Extracting low-precision floating-point adders from embedded hard FP DSP Blocks on FPGAs

Bogdan Pasca
Intel Corporation, France
bogdan.pasca@intel.com

Martin Langhammer
Intel Corporation, UK
martin.langhammer@intel.com

***Abstract*—This work presents a set of techniques that allow implementing low-precision floating-point adders based on the embedded hard FP DSP Blocks available in contemporary Intel FPGAs. The presented architectures exploit the properties of these formats during exponent handling to obtain efficient implementations in terms of logic utilization. For instance, a half-precision floating-point adder implementation only requires 1 DSP Block and no extra logic. The newly available IEEE-754 compliant implementations can then be used as drop-in replacements in designs making use of these exact floating-point adder blocks. We present the case of a floating-point FFT implementation that benefits from these proposed architectures in order to substantially reduce logic utilization at the expense of using more DSP blocks.**

***Index Terms*—FPGA, DSP, floating-point, adder, extraction, mapping, half-precision, bfloat16, FP-DSP**

I. Introduction

Contemporary FPGA devices now include thousands of DSP Blocks in their architecture. The DSP Blocks have also evolved from fixed-point multiply-add-based functionality to include floating-point (FP) arithmetic support: single-precision (SP) multiply-add support was introduced in Arria 10 [1] and half-precision-based sum-of-products arithmetic has been included in Agilex devices [2]. In this work we are interested in generating half-precision (HP) adder architectures that utilize these novel FP DSP Block features. Our goal is to provide implementations that produce correctly rounded results for the round-to-nearest (tie breaks to even) rounding mode, but also for the relaxed accuracy faithful rounding mode. In all cases we diverge from the IEEE-754 2008 standard [3] by focusing on implementations that flush subnormals to zero on both input and output - which is a common practice in FPGA designs.

The primary goal is to provide better mappings to FP formats supported by the Agilex DSP Block (that supports several low-precision arithmetic modes), with a secondary goal of support architectures based on SP FP addition which can also be used on previous generation devices such as Arria10 and Stratix10.

The main contributions of these work are two classes FP adder architectures: (i) a specific HP adder based on the Agilex DSP Block, and (ii) low-precision adder implementation for 5-bit exponents on SP Hard FP DSP Blocks available in all contemporary Intel FPGA devices.

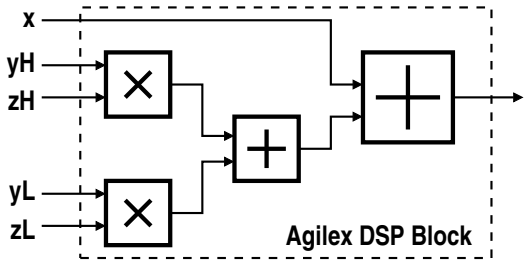

Fig. 1: Agilex DSP Block in low-precision FP mode

This paper is organized as follows: after a brief background on FPGA DSP Block features relevant to this work, we introduce an Agilex-specific HP adder mapping in Section III-A. Next, a second HP architecture is presented targeting SP DSP Blocks in Section III-B. We present area utilization numbers for these proposed architectures in Section IV to finally conclude in Section V.

II. Background

Here we summarize the main features of the Agilex DSP Block [2] . We mainly focus on the reduced-precision FP modes, which are the topic of the following sections. When configured in one of these modes the Agilex DSP includes a pair of low-precision FP multipliers, together with a low-precision FP adder that is used to sum these products. The output of the low-precision adder is then fed to SP adder which receives its second input either from general-purpose logic or from a neighboring DSP Block. The DSP block supports 3 reduced-precision modes: bfloat16 [4], HP, and bfloat16+ (a hybrid between bfloat16 and HP). Here we restrict our focus to the HP mode.

In HP mode, the 4 inputs (yH, zH, yL, zL) of the 2 HP multipliers arrive on 16 bits and encode FP values according to the binary16 representation. A DSP Block mode setting then allows to alter the behavior of the internal multiply-add operation. In *flushed* mode, HP subnormal inputs provided on (aH, bH, aL, bL) will be flushed to zero. The multipliers output a correctly rounded HP product which flushes subnormals to zero

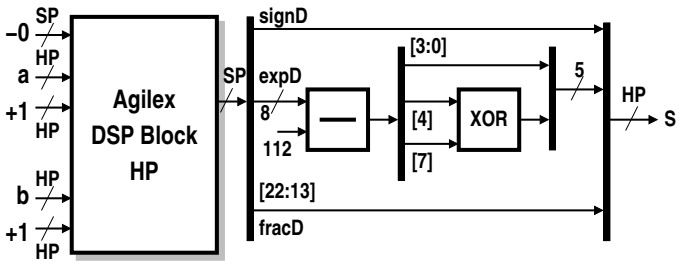

Fig. 2: Correctly rounded HP implementation based on the Agilex DSP Block

TABLE I: Legal exponent values for the HP addition

e_D^b	e_D^u	Binary	Class	e_D^b-112	Binary	Goal
255	-	1111 1111	Inf/NaN	143	1000 1111	1 1111
142	15	1000 1110	Regular	30	0001 1110	1 1110
141	14	1000 1101	Regular	29	0001 1101	1 1101
...						
128	1	1000 0000	Regular	16	0001 0000	1 0000
127	0	0111 1111	Regular	15	0000 1111	0 1111
126	-1	0111 1110	Regular	14	0000 1110	0 1110
...						
114	-13	0111 0010	Regular	2	0000 0010	0 0010
113	-14	0111 0001	Regular	1	0000 0001	0 0001
0	-	0000 0000	Zero	-112	1001 0000	0 0000

on output (after rounding). The subsequent HP adder is similarly configured to flush subnormals to zero on output. The result of the HP adder will then be converted to SP and provided as an input to the SP DSP Block output.

Similar to Arria 10 [1] and Stratix 10 [5] devices, the Agilex DSP Block can also be configured in SP multiply-add mode. In this mode too subnormals are flushed to zero on both input and output.

III. Architectures

A. Half-precision – Agilex specific

A HP adder is constructed starting from the DSP Block on Agilex devices by using the FP16MULTADD_SUM flushed mode. In this mode the DSP Block computes:

$$D = (yH \cdot zH + yL \cdot zL) + x$$

where yY, yL, zH and zL are all HP values, and x is in SP.

The desired operation is:

$$S = (a + b)$$

where both a and b are HP values. The first step is mapping this operation to the DSP Block mode:

- $yH = a$, $zH = +1$ (HP)
- $yL = b$, $zL = +1$ (HP)
- $x = -0$ (SP).

The mapping contains a subtlety which is that the value added to the sum of products is -0. This is required in order to propagate the correct sign for zero. For instance, if the sum of products $(yH \cdot zH + yL \cdot zL)$ returns -0, then if the value added on the the x input would be +0, the SP adder sum would be $+0$, which would be incorrect. Consequently, the value -0 is used so to preserve the sign of the second adder input.

The DSP Block output D is now a SP value containing a 10-bit populated fraction. The fraction is extracted directly from D[22:13]. However, the biased 8-bit exponent e_D^b[30:23] needs to be converted a 5-bit HP exponent.

For SP the exponents are stored biased:

$$e_D^b = e_D^u + 127.$$

For HP, the exponent stored is:

$$e_S^b = e_S^u + 15.$$

Knowing that $e_S^u == e_D^u$, in order to compute e_S^b from e_D^b one needs to subtract $112 = 127 - 15$ from e_D^b:

$$\begin{aligned} e_S^b &= (e_D^b - 127) + 15 \\ &= e_D^b - 112. \end{aligned}$$

We next check the effects of this exponent subtraction on the exception case encodings. For infinity and NaN $e_D^b = 255$. Upon subtracting 112 we obtain:

$$\begin{aligned} e_S^b &= 255 - 112 = 143 \\ &= 10001111 \text{ (in binary encoding).} \end{aligned}$$

For zero $e_D^b = 0$. Upon subtracting 112 we obtain:

$$\begin{aligned} \exp_S^{\text{biased}} &= 0 - 112 = -112 \\ &= 10010000\text{(in binary encoding).} \end{aligned}$$

In both cases the lower 4-bits of the representation already match the desired encoding, and bit 5 needs inverting.

For the general case, since the SP adder simply re-encodes the HP exponents, only a very small exponent range (corresponding to the HP unbiased exponent range [-14, +15]) will actually be used. These legal exponent values are depicted in Table I.

It can be observed that the lower 5 bits of $e_D^b - 112$ match exactly the desired output encoding shown in column *Goal* for all regular exponent values. Moreover, for the special cases the lower 4 bits also match the goal, but the fifth bit (highlighted in red) is negated. We use the fact that bit index 7 from the same subtraction will be 1 when these bits are flipped, and will be 0 for all regular exponent values. Consequently, we can use a XOR between bits index 4 and 7 of $e_D^b - 112$ to produce the 5th exponent bit. The architecture depicting these operations is presented in Figure 2.

The implementation may be further optimized based on the observation that the goal exponent representation can be composed from concatenating e_D^b MSB (index 7) with the four LSBs (index 3 to 0) to create the final exponent. This new architecture is depicted in Figure 3.

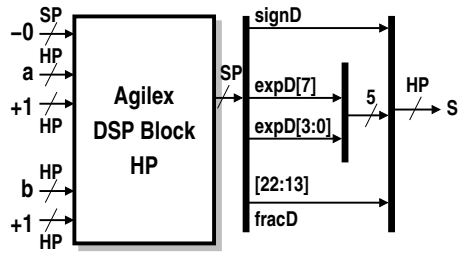

Fig. 3: Correctly rounded HP implementation based on the Agilex DSP Block using observation

B. Half-precision – SP Hard FP general solution

A generic HP adder architecture that utilizes the SP DSP Block present in Arria 10, Stratix 10 or Agilex devices involves in the first step transforming the HP inputs to SP. On the fraction side this is achieved by zero padding (to the right) – with a string of 13 zeros.

The exponent conversion may be implemented by means of a table lookup. A total of 8 LUT5s are required per exponent conversion, for a total of 4 ALMs (basic logic cell in modern Intel FPGA devices) per exponent. We denote by x_{HP} the HP variable, and by x_{SP} the SP value. Instead of performing a traditional conversion, where $x_{\mathrm{HP}} == x_{\mathrm{SP}}$, we will apply a conversion that allows us to benefit from some of the exception handling of the SP adder. In particular, we preserve the special case zero mapping and we map the entire HP range to the 'top' of the SP exponent range. The mapping is detailed in Algorithm 1 and has the advantage that it allows benefiting from the overflow exception handling of the SP FP adder.

The biased exponent range for HP is {0, 31} where the value 0 encodes the special case for zero (or subnormal if supported), and the value 31 encodes either infinity or NaN, depending on the mantissa contents. For SP, the biased exponent range is {0, 255}, with 0 and 255 similarly encoding the special cases. Our proposed mapping will map the range {1, 31} to {225, 255} and map {0} to {0}. For regular values the biased exponent difference will be 224 = 255 - 31, which corresponds to a classical exponent difference of 112.

Once the SP addition is performed, we can obtain correct rounding for RNE on 8-bit exponents and 10-bit fractions (generically the fraction can be anywhere up to 11 bit) by simply rounding the SP sum to HP. This can be guaranteed since the sum cannot be close to a HP midpoint when actual SP rounding (information loss) has happened. Next, we need to perform the 'reverse' mapping to the HP format. For the reversed mapping, we need to subtract 224 from the computed 8-bit exponent. Here we need to note that the minimum 8-bit exponent value that can be observed will be 224-10=214 (smallest subnormal). Instead of performing the subtraction, and applying potential fixes for the special cases, we revert again to a tabulation-based approach. For this we list in Table II the potential exponent values that are valid at the output of the FP32 adder.

TABLE II: Potential exponent values in the SP Addition

Exponent Value	Binary Encoding
255	1111 1111
...	111X XXXX
224	1110 0000
223	1101 1111
...	1101 XXXX
215	1101 0111
214	1101 0110 (smallest denormal, half)
0	0000 0000

We note that by observing the highlighted bottom 6 bits of this exponent, we can compute by tabulation the 5-bit HP exponent. Note that the only combination where all 6 bottom bits are '000000' corresponds to 0, since in binary the exponent values >= 214 will have the lower 6 bits > 0. Consequently, we can deterministically decide on the exponent update. Since we will fill-up a 64-element LUT6, we will in fact cover all exponents down to (biased) 128 + 64. Some of the input combinations will not be valid - and these are set to zero. The lookup table indexed by the 6 LSBs and which outputs the 5-bit exponent is populated in Algorithm 2.

The architecture diagram implementing this adder is depicted in Figure 4.

IV. Results

We have compared the synthesis results for the proposed architectures against state-of-the-art logic-based implementations available with the vendor tools. In particular, we have used the Intel Quartus Prime Pro 22.4 [6] to obtain the resource utilization post place-and-route and we have used the FP_FUNCTIONS Megacore IP [7] to generate the logic-based HP implementations. We note that we have used fastest speed-grade devices for this test – for both Agilex and Stratix 10, and we have set a target frequency of 500 MHz in FP_FUNCTIONS. For all comparisons we list the pipeline latency, number of ALMs, the number of DSP blocks alongside the *Ratio* column that tells how many ALMs are traded for a

ALGORITHM 1: LUT1: 5-bit to 8-bit exponents

```
Output: LUT1 (index by 5-bit exponent)
LUT[0] = 0;
for i from −14 to 16 do
    LUT1[i+15] = i + 255 - 16;
end for
```

ALGORITHM 2: LUT2: 8-bit to 5-bit exponents

```
Output: LUT2 (indexed by 6-bit exponent LSB)
for i from 0 to 63 do
    LUT2[i] = Max(0, i + 128 + 64 -224);
end for
```

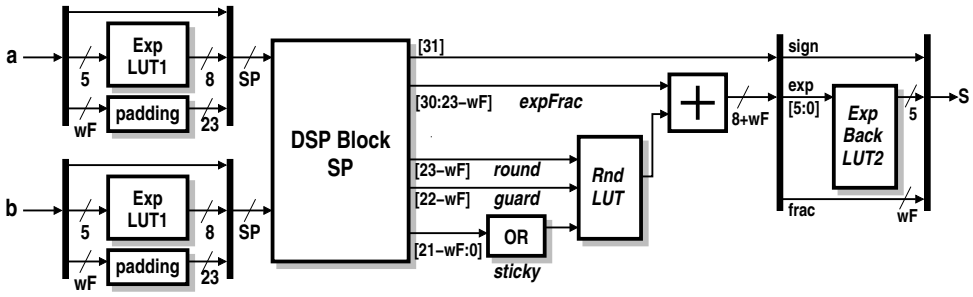

Fig. 4: Correctly rounded wE=5 wF<=11 implementation (HP included) based on SP DSP Blocks

TABLE III: Half-precision Adder Synthesis results

Arch	Target	Latency	ALMs	DSPs	Ratio
Proposed-A1	Agilex	5	5	1	147
Proposed-A2	Agilex	5	0	1	152
Logic	500MHz, -1	11	152	0	-
Proposed-B	Statix10	6	26	1	174
Logic	500MHz, -1	16	200	0	-

DSP (comparison made against the logic-only baseline implementation).

The comparison results are presented in Table III. First, for the Agilex device we have synthesized both architectures discussed in Section III-A: we denote by *Proposed-A1* the architecture depicted in Figure 2 and by *Proposed-A2* the architecture from Figure 3. As expected, A2 outperforms A1 since it uses absolutely no logic. They both provide a desirable tradeoff (152 ALMs / 1 DSP) which is important since recent devices now contain thousands of DSP Blocks.

In the case of the HP architecture based on a SP DSP Block the results are reported for a Stratix 10 device. It can be seen in this case that the extra logic used in Figure 4 amounts to 26 ALMs. This is still significantly less than the 200 ALMs reported for the logic-only HP implementation on this device. It must be noted that the ALM difference between the Agilex and Stratix 10 implementations (152 vs 200) is attributed to the latency difference (11 vs 16) which is a consequence of Agilex being faster than its Stratix 10 counterpart; in order to meet the same target frequency of 500MHz, more pipelining stages are required on the slower device thus increasing resource utilization.

Finally we report in Table IV resource utilization results for an 8K-point FFT design implemented in DSP Builder Advanced [8] from the a sample FFT design presented in Chapter 6.2.5. and which targets an Agilex device. The HP adders in this design use the Proposed-A2 architecture in Table III. It can be observed from this table that the ALM utilization has dropped by nearly 6K ALMs at the expense of 32 additional DSPs, for a ratio of approximately 187 ALMs per DSP. This is slightly higher than the reported ratio of 152 ALMs/DSP from Table III . This again is explained by the significantly lower latency

TABLE IV: 8K FFT Synthesis results on Agilex

Architecture	ALMs	M20K	DSPs
Old	8116	62	12
Proposed	2183	46	44

introduced by these adders which has a compound effect in a larger design (fewer synchronization registers are required between parallel paths in the design).

V. Conclusions

In this work we have shown how FPGA DSP Blocks supporting certain FP arithmetic functions can be used to implement HP FP adders using only a small amount of extra logic - if any. In particular, we have shown two sets of architectures: one targeting the sum-of-binary16 mode from Agilex devices, and a second targeting DSP Blocks that implement SP addition. For the first set of architectures, we have shown that HP addition can be implemented with no additional logic cost, whereas for the generic architecture we have shown that a total of 26 ALMs are required in addition to the DSP Block for the implementation.

References

[1] *Intel Arria®10 Device Overview*, 2018, https://www.intel.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/arria-10/a10_overview.pdf.

[2] *Intel Agilex Variable Precision DSP Blocks User Guide*, 2019, https://www.intel.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/agilex/ug-ag-dsp.pdf.

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

[4] Intel Corporation, "BFLOAT16 - Hardware Numerics Definition," 11 2018. [Online]. Available: https://software.intel.com/sites/default/files/managed/40/8b/bf16-hardware-numerics-definition-white-paper.pdf

[5] *Intel Stratix®10 GX/SX Device Overview*, 2018, https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/stratix-10/s10-overview.pdf.

[6] "Intel Quartus Prime Software," 2023, https://www.intel.com/content/www/us/en/products/details/fpga/development-tools/quartus-prime.html.

[7] "FP_FUNCTIONS Intel FPGA IP or Floating Point Functions Intel FPGA IP Core," 2023, https://www.intel.com/content/www/us/en/docs/programmable/683750/20-1/fp-functions-or-floating-point-functions-72394.html.

[8] "DSP Builder for Intel FPGAs (Advanced Blockset): Handbook," 2023, https://www.intel.com/programmable/technical-pdfs/683337.pdf.