Designing a Custom Architecture for DCT Using NISC Design Flow

Bita Gorjiara, Mehrdad Reshadi, Daniel Gajski
Center for Embedded System Computers, University of California Irvine
{bgorjiar, reshadi, gajski}@cecs.uci.edu

Abstract

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. 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.

1. 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) [3] in which a *cycle-accurate compiler* generates code to control a given custom datapath at every clock cycle. However, instead of using any abstraction such as instruction-set or microcode, the NISC compiler directly generates the control signal values of every component in the datapath for every clock cycle. A NISC designer needs to only focus on designing the datapath, i.e. selecting the components and connecting them together. Unlike ASIPS, in NISC, there is no need for designing instruction-set and instruction decoder, or updating the compiler. The NISC compiler inputs the datapath as a netlist of RTL components, and automatically analyzes and extracts branch delay and possible operations. The datapath netlist contains components such as bus, multiplexer, register, register-file, memory, and functional unit. After compiling the program onto the given datapath, the compiler generates a string of control values, called Control Word (CW), for each cycle. These control words are stored in a control memory and are applied to the datapath by the controller at every cycle.

In this case-study, first we compile the C code of DCT algorithm on a general-purpose datapath similar to a MIPS processor. Next, we apply several software transformations and hardware customization to improve the performance, power, energy and area.

2. 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 $N \times N$ matrix of pixels is as follows:

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

Where u, v are discrete frequency variables ($0 \le u, v \le N-1$), $f[i, j]$ 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:

$$C[u][n] = \frac{1}{8}\cos\frac{(2n+1)u\pi}{16}$$

Based on matrix C, an integer matrix $C1$ is defined as follows: $C1$ = round(*factor* × C). The $C1$ matrix is used in calculation of DCT and IDCT: $F = C1 \times f \times C2$, where, $C2 = C1^T$. 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 nested loops. Using a MIPS M4K ™ Core processor [4], the matrix-multiplication-based DCT takes 13058 cycles to compute [3]. However, given the MIPS datapath, the NISC-style processor takes 10772 cycles to compute DCT. The 20% reduction in number of cycles is because of the finer-grained control that NISC compiler has over the datapath compared to traditional compilers that use instruction-set abstraction. We developed the synthesizable hardware description for our NISC-style MIPS (called NMIPS), and synthesized it using Xilinx ISE 6.3. In our implementation, the bus-width of the datapath is 16-bit for a 16-bit DCT precision, and the datapath does not have any integer divider or floating point unit. The clock frequency of 78.3MHz was achieved after synthesis and Placement-and-Routing (PAR). All of the experiments in this paper are synthesized and mapped on Xilinx FPGA package Virtex2V250-6 using Xilinx ISE 6.3 tool. Two synthesis optimizations of retiming and buffer-to-multiplexer conversions are applied to improve the performance. In these experiments, we set the PAR effort to the highest level possible for maximum clock speed.

```
for(int i=0; i<8; i++)
  for(int j=0; j<8; j++){
    sum=0;
    for(int k=0; k<8; k++)
      sum= sum+A[i][k]×B[k][j];
    C[i][j]= sum;
  }
```
(a)

```
ij=0;
do {
  i8 = ij & 0xF8;
  j = ij & 0x7;
  aL= *(A+(i8|0) ); bL= *(B + (0|j) );  sum =  aL × bL;
  aL= *(A+(i8|1) ); bL= *(B + (8|j) );  sum+= aL × bL;
  aL= *(A+(i8|2) ); bL= *(B + (16|j) ); sum+= aL × bL;
  aL= *(A+(i8|3) ); bL= *(B + (24|j) ); sum+= aL × bL;
  aL= *(A+(i8|4) ); bL= *(B + (32|j) ); sum+= aL × bL;
  aL= *(A+(i8|5) ); bL= *(B + (40|j) ); sum+= aL × bL;
  aL= *(A+(i8|6) ); bL= *(B + (48|j) ); sum+= aL × bL;
  aL= *(A+(i8|7) ); bL= *(B + (56|j) );
  *(C + ij) = sum + (aL × bL);
} while(++ij!=64);
```
(b)

Figure 1. (a) Original and (b) Transformed matrix multiplication

3. 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 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).

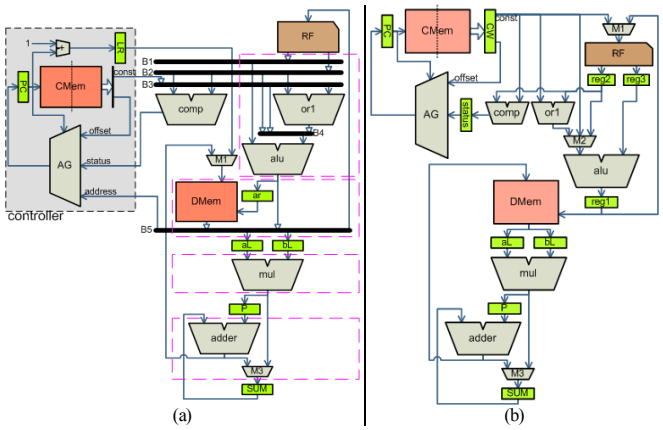

Figure 2. Block diagram of (a) CDCT1, (b) CDCT7

4. 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.

Table 1. Performance, power, energy, and area of the NISCs

	No. of cycles	Clock Freq	DCT exec. time(us)	Power (mW)	Enegy (uJ)	Normalized area
NMIPS	10772	78.3	137.57	177.33	24.40	1.00
CDCT1	3080	85.7	35.94	120.52	4.33	0.81
CDCT2	2952	90.0	32.80	111.27	3.65	0.71
CDCT3	2952	114.4	25.80	82.82	2.14	0.40
CDCT4	3080	147.0	20.95	125.00	2.62	0.46
CDCT5	3208	169.5	18.93	106.00	2.01	0.43
CDCT6	3208	171.5	18.71	104.00	1.95	0.34
CDCT7	3460	250.0	13.84	137.00	1.90	0.35

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.

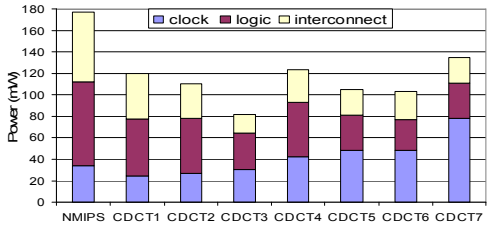

Figure 3. Power breakdown of the DCT implementations

Figure 4 shows the performance, power, energy and area of the designs normalized against NMIPS. As shown in Figure 4, CDCT7 is the best design in terms of delay and energy consumption, while CDCT3 is the best in terms of power, and CDCT6 is the best in terms of area. As a result, CDCT3, CDCT6, and CDCT7 are considered the pareto-optimial solutions. Note that minimum energy and minimum power are achieved by two different designs: CDCT7 and CDCT3, respectively. Compared to NMIPS, CDCT7 runs 10 times faster, consumes 1.3 times less power and 12.8 times less energy. Also, it occupies 2.9 times less area than NMIPS. The minimum power consumption is achieved by the CDCT3, which consumes 2 times less power compared to NMIPS. Note that performance of NMIPS is 20% better than performance of a MIPS core. Also, since NMIPS does not have instruction decoder, its area and power are less than MIPS. In our experiments, we compared the results to NMIPS which is conservative relative to the MIPS core.

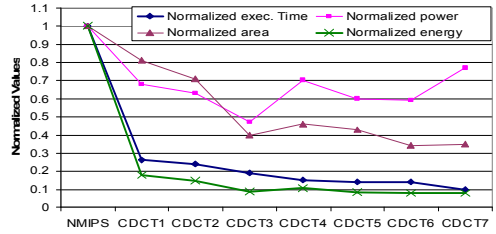

Figure 4. Comparing different DCT implementations

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 5. Xilinx Virtex-II multimedia board

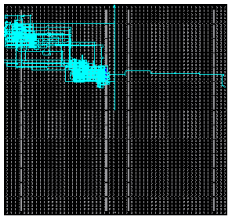

Figure 6. CDCT7 after placement and routing

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 [5].

References

[1] N. Ahmed, T. Natarajan, and K.R. Rao, Discrete Cosine Transform, *IEEE Trans. On Computers*, vol. C- 23, pp. 90-93, Jan 1974

[2] M.K. Jain, M. Balakrishnan, and A. Kumar, ASIP Design Methodologies: Survey and Issues, *In Proc. of International Conference on VLSI Design*, 2001.

[3] M. Reshadi, D. Gajski, A Cycle-Accurate Compilation Algorithm for Custom Pipelined Datapaths, *In Proc. ISSS05*, 2005.

[4] MIPS32® M4K™ Core, http://www.mips.com

[5] http://newport.eecs.uci.edu/~bgorjiar/projects/NISC/customDCT/