Low-Cost Approximate Constant Coefficient Hybrid Binary-Unary Multiplier for DSP Applications

S. Rasoul Faraji
Department of ECE
University of Minnesota, Twin Cities
Minneapolis, MN, USA
faraj008@umn.edu

Pierre Abillama
Department of ECE
University of Minnesota, Twin Cities
Minneapolis, MN, USA
abill001@umn.edu

Kia Bazargan
Department of ECE
University of Minnesota, Twin Cities
Minneapolis, MN, USA
kia@umn.edu

Abstract—**Multipliers are used in virtually all Digital Signal Processing (DSP) applications, such as image and video processing. Multiplier efficiency has a direct impact on the overall performance of such applications, especially when real-time processing is needed, as in 4K video processing, or where hardware resources are limited, as in mobile and IoT devices. We propose a novel, low-cost, low energy and high-speed approximate constant coefficient multiplier (CCM) using a hybrid binary-unary encoding method. The proposed method implements a CCM using simple routing networks with no logic gates in the unary domain, which results in more efficient multipliers compared to Xilinx LogiCORE IP CCMs and table-based KCM CCMs on average. We evaluate the proposed multipliers on 2-D discrete cosine transform and fast Fourier transform algorithms as two common DSP modules. Post-routing FPGA results show that the proposed multipliers can improve the $\{$area $\times$ delay cost, and energy consumption per sample$\}$ of 8-bit fixed-point 2-D discrete cosine transform on average by $\{33\%, 36\%\}$. The improvement for 128-point 16-bit fixed-point fast Fourier transform on these metrics is $\{45\%, 54\%\}$. Moreover, the throughput of the proposed 2-D discrete cosine transform and 128-point fast Fourier transform architectures are on average 1.04$\times$ and 1.76$\times$ of the throughput of the binary architectures implemented using Xilinx LogiCORE IP CCMs, respectively.**

I. Introduction

Digital signal processing (DSP) blocks are widely used in processing multimedia input, such as video, image and cellular communication signals. Most applications require real-time processing, with a demand for computationally intensive DSP functions while processing large amounts of data under stringent power and battery usage constraints, especially in mobile platforms such as cell phones and satellite applications. The basic computational block of most DSP algorithms is the multiplier unit. Many DSP algorithms that transform input data from a domain into another one, such as discrete cosine transform (DCT) and fast Fourier transform (FFT), use constant coefficient multipliers (CCMs). The need for real-time processing in today's applications, *e.g.*, 4K video processing, motivated us to improve the performance of CCMs and to propose highly efficient and low-latency DSP accelerators.

Two general techniques that have been explored to implement constant coefficient multiplication are shift-and-add tree [1] and table-based Constant-coefficient multiplier (KCM) [2], [3]. The first technique shifts the input according to the position of non-zero bits of the constant and then adds all shifted-versions of the input together. The number of adders depends on the number of non-zero bits of the constant [4]. The second technique splits the input into chunks of α bits and then implements each partial product using look-up tables. Finally, KCM adds all partial products together based on their weights. For an FPGA architecture that uses K-input lookup tables, α is set to K. The cost of KCM mostly depends on the size of the constant while the cost of shift-and-add mostly depends on the complexity of the constant. Moreover, KCM can implement a real constant such as log(2) much more accurately than the shift-and-add tree technique when the input and output widths are limited [5].

The KCM and shift-and-add tree techniques implement a CCM using binary encoding. An alternative encoding called **hybrid binary-unary (HBU)** was introduced in our recent paper [6], which represents the most significant bits of a number using binary, and the lower bits in unary. In the unary encoding, N parallel wires are used to represent a number between $0..N$. To represent the number $P \leq N$, the first P wires are set to logic 1, and the rest to logic 0 (thermometer encoding). Computing using the HBU encoding results in remarkable improvements in $A \times D$ compared to binary. The HBU method is essentially an evolution of the field of Stochastic Computing (SC) [7]–[12], which was improved for better accuracy using deterministic coding [13]–[15], and lower latency using parallelism [16], [17].

Our HBU work [6] showed results on isolated functions, *e.g.*, $\sin(15x)$, x^γ, and $\tanh(x)$. We showed significant $A \times D$ improvements over conventional binary ($A \times D$ was only 2.5% of the binary at 8-bit resolution), which is remarkable but does not necessarily convince application developers because the real question is how much cost improvement we get when using such techniques in a complete system. That is the main contribution of this work. In this paper, we use the main idea in [6] to propose a new architecture to implement *approximate constant coefficient* multipliers and then we show an *overall system* that uses the HBU CCMs. We evaluate the proposed multipliers first by extracting the hardware costs for an FPGA implementation and comparing it against Xilinx LogiCORE IP CCMs and table-based KCM approach. Then we implement the two most common DSP algorithms, 2-D DCT and FFT algorithms, using the proposed CCMs. Compared

2576-2621/20/$31.00 ©2020 IEEE
DOI 10.1109/FCCM48280.2020.00022

to the 2-D DCT implementation using Xilinx LogiCORE IP CCMs, our architecture improves the $A \times D$ cost by 37%, while increasing the throughput by a factor of 1.04x, reducing power by 33%, and the energy per sample by 36%. Our implementation of 8- to 128-point FFT reduces dynamic power consumption and energy per sample consumption compared to implementations using Xilinx LogiCORE IP CCMs by 16% and 14%, respectively. (Table III).

II. The Proposed Approach

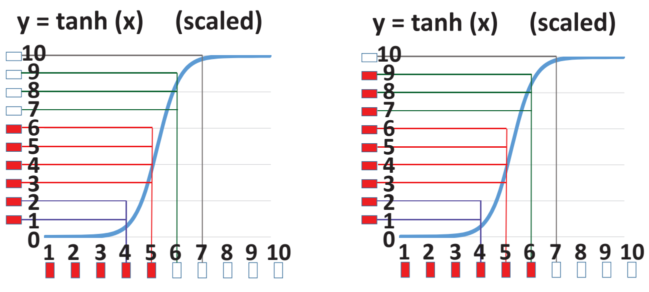
In this section, we first discuss the basic idea of fully unary and HBU computing, which was proposed in [6]. Then, we use those ideas to develop our HBU CCM, and finally we introduce the overall architecture of our method.

A. The Basic Idea (Our Previous Work)

Recently, the parallel thermometer number representation [17] was explored as an evolution of the original "random", serial bitstreams used in stochastic computing. This approach represents an N-bit precision binary number using 2^N parallel bits. Any integer $0 \leq M \leq 2^N - 1$, is represented using 2^N parallel bits where the first M bits are ones and the rest are zeros. We call this representation **fully unary**. The approach in [17] first converts binary numbers to the thermometer format, then implements desired functions using a "scaling network" (relevant to this paper) and "alternator logic" (not relevant to this paper) in the fully unary domain, and finally converts unary data back to the binary domain using an adder tree.

Fig. 1 shows an example of this method on $y = \tanh(x)$, where both input and output are scaled and quantized to the set of integer numbers between 0 and 10. Part (a) of the figure shows that for $x = 5$, y evaluates to 6. Note that both the input (the boxes under the x axis) and the output (the boxes to the left of the y axis) are in thermometer format: when $x = 5$, all boxes from 1-5 are lit up. When x increases to 6 in part (b) of the figure, it lights up three y outputs, because the slope of the function at $x = 5$ is 3. Note that in this implementation, only wires are used with no logic gates. Each wire is derived from a gate in the thermometer encoder. The fanout of each gate is determined by the derivative of the function at that point. Even though it is very efficient for complex functions such as $y = \tanh(x)$, the method of [17] can not beat conventional binary implementations on simpler functions, such as high-resolution CCMs, in terms of the area or $A \times D$ costs, especially as the bit resolution increases due to the exponential growth of the number representation (2^N wires for an N-bit binary number).

To address the scalability problem, we proposed a **hybrid binary-unary (HBU)** computing approach that can implement most complex and some simple functions more efficiently [6]. The method takes advantage of the compact representation of binary on higher bits of the number, and the simplicity of the computation logic in unary on the lower bits of the number. It calculates functions by first dividing the target function into a few sub-functions, then implementing each sub-function in the fully unary domain, and finally multiplexing



(a) x=5/10 (b) x=6/10

Fig. 1. Scaled y=tanh(x) quantized and implemented using the method of [17].

the appropriate sub-functions' output to the final output. In this method, the input range of each sub-function is a power of 2 and would not necessarily have the same length. The proposed method uses smaller individual or shared binary-to-thermometer encoders to encode each region. The method decomposes a target function as follows:

$$f(x) = \begin{cases} f_1(x) & 0 \leq x < x_1 \\ \ldots & \\ f_K(x) & x_{k-1} \leq x < x_k \end{cases} \tag{1}$$

where the input domain of any f_i and f_j ($\forall i \neq j$) do not intersect. The advantages of the HBU approach are: (1) preserving the higher bits of the binary data makes the encoding logarithmic as in the conventional binary representation, and (2) dividing each function into few sub-functions makes both the unary encoding and the unary function evaluation exponentially less costly. Therefore, the HBU approach can remarkably improve the FPGA and ASIC implementation costs compared to the conventional binary, classic stochastic computing, and the fully unary approaches in terms of area, power, energy and throughput [6].

The fully unary approach implements monotonic functions using only wires and **NO gates**[1] [6], [17], and non-monotonic functions using wiring and XOR gates. However, the HBU approach tries to split non-monotonic functions into completely monotonic sub-functions. Therefore, because of smaller thermometer encoders, simple routing networks, and smaller decoders, HBU reduces the implementation cost drastically compared to other approaches. In this paper, we use the idea behind the HBU approach and propose a new architecture to implement CCMs that are widely used in DSP applications such as FFT and DCT.

B. Fully Unary Implementation

The HBU approach uses the fully unary approach to implement sub-functions. Fig. 2 shows fully unary cores for implementing an unsigned CCM. Just as in Fig. 1, the horizontal and vertical box arrays represent the input and the output wire bundles in the thermometer format, respectively. The lines connecting the boxes are the wiring network that implements the function. Fig. 2 implements $y = \frac{1}{2}x$. Since the derivative of the function is $\frac{1}{2}$, a single change in the output of its circuits

[1]Functions with negative slopes need inverter gates.

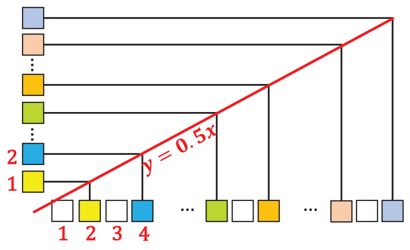


Fig. 2. Unsigned multiplier evaluation in the unary domain ($y = \frac{1}{2}x$).

corresponds to two changes in the input. As illustrated in this figure, implementing an unsigned CCM only requires a simple routing network. It should be mentioned that a signed CCM is a monotonic function that the fully unary approach implements using only a simple routing network for positive constants and a simple routing network with inverter gates at the output port for negative constants (vertical box in Fig. 2).

Even though the fully unary approach can implement CCMs using a cheap routing network, the required encoder and decoder units to convert between binary and unary encodings cause the implementation cost for high-resolution computations to increase to unacceptable levels, making it not competitive with binary computations. In addition, since a signed CCM is a constant slope monotonic function that exhibits symmetry around the middle of the input range, the HBU approach can work very well on these functions and decompose them into smaller sub-functions. In Section II-C, we will show that our new architecture based on the HBU approach uses a smaller encoder and decoder to implement a CCM, which results in a lower hardware cost.

C. Fixed-Point Constant Coefficient Multiplier (Our Work)

We modified the HBU method [6] to develop unsigned and signed CCMs. Since a constant coefficient multiplier is a constant slope monotonic increasing/decreasing function, Equation 1 can be simplified as:

$$f(x) \approx g(x) = f_{base}(x) + \begin{cases} b_1 & 0 \le x < x_1 \\ \dots & \\ b_K & x_{k-1} \le x < x_k \end{cases} \tag{2}$$

$$f_{base}(x) = f(x) \qquad 0 \le x < x_1 \tag{3}$$

Where b_i $(1 \le i \le k)$ are the bias values added to the base function $f_{base}(x)$. It is important to note that Equation 2 implements the multiplication operation with complete accuracy. However, to save hardware, we can implement an approximate version of the function. We use a guiding example to explain the source of error in our approximation. Fig. 3 shows the original and the truncated (hardware optimized) version of a 5-bit CCM using Equation 2. In this example, we divided the input range into two sections ($k = 2$). According to Equation 3, $f_{base}(x)$ is equal to $f(x)$ for $0 \le x < 16$. For $16 \le x < 31$, the proposed method produces the output values $g(x)$ by adding 5, as the bias value, to the base function. By adding 5 to the base function, the method will output 5 for $x = [16, 17]$ and 6 for $x = [18, 19]$, while the correct value for $x = [16, 19]$ is 5. In fact, since our approach might split the

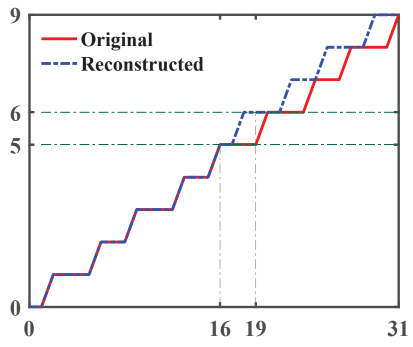


Fig. 3. Aliasing issue phenomena.

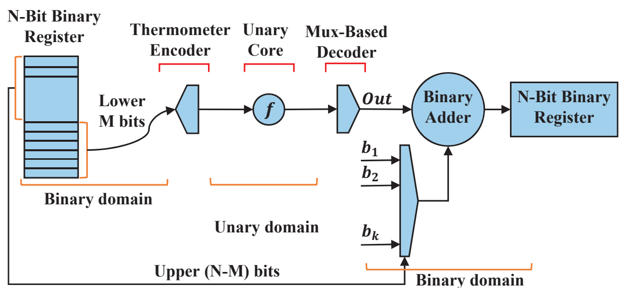


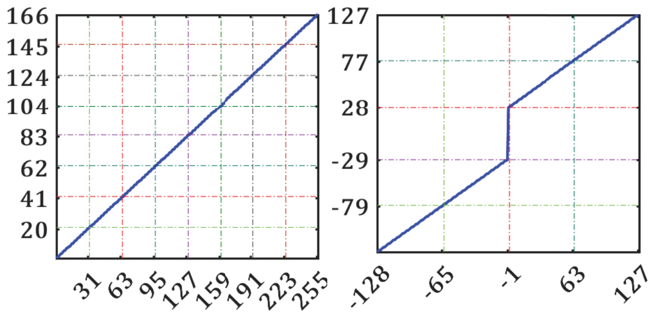
Fig. 4. The overall architecture.

output range of a truncated CCM into non-homogeneous sub-ranges, the error stems from an aliasing issue when designing truncated CCMs. In Section IV, we will evaluate the accuracy of the proposed CCMs for different resolutions by capturing the value and frequency of occurrence of the error.

We used a synthesis methodology similar to what was proposed in [6] to implement equations 2 and 3. Our synthesizer uses just one parameter, K, as opposed to the many parameters in [6]. Parameter K splits the input range into a number of sub-ranges of length 2^K, where $3 \le K \le N-1$, N being the input resolution. Therefore, the modified synthesizer generates $N-4$ unique designs for each CCM and then finds the best design with the minimum hardware cost. The modified synthesizer generates designs based on the proposed HBU CCM architecture shown in Fig. 4. The proposed architecture implements a CCM using equations 2 and 3. The architecture consists of four units: a thermometer encoder, a fully unary computational unit, a decoder, and a bias adder unit. The first stage converts binary numbers corresponding to the base function's input using a thermometer encoder. The proposed method uses the lower M bits of the input value to feed the encoder and uses the remaining $N-M$ upper bits to add an appropriate bias to the output of the decoder. The second stage consists of a fully unary core that implements the base function $f_{base}(x)$ using the fully unary approach (Section II-B). The third stage consists of a multiplexer-based decoder that converts the base function's output to the binary format.

D. Truncated Fixed-Point Constant Coefficient Multiplier

We use two guiding examples to show how the modified synthesizer decomposes and rebuilds a truncated CCM using



(a) Unsigned $f(x) = 0.6523x$ (b) Signed $f(x) = 0.78125x$

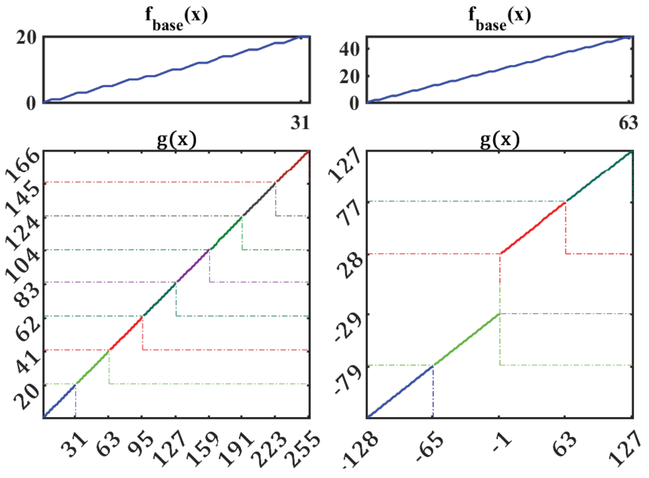
Fig. 5. Unsigned and signed multiplier behaviors.

the proposed architecture. In this paper, we use the term "truncated" to mean multiplying an N-bit input X by a N-bit input constant, and generating an N-bit output using floor/round quantization schemes. Figures 5a and 5b show the behavior of an 8-bit unsigned and signed truncated CCM with positive coefficients, respectively. The x-axis and y-axis show the input and the output range. In these particular examples, the modified synthesizer splits the input range of the unsigned and the signed CCMs into 8 and 4 equal sub-regions, respectively. Since the lengths of these sub-regions are 32- and 64-bits in the unary domain, 5- and 6-bit thermometer encoders and multiplexer-based decoders are needed to implement base functions corresponding to these two CCMs, respectively. Figures 6a and 6b show base functions and reconstructed versions of an unsigned and signed CCM with positive coefficient using equation 2 and 3, respectively, which correspond to figures 5a and 5b, respectively. The base functions look jagged because of quantization, *i.e.*, generating an N-bit output from an N-bit by N-bit multiplication, as opposed to $2N$-bit output. Since the synthesizer splits the input range of the mentioned CCMs into 8 and 4 different sub-regions, the proposed architecture uses 7 and 3 different non-zero biases to reconstruct the original unsigned and signed CCM outputs, respectively. Therefore, the proposed architecture implements the unsigned/signed CCM using a 5/6-bit thermometer encoder, a simple routing network, and a 5/6-bit decoder. It should be noted that the proposed approach can implement CCMs combined with rounding at no extra cost.

E. Non-Truncated Fixed-Point Constant Coefficient Multiplier

In the previous section, we looked at truncated CCMs, *e.g.*, both input and output being N bits wide. In this section, we look at non-truncated constant multiplication, *i.e.*, an N-bit input number multiplied by an P-bit constant value resulting in an $(N + P)$-bit number, hence the term non-truncated CCM. Since a non-truncated CCM has a wider output than a truncated CCM, the cost of the decoder in Fig. 4 increases exponentially and becomes the most costly part of the architecture to implement. To address this increase in cost, we split the coefficient into two sections and perform non-truncated multiplication using each section:

$$C = c_{N-1}...c_0 \rightarrow C_1 = c_{N-1}...c_M,\ C_0 = c_{M-1}...c_0 \quad (4)$$



(a) Unsigned $f(x) = 0.6523x$ (b) Signed $f(x) = 0.78125x$

Fig. 6. Multiplier evaluation using the HBU approach.

where $0 < M < N$. Therefore, a non-truncated multiplication can be re-written as follows:

$$\begin{aligned} f(x) = C \times x = f_1(x) \times 2^M + f_0(x) \\ f_1(x) = C_1 \times x, \quad f_0(x) = C_0 \times x \end{aligned} \quad (5)$$

Where $f_1(x)$ and $f_0(x)$ are non-truncated CCMs. Our experience shows that for 8-bit non-truncated CCMs, the best value for M is either $\frac{N}{2} - 1$, $\frac{N}{2}$, or $\frac{N}{2} + 1$. In fact, choosing a value out of this range for M results in a pair of small and a large decoders for f_1 and f_0. The cost overhead due to the large decoder makes the proposed design unattractive in terms of area and $A \times D$ costs. Splitting the coefficient not only reduces the decoder complexity, but also reduces the total cost by sharing the encoder between partial multipliers if possible. The original input is encoded into the unary format and fed to fully unary cores to perform partial multiplications. The output of each core is decoded to the binary format using smaller decoders compared to the original one. The outputs of the partial multipliers are summed together based on their weights to recover the final non-truncated output. It should be mentioned that the proposed non-truncated CCMs are completely accurate.

F. The Proposed Cost Optimizer

We propose a framework to further reduce the area cost of the proposed CCMs. The proposed optimizer tries to build a coefficient using sub-coefficients with simpler hardware. The proposed optimizer splits a coefficient 'C' into a set of sub-coefficients $C_1, \ldots, C_n$ with the same resolution as 'C' such that $C = \sum_{i=1}^{n} \alpha_i C_i$, where $\alpha_i \in$ {-1, 0, 1}, and the total hardware cost of $\sum_{i=1}^{n} \alpha_i C_i X$ is less than the hardware cost of CX.

To ensure that the cost is reduced, the number of sub-coefficients must be limited; otherwise, the cost of adders will become an overhead. Based on our experience, the maximum number of sub-coefficients should be three. Moreover, another way to reduce the total cost of the set is to share the encoders among the sub-coefficient CCMs. This can be done by selecting a set where the sub-coefficients have encoders of the same size. There are many sets of sub-coefficients for each coefficient. However, only those of them that are very likely to yield lower cost need to be considered. To find optimal or sub-optimal sets, we develop a framework that uses a few constraints to shrink the number of sets.

The proposed framework computes an estimate of the total cost for each set of sub-coefficients by summing the individual costs of each sub-coefficient. It then discards the sets of which the total cost is greater than 70% of the cost of the original CCM, regardless of other constraints. To further reduce the number of candidates, eight groups of sets are sorted based on their priority. The optimizer assigns a factor to each group and removes sets from each group whose total cost exceeds the minimum total cost between all possible candidates weighted by the group's factor. The sets with first priority contain two equal sub-coefficients (e.g. $10X = 5X + 5X$). The sets with second priority contain two non-equal sub-coefficients with the same encoder size. For instance, if the encoder size of $26X$ and $42X$ are equal, then these two coefficients can be a candidate to implement $68X$ ($68X = 26X + 42X$). The third priority is given to sets containing all three equal sub-coefficients. The sets with fourth priority contain three non-equal sub-coefficients with the same encoder size. In these cases, just a single encoder is needed for all sub-coefficient multipliers. The fifth priority is given to sets containing two non-equal sub-coefficients with different encoder sizes (e.g. $40X = 32X + 8X$). The sets with sixth priority have three sub-coefficients of which two out of three sub-coefficients are exactly the same (e.g. $22X = 10X + 10X + 2X$), regardless of the encoder size. The seventh priority is given to sets containing all three sub-coefficients of which two out of three sub-coefficients have the same encoder size, regardless of sub-coefficient values. The last priority is given to sets containing all remaining three sub-coefficients. Then, the framework sorts all remaining sets based on their total cost and chooses the top 20 sets for each coefficient. Finally, the framework generates Verilog codes to implement the candidate sets and then synthesizes them using the Xilinx Vivado 2018.2 default design flow. For optimized truncated CCMs, since all sub-coefficients have at most 1 bit inaccuracy, the optimized designs can have at most 2 bit inaccuracy. In Section IV, we evaluate the accuracy of the best design for each CCM.

III. High-Resolution Constant Coefficient Multiplier

In this section, we propose low-cost, *approximate*, high-resolution CCMs for applications that can tolerate approximation. We use truncated and non-truncated multipliers proposed in sections II-D and II-E as building blocks to design such multipliers. Before we present our proposed approximate architecture, we should mention that the methods of sections II-D and II-E are not scalable with respect to the width of variable input X: no matter how we tune synthesis parameters, hardware cost of the HBU CCMs increases beyond that of Xilinx LogiCORE IP CCMs, and table-based KCM CCMs. If the input is broken into many small sections to bring down the cost of the encoder/decoder blocks, the cost of the multiplexer and the bank of bias values increases prohibitively. As a result, we are forced to break the input variable, as well as the constant into smaller chunks.

We take advantage of the binary format representation to split the multiplicand and the multiplier into three sections to reduce the required encoder and decoder length. We use the pencil-and-paper multiplication method to break a 16-bit CCM into a non-truncated (16-bit output) and two truncated CCMs (8-bit output). Equation 6 illustrates the approach used to implement 16-bit CCMs.

$$\begin{gathered} f(x) = C \times x = f_2(x) + f_1(x) + f_0(x) \\ f_0(x_L) = c_L \underline{\times} x_H, f_1(x_L) = c_H \underline{\times} x_L, f_2(x_H) = c_H \times x_H \\ C = c_{15}...c_0 = c_H \times 2^M + c_L \end{gathered} \tag{6}$$

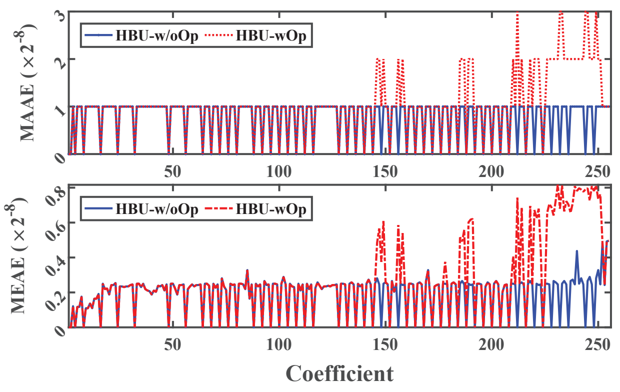
where $\underline{\times}$ represents truncated and $\times$ represents non-truncated multiplication. It follows that for N=8, the output of $f_2(x_H)$ is a non-truncated 16-bit output, while the outputs obtained from $f_0(x_L)$ and $f_1(x_H)$ are truncated 8-bit outputs. Note that we discard the term $c_L \times x_L$, and use the truncated rounded multiplications.

IV. Accuracy Analysis

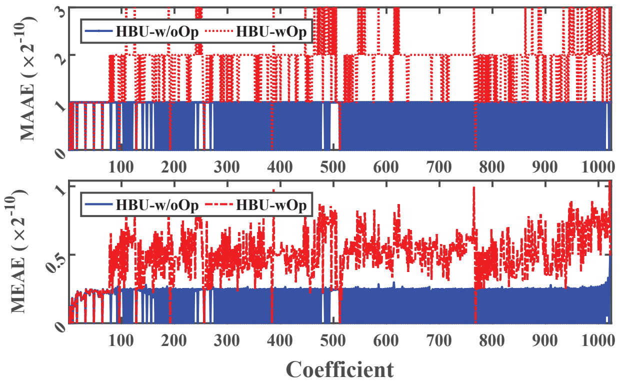
In this section, we evaluate the accuracy of all proposed unsigned CCMs using maximum absolute error (MAAE) and mean absolute error (MEAE) metrics, comparing the proposed CCMs with conventional truncated round CCMs [2]. Fig. 7 shows the error analysis of the proposed 8- and 10-bit CCMs. Moreover, this figure compares the error of the proposed CCMs before applying the proposed cost optimizer (HBU-w/oOp) and the error of the optimal HBU CCMs (HBU-wOp).

As seen in Fig. 7, the maximum absolute error of HBU-w/oOp CCMs is one bit, while the mean absolute error of each CCM is less than $\frac{0.5}{2^N}$ which means each CCM has 0 bit error for most inputs and at most one bit error for the remaining inputs. For instance, MEAE of 0.3 for a particular constant means that the CCM has one bit error for 30% of the input X values, and 0 bit error for the rest. However, the MAAE of the HBU-wOp CCMs can be $\frac{2}{2^N}$ or $\frac{3}{2^N}$ which means that those CCMs can have a maximum of 2-bit error. In fact, when we apply the cost optimizer to fixed point CCMs, the resulting designs can have up to 2 bits of inaccuracy for some inputs since all original sub-coefficients have up to 1 bit of inaccuracy and the errors accumulate. Their MEAE is closer to $\frac{1}{2^N}$ which means that the probability of having a one bit error is higher than the probability of having a one bit error using HBU-w/oOp CCMs, especially for 10-bit truncated CCMs. It should be mentioned that there is a trade-off between cost and accuracy. To have less than one bit inaccuracy on average, one can use another metric instead of minimum area cost to pick sub-optimal designs between the HBU-w/oOp and HBU-wOp CCMs. Therefore, our approximate CCMs can be used

[2]We have used quantization scheme to generate truncated multipliers output

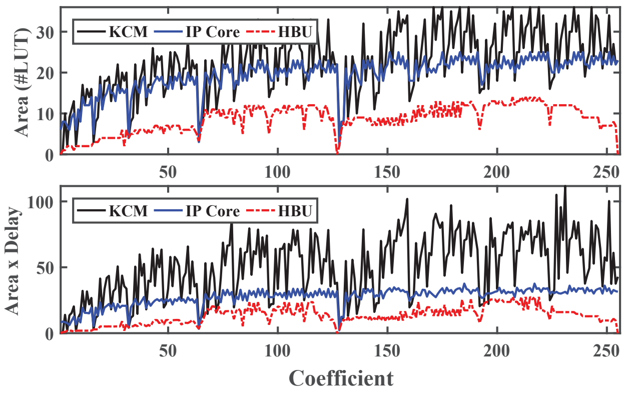


(a) 8-bit truncated CCMs

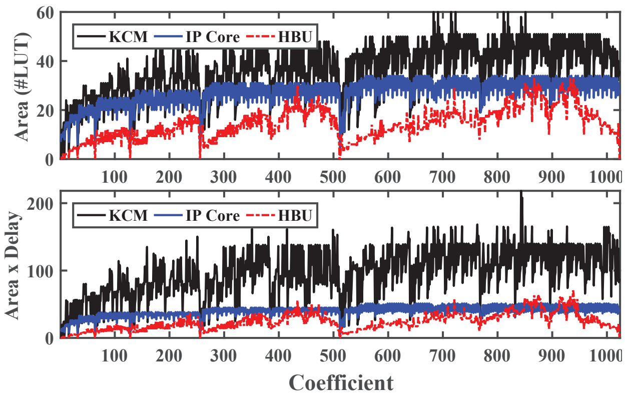


(b) 10-bit truncated CCMs

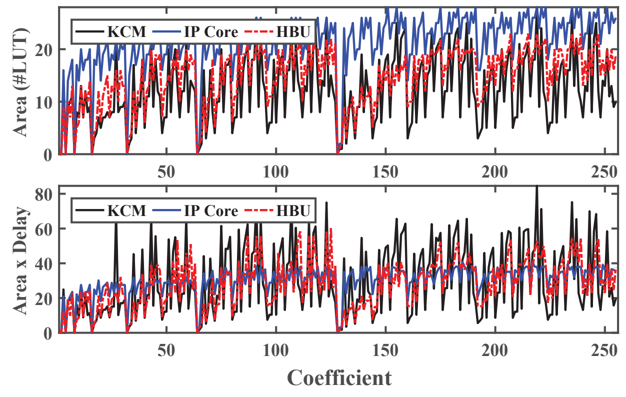
Fig. 7. Error analysis of the proposed truncated CCMs.



(a) 8-bit truncated CCMs



(b) 10-bit truncated CCMs



(c) 8-bit non-truncated CCMs

Fig. 8. Hardware cost comparison of the proposed CCMs.

to implement applications that can tolerate slight inaccuracies, such as image processing applications or neural networks. Also, it should be mentioned that non-truncated HBU-w/oOp and HBU-wOp CCMs are completely accurate (Fig. 7 showed results for truncated architectures).

V. Hardware Cost Evaluation

We developed Verilog hardware descriptions to implement the proposed unsigned CCMs for different resolutions. We evaluate all designs on Kintex7$XC7K70TFBG$676-2 FPGAs and synthesized them using the Xilinx Vivado 2018.2 default design flow. The synthesis clock speed is 250 MHz. We have extracted the implementation costs in terms of area, as the number of LUTs, and $A \times D$ for each coefficient using the proposed architecture, Xilinx LogiCORE IP CCMs, and table-based KCM approach [3], [5]. We have reported hardware costs of the optimized HBU CCMs. To implement each CCM using table-based KCM, we used the decomposition technique mentioned in [5] to split the input into 4-, 5-, and 6-bit chunks as well as the decomposition technique proposed by [3] which results in four different designs. Then, we chose the best design among all designs for each CCM.

Figures 8a, 8b, and 8c show the area and $A \times D$ costs of each coefficient for 8-bit truncated, 10-bit truncated, and 8-bit non-truncated CCMs, respectively. As we can see, the proposed CCMs have lower area and $A \times D$ costs compared to other approaches for most cases. For further evaluation of the proposed CCMs as well as the proposed cost optimizer, Table I compares the area cost statistics of the proposed CCMs against the cost of Xilinx LogiCORE IP CCMs before and after applying the proposed cost optimizer for different resolutions. This table shows the number of cases that the proposed approach has higher, equal, and lower area cost compared to Xilinx LogiCORE IP CCMs. Also, this table reports the total hardware cost of all coefficients for each resolution using the proposed and Xilinx LogiCORE IP CCMs. In this table, the 'Ratio' column is the ratio of the total cost of the proposed CCMs to the total cost of Xilinx LogiCORE IP CCMs. Table I shows that the proposed method can improve the area cost of 7- to 9-bit truncated CCMs by 53% to 25% as well the area

TABLE I
HARDWARE COST COMPARISON STATISTICS OF THE PROPOSED CCMS BEFORE AND AFTER APPLYING OUR PROPOSED OPTIMIZER.

N	Before applying our optimizer						After applying our optimizer				
	IP<HBU	IP=HBU	IP>HBU	Tot.Cost IP(LUT)	Tot.Cost HBU(LUT)	Ratio	IP<HBU	IP=HBU	IP>HBU	Tot.Cost HBU(LUT)	Ratio
7, Truncated	1	1	125	1847	877	0.475	1	1	125	758	0.410
8, Truncated	0	4	251	4950	2529	0.511	0	4	251	2211	0.446
9, Truncated	30	35	446	11794	8882	0.753	2	4	505	6172	0.523
10, Truncated	806	36	181	27802	33034	1.188	11	12	1000	14718	0.529
11, Truncated	1944	6	97	50421	119398	2.368	580	72	1395	41495	0.823
8, non-truncated	0	10	245	5462	3911	0.716	0	10	245	3738	0.684

TABLE II
JPEG ENCODER ACCURACY RESULTS FOR 8-BIT RESOLUTION.

Design approach	PSNR (dB)	SSIM
Floating-point	39.95	0.9634
Binary Fixed-point	39.86	0.9613
The proposed CCMs-Original	39.76	0.9603
The proposed CCMs-Optimized	39.44	0.9581

cost of 8-bit non-truncated CCMs by 29% without applying the proposed optimizer. As we can see, it cannot beat Xilinx LogiCORE IP for high-resolution CCMs, such as 10- and 11-bit truncated CCMs. However, after applying our optimizer, the proposed method can beat Xilinx LogiCORE IP-based CCMs and improve the area cost of the reported resolutions from 59% to 18% on average. For example, if we add up the area cost of all LogiCORE IP-based and optimized HBU-based of 8-bit truncated CCMs, the optimized HBU reduces the area cost by 54%. Another technique to implement high-resolution CCMs is the coefficient-splitting technique, and will be discussed in Section III. The implementation cost results of 16-bit truncated CCMs, which are used to implement parallel fast Fourier transform (FFT) architectures, will be discussed in Section VI-B.

VI. CASE STUDIES

In this section, we evaluate the proposed CCMs on two common digital signal processing (DSP) applications: 2-D DCT and Radix-2 Decimation in Time-FFT (DIT-FFT). We have implemented these applications using Xilinx LogiCORE IP CCMs, referred to as the binary architecture, and the proposed optimized CCMs, referred to as the HBU architecture. All designs have been evaluated on Kintex7$XC7K70TFBG$676-2 FPGAs and placed and routed them using Xilinx Vivado 2018.2 with default design flow. The synthesis clock speed is 250 MHz. However, we have reduced the clock frequency to 100 MHz in order to place and route 128-point Radix-2 DIT-FFT.

A. 2-D DCT

We introduce a new fully parallel HBU 2-D DCT engine to implement a JPEG encoder, which results in remarkable improvements in hardware cost and throughput compared to Xilinx LogiCORE IP CCMs-based engine. A major portion of a JPEG encoder's computational complexity is due to the DCT unit. We have evaluated the performance of a JPEG encoder in terms of Peak Signal to Noise Ratio (PSNR) and structural similarity (SSIM) using floating- and fixed-point operations in MATLAB. We realized that using floor truncation in signed multiplication results in a significant drop in accuracy, while using rounding in signed multiplication or flooring in sign-magnitude multiplication[3] keeps the accuracy the same as that of the non-truncated implementation. Therefore, we used sign-magnitude multiplication to evaluate and implement the 2-D DCT algorithm. We have evaluated the performance of the JPEG encoder using the proposed CCMs before and after applying the proposed cost optimizer, the original and the optimized CCMs, respectively. For the accuracy test and analysis, we used the quantization matrix corresponding to the quality factor of 90%. We also used Lena512 image as a test case. Table II shows the accuracy analysis of a JPEG encoder. It shows that the drop in accuracy of the 8-bit fixed-point implementation compared to the floating-point implementation is negligible. Moreover, the table shows that the drop in accuracy using the proposed original CCMs and the proposed optimized CCMs is around 0.1-0.3-dB compared to 8-bit accurate fixed-point Xilinx LogiCORE IP CCMs, which is negligible.

We have developed the HDL code to implement a 2D-DCT architecture using Xilinx LogiCORE IP CCMs and the proposed optimized CCMs for 8-bit resolution. The proposed kernel uses 64 pipelined-parallel sub-kernels to produce all output elements in parallel. Each kernel has 64 CCMs and a fully pipelined adder tree. The latency of an IP core CCM-based and the proposed CCM-based sub-kernel are 10 and 8 clock cycles, respectively. We have implemented the architectures using IP core CCMs (IP CCM) and the proposed CCMs (HBU CCM) with cost optimization. Table IV shows hardware implementation results. Columns 2-4 show the number of LUTs, the number of FFs and the critical path delay, respectively. Columns 5-7 show the throughput in Mega Samples per Seconds, power and energy per sample, respectively. The last column shows the total $A \times D$ value. The table shows that the HBU architecture results in 34% and 37% improvement in terms of area and $A \times D$, respectively. Moreover, our architecture not only reduces the energy consumption per sample by 36%, but also improves the throughput by 4%. The 8K and 4K ultra HD images have around 33.2M and 8.2M pixels, respectively. The proposed kernel can be used in real 8K and 4K video processing, because it can process 783 and 3165 frames per second of 8K and 4K UHD streams, respectively. These numbers can be improved by pipelining

[3]We use the floor operator to truncate the output magnitude.

TABLE III
FFT hardware implementation results. 'T' is the (Giga samples per second) throughput, 'P' is power, and 'E/S' is energy per sample consumption. We do not have data on the power consumption of 256-point design as Vivado crashes on our computers.

FFT Size	Binary approach (Xilinx LogiCORE IP CCM-based)							HBU approach							Ratio			
	LUT	D(ns)	T(GS)	P(W)	E/S(pJ)	$A \times D$	SNR(dB)	LUT	D(ns)	T(GS)	P(W)	E/S(pJ)	$A \times D$	SNR(dB)	T	Area	$A \times D$	E/S
8-Point	$4.07K$	2.76	2.89	0.19	67	$1.12e$-4	Inf	$3.94K$	2.93	2.73	0.17	62.6	$1.15e$-4	Inf	0.94	0.96	1.0	0.93
16-Point	$12.8K$	2.88	5.57	0.50	90	$3.68e$-4	79.9	$12.0K$	3.33	4.80	0.43	90.2	$4.03e$-4	77.5	0.86	0.94	1.0	1.0
32-Point	$36.1K$	3.11	10.3	1.29	125	$11.2e$-4	76.3	$35.0K$	3.43	9.32	1.17	125	$12.0e$-4	73.7	0.90	0.97	1.0	1.0
64-Point	$90.0K$	3.42	18.7	2.87	154	$3.90e$-4	72.8	$88.2K$	3.49	18.3	2.53	138	$30.8e$-4	70.3	0.98	0.97	0.99	0.89
128-Point	$224K$	8.68	14.7	7.19	487	$194e$-4	69.6	$219K$	4.92	26.0	5.85	225	$108e$-4	67.1	1.76	0.98	0.55	0.46
256-Point	$493K$	8.96	25.6	–	–	$441e$-4	66.4	$483K$	6.31	40.6	–	–	$304e$-4	64.1	1.58	0.97	0.69	–

TABLE IV
2-D DCT hardware implementation results using Xilinx LogiCORE IP CCMs (IP CCM-based) and the proposed CCMs (HBU CCM-based).

Design method	LUT	FF	D(ns)	T(MS)	P(W)	E/S(pJ)	$A \times D$
IP CCM	$109K$	$93K$	2.57	389	3.63	145	$280e$-6
HBU CCM	$72K$	$57K$	2.46	406	2.43	93	$177e$-6
Ratio	0.66	0.61	0.95	1.04	0.67	0.64	0.63

TABLE V
Hardware cost comparison statistics of the proposed 16-bit truncated CCMs.

Bin<HBU	Bin=HBU	Bin>HBU	Tot.Cost Bin(LUT)	Tot.Cost HBU(LUT)	Ratio
5	2	248	16962	12356	0.728

and retiming techniques.

B. Fast Fourier Transform

We introduce a pipelined-parallel Radix-2 DIT-fast Fourier transform (FFT) architecture using the HBU CCMs, which results in a processing power of 321 Giga samples per second. The parallel FFT can be used to implement high order filters for many DSP applications such as compensation of chromatic dispersion inherent in optical fibers, object detection over large bandwidths using radar, cellular communication, and so on. As we mentioned in Section III, we used the proposed truncated and non-truncated 8-bit CCMs to implement 16-bit truncated CCMs based on Equation 6. We only implemented the required CCMs to implement 8- to 128-point Radix-2 16-bit fixed-point DIT-FFT. Table V shows the hardware cost comparison statistics of the proposed 16-bit truncated CCMs. The proposed optimized 16-bit truncated CCMs can beat the 16-bit truncated fixed-point Xilinx LogiCORE IP CCMs in 97.6% of the coefficients. The proposed approach can decrease the area cost of the required CCMs by 54.65%. To evaluate the proposed multipliers, we have implemented different sizes of 16-bit fixed-point DIT-FFT with the Xilinx LogiCORE IP and the proposed optimized CCMs. Each stage has a latency of 4 clock cycles in the binary and the HBU architectures. We have reported the hardware cost in Table III. As we can see, the proposed architecture improves power and the energy per sample by 16% and 14% on average for 8- to 128-point Radix-2 DIT-FFT. These number can be improved by using different constraints for place and route steps. We have evaluated the accuracy of fixed-point FFT engines against floating-point engines using signal-to-noise ratio (SNR). Table III shows that the SNR of our designs drops around 2.5dB (0.415 in terms of bits) based on the level of quantization error induced by our proposed truncated CCMs.

TABLE VI
Routability resource utilization test results.

Design	FPGA	LUT(K)		Resource Utilization
		Capacity	Used	
3× HBU	xc7vx415tffv1927-1	260	223	85.2%
2× LogiCORE	xc7vx415tffv1927-1	260	unroutable	
2× LogiCORE	xc7k480tiffv901-2L	298	270	90.5%

VII. Routability Resource Utilization Test

Given that our method uses routing resources to perform "logic", one might be concerned that even though it uses fewer LUTs, it might use more routing resources and hence be unroutable when chip utilization is high. We designed an experiment to test this hypothesis. As mentioned in Section VI-A, a fully parallel 8×8 HBU CCM-based 2D-DCT engine needs 30% fewer LUTs than LogiCORE IP CCM-based 2D-DCT engine. As a result, in theory one should be able to use a small FPGA that can fit two fully-routed copies of LogiCORE IP DCT blocks with high utilization, and use the same FPGA to fit three copies of HBU DCT engines. In our experiments, the opposite of what we expected happened: we could fit three copies of the HBU DCT engine on an FPGA with 85.2% logic resource utilization, but could *not* successfully place and route two LogiCORE IP DCT engines on the same FPGA. We then used larger and larger FPGAs to be able to fit two or even three LogiCORE IP DCT engines. Table VI shows the post place and route results. This shows that our method has *less* routability issues compared to binary implementations.

VIII. Conclusions

We proposed a novel HBU approximate CCM with lower costs compared to Xilinx LogiCORE IP and table-based KCM CCMs on average. We evaluated the proposed multipliers on two common DSP algorithms: 8-bit fixed-point 2-D DCT and 16-bit fixed-point FFT. The proposed architecture solidly outperforms the binary architecture implemented using Xilinx LogiCORE IP CCMs on average in terms of {area × delay, throughput, power, and energy per sample consumption}, by {37%, 4%, 33%, and 36%} for the 2-D DCT algorithm and by {6%, 9%, 16%, and 14%} for the FFT algorithm. Moreover, we showed that our method has less routability issues compared to binary implementations at least in DCT.

References

[1] O. Gustafsson *et al.*, “Simplified design of constant coefficient multipliers”, *Circuits, Systems and Signal Processing*, vol. 25, no. 2, pp. 225–251, Apr. 2006.

[2] K. Chapman, “Fast integer multipliers fit in fpgas”, in *EDN magazine*, 1993, p. 80.

[3] E. George Walters, “Reduced-area constant-coefficient and multiple-constant multipliers for xilinx fpgas with 6-input luts”, English (US), *Electronics*, vol. 6, no. 4, Dec. 2017.

[4] V. Dimitrov, L. Imbert, and A. Zakaluzny, “Multiplication by a constant is sublinear”, in *18th IEEE Symposium on Computer Arithmetic (ARITH ’07)*, Jun. 2007, pp. 261–268.

[5] F. de Dinechin *et al.*, “Table-based versus shift-and-add constant multipliers for fpgas”, in *ARITH 2019 - 26th IEEE Symposium on Computer Arithmetic*, Kyoto, Japan: IEEE, 2019, pp. 1–8.

[6] S. R. Faraji and K. Bazargan, “Hybrid Binary-Unary Hardware Accelerators”, in *2019 24th Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan. 2019.

[7] A. Alaghi, W. Qian, and J. P. Hayes, “The promise and challenge of stochastic computing”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. PP, no. 99, pp. 1–1, 2017.

[8] A. Ardakani, F. Leduc-Primeau, and W. J. Gross, “Hardware implementation of fir/iir digital filters using integral stochastic computation”, in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Mar. 2016, pp. 6540–6544.

[9] K. Han *et al.*, “Stochastic bit-wise iterative decoding of polar codes”, *IEEE Transactions on Signal Processing*, vol. 67, no. 5, pp. 1138–1151, Mar. 2019.

[10] H. Sim and J. Lee, “A New Stochastic Computing Multiplier with Application to Deep Convolutional Neural Networks”, in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, Jun. 2017.

[11] V. T. Lee *et al.*, “Architecture considerations for stochastic computing accelerators”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2277–2289, Nov. 2018.

[12] M. H. Najafi *et al.*, “Accelerating deterministic bit-stream computing with resolution splitting”, in *20th International Symposium on Quality Electronic Design (ISQED)*, Mar. 2019, pp. 157–162.

[13] S. R. Faraji *et al.*, “Energy-Efficient Convolutional Neural Networks with Deterministic Bit-Stream Processing”, in *Design, Automation, and Test in Europe (DATE), 2019*, Mar. 2019.

[14] S. Liu and J. Han, “Energy efficient stochastic computing with sobol sequences”, in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, Mar. 2017, pp. 650–653.

[15] M. H. Najafi, D. J. Lilja, and M. Riedel, “Deterministic Methods for Stochastic Computing Using Low-discrepancy Sequences”, in *Proceedings of the International Conference on Computer-Aided Design*, ser. ICCAD ’18, San Diego, California: ACM, 2018, 51:1–51:8.

[16] P. Ting and J. P. Hayes, “Maxflow: Minimizing latency in hybrid stochastic-binary systems”, in *Proceedings of the 2018 on Great Lakes Symposium on VLSI*, ser. GLSVLSI ’18, Chicago, IL, USA: ACM, 2018, pp. 21–26.

[17] S. Mohajer, Z. Wang, and K. Bazargan, “Routing magic: Performing computations using routing networks and voting logic on unary encoded data”, in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA ’18, Monterey, CALIFORNIA, USA: ACM, 2018, pp. 77–86.