A VLSI Implementation of a JPEG Decoder

Smitha Shyam, Deepesh John,
Tejasvi Kachru, Sujay Phadke

Department of Electrical Engineering and Computer Science,
University of Michigan, Ann Arbor
EECS – 427 Fall 2004

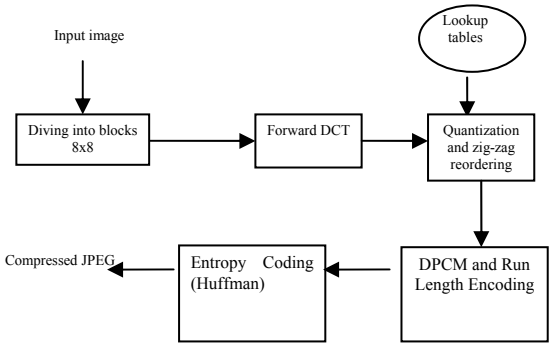
ABSTRACT

In this paper, we describe the VLSI implementation of a JPEG decoder. The implementation used custom designed hardware as well as synthesized components. We have implemented a Huffman decoder, a run length decoder, a booth multiplier alongwith a custom datapath consisting of a register file, shifter and ALU.

Keywords: JPEG, Huffman decoder, Run Length Decoder, IDCT, Booth Multiplier

1. INTRODUCTION

The JPEG image compression standard uses transform based coding to compress images without loss in resolution. It consists of the following modules:



After dividing the input image into 8x8 blocks, a forward Discrete Cosine Transform (FDCT) is performed on each block according to the expression:

$$\text{FDCT}\quad S_{u,v}(u,v)=\frac{1}{4}C_uC_v\sum_{x=0}^{7}\sum_{y=0}^{7}s_{x,y}\cos\left(\frac{(2x+1)\cdot u\pi}{16}\right)\cos\left(\frac{(2y+1)\cdot v\pi}{16}\right)$$

$$\text{IDCT}\quad s_{x,y}(x,y)=\frac{1}{4}\sum_{u=0}^{7}\sum_{v=0}^{7}C_uC_vS_{u,v}\cos\left(\frac{(2x+1)\cdot u\pi}{16}\right)\cos\left(\frac{(2y+1)\cdot v\pi}{16}\right)$$

$$C_n=\begin{cases}1/2 & n=0\\ 1 & n\neq 0\end{cases}$$

This generates coefficients corresponding to the visual frequency content in the image. Since these coefficients could vary over a large range, a large number of bits would be required to store every one of them in binary form. Hence a quantization is performed to reduce the amplitude of these bits. If this is done using floating point numbers then the compression is lossless. The output of the quantizer consists of 64 scaled coefficients. The first of these is the DC coefficient (analogous to the average DC value of the image bits) and the remaining 63 coefficients are AC coefficients representing the frequency domain variations. It is observed that after quantization, a large number of 0s are obtained in the AC coefficients, and they are arranged in a zig-zag pattern starting from the top-left corner. Hence if re-ordering is done, a string is produced with large number of 0s bounded by small number of 1s. This stream can be efficiently compressed using Run Length Encoding (RLE) which replaces it with a (run,size) pair representing the amplitude of a non-zero coefficient *(size)* preceded by *run* number of zeroes. A further compression is obtained by using Huffman encoding which encodes these (run,size) pairs as bits based on their probabilistic occurrence in the transformed image. The DC coefficient is encoded using Differential Pulse Code Modulation (DPCM) and hence stored as the difference over the previous value. A typical Huffman coding scheme is as shown in the following tables:

Huffman codes for DC coefficient:

DC coefficient	Size	Typical code	Additional bits
0	0	00	-
-1,1	1	010	0,1
-3,-**2,2**,3	2	011	0,1
-7,-6,-**5**,-4,…,**5**,6,7	3	100	0,1
-15, -**11**, …0…**11**,…15	4	101	0,1

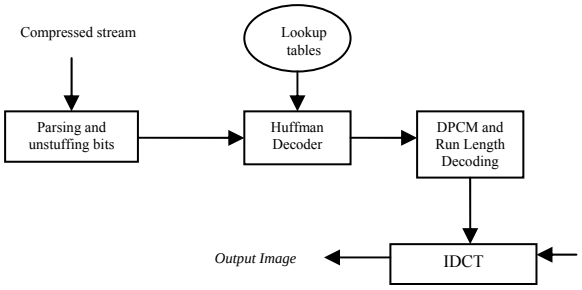
Huffman code for AC coefficients:

(run, size)	Code byte (Hex)	Code word
0,1	01	00
0,2	02	01
0,3	03	100
EOB	FF	1010

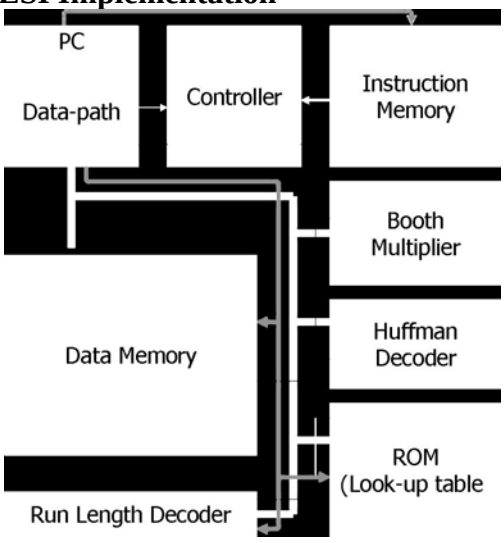
0,4	04	1011
1,1	11	1100

The entire Huffman coding is upto 32 bits and would require a large amount of memory and hardware. Hence we decided to implement a reduced Huffman decoder which uses the tables given above. The output of the encoder are these encoded bits.

A JPEG decoder implements the opposite processes. It takes the input stream and after pre-processing, implements a Huffman decoding to extract the (run,size) information. The Huffman tables are part of the JPEG header. Next a DPCM is performed on the DC coefficient and an Run Length Decoding is performed on the AC coefficients which extract a stream of frequency coefficients of the original image. In order to return to the spatial domain, a Inverse DCT is performed which reconstructs the original image and scales it up to the original amplitudes. The block diagram below illustrates this process flow.

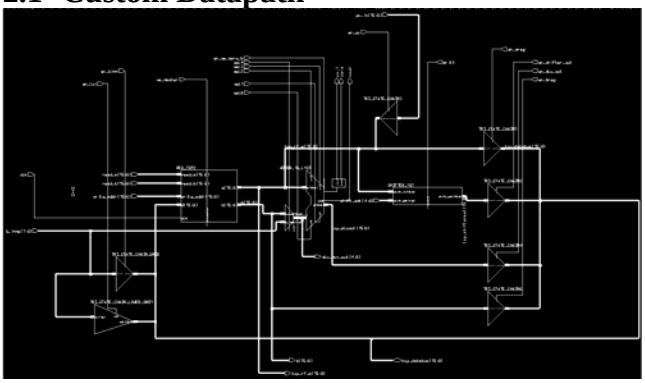


2. VLSI Implementation



The JPEG decoder is implanted in part as a custom design and synthesized modules. The Huffman decoder and Run Length Decoder modules are synthesized from Verilog. A Booth multiplier has been designed and synthesized for performing multiplications operations for the IDCT. The custom designed datapath consists of a dual-port Register File, a Logarithmic Shifter and the ALU. These form part of the core ISA architecture. The controller is synthesized and it acts as the master control for all the different modules.

2.1 Custom Datapath

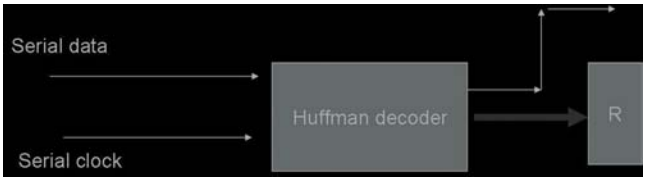


The datapath is custom designed in 0.25µ technology using Mentor Graphics CAD tools. The process flow is : Schematic → Functional Simulation → Sizing critical paths and simulation in SPICE → custom layout → DRC → LVS → PEX → re-simulation in SPICE for delays → functional simulation with delays and verification. The databus is 16 bit, address bus is 8 bit. The adder is 16-bit carry-bypass adder.

2.2 Booth Multiplier

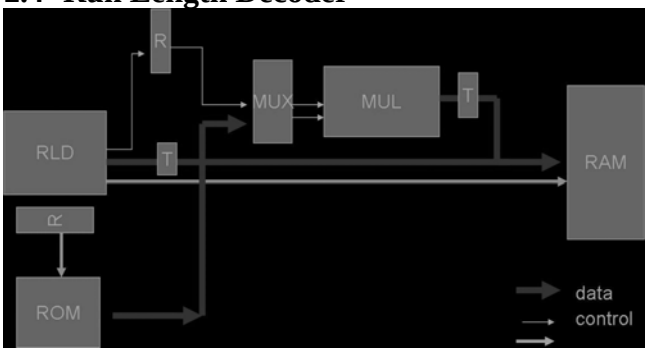
In our implementation, the IDCT is performed using the datapath ALU and a Booth Multiplier which is synthesized. The Booth multiplier has been designed using modified Booth encoding. It is a 8 bit multiplier. 3 bits of the multiplier are scanned together to generate control signals viz. NEG, ZERO, and SHIFT which act on the multiplicand to negate, zero or shift the multiplicand. These bits are fed to a module which takes the multiplicand as the input and based on the values of the 3 control bits, generates the partial product. For a 8-bit multiplication, we would have 4 partial products. The output of these 4 partial product generators is sent to the final ripple carry adder which generates the final 16 bit product. The multiplication is implemented as a single cycle instruction.

2.3 Huffman Decoder



The Huffman decoder decodes the input stream based on the tables shown before. It is implemented in the form of a FSM and takes in 1 bit at a time. It has a separate clock which runs at a higher frequency than the master clock. The Huffman decoder is activated by 2 separate instructions – HuffDC and HuffAC which generates either the amplitude of the DC coefficient or the (run,size) pair. The output of Huffman decoder is fed to a register which is read by the Run Length Decoder (RLD). Every instruction decodes one coefficient. The input to the Huffman decoder is through a serial port, where data is fed externally.

2.4 Run Length Decoder

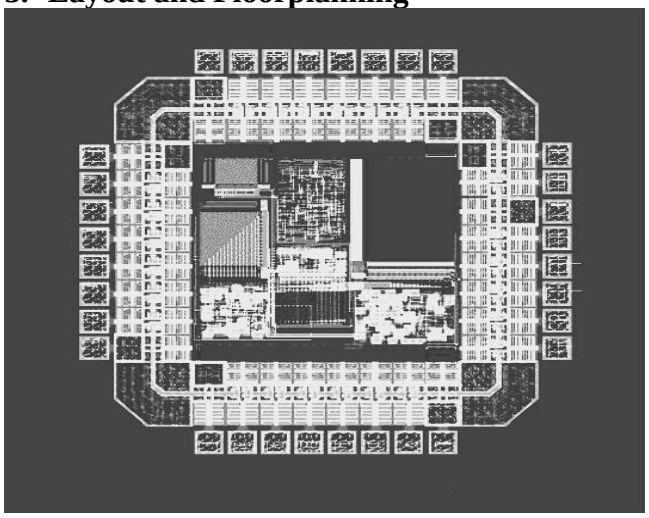


The Run Length Decoder takes input from the Huffman decoder and produces a decoded stream of 1s and 0s from the (run,size) input where size is the magnitude of the AC coefficient and run is the number of preceding 0s. Thus, a (0,3) is decoded to a 00000011 and a (1,1) is decoded to a 00000000 00000001. These values are generated as part of 2 special instructions MRLD_DC and MRLD_AC. Since the IDCT consists of multiplications and additions, the output of the RLD is sent to the multiplier which takes the constant scaling factors from the ROM for multiplication.

2.5 Controller

The control logic is written as a behavioral description in Verilog and synthesized.

3. Layout and Floorplanning



The datapath was designed as a custom layout. The remaining modules have been synthesized.

4. Results

Area	1.2 mm x 1.2 mm
Clock period	18ns
Serial clock period	6ns
I/O pads	16/32 used

5. ACKNOWLEDGMENTS

We express our heartfelt thanks to Prof. Marios Papaefthymiou, Luke Frankfart and Mark Who for their support throughtout this project.

6. REFERENCES

[1] **"JPEG Decoder Design,** Team # 11", John P. Jones, Frank Vahid, Barry S. Todd

[2] **"A JPEG Decoder IP Module Designed Using VHDL Module Generator",** ***Koay Kah Hoe***

[3] http://bw-www.ie.u-ryukyu.ac.jp/~wada/design03/spec_e.html