# Efficient Multiple Constant Multiplication Using DSP Blocks in FPGA

Ahmet Can Mert, Hasan Azgin, Ercan Kalali, Ilker Hamzaoglu
Faculty of Engineering and Natural Sciences
Sabanci University
Istanbul, Turkey
{ahmetcanmert, hasanazgin, ercankalali, hamzaoglu}@sabanciuniv.edu

***Abstract*—Multiple constant multiplication (MCM) operation multiplies an input variable with multiple constants. MCM operations are widely used in many applications such as video processing and compression. In this paper, a method is proposed for efficient implementation of MCM operations using DSP blocks in Xilinx FPGAs. The proposed method reduces number of DSP blocks used for implementing a given MCM operation by manipulating the multiple constants used in this MCM operation. In this paper, a high level synthesis tool implementing the proposed method is also proposed. The proposed tool takes the input variable bit length and multiple constants as inputs, and generates a Verilog RTL code which efficiently implements this MCM operation using DSP blocks. The proposed method and tool are used for one of the most complex video compression algorithms, HEVC 2D DCT. They reduced number of DSP blocks used in the FPGA implementation of HEVC 2D DCT algorithm by 35.8%.**

***Keywords—MCM, FPGA, DSP Block, HEVC***

## I. Introduction

Modern FPGAs provide different full-custom built-in blocks in addition to look up tables (LUTs) and registers. FPGA implementations using these built-in blocks can perform operations faster with less resources and power consumption than FPGA implementations using LUTs. Therefore, they are used in wide range of applications. DSP blocks, which have full-custom multiplier hardware, are one of these built-in blocks in FPGAs. They are used in FPGA implementations of many applications such as video processing and compression, and machine learning [1]-[2].

Multiple constant multiplication (MCM) operation multiplies a single variable with multiple constants. It is used in many digital signal processing (DSP) applications such as finite impulse response (FIR) filter, discrete cosine transform (DCT) and fast Fourier transform (FFT).

DSP blocks perform constant multiplications faster and with less energy than adders and shifters. A DSP block can be used to perform different constant multiplications by providing proper constant value to its input. Therefore, it is more efficient to implement constant multiplications using DSP blocks instead of using adders and shifters in an FPGA implementation.

In this paper, a method is proposed for efficient implementation of MCM operations using DSP blocks in Xilinx FPGAs. The proposed method reduces number of DSP blocks used for implementing a given MCM operation by manipulating the multiple constants used in this MCM operation. In this paper, a high level synthesis tool implementing the proposed method is also proposed. The proposed tool takes the input variable bit length and multiple constants as inputs, and generates a Verilog RTL code which efficiently implements this MCM operation using DSP blocks.

In this paper, to demonstrate effectiveness of the proposed method and tool, an FPGA implementation of one of the most complex video compression algorithms, HEVC Two Dimensional (2D) DCT, is done using them. They reduced the number of DSP blocks used in the FPGA implementation of HEVC 2D DCT algorithm by 35.8%.

There are many techniques proposed in the literature to optimize multiple constant multiplication operations [3]-[5]. These techniques try to find common sub-expressions between multiple constants, and they implement MCM operations using adders and shifters. They provide efficient solutions for ASIC implementations. Since their FPGA implementations use LUTs instead of DSP blocks, they are inefficient for FPGA implementations compared to using DSP blocks in FPGAs.

The techniques proposed in [6]-[7] use pipeline registers in DSP blocks to schedule operations efficiently. They also use different scheduling techniques to increase the resource sharing on DSP blocks. In this way, they can increase the throughput. The method proposed in [8] can map two constant multiplication operations into one DSP block by concatenating two constants and assigning them to one input of DSP block. Therefore, it reduces the number of DSP blocks used for MCM operations. The method proposed in [8] uses only two inputs (A, B) of a DSP block. It works only for unsigned input variables. However, the method proposed in this paper can map more constant multiplication operations to one DSP block by manipulating the constants and using three inputs (A, B, C) of a DSP block. In addition, it works for both unsigned and signed input variables.

## II. Xilinx DSP Block Architecture

Simplified architecture of Xilinx DSP48E1 block is shown in Fig. 1. A DSP48E1 block has a signed 25 bit pre-adder, a signed 25x18 bit multiplier and an ALU which has 48 bit adder/subtractor and pattern detector. Since these sub-blocks are implemented as full-custom hardware on FPGA, they provide higher speed and lower power consumption than equivalent LUT implementations of the same operations. These sub-blocks can be configured to implement different operations in each clock cycle. In this way, a DSP48E1 block can perform different operations such as A×B and A×B+C.

## III. Proposed Method

The proposed method manipulates the constants in a multiple constant multiplication operation to map more constant multiplication operations to one DSP block.

Multiplication of a $v$ bit input variable $V$ with two constants $m_1$bit $M1$ and $m_2$ bit $M2$ can be performed using two DSP blocks, where each DSP block performs multiplication with one constant. The method proposed in [8] concatenates two constants to perform two constant multiplications using one DSP block as shown in (1). A DSP block has a 25x18 bit signed multiplier. Therefore, $v$ and $(v + m_1 + m_2)$ should be less than 18 and 25, respectively, to map two constant multiplications to one DSP block.

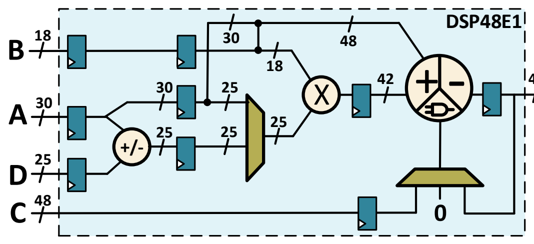

Fig. 1. Simplified architecture of Xilinx DSP48E1 block.

| Input Bit Length: 9 Bit Signed | Constants: [78913, 100663360] |
|---|---|
| $78913 = 2^0(1 + 2^6 \times 1233)$, <br> $100663360 = 2^6(1 + 2^{19} \times 3)$, | $m_1$=17, $s_1$=0, $n_1$=6, $MM_1$=1233 <br> $m_2$=27, $s_2$=6, $n_2$=19, $MM_2$=3 |

$\text{cost} = v + m_1 + m_2 - (s_1 + s_2) - (n_1 + n_2) = 22 < 25$ ✓

$\text{signext}_{78913} = V[8]*[2^{17} - 78913\times 2^0][16:0]$
$\text{signext}_{100663360} = V[8]*[2^{21} - 100663360\times 2^{-6}][20:0]$

$V[v\text{-}1:n_1] = V[8:6]$   $V[v\text{-}1:n_2] = V[8:19]$ ($\varnothing$)

***DSP48E1 Inputs:***
A = {$MM_2$ << (v+m1-(n1+s1)) | $MM_1$)}  A = {3 << 20 | 1233}
B = 9 Bit Signed Input
C = {$\text{signext}_{100663360}$, $\text{signext}_{78913}$, V[8:6]}

Fig. 2. An example constant manipulation.

TABLE I. MAXIMUM NUMBER OF CONSTANTS PER DSP BLOCK

| Input Variable Bit Length (v) | $1<v\le2$ | $2<v\le3$ | $3<v\le4$ | $4<v\le6$ | $6<v\le8$ | $8<v\le12$ | $12<v\le18$ |
|---|---|---|---|---|---|---|---|
| MaxConstDSP | 24 | 12 | 8 | 6 | 4 | 3 | 2 |

$$\{V \times M2, V \times M1\} = V \times \big((M2 \ll (v + m1))|M1\big) \quad (1)$$

The symbols "×", "{ , }", "|", "≪" and "≫" represent multiplication, concatenation, bit-wise or, left shift and right shift operations, respectively. This method can be used for k constant multiplications if the bit lengths of input variable and constants satisfy the condition in (2).

$$((k-1) \times v + \sum_{i=1}^{k} m_i) < 25 \quad (2)$$

The proposed method reduces bit length of a constant by manipulating the constant and utilizing C input of DSP block. Manipulation of an $m$ bit constant $M$ multiplied with $v$ bit input variable $V$ is shown in (3)-(10). First, $V$ and $M$ can be represented as shown in (4)-(5). Thus, $MM$ is equal to $\big((M \gg s) - 1\big) \gg n$. $s$ and $n$ can take different values. The proposed method uses $s$ and $n$ values that minimize $MM$. Then, $V$ and $M$ are substituted into (3).

After the proposed manipulation, $V \times M$ is converted to $\{(MM \times V + V[v-1:n]), V[n-1:0]\} \ll s$ which means multiplication of $v$ bit input variable and $m$ bit constant is converted to multiplication of $v$ bit input variable and $(m - (n+s))$ bit manipulated constant, addition of $(v-n)$ bit input variable, concatenation and shift operations. Since concatenation and shift operations require no computation, only the equation $(MM \times V + V[v-1:n])$ needs to be implemented using DSP block.

$$V \times M = V[v-1:0] \times M[m-1:0] \quad (3)$$
$$M = 2^s \times (1 + 2^n \times MM) \quad (4)$$
$$V = 2^n \times V[v-1:n] + V[n-1:0] \quad (5)$$
$$where\ s \ge 0, n \ge 0$$
$$V \times M = V \times 2^s \times (1 + 2^n \times MM) \quad (6)$$
$$V \times M = 2^s \times (V + (V \times 2^n \times MM)) \quad (7)$$
$$\text{V}\times\text{M}=2^{\text{s}}\times(2^{\text{n}}\times\text{V[v-1:n]}+\text{V[n-1:0]}+(\text{V}\times2^{\text{n}}\times\text{MM})) \quad (8)$$
$$\text{V}\times\text{M}=2^{\text{s}}\times(2^{\text{n}}\times(\text{V}\times\text{MM}+\text{V[v-1:n]})+\text{V[n-1:0]}) \quad (9)$$
$$\text{V}\times\text{M}=\{(\text{MM}\times\text{V}+\text{V[v-1:n]}),\text{V[n-1:0]}\}\ll s \quad (10)$$

$MM$, $V$ and $V[v-1:n]$ are assigned to A, B and C inputs of DSP block, respectively. The bit length of $MM$ is smaller than the bit length of $M$. Therefore, more constant multiplications can be mapped to one DSP block by manipulating the constants before multiplication.

If the input variable is a signed number, sign extension of $V[v-1:n]$ for each constant multiplication should also be added to multiplication result using C input of DSP block. Sign extension calculation for an m bit constant $M$ and $v$ bit signed input variable $V$ is shown in (11). After sign extension, $\{signext, V[v-1:n]\}$ should be assigned to C input of DSP block.

$$signext = \big(\text{V[v-1]}\times(2^{\text{m-s}}\text{-M}\times2^{\text{-s}})\big)\text{[m-s-1:0]} \quad (11)$$

Multiplication of a $v$ bit input variable $V$ with two constants $m_1$ bit $M1$ and $m_2$ bit $M2$ can be performed using the proposed method as shown in (12)-(14). Final concatenation and shift operations are not shown for simplicity.

$$M1 = 2^{s_1} \times (1 + 2^{n_1} \times MM1) \quad with\ signext_1 \quad (12)$$
$$M2 = 2^{s_2} \times (1 + 2^{n_2} \times MM2) \quad with\ signext_2 \quad (13)$$
$$\{V \times M2, V \times M1\} =$$
$$V \times \big(MM2 \ll (v + m_1 - (s_1 + n_1))\ |\ MM1\big) \quad (14)$$
$$+\{signext_2, V[v-1:n_2], signext_1, V[v-1:n_1]\}$$

The proposed method can be used for k constant multiplications if the bit lengths of input variable and manipulated constants satisfy the condition in (15).

$$((k-1) \times v + \sum_{i=1}^{k} (m_i - (s_i + n_i))) < 25 \quad (15)$$

An MCM example using the proposed method is shown in Fig. 2. In this example, a 9 bit signed input variable is multiplied with two constants; 78913 and 10066336. Since a DSP block has a 25x18 bit multiplier, multiplication with 27 bit constant 100663360 cannot be mapped to one DSP block without the proposed constant manipulation. Therefore, concatenation method proposed in [8] cannot map this constant multiplication to a DSP block. However, the proposed method can map these two constant multiplications to one DSP block.

## IV. PROPOSED HIGH-LEVEL SYNTHESIS (HLS) TOOL

In this paper, a high level synthesis tool implementing the proposed algorithm is also proposed. As shown in Fig. 3, the proposed tool takes the input variable bit length and multiple constants as inputs, and generates a Verilog RTL code which efficiently implements this MCM operation using DSP blocks. If a constant is power of 2, this constant multiplication is implemented with shift operation. If a constant is a power of 2 multiple of another constant in the input constants, this constant multiplication is also implemented with shift operation. The remaining constant multiplications are mapped to DSP blocks using the proposed DSP mapping algorithm. Finally, a Verilog RTL code is generated for this MCM operation.

Flow chart of the proposed DSP mapping algorithm is shown in Fig. 4. The proposed algorithm takes bit length of input variable and multiple constants as input. It groups and maps the multiple constant multiplications to minimum number of DSP blocks.

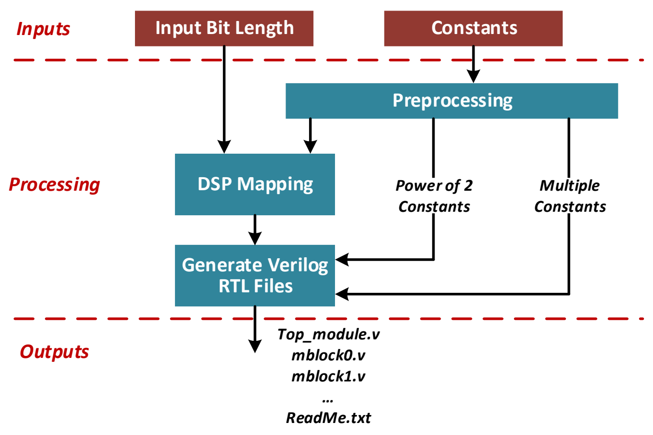

Fig. 3. Flow chart of the proposed DSP mapping HLS tool.

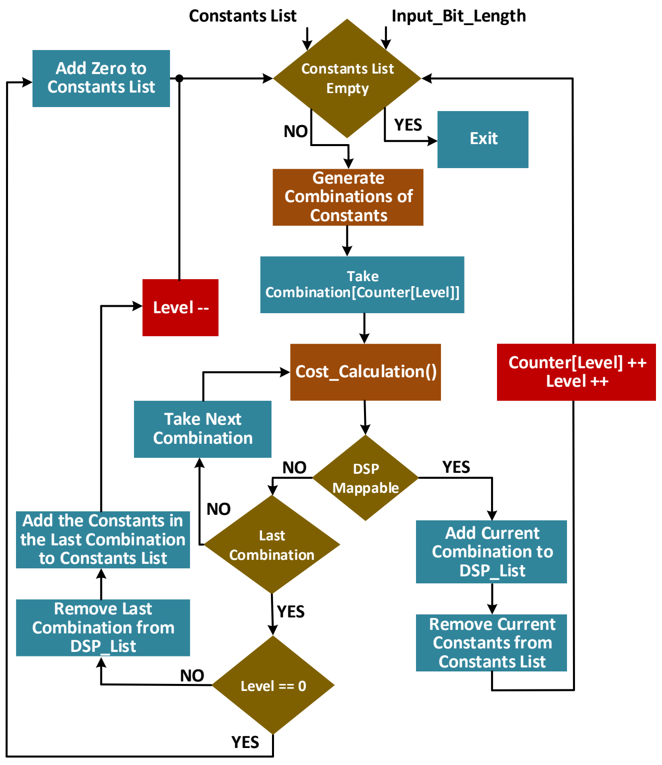
Constants List
Input_Bit_Length
Add Zero to Constants List
Constants List Empty
NO
YES
Exit
Generate Combinations of Constants
Take Combination[Counter[Level]]
Level --
Cost_Calculation()
Counter[Level] ++ Level ++
Take Next Combination
DSP Mappable
NO
YES
NO
Last Combination
Add the Constants in the Last Combination to Constants List
Add Current Combination to DSP_List
Remove Current Constants from Constants List
YES
Remove Last Combination from DSP_List
NO
Level == 0
YES

Fig. 4. Flow chart of the proposed DSP mapping algorithm.

```
$cost\_calculation(input_{bitlength}, combination)\{$
$cost = 0, i = 0, comb_{size} = size(combination)$
WHILE($i < comb_{size}$)
    IF($combination[i]\ != 0$)
        $combination[i] = 2^s \times (1 + 2^n \times MM)$
        $cost = cost + bitlength((1 + 2^n \times MM)) - n$
        $i = i + 1$
ENDWHILE
$cost = cost + input_{bitlength} \times (comb_{size} - 1)$
IF($cost > 24$)    $DSP\_mappable = 0$
ELSE               $DSP\_mappable = 1$
$return\ DSP\_mappable\ \}$
```

Fig. 5. The proposed cost calculation function.

The proposed DSP mapping algorithm is an iterative algorithm. It starts with *Iteration* 0 and *Level* 0. It puts all the constants that will be multiplied with input variable to *Constants_List*. It sorts the constants in *Constants_List*, generates all possible combinations of these constants and puts them to *Combinations_List*. Each combination has the same number of constants, and the number of constants in a combination is determined by input variable bit length. Table I shows maximum number of constant multiplications that can be mapped to a DSP block (MaxConstDSP) for different input variable bit lengths. MaxConstDSP can be calculated by using (15).

*Counter* keeps the number of combinations tried to be mapped to a DSP block at each level. Initially, *Counter* is set to zero for each level. The proposed algorithms takes the first combination in *Combinations_List* of the current level and determines whether it can be mapped to one DSP block or not by using the cost calculation function shown in Fig. 5.

If the current combination cannot be mapped to a DSP block, *Counter* for the current level is incremented by one, the next combination in *Combinations_List* is taken and determined whether it can be mapped to one DSP block or not by using the cost calculation function shown in Fig. 4. This process continues until either a combination that can be mapped to one DSP block is found or it is determined that none of the combinations in *Combinations_List* can be mapped to a DSP block.

If the current combination can be mapped to one DSP block, then it is added to *DSP_List* and the constants in the current combination are removed from *Constants_List*. *DSP_List* contains the combinations of constants that are mapped to DSP blocks. *Counter* for the current level is incremented by one. *Level* is incremented by one and the proposed algorithm continues with the next level. *Level* keeps the recursion depth of the current iteration.

If there are no constants left in *Constants_List*, this means all the constant multiplications are mapped to DSP blocks and the proposed algorithm terminates successfully.

If none of the combinations in *Combinations_List* of the current level can be mapped to a DSP block and *Level* is greater than 0, *Counter* for the current level is reset to 0, *Level* is decremented by one, the last combination is removed form *DSP_List*, the constants in that combination are added to *Constants_List*, and the proposed algorithm continues with the previous level.

If none of the combinations in *Combinations_List* of the current level can be mapped to a DSP block and *Level* is 0, this means the proposed algorithm cannot map the constant multiplications to DSP blocks in groups of MaxConstDSP and the proposed algorithm terminates the current iteration.

In that case, the proposed algorithm adds constant 0 to *Constants_List* and it starts the next iteration with this new *Constants_List*. In these iterations, if a combination contains 0 as a constant, it means that only nonzero constants in this combination are mapped to a DSP block. Therefore, some combinations with fewer nonzero constants than MaxConstDSP can be mapped to DSP blocks.

The proposed algorithm continues with next iterations by adding constant 0 to *Constants_List* until all the constant multiplications are mapped to DSP blocks successfully.

The proposed algorithm sorts the constants in *Constants_List* before generating *Combinations_List*. In this way, it generates combinations of constants in the same order as it goes back and forth between different levels.

An example of mapping constant multiplications to DSP blocks using the proposed algorithm is shown in Fig. 6. In iteration 0, the proposed algorithm cannot map the four constant multiplications to two DSP blocks. Constant 0 is added to *Constants_List* in iteration 1 and iteration 2. The proposed algorithm mapped the four constant multiplications to three DSP blocks successfully in iteration 2.

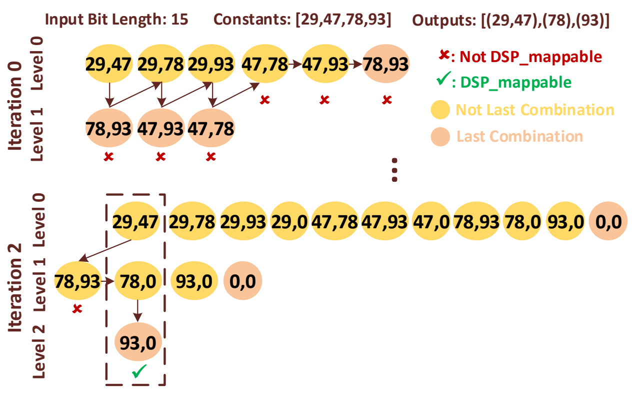

Fig. 6. An example DSP mapping using the proposed algorithm.

TABLE II. CONSTANT MULTIPLICATIONS ON DSP BLOCKS

| | Mult. Block | Input Length | Constants List | DSP Mapping | # of DSPs |
|---|---|---|---|---|---|
| DCT_Column | 1st 4x4 | 13 bit | 36,64,83 | (36,83) | 1 |
| | 2nd 4x4 | 12 bit | 18,50,75,89 | (18,75),(50,89) | 2 |
| | 8x8 | 11 bit | 9,25,43,57, 70,80,87,90 | (9,87),(80,70), (25,43),(57,90) | 4 |
| | 16x16 | 10 bit | 4,13,22,31,38, 46,54,61,67,73, 78,82,85,90 | (13,67),(22,85), (82,78),(31,90), (38,73),(46,61),(54) | 7 |
| DCT_Row | 1st 4x4 | 20 bit | 36,64,83 | (36),(83) | 2 |
| | 2nd 4x4 | 19 bit | 18,50,75,89 | (18),(50),(75),(89) | 4 |
| | 8x8 | 18 bit | 9,25,43,57, 70,80,87,90 | (25,90),(80,43), (9,70),(57),(87) | 5 |
| | 16x16 | 17 bit | 4,13,22,31,38, 46,54,61,67,73, 78,82,85,90 | (82,73),(22,90), (13,85),(31),(38),(46), (54),(61),(78),(67) | 10 |

TABLE III. IMPLEMENTATION RESULTS

| | Baseline | Concatenate | Proposed |
|---|---|---|---|
| **FPGA** | Virtex-6 | Virtex-6 | Virtex-6 |
| **DFF** | 7123 | 7119 | 7117 |
| **LUT** | 17176 | 17100 | 17086 |
| **DSP** | 584 | 438 | 380 |
| **Freq. (MHz)** | 147 | 130 | 147 |

## V. CASE STUDY: HEVC 2D DCT

HEVC uses DCT-II for DCT operations. It uses 4x4, 8x8, 16x16, 32x32 Transform Unit (TU) sizes. HEVC performs 2D transform operation by applying 1D transforms in vertical and horizontal directions. The coefficients in the HEVC 1D transform matrices are derived from the DCT-II basis functions. However, integer coefficients are used for simplicity.

In this paper, three different HEVC 2D DCT hardware for all TU sizes are designed and implemented. The first (baseline) hardware uses DSP blocks for multiple constant multiplications. In this hardware, each multiplication is implemented using one DSP block. The second (concatenate) hardware uses concatenation method proposed in [8]. In the baseline and concatenate hardware, if a constant is a power of 2 or a power of 2 multiple of another constant, it is implemented using shift operation instead of DSP block. Finally, the third (proposed) hardware uses the proposed DSP mapping algorithm to reduce number of DSP blocks.

The proposed hardware perform 2D DCT by first performing 1D DCT transform on the columns of a TU, and then performing 1D DCT transform on the rows of the TU. After 1D column DCT, the resulting coefficients are stored in a transpose memory, and they are used as input for 1D row DCT. One 4x4 datapath is used for 4x4 TU size. Two 4x4 datapaths are used for 8x8 TU size. Two 4x4 datapaths and one 8x8 datapath are used for 16x16 TU size. Two 4x4, one 8x8 and one 16x16 datapaths are used for 32x32 TU size [9].

Since different constants are used in HEVC 2D DCT for 4x4, 8x8, 16x16 and 32x32 TU sizes, four different multiplier blocks are used in the proposed hardware. Multiplier blocks in the first 4x4, second 4x4, 8x8 and 16x16 datapaths multiply a single input with 3, 4, 8 and 16 different constants, respectively. The proposed DSP mapping algorithm is used for MCM operations implemented in these datapaths. As shown in Table II, the proposed DSP mapping algorithm performs 27 different constant multiplications in HEVC 2D DCT using only 14 and 21 DSP blocks in the column (DCT_Column) and row (DCT_Row) transforms, respectively.

The Verilog RTL codes are synthesized and mapped to a Xilinx XC6VLX550T FF1760 FPGA with speed grade 2 using Xilinx ISE 14.7. FPGA implementations are verified with post place and route simulations. The implementation results are shown in Table III. The proposed FPGA implementation uses 35.8% and 13% less DSP blocks than baseline and concatenate FPGA implementations, respectively.

## VI. CONCLUSIONS

In this paper, a method is proposed for efficient implementation of MCM operations using DSP blocks in FPGAs. The proposed method reduces number of DSP blocks used for implementing a given MCM operation by manipulating the multiple constants used in this MCM operation. In this paper, a high level synthesis tool implementing the proposed method is also proposed. The proposed tool takes the input variable bit length and multiple constants as inputs, and generates a Verilog RTL code which efficiently implements this MCM operation using DSP blocks. The proposed method and tool reduced number of DSP blocks used in the FPGA implementation of HEVC 2D DCT algorithm by 35.8%.

## REFERENCES

[1] H. Nakahara and T. Sasao, "A deep neural network based on nested residue number system," Int. Conf. on Field Programmable Logic and Applications (FPL), Sep. 2015.

[2] H. Azgin, A. C. Mert, E. Kalali, and I. Hamzaoglu, "An efficient FPGA implementation of HEVC intra prediction," IEEE Int. Conf. on Consumer Electronics, Jan. 2018.

[3] Y. Voronenko and M. Püschel, "Multiplierless constant multiple multiplication," ACM Trans. on Algorithms, vol. 3, no. 2, May 2007.

[4] A. K. Oudjida, A. Liacha, M. Bakiri, and N. Chaillet, "Multiple constant multiplication algorithm for high-speed and low-power design," IEEE Trans. on Circuits and Systems-II: Express Briefs, vol. 63, no. 2, pp. 176-180, Aug. 2015.

[5] M. Kumm, M. Hardieck, J. Willkomm, P. Zipf, and U. M. Baese, "Multiple constant multiplication with ternary adders," Int. Conf. on Field Programmable Logic and Applications (FPL), Sep. 2013.

[6] B. Bonak and S. A. Fahmy, "Efficient mapping of mathematical expressions into DSP blocks," Int. Conf. on Field Programmable Logic and Applications (FPL), Sep. 2014.

[7] B. Bonak and S. A. Fahmy, "Improved resource sharing for FPGA blocks," Int. Conf. on Field Programmable Logic and Applications (FPL), Sep. 2016.

[8] Y. Fu, E. Wu, A. Sirasao, S. Attia, K. Khan, and R. Witting, "Deep learning with INT8 optimization on Xilinx devices," Xilinx White Paper, WP486, Nov. 2016.

[9] E. Kalali, A. C. Mert, and I. Hamzaoglu, "A computation and energy reduction technique for HEVC Discrete Cosine Transform," IEEE Trans. on Consumer Electronics, vol. 62, no. 2, pp. 166-174, May 2016.