Design and Implementation of an Embedded FPGA Floating Point DSP Block

Martin Langhammer
Altera European Technology Centre
High Wycombe, UK

Bogdan Pasca
Altera European Technology Centre
High Wycombe, UK

***Abstract*—This paper describes the architecture and implementation, from both the standpoint of target applications as well as circuit design, of an FPGA DSP Block that can efficiently support both fixed and single precision (SP) floating-point (FP) arithmetic. Most contemporary FPGAs embed DSP blocks that provide simple multiply-add-based fixed-point arithmetic cores. Current FP arithmetic FPGA solutions make use of these hardened DSP resources, together with embedded memory blocks and soft logic resources, however, larger systems cannot be efficiently implemented due to the routing and soft logic limitations on the devices, resulting in significant area, performance, and power consumption penalties compared to ASIC implementations. In this paper we analyse earlier proposed embedded FP implementations, and show why they are not suitable for a production FPGA. We contrast these against our solution – a unified DSP Block – where (a) the SP FP multiplier is overlaid on the fixed point constructs, (b) the SP FP Adder/Subtracter is integrated as a separate unit; and (c) the multiplier and adder can be combined in a way that is both arithmetically useful, but also efficient in terms of FPGA routing density and congestion. In addition, a novel way of seamlessly combining any number of DSP Blocks in a low latency structure will be introduced. We will show that this new approach allows a low cost, low power, and high density FP platform on current production 20nm FPGAs. We also describe a future enhancement of the DSP block that can support subnormal numbers.**

***Keywords*-FPGA; floating-point; single precision; DSP; CPA; subnormal**

I. Introduction

With increasing logic capacity and new embedded DSP features, the potential application space for FPGAs continues to increase. For some of these new applications, floating-point (FP) arithmetic is a requirement, either because the dynamic range of data, a higher accuracy requirement, or the complexity of the development process using fixed-point arithmetic. For instance, many new applications, such as matrix decompositions, require FP for numerical stability. FPGAs have been shown to outperform competing platforms in custom FP arithmetic [1], or when using exotic functions [2]. However, applications requiring large numbers of basic multiply-add operators will not find in FPGAs an ideal platform. Each core implementation will roughly take 10x the silicon area of an ASIC implementation (which is the technology gap between ASICs and FPGAs) which leads to poor performance and power results. Embedded FPGA FP units are required for FPGAs to be competitive in this space, but the commercial nature of general-purpose FPGAs means that the cost impact of the FP block needs to be minimal.

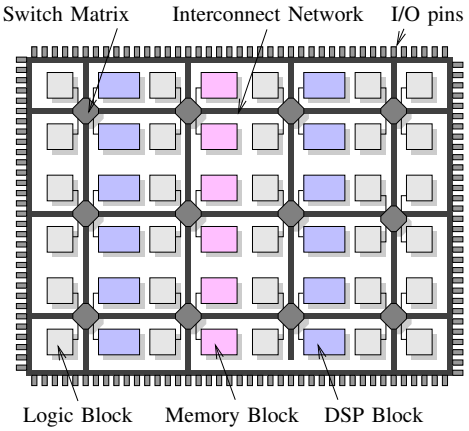
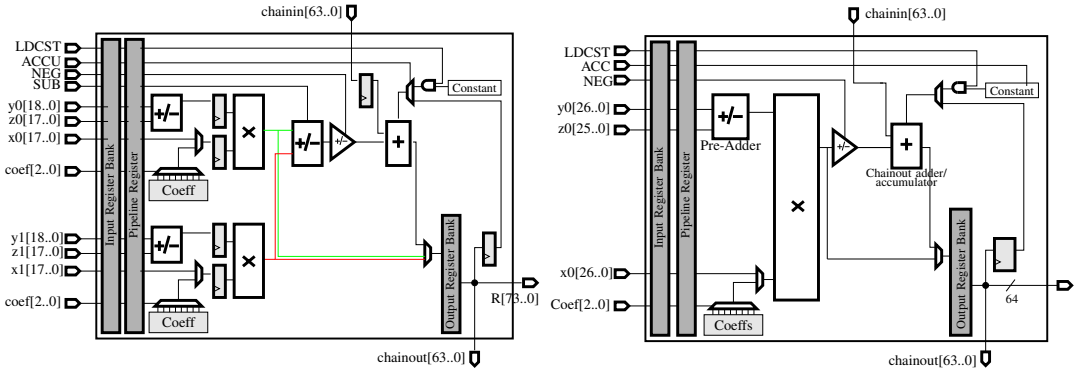


Figure 1. Simplified view of a modern FPGA architecture

It is increasingly expensive to design, develop, and support FPGAs throughout the tools ecosystem, so the FP features need to be cost-effective enough to be on every device – taking advantage of the economies of scale – but at same time, existing fixed point DSP features must still be supported. Therefore, FP has to fit into the same block as fixed point. One of the design challenges is that the DSP dimensions are relatively fixed at least in one direction (usually height), as they must be pitch matched to the regular logic and routing structure in the FPGA. Fig. 1 shows a simplified diagram of a modern FPGA, with a regular arrangement of logic, memory, and DSP resources, connected by a switched routing fabric. It is clear that a block of non-standard size or aspect ratio would have a significant impact on the layout of the device.

This paper describes the architecture and implementation of a DSP block that meets these goals by supporting single precision (SP) FP arithmetic. The FP multiplier is a superset of the fixed-point multiplier pipeline; the additional functionality required has a minimal area, and virtually no performance impact on the fixed point modes. An efficient, separate, FP adder is designed and added to be block; the challenge is to allow the FP adder and FP multiplier to work together, not only in one block, but across multiple blocks. This fixed and floating point DSP block has now been implemented and proven in a production FPGA [3].

1063-6889/15 $31.00 © 2015 IEEE
DOI 10.1109/ARITH.2015.18



(a) DSP block in dual 18x18 fixed-point multiplier mode

(b) DSP block in 27-bit fixed-point multiplier mode

Figure 2. DSP block in two fixed-point configurations

The rest of the paper is organized as follows. After briefly describing related works in Section II, Section III describes the functional architecture of the block, and shows some of the common use cases. Section IV focuses on the changes to the fixed point multiplier pipeline to efficiently support FP multiplication, as well as a brief description of the separate FP adder component. Section V describes an potential future enhancement of the FP DSP Block in order to support subnormal numbers, and how this is interesting in the context of the IEEE754-2008 standard. Finally, Section VI presents results, and Section VII our conclusions.

II. Related work

We will first briefly review the previous works in the area of FPGA FP functions. We will contrast these works against our proposed solution of adding SP FP arithmetic support (addition and multiplication) to the existing DSP block. Historically, there have been no commercial FPGAs that support FP directly, although there a number of published works propose solutions [4], [5], [6], [7], [8].

In [4] Beauchamp et.al. propose adding a dedicated double-precision (DP) FP block into the FPGA fabric with area and performance estimated from published microprocessor FPU data. In order to compare with our work, we scale their reported area using a 4:1 logic ratio (4 SP operators for 1 DP); the newly estimated area for a SP FP block is equivalent to over 300 LUT-register pairs – which is too large for an FPGA integration.

In [6], Ho, et. al. evaluate another FPGA DSP block, with the area and performance extrapolated from published IBM Power PC results. The area is smaller than the estimated area of [4], but again the proposed DSP block is a separate structure with considerable area compared to other common FPGA embedded blocks. In [8], Ho, et.al. update their earlier work with a more detailed analysis that also considers SP only results. Scaling their updated results to a 20nm mid-range FPGA, 1600 SP FP operators would require about 7% die area. While this is considerably improved over earlier work, this would still make the FP device a specialized ASSP.

In [7], Chong and Parameswaran design a DP FPU, which can be configured to support two SP multiplier and adder pairs. Their gate count is smaller than the equivalent estimates in [4], [6], [8], but the interface structure forces a particular aspect ratio which in turn increases the area to the range of the those estimates. Their proposal still suffers from the application specific nature of the FPU (although they also support some integer operations, standard fixed point DSP elements are still included in the FPGA), as well as only small number of the FPUs, largely because of the routing density effects.

III. Application architecture

Our work started with the StratixV DSP block [9] architecture. Although this block has many different modes that users can configure for various applications, we will only describe a subset of features and structures that we can use to build FP operators. The StratixV DSP block supports two 18x18 multipliers, depicted in Fig. 2(a) which can be used individually, or summed. DSP blocks can be chained together, for example to implement systolic FIR filters, using dedicated inter-block connections, which are more efficient than using general purpose FPGA routing. The 18x18 multipliers can also be combined to create a 27x27 multiplier, depicted in Fig. 2(b), which also can be used with the interblock routing. The proposed SP multiplier is based on the 27x27 fixed-point multiplier mode.

The exception handling and exponent calculation are relatively small and have little effect on area or performance. The critical part of the design is the round to nearest, tie breaks to even (RNE) implementation; the pedantic method requires a carry-propagate adder (CPA) after the carry-save adder (CSA) tree of the multiplier, followed by a 1-bit normalization, followed by another CPA for the rounding. The fixed-point modes only need the first CPA; the second CPA would have a significant area and delay impact on them.

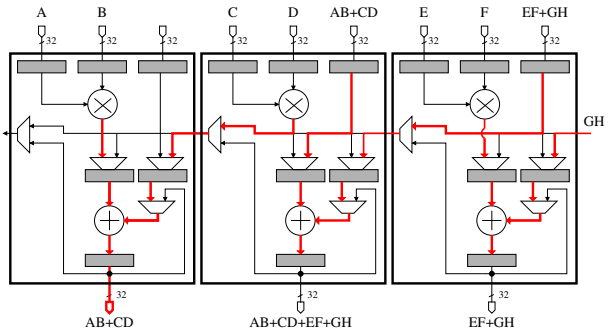


Figure 3. DSP block in Recursive Vector Mode

Although there are many known methods for combining the FP multiplier CPA and rounding CPA into a single structure, all of the many fixed-point modes still need to be supported. The multiplier pipeline CPA is therefore the most complex portion of the DSP block.

The FP adder has no logic or function in common with any of the legacy fixed-point functions, and is implemented as a separate block (FMA is not supported). In Fig. 3 showing multiple DSP blocks in FP mode chained together, one can observe the logical connections between the FP multiplier and adder, as well as the connections to and from adjacent DSP blocks.

As previously stated, some applications such as matrix decompositions require FP arithmetic for accuracy and handling the dynamic range. In contrast to traditional FPGA applications such as FIR filters, both the volume of processing (e.g.the size of the dot product for a large matrix calculation, for example), as well as the iterative nature of processing, require a much lower latency. Typically, FPGA implementations of FIR filters use a systolic, or feed-forward flow, which is proportional to one clock per input element; FP applications containing a long dot product (e.g. 256 elements) cannot wait for 256 cycles between successive intermediate results. A low latency, hardware reduction structure is therefore required.

The reduction structures are based on the adjacent DSP blocks being configured in different modes allowing to input operands from different sources. A logical block diagram, omitting many of the balancing registers for illustrative simplicity is shown in Fig. 3. The multiplier has a latency of two or three, with the first adder tree level adding an additional latency cycle. Every subsequent level of the adder tree adds three cycles of latency. The example 256 length vector will therefore have a latency of $3\lceil \log_2(\text{length}) \rceil$, or 24 clock cycles. Another pipeline stage is available in the multiplier (requiring the use of the hidden balancing registers for the first level of the tree). For large dot-product sizes and depending on placement, physical routing distance may grow so that frequency is affected. In such a case any depth of logic-based registers may be used at any point in the reduction tree, as long as the same number of registers are used for all soft routing connections at that level of the tree.

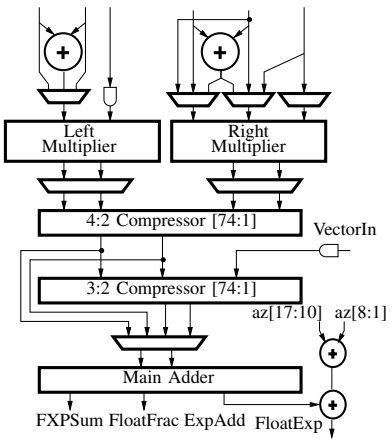


Figure 4. DSP block main datapath portion

IV. Arithmetic Datapath Architecture

There were a number of constraints on the implementation of the FP features: (a) full backwards compatibility with the existing fixed-point modes, (b) no performance degradation of the fixed-point modes while operating at similar frequencies for FP modes, and (c) only a minimal area increase was allowed. In order to maintain the structure and efficiency of the FPGA, a narrow range of X and Y dimensions are allowable for any embedded block included in the FPGA fabric, which further restricted the scope of the FP architectural changes.

A block diagram of the main datapath portion of the DSP block is shown in Fig. 4. Two smaller 18x18 multipliers, preceded by optional input pre-adders, can be used to implement independent multipliers, sums of multipliers, or larger 27x27 multipliers, using the multiplexers and compressors shown in the figure. The final CPA in the DSP block can be split in two in the case of individual multipliers.

To support all of the fixed-point modes and also FP multiplication, the final CPA was decomposed into a set of prefix structures aligned with both the fixed and FP boundaries. The three FP multiplier steps post compression: final sub-product reduction, normalization and rounding were combined into a single-level CPA using a flagged prefix approach [10]. Our flagged prefix approach, in order to support both fixed and FP calculations, uses a flagged prefix structure, overlaid on the carry-select structure, overlaid on a prefix adder structure. A simplified block diagram of the adder decomposition, without the prefix structures, is shown in Fig. 5.

The low adder boundary is the sticky bit boundary. The float/round block uses multiple intermediate values to make a rounding and normalization determination of the FP multiplication: the OR of the 22 result bits of the low adder (S), the MSB of the low adder (L[23]), the carry-out of the

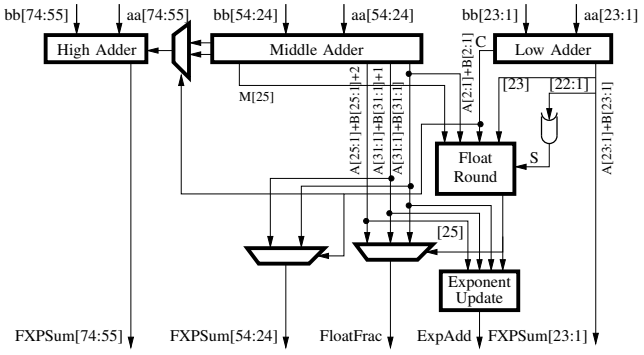


Figure 5. Simplified Final CPA Decomposition. The CPA may also be split in two (not highlighted here for readability) when the DSP block is used in fixed-point mode.

low adder (C), 2 LSBs of the middle adder (M[2:1]) and the MSB of the middle adder when the DSP is in FP mode (M[25]). The logic drives the select line of a 3:1 multiplexer (MXSEL) where the output sum is aa+bb+MXSEL. The selection logic is presented in Table I. When the middle adder is used for FP multiplication, a set of half adders is used to first insert a 0 into the LSB of the carry vector input to the middle adder. This is necessary for our flagged prefix approach (+1 and +2) implementation.

From Fig. 6, the two 32 bit sum and carry vectors from the half adders input a prefix network. The prefix network uses a Kogge-Stone architecture and is split two modules, which are required for the independent multiplier fixed-point modes. This prefix network generates two 32 bit carry and propagate vectors. Three values are needed for the FP multiplier calculation: the 24 bit sum of the inputs, the 23 bit sum plus one, and the 24 bit sum plus two. Depending on the output of the float/round block, along with the 25th bit of the sum, the lower 24 bits of the sum may be normalized by right shifting one bit. The 24 bits sum plus two will therefore become a 23 bit fraction rounded up. The calculation of the three pre-shifted values is shown in Table II.

Although only 25 bit outputs are shown, 32 bit results are calculated using the above methods for both aa+bb and aa+bb+1, as these values are also used for the fixed-point multiplications, as a carry-select adder topology spans across the three adder groups shown in Fig 5.

The mantissa of the FP multiplication is selected from the three outputs shown in Table II. There is a small amount of post computation to detect and apply error conditions and exceptions to the FP multiplier output, but this is trivial in effect compared with the bulk of the multiplication operation. Likewise, the exponent calculation is very small compared to the mantissa multiplication operation, and can be described behaviourally, and therefore implemented by the synthesis tool in the simplest possible structure.

The fixed-point multiplications, except for the mode of the two independent 18x18 multipliers, use a single adder

Table I
FLOAT/ROUND SELECTION LOGIC; X = MXSEL

M[25]	M[2:1]	C	L[23]	S	X
0/1	00	0	0	0	0/0
0/1	00	0	0	1	0/0
0/1	00	0	1	0	0/0
0/1	00	0	1	1	1/1
0/1	00	1	0	0	1/1
0/1	00	1	0	1	1/2
0/1	00	1	1	0	2/2
0/1	00	1	1	1	2/2
0/1	01	0	0	0	0/0
0/1	01	0	0	1	0/1
0/1	01	0	1	0	1/1
0/1	01	0	1	1	1/1
0/1	01	1	0	0	1/1
0/1	01	1	0	1	1/1
0/1	01	1	1	0	1/1
0/1	01	1	1	1	2/1

M[25]	M[2:1]	C	L[23]	S	X
0/1	10	0	0	0	0/0
0/1	10	0	0	1	0/0
0/1	10	0	1	0	0/0
0/1	10	0	1	1	1/0
0/1	10	1	0	0	1/2
0/1	10	1	0	1	1/2
0/1	10	1	1	0	2/2
0/1	10	1	1	1	2/2
0/1	11	0	0	0	0/1
0/1	11	0	0	1	0/1
0/1	11	0	1	0	1/1
0/1	11	0	1	1	1/1
0/1	11	1	0	0	1/1
0/1	11	1	0	1	1/1
0/1	11	1	1	0	1/1
0/1	11	1	1	1	2/1

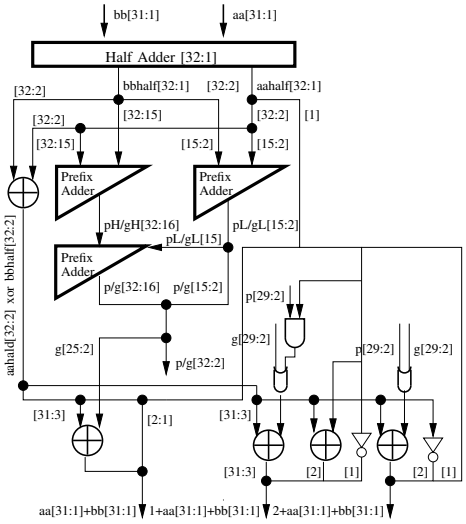


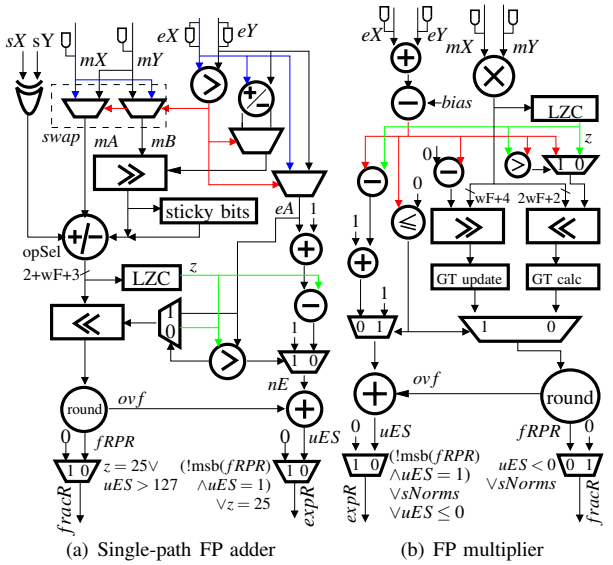
Figure 6. Middle Adder Flagged Prefix Structure

result that ranges in width from 38 to 64 bits, and therefore either spans the lower two, or all three adders of Fig. 5. The middle and high adders output both aa+bb and aa+bb+1 (the high adder also consists of a flagged prefix adder), which allows a carry select network to be constructed from the lowest to highest adder.

In the case of two independent 18x18 multipliers, one of the multipliers uses the low adder and the lower 14 bits of the middle adder, and the other multiplier uses the upper 17 bit of the middle adder and the high adder. The half adders in the middle adder are bypassed in this mode, and the two

Table II
FP CPA PRE-OUTPUTS: V0 = AA+BB; V1=AA+BB+1; V2=AA+BB+2

Val	Result[25:3]	Result[2]	Result[1]
v0	$aa_{half}^{[24:3]} \oplus bb_{half}^{[24:3]} \oplus g^{[25:2]}$	$aa_{half}^{[2]} \oplus bb_{half}^{[2]}$	$aa_{half}^{[1]}$
v1	$aa_{half}^{[24:3]} \oplus bb_{half}^{[24:3]} \oplus (g^{[23:2]} + (p^{[23:2]} \cdot aa_{half}^{[1]}))$	$aa_{half}^{[2]} \oplus bb_{half}^{[2]} \oplus aa_{half}^{[1]}$	$\neg(aa_{half}^{[1]})$
v2	$aa_{half}^{[25:3]} \oplus bb_{half}^{[25:3]} \oplus (g^{[24:2]} + p^{[24:2]})$	$\neg(aa_{half}^{[2]} \oplus bb_{half}^{[2]})$	$aa_{half}^{[1]}$



(a) Single-path FP adder (b) FP multiplier

Figure 7. FPAdd and FPMul with subnormal support. For single-precision wF=23.

prefix networks in the adder are separated.

The FP adder is a separate, monolithic structure, that does not share logic or functionality with any other operation in the DSP block. We used a dual-path architecture [11], with a textbook implementation that separates an otherwise long critical path consisting of alignment, operation and normalization into two paths: a *near path* used when inputs are within one exponent difference and have opposite signs (subtraction) and a *far path* used for the rest of cases. The near path critical path consists of a trivial alignment, a subtraction and a normalization stage whereas the far path consists of an alignment, an operation and a trivial normalization.

An important consideration for the adder is that it had to fit within the DSP block without affecting the height of the block, in order to maintain the regular structure of the FPGA. This required the adder to be malleable to fit around the deeper logic structures of the multiplier datapath, which was possible with a Verilog description of the adder.

V. Future work: subnormal support

Our implemented SP FP multiplier and adder designs in the Arria10 FPGA [3] flush subnormal inputs and outputs to zero. We made this choice out of two reasons. Firstly, in an FPGA context most available IP implementations do not support subnormals [12], [13], [14]. Moreover, one of the recent use models for FP on FPGAs – OpenCL [15] – does not require subnormal number support for achieving single-precision numerical conformance. Secondly, subnormal support in both the adder and multiplier would have too much of a impact on the DSP block size.

In this paper, we present a possible enhancement which allows supporting subnormal arithmetic at only a small increase in DSP block size. Our proposed solution consists of an enhanced FP adder which supports subnormal arithmetic but which can also function as an external subnormal handling structure for the preceding multiplier.

In the following we provide details on how the subnormal handling structures are built. We will use a single-path FP adder to illustrate the changes, but one should note that similar changes can be applied to a dual-path adder. For generality, hereforth we use the notation wF to denote the fraction width (23 bits for SP), and wE to denote the exponent width (8 bits for SP).

A. FP addition with subnormal support

In the implementation of a single-path FP adder, depicted in Fig. 7(a), the two mantissas (mX and mY) input the first set of multiplexers (implicit one is masked if input is subnormal). Their respective exponents are compared, the mantissa of the larger value is multiplexed to the left datapath (mA), and the smaller value is multiplexed to the right datapath (mB). Not shown is the comparison between the two mantissas, the result of which is used as a tie-break in case the exponents are equal; by ensuring that the left datapath value is always greater than the right datapath value, a positive output from the fixed-point adder/subtracter is ensured. On the right datapath, the smaller value (mB having $1+wF$ bits) is shifted right up to $1+wF+1$ positions resulting in $2wF+3$ bits (one integer bit, rest fraction). This wide fraction will be compacted to $wF+3$ bits by keeping a guard bit (G) in position 2^{-wF-1}, a round bit (R) right of G and a sticky bit (T) as an OR of the remaining wF bits.

The result of the addition/subtraction will always be positive, and will have will have 2 integer bits and $wF+3$ bits fraction bits, with the lower 3 bits: G, R and T.

The counted number of leading zeros z (maximum of 25 for $a-a$) and the exponent eA are used to drive the left shifter which is used for normalization. Unless dealing with subnormals, normalization requires aligning the leading one to the fixed-point position with weight 1 (implicit "1" position): either of a 1-bit right shift - if the mantissa sum ≥ 2, no shift if the sum $\in [1,2)$, or a left shift of at most wF positions when sum < 1. The right shift is avoided by aligning the sum to the leftmost position (weight 2). The left shift value is either eA if $z > eA$ or z otherwise. The exponent post normalization (nE) will be set to 1 if $z > eA$, else set to $eA+1-z$.

Finally, rounding will use updated values for the LSB (L), G, R and T in order to generate the rounding value $rnd = G(L+R+T)$, which is added to the L position. An overflow bit is used to increment the exponent; the exponent will reset to zero if the exponent value post overflow update $uES = 1$ and the MSB of the fraction post rounding is 0 (in other words the fixed-point fraction is smaller than 1 but the exponent is set to 1). The fraction is similarly reset for

$z = 25$ ($a - a$) or $uES > (2^{wE-1} - 1)$. Exception handling (not shown) generates the correct outputs for the special cases.

In this paper we propose enhancing this adder structure, motivated by the lack of subnormal handling support of the presented FP multiplier. The difference between this structure and previously reported implementations is therefore the ability of the FP adder logic to also be configured as a subnormal handling unit for the multiplier when required.

In a configuration where subnormals are supported during multiplication, the FP multiplier will use the following adder in the same DSP block for subnormal number support. In this use case, the adder is not available for its normal uses such as FP addition, accumulation, multiply-add, or the recursive modes shown previously. The arrangement of the adder as the multiplier subnormal handler is done at the configuration time of the FPGA and can be independently set for each DSP block in the device.

Not all use cases would require one FP adder per multiplier to support subnormals. Reduction under IEEE754-2008 can support extended range exponents so only the output of the reduction tree would have to implement subnormal support. This would, however require enhanced FP operators with IEEE754-2008 exponent range handling.

B. Subnormal number generation from the FP multiplier

The architecture of a FP multiplier with subnormal support is shown for reference in Fig. 7(b). If the adder is used as a subnormal handler for the Arria10 multiplier (Fig. 8), the original multiplier rounding and exception handling stages are bypassed and a number of signals are forwarded to the adder together with additional generated signals:

Forwarded signals:

- the pre-rounding mantissa (MidAdder[25:1])
- the pre-rounding low product bits (LowAdder[23:1])
- a modified (explained next) value of the exponent used for driving the subnormal stage shifters.

Generated signals:

- a negOrZero_exponent flag, which indicates that the 8 bit exponent contains a negative number (and not a large positive number) or that the product exponent pre-normalization is zero;

A subnormal number can only be generated in two cases: the product of two normalized numbers that underflows, or the product of a normalized and a subnormal which underflows. The product of two subnormal numbers will always generate zero due exponent underflow; this information is used to reduce the size of the LZC.

We next decompose the multiplication operation into its subcomponents – mantissa multiplication and exponent addition – and evaluate the cases which can generate subnormal outputs, based on the temporary exponent $erT = eX + eY - bias$ (eX, eY denote updated exponent value: 1 for subnormals), and the temporary mantissa $mR = mX \times mY$ (mantissas will have the implicit "1" masked if a subnormal).

Table III
ADDER SUBNORMAL NUMBER HANDLING USE CASES: *noz* - NEGORZERO_EXPONENT, *eRGen* - EXPONENT OUTPUT

Case	Mantissa	erT	noz	$eRGen$	$sVGen$
1	$1 < mR < 2$	≤ 0	1	1	$-erT$
2	$2 \leq mR$	< 0	1	1	$-erT$
3	$mR < 1$	$= 0$	1	1	0
4	$mR < 1$	< 0	1	1	$-erT$
5a	$mR < 1$, $erT \geq z$	> 0	0	$erT - z + 1$	z
5b	$mR < 1$, $erT < z$	> 0	0	1	eA

For the product of two normalized numbers with mantissas (mX and mY) $\in [1,2)$, the product mR (excluding exponent manipulation) is in $[1,4)$. A subnormal number may be produced in two cases. *Case 1*: $erT \leq 0$, $1 \leq mR < 2$ with $z = 1$ or *Case 2*: $erT < 0$, $mR \geq 2$ with $z = 0$. In both these cases negOrZero_exponent=1 and the generated exponent $eRGen = 1$. The right shift will align mR against position weight 2 (similar as for the adder) and the shift value is $sVGen = -erT$. The sticky (T) and guard bits (G) are generated prior of the right shifter and are updated after the denormalization right shift.

A subnormal output may also arise during the multiplication of a normalized number (mantissa $\in [1,2)$) with a subnormal (mantissa $\in [2^{-p},1)$), the mantissa of the product will be in $[2^{-p},2)$. When the $mR \in [1,2)$ and the $eRTemp < 1$ we apply the rules from *Case 1*. Otherwise the following cases are possible: *Case 3*: $erT = 0$, $mR < 1$ number of leading zeros $z \geq 2$, this requires negOrZero_exponent = 1; the fraction is already aligned. *Case 4*: $erT < 0$, $mR < 1$, negOrZero_exponent = 1, $z \geq 2$, the shift value $sVGen = -erT$, $eRGen = 1$. Finally, the most complex case *Case 5* arises when $erT > 0$, $mR < 1$. In this case a left shift is required (negOrZero_exponent = 0). The left shift can potentially create a normal result when $erT \geq z$; in this case $sVGen = z$ and the propagated exponent is $erT - z + 1$ (5a). A subnormal result is potentially generated when $erT < z$ with $sVGen = eA$ and $erT = 1$.

Fig. 8 shows the modifications to the single-path architecture to add subnormal support to an individual multiplier, as well as handle subnormal numbers in the addition/subtraction operation. Both the left and right side of the datapath are used when the adder supports subnormals; the left path and right paths are used independently when the adder is used to support subnormals for a multiplier.

If the adder is used for the subnormal support for a single multiplier then the select line for the mantissa swapping multiplexers will be forced to select from the left inputs.

In multiplier support mode (MSM) the right shifter will input the upper $2 + wF$ bits of the product $middAdd[25:1]$, a guard bit $G = lowAdd[23]$ and a sticky bit $T = Or(lowAdd[22:1])$. In adder mode a 0 is padded left of the implicit "1" position, and two zeros padded right of the

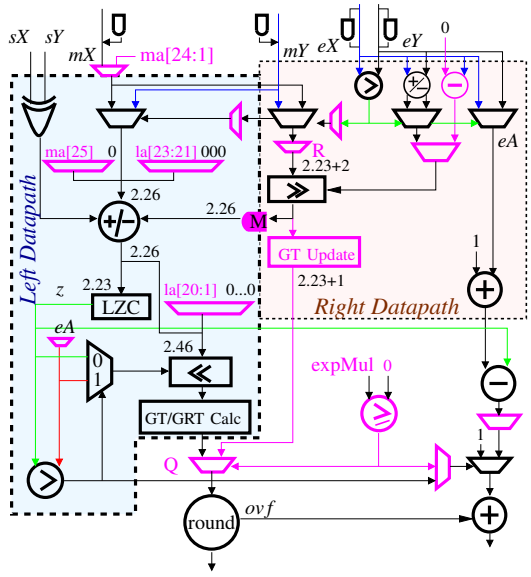


Figure 8. Single-path FP adder with subnormal handling and multiplier subnormal handling; ma – midAdder, la – lowAdder

LSB position. This selection is abstracted by multiplexer R in Fig. 8. The right shifted value (maximum right shift $1+wF+1$) will be reduced to $2+wF+3$ bits in adder mode (shifted out bits found in the sticky) and inputs the add/sub unit by the AND gate M. In MSM the M gate will output 0 to the add/sub unit; the right shifter output is reduced to $2+wF+1$ bits through the *GT Update* unit and is directed to multiplexer Q (selects among the two datapaths in MSM, or the left datapath in adder mode).

In the case of the FP adder, the left datapath shifts left a value on $2+wF+3$ bits, representing the sum of the mantissas. The result is reduced to $1+wF+3$ bits (lower bits are G, R and T). In MSM, the input to the shifter is a full $2+wF+wF$ mantissa, and the shifted value is reduced to $1+wF+2$ bits, to feed into the same multiplexer preceding the rounding block. The LZC is used on the top $2+wF$ bits of the mantissa.

Our proposed architectures do not output flags. However, when it comes to underflow signaling, one should also note that the IEEE754-2008 standard [16] allows two behaviours depending how tiny non-zero numbers are detected: tiny detection before rounding with unbounded exponent range and precision, or after rounding with unbounded exponent range. The presented architectures for adder and multiplier can be easily extended to support the underflow flag based on tiny detection tiny detection before rounding.

VI. Results

In this section we present the relative area increase required to support FP in the DSP block of a commercial production device. We also present estimated DSP block area increases for supporting subnormals, and show how our proposed solution in Section V can be a commercially viable solution in terms of area.

Table IV
Normalized silicon area for various cores

Core	Area
18x18 Multiplier	1
SP FP ALU (no subnormal) latency 3	0.9
SP FP ALU (subnormal) latency 3	1
SP FP Multiplier (no subnormal) latency 3	2
SP FP Multiplier (subnormal) latency 3	2.4

The percentage increase of our design (without subnormal support) for adding SP FP to the DSP block compared to the fixed-point-only implementation has two contributions:

- by reusing the existing fixed-point multiplier structures the FP multiplier requires additionally only exception and exponent handling, normalization and rounding (extending the final fixed-point multiplier adder into a flagged-prefix adder). The logic increase by supporting the FP multiplier block is only $\approx 3\%$.
- the FP adder is a stand-alone component. This allows us to optimize its implementation so it takes $\approx 10\%$.

Between these two contributions the final increase of the DSP block was around 15%, but the effective increase of the entire DSP cost to the FPGA architecture, considering the fixed cost of the existing routing interface, is only about 10%. Depending on the ratio of logic, memory, DSP blocks and transceivers in a particular device the DSP block area increase corresponds to a die area increase ranging between 0.5% and 5%.

To validate our assumptions about the cost of supporting subnormals we synthesized some fixed and FP commercial IPs from the Synopsys Design Ware library [17] for 20nm. We report the results, for the same synthesized performance, area normalized relative only to themselves, in Table IV.

We can see that the subnormal support for a FP multiplier is about half the size of an 18x18 multiplier or an SP FP ALU. This is in line with our expectations, as the subnormal support logic would be approximately the complexity of the near path of the FP ALU or the bottom half of a single path FP ALU (albeit the wider shifter). These results will not scale directly to our design, as this IP library has all pipelined components, while our FP ALU is combinatorial to support FP multiply accumulate; achieving the performance of the pipelined multiplier using only combinatorial logic would likely require a larger area.

The three main arithmetic components in our FP DSP block are two 18x18 multipliers and one FP ALU. We also need some arithmetic structures equivalent to about half an 18x18 multiplier to combine the two multipliers into one 27x27 multiplier, which is the basis of the FP multiplier. The two multipliers and the FP ALU are approximately the same area, with the FP ALU slightly larger because of the combinatorial area scaling explained above. The relative areas (our functions were all without subnormal support)

are similar to the relative areas shown in Table IV. A first order analysis shows that adding subnormal support to both the multiplier and the adder would increase the arithmetic component (other logic and routing is needed to support the configurability of the many options of the DSP block) of the DSP block by about 4%, which we felt was not an acceptable penalty for most of the applications we were targeting.

Adding subnormal support directly to the DSP block would cause other issues, such as the increased propagation delay in the last stage of the DSP block, which might negatively affect the supported fixed point applications. As our FP ALU is combinatorial, the additional area for the ALU might cause a non-linear increase in the area as well.

Our second proposed solution uses the FP adder as the subnormal handling unit for the FP multiplier. The implies an increase of roughly 10% in the FP adder according to Table IV together with another $\approx$ 5% required from our experiments for supporting the multiplier subnormal cases. Area alone results show that our proposed solution can provide subnormal support for both the FP multiplier and the adder (admittedly, not a the same time) for an extra DSP increase of $\approx$ 1%.

VII. Conclusion

The task of building an FPGA with hardware support for FP arithmetic is a research topic that was well addressed in several previous works. A significantly more difficult task is making these choices so that the FPGA is still viable for traditional FPGA fixed-point applications. Most previous approaches propose adding the FP units as a separate block, possibly grouping them in clusters for increased efficiency. However, when these blocks are unused the FPGA architecture performs sub-optimally both in terms of area as well as power. Our design adds FP capabilities to the existing DSP block by reusing and enhancing the fixed-point components such that the cost of FP support is a roughly 10% increase in DSP block area. The main challenges are keeping fixed-point performance unaltered, while providing similar FP performance, and meeting the area budget. For the FP multiplier we had to fuse the rounding stage together with an existing carry-propagate adder – used by the fixed-point modes – by using a flagged-prefix approach in order to reduce propagation delay. We also presented a possible future enhancement for adding subnormal support to the adder and multiplier. In contrast with adding support for both units, which would involve an additional 4% DSP increase, our proposed solution increases the block size by only 1% more by using the existing FP adder having subnormal support as a subnormal handling unit for the FP multiplier.

References

[1] F. de Dinechin, B. Pasca, O. Creţ, and R. Tudoran, "An FPGA-specific approach to floating-point accumulation and sum-of-products," in *IEEE International Conference on Field-Programmable Technology*. IEEE, 2008, pp. 33–40.

[2] F. de Dinechin and B. Pasca, "Floating-point exponential functions for DSP-enabled FPGAs," in *IEEE International Conference on Field-Programmable Technology*. IEEE, 2010.

[3] *Arria10 Device Overview*, 2014, http://www.altera.com/literature/hb/arria-10/a10_overview.pdf.

[4] M. Beauchamp, S. Hauck, K. Underwood, and K. Hemmert, "Architectural modifications to enhance the floating-point performance of FPGAs," *Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 2, pp. 177–187, Feb 2008.

[5] C. Ho, P.-W. Leong, W. Luk, S. J. E. Wilton, and S. Lopez-Buedo, "Virtual embedded blocks: A methodology for evaluating embedded elements in fpgas," in *Field-Programmable Custom Computing Machines, 2006. FCCM '06*, April 2006, pp. 35–44.

[6] C. H. Ho, C. W. Yu, P.-W. Leong, W. Luk, and S. J. E. Wilton, "Domain-specific hybrid FPGA: Architecture and floating point applications," in *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*, Aug 2007, pp. 196–201.

[7] Y. J. Chong and S. Parameswaran, "Configurable multimode embedded floating-point units for FPGAs," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 19, no. 11, pp. 2033–2044, Nov 2011.

[8] C. H. Ho, C. W. Yu, P. Leong, W. Luk, and S. J. E. Wilton, "Floating-point FPGA: Architecture and modeling," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 17, no. 12, pp. 1709–1718, Dec 2009.

[9] *StratixV Device Handbook*, 2011, http://www.altera.com/literature/hb/stratix-v/stratix5_handbook.pdf.

[10] N. Burgess, "The flagged prefix adder and its applications in integer arithmetic," *J. VLSI Signal Process. Syst.*, vol. 31, no. 3, pp. 263–271, Jul. 2002. [Online]. Available: http://dx.doi.org/10.1023/A:1015421507166

[11] M. D. Ercegovac and T. Lang, *Digital Arithmetic*. Morgan Kaufmann Publishers, 2004.

[12] *LogiCORE IP CORDIC v7.0*, 2013, http://www.xilinx.com/support/documentation/ip_documentation/floating_point/v7_0/pg060-floating-point.pdf.

[13] "Megawizard plug-in manager," http://www.altera.com.

[14] F. de Dinechin and B. Pasca, "Designing custom arithmetic data paths with FloPoCo," *IEEE Design and Test*, 2011.

[15] Khronos OpenCL Working Group, *The OpenCL Specification, version 1.0.29*, 8 December 2008. [Online]. Available: http://khronos.org/registry/cl/specs/opencl-1.0.29.pdf

[16] "IEEE Standard for Floating-Point Arithmetic," *IEEE Std 754-2008*, pp. 1–58, 29 2008.

[17] "DesignWare Library – Datapath and Building Block IP," https://www.synopsys.com/dw/buildingblock.php.

[18] P. M. Farmwald, "On the design of high performance digital arithmetic units," Ph.D. dissertation, Stanford, CA, USA, 1981.

[19] P.-M. Seidel and G. Even, "On the design of fast IEEE floating-point adders," in *Computer Arithmetic, 2001. Proceedings. 15th IEEE Symposium on*, 2001, pp. 184–194.

[20] ——, "Delay-optimized implementation of IEEE floating-point addition," *IEEE Trans. Comput.*, vol. 53, no. 2, pp. 97–113, Feb. 2004. [Online]. Available: http://dx.doi.org/10.1109/TC.2004.1261822