

Clockgating schemes used in previous designs include unit based clockgating [4] and stage based clock gating [1]. Unit based clockgating turns off the functional units that do not execute any instruction and stage based clockgating turns off the pipeline stages that do not hold a valid instruction.

Unit based clockgating [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 datapath. Hence, with such a coarse-grain clockgating 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 clockgating [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 clockgating has little impact on the FPU power consumption. We therefore introduce three new clockgating schemes that are implemented in addition to stage based clockgating.

Instruction based clockgating [2] extends the idea of unit based clockgating. For this, we partition the FPU datapath 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 clockgating 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 clockgating (Section 3).

Section 4 describes data based clockgating [2]. In addition to the instruction type, this clockgating 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.

2. Instruction Based Clockgating

For instruction based clockgating, the FPU is carefully partitioned into blocks that can be turned off independently. Some clockgating 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 clockgating.

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 clockgated. 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 datapaths for the two result types and merge the results in the final packing step (Figure 5). Any given instruction uses only one of the datapaths and clockgates

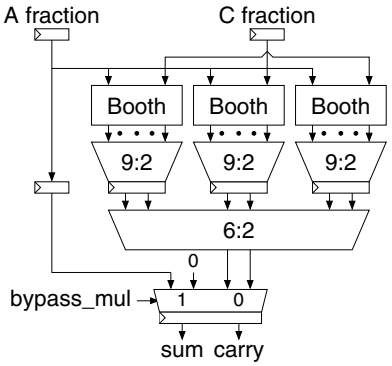

Figure 4. Multiplier bypass

the other. In addition, the LZA which controls the normalizer shift amount can be clockgated for fixed-point results. Normalizer and LZA account for 6% of the registers and the fixed-point result logic for 2%.

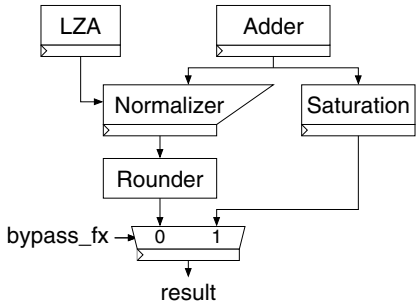

Figure 5. Fixed-point result bypass

2.3. Partial Adder Bypass

The logic used for the addition of product and addend is split into an adder and an incrementer 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 incrementer 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 incrementer 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 incrementer (about 5% of the total register bits) independently. The only interaction between these circuits is the carry bit from the adder to the incrementer. Hence, the incrementer can be turned off without impacting the adder. In order to turn off the adder and use only the incrementer, 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 incrementer.

For instructions which do not need the adder, the multiplier and the LZA can also be turned off. These three clock domains together account for 49% of the FPU.

2.4. Clock Domains

Table 2 lists all clock domains needed for the instruction based clockgating and the percentage of the overall register bits that belong to these domains. Figure 6 depicts the FPU dataflow with the main clock domains.

Name	%	Description
EC	22.5	Exponent, control, and result register
FA	1.6	A operand fraction
FB	1.6	B operand fraction
MR	25.3	Multiplier intermediate register
MB	1.7	Multiplier bypass
AL	1.8	Aligner staging register
AC	2.4	Aligner control
AD	20.2	Adder (including input register)
LZ	3.4	Leading zero counter
IN	5.4	Incrementer (including input register)
NR	2.6	Normalizer
FX	2.0	Fixed point result logic
NN	3.6	A and C operand (NaN) forwarding
ES	5.9	Reciprocal estimate logic

Table 2. FPU clock domains

The domain EC contains all exponent and control logic as well as the result registers. This domain is active for all instructions. Dividing the exponent or control logic into multiple clock domains would lead to very small clock domains per cycle. This would increase the overall power due to the added LCBs.

The fractions of the A and the B operand have separate clock domains (FA and FB). These domains are only active for instructions that use the respective operand. The C operand is only used for multiply-add type operations. Since these instructions always activate the multiplier, the fraction of the C operand is merged with the domain for the reduction tree of the multiplier (MR). As discussed in Section 2.1, the multiplier further contains a clock domain MB for the multiplier bypass.

The aligner contains several clock domains. AL contains the staging register for the fraction and AC contains the control logic used for the shift amount computation and special case detection. This logic is not needed if the aligner is only used to stage the B operand. The aligner output registers are in a separate domain.

The adder and incrementer domains (AD and IN) contain also the input registers to the respective circuits, i.e., the output registers of aligner and multiplier. The registers of the leading-zero-anticipator in the adder have their own clock domain LZ, since some instructions use the adder but not the LZA. The normalizer and the fixed-point result bypass (Section 2.2) are contained in the clock domains NR respectively FX.

Finally, there are two clock domains for special functions. The domain NN is used to stage the A or C operand for NaN forwarding and to stage the A operand of a minimum/maximum operation. The special logic for estimate instructions is summarized in the ES domain.

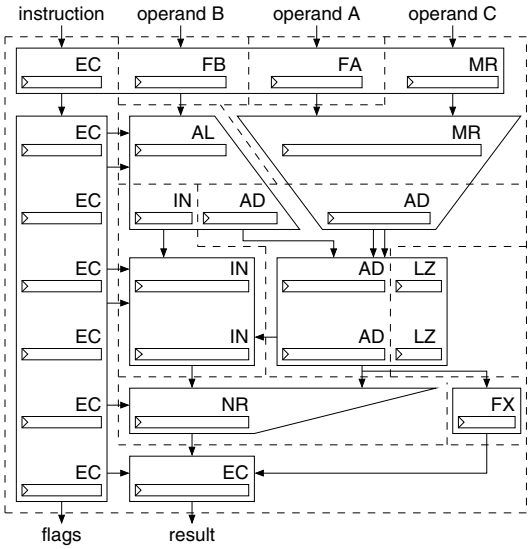

Figure 6. FPU dataflow with clock domains

2.5. Evaluation

Table 3 lists for each instruction type the clock domains which have to be activated. The instruction types are sorted by decreasing number of active registers.

Since the pipeline is primarily designed for floating-point multiply-add (fma) operations, it is no surprise that almost all clock domains are active for these instructions. Only the multiplier bypass MB and fixed-point-result bypass FX can be disabled, as well as the domain ES containing dedicated hardware for estimate instructions.

Note that every instruction that can propagate an input NaN needs the IN domain. This applies to each instruction with floating point operands and a floating point result. The NN domain is required in addition if the instruction has an A or C operand (fma, mul, add).

Floating-point multiply (mul) operations compute A·C+0 and have no B-operand. It is executed like a multiply-add, but since the addend is zero, the FB operand domain and the aligner domains AL and AC can be disabled. The aligner output register in domain AD is forced to zero.

	EC	FA	FB	MR	MB	AL	AC	AD	LZ	IN	NR	FX	NN	ES	Σ(%)
fma	×	×	×	×		×	×	×	×	×	×		×		90.4
mul	×	×		×				×	×	×	×		×		84.6
int-muladd	×	×	×	×	×	×		×				×			76.7
add	×	×	×		×	×	×	×	×	×	×		×		66.8
round-to-int	×		×			×	×	×	×	×	×				63.5
int-to-float	×		×			×		×	×		×				52.1
float-to-int	×		×			×	×	×				×			50.5
estimate	×		×			×	×			×	×			×	42.2
min/max	×	×	×			×				×	×		×		39.1
move	×		×			×				×	×				33.9
compare	×	×	×												25.7
Size (%)	22.5	1.6	1.6	25.3	1.7	1.8	2.4	20.2	3.4	5.4	2.6	2.0	3.6	5.9	

Table 3. Impact of instruction based clockgating

Fixed-point multiply-add (int-muladd) operations are executed similar to floating-point multiply-add instructions, with the difference that their result is a fixed-point number. As described in Section 2.2, the fixed-point result bypass (FX) is used instead of the regular floating-point result path (IN, LZ, NR).

Floating-point add-type operations (add) compute A+B or A-B. Instead of setting C to 1.0 and executing a multiply-add operation A·C+B, the multiplier domain MR is disabled and the multiplier bypass MB is used (Section 2.1).

The convert, estimate and move instructions only have a B-operand. Thus, all domains driven by the A or C operand can be disabled; that are the domains FA, MR, MB and NN. For the converts and moves, the route through the datapath depends on the datatype (int vs float) of operand and result. The estimate instructions, delivering an approximate value for $1/B$ or $1/\sqrt{B}$, are executed on the dedicated datapath ES.

Mininum and maximum instructions (min/max) use the NaN propagation datapath in domain NN to stage the A operand. After a selection is made between A or B, the instruction is executed like a move instruction.

For performance reasons, compare instructions are executed by an integer comparator connected to the outputs of the operand A and B registers. With that special hardware, the compare result is available after 2 cycles rather than after 6 cycles when using a floating-point subtraction operation. The compare operation only uses the control domain EC and the operand registers FA and FB.

3. Precision Based Clockgating

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 halfs can be clockgated. 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·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 clockgate 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 precison based clockgating reduces the number of clocked register bits for SP instructions by up to 11.9%.

The precision based clockgating in the multiplier can

	MR		AD		NR		Saving
	HI	LO	HI	LO	HI	LO	(%)
DP	×	×	×	×	×	×	0.0
SP	×		×		×		-11.9
Size (%)	17.4	7.9	17.2	3.0	1.6	1.0	

Table 4. Precision based clockgating for fma

also be 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 Clockgating

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 incrementer 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 clockgated.

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 clockgating 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 incrementer 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 incrementer part which is sent to the incrementer 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 ignored and the addend is forced into the incrementer part of the aligned addend [14].

Based on the position of the input addend within the aligned addend, we distinguish three cases: inconly, addonly, and overlap (Figure 7). The case where the whole addend is placed in the incrementer part of the aligned addend is called inconly. Addonly 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 clockgating of the incrementer and adder inputs.

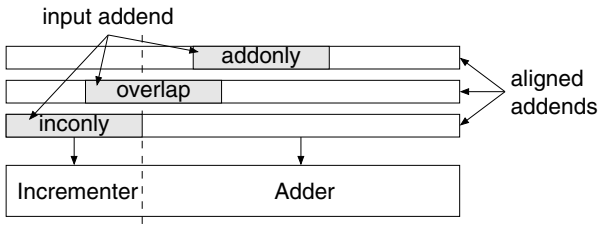

Figure 7. Alignment Cases

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 inconly case, the addend is much larger than the product. The final result is either addend B or B $\pm$ ulp, 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 incrementer 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 incrementer 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 inconly case, adder and LZA can be turned off.

In the overlap case, both the incrementer and the adder are needed. However, since the leading one of the result is in the incrementer 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 clockgating for a double-precision multiply-add instruction. Both the adder domain AD and the leading zero anticipator domain LZ are split up in a upper half HI and a lower half LO. In the overlap case, data based clockgating reduces the number of clocked registers by 8.3%. In the inconly case, data based clockgating reduces the number of clocked registers even by 27.2%.

	AD		LZ		IN	NN	Saving
	HI	LO	HI	LO			(%)
overlap	×			×	×		-8.3
addonly	×	×	×	×			-9.0
inconly					×		-27.2
Size (%)	17.2	3.0	1.7	1.7	5.4	3.6	

Table 5. Data based clockgating for DP fma

5. Conclusion

Traditional clockgating 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 clockgating schemes are not efficient for the FPU. We have developed new clockgating schemes that address exactly this scenario, i.e., they save power even if the FPU executes an instruction every cycle. The schemes clockgate 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 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			

Table 6. Clock activation (in %)

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 clockgate 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 clockgating; the remaining 10% are disabled by instruction based clockgating.

Note that for area and power efficient designs, it is better to avoid extra hardware rather than to clockgate 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 clockgating 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 floating-point 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 Floating-Point 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 multiply-add 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.

[13] A. Raghunathan, S. Dey, and N. K. Jha. Register transfer level power optimization with emphasis on glitch analysis and reduction. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 18(8):1114–1131, 1999.

[14] E. M. Schwarz, M. S. Schmookler, and S. D. Trong. Fpu implementations with denormalized numbers. *IEEE Transactions on Computers*, 54(7):825–836, 2005.

[15] P.-M. Seidel. Multiple path IEEE floating-point fused multiply-add. *Circuits and Systems, 2003. MWSCAS '03. Proceedings of the 46th IEEE International Midwest Symposium on*, 3:1359–1362 Vol. 3, Dec. 2003.

[16] O. Takahashi et al. The power conscious synergistic processor element of a CELL processor. *Asian Solid-State Circuits Conference, 2005*, pages 21–24, Nov. 2005.

[17] Top500.org. Top 500 supercomputer sites, June 2008. http://www.top500.org/lists/2008/06.

[18] Q. Wu, M. Pedram, and X. Wu. Clock-gating and its application to low power design of sequential circuits. *IEEE Trans. on Circuits and Systems I: Fundamental Theory and Applications*, 47:415–420, 2000.