2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines

Highly Versatile DSP Blocks for Improved FPGA Arithmetic Performance

Hadi Parandeh-Afshar and Paolo Ienne
Ecole Polytechnique Fédérale de Lausanne (EPFL)
{hadi.parandehafshar and paolo.ienne}@epfl.ch

Abstract

Integrating DSP blocks into FPGAs is an effective approach to close the existing gap between FPGAs and ASICs. A much wider range of applications could benefit from DSP blocks if they were more versatile than those currently found in commercial devices. In this paper we propose a novel DSP block which resembles commercially available ones and yet additionally supports a wide variety of multiplier bit widths as well as multi-input addition with negligible overhead. The novel DSP block uses much more efficiently the limited available input/output bandwidth. Experimental results show that on average the area overhead of the novel features added to a base design is a mere 3% with practically no delay penalty. Moreover, for multi-input addition, which is not supported in current DSP blocks, the proposed DSP block is more than 50% faster on average compared to FPGA soft logic.

1. Introduction

One fundamental step to bridge the gap between FPGAs and ASICs for arithmetic-dominated circuits is to integrate the DSP blocks that perform commonly-used arithmetic functions into an FPGA. The current trend in FPGA design is to add as many features as possible to the DSP blocks to improve a wider range of applications. The challenging issue is that the new features should not impose a considerable overhead to the original DSP block.

The most important feature of the DSP blocks in current FPGAs is multiplication. Most DSP blocks have a fixed bit-width multiplier as the base and other few multiplication bit-widths are formed on top of the base multiplier. For instance, the DSP blocks in Altera Stratix-II only support 9-bit and 36-bit multiplier on top of the base 18-bit multiplier. Moreover, despite the availability of IO bandwidth, when a DSP block is configured as a 36-bit multiplier, no other resources remain for other functionalities.

In this paper, first we present a base architecture for the DSP block, which consists of Partial Product Generator (PPG) and Partial Product Reduction (PPR) units. To support additional features such as various multiplication bit-widths as well as multi-input addition, we make the PPG unit reconfigurable and keep the PPR unit almost the same. This way the PPR remains small and fast, which is ideal for multi-input addition. However, the PPG unit can get complicated, since it should provide a different set of inputs for the PPR based on DSP block configuration. We propose a number of techniques, which significantly reduce the costs of making PPG reconfigurable.

The paper structure is as follows. Related work and preliminary arithmetic concepts are discussed in sections 2 and 3 respectively. Section 4 presents the architecture of the proposed base DSP block. Sections 5 and 6 describe how new features are added to the base DSP block. Experimental results are presented in section 7 and the final section is conclusion.

2. Related work

Two architectures [1,2] have been proposed in the past for FPGAs to improve multi-input addition. In both architectures, an accelerator for carry-save arithmetic is proposed which is intended to be placed in an FPGA as a DSP block. The problem with these two architectures is that they can only implement multi-input addition. To perform multiplication, one can use the FPGA soft-logic for the PPG and the proposed architectures for the PPR. The first drawback is that only a few multipliers can be implemented in each DSP block due to the input bandwidth and resource constraints. Moreover, such multipliers have poor performance and area utilization, since the PPG is generated by the soft logic and general routing network is used to connect PPG and PPR units.

In a subsequent work [3] by the same group, the PPG was integrated into the accelerator. This removed the second drawback, but still only two 9-bit multipliers can be implemented in the DSP block,

978-0-7695-4056-6/10 $26.00 © 2010 IEEE
DOI 10.1109/FCCM.2010.42

while one can fit nine 9-bit multipliers to the DSP block of Stratix-II.

There are several academic proposals [4-8] to improve the scientific applications on FPGAs. Such applications require floating-point arithmetic and to implement floating-point arithmetic operations on an FPGA, a large amount of resources are required. Therefore, the primary goal in these proposals is to integrate the floating-point functionality into DSPs, whereas our goal is different and complementary. Our contribution is to propose a base DSP block in which we can add various multiplier bit-widths as well as multi-input addition with small overhead.

3. Arithmetic preliminaries

In this section, we explain a number of arithmetic concepts such as Radix-4 Booth multiplication and compressor tree that are used in this paper.

Radix-4 Booth [9] multiplication is a standard technique to design smaller and faster multipliers, by recoding the numbers that are multiplied. By Radix-4 Booth recoding, the number of Partial Products (PP) is reduced by half. The basic idea is to take every second column, and multiply by ±1, ±2, or 0, instead of shifting and adding for every column of the multiplier term and multiplying by 1 or 0. To Booth recode the multiplier term, the bits in blocks of three are considered, such that each block overlaps the previous block by one bit. The overlap is necessary so that we know what happened in the last block, as the MSB of the block acts like a sign bit.

To compute the sum of two integers, a *carry-propagate adder (CPA)* [9] such as ripple-carry and carry-select adders is used. To add more numbers, a *compressor tree* [9] is used. The building block of compressor tree can be either a counter or a compressor. Compressors are, in principal, similar to counters, but in contrast, they have explicit carry inputs and outputs. Each compressor tree may consist of several layers of either counters or compressors. However, no ripple carry propagation occurs within the same layer of compressor trees. Only, a final CPA is required to compute the final sum.

4. Proposed base architecture of DSP

The fundamental building block of our base DSP block, similar to DSP blocks of *Altera* FPGAs, consists of two paired 18-bit multipliers followed by an optional adder stage. The adder unit is mainly used for complex arithmetic multiplication [12]. We proposed to use Radix-4 Booth architecture for the multipliers.

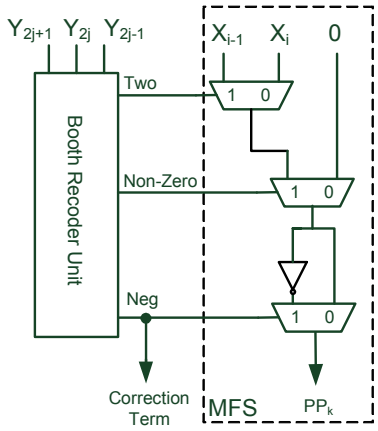

Figure 1. Radix-4 Booth PPG unit. Booth recoder is common for the bits of a PP, but each PP bit needs a separate MFS unit.

There are two reasons for choosing Radix-4 Booth multiplier. First, by modifying the PPG structure of Radix-4 Booth multiplier and doing some transformations on the sign extension parts of the PPs, we can significantly reduce the costs of adding new multiplier bit-widths to the base architecture. Second, we use the PPR unit of the multipliers in the base DSP block as a compressor tree for implementing multi-input addition. In Radix-4 Booth multiplier, the number of PPs is half compared to that of parallel array multiplier. Therefore, we can exploit faster and smaller compressors to build the compressor tree. This is a key factor in designing compressor trees for multi-input addition. We will provide more details about these advantages of Radix-4 Booth, once we unveiled the complete architecture in subsequent sections.

In the following, we give a brief overview of the base DSP architecture as it is needed to understand the modifications that we will made to increase the flexibility. There are a couple of differences from standard Radix-4 architecture in the way that we design the PPG and PPR units. For the PPG, to multiply the multiplicand by 1, 2, or 0, all that is needed is a few multiplexers, which have a delay time that is independent of the size of the inputs. The only complexity relates to negating a 2's complement number, where a '1' is added to an inverted number. This complexity can be avoided in the PPG, if we move the summation part into the PPR unit. For this purpose, a correction bit corresponding to each PP is added to the PPR. Figure 1 illustrates the PPG unit of the Radix-4 multiplier. For each PP, one Booth recoder is required, while for each bit of PP we need the multiplicand factor selection (MFS) unit shown in this figure. All MFS units get the same set of select signals.

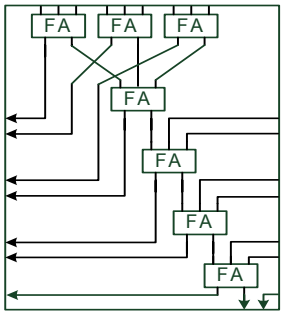

Figure 2. Proposed design for 9:2 compressor.

In Booth multiplication each PP should be sign extended and also should be shifted to the left by two bits. Therefore, the PPR unit of an 18-bit multiplier is 36-bit wide. Moreover, the height of the PPR is nine, since we have nine PPs. We propose to exploit a layer of 9:2 compressors followed by a final CPA for the PPR unit. In this layer, there is a 9:2 compressor per each column. So for an 18-bit multiplier, thirty-six 9:2 compressors are required. Figure 2 shows the proposed circuit level design of 9:2 compressor. All of its inputs, including the carry bits, have the same bit position, *rank i*. The two outputs also have *rank i*, but the carry outputs have *rank i+1*. The delay of 9:2 layer is independent of the layer width, since no ripple carry path exist in the layer. The longest path that a carry can propagate contains 3 cells.

Since the compressor layer will be reused for other DSP block configurations, the 9:2 layers of all 18-bit multipliers in the base DSP block are chained, but at the multiplier boundaries, each carry input is set to '0', by a simple AND gate.

As explained, besides the PPs, we have a number of correction bits that need to be added to the PPR unit. Each correction bit (Cbit) is aligned with the first bit of the corresponding PP. Due to the shifting of the PPs, there is always a free place in the 9:2 layer for every Cbit, except for the ninth one.

To avoid the overhead of exploiting 10:2 compressors instead of 9:2 ones, we merge the ninth Cbit with the first PP as shown in Figure 3. In this figure S is the the sign bit of the PP and C is the correction bit. Since the Cbit is aligned with the 17th bit of the PP, the MSB bits are modified from that bit position as shown.

$$\begin{array}{llllll} \ldots & S & S & S & P_{16} & \ldots \\ + & & & & C & \\ \hline \ldots & (\overline{C}+\overline{P_{16}}).S & (\overline{C}+\overline{P_{16}}).S & C.P_{16}\oplus S & C\oplus P_{16} & \ldots \end{array}$$

Figure 3. Merging the 9th Cbit with the 1st PP. MSB bits of the PP from bit 16 are modified.

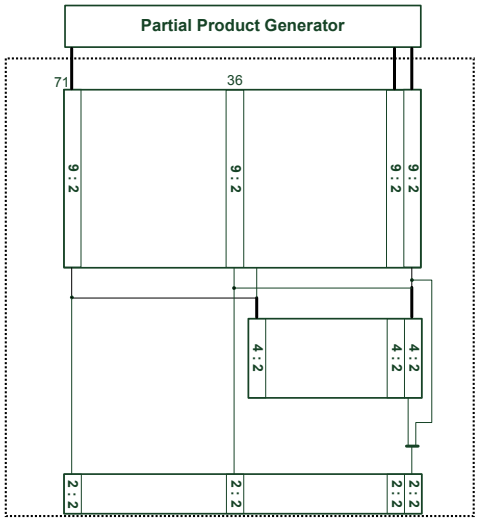

Figure 4. Building block of proposed DSP block.

In addition to 9:2 layer, we need a 4:2 layer to sum the results of the two paired 18-bit multipliers for complex arithmetic multiplication. A 4:2 compressor is similar to the 9:2 compressor, which has fewer number of inputs and carry bits. This layer is added between the 9:2 layer and the final CPA. This layer should be 36-bit wide for each multiplier pair. Figure 4 shows the diagram of the building block in the proposed DSP block. The dashed box represents the PPR unit.

5. Supporting various multiplier bit-widths

In this section we will describe how we can reduce the cost of adding new multiplier bit-width to the base DSP block by modifying the PPG structure of Radix-4 Booth multiplier and performing some transformations on the sign extension parts of the PPs. We use the same PPR unit for all multiplier bit-widths and the PPG unit is modified to provide the required inputs for the PPR, based on DSP block configuration. Providing such flexibility in the PPG can make it considerably complex. For instance as shown in Figure 5, when a new bit-width is added to the base DSP, for a certain bit position, we may need to select between a Booth encoded bit or a sign bit. This requires exploiting a multiplexer to choose the right configuration. In this figure, the PPG corresponding to the first PP of an 18-bit multiplier is illustrated on the top. Within the same number of bits, two 9-bit multipliers can fit. In this figure, we can see the first PPG of both 9-bit multiplies below that of the 18-bit multiplier. Since both configurations use the same PPR unit, we need to exploit several multiplexers to choose between these two configurations. Note that in most of the cases, we should select between a sign bit and another bit, which

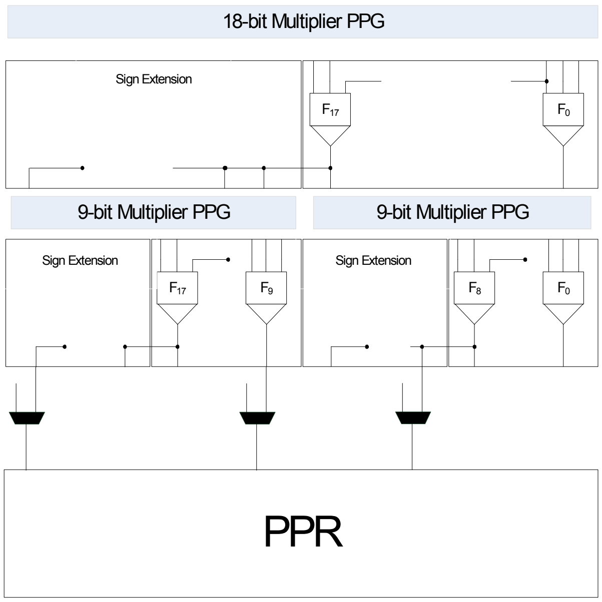

Figure 5. Overlap between the first PPG of two different multiplier configurations. Since the same PPR is used for both configurations, several multiplexers are required to select between a sign bit and an encoded-bit or other sign bit.

can be a Booth-encoded bit or a different sign bit. For the bit positions that the encoded parts of two multiplier configurations overlaps, we need to use one encoder for both, but with some multiplexers at the inputs. In this case, such overlap occurs in the first nine bit and since the encoder inputs are the same, no multiplexer is required.

Therefore, one efficient approach to reduce the PPG complexity is to avoid this huge amount of multiplexers that are added for each bit of partial product, when a new multiplier configuration is added. The first step is to eliminate the sign extension parts of the PPs. Here we use a similar technique to the one that is used in Baugh-Wooly multiplier [10]. This technique is illustrated in Figure 6. In this figure, the sign extension part of a PP is first added with +1 and then with −1. As shown, when it is added with +1, the whole sign part is reduced to a single inverted sign bit.

	S	. . . .	S	S	S	S	S
+							1
	1	. . . .	1	1	1	1	1
+							$\overline{S}$
	1	. . . .	1	1	1	1	1

Figure 6. Reducing the repetitive sign bits by adding with ±1.

```
1 . . . . . . . . . . . . . . . . .  1 1 1 1 1 1 1 1 1 1 1 1 1
1 . . . . . . . . . . . . . . . . .  1 1 1 1 1 1 1 1 1 1 1 0 0
1 . . . . . . . . . . . . . . . . .  1 1 1 1 1 1 1 1 1 0 0 0 0
                                     .
                                     .
---------------------------------------------------------------
 . . . . . . . . . . . . . . . . . . . . . . . . . . .1 0 1 0 1 1
```

Figure 7. Reducing the constant numbers to one number.

+							$\overline{S}$
	. . . . .	1	0	1	0	1	1
	. . . . .	1	0	1	$\overline{S}$	S	S

Figure 8. Merging the constant number into first partial product.

Now, if this rule is applied to the sign extension parts of N PPs, then we will have N constant numbers and N single inverted sign bits. Now, we can reduce the N constant numbers to only one number, by summing up all the constant numbers, as shown in Figure 7. Since the first bit of this constant number is aligned against the inverted sign bit of the first PP, we can append the constant number to the first PP as shown in Figure 8. The resulted value is then appended to first PP from its sign bit position.

With this technique, we replace the sign parts of all PPs with a set of '0s' and a single inverted bit, except for the first PP, where there are three sign bits and a number of '0s' and '1s'. This means that we need to choose between constant bits ('0' or '1'), single sign bits (inverted or non-inverted) and the normal Booth encoded bits. To avoid multiplexers, we modified the multiplicand factor selection (MFS) unit of the Radix-4 Booth PPG in Figure 2 and added two extra control signals as shown in Figure 9. Compared to the original design, two control signals, *Const* and *Inv*, and two simple two input gates have been added on the selection logic of the last two multiplexers in MFS. When *Inv* signal is set, the output of the MFS is inverted and when *Const* bit is set, '0' is selected as the output of the second multiplexer. Table 1 shows the operation modes of the modified PPG based on these two control signals.

With the modified MFS, for having a constant output, the *Const* signal should be set and based on the required value the *Inv* input is defined. This operation mode, resolves the conflicts of a constant bit ('0' or

Table 1. Operation modes of modified Radix-4 Booth PPG.

Const	Inv	Func
0	0	PP_k
0	1	$\overline{PP_k}$
1	0	0
1	1	1

'1') with the opposite constant bit, a Booth encoded bit and an inverted sign bit. To produce the inverted sign bit, only we need to set the *Inv* signal. For a normal encoding, the two control bits are tied to '0'.

In contrast to PPG, the PPR unit does not require any major modification. The compressor tree in the PPR was designed in a way that can be reused for several different bit-widths. For the bit widths smaller than 18, since the number of PPs is less than nine, they can be reduced within a certain number of slices in the 9:2 layer. Nevertheless, for the bigger bit-widths up to 36, we should split the PPs into two groups and reduce each group separately by two disjoint chunks of 9:2 compressors. Then, we need to sum the results of the two chunks. For this part, we use the 4:2 layer of the base DSP. The number of 9:2 slices that are required for a set of PP is obtained from the following equation. In this equation, MulBW represents the multiplier bit-width.

$$Slice\ BW = MulBW + 2 \cdot Num\ of\ PPs$$

For example, in 36-bit multiplier, we have two sets of nine PPs. Therefore, each set requires 54 9:2 slices. In this case, we allocate the first 108 slices of the 9:2 layer to the 36-bit multiplier. Similarly, for a 24-bit multiplier, we need 72 slices. The required numbers of 4:2 slices for 36- and 24-bit multipliers are 54 and 30 respectively. In fact, for both cases, the LSB of the second PP set aligns at the 19th bit of the first set and the first 18 bits go directly to the final CPA.

6. Supporting multi-input addition

It is well known in ASIC design that the right way of implementing multi-input addition is to use compressor trees. In general, using smaller compressors as the building blocks of the compressor tree leads to have faster and more flexible designs. From the multi-input addition perspective, the advantage of using the PPR of Radix-4 Booth multiplier compared to that of parallel array multiplier is that we can use smaller, faster and more flexible

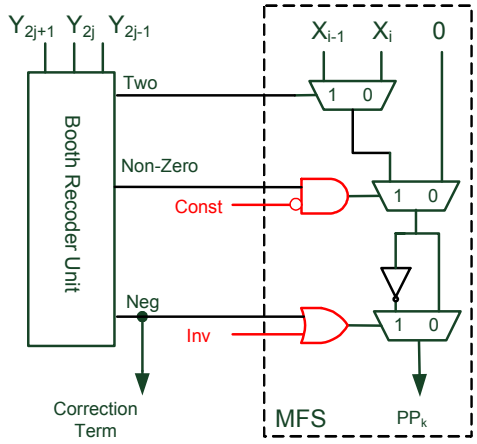

Figure 9. Modified Radix-4 Booth PPG encoder for resolving the conflicts of PPG parts of various multiplier bit-widths.

compressors for the synthesis. The compressor tree in PPR can be used for multi-input addition by bypassing the PPG. This is a missing feature in the DSP blocks of current FPGAs. In such DSP blocks, the PPG cannot be bypassed and the PPR has not been designed for multi-input addition.

Assuming that there is no connectivity constraint between the DSP block inputs and the inputs of the compressor unit inside the PPR, we can efficiently map any regular and irregular multi-input addition by the PPR. However, such a connectivity requires a fully populated switch box which is extremely costly. Therefore, the real challenge is to find a way to benefit from the inherent flexibility of the PPR compressor tree with minimum overhead.

Our solution for this problem is to define a set of fixed rectangular blocks (rc-block) within the 9:2 layer of the PPR for mapping different patterns. In fact, for different multi-input addition patterns, a different combination of the rc-blocks is used. The advantage of this approach is that no crossbar is required to make a direct connection between the DSP inputs and the PPR inputs. Since each rc-block is placed in a fixed region of the 9:2 layer, there will be a predefined pattern to connect DSP blocks inputs to the PPR for multi-input addition. Table 2 shows the dimensions of each rc-block for a half-DSP with 72 inputs. Here we assume that each half-DSP gets half of the total inputs for multi-input addition. The maximum height of an rc-block is nine since it should fit into the 9:2 layer. The width of an rc-block in this table, defines the number of slices of 9:2 that are used.

Figure 10 shows how different DSP inputs are referenced in different rc-blocks. In this figure, the first four rc-blocks in Table 2 are shown. The numbers

Table 2. Different rectangular blocks that are used for mapping the bits in an adder tree.

Block	Heigth	Width
0	9	8
1	8	9
2	7	10
3	6	12
4	5	14
5	4	18
6	3	24

inside each rc-block specify the DSP input indices. Note that the PPG unit should be bypassed for the indicated bit positions. Therefore, we insert some multiplexers into the PPG unit for this purpose. To minimize the number of multiplexers, the rc-blocks are overlapped maximally as shown in this figure. For a maximum overlap, we align all the rc-blocks to the left for the right half-DSP and to the right for left half-DSP. The reason for such alignment is that we can connect two rc-blocks of each half-DSP and form bigger blocks. For instance, rc-block 0 in the first half can be chained to rc-block 5 in the second half and form a bigger non-rectangular block for covering the input bits; similarly, two identical rc-blocks can be chained and form a wider rectangular block with the same height. Having fixed block sizes with fixed placement removes the need to route the DSP inputs to the compressor unit of PPR through a crossbar.

6.1. Multi-input addition mapping algorithm

In this section, the mapping algorithm of multi-input addition is described. Mapping algorithm is an important step, since we have to choose the best combination of the mentioned rc-blocs to efficiently cover the input bits, which almost have irregular shapes. This is an important decision, which can affect both area utilization and performance.

We define the *compression ratio (CR)* parameter for each block and based on that we prioritize them for the mapping. This parameter is defined as follows:

$$CR = \frac{\sum_{i=0}^{W_b} h_i}{2 \cdot W_b}$$

In this equation, W_b is the width of the bits that are generated by a block and h_i is the number of bits in the i-th column of the block. The higher the CR is the

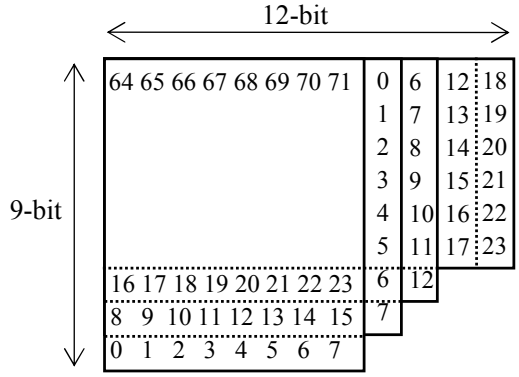

Figure 10. DSP Block input indices that are connected to each rc-block. Rc-blocks are aligned for maximum input sharing.

greater is the overall compression ratio of the rc-block. Blocks with higher CRs tend to compress more.

Although, the maximum height of a column for a specific rc-block is limited to the numbers illustrated in Table 2, more bits can be mapped to some columns in certain situations if the heights of other columns do not reach to the expected height. As an example, assume that rc-block-2 covers a set of bits, where all columns have 7 bits except the two right most columns, which only have 5 bits. This means that two of the bits in each of these two columns remain unmapped. Now if we find an input index in other columns outside the rc-block boundary that matches an input index in the short columns, then we have the chance to cover one more bit in the higher column. In this example, we can add two bits to the first column since the indices of the two exterior bits are '0' and '8' and these two inputs are found in the short columns. By using this trick, we can cover more bits by a block, when it is not fully utilized. We call this process *block refinement*.

The proposed mapping algorithm has three major steps: In the first step, the best block for covering a set of input bits is defined. This block can be either a single rc-block or two joined rc-blocks as explained. The best block is the one that has the highest CR after the block refinement. Then, the covered bits are removed from the uncovered set of bits and this step is repeated until we reach to the termination condition. Termination condition is either covering all the bits or not able to find a block with more than 50% covering ratio. The covering ratio for a block is the ratio between the number of bits that are covered and the maximum number of bits that can be covered.

The second step of the mapping is to generate the output bits corresponding to the covered bits. If this is the last level of the whole compressor tree and no other DSP block is required for the mapping, the final adder of the DSP has to be used to generate the result,

otherwise the final adder is skipped and the output will stay in the form of carry and save.

In the third step of the mapping, we place the selected blocks in the DSPs and connect the DSP blocks based on the mapping. Each DSP can hold two rc-blocks, one in each half, and the joined rc-blocks are placed in the same DSP where the two halves are chained.

7. Experiments

To evaluate the proposed DSP block architecture, we designed a sample base DSP block with 144 inputs and outputs. This is the IO bandwidth of a DSP block in Altera Stratix-II. Since we use 90nm CMOS technology for the DSP block design and we have to estimate the inter DSP net delays for our experiments, Stratix-II was selected as a baseline FPGA. Considering the IO bandwidths, two of the building blocks shown in Figure 5 can be implemented and so we will have four 18-bit multiplier. Therefore, we need a 144-bit wide 9:2 layer. So each half-DSP has two 18-bit multiplier followed by an optional 36-bit 4:2 layer for complex arithmetic multiplication. For our experiments, we add 9/12/24/36 multiplier bit-widths to the base DSP block and we evaluate the overhead that is imposed by each of them. Note that 24-bit multiplier is used for single precision floating-point multiplication. We also measure the overhead of adding multi-input addition feature to the sample DSP block. Based on the available bandwidth and resources, the designed DSP block can implement up to eight 9-bit, six 12-bit, two 24-bit and one 36-bit multipliers. For the 36-bit multiplier, some parts of the second half-DSP are used in conjunction with the complete first half-DSP. However, we can implement two 9-bit or one 12-bit multipliers in the remained part of the second half-DSP. Moreover, we can use the multi-input

Table 3. Overhead of adding new features to the base DSP. The delay numbers show the 18-bit multiplier delay in each case.

DSP Features	Delay (ns)	Area (μm^2)
Base DSP	**3.11**	**41439**
Base DSP + 9-bit Mul	3.11	43269
Base DSP + 9/12 Mul	3.12	44633
Base DSP + 9/12/24 Mul	3.14	45852
Base DSP + 9/12/24/36 Mul	3.15	46271
Base DSP + 9/12/24/36 Mul and MADD	3.15	46973

addition feature of the second half at the same time that we have 36-bit multiplier.

We modeled the sample DSP in Verilog and used Synopsys Design Compiler with *90nm Artisan* standard cell library for the synthesis. The mapping algorithm of the multi-input addition was developed in C++. The methodology that we used for estimating the net delay of DSP inter connections was to replace the DSPs of Stratix-II in the netlist and extract the real net delays by Quartus tool.

7.1. Results

Table 3 compares the synthesis results of the proposed base DSP with different features. The delay values show the delay of the 18-bit multiplier in each case and the area is the total area of the DSP with all mentioned features. The interesting point is that the delay overhead of adding new features to the base DSP is almost nothing; while the area overhead of all bit-widths before and after supporting multi-input addition feature is 11% and 13% respectively. This means that on average, less than 3% area overhead is imposed by supporting a new multiplier bit-width and the area overhead of adding multi-input addition is 2%.

Table 4 shows the combinational delay of each multiplier in the final DSP with all features included. These numbers can further be improved by inserting some pipeline registers between the layers of the DSP. In Stratix-II, the combinational delays of 9/18/36 bit multipliers are 2.99 ns, 3.17 ns and 4.57 respectively. The delay of 9-bit multiplier of DSP in [3] is 1.71 ns.

To evaluate the multi-input addition feature, we compared the synthesis results of the multi-input addition parts of some real arithmetic, multimedia and signal processing applications on Stratix-II FPGA soft logic [11], FPCT [2] and our DSP block. Note that, this feature is not supported by the DSP blocks of current FPGAs. Table 5 shows the delay results. Compared to FPCT, our DSP has a lower delay for the FIR benchmarks. On average, the delay of our DSP is around 4% slower than FPCT, which only does multi-input addition. Compared to the soft-logic, our DSP

Table 4. Delay (ns) of each multiplier in the final DSP block.

Multiplier BW	Our DSP	Stratix-II
9-bit	1.89	2.99
12-bit	2.43	-
18-bit	3.15	3.17
24-bit	4.01	-
36-bit	5.17	4.57

Table 5. Delays (ns) of multi-input addition benchmarks on different hardwares.

Benchmark	Soft Logic[11]	FPCT [2]	Our DSP
DCT	4.32	2.46	3.08
Motion Es.	4.2	1.5	2.09
G72x	4.31	3.22	3.79
ADPCM	2.41	1.41	1.83
Fir3	5.21	3.76	3.21
Fir6	6.79	4.53	3.39
Hpoly	5.55	3.36	3.81
Average	4.68	2.89	3.02

Table 6. Areas of multi-input addition benchmarks on different hardwares.

Benchmark	Soft Logic (LAB)	FPCT (DSP)	Ours (DSP)
DCT	17	2	2.5
Motion Es.	11	4	1
G72x	22	3	2.5
ADPCM	3	1	0.5
Fir3	35	6	3
Fir6	76	8	6
Hpoly	35	3	3.5

has a lower delay for all the benchmarks and on average is 54% faster.

Area comparison of these three methods is not that straight forward. Table 6 shows the area of each benchmark in terms of the basic blocks that is used. For the soft logic, the area is in terms of LABs number, for FPCT and our DSP block, the numbers represent the DSPs number. The area of FPCT is 56000 μm^2, while ours is less than 48000 μm^2 with all mentioned features.

8. Conclusion

In this paper, we proposed a base DSP block architecture for FPGAs with the potential of adding various multiplier bit-widths and multi-input addition without a considerable overhead. This was achieved by employing a number of techniques that was used to simplify the reconfigurable PPG unit. We studied the effects of adding these features on top the base architecture and we designed a sample DSP block with 9/12/18/24/36 multiplier bit-widths and multi-input addition. Moreover, the novel DSP block uses much more efficiently the limited available input/output bandwidth.

9. References

[1] H. Parandeh-Afshar, P. Brisk, and P. Ienne, "Integrating generalized parallel counters into FPGAs for improved arithmetic performance," to appear in in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* 2010.

[2] A. Cevrero, et al., "Field programmable compressor trees: acceleration of multi-input addition of FPGAs," *ACM Trans. Reconfigurable Technology and Systems*, vol. 2, no. 2, article no. 13, June, 2009.

[3] H. Parandeh-Afshar, et al., "A flexible DSP block to enhance FPGA arithmetic performance," in *Proc. of the IEEE International Conference on Field Programmable Technology, FPT'09,* pp. 210-218, 2009..

[4] M. J. Beauchamp, S. Hauck, K. D. Underwood, and K. S. Hemmert, "Architectural modifications to enhance the floating-point performance of FPGAs," *IEEE Trans. VLSI*, vol. 16, no. 2, pp. 177-187, Feb. 2002.

[5] Y. J. Chong and S. Parameswaran, "Flexible multi-mode embedded floating-point unit for field programmable gate arrays," in *Proc. Int. Symp. FPGAs '09*, 2009, pp. 171-180.

[6] C. H. Ho, et al., "Virtual embedded blocks: a methodology for evaluating embedded elements in FPGAs," in *Proc. IEEE Symp. Field Programmable Custom Computing Machines '06*, 2006, pp. 35-44.

[7] P. Jamieson and J. Rose, "Architecting hard crossbars on FPGAs and increasing their area-efficiency with shadow clusters," in *IEEE Int. Conf. Field Programmable Technology '07*, 2007, pp. 57-64.

[8] P. Jamieson and J. Rose, "Enhancing the area-efficiency of FPGAs with hard circuits using shadow clusters," in *IEEE Int. Conf. Field Programmable Technology '06*, 2006, pp. 1-8.

[9] M.D. Ercegovac and T. Lang, "Digital Arithmetic," Morgan Kaufmann Publishers, San Francisco, 2004.

[10] C. R. Baugh and B. A. Wooley, "A two's complement parallel array multiplication algorithm," in *IEEE Trans. Computers*, vol. C-22, 1973 pp.1045-1047.

[11] H. Parandeh-Afshar, P. Brisk, and P. Ienne, "Exploiting fast carry-chains of FPGAs for designing compressor trees," in *Proc. 19th international conference on Field Programmable Logic and Applications. FPL'09*, 2009, pp. 242-249.

[12] Aletra Corporation, "DSP Blocks in Stratix-III Devices," in Stratix-III Device Handbook, Volume 1. May 2009.