A Novel Low-Power and High-PSNR Architecture Based on ARC for DCT/IDCT

Yiliu Feng(✉), Jianfeng Zhang, and Hengzhu Liu

National University of Defense and Technology, Changsha, Hunan, China
endlessfyl@gmail.com

Abstract. Discrete cosine transform (*DCT*) and its inverse (*IDCT*) play a key role in image and video systems. In this paper, we propose an efficient DCT/IDCT architecture based on adaptive recoding coordinate rotation digital computer (*ARC*), which has been validated on an FPGA platform. Compared to the state-of-the-art DCT, the proposed architecture dissipates 8.2% less power and improves PSNR by 3.21 dB while maintaining nearly the same area and speed. The proposed architecture uses 37.6% less hardware resources, saves 31.6% in power dissipation, provides a 2.15 times speed-up and improves PSNR slightly when compared with the newest DCT/IDCT architecture.

Keywords: ARC · CORDIC · DCT · FPGA · IDCT · PSNR

1 Introduction

Recent years have experienced a significant demand for low-power implementations of algorithms in image and video processing systems, especially like discrete cosine transform *(DCT*) and inverse discrete cosine transform (*IDCT*). The reason is that DCT and IDCT are one of the most computationally intensive transforms in image and video compression standards, which have been demonstrated to provide perfect energy packing for images [1] and are very close approximation of the optimal Karhunen-Loeve transform (*KLT*) [2], such as JPEG [3], MPEG [4] and HEVC [5].

Hence, many fast algorithms are proposed to reduce the computational complexity of computing DCT/IDCT based on three different methods: multiplier-based algorithms [6], distributed arithmetic (*DA*) based algorithms [7] and coordinate rotation digital computer (*CORDIC*) based algorithms [8–12]. As the multiplier has high hardware complexity, which restricts the computation speed, and the ROM size increases exponentially with the transform length for DA-based DCT, it is an efficient way to implement DCT and IDCT based on CORDIC, which only requires shifters and adders. Huang et al. [9–11] presented a radix-2 unified DCT and IDCT algorithm based on unfolded CORDIC, but it consumes too many hardware resources and dissipates too much power. Lee et al. [12] discussed a low-power DCT based on look ahead CORDIC, while the peak signal-to-noise ratio (*PSNR*) is reduced.

In this paper, we propose an efficient multiplier less unified DCT and IDCT architecture based on adaptive recoding CORDIC (*ARC*) [13] combined with the conventional CORDIC [14]. ARC accelerates the vector rotation and improves the

W. Xu et al. (Eds.): NCCET 2016, CCIS 666, pp. 55–68, 2016.
DOI: 10.1007/978-981-10-3159-5_6

accuracy with no performance penalty, and the post-processing of DCT/IDCT is correlated with the scale factor of the conventional CORDIC. Therefore, compared to existing DCT and unified DCT/IDCT architectures, the proposed one has outstanding performance. The proposed architecture has been synthesized on Xilinx Virtex-5 LX150T to verify the correctness and performance.

The rest of this paper is organized as follows: In Sect. 2, we discuss the efficient unified DCT and IDCT architecture. The vector rotation schemes are described in Sect. 3. Section 4 analyzes the simulation and comparison results. Conclusions are drawn in Sect. 5.

2 Efficient Unified DCT/IDCT Architecture

As a 2-D DCT is commonly calculated by first applying a 1-D DCT over the rows followed by another 1-D DCT applied to the columns of the input matrix [12], 1-D DCT is the kernel processing element. Meanwhile, both DCT and IDCT are used in image and video systems, and then designing a unified efficient architecture for 1-D DCT and IDCT is very important. In this paper, we propose a novel unified architecture for 1-D DCT and IDCT based on CORDIC.

The N-point 1-D DCT transforms a real data sequence from time domain $\{x(n), n = 0, 1, 2, \ldots, N-1\}$ to frequency domain $\{y(n), n = 0, 1, 2, \ldots, N-1\}$, which is defined as

$$y(k) = \sqrt{\frac{2}{N}} * c(k) * \sum_{n=0}^{N-1} x(n) * \cos\left[\frac{(2n+1)k\pi}{2N}\right], (k = 0, 1, \ldots, N-1) \tag{1}$$

Where $c(0) = 1/\sqrt{2}$ and $c(k) = 1$ for $k = 0, 1, \ldots, N-1$. As the 8-point DCT is commonly used in image and video standards [3–5], we can decompose the 8-point DCT into four parts. First, the 8-point input signals are preprocessed and represented as follows:

$$\begin{cases} a_0 = x(0) + x(7) \\ a_1 = x(0) - x(7) \end{cases}, \begin{cases} a_2 = x(1) + x(6) \\ a_3 = x(1) - x(6) \end{cases} \\ \begin{cases} a_4 = x(2) + x(5) \\ a_5 = x(2) - x(5) \end{cases}, \begin{cases} a_6 = x(3) + x(4) \\ a_7 = x(3) - x(4) \end{cases} \tag{2}$$

Then, the outputs of the 8-point DCT can be written as:

$$\begin{bmatrix} y(0) \\ y(4) \end{bmatrix} = \frac{1}{\sqrt{8}} \times \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \times \begin{bmatrix} a_0 + a_6 \\ a_2 + a_4 \end{bmatrix} \tag{3}$$

$$\begin{bmatrix} y(2) \\ y(6) \end{bmatrix} = \frac{1}{2} \times \begin{bmatrix} c_2 & -s_2 \\ s_2 & c_2 \end{bmatrix} \times \begin{bmatrix} a_0 - a_6 \\ a_4 - a_2 \end{bmatrix} \tag{4}$$

$$\begin{bmatrix} y(1) \\ y(7) \end{bmatrix} = \frac{1}{2} \times \begin{bmatrix} c_1 & -s_1 \\ s_1 & c_1 \end{bmatrix} \times \begin{bmatrix} a_1 \\ -a_7 \end{bmatrix} + \frac{1}{2} \times \begin{bmatrix} c_3 & s_3 \\ -s_3 & c_3 \end{bmatrix} \times \begin{bmatrix} a_3 \\ a_5 \end{bmatrix}$$
$$= \frac{1}{2} \times \begin{bmatrix} c_1 & -s_1 \\ s_1 & c_1 \end{bmatrix} \times \begin{bmatrix} a_1 \\ -a_7 \end{bmatrix} + \frac{1}{\sqrt{8}} \times \begin{bmatrix} c_3 & -s_1 \\ s_1 & c_1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} \times \begin{bmatrix} a_3 \\ a_5 \end{bmatrix} \tag{5}$$

$$\begin{bmatrix} y(5) \\ y(3) \end{bmatrix} = \frac{1}{2} \times \begin{bmatrix} c_3 & s_3 \\ -s_3 & c_3 \end{bmatrix} \times \begin{bmatrix} a_7 \\ a_1 \end{bmatrix} - \frac{1}{2} \times \begin{bmatrix} c_1 & -s_1 \\ s_1 & c_1 \end{bmatrix} \times \begin{bmatrix} a_3 \\ a_5 \end{bmatrix}$$
$$= \frac{1}{\sqrt{8}} \times \begin{bmatrix} c_1 & -s_1 \\ s_1 & c_1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} \times \begin{bmatrix} a_7 \\ a_1 \end{bmatrix} - \frac{1}{2} \times \begin{bmatrix} c_1 & -s_1 \\ s_1 & c_1 \end{bmatrix} \times \begin{bmatrix} a_3 \\ a_5 \end{bmatrix} \tag{6}$$

in which $c_m = \cos(m\pi/16)$ and $s_m = \sin(m\pi/16)$. As the rearranged 8-point DCT is presented as vector rotation matrices, it can be realized by CORDIC. For Eqs. (4), (5) and (6), we can put the constant scale factors into CORDIC to save area and power, and reduce the truncation and quantization errors.

Figure 1 shows the data flow of the DCT architecture, which consists of three adder arrays (*Adder_1, Adder_2, and Adder_3*), two processing elements (*PE_1, PE_2*) and the DCT/IDCT mode controller. The Adder_1 and Adder_2 consist of four adders and four subtractors, and two adders and two subtractors, respectively. The Adder_3 is made up of one adder, one subtractor and two constant scale factor $1/\sqrt{8}$ compensation logics. The ARC rotation ARC_1, the rotation angle of which is $\pi/8$, and the scale

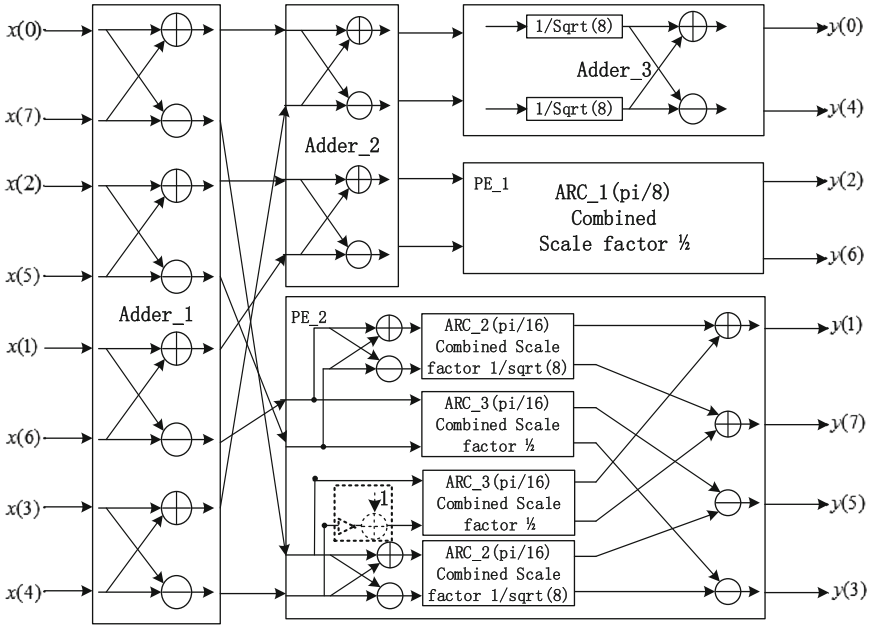

Fig. 1. Data flow of the DCT architecture.

factor is combined with the constant value $1/2$, builds the PE_1. The PE_2 consists of one inverter, five adders, four subtractors, two identical ARC_2 rotators, whose rotation angle is $\pi/16$ and the scale factor is combined with $1/2$, and two ARC_3, the rotation angle of which is also $\pi/16$, but the scale factor is combined with $1/\sqrt{8}$. According to Eqs. (5) and (6), the two outputs of ARC_2 scaled by $1/\sqrt{8}$ pass to the following adder and subtractor to rotate the results by $-3\pi/16$. The inverter and the following constant adder in the black dotted box are to get the 2's complementary of.

In the meantime, the 8-point IDCT is represented as:

$$x(n) = \sqrt{\frac{2}{N}} \times \sum_{n=0}^{N-1} c(k) \times y(k) \times \cos\left[\frac{(2n+1)k\pi}{2N}\right], (n = 0, 1, \ldots, N-1) \tag{7}$$

We can also use the decomposition method. First, the input {y(n), n = 0, 1, 2,…, N−1} are proprocessed and expressed as

$$\begin{cases} \begin{bmatrix} b_1 \\ b_0 \end{bmatrix} = \frac{1}{\sqrt{8}} \times \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} \times \begin{bmatrix} y(0) \\ y(4) \end{bmatrix}, \begin{bmatrix} b_3 \\ b_2 \end{bmatrix} = \frac{1}{2} \times \begin{bmatrix} c_2 & -s_2 \\ s_2 & c_2 \end{bmatrix} \times \begin{bmatrix} y(6) \\ y(2) \end{bmatrix} \\ \begin{bmatrix} b_5 \\ b_4 \end{bmatrix} = \frac{1}{2} \times \begin{bmatrix} c_1 & -s_1 \\ s_1 & c_1 \end{bmatrix} \times \begin{bmatrix} y(7) \\ y(1) \end{bmatrix}, \begin{bmatrix} b_7 \\ b_6 \end{bmatrix} = \frac{1}{2} \times \begin{bmatrix} c_1 & -s_1 \\ s_1 & c_1 \end{bmatrix} \times \begin{bmatrix} y(3) \\ y(5) \end{bmatrix} \\ \begin{bmatrix} b_9 \\ b_8 \end{bmatrix} = \frac{1}{\sqrt{8}} \times \begin{bmatrix} c_1 & -s_1 \\ s_1 & c_1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} \times \begin{bmatrix} y(7) \\ y(1) \end{bmatrix} \\ \begin{bmatrix} b_{11} \\ b_{10} \end{bmatrix} = \frac{1}{\sqrt{8}} \times \begin{bmatrix} c_1 & -s_1 \\ s_1 & c_1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} \times \begin{bmatrix} y(3) \\ y(5) \end{bmatrix} \end{cases} \tag{8}$$

Substituting Eq. (8) into (7), we can get

$$\begin{cases} x(0) = (b_1 + b_2) + (b_4 + b_{11}) \\ x(7) = (b_1 + b_2) - (b_4 + b_{11}) \end{cases}, \begin{cases} x(2) = (b_0 + b_3) + (b_9 - b_7) \\ x(5) = (b_0 + b_3) - (b_9 - b_7) \end{cases}$$
$$\begin{cases} x(1) = (b_0 - b_3) + (b_8 - b_6) \\ x(6) = (b_0 - b_3) - (b_8 - b_6) \end{cases}, \begin{cases} x(3) = (b_1 - b_2) + (b_{10} - b_5) \\ x(4) = (b_1 - b_2) - (b_{10} - b_5) \end{cases} \tag{9}$$

The corresponding data flow of the IDCT architecture is shown in Fig. 2.

Note that the dataflow is from right to left to keep the layout of the functional units the same as in Fig. 1. The required modules in the IDCT, including Adder_1, Adder_2, Adder_3 and PE_1, are the same as the ones in the DCT except for PE_2. The inverter and the adder in the black dotted box in PE_2 of the IDCT is connected to the output of ARC_3, while the output of the black dotted box in PE_2 of the DCT is at the input of ARC_3. Meanwhile, the signal flows between these modules of IDCT and DCT are different.

Figure 3 shows the data flow of the unified architecture for DCT and IDCT, which consists of three adder arrays (*Adder_1, Adder_2, and Adder_3*), two processing elements (*PE_1, PE_2*) and the DCT/IDCT mode controller. The DCT/IDCT mode controller is to reconfigure the architecture to ensure the unified structure can work well

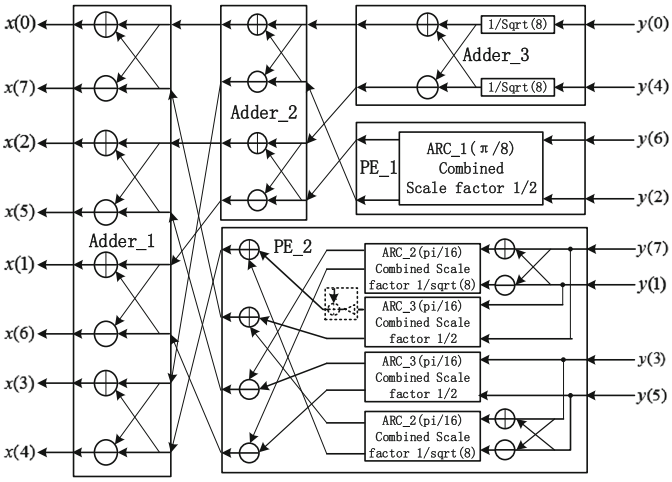

Fig. 2. Data flow of the IDCT architecture.

for DCT mode and IDCT mode, the signals of which are illustrated as the purple arrows. The inverter in the PE_2 marked with black dotted arrows are the DCT and IDCT data flows respectively, and the blue arrows are the signals which can either be DCT or IDCT data flows.

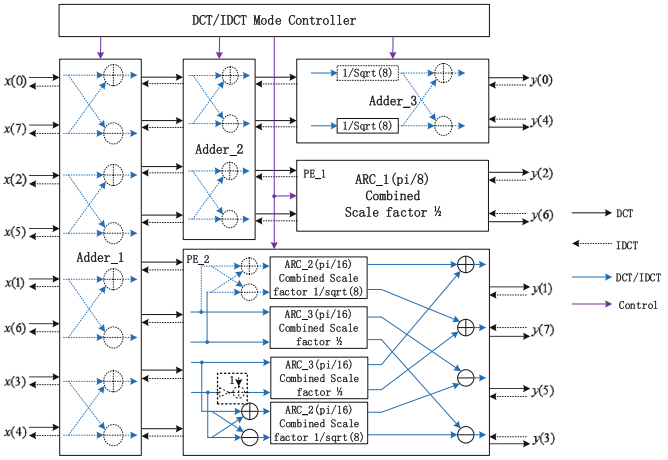

Fig. 3. Data flow of unified DCT and IDCT.

3 ARC Rotation

As discussed in Sect. 2, the unified DCT and IDCT architecture requires three different vector rotation: ARC_1, ARC_2 and ARC_3. In this Section, we will analyze how to realize these rotations based on ARC. Meanwhile, the scale factor compensation for in the Adder_3 is also discussed. To conduct a fair comparison, we also assume the word length is 12-bit, and the natural number 1 equals 12'b010000000000.

3.1 First Rotation ARC_1

The rotation angle $\pi/\sqrt{8}$ of ARC_1 is represented as 12'b010000000000. We first implement the two conventional $i = 2$ and $i = 3$ CORDIC iterations to diminish the rotation angle, and then use $i = 7$ and $i = 8$ iterations of ARC to finish the residual angle rotation.

The basic rotation Matrix of the conventional CORDIC is expressed as

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = \begin{bmatrix} 1 & -2^{-i} \\ 2^{-i} & 1 \end{bmatrix} \times \begin{bmatrix} y_i \\ x_i \end{bmatrix} \tag{10}$$

and the basic iteration of ARC is written as

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = \begin{bmatrix} 1 - 2^{-2i+1} & -2^{-i+1} \\ -2^{-i+1} & 1 - 2^{-2i+1} \end{bmatrix} \times \begin{bmatrix} y_i \\ x_i \end{bmatrix} \tag{11}$$

Therefore, the iteration sequencing of ARC_1 is

$$\begin{bmatrix} 1 & -2^{-2} \\ 2^{-2} & 1 \end{bmatrix}, \begin{bmatrix} 1 & -2^{-3} \\ 2^{-3} & 1 \end{bmatrix}, \begin{bmatrix} 1 & -2^{-6} \\ 2^{-6} & 1 \end{bmatrix}, \begin{bmatrix} 1 & -2^{-7} \\ 2^{-7} & 1 \end{bmatrix} \tag{12}$$

where the first and the fourth iterations, and the other two iterations can be replaced by two iterations to accelerate the rotation process, respectively. The replaced iterations are illustrated as

$$\begin{bmatrix} 1 & -2^{-2} - 2^{-7} \\ 2^{-2} + 2^{-7} & 1 \end{bmatrix}, \begin{bmatrix} 1 & -2^{-3} - 2^{-6} \\ 2^{-3} + 2^{-6} & 1 \end{bmatrix} \tag{13}$$

where the components right shifting 9 bits have been eliminated not only to simplify the architecture, but also to save hardware resources.

Compared to the separate iterations, the results have been slightly amplified after operated by Eq. (13). However, we can compensate for the rounding errors in the scale factor compensation units.

As the scale factor $\cos(\arctan(2^{-2}) \times \cos\tan(2^{-3}))$ of conventional CORDIC needs to be scaled by $1/2$, it is hard for hardware implementation. We propose an adder and shifter-based approximation to approach the new scale factor, the method is shown in Algorithm 1.

The scale factor k_i of ARC_1, the required accuracy R_Accuracy and the initial value of the achieved accuracy A_Accuracy are first set. For lines 5–9, we propose five different approximation expressions to approach the scale factor, and each part of the proposed expressions has the same critical path delay as the required iterations depicted in Eq. (13). As the scale factor is always smaller than natural number 1, lines 5–9 cover all the cases of two separate parts multiplication. The reason for using two parts multiplication to approximate the scale factor is that only one separate part cannot satisfy the required accuracy. Then, the approximation expression which has the smallest error is computed and set to the calculated accuracy in line 10. Third, the C_Accruacy is assigned to the A_Accuracy and the corresponding values consisted of the approximation expression are stored under the condition that if the C_Accuracy is smaller than the R_Accuracy and the A_Accuracy in lines 11–16. After opeated by lines 1–20, we can get the best approximation expression and the corresponding values of its components, the accuracy of which is highest for all tested values.

Algorithm 1 Scale Factor Approximation

```
Initialization:
Set k_1=1/(2×√(1+2^(-4)×√(1+2^(-6)) ));R_Accuracy= 2^(-10);A_Accuracy=0;
Iteration:
1: for m = 1 to 10 do
2:   for n = 1 to 10 do
3:     for k = 1 to 10 do
4:       for l = 1 to 10 do
5:         A_1=(1+2^(-m)+2^(-n))×(1+2^(-k)-2^(-l))
6:         A_2=(1+2^(-m)+2^(-n))×(1-2^(-k)-2^(-l) )
7:         A_3=(1+2^(-m)-2^(-n))×(1+2^(-k)-2^(-l))
8:         A_4=(1+2^(-m)-2^(-n))×(1-2^(-k)-2^(-l))
9:         A_5=(1-2^(-m)-2^(-n))×(1-2^(-k)-2^(-l))
10:          Set C_Accuracy = Minimum(|k_1-A_i|,iϵ[1,5])
11:          if C_Accuracy < R_Accuracy then
12:            if C_Accuracy < A_Accuracy then
13:              A_Accuracy = C_Accuracy;
14:              q = i; w = m; r = n; t = k; s = l;
15:            end if
16:          end if
17:        end for
18:      end for
19:   end for
20: end for
21: return A_Accuracy, q, w, r, s, t;
```

The A_Accuracy, the corresponding values and the type of the approximation expression are returned in line 21. After using Algorithm 1, the scale factor k_1 is represented as:

$$
\begin{aligned}
k_1 &= 1/2 * \cos(\arctan(2^{-2}) \times \cos\tan(2^{-3})) \\
&= \frac{1}{2 \times \sqrt{1+2^{-4}} \times \sqrt{1+2^{-6}}} \\
&\cong (1 - 2^{-1} + 2^{-6}) \times (1 - 2^{-4} - 2^{-8})
\end{aligned} \tag{14}
$$

To compensate for the rounding errors in Eq. (13), we scale the approximation Eq. (14) into $(1 - 2^{-1} + 2^{-6}) \times (1 - 2^{-4} - 2^{-8})$ after using Matlab to verify the accuracy of the results.

Figure 4 shows the data flow of ARC_1, which only consist of shifters and adders, and the "−" symbols placed near the arrows represent subtractions A pipeline balancing method is proposed to implement the architecture, which is divided into four stages. The first two stages are to execute the required iterations, and the scale factor compensation is implemented in the remaining stages. We have split the bigger shifter in evert stage into two subshifters: one has the same right shifting bits as the smaller shifter and the other executes the rest bits shifting, which means that the bigger shifter can be realized based on the smaller shifting to save hardware resources. The critical path delay of each stage is two shifts and two additions.

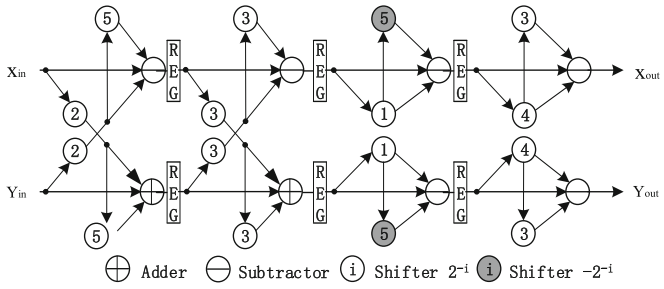

Fig. 4. Data flow of ARC_1

3.2 First Rotation ARC_2

As mentioned in Sect. 2, ARC_2 actually need to rotate the input vector by $-3\pi/16$. Hence, we combine ARC_2 with one adder and one subtractor to fulfill the rotation, and the scale factor of ARC_2 needs to multiply $1/\sqrt{8}$. Due to the scaling-free property of ARC, if ARC_2 only adopts ARC, it would require extra resources to execute the scale factor compensation. Hence, we combine the conventional CORDIC with ARC to implement $\pi/16$ rotation, and then the constant value $1/\sqrt{8}$ compensation can be combined with the scale factor of the conventional CORDIC.

As $\pi/16$ is expressed as 12'b000011001001, the conventional i = 3 iteration is first implemented, and then the i = 5, i = 8 and i = 10 iterations of ARC are executed. Meanwhile, we can also put the conventional i = 3 iteration and ARC i = 10 iteration into one iteration, and the rest are also rearranged into one iteration to get an area efficient architecture. The new iterations are expressed as:

$$\begin{bmatrix} 1 & -2^{-3}-2^{-9} \\ 2^{-3}+2^{-9} & 1 \end{bmatrix}, \begin{bmatrix} 1 & -2^{-4}-2^{-7} \\ 2^{-4}+2^{-7} & 1 \end{bmatrix} \tag{15}$$

where the components right shifting over 10 bits which equals machine zero have been eliminated. Compared to the separate iterations, the results of Eq. (15) have also been slightly amplified.

After using the scale factor k_2 of ARC_2 to replace the k_1 in Algorithm 1, the k_2 is represented as:

$$\begin{aligned} k_2 &= 1/\sqrt{8} \times \cos(\cos\tan(2^{-3})) \\ &= \frac{1}{\sqrt{8} \times \sqrt{1+2^{-6}}} \\ &\cong (1-2^{-1}-2^{-3}) \times (1-2^{-4}-2^{-9}) \end{aligned} \tag{16}$$

Using the same method as ARC_1, the approximation Eq. (16) is scaled into to compensate for the errors introduced in Eq. (15). The data flow of the ARC_2 is shown in Fig. 3, which is also divided into four stages to keep the critical path delay constant with ARC_1. The required iterations are realized in the first two stages, and the remaining stages are to execute the scale factor compensation. Meanwhile, the bigger shifters are also split into two subshifters to save area (Fig. 5).

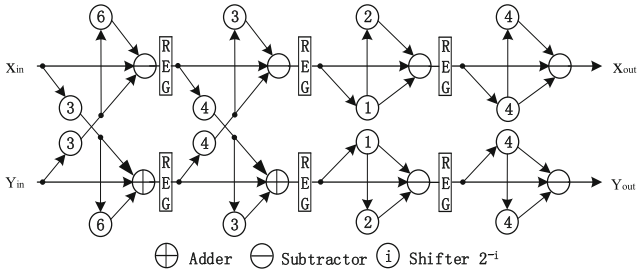

Fig. 5. Data flow of ARC_2

3.3 First Rotation ARC_3

The rotation angle of ARC_3 is the same as ARC_2, but the scale factor is different, which only needs to be scaled by $1/2$ through right shifting the results one bit.

Hence, we only adopt ARC $(i = 4, i = 5, i = 8, i = 11)$ to design ARC_3, and the i = 8 and i = 11 iterations combined with $1/2$ are put together into one iteration. The rotation sequencing of ARC_3 is

$$\begin{bmatrix} 1-2^{-7} & -2^{-3} \\ 2^{-3} & 1-2^{-7} \end{bmatrix}, \begin{bmatrix} 1-2^{-9} & -2^{-4} \\ 2^{-4} & 1-2^{-9} \end{bmatrix}, \begin{bmatrix} 2^{-1} & -2^{-8} \\ 2^{-8} & 2^{-1} \end{bmatrix} \tag{17}$$

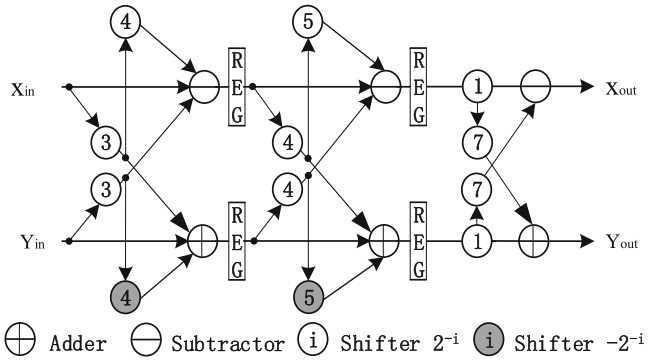

Fig. 6. Data flow of ARC_3

which is realized by three stages as illustrated in Fig. 6. As ARC_3 does not need scale factor compensation, all the stages are to execute the required iterations. We also split the bigger into two sub-shifters to reduce area.

3.4 Scale Factor Compensation for Adder_3

The scale factor in the Adder_3 is $1/\sqrt{8}$. After operated by Algorithm 1, it is approximated as:

$$K_{Adder\_3} = 1/\sqrt{8} \cong (1 - 2^{-1} - 2^{-5}) \times (1 - 2^{-2} + 2^{-8}) \tag{18}$$

where the achieved accuracy is above 1.6E-4. The corresponding date flow the scale factor compensation is shown in Fig. 7, which requires two clock cycles. The splitting scheme is also used to reduce the required resources.

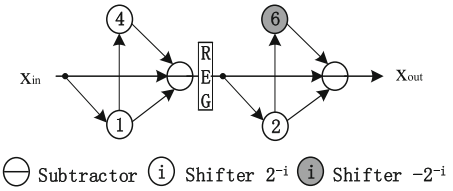

Fig. 7. Scale factor compensation for Adder_3

4 Performance Evaluation and Comparison

In this section, we compare our design with the state-of-the-art DCT discussed in [12] and the newest unified DCT and IDCT architecture presented in [10]. Meanwhile, as there are some different architectures proposed in [10, 12], we only focus on the

common architectures, which can process 8-point concurrently and do not save the hardware resources at the expense of PSNR.

4.1 Area Comparison

As the Lee architecture only works in DCT mode, we synthe-size both the Huang and the proposed architectures under DCT mode and DCT/IDCT mode, respectively. The required hardware resources of the three different architecture are all shown in Table 1. For DCT mode, compared with the Lee architecture, the proposed architecture uses 52 and 1 more Slice Registers and LUT-FF pairs respectively, but requires 52 less Slice LUTs. Compared with the Huang architecture, due to the proposed vector rotation schemes, no matter working in which mode, the proposed architecture uses less hardware resources. Taking Slice Registers under DCT/IDCT mode as an example, the required number of the proposed architecture is reduced by 37.6%.

Table 1. Area, speed and power comparisons.

Comparison	Lee. [12]	Huang. [10]		Proposed arch.	
Mode	DCT	DCT	DCT/IDCT	DCT	DCT/IDCT
Slice registers	992	1506	1714	1044	1070
Slice LUTs	1172	1526	1812		1502
LUT-FF pairs	1216	1660	2019	1120 1217	1517
CPD (ns)	3.16	3.05	3.2	3.08	3.23
Latency	6	13	113	6	6
PTime(ns)	18.96	39.65	41.6		19.38
T (M samples/s)	2532	2623	2500	18.48 2597	2477
Power(mW)	98	148	212	90	145

4.2 Speed Comparison

We compare the speed in terms of critical path delay, latency, processing time and throughput. As shown in Table 1, CPD means Critical Path Delay, PTime means Processing Time, T means Throughput. The latencies of the Lee, the Huang and the proposed architectures 6, 13 and 6, respectively. For DCT mode, the critical path delays of the three different architectures are 3.16 *ns*, 3.05 *ns* and 3.08 *ns*, respectively. After the pipelines set up, the throughput of the Lee, the Huang and the proposed architectures are 2532(*M samples/s*), 2623(*M samples/s*) and 2597(*M samples/s*), respectively, which means the throughput of the proposed architecture is slightly lower than the Huang architecture, but exceeds the Lee architecture. The processing time is defined as how long the pipeline needs to finish a complete computation, which equals the critical path delay times and latency. The processing time of the Lee and Huang are 18.96 ns and 39.65 ns, respectively, but the proposed architecture only requires 18.47 ns. For DCT/IDCT mode, compared to the Huang architecture, even the critical

path delay and throughput of the proposed architecture are slightly worse, the processing time provides a factor of 41.6/19.38 = 2.15-fold improvement because of its less latency.

4.3 Accuracy Comparison

We first investigate the accuracy of the rotation because they have an impact on the PSNR [16] of the full DCT and IDCT blocks.

We use bit error ratio (BER) to compare the accuracy of the proposed rotation elements with the Huang CORDIC [12] and the Lee CORDIC [10]. Taking the rotation angle $-3\pi/16$ as an example, a pseudorandom sequence of 1000 vectors lying evenly in the convergence range $[0, 2^8 - 1)$ is generated. We use these vectors as the inputs of the proposed ARC_2, Huang CORDIC and Lee CORDIC, the corresponding BERs of rotation of the three different architectures are shown in Fig. 8. The BERs of the proposed ARC_2 are the smallest, which are below $3E - 4$, while the BERs of the Huang CORDIC are the biggest and exceed $2E - 3$, and the BERs of the Lee CORDIC approximately equal $1E - 3$. The reason that ARC_2 has improved BERs is that it uses ARC combined with conventional CORDIC to implement the rotation elements and the proposed scale factor approximation scheme.

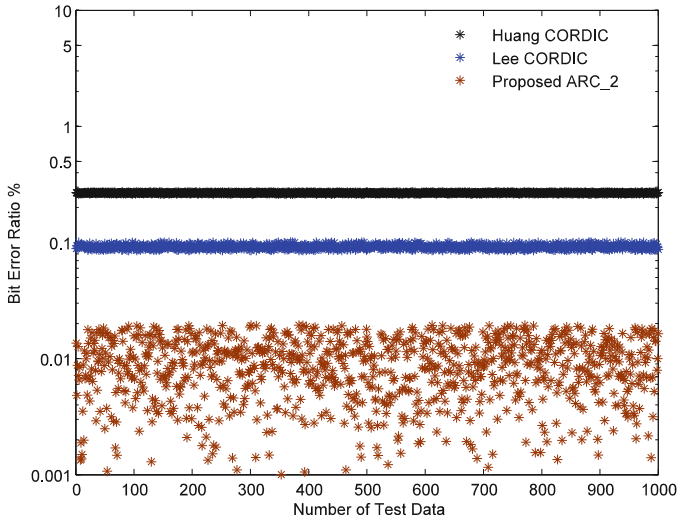

Fig. 8. BER comparison

Using PSNR to evaluate the performance of the three different DCT architectures, including the Lee architecture [10], the Huang architecture [12] and the proposed architecture, we generate 500 8×8 test matrices to obtain the average PSNR. The PSNRs of the Lee, the Huang and the proposed architectures are 43.16 dB, 45.98 dB and 46.37 dB, respectively, which means that the PSNR of the proposed architecture

exceeds the Lee architecture is improved by 3.21 dB. The reason for the improved PSNR is that the BERs of rotation elements of the proposed architecture are lower than the rotation elements in the Lee architecture. Compared with the Huang architecture, the PSNR of the proposed architecture is also higher. The reason is not only the proposed novel architecture, which shortens the required number of iterations to reduce the errors, but also the rotation elements based on the efficient ARC combined with conventional CORDIC, which have higher accuracy.

4.4 Power Comparison

We use the Xpower tool [15] to estimate the power dissipation of the Lee, the Huang and the proposed architectures. The method used is to build a harness that can connect to all of the inputs and outputs of the circuit. Xpower is used to estimate the power of the device with and without the measured circuit in the harness. The difference between the two versions is the power dissipation of the circuit of interest. This approach is described in more detail in [18]. Since the power estimation is a function of the toggle rate of the signals, we set the toggle rate of the signals to 12.5% to bracket the estimated power dissipation.

As illustrated in Table 1, for DCT mode, the three different architectures dissipate 98 mW, 148 mW and 30 mW, respectively, which means the proposed architecture dissipates 8.2% and 39.2% less power than the Lee and the Huang architectures, respectively. For DCT/IDCT mode, the Huang and the proposed architectures consume 212 mW and 145 mW, respectively, which means the dissipated power of the proposed architecture is reduced by 31.6%.

5 Conclusion

This paper has presented an Improved FPGA implementation of DCT/IDCT architecture based on ARC. An efficient architecture for DCT/IDCT is first described. For the different vector rotations, we adopt ARC and conventional CORDIC combined with different scale factors to simplify the architecture and improve the accuracy. The result demonstrate that the proposed architecture has the best PSNR and dissipate the least power. Compared with the Lee architecture, both the throughput and the processing time of the proposed architecture are also slightly better. Compared with the Huang architecture, all of the required hardware resources, the latency and the processing time are highly reduced with a little loss in critical path delay and throughput.

Acknowledgments. This work is supported by Xilinx. We also would like to thank Jianfeng Zhang and the reviewer for their revisions and suggestions.

References

1. Rao, K.R., Yip, P.: Discrete Cosine Transform: Algorithms, Advantages. Applications. Academic press, New York (2014)
2. Clarke, R.: Relation between the Karhunen Loèvet and cosine transforms. Commun. Rada Signal Process. IEE Proc. F **128**(6), 359–360 (1981)
3. Wallace, G.K.: The JPEG still picture compression standard. IEEE Trans. Consum. Electron. **38**(1), xviii–xxxiv (1992)
4. Le Gall, D.: MPEG: a video compression standard for multimedia applications. Commun. ACM **34**(4), 46–58 (1991)
5. Ahmed, A., Shahid, M.U.: N point DCT VLSI architecture for emerging HEVC standard. VLSI Des. **2012**, 6 (2012)
6. Li, C.-Y., et al.: A probabilistic estimation bias circuit for fixed-width booth multiplier and its DCT applications. IEEE Trans. Circuits Syst. II Express Briefs **58**(4), 215–219 (2011)
7. Yu, S., Swartziander, E.: DCT implementation with distributed arithmetic. IEEE Trans. Comput. **50**(9), 985–991 (2001)
8. Xiao, L., Huang, H.: A novel CORDIC based unified architecture for DCT and IDCT. In: 2012 International Conference on Optoelectronics and Microelectronics (ICOM). IEEE (2012)
9. Huang, H., Xiao, L.: CORDIC based fast radix-2 DCT algorithm. IEEE Sig. Process. Lett. **20**(5), 483–486 (2013)
10. Huang, H., Xiao, L.: CORDIC based fast algorithm for power-of-two point DCT and its efficient VLSI implementation. Microelectron. J. **45**(11), 1480–1488 (2014)
11. Huang, H., Xiao, L., Liu, J.: CORDIC-Based Unified Architectures for Computation of DCT/IDCT/DST/IDST. Circ. Syst. Sig. Process. **33**(3), 799–814 (2014)
12. Lee, M.W., Yoon, J.H., Park, J.: Reconfigurable CORDIC-based low-power DCT architecture based on data priority. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **22** (5), 1060–1068 (2014)
13. Zhang, J., et al.: Adaptive recoding CORDIC. IEICE Electron. Express **9**(8), 765–771 (2012)
14. Meher, P.K., et al.: 50 years of CORDIC: algorithms, architectures, and applications. IEEE Trans. Circ. Syst. I Regul. Pap. **56**(9), 1893–1907 (2009)
15. Xilinx, X.E.U.G., Xilinx power tools tutorial (2010). 2012
16. Zhang, J., Chow, P., Liu, H.: An efficient FPGA implementation of QR decomposition using a novel systolic array architecture based on enhanced vectoring CORDIC. In: 2014 International Conference on Field-Programmable Technology (FPT). IEEE (2014)