Designing a Custom Architecture for DCT Using NISC Technology
Bita Gorjiara, Mehrdad Reshadi, Daniel Gajski
Center for Embedded System Computers, University of California Irvine
{bgorjiar, reshadi, gajski}@cecs.uc.edu

Abstract—This paper presents design of a custom architecture for Discrete Cosine Transform (DCT) using No-Instruction-Set Computer (NISC) technology that is developed for fast processor customization. Using several software transformations and hardware customization, we achieved up to 10 times performance improvement, 2 times power reduction, 12.8 times energy reduction, and 3 times area reduction compared to an already-optimized soft-core MIPS implementation.

I. Introduction
This paper presents design of a custom architecture for Discrete Cosine Transform (DCT) using No-Instruction-Set Computer (NISC) design flow that is developed for fast processor customization. Processor customization techniques such as designing Application-Specific Instruction-Set Processors (ASIPs) [2] have recently emerged to meet the performance and power constraints of designs starting from high-level languages such as C. A new alternative to ASIP is No-Instruction-Set-Computer (NISC) technology that is developed for fast processor customization. Using several software transformations and Computer (NISC) technology that is developed for fast processor customization.

II. DCT algorithm
The Discrete Cosine Transform (DCT) [1] and Inverse Discrete Cosine Transform (IDCT) are important parts of JPEG and MPEG standards. MPEG encoders use both DCT and IDCT, whereas MPEG decoders only use IDCT. The definition of DCT for a 2-D N×N matrix of pixels is as follows:

\[ F[u,v] = \frac{1}{N^2} \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} f[m,n] \cos \left( \frac{2m+1}{2N} \pi \right) \cos \left( \frac{2n+1}{2N} \pi \right) \]

Where \( u, v \) are discrete frequency variables (0 ≤ u, v < N), \( f[i,j] \) gray level of pixel at position \( (i,j) \), and \( F[u,v] \) coefficients of point \( (u,v) \) in spatial frequency. Assuming \( N=8 \), matrix \( C \) is defined as follows:

\[ C[u][v] = \frac{1}{8} \cos \left( \frac{2n+1}{2} \pi \right) \]

Based on matrix \( C \), an integer matrix \( C1 \) is defined as follows: \( C1 = \text{round} \times C \). The \( C1 \) matrix is used in calculation of DCT and IDCT: \( F = C1 \times F \times C2 \), where \( C2 = C1 \times \). As a result, DCT can be calculated using two consecutive matrix multiplications. Figure 1(a) shows the C code of multiplying two given matrix \( A \) and \( B \) using three

III. Custom DCT implementations
In general, customization of a design involves both software and hardware transformations. To increase the parallelism in code, we unroll the inner-most loop of the matrix multiplication code, merge the two outer loops, and convert some of the costly operations such as addition and multiplication to OR AND. In DCT, the operation conversions are possible because of the special values of the constants and variables. The transformed code is shown in Figure 1(b). By looking at the body of loop, four steps of computation can be identified: (1) calculation of the memory addresses of the matrix elements; (2) loading the values from data memory; (3) multiplying the two values; (4) accumulating the multiplication results. We design our custom datapath in a way that each of these steps is a pipeline stage. Figure 2(a) shows the proposed custom pipelined datapath called CDCT1. The datapath includes four major pipeline stages that are marked in the figure. In NISC, Comparator (Comp) and Address Generator (AG) are used for handling jumps, while Link Register (LR) and direct address are used for supporting function calls. We have used operation chaining to reduce RF file accesses and decrease register pressure. The OR and ALU, as well as the Mul and Adder are chained. Note that the chaining of multiply and add forms a MAC unit in the datapath. After compilation, synthesis and PAR, the total number of cycles of the DCT is reduced to 3080, and the maximum clock frequency is 85.7MHz. Next, we iteratively apply the following datapath refinements to improve the performance, power, and area of DCT implementation:

1) To reduce critical path delay that includes ALU delay and RF setup time, we add an extra register between the output of ALU and the input of RF. Also, LR and direct address are removed because, there is no need for a function call (the matrix multiplication code is inlined).
Additionally, buses are simplified to point-to-point connections that are actually used by DCT. The result architecture is called CDCT2.

2) The unused parts of ALU, Comp and RF are removed. A general-purpose ALU and Comp supports many operations. However, as shown in Figure 1, only Add, Or, And, Multiply, and Not-equal (\(=\)) operations are used in DCT. Therefore, the ALU and Comp can be simplified. Additionally, instead of 32-register RF, we can use a 16-register RF because the rest is not used by the application. We call this new architecture CDCT3.

3) After synthesizing CDCT3, we observe that the critical path includes the Control Memory (CMem) read delay, CW wire delay, and AG's delay. Therefore, to reduce critical path delay, we apply controller pipelining by adding CW register and status register, and call the new architectures CDCT4 and CDCT5, respectively.

4) To decrease the area, we reduce the bit-width of the components in address calculation pipeline stage without affecting the precision of DCT calculations. The result architecture is called CDCT6.

5) After synthesizing CDCT6, we observe that the critical path goes through the Mul. Since Mul is a ASIC multiplier, we cannot reduce the critical path any further. However, if we consider Mul as a two-cycle unit, we can further improve the clock frequency by adding pipeline registers at the outputs of the RF. The final architecture (CDCT7) is shown in Figure 2(b).

IV. Comparing the DCT implementations

Table 1 compares the performance, power, energy, and area of the all NISC implementations after synthesizing them on FPGA. The fourth column of Table 1 shows the total execution time of the DCT algorithm. Note that although in some cases (CDCT4, CDCT5, and CDCT7) the number of cycles increases, the clock frequency improvement compensates for that. Therefore, the total execution delay maintains a decreasing trend.

\[
\text{Table 1. Performance, dynamic power, energy, and area of the NISCs}
\]

<table>
<thead>
<tr>
<th>No. of cycles</th>
<th>Clock Freq</th>
<th>DCT exec. time (us)</th>
<th>Power (mW)</th>
<th>Energy (uJ)</th>
<th>Normalized area</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMIPS</td>
<td>3072</td>
<td>583</td>
<td>157.57</td>
<td>127.33</td>
<td>24.40</td>
</tr>
<tr>
<td>CDCT1</td>
<td>3008</td>
<td>55.7</td>
<td>155.94</td>
<td>120.52</td>
<td>4.33</td>
</tr>
<tr>
<td>CDCT2</td>
<td>2952</td>
<td>90.7</td>
<td>155.50</td>
<td>115.57</td>
<td>6.38</td>
</tr>
<tr>
<td>CDCT3</td>
<td>2952</td>
<td>114.4</td>
<td>25.80</td>
<td>82.82</td>
<td>2.14</td>
</tr>
<tr>
<td>CDCT4</td>
<td>3008</td>
<td>147.0</td>
<td>20.95</td>
<td>125.00</td>
<td>2.62</td>
</tr>
<tr>
<td>CDCT5</td>
<td>3206</td>
<td>199.5</td>
<td>18.93</td>
<td>190.00</td>
<td>2.01</td>
</tr>
<tr>
<td>CDCT6</td>
<td>3208</td>
<td>171.5</td>
<td>18.71</td>
<td>104.00</td>
<td>1.95</td>
</tr>
<tr>
<td>CDCT7</td>
<td>3460</td>
<td>250.0</td>
<td>13.84</td>
<td>137.00</td>
<td>1.90</td>
</tr>
</tbody>
</table>

Dynamic power consumption (column fifth), also decreases as we introduce customization and datapath pipelining. However, in CDCT4, power consumption increases because of extra logic added by retiming algorithm. In general, as frequency increases the clock power of the datapaths increases. The power-breakdown of the designs (Figure 3) confirms this fact.

![Figure 2. Block diagram of (a) CDCT1, (b) CDCT7](image)

We configured the Xilinx Virtex-II Multimedia development board to run CDCT7. The board has a Virtex-II XC2V2000-FF896 FPGA package and only supports 27MHz, 53MHz, and 108MHz clock frequencies. Although CDCT7 could achieve the maximum clock frequency of 250MHz, we ran it with clock frequency of 108MHz on the board due to unavailability of higher clock frequencies.

![Figure 3. Dynamic power of the DCT implementations](image)

![Figure 4. Comparing different DCT implementations](image)

![Figure 5. Xilinx Virtex-II multimedia board](image)

![Figure 6. CDCT7 after placement and routing](image)

For all the DCT implementations, the synthesizable Verilog files, timing constraints, the synthesis scripts, and the Placement-and-Routing results are available for download at [6].

References