A VHDL design of a JPEG still image compression standard decoder

Douglas Carpenter

Follow this and additional works at: http://scholarworks.rit.edu/theses

Recommended Citation
A VHDL Design of a JPEG Still Image Compression Standard Decoder

by

Douglas A. Carpenter

A Thesis Submitted
in
Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Computer Engineering

Approved by: _______________________________
Graduate Advisor - Prof. George A. Brown

______________________________
Committee Member - Dr. Ronald G. Matteson

______________________________
Department Head - Dr. Roy S. Czemikowski

DEPARTMENT OF COMPUTER ENGINEERING
COLLEGE OF ENGINEERING
ROCHESTER INSTITUTE OF TECHNOLOGY
ROCHESTER, NEW YORK

August, 1994
Title: A VHDL Design of a JPEG Still Image Compression Standard Decoder

I, Douglas A. Carpenter, hereby grant permission to the Wallace Memorial Library of RIT to reproduce my thesis in whole or in part.

Signature: _________________________________
Date: 15 sep 94
ABSTRACT

Digital images require large amounts of memory to be stored in a computer system. The JPEG compression standard allows the amount of memory storage required by a digital image to be reduced with little to no perceptible loss of image quality. This thesis is a design of an ASIC that implements a decoder of JPEG compressed images. The decoder implements the baseline decoder defined by the JPEG standard with a few exceptions, the most notable being that only grayscale images can be decompressed. With such an ASIC, the speed of decompressing images is greatly increased. The decoder was designed by writing VHDL source code, which in turn was used to synthesize the ASIC using standard cells.
## TABLE OF CONTENTS

ABSTRACT ........................................................................................................................................ iii

LIST OF FIGURES ......................................................................................................................... vi

LIST OF TABLES ........................................................................................................................... x

GLOSSARY OF TERMS ..................................................................................................................... xii

1.0 INTRODUCTION ...................................................................................................................... 1-1
   1.1 OVERVIEW ............................................................................................................................. 1-1
   1.2 IMAGES ................................................................................................................................. 1-1
   1.3 DIGITAL IMAGING ................................................................................................................ 1-2
   1.4 JPEG .................................................................................................................................... 1-3
   1.5 TYPES OF JPEG IMPLEMENTATIONS ................................................................................. 1-6
   1.6 VHDL .................................................................................................................................... 1-8
   1.7 DESIGN BRIEF ..................................................................................................................... 1-10
   1.8 SOFTWARE USED ............................................................................................................... 1-11

2.0 CONCEPTS ................................................................................................................................. 2-1
   2.1 JPEG .................................................................................................................................... 2-1
      2.1.1 JPEG BRIEF .................................................................................................................. 2-1
      2.1.2 HUFFMAN CODING .................................................................................................... 2-7
      2.1.3 DISCRETE COSINE TRANSFORM ................................................................................. 2-12
   2.2 A JPEG ENCODER AND DECODER ................................................................................... 2-13
   2.3 INTERCHANGE FORMAT ...................................................................................................... 2-20
   2.4 VHDL .................................................................................................................................... 2-21
      2.4.1 BEHAVIORAL MODELING ......................................................................................... 2-21
      2.4.2 STRUCTURAL MODELING ........................................................................................... 2-23

3.0 IMPLEMENTATION ..................................................................................................................... 3-1
   3.1 C PROGRAM BEHAVIORAL MODEL ................................................................................... 3-1
   3.2 LIMITATIONS OF STRUCTURAL MODEL .......................................................................... 3-1
   3.3 VHDL STRUCTURAL MODEL ............................................................................................... 3-2
      3.3.1 VHDL TEST BENCH .................................................................................................... 3-2
      3.3.2 CONTROL SIGNALS ................................................................................................... 3-5
      3.3.3 DECODER MODULE ..................................................................................................... 3-7
      3.3.4 DECODING A SCAN .................................................................................................... 3-35
      3.3.5 MISCELLANEOUS MODULES ..................................................................................... 3-43
LIST OF FIGURES

Figure 1-1  Example Compressed Images ............................................................ 1-5
Figure 1-2  Overview of Design Test Flow ......................................................... 1-10
Figure 2-1  Encoder Overview ................................................................. 2-2
Figure 2-2  Decoder Overview ................................................................. 2-3
Figure 2-3  Interchange Format ................................................................. 2-4
Figure 2-4  JPEG hierarchy ................................................................. 2-5
Figure 2-5  DPCM Model for Encoding ........................................................... 2-8
Figure 2-6  Example Quantization ................................................................. 2-15
Figure 2-7  Zig Zag Ordering ................................................................. 2-17
Figure 2-8  Encoding Flow ................................................................. 2-18
Figure 2-9  Decoding Flow ................................................................. 2-19
Figure 2-10  Typical Compressed Image Data Format ........................................ 2-20
Figure 2-11  A Decoder Circuit ................................................................. 2-22
Figure 2-12  Example VHDL Circuit ............................................................. 2-22
Figure 2-13  Example VHDL Behavioral Model ............................................... 2-23
Figure 2-14  Example VHDL Structural Model ............................................... 2-24
Figure 3-1  VHDL Test Bench ................................................................. 3-2
Figure 3-2  VHDL Test Bench Basic Structure ............................................... 3-4
Figure 3-3  Control Signals Example Schematic ............................................. 3-5
Figure 3-4  Example Control Signals Timing .................................................. 3-7
| Figure 3-5 | Major Components of Decoder Module | 3-8 |
| Figure 3-6 | Layout of Decoder VHDL Source Code | 3-9 |
| Figure 3-7 | Controller Symbol | 3-10 |
| Figure 3-8 | NextByte Symbol | 3-15 |
| Figure 3-9 | NextByte Espresso Input | 3-17 |
| Figure 3-10 | NextByte Espresso Output File | 3-18 |
| Figure 3-11 | VHDL Source Code From Espresso Output | 3-19 |
| Figure 3-12 | ISR Symbol | 3-19 |
| Figure 3-13 | FindSOI Symbol | 3-21 |
| Figure 3-14 | Example Wired Or Schematic | 3-22 |
| Figure 3-15 | FindSOF Symbol | 3-23 |
| Figure 3-16 | Frame Header Syntax | 3-24 |
| Figure 3-17 | Fr_Header Symbol | 3-25 |
| Figure 3-18 | FindEOI Symbol | 3-27 |
| Figure 3-19 | dec_frame symbol | 3-28 |
| Figure 3-20 | Find_SOS symbol | 3-29 |
| Figure 3-21 | Decode_Scan Symbol | 3-30 |
| Figure 3-22 | Differing Uses of else Clause | 3-32 |
| Figure 3-23 | Example VHDL Direct Coded State Machine | 3-33 |
| Figure 3-24 | dec_scan_header Symbol | 3-34 |
| Figure 3-25 | Compute_MCUs Symbol | 3-35 |
| Figure 3-26 | Example Pipelined and Non-pipelined Systems | 3-36 |
| Figure 3-27 | Scan Decoding Modules | 3-37 |
Figure 4-18  Controller with I/O Pads .................................................................4-16

Figure 4-19  Controller Layout.................................................................4-18

Figure 5-1  Two Types of Buffering in Pipelines ...........................................5-3
### LIST OF TABLES

<table>
<thead>
<tr>
<th>Table 1-1</th>
<th>Examples of Compression</th>
<th>1-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1-2</td>
<td>Advantages and Disadvantages of Differing Implementations</td>
<td>1-8</td>
</tr>
<tr>
<td>Table 1-3</td>
<td>Functionality of Programs</td>
<td>1-11</td>
</tr>
<tr>
<td>Table 2-1</td>
<td>Levels of JPEG implementation</td>
<td>2-5</td>
</tr>
<tr>
<td>Table 2-2</td>
<td>SSSS Values for DC coding</td>
<td>2-9</td>
</tr>
<tr>
<td>Table 2-3</td>
<td>Additional bits for DC and AC coding</td>
<td>2-9</td>
</tr>
<tr>
<td>Table 2-4</td>
<td>Example DC Huffman coding</td>
<td>2-10</td>
</tr>
<tr>
<td>Table 2-5</td>
<td>SSSS Symbols for AC Huffman coding</td>
<td>2-11</td>
</tr>
<tr>
<td>Table 2-6</td>
<td>Example AC Huffman coding</td>
<td>2-11</td>
</tr>
<tr>
<td>Table 2-7</td>
<td>JPEG Markers</td>
<td>2-21</td>
</tr>
<tr>
<td>Table 3-1</td>
<td>VHDL Test Bench Modules</td>
<td>3-3</td>
</tr>
<tr>
<td>Table 3-2</td>
<td>VHDL Test Bench Interconnecting Signals</td>
<td>3-3</td>
</tr>
<tr>
<td>Table 3-3</td>
<td>Controller Signals</td>
<td>3-11</td>
</tr>
<tr>
<td>Table 3-4</td>
<td>Work Request Codes</td>
<td>3-11</td>
</tr>
<tr>
<td>Table 3-5</td>
<td>NextByte Signals</td>
<td>3-15</td>
</tr>
<tr>
<td>Table 3-6</td>
<td>NextByte State Table</td>
<td>3-16</td>
</tr>
<tr>
<td>Table 3-7</td>
<td>ISR Signal Definitions</td>
<td>3-20</td>
</tr>
<tr>
<td>Table 3-8</td>
<td>ISR Codes</td>
<td>3-20</td>
</tr>
<tr>
<td>Table 3-9</td>
<td>FindSOI Signal Definitions</td>
<td>3-21</td>
</tr>
<tr>
<td>Table 3-10</td>
<td>FindSOF Signal Definitions</td>
<td>3-24</td>
</tr>
</tbody>
</table>
Table 3-11  Frame Header Descriptors ................................................................. 3-25
Table 3-12  Fr_Header Signal Definitions .............................................................. 3-26
Table 3-13  dec_frame Interface Definitions ....................................................... 3-28
Table 3-14  FindSOS Signal Definitions ............................................................... 3-29
Table 3-15  Decode_Scan Signal Definitions .......................................................... 3-31
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASIC</td>
<td>Application Specific Integrated Circuit</td>
</tr>
<tr>
<td>bit</td>
<td>A binary digit</td>
</tr>
<tr>
<td>chrominance</td>
<td>The difference between a color and a reference white at the same luminance.</td>
</tr>
<tr>
<td>CMY</td>
<td>Cyan, Magenta, and Yellow: used in a subtractive color device</td>
</tr>
<tr>
<td>CMYK</td>
<td>Cyan, Magenta, Yellow and black: used in a subtractive color device</td>
</tr>
<tr>
<td>CODEC</td>
<td>Combined encoder and decoder</td>
</tr>
<tr>
<td>continuous tone</td>
<td>An image whose components have more than one bit per sample</td>
</tr>
<tr>
<td>DCT</td>
<td>Discrete Cosine Transform</td>
</tr>
<tr>
<td>DIS</td>
<td>Draft International Standard</td>
</tr>
<tr>
<td>DPCM</td>
<td>Differential Pulsed Code Modulation</td>
</tr>
<tr>
<td>DOD</td>
<td>Department Of Defense</td>
</tr>
<tr>
<td>DSP</td>
<td>Digital Signal Processing</td>
</tr>
<tr>
<td>EDIF</td>
<td>Electronic Design Interchange Format</td>
</tr>
<tr>
<td>GIF</td>
<td>Graphics Interchange Format</td>
</tr>
<tr>
<td>IEEE</td>
<td>Institute of Electrical and Electronics Engineers</td>
</tr>
<tr>
<td>JPEG</td>
<td>Joint Photographic Experts Group</td>
</tr>
<tr>
<td>luminance</td>
<td>Provides a grayscale version of an image. The absolute, rather than relative, amount of light from each primary color source.</td>
</tr>
<tr>
<td>pixel</td>
<td>Picture Element. Sample in a digital image. A digital image is made up of an array of pixels.</td>
</tr>
<tr>
<td>RGB</td>
<td>Red, Green, and Blue: used in an additive color device</td>
</tr>
<tr>
<td>RTL</td>
<td>Register Transfer Language</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>VHDL</td>
<td>VHSIC Hardware Description Language</td>
</tr>
<tr>
<td>VHSIC</td>
<td>Very High Speed Integrated Circuits</td>
</tr>
<tr>
<td>VLSI</td>
<td>Very Large Scale Integration</td>
</tr>
<tr>
<td>YCbCr</td>
<td>A color transformation, related to YUV</td>
</tr>
<tr>
<td>YIQ</td>
<td>A color transformation, related to YUV</td>
</tr>
<tr>
<td>YUV</td>
<td>A color transformation, Y is the luminance, and U, V make up the chrominance.</td>
</tr>
</tbody>
</table>
1. Introduction

1.1 Overview

This thesis is divided into 5 chapters, plus appendices. Chapter one is a brief introduction to images, the JPEG (Joint Photographic Experts Group) still image compression standard and VHDL (Very high speed integrated circuits Hardware Description Language). A more in depth coverage of JPEG, especially relating to decompressing images, is found in chapter two. Chapter three covers the design and implementation of the JPEG decoder in VHDL. The results from the implementation of the decoder are described in chapter four. The last chapter is conclusions, containing ideas for future work, and reflections of lessons learned from this thesis.

1.2 Images

What is an image? Webster's Third New International Dictionary has one definition of an image as “a thing actually or seemingly reproducing another, as any likeness of an object produced on photographic-material” [Web86]. When people think of an image, most would refer to the traditional image associated with a piece of photographic paper or a color negative. Traditional photography relies upon chemical processes to store representations of an image. This method of storing images is similar to the use of analog circuits in electronics. To relate why digital still image compression is needed, a brief discussion of traditional photographs is used. A traditional photograph is taken by creating a negative of the image on filmbase called a “negative” To view a picture, the image is enlarged and printed as a positive image on photographic paper. If
the image is enlarged too much, the picture will be distorted and blurry. The negative can be thought of as a compressed image of the enlarged picture. But, the image quality is not distorted due to the negative, rather it is a result of enlarging the picture. This is the opposite case from a digital image, as will be seen in section 1.3. With digital images, it is the compressing of the image that causes the loss of data and picture quality. For a more in depth coverage of images and the human visual system refer to Digital Image Processing chapters 1 to 4 [Pra91].

1.3 Digital Imaging

With the advent of the electronic information age, the methods of capturing, transferring, displaying and storing images has expanded. Digital images can be created by a number of different methods. Some digital images are created by the use of scanners, digital copiers or facsimile machines. These methods are essentially creating copies of images that exist in another form. Another method for creating digital images is the use of digital cameras. With this method, the image is created and exists only in a digital format. With the expansion in the methods of using digital images comes an increase in the number of digital images. It is due to the increase in the number of digital images and the size of digital images that a compression scheme was developed. To understand why a compression scheme was needed, a brief discussion of what comprises a digital image is required. The basic element of a digital image is a pixel. A pixel is the smallest sample in a digital image. Pixels can be samples of differing precision depending upon the application. A Bi-level image, such as a facsimile will only have 1 bit per pixel. A monochrome still image will have typically 8-bits per pixel. For more precise images such as digitized X-
rays, additional bits per pixel are used. The addition of color adds more bits per pixel. Color can be represented by a number of methods which separate the color value into a number of primary color values. Some examples are RGB (Red, Green, Blue), CMY (Cyan, Magenta, Yellow), CMYK (Cyan, Magenta, Yellow, Black), YUV (Y = luminance level, U, V = chrominance levels), YIQ (Y = luminance level, I, Q = chrominance levels related to U and V), and YCbCr (Y = luminance level, Cb, Cr = chrominance levels related to U and V). To display an image in color, every pixel needs data from each part of the color representation. For example, to display a RGB image, data elements for the red, green and blue primary levels are needed. A typical RGB image will have 8-bits per primary color per pixel, thus 24-bits are needed for each pixel. It should be noted that the JPEG standard does not specify the method for representing color images; any one of the previously mentioned methods may be used. A color image made up of 640 pixels by 480 pixels requires a total of 307,200 pixels. Since each pixel is 3 bytes (1 byte = 8-bits, each color needs 8-bits), the image requires 921,600 bytes of space to be stored. Relating this to a roll of film, 24 images of this size would need 22,118,400 (approx. 22 MB) bytes of space. Storing multiple rolls of “digital film” would quickly use most computer systems disk space. One method to avoid this problem is to purchase more disk space, but that is not very cost effective. The other method is to compress the images. With this example it is readily apparent that some means of compression is needed.

1.4 JPEG

JPEG stands for Joint Photographic Experts Group. JPEG was formed to create an international standard for a compression technique for color and grayscale continuous tone
still images. The standard was to fill a gap in compression techniques between the Bi-Level standard, JBIG, and the future digital motion compression standard MPEG. It should be noted that JPEG is designed for still images much like those in photographs. It will not perform well on non-continuous tone images, such as Bi-level images. For a further introduction on JPEG refer to references [Leg91], [Mit91], [Mit92], and [Wal92]. For a more in depth coverage of JPEG, William Pennebaker and Joan Mitchell’s book JPEG Still Image Data Compression Standard [Pen93] is recommended. This book contains an excellent overview of JPEG for all levels of users. It also contains a draft copy of the JPEG standard, which is useful for those who are implementing this standard.

An example of the amount of compression that JPEG is capable of is shown using a 125 x 125 pixel grayscale image. With 8-bits per pixel, the uncompressed image would need 15,625 storage bytes. With some additional overhead involved to store the image size, the number of bits per pixel and the number of colors, the actual file size is 15,722 bytes. A visual example of the effects of data compression on an image is shown in figure 1-1. Picture A is the original uncompressed image. Table 1-1 shows the file size of each image and the amount of compression done. On a workstation display, a very slight distortion was noted for images C and D and image E was very distorted. Picture F is the original image saved in Graphics Interchange Format (GIF). GIF format is a popular format found on computer systems today. Its drawbacks are that it does not compress well, and can only use 8-bits per pixel. Note, for small images, the GIF format will not reduce the amount of image data, but will take up more space than the original image. This can be seen by comparing the sizes of image A to image F in table 1-1. For larger
images, GIF will compress the amount of data used to a size less than the original. A GIF image will still take up more space than a JPEG compressed image.

<table>
<thead>
<tr>
<th>Figure</th>
<th>File Size (bytes)</th>
<th>Compression (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A (original)</td>
<td>15722</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>3676</td>
<td>77</td>
</tr>
<tr>
<td>C</td>
<td>3225</td>
<td>79</td>
</tr>
<tr>
<td>D</td>
<td>2041</td>
<td>87</td>
</tr>
<tr>
<td>E</td>
<td>1006</td>
<td>94</td>
</tr>
<tr>
<td>F</td>
<td>18158</td>
<td></td>
</tr>
</tbody>
</table>

Table 1-1 Examples of compression

Figure 1-1 Example compressed images
1.5 Types of JPEG implementations

Various factors will determine how the JPEG standard is implemented in systems. Factors include cost, speed of processing needed, ease of implementation, ease of making changes and other design considerations. Three methods of implementing JPEG are discussed in this section, and some of the factors that drive their use are noted. One popular method is the use of software to implement the JPEG standard. This method can be found embedded in commercial software programs that display and manipulate digital images. The JPEG standard has also been implemented by the Independent JPEG Group, which offers free C software for non-commercial and commercial use. Refer to page 280 in [Pen93] for more information on how to obtain this software. Implementing JPEG in software has the advantages that it can be easily and quickly created, it can be easily updated with new features, it is a low cost project, and it can be used across many computer platforms. A downside to implementing the project in software is the slow speed at which images are compressed or decompressed.

Another method for implementing the JPEG still image standard is the use of specialized image processing DSP chip sets. An example of such a DSP is the DSP96002 processor. Refer to [Ram93] for an example such of a DSP implementation. A DSP processor implementation is a mix of a hardware and a software solution. The DSP chip set is designed specifically to implement commonly used image processing functions. Specialized software is provided to allow a designer to implement complex image processing tasks. It should be noted that DSPs do not directly implement the JPEG standard, they only provide a set of hardware/software tools that can be used to implement
the standard. Advantages of a DSP solution are much faster compression/decompression speeds. Also since the actual standard is implemented using software, the ease of creating and making changes is also an advantage. The disadvantages are that the implementation is limited only to one specific DSP processor. This may not be a disadvantage, if the implementation is to be embedded into a product. If space is a major concern, the DSP solution may not be as viable as a total hardware implementation.

The last method of implementing the JPEG standard is a pure hardware design. The complete JPEG standard can be implemented into a single VLSI integrated circuit. Some examples of this can be found in [Jai92], [Leo91], [Oga92], and [Rue93]. The main advantage of a VLSI implementation is speed. The second advantage is size. A single chip will occupy much less space than a DSP set, or a computer that runs software. The disadvantages are that a VLSI design is harder to implement and to make changes. Cost, speed, space and market focus are some of the factors that will determine which of the three methods are chosen to implement the standard. Another factor is how much of the standard is to be implemented. Requirements may call for the complete standard, or only a portion, such as just the decoder. Table 1-2 outlines the advantages and disadvantages of each method.
<table>
<thead>
<tr>
<th>Need</th>
<th>Software</th>
<th>DSP</th>
<th>Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost</td>
<td>low</td>
<td>medium</td>
<td>high</td>
</tr>
<tr>
<td>Speed</td>
<td>slow</td>
<td>fast</td>
<td>fastest</td>
</tr>
<tr>
<td>Ease of</td>
<td>easy</td>
<td>medium, requires</td>
<td>complex</td>
</tr>
<tr>
<td>implementation</td>
<td></td>
<td>knowledge of specialized software</td>
<td></td>
</tr>
<tr>
<td>Ease to change</td>
<td>easy, but must be distributed</td>
<td>easy, but requires distribution</td>
<td>complex, requires new device</td>
</tr>
<tr>
<td>Cross Platform</td>
<td>Yes</td>
<td>No, specific to only that DSP set</td>
<td>No, specific to only that VLSI chip</td>
</tr>
<tr>
<td>Size</td>
<td>large</td>
<td>medium</td>
<td>small</td>
</tr>
</tbody>
</table>

Table 1-2 Advantages and Disadvantages of Differing Implementations

1.6 VHDL

VHDL stands for VHSIC Hardware Description Language, with VHSIC meaning Very High Speed Integrated Circuits. VHDL is much like a typical software language, such as C or FORTRAN, except that it has some concurrent constructs that can specifically be used to design digital circuits. VHDL is a result of a DOD requirement to have one hardware description language for use by vendors. One common language allows different vendors to be able to share and connect designs. Due to increased needs, the language was transferred to the IEEE for standardization. Since then it has undergone numerous reviews and changes and the latest official version has been documented in the VHDL 1993 Language Reference Manual.

"Why use VHDL when current schematic capture programs do the same function?" is a question often asked. One reason to use VHDL over traditional schematic
capture is that VHDL is a recognized IEEE standard, and thus can be exchanged between companies and vendors. No matter what vendor creates a VHDL compiler, the VHDL source code should be acceptable and usable. Note this assumes that the compiler does in fact comply with the IEEE VHDL standard. This is not so with schematic capture. A circuit diagram often can not be exchanged between two parties using different vendor schematic capture software unless interchanged using the Electronic Design Interchange Format (EDIF). Another reason is that changing the source code is easier than having to edit circuit diagrams. An additional reason is that VHDL can model different levels of abstraction of a design. The different models of abstraction are behavioral, data flow and structural. The level of abstraction goes from high level, for behavioral, to a low level for the structural model. A behavioral model is a sequential model of the design that is very similar to a high level programming language such as “C” or Pascal. The data flow model is the next level down in abstraction and it models the flow of data in a design in a Register Transfer Language (RTL) type format. The lowest level of abstraction is the structural model, which is a model of a set of interconnects in a design. This is much like a schematic layout. At this level, the model is not sequential, but concurrent. Events occur concurrently to each other, just as they do in a circuit. From the lower levels of abstraction, circuits and layouts can be synthesized. Thus an ASIC can be entirely written, simulated and designed using VHDL without the need for schematics. For further reading on VHDL references [Bha92] and [Coe89] are recommended. References [Hub91] and [Leu89] incorporate the design cycle of an ASIC with VHDL.
1.7 Design Brief

This design was accomplished in two parts. The first part was a behavioral model of the JPEG decoder. The behavioral model was done in C. A C program was chosen to do the behavioral model over VHDL due to a change in the computer systems at the time this thesis was started. Appendix C contains the C program dec.c that implements the baseline decoder. This program implements color images, whereas the design in VHDL only does grayscale images. Ideally the behavioral model would be done in VHDL, to help in the design of the structural model. Figure 1-2 shows an overview of how the VHDL design model simulates decoding JPEG compressed images.

![Figure 1-2 Overview of design test flow](imageURL)
Table 1-3 defines the functionality of each block in the above figure.

<table>
<thead>
<tr>
<th>Block</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>orig.jpg</td>
<td>Original, raw data JPEG compressed grayscale image</td>
</tr>
<tr>
<td>conv</td>
<td>Converts the raw JPEG file into an ASCII file. This is due to the textio</td>
</tr>
<tr>
<td></td>
<td>functions available in VHDL which only read ASCII.</td>
</tr>
<tr>
<td>vhdl.dat</td>
<td>ASCII JPEG file that will be read in by the VHDL test bench</td>
</tr>
<tr>
<td>VHDL test</td>
<td>Simulation of VHDL design.</td>
</tr>
<tr>
<td>bench</td>
<td></td>
</tr>
<tr>
<td>vhdl.pgm</td>
<td>Output from VHDL simulator</td>
</tr>
<tr>
<td>display</td>
<td>Converts the output from blocks of 8x8 to line by line for proper display</td>
</tr>
<tr>
<td>orig.pgm</td>
<td>original uncompressed image</td>
</tr>
<tr>
<td>compare</td>
<td>Compares original image to the uncompressed image</td>
</tr>
<tr>
<td>results</td>
<td>notes any pixel that differs from the original by +/- 1</td>
</tr>
<tr>
<td>XV</td>
<td>Display program</td>
</tr>
</tbody>
</table>

Table 1-3 Functionality of programs

1.8 Software used

This thesis was designed using Mentor Graphics Corporation software tools on HP 9000/700 series workstations. The tools used from the Mentor package were: Design Architecture, Quicksim II, and Autologic. Design Architecture contains System 1076 which was used to write and compile the VHDL source code. Quicksim II is the digital simulator, and it was used to simulate the design. The design was partially synthesized using the Autologic tool. The entire design was not synthesized due to its large size, and a lack of time and disk space. To display the decompressed images the XV program by John Bradley was used on the HP workstations. This thesis was written using Microsoft Word 6.0.
2. Concepts

This chapter is divided into two parts. The first part is a condensed overview of what comprises this thesis. The JPEG decoder is presented in some detail in the first section of this chapter. Again, the book JPEG Still Image Data Compression Standard [Pen93] by William Pennebaker and Joan Mitchell is highly recommended for a more in-depth coverage. Also the actual standard is very useful in understanding how JPEG works. If any discrepancies are noted between this thesis and the standard, the standard is the correct source. The second part of this chapter is a short introduction to two of the methods for modeling designs in VHDL. Behavioral and structural modeling examples are given and contrasted.

2.1 JPEG

2.1.1 JPEG Brief

There are three major parts to the JPEG still image compression standard. They are the encoder, the decoder and the interchange format. The encoder is the part that compresses images, the decoder takes a compressed image and expands it for viewing. The interchange format is the data that are passed along with the compressed image that allows a decoder designed by any second party to decompress that image. The three parts are discussed in some detail in the following three paragraphs.

The encoder’s function is to take a raw image, whose format is not specified by JPEG, and compress it into a smaller size. The JPEG baseline encoder uses a lossy compression scheme. Lossy refers to the fact that data will be lost when the image is
compressed. This is usually not a problem with still images compressed by JPEG. There will be some data loss, but the image quality will remain the same to the viewer’s eye. Users can trade off the amount of compression against degradation by modifying values in a “quantization table”. The major purpose of JPEG, is to obtain as much compression as possible with little or no perceptible loss in visual image quality.

Figure 2-1 shows a basic layout of the encoder. The inputs to the encoder are the digital source image data and table specifications. The digital image data can be just grayscale data, or it can be different planes of color data. In a baseline encoder, only 8-bits per pixel are used. The other input to the encoder is table specifications, which are the Huffman tables and quantization tables. These tables can be optimized from the current image being compressed, or they can be a set of standard (hard coded) tables permanently set into the encoder. JPEG does not require which method is used. The output of the encoder is the compressed image data.

![Figure 2-1 Encoder](image)

The decoder is the second major part of the JPEG standard. The decoder takes compressed image data and uncompresses that data in some usable form. The final structure of the uncompressed image data is not dependent upon the JPEG standard. The
input to the decoder is compressed image data. From the compressed data the table specifications, Huffman and quantization, may be obtained. Note, if the decoder is part of a system that has advance knowledge of the contents of these tables, they need not be present in the compressed data, but will already be available in the decoder. An example would be a system that transits image data from point to point that is built by one vendor. Figure 2-2 shows the layout of the JPEG decoder.

![Diagram of JPEG decoder]

Figure 2-2 Decoder

The third major portion of the JPEG standard is the interchange format. The interchange format is the compressed data and all other associated data involved in the encoding and decoding of an image. The interchange format guarantees that a compressed image can cross the boundary between application environments, regardless of how each environment internally associates tables with compressed images [ISO93]. Figure 2-3 shows an example of what is meant by the interchange format.
There are three types of interchange format. The first type of interchange format contains all information needed to decode an image. The second type is an abbreviated format for compressed image data. The abbreviated format does not need to include all tables used by the decoder. This would be used in applications where the tables were known at the decoder beforehand. The third type of interchange format is abbreviated format for table specification data. With this format, only the table data are contained in the compressed data. This would be used to set up the decoder, so that the following images would not have to carry the table data (Note: all images using this method would be of the second type of interchange format).

There are four different levels at which the JPEG standard can be implemented. The first is the baseline level. All encoders, decoders and codecs except the lossless, must implement this lowest level. The other three levels are extended discrete cosine transform (DCT) based, lossless and hierarchical. Figure 2-4 shows the hierarchy of the JPEG
standard. Table 2-1 lists the four levels of implementation, and the major differences between them.

![Diagram of JPEG hierarchy]

**Figure 2-4 JPEG hierarchy**

<table>
<thead>
<tr>
<th>Levels of JPEG implementation</th>
<th>Baseline - sequential DCT based</th>
<th>Extended DCT based process</th>
<th>Lossless</th>
<th>Hierarchical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline - sequential DCT based</td>
<td>DCT based process 8-bits per pixel Sequential coding Huffman coding, 2 AC tables and 2 DC tables Decoders shall process scans with 1,2,3 and 4 components Interleaved and non-interleaved scans</td>
<td>DCT based process 8 or 12-bits per pixel Sequential or progressive processing Huffman or arithmetic coding, 4 AC and 4 DC tables Decoders shall process scans with 1,2,3 and 4 components Interleaved and non-interleaved scans</td>
<td>Predictive process, not DCT based 2 to 16 bits per pixel Sequential processing Huffman or arithmetic coding : 4 DC tables Decoders shall process scans with 1,2,3 and 4 components Interleaved and non-interleaved scans</td>
<td>Multiple frames (non-differential and differential) Uses extended DCT based or lossless process Decoders shall process scans with 1,2,3 and 4 components Interleaved and non-interleaved scans</td>
</tr>
</tbody>
</table>

Table 2-1 Levels of JPEG implementation
The following is a list of each term used to define the baseline system. For more detail on the other three levels refer to the standard.

DCT based: Encoders and decoders are based upon the use of the discrete cosine transform. Refer to section 2.1.2 for more information on the discrete cosine transform.

8-bits per pixel: For each scan, eight bits only are used.

sequential coding: Image is coded from upper left to the lower right in 8 by 8 blocks. This is done all in one scan. The other method is progressive, which uses multiple scans in which the picture quality starts out very low and progressively gets better; much like focusing a lens.

Huffman coding: Arithmetic coding is not used to code the coefficients. Refer to section 2.1.3 for more information on Huffman coding.

1,2,3, or 4 components: An image can have one to four components. Examples are 1 component for grayscale, 3 for RGB and 4 components for CMYK.

interleaved or non-interleaved: For color images only. An interleaved scan is one where the components are intercoded into the compressed data. An example would be to code a 8x8 red unit, then a 8x8 green and then a 8x8 blue. As each set of 8x8 blocks (each set has the three primary color components) are decoded that part of the image can be displayed in full color. With non-interleaved, each color is coded...
separately in the scan. All of the red is done, the green and finally the blue. When this method is decoded, the image displayed will not look correct until the final color is decoded.

2.1.2 Huffman Coding

This section explains how Huffman coding is used in the JPEG image compression standard. For a complete background on the theory of Huffman coding, refer to [Huf62]. Huffman coding is performed on the compressed data after it has been quantized and transformed by the discrete cosine transform. Once the data has been transformed it is decorrelated, and thus can be "compressed independently without concern about correlation between coefficients" [Pen93]. The coefficients are coded in two parts, the DC coefficient and AC coefficients. The DC coefficient is coded using a differential pulse code modulation (DPCM) model. A DPCM model is one in which the current DC coefficient is coded with respect to its difference from the last DC coefficient. AC coefficients are coded in a manner which uses the runs of zeros created by zigzag ordering to an advantage.
DC coding is done by using the following steps. A step by step example is included.

1. Find difference: DIFF = Current DC value - Last DC value. The Last DC value is also called the predictor value (PRED). Initialize last DC value to zero. Figure 2-5 shows a diagram of the DPCM model for encoding.

![DPCM Model for Encoding](image)

**Example, Step 1**

- Current DC = 8
- PRED = 0 \((Initial\ Value)\)
- Diff = -8

2. Look up SSSS (SSSS is a variable) code, see table 2-2 below. Select SSSS according to the value of the DPCM difference.

**Example, Step 2**

SSSS = 4
3. Select additional bits. Note: a leading 1 means the difference is positive, and a leading 0 means the difference is negative. See table 2-3

**Example, Step 3**

*Additional Bits = 0111*

<table>
<thead>
<tr>
<th>SSSS</th>
<th>DPCM Difference</th>
<th>Additional Bits (binary)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>-1,1</td>
<td>0,1</td>
</tr>
<tr>
<td>2</td>
<td>-3,-2,2,3</td>
<td>000,011,100,111</td>
</tr>
<tr>
<td>3</td>
<td>-7,...,-4,4,...,7</td>
<td>000,...,011,100,111</td>
</tr>
<tr>
<td>4</td>
<td>-15,...,-8,8,...,15</td>
<td>000,...,011,100,111</td>
</tr>
<tr>
<td>5</td>
<td>-31,...,-16,16,...,31</td>
<td>00000,...,01111,10000,11111</td>
</tr>
<tr>
<td>6</td>
<td>-63,...,-32,32,...,63</td>
<td>00000,...,01111,10000,11111</td>
</tr>
<tr>
<td>7</td>
<td>-127,...,-64,64,...,127</td>
<td>0000000,...,0111111,1000000,1111111</td>
</tr>
<tr>
<td>8</td>
<td>-255,...,-128,128,...,255</td>
<td>000000000,...,011111111,100000000,111111111</td>
</tr>
<tr>
<td>9</td>
<td>-511,...,-256,256,...,511</td>
<td>0000000000,...,0111111111,1000000000,1111111111</td>
</tr>
<tr>
<td>10</td>
<td>-1023,...,-512,512,...,1023</td>
<td>00000000000,...,01111111111,10000000000,11111111111</td>
</tr>
<tr>
<td>11</td>
<td>-2047,...,-1024,1024,...,2047</td>
<td>000000000000,...,011111111111,100000000000,111111111111</td>
</tr>
</tbody>
</table>

Table 2-2 SSSS values for DC coding

<table>
<thead>
<tr>
<th>SSSS</th>
<th>DPCM Difference</th>
<th>Additional bits (binary)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>-1,1</td>
<td>0,1</td>
</tr>
<tr>
<td>2</td>
<td>-3,-2,2,3</td>
<td>000,011,101,111</td>
</tr>
<tr>
<td>3</td>
<td>-7,...,-4,4,...,7</td>
<td>000,...,011,100,111</td>
</tr>
<tr>
<td>4</td>
<td>-15,...,-8,8,...,15</td>
<td>000,...,011,100,111</td>
</tr>
<tr>
<td>5</td>
<td>-31,...,-16,16,...,31</td>
<td>00000,...,01111,10000,11111</td>
</tr>
<tr>
<td>6</td>
<td>-63,...,-32,32,...,63</td>
<td>00000,...,01111,10000,11111</td>
</tr>
<tr>
<td>7</td>
<td>-127,...,-64,64,...,127</td>
<td>0000000,...,0111111,1000000,1111111</td>
</tr>
<tr>
<td>8</td>
<td>-255,...,-128,128,...,255</td>
<td>000000000,...,011111111,100000000,111111111</td>
</tr>
<tr>
<td>9</td>
<td>-511,...,-256,256,...,511</td>
<td>0000000000,...,0111111111,1000000000,1111111111</td>
</tr>
<tr>
<td>10</td>
<td>-1023,...,-512,512,...,1023</td>
<td>00000000000,...,01111111111,10000000000,11111111111</td>
</tr>
<tr>
<td>11</td>
<td>-2047,...,-1024,1024,...,2047</td>
<td>000000000000,...,011111111111,100000000000,111111111111</td>
</tr>
</tbody>
</table>

Table 2-3 additional bits for DC and AC coding
4. Concatenate the binary value of SSSS and the additional bits together. This becomes the code used for the DC coefficient. The size of the code will vary for each DPCM difference.

*Example, Step 4*

code (binary) = 1000111

Table 2-4 illustrates an example of Huffman coding of DC coefficients. Note, the first column is from the example used previously.

<table>
<thead>
<tr>
<th>Quantized DC value</th>
<th>8</th>
<th>9</th>
<th>8</th>
<th>-6</th>
<th>-8</th>
<th>-3</th>
<th>3</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRED value</td>
<td>0</td>
<td>8</td>
<td>9</td>
<td>8</td>
<td>-6</td>
<td>-8</td>
<td>-3</td>
<td>3</td>
</tr>
<tr>
<td>DPCM difference</td>
<td>-8</td>
<td>1</td>
<td>-1</td>
<td>-14</td>
<td>-2</td>
<td>5</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>SSSS</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Additional bits</td>
<td>111</td>
<td>1</td>
<td>0</td>
<td>001</td>
<td>00</td>
<td>101</td>
<td>110</td>
<td>-</td>
</tr>
<tr>
<td>code (binary)</td>
<td>1000111</td>
<td>11</td>
<td>10</td>
<td>100001</td>
<td>1000</td>
<td>11101</td>
<td>11110</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2-4 Example DC Huffman coding

AC Huffman coding is accomplished by using the following steps:

1. Calculate RRRR. RRRR is the number of repeating zeros in a row before a non-zero coefficient. If there are 16 zeros in a row, they are coded as using the ZRL (zero run length) symbol. If the end of the buffer is reached and no non-zero coefficients are reached, the EOB (end of buffer) code is used.
2. Calculate $SSS$. $SSS$ is derived from the non-zero number reached when determining $RRR$. Use table 2-5 to calculate $SSS$.

<table>
<thead>
<tr>
<th>$SSS$</th>
<th>non-zero coefficient value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-1,1</td>
</tr>
<tr>
<td>2</td>
<td>-3, -2, 2, 3</td>
</tr>
<tr>
<td>3</td>
<td>-7, ..., -4, 4, ..., 7</td>
</tr>
<tr>
<td>4</td>
<td>-15, ..., -8, 8, ..., 15</td>
</tr>
<tr>
<td>5</td>
<td>-31, ..., -16, 16, ..., 31</td>
</tr>
<tr>
<td>6</td>
<td>-63, ..., -32, 32, ..., 63</td>
</tr>
<tr>
<td>7</td>
<td>-127, ..., -64, 64, ..., 127</td>
</tr>
<tr>
<td>8</td>
<td>-255, ..., -128, 128, ..., 255</td>
</tr>
<tr>
<td>9</td>
<td>-511, ..., -256, 256, ..., 511</td>
</tr>
<tr>
<td>10</td>
<td>-1023, ..., -512, 512, ..., 1023</td>
</tr>
<tr>
<td>11</td>
<td>-2047, ..., -1024, 1024, ..., 2047</td>
</tr>
</tbody>
</table>

Table 2-5 $SSS$ Symbols for AC Huffman coding

3. Calculate $RUN-SIZE$. $RUN-SIZE$ is equal to $(16 \times RRRR) + SSS$.  

4. Add on additional bits to code the sign and magnitude of the non-zero value in the $RUN-SIZE$ symbol. Use the non-zero value and table 2-3 to determine the additional bits. For example, a -14 would need additional bits of 0001 (see table 2-3: for $SSS = 4$, -15 is 0000, -14 is 0001 and -8 is 0111).

See appendix F1.2 in the draft international standard [ISO93] for specific information on how Huffman coding for the JPEG standard is accomplished. Shown below is table 2-6 [Pen93] which is an example of AC Huffman coding.

<table>
<thead>
<tr>
<th>Zigzag index</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>...</th>
<th>63</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC descriptor</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-14</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>RRRR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&lt;---</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSSS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>RUN-SIZE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&lt;---</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Additional bits</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Huffman code</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&lt;---</td>
<td>EOB</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2-6 Example AC Huffman Coding
2.1.3 Discrete Cosine Transform

The two dimensional discrete cosine transform is one of the major components that allows compression to be possible in the JPEG standard. The transform does not actually do any compression, but it transforms coefficients into a format so that compression can be done. For an in-depth coverage of the discrete cosine transform, the book *Discrete Cosine Transform* [Rao90] by K.R. Rao and P. Yip is recommended. Also chapter 4 of Pennebaker and Mitchell’s book [Pen93] is a good source for an introduction to the transform. The 2-D forward discrete cosine transform used by JPEG is defined by

\[
S_{uv} = \frac{1}{4} C_u C_v \sum_{x=0}^{7} \sum_{y=0}^{7} S_{x,y} \cos \left( \frac{(2x + 1)u\pi}{16} \right) \cos \left( \frac{(2y + 1)v\pi}{16} \right)
\]

where \( C_u \) and \( C_v = \frac{1}{\sqrt{2}} \) for \( u, v = 0 \); \( C_u, C_v = 1 \) otherwise.

One of the properties of two dimensional DCT is that it can be broken down into two 1-dimensional transforms. The one dimensional transform that applies to JPEG is defined as:

\[
S_u = \frac{1}{2} C_u \sum_{x=0}^{7} S_x \cos \left( \frac{(2x + 1)u\pi}{16} \right)
\]

This thesis uses the two 1-D DCT approach to model the Inverse DCT. If an actual design were to be constructed in VLSI, then the most appropriate DCT method would be used. Another option is to use a commercial chip that is designed for the DCT. The DCT transform, if implemented as defined, would require a large amount of time and computational power to implement. Thus simplification exists for the discrete cosine
transform to reduce the time to perform. Most of these simplifications attempt to reduce the number of multiplications and/or additions. Doing the DCT using separable 1-D transforms results in 1,024 multiplications and 896 additions. A method developed by Feig [Fei92] has reduced the number of multiplications to 54 with 464 additions and 6 arithmetic shifts. Other methods use the fast fourier transform (FFT) to implement the DCT. Different implementations will use different approaches to speed up the transform. A platform that can do fast additions but slow multiplications should use an algorithmic technique which reduces the number of multiplications.

2.2 A JPEG encoder and decoder

The JPEG encoder and decoder are covered in this section. The following is a step-by-step coverage of how the JPEG encoder functions. Each step contains a brief description of its functions. Take note of where the actual data loss and compression occur.

1. Level shift to a signed representation. For \( P \) bits, subtract from each pixel \( 2^{P-1} \).

For an 8 bit mode, the level shift is 128. For the baseline system this will take an unsigned 8 bit number (0 to 255) and create a signed 8 bit number (-128 to 127). This step can be done as part of step 2.

2. Image component samples are grouped into 8x8 blocks. A 8x8 block of pixels is the basic data unit that is compressed.

3. Each block is transformed by the forward discrete cosine transform (FDCT) into 64 values, called the DCT coefficients. One coefficient (upper left) is the DC
coefficient, the other 63 are the AC coefficients. No specific algorithm for implementing the DCT is specified by the JPEG standard. Note, there is no data loss or compression from this step. If an inverse transform were performed at this point, the original data would be recovered (note: there may be a slight loss due to rounding and accuracy of floating point operations with cosines). The purpose of this step is to take a highly correlated group of data and decorrelate it. The coefficients towards the upper left corner are the lower frequency points. The coefficients towards the lower right corner are the higher frequency points. The purpose of decorrelating the data is to be able to eliminate some data points without adversely affecting other data points. The following is the forward discrete cosine transform (FDCT) used by the JPEG standard. For an input of 8-bit precision (baseline) the FDCT transform will require an 11-bit two's complement output. 12-bit inputs require an 15-bit output.

FDCT:

\[ S_{uv} = \frac{1}{4} C_u C_v \sum_{x=0}^{7} \sum_{y=0}^{7} a_{xy} \cos \left( \frac{(2x + 1)u\pi}{16} \right) \cos \left( \frac{(2y + 1)v\pi}{16} \right) \]

where \( C_u and C_v = \frac{1}{\sqrt{2}} \) for \( u, v = 0 \); \( C_u, C_v = 1 \) otherwise

4. The 64 coefficients are quantized using one of 64 corresponding values from a quantization table. Quantization is done by dividing each coefficient by a
corresponding number in the quantization array. Results are rounded to the nearest integer. Figure 2-6 shows an example.

Figure 2-6 Example quantization

No default table is specified by the JPEG standard. The table used in this example is the default table from the Independent JPEG Group software. It is also the same table specified by the draft ISO compliance testing document.
There is no compression at this stage, as there are still 64 coefficients. But at this step data loss will occur. The lower frequency points are the most important in reconstructing an image, while the high frequency points are the least important in reconstructing an image. Thus the higher frequency coefficients are divided down to obtain as many zeroes as possible, hence the high frequency points are divided by large quantization coefficients. The values in the quantization table are selected so that step 6 will be able to maximize the number of zeroes that can be placed together in a row. The low frequency points are quantized by much smaller numbers to prevent them from becoming zero. All numbers that do not quantize to zero can be restored by a reverse quantization using the same table, and thus have no loss. All numbers in the 8x8 data block that do divide to zero cannot be recovered, and thus are lost. Dividing too many numbers, especially in the lower frequency range, to zero will result in the reconstructed image being distorted.

5. The DC coefficient is encoded using the difference with respect to the last DC coefficient.

\[
\text{coded DC term} = \text{Present Quantized DC} - \text{Last Quantized DC}
\]

This preserves the DC coefficient with a lossless coding scheme, except for quantization error.
6. The coefficients are ordered with a zigzag ordering. This is done to place as many zeros in a row as possible.

7. Entropy encode the coefficients using Huffman coding. Tables must be supplied to the encoder. The DC coefficient is Huffman coded using the DC tables, and the AC coefficients are coded using the AC tables. At this stage the compression will occur. With more zero runs, more compression will occur. Figure 2-9 shows the complete overall encoding flow.
Decoding is essentially the reverse of encoding, therefore this section on the decoder has only the essential information on its function. The following is a list of the steps done to decode compressed image data. Figure 2-9 shows the overall decoding flow.

1. Entropy decoding, using Huffman coding. The tables must be supplied. The result will be 64 coefficients in a 8 x 8 array.
   
   A. Decode the DC coefficient, using DC Huffman table and predictor from last DC coefficient. DC value = Last DC value + decode value.
   
   B. Decode the AC coefficients using the AC Huffman table.

2. Undo the zigzag reordering.

3. De-quantize the coefficients. Table(s) must be supplied.
4. Run the coefficients through the inverse discrete cosine transform (IDCT).

\[ s_{xy} = \frac{1}{4} \sum_{u=0}^{2} \sum_{v=0}^{2} C_u C_v S_{uw} \cos \left( \frac{2x + 1)u \Pi}{16} \right) \cos \left( \frac{2y + 1)v \Pi}{16} \right) \]

where \( C_u \text{ and } C_v = \frac{1}{\sqrt{2}} \) for \( u, v = 0 \), \( C_u, C_v = 1 \) otherwise

5. Level shift; for an 8 bit image, add 128 to each pixel.

6. Data will be an 8x8 block. Reconstruct the image with 8x8 blocks.

![Decoding Flow Diagram](image)

Figure 2-9 Decoding Flow

The JPEG standard does not specify how to implement the above steps, it only specifies what comprises the encoder and decoder. Different implementations, such as software and hardware, would implement each block using the strong points of their respective technologies.
2.3 Interchange Format

This section contains a brief description of the interchange format found in a compressed image that can be decoded by this project. This is not meant to be an exhaustive description of the complete interchange format specified by the JPEG standard. For the complete description refer to the standard. The interchange format consists of marker, table data and the image data. Markers denote specific sections in the compressed data. An example of such a marker is the Define Huffman Table(s), DHT, marker. It denotes a section where Huffman table data are defined. Table data are the actual Huffman or quantization tables. Figure 2-10 shows a typical format of the compressed image data. For a complete description of each part of the compressed data refer to section B of the DIS [ISO94].

Figure 2-10 Typical compressed image data format.

The markers found in the compressed data are SOI, EOI, SOF, SOS, DQT, DHT, COM and APP. They are defined in table 2-7 along with their codes.
<table>
<thead>
<tr>
<th>Marker</th>
<th>Definition</th>
<th>Code (hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOI</td>
<td>Start Of Image</td>
<td>FFD8</td>
</tr>
<tr>
<td>EOI</td>
<td>End Of Image</td>
<td>FFD9</td>
</tr>
<tr>
<td>SOF</td>
<td>Start Of Frame</td>
<td>FFC0</td>
</tr>
<tr>
<td>SOS</td>
<td>Start Of Scan</td>
<td>FFDA</td>
</tr>
<tr>
<td>DQT</td>
<td>Define Quantization Table(s)</td>
<td>FFDB</td>
</tr>
<tr>
<td>DHT</td>
<td>Define Huffman Table(s)</td>
<td>FFC4</td>
</tr>
<tr>
<td>COM</td>
<td>COMment</td>
<td>FFFE</td>
</tr>
<tr>
<td>APP</td>
<td>Application specific information</td>
<td>FFE0-FFEF</td>
</tr>
</tbody>
</table>

Table 2-7 JPEG Markers

The SOF marker is found at the start of the frame header. At the start of the scan header, the SOS marker will be found. In the tables/misc. boxes there may be Huffman tables, quantization tables, comments or application specific information. These will all be preceded by their appropriate markers.

2.4 VHDL

2.4.1 Behavioral modeling

“A behavioral model is a model that describes the behavior of the hardware entity under test” [Fan94]. The entity under test can be thought of as a black box. There are inputs and outputs to this black box, but what implements the functionality of the black box need not be the actual circuitry. Behavioral modeling is used as the start of a top down design. Its use is to understand how a system is going to operate, not how to implement the system. Behavioral modeling can be implemented much quicker than lower level designs, and thus can give designers and customers an early look at how a system will work. Once a behavioral model is functional and approved, by both the designers and
customers, then a lower level design can be done. Figure 2-11[Bha92] shows a schematic that will be designed as both a behavioral model and a structural model (section 2.4.2).

![Diagram of a decoder circuit](image)

**Figure 2-11 A Decoder Circuit**

Below is the entity that implements the interconnections for the decoder circuit. It is common to both the behavioral and structural models. Thus the analogy to a black box, the view to the outside stays the same, but what goes on inside changes.

```vhdl
entity DECODER2x4 is
  port (A,B, ENABLE : in BIT; Z: out BIT_VECTOR(0 to 3));
end DECODER2x4;
```

**Figure 2-12 Example VHDL Entity**
The code that implements this entity as a behavioral model is shown below.

```vhdl
architecture DEC_SEQUENTIAL of DECODER2x4 is
begin
  process (A,B,ENABLE)
  variable ABAR, BBAR: BIT;
  begin
    ABAR := not A
    BBAR := not B;
    if (ENABLE = '1') then
      Z(3) <= not (A and B);
      Z(0) <= not (ABAR and BBAR);
      Z(2) <= not (A and BBAR);
      Z(1) <= not (ABAR and B);
    else
      Z <= "1111";
    end if;
  end process;
end;
```

Figure 2-13 Example VHDL Behavioral Model

Chapter two of *A VHDL Primer* by Jayaram Bhasker [Bha92] has a more in depth coverage of this and other types of VHDL modeling.

### 2.4.2 Structural Modeling

Structural modeling is at a lower level than behavioral modeling. It describes the model as a set of interconnected components, much like a schematic. Figure 2-10 uses inverters and NAND gates that are interconnected. In the structural model that implements the decoder, inverters and NAND gates are interconnected. The inverters and gates are the lowest level devices, each have a behavioral model format. Shown below is the structural portion of the VHDL code that implements the decoder example from section 2.4.1.
architecture DEC_STR of DECODER2x4 is 
    component INV
        port (A: in BIT; Z: out BIT);
    end component;
    component NAND3
        port (A,B,C: in BIT; Z: out BIT);
    end component;
signal ABAR, BBAR: BIT;
begin
    I0 : INV port map (A,ABAR);
    I1 : INV port map (B, BBAR);
    N0 : NAND3 port map (ABAR, BBAR, ENABLE, Z(0));
    N1 : NAND3 port map (ABAR, B, ENABLE, Z(1));
    N2 : NAND3 port map (A, BBAR, ENABLE, Z(2));
    N3 : NAND3 port map (A, B, ENABLE, Z(3));
end dec_str

Figure 2-14 Example VHDL Structural Model
3. Implementation

3.1 C program behavioral model

The C program, dec.c was written to be a behavioral model of the JPEG decoder. Its main purpose was to be able to understand how a JPEG decoder worked. It was not meant to be an exact match to the VHDL structural design. It does not have the same input, output or modules that the VHDL implementation has. Since the C program was only used for an understanding of how JPEG works, and not as a part of the implementation of the design, it will not be discussed in detail. Refer to appendix A for the source code to the dec.c program.

3.2 Limitations of Structural Model

This section contains a list of functions that are not available in the decoder. For each omission, the JPEG baseline standard requirement is stated.

1. The VHDL decoder will only work on grayscale images. This was done to limit the size of the design to a level that could be implemented in a reasonable time period. This limitation also reduces the simulation times to a manageable level.

2. The minimum coded unit (MCU) size was limited to one 8x8 block. Most grayscale images are compressed in this manner; therefore, it was decided not to add in the extra capability for more than one 8x8 block in a MCU for the few cases that do not compress this way.

3. Restart markers were not included in the design of the decoder. Restart markers reset the decoding of a scan to an initial state at set intervals. This is helpful if errors are introduced into the compressed image data. With restarts, only the
section in a given interval will be affected by the error. Without restarts, the entire image after the error will be affected.

3.3 VHDL Structural model

This section describes how the VHDL behavioral model of the JPEG decoder was designed and implemented. The structural model will be described in the same manner it was designed, using a top down approach. A top down design starts with high level abstract blocks and works down to specific detailed small blocks. An example would be an upper level abstract block of a car, with some lower levels being brakes, tires and engine. These lower levels may be abstract blocks themselves that can be broken down into smaller blocks.

3.3.1 VHDL Test Bench

As seen in figure 1-2, the upper most level of the VHDL design is the test bench. This top level block is expanded to the next lower level and is shown in figure 3-1.

![Diagram of VHDL Test Bench](image)

Figure 3-1 VHDL Test Bench
The three parts that make up the VHDL test bench are the decoder, memory and display blocks. These blocks are defined in table 3-1. The interconnecting signals between the three blocks are defined in table 3-2. The exact behavior of these signals is shown in more detail in the decoder section, 3.3.2.

<table>
<thead>
<tr>
<th>Memory</th>
<th>A VHDL behavioral model of the memory that stores the compressed image data. This module reads the compressed image data from a file.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decoder</td>
<td>The JPEG decoder, described in detail in section 3.3.2</td>
</tr>
<tr>
<td>Display</td>
<td>A VHDL behavioral model of a display module. This module accepts the decompressed image and writes the data to a file.</td>
</tr>
</tbody>
</table>

Table 3-1 VHDL Test Bench Modules

<table>
<thead>
<tr>
<th>clock</th>
<th>system clock</th>
</tr>
</thead>
<tbody>
<tr>
<td>reset</td>
<td>Resets the decoder to an initial state. Initial state is awaiting signal to begin decoding. Reset has precedence over all other signals.</td>
</tr>
<tr>
<td>start_decode</td>
<td>Signals the decoder to start decoding a compressed image. The decoder looks for a rising edge on this signal.</td>
</tr>
<tr>
<td>decoding</td>
<td>When on, signifies that the decoder is decompressing an image. When off, decoder is awaiting a start decode signal.</td>
</tr>
<tr>
<td>complete</td>
<td>Signals that the decoder has completed decompressing data.</td>
</tr>
<tr>
<td>complete_ack</td>
<td>Signals the decoder that the complete signal was seen.</td>
</tr>
<tr>
<td>intr</td>
<td>Signals that an error occurred while the decoder was decompressing data.</td>
</tr>
<tr>
<td>isr_data</td>
<td>Interrupt data from which the cause of an error can be determined.</td>
</tr>
<tr>
<td>cmprssed_data</td>
<td>Compressed data that is input into the decoder a byte at a time.</td>
</tr>
<tr>
<td>next_byte_req</td>
<td>A request from the decoder for the next byte of compressed data.</td>
</tr>
<tr>
<td>next_byte_ack</td>
<td>Acknowledgment that the next byte request was signaled, compressed data is valid on cmprssed_data line.</td>
</tr>
<tr>
<td>read_err</td>
<td>Data could not be obtained from memory.</td>
</tr>
<tr>
<td>result_data</td>
<td>Decompressed data, output from the decoder a byte at a time.</td>
</tr>
<tr>
<td>result_data_valid</td>
<td>Decompressed data is valid on the result_data line.</td>
</tr>
<tr>
<td>result_data_ack</td>
<td>Decompressed data has been delivered to display module.</td>
</tr>
</tbody>
</table>

Table 3-2 VHDL Test Bench Interconnecting Signals
To implement the VHDL test bench, a source file that connects the modules together is needed. This file also provides the stimulus to the inputs signals; clock, reset, start_decode, and complete_ack. It also checks the output signals decoding, complete, intr and isr_data. The basic structure of the VHDL test bench is shown in figure 3-2. For the complete test bench, refer to the JPEG source file in appendix A.

```vhdl
entity jpeg is
end jpeg;
architecture jpeg_arch of jpeg is
  -- constant definitions
  -- component declarations
  component memory
    .
  end component;
  component decoder
    .
  end component;
  component display
    .
  end component;
  -- define signals
  signal clock std_logic;
  signal result_data_ack std_logic;
begin
  -- connect components
  mem : memory
    port map(
      signals
    );
  dec : decoder
    port map(
      signals
    );
  dis : display
    port map(
      signals
    );
  process MAIN
  begin
    -- control signals
  end process MAIN;
end jpeg_arch;
```

Figure 3-2 VHDL Test Bench Basic Structure
3.3.2 Control Signals

Throughout the design of the JPEG decoder many control signals are used. This section describes how control signals are used to communicate between two modules. Control between different modules is accomplished using a handshaking scheme. The handshaking methodology is shown in the following example. This example has two modules, A and B, which need to communicate some task between themselves. There are two control lines between the modules, REQ (request) and REQ_ACK (request acknowledged). These two control lines are used in conjunction with a data line. In this example, module A will need to request data from module B. Figure 3-3 shows the schematic layout of the modules.

![Figure 3-3 Control Signals Example Schematic](image)

The REQ line is used by module A to request data from module B. Module B signals that the data are valid on the DATA line by using the REQ_ACK line. The exact syntax for requesting and acknowledging is as follows:

1. Reset state, both REQ and REQ_ACK are set to 0. This is the initial state.

   Module A waits to do work, and module B waits for the REQ signal to change to 1. The DATA line starts in an unknown state.
2. Module A receives an external signal to start work, it then changes REQ line from 0 to 1.

3. Module A waits for REQ_ACK line to change from 0 to 1.

4. Module B receives REQ change from 0 to 1. Does internal work to obtain data for module A.

5. Module B puts data onto DATA line.

6. Module B changes REQ_ACK line from 0 to 1.

7. Module B waits for REQ line to change from 1 to 0.

8. Module A receives REQ_ACK line change from 0 to 1.

9. Module A returns REQ line to 0. (Note, it would be an improvement to take the data from B at this point.)

10. Module A waits for REQ_ACK line to return to 0.

11. Module B receives REQ signal change back to 0. Module B returns to initial state, restoring REQ_ACK signal to 0. Data line is not changed in the initial state.

12. Module A receives REQ_ACK signal change to 0. It will now take data from the DATA line and continue with work. Module B will not change the data line until the next request. Note, in this design only these two modules may talk to each other. Thus the data will remain valid until module A is done with the current data and make a new request.
Note that this is a synchronous design, and all actions occur with an edge of a clock. In all modules used in the design of the JPEG decoder this is the method used with control signals. An example timing diagram is shown below in figure 3-4.

![Example Control Signals Timing](image)

**Figure 3-4 Example Control Signals Timing**

### 3.3.3 Decoder Module

The decoder module is the structural model of the JPEG decoder. It consists of seven major components that control and handle the decompressing of image data. The main unit in the decoder is the controller. The other major units are the NextByte module, ISR (Interrupt Status Register) module, FindSOI (Find Start of Image) module, FindSOF (Find Start of Frame), FrHeader (Frame Header) module, FindEOI (Find End of Image) module, FindSOS (Find Start of Scan) and DecodeScan (Decode Scan) module. Figure 3-5 shows the major components that that make up the decoder. Note, this is not a complete diagram of the decoder, it only shows the major control and data signals. Later sections in this chapter cover the remaining modules and signals.
The decoder module is made up of interconnections between many submodules. The actual source code for the decoder has no functionality, it only connects modules that accomplish the parts of the decoding. Figure 3-6 shows the basic structure of the decoding module. See appendix A for the complete decoder source code.

```vhdl
--
-- Name    decoder_entity
--
-- Purpose : JPEG baseline decoder
-- Author  : Douglas A. Carpenter
-- Created : 12-Mar-1994
-- Revised :

-- library and use clauses
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164_extensions.all;

library my_packages;
use my_packages.package_1.all;

library my_components;
use my_components.isr.all;
-- include all components
use my_components.huff_adcisel6.all;

entity decoder is
port (  -- control inputs/outputs
  clock    : in  std_logic;
  start_decode      in  std_logic;  -- start decoding compressed data in memory
  decoding        : out std_logic;  -- acknowledge start decode signal
  complete       : out std_logic;  -- decoding complete
  complete_ack   : in  std_logic;  -- decoding complete acknowledge
  intr          : out std_logic;  -- interrupt flag
  isr_data      : out std_logic_vector(7 downto 0);  -- interrupt status register
);  
end decoder;
```
reset : in std_ulogic; -- reset flag

-- compressed data in
compressed_data : in std_ulogic_vector(7 downto 0); -- input compressed data
next_byte_req : out std_ulogic; -- request for next byte of compressed data
inc_data_ack : in std_ulogic; -- next byte is valid for read
read_err : in std_ulogic; -- error reading next byte

result_data_valid : out std_ulogic;
result_data : out std_ulogic_vector(7 downto 0);
result_data_ack : in std_ulogic);
end decoder;

-- Name : decoder_arch
-- Purpose : decoder module
-- Author : Douglas A. Carpenter
-- Created : 12-Mar-1994 DAC
-- Revised :

architecture decoder_arch of decoder is

component isr:
  port ( reset : in std_ulogic;
        -- interconnections
clock : in std_ulogic");
end component;

-- define components
--

component doUnload
  port ( clock : in std_ulogic;
        -- interconnections
result_data_ack in std_ulogic);
end component;

signal work_req std_ulogic_vector(2 downto 0);
-- define internal signals
--
signal doUnload_ack : std_ulogic;

begin

ISReg :
  isr
  port map ( reset,
            -- interconnecting signals
clock);

-- Instantiate components
--
--
DoUnload :
  doUnload
  port map ( clock,
            -- interconnecting signals
result_data_ack);

end decoder_arch;

Figure 3-6 Layout of Decoder VHDL Source Code
Controller Module

The controller module is the brains of the decoder. It has the external inputs and outputs to the decoder that signal when to start decoding. The controller determines which sub-module in the decoder will be working on the compressed data. Figure 3-7 shows the symbol for the controller. A feature of designing with Mentor Graphic's Corporation System 1076 VHDL is that symbols can be created from the entity definitions. Source code for the controller is found in Appendix A.

![Controller Symbol](image)

Figure 3-7 Controller Symbol
The signals used by the controller are defined as follows in table 3-4

<table>
<thead>
<tr>
<th>Signal</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock</td>
<td>System clock</td>
</tr>
<tr>
<td>reset</td>
<td>System reset</td>
</tr>
<tr>
<td>start_decode</td>
<td>External signal to start decoding compressed data.</td>
</tr>
<tr>
<td>decoding</td>
<td>External output to signal that decoding is in process.</td>
</tr>
<tr>
<td>complete</td>
<td>External output to signal that decoding is complete.</td>
</tr>
<tr>
<td>complete_ack</td>
<td>External input to signal that complete has been received.</td>
</tr>
<tr>
<td>intr</td>
<td>External output, an error has occurred decoding the compressed data. Check ISR register for error cause.</td>
</tr>
<tr>
<td>work_req</td>
<td>Internal control bus. Different requests are put on the bus to request a module perform its operation. See table 3-4 for work request codes.</td>
</tr>
<tr>
<td>ack</td>
<td>Work requested is complete</td>
</tr>
<tr>
<td>req_err</td>
<td>Work requested can not be completed, an error occurred</td>
</tr>
</tbody>
</table>

Table 3-3 Controller Signals

<table>
<thead>
<tr>
<th>work_req code</th>
<th>definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>No request, also signals that the last ACK was received.</td>
</tr>
<tr>
<td>001</td>
<td>Request that FindSOI module perform its operation</td>
</tr>
<tr>
<td>010</td>
<td>Request that FindSOF module perform its operation</td>
</tr>
<tr>
<td>011</td>
<td>Request that FrHeader module perform its operation</td>
</tr>
<tr>
<td>100</td>
<td>Request that dec_frame module perform its operation</td>
</tr>
<tr>
<td>110</td>
<td>Request that FindEOI module perform its operation</td>
</tr>
<tr>
<td>all others</td>
<td>not used</td>
</tr>
</tbody>
</table>

Table 3-4 Work Request Codes
For the remaining modules, the flow will not be shown. See the appropriate module’s corresponding state table or VHDL source code. The flow of the controller is as follows:

Step 1. Reset state, set all outputs to their initial state. Wait for start_decode signal.

Step 2. Received start_decode signal. Send out work_req code to find start of image (SOI).

Step 3. Wait for acknowledgment from FindSOI module that SOI marker was found. If a req_err is returned, go to error step.

Step 4. Received ACK from FindSOI, send out work_req code to find start of frame (SOF).

Step 5. Wait for acknowledgment from FindSOF module that SOF marker was found. If a req_err is returned, go to error step.

Step 6. Received ACK from FindSOF, send out work request to the frame header module to perform its operation.

Step 7. Wait for acknowledgment from FrHeader module that the frame header data were read in correctly. If a req_err is returned, go to error step.

Step 8. Received ACK from FrHeader module, send out work request code to the DecFrame module.

Step 9. Wait for acknowledgment from DecFrame module that the frame was decoded properly. If a req_err is returned, go to error step.

Step 10. Received ACK from DecFrame module, send out work request code to the module to find the end of image (EOI) marker.
Step 11. Wait for acknowledgment from FindEOI module that the EOI marker was found. If a req_err is returned, go to error step.

Step 12. Received ACK from FindEOI module that decoding is complete. Send proper sequence of signals that decoding is complete.

Step 13. Return to step 1.

Error Step: Send proper sequence of control signals using the INTR line to alert an interrupt. Once complete return to step 1.

The controller is implemented using a state machine. Most modules in the decoder design are implemented using state machines. Two methods for designing state machines were used. The first method was not an efficient method, and it was used only on a few modules at the start of the design process. The basic process of this first method will be explained for the controller. Since the controller is rather large an in depth discussion will not be shown for the module. The ISR module will cover this method in detail later in the thesis. All other modules that use this method will only be noted. The downfall of this first method is that it is very cumbersome to use and when changes are necessary on a module it is a difficult and time consuming process. Also, understanding the source code is very difficult to do without a state table. The steps in the process to implement the state machine are:

1. Design a state table. The state table for the controller is found in appendix B.

2. Code the state table into an espresso program input.

3. Run the espresso program, obtain the espresso output. The espresso output for the controller is found in appendix B.
4. Use the espresso output to write the VHDL code. Code is written using minterms. Minterms are found as the rows in the espresso output. They are labeled A0 to A30 for the controller. With these minterms, the state variables (D3 to D0) and outputs are derived using sums.

Making improvements using this method results in major changes to all the minterms and the sums used. Usually a complete rewrite of the source code is required. For this reason, this method of implementing state machines was abandoned for the second method. The first method will be referred to as the espresso method throughout the rest of this thesis. The second method is the direct method and it will be covered in the first module used.

NextByte Module

The NextByte module is used to supply one common interface to the compressed data memory. If changes are made to the way the compressed data is accessed, only this module needs to be changed. The NextByte module was designed as a state machine. The method used in the design was the espresso method. A detailed description of the espresso method is shown for the NextByte module following the symbol and signal tables. Figure 3-8 is the interface of the NextByte module. All compressed data are read from memory sequentially.
Figure 3-8 NextByte Symbol

Table 3-5 defines the signals used by the NextByte module.

<table>
<thead>
<tr>
<th>Signal</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock</td>
<td>System clock</td>
</tr>
<tr>
<td>reset</td>
<td>System reset</td>
</tr>
<tr>
<td>next_byte_req</td>
<td>Request to memory for next byte of data.</td>
</tr>
<tr>
<td>inc_data_ack</td>
<td>ACK from memory, data is valid on compressed_data line.</td>
</tr>
<tr>
<td>read_err</td>
<td>Error at memory providing next byte of data</td>
</tr>
<tr>
<td>compressed_data</td>
<td>Compressed data from memory. Input a byte at a time.</td>
</tr>
<tr>
<td>next_req</td>
<td>Internal request for next byte of data.</td>
</tr>
<tr>
<td>next_ack</td>
<td>Internal response to next_req. Will signal that data are valid on data line.</td>
</tr>
<tr>
<td>data</td>
<td>Internal data line. The size is a byte.</td>
</tr>
<tr>
<td>err</td>
<td>Internal error line. Signals that there was an error reading from memory.</td>
</tr>
</tbody>
</table>

Table 3-5 NextByte Signals
The following is a step-by-step description of the espresso method for coding state tables.

Step 1: Create a state table of the functionality needed for the module. Table 3-6 is the state table for the NextByte module.

<table>
<thead>
<tr>
<th>O2</th>
<th>Q1</th>
<th>Q0</th>
<th>next_req</th>
<th>req</th>
<th>err</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
<th>next_ack</th>
<th>err</th>
<th>next_byte_req</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Wait for next request</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Got next request, send out to memory a request</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>wait for inc_ack</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>err on memory req</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>got inc_ack, no err</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>wait for inc_ack to return to 0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>0</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>inc_ack returned to 0, send next_ack</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>wait for next_req to return to 0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>next_req returned to 0, go to 000</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>err state, wait for next_req to return to 0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>err state, next_req returned to 0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Not Used</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Not Used</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Not Used</td>
</tr>
</tbody>
</table>

Table 3-6 NextByte State Table

Step 2. Create an espresso input file from the state table. Figure 3-9 is the espresso input file. The comments in the espresso input file define the inputs and states needed for the NextByte module. The data in the espresso input file are the same as the state table.

```
# NextByte espresso input
.i 6    # number of inputs
.o 6    # number of outputs
.type fr
.phase 111111    # non inverted outputs
#
# The first six columns are inputs
# The second six columns are outputs
```
The columns are defined as follows, from left to right:

- Q2
- Q1
- Q0
- next_req
- inc_data_ack
- read_err
- D2
- D1
- D0
- next_ack
- err
- next_byte_req

- stands for don’t care, same as X in state table.

```
000 0-- 000 000
000 1-- 001 001
001 -00 001 001
001 001 111 010
001 -10 011 000
011 -1- 011 000
011 -0- 010 100
010 1-- 010 100
010 0-- 000 000
111 1-- 111 010
111 0-- 000 000
```

.end

Figure 3-9 NextByte Espresso Input File

Step 3. Run the espresso input file through the Espresso program. It will create an espresso output file. Figure 3-10 shows the espresso output for the NextByte module.

```
# NextByte espresso output
.i 6    # six inputs
.o 6    # six outputs
# .phase 111111
#
# The columns are defined the same as in the input file
#
# The first size columns of each row defines a minterm
#
.p 7
-01--1 111010  # A0
0-1-1- 011000  # A1
-01-00 001001  # A2
1--1-- 111010  # A3
-101-- 010100  # A4
-001-- 001001  # A5
011-0- 010100  # A6
#
# Example:
```

3-17
Step 3: Create the VHDL code from the espresso results. Figure 3-11 Shows the VHDL code that implements the results from espresso.

```vhdl
-- Name: nextbyte_arch
-- Purpose: nextbyte module
-- Author: Douglas A. Carpenter
-- Created: 30-Mar-1994 DAC
-- Revised:

architecture nextbyte_arch of nextbyte is
  signal D : std_ulogic_vector (2 downto 0);
  signal a0 : std_ulogic;
  signal a1 : std_ulogic;
  signal a2 : std_ulogic;
  signal a3 : std_ulogic;
  signal a4 : std_ulogic;
  signal a5 : std_ulogic;
  signal a6 : std_ulogic;
  signal idata : std_ulogic_vector (7 downto 0);
begin
  next_process : process (reset,clock)
  begin
    if (reset = '0') then
      D <= "000";
      a0 <= '0';
      a1 <= '0';
      a2 <= '0';
      a3 <= '0';
      a4 <= '0';
      a5 <= '0';
      a6 <= '0';
      idata <= "00000000";
    elsif ((clock'event) and (clock = '1') and (clock'last_value = '0')) then
      a0 <= ((D(1) = '0') and (D(0) = '1') and (read_err = '1'))
          or ((D(2) = '1') and (next_req = '1'));
      a1 <= ((D(1) = '0') and (D(0) = '1') and (inc_data_ack = '1'))
          or ((D(2) = '0') and (next_req = '1'));
      a2 <= ((D(1) = '0') and (D(0) = '0') and (inc_data_ack = '1'))
          or ((D(2) = '1') and (next_req = '0'));
      a3 <= ((D(1) = '1') and (D(0) = '0') and (next_req = '1'))
          or ((D(2) = '0') and (next_req = '1'));
      a4 <= ((D(1) = '1') and (D(0) = '0') and (next_req = '1'))
          or ((D(2) = '1') and (next_req = '0'));
      a5 <= ((D(1) = '0') and (D(0) = '0') and (next_req = '1'))
          or ((D(2) = '1') and (next_req = '1'));
      a6 <= ((D(1) = '1') and (D(0) = '0') and (next_req = '1'))
          or ((D(2) = '0') and (next_req = '1'));

      if (inc_data_ack = '1') then
        idata <= compressed_data;
      end if;
    end if;
  end process next_process;

-- assign output values
  next_ack <= '1' when ((a4 = '1') or (a6 = '1')) else '0';
  err <= '1' when ((a0 = '1') or (a3 = '1')) else '0';
  next_byte_req <= '1' when ((a2 = '1') or (a5 = '1')) else '0';
end
```
The VHDL source code implements the results from espresso directly. Minterms are modified in the main process; the sum of these minterms are used for each outputs.

**ISR Module**

The ISR Module is the interrupt status register. When an error occurs in the decoding process, the intr signal line will be changed from 0 to 1. Along with this line, the reason for the interrupt will be placed into the ISR. The ISR is an eight bit register that holds codes for various error conditions. Shown below are the symbol for the ISR module (figure 3-12), a table of signal definitions (table 3-7) and a table of the current in use error codes (table 3-8). Only a few codes have been used as of the current implementation. All extra codes are for future enhancements.
<table>
<thead>
<tr>
<th>Signal</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock</td>
<td>System clock</td>
</tr>
<tr>
<td>reset</td>
<td>System reset</td>
</tr>
<tr>
<td>isr_latch</td>
<td>Input from modules to latch isr_data_in.</td>
</tr>
<tr>
<td>isr_latch_ack</td>
<td>Output to modules that data have been latched.</td>
</tr>
<tr>
<td>isr_data_in</td>
<td>Interrupt code data from internal modules</td>
</tr>
<tr>
<td>isr_data</td>
<td>External output of interrupt code.</td>
</tr>
</tbody>
</table>

Table 3-7 ISR Signal Definitions

<table>
<thead>
<tr>
<th>Code</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000001</td>
<td>Error finding SOI marker</td>
</tr>
<tr>
<td>00000011</td>
<td>Error loading quantization table</td>
</tr>
<tr>
<td>00000100</td>
<td>Error loading Huffman table</td>
</tr>
<tr>
<td>00000101</td>
<td>Error finding SOF marker</td>
</tr>
<tr>
<td>00000111</td>
<td>Error finding EOI marker</td>
</tr>
</tbody>
</table>

Table 3-8 ISR Codes

The ISR was implemented as a state machine using the espresso method. Appendix B contains the state table and the espresso results.

**FindSOI Module**

The purpose of the FindSOI module is to read in compressed data until the SOI marker is found. Once the SOI marker is found an ACK is returned to the controller. If all of the compressed data are searched and no SOI marker is found or there is a read error from memory, then the REQ_ERR is returned to the controller. Figure 3-13 shows the FindSOI interface. Following that, table 3-9 defines the signals used by the module. The FindSOI module is a state machine and was designed using the espresso method. See appendix B for the state table and espresso output.
Some of the signals used in the design of the decoder are shared by multiple modules. One such example is the ACK signal. All of the modules that receive work requests use the ACK signal. Multiple modules can drive a common signal since only one module is working at any one time. To code such a line, a wired or resolution function is
A resolution function is used in VHDL when there are multiple inputs to the same signal. As the name implies, the function will resolve what is the correct output. A wired or function takes all of the modules signals for the one line and puts them through an OR gate. The output of the OR gate is the resolved function. Thus when all of the lines are off, the signal will be off. If any of the modules are driving on, the signal will be on. In this design, only one module is working at any on time, thus only one module will drive the ACK line. Figure 3-14 shows an example of the wired or function as a schematic diagram.

![Figure 3-14 Example Wired Or Schematic](image)

**FindSOF Module**

FindSOF reads compressed data until the SOF marker is found. Once the marker is found an ACK is returned to the controller. As the module is searching for the SOF marker, if any DHT or DQT markers are found then the corresponding table load modules are called. A DHT marker will result in the loading of a Huffman table. The DQT marker causes a quantization table to be read. The modules that accomplish the loading of these tables are covered later in this section. Figure 3-15 shows the interface of the FindSOF marker.
The definitions of the signals used to interface to the FindSOF module are described in table 3-10.

<table>
<thead>
<tr>
<th>Signal</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock</td>
<td>System clock</td>
</tr>
<tr>
<td>reset</td>
<td>System reset</td>
</tr>
<tr>
<td>work_req</td>
<td>Work request line from controller. FindSOF waits until work_req line has data that matches the SOF req value.</td>
</tr>
<tr>
<td>ack</td>
<td>Acknowledgment back to controller that work is complete, SOF was found successfully.</td>
</tr>
<tr>
<td>req_err</td>
<td>An error occurred when attempting to locate the SOF marker. See ISR register for the cause.</td>
</tr>
<tr>
<td>next_req</td>
<td>Request from NextByte for a byte of compressed data.</td>
</tr>
<tr>
<td>next_ack</td>
<td>Acknowledgment from NextByte that byte on data line is valid.</td>
</tr>
<tr>
<td>err</td>
<td>Error obtaining a byte from memory</td>
</tr>
<tr>
<td>data</td>
<td>Compressed data, size is byte.</td>
</tr>
<tr>
<td>isr_latch</td>
<td>Message to ISR to latch in data.</td>
</tr>
<tr>
<td>isr_latch_ack</td>
<td>Return message from ISR that the data was latched in.</td>
</tr>
<tr>
<td>isr_data_in</td>
<td>Data to be latched into ISR register.</td>
</tr>
<tr>
<td>qtable_req</td>
<td>Request to load a quantization table.</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------------------</td>
</tr>
<tr>
<td>qtable_ack</td>
<td>Acknowledgment from quantization table loader module that table was loaded successfully.</td>
</tr>
<tr>
<td>htable_req</td>
<td>Request to load a Huffman table.</td>
</tr>
<tr>
<td>htable_ack</td>
<td>Acknowledgment from Huffman table loader module that table was loaded successfully.</td>
</tr>
<tr>
<td>table_err</td>
<td>Error loading a table. Both quantization and Huffman loaders used the same signal. Since only one table is loading at a time, they can share the line.</td>
</tr>
</tbody>
</table>

Table 3-10 FindSOF Signal Definitions

**Fr_Header Module**

The Fr_Header module was designed to read in the data that are located after the SOF marker. This header contains important data related to the image being decompressed. Figure 3-16 shows the makeup of the frame header.

![Frame Header](image)

**Figure 3-16 Frame Header Syntax**

For a grayscale image, most of the parameters are will only be one value. Table 3-11 lists what each parameter is, and the possible or preset values. For a detailed description of the complete frame header refer to section B.2.2 in the draft international standard.
<table>
<thead>
<tr>
<th>SOF</th>
<th>Start of Frame marker, read in by FindSOF module.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lf</td>
<td>Length of the frame header, excludes the SOF marker. Lf is the number of bytes including the two byte length parameter.</td>
</tr>
<tr>
<td>P</td>
<td>Sample precision, for baseline set to 8 bits.</td>
</tr>
<tr>
<td>Y</td>
<td>Number of lines.</td>
</tr>
<tr>
<td>X</td>
<td>Number of sample per line.</td>
</tr>
<tr>
<td>Nf</td>
<td>Number of image components in a frame. Set to one for grayscale images.</td>
</tr>
<tr>
<td>C</td>
<td>Component identifier. Used for multiple component frames. Set to 1 for grayscale images.</td>
</tr>
<tr>
<td>H</td>
<td>Horizontal sampling factor. Set to one for this thesis.</td>
</tr>
<tr>
<td>V</td>
<td>Vertical sampling factor. Set to one.</td>
</tr>
<tr>
<td>Tq</td>
<td>Quantization table selector for the component. Since only one component for a grayscale image, the Tq parameter is set to one.</td>
</tr>
</tbody>
</table>

Table 3-11 Frame Header Descriptors

Figure 3-17 shows interface of the FrHeader module, and following that table 3-12 describes the signals used.

![Figure 3-17 Fr_Header Symbol](image-url)
<table>
<thead>
<tr>
<th>Signal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock</td>
<td>System clock</td>
</tr>
<tr>
<td>reset</td>
<td>System reset</td>
</tr>
<tr>
<td>work_req</td>
<td>Work request from the controller. Frame header module waits for the correct code.</td>
</tr>
<tr>
<td>ack</td>
<td>Acknowledgment to controller that frame header was read in correctly.</td>
</tr>
<tr>
<td>req_err</td>
<td>Signal controller that an error occurred reading in frame header data.</td>
</tr>
<tr>
<td>next_req</td>
<td>Request to NextByte module for next byte of compressed data.</td>
</tr>
<tr>
<td>next_ack</td>
<td>Return signal from NextByte module that byte on data line is valid.</td>
</tr>
<tr>
<td>err</td>
<td>NextByte module signals that an error occurred reading from memory.</td>
</tr>
<tr>
<td>data</td>
<td>Compressed data from memory.</td>
</tr>
<tr>
<td>isr_latch</td>
<td>Message to ISR to latch in data.</td>
</tr>
<tr>
<td>isr_latch_ack</td>
<td>Return message from ISR that data were latched in.</td>
</tr>
<tr>
<td>isr_data_in</td>
<td>Data to be latched into ISR register.</td>
</tr>
<tr>
<td>x_latch</td>
<td>Signal to X register to latch in data.</td>
</tr>
<tr>
<td>y_latch</td>
<td>Signal to Y register to latch in data.</td>
</tr>
<tr>
<td>xy_data</td>
<td>X or Y dimension data, fed into both registers. X or Y latch signal determines which register latches data.</td>
</tr>
</tbody>
</table>

Table 3-12 Fr_Header Signal Definitions

The frame header module was designed using the espresso method. Refer to appendix B for the state table and espresso results.

**FindEOI Module**

The FindEOI module has one purpose, and that is to find the EOI marker. It requests compressed data until the EOI marker is found. When found, it returns an acknowledgment to the controller. If an error occurs when searching for the marker, the controller is notified.
The FindEOI module was designed using the espresso method. Refer to appendix B for the state table and espresso results.

**dec_frame**

The decode frame module, dec_frame, is a mini controller. It receives a request from the main controller to decode a frame. When this request is received, in the form of a code on the work_req line, the module sends requests to two sub-modules. The module first requests that the start of scan marker be found. Once the start of scan marker is found, the module requests that the scan be decoded. Once both these requests are performed, in order, the main controller signals that the frame was decoded successfully. Since there is only one scan in a frame in a grayscale image, no extra code is needed for multiple scans. Figure 3-19 shows the interface used by the dec_module. Table 3-13 defines the signals used by the module.
Figure 3-19 dec_frame Symbol

| clock       | System clock |
| reset       | System reset |
| work_req    | Work request from the controller. Frame header module waits for the correct code. |
| ack         | Acknowledgment to controller that frame header was read incorrectly. |
| req_err     | Signal controller that an error occurred reading in frame header data. |
| find_sos    | Request that the find_sos module find the SOS marker. |
| find_sos_ack| Return signal that the find_sos module found the SOS marker successfully. |
| find_sos_err| An error occurred attempting to find the SOS marker. |
| decode_scan | Request that the scan be decoded. |
| decode_scan_ack | Return signal that the scan was successfully decoded. |
| decode_scan_err | An error occurred decoding the scan. |

Table 3-13 dec_frame interface definitions

Find_SOS

The FindSOS module has the same function as the FindSOF module, except that it finds the SOS marker. If any DHT or DQT markers are found, their respective modules are called to load the tables. Figure 3-20 shows the interface used by the FindSOS module. Table 3-14 has the definitions for the signals used by the module.
Table 3-14 FindSOS Signal Definitions

<table>
<thead>
<tr>
<th>Signal</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock</td>
<td>System clock</td>
</tr>
<tr>
<td>reset</td>
<td>System reset</td>
</tr>
<tr>
<td>find_sos_req</td>
<td>Request that this module find the SOS marker.</td>
</tr>
<tr>
<td>find_sos_ack</td>
<td>Acknowledgement to decode_frame module that SOS marker was found.</td>
</tr>
<tr>
<td>find_sos_err</td>
<td>Error signal to decode_frame module that SOS marker was not found in compressed data.</td>
</tr>
<tr>
<td>next_req</td>
<td>Request from NextByte for a byte of compressed data.</td>
</tr>
<tr>
<td>next_ack</td>
<td>Acknowledgement from NextByte that byte on data line is valid.</td>
</tr>
<tr>
<td>err</td>
<td>Error obtaining a byte from memory.</td>
</tr>
<tr>
<td>data</td>
<td>Compressed data, size is byte.</td>
</tr>
<tr>
<td>isrLatch</td>
<td>Message to ISR to latch in data.</td>
</tr>
<tr>
<td>isrLatch_ack</td>
<td>Return message from ISR that data were latched in</td>
</tr>
<tr>
<td>isr_data_in</td>
<td>Data to be latched into ISR register.</td>
</tr>
<tr>
<td>qtable_req</td>
<td>Request to load a quantization table.</td>
</tr>
<tr>
<td>qtable_ack</td>
<td>Acknowledgement from quantization table loader module that table was loaded successfully.</td>
</tr>
<tr>
<td>htable_req</td>
<td>Request to load a Huffman table.</td>
</tr>
<tr>
<td>htable_ack</td>
<td>Acknowledgement from Huffman table loader module that table was loaded successfully.</td>
</tr>
<tr>
<td>table_err</td>
<td>Error loading a table. Both quantization and Huffman loaders used the same signal. Since only one table is loading at a time, they can share the line.</td>
</tr>
</tbody>
</table>

Figure 3-20 Find_SOS Symbol
The FindSOS module uses the espresso method for implementing a state machine. Appendix B contains the state table and espresso results.

**Decode_Scan**

The Decode_Scan module controls the actual decoding of the compressed data for a scan. It has three functions that it performs sequentially. Once these three have been completed, the decode_scan module signals that it is complete. The three sequential functions are:

1. Request and wait for the scan header to be read in.
2. Request and wait for the number of MCU to be computed.
3. Request and wait for the scan be decoded. This will be done by a pipeline of modules.

![Decode_Scan Symbol](Figure 3-21 Decode_Scan Symbol)
<table>
<thead>
<tr>
<th>decode_type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock</td>
<td>System clock</td>
</tr>
<tr>
<td>reset</td>
<td>System reset</td>
</tr>
<tr>
<td>decode_scan</td>
<td>Signal for this module to start.</td>
</tr>
<tr>
<td>decode_scan_ack</td>
<td>Return signal to dec_frame module that the scan has been decoded successfully.</td>
</tr>
<tr>
<td>decode_scan_err</td>
<td>Return signal to the dec_frame module that an error occurred decoding the scan.</td>
</tr>
<tr>
<td>decode_scan_header</td>
<td>Signal to the scan header module to read in the header</td>
</tr>
<tr>
<td>decode_scan_header_ack</td>
<td>Return signal from the scan header module that it was successful.</td>
</tr>
<tr>
<td>decode_scan_header_err</td>
<td>Return signal from the scan header module that an error occurred.</td>
</tr>
<tr>
<td>computenummcus</td>
<td>Signal to compute number of MCUs module to perform its operation.</td>
</tr>
<tr>
<td>computenummcusack</td>
<td>Return signal that the number of MCUs was successfully computed.</td>
</tr>
<tr>
<td>start_scan</td>
<td>Signal to the scan pipeline to start.</td>
</tr>
<tr>
<td>scan_complete</td>
<td>Return signal from the scan pipeline that it is complete.</td>
</tr>
</tbody>
</table>

Table 3-15 Decode_Scan Signal Definitions

The decode scan module does not use the espresso method for implementing a state machine; instead it uses a direct approach. No outside tables or programs are needed to implement the state machine. The state machine is implemented directly into the source code. Refer to the source code in appendix A for this module to see an example of this method. This method, the direct method, is much like a high level language program. It is very readable, and making changes is very simple. When design changes are needed on a state, only the code associated with that state is affected. All modules here after use this method. Two notes on coding this way are:

1. Code all states, even unused ones.
2. Use caution when using **else** clauses. If they are to be used, ensure that no unknown states can cause the **else** statement to be executed. An example of this is shown in figure 3-22.

```vhdl
-- Improper method of coding
-- if decode_scan_ack is any value but 0, the next
-- state will be reached. Thus if decode_scan_ack = '0'
-- the next state will be reached.
--
-- if (decode_scan_header_ack = '0') then
  state = same;
else
  state = next;
end if;

-- Proper method of coding
-- Only when next_state_ack is equal to 1 will the
-- next state be reached
--
-- if (decode_scan_header_ack = '1') then
  state = next;
else
  state = same;
end if;
```

Figure 3-22 Differing uses of else clause

An example state machine coded in VHDL is shown in figure 3-23. The module has four states, uses a synchronous clock, and has an asynchronous reset. Actions occur on the rising edge of the clock, and resets can occur at any time. In the clock section each possible state is covered, including any unknown states. Only one state will be executed each time the process is entered for a clock cycle. The state machine is designed as follows:

1. State 0 is the reset state, the state machine will remain in this state until input1 is equal to 1. When this occurs it will go to state 1.

2. State 1 waits until input1 returns to 0. When this occurs, the state changes to state 2.
3. State 2 watches the signal input2. When input2 changes to 1, the next state will be state 3.

4. State 3 waits until input2 returns to 0. When input2 returns to 0, the state returns to 0, the reset state.

During any of the states the output values may be changed as needed. Notice how the code is a direct implementation of the four steps noted above. That is the reason for the name direct approach for implementing state machines.

```vhdl
example_process : process (reset, clock)
begin
  if (reset = '0') then
    Q <= "00";
    -- set outputs to initial states
  elsif ((clock = '1') and (clock'last_value = '0') and (clock'event)) then
    if (D = "00") then
      if (input1 = '1') then
        -- change state to 1
        Q <= "01";
      else
        -- remain at state 0
        Q <= "00";
      end if;
    elseif (D = "01") then
      if (input1 = '0') then
        -- change state to 2
        Q <= "10";
      else
        -- remain at state 1
        Q <= "01";
      end if;
    elseif (D = "10") then
      if (input2 = '1') then
        -- change state to 3
        Q <= "11";
      else
        -- remain at state 2
        Q <= "10";
      end if;
    elseif (D = "11") then
      if (input2 = '0') then
        -- change state to 0
        Q <= "00";
      else
        -- remain at state 3
        Q <= "11";
      end if;
    else
      Q <= "00" -- unknown states
    end if;
  else
    Q <= "00" -- unknown states
  end if;
end process ; example_process
```

Figure 3-23 Example VHDL direct coded state machine
dec_scan_header

The purpose of the dec_scan_header module is to read in and check the scan header data. The header data must be checked so that they conform to the restrictions of this design. Refer to section B.2.3 of the DIS for the syntax of the header. The source code for this module contains the preset values that must be present in the header.

![Figure 3-24 dec_scan_header symbol](image)

**Compute_MCUs**

The Compute_MCUs module computes the number of 8x8 blocks that are to be decoded in a scan. For this thesis a minimum coded unit (MCU) is one 8x8 block. The module takes in the x and y dimensions of the image, multiplies the two values together, and then divides that result by eight. The result is rounded upwards to the nearest whole number. When an image is compressed, additional pixels are added to either the x or y dimension to make them multiples of eight. These additional pixels can be discarded when the image is decompressed. Figure 3-25 shows the symbol for the Compute_MCUs module. Refer to the VHDL source code for a detailed description on how the computation is done.
3.3.4 Decoding a Scan

The actual decoding of a scan is accomplished by a set of modules. Several modules were used to break up the decoding process into smaller blocks. With smaller blocks a pipeline can be created. With a pipeline, data can be decompressed at a faster rate than a system without a pipeline. The pipeline is created by breaking up the decoding of an 8x8 block into four parts. The four parts are: loading an 8x8 block from memory, dequantization, inverse discrete cosine transform and unloading data to display memory. The loading module involves Huffman decoding of the compressed data into 8x8 blocks of data. The speed of the pipeline will be limited to the slowest module in the pipeline. There will be a latency in the pipeline that is due to waiting for the first block to be decoded. This pipelined implementation uses a single buffering scheme. Single buffering occurs when one stage must wait until the other is done with the buffer before it can be used. The advantage of a pipeline can be shown by a simple example. In this example a 64x64 pixel image is to be decoded. The 64x64 pixel image has 64 blocks of size 8x8 to be decoded. Figure 3-26 shows the two methods which are to be used to decode this image. Each
block has a time associated with it that represents how long it takes to process data. The total time it takes to process a given number of blocks is defined by:

\[ T_{\text{total}} = (N+(P-1)) \times T_{\text{slowest stage}} \] [Cok91]

where \( N \) number of 8x8 pixel blocks in the image

\( P \) : number of stages in a pipeline

and \( T_{\text{total}} \) excludes the time to load the pipeline.

For the non-pipelined stage the number of stages (\( p \)) is one. The pipelined method has a \( P \) of four. The total times are:

- \( T_{\text{total}} = 2048\text{ns} \) non-pipelined method, \( T_{\text{slowest}} = 32\text{ns} \)
- \( T_{\text{total}} = 670\text{ns} \) pipelined method, \( T_{\text{slowest}} = 10\text{ns} \)

With larger numbers of blocks, the pipelined method will have even faster times versus the non-pipelined method. For a more detailed analysis refer to chapter four of Parallel Programs for the Transputer by Ronald S. Cok [Cok91]. Improvements that can be made to the pipeline will be discussed in the conclusions section of this thesis.

![Diagram](image)

Figure 3-26 Example Pipelined and Non-pipelined Systems
The modules used to implement the decoding of a scan process are shown in figure 3-27. Each module is explained in greater detail following this figure.

Figure 3-27 Scan Decoding Modules

**load_coeff**

The load_coeff module is that part of the pipeline that decodes the scan. It controls the loading of data from the compressed data stream into 8x8 blocks. The load_coeff module reads in the number of MCUs to be decoded. Once this number of MCUs has been decoded, a complete signal is sent to the dequantize module. Control of the memory buffer, related to this module, is done by the use of two handshaking signals, a_empty_right and a_full_right. Using these two signals data is only loaded when the dequantization module is ready. Thus data are not overwritten at any time. See the source code for the detailed syntax of this handshaking. Loading is done by Huffman decoding. The Huffman decoding is accomplished using two steps. First the DC coefficient is decoded; then the remaining 63 AC coefficients are done. The module do_load_coeff controls the Huffman decoding process for each 8x8 block of data. Figure 3-28 shows the symbol of the load_coeff module.
The dequant_coeff module controls the dequantizing of 8x8 blocks. The actual dequantizing is done by the dequantize module. Access to the input and output buffers is done by the dequantize_coeff module. Figure 3-29 is the symbol for the interface used by this module. The simplified steps of this module are:

1. Check for a full signal on the input buffer. When full, go to unload step.
2. Unload input buffer and signal that buffer is unloaded.
3. Dequantize data, done by calling the dequantize module.
4. Check if output buffer is empty. When empty, load data to the output buffer.
5. Go to step 1.

See the source code for a detailed description of this process.
idct_coeff

The idct_coeff module controls the dequantizing of 8x8 blocks. The actual transform is done by the idct module. Access to the input and output buffers is done by the idct_coeff module. Figure 3-30 shows the symbol for the idct_coeff module. The simplified steps of this module are:

1. Check for a full signal on the input buffer. When full go to unload step.
2. Unload input buffer and signal that buffer is unloaded.
3. Do the inverse discrete cosine transform on the data, calling the idct module.
4. Check if output buffer is empty. When empty, load data to the output buffer.
5. Go to step 1.

See the source code for a detailed description of this process.
unload_coeff

The purpose of this module is to control the unloading of image data to the display memory. This module controls the access to the input buffer, and calls the do_unload module to do the actual unloading. When the idct_coeff_complete signal is received and the final buffer is unloaded, the scan_complete signal is sent to the decode_scan module. Refer to the source code for a detailed description of this module. Figure 3-31 shows the interface of this module in symbol format.

Figure 3-31 unload_coeff symbol
mem64x8 and mem64x16

The mem64x8 module is a 64 element register that has 8 bits for each element. The mem64x16 module is the same except it has 16 bits per element. Figure 3-32 shows the mem64x8 symbol. Change the size of the data lines to 16 bits for the mem64x8. These memory modules are used in between stages in the pipeline. Most data are 8 bits, except at the output of the dequantize module, which has 16 bit data.

![Figure 3-32 mem64x8 Symbol](image)

The following is a list of the remaining components that make up the pipeline. A short description is given for each module. For the actual interface used, refer to the VHDL source code in appendix A.

**do_load_coeff**

The do_decode_coeff module controls the do_load_DC and do_load_AC modules. It has two basic steps. The first is to signal do_load_DC to operate, and when that step is complete the do_load_AC module is signaled. When both steps have been completed in order, a complete ack is returned to the load_coeff module.
do_load_DC and do_load_AC

These two modules control the actual Huffman decoding of the compressed image data. The do_load_DC controls the Huffman decoding of the DC coefficient in an 8x8 block. Do_load_AC controls the Huffman decoding of the remaining 63 AC coefficients in the 8x8 block. These two routines use the following modules to accomplish their tasks: decode, receive, extend and nextbit. The functionality of these routines can be found in the DIS section F.2.2. There are also modules used by these routines to access the Huffman tables. These are mentioned in section 3.3.5 of this thesis.

dequantize

The dequantize module takes the data that were Huffman decoded and dequantizes it using the quantization table.

idct

The idct module does the actual inverse transform on the dequantized data. This module is a behavioral model of the inverse discrete cosine transform. Coding this for synthesis would require a rather complex design considering the use of cosines and floating point numbers. The transform was split into the separable one-dimensional transform. Once the transform is complete, the data are sent to the unload buffer.

do_unload

The do_unload module is responsible for unloading the decompressed image data to the display module. It is a very simple module that first reads in the input buffer; and then sends the data out to the display module.
3.3.5 Miscellaneous Modules

The following is an abbreviated list of the remaining modules; only the purpose is stated for each module. No symbols or interconnections are given.

Quantization table modules

The following modules are used for the loading and storage of the quantization table:

- define_qtable: This module controls the loading of the quantization table data.
- qaddr: The address that accesses the memory that stores the quantization data is controlled by this module. It is used only for writing data. This was a very inefficient method of accessing memory, and was used only once.
- qmem: Qmem is the actual memory where the quantization data are stored. It is a very poor implementation of a memory module. Refer to any of the Huffman table register banks for a better example of implementing memory. A feature for speeding up simulations is included in this module. A predetermined table is loaded upon reset. Thus the loading of the quantization table can be eliminated. This is useful when testing the pipelined stages.

Huffman decoder loader modules

The following routines are used for the loading, storage and access to the Huffman tables. There are two Huffman tables, one for the DC data, and the other for the AC data. The routines are:

1. define_htable: Controls the following routines that read in and load Huffman tables.
Routines controlled by define_htable are huff_controller, huff_read_in,
huff_load, huff_bits_addr, huff_bits, huff_vals_addr, huff_vals, huff_lencntr,
huff_acdcsel, huff_acdc_reg, huff_len_loader, huff_bits_loader,
huff_vals_loader, huff_load_size, huff_load_code, huff_reg256by8,
huff_reg256by16, and huff_gen.

2. huff_reg256by8, huff_reg256by16, huff2_256by8, huff_reg16by16 and
huff_reg16by8: These modules are used to store the Huffman tables. There are
two modules of each type; one is for the DC table and the other is for AC table
data. These registers contain a special feature for speeding up simulations. Upon
preset they are preloaded with a standard set of tables. Thus, if a test image uses
the same set of tables, it can be decoded without the need for including the table
data. This is useful when testing the pipeline stages. The time to load the tables
is eliminated.

3. huff_acdcsel, huffacdcSEL16 and huffacdcSEL6: These modules are used to select
which table, DC or AC, provide input to the Huffman decoder modules decode
and receive.

Miscellaneous

reg16 and reg32: These are general purpose registers of size 16 and 32 bits
respectively.
4. Results

4.1 Timing Diagrams

Figure 4-1 shows a timing diagram from a simulation of the VHDL decoder. An expanded view of figure 4-1 is shown in figure 4-2. A 16x16 pixel image was decoded using the structural VHDL model. Using a clock with a 10 nanosecond period, the decoding of the compressed data took 450,898 ns. The compressed data contained one quantization table, a DC huffman table, a AC huffman table and the compressed image data. The operation of the pipeline can be seen by examining the do_load, do_dequant, do_idct and do_unloading signals. By examining the do_idct transform signal, it can be seen that this module has the longest execution time, and thus controls the speed of the pipeline.
Figure 4.1 Decoding Timing Diagram
Figure 4-2 Decoding Timing Diagram, Expanded
Figure 4.2 Timing Diagram, Expanded - Continued

<table>
<thead>
<tr>
<th>Event</th>
<th>4.1339.2</th>
<th>417385.8</th>
<th>420432.4</th>
<th>423479.0</th>
<th>426525.6</th>
<th>429572.2</th>
<th>432618.8</th>
<th>435665.5</th>
<th>438712.1</th>
<th>441758.7</th>
</tr>
</thead>
<tbody>
<tr>
<td>start_decode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decoding</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>complete</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>complete_ack</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>encoder/work_req</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decoder/qt_table_req</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decoder/qt_table_ack</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decoder/qt_table</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decoder/startscan</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_coeff_complete</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>adtodeq_a_full</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>adtodeq_a_empty</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decoder/do_load</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decoder/do_load_ack</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_coeff_complete</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>eqtoict_a_full</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>qtoict_a_empty</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decoder/do_dequant</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decoder/do_dequant_ack</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_coeff_complete</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tounload_a_full</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tounload_a_empty</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decoder/do_idct</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>idctUnloadAck</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decoder/do_idct</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decoder/do_unloading</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decoder/do_unload_ack</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_coeff_complete</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>er/scancomplete</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


4.2 Decoded Images

A 16x16 pixel image was used to test the decoder implemented in VHDL. A small image was chosen to limit the time needed to run the simulations. The 16x16 size was chosen because it contained more than one 8x8 block, and had data in more than one row or column. The original image is shown in figure 4-3. The actual pixel data values are shown in figure 4-4. For all images shown in this chapter, 0 is for black, and 255 is for white.

![Figure 4-3 Test Image - Original](image)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

![Figure 4-4 Test Image - Original, Values](image)
The original image was then compressed using the Independent JPEG Group compression software. The compressed data was then decompressed using four different methods. The methods used are: the Independent JPEG Group decompression routine, the XV software, the dec.c program and the VHDL implementation. The Independent JPEG Group was then used as a standard. The three other methods were compared to this standard, and difference images were created showing how accurate the decompressing methods are. Figure 4-5 through figure 4-15 show the decompressed images, the actual data and the difference data for all methods.
Figure 4-7 XV Result, Image

<table>
<thead>
<tr>
<th></th>
<th>97</th>
<th>106</th>
<th>114</th>
<th>114</th>
<th>106</th>
<th>100</th>
<th>100</th>
<th>103</th>
<th>104</th>
<th>117</th>
<th>106</th>
<th>90</th>
<th>112</th>
<th>146</th>
<th>194</th>
<th>147</th>
</tr>
</thead>
<tbody>
<tr>
<td>161</td>
<td>154</td>
<td>145</td>
<td>141</td>
<td>143</td>
<td>150</td>
<td>158</td>
<td>162</td>
<td>163</td>
<td>168</td>
<td>158</td>
<td>150</td>
<td>157</td>
<td>162</td>
<td>161</td>
<td>164</td>
<td></td>
</tr>
<tr>
<td>159</td>
<td>155</td>
<td>153</td>
<td>160</td>
<td>170</td>
<td>172</td>
<td>164</td>
<td>155</td>
<td>171</td>
<td>172</td>
<td>172</td>
<td>180</td>
<td>183</td>
<td>169</td>
<td>164</td>
<td>179</td>
<td></td>
</tr>
<tr>
<td>195</td>
<td>187</td>
<td>179</td>
<td>177</td>
<td>181</td>
<td>187</td>
<td>189</td>
<td>188</td>
<td>178</td>
<td>179</td>
<td>179</td>
<td>185</td>
<td>184</td>
<td>164</td>
<td>151</td>
<td>161</td>
<td></td>
</tr>
<tr>
<td>162</td>
<td>160</td>
<td>152</td>
<td>136</td>
<td>117</td>
<td>104</td>
<td>100</td>
<td>100</td>
<td>101</td>
<td>110</td>
<td>108</td>
<td>108</td>
<td>114</td>
<td>109</td>
<td>98</td>
<td>96</td>
<td></td>
</tr>
<tr>
<td>113</td>
<td>129</td>
<td>149</td>
<td>158</td>
<td>155</td>
<td>149</td>
<td>148</td>
<td>150</td>
<td>131</td>
<td>146</td>
<td>146</td>
<td>142</td>
<td>156</td>
<td>168</td>
<td>164</td>
<td>159</td>
<td></td>
</tr>
<tr>
<td>166</td>
<td>165</td>
<td>162</td>
<td>158</td>
<td>155</td>
<td>155</td>
<td>158</td>
<td>161</td>
<td>170</td>
<td>184</td>
<td>180</td>
<td>168</td>
<td>172</td>
<td>177</td>
<td>174</td>
<td>173</td>
<td></td>
</tr>
<tr>
<td>173</td>
<td>171</td>
<td>174</td>
<td>182</td>
<td>191</td>
<td>191</td>
<td>182</td>
<td>172</td>
<td>169</td>
<td>182</td>
<td>182</td>
<td>176</td>
<td>177</td>
<td>177</td>
<td>179</td>
<td>190</td>
<td></td>
</tr>
<tr>
<td>189</td>
<td>161</td>
<td>152</td>
<td>161</td>
<td>161</td>
<td>156</td>
<td>154</td>
<td>149</td>
<td>149</td>
<td>120</td>
<td>139</td>
<td>96</td>
<td>106</td>
<td>109</td>
<td>101</td>
<td>100</td>
<td>105</td>
</tr>
<tr>
<td>109</td>
<td>103</td>
<td>106</td>
<td>112</td>
<td>112</td>
<td>154</td>
<td>162</td>
<td>132</td>
<td>118</td>
<td>110</td>
<td>113</td>
<td>115</td>
<td>108</td>
<td>104</td>
<td>106</td>
<td></td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>115</td>
<td>116</td>
<td>100</td>
<td>96</td>
<td>115</td>
<td>138</td>
<td>146</td>
<td>152</td>
<td>148</td>
<td>144</td>
<td>143</td>
<td>147</td>
<td>149</td>
<td>149</td>
<td>148</td>
<td></td>
</tr>
<tr>
<td>145</td>
<td>163</td>
<td>166</td>
<td>156</td>
<td>155</td>
<td>160</td>
<td>166</td>
<td>171</td>
<td>156</td>
<td>162</td>
<td>163</td>
<td>156</td>
<td>157</td>
<td>168</td>
<td>175</td>
<td>175</td>
<td></td>
</tr>
<tr>
<td>161</td>
<td>170</td>
<td>170</td>
<td>174</td>
<td>185</td>
<td>177</td>
<td>165</td>
<td>168</td>
<td>180</td>
<td>190</td>
<td>190</td>
<td>176</td>
<td>168</td>
<td>176</td>
<td>185</td>
<td>187</td>
<td></td>
</tr>
<tr>
<td>179</td>
<td>183</td>
<td>172</td>
<td>175</td>
<td>192</td>
<td>181</td>
<td>161</td>
<td>165</td>
<td>155</td>
<td>162</td>
<td>161</td>
<td>150</td>
<td>142</td>
<td>146</td>
<td>151</td>
<td>152</td>
<td></td>
</tr>
<tr>
<td>136</td>
<td>155</td>
<td>146</td>
<td>140</td>
<td>162</td>
<td>164</td>
<td>151</td>
<td>158</td>
<td>162</td>
<td>161</td>
<td>159</td>
<td>160</td>
<td>163</td>
<td>164</td>
<td>159</td>
<td>154</td>
<td></td>
</tr>
<tr>
<td>153</td>
<td>190</td>
<td>184</td>
<td>162</td>
<td>176</td>
<td>181</td>
<td>169</td>
<td>172</td>
<td>182</td>
<td>173</td>
<td>170</td>
<td>179</td>
<td>189</td>
<td>186</td>
<td>171</td>
<td>158</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4-8 XV Decompression Result - Values

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 4-9 XV Difference Table

4-11
**Figure 4-10 Dec.c Result, Image**

![Image](image_url)

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 97 | 106 | 115 | 115 | 107 | 101 | 101 | 100 | 104 | 104 | 118 | 106 | 91 | 113 | 145 | 153 | 147 |
| 161 | 153 | 145 | 140 | 143 | 150 | 157 | 162 | 162 | 167 | 158 | 149 | 157 | 162 | 161 | 164 |
| 159 | 154 | 153 | 160 | 169 | 172 | 164 | 154 | 171 | 172 | 172 | 179 | 183 | 169 | 163 | 178 |
| 194 | 186 | 178 | 176 | 180 | 186 | 186 | 187 | 178 | 178 | 179 | 164 | 164 | 163 | 150 | 160 |
| 161 | 159 | 151 | 135 | 118 | 105 | 100 | 101 | 102 | 111 | 109 | 108 | 115 | 110 | 99 | 97 |
| 1113 | 129 | 149 | 157 | 154 | 148 | 147 | 145 | 130 | 146 | 146 | 142 | 155 | 167 | 164 | 158 |
| 166 | 165 | 161 | 157 | 155 | 154 | 157 | 160 | 169 | 183 | 179 | 168 | 171 | 176 | 173 | 172 |
| 172 | 171 | 173 | 182 | 191 | 191 | 182 | 172 | 168 | 181 | 181 | 176 | 177 | 176 | 179 | 190 |
| 188 | 160 | 152 | 160 | 160 | 156 | 153 | 149 | 121 | 104 | 97 | 107 | 110 | 101 | 100 | 109 |
| 110 | 104 | 107 | 113 | 132 | 153 | 161 | 131 | 118 | 110 | 114 | 115 | 109 | 105 | 107 |
| 105 | 116 | 116 | 101 | 97 | 116 | 130 | 146 | 151 | 147 | 143 | 143 | 146 | 148 | 148 | 147 |
| 144 | 162 | 165 | 155 | 154 | 160 | 166 | 170 | 155 | 162 | 162 | 156 | 157 | 167 | 174 | 174 |
| 160 | 170 | 170 | 173 | 184 | 177 | 164 | 167 | 180 | 189 | 189 | 175 | 167 | 175 | 184 | 186 |
| 179 | 183 | 172 | 174 | 191 | 180 | 160 | 165 | 155 | 162 | 161 | 149 | 142 | 145 | 151 | 151 |
| 135 | 155 | 146 | 140 | 162 | 163 | 150 | 157 | 161 | 160 | 159 | 160 | 163 | 163 | 159 | 153 |
| 153 | 190 | 163 | 162 | 176 | 181 | 168 | 171 | 182 | 173 | 170 | 179 | 188 | 185 | 170 | 157 |

**Figure 4-11 Dec.c Result, Values**

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 0  | 0  | 1  | 1  | 1  | 1  | 1  | 1  | 0  | 1  | 0  | 1  | 1  | -1  | -1  | 0  | 0  |
| -1  | 0  | -1  | 0  | -1  | 0  | -1  | 0  | -1  | 0  | 0  | 0  | 0  | 0  | -1  |
| -1  | -1  | -1  | -1  | -1  | 0  | -1  | 0  | -1  | -1  | -1  | -1  | -1  | -1  | -1  | -1  |
| 1  | 1  | 0  | 1  | 1  | 1  | 0  | 1  | 1  | 1  | 0  | 0  | 0  | 0  | -1  |
| -1  | -1  | -1  | 0  | 0  | -1  | -1  | 0  | -1  | -1  | 0  | 0  | 0  | 0  | -1  |
| -1  | -1  | -1  | -1  | 0  | -1  | -1  | -1  | -1  | 0  | 0  | 0  | 0  | 0  |
| 0  | -1  | -1  | -1  | 0  | 0  | -1  | 0  | -1  | -1  | 0  | -1  | 0  | -1  |
| 1  | 1  | 1  | 1  | 1  | 0  | 0  | 0  | 1  | 1  | 1  | 0  | -1  | -1  |
| 0  | 0  | 1  | 0  | 1  | 1  | 1  | 0  | 1  | 1  | 1  | 0  | 0  | -1  |
| -1  | 0  | -1  | -1  | -1  | -1  | -1  | -1  | -1  | -1  | 0  | 0  | -1  |
| 0  | 0  | -1  | 0  | -1  | -1  | 0  | -1  | -1  | 0  | -1  | -1  | -1  |
| -1  | -1  | -1  | -1  | 0  | 0  | -1  | -1  | -1  | 0  | 0  | -1  |
| -1  | 0  | -1  | -1  | 0  | 0  | -1  | -1  | -1  | 0  | 0  | -1  |
| 0  | -1  | 0  | -1  | 0  | 0  | -1  | -1  | 0  | 0  | 0  | 0  |
| -1  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

**Figure 4-12 Dec.c Difference Table**

![Difference Table](image_url)

4-12
From the results it can be seen that the XV program was a perfect match to the Independent JPEG Groups software. The dec.c program and the VHDL software had errors that were only plus or minus one from the standard. The reason for the differences
is most likely due to differing implementations of the IDCT. The dec.c and VHDL code use separable 1-D transforms to implement the IDCT, while the XV software and Independent JPEG group use other methods for the IDCT.

4.3 Synthesis of a module

This section contains an example synthesis of one module used in the design of the decoder. The controller module will be as the example module. Only the results of each step in the synthesis of the module will be presented. Refer to the Mentor Graphics manuals for more information on how to perform the synthesis. The first step of the synthesis is to run the compiled VHDL code through the Autologic software. This will produce a symbol for the module, and a schematic of the circuitry needed to implement the module. Figure 4-16 shows the symbol created, figure 4-17 and shows the circuit diagram.

![Figure 4-16 Controller Symbol](image)
Figure 4-17 Controller Schematic
Once the schematic and symbol have been created, the circuit must be prepared to be constructed from standard cells. Input, output, power and ground pads must be added before the circuit can be designed with the standard cells. Note, the pads used here are only for an example of the synthesis of a complete design. If module was to be synthesized as part of a larger module, the pads would not be needed. Figure 4-18 shows the addition of the pads to the controller circuit. The schematic symbol is used on a sheet to add the pads.

Figure 4-18 Controller with I/O Pads
Once the pads have been added, the circuit can be opened in Mentor Graphics layout tool called IC. In IC the standard cells can be automatically placed and routed. The final resulting layout is shown in figure 4-19. This whole procedure could be done on the complete design, except for the IDCT module (the IDCT module is only a behavioral model), to create a complete decoder chip at the cell level.
Figure 4-19 Controller Layout
5. Conclusion

5.1 Accomplishments

This thesis was able to produce a VHDL design of a JPEG decoder. The major accomplishments that were achieved are:

1. A "C" program that was used as a pseudo behavioral model to understand how the JPEG decompression standard worked.

2. A VHDL structural design of a baseline JPEG decoder that can be synthesized. This excludes the IDCT module which is only a behavioral model and is not synthesizable.

3. A test bench, plus additional software that can provide stimulus and receive output from the decoder during a simulation.

4. The successful simulation of the decompression of a 16x16 image using the decoder designed in VHDL.

5. The synthesis of modules, see lessons learned for more comments on this achievement.

5.2 Improvements

Improved IDCT: The current IDCT used in the VHDL design is very inefficient in terms of computations and time. A complete investigation of the available methods for implementing the discrete cosine transform should be done. Then the best method for implementing the DCT in hardware should be selected. Since this stage is the most critical, it consumes the greatest amount of time in the pipeline; it may even have to be custom
designed. Note, a custom design would still involve using VHDL for simulation, but the layout of the circuits at the silicon level would be done by hand, not by a synthesis unit.

**Interface:** If the decoder device were to be used in a system, changes to the input and output modules can be changed. They should be changed to meet the need of the system the decoder is being used in. Only the NextByte and DoUnload modules would need to be changed for any input and/or output requirements.

**Testability:** One major requirement that would need to be added would be to incorporate testability features. Currently the only method to test this device is to decode a complete image. The few ISR (interrupt status register) errors that have been included in the design are only indicators of corrupt data. The design should include methods to test submodules in the design for proper operation. Running a full operational test on every device fabricated is very time inefficient. To avoid this method of testing, some method of designing for testability should be included.

**Baseline:** The decoder should be improved until it meets all baseline requirements. This would involve incorporating the following improvements. Additional controls circuitry would be added to allow for the restart marker to be used. Color images would need to be decoded; this would necessitate the addition of multiscan images. Also the number of blocks in each MCU would have to become a variable size, instead of the hardcoded one 8x8 block now used (As an example, a RGB MCU will have at least three 8x8 blocks, one for each color).

**Pipeline:** Two improvements can be made to the pipelining of the decoder. One improvement is to change the pipeline from single-buffered to doubled-buffered. A
double buffered pipeline allows each stage of the pipeline to work on data without waiting for the next stage to unload the buffer. Figure 5-1 shows the two type of buffering methods used in a pipeline.

![Diagram of single and double buffered pipelines](image)

**Figure 5-1 Two Types of Buffering in Pipelines**

In the double buffered pipeline, the load module will first fill buffer A. Then while the dequant module is unloading buffer A, the load module can fill buffer B. Once the B buffer has been filled, and the A buffer has been emptied by the dequant module, the load module can then switch back to filling the A buffer. This continues until the image has been completely decoded. The other modules work in the same fashion.

The other improvement to the pipeline is to divided the IDCT module into two separate modules that each implement a 1-D transform. Currently the IDCT is implemented as two 1-D transforms in one module. With this implementation, only one transform is being performed at any one time. By separating the transform into two
separate modules, each 1-D transform can be operating on data simultaneously. Combining this with a double buffered pipeline will further increase the speed of the IDCT.

To increase the speed of the decoder, the module that is the slowest in the pipeline must be improved. Since the IDCT module is the slowest, it must be improved first before any other module. Using a double buffered pipelined, splitting the transform into two modules and choosing the most efficient method for implementing the transform are improvements that will speed up the decoder.

5.3 Lessons learned

The following paragraph contains lessons learned during the course of this thesis. Anyone just starting a VHDL design for the first time should find this information helpful.

1. Designing with VHDL using a top down design approach does work. The ability to mix the abstract upper part of the design (behavioral model) with lower levels (structural models) is very useful. To be able to write the design as an abstraction using behavioral modeling to fully understand how the device operates is extremely useful. Then to be able to substitute in structural parts and still have a complete working device is an even greater advantage. Throughout the complete structural design one is able to ensure that the device operates properly as a whole. Simulation time can also be reduced by using a large number of behavioral models and a few structural models that need to be tested. This design would have benefited greatly by the use of the complete top down design in VHDL.

2. The use of more medium size modules, such as the load_hetable module, will allow the design to be synthesized. As the design stands now, most of the modules are
interconnected in the decoder module. This creates one large module that is too big to be synthesized with the current system power and disk space. Also the time that would be needed by the current system to synthesize this module would run into the days. Thus to actually create a design that can be synthesized, many small modules, which can be grouped into a small number of medium sized modules, should be used. For simulation purposes, these medium sized modules can be interconnected in one module. This large module would not be used for synthesis. Rather, the medium sized modules, once synthesized, could be hand connected at the chip level. By keeping the number of medium sized modules as small as possible, the interconnection should not be too tedious. What determines the size of a small or medium module depends on the system in use and some experience. I have found that a small module is typically the size of a small state machine. Usually a small module will have only one or two “tasks” that it performs. Medium modules are collections of small modules that interconnect to one another.

3. The use of medium and small blocks help in the simulation of the device. Testing small modules is very quick, and can be done very thoroughly since there are few signals. Once the small modules have been tested, they can be interconnected into medium sized modules and tested. The use of test benches on these small and medium modules is very helpful. Instead of having to check the results and force the proper inputs each time a simulation is run, a test bench, written in VHDL can be written one time. It is a lot easier to let a program remember all the details that need to be checked.

4. Two additional hints for coding in VHDL are to use as many variables as possible and keep track of all the interconnects. With a large module, such as the decoder that
connected many small modules, it was difficult to keep track of all the interconnecting signals. The use of more medium sized modules will solve this problem. Variables should be used whenever possible to speed up simulation times. Signals must be checked by the simulator whenever any event occurs on them, thus they are very time consuming. Variables are only used whenever they are changed. A tool that shows the activity of each signal would be helpful. The signals that have a high transaction traffic, but few actual events could be analyzed for possible improvements. By cutting down on the events on a signal, the number of times it is checked in a simulation is reduced, and thus simulations are faster.

5. The design of the scan decoder with a pipeline was very useful. By using a pipeline, the design was broken down into smaller parts that could be tested individually. The other by-product of the pipeline was that the device was able to process compressed data faster. Adding in the improvements to the pipeline from the previous section will increase the speed of the pipeline even more.

6. One additional lesson learned from programming in VHDL is it should be thought of as a programming language. Early on in the design, the espresso method was used for designing state machines. This is an old method that is used to design state machines with circuits. It is a very difficult and inefficient method when used in VHDL. By thinking of VHDL as a programming language, coding state machines is very simple. there are times when old methods have to be traded in on new ones.
Bibliography


COMPONENT: jpeg
DESCRIPTION: Test bench for JPEG decoder (entity)

--
-- Name : jpeg_entity
--
-- Purpose: test bench for jpeg decoder/encoder
-- Author: Douglas A. Carpenter
-- Created: 12-Mar-1994 DAC
-- Revised:

-- library and use clauses
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164_extensions.all;

library my_packages;
use my_packages.package_l.all;

library my_components;
use my_components.decoder.all;
use my_components.memory.all;
use my_components.compare.all;

end jpeg;

COMPONENT: jpeg
DESCRIPTION: Test bench for JPEG decoder (architecture)

--
-- Name : jpeg_arch
--
-- Purpose: test bench for jpeg decoder/encoder : architecture
-- Author: Douglas A. Carpenter
-- Created: 12-Mar-1994 DAC
-- Revised:

architecture jpeg_arch of jpeg is

constant CLOCK_CYCLE : time := 5ns;
constant WAIT_TIME : time := 15ns;

component memory
port(next_byte_req: in std_logic;
compressed_data: out
std_logic_vector(7 downto 0);
inc_data_ack : out std_logic;
read_err : out std_logic);
end component;

component decoder
port( -- control inputs/outputs
clock : in std_logic;
start_decode : in std_logic;
decoding : out std_logic;
complete : out std_logic;
complete_ack : in std_logic;
inr : in std_logic;
isr_data : out
std_logic_vector(7 downto 0);
reset : in std_logic;
-- compressed data in
compressed_data : in
std_logic_vector(7 downto 0);
next_byte_req : out std_logic;
inc_data_ack : in std_logic;
result_data_valid : out std_logic;
result_data : out
std_logic_vector(7 downto 0);
result_data_ack : in std_logic);
end component;

component display
port(result_data_valid : in std_logic;
result_data : in
std_logic_vector(7 downto 0);
result_data_ack : out
std_logic;
complete : in
std_logic);
end component;

begin

mem :
memory port map (next_byte_req,
compressed_data,
inc_data_ack,
read_err);

dec :
decoder port map (clock,
start_decode,
decoding,
complete,
complete_ack,
inr,
isr_data,
reset,
compressed_data,
next_byte_req,
inc_data_ack,
read_err,
result_data_valid,
result_data,
result_data_ack);

TheDisplay :
display port map (result_data_valid,
result_data,
result_data_ack,
complete);

A-1
clk_process : process
begin
  clock <= '1';
  wait for CLOCK_CYCLE;
  clock <= '0';
  wait for CLOCK_CYCLE;
end process clk_process;

process
begin
  assert FALSE
  report "Started"
  severity NOTE;
  reset <= '1';
  start_decode <= '0';
  complete_ack <= '0';
  wait for WAIT_TIME;
  reset <= '0';
  wait for 30ns;
  reset <= '1';
  wait for 30ns;
  start_decode <= '1';
  wait until (decoding = '1');
  wait for WAIT_TIME;
  complete_ack <= '1';
  wait until (complete = '1');
  wait for WAIT_TIME;
  complete_ack <= '0';
  wait for WAIT_TIME;
  assert FALSE
  report "Decoding complete"
  severity ERROR;
end process;
end jpeg_arch;

---
--- Name : memory_entity
---
--- Purpose : memory module, provides a byte at a time when requested.
--- Author : Douglas A. Carpenter
--- Created : 12-Mar-1994
--- Revised : 

--- library and use clauses
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164_extensions.all;
use std.textio.all;
library my_packages;
use my_packages.package_l.all;

entity memory is
  port (next_byte_req : in std_ulogic;
    cmprssed_data : out std_ulogic_vector(7 downto 0);
    inc_data_ack : out std_ulogic;
    read_err : out std_ulogic);
end memory;

COMPONENT: memory
DESCRIPTION: compressed data memory, test bench (architecture)

-- Name : memory_arch
--
-- Purpose : memory module
-- Author : Douglas A. Carpenter
-- Created : 12-Mar-1994 DAC
-- Revised :

USE std.textio.all;
architecture memory_arch of memory is
  file IN_FILE: TEXT is
    in "/home/stul/dac4927/jpg/vhdl.dat";
  constant WAIT_DELAY : time := 15ns;
begin
  process
  variable linel
  variable d_string
  variable good
  variable d_byte
  downto 0
  variable j
  variable d_char
begin
  inc_data_ack <= '0';
  read_err <= '0';
  cmprssed_data <= "00000000";
  wait until (next_byte_req = '1');
  while (not ENDFILE (IN_FILE)) loop
    READLINE(IN_FILE, linel);
    wait for WAIT_DELAY;
    READ(linel, d_string, good);
    if not(good) then
      assert FALSE
      report "bad read to string"
      severity NOTE;
      read_err <= '1' after WAIT_DELAY;
    else
      j := 1;
      while (j <61) loop
        d_char := d_string(j);
        case d_char is
          when '0' => d_byte(7 downto 4) := "0000";
          when '1' => d_byte(7 downto 4) := "0001";
          when '2' => d_byte(7 downto 4) := "0010";
          when '3' => d_byte(7 downto 4) := "0011";
          when '4' => d_byte(7 downto 4) := "0100";
          when '5' => d_byte(7 downto 4) := "0101";
          when '6' => d_byte(7 downto 4) := "0110";
          when '7' => d_byte(7 downto 4) := "0111";
        end case;
        j := j + 1;
      end loop;
    end if;
  end loop;
end process;
end architecture memory_arch;
"1000";
when '9' => d_byte(7 downto 4) :=
"1001";
when 'A' => d_byte(7 downto 4) :=
"1010";
when 'B' => d_byte(7 downto 4) :=
"1011";
when 'C' => d_byte(7 downto 4) :=
"1100";
when 'D' => d_byte(7 downto 4) :=
"1101";
when 'E' => d_byte(7 downto 4) :=
"1110";
when 'F' => d_byte(7 downto 4) :=
"1111";
when others => d_byte(7 downto 4) :=
"0000";
end case;
d_char := d_string(j+1);
case d_char is
"0000";
when '0' => d_byte(3 downto 0) :=
"0001";
when '1' => d_byte(3 downto 0) :=
"0010";
when '2' => d_byte(3 downto 0) :=
"0011";
when '3' => d_byte(3 downto 0) :=
"0100";
when '4' => d_byte(3 downto 0) :=
"0101";
when '5' => d_byte(3 downto 0) :=
"0110";
when '6' => d_byte(3 downto 0) :=
"0111";
when '7' => d_byte(3 downto 0) :=
"1000";
when '8' => d_byte(3 downto 0) :=
"1001";
when '9' => d_byte(3 downto 0) :=
"1010";
when 'A' => d_byte(3 downto 0) :=
"1011";
when 'B' => d_byte(3 downto 0) :=
"1100";
when 'C' => d_byte(3 downto 0) :=
"1101";
when 'D' => d_byte(3 downto 0) :=
"1110";
when 'E' => d_byte(3 downto 0) :=
"1111";
when 'F' => d_byte(3 downto 0) :=
"0000";
end case;
j := j + 2;
compressed_data <= d_byte after WAIT_DELAY;
wait for PROP_DELAY;
inc_data_ack <= '1' after WAIT_DELAY;
wait until (next_byte_req = '0');
inc_data_ack <= '0' after WAIT_DELAY;
wait until (next_byte_req = '1');
end loop;
end if;
end loop;
inc_data_ack <= '0';
read_err <= '1';
end process;

COMPONENT: decoder
DESCRIPTION: decoder module (entity)
decoding : out std_ulogic; --ack start
decode signal : out std_ulogic; --decoding
complete : out std_ulogic; --decoding
complete_ack : in std_ulogic; --decoding
complete acknowledge : in std_ulogic; --interrupt
flag
isr_data : out std_logic_vector(7 downto 0); --intr stat reg
reset : in std_ulogic; --reset flag

-- compressed data in
compressed_data : in std_logic_vector(7 downto 0); -- input compressed data
next_byte_req : out std_ulogic; -- request for next byte of compressed data
inc_data_ack : in std_logic_vector(7 downto 0); -- next byte is valid for read
read_err : in std_logic; -- error reading next byte

result_data_valid : out std_ulogic;
result_data : out std_logic_vector(7 downto 0);
result_data_ack : in std_ulogic);
end decoder;

COMPONENT: decoder
DESCRIPTION: decoder module (architecture)

--
-- Name . decoder_arch
--
-- Purpose : decoder module
-- Author : Douglas A. Carpenter
-- Created 12-Mar-1994 DAC
-- Revised .

architecture decoder_arch of decoder is

component isr
port ( reset : in std_ulogic; -- reset isr output to all zeroes
      isr_data_in : in std_logic_vector(7 downto 0); -- incoming ISR data
      isr_data : out std_logic_vector(7 downto 0); -- outgoing ISR data
      isr_latch : in std_ulogic; -- latch the incoming data
      isr_latch_ack : out std_ulogic; -- acknowledge the incoming data latch
      clock : in std_ulogic );
end component;

component nextbyte
port ( clock : in std_ulogic;
       reset : in std_ulogic;
       inc_data_ack : in std_ulogic;
       compressed_data : in std_logic_vector(7 downto 0);
       read_err : in std_ulogic;
       next_req : in std_ulogic;
       next_byte_req : out std_ulogic;
       data : out std_logic_vector(7 downto 0);
       next_ack : out std_ulogic;
       err : out std_ulogic );
end component;

component controller
port( clock : in std_ulogic;
      reset : in std_ulogic;
      work_req : in std_logic_vector(2 downto 0));
end component;

component find_soi
port(clock : in std_ulogic;
     reset : in std_ulogic;
     work_req : in std_logic_vector(2 downto 0);
     work : in std_logic_vector(2 downto 0);
     data : in std_logic_vector(7 downto 0);
     next_ack : in std_ulogic;
     err : in std_ulogic;
     isr_latch_ack : in std_logic;
     req_err : out std_logic;
     ack : out std_logic;
     next_req : out std_logic;
     isr_latch : out std_logic;
     isr_data_in : out std_logic_vector(7 downto 0));
end component;

component fr_header
port(clock : in std_ulogic;
     reset : in std_ulogic;
     work_req : in std_logic_vector(2 downto 0);
     work : in std_logic_vector(7 downto 0);
     data : in std_logic_vector(7 downto 0);
     next_ack : in std_ulogic;
     err : in std_ulogic;
     isr_latch_ack : in std_logic;
     req_err : out std_logic;
     ack : out std_logic;
     next_req : out std_logic;
     isr_latch : out std_logic;
     isr_data_in : out std_logic_vector(7 downto 0));
end component;
component dec_frame
  port(clock : in std_ulogic;
    reset : in std_ulogic;
    work_req : in std_ulogic;
  std_ulogic_vector(2 downto 0);
    find_sos_ack : out std_ulogic;
    decode_scan_ack : in std_ulogic;
    find_sos_err : out std_ulogic;
    decode_scan_err : in std_ulogic;
    req_err : out std_ulogic;
    ack : out std_ulogic;
    find_sos : out std_ulogic;
    decode_scan : out std_ulogic);
  end component;

component define_qtable
  port(clock : in std_ulogic;
    reset : in std_ulogic;
    table_err : out std_ulogic;
    htable_err : in std_ulogic;
    htable_ack : out std_ulogic;
    next_req : out std_ulogic;
    next_ack : in std_ulogic;
    err : in std_ulogic;
    data : in std_ulogic_vector(7 downto 0);
    next_req : out std_ulogic;
    next_ack : in std_ulogic;
    err : in std_ulogic;
    qdata_in : out std_ulogic_vector(7 downto 0);
    qload : out std_ulogic;
    qload_ack : out std_ulogic);
  end component;

component qmem
  port(clock : in std_ulogic;
    inc : in std_ulogic;
    qtable_addr : out std_ulogic_vector(5 downto 0);
    inc_ack : out std_ulogic);
  end component;

component define_hhtable
  port(clock : in std_ulogic;
    reset : in std_ulogic;
    table_err : out std_ulogic;
    htable_err : in std_ulogic;
    htable_ack : out std_ulogic;
    next_req : out std_ulogic;
    next_ack : in std_ulogic;
    err : in std_ulogic;
    data : in std_ulogic_vector(7 downto 0);
    std_ulogic_vector(15 downto 0);
    huff_WADR : out std_ulogic_vector(3 downto 0);
    huff_LD1 : out std_ulogic;
    val_LD1 : out std_ulogic_vector(7 downto 0);
    huff_WADR : out std_ulogic_vector(7 downto 0);
    val_RWADR : out std_ulogic_vector(7 downto 0);
    val_LD1 : out std_logic);
  end component;

component define_qtable
  port(clock : in std_ulogic;
    reset : in std_ulogic;
    table_err : out std_ulogic;
    qtable_req : in std_ulogic;
    qtable_ack : out std_ulogic;
    qreset : out std_ulogic;
    inc : out std_ulogic;
    qtable_addr : in std_ulogic_vector(5 downto 0);
    inc_ack : in std_ulogic;
  end component;
component reg32
  port (reset : in std_ulogic;
        clock : in std_ulogic;
        in_data : in std_ulogic_vector(31 downto 0);
        in_latch : in std_ulogic;
        out_data : out std_ulogic_vector(31 downto 0));
end component;

component reg16
  port (reset : in std_ulogic;
        clock : in std_ulogic;
        in_data : in std_ulogic;
        in_latch : in std_ulogic;
        out_data : out std_ulogic);
end component;

component find_sos
  port (clock : in std_ulogic;
        reset : in std_ulogic;
        find_sos_req : in std_ulogic;
        data : in std_ulogic_vector(7 downto 0);
        next_ack : in std_ulogic;
        err : in std_ulogic;
        isr_latch_ack : in std_ulogic;
        qtable_ack : in std_ulogic;
        htable_ack : in std_ulogic;
        table_err : in std_ulogic;
        find_sos_ack : out std_ulogic;
        find_sos_err : out std_ulogic;
        next_req : out std_ulogic;
        isr_latch : out std_ulogic;
        isr_data_in : out std_logic_vector(7 downto 0);
        qtable_req : out std_ulogic;
        htable_req : out std_ulogic);
end component;

component dec_scan
  port (clock : in std_ulogic;
        reset : in std_ulogic;
        decode_scan : in std_ulogic;
        decode_scan_ack : out std_ulogic;
        decode_scan_err : out std_ulogic;
        computeNumMCUs : out std_ulogic;
        computeNumMCUsack : in std_ulogic;
        startscan : out std_ulogic;
        scancomplete : in std_ulogic;
        decode_scan_header : out std_ulogic;
        decode_scan_header_ack : in std_ulogic;
        decode_scan_header_err : in std_ulogic);
end component;

component dec_scan_header
  port (clock : in std_ulogic;
        reset : in std_ulogic;
        data : in std_ulogic_vector(7 downto 0);
        next_ack : in std_ulogic;
        err : in std_ulogic;
        isr_latch_ack : in std_ulogic;
        next_req : out std_ulogic;
        isr_latch : out std_ulogic;
        isr_data_in : out std_logic_vector(7 downto 0);
        decode_scan_header_ack : out std_ulogic;
        decode_scan_header_err : out std_ulogic);
end component;

component huff_reg256by16
  port (DIN1 : in std_ulogic_vector(15 downto 0);
        RADR1 : in std_ulogic_vector(7 downto 0);
        WADR1 : in std_ulogic_vector(7 downto 0);
        DOUT1 : out std_ulogic_vector(15 downto 0);
        LD1 : in std_ulogic;
        WE1 : in std_ulogic);
end component;

component huff_reg256by8
  port (DIN1 : in std_ulogic_vector(7 downto 0);
        RADR1 : in std_ulogic_vector(3 downto 0);
        WADR1 : in std_ulogic_vector(3 downto 0);
        DOUT1 : out std_ulogic_vector(7 downto 0);
        LD1 : in std_ulogic;
        WE1 : in std_ulogic);
end component;

component huff_reg16byl6
  port (DIN1 : in std_ulogic_vector(15 downto 0);
        RADR1 : in std_ulogic_vector(3 downto 0);
        WADR1 : in std_ulogic_vector(3 downto 0);
        DOUT1 : out std_ulogic_vector(15 downto 0);
        LD1 : in std_ulogic;
        WE1 : in std_ulogic;
        reset : in std_ulogic;
        acdc : in std_ulogic;
        minmax : in std_ulogic);
end component;
end component;

cOMPONENT huff_reg16by8
    PORT (DIN1 : IN std_ulogic_VECTOR(7 DOWNTO 0);
          RAD1 : IN std_ulogic_VECTOR(3 DOWNTO 0);
          WADR1 : IN std_ulogic_VECTOR(3 DOWNTO 0);
          DOUT1 : OUT std_ulogic_VECTOR(7 DOWNTO 0);
          LD1 : IN std_ulogic;
          WE1 : IN std_ulogic;
          reset : IN std_ulogic;
          acdc : IN std_ulogic);
    end COMPONENT;

COMPONENT compute_MCUs
    PORT (clock : IN std_ulogic;
          reset : IN std_ulogic;
          computeNumMCUs : IN std_ulogic;
          computeNumMCUsack : OUT std_ulogic;
          x_data : IN std_ulogic_vector(15 DOWNTO 0);
          y_data : IN std_ulogic_vector(15 DOWNTO 0);
          numMCUs_in : IN std_ulogic_vector(31 DOWNTO 0);
          latchMCUs : OUT std_ulogic);
    end COMPONENT;

COMPONENT mem64by8
    PORT (DIN1 : IN std_ulogic_VECTOR(7 DOWNTO 0);
          RAD1 : IN std_ulogic_VECTOR(5 DOWNTO 0);
          WADR1 : IN std_ulogic_VECTOR(5 DOWNTO 0);
          DOUT1 : OUT std_ulogic_VECTOR(7 DOWNTO 0);
          LD1 : IN std_ulogic;
          WE1 : IN std_ulogic;
          mempreset_in : IN std_ulogic);
    end COMPONENT;

COMPONENT mem64by16
    PORT (DIN1 : IN std_ulogic_VECTOR(15 DOWNTO 0);
          RAD1 : IN std_ulogic_VECTOR(5 DOWNTO 0);
          WADR1 : IN std_ulogic_VECTOR(5 DOWNTO 0);
          DOUT1 : OUT std_ulogic_VECTOR(15 DOWNTO 0);
          LD1 : IN std_ulogic;
          WE1 : IN std_ulogic;
          mempreset_in : IN std_ulogic);
    end COMPONENT;

COMPONENT load_coeff
    PORT (clock : IN std_ulogic;
          reset : IN std_ulogic;
          startscan : IN std_ulogic;
          numMCUsout : IN std_ulogic_VECTOR(31 DOWNTO 0);
          load_coeff_complete : OUT std_ulogic;
          a_full_right : OUT std_ulogic;
          a_empty_right : IN std_ulogic;
          do_load : OUT std_ulogic;
          do_load_ack : IN std_ulogic);
    end COMPONENT;

COMPONENT dequant_coeff
    PORT (clock : IN std_ulogic;
          startscan : IN std_ulogic;
          dequant_coeff_complete : OUT std_ulogic;
          a_full_right : OUT std_ulogic;
          a_empty_right : IN std_ulogic;
          do_dequant : OUT std_ulogic;
          do_dequantUnload_ack : IN std_ulogic;
          do_dequant_ack : IN std_ulogic);
    end COMPONENT;

COMPONENT idct
    PORT (clock : IN std_ulogic;
          reset : IN std_ulogic;
          do_idct : IN std_ulogic;
          do_idctUnload_ack : OUT std_ulogic;
          do_idct_done_ack : OUT std_ulogic;
          DeqToIDCT_RAD1 : OUT std_ulogic_VECTOR(5 DOWNTO 0);
          DeqToIDCT_DOUT1 : IN std_ulogic_VECTOR(15 DOWNTO 0);
          IDCT_ToUnload_DIN1 : OUT std_ulogic_VECTOR(7 DOWNTO 0);
          IDCT_ToUnload_DOUT1 : OUT std_ulogic_VECTOR(5 DOWNTO 0);
          IDCT_ToUnload_LD1 : OUT std_ulogic;
          IDCT_ToUnload_WE1 : OUT std_ulogic);
    end COMPONENT;

COMPONENT unload_coeff
    PORT (clock : IN std_ulogic;
          reset : IN std_ulogic;
          startscan : IN std_ulogic;
          scancomplete : OUT std_ulogic;
          idct_coeff_complete : IN std_ulogic;
          a_full_right : IN std_ulogic;
          a_empty_right : OUT std_ulogic;
          do_unload : OUT std_ulogic);
    end COMPONENT;
do_unload_ack : in std_ulogic;
end component;

component do_load_coeff
port(clock : in std_ulogic;
reset : in std_ulogic;
do_load : in std_ulogic;
do_load_ack : out std_ulogic;
loadDC : out std_ulogic;
loadDC_ack : in std_ulogic;
loadAC : out std_ulogic;
loadAC_ack : in std_ulogic);
end component;

component do_loadDC
port(clock : in std_ulogic;
reset : in std_ulogic;
loadDC : in std_ulogic;
loadDC_ack out std_ulogic;
decode : out std_ulogic;
de_code_ack : in std_ulogic;
decode : out std_ulogic;
de_decode_ack : in std_ulogic;
extend : out std_ulogic;
extend_ack : in std_ulogic;
Tdecode : in std_ulogic_vector(7 downto 0);
D DIFF : in std_ulogic_vector(7 downto 0);
RetV : in std_ulogic_vector(7 downto 0);
LoadDIN1 : out std_ulogic_vector(7 downto 0);
LoadWADR1 : out std_ulogic_vector(5 downto 0);
LoadLD1 : out std_ulogic;
LoadWE1 : out std_ulogic);
end component;

component do_loadAC
port(clock : in std_ulogic;
reset : in std_ulogic;
loadAC : in std_ulogic;
loadAC_ack : out std_ulogic;
huff outSEL : out std_ulogic_vector(0 downto 0);
LoadDIN2 : out std_ulogic_vector(7 downto 0);
LoadWADR2 : out std_ulogic_vector(5 downto 0);
LoadLD1 : out std_ulogic;
LoadWE1 : out std_ulogic;
decode : out std_ulogic;
decode_ack : in std_ulogic;
Tdecode : in std_ulogic_vector(7 downto 0);
receive : out std_ulogic;
receive_ack : in std_ulogic;
extend : out std_ulogic;
extend_ack : in std_ulogic;
recvSSSS : out std_ulogic_vector(7 downto 0);
RetV : in std_ulogic_vector(7 downto 0);
Eval : in std_ulogic_vector(7 downto 0);
mempreset : out std_ulogic);
end component;

component decode
port(clock : in std_ulogic;
reset : in std_ulogic;
decode : in std_ulogic;
decode_ack : out std_ulogic);
component huff_accacsel6
port(IN0 : in std_ulogic_vector(5 downto 0);
IN1 : in std_ulogic_vector(5 downto 0);
SEL : in std_ulogic_vector(0 downto 0);
DOUT : out std_ulogic_vector(15 downto 0));
end component;

component do_unload
port(clock : in std_ulogic;
reset : in std_ulogic;
do_unload : in std_ulogic;
do_unload_ack : out std_ulogic;
IDCT_ToUnload_RADR : in std_ulogic_vector(5 downto 0);
IDCT_ToUnload_DOUT : in std_ulogic_vector(7 downto 0);
result_data_valid : out std_ulogic;
result_data : out std_ulogic_vector(7 downto 0);
result_data_ack : in std_ulogic);
end component;

signal work_reg : std_ulogic_vector(2 downto 0);
signal isr_data_in : std_logic_vector(7 downto 0);
signal isr_latch : std_logic_vector(7 downto 0);
signal isr_latch_ack : std_logic;
signal next_req : std_logic_vector(0 downto 0);
signal next_ack : std_logic;
signal ack : std_logic_vector(0 downto 0);
signal err : std_logic_vector(0 downto 0);
signal table_error : std_logic_vector(0 downto 0);
signal data : std_logic_vector(0 downto 0);
signal req_error : std_logic_vector(0 downto 0);
signal htable : std_logic_vector(0 downto 0);
signal qtable : std_logic_vector(0 downto 0);
signal htable_ack : std_logic;
signal qtable_ack : std_logic;
signal qreset : std_logic;
signal inc : std_logic;
signal inc_ack : std_logic;
signal qtable_raddr : std_logic_vector(5 downto 0);
signal qtable_addr : std_logic_vector(5 downto 0);
signal qdata_in : std_logic_vector(7 downto 0);
signal qload : std_logic;
signal qload_ack : std_logic;
signal qreq : std_logic;
signal qreq_ack : std_logic;
signal qdata_out : std_logic_vector(7 downto 0);
signal x_latch : std_logic;
signal y_latch : std_logic;
signal xy_data : std_logic;
signal huffmax_DOUT : std_logic_vector(15 downto 0);
signal x_data : std_logic_vector(15 downto 0);
signal y_data : std_logic_vector(15 downto 0);
signal huffACmin_DOUT : std_logic;
signal find_sos_req : std_logic;
signal decode_scan : std_logic;
signal find_sos_ack : std_logic;
signal decode_scan_ack : std_logic;
signal find_sos_err : std_logic;
signal decode_scan_err : std_logic;
signal huff_DIN : std_logic_vector(15 downto 0);
signal huffDCmin_DOUT : std_logic_vector(15 downto 0);
signal huffDCmax_DOUT : std_logic_vector(15 downto 0);
signal huffDCvalptr_DOUT : std_logic_vector(15 downto 0);
signal huffACmin_DOUT : std_logic_vector(15 downto 0);
signal huffACmax_DOUT : std_logic_vector(15 downto 0);
signal huffDCvalptr_DOUT : std_logic_vector(15 downto 0);
signal huffmin_DOUT : std_logic_vector(15 downto 0);
signal huffmax_DOUT : std_logic_vector(15 downto 0);
signal huffvalptr_DOUT : std_logic_vector(15 downto 0);
signal huffval_DOUT : std_logic_vector(15 downto 0);
signal huffoutSEL : std_logic_vector(0 downto 0);
signal huff_WADR : std_logic_vector(3 downto 0);
signal huff_RADR : std_logic_vector(3 downto 0);
signal huff_LDI : std_logic;
signal DCmin_WE1 : std_logic;
signal DCmax_WE1 : std_logic;
signal DCvalptr_WE1 : std_logic;

A-9
signal DCval_WE1 : std_ulogic;
signal ACmin_WE1 : std_ulogic;
signal ACmax_WE1 : std_ulogic;
signal ACvalptr_WE1 : std_ulogic;
signal ACval_WE1 : std_ulogic;

signal val_DIN : std_ulogic_vector(7 downto 0);
signal val_WADR : std_ulogic_vector(7 downto 0);
signal val_RADR : std_ulogic_vector(7 downto 0);
signal valACval_DOUT : std_ulogic_vector(7 downto 0);
signal val_LDl : std_ulogic;

signal computeNumMCUs : std_ulogic;
signal computeNumMCUsack : std_ulogic;

signal numMCUs_in : std_ulogic_vector(31 downto 0);
signal numMCUs_out : std_ulogic_vector(31 downto 0);
signal latchMCUs : std_ulogic;

signal startscan : std_ulogic;
signal scancomplete : std_ulogic;

signal load_coeff_complete : std_ulogic;
signal loadtodeq_a_full : std_ulogic;
signal loadtodeq_a_empty : std_ulogic;

signal dequant_coeff_complete : std_ulogic;
signal deqtoidct_a_full : std_ulogic;
signal deqtoidct_a_empty : std_ulogic;

signal idct_coeff_complete : std_ulogic;
signal idcottounload_a_full : std_ulogic;
signal idcottounload_a_empty : std_ulogic;

signal do_load : std_ulogic;
signal do_load_ack : std_ulogic;

signal loadAC : std_ulogic;
signal loadAC_ack : std_ulogic;

signal ddecode : std_ulogic_wired_or std_ulogic;

signal DecToIDCT_DIN1 : std_ulogic_vector(15 downto 0);
signal DecToIDCT_RADR1 : std_ulogic_vector(5 downto 0);
signal DeqToIDCT_WADRl : std_ulogic_vector(5 downto 0);
signal DeqToIDCT_DOUT1 : std_ulogic_vector(15 downto 0);
signal DeqToIDCT_LDl : std_ulogic_wired_or std_ulogic;
signal DeqToIDCT_WEl : std_ulogic_wired_or std_ulogic;
signal IDCT_ToUnload_DINl : std_ulogic_vector(7 downto 0);
signal IDCT_ToUnload_RADRL : std_ulogic_vector(5 downto 0);
signal IDCT_ToUnload_WADRl : std_ulogic_vector(5 downto 0);
signal IDCT_ToUnload_WEl : std_ulogic_wired_or std_ulogic;
signal IDCT_ToUnload_DOUTl : std_ulogic_vector(7 downto 0);
signal IDCT_ToUnload_LDl : std_ulogic_wired_or std_ulogic;
signal IDCT_ToUnload_WEEl : std_ulogic_wired_or std_ulogic;

signal sel_min : std_ulogic;
signal sel_max : std_ulogic;
signal sel_dc : std_ulogic;
signal sel_ac : std_ulogic;
signal recSSSS : std_ulogic_vector(7 downto 0);
signal RecV : std_ulogic_vector(7 downto 0);
signal Eval : std_ulogic_vector(7 downto 0);
signal memreset : std_ulogic;
signal do_dequant : std_ulogic;
signal do_dequant_ack : std_ulogic;
signal do_dequant_unload_ack : std_ulogic;
signal do_IDCT : std_ulogic;
signal do_IDCT_done_ack : std_ulogic;
signal do_IDCT_unload_ack : std_ulogic;
signal do_unloading : std_ulogic;
signal do_unload_ack : std_ulogic;

begin

 sel_min <= '0';
 sel_max <= '1';
 sel_dc <= '0';
 sel_ac <= '1';

NB :

 nextbyte
 port map (clock,
 inc_data_ack,
 comprsed_data,
 read_err,
 next_req,
 next_byte_req,
 data,
 next_ack,
 err);

Cont :

controller
 port map (clock,
 start_decode,
 complete_ack,
 reset,
 decoding,
 complete,
 intr,
 ack,
 req_err,
 work_req);

fSOF :

 find_soi
 port map (clock,
 reset,
 work_req,
 data,
 next_ack,
 req_err,
 ack,
 next_req,
 isr_latch,
 isr_data_in);

fSOF :

 find_sof
 port map (clock,
 reset,
 work_req,
 data,
 next_ack,
 err,
 isr_latch_ack,
 qtable_ack,
 htable_ack,
 table_err,
 req_err,
 ack,
 next_req,
 isr_latch,
 isr_data_in,
 qtable_request,
 htable_request);

begin

frame_header :

fr_header
 port map (clock,
 reset,
 work_req,
 data,
 next_ack,
 err,
 isr_latch_ack,
 req_err,
 ack,
 next_req,
decode_frame :
  dec_frame
  port map (  
  clock,
  reset,
  work_req,
  find_sos_ack,
  decode_scan_ack,
  find_sos_err,
  decode_scan_err,
  req_err,
  ack,
  find_sos_req,
  decode_scan);

fEOI :
  find_eoi
  port map (  
  clock,
  reset,
  work_req,
  data,
  next_ack,
  err,
  isr_latch_ack,
  req_err,
  ack,
  next_req,
  isr_latch,
  isr_data_in);

DH :
  define_hetable
  port map (  
  clock,
  reset,
  table_err,
  htable_req ,
  htable_ack ,
  next_req,
  next_ack,
  err,
  data,
  DCmin_WEl ,
  DCmax_WEl ,
  DCvalptr_WE1 ,
  DCval_WE1 ,
  ACmin_WE1 ,
  ACmax_WE1 ,
  ACvalptr_WE1 ,
  ACval_WE1 ,
  huff_DIN ,
  huff_WADR ,
  huff_LDl ,
  val_DIN ,
  val_WADR ,
  val_LDl );

DQ :
  define_qtable
  port map (  
  clock,
  reset,
  table_err,
  qtable_req ,
  qtable_ack ,
  qreset,
  inc,
  qtable_addr,
  inc_ack,
  data,
  next_req,
  next_ack,  
  err,
  qdata_in ,
  qload,
  qload_ack );

QA :  
  qaddr
  port map (  
  qreset,
  inc ,
  clock,
  qtable_addr ,
  inc_ack );

QM :  
  qmem
  port map (  
  clock,
  reset,
  qdata_in ,
  qload,
  qload_ack ,
  qdata_out ,
  qreq,
  qreq Ack ,
  qtable_addr ,
  qtable_raddr );

DEQ :  
  dequantize
  port map (  
  clock,
  reset,
  do_dequant ,
  do_dequant_unload_ack ,
  do_dequant_ack ,
  qdata_out ,
  qreq,
  qreq Ack ,
  qtable_addr ,
  loadToDeqRAADRl ,
  loadToDeqDOUTl ,
  DeqToIDCT_DINl ,
  DeqToIDCT_WAdrl ,
  DeqToIDCT_Ldl ,
  DeqToIDCT_WE1l );

Xreg :  
  reg16
  port map (  
  reset,
  clock,
  xy_data ,
  x_latch,
  x_data );

Yreg :  
  reg16
  port map (  
  reset,
  clock,
  xy_data ,
  y_latch,
  y_data );

NumMCUs :  
  reg32
  port map (  
  reset,
  clock,
  numMCUs_in ,
  latchMCUs ,
  numMCUs_out );

FStOfSc :  
  find_sos
  port map (  
  clock,
  reset,
  find_sos_req ,
  data,
  next_ack,
DecTheScan:
    dec_scan
    port map {
      clock, reset, decode_scan, decode_scan_ack, decode_scan_err, computeNumMCUs, computeNumMCUsack, startscan, scancomplete, decode_scan_header, decode_scan_header_ack, decode_scan_header_err};

DecTheScanHeader:
    dec_scan_header
    port map {
      clock, reset, data, next_ack, err, isr_latch_ack, next_req, isr_latch, isr_data_in, decode_scan_header, decode_scan_header_ack, decode_scan_header_err};

computeMCU:
    compute_MCU
    port map {
      clock, reset, computeNumMCUs, computeNumMCUsack, x_data, y_data, numMCUs_in, latchMCUs};

huffDCmincode:
    huff_reg16by8
    port map {
      huff_DIN(7 downto 0), huff_RADR, huff_WADR, huffDCvalptr_DOUT, huff_LDl, DCvalptr_WE1, reset, sel_dc, sel_max); huff_DIN, huff_RADR, huff_WADR, huffACval_DOUT, huff_LDl, ACval_DOUT, ACval1_WE1, reset, sel_ac, sel_min); huff_DIN, huff_RADR, huff_WADR, huffACval_DOUT, huff_LDl, ACval1_WE1, reset, sel_ac, sel_max); huff_DIN, huff_RADR, huff_WADR, huffACval_DOUT, huff_LDl, ACval_DOUT, ACval1_WE1, reset, sel_ac, sel_max);
CoeffLoader:
  load_coeff
  port map (clock,
            reset,
            startscan,
            numMCUs_out,
            load_coeff_complete,
            loadDQDq_a_full,
            loadDQDq_a_empty,
            do_load,
            do_load_ack);

CoeffUnLoad:
  unload_coeff
  port map (clock,
            reset,
            startscan,
            scancomplete,
            idct_coeff_complete,
            idcttunload_a_full,
            idcttunload_a_empty,
            do_unloading,
            do_unload_ack);

CoeffDeQuant:
  dequant_coeff
  port map (clock,
            reset,
            startscan,
            load_coeff_complete,
            dequant_coeff_complete,
            loadDQDq_a_full,
            loadDQDq_a_empty,
            deqtoIDCT_a_full,
            deqtoIDCT_a_empty,
            do_dequant,
            do_dequant_unload_ack,
            do_dequant_ack);

CoeffIDCT:
  idct_coeff
  port map (clock,
            reset,
            startscan,
            dequant_coeff_complete,
            idct_coeff_complete,
            deqtoIDCT_a_full,
            deqtoIDCT_a_empty,
            idcttunload_a_full,
            idcttunload_a_empty,
            do_IDCT,
            do_IDCT_unload_ack,
            do_IDCT_done_ack);

DoLoad:
  do_load_coeff
  port map (clock,
            reset,
            do_load,
            do_load_ack,
            loadDC,
            loadDC_ack,
            loadAC,
            loadAC_ack);

DCload:
  do_loadDC
  port map (clock,
            reset,
            loadDC,
            loadDC_ack,
ddecode,
ddecode_ack,
dreceive ,
dreceive_ack,
eextend ,
eextend_ack,
Tdecode,
RecVal,
RetV,
LoadToDeqDIN1,
LoadToDeqWADR1,
LoadToDeqLDI1,
LoadToDeqWEI1);

 AçLoad :
  do_loadAC
  port map (clock,
            reset,
            loadAC,
            loadAC_ack,
huff_outSEL,
LoadToDeqDIN2,
LoadToDeqWADR2,
LoadToDeqLDI1,
LoadToDeqWEI1,
ddecode,
ddecode_ack,
Tdecode,
acrreceive ,
acrreceive_ack,
acextend,
acextend_ack,
reSSSS,
RecV,
Eval,
mempreset);

CoeffACRECEIVE:
  receive
  port map (clock,
            reset,
            receive,
            receive_ack,
Tdecode,
RecVal,
nextbit_req,
nextbit_ack,
nextbit_data,
nextbit_err,
huff_RADR,
val_RADR,
huffval_DOUT,
huffvalptr_DOUT,
huffmin_DOUT,
huffmax_DOUT);
CoeffDCEXTEND:
  extend
  port map (CoefACEXTEND:
    extend
    port map (GetNextBit:
      nextbit
      port map (clock,
                   reset,
                   extend_ack,
                   RecV,
                   nextbit_req,
                   nextbit_data,
                   nextbit_ack,
                   nextbit_err);
    clock,
    reset,
    aceextend,
    aceextend_ack,
    RecV,
    recSSSS,  
    Eval));

huff_VALsel:
  huff_acdcsel
  port map (valDCval_DOUT,  
            valACval_DOUT,  
            huff_outSEL,  
            huffval_DOUT);

huff_VALPTRsel:
  huff_acdcsel
  port map (huffDCvalptr_DOUT,  
            huffACvalptr_DOUT,  
            huff_outSEL,  
            huffvalptr_DOUT);

huff_MINsel:
  huff_acdcsel16
  port map (huffDCmin_DOUT,  
            huffACmin_DOUT,  
            huff_outSEL,  
            huffmin_DOUT);

huff_MAXsel:
  huff_acdcsel16
  port map (huffDCmax_DOUT,  
            huffACmax_DOUT,  
            huff_outSEL,  
            huffmax_DOUT);

LoadToDeqMem:
  mem64by8
  port map (LoadToDeqDIN,
            LoadToDeqDIN2,
            LoadToDeqDIN3,
            DeqIDCTMem :  
            mem64by16
            port map (DeqToIDCT_DIN1,
                      DeqToIDCT_DIN2,
                      DeqToIDCT_DIN3,
                      DeqToIDCT_DOUT1,
                      DeqToIDCT_DOUT2,
                      DeqToIDCT_DOUT3,
                      DeqToIDCT_WADR1,
                      DeqToIDCT_WADR2,
                      DeqToIDCT_WADR3,
                      DeqToIDCT_WDIR,
                      DeqToIDCT_WSEL,
                      DeqToIDCT_WE1,
                      DeqToIDCT_WE2,
                      DeqToIDCT_WE3);

LoadtoEqDINSelect:
  huff_acdcsel
  port map (LoadToEqDIN1,
            LoadToEqDIN2,
            LoadToEqDIN3,
            huff_outSEL,  
            huffval_DOUT);

LoadtoEqSelect:
  huff_acdcsel16
  port map (LoadToEqWADR1,
            LoadToEqWADR2,
            huff_outSEL,  
            huffval_DOUT);

TheIDCT:
  idct
  port map (clock,
            reset,
            do_idct,
            do_idct_unload_ack,
            do_idct_done_ack,  
            DeqToIDCT_RADR1,  
            DeqToIDCT_DOUT1,
            IDCTToUnload_DIN1,  
            IDCTToUnload_WADR1,  
            IDCTToUnload_LD1,  
            IDCTToUnload_WE1);

IDCTToUnloadMem:
  mem64by8
  port map (IDCTToUnload_DIN1,  
            IDCTToUnload_WADR1,  
            IDCTToUnload_LD1,
            IDCTToUnload_WE1,
            IDCTToUnload_LD2,
            IDCTToUnload_WE2,  
            IDCTToUnload_LD3,
            IDCTToUnload_WE3);

DoUnload:
  doUnload
  port map (clock,
            reset,
            do_unloading,
            do_unload_ack,  
            IDCTToUnload_RADR1,  
            IDCTToUnload_DOUT1,  
            IDCTToUnload_WADR1,  
            IDCTToUnload_LD1,
            IDCTToUnload_WE1,
            IDCTToUnload_LD2,
            IDCTToUnload_WE2,
            IDCTToUnload_LD3,
            IDCTToUnload_WE3);

end decoder_arch;

COMPONENT: display
DESCRIPTION: display module, for test bench (entity)

-- Name: display_entity
-- Purpose: display module, used to simulate output connection.
-- Receives uncompressed image data as input.
-- Author   Douglas A. Carpenter
-- Created  12-Mar-1994
-- Revised   

-- library and use clauses
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164_extensions.all;
use std.textio.all;

library my_packages;
use my_packages.package_l.all;

entity display is
  port(result_data_valid : in std_ulogic;
       result_data : in std_logic_vector(7 downto 0);
       complete : in std_logic);
  end display;

architecture display_arch of display is
  file OUT_FILE: TEXT is "/home/stul/dac4927/jpeg/vhdl.ppm";
begin
  process
    variable linel : line;
    variable d_string : string(1 to 1);
    variable good : boolean;
    variable d_byte : std_logic_vector(7 downto 0);
    variable d_int : integer;
    variable j : integer;
  begin
    result_data_ack := '0';
    j := 0;
    d_string := " ";
    
    forever : loop
    wait until ((result_data_valid = '1')
or (complete = '1'))
    if (complete = '1') then
      if (j /= 0) then
        writeln(OUT_FILE,linel);
        j := 0;
      end if;
    else
      d_byte := result_data;
      d_int := 0;
      if (d_byte(0) = '1') then
        d_int := d_int + 4;
      end if;
      if (d_byte(3) = '1') then
        d_int := d_int + 8;
      end if;
      if (d_byte(4) = '1') then
        d_int := d_int + 16;
      end if;
      if (d_byte(5) = '1') then
        d_int := d_int + 32;
      end if;
      if (d_byte(6) = '1') then
        d_int := d_int + 64;
      end if;
      if (d_byte(7) = '1') then
        d_int := d_int + 128;
      end if;
      write (linel,d_int);
      write (linel,d_string);
      j := j + 1;
      if (j = 17) then
        writeln(OUT_FILE,linel);
        j := 0;
      end if;
    result_data_ack <= '1' after PROP_DELAY;
    end if;
  end loop;
end display_arch;

COMPONENT: display
DESCRIPTION: display module (architecture)

-- Name : display_arch
-- Purpose : display module
-- Author  Douglas A. Carpenter
-- Created : 12-Mar-1994 DAC
-- Revised :

architecture display_arch of display is
  file OUT_FILE: TEXT is "/home/stul/dac4927/jpeg/vhdl.ppm";
begin
  process
    variable linel : line;
    variable d_string : string(1 to 1);
    variable good : boolean;
    variable d_byte : std_logic_vector(7 downto 0);
    variable d_int : integer;
    variable j : integer;
  begin
    result_data_ack := '0';
    j := 0;
    d_string := " ";
    
    forever : loop
    wait until ((result_data_valid = '1')
or (complete = '1'))
    if (complete = '1') then
      if (j /= 0) then
        writeln(OUT_FILE,linel);
        j := 0;
      end if;
    else
      d_byte := result_data;
      d_int := 0;
      if (d_byte(0) = '1') then
        d_int := d_int + 4;
      end if;
      if (d_byte(3) = '1') then
        d_int := d_int + 8;
      end if;
      if (d_byte(4) = '1') then
        d_int := d_int + 16;
      end if;
      if (d_byte(5) = '1') then
        d_int := d_int + 32;
      end if;
      if (d_byte(6) = '1') then
        d_int := d_int + 64;
      end if;
      if (d_byte(7) = '1') then
        d_int := d_int + 128;
      end if;
      write (linel,d_int);
      write (linel,d_string);
      j := j + 1;
      if (j = 17) then
        writeln(OUT_FILE,linel);
        j := 0;
      end if;
    result_data_ack <= '1' after PROP_DELAY;
    end if;
  end loop;
end display_arch;

COMPONENT: isr
DESCRIPTION: Interrupt Status Register (entity)

-- Name : isr_entity
-- Purpose : Interrupt Status Register (ISR) module
-- Author   Douglas A. Carpenter
-- Created   30-Mar-1994
-- Revised   

-- library and use clauses
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164_extensions.all;

library my_packages;
use my_packages.package_l.all;

entity isr is
  port( reset : in std_logic; -- reset isr output to all zeroes
        isr_data_in : in std_logic_vector(7 downto 0); -- incoming ISR data
        isr_data : out std_logic_vector(7 downto 0); -- outgoing ISR data
        isr_latch : in std_logic; -- latch the incoming data
        isr_latch_ack : out std_logic; -- acknowledge the incoming data latch
        clock : in std_logic -- clock signal
    );
Interrupt Status Register (architecture)

---
- **Name**: ISR_arch
- **Purpose**: ISR module
- **Author**: Douglas A. Carpenter
- **Created**: 30-Mar-1994 DAC
- **Revised**: A. Carpenter

**DESCRIPTION**

---

\[

data : \text{out} \\
\text{std}_\text{ulogic}\_\text{vector}(7 \text{ downto } 0); \\
next\_ack : \text{out} \hspace{1em} \text{std}_\text{ulogic}; \\
err : \text{out} \hspace{1em} \text{std}_\text{ulogic}; \\
\]

end nextbyte;

NextByte of compressed data from memory (entity)

---
- **Name**: nextbyte_entity
- **Purpose**: nextbyte module
- **Author**: Douglas A. Carpenter
- **Created**: 30-Mar-1994
- **Revised**: A. Carpenter

**DESCRIPTION**

---

\[

data : \text{out} \\
\text{std}_\text{ulogic}\_\text{vector}(7 \text{ downto } 0); \\
next\_ack : \text{out} \hspace{1em} \text{std}_\text{ulogic}; \\
err : \text{out} \hspace{1em} \text{std}_\text{ulogic}; \\
\]

end nextbyte;
COMPONENT: controller
DESCRIPTION: Controls the decoder module. Determines what submodule will be working.

--
-- Name: controller_entity
--
-- Purpose: Controller module
-- Author: Douglas A. Carpenter
-- Created: 30-Mar-1994
-- Revised:

-- library and use clauses
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164_extensions.all;

library my_packages;
use my_packages.package_1.all;

entity controller is
port
  -- external inputs
  clock : in std_ulogic;
  start_decode : in std_ulogic;
  complete_ack : in std_ulogic;
  reset : in std_ulogic;
  -- external outputs
  decoding : out std_ulogic;
  complete : out std_ulogic;
  intr : out std_ulogic;
  -- internal inputs
  ack : in std_ulogic;
  req_err : in std_ulogic;
  -- internal outputs
  work_req : out std_ulogic_vector(2 downto 0));
end controller;

COMPONENT: Controller
(architecture)

--
-- Name: controller_arch
--
-- Purpose: Controller module
-- Author: Douglas A. Carpenter
-- Created: 30-Mar-1994
-- Revised:

architecture controller_arch of controller is
  signal al4 : std_ulogic;
  signal al5 : std_ulogic;
  signal al6 : std_ulogic;
  signal al7 : std_ulogic;
  signal al8 : std_ulogic;
  signal al9 : std_ulogic;
  signal a10 : std_ulogic;
  signal a11 : std_ulogic;
  signal a12 : std_ulogic;
  signal a13 : std_ulogic;
  signal a14 : std_ulogic;
  signal a15 : std_ulogic;
  signal a16 : std_ulogic;
  signal a17 : std_ulogic;
  signal a18 : std_ulogic;
  signal a19 : std_ulogic;
  signal a20 : std_ulogic;
  signal a21 : std_ulogic;
  signal a22 : std_ulogic;
  signal a23 : std_ulogic;
  signal a24 : std_ulogic;
  signal a25 : std_ulogic;
  signal a26 : std_ulogic;
  signal a27 : std_ulogic;
  signal a28 : std_ulogic;
  signal a29 : std_ulogic;
  signal a30 : std_ulogic;

begin
  ContProcess : process (reset, clock)
  begin
    if (reset = '0') then
      D <= "0000";
      al <= '0';
      a2 <= '0';
      a3 <= '0';
      a4 <= '0';
      a5 <= '0';
      a6 <= '0';
      a7 <= '0';
      a8 <= '0';
      a9 <= '0';
      al0 <= '0';
      al1 <= '0';
      al2 <= '0';
      al3 <= '0';
      a14 <= '0';
      a15 <= '0';
      a16 <= '0';
      a17 <= '0';
      a18 <= '0';
      a19 <= '0';
      a20 <= '0';
      a21 <= '0';
      a22 <= '0';
      a23 <= '0';
      a24 <= '0';
      a25 <= '0';
      a26 <= '0';
      a27 <= '0';
      a28 <= '0';
      a29 <= '0';
      a30 <= '0';
    elsif ((clock'event) and (clock = '1') and
      (clock'last_value = '0')) then
      a1 <= '(D(3)="0") and (D(2)="0") and (D(1) =
        '0') and (start_decode = '1') and (ack = '0'));
      a2 <= '(D(3)="0") and (D(2)="0") and (D(1) =
        '0') and (start_decode = '0') and (ack = '0'));
      a3 <= '(D(3)="1") and (D(2)="1") and (D(1) =
        '0') and (complete_ack = '0'));
      a4 <= '(D(3)="0") and (D(2)="0") and (D(1) =
        '0') and (req_err = '0'));
      a5 <= '(D(3)="0") and (D(2)="0") and (D(1) =
        '0') and (req_err = '1'));
      a6 <= '(D(3)="1") and (D(2)="1") and (D(1) =
        '0') and (req_err = '0'));
      a7 <= '(D(3)="1") and (D(2)="1") and (D(1) =
        '0') and (req_err = '1'));
      a8 <= '(D(3)="0") and (D(2)="0") and (D(1) =
        '0') and (req_err = '0'));
      a9 <= '(D(3)="0") and (D(2)="0") and (D(1) =
        '0') and (req_err = '1'));
  end if;
end ContProcess;
end controller_arch;
component Find_SOI
end component;
architecture find_soi_arch of find_soi is
signal D : std_ulogic_vector(3 downto 0);
signal a0 : std_ulogic;
signal a1 : std_ulogic;
signal a2 : std_ulogic;
signal a3 : std_ulogic;
signal a4 : std_ulogic;
signal a5 : std_ulogic;
signal a6 : std_ulogic;
signal a7 : std_ulogic;
signal a8 : std_ulogic;
signal a9 : std_ulogic;
signal a10 : std_ulogic;
signal a11 : std_ulogic;
signal a12 : std_ulogic;
signal a13 : std_ulogic;
signal a14 : std_ulogic;
signal a15 : std_ulogic;
signal a16 : std_ulogic;
signal a17 : std_ulogic;
signal a18 : std_ulogic;
signal a19 : std_ulogic;
signal a20 : std_ulogic;
signal a21 : std_ulogic;
signal a22 : std_ulogic;
signal a23 : std_ulogic;
signal a24 : std_ulogic;
signal a25 : std_ulogic;
signal a26 : std_ulogic;
signal a27 : std_ulogic;
signal a28 : std_ulogic;
signal a29 : std_ulogic;
signal a30 : std_ulogic;
signal a31 : std_ulogic;
signal a32 : std_ulogic;
signal a33 : std_ulogic;
signal a34 : std_ulogic;
signal a35 : std_ulogic;

begin

soi process : process (reset, clock)
begin
if (reset = '0') then
D <= "0000";
a0 <= '0';
a1 <= '0';
a2 <= '0';
a3 <= '0';
a4 <= '0';
a5 <= '0';
a6 <= '0';
a7 <= '0';
a8 <= '0';
a9 <= '0';
a10 <= '0';
a11 <= '0';
a12 <= '0';
a13 <= '0';
a14 <= '0';
a15 <= '0';
a16 <= '0';
a17 <= '0';
a18 <= '0';
a19 <= '0';
a20 <= '0';
a21 <= '0';
a22 <= '0';
a23 <= '0';
a24 <= '0';
a25 <= '0';
a26 <= '0';
a27 <= '0';
a28 <= '0';
a29 <= '0';
a30 <= '0';
a31 <= '0';
a32 <= '0';
a33 <= '0';
a34 <= '0';
a35 <= '0';
elif (clock'event) and (clock = '1') and
(clock'last_value = '0') then
a0 <= ((D(3)='0') and (D(2)='0') and
(D(1)='1') and (D(0)='0') and (data(2)='0') and
(next_ack='0')));
a1 <= ((D(3)='0') and (D(2)='0') and
(D(1)='1') and (D(0)='1') and (data(2)='1') and
(next_ack='0')));
a2 <= ((D(3)='0') and (D(2)='0') and
(D(1)='0') and (D(0)='1') and (data(7)='0') and
(next_ack='0')));
a3 <= ((D(3)='0') and (D(2)='0') and
(D(1)='0') and (D(0)='0') and (data(7)='1') and
(next_ack='0')));
a4 <= ((D(3)='0') and (D(2)='0') and
(D(1)='0') and (D(0)='0') and (data(0)='0') and
(next_ack='0')));
a5 <= ((D(3)='0') and (D(2)='0') and
(D(1)='1') and (D(0)='1') and (data(4)='1') and
(next_ack='0')));
a6 <= ((D(3)='0') and (D(2)='0') and
(D(1)='1') and (D(0)='1') and (data(5)='0') and
(next_ack='0')));
a7 <= ((D(3)='0') and (D(2)='0') and
(D(1)='1') and (D(0)='0') and (data(4)='0') and
(next_ack='0')));
a8 <= ((D(3)='0') and (D(2)='0') and
(D(1)='1') and (D(0)='0') and (data(3)='0') and
(next_ack='0')));
a9 <= ((D(3)='0') and (D(2)='0') and
(D(1)='1') and (D(0)='0') and (data(3)='0') and
(next_ack='0')));
a10 <= ((D(3)='0') and (D(2)='0') and
(D(1)='1') and (D(0)='0') and (data(3)='0') and
(next_ack='0')));
a11 <= ((D(3)='0') and (D(2)='0') and
(D(1)='1') and (D(0)='0') and (data(3)='0') and
(next_ack='0')));
a12 <= ((D(3)='0') and (D(2)='0') and
(D(1)='1') and (D(0)='0') and (data(3)='0') and
(next_ack='0')));
a13 <= ((D(3)='0') and (D(2)='0') and
(D(1)='1') and (D(0)='0') and (data(3)='0') and
(next_ack='0')));
a14 <= ((D(3)='0') and (D(2)='0') and
(D(1)='1') and (D(0)='0') and (data(3)='0') and
(next_ack='0')));
a15 <= ((D(3)='0') and (D(2)='0') and
(D(1)='1') and (D(0)='0') and (data(3)='0') and
(next_ack='0')));
a16 <= ((D(3)='0') and (D(2)='0') and
(D(1)='1') and (D(0)='0') and (data(3)='0') and
(next_ack='0')));
a17 <= ((D(3)='0') and (D(2)='0') and
(D(1)='1') and (D(0)='0') and (next_ack='1'));
a18 <= ((D(3)='0') and (D(2)='0') and
(D(1)='1') and (D(0)='0') and (next_ack='1'));
a19 <= ((D(3)='0') and (D(2)='0') and
(D(1)='1') and (D(0)='0') and (next_ack='1'));
a20 <= ((D(3)='0') and (D(2)='0') and
(D(1)='1') and (D(0)='0') and (next_ack='1'));
a21 <= ((D(3)='0') and (D(2)='0') and
(D(1)='1') and (D(0)='0') and (next_ack='1'));
a22 <= ((D(3)='0') and (D(2)='0') and
(D(1)='1') and (D(0)='0') and (next_ack='1'));
a23 <= ((D(3)='0') and (D(2)='0') and
(D(1)='1') and (D(0)='0') and (next_ack='1'));
a24 <= ((D(3)='0') and (D(2)='0') and
(D(1)='1') and (D(0)='0') and (next_ack='1'));
a25 <= ((D(3)='0') and (D(2)='0') and
(D(1)='1') and (D(0)='0') and (next_ack='1'));
a26 <= ((D(3)='0') and (D(2)='0') and
(D(1)='1') and (D(0)='0') and (next_ack='1'));
a27 <= ((D(3)='0') and (D(2)='0') and
(D(1)='1') and (D(0)='0') and (next_ack='1'));
end
end

(next_ack = '1') and (err = '0')));

a18 <= (D(3)='1') and (D(2)='0') and (D(1) =
'0') and (D(0) = '1') and
(D(latch_ack = '0')));

a19 <= (D(3)='1') and (D(2)='0') and
(D(0) = '1') and
(latch_ack = '0')));

a20 <= (D(3)='0') and (D(2)='1') and (D(1) =
'1') and (D(0) = '0') and
(data(0) = '1') and (next_ack =
'0')));

a21 <= (D(3)='0') and (D(2)='1') and (D(1) =
'1') and (D(0) = '0') and (data(3) = '0') and
(next_ack = '0')));

a22 <= (D(3)='0') and (D(2)='1') and (D(1) =
'1') and (D(0) = '0') and (data(4) = '0') and
(next_ack = '0')));

a23 <= (D(3)='0') and (D(2)='1') and (D(1) =
'1') and (D(0) = '0') and (data(5) = '1') and
(next_ack = '0')));

a24 <= (D(3)='0') and (D(2)='1') and (D(1) =
'1') and (D(0) = '0') and (data(6) = '0') and
(next_ack = '0')));

a25 <= (D(3)='0') and (D(2)='1') and (D(1) =
'1') and (D(0) = '0') and (data(7) = '0') and
(next_ack = '0')));

a26 <= (D(3)='0') and (D(2)='1') and
(D(1) =
'1') and (D(0) = '0') and
(next_ack = '1') and (err = '0')));

a27 <= (D(3)='0') and (D(2)='0') and (D(1) =
'1') and (D(0) = '0') and (data(3) = '0') and
(next_ack = '0')));

a28 <= (D(3)='0') and (D(2)='1') and
(D(1)='1') and (D(0) = '0') and (work_req(1) =
'1')));

a29 <= (D(3)='1') and (D(2)='1') and
(D(1)='1') and (D(0) = '0') and (work_req(2) =
'1')));

a30 <= (D(3)='1') and (D(2)='1') and
(D(1)='1') and (D(0) = '0') and (work_req(0) =
'1')));

a31 <= (D(3)='1') and (D(2)='1') and
(D(1)='1') and (D(0) = '1') and (work_req(1) =
'1')));

a32 <= (D(3)='1') and (D(2)='1') and
(D(1)='1') and (D(0) = '1') and (work_req(2) =
'1')));

a33 <= (D(3)='1') and (D(2)='1') and
(D(1)='1') and (D(0) = '1') and (work_req(0) =
'1')));

a34 <= (D(3)='1') and (D(2)='0') and
(D(1)='1') and (D(0) = '1') and
(latch_ack = '0')));

a35 <= (D(3)='0') and (D(2)='0') and
(D(1)='1') and (D(0) = '0') and (err = '1'));

D3 <= ((a3 = '1') or (a4 = '1') or (a5 = '1')
or (a9 = '1') or (a10 = '1') or (a11 = '1')
or (a12 = '1') or (a13 = '1') or (a16 = '1') or
(a17 = '1') or (a19 = '1') or (a32 = '1')
or (a33 = '1') or (a34 = '1')
or
(a34 = '1') and (a35 = '1')));

D2 <= ((a14 = '1') or (a15 = '1') or (a26 =
'1') or (a27 = '1') or (a28 = '1')) or (a29 = '1')
or (a30 = '1') or (a31 = '1') or (a32 =
'1') or (a33 = '1') or (a34 = '1')
or
(a34 = '1') and (a35 = '1')));

D1 <= ((a1 = '1') or (a9 = '1') or (a14 =
'1') or (a15 = '1') or (a16 = '1') or (a17 =
'1') or (a19 = '1') or (a26 = '1')
or (a27 = '1') or (a28 = '1') or (a29 = '1')
or (a30 = '1') or (a31 = '1') or (a32 =
'1') or (a33 = '1') or (a34 = '1'));

D0 <= ((a0 = '1') or (a2 = '1') or (a3 =
'1') or (a4 = '1') or (a5 = '1') or
(a6 = '1') or (a7 = '1') or (a8 =
'1') or (a10 = '1') or (a12 = '1') or
(a13 = '1') or (a16 = '1') or (a17 =
'1') or (a18 = '1') or (a19 = '1') or (a20 =
'1') or (a21 = '1') or (a22 = '1') or (a23 = '1')
or (a24 = '1') or (a25 = '1') or (a27 =
'1') or (a31 = '1') or (a32 = '1') or (a33 = '1')
or (a34 = '1') or (a35 = '1')));
-- outputs
req_err    : out std_ulogic;
ack        : out std_ulogic;
next_req   : out std_ulogic;
islatch    : out std_ulogic;
isrdatain  : out std_logic_vector(7 downto 0);
htable_reg : out std_ulogic;
htable_req : out std_ulogic;
end find_sof;

COMPONENT find_sof
DESCRIPTION: (architecture)

-- -- Name     : find_SO鹃_arch
-- -- Purpose : find_SO鹃 module
-- -- Author  : Douglas A. Carpenter
-- -- Created : 04-APR-1994
-- -- Revised :

architecture find_sof_arch of find_sof is
  signal D : std_ulogic_vector(3 downto 0);
  signal a : std_ulogic_vector(60 downto 0);

begin
  sof_process : process(reset, clock)
  begin
    if (reset = '0') then
      D <= "0000";
      a <= "0000000000000000000000000000000000000000000000000000000000000000"
    elsif ((clock'event) and (clock = '1')) then
      a(0) <= ((D(3) = '0') and (D(2) = '0') and (D(1) = '1') and (D(0) = '1') and (data = "11111111") and (next_ack = '0'));
      a(1) <= ((D(3) = '0') and (D(2) = '1') and (D(1) = '1') and (D(0) = '0') and (data = "11011011") and (next_ack = '0'));
      a(2) <= ((D(3) = '0') and (D(2) = '1') and (D(1) = '0') and (D(0) = '0') and (data = "11000100") and (next_ack = '0'));
      a(3) <= ((D(3) = '1') and (D(2) = '0') and (D(1) = '1') and (D(0) = '0') and (data = "11000000") and (next_ack = '0'));
      a(4) <= (D(2) = '0') and (D(0) = '0') and (work_req = "010") and (next_ack = '1');
      a(5) <= (D(3) = '0') and (D(2) = '0') and (D(1) = '0') and (work_req = "010") and (next_ack = '0') and (err = '0'));
      a(6) <= (D(3) = '0') and (D(2) = '0') and (D(1) = '0') and (work_req = "010") and (next_ack = '0') and (err = '1'));
      a(7) <= (D(3) = '1') and (D(2) = '1') and (D(1) = '0') and (D(0) = '0') and (htable_ack = '0') and (table_err = '0') and (next_ack = '1') and (err = '0'));
      a(8) <= (D(3) = '0') and (D(2) = '0') and (D(1) = '0') and (D(0) = '0') and (htable_ack = '0') and (table_err = '0') and (next_ack = '1') and (err = '0'));
      a(9) <= ((D(3) = '0') and (D(2) = '0') and (D(1) = '1') and (D(0) = '0') and (next_ack = '0') and (err = '0'));
      a(10) <= (D(3) = '1') and (D(2) = '1') and (D(0) = '0') and (htable_ack = '1') and (table_err = '0') and (next_ack = '0'));
      a(11) <= (D(3) = '1') and (D(2) = '0') and (D(1) = '1') and (D(0) = '0') and (data(2) = '1') and (next_ack = '0'));
      a(12) <= (D(3) = '1') and (D(2) = '0') and (D(1) = '1') and (D(0) = '0') and (data(5) = '1') and (next_ack = '0'));
      a(13) <= ((D(3) = '1') and (D(2) = '0') and (D(1) = '1') and (D(0) = '0') and (data(6) = '0') and (next_ack = '0'));
      a(14) <= ((D(3) = '1') and (D(2) = '0') and (D(1) = '1') and (D(0) = '0') and (data(7) = '0') and (next_ack = '0'));
      a(15) <= ((D(3) = '1') and (D(2) = '0') and (D(1) = '1') and (D(0) = '0') and (data(0) = '1') and (next_ack = '0'));
      a(16) <= ((D(3) = '1') and (D(2) = '0') and (D(1) = '1') and (D(0) = '0') and (data(1) = '1') and (next_ack = '0'));
      a(17) <= ((D(3) = '1') and (D(2) = '0') and (D(1) = '1') and (D(0) = '0') and (data(3) = '1') and (next_ack = '0'));
      a(18) <= ((D(3) = '1') and (D(2) = '0') and (D(1) = '1') and (D(0) = '0') and (data(4) = '1') and (next_ack = '0'));
      a(19) <= ((D(3) = '0') and (D(2) = '1') and (D(1) = '1') and (D(0) = '0') and (next_ack = '1'));
      a(20) <= ((D(3) = '1') and (D(2) = '0') and (D(1) = '0') and (D(0) = '0') and (isr_latch_ack = '0'));
      a(21) <= ((D(3) = '1') and (D(2) = '0') and (D(1) = '0') and (D(0) = '0') and (isr_latch_ack = '0'));
      a(22) <= ((D(3) = '0') and (D(2) = '0') and (D(1) = '1') and (D(0) = '1') and (next_ack = '0'));
      a(23) <= ((D(3) = '0') and (D(2) = '0') and (D(1) = '1') and (D(0) = '0') and (data(1) = '0') and (next_ack = '0'));
      a(24) <= ((D(3) = '0') and (D(2) = '0') and (D(1) = '1') and (D(0) = '0') and (data(0) = '0') and (next_ack = '0'));
      a(25) <= ((D(3) = '0') and (D(2) = '0') and (D(1) = '1') and (D(0) = '0') and (data(4) = '0') and (next_ack = '0'));
      a(26) <= ((D(3) = '0') and (D(2) = '0') and (D(1) = '1') and (D(0) = '0') and (data(7) = '0') and (next_ack = '0'));
      a(27) <= ((D(3) = '0') and (D(2) = '0') and (D(1) = '1') and (D(0) = '0') and (next_ack = '0'));
      a(28) <= ((D(3) = '0') and (D(2) = '0') and (D(1) = '1') and (D(0) = '0') and (next_ack = '0'));
      a(29) <= (D(3) = '0') and (D(2) = '0') and (D(1) = '1') and (D(0) = '0') and (next_ack = '0'));
      a(30) <= (D(3) = '0') and (D(2) = '1') and (D(1) = '0') and (next_ack = '0'));
      a(31) <= (D(3) = '0') and (D(2) = '0') and (D(1) = '0') and (next_ack = '0') and (err = '0'));
      a(32) <= (D(3) = '1') and (D(2) = '1') and (D(1) = '1') and (D(0) = '0') and (htable_ack = '0') and (err = '0'));
      a(33) <= (D(3) = '0') and (D(2) = '1') and (D(1) = '0') and (next_ack = '0'));
    end if;
  end process;
end architecture;

A-22
a(34) <= ((D(3) = '0') and (D(2) = '1') and (D(1) = '0') and (data = '0') and (next_ack = '0'))
and (data(5) = '1') and (next_ack = '0'))
a(35) <= ((D(3) = '0') and (D(2) = '1') and (D(1) = '0') and (data = '0') and (next_ack = '0'))
a(36) <= ((D(3) = '1') and (D(2) = '1') and (D(1) = '1') and (D(0) = '1') and (data(7) = '0'))
a(37) <= ((D(3) = '0') and (D(2) = '1') and (D(1) = '0') and (data(0) = '1') and (next_ack = '0'))
a(38) <= ((D(3) = '0') and (D(2) = '1') and (D(1) = '0') and (data(1) = '1') and (next_ack = '0'))
a(39) <= ((D(3) = '0') and (D(2) = '1') and (D(1) = '0') and (data(2) = '0') and (next_ack = '0'))
a(40) <= ((D(3) = '0') and (D(2) = '1') and (D(1) = '0') and (data(3) = '1') and (next_ack = '0'))
a(41) <= ((D(3) = '0') and (D(2) = '1') and (D(1) = '0') and (data(4) = '1') and (next_ack = '0'))
a(42) <= ((D(3) = '1') and (D(2) = '0') and (D(1) = '1') and (D(0) = '0') and (next_ack = '1'))
a(43) <= ((D(3) = '0') and (D(2) = '1') and (D(1) = '1') and (D(0) = '1') and (table_err = '0'))
a(44) <= ((D(3) = '1') and (D(2) = '1') and (D(1) = '1') and (D(0) = '0') and (table_err = '0'))
a(45) <= ((D(3) = '1') and (D(2) = '1') and (D(1) = '0') and (D(0) = '0') and (table_err = '0'))
a(46) <= ((D(3) = '0') and (D(2) = '1') and (D(1) = '0') and (D(0) = '1') and (next_ack = '0'))
a(47) <= ((D(3) = '0') and (D(2) = '1') and (D(1) = '1') and (D(0) = '0'))
a(48) <= ((D(3) = '0') and (D(2) = '0') and (D(1) = '1') and (D(0) = '1') and (next_ack = '1'))
a(49) <= ((D(3) = '0') and (D(2) = '1') and (D(1) = '0') and (D(0) = '0') and (table_err = '0'))
a(50) <= ((D(3) = '0') and (D(2) = '0') and (D(1) = '1') and (D(0) = '0') and (next_ack = '0'))
a(51) <= ((D(3) = '1') and (D(2) = '0') and (D(1) = '1') and (D(0) = '1') and (work_req = '0'))
a(52) <= ((D(3) = '1') and (D(2) = '0') and (D(1) = '1') and (D(0) = '1') and (work_req = '1'))
a(53) <= ((D(3) = '1') and (D(2) = '0') and (D(1) = '1') and (D(0) = '1') and (work_req = '1'))
a(54) <= ((D(3) = '1') and (D(2) = '1') and (D(1) = '1') and (D(0) = '1') and (table_ack = '1'))
a(55) <= ((D(3) = '1') and (D(2) = '1') and (D(1) = '0') and (D(0) = '1') and (work_req = '1'))
a(56) <= ((D(2) = '1') and (D(1) = '0') and (D(0) = '1') and (work_req(2) = '1'))
a(57) <= ((D(2) = '1') and (D(1) = '0') and (D(0) = '1') and (work_req(1) = '1'))
a(58) <= ((D(3) = '0') and (D(1) = '0') and (D(0) = '1') and (err = '1'))
a(59) <= ((D(3) = '1') and (D(1) = '0') and (D(0) = '1'))
a(60) <= ((D(3) = '0') and (D(2) = '1') and (D(0) = '1') and (table_err = '1'))

D(3) <= (a(2) = '1') or (a(5) = '1') or (a(4) = '1') or (a(6) = '1') or (a(7) = '1') or (a(10) = '1') or (a(12) = '1') or (a(13) = '1') or (a(14) = '1') or (a(15) = '1') or (a(16) = '1') or (a(17) = '1') or (a(18) = '1') or (a(20) = '1') or (a(21) = '1') or (a(22) = '1') or (a(23) = '1') or (a(24) = '1') or (a(25) = '1') or (a(26) = '1') or (a(27) = '1') or (a(28) = '1') or (a(29) = '1') or (a(31) = '1') or (a(32) = '1') or (a(36) = '1') or (a(43) = '1') or (a(45) = '1') or (a(46) = '1') or (a(47) = '1') or (a(48) = '1') or (a(49) = '1') or (a(50) = '1') or (a(51) = '1') or (a(52) = '1') or (a(53) = '1') or (a(54) = '1')

D(0) <= (a(1) = '1') or (a(3) = '1') or (a(5) = '1') or (a(6) = '1') or (a(11) = '1') or (a(12) = '1') or (a(13) = '1') or (a(14) = '1') or (a(15) = '1') or (a(16) = '1') or (a(17) = '1') or (a(18) = '1') or (a(20) = '1') or (a(21) = '1') or (a(22) = '1') or (a(23) = '1') or (a(24) = '1') or (a(25) = '1') or (a(26) = '1') or (a(27) = '1') or (a(28) = '1') or (a(29) = '1') or (a(31) = '1') or (a(32) = '1') or (a(36) = '1') or (a(43) = '1') or (a(45) = '1') or (a(46) = '1') or (a(47) = '1') or (a(48) = '1') or (a(49) = '1') or (a(50) = '1') or (a(51) = '1') or (a(52) = '1') or (a(53) = '1') or (a(54) = '1') or (a(56) = '1') or (a(57) = '1')

end if;
end process sof_process;

-- assign output values
req_err <= '1' when ((a(50) = '1') or (a(55) = '1')) or (a(56) = '1') or (a(57) = '1') else '0';
ack <= '1' when ((a(3) = '1') or (a(51) = '1')) or (a(52) = '1') or (a(53) = '1') else '0';
next_req <= '1' when ((a(0) = '1') or (a(5) = '1')) or (a(9) = '1') or (a(11) = '1') or (a(12) = '1') or
component fr_header

description: Decodes the frame header data from the compressed data stream. Found after the SOF marker (entity)

--
-- Name: fr_header entity
-- Purpose: Decode frame header module.
-- Author: Douglas A. Carpenter
-- Created: 04-Apr-1994
-- Revised:

-- library and use clauses
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164_extensions.all;

library my_packages;
use my_packages.package_1.all;

entity fr_header is
port ( -- inputs
  clock : in std_logic;
  reset : in std_logic;
  work req : in std_logic_vector(2 downto 0);
  data : in std_logic_vector(7 downto 0);
  next ack : in std_logic;
  err : in std_logic;
  isr latch ack : in std_logic;
  -- outputs
  req err : out std_logic;
  ack : out std_logic;
  next req : out std_logic;
  isr latch : out std_logic;
  isr data in : out std_logic_vector(7 downto 0);
  x latch : out std_logic;
  y latch : out std_logic;
  xy data : out std_logic_vector(15 downto 0);
);
end fr_header;

ARCHITECTURE fr_header_arch OF fr_header IS

signal D : std_logic_vector(5 downto 0);
signal a0 : std_logic;
signal a1 : std_logic;
signal a2 : std_logic;
signal a3 : std_logic;
signal a4 : std_logic;
signal a5 : std_logic;
signal a6 : std_logic;
signal a7 : std_logic;
signal a8 : std_logic;
signal a9 : std_logic;
signal a10 : std_logic;
signal a11 : std_logic;
signal a12 : std_logic;
signal a13 : std_logic;
signal a14 : std_logic;
signal a15 : std_logic;
signal a16 : std_logic;
signal a17 : std_logic;
signal a18 : std_logic;
signal a19 : std_logic;
signal a20 : std_logic;
signal a21 : std_logic;
signal a22 : std_logic;
signal a23 : std_logic;
signal a24 : std_logic;
signal a25 : std_logic;
signal a26 : std_logic;
signal a27 : std_logic;
signal a28 : std_logic;
signal a29 : std_logic;
signal a30 : std_logic;
signal a31 : std_logic;
signal a32 : std_logic;
signal a33 : std_logic;
signal a34 : std_logic;
signal a35 : std_logic;
signal a36 : std_logic;
signal a37 : std_logic;
signal a38 : std_logic;
signal a39 : std_logic;
signal a40 : std_logic;
signal a41 : std_logic;
signal a42 : std_logic;
signal a43 : std_logic;
signal a44 : std_logic;
signal a45 : std_logic;
signal a46 : std_logic;
signal a47 : std_logic;
signal a48 : std_logic;
signal a49 : std_logic;
signal a50 : std_logic;
signal a51 : std_logic;
signal a52 : std_logic;
signal a53 : std_logic;
signal a54 : std_logic;
signal a55 : std_logic;
signal a56 : std_logic;
signal a57 : std_logic;
signal a58 : std_logic;

end architecture fr_header_arch;
begin
downto 0
begin
  sof i xy da
  signal all9 std ulogic
  signal all8 std ulogic
  signal all7 std ulogic
  signal all6 std ulogic
  signal all5 std ulogic
  signal all4 std ulogic
  signal all3 std ulogic
  signal all2 std ulogic
  signal all1 std ulogic
  signal all0 std ulogic

  a72 signal
  a71 signal
  a70 signal
  a69 signal
  a68 signal
  a67 signal
  a66 signal
  a65 signal
  a64 signal
  a63 signal
  a62 signal
  a61 signal
  a60 signal
  a59 signal
  a58 signal
  a57 signal
  a56 signal
  a55 signal
  a54 signal
  a53 signal
  a52 signal
  a51 signal
  a50 signal
  a49 signal
  a48 signal
  a47 signal
  a46 signal
  a45 signal
  a44 signal
  a43 signal
  a42 signal
  a41 signal
  a40 signal
  a39 signal
  a38 signal
  a37 signal
  a36 signal
  a35 signal
  a34 signal
  a33 signal
  a32 signal
  a31 signal
  a30 signal
  a29 signal
  a28 signal
  a27 signal
  a26 signal
  a25 signal
  a24 signal
  a23 signal
  a22 signal
  a21 signal
  a20 signal
  a19 signal
  a18 signal
  a17 signal
  a16 signal
  a15 signal
  a14 signal
  a13 signal
  a12 signal
  a11 signal
  a10 signal
  a9 signal
  a8 signal
  a7 signal
  a6 signal
  a5 signal
  a4 signal
  a3 signal
  a2 signal
  a1 signal
  a0 signal

if (reset <= a3 <= a2 <= a1 <= a0) then
  D <= "0000000";
a0 <= '0';
a1 <= '0';
a2 <= '0';
a3 <= '0';
a4 <= '0';
end if

signal i xy_data : std_ulogic_vector(15 downto 0);
begin
  sof_process : process(reset, clock)
  begin
    if (reset = '0') then
      D <= "00000000";
a0 <= '0';
a1 <= '0';
a2 <= '0';
a3 <= '0';
a4 <= '0';
    end if
  end process
end begin

a80 <= '0';
a81 <= '0';
a82 <= '0';
a83 <= '0';
a84 <= '0';
a85 <= '0';
a86 <= '0';
a87 <= '0';
a88 <= '0';
a89 <= '0';
a90 <= '0';
a91 <= '0';
a92 <= '0';
a93 <= '0';
a94 <= '0';
a95 <= '0';
a96 <= '0';
a97 <= '0';
a98 <= '0';
a99 <= '0';
a100 <= '0';
a101 <= '0';
a102 <= '0';
a103 <= '0';
a104 <= '0';
a105 <= '0';
a106 <= '0';
a107 <= '0';
a108 <= '0';
a109 <= '0';
a110 <= '0';
a111 <= '0';
a112 <= '0';
a113 <= '0';
a114 <= '0';
a115 <= '0';
a116 <= '0';
a117 <= '0';
a118 <= '0';
a119 <= '0';

i_xy data <= "0000000000000000"

elsif (clock'event and clock = '1') and
(clock'last_value = '0') then
  a0 <= ((D(5) = '0') and (D(4) = '0') and
  (D(3) = '1') and (D(2) = '1') and (D(1) = '0') and
  (D(0) = '1') and (data =
  "000010000"));
a1 <= (D(4) = '0') and
  (D(3) = '0') and (D(2) = '0') and (D(1) = '1') and
  (D(0) = '0') and (data =
  "0000000000"));
a2 <= ((D(5) = '1') and (D(4) = '0') and
  (D(3) = '1') and (D(2) = '1') and (D(1) = '0') and
  (D(0) = '0') and (data =
  "0000000000"));
a3 <= (D(3) = '0') and (D(2) = '1') and (D(1) = '0') and
  (D(0) = '1') and (data =
  "000010110"));
a4 <= ((D(5) = '1') and (D(4) = '1') and
  (D(2) = '1') and (D(1) = '1') and
  (D(0) = '0') and (data =
  "000000001"));
a5 <= ((D(5) = '1') and (D(4) = '1') and
  (D(2) = '1') and (D(1) = '1') and
  (D(0) = '1') and (data =
  "00010001"));
a6 <= ((D(5) = '1') and (D(4) = '1') and
  (D(2) = '0') and (D(1) = '0') and
  (D(0) = '0') and (data =
  "00000001"));
a78 <= (D(4) = '0') and (D(3) = '1') and (D(2) = '0') and (D(1) = '1') and (D(0) = '1'));
a79 <= (D(3) = '0') and (D(2) = '1') and (D(1) = '1') and (err = '0'));
a80 <= ((D(5) = '1') and (D(4) = '1')) and (D(2) = '1') and (D(1) = '1') and (D(0) = '1') and (data(4) = '0'));
a81 <= ((D(5) = '1') and (D(4) = '1')) and (D(2) = '1') and (D(1) = '1') and (D(0) = '0') and (data(4) = '1'));
a82 <= (D(3) = '1') and (D(2) = '0') and (D(0) = '0'));
a83 <= ((D(5) = '1')) and (D(3) = '0') and (D(1) = '0') and (ISR_LATCH_ACK = '0')));
a84 <= ((D(5) = '1')) and (D(4) = '0') and (D(3) = '1') and (D(2) = '0') and (D(0) = '1'));
a85 <= (D(4) = '1') and (D(2) = '0') and (D(0) = '1'));
a86 <= (D(3) = '0') and (D(2) = '1') and (D(1) = '0') and (D(0) = '0'));
a87 <= (D(4) = '1') and (D(3) = '0') and (D(2) = '0') and (D(1) = '0'));
a88 <= ((D(5) = '1')) and (D(4) = '0') and (D(3) = '1') and (D(2) = '1') and (D(1) = '1') and (D(0) = '0') and (work_req(2) = '1'));
a89 <= ((D(5) = '1')) and (D(4) = '0') and (D(3) = '0') and (D(1) = '1') and (ISR_LATCH_ACK = '1'));
a90 <= ((D(5) = '1')) and (D(4) = '0') and (D(3) = '1') and (D(1) = '1') and (D(0) = '0') and (work_req(0) = '1'));
a91 <= ((D(5) = '1')) and (D(4) = '0') and (D(3) = '0') and (D(1) = '1') and (D(0) = '0') and (work_req(1) = '1'));
a92 <= ((D(5) = '0')) and (D(4) = '1') and (D(3) = '0') and (D(1) = '1') and (err = '0'));
a93 <= (D(4) = '1') and (D(3) = '0') and (D(2) = '1') and (D(1) = '0'));
a94 <= (D(3) = '0') and (D(2) = '1') and (D(1) = '1') and (D(0) = '1') and (err = '1'));
a95 <= ((D(5) = '1')) and (D(4) = '1') and (D(2) = '0') and (D(1) = '1') and (err = '0'));
a96 <= (D(4) = '0') and (D(3) = '0') and (D(2) = '1') and (D(1) = '1') and (err = '1'));
a97 <= ((D(5) = '1')) and (D(4) = '1') and (D(2) = '0') and (D(1) = '0') and (data(2) = '1'));
a98 <= ((D(5) = '1')) and (D(4) = '1') and (D(2) = '0') and (D(1) = '0') and (data(5) = '1'));
a99 <= ((D(5) = '1')) and (D(4) = '1') and (D(2) = '0') and (D(1) = '0') and (data(6) = '1'));
a100 <= ((D(5) = '1')) and (D(4) = '1') and (D(2) = '0') and (D(1) = '0') and (data(7) = '1'));
a101 <= ((D(5) = '1')) and (D(4) = '1') and (D(2) = '0') and (D(1) = '0') and (data(0) = '0'));
a102 <= ((D(5) = '1')) and (D(4) = '1') and (D(2) = '0') and (D(1) = '0') and (data(1) = '1'));
a103 <= ((D(5) = '1')) and (D(4) = '1') and (D(2) = '0') and (D(1) = '0') and (data(3) = '1'));
a104 <= ((D(5) = '1')) and (D(4) = '1') and (D(2) = '0') and (D(1) = '0') and (data(4) = '1'));
a105 <= ((D(5) = '1')) and (D(4) = '1') and (D(1) = '0') and (D(0) = '0') and (err = '1'));
a106 <= ((D(5) = '1')) and (D(4) = '1') and (D(2) = '1') and (D(1) = '1') and (data(2) = '1'));
a107 <= ((D(5) = '1')) and (D(4) = '1') and (D(2) = '1') and (D(1) = '1') and (data(5) = '1'));
a108 <= ((D(5) = '1')) and (D(4) = '1') and (D(1) = '1') and (data(6) = '1'));
a109 <= ((D(5) = '1')) and (D(4) = '1') and (D(2) = '1') and (D(1) = '1') and (data(7) = '1'));
a110 <= ((D(5) = '1')) and (D(4) = '1') and (D(2) = '1') and (D(1) = '1') and (data(0) = '0'));
a111 <= ((D(5) = '1')) and (D(4) = '1') and (D(2) = '1') and (D(1) = '1') and (data(1) = '1'));
a112 <= ((D(5) = '1')) and (D(4) = '1') and (D(2) = '1') and (D(1) = '1') and (data(3) = '1'));
a113 <= ((D(5) = '1')) and (D(4) = '1') and (D(1) = '0') and (D(0) = '0')));
a114 <= ((D(5) = '1')) and (D(1) = '1') and (work_req(2) = '1'));
a115 <= ((D(5) = '1')) and (D(3) = '0') and (D(1) = '1') and (work_req(0) = '1'));
a116 <= ((D(5) = '1')) and (D(3) = '0') and (D(1) = '1') and (work_req(1) = '1'));
a117 <= ((D(5) = '1')) and (D(3) = '0') and (D(1) = '1') and (D(0) = '0') and (ISR_LATCH_ACK = '0'));
a118 <= ((D(5) = '1')) and (D(2) = '1') and (D(1) = '0') and (D(0) = '1'));
a119 <= ((D(5) = '1')) and (D(4) = '1') and (D(2) = '0') and (D(0) = '0'));

if ((a16 = '1') or (a25 = '1')) then
  i_xy_data(15 downto 0) <= data;
end if;

if ((a11 = '1') or (a18 = '1')) then
  i_xy_data(7 downto 0) <= data;
end if;

D(5) <= (a2 = '1') or (a4 = '1') or (a5 = '1') or (a6 = '1') or (a8 = '1') or (a29 = '1') or (a30 = '1') or (a34 = '1') or (a35 = '1') or (a44 = '1') or (a45 = '1') or (a47 = '1') or (a48 = '1') or (a49 = '1') or (a50 = '1') or (a51 = '1') or (a52 = '1') or (a53 = '1') or (a54 = '1') or (a55 = '1') or (a57 = '1') or (a58 = '1') or (a59 = '1') or (a60 = '1') or (a61 = '1') or (a63 = '1') or (a64 = '1') or (a66 = '1') or (a67 = '1') or (a68 = '1') or
(a69 = '1') or (a70 = '1') or (a71 = '1') or (a72 = '1') or (a73 = '1') or (a74 = '1') or
(a76 = '1') or (a77 = '1') or (a80 = '1') or (a81 = '1') or (a83 = '1') or (a84 = '1') or
(a88 = '1') or (a89 = '1') or (a90 = '1') or (a91 = '1') or (a94 = '1') or (a95 = '1') or
(a96 = '1') or (a97 = '1') or (a98 = '1') or (a99 = '1') or (a100 = '1') or (a101 = '1') or
(a102 = '1') or (a103 = '1') or (a104 = '1') or (a105 = '1') or (a106 = '1') or (a107 = '1') or
(a108 = '1') or (a109 = '1') or (a110 = '1') or (a111 = '1') or (a112 = '1') or (a114 = '1') or
(a115 = '1') or (a116 = '1') or (a117 = '1') or (a118 = '1') or (a119 = '1');
D(4) <= ((a4 = '1') or (a6 = '1') or (a11 = '1') or (a18 = '1') or (a23 = '1') or (a25 = '1') or
(a28 = '1') or (a36 = '1') or (a42 = '1') or (a44 = '1') or (a45 = '1') or (a56 = '1') or
(a62 = '1') or (a92 = '1') or (a95 = '1') or (a113 = '1') or (a119 = '1');
D(3) <= ((a0 = '1') or (a2 = '1') or (a4 = '1') or (a5 = '1') or (a6 = '1') or (a9 = '1') or
(a15 = '1') or (a16 = '1') or (a27 = '1') or (a31 = '1') or (a32 = '1') or (a33 = '1') or
(a43 = '1') or (a44 = '1') or (a45 = '1') or (a56 = '1') or (a65 = '1') or (a75 = '1') or
(a78 = '1') or (a82 = '1') or (a84 = '1') or (a88 = '1') or (a90 = '1') or (a91 = '1') or
(a95 = '1') or (a118 = '1') or (a119 = '1');
D(2) <= ((a0 = '1') or (a1 = '1') or (a2 = '1') or (a3 = '1') or (a4 = '1') or (a5 = '1') or
(a9 = '1') or (a15 = '1') or (a20 = '1') or (a25 = '1') or (a32 = '1') or (a36 = '1') or
(a38 = '1') or (a43 = '1') or (a45 = '1') or (a46 = '1') or (a47 = '1') or (a56 = '1') or (a79 =
'1') or
(a84 = '1') or (a86 = '1') or (a88 = '1') or (a90 = '1') or (a91 = '1') or (a93 = '1') or
(a118 = '1');
D(1) <= ((a1 = '1') or (a2 = '1') or (a3 = '1') or (a5 = '1') or (a6 = '1') or (a10 = '1') or
(a13 = '1') or (a15 = '1') or (a16 = '1') or (a20 = '1') or (a21 = '1') or (a22 = '1') or
(a23 = '1') or (a24 = '1') or (a26 = '1') or (a27 = '1') or (a32 = '1') or (a36 = '1') or
(a40 = '1') or (a46 = '1') or (a47 = '1') or (a56 = '1') or (a75 = '1') or (a83 = '1') or (a85 = '1') or
(a88 = '1') or (a90 = '1') or (a91 = '1') or (a95 = '1') or (a114 = '1') or (a115 = '1') or
(a116 = '1') or (a119 = '1');
D(0) <= ((a3 = '1') or (a5 = '1') or (a6 = '1') or (a7 = '1') or (a10 = '1') or (a12 = '1') or
(a14 = '1') or (a17 = '1') or (a19 = '1') or (a24 = '1') or (a27 = '1') or (a31 = '1') or
(a37 = '1') or (a39 = '1') or (a41 = '1') or (a45 = '1') or (a62 = '1') or (a65 = '1') or
(a75 = '1') or (a78 = '1') or (a83 = '1') or (a87 = '1') or (a89 = '1') or (a93 = '1') or
(a114 = '1') or (a115 = '1') or (a116 = '1');
end if;
end process sof_process;

-- assign output values
next_req <= '1' when ((a0 = '1') or (a1 = '1') or (a3 = '1') or (a4 = '1') or (a5= '1') or (a6 =
'1') or (a7 = '1') or (a9 = '1') or (a12 = '1') or (a13 = '1') or (a14 = '1') or (a21 =
'1') or (a22 = '1') or (a24 = '1') or (a28 = '1') or (a36 = '1') or (a42 = '1') or (a44 =
'1') or (a46 = '1') or (a78 = '1')) else '0';
ack <= '1' when ((a2 = '1') or (a88 = '1') or (a90 = '1') or (a91 = '1')) else '0';

isr_latch <= '1' when ((a8='1') or (a29='1') or (a34='1') or (a35='1') or (a47 = '1') or (a48 =
'1') or (a49 = '1') or (a50 = '1') or (a51 = '1') or (a52 = '1') or (a53 = '1') or (a54 =
'1') or (a55 = '1') or (a57 = '1') or (a58 = '1') or (a59 = '1') or (a60 = '1') or (a61 =
'1') or (a63 = '1') or (a64 = '1') or (a66 = '1') or (a67 = '1') or (a68 = '1') or (a69 =
'1') or (a70 = '1') or (a71 = '1') or (a72 = '1') or (a73 = '1') or (a74 = '1') or (a76 =
'1') or (a77 = '1') or (a80 = '1') or (a81 = '1') or (a94 = '1') or (a96 = '1') or (a97 =
'1') or (a98 = '1') or (a99 = '1') or (a100 = '1') or (a101='1') or (a102='1') or (a103 =
'1') or (a104 = '1') or (a105 = '1') or (a106 = '1') or (a107='1') or (a108='1') or (a109='1') or
(a110 = '1') or (a111 = '1') or (a112 = '1') or (a117 = '1')) else '0';
isr_data_in <= "00110011" when ((a8='1') or (a29='1') or (a34='1') or (a35='1') or (a47 = '1') or (a48 =
'1') or (a49 = '1') or (a50 = '1') or (a51 = '1') or (a52 = '1') or (a53 = '1') or (a54 =
'1') or (a55 = '1') or (a57 = '1') or (a58 = '1') or (a59 = '1') or (a60 = '1') or (a61 =
'1') or
(a63 = '1') or (a64 = '1') or 
(a66 = '1') or (a67 = '1') or (a68 = '1') or (a69 = '1') or
(a70 = '1') or (a71 = '1') or 
(a72 = '1') or (a73 = '1') or (a74 = '1') or (a76 = '1') or
(a77 = '1') or (a80 = '1') or 
(a81 = '1') or (a94 = '1') or (a96 = '1') or (a97 = '1') or
(a98 = '1') or (a99 = '1') or (a100 = '1') or (a101 = '1') or (a102 = '1') or (a103 = '1') or
(a104 = '1') or (a105 = '1') or (a106 = '1') or (a107 = '1') or (a108 = '1') or (a109 = '1') or
(a110 = '1') or (a111 = '1') or (a112 = '1') or (a117 = '1') else
"0000000000; 

x_latch <= '1' when (a87 = '1') else '0'; 
y_latch <= '1' when (a56 = '1') else '0'; 
xy_data <= i_xy_data; 

req_err <= '1' when ((a83 = '1') or (a14 = '1') or (a15 = '1') or (a16 = '1')) else '0'; 
end fr_header_arch;

COMPONENT: dec_frame
DESCRIPTION: Module controls the decoding of a frame. (entity)

---
-- Name : dec_frame_entity
---
-- Purpose : Decode Frame.
-- Author : Douglas A. Carpenter
-- Created : 04-APR-1994
-- Revised : 04-APR-1994

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164_extensions.all;

library my_packages;
use my_packages.package_1.all;

dec_frame: 
entity dec_frame is
  port ( 
    -- inputs
    clock : in  std_ulogic;
    reset : in  std_ulogic;
    work_req : in std_ulogic;
    std_ulogic_vector[2 downto 0];
    find_sos_ack : in  std_ulogic;
    decode_scan_ack : in  std_ulogic;
    find_sos_err : in  std_ulogic;
    decode_scan_err : in  std_ulogic;
  );
end entity dec_frame;

COMPONENT: dec_frame
DESCRIPTION: (architecture)

---
-- Purpose : Decode frame module
-- Author : Douglas A. Carpenter
-- Created : 04-APR-1994
-- Revised : 04-APR-1994

architecture dec_frame_arch of dec_frame is
  signal D : std_ulogic_vector[2 downto 0];
  signal a0 : std_ulogic;
  signal a1 : std_ulogic;
  signal a2 : std_ulogic;
  signal a3 : std_ulogic;
  signal a4 : std_ulogic;
  signal a5 : std_ulogic;
  signal a6 : std_ulogic;
  signal a7 : std_ulogic;
  signal a8 : std_ulogic;
  signal a9 : std_ulogic;
  signal a10 : std_ulogic;
  signal a11 : std_ulogic;
  signal a12 : std_ulogic;
  signal a13 : std_ulogic;
  signal a14 : std_ulogic;

begin
  sof_process : process(reset, clock)
  begin
    if (reset = '0') then
      D <= "000";
      a0 <= '0';
      a1 <= '0';
      a2 <= '0';
      a3 <= '0';
      a4 <= '0';
      a5 <= '0';
      a6 <= '0';
      a7 <= '0';
      a8 <= '0';
      a9 <= '0';
      a10 <= '0';
      a11 <= '0';
      a12 <= '0';
      a13 <= '0';
      a14 <= '0';
    elsif ((clock'event) and (clock = '1') and 
      (clock'last_value = '0')) then
      a0 <= (D(2) = '0') and (D(1) = '1') and 
      (D(0) = '1') and (find_sos_ack = '1'));
      a1 <= ( (D(1) = '0')) and 
      (D(0) = '1') and (find_sosAck = '1'));
      a2 <= ( (D(1) = '0')) and 
      (work_req = "100") and 
      (find_sosAck = '0') and 
      (find_sosErr = '0'));
      a3 <= ( (D(1) = '0')) and 
      (find_sosErr = '1'));
      a4 <= ( (D(1) = '0')) and 
      (D(0) = '1') and (find_sosAck = '0') and 
      (find_sosErr = '0'));
      a5 <= ( (D(2) = '0') and (D(1) = '1') and 
      (decode_scan_ack = '0') and 
      (decode_scan_err = '0'));
      a6 <= ( (D(2) = '0') and (D(1) = '1') and 
      (D(0) = '0') and (decode_scanAck = '1') and 
      (decode_scanErr = '0'));
      a7 <= ( (D(1) = '0')) and 
      (D(0) = '1') and (find_sosErr = '1'));
      a8 <= ( (D(2) = '1') and 
      (D(0) = '0') and (work_req = '1'));
      a9 <= ( (D(2) = '1')) and 
      (D(0) = '0') and (work_req = '1'));
      a10 <= ( (D(2) = '1')) and 
      (D(0) = '0') and (work_req = '1'));
  end process sof_process;
end architecture dec_frame_arch;
all <= ((D(2) = '1') and
(D(0) = '1') and (work_req(0) = '1')));

a12 <= ((D(2) = '1') and
(D(0) = '1') and (work_req(1) = '1'));
a13 <= ((D(2) = '1') and
(D(0) = '1') and (work_req(2) = '1'));

al4 <= ((D(2) = '0') and (D(1) = '1') and
(decode_scan_err = '1')));

D(2) <= ((a3 = '1') or (a6 = '1') or (a7 = '1') or (a8 = '1') or (a9 = '1') or (a10 = '1') or
(a11 = '1') or (a12 = '1') or (a13 = '1') or (a14 = '1'));

D(1) <= ((a0 = '1') or (a1 = '1') or (a3 = '1') or (a5 = '1') or (a6 = '1') or (a7 = '1') or
(a8 = '1') or (a9 = '1') or (a10 = '1') or (a11 = '1') or (a12 = '1') or (a13 = '1') or
(a14 = '1'));

D(0) <= ((a0 = '1') or (a1 = '1') or (a2 = '1') or (a3 = '1') or (a4 = '1') or (a7 = '1') or
(a11 = '1') or (a12 = '1') or (a13 = '1') or (a14 = '1'));

end if;
end process sof_process;

-- assign output values

ack <= '1' when ((a6 = '1') or (a8 = '1') or (a9 = '1') or (a10 = '1')) else '0';
req_err <= '1' when ((a3 = '1') or (a5 = '1') or (a7 = '1') or
(a11 = '1') or (a12 = '1') or (a13 = '1') or (a14 = '1'))
else '0';

find_sos <= '1' when ((a2 = '1') or (a4 = '1')) else '0';

decode_scan <= '1' when (a5 = '1') else '0';
end dec_frame_arch;

COMPONENT: find_eoi
DESCRIPTION: Finds the EOI marker in the compressed data

--
-- Author: Douglas A. Carpenter
-- Created: 14-APR-1994
-- Revised: 

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164_extensions.all;
library my_packages;
use my_packages.package_1.all;

entity find_eoi is
  port (  
   -- inputs
   clock : in std_ulogic;
   reset : in std_ulogic;
   work_req : in std_ulogic_vector(7 downto 0);
   data : in std_ulogic_vector(7 downto 0);
   next_ack : in std_ulogic;
   err : in std_ulogic;
   isr_latch_ack : in std_ulogic;
  -- outputs
   req_err : out std_ulogic;
   ack : out std_ulogic;
   next_req : out std_ulogic;
   isr_latch : out std_ulogic;
   isr_data_in : out std_ulogic;
  );
end find_eoi;

architecture find_eoi_arch of find_eoi is

begin

end find_eoi_arch;
a9 <= '0';
a10 <= '0';
a11 <= '0';
a12 <= '0';
a13 <= '0';
a14 <= '0';
a15 <= '0';
a16 <= '0';
a17 <= '0';
a18 <= '0';
a19 <= '0';
a20 <= '0';
a21 <= '0';
a22 <= '0';
a23 <= '0';
a24 <= '0';
a25 <= '0';
a26 <= '0';
a27 <= '0';
a28 <= '0';
a29 <= '0';
a30 <= '0';
a31 <= '0';
a32 <= '0';
a33 <= '0';
a34 <= '0';
a35 <= '0';
else if ((clock'event) and (clock = '1')) then
  a0 <= (D(3) = '0') and (D(2) = '1') and (D(1) = '1') and (D(0) = '0') and (data(2) = '1') and (next_ack = '0');
  a1 <= (D(3) = '0') and (D(2) = '0') and (D(1) = '1') and (D(0) = '1') and (data = "11111111" and (next_ack = '0'));
  a2 <= (D(3) = '0') and (D(2) = '0') and (D(1) = '0') and (work_req = "110") and (err = '1');
  a3 <= (D(3) = '0') and (D(2) = '0') and (D(1) = '1') and (D(0) = '1') and (data(7) = '0') and (next_ack = '0');
  a4 <= (D(3) = '0') and (D(2) = '0') and (D(1) = '0') and (D(0) = '0') and (work_req = "110") and (next_ack = '0');
  a5 <= (D(3) = '0') and (D(2) = '0') and (D(1) = '1') and (D(0) = '1') and (data(6) = '0') and (next_ack = '0');
  a6 <= (D(3) = '0') and (D(2) = '0') and (D(1) = '1') and (D(0) = '1') and (data(5) = '0') and (next_ack = '0');
  a7 <= (D(3) = '0') and (D(2) = '0') and (D(1) = '1') and (D(0) = '1') and (data(4) = '0') and (next_ack = '0');
  a8 <= (D(3) = '0') and (D(2) = '0') and (D(1) = '1') and (D(0) = '1') and (data(3) = '0') and (next_ack = '0');
  a9 <= (D(3) = '0') and (D(2) = '0') and (D(1) = '1') and (D(0) = '0') and (next_ack = '0');
  a10 <= (D(3) = '0') and (D(2) = '0') and (D(1) = '1') and (D(0) = '1') and (data(2) = '0') and (next_ack = '0');
  a11 <= (D(3) = '0') and (D(2) = '0') and (D(1) = '0') and (D(0) = '1') and (data(1) = '0') and (next_ack = '0');
  a12 <= (D(3) = '0') and (D(2) = '0') and (D(1) = '1') and (D(0) = '1') and (data(0) = '0') and (next_ack = '0');
  a13 <= (D(3) = '0') and (D(2) = '0') and (D(1) = '1') and (D(0) = '0') and (data(7) = '1') and (next_ack = '0');
  a14 <= (D(3) = '0') and (D(2) = '0') and (D(1) = '0') and (D(0) = '1') and (data(6) = '1') and (data(5) = '0') and (data(4) = '1') and (data(3) = '1') and (data(2) = '0') and (data(1) = '1') and (data(0) = '1') and (next_ack = '0');
  a15 <= (D(3) = '0') and (D(2) = '0') and (D(1) = '0') and (D(0) = '0') and (next_ack = '0');
  a16 <= (D(2) = '0') and (D(1) = '1') and (D(0) = '1') and (next_ack = '0');
  a17 <= (D(3) = '0') and (D(2) = '0') and (D(1) = '1') and (D(0) = '1') and (next_ack = '1') and (err = '0');
  a18 <= (D(3) = '1') and (D(2) = '0') and (D(1) = '0') and (D(0) = '0') and (data(4) = '0') and (data(3) = '0') and (data(2) = '0') and (data(1) = '1') and (data(0) = '0') and (next_ack = '0');
  a19 <= (D(3) = '1') and (D(2) = '0') and (D(1) = '0') and (D(0) = '0') and (next_ack = '0');
  a20 <= (D(3) = '0') and (D(2) = '0') and (D(1) = '1') and (D(0) = '1') and (next_ack = '0');
  a21 <= (D(3) = '0') and (D(2) = '0') and (D(1) = '0') and (D(0) = '0') and (data(3) = '0') and (next_ack = '0');
  a22 <= (D(3) = '0') and (D(2) = '0') and (D(1) = '1') and (D(0) = '0') and (next_ack = '0');
  a23 <= (D(3) = '0') and (D(2) = '0') and (D(1) = '1') and (D(0) = '0') and (data(5) = '1') and (next_ack = '0');
  a24 <= (D(3) = '0') and (D(2) = '1') and (D(1) = '1') and (D(0) = '0') and (data(6) = '0') and (next_ack = '0');
  a25 <= (D(3) = '0') and (D(2) = '0') and (D(1) = '1') and (D(0) = '0') and (data(7) = '0') and (next_ack = '0');
  a26 <= (D(3) = '0') and (D(2) = '1') and (D(1) = '1') and (D(0) = '0') and (next_ack = '1') and (err = '0');
  a27 <= (D(3) = '1') and (D(2) = '1') and (D(1) = '1') and (D(0) = '0') and (work_req0 = '1');
  a28 <= (D(3) = '0') and (D(2) = '0') and (D(1) = '0') and (D(0) = '1') and (next_ack = '1');
  a29 <= (D(3) = '1') and (D(2) = '1') and (D(1) = '1') and (D(0) = '0') and (next_ack = '1');
  a30 <= (D(3) = '1') and (D(2) = '0') and (D(1) = '1') and (D(0) = '0') and (next_ack = '1');
  a31 <= (D(3) = '1') and (D(2) = '1') and (D(1) = '1') and (D(0) = '1') and (work_req0 = '1');
a32 <= ((D(3) = '1') and (D(2) = '1') and (D(1) = '1') and (D(0) = '1')) and (work_req(1) = '1');
a30 <= ((D(3) = '1') and (D(2) = '1') and (D(1) = '1') and (D(0) = '0')) and (work_req(2) = '1');
a31 <= ((D(3) = '0') and (D(2) = '0')) and (D(1) = '1') and (D(0) = '0') and (err = '1');
D(3) <= (a2 = '1') or (a14 = '1') or (a18 = '1') or (D(1) = '1') or (a19 = '1') or (a27 = '1') or (a28 = '1') or (a29 = '1') or (a30 = '1') or (a31 = '1') or (a32 = '1') or (a33 = '1');
end if;
end process eoi_process;

-- assign output values
req_err <= '1' when ((a31 = '1') or (a32 = '1')) or (a33 = '1') or (a34 = '1');
ack <= '1' when ((a14 = '1') or (a27 = '1')) or (a29 = '1') or (a30 = '1') else '0';
next_req <= '1' when ((a0 = '1') or (a1 = '1')) or (a3 = '1') or (a4 = '1') or (a5 = '1') or (a6 = '1') or (a7 = '1') or (a8 = '1') or (a9 = '1') or (a10 = '1') or (a11 = '1') or (a12 = '1') or (a13 = '1') or (a16 = '1') or (a17 = '1') or (a18 = '1') or (a19 = '1') or (a20 = '1') or (a21 = '1') or (a22 = '1') or (a23 = '1') or (a24 = '1') or (a25 = '1') else '0';
isr_latch <= '1' when ((a2 = '1') or (a18 = '1') or (a28 = '1')) or (a34 = '1') or (a35 = '1');
isr_data_in <= 00000000 when ((a2 = '1') or (a18 = '1') or (a28 = '1')) or (a34 = '1') else 
'0';
eoi_process;
end define_hhtable;

-- Name define_hhtable
-- Purpose : Module that controls the loading
-- of an huffman table
-- Author : Douglas A. Carpenter
-- Created : 18-APR-1994
-- Revised :

-- library and use clauses
library mgc_portable, ieee;
use mgc_portable.qsim_logic.all;
use mgc_portable.qsim_relations.all;
use leee.std_logic_1164.all;
use leee.std_logic_1164_extensions.all;

library my_packages;
use my_packages.package_1.all;

library my_components;
use my_components.huff_controller.all;
use my_components.huff_load.all;
use my_components.huff_load_addr.all;
use my_components.huff_bits_addr.all;
use my_components.huff_bits.all;
use my_components.huff_vals_addr.all;
use my_components.huff_vals.all;
use my_components.huff_lencntr.all;
use my_components.huff_acdc_reg.all;
use my_components.huff_huff_loader.all;
use my_components.huff_bits_loader.all;
use my_components.huff_vals_loader.all;
use my_components.huff_load_size.all;
use my_components.huff_load_code.all;
use my_components.huff_reg256by8.all;
use my_components.huff_reg256by16.all;
use my_components.huff_gen.all;

entity define_hhtable is
port
--
clock : in std_logic;
reset : in std_logic;
table_err : out std_logic;
htable_req : in std_logic;
htable_ack : out std_logic;
next_req : out std_logic;
next_ack : in std_logic;
err : in std_logic;
data : in std_logic_vector(7 downto 0);

Dmin_WE1 : out std_logic;
Dmax_WE1 : out std_logic;
Dvalpdr_WE1 : out std_logic;
ACmin_WE1 : out std_logic;
ACmax_WE1 : out std_logic;
ACvalpdr_WE1 : out std_logic;
ACval_WE1 : out std_logic;
huff_DIN : out std_logic_vector(7 downto 0);
huff_WADR : out std_logic_vector(3 downto 0);
huff_LD1 : out std_logic;
val_DIN1 : out std_logic;
val_RWADR : out std_logic_vector(7 downto 0);
val_LD1 : out std_logic;

end define_hhtable;

 COMPONENT : define_hhtable
 DESCRIPTION : (entity)
architecture define_htable_arch of define_htable is

component huff_controller
port ( clock : in std_ulogic;
      reset : in std_ulogic;
      table_req : in std_ulogic;
      htable_ack : out std_ulogic;
      hreset : out std_ulogic;
      hread_in_req : out std_ulogic;
      hload_req : out std_ulogic;
      hload_ack : in std_ulogic;
      herr : in std_ulogic);
end component;

component huff_read_in
port ( clock : in std_ulogic;
      reset : in std_ulogic;
      hread_in_req : in std_ulogic;
      hread_in_ack : out std_ulogic;
      herr : out std_ulogic;
      hlen_ack : in std_ulogic;
      hbits_ack : in std_ulogic;
      hvals_ack : in std_ulogic;
      hreset : in std_ulogic;
      hreset : in std_ulogic;
      hlen_req : out std_ulogic;
      hload_req : out std_ulogic;
      hvals_req : out std_ulogic;
      hvals_req : out std_ulogic);
end component;

component huff_load
port ( clock : in std_ulogic;
      reset : in std_ulogic;
      hload_req : in std_ulogic;
      hload_ack : out std_ulogic;
      hreset : out std_ulogic;
      hsize_ack : in std_ulogic;
      hcode_ack : in std_ulogic;
      hgen_ack : in std_ulogic;
      hsize_req : out std_ulogic;
      hcode_req : out std_ulogic;
      hgen_req : out std_ulogic);
end component;

component huff_len_loader
port ( clock : in std_ulogic;
      reset : in std_ulogic;
      hlen_req : in std_ulogic;
      hlen_ack : out std_ulogic;
      hreset : out std_ulogic;
      hlen_req : in std_ulogic;
      hlen_ack : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_ulogic;
      hsize : out std_u
component huff_lencntr
port ( set_data : in std_ulogic_vector(15 downto 0);
dec : in std_ulogic;
set : in std_ulogic;
clock : in std_ulogic;
length : out std_ulogic_vector(15 downto 0);
dec_ack : out std_ulogic);
end component;

component huff_acdc_reg
port ( D : in std_ulogic_vector(7 downto 0);
Q : out std_ulogic_vector(0 downto 0);
CLK : in std_ulogic;
R : in std_ulogic);
end component;

component huff_bits
port ( DIN1 : in std_ulogic_vector(7 downto 0);
RADR1 : in std_ulogic_vector(3 downto 0);
WADR1 : in std_ulogic_vector(3 downto 0);
LD1 : in std_ulogic;
WE1 : in std_ulogic);
end component;

component huff_bits_addr
port ( hreset : in std_ulogic;
inc : in std_ulogic;
clock : in std_ulogic;
hbits_addr : out std_ulogic_vector(3 downto 0);
inc_ack : out std_ulogic);
end component;

component huff_vals_addr
port ( hreset : in std_ulogic;
inc : in std_ulogic;
clock : in std_ulogic;
hvals_addr : out std_ulogic_vector(7 downto 0);
inc_ack : out std_ulogic);
end component;

component huff_vals
port ( DIN1 : in std_ulogic_vector(7 downto 0);
RADR1 : in std_ulogic_vector(7 downto 0);
WADR1 : in std_ulogic_vector(7 downto 0);
DOUT1 : out std_ulogic_vector(7 downto 0);
LD1 : in std_ulogic;
WE1 : in std_ulogic);
end component;

component huff_load_size
port ( clock : in std_ulogic;
hreset : in std_ulogic;
hsiz_req : in std_ulogic;
hsiz_ack : out std_ulogic;
hsiz_data : out std_ulogic;
bits_addr_reset : out std_ulogic;
bits_inc : out std_ulogic;
bits_inc_ack : in std_ulogic;
bits_data_out : in std_ulogic;
hsiz_LD1 : out std_ulogic;
hsiz_WE1 : out std_ulogic;
hsiz_addr : out std_ulogic_vector(7 downto 0);
hsiz_data : out std_ulogic_vector(7 downto 0));
end component;

component huff_gen
port ( hreset : in std_ulogic;
hcode_req : in std_ulogic;
hcode_ack : out std_ulogic;
hscg_err : out std_ulogic;
hsize_LD1 : out std_ulogic;
hsiz_WE1 : out std_ulogic;
hsiz_addr : out std_ulogic_vector(7 downto 0);
hsiz_data : out std_ulogic_vector(7 downto 0));
end component;

component huff_load_code
port ( clock : in std_ulogic;
hreset : in std_ulogic;
hcode : in std_ulogic_vector(7 downto 0);
hcode_LD1 : out std_ulogic;
hcode_WE1 : out std_ulogic;
hcode_DIN1 : out std_ulogic_vector(15 downto 0);
hsiz_SEL : out std_ulogic_vector(0 downto 0));
end component;

component huff_reg256by8
port ( clock : in std_ulogic;
hreset : in std_ulogic;
hsiz_req : in std_ulogic;
hsiz_ack : out std_ulogic;
hsiz_err : out std_ulogic;
bits_addr_reset : out std_ulogic;
bits_inc : out std_ulogic;
bits_inc_ack : in std_ulogic;
bits_data_out : in std_ulogic;
hsiz_LD1 : out std_ulogic;
hsiz_WE1 : out std_ulogic;
hsiz_addr : out std_ulogic_vector(7 downto 0);
hsiz_data : out std_ulogic_vector(7 downto 0));
end component;
component huff_reg256by16
port(
  DIN1 : in std_ulogic_vector(15 downto 0);
  RADR1 : in std_ulogic_vector(7 downto 0);
  WADR1 : in std_ulogic_vector(7 downto 0);
  DOUT1 : out std_ulogic_vector(15 downto 0);
  LD1 : in std_ulogic;
  WE1 : in std_ulogic);
end component;

component huff_adcsel
port(
  IN0 : in std_ulogic_vector(7 downto 0);
  IN1 : in std_ulogic_vector(7 downto 0);
  SEL : in std_ulogic_vector(0 downto 0);
  DOUT : out std_ulogic_vector(7 downto 0));
end component;

signal hreset : std_ulogic;
signal hload_req : std_ulogic;
signal hload_ack : std_ulogic;
signal hload_req : std_ulogic;
signal hload_ack : std_ulogic;
signal hlen : std_ulogic;
signal hlen_ack : std_ulogic;
signal hsize : std_ulogic_vector(15 downto 0);
signal hsizeAck : std_ulogic_vector(15 downto 0);
signal hcode : std_ulogic_vector(15 downto 0);
signal hcodeAck : std_ulogic_vector(15 downto 0);
signal hsize_DIN1 : std_ulogic;
signal hsize_WEl : std_ulogic;
signal hsize_LDl : std_ulogic;
signal hsize_REQ : std_ulogic;
signal hsize_ADR : std_ulogic;
signal hsizeADR1 : std_ulogic;
signal hsizeADR2 : std_ulogic;
signal hsize_DOUT1 : std_ulogic;
signal hcode_DIN1 : std_ulogic;
signal hcode_WEl : std_ulogic;
signal hcode_LDl : std_ulogic;
signal hcodeREQ : std_ulogic;
signal hcodeADR1 : std_ulogic;
signal hcodeADR2 : std_ulogic;
signal hcode_DOUT1 : std_ulogic;
signal hload_REQ : std_ulogic;
signal hload_ACK : std_ulogic;
signal hreset : std_ulogic;
begin
  huff_controller
  port map (clock, reset, table_err, htable_req, htable_ack, hreset, hread_in_req, hload_req, hread_in_ack, hload_ack, herr);
end

A-36
hreadin :
    huff_read_in
    port map (
        clock,
        hreset,
        hread_in_req,
        hread_in_ack,
        herr,
        hlen_ack,
        hbits_ack,
        hvals_ack,
        hlbv_err,
        hlen_req,
        hbits_req,
        hvals_req);

hload :
    huff_load
    port map (
        clock,
        hreset,
        hload_req,
        hload_ack,
        herr,
        hsiz_ack,
        hcode_ack,
        hgen_ack,
        hscg_err,
        hsiz_req,
        hcode_req,
        hgen_req);

hlen_load :
    huff_len_loader
    port map ( 
        clock,
        hreset,
        hlen_req,
        hlen_ack,
        hlbv_err,
        h_next_req,
        next_ack,
        err,
        data,
        set_data,
        dec,
        set,
        hlength,
        dec_ack,
        acdc_data,
        acdc_en);

hbits_load :
    huff_bits_loader
    port map ( 
        clock,
        hreset,
        hbits_req,
        hbits_ack,
        hlbv_err,
        h_next_req,
        next_ack,
        err,
        data,
        hlength,
        dec,
        dec_ack,
        bits_reset,
        bits_inc,
        bits_inc_ack,
        vals_inc,
        vals_inc_ack,
        vals_DINl,
        vals_LDl,
        vals_WE1,
        bits_addr,
        bits_data_out,
        val_DINl,
        val_RWADR,
        val_LDl,
        DOval_WE1,
        ACval_WE1,
        acdc_data_out);

hlenreg :
    huff_lencntr
    port map ( 
        set_data,
        dec,
        set,
        clock,
        hlength,
        dec_ack);

hacdc_reg :
    huff_acdc_reg
    port map ( 
        acdc_data,
        acdc_data_out,
        clock,
        acdc_en,
        hreset);

hbits :
    huff_bits
    port map ( 
        bits_DINl,
        bits_addr,
        bits_addr,
        bits_data_out,
        bits_LDl,
        bits_WE1);

hbitsaddr :
    huff_bits_addr
    port map ( 
        bits_reset,
        bits_inc,
        clock,
        bits_addr,
        bits_inc_ack);

hvals :
    huff_vals
    port map ( 
        vals_DINl,
        vals_ADRl,
        vals_ADR1,
vals_DOUTl,
vals_LDl,
vals_WE1);

hvals
addr :
huff_vals_addr
port map (bits_reset,
valsa_inc,
clock,
vals_ADRl,
valsa_inc_ack);

hloadsize :
huff_load_size
port map (clock,
hrset,
hsize_req,
hsiz_ack,
hsiz_msg,
bits_reset,
bits_inc,
bits_addr,
hsiz_data_out,
hsiz_DOUTl,
hsiz_LDl,
hsiz_WE1,
hsiz_ADR1,
hsiz_DIN1);

hloadcode :
huff_load_code
port map (clock,
hrset,
hsize_req,
hsize_ack,
hsiz_msg,
hsiz_ADR1,
hsiz_LDl,
hsiz_WE1,
hsiz_DIN1);

huffgen
huff_gen
port map (clock,
hrset,
hsize_msg,
hsize_ADR1,
hsiz_LDl,
hsiz_WE1,
hsiz_DIN1,
hsiz_SEL);

huffregreg:
huff_reg256by8
port map (hsiz_DIN1,
hsiz_ADR,
hsiz_ADR,
hsiz_DOUTl,
hsiz_LDl,
hsiz_WE1);

huffcodereg :
huff_reg256by16
port map (hc0de_DIN1,
hc0de_ADR,
hc0de_ADR,
hc0de_DOUTl,
hc0de_LDl,
hc0de_WE1);

hsize_adr_sel :
huff_adc_dbsel
port map (hc0de_ADR1,
hc0de_ADR2,
hc0de_SEL,
hc0de_ADR);

hcode_adr_sel :
huff_adc_dbsel
port map (hc0de_ADR1,
hc0de_ADR2,
hc0de_SEL,
hc0de_ADR);

next_req <= h_next_req;
end define_htable_arch;

COMPONENT: huff_controller
DESCRIPTION: (entity)

-- Name : huff_controller_entity
-- Purpose :
-- Author : Douglas A. Carpenter
-- Created 02-MAY-1994
-- Revised :

-- library and use clauses
library mgc_portable, ieee;
use mgc_portable.qsim_logic.all;
use mgc_portable.qsim_relations.all;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164_extensions.all;

library my_packages;

entity huff_controller is
port (-- external signals
clock : in std_ulogic;
reset : in std_ulogic;
table_err : out std_ulogic;
htable_req : in std_ulogic;
htable_ack : out std_ulogic;
-- internal signals
hrset : out std_ulogic;
hrset_in_req : out std_ulogic;
hload_req : out std_ulogic;

haffsizereg :
architecture huff_controller of
huff_controller is

signal D : std_ulogic_vector(3 downto 0);
signal a0 : std_ulogic;
signal a1 : std_ulogic;
signal a2 : std_ulogic;
signal a3 : std_ulogic;
signal a4 : std_ulogic;
signal a5 : std_ulogic;
signal a6 : std_ulogic;
signal a7 : std_ulogic;
signal a8 : std_ulogic;
signal a9 : std_ulogic;
signal a10 : std_ulogic;
signal a11 : std_ulogic;
signal a12 : std_ulogic;
signal a13 : std_ulogic;
signal a14 : std_ulogic;
signal a15 : std_ulogic;
signal a16 : std_ulogic;
signal a17 : std_ulogic;

begin
hcontr_process : process(reset, clock)
begins
if (reset = '0') then
D <= "0000";
a0 <= '0';
a1 <= '0';
a2 <= '0';
a3 <= '0';
a4 <= '0';
a5 <= '0';
a6 <= '0';
a7 <= '0';
a8 <= '0';
a9 <= '0';
a10 <= '0';
a11 <= '0';
a12 <= '0';
a13 <= '0';
a14 <= '0';
a15 <= '0';
a16 <= '0';
a17 <= '0';
elsif ((clock'event) and (clock = '1') and (clock'last_value = '0')) then
a0 <= ((D(3) = '0') and (D(2) = '1') and (D(1) = '1') and (D(0) = '1') and (hload_ack = '0'))
and (herr = '0'));
a1 <= ((D(3) = '0') and (D(2) = '0') and (D(1) = '0')
and (herr = '0'))
and (hread_in_ack = '0')));
a2 <= ((D(3) = '0') and (D(2) = '1') and (D(1) = '0') and (D(0) = '1') and (hload_ack = '0'));

end process hcontr_process;

-- assign output values

htable_req <= '1' when ((a5 = '1') or (a9 = '1')) or (a12 = '1') or (a14 = '1')
or (a15 = '1'));
hread_in_req <= '1' when ((a4 = '1') or (a13 = '1')) else '0';
hread_ack <= '1' when ((a2 = '1') or (a3 = '1')) else '0';

reset <= '1' when ((a4 = '1') or (a10 = '1') or (a16 = '1') or (a17 = '1')) else '0';

hreset <= '1' when ((a4 = '1') or (a13 = '1')) else '0';
hload_req <= '1' when ((a0 = '1') or (a7 = '1')) else '0';
begin
hread_process : process(hread, clock)
begin
if (hreset = '0') then

D <= "0000";
a0 <= '0';
a1 <= '0';
a2 <= '0';
a3 <= '0';
a4 <= '0';
a5 <= '0';
a6 <= '0';
a7 <= '0';
a8 <= '0';
a9 <= '0';
a10 <= '0';
a11 <= '0';
a12 <= '0';
a13 <= '0';
a14 <= '0';
a15 <= '0';
else if (clock'event) and (clock = '1') and
(clock'last_value = '0') then
a0 <= (D(2) = '0') and
(D(1) = '0')
and (hlbv_err = '1');
a1 <= (D(2) = '0')
and (hlbv_err = '1');
a2 <= (D(2) = '0')
and (D(1) = '0')
and (D(0) = '0')
and (hbval_ack = '0');
a3 <= (D(2) = '0')
and (D(1) = '0')
and (D(0) = '1')
and (hlbv_err = '0');
a4 <= (D(2) = '1')
and (D(1) = '0')
and (D(0) = '0')
and (hbval_ack = '0');
a5 <= (D(3) = '1')
and (hread_in_req = '1');
a6 <= (D(1) = '1')
and (D(0) = '0')
and (hbval_ack = '1');
a7 <= (D(2) = '1')
and (hbval_ack = '1');
a8 <= (D(2) = '0')
and (D(0) = '1')
and (hval_ack = '1');
a9 <= (D(2) = '1')
and (D(1) = '1')
and (D(0) = '0')
and (hbval_ack = '0');
a10 <= (D(2) = '0')
and (D(1) = '1')
and (D(0) = '1')
and (hval_ack = '0');
a11 <= (D(2) = '1')
and (D(1) = '0')
and (D(0) = '1')
and (hval_ack = '0');
a12 <= (D(3) = '0')
and (D(2) = '1')
and (D(1) = '1')
and (D(0) = '1')
and (hread_in_req = '1');
a13 <= (D(2) = '0')
and (D(1) = '1')
and (D(0) = '0')
and (hbval_err = '1');
a14 <= (D(2) = '1')
and (D(1) = '0')
and (D(0) = '0')
and (hbval_err = '1');
a15 <= (D(2) = '0')
and (D(1) = '0')
and (D(0) = '1')
and (hbval_err = '1');
end if;
end process hread_process;

begin
end hread_controller_arch;

end huff_read_in;

architecture huff_read_in_arch of huff_read_in is


signal D : std_ulogic_vector(3 downto 0);
signal a0 : std_ulogic;
signal a1 : std_ulogic;
signal a2 : std_ulogic;
signal a3 : std_ulogic;
signal a4 : std_ulogic;
signal a5 : std_ulogic;
signal a6 : std_ulogic;
signal a7 : std_ulogic;
signal a8 : std_ulogic;
signal a9 : std_ulogic;
signal a10 : std_ulogic;
signal a11 : std_ulogic;
signal a12 : std_ulogic;
signal a13 : std_ulogic;
signal a14 : std_ulogic;
signal a15 : std_ulogic;

begin

end huff_read_in_arch;

COMPONENT: huff_read_in
DESCRIPTION: (entity)

-- Name : huff_read_in_entity
-- Purpose :
-- Author : Douglas A. Carpenter
-- Created : 02-MAY-1994
-- Revised : 

-- library and use clauses
library mgc_portable, ieee;
use mgc_portable.qsim_logic.all;
use mgc_portable.qsim_relations.all;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164_extensions.all;

library my_packages;

entity huff_read_in is

port {

-- external signals
clock : in std_logic;

-- internal signals
hreset : in std_logic;
hread_in_req : in std_logic;
hread_in_ack : out std_logic;
herr : out std_ulogic;
hlen_ack : in std_logic;
hibits_ack : in std_logic;
hvals_ack : in std_logic;
hiblv_err : in std_logic;
hlen_req : out std_logic;
hibits_req : out std_logic;
hvals_req : out std_logic;

}
end huff_read_in;

COMPONENT: huff_read_in
DESCRIPTION: (architecture)

-- Name : huff_read_in_arch
-- Purpose :
-- Author : Douglas A. Carpenter
-- Created : 02-MAY-1994
-- Revised :

architecture huff_read_in_arch of huff_read_in is

signal D : std_ulogic_vector(3 downto 0);
signal a0 : std_ulogic;
signal a1 : std_ulogic;
signal a2 : std_ulogic;
signal a3 : std_ulogic;
signal a4 : std_ulogic;
signal a5 : std_ulogic;
signal a6 : std_ulogic;
signal a7 : std_ulogic;
signal a8 : std_ulogic;
signal a9 : std_ulogic;
signal a10 : std_ulogic;
signal a11 : std_ulogic;
signal a12 : std_ulogic;
signal a13 : std_ulogic;
signal a14 : std_ulogic;
signal a15 : std_ulogic;

begin

end architecture huff_read_in_arch;
D(3) <= ((a0 = '1') or (a5 = '1') or (a13 = '1') or (a14 = '1') or (a15 = '1')) or (a3 = '1') or (a14 = '1') or (a15 = '1');
D(2) <= ((a0 = '1') or (a4 = '1') or (a5 = '1') or (a6 = '1') or (a7 = '1') or (a9 = '1') or (a11 = '1') or (a12 = '1') or (a13 = '1') or (a14 = '1') or (a15 = '1'));
D(1) <= ((a0 = '1') or (a2 = '1') or (a5 = '1') or (a6 = '1') or (a8 = '1') or (a10 = '1') or (a11 = '1') or (a12 = '1') or (a13 = '1') or (a14 = '1') or (a15 = '1'));
D(0) <= ((a0 = '1') or (a1 = '1') or (a3 = '1') or (a5 = '1') or (a7 = '1') or (a8 = '1') or (a12 = '1') or (a13 = '1') or (a14 = '1') or (a15 = '1'));
end if;
end process hread_process;

-- assign output values
hread_in_ack <= '1' when ((a1 = '1') or (a2 = '1')) else '0';
herr <= '1' when ((a0 = '1') or (a5 = '1') or
(a3 = '1') or (a14 = '1') or (a15 = '1')) else '0';
hlen_req <= '1' when ((a1 = '1') or (a3 = '1')) else '0';
hbts_req <= '1' when ((a2 = '1') or (a10 = '1')) else '0';
hvals_req <= '1' when ((a4 = '1') or (a9 = '1')) else '0';
end huff_read_in_arch;

-- Name : huff_load_entity
-- Purpose :
-- Author : Douglas A. Carpenter
-- Created : 02-MAY-1994
-- Revised :
--
-- library and use clauses
library mgc_portable, ieee;
use mgc_portable.qsim_logic.all;
use mgc_portable.qsim_relations.all;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164_extensions.all;

library my_packages;

entity huff_load is
port (  
-- external signals
  clock : in  std_ulogic;
  hreset : in  std_ulogic;
  hload_req : in  std_ulogic;
  hload_ack : out  std_ulogic;
  herr : out  std_ulogic;
  hsz_ack : in  std_ulogic;
  hcode_ack : in  std_ulogic;
  hgen_ack : in  std_ulogic;

  hscg_err : in  std_ulogic;
  hsz_req : out  std_ulogic;
  hcode_req : out  std_ulogic;
  hgen_req : out  std_ulogic
);
end huff_load;

architecture huff_load_arch of huff_load is
begin
  hload_process : process(hreset, clock)
  begin
    if (hreset = '0') then
      Q <= "0000";
      i_outs <= "00000";
    elsif ((clock'event) and (clock = '1') and
    (clock'last_value = '0')) then
      if (D="0000") then
        if (hload_req = '0') then
          Q <= "0000";
          i_outs <= "00000";
        elsif ((hload_req = '1') and (hsiz_ack = '0')) and
        (hscg_err = '0') then
          Q <= "0000";
          i_outs <= "00000";
        elsif ((hload_req = '1') and (hscg_err = '1')) then
          Q <= "1111";
          i_outs <= "01000";
        elsif (D = "0001") then
          if ((hsiz_ack = '0') and (hscg_err = '0')) then
            Q <= "0001";
            i_outs <= "00100";
            elsif ((hsiz_ack = '1') and (hscg_err = '1')) then
            Q <= "1111";
            i_outs <= "01000";
          elsif ((hsiz_ack = '1') and (hscg_err = '0')) then
            Q <= "0001";
            i_outs <= "00100";
          end if;
      end if;
    end if;
  end process hload_process;

-- library and use clauses
library mgc_portable, ieee;
use mgc_portable.qsim_logic.all;
use mgc_portable.qsim_relations.all;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164_extensions.all;

library my_packages;

entity huff_load is
port (  
-- external signals
  clock : in  std_ulogic;
  hreset : in  std_ulogic;
  hload_req : in  std_ulogic;
  hload_ack : out  std_ulogic;
  herr : out  std_ulogic;
  hsz_ack : in  std_ulogic;
  hcode_ack : in  std_ulogic;
  hgen_ack : in  std_ulogic;

  hscg_err : in  std_ulogic;
  hsz_req : out  std_ulogic;
  hcode_req : out  std_ulogic;
  hgen_req : out  std_ulogic
);
end huff_load;

architecture huff_load_arch of huff_load is
begin
  hload_process : process(hreset, clock)
  begin
    if (hreset = '0') then
      Q <= "0000";
      i_outs <= "00000";
    elsif ((clock'event) and (clock = '1') and
    (clock'last_value = '0')) then
      if (D="0000") then
        if (hload_req = '0') then
          Q <= "0000";
          i_outs <= "00000";
        elsif ((hload_req = '1') and (hsiz_ack = '0')) and
        (hscg_err = '0') then
          Q <= "0000";
          i_outs <= "00000";
        elsif ((hload_req = '1') and (hscg_err = '1')) then
          Q <= "1111";
          i_outs <= "01000";
        elsif (D = "0001") then
          if ((hsiz_ack = '0') and (hscg_err = '0')) then
            Q <= "0001";
            i_outs <= "00100";
            elsif ((hsiz_ack = '1') and (hscg_error = '1')) then
            Q <= "1111";
            i_outs <= "01000";
          elsif ((hsiz_ack = '1') and (hscg_error = '0')) then
            Q <= "0001";
            i_outs <= "00100";
          end if;
      end if;
    end if;
  end process hload_process;

-- library and use clauses
library mgc_portable, ieee;
use mgc_portable.qsim_logic.all;
use mgc_portable.qsim_relations.all;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164_extensions.all;

library my_packages;
Q <= "0010";
i_outs <= "00010";
elif (hscg_err = '1') then
    Q <= "1111";
i_outs <= "01000";
elif (hcode_ack = '1') then
    Q <= "0110";
i_outs <= "00000";
end if;
elif (D = '0110') then
if (hcode_ack = '1') then
    Q <= "0110";
i_outs <= "00000";
elif (hcode_ack = '0') then
    Q <= "0100";
i_outs <= "00001";
elseif (hcode_ack = '0') then
    Q <= "0100";
i_outs <= "00000";
elif (D = '0110') then
if (hload_req = '1') then
    Q <= "0111";
i_outs <= "10000";
elif (hload_req = '0') then
    Q <= "0000";
i_outs <= "00000";
end if;
elif (D = "1111") then
if (hload_req = '1') then
    Q <= "0111";
i_outs <= "10000";
elif (hload_req = '0') then
    Q <= "0000";
i_outs <= "00000";
end if;
elif (D = "1111") then
if (hload_req = '1') then
    Q <= "0111";
i_outs <= "10000";
elif (hload_req = '0') then
    Q <= "0000";
i_outs <= "00000";
end if;
else
    Q <= "0000";
i_outs <= "00000";
end if;
end if;
end process hload_process;

-- assign output values
D <= Q;
hload_ack <= i_outs(4);
herr <= i_outs(3);
hsiz_req <= i_outs(2);
hcode_req <= i_outs(1);
hgen_req <= i_outs(0);

end huff_load_arch;

COMPONENT: huff_bits_addr
DESCRIPTION: ()

--
-- Name: huff_bits_addr_entity
--
-- Purpose: Huffman bits array addresser counter module
-- Author: Douglas A. Carpenter
-- Created: 06-MAY-1994
-- Revised: 

-- library and use clauses
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164_extensions.all;
library my_packages;
use my_packages.package_1.all;

entity huff_bits_addr is
    port ( -- inputs
        hreset : in std_ulogic;
        inc : in std_ulogic;
        clock : in std_logic;
        -- outputs
        hbits_addr : out std_logic_vector(3 downto 0);
        inc_ack : out std_ulogic);
end huff_bits_addr;

COMPONENT: huff_bits_addr
DESCRIPTION: (architecture)

--
-- Name: huff_bits_addr_arch
--
-- Purpose: huff_bits_addr_module
-- Author: Douglas A. Carpenter
-- Created: 06-MAY-1994 DAC
-- Revised: 

architecture huff_bits_addr_arch of huff_bits_addr is
    signal b_addr : std_logic_vector (3 downto 0);
    signal i_ack : std_logic;
begin

    h1_Process : process(hreset,inc)
    begin
        if (hreset = '1') then
            b_addr <= "0000";
        elsif ((inc'event) and (inc = '1')) and
            (inc'last_value = '0') then
            b_addr <= b_addr + "0001";
        end if;
    end process h1_Process;

    h2_Process : process(hreset,clock)
    begin
        if (hreset = '1') then
            i_ack <= '0';
        elsif ((clock'event) and (clock = '1')) and
            (clock'last_value = '0') then
            if (hreset = '1') then
                b_addr <= "0000";
            else
                if (hreset = '1') then
                    b_addr <= "0000";
                end if;
            end if;
        end if;
    end process h2_Process;
end huff_bits_addr_arch;
COMPONENT: huff_bits
DESCRIPTION:       

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_1164_extensions.all;

-- Written by LL_to_VHDL at Tue May 3 09:08:37 1994
-- Parameterized Generator Specification to VHDL Code
--
-- LogicLib generator called: REGISTER_FILE
-- Passed Parameters are:
--  tinst name = huff_bits
--  parameters are:
--    W = 8
--    D = 16
--    read_ports = 1
--
-- huff_bits Entity Description
entity huff_bits is
  port (    
    DIN1 : in std_logic_vector(7 downto 0);
    RADR1 in std_logic_vector(3 downto 0);
    WADR1 : in std_logic_vector(3 downto 0);
    LD1 , in std_logic;
    WEI, in std_logic;
    inc_ack < std_logic;
    hbits_addr <= std_logic_vector(7 downto 0);
  );
end huff_bits;

COMPONENT: huff_bits
DESCRIPTION:       

-- huff_bits Architecture Description
architecture rtl of huff_bits is
  subtype ramword is std_logic_vector(7 downto 0);
type rammemory is array (15 downto 0) of ramword;
signal ram : rammemory;
begin
  REGISTER_FILE_read_Process: process(ram,RADR1)
  variable waddr : integer range 0 to 15;
  variable load : std_logic;
  begin
    -- write mode? (need both WEI and LD1 to be high)
    if (load = '1') then
      ram(waddr) <= DIN1;
    end if;
  end process;

  -- assign output values
  inc_ack <= '0';
hbits_addr <= b_addr;
end huff_bits_addr_arch;

COMPONENT: huff_vals_addr
DESCRIPTION:       

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_1164_extensions.all;

-- Name . huff_vals_addr_entity
-- Purpose : Huffman vals array addresser
counter module
-- Author : Douglas A. Carpenter
-- Created : 06-MAY-1994
-- Revised :

-- library and use clauses
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164_extensions.all;
library my_packages;
use my_packages.package_1.all;

entity huff_vals_addr is
  port (    
    hreset : in std_logic;
    inc : in std_logic;
    clock : in std_logic;
    -- inputs
    hvals_addr : out std_logic_vector(7 downto 0);
    inc_ack : out std_logic;
  );
end huff_vals_addr;

COMPONENT: huff_vals_addr
DESCRIPTION:       

architecture huff_vals_addr_arch of huff_vals_addr is
  signal v_addr : std_logic_vector(7 downto 0);
signal i_ack : std_logic;
begin
  h1_Process : process(hreset,inc)
  begin
    if (hreset = '1') then
      v_addr <= "00000000";
    elsif ((inc'event) and (inc = '1') and (inc'last_value = '0')) then
      v_addr <= v_addr + "00000001";
  end process;
end huff_vals_addr_arch;

A-43
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_1164_extensions.all;

-- Written by LL_TO_VHDL at Tue May 3 09:08:37 1994
-- Parameterized Generator Specification to VHDL Code
--
-- LogicLib generator called: REGISTER_FILE
-- Passed Parameters are:
-- tinat name = huff_bits
-- parameters are:
-- W = 8
-- D = 256
-- read_ports = 1
--
-- huff_bits Entity Description
-- entity huff_bits is
-- port(
-- DIN1: in std_ulogic_vector(7 downto 0);
-- RADR1: in std_ulogic_vector(7 downto 0);
-- WADR1: in std_ulogic_vector(7 downto 0);
-- DOUT1: out std_ulogic_vector(7 downto 0);
-- LD1, WE1: in std_ulogic
-- );
-- end huff_bits;

component: huff_vals
description: (entity)

library IEEE;
use IEEE.std_logic_1164.all; 
use IEEE.std_logic_1164_extensions.all;

-- Name: huff_lencntr_entity
-- Purpose: huffman data length counter module
-- Author: Douglas A. Carpenter
-- Created: 05-MAY-1994
-- Revised:

-- library and use clauses
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164_extensions.all;
use std.textio.all;
library my_packages;
use my_packages.package_1.all;

entity huff_lencntr is
port ( -- inputs
set_data : in std_logic_vector(15 downto 0);
dec : in std_logic;
set : in std_logic;
 clock : in std_logic;
-- outputs
hlength : out std_logic_vector(15 downto 0);
dec_ack : out std_logic);
end huff_lencntr;

component: huff_lencntr
description: (architecture)

architecture rtl of huff_lencntr is
subtype ramword is std_logic_vector (7 downto 0);
type rammemory is array (255 downto 0) of ramword;
signal ram : rammemory;
begin
REGISTER_FILE_read_Process: process(ram,RADR1)
variable raddr1 : integer range 0 to 255;
begin
-- Process the read port number 1
-- convert address to integer
raddr1 := to_integer('0' & RADR1,0);
DOUT1 <= ram(raddr1);
end process REGISTER_FILE_read_Process;
end rtl;
H1_Process : process(set,dec,set_data)
begin
  if (set = '1') then
    d_len <= set_data;
  elsif ((dec'event) and (dec = '1') and (dec'last_value = '0')) then
    d_len <= d_len "0000000000000001";
  end if;
end process H1_Process;

H2_Process : process(set,clock)
begin
  if (set = '1') then
    d_ack <= '0';
  elsif ((clock'event) and (clock = '1') and (clock'last_value = '0')) then
    if (dec = '1') then
      d_ack <= '1';
    else
      d_ack <= '0';
    end if;
  end if;
end process H2_Process;

-- assign output values
dec_ack <= d_ack;
hlength <= d_len;
end huff_lencntr_arch;

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_1164_extensions.all;

-- Written by LL_to_VHDL at Tue May 3 10:07:38 1994
-- Parameterized Generator Specification to VHDL Code
--
-- LogicLib generator called: MULTIPLEXER
-- Passed Parameters are:
--  tinst name = huff_acdcsel
--  parameters are:
--    type = SIMPLE
--    W = 8
--    numin = 2
--    SW = 1
--    bus_mask = 0
--    comp_out = NO
--
-- huff_acdcsel Entity Description
entity huff_acdcsel is
  port(
    IN0 : in std_ulogic_vector(7 downto 0);
    IN1 : in std_ulogic_vector(7 downto 0);
    SEL : in std_ulogic_vector(0 downto 0);
    DOUT : out std_ulogic_vector(7 downto 0)
  );
end huff_acdcsel;

component huff_acdcsel
description:
begin
  huff_acdcsel_Process: process(IN0,IN1,SEL)
  variable iaddress : integer range 0 to 1;
  variable state : std_ulogic_vector(7 downto 0);
  begin
    iaddress := to_Integer('0' & SEL,0);
    case iaddress is
      when 0 =>
        state := IN0;
      when 1 =>
        state := IN1;
      when others =>
        state := IN1;
    end case;
    -- Assign outputs
    DOUT <= state;
  end process huff_acdcsel_Process;
end rtl;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164_extensions.all;

library my_packages;

entity huff_len_loader is
  port (   -- external signals
    clock : in std_ulogic;
  -- internal signals
    hreset : in std_ulogic;
    hlen_req : out std_ulogic;
    hlen_ack : out std_ulogic;
    hlbv_err : out std_ulogic;
    next_req : out std_ulogic;
    next_ack : in std_ulogic;
    err : in std_ulogic;
    data : in std_ulogic_vector(7 downto 0);
  -- Huffman center signals
    set_data : out std_ulogic;
  -- Huffman decoder signals
    dec : out std_ulogic;
    dec_ack : in std_ulogic;
    dec_data : out std_ulogic_vector(15 downto 0);
    acdc_en : out std_ulogic;
    acdc_data : out std_ulogic_vector(0 downto 0);
  end huff_len_loader;

COMPONENT: huff_len_loader
DESCRIPTION: (architecture)
  --
  -- Name: huff_len_loader_arch
  -- Purpose:
  -- Author: Douglas A. Carpenter
  -- Created: 07-MAY-1994
  -- Revised:

architecture huff_len_loader_arch of huff_len_loader is
  signal D : std_ulogic_vector(4 downto 0);
signal Q : std_ulogic_vector(4 downto 0);
signal i_length : std_ulogic_vector(15 downto 0);
signal i_acdc : std_ulogic;
signal i_outs : std_ulogic_vector(7 downto 0);
begin
  hload_process: process (hreset, clock)
  begin
    if (hreset = '0') then
      i_length <= "0000000000000000";
      i_outs <= "00000000";
      Q <= "000000";
      elsif ((clock'event) and (clock = '1')) and (clock'last_value = '0')
      then
        if ((D = "00000") and (hlen_req = '0')) then
          Q <= "00000";
          i_outs <= "00000000";
          elsif ((D = "00000") and (hlen_req = '1'))
          and (next_ack = '1') and (err = '0') then
            Q <= "00000";
            i_outs <= "00000000";
        elsif ((D = "00000") and (hlen_req = '1'))
          and (next_ack = '0') and (err = '1') then
          Q <= "00000";
          i_outs <= "00000000";
          elsif ((D = "00001") and (next_ack = '0')
          and (err = '0')) then
            Q <= "00001";
            i_outs <= "00000000";
            elsif ((D = "00001") and (next_ack = '1')
            and (err = '0')) then
            Q <= "00001";
            i_outs <= "00000000";
            elsif ((D = "00010") and (next_ack = '0'))
            then
            Q <= "00010";
            i_outs <= "00000000";
            elsif ((D = "00010") and (next_ack = '1'))
            then
            Q <= "00010";
            i_outs <= "00000000";
            elsif ((D = "00100") and (next_ack = '1'))
            then
            Q <= "00100";
            i_outs <= "00000000";
            elsif ((D = "00100") and (next_ack = '0'))
            then
            Q <= "00100";
            i_outs <= "00000000";
            elsif ((D = "00101") and (next_ack = '0'))
            then
            Q <= "00101";
            i_outs <= "00000100";
            elsif ((D = "00101") and (next_ack = '1'))
            then
            Q <= "00101";
            i_outs <= "00000100";
            elsif ((D = "01000") and (next_ack = '1'))
            then
            Q <= "01000";
            i_outs <= "00000000";
            elsif ((D = "01000") and (next_ack = '0'))
            then
            Q <= "01000";
            i_outs <= "00000000";
            elsif ((D = "01001") and (next_ack = '0'))
            then
            Q <= "01001";
            i_outs <= "00000000";
            elsif ((D = "01001") and (next_ack = '1'))
            then
            Q <= "01001";
            i_outs <= "00000000";
          end if;
      end if;
  end process hload_process;
end architecture huff_len_loader_arch;
then
  Q <= "01100";
  i_outs <= "00000000";
elsif ((D = "01100") and (dec_ack = '1'))
then
  Q <= "01100";
  i_outs <= "00000000";
elsif ((D = "01100") and (dec_ack = '0'))
then
  if (hlength = "0000000000000000")
then
    Q <= "11111";
    i_outs <= "01000000";
else
    Q <= "01110";
    i_outs <= "00010000";
  end if;
elsif ((D = "01110") and (dec_ack = '0'))
then
  Q <= "01110";
  i_outs <= "00000000";
elsif ((D = "01101") and (dec_ack = '1'))
then
  Q <= "01101";
  i_outs <= "00000000";
elsif ((D = "01101") and (dec_ack = '0'))
then
  if (hlength = "0000000000000000")
then
    Q <= "11111";
    i_outs <= "01000000";
else
    Q <= "11111";
    i_outs <= "10000000";
end if;
elsif ((D = "11111") and (hlen_req = '1'))
then
  Q <= "11111";
  i_outs <= "01000000";
elsif ((D = "11111") and (hlen_req = '0'))
then
  Q <= "00000";
  i_outs <= "00000000";
end if;
end if;

end process hload_process;

-- assign output values
D <= Q;
hlen_ack <= i_outs(7);
hlbv_err <= i_outs(6);
dec <= i_outs(4);
set <= i_outs(3);
acdc_en <= i_outs(1);
nexr_req <= i_outs(0);
set_data <= i_length;
acdc_data(0) <= i_acdc;

end huff_len_loader_arch;

COMPONENT: huff_bits_loader
DESCRIPTION: (entity)

-- Name: huff_bits_loader_entity
-- Purpose:
-- Author: Douglas A. Carpenter
-- Created: 07-MAY-1994
-- Revised:

library and use clauses
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164_extensions.all;

library my_packages;

entity huff_bits_loader is
  port (-- external signals
    clock : in std_ulogic;
    -- internal signals
    hreset : in std_ulogic;

  );
begin
if (hreset = '0') then
  Q <= "0000";
  i_outs <= "0000000000";
  i_counter <= "1111";
  i_data <= "0000000000";
elsif ((hbits_req = '1') and (next_ack = '1') and (err = '0')) then
  Q <= "1101";
  i_outs <= "0000000000";
elsif ((hbits_req = '1') and (next_ack = '1') and (err = '1')) then
  Q <= "1101";
  i_outs <= "0000000000";
else
  Q <= "1101";
  i_outs <= "0000000000";
end if;
send if;
end if;
end process
end architecture

end component

COMPONENT: huff_bits_loader
DESCRIPTION: (entity)
huffvalsloader is
entity my_packages;
library ieee;
end entity;

architecture huffvals_loader_arch of huffvals_loader is
signal D    : std_logic_vector(4 downto 0);
signal Q    : std_logic_vector(4 downto 0);
signal i_outs : std_logic_vector(13 downto 0);
signal i_vals_data : std_logic_vector(7 downto 0);
signal i_valaddr : std_logic_vector(7 downto 0);
begin
  hload_process : process(hreset, clock)
  variable bits_data_counter : std_logic_vector(1 downto 0);
  begin
    if (hreset = '0') then
      Q <= "00000";
    elsif (hreset = '1') then
      Q <= "00000";
    end if;
    bits_DINl <= i_data;
    hbits_ack <= i_outs(9);
    hlbv_err <= i_outs(8);
    dec <= i_outs(7);
    bits_reset <= i_outs(6);
    bits_inc <= i_outs(5);
    bits_LDl <= i_outs(3);
    bits_WE1 <= i_outs(2);
    next_req <= i_outs(0);
  end process hload_process;

-- assign output values
D <= Q;
hbits_ack <= i_outs(9);
hlbv_err <= i_outs(8);
dec <= i_outs(7);
bits_reset <= i_outs(6);
bits_inc <= i_outs(5);
bits_LDl <= i_outs(3);
bits_WE1 <= i_outs(2);
next_req <= i_outs(0);
end architecture;

component huff_vals_loader is
begin
end component;

description:

-- Name: huff_vals_loader
-- Purpose: DOUGLAS A. CARPENTER
-- Created: 07-MAY-1994
-- Revised: 07-MAY-1994

-- library and use clauses
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164_extensions.all;
library my_packages;
entity huff_vals_loader is
port (-- external signals
clock : in std_ulogic;

-- internal signals
hreset : in std_ulogic;
hvals_req : in std_ulogic;
hvals_ack : std_logic;
hlbv_err : std_ulogic;
next_req : out std_ulogic;
next_ack : in std_ulogic;
err : in std_ulogic;
data : in

std_logic_vector(7 downto 0);

std_logic_vector(15 downto 0);

dec : out std_ulogic;
dec_ack : in std_ulogic;
vb_reset : out std_ulogic;
bits_inc : out std_ulogic;
bits_inc_ack : in std_ulogic;
vals_inc : out std_ulogic;
vals_inc_ack : in std_ulogic;
vals_DINI : out

std_logic_vector(7 downto 0);

vals_LDl : out std_ulogic;
vals_WE1 : out std_ulogic;
bits_addr : in

std_logic_vector(3 downto 0);

bits_data : in

std_logic_vector(7 downto 0);

val_DINI : out

std_logic_vector(7 downto 0);

val_RWADR : out

std_logic_vector(7 downto 0);

val_LDl : out std_ulogic;
DCval_WE1 : out std_ulogic;
ACval_WE1 : out std_ulogic;
ACDC_data_out : in

std_logic_vector(0 downto 0);
)

end component;
i_outs <= "00000000000000";
bits_data_counter := "00000000"

elsif ((clock'sevent) and (clock = '1') and
(clock'last_value = '0')) then
if (D = "00000") then
if (hvals_req = '0') then
Q <= "00000";
i_outs <= "00000000000000"
bits_data_counter := "00000000"
i_valaddr <= "00000000";
elsif ((hvals_req = '1') and (next_ack = '1') and (err = '0')) then
Q <= "00000";
i_outs <= "00000000000000";
elsif ((hvals_req = '1') and (err = '1')) then
then
Q <= "11111";
i_outs <= "00000000000000"
elsif (D = "00001") then
Q <= "00111";
i_outs <= "00000000000000"
elsif (D = "00011") then
Q <= "00010";
i_outs <= "00000000000010"
bits_data_counter := bits_data;
elsif (D = "00100") then
if (bits_inc_ack = '0') then
Q <= "00100"
i_outs <= "00000000000001";
elsif (bits_inc_ack = '1') then
Q <= "00100"
i_outs <= "00000000000000";
elsif (bits_inc_ack = '0') then
Q <= "01011"
i_outs <= "00000000000000"
elsif (bits_inc_ack = '1') then
Q <= "01010"
i_outs <= "00000000000000"
end if;
elsif (D = "00110") then
if (bits_data_counter = "00000000") then
Q <= "00110"
i_outs <= "00000001000000"
else
Q <= "00101"
i_outs <= "00000000000001";
end if;
elsif (D = "00110") then
if (bits_inc_ack = '0') then
Q <= "00110"
i_outs <= "00000000000000";
elsif (bits_inc_ack = '1') then
Q <= "01000"
i_outs <= "00000000000000";
elsif (bits_inc_ack = '0') then
Q <= "01011"
i_outs <= "00000000000000"
elsif (D = "11111") then
if ((next_ack = '0') and (err = '0')) then
Q <= "00101"
i_outs <= "00000000000001";
elif (err = '1') then
Q <= "11111";
i_outs <= "00010000000000"
elsif (clock'last_value = '0')) then
Q <= "00011"
i_outs <= "00000000000000";
end if;
elsif (D = "00000") then
if (hvals_req = '1') then
Q <= "00111"
i_outs <= "00000000000000";
elsif (hvals_req = '0') then
Q <= "01111"
i_outs <= "00010000000000";
elsif (hvals_req = '1') then
Q <= "01000"
i_outs <= "00000000000000";
elsif (hvals_req = '0') then
Q <= "00000"
i_outs <= "00000000000000";
end if;
end if;
else
Q <= "00000";
i_outs <= "00000000000000";
end if;
end process hload_process;

-- assign output values
D <= Q;
hvals_ack <= i_outs(11);
hlbv_err <= i_outs(10);
dec <= i_outs(9);
vb_reset <= i_outs(8);
bits_inc <= i_outs(7);
vals_inc <= i_outs(5);
vals_DINl <= i_vals_data;
vals_LDl <= i_outs(3);
vals_WE1 <= i_outs(2);
next_req <= i_outs(0);
val_DINl <= i_vals_data;
val_LDl <= i_outs(3);
ACval_WE1 <= i_outs(13);
end huff_vals_loader_arch;

begin
if (hreset = '0') then
i_outs <= "0000000";
Q <= "0000";
i <= "00001";
j <= "00000001";
k <= "00000000";
elsif (hsiz_req = '1') then
Q <= "0001";
i_outs <= "0000001";
elsif (D = "0001") then
Q <= "0011";
i_outs <= "00000001";
elsif (D = "0011") then
Q <= "0010";
i_outs <= "00000000";
elsif (D = "0010") then
Q <= "1010";
i_outs <= "0000000";
elif (D = "0001") then
Q <= "00101";
i_outs <= "0000000";
elif (D = "0000") then
Q <= "00001";
i_outs <= "0000001";
elif (hsiz_req = '1') then
Q <= "0001";
i_outs <= "00000001";
end if;
elif (D = "0001") then
Q <= "0011";
i_outs <= "00000001";
elif (D = "0011") then
Q <= "00101";
i_outs <= "0000000";
elif (D = "0010") then
Q <= "1010";
i_outs <= "00001010";
i_data <= "0000000";
else
Q <= "0110";
end if;
end if;
end process hload_process;

COMPONENT: huff_load_size
DESCRIPTION: (architecture)

-- Name : huff_load_size_arch

-- Purpose :
-- Author : Douglas A. Carpenter
-- Created : 16-MAY-1994
-- Revised :

architecture huff_load_size_arch of huff_load_size is

begin
if (hreset = '0') then
i_outs <= "0000000";
Q <= "0000";
i <= "00001";
j <= "00000001";
k <= "00000000";
elsif (hsiz_req = '1') then
Q <= "0001";
i_outs <= "0000001";
elsif (D = "0001") then
Q <= "0011";
i_outs <= "00000001";
elsif (D = "0011") then
Q <= "0010";
i_outs <= "00000000";
elsif (D = "0010") then
Q <= "1010";
i_outs <= "00001010";
i_data <= "0000000";
else
Q <= "0110";
end if;
end if;
end process hload_process;

COMPONENT: huff_load_size
DESCRIPTION: (entity)

-- Name : huff_load_size_entity

-- Purpose :
-- Author : Douglas A. Carpenter
-- Created : 16-MAY-1994
-- Revised :

-- library and use clauses
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164_extensions.all;

library my_packages;

entity huff_load_size is
port (
-- external signals
  clock : in std_logic;

-- internal signals
  hreset : in std_logic;
  hsiz_req : in std_logic;
  hsiz_ack : out

  std_ulogic:
    bits_addr_reset : out
    bits_inc : out
    bits_inc_ack in std_ulogic;
    bits_data_out in std_ulogic_vector(7 downto 0);
  std_ulogic:
    hsize_LDl : out
    hsize_WE1 : out
    hsize_addr : out
    hsize_data : out
    hsize_data : out
    hsize_vector(7 downto 0);

end entity huff_load_size;

end
DESCRIPTION:
COMPONENT;  huff_load_code

end

-- Name : huff_load_code_entity
-- Purpose :
-- Author : Douglas A. Carpenter
-- Created : 16-MAY-1994
-- Revised :

-- library and use clauses
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164_extensions.all;

library my_packages;

entity huff_load_code is
port [ ]
end;

-- external signals
clock : in std_ulogic;

-- internal signals
hreset in std_ulogic;
hcode_req : in std_ulogic;
hcode_ack : out std_ulogic;
hscg_err : out std_ulogic;

hsiz_ADRl : out std_ulogic_vector(7 downto 0);
hsiz_DOUTl : in std_ulogic_vector(7 downto 0);

hcode_ADRl out std_ulogic_vector(7 downto 0);

hsiz_LDpl : out std_ulogic;
hsiz_WEpl : out std_ulogic;
hsiz_DINpl : out std_ulogic_vector(15 downto 0);
hsiz_SEL , out std_ulogic_vector(0 downto 0);
end huff_load_code;

COMPONENT:  huff_load_code
DESCRIPTION:  (architecture)

-- Name : huff_load_code_arch
-- Purpose :
-- Author : Douglas A. Carpenter
-- Created : 16-MAY-1994
-- Revised :

architecture huff_load_code_arch of huff_load_code is
signal D : std_ulogic_vector(3 downto 0);
signal Q : std_ulogic_vector(3 downto 0);
signal i_outs : std_ulogic_vector(3 downto 0);

end process hload_process;

-- assign output values
D <= Q;
hsize_addr <= k;
hsize_data <= i_data;

hsiz_ack <= i_outs(6);
hscg_err <= i_outs(5);
bits_inc <= i_outs(4);
hsize_LDpl <= i_outs(3);
hsiz_WEpl <= i_outs(2);
bias_addr_reset <= i_outs(0);

end huff_load_size_arch;
signal SI : std_ulogic_vector(7 downto 0);
signal k : std_ulogic_vector(7 downto 0);
signal CODE : std_ulogic_vector(15 downto 0);
signal i_data : std_ulogic_vector(15 downto 0);
signal i_sel : std_ulogic_vector(0 downto 0);

begin

hloadc_process : process(hreset, clock)
begin
if (hreset = '0') then
  i_outs <= "0000";
  Q <= "0000";
  SI <= "00000000";
  k <= "00000000";
  CODE <= "0000000000000000";
  i_data <= "0000000000000000";
  i_sel <= "0";
elsif ((clock'event) and (clock = '1')) and (clock'last_value = '0') then
  if (D = "0000") then
    if (hcode_req = '0') then
      Q <= "0000";
      i_outs <= "0000";
      i_sel <= "0";
      SI <= "00000000";
      k <= "00000000";
      CODE <= "0000000000000000";
      i_data <= "0000000000000000";
      elsif (hcode_req = '1') then
        Q <= "0001";
        i_outs <= "0000";
        i_sel <= "1";
        SI <= hsize_DOUT1;
      end if;
    elsif (D = "0001") then
      Q <= "0111";
      i_outs <= "0010";
      i_sel <= "1";
      i_data <= CODE;
    elsif (D = "0011") then
      Q <= "0110";
      i_outs <= "0000";
      i_sel <= "1";
    elsif (D = "0101") then
      Q <= "0100";
      i_outs <= "0000";
      i_sel <= "1";
      CODE <= CODE + "0000000000000000";
      k <= k + "00000000";
    elsif (D = "0100") then
      if (hsize_DOUT1) then
        Q <= "0001";
        i_outs <= "0000";
        i_sel <= "1";
      else
        Q <= "0101";
        i_outs <= "0000";
        i_sel <= "1";
      end if;
    elsif (D = "0101") then
      if (hsize_DOUT1 = "00000000") then
        Q <= "1010";
        i_outs <= "1000";
        i_sel <= "1";
      else
        Q <= "0111";
        i_outs <= "0000";
        i_sel <= "1";
        CODE <= CODE(14 downto 0) & "0";
      end if;
    else
      if (SI = hsize_DOUT1) then
        Q <= "0000";
        i_outs <= "0000";
        i_sel <= "1";
        CODE <= CODE(14 downto 0) & "0";
      end if;
    end if; end process hloadc_process;

-- assign output values
D <= Q;
hcode_ack <= i_outs(3);
hscan_err <= i_outs(2);
hcode_LD1 <= i_outs(1);
hcode_WE1 <= i_outs(0);
hcode_DIN1 <= i_data;
hcodeADR1 <= k;
hsizeADR1 <= k;
hsizeSEL <= i_sel;
end huff_load_code_arch;

COMPONENT: huff_reg256by8
DESCRIPTION: (entity)

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_1164_extensions.all;

-- Written by LL_to_VHDL at Tue May 3 09:08:37 1994
-- Parameterized Generator Specification to VHDL Code
--
-- LogicLib generator called: REGISTER_FILE
-- Passed Parameters are:
-- tinst name = huff_reg256by8
-- parameters are:
-- W = 8
-- D = 256
-- read ports = 1

-- huff_reg256by8 Entity Description
entity huff_reg256by8 is
port(
  DIN1: in std_ulogic_vector(7 downto 0);
  RADR1: in std_ulogic_vector(7 downto 0);
  WADR1: in std_ulogic_vector(7 downto 0);
  DOUT1: out std_ulogic_vector(7 downto 0);
  LD1, WE1: in std_ulogic
);
end huff_reg256by8;

COMPONENT: huff_reg256by8
DESCRIPTION: (architecture)

-- huff_reg256by8 Architecture Description
architecture rtl of huff_reg256by8 is
  subtype ramword is std_ulogic_vector (7 downto 0);
  type rammemory is array (255 downto 0) of ramword;
  signal ram : rammemory;
begin
  REGISTER_FILE_read_Process: process(ram, RADR1)
    variable raddr1 : integer range 0 to 255;
  begin
    -- Process the read port number 1
    -- convert address to integer
    raddr1 := to_Integer('0' & RADR1, 0);
    DOUT1 <= ram(raddr1);
  end process REGISTER_FILE_read_Process;
  REGISTER_FILE_write_Process:
  process(DIN1, WE1, LD1, WADR1)
    variable waddr : integer range 0 to 255;
    variable load : std_ulogic;
  begin
    -- write mode? (need both WE1 and LD1 to be high)
    -- convert address to integer
    waddr := to_Integer('0' & WADR1, 0);
    load := LD1 and WE1;
    if (load = '1') then
      ram(waddr) <= DIN1;
    end if;
  end process REGISTER_FILE_write_Process;
end rtl;

COMPONENT: huff_reg256by16
DESCRIPTION: (entity)

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_1164_extensions.all;

-- Written by LL_to_VHDL at Tue May 3 09:08:37 1994
-- Parameterized Generator Specification to VHDL Code
--
-- LogicLib generator called: REGISTER_FILE
-- Passed Parameters are:
--  tinst name = huff_reg256by16
--  parameters are:

-- W = 8
-- D = 256
-- read ports = 1

-- huff_reg256by16 Entity Description
entity huff_reg256by16 is
port(
  DIN1: in std_ulogic_vector(15 downto 0);
  RADR1: in std_ulogic_vector(7 downto 0);
  WADR1: in std_ulogic_vector(7 downto 0);
  DOUT1: out std_ulogic_vector(15 downto 0);
  LD1, WE1: in std_ulogic
);
end huff_reg256by16;

COMPONENT: huff_reg256by16
DESCRIPTION: (architecture)

-- huff_reg256by16 Architecture Description
architecture rtl of huff_reg256by16 is
  subtype ramword is std_ulogic_vector (15 downto 0);
  type rammemory is array (255 downto 0) of ramword;
  signal ram : rammemory;
begin
  REGISTER_FILE_read_Process: process(ram, RADR1)
    variable raddr1 : integer range 0 to 255;
  begin
    -- Process the read port number 1
    -- convert address to integer
    raddr1 := to_Integer('0' & RADR1, 0);
    DOUT1 <= ram(raddr1);
  end process REGISTER_FILE_read_Process;
  REGISTER_FILE_write_Process:
  process(DIN1, WE1, LD1, WADR1)
    variable waddr : integer range 0 to 255;
    variable load : std_ulogic;
  begin
    -- write mode? (need both WE1 and LD1 to be high)
    -- convert address to integer
    waddr := to_Integer('0' & WADR1, 0);
    load := LD1 and WE1;
    if (load = '1') then
      ram(waddr) <= DIN1;
    end if;
  end process REGISTER_FILE_write_Process;
end rtl;

COMPONENT: huff_gen
DESCRIPTION: (entity)

-- Name : huff_gen_entity
-- Purpose :
-- Author : Douglas A. Carpenter
-- Created : 16-MAY-1994
-- Revised :

-- library and use clauses
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164_extensions.all;
library my_packages;

entity huff_gen is
  port (
-- external signals
  clock : in std_ullogic;
-- internal signals
  hreset : in std_ullogic;
  hgen_req : in std_ullogic;
  hgen_ack : out std_ullogic;
  hscq_err : out std_ullogic;
  DMin_WE1 : out std_ullogic;
  DMax_WE1 : out std_ullogic;
  DValpr1_WE1 : out std_ullogic;
  AMin_WE1 : out std_ullogic;
  AValpr1_WE1 : out std_ullogic;
  huff_DIN : out std_ullogic;

  std_ullogic_vector(15 downto 0): huff_WADR;
  std_ullogic_vector(3 downto 0):
    huff_LD1: out std_ullogic;
    bits_reset: out std_ullogic;
    bits_inc: out std_ullogic;
    bits_inc_ack: in std_ullogic;
    bits_addr: in std_ullogic;

  std_ullogic_vector(7 downto 0):
    hcode_ADDR: out std_ullogic;
    hcode_DOUT: in std_ullogic;

  std_ullogic_vector(15 downto 0):
    hcode_DOUT1: in std_ullogic;

  std_ullogic_vector(0 downto 0):
    acdc_data_out: in std_ullogic;

end huff_gen;

COMPONENT:  huff_gen
DESCRIPTION: (architecture)

-- Name : huff_gen_arch
-- Purpose : Douglas A. Carpenter
-- Created : 16-MAY-1994
-- Revised :

architecture huff_gen_arch of huff_gen is
  signal D : std_ullogic_vector(4 downto 0);
  signal Q : std_ullogic_vector(4 downto 0);
  signal i_outs : std_ullogic_vector(14 downto 0);
  signal i : std_ullogic_vector(3 downto 0);
  signal j : std_ullogic_vector(7 downto 0);
  signal i_data : std_ullogic_vector(15 downto 0);

begin

  hloadg_process : process(hreset, clock)
  begin
    if (hreset = '0') then
      i_outs <= "0000000000000000";
      Q <= "0000";
      j <= "00000000";
      i_data <= "0000000000000000";
    elsif (D = "0000") then
      if (hgen_req = '0') then
        i_outs <= "0000000000000000";
        Q <= "0000";
        j <= "00000000";
      else
        Q <= "0000100000000000";
        i_outs <= "0001000000000000";
      end if;
    elsif (D = "0001") then
      Q <= "0000110000000000";
      i_outs <= "0000000000000000";
      i_data <= "0000000000000000";
    elsif (D = "0010") then
      if (bits_data = "00000000") then
        Q <= "11111";
        i_outs <= "0000100000000000";
        i_data <= "1111111111111111";
      else
        Q <= "0010";
        i_outs <= "0000010000000000";
        i_data <= "00000000" & j;
      end if;
    elsif (D = "0011") then
      if (acdc_data_out = "0") then
        Q <= "01100";
        i_outs <= "0000100000000000";
        else
        Q <= "01100";
        i_outs <= "0000100000000000";
      end if;
    elsif (D = "0101") then
      Q <= "011011";
      if (acdc_data_out = "0") then
        i_outs <= "0000100000000000";
      else
        Q <= "0000100000000000" & j;
      end if;
    elsif (D = "0111") then
      if (acdc_data_out = "0") then
        i_outs <= "0000100000000000";
      else
        Q <= "0000100000000000" & j;
      end if;
    elsif (D = "1111") then
      Q <= "1111111111111111";
      if (acdc_data_out = "0") then
        i_outs <= "0000100000000000";
      else
        Q <= "0000100000000000" & j;
      end if;
  end huff_gen;
i_outs <= "0001000000000000";
else
  Q <= "01001";
  i_outs <= "0000100000000000";
end if;
elsif (D = "01001") then
  if (i = "0000") then
    Q <= "01000";
    i_outs <= "1000000000000000";
  else
    Q <= "00010";
    i_outs <= "0000100000000000";
  end if;
elsif (D = "01000") then
  if (hgen_req = '1') then
    Q <= "01000";
    i_outs <= "0000000000000000";
  else
    Q <= "00000";
    i_outs <= "0000000000000000";
  end if;
elsif (D = "11111") then
  if (acdc_data_out = "0") then
    i_outs <= "0000100000000001";
  else
    i_outs <= "00001000001101";
  end if;
elsif (D = "11110") then
  if (acdc_data_out = "0") then
    i_outs <= "0001000000000001";
  else
    i_outs <= "0000000000000000";
  end if;
else
  Q <= "00000";
  i_outs <= "0000000000000000";
end if;
end process hloadg_process;

-- assign output values
D <= Q;
huff_ack <= i_outs(14);
hscg_err <= i_outs(13);
bits_reset <= i_outs(12);
bits_inc <= i_outs(11);
hcode_SEL(0) <= i_outs(10);
DCmin_WEl <= i_outs(9);
DCmax_WEl <= i_outs(8);
DCvalptr_WEl <= i_outs(7);
ACmin_WEl <= i_outs(5);
ACmax_WEl <= i_outs(4);
ACvalptr_WEl <= i_outs(3);
huff_LD1 <= i_outs(0);
hcodeADR1 <= j;
huff_WADR <= i;
huff_DIN <= i_data;
end huff_gen_arch;

COMPONENT: define_qtable
DESCRIPTION: (entity)

-- Name: define_qtable_entity
-- Purpose:
-- Author: Douglas A. Carpenter
-- Created: 18-APR-1994
-- Revised:

-- library and use clauses
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164_extensions.all;
library my_packages;
use my_packages.package_1.all;

entity define_qtable is
port (clock : in std_logic;
  reset : in std_logic;
  table_err : out std_logic;
  qtable_req : in std_logic;
  qtable_ack : out std_logic;
  inc : out std_logic;
  table_addr : in std_logic_vector(5 downto 0);
  inc_ack : in std_logic;
  data : in std_logic_vector(7 downto 0);
  qload : out std_logic;
  qload_ack : in std_logic)
);
end define_qtable;

COMPONENT: define_qtable
DESCRIPTION: (architecture)

--
-- Name: define_qtable_arch
--
-- Purpose: define quant table module
-- Author: Douglas A. Carpenter
-- Created: 18-APR-1994
-- Revised: 
architecture define_qtable_arch of define_qtable is
signal D : std_logic_vector(4 downto 0);
signal a : std_logic_vector(59 downto 0);
signal i_data : std_logic_vector(7 downto 0);
begin
  sof_process : process(reset, clock)
  begin
    if (reset = '0') then
      i_data <= "00000000";
    end if;
end process;
a(48) <= ('1') when ((D(3) = '0') and (D(2) = '0') and (D(1) = '1') and (D(0) = '0')) and (a(49) = '1') or (a(50) = '1') or (a(51) = '1') or (a(52) = '1');

if ((D(3) = '1') and (D(2) = '0') and (D(1) = '0') and (D(0) = '1')) then
  i_data <= data;
else
  i_data <= i_data;
end if;

D4 <= (a(4) = '1') or (a(20) = '1') or (a(29) = '1') or (a(31) = '1') or (a(32) = '1') or (a(34) = '1') or (a(35) = '1') or (a(36) = '1') or (a(37) = '1') or (a(38) = '1') or (a(39) = '1') or (a(40) = '1') or (a(41) = '1') or (a(42) = '1') or (a(43) = '1') or (a(44) = '1') or (a(45) = '1') or (a(46) = '1') or (a(47) = '1') or (a(48) = '1') or (a(50) = '1');

D3 <= (a(3) = '1') or (a(6) = '1') or (a(9) = '1') or (a(10) = '1') or (a(11) = '1') or (a(12) = '1') or (a(13) = '1') or (a(14) = '1') or (a(15) = '1') or (a(16) = '1') or (a(17) = '1') or (a(18) = '1') or (a(19) = '1') or (a(20) = '1') or (a(21) = '1') or (a(22) = '1') or (a(23) = '1') or (a(24) = '1') or (a(25) = '1') or (a(26) = '1') or (a(27) = '1') or (a(28) = '1');

D2 <= ((a(1) = '1') or (a(9) = '1') or (a(10) = '1') or (a(11) = '1') or (a(12) = '1') or (a(13) = '1') or (a(14) = '1') or (a(20) = '1') or (a(21) = '1') or (a(22) = '1') or (a(23) = '1') or (a(24) = '1') or (a(25) = '1') or (a(26) = '1') or (a(27) = '1');

}$

D1 <= (a(1) = '1') or (a(8) = '1') or (a(9) = '1') or (a(10) = '1') or (a(11) = '1') or (a(12) = '1') or (a(13) = '1') or (a(14) = '1');

D0 <= (a(5) = '1') or (a(7) = '1') or (a(16) = '1') or (a(17) = '1') or (a(18) = '1') or (a(19) = '1') or (a(30) = '1') or (a(31) = '1') or (a(32) = '1') or (a(33) = '1') or (a(34) = '1') or (a(35) = '1') or (a(36) = '1') or (a(37) = '1') or (a(38) = '1') or (a(46) = '1') or (a(47) = '1') or (a(48) = '1');

$qdata_in <= i_data;

table_err <= '1' when ((a(31) = '1') or (a(32) = '1') or (a(34) = '1') or (a(37) = '1') or (a(41) = '1') or (a(42) = '1') or (a(43) = '1') or (a(44) = '1') or (a(45) = '1') or (a(46) = '1') or (a(47) = '1') or (a(48) = '1') or (a(50) = '1');

$qtable_ack <= '1' when ((a(4) = '1') or (a(29) = '1') or (a(49) = '1')) else '0';

$qreset <= '0' when ((a(0) = '1') or (a(9) = '1') or (a(10) = '1') or (a(11) = '1') or (a(12) = '1') or (a(13) = '1') or (a(14) = '1') or (a(15) = '1') or (a(16) = '1') or (a(17) = '1') or (a(18) = '1') or (a(19) = '1') or (a(20) = '1') or (a(21) = '1') or (a(22) = '1') or (a(23) = '1') or (a(24) = '1') or (a(25) = '1') or (a(26) = '1') or (a(27) = '1');

$inc <= '1' when ((a(20) = '1') or (a(40) = '1')) else '0';

$next_req <= '1' when ((a(1) = '1') or (a(2) = '1') or (a(3) = '1') or (a(5) = '1') or (a(6) = '1') or (a(7) = '1') or (a(8) = '1') or (a(24) = '1') or (a(25) = '1') or (a(26) = '1') or (a(27) = '1') or (a(28) = '1')) else '0';
qload <= '1' when (a(22) = '1') else '0';

end define_qtable_arch;

COMPONENT: qaddr
DESCRIPTION: (entity)

--
-- Name : qaddr_entity
--
-- Purpose: quant table addr counter module
-- Author  : Douglas A. Carpenter
-- Created: 19-APR-1994
-- Revised:

-- library and use clauses
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164_extensions.all;
use std.textio.all;
library my_packages;
use my_packages.package_l.all;
entity qaddr is
  port (  
    qreset     : in  std_ulogic;
    inc        : in  std_ulogic;
    clock      : in  std_ulogic;
    -- outputs
    qtable_addr : out std_ulogic_vector(5 downto 0);  
    inc_ack     : out std_ulogic);
end qaddr;

COMPONENT: qaddr
DESCRIPTION: (architecture)

--
-- Name : qaddr_arch
--
-- Purpose: qaddr module
-- Author  : Douglas A. Carpenter
-- Created: 19-APR-1994 DAC
-- Revised:

architecture qaddr_arch of qaddr is
  signal i_addr : std_ulogic_vector (5 downto 0);
  signal i_ack  : std_ulogic;
begin

  Q1_Process : process(qreset,inc)
  begin
    if (qreset = '0') then
      i_addr <= "000000";
    elsif ((inc'event) and (inc = '1') and
      (inc'last_value = '0')) then
      i_addr <= i_addr + "000001";
    end if;
  end process Q1_Process;

  Q2_Process : process(qreset,clock)
  begin
    if (qreset = '0') then
      i_ack <= '0';
    elsif ((clock'event) and (clock = '1') and
      (clock'last_value = '0')) then
      i_ack <= '1';
    end if;
  end process Q2_Process;

  -- assign output values
  inc_ack <= i_ack;
  qtable_addr <= i_addr;
end qaddr_arch;

COMPONENT: qmem
DESCRIPTION: (entity)

--
-- Name : qmem_entity
--
-- Purpose:
-- Author  : Douglas A. Carpenter
-- Created: 18-APR-1994
-- Revised:

-- library and use clauses
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164_extensions.all;
use std.textio.all;
library my_packages;
use my_packages.package_l.all;
entity qmem is
  port (  
    clock  . in  std_ulogic;
    reset  . in  std_ulogic;
    qdata_in : in std_ulogic_vector(7 downto 0);
    qload    : in std_ulogic;
    qload_ack : out std_ulogic;
    qdata_out : out std_ulogic_vector(7 downto 0);
    qreq     : in std_ulogic;
    qreq_ack : out std_ulogic;
    qtable_addr : in std_ulogic_vector(5 downto 0);
    qtable_raddr : in std_ulogic_vector(5 downto 0)
  );
end qmem;

COMPONENT: qmem
DESCRIPTION: (architecture)

--
-- Name : qmem_arch
--
-- Purpose:
-- Author  : Douglas A. Carpenter
-- Created: 21-APR-1994
-- Revised:

architecture qmem_arch of qmem is
  signal q0 : std_ulogic_vector(7 downto 0);
  signal q1 : std_ulogic_vector(7 downto 0);
  signal q2 : std_ulogic_vector(7 downto 0);
  signal q3 : std_ulogic_vector(7 downto 0);
  signal q4 : std_ulogic_vector(7 downto 0);
  signal q5 : std_ulogic_vector(7 downto 0);
  signal q6 : std_ulogic_vector(7 downto 0);
  signal q7 : std_ulogic_vector(7 downto 0);
  signal q8 : std_ulogic_vector(7 downto 0);
  signal q9 : std_ulogic_vector(7 downto 0);
end qmem_arch;
signal q10 : std_logic_vector(7 downto 0);
signal q11 : std_logic_vector(7 downto 0);
signal q12 : std_logic_vector(7 downto 0);
signal q13 : std_logic_vector(7 downto 0);
signal q14 : std_logic_vector(7 downto 0);
signal q15 : std_logic_vector(7 downto 0);
signal q16 : std_logic_vector(7 downto 0);
signal q17 : std_logic_vector(7 downto 0);
signal q18 : std_logic_vector(7 downto 0);
signal q19 : std_logic_vector(7 downto 0);
signal q20 : std_logic_vector(7 downto 0);
signal q21 : std_logic_vector(7 downto 0);
signal q22 : std_logic_vector(7 downto 0);
signal q23 : std_logic_vector(7 downto 0);
signal q24 : std_logic_vector(7 downto 0);
signal q25 : std_logic_vector(7 downto 0);
signal q26 : std_logic_vector(7 downto 0);
signal q27 : std_logic_vector(7 downto 0);
signal q28 : std_logic_vector(7 downto 0);
signal q29 : std_logic_vector(7 downto 0);
signal q30 : std_logic_vector(7 downto 0);
signal q31 : std_logic_vector(7 downto 0);
signal q32 : std_logic_vector(7 downto 0);
signal q33 : std_logic_vector(7 downto 0);
signal q34 : std_logic_vector(7 downto 0);
signal q35 : std_logic_vector(7 downto 0);
signal q36 : std_logic_vector(7 downto 0);
signal q37 : std_logic_vector(7 downto 0);
signal q38 : std_logic_vector(7 downto 0);
signal q39 : std_logic_vector(7 downto 0);
signal q40 : std_logic_vector(7 downto 0);
signal q41 : std_logic_vector(7 downto 0);
signal q42 : std_logic_vector(7 downto 0);
signal q43 : std_logic_vector(7 downto 0);
signal q44 : std_logic_vector(7 downto 0);
signal q45 : std_logic_vector(7 downto 0);
signal q46 : std_logic_vector(7 downto 0);
signal q47 : std_logic_vector(7 downto 0);
signal q48 : std_logic_vector(7 downto 0);
signal q49 : std_logic_vector(7 downto 0);
signal q50 : std_logic_vector(7 downto 0);
signal q51 : std_logic_vector(7 downto 0);
signal q52 : std_logic_vector(7 downto 0);
signal q53 : std_logic_vector(7 downto 0);
signal q54 : std_logic_vector(7 downto 0);
signal q55 : std_logic_vector(7 downto 0);
signal q56 : std_logic_vector(7 downto 0);
signal q57 : std_logic_vector(7 downto 0);
signal q58 : std_logic_vector(7 downto 0);
signal q59 : std_logic_vector(7 downto 0);
signal q60 : std_logic_vector(7 downto 0);
signal q61 : std_logic_vector(7 downto 0);
signal q62 : std_logic_vector(7 downto 0);
signal q63 : std_logic_vector(7 downto 0);
signal a : std_logic_vector(5 downto 0);
signal D : std_logic_vector(2 downto 0);
signal out_data : std_logic_vector(7 downto 0);

begin

Qmem : process(reset, clock)
begin
if (reset = '0') then
if (reset = '0') then
D <= "000";
d <= "000000";
g0 <= To_StdUlogicVector(8,8);
g1 <= To_StdUlogicVector(6,8);
g2 <= To_StdUlogicVector(8,8);
g3 <= To_StdUlogicVector(7,8);
g4 <= To_StdUlogicVector(6,8);
g5 <= To_StdUlogicVector(5,8);
g6 <= To_StdUlogicVector(8,8);
elseif (clock'event and (clock = '1') and
(last_value = '0')) then
a(0) <= ([D(2) = '0'] and (D(1) = '0')) and
(D(0) = '0') and (qload = '1') and (qreq = '0'))
and (D(0) = '0') and (qload = '1')
and (qreq = '0'))
a(1) <= ([D(2) = '0'] and (D(1) = '0')) and
(D(0) = '0') and (qload = '0') and (qreq = '1'))
a(2) <= ([D(2) = '0'] and (D(1) = '1')) and
(D(0) = '1') and (qreq = '1'))
a(3) <= ([D(2) = '1'] and
(D(0) = '0') and (qload = '1'))
a(4) <= ([D(2) = '1'] and (D(1) = '0')
and (D(0) = '0'))
a(5) <= ([D(2) = '0'] and (D(1) = '0')
and (D(0) = '1'))
if (D(2) = '0') and (D(1) = '0') and (D(0) = '0') and (qload = '0') and (qreq = '1')) then
end if;
if (qtable_raddr = "000000") then
  out_data <= q0;
elsif (qtable_raddr = "000001") then
  out_data <= q1;
elsif (qtable_raddr = "000010") then
  out_data <= q2;
elsif (qtable_raddr = "000011") then
  out_data <= q3;
elsif (qtable_raddr = "000100") then
  out_data <= q4;
elsif (qtable_raddr = "000101") then
  out_data <= q5;
elsif (qtable_raddr = "000110") then
  out_data <= q6;
elsif (qtable_raddr = "000111") then
  out_data <= q7;
elsif (qtable_raddr = "001000") then
  out_data <= q8;
elsif (qtable_raddr = "001001") then
  out_data <= q9;
elsif (qtable_raddr = "001010") then
  out_data <= q10;
elsif (qtable_raddr = "001011") then
  out_data <= q11;
elsif (qtable_raddr = "001100") then
  out_data <= q12;
elsif (qtable_raddr = "001101") then
  out_data <= q13;
elsif (qtable_raddr = "001110") then
  out_data <= q14;
elsif (qtable_raddr = "001111") then
  out_data <= q15;
elsif (qtable_raddr = "010000") then
  out_data <= q16;
elsif (qtable_raddr = "010001") then
  out_data <= q17;
elsif (qtable_raddr = "010010") then
  out_data <= q18;
elsif (qtable_raddr = "010011") then
  out_data <= q19;
elsif (qtable_raddr = "010100") then
  out_data <= q20;
elsif (qtable_raddr = "010101") then
  out_data <= q21;
elsif (qtable_raddr = "010110") then
  out_data <= q22;
elsif (qtable_raddr = "010111") then
  out_data <= q23;
elsif (qtable_raddr = "011000") then
  out_data <= q24;
elsif (qtable_raddr = "011001") then
  out_data <= q25;
elsif (qtable_raddr = "011010") then
  out_data <= q26;
elsif (qtable_raddr = "011011") then
  out_data <= q27;
elsif (qtable_raddr = "011100") then
  out_data <= q28;
elsif (qtable_raddr = "011101") then
  out_data <= q29;
elsif (qtable_raddr = "011110") then
  out_data <= q30;
elsif (qtable_raddr = "011111") then
  out_data <= q31;
elsif (qtable_raddr = "100000") then
  out_data <= q32;
elsif (qtable_raddr = "100001") then
  out_data <= q33;
elsif (qtable_raddr = "100010") then
  out_data <= q34;
elsif (qtable_raddr = "100011") then
  out_data <= q35;
elsif (qtable_raddr = "100100") then
  out_data <= q36;
elsif (qtable_raddr = "100101") then
  out_data <= q37;
elsif (qtable_raddr = "100110") then
  out_data <= q38;
elsif (qtable_raddr = "100111") then
  out_data <= q39;
elsif (qtable_raddr = "101000") then
  out_data <= q40;
elsif (qtable_raddr = "101001") then
  out_data <= q41;
elsif (qtable_raddr = "101010") then
  out_data <= q42;
elsif (qtable_raddr = "101011") then
  out_data <= q43;
elsif (qtable_raddr = "101100") then
  out_data <= q44;
elsif (qtable_raddr = "101101") then
  out_data <= q45;
elsif (qtable_raddr = "101110") then
  out_data <= q46;
elsif (qtable_raddr = "101111") then
  out_data <= q47;
elsif (qtable_raddr = "110000") then
  out_data <= q48;
elsif (qtable_raddr = "110001") then
  out_data <= q49;
elsif (qtable_raddr = "110010") then
  out_data <= q50;
elsif (qtable_raddr = "110011") then
  out_data <= q51;
elsif (qtable_raddr = "110100") then
  out_data <= q52;
elsif (qtable_raddr = "110101") then
  out_data <= q53;
elsif (qtable_raddr = "110110") then
  out_data <= q54;
elsif (qtable_raddr = "110111") then
  out_data <= q55;
elsif (qtable_raddr = "111000") then
  out_data <= q56;
elsif (qtable_raddr = "111001") then
  out_data <= q57;
elsif (qtable_raddr = "111010") then
  out_data <= q58;
elsif (qtable_raddr = "111011") then
  out_data <= q59;
elsif (qtable_raddr = "111100") then
  out_data <= q60;
elsif (qtable_raddr = "111101") then
  out_data <= q61;
elsif (qtable_raddr = "111110") then
  out_data <= q62;
elsif (qtable_raddr = "111111") then
  out_data <= q63;
else
  out_data <= q0;
end if;
end if;

if (D(2) = '0') and (D(1) = '0') and (D(0) = '0') and (qload = '1') and (qreq = '0') then
  if (tqtable_addr = "000000") then
    q0 <= qdata_in;
  elsif (tqtable_addr = "000001") then
    q1 <= qdata_in;
  elsif (tqtable_addr = "000010") then
    q2 <= qdata_in;
  elsif (tqtable_addr = "000011") then
    q3 <= qdata_in;
  elsif (tqtable_addr = "000100") then
    q4 <= qdata_in;
  elsif (tqtable_addr = "000101") then
    q5 <= qdata_in;
  elsif (tqtable_addr = "000110") then
    q6 <= qdata_in;
elsif (tqtable_addr = "000111") then
  q7 <= qdata_in;
else
  q0 <= qdata_in;
end if;
end if;
q7 <= qdata_in;
elsif (qtable_addr = "001000") then
q8 <= qdata_in;
elsif (qtable_addr = "001001") then
q9 <= qdata_in;
elsif (qtable_addr = "001010") then
q10 <= qdata_in;
elsif (qtable_addr = "001011") then
q11 <= qdata_in;
elsif (qtable_addr = "001100") then
q12 <= qdata_in;
elsif (qtable_addr = "001101") then
q13 <= qdata_in;
elsif (qtable_addr = "001110") then
q14 <= qdata_in;
elsif (qtable_addr = "001111") then
q15 <= qdata_in;
elsif (qtable_addr = "010000") then
q16 <= qdata_in;
elsif (qtable_addr = "010001") then
q17 <= qdata_in;
elsif (qtable_addr = "010010") then
q18 <= qdata_in;
eelsif (qtable_addr = "010011") then
q19 <= qdata_in;
eelsif (qtable_addr = "010100") then
q20 <= qdata_in;
eelsif (qtable_addr = "010101") then
q21 <= qdata_in;
eelsif (qtable_addr = "010110") then
q22 <= qdata_in;
eelsif (qtable_addr = "010111") then
q23 <= qdata_in;
eelsif (qtable_addr = "011000") then
q24 <= qdata_in;
eelsif (qtable_addr = "011001") then
q25 <= qdata_in;
eelsif (qtable_addr = "011010") then
q26 <= qdata_in;
eelsif (qtable_addr = "011011") then
q27 <= qdata_in;
eelsif (qtable_addr = "011100") then
q28 <= qdata_in;
eelsif (qtable_addr = "011101") then
q29 <= qdata_in;
eelsif (qtable_addr = "011110") then
q30 <= qdata_in;
eelsif (qtable_addr = "011111") then
q31 <= qdata_in;
eelsif (qtable_addr = "100000") then
q32 <= qdata_in;
eelsif (qtable_addr = "100001") then
q33 <= qdata_in;
eelsif (qtable_addr = "100010") then
q34 <= qdata_in;
eelsif (qtable_addr = "100011") then
q35 <= qdata_in;
eelsif (qtable_addr = "100100") then
q36 <= qdata_in;
eelsif (qtable_addr = "100101") then
q37 <= qdata_in;
eelsif (qtable_addr = "100110") then
q38 <= qdata_in;
eelsif (qtable_addr = "100111") then
q39 <= qdata_in;
eelsif (qtable_addr = "101000") then
q40 <= qdata_in;
eelsif (qtable_addr = "101001") then
q41 <= qdata_in;
eelsif (qtable_addr = "101010") then
q42 <= qdata_in;
eelsif (qtable_addr = "101011") then
q43 <= qdata_in;
eelsif (qtable_addr = "101100") then
q44 <= qdata_in;
eelsif (qtable_addr = "101101") then
q45 <= qdata_in;
eelsif (qtable_addr = "101110") then
q46 <= qdata_in;
eelsif (qtable_addr = "101111") then
q47 <= qdata_in;
eelsif (qtable_addr = "110000") then
q48 <= qdata_in;
eelsif (qtable_addr = "110001") then
q49 <= qdata_in;
eelsif (qtable_addr = "110010") then
q50 <= qdata_in;
eelsif (qtable_addr = "110011") then
q51 <= qdata_in;
eelsif (qtable_addr = "110100") then
q52 <= qdata_in;
eelsif (qtable_addr = "110101") then
q53 <= qdata_in;
eelsif (qtable_addr = "110110") then
q54 <= qdata_in;
eelsif (qtable_addr = "110111") then
q55 <= qdata_in;
eelsif (qtable_addr = "111000") then
q56 <= qdata_in;
eelsif (qtable_addr = "111001") then
q57 <= qdata_in;
eelsif (qtable_addr = "111010") then
q58 <= qdata_in;
eelsif (qtable_addr = "111011") then
q59 <= qdata_in;
eelsif (qtable_addr = "111100") then
q60 <= qdata_in;
eelsif (qtable_addr = "111101") then
q61 <= qdata_in;
eelsif (qtable_addr = "111110") then
q62 <= qdata_in;
eelsif (qtable_addr = "111111") then
q63 <= qdata_in;
end if;
end if;

D(2) <= ((a(0) = '1') or (a(3) = '1') or (a(4) = '1'));
D(1) <= ((a(2) = '1') or (a(3) = '1') or (a(4) = '1'));
D(0) <= ((a(1) = '1') or (a(2) = '1') or (a(5) = '1'));
end if;
end process qmem;
-- outputs
qload_ack <= '1' when (a(3) = '1') or (a(4) = '1') or (a(5) = '1') else '0';
qreq_ack <= '1' when (a(2) = '1') else '0';
qdata_out <= out_data;
end qmem_arch;

COMPONENT: dequantize
DESCRIPTION: (entity)

-- Name: dequantize_entity
-- Created : 22-APR-1994
-- Revised : 
-- library and use clauses
library mgc_portable, ieee;
use mgc_portable.qsim_relations.all;
use mgc_portable.qsim_logic.all;
use ieee.std_logic_1164.all;

A-62
use ieee.std_logic_1164_extensions.all;
library my_packages;
use my_packages.package_1.all;

entity dequantize is
  port (  
clock : in
std_ulogic;
reset : in
std_ulogic;
do_dequant : in
std_ulogic;
do_dequantUnload_ack : out
std_ulogic;
do_dequant_ack : out
std_ulogic;
qudata_out : in
std_ulogic_vector(7 downto 0);
qureq : out
std_ulogic;
qureq_ack : in
std_ulogic;
qtable_addr : out
std_ulogic_vector(5 downto 0);
  );
end dequantize;

COMPONENT: dequantize
DESCRIPTION: (architecture)

-- Name : dequantize_arch
-- Purpose :
-- Author : Douglas A. Carpenter
-- Created : 22-APR-1994
-- Revised :

architecture dequantize_arch of dequantize is
  signal D : std_ulogic_vector(3 downto 0);
  signal Q : std_ulogic_vector(3 downto 0);
  subtype ramword is std_ulogic_vector(6 downto 0);
type rammemory is array (63 downto 0) of
ramword;
signal ZigZagRam : rammemory;
signal i_outs : std_ulogic_vector(3 downto 0);
signal i : integer range 0 to 7;
signal j : integer range 0 to 7;
signal unzigaddr : integer range 0 to 63;
signal ziggedaddr : integer range 0 to 63;
signal i_data : std_ulogic_vector(15 downto 0);

begin
  Dequant : process(reset, clock)
  begin
    if (reset = '0') then
      Q <= "0000";
      unzigaddr <= 0;
      ziggedaddr <= 0;
      i <= 0;
      j <= 0;
      ZigZagRam(0) <= "0" &
      To_StdUlogicVector(0,6);
      ZigZagRam(1) <= "0" &
      To_StdUlogicVector(1,6);
      ZigZagRam(2) <= "0" &
      To_StdUlogicVector(5,6);
      ZigZagRam(3) <= "0" &
      To_StdUlogicVector(6,6);
      ZigZagRam(4) <= "0" &
      To_StdUlogicVector(14,6);
      ZigZagRam(5) <= "0" &
      To_StdUlogicVector(15,6);
      ZigZagRam(6) <= "0" &
      To_StdUlogicVector(27,6);
      ZigZagRam(7) <= "0" &
      To_StdUlogicVector(28,6);
      ZigZagRam(8) <= "0" &
      To_StdUlogicVector(2,6);
      ZigZagRam(9) <= "0" &
      To_StdUlogicVector(4,6);
      ZigZagRam(10) <= "0" &
      To_StdUlogicVector(7,6);
      ZigZagRam(11) <= "0" &
      To_StdUlogicVector(13,6);
      ZigZagRam(12) <= "0" &
      To_StdUlogicVector(16,6);
      ZigZagRam(13) <= "0" &
      To_StdUlogicVector(26,6);
      ZigZagRam(14) <= "0" &
      To_StdUlogicVector(29,6);
      ZigZagRam(15) <= "0" &
      To_StdUlogicVector(42,6);
      ZigZagRam(16) <= "0" &
      To_StdUlogicVector(3,6);
      ZigZagRam(17) <= "0" &
      To_StdUlogicVector(8,6);
      ZigZagRam(18) <= "0" &
      To_StdUlogicVector(12,6);
      ZigZagRam(19) <= "0" &
      To_StdUlogicVector(17,6);
      ZigZagRam(20) <= "0" &
      To_StdUlogicVector(25,6);
      ZigZagRam(21) <= "0" &
      To_StdUlogicVector(30,6);
      ZigZagRam(22) <= "0" &
      To_StdUlogicVector(41,6);
      ZigZagRam(23) <= "0" &
      To_StdUlogicVector(43,6);
      ZigZagRam(24) <= "0" &
      To_StdUlogicVector(9,6);
      ZigZagRam(25) <= "0" &
      To_StdUlogicVector(11,6);
      ZigZagRam(26) <= "0" &
      To_StdUlogicVector(18,6);

    elsif ((clock'event) and (clock = '1')) and
    (clock'last_value = '0')) then
      if (D = "0000") then
        unzigaddr <= 0;
        ziggedaddr <= 0;
        i <= 0;
        j <= 0;
        ZigZagRam(0) <= "0" &
        To_StdUlogicVector(0,6);
        ZigZagRam(1) <= "0" &
        To_StdUlogicVector(1,6);
        ZigZagRam(2) <= "0" &
        To_StdUlogicVector(5,6);
        ZigZagRam(3) <= "0" &
        To_StdUlogicVector(6,6);
        ZigZagRam(4) <= "0" &
        To_StdUlogicVector(14,6);
        ZigZagRam(5) <= "0" &
        To_StdUlogicVector(15,6);
        ZigZagRam(6) <= "0" &
        To_StdUlogicVector(27,6);
        ZigZagRam(7) <= "0" &
        To_StdUlogicVector(28,6);
        ZigZagRam(8) <= "0" &
        To_StdUlogicVector(2,6);
        ZigZagRam(9) <= "0" &
        To_StdUlogicVector(4,6);
        ZigZagRam(10) <= "0" &
        To_StdUlogicVector(7,6);
        ZigZagRam(11) <= "0" &
        To_StdUlogicVector(13,6);
        ZigZagRam(12) <= "0" &
        To_StdUlogicVector(16,6);
        ZigZagRam(13) <= "0" &
        To_StdUlogicVector(26,6);
        ZigZagRam(14) <= "0" &
        To_StdUlogicVector(29,6);
        ZigZagRam(15) <= "0" &
        To_StdUlogicVector(42,6);
        ZigZagRam(16) <= "0" &
        To_StdUlogicVector(3,6);
        ZigZagRam(17) <= "0" &
        To_StdUlogicVector(8,6);
        ZigZagRam(18) <= "0" &
        To_StdUlogicVector(12,6);
        ZigZagRam(19) <= "0" &
        To_StdUlogicVector(17,6);
        ZigZagRam(20) <= "0" &
        To_StdUlogicVector(25,6);
        ZigZagRam(21) <= "0" &
        To_StdUlogicVector(30,6);
        ZigZagRam(22) <= "0" &
        To_StdUlogicVector(41,6);
        ZigZagRam(23) <= "0" &
        To_StdUlogicVector(43,6);
        ZigZagRam(24) <= "0" &
        To_StdUlogicVector(9,6);
        ZigZagRam(25) <= "0" &
        To_StdUlogicVector(11,6);
        ZigZagRam(26) <= "0" &
        To_StdUlogicVector(18,6);

    end if;
  end if;
end Dequant;
ZigZagRam(27) <= "0" & To_StdUlogicVector(24,6); ZigZagRam(28) <= "0" & To_StdUlogicVector(31,6); ZigZagRam(29) <= "0" & To_StdUlogicVector(40,6); ZigZagRam(30) <= "0" & To_StdUlogicVector(44,6); ZigZagRam(31) <= "0" & To_StdUlogicVector(53,6); ZigZagRam(32) <= "0" & To_StdUlogicVector(10,6); ZigZagRam(33) <= "0" & To_StdUlogicVector(19,6); ZigZagRam(34) <= "0" & To_StdUlogicVector(23,6); ZigZagRam(35) <= "0" & To_StdUlogicVector(32,6); ZigZagRam(36) <= "0" & To_StdUlogicVector(39,6); ZigZagRam(37) <= "0" & To_StdUlogicVector(45,6); ZigZagRam(38) <= "0" & To_StdUlogicVector(52,6); ZigZagRam(39) <= "0" & To_StdUlogicVector(54,6); ZigZagRam(40) <= "0" & To_StdUlogicVector(20,6); ZigZagRam(41) <= "0" & To_StdUlogicVector(22,6); ZigZagRam(42) <= "0" & To_StdUlogicVector(33,6); ZigZagRam(43) <= "0" & To_StdUlogicVector(38,6); ZigZagRam(44) <= "0" & To_StdUlogicVector(46,6); ZigZagRam(45) <= "0" & To_StdUlogicVector(51,6); ZigZagRam(46) <= "0" & To_StdUlogicVector(55,6); ZigZagRam(47) <= "0" & To_StdUlogicVector(60,6); ZigZagRam(48) <= "0" & To_StdUlogicVector(21,6); ZigZagRam(49) <= "0" & To_StdUlogicVector(34,6); ZigZagRam(50) <= "0" & To_StdUlogicVector(37,6); ZigZagRam(51) <= "0" & To_StdUlogicVector(47,6); ZigZagRam(52) <= "0" & To_StdUlogicVector(50,6); ZigZagRam(53) <= "0" & To_StdUlogicVector(56,6); ZigZagRam(54) <= "0" & To_StdUlogicVector(59,6); ZigZagRam(55) <= "0" & To_StdUlogicVector(61,6); ZigZagRam(56) <= "0" & To_StdUlogicVector(35,6); ZigZagRam(57) <= "0" & To_StdUlogicVector(36,6); ZigZagRam(58) <= "0" & To_StdUlogicVector(48,6); ZigZagRam(59) <= "0" & To_StdUlogicVector(49,6); ZigZagRam(60) <= "0" & To_StdUlogicVector(57,6); ZigZagRam(61) <= "0" & To_StdUlogicVector(58,6); ZigZagRam(62) <= "0" & To_StdUlogicVector(62,6); ZigZagRam(63) <= "0" & To_StdUlogicVector(63,6);

if (do_dequant = '1') then
  dequant = "00001" -- got do
  i_outs <= "0000";
  else
    Q <= "00000";
    -- waiting for
start signal
  i_outs <= "00000";
  else (D = "0001") then
    Q <= "00111";
    i_outs <= "00000";
    elsif (D = "00111") then
      ziggedaddr <= To_Integer(ZigZagRam(unzigaddr));
      Q <= "00100";
      i_outs <= "00000";
      elsif (D = "00101") then
        if (qreq_ack = '0') then
          Q <= "01100";
          i_outs <= "00100";
        else
          Q <= "00101";
          -- qreq not ready, wait
        end
        else
          i_outs <= "00000";
        end
      elsif (D = "01111") then
        if (qreq Ack = '1') then
          Q <= "00101";
          -- qreq ack
        signalled
          i_outs <= "00000";
        else
          Q <= "01000";
          -- wait
          i_outs <= "00000";
          end
        elsif (D = "01011") then
          int_data <= To_Integer(q_data) * To_Integer(LoadToDeqDOUTl);
          Q <= "01011";
          i_outs <= "00000";
          elsif (D = "01111") then
            Q <= "11111";
            i_outs <= "00000";
            elsif (D = "11111") then
              Q <= "11111";
              i_outs <= "00000";
              -- do write
              if (j = 7) then
                j <= 0;
                i_outs <= "00000";
                Q <= "11100";
              else
                j <= j + 1;
                i_outs <= "00000";
                Q <= "00011";
              end
            elsif (D = "11000") then
              if (i = 7) then
                i <= 0;
                i_outs <= "00011";
                Q <= "10000";
              -- end of
            else
              if (do_dequant = '1') then
                Q <= "00001";
                i_outs <= "00000";
                -- got do
                dequant = "00001";
                i_outs <= "00000";
                else

A-64
i <= i + 1;
i_outs <= "0000";
Q <= "0001";
end if;
elif (D = "1000") then
if (do_dequant = '0') then
Q <= "0000";
i_outs <= "0000";
else
Q <= "1000";
i_outs <= "0001";
end if;
else
Q <= "0000";
i_outs <= "0000";
end if;
end process Dequant;

-- assign output values
D <= Q;
qreq <= i_outs(1);
do_dequant_unload_ack <= i_outs(0);
do_dequant_ack <= i_outs(0);
DeqToIDCT_LDl <= i_outs(2);
DeqToIDCT_WEi <= i_outs(3);
LoadToDeq_RADRl <=
To.StdUlogicVector(ziggedaddr,6);
qtable_raddr <=
To.StdUlogicVector(ziggedaddr,6);
DeqToIDCT_WADRl <=
To.StdUlogicVector(unzigaddr,6);
DeqToIDCT_DINl <= i_data;
end dequantize_arch;

COMPONENT:  reg16
DESCRIPTION:  (architecture)

--
-- Name :  reg16_arch
--
-- Purpose :  reg16 module
-- Author :  Douglas A. Carpenter
-- Created :  23-APR-1994 DAC
-- Revised :  .

architecture reg16_arch of reg16 is
signal i_data : std_ulogic_vector(15 downto 0);
begin
reg_process process(reset,clock)
begin
if (reset = '0') then
i_data <= "0000000000000000";
else if ((clock'event) and (clock = '1') and
(clock'last_value = '0')) then
if (in_latch = '1') then
i_data <= in_data;
else
i_data <= i_data;
end if;
end if;
end process reg_process;

-- assign output values
out_data <= i_data;
end reg16_arch;

COMPONENT:  reg32
DESCRIPTION:  (entity)

--
-- Name :  reg32_entity
--
-- Purpose :  A general purpose 16 bit register
-- Author :  Douglas A. Carpenter
-- Created :  26-MAY-1994
-- Revised :  .

-- library and use clauses
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164_extensions.all;
library my_packages;
use my_packages.package_l.all;

entity reg32 is
port ( -- inputs
reset : in  std_ulogic;
clock : in  std_ulogic;
in_data : in  std_ulogic_vector(15 downto 0);
in_latch : in  std_ulogic;
-- outputs
out_data : out  std_ulogic_vector(15 downto 0)
);
end reg32;
```
-- Architecture: reg32_arch
-- Name: reg32_arch
-- Purpose: Reg 32 module
-- Author: Douglas A. Carpenter
-- Created: 26-MAY-1994 DAC
-- Revised: -

architecture reg32_arch of reg32 is
  signal i_data : std_ulogic_vector(31 downto 0);
begin
  reg_process : process(reset,clock)
  begin
    if (reset = '0') then
      i_data <= "00000000000000000000000000000000";
    elsif ((clock'event) and (clock = '1')) and 
      (clock'last_value = '0')) then
      if (in_latch = '1') then
        i_data <= i_data;
      else
        i_data <= i_data;
      end if;
    end if;
  end process reg_process;
-- assign output values
  out_data <= i_data;
end reg32_arch;

-- COMPONENT: find_sos
-- DESCRIPTION: (entity)

-- Purpose: Find Start of Scan
-- Author: Douglas A. Carpenter
-- Created: 27-APR-1994
-- Revised: -

-- library and use clauses
library mgc_portable, ieee;
use mgc_portable.qsim_logic.all;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164_extensions.all;
library my_packages;
use my_packages.package_1.all;

entity find_sos is
  port (  -- inputs
    clock : in  std_ulogic;
    reset : in  std_ulogic;
    find_sos_req : in  std_ulogic;
    data : in  std_ulogic_vector(7 downto 0);
    next_ack : in  std_ulogic;
    err : in  std_ulogic;
    isr_latch_ack : in  std_ulogic;
    qtable_ack : in  std_ulogic;
    htable_ack : in  std_ulogic;
    table_err : in  std_ulogic;
  -- outputs
    find_sos_ack : out  std_ulogic;
    find_sos_err : out  std_ulogic;
    next_req : out  std_ulogic;
    isr_latch : out  std_ulogic;
    isr_data_in : out  std_logic_vector(7 downto 0);
    qtable_req : out  std_ulogic;
  );
end find_sos;
```

```
architecture find_sos_arch of find_sos is
  signal D : std_logic_vector(3 downto 0);
  signal a0 : std_logic;
  signal a1 : std_logic;
  signal a2 : std_logic;
  signal a3 : std_logic;
  signal a4 : std_logic;
  signal a5 : std_logic;
  signal a6 : std_logic;
  signal a7 : std_logic;
  signal a8 : std_logic;
  signal a9 : std_logic;
  signal a10 : std_logic;
  signal a11 : std_logic;
  signal a12 : std_logic;
  signal a13 : std_logic;
  signal a14 : std_logic;
  signal a15 : std_logic;
  signal a16 : std_logic;
  signal a17 : std_logic;
  signal a18 : std_logic;
  signal a19 : std_logic;
  signal a20 : std_logic;
  signal a21 : std_logic;
  signal a22 : std_logic;
  signal a23 : std_logic;
  signal a24 : std_logic;
  signal a25 : std_logic;
  signal a26 : std_logic;
  signal a27 : std_logic;
  signal a28 : std_logic;
  signal a29 : std_logic;
  signal a30 : std_logic;
  signal a31 : std_logic;
  signal a32 : std_logic;
  signal a33 : std_logic;
  signal a34 : std_logic;
  signal a35 : std_logic;
  signal a36 : std_logic;
  signal a37 : std_logic;
  signal a38 : std_logic;
  signal a39 : std_logic;
  signal a40 : std_logic;
  signal a41 : std_logic;
  signal a42 : std_logic;
  signal a43 : std_logic;
  signal a44 : std_logic;
  signal a45 : std_logic;
  signal a46 : std_logic;
  signal a47 : std_logic;
  signal a48 : std_logic;
  signal a49 : std_logic;
  signal a50 : std_logic;
  signal a51 : std_logic;
  signal a52 : std_logic;
  signal a53 : std_logic;
  signal a54 : std_logic;
  signal a55 : std_logic;
```

```
A-66
```
begin

fsos_process : process(reset, clock)
begin
if (reset = '0') then
 D <= "00000";
 a0 <= '0';
 a1 <= '0';
 a2 <= '0';
 a3 <= '0';
 a4 <= '0';
 a5 <= '0';
 a6 <= '0';
 a7 <= '0';
 a8 <= '0';
 a9 <= '0';
a10 <= '0';
a11 <= '0';
a12 <= '0';
a13 <= '0';
a14 <= '0';
a15 <= '0';
a16 <= '0';
a17 <= '0';
a18 <= '0';
a19 <= '0';
a20 <= '0';
a21 <= '0';
a22 <= '0';
a23 <= '0';
a24 <= '0';
a25 <= '0';
a26 <= '0';
a27 <= '0';
a28 <= '0';
a29 <= '0';
a30 <= '0';
a31 <= '0';
a32 <= '0';
a33 <= '0';
a34 <= '0';
a35 <= '0';
a36 <= '0';
a37 <= '0';
a38 <= '0';
a39 <= '0';
a40 <= '0';
a41 <= '0';
a42 <= '0';
a43 <= '0';
a44 <= '0';
a45 <= '0';
a46 <= '0';
a47 <= '0';
a48 <= '0';
a49 <= '0';
a50 <= '0';
a51 <= '0';
a52 <= '0';
a53 <= '0';
a54 <= '0';
a55 <= '0';
elsif ((clock'event) and (clock = '1'') and (clock'last_value = '0')) then
 a0 <= (D(3) = '0') and (D(2) = '0') and (D(1) = '1') and (D(0) = '0') and (data = "1111111") and (next_ack = '0');
 a1 <= (D(3) = '0') and (D(2) = '1') and (D(1) = '0') and (data = "11000100") and (next_ack = '0');
 a2 <= (D(3) = '0') and (D(2) = '1') and (D(1) = '1') and (D(0) = '0') and (data = "11011011") and (next_ack = '0');
end if;
end process;
end

and (next_ack = '0'));
a3 <= (D(3) = '1') and (D(2) = '0') and (D(1) = '1') and (D(0) = '0') and (data = "11011010") and (next_ack = '0');
a4 <= (D(3) = '1') and (D(2) = '1') and (D(1) = '0') and (D(0) = '0') and (table_ack = '0') and (table_err = '0');
a5 <= (D(3) = '0') and (D(2) = '0') and (D(1) = '0') and (find_sos = '1') and (next_ack = '0') and (err = '0'));
a6 <= (D(3) = '1') and (D(2) = '1') and (D(0) = '0') and (table_ack = '1') and (table_err = '0');
a7 <= (D(3) = '0') and (D(2) = '0') and (D(1) = '1') and (D(0) = '0') and (next_ack = '1') and (err = '0'));
a8 <= (D(3) = '1') and (D(2) = '0') and (D(1) = '1') and (D(0) = '0') and (data(1) = '0') and (next_ack = '0');
a9 <= (D(3) = '1') and (D(2) = '0') and (D(1) = '1') and (D(0) = '0') and (data(2) = '1') and (next_ack = '0');
a10 <= (D(3) = '1') and (D(2) = '0') and (D(1) = '1') and (D(0) = '0') and (data(3) = '0') and (next_ack = '0');
a11 <= (D(3) = '1') and (D(2) = '0') and (D(1) = '1') and (D(0) = '0') and (data(4) = '0') and (next_ack = '0');
a12 <= (D(3) = '1') and (D(2) = '0') and (D(1) = '1') and (D(0) = '0') and (data(5) = '1') and (next_ack = '0');
a13 <= (D(3) = '1') and (D(2) = '0') and (D(1) = '1') and (D(0) = '0') and (data(6) = '0') and (next_ack = '0');
a14 <= (D(3) = '1') and (D(2) = '0') and (D(1) = '1') and (D(0) = '0') and (data(7) = '0') and (next_ack = '0');
a15 <= (D(3) = '0') and (D(2) = '1') and (D(1) = '1') and (D(0) = '0') and (next_ack = '1');
 a16 <= (D(3) = '1') and (D(2) = '0') and (D(1) = '1') and (D(0) = '0') and (data(0) = '1') and (next_ack = '0');
a17 <= (D(3) = '0') and (D(2) = '0') and (D(1) = '1') and (D(0) = '0') and (next_ack = '1') and (err = '0');
 a18 <= (D(3) = '1') and (D(2) = '0') and (D(1) = '0') and (D(0) = '0') and (data(0) = '0') and (data(1) = '0') and (data(2) = '0') and (data(3) = '0') and (data(4) = '0') and (data(5) = '0') and (data(6) = '0') and (data(7) = '0') and (next_ack = '0');
end process fsos_process;
-- assign output values
find_sos_ack <= '1' when ((a3 = '1') or (a49 = '1')) else '0';
find_sos_err <= '1' when ((a19 = '1') or (a52 = '1')) else '0';
next_req <= '1' when ((a0 = '1') or (a5 = '1')) or (a8 = '1') or (a3 = '1') or (a13 = '1') or (a14 = '1') or (a16 = '1') or (a18 = '1') or (a19 = '1') or (a20 = '1') or (a21 = '1') or (a22 = '1') or (a23 = '1') or (a24 = '1') or (a25 = '1') or (a27 = '1') or (a28 = '1') or (a31 = '1') or (a52 = '1') or (a54 = '1') or (a55 = '1');
architecture dec_scan_arch of dec_scan is
signal D : std_logic_vector(3 downto 0);
signal Q : std_logic_vector(3 downto 0);
signal i_outs : std_logic_vector(4 downto 0);
begin

dscan_process : process(reset, clock)
begin
  if (reset = '0') then
    Q <= "0000";
    i_outs <= "00000";
  elsif (clock'event) and (clock = '1') and
        (clock'last_value = '0') then
    if (D = "0000") then
      if (decode_scan = '1') then
        Q <= "0001";
      end if;
    else
      Q <= "0001";
      i_outs <= "00010";
    end if;
  elsif (D = "0001") then
    if (computeNumMCUsack = '1') then
      Q <= "0011";
    end if;
  elsif (D = "0011") then
    if (computeNumMCUsack = '0') then
      Q <= "0101";
      i_outs <= "10000";
    end if;
  else
    i_outs <= "00000";
  end if;
end if;

-- Purpose : Decode scan module
-- Author : Douglas A. Carpenter
-- Created : 27-APR-1994
-- Revised :

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164_extensions.all;

library my_packages;
use my_packages.package_l.all;

declration

entity dec_scan is
  -- inputs
  std_ulogic;
  reset : in
  std_ulogic;
  decode_scan : in
  std_ulogic;
  -- outputs
  decode_scan_ack : out
  std_ulogic;
  decode_scan_err : out
  std_ulogic;
  computeNumMCUs : out
  std_ulogic;
  computeNumMCUsack : in
  std_ulogic;
  decode_scan_header : out
  std_ulogic;
  decode_scan_header_ack : in
  std_ulogic;
  decode_scan_header_err : in
  std_ulogic;
  startscan : out
  std_ulogic;
  scancomplete : in
  std_ulogic;
end dec_scan;

begin
Q <= "0010";  -- wait for scan
    i_outs <= "00001";
    end if;
    elsif (Q = "0110") then
      if (scancomplete = '0') then
        Q <= "0100";  -- ack dropped
      back to 0
        i_outs <= "00000";
      else
        Q <= "0110";  -- wait for ack to drop
        i_outs <= "00000";
      end if;
    end if;
  end if;
end;
}

begin
  i_outs <= "00000";
  elsif (Q = "0100") then
    if (decode_scan = '0') then
      -- decode scan
    signal turned off, back to start
      i_outs <= "00000";
      elsif (Q = "1111") then  -- error
        Q <= "0100";  -- wait for signal
      to return to off
        i_outs <= "01000";
      end if;
    end if;
  else
    Q <= "0000";
    i_outs <= "00000";
    end if;
  end if;
end process dscan_process;

-- assign output values
   D <= Q;
   decode_scan_ack <= i_outs(3);
   decode_scan_err <= i_outs(2);
   computeNumMCUs <= i_outs(1);
   startscan <= i_outs(0);
   decode_scan_header <= i_outs(4);
end dec_scan_arch;

---
-- Name . dec_scan_header
-- Purpose :
-- Author : Douglas A. Carpenter
-- Created : 07-JUN-1994
-- Revised :

-- library and use clauses
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164_extensions.all;
library my_packages;

use my_packages.package_1.all;

entity dec_scan_header is
  port ( -- inputs
    clock : in
    std_ulogic;
    reset : in
    std_ulogic;
    data : in
    std_ulogic_vector(7 downto 0);
    next_ack : in
    std_ulogic;
    err : in
    std_ulogic;
    isr_latch_ack : in
    std_ulogic;
    -- outputs
    next_req : out
    std_ulogic;
    isr_latch : out
std_ulogic;
    isr_data_in : out
std_logic_vector(7 downto 0);
  decode_scan_header : in
std_ulogic;
  decode_scan_header_ack : out
std_ulogic;
  decode_scan_header_err : out
std_ulogic;
);
end dec_scan_header;

COMPONENT: dec_scan_header
DESCRIPTION: (architecture)

-- Purpose : Decode scan header module
-- Author : Douglas A. Carpenter
-- Created : 07-JUN-1994
-- Revised :

architecture dec_scan_header_arch of dec_scan_header is
signal D : std_ulogic_vector(2 downto 0);
signal Q : std_ulogic_vector(2 downto 0);
signal i_outs : std_logic_vector(3 downto 0);
begin
dscanh_process : process(reset, clock)
variable len_cntr : std_logic_vector(3
downto 0);
begin
  if (reset = '0') then
    Q <= "000";
i_outs <= "00000";
  len_cntr := "1000";
  elsif ((clock'event) and (clock = '1') and
    (clock'last value = '0')) then
  if (D = "000") then
    Q <= "001";
i_outs <= "00000";
  else
    Q <= "000";
i_outs <= "00000";
  len_cntr := "1000";
  end if;
  elsif (D = "001") then
  if (len_cntr = "0000") then
    Q <= "101";  -- end of scan
  header
    i_outs <= "0010";
  else

A-70
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_1164_extensions.all;

-- Written by LL_to_VHDL at Tue May 3 09:08:37 1994
-- Parameterized Generator Specification to VHDL Code
--
-- LogicLib generator called: REGISTER_FILE
-- Passed Parameters are:
--   tinst name = huff2_reg256by8
--   parameters are:
--     W = 8
--     D = 256
--     read_ports = 1

-- huff2_reg256by8 Entity Description
entity huff2_reg256by8 is
  port(
    DIN1: in std_ulogic_vector(7 downto 0);
    RADR1: in std_ulogic_vector(7 downto 0);
    WADR1: in std_ulogic_vector(7 downto 0);
    DOUT1: out std_ulogic_vector(7 downto 0);
    LD1,WE1: in std_ulogic;
    reset: in std_ulogic;
    acdc . in std_logic
  );
end huff2_reg256by8;

-- huff2_reg256by8 Architecture Description
-- Used for the storage of huffman vals

architecture rtl of huff2_reg256by8 is
  subtype ramword is std_ulogic_vector(1 downto 0);
type rammemory is array (255 downto 0) of ramword;
signal ram : rammemory;
begin
  REGISTER_FILE_read_Process: process(ram,RADR1)
  variable raddrl : integer range 0 to 255;
  begin
    -- Process the read port number 1
    -- convert address to integer
    raddrl := to_integer('0' & RADR1,0);
    DOUT1 <= ram(raddrl);
  end process REGISTER_FILE_read_Process;

  REGISTER_FILE_write_Process: process(DIN1,WE1,L0,LD1,reset,acdc)
  variable waddr : integer range 0 to 255;
  variable laddr : integer range 0 to 255;
  variable load : std_ulogic;
  begin
    if (reset = '0') then
      if (acdc = '0') then
        -- 0 is for DC tables
    end if;
  end if;
end if;
end if;
end process dscanh_process;

-- assign output values
D <= Q;
irr_data_in <= "00000000";
nexr_req <= i_outs(3);
irr_latch <= i_outs(2);
dec_scan_header_ack <= i_outs(4);
dec_scan_header_err <= i_outs(0);
end dec_scan_header_arch;

COMPONENT:  huff2_reg256by8
DESCRIPTION:  (entity)

...
-- 0 ram(0) <= To_StdUlogicVector(0,8);
-- 1 ram(1) <= To_StdUlogicVector(1,8);
-- 2 ram(2) <= To_StdUlogicVector(2,8);
-- 3 ram(3) <= To_StdUlogicVector(3,8);
-- 4 ram(4) <= To_StdUlogicVector(4,8);
-- 5 ram(5) <= To_StdUlogicVector(5,8);
-- 6 ram(6) <= To_StdUlogicVector(6,8);
-- 7 ram(7) <= To_StdUlogicVector(7,8);
-- 8 ram(8) <= To_StdUlogicVector(8,8);
-- 9 ram(9) <= To_StdUlogicVector(9,8);
-- 10 ram(10) <= To_StdUlogicVector(10,8);
-- 11 ram(11) <= To_StdUlogicVector(11,8);
else
    ram(0) <= To_StdUlogicVector(1,8);
    ram(1) <= To_StdUlogicVector(2,8);
    ram(2) <= To_StdUlogicVector(3,8);
    ram(3) <= To_StdUlogicVector(0,8);
    ram(4) <= To_StdUlogicVector(4,8);
    ram(5) <= To_StdUlogicVector(17,8);
    ram(6) <= To_StdUlogicVector(5,8);
    ram(7) <= To_StdUlogicVector(18,8);
    ram(8) <= To_StdUlogicVector(33,8);
    ram(9) <= To_StdUlogicVector(49,8);
    ram(10) <= To_StdUlogicVector(65,8);
    ram(11) <= To_StdUlogicVector(6,8);
    ram(12) <= To_StdUlogicVector(19,8);
    ram(13) <= To_StdUlogicVector(81,8);
    ram(14) <= To_StdUlogicVector(97,8);
    ram(15) <= To_StdUlogicVector(7,8);
    ram(16) <= To_StdUlogicVector(34,8);
    ram(17) <= To_StdUlogicVector(113,8);
    ram(18) <= To_StdUlogicVector(20,8);
    ram(19) <= To_StdUlogicVector(50,8);
    ram(20) <= To_StdUlogicVector(129,8);
    ram(21) <= To_StdUlogicVector(145,8);
    ram(22) <= To_StdUlogicVector(161,8);
    ram(23) <= To_StdUlogicVector(19,8);
    ram(24) <= To_StdUlogicVector(35,8);
    ram(25) <= To_StdUlogicVector(66,8);
    ram(26) <= To_StdUlogicVector(177,8);
    ram(27) <= To_StdUlogicVector(193,8);
    ram(28) <= To_StdUlogicVector(21,8);
    ram(29) <= To_StdUlogicVector(82,8);
    ram(30) <= To_StdUlogicVector(209,8);
    ram(31) <= To_StdUlogicVector(240,8);
    ram(32) <= To_StdUlogicVector(36,8);
--
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_1164_extensions.all;

-- Written by LL_to_VHDL at Tue May 3 09:08:37 1994
-- Parameterized Generator Specification to VHDL
-- Code
--
-- LogicLib generator called: REGISTER_FILE
-- Passed Parameters are:
-- tname = huff_reg256by16
-- parameters are:
-- W = 8

-- D = 256
-- read_ports = 1

-- huff_reg16by16 Entity Description
entity huff_reg16by16 is
  port(
    DIN1 : in  std_logic_vector(15 downto 0);
    RADDR1 : in  std_logic_vector(3 downto 0);
    WADDR1 : in  std_logic_vector(3 downto 0);
    DOUT1 : out  std_logic_vector(15 downto 0);
    LD1 : in  std_logic;
    WE1 : in  std_logic;
    reset : in  std_logic;
    acce : in  std_logic;
    minmax : in  std_logic
  );
end huff_reg16by16:

COMPONENT: huff_reg16by16
DESCRIPTION: (architecture)

-- huff_reg16by16 Architecture Description
architecture rtl of huff_reg16by16 is
  subtype ramword is std_logic_vector(15 downto 0);
type rammemory is array (15 downto 0) of ramword;
signal ram : rammemory;
begin
  REGISTER_FILE_read_Process: process(ram,RADDR1)
  variable raddr1 : integer range 0 to 15;
  begin
    Process the read port number 1
    -- convert address to integer
    raddr1 := to_integer('0' & RADDR1,0);
    DOUT1 <= ram(raddr1);
  end process REGISTER_FILE_read_Process;

  REGISTER_FILE_write_Process: process(DIN1,WE1,LD1,RADDR1,reset,acce,minmax)
  variable waddr : integer range 0 to 15;
  variable load : std_logic;
  begin
    if (reset = '0') then
      if (acce = '0') then
        -- 0 is
        for DC tables
        if (minmax = '0') then
          -- 0 is
          for min table
          ram(0) <= To_StdUlogicVector(0,16);
          ram(1) <= To_StdUlogicVector(0,16);
          ram(2) <= To_StdUlogicVector(0,16);
          ram(3) <= To_StdUlogicVector(0,16);
          ram(4) <= To_StdUlogicVector(14,16);
          ram(5) <= To_StdUlogicVector(30,16);
          ram(6) <= To_StdUlogicVector(62,16);
          ram(7) <= To_StdUlogicVector(126,16);
          ram(8) <= To_StdUlogicVector(254,16);
          ram(9) <= To_StdUlogicVector(510,16);
          ram(10) <= To_StdUlogicVector(1022,16);
          ram(11) <= To_StdUlogicVector(0,16);
          ram(12) <= To_StdUlogicVector(0,16);
          ram(13) <= To_StdUlogicVector(0,16);
          ram(14) <= To_StdUlogicVector(0,16);
          ram(15) <= To_StdUlogicVector(0,16);
          else
            -- max table
            ram(0) <= To_StdUlogicVector(0,16);
            ram(1) <= "1111111111111111";
            ram(2) <= To_StdUlogicVector(0,16);
            ram(3) <= To_StdUlogicVector(6,16);
          end
        end
      end if;
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_1164_extensions.all;
--
-- Written by LL_to_VHDL at Tue May 3 09:08:37 1994
-- Parameterized Generator Specification to VHDL Code
--
-- LogicLib generator called: REGISTER_FILE
-- Passed Parameters are:
-- tinst name = huff_regl6by8
-- parameters are:
-- W = 8
-- D = 16
-- read ports = 1
--
-- huff_regl6by8 Entity Description
entity huff_regl6by8 is
port(
  DIN1: in std_logic_vector(7 downto 0);
  RADR1: in std_logic_vector(3 downto 0);
  WADR1: in std_logic_vector(3 downto 0);
  DOUT1: out std_logic_vector(7 downto 0);
  LD1, WE1: in std_logic;
  acdc: in std_logic
);
end huff_regl6by8;

COMPONENT: huff_regl6by8
DESCRIPTION: (architecture)
--
-- huff_regl6by8 Architecture Description
--
-- holds huffman val ptrs
--
architecture rtl of huff_regl6by8 is
  subtype rammemory is array (15 downto 0) of ramword;
  signal ram : rammemory;
begin
  REGISTER_FILE_read_Process: process(ram,RADR1)
    variable raddr : integer range 0 to 15;
    begin
      -- Process the read port number 1
      raddr := to_integer('0' & RADR1,0);
      DOUT1 <= ram(raddr);
      end process REGISTER_FILE_read_Process;
  REGISTER_FILE_write_Process: process(DIN1, WE1, LD1, WADR1, reset, acdc)
    variable waddr : integer range 0 to 15;
    variable load : std_logic;
    begin
      if (reset = '0') then
        if (acdc = '0') then
          -- 0 is for DC tables
          ram(0) <= To_StdUlogicVector(0,8);
          ram(1) <= To_StdUlogicVector(0,8);
          ram(2) <= To_StdUlogicVector(0,8);
          ram(3) <= To_StdUlogicVector(1,8);
          --
          ram(4) <= To_StdUlogicVector(1,16);
          ram(5) <= To_StdUlogicVector(1,16);
          ram(6) <= To_StdUlogicVector(1,16);
          ram(7) <= To_StdUlogicVector(1,16);
          ram(8) <= To_StdUlogicVector(1,16);
          ram(9) <= To_StdUlogicVector(0,16);
          ram(10) <= To_StdUlogicVector(0,16);
          ram(11) <= To_StdUlogicVector(0,16);
          ram(12) <= To_StdUlogicVector(4,16);
          ram(13) <= To_StdUlogicVector(4,16);
          ram(14) <= To_StdUlogicVector(4,16);
          ram(15) <= To_StdUlogicVector(4,16);
        else
          -- write mode? (need both WE1 and LD1 to be high)

          -- convert address to integer
          waddr := to_integer('0' & WADR1,0);
          load := LD1 and WE1;
          if (load = '1') then
            ram(waddr) <= DIN1;
          end if;
        end if;
        end process REGISTER_FILE_write_Process;
end rtl;

COMPONENT: huff_regl6by8
DESCRIPTION: (entity)
-- Purpose : ComputeMCUs
-- Author : Douglas A. Carpenter
-- Created : 26-MAY-1994
-- Revised :

-- library and use clauses
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164_extensions.all;

library my_packages;
use my_packages.package_1.all;

entity compute_MCUs is
port (...)
end;

COMPONENT: compute_MCUs
DESCRIPTION: (architecture)

-- Purpose : compute number of MCUs to decode module
-- Author : Douglas A. Carpenter
-- Created : 26-MAY-1994
-- Revised :

architecture compute_MCUs_arch of compute_MCUs is
signal D : std_logic_vector(2 downto 0);
signal Q : std_logic_vector(2 downto 0);
signal i_outs : std_logic_vector(1 downto 0);
signal i_data : std_logic_vector(31 downto 0);
begin

dscan_process : process(reset, clock)
variable width : std_logic_vector(15 downto 0);
variable height : std_logic_vector(15 downto 0);
begin
if (reset = '0') then
Q <= "000";
i_outs <= "00";
width := "00000000000000000000000000000000";
height := "00000000000000000000000000000000";
i_data <= "00000000000000000000000000000000";
extif ((clock'event) and (clock = '1')) and
(clock'last_value = '0') then
if (D = '000') then
if (computeNumMCUs = '0') then

end rtl;
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_1164_extensions.all;

-- Written by LL_to_VHDL at Tue May 3 09:08:37 1994
-- Parameterized Generator Specification to VHDL
--
-- LogicLib generator called: REGISTER_FILE
-- Passed Parameters are:
--  tinst name = huff_reg16by8
--  parameters are:
--    W = 8
--    D = 64
--    read_ports = 1
--
-- mem64by8 Entity Description
entity mem64by8 is
port (DIN1 : in std_ulogic_vector(7 downto 0);
      RADR1 : in std_ulogic_vector(5 downto 0);
      WADR1 : in std_ulogic_vector(5 downto 0);
      DOUT1 : out std_ulogic_vector(7 downto 0);
      LD1 : in std_logic;
      WE1 : in std_logic;
      mempreset : in std_ulogic);
end mem64by8;

architecture rtl of mem64by8 is
subtype ramword is std_ulogic_vector(7 downto 0);
type rammemory is array (63 downto 0) of ramword;
signal ram : rammemory;
begin
  REGISTER_FILE_read_Process: process(ram,RADR1)
  variable raddr1 : integer range 0 to 63;
type rammemory is array (63 downto 0) of ramword;
signal ram : rammemory;
begin
  REGISTER_FILE_read_Process: process(DIN1,WE1,LD1,RADR1,mempreset)
variable waddr : integer range 0 to 63;
variable load : std_ulogic;
begin
  if (mempreset = '0') then
    ram(1) <= "00000000";
    ram(2) <= "00000000";
    ram(3) <= "00000000";
    ram(4) <= "00000000";
    ram(5) <= "00000000";
    ram(6) <= "00000000";
    ram(7) <= "00000000";
    ram(8) <= "00000000";
    ram(9) <= "00000000";
    ram(10) <= "00000000";
    ram(11) <= "00000000";
    ram(12) <= "00000000";
    ram(13) <= "00000000";
    ram(14) <= "00000000";
    ram(15) <= "00000000";
    ram(16) <= "00000000";
    ram(17) <= "00000000";
end process;

--assign output values
D <= Q;
computeNumMCUsack <= i_outs(1);
latchMCUs <= i_outs(0);
umMCUs_in <= i_data;
end compute_MCUs_arch;

end COMPONENT;
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_1164_extensions.all;

-- Written by LL_to_VHDL at Tue May 3 09:08:37 1994
-- Parameterized Generator Specification to VHDL Code
--

COMPONENT:  mem64by16
DESCRIPTION:  

architecture rtl of mem64by16 is
subtype ramword is std_logic_vector(15 downto 0);
variable waddr : integer range 0 to 63;
variable raddr1 : integer range 0 to 63;
variable load : std_logic;
begin
  REGISTER_FILE_write_Process: process(ram,RADR1)
  variable raddr1 : integer range 0 to 63;
  begin
    -- convert address to integer
    waddr := to_integer('0' & WADR1,0);
    if (load = '1') then
      ram(waddr) <= '0';
    end if;
  end process REGISTER_FILE_write_Process;
end rtl;

-- LogicLib generator called: REGISTER_FILE
-- Passed Parameters are:
-- tinst name = huff_reg16by8
-- parameters are:
-- W = 16
-- D = 64
-- read ports = 1
--
-- mem64by16 Entity Description
entity mem64by16 is
  port
    (DIN1 : in std_logic_vector(15 downto 0);
     RADR1 : in std_logic_vector(5 downto 0);
     WADR1 : in std_logic_vector(5 downto 0);
     DOUT1 : out std_logic_vector(15 downto 0);
     LD1 : in std_logic;
     WE1 : in std_logic;
     memreset : in std_logic);
end mem64by16;

architecture rtl of mem64by16 is
begin
  REGISTER_FILE_read_Process: process(ram,RADR1)
  variable raddr1 : integer range 0 to 63;
  begin
    -- Process the read port number 1
    raddr1 := to_integer('0' & RADR1,0);
    DOUT1 <= ram(raddr1);
  end process REGISTER_FILE_read_Process;
end rtl;
component load_coeff

description:

-- Purpose : load coeff module
-- Author : Douglas A. Carpenter
-- Created : 27-MAY-1994
-- Revised :

architecture load_coeff_arch of load_coeff is

signal D : std_logic_vector(3 downto 0);
signal Q : std_logic_vector(3 downto 0);
signal i_comp : std_logic;
signal i_afull : std_logic;
signal i_load : std_logic;

constant FULL : std_logic := '1';
-- ok to unload
constant NOT_FULL : std_logic := '0';
-- don't unload
constant UNLOADING : std_logic := '1';
-- don't fill
constant DONE_UNLOADING : std_logic := '0';
-- ok to fill

begin

load_process : process(reset, clock)
variable numMCU : std_logic_vector(31 downto 0);
begin

if (reset = '0') then

Q <= "0000";
i_comp <= '0';
i_afull <= NOT_FULL;
i_load <= '0';
numMCU := "00000000000000000000000000000000";
else if ((clock='event) and (clock = '1') and
(clock'last_value = '0')) then

-- 0. Send not_full ->
if (D = "0000") then

if (startscanning = '0') then

Q <= "0000";
i_comp <= '0';
i_afull <= NOT_FULL;
i_load <= '0';
else

Q <= "0011";
i_comp <= '0';
i_afull <= NOT_FULL;
numMCU <= numMCUout;
end if;
end if;

-- 1. Check if all MCU's have been loaded

elsif (D = "0001") then

end if;

end process load_process;
end rtl;

---
if (numMCU = "000000000000000000000000000000000") then
  Q <= "1101"; -- complete
  i_comp <= '1';
  i_afull <= NOT_FULL;
else
  Q <= "0011"; -- do next MCU
  i_afull <= NOT_FULL;
end if;
elsif (D = "0011") then -- check if A buffer is empty
  if (a_empty_right = DONE_UNLOADING) then
    Q <= "1111"; -- got done unloading
    i_load <= '1';
    i_afull <= NOT_FULL;
  else
    Q <= "0011"; -- still waiting for
  end if;
  done_unloading, stay at this step
  i_load <= '0';
  i_afull <= NOT_FULL;
end if;
elsif (D = "1111") then
  if (do_load_ack = '0') then
    Q <= "0010"; -- ready to load,
    therefore load
    i_load <= '1';
    i_afull <= NOT_FULL;
  else
    Q <= "1111"; -- wait till ready
to load, ack = 0
    i_load <= '0';
    i_afull <= NOT_FULL;
end if;
elsif (D = "0110") then
  if (do_load_ack = '1') then
    Q <= "0100"; -- load complete
    i_afull <= FULL; -- signal that
  end if;
else
  Q <= "0110"; -- load complete ack
  has not dropped back to 0.
  i_load <= '0';
  i_afull <= NOT_FULL;
end if;
elsif (D = "1010") then
  if (startscan = '1') then
    Q <= "1010"; -- wait until
    startscan drops, acking complete seen
    i_comp <= '1';
    i_afull <= NOT_FULL;
  else
    Q <= "0000"; -- complete seen
    i_comp <= '0';
    i_afull <= NOT_FULL;
  end if;
elsif (D = "0010") then
  load_coeff_complete <= i_comp;
  a_full_right <= i_afull;
  do_load <= i_load;
end load_coeff_arch;

COMPONENT: dequant_coeff
DESCRIPTION: (entity)

-- Purpose : DeQuantize Coefficients
-- Author : Douglas A. Carpenter
-- Created : 27-MAY-1994
-- Revised :

-- library and use clauses
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164_extensions.all;

library my_packages;
use my_packages.package_l.all;

entity dequant_coeff is
  port (1 -- inputs
    clock : in
    std_ulogic;
    reset : in
    std_ulogic;
    startscan : in
    std_ulogic;
    load_coeff_complete : in
    std_ulogic;
    dequant_coeff_complete : out
    std_ulogic;
    a_full_left : in
    std_ulogic;
    a_empty_left : out
    std_ulogic;
    a_full_right : out
    std_ulogic;
    do_dequant : out
    std_ulogic;
    do_dequant_unload_ack : in
    std_ulogic;
  );

A-79
COMPONENT: dequant_coef

DESCRIPTION: (architecture)

-- Purpose: dequant coeff module
-- Author: Douglas A. Carpenter
-- Created: 27-MAY-1994

architecture dequant_coef_arch of dequant_coef is
signal D : std_ulogic_vector(3 downto 0);
signal Q : std_ulogic_vector(3 downto 0);
signal i_comp : std_ulogic;
signal i_aempty_left : std_ulogic;
signal i_afull_right : std_ulogic;
signal i_outs : std_ulogic_vector(0 downto 0);

constant FULL : std_ulogic := '1';
constant NOT_FULL : std_ulogic := '0';
constant UNLOADING : std_ulogic := '1';
constant DONE_UNLOADING : std_ulogic := '0';

begin

deoquant_process : process(reset, clock) begin
if (reset = '0') then
  Q <= "0000";
i_comp <= '0';
i_aempty_left <= DONE_UNLOADING;
i_afull_right <= NOT_FULL;
i_outs <= "0";
elsif ((clock'event) and (clock = '1') and
(clock'last_value = '0')) then
  if (D = "0000") then
    if (startscan = '0') then
      Q <= "0000";
    else
      Q <= "0001";
    end if;
  end if;
end if;

if (D = "0001") then
  if (load_coeff_complete = '1') then
    if (D = "0001") then
      if (a_full_left = FULL) and
(a_empty_right = DONE_UNLOADING) then
      Q <= "0011";
    else
      if (a_full_left = NOT_FULL) then
        Q <= "0010";
      else
        Q <= "0011";
      end if;
    end if;
  else
    if (N0T_FULL then
      Q <= "0001";
    else
      Q <= "0000";
    end if;
  end if;
edu

if (D = "0101") then
  if (a_empty_left = DONE_UNLOADING) then
    Q <= "1011";
  else
    Q <= "0011";
  end if;
end if;
else
  if (DONE_UNLOADING) then
    Q <= "1011";
  else
    Q <= "0011";
  end if;
end if;
end if;
end if;
edu

else
  if (D = "1011") then
    if (a_full_left = NOT_FULL) then
      Q <= "1011";
    else
      Q <= "0011";
    end if;
  else
    if ( DONEUNLOADING) then
      Q <= "1011";
    else
      Q <= "0011";
    end if;
  end if;
else
  if (D = "1111") then
    if (do_dequant_ack = '0') and
(do_dequant_unload_ack = '0') then
      Q <= "1001";
    else
      Q <= "1111";
    end if;
else
  if (D = "1101") then
    if (do_dequant_unload_ack = '1') then
      Q <= "1101";
    else
      Q <= "1111";
    end if;
else
  if (D = "1100") then
    if (do_dequant_ack = '0') and
(do_dequant_unload_ack = '0') then
      Q <= "0010";
    else
      Q <= "1100";
    end if;
else
  Q <= "0001";
end if;
end if;
edu

for in/out buffers
  i_outs <= "0";
end if;
edu
else
  Q <= "0011";
end if;
edu
else
  Q <= "1001";
end if;
edu
end if;
edu
end if;
edu
end if;
edu
end if;
edu
end if;
edu
end if;
edu
end if;
edu
end if;
edu
end if;
edu
end if;
edu
end if;
edu
end if;
edu
end if;
edu
end if;
edu
end if;
edu
end if;
edu
end if;
edu
end if;
edu
end if;
edu
end process
end dequant_coef_arch;
end dequant_coef;
else if (D = "0010") then  
  Q <= "0110";  
  -- done  
unloading/loading  
i_empty_left <= DONE_UNLOADING;  
-- signal buffer empty  
i_afull_right <= NOT_FULL;  
-- signal that data is not loaded at output  
i_outs <= "0";  
elsif (D = "0100") then  
  if (a_empty_right = DONE_UNLOADING) then  
    -- wait for signals to change  
    i_empty_left <= DONE_UNLOADING;  
    -- signal buffer empty  
    i_afull_right <= FULL;  
    -- signal that data is loaded at output  
    i_outs <= "0";  
  else  
    Q <= "0110";  
    i_empty_left <= DONE_UNLOADING;  
signal buffer empty  
i_afull_right <= NOT_FULL;  
-- signal that data is loaded at output  
i_outs <= "0";  
else  
  Q <= "0110";  
  i_empty_left <= DONE_UNLOADING;  
signal buffer empty  
i_afull_right <= NOT_FULL;  
-- signal that data is not loaded at output  
i_outs <= "0";  
end if;  
elsif (D = "0100") then  
  if (a_empty_right = DONE_UNLOADING) then  
    Q <= "0001";  
    i_empty_left <= DONE_UNLOADING;  
signal buffer empty  
i_afull_right <= NOT_FULL;  
-- signal that data is loaded at output  
i_outs <= "0";  
else  
  Q <= "0100";  
  i_empty_left <= DONE_UNLOADING;  
signal buffer empty  
i_afull_right <= NOT_FULL;  
-- signal that data is not loaded at output  
i_outs <= "0";  
end if;  
elsif (D = "1010") then  
  if (startscan = '1') then  
    Q <= "1010";  
    -- wait for ack  
    i_comp <= '1';  
    i_empty_left <= DONE_UNLOADING;  
    i_afull_right <= NOT_FULL;  
    i_outs <= "0";  
  else  
    Q <= "0000";  
    i_comp <= '0';  
    i_empty_left <= DONE_UNLOADING;  
    i_afull_right <= NOT_FULL;  
    i_outs <= "0";  
  end if;  
else  
  Q <= "0000";  
  i_comp <= '0';  
  i_empty_left <= DONE_UNLOADING;  
  i_afull_right <= NOT_FULL;  
  i_outs <= "0";  
end if;  
end process dequant_process;  

-- assign output values  
D <= Q;  
dequant_coeff_complete <= i_comp;  
end_dequant <= i_outs(0);  
end dequant_coeff_arch;  

COMPONENT: idct_coeff
DESCRIPTION: (entity)

-- Purpose : IDCT Coefficients
-- Author : Douglas A. Carpenter
-- Created : 27-MAY-1994
-- Revised :

-- library and use clauses
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164_extensions.all;
library my_packages;
use my_packages.package_1.all;

entity idct_coeff is
  port (
    -- inputs
    std_ulogic;  
    clock : in
    std_ulogic;  
    reset : in
    std_ulogic;  
    startscan : in
    std_ulogic;  
    dequant_coeff_complete : in
    std_ulogic;  
    idct_coeff_complete : out
    std_ulogic;  
    a_full_left : in
    std_ulogic;  
    a_empty_left : out
    std_ulogic;  
    a_full_right : out
    std_ulogic;  
    a_empty_right : in
    std_ulogic;  
    do_IDCT : out
    std_ulogic;  
    do_IDCT_unload_ack : in
    std_ulogic;  
    do_IDCT_done_ack : in
    std_ulogic;  
  );
end idct_coeff;

COMPONENT: idct_coeff
DESCRIPTION: (architecture)

-- Purpose : idct coeff module
-- Author : Douglas A. Carpenter
-- Created : 27-MAY-1994
-- Revised :

architecture idct_coeff_arch of idct_coeff is
signal D : std_ulogic_vector(3 downto 0);
signal Q : std_ulogic_vector(3 downto 0);
signal i_comp : std_ulogic;
signal i_empty_left : std_ulogic;
signal i_afull_right : std_ulogic;
signal i_outs : std_ulogic_vector(0 downto 0);

constant FULL : std_ulogic := '1';
constant NOT_FULL : std_ulogic := '0';
constant UNLOADING : std_ulogic := '1';
constant DONE_UNLOADING : std_ulogic := '0';

idct_process : process(reset, clock)
begin
  if (reset = '0') then
    Q <= "0000";
    i_comp <= '0';
    i_empty_left <= DONE_UNLOADING;
    i_afull_right <= NOT_FULL;
    i_outs <= "0";
  elsif (clock'event) and (clock = '1') and
  (clock'last_value = '0') then
    -- 1. Send <= done_unload and
    Q <= "0001";
    i_empty_left <= DONE_UNLOADING;
    i_outs <= "0";
  end if;

  done_unloading <=
  elsif (D = "0001") then
    if (startscan = '0') then
      Q <= "0100";
      i_comp <= '1';
      i_empty_left <= DONE_UNLOADING;
      i_afull_right <= NOT_FULL;
      i_outs <= "0";
    else
      Q <= "0011";
      i_empty_left <= DONE_UNLOADING;
      i_outs <= "0";
    end if;

  end if;

  done_unloading <=
  elsif (D = "0011") then
    if ((a_full_left = FULL) and
    (a_empty_right= DONE_UNLOADING)) then
      Q <= "0010";
      -- data ready to
    load, out buffer empty
      i_empty_left <= UNLOADING;
      i_afull_right <= NOT_FULL;
      signal that data is not loaded at output
      i_outs <= "0";
    else
      Q <= "0001";
      -- wait for in/out
    buffers
      i_outs <= "0";
    end if;

  else
    -- 4. Send <=unloading and
    not_full <=
    elsif (D = "0100") then
      i_empty_left <= UNLOADING;
      i_afull_right <= NOT_FULL;
      signal that data is not loaded at output
      i_outs <= "0";
      -- signal to do
    IDCT
    -- 5. Wait for ->not_full
  end if;

  if (a_full_left = NOT_FULL) then
    Q <= "1111";
    -- 6. Got ->
  not_full
    elsif (D = "1011") then
    if (a_full_left = NOT_FULL) then
      Q <= "0111";
      -- 7. unload
    elsif (D = "1111") then
      if ((do_IDCT_done_ack = '0') and
      (do_IDCT_unload_ack = '0')) then
        Q <= "1101";
        -- data ready to
      unload;
      elsif (D = "1101") then
        if (do_IDCT_unload_ack = '1') then
          Q <= "1110";
          i_outs <= "1";
          i_empty_left <= DONE_UNLOADING;
        end if;

      else
        if (do_IDCT_done_ack = '1') and
        (do_IDCT_unload_ack = '1') then
          Q <= "1110";
          -- data was IDCT
        else
          Q <= "1110";
          -- data was not
        end if;

      elsif (D = "1010") then
        if ((do_IDCT_done_ack = '0') and
        (do_IDCT_unload_ack = '0')) then
          Q <= "0010";
          -- data done
        else
          Q <= "1010";
          -- data not done
        end if;

      elsif (D = "0010") then
        Q <= "0110";
        -- done
      end if;

      elsif (D = "0110") then
        Q <= "1001";
        -- unload/load
      elsif (D = "0101") then
        Q <= "1011";
        -- load, out buffer empty
      end if;

      else
        Q <= "0101";
        -- signal that data is loaded at output
      i_outs <= "0";
    end if;
end if;
end if;
end process;

A-82
if (a_empty_right = DONE_UNLOADING) then
  Q <= "0110"; -- wait for signals to change
  i_aempty_left <= DONE_UNLOADING; -- signal
signal buffer empty
  i_afull_right <= FULL; -- signal
  that data is not loaded at output
  i_outs <= "0";
elsif (a_empty_right = UNLOADING) then
  Q <= "0100"; -- signals asked,
go to wait until right is empty.
  i_aempty_left <= DONE_UNLOADING; -- signal
signal buffer empty
  i_afull_right <= NOT_FULL; -- signal
  that data is loaded at output
  i_outs <= "0";
  i_aempty_left <= DONE_UNLOADING; -- signal
signal buffer empty
  i_afull_right <= FULL; -- signal
  that data is not loaded at output
  i_outs <= "0";
  end if;
else
  Q <= "0110";
  i_aempty_left <= DONE_UNLOADING; -- signal
  that data is not loaded at output
  i_outs <= "0";
  end if;
elsif (D = "0100") then
  if (a_empty_right = DONE_UNLOADING) then
    Q <= "0001"; -- go to next MCU
  i_aempty_left <= DONE_UNLOADING; -- signal
  else
    Q <= "0100";
  i_aempty_left <= DONE_UNLOADING; -- signal
  signal buffer empty
  i_afull_right <= NOT_FULL; -- signal
  that data is loaded at output
  i_outs <= "0";
  else
    Q <= "0000";
    i_comp <= '0';
  i_aempty_left <= DONE_UNLOADING;
  i_afull_right <= NOT_FULL;
  i_outs <= "0";
  end if;
  else
    Q <= "0000";
    i_comp <= '0';
    i_aempty_left <= DONE_UNLOADING;
    i_afull_right <= NOT_FULL;
    i_outs <= "0";
  end if;
end if;
end process idct_process;
-- assign output values
D <= Q;
idct_coeff_complete <= i_comp;
a_empty_left <= i_aempty_left;
a_full_right <= i_afull_right;
dIDCT <= i_outs(0);
end idct_coeff_arch;

begin
  -- Name : dequantize_entity
  -- Created : 22-APR-1994
  -- Revised :
  -- library and use clauses
  library ieee;
  use ieee.std_logic_1164.all;
  use ieee.std_logic_1164_extensions.all;
  use std.math.all;
  library my_packages;
  use my_packages.package_1.all;
  entity idct is
    port (clock in std_ulogic;
          reset : in std_ulogic;
          do_idct in std_ulogic;
          do_idctUnload_ack out std_ulogic;
          do_idct_done_ack : out std_ulogic;
          DeqToIDCT_RADRl : out std_ulogic_vector(5 downto 0);
          DeqToIDCT_RADRl : out std_ulogic_vector(15 downto 0);
          IDCT_ToUnload_DINl : out std_ulogic_vector(3 downto 0);
          IDCT_ToUnload_DINl : out std_ulogic_vector(7 downto 0);
          IDCT_ToUnload_WADRl : out std_ulogic_vector(5 downto 0);
          IDCT_ToUnload_WEl : out std_ulogic;
          IDCT_ToUnload_WEl : out std_ulogic;
          IDCT_ToUnload_WEl : out std_ulogic)
      end entity idct;
  architecture behav_arch of idct is
    signal i_outs : std_ulogic_vector(3 downto 0);
    signal i_data : std_ulogic_vector(7 downto 0);
    signal load_addr : std_ulogic_vector(5 downto 0);
    constant i_delay : time := 15 ns;
    begin
      IDCT : process

A-83
```verilog
type rammemory is array (63 downto 0)
of ramword;
subtype iramword is integer range 0 to 255;
type irammemory is array (63 downto 0)
of iramword;
  variable ram : rammemory;
  variable sss : rammemory;
  variable tmp : integer range -32768 to 32768;
  variable ftmp : real range -32768.0 to 32768.0;
  variable tworoot : real range 0.0 to 10.0;
  variable sum : real range -32768.0 to 32768.0;
  variable a : real range -32768.0 to 32768.0;
  variable b : real range -32768.0 to 32768.0;
  variable Cu : real range 0.0 to 10.0;
  variable Cv : real range 0.0 to 10.0;
  variable x,y,u,v,NUMBER : integer range 0 to 64;
  variable ss : irammemory;
begin
  i_outs <= "000000";
  load_addr <= "00000000";
  wait for lns;
  wait until ((reset = '0') or (do_idct'event));
  if (reset = '0') then
    i_outs <= "000000000";
  assert FALSE
  report "resetting until reset complete in IDCT" severity NOTE;
  load_addr <= "00000000";
  wait for i_delay;
  NUMBER := 0;
  wait for lns;
  LoadLoop : while (NUMBER < 64) loop
    tmp := To_Integer(DegToIDCT_DOUT1);
    wait for 5 ns;
    ram(NUMBER) := tmp;
    wait for i_delay;
    load_addr <= load_addr + "0000001";
    wait for i_delay;
    NUMBER := NUMBER + 1;
    wait for 1 ns;
  end loop LoadLoop;
  wait for i_delay;
  i_outs <= "00001";
  tworoot := 1.0 / sqrt(2.0);
  wait for i_delay;
  y := 0;
  wait for i_delay;
  Yloop : while (y < 8) loop
    x := 0;
  ```
-- unload IDCT data to next buffer
load_addr <= "000000";
for y in 0 to 7 loop
    for x in 0 to 7 loop
        i_data <= To_StdLogicVector(ss((y*8)+x),8);
        wait for i_delay;
        i_outs <= "0101";
        wait for i_delay;
        i_outs <= "1101";
        wait for i_delay;
        Q <= "0001";
        wait for i_delay;
        load_addr <= load_addr + "000001";
        wait for i_delay;
    end loop;
end loop;
wait for 15 ns;
i_outs <= "0011";
wait until (do_idct = '0');
wait for 15ns;
i_outs <= "0000";
else
    i_outs <= "0000";
end if;
end process IDCT;

-- assign outputs
do_idct_unload_ack <= i_outs(0);
do_idct_done_ack <= i_outs(1);
IDCT_ToUnload_LDl <= i_outs(2);
IDCT_ToUnload_WEl <= i_outs(3);
DeqToIDCT_RADRl <= load_addr;
IDCT_ToUnload_WADRl <= load_addr;
IDCT_TOUnload_DINl <= i_data;
end behav_arch;

-- Purpose : Unload Coefficients
-- Author : Douglas A. Carpenter
-- Created : 27-MAY-1994
-- Revised :

-- library and use clauses
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164_extensions.all;
library my_packages;
use my_packages.package_1.all;

entity unload_coeff is
    port ( -- inputs
        clock : in
        std_ulogic;
        reset : in
        std_ulogic;
        startscan : in
        std_ulogic;
        scancomplete : out
        std_ulogic;
        idct_coeff_complete : in
        std_ulogic;
        a_full_left : in
        std_ulogic;
        a_empty_left : out
        std_ulogic;
        do_unload : out
        std_ulogic;
        do_unload_ack : in
        std_ulogic );
end unload_coeff;

-- Purpose : unload coeff module
-- Author : Douglas A. Carpenter
-- Created : 27-MAY-1994
-- Revised :

architecture unload_coeff_arch of unload_coeff is
    signal D : std_ulogic_vector(3 downto 0);
signal Q : std_ulogic_vector(3 downto 0);
signal i_comp : std_ulogic;
signal i_aempty : std_ulogic;
signal i_outs : std_ulogic_vector(0 downto 0);
constant FULL : std_ulogic := '1';
constant NOT_FULL : std_ulogic := '0';
constant UNLOADING : std_ulogic := '1';
constant DONE_UNLOADING : std_ulogic := '0';
begin
    unload_process : process(reset, clock)
    begin
        if (reset = '0') then
            Q <= "0000";
i_comp <= '0';
i_aempty <= DONE_UNLOADING;
i_outs <= "0";
        elsif ((clock'event) and (clock = '1') and
                (clock'last_value = '0')) then
            1. Send done_unloading <=
            if (D = "0000") then
                Q <= "0000";
i_comp <= '0';
i_aempty <= DONE_UNLOADING;
i_outs <= "0";
            else
                Q <= "0001";
i_aempty <= DONE_UNLOADING;
i_outs <= "0";
            end if;
            2. wait for -> full
        elsif (D = "0001") then
            if (idct_coeff_complete = '1') then
                Q <= "1010";
                2. recvd
            complete signal
            i_aempty <= DONE_UNLOADING;
i_comp <= '1';
        end if;
    end if;
end unload_process;

end unload_coeff_arch;
else
if (a_full_left = FULL) then
Q <= "0011"; -- 3. got ->
end if;
end if;
end if;
-> full
i_empty <= UNLOADING;
else
Q <= "0001"; -- 2. wait for
end if;
-> full
i_empty <= DONE_UNLOADING;
end if;
end if;
-- 4. Send unloading
elsif (D = "0011") then
Q <= "1110"; -- unload
i_empty <= UNLOADING;
elsif (D = "1110") then
if (a_full_left = NOT_FULL) then
Q <= "1111"; -- 5. Wait for not_full
end if;
elsif (D = "1101") then
if (do_unload_ack = '0') then
Q <= "1101";
end if;
exsif (D = "1100") then
if (do_unload_ack = '0') then
Q <= "0010";
end if;
exsif (D = "0010") then
Q <= "0001";
else
Q <= "1110";
end if;
not_full
elsif (D = "1111") then
Q <= "1110";
end if;
-- 6. got
not_full
elsif (D = "1110") then
Q <= "1110";
end if;
-- 7. unload
elsif (D = "1101") then
if (do_unload_ack = '1') then
Q <= "1100";
end if;
end if;
dropped back to 0
elsif (D = "0010") then
Q <= "0001";
end if;
else
Q <= "0000";
end if;
end if;
end if;
end if;
else
Q <= "0000";
end if;
end if;
end if;
end if;
end if;
end process unload_process;
-- assign output values
D <= Q;
scancomplete <= i_comp;
a_empty_left <= i_empty;
do_unload <= i_outs(0);
end unload_coeff_arch;

COMPONENT: do_unload
DESCRIPTION: (entity)
-- Purpose : Do Load Coefficients
-- Author : Douglas A. Carpenter
-- Created : 27-MAY-1994
-- Revised :

-- library and use clauses
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164_extensions.all;
library my_packages;
use my_packages.package_l.all;

entity do_unload is
port ( inputs clock reset do_unload do unload ack std_ulogic; std_ulogic; std_ulogic; std_ulogic; std_ulogic; std_ulogic_vector(5 downto 0); std_ulogic_vector(7 downto 0); std_ulogic; std_ulogic_vector(7 downto 0); std_ulogic_vector(7 downto 0); std_ulogic; std_ulogic_vector(7 downto 0); std_ulogic_vector(7 downto 0); std_ulogic_vector(7 downto 0); std_ulogic_vector(7 downto 0); std_ulogic_vector(7 downto 0); std_ulogic_vector(7 downto 0); std_ulogic_vector(7 downto 0); std_ulogic_vector(7 downto 0); std_ulogic_vector(7 downto 0));
result_data_valid : out std_ulogic;
result_data : out std_ulogic;
result_data_ack : in std_ulogic);
end do_unload;

COMPONENT: do_unload
DESCRIPTION: (architecture)
-- Purpose : do unload module
-- Author : Douglas A. Carpenter
-- Created : 22-JUN-1994
-- Revised :

architecture do_unload_arch of do_unload is
signal D : std_ulogic_vector(3 downto 0); signal Q : std_ulogic_vector(3 downto 0); signal i_outs : std_ulogic_vector(1 downto 0); signal i_data : std_ulogic_vector(7 downto 0); signal i_radr : std_ulogic_vector(5 downto 0);
unload_process : process(reset, clock)
begin
if (reset = '0') then
  Q <= "0000";
i_outs <= "00";
i_radr <= "000000";
elsif (clock'event) and (clock = '1') and (clock'last_value = '0') then
  if (D = "0000") then
    i_outs <= "00";
i_radr <= "000000";
  elsif (D = "0001") then
    Q <= "0001";
i_outs <= "00";
i_radr <= "000000";
  elsif (D = "0011") then
    Q <= "0010";
i_outs <= "00";
i_data <= IDCT_ToUnload_DOUT1;
  elsif (D = "0101") then
    Q <= "0110";
i_outs <= "10";
  elsif (D = "0100") then
    Q <= "0101";
i_outs <= "00";
  elsif (D = "0110") then
    Q <= "0100";
i_outs <= "00";
  elsif (D = "0101") then
    i_radr <= i_radr + "000001";
    Q <= "0111";
  elsif (D = "0111") then
    Q <= "1000";
i_outs <= "01";
    i_radr <= "000000";
  elsif (D = "1000") then
    i_radr <= i_radr + "000000";
  else
    Q <= "0001";
i_outs <= "00";
  end if;
elsif (D = "0000") then
  Q <= "0000";
i_outs <= "00";
end if;
end process unload_process;

-- assign outputs
D <= Q;
result_data <= i_data;
result_data_valid <= i_outs(1);
IDCT_ToUnload_RADR1 <= i_radr(5 downto 0);
do_unload_ack <= i_outs(0);
end do_unload_arch;

COMPONENT: do_load_coeff
DESCRIPTION: (entity)

-- Purpose : Do Load Coefficients
-- Author : Douglas A. Carpenter
-- Created : 27-MAY-1994
-- Revised :

-- library and use clauses
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164_extensions.all;
library my_packages;
use my_packages.package_1.all;

entity do_load_coeff is
  port ( -- inputs
    clock : in std_logic;
    reset : in std_logic;
    do_load : in std_logic;
    do_load_ack : out std_logic;
    loadDC : out std_logic;
    loadDC_ack : out std_logic;
    loadAC : out std_logic;
    loadAC_ack : out std_logic;
  );
end do_load_coeff;

COMPONENT: do_load_coeff
DESCRIPTION: (architecture)

-- Purpose : do_load coeff module
-- Author : Douglas A. Carpenter
-- Created : 27-MAY-1994
-- Revised :

architecture do_load_arch of do_load_coeff is
signal D : std_logic_vector(2 downto 0);
signal Q : std_logic_vector(2 downto 0);
signal i_outs : std_logic_vector(2 downto 0);

begin
  do_load_process : process(reset, clock)
  begin
    if (reset = '0') then
      Q <= "000";
i_outs <= "00";
    elsif (clock'event) and (clock = '1') and (clock'last_value = '0') then
      if (D = "000") then
        Q <= "000";
        i_outs <= "000";
        -- wait for do load
      else
      end if;
  end process do_load_process;
Q <= "001";  -- got do load, send
load DC coeff
  i_outs <= "010";
end if;
elsif (D = "001") then
  if (loadDC_ack = '0') then
    Q <= "001";  -- wait for load DC
  ack
    i_outs <= "010";
  else
    Q <= "011";  -- got load DC ack,
go to wait for loadDCack to turn off
    i_outs <= "000";
  end if;
elsif (D = "011") then
  if (loadDC_ack = '1') then
    Q <= "011";
  ack
    i_outs <= "000";  -- load DC ack
  still on
  else
    Q <= "010";  -- load DC ack
  done, go to load AC coeff
  end if;
elsif (D = "100") then
  if (loadAC_ack = '0') then
    Q <= "100";
  ack
  i_outs <= "100";  -- load AC ack
  still on
  else
    Q <= "110";  -- got load AC ack,
go to wait for loadACack to turn off
    i_outs <= "000";
  end if;
elsif (D = "110") then
  if (loadAC_ack = '1') then
    Q <= "110";
  ack
    i_outs <= "000";  -- load AC ack
  still on
  else
    Q <= "100";  -- got load AC ack,
go to wait for load AC coeff
    i_outs <= "100";
  end if;
elsif (D = "100") then
  if (do_load = '1') then
    Q <= "100";  -- do_load still on,
wait till off
    i_outs <= "100";
  else
    Q <= "000";
    i_outs <= "000";
  end if;
else
    Q <= "000";
    i_outs <= "000";
  end if;
end process do_load_process;
D <= Q;
do_load_ack <= i_outs(2);
loadDC <= i_outs(1);
loadAC <= i_outs(0);
end do_load_arch;

-- Created : 27-MAY-1994
-- Revised :

-- library and use clauses
library mgc_portable, ieee;
use mgc_portable.qsim_logic.all;
use mgc_portable.qsim_relations.all;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164_extensions.all;
library my_packages;
use my_packages.package_l.all;
entity do_loadDC is
  port (  
    clock : in std_ulogic;
    reset : in std_ulogic;
    loadDC : in std_ulogic;
    loadDC_ack : out std_ulogic;
    decode : out std_ulogic;
    decode_ack : in std_ulogic;
    receive : out std_ulogic;
    receive_ack : in std_ulogic;
    extend : out std_ulogic;
    extend_ack : in std_ulogic;
    Tdecode in std_logic_vector(7 downto 0);
    DIFF in std_logic_vector(7 downto 0);
    RetV in std_logic_vector(7 downto 0);
    LoadDINl in std_logic_vector(7 downto 0);
    LoadWRDl in std_logic_vector(5 downto 0);
    LoadLEl in std_logic;
    LoadWEl in std_logic;
  );
end do_loadDC;

COMPONENT: do_loadDC
DESCRIPTION: (architecture)

-- Purpose : do_loadDC module
-- Author : Douglas A. Carpenter
-- Created : 02-JUN-1994
-- Revised :

architecture do_loadDC_arch of do_loadDC is
  signal D : std_logic_vector(7 downto 0);
  signal Q : std_logic_vector(7 downto 0);
  signal i_outs : std_logic_vector(5 downto 0);
  signal PRED : integer range -1024 to 1024;
  signal TMP : integer range -1024 to 1024;
  signal i_data : std_logic_vector(7 downto 0);
begin
  do_loadDC_process : process(reset, clock)
variable IDIFF : std_logic_vector(7 downto 0);
variable T : std_logic_vector(7 downto 0);
begin
  if (reset = '0') then
    Q <= "0000";
    i_outs <= "000000";
    PRED <= 0;
  elsif ((clock'event) and (clock = '1') and
    (clock'last_value = '0')) then
    if (D = "0000") then

-- Author : Douglas A. Carpenter
if (loadDC = '1') then
  Q <= "0001";
  -- got load, go to next step
end if;

elsif (D = "0001") then
  if (decode_ack = '1') then
    Q <= "0011";
    -- ensure that decode is not in use
  else
    Q <= "0000";
    i_outs <= "000000";
  end if;
else
  Q <= "0011";
  i_outs <= "000100";
end if;

elsif (D = "0011") then
  if (decode_ack = '0') then
    Q <= "0010";
  else
    Q <= "000000";
    i_outs <= "000000";
  end if;
end if;

elsif (D = "0100") then
  if (decode_ack = '0') then
    Q <= "0100";
  else
    Q <= "001000";
    i_outs <= "001000";
  end if;
end if;

elsif (D = "0110") then
  if (decode_ack = '1') then
    Q <= "0110";
  else
    Q <= "000000";
    i_outs <= "000000";
  end if;
end if;

elsif (D = "1010") then
  if (loadDC = '1') then
    Q <= "1010";
    -- wait for loadDC
  else
    Q <= "000000";
    i_outs <= "000000";
  end if;
end if;

elsif (D = "0111") then
  if (extend_ack = '0') then
    Q <= "1111";
    -- extend ack turned off, signal done
  else
    Q <= "00111";
    -- wait for it
  end if;
elsif (D = "1111") then
  Q <= "11101";
  i_outs <= "000000";
  TMP <= PRED + To_Integer(RetV);
elsif (D = "1100") then
  Q <= "1100";
  i_outs <= "010000";
  PRED <= TMP;
  i_data <= ToStdUlogicVector(TMP, 8);
elsif (D = "1000") then
  Q <= "1010";
  i_outs <= "000001";
elsif (D = "1010") then
  Q <= "1010";
  i_outs <= "000000";
end if;

end process do_loadDC_process;

D <= Q;
loadDC_ack <= i_outs(0);
decode <= i_outs(3);
receive <= i_outs(2);
extend <= i_outs(1);
LoadRAD1 <= "000000";
LoadWE1 <= i_outs(5);
LoadLD1 <= i_outs(4);
LoadDIN1 <= i_data;
end if;
end if;
end if;
end if;

-- Author: Douglas A. Carpenter
-- Created: 02-JUN-1994
-- Revised:

-- library and use clauses
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164_extensions.all;

library my_packages;
use my_packages.package_1.all;

entity do_loadAC is
port ( -- inputs
  clock : in std_ulogic;
  reset : in std_ulogic;
  loadAC : in std_ulogic;
  huff_outSEL : out std_ulogic;
  std_ulogic_vector(0 downto 0);
end
doloadAC;

COMPONENT: do_loadAC
DESCRIPTION: (architecture)

-- Purpose : do_loadAC module
-- Author : Douglas A. Carpenter
-- Created 02-JUN-1994
-- Revised :

architecture do_loadAC_arch of do_loadAC is
signal D : std_ulogic_vector(4 downto 0);
signal Q : std_ulogic_vector(4 downto 0);
signal i_outs : std_ulogic_vector(7 downto 0);
signal k : std_ulogic_vector(5 downto 0);
signal i_data : std_ulogic_vector(7 downto 0);
signal SSSS : std_ulogic_vector(7 downto 0);
signal RS : std_ulogic_vector(5 downto 0);

begin
doloadAC_process : process(reset, clock)
begin
  if (reset = '0') then
    Q <= "00000";
    k <= "000001";
    i_data <= "00000000";
  else if (D = "00001") then
    Q <= "00011";
    -- start
    i_outs <= "10000010";
    -- turn on AC
    sel
    k <= "00001";
    i_data <= "00000000";
  else
    Q <= "00000";
    -- wait for do load
    i_outs <= "10000000";
    k <= "000011";
    i_data <= "00000000";
    end if;
  end if;
elsif (D = "00011") then
  Q <= "00011";
  -- start
  i_outs <= "10000010";
  -- turn on AC
  sel
  k <= "00001";
  i_data <= "00000000";
else
  Q <= "00000";
  -- wait for do load
  i_outs <= "10000000";
  k <= "000011";
  i_data <= "00000000";
elsif (D = "00011") then
  Q <= "00011";
  -- start
  i_outs <= "10000010";
  -- turn on AC
  sel
  k <= "00001";
  i_data <= "00000000";
else
  Q <= "00000";
  -- wait for do load
  i_outs <= "10000000";
  k <= "000011";
  i_data <= "00000000";
elsif (D = "00110") then
  Q <= "00110";
  -- turn preset off
  i_outs <= "10000010";
  -- turn preset off
  i_outs <= "10000010";
else
  Q <= "00011";
  -- preset to
  i_outs <= "00000110";
  -- preset
  i_outs <= "00000110";
else
  Q <= "00010";
  -- preset to
  i_outs <= "00000000";
  -- preset to
  i_outs <= "00000000";
else
  Q <= "000001";
  -- preset to
  i_outs <= "00000011";
  -- preset
  i_outs <= "00000011";
else
  Q <= "00000001";
  -- preset
  i_outs <= "00000000";
else
  Q <= "00000000";
  -- preset
  i_outs <= "00000000";
end if;
elsif (D = "00101") then
  Q <= "00101";
  -- decode
  i_outs <= "10000010";
  -- decode
  i_outs <= "10000010";
else
  Q <= "00011";
  -- decode
  i_outs <= "10000010";
  -- decode
  i_outs <= "10000010";
else
  Q <= "00010";
  -- decode
  i_outs <= "10000010";
  -- decode
  i_outs <= "10000010";
else
  Q <= "00001";
  -- decode
  i_outs <= "10000010";
  -- decode
  i_outs <= "10000010";
else
  Q <= "00000";
  -- decode
  i_outs <= "10000010";
  -- decode
  i_outs <= "10000010";
end if;
elsif (D = "00111") then
  Q <= "00111";
  -- decode
  i_outs <= "10000010";
  -- decode
  i_outs <= "10000010";
else
  Q <= "01111";
  -- decode
  i_outs <= "10000010";
  -- decode
  i_outs <= "10000010";
else
  Q <= "00101";
  -- decode
  i_outs <= "10000010";
  -- decode
  i_outs <= "10000010";
else
  Q <= "00010";
  -- decode
  i_outs <= "10000010";
  -- decode
  i_outs <= "10000010";
else
  Q <= "00000";
  -- decode
  i_outs <= "10000010";
  -- decode
  i_outs <= "10000010";
end if;
elsif (D = "01100") then
  Q <= "01100";
  -- decode
  i_outs <= "10000010";
  -- decode
  i_outs <= "10000010";
else
  Q <= "01100";
  -- decode
  i_outs <= "10000010";
  -- decode
  i_outs <= "10000010";
else
  Q <= "00001";
  -- decode
  i_outs <= "10000010";
  -- decode
  i_outs <= "10000010";
end if;
elsif (D = "01101") then
  Q <= "01101";
  -- decode
  i_outs <= "10000010";
  -- decode
  i_outs <= "10000010";
else
  Q <= "01101";
  -- decode
  i_outs <= "10000010";
  -- decode
  i_outs <= "10000010";
else
  Q <= "00000";
  -- decode
  i_outs <= "10000010";
  -- decode
  i_outs <= "10000010";
end if;
k <= k + R;
Q <= "01010";
l Outs <= "10000010";
else
  Q <= "01010";
  l Outs <= "10000010";
elsif (D = "01010") then
  if (receive Ack = '0') then -- receive
    Q <= "01011";
    l Outs <= "10100010";
    end if;
elsif (D = "01011") then
  if (receive Ack = '1') then
    Q <= "00000";
    l Outs <= "10000000";
  else
    Q <= "00000";
    l Outs <= "10000000";
  end if;
end proc ess do loadAC_process;
D <= Q;
loadAC_ack <= l Outs(0);
huff_outSEL(0) <= l Outs(1);
loadLDI <= l Outs(2);
loadWEI <= l Outs(3);
loadDIN2 <= i data;
loadWADR2 <= k;
decode <= l Outs(4);
receive <= l Outs(5);
extend <= l Outs(6);
recSSSS <= SSSS;
mempreset <= l Outs(7);
end do loadAC_arch;

COMPONENT: decode
DESCRIPTION: (entity)

-- Author: Douglas A. Carpenter
-- Created: 02-JUN-1994
-- Revised:

-- library and use clauses
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164_extensions.all;
library my_packages;
use my_packages.package_1.all;

entity decode is
  port (...
    -- inputs
    clock : in std_ulogic;
    reset : in std_ulogic;
    decode : in std_ulogic;
    decode_ack : out std_ulogic;
    value : out
    std_ulogic_vector(7 downto 0);
    nextbit_req : out std_ulogic;
    nextbit_ack : in std_ulogic;
    nextbit_data : in std_ulogic;
    nextbit_err : in std_ulogic;
    huff_RADR : out
    std_ulogic_vector(3 downto 0);
    val_RADR : out
    std_ulogic_vector(7 downto 0);
    huffvalDOUT : in
    std_ulogic_vector(7 downto 0);
    huffvalptr_DOUT : in
    std_ulogic_vector(7 downto 0);
    huffmin_DOUT : in
    std_ulogic_vector(15 downto 0);
COMPONENT: decode

DESCRIPTION: (architecture)  

-- Purpose : decode module
-- Author : Douglas A. Carpenter
-- Created : 02-JUN-1994
-- Revised :  

architecture decode_arch of decode is
  signal D : std_ulogic_vector(3 downto 0);
  signal Q : std_ulogic_vector(3 downto 0);
  signal i_outs : std_ulogic_vector(1 downto 0);
  signal i : std_ulogic_vector(3 downto 0);
  signal j : std_ulogic_vector(7 downto 0);
  signal CODE : std_ulogic_vector(15 downto 0);
  signal i_data : std_ulogic_vector(7 downto 0);
  signal huffMIN : std_ulogic_vector(15 downto 0);
  signal DIFF : std_ulogic_vector(15 downto 0);

begin

  decode_process : process(reset, clock)
  begin
    if (reset = '0') then
      Q <= "0000";
      i_outs <= "00";
      i <= "0001";
      CODE <= "0000000000000000";
    elsif ((clock'event) and (clock = '1')) and
           (clock'last_value = '0')) then
      if (D = "0000") then
        if (decode = '1') then
          Q <= "0001"; -- decode was signaled
          i_outs <= "00";
          i <= "0001";
          CODE <= "0000000000000000";
        else
          Q <= "0000"; -- wait for decode
          i_outs <= "00";
          i <= "0001";
          CODE <= "0000000000000000";
        end if;
      elsif (D = "0001") then
        if ((nextbit_ack = '0') and (nextbit_err = '0')) then
          Q <= "0011"; -- get next bit
          i_outs <= "10";
          i <= "0001";
          CODE <= "00101";
          i_outs <= "01"; -- error, need to signal here some how
        else
          Q <= "0001"; -- wait until nextbit is ready
          i_outs <= "00";
        end if;
      elsif (D = "0011") then
        if ((nextbit_ack = '1') and (nextbit_err = '0')) then
          Q <= "0100"; -- got next bit
          i_outs <= "10";
          i <= "0001";
          CODE <= "00100";
          i_outs <= "01"; -- error, need to signal here some how
        else
          Q <= "0001"; -- wait until nextbit is ready
          i_outs <= "00";
        end if;
      elsif (D = "0101") then
        if ((nextbit_ack = '0') and (nextbit_err = '1')) then
          Q <= "1000"; -- err at next bit
          i_outs <= "00";
          i <= "0001";
          CODE <= "01010";
          i_outs <= "10"; -- wait
        else
          Q <= "0101"; -- wait
          i_outs <= "10";
        end if;
      elsif (D = "0111") then
        if ((nextbit_ack = '1') and (nextbit_err = '0')) then
          Q <= "1010"; -- got next bit
          i_outs <= "00";
          i <= "0001";
          CODE <= "01101";
          i_outs <= "10"; -- wait
        else
          Q <= "0111"; -- wait
          i_outs <= "00";
        end if;
      elsif (D = "1111") then
        if ((nextbit_ack = '1') and (nextbit_err = '1')) then
          Q <= "1100"; -- err at next bit
          i_outs <= "00";
          i <= "0001";
          CODE <= "11110";
          i_outs <= "10"; -- wait
        else
          Q <= "1111"; -- wait
          i_outs <= "00";
        end if;
      elsif (D = "1011") then
        if ((nextbit_ack = '0') and (nextbit_err = '1')) then
          Q <= "1001"; -- err at next bit
          i_outs <= "10";
          i <= "0001";
          CODE <= "10110";
          i_outs <= "01"; -- error, need to signal here some how
        else
          Q <= "1011"; -- wait
          i_outs <= "10";
        end if;
      end if;
    end if;
  end process decode_process;
end decode_arch;
elsif (D = "1101") then
  huffMIN <= huffmin_DOUT(15 downto 0);
  Q <= "1110";
  i_outs <= "00";
elsif (D = "1110") then
  j <= huffvalpr_DOUT;
  Q <= "1110";
  i_outs <= "00";
elsif (D = "1100") then
  DIFF <= (CODE - huffMIN);
  Q <= "1110";
  i_outs <= "00";
elsif (D = "1111") then
  if (decode = '0') then
    Q <= "0000";
    i_outs <= "00";
  else
    Q <= "1111";
    i_outs <= "01";
  end if;
elsif (D = "1010") then
  i_data <= huffval_DOUT;
  Q <= "1010";
  i_outs <= "00";
elsif (D = "1011") then
  if (decode = '0') then
    Q <= "0000";
    i_outs <= "00";
  else
    Q <= "1011";
    i_outs <= "01";
  end if;
elsif (D = "1000") then
  if (decode = '0') then
    Q <= "0000";
    i_outs <= "00";
  else
    Q <= "1000";
    i_outs <= "01";
  end if;
end if;
end process decode_process;

D <= Q;
decode_ack <= i_outs(0);
nexbit_req <= i_outs(1);
huff_RADR <= i;
val_RADR <= j;
value <= i_data;
end decode_arch;

COMPONENT: receive
DESCRIPTION: (entity)

-- Author: Douglas A. Carpenter
-- Created: 02-JUN-1994
-- Revised:

-- library and use clauses
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164_extensions.all;
library my_packages;
use my_packages.package_l.all;

entity receive is
  port (clock : in
    std_ulogic);
end receive;

COMPONENT: receive
DESCRIPTION: (architecture)

-- Author: Douglas A. Carpenter
-- Created: 02-JUN-1994
-- Revised:

-- library and use clauses
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164_extensions.all;
library my_packages;
use my_packages.package_l.all;

architecture receive_arch of receive is
  signal D : std_logic_vector(2 downto 0);
  signal Q : std_logic_vector(2 downto 0);
  signal i_outs : std_logic_vector(1 downto 0);
  signal v : std_logic_vector(7 downto 0);
  signal i_v : std_logic_vector(7 downto 0);
begin
  receive_process : process (reset, clock)
  variable i : std_logic_vector(7 downto 0);
  begin
    if (reset = '0') then
      Q <= "000";
      i_outs <= "00";
      i := "00000000";
      v <= "00000000";
      elsif ((clock'event) and (clock = '1') and
        (clock'last value = '0')) then
      if (D = "000") then
        if (receive = '1') then
          Q <= "001";
          -- wait for do receive
          i_outs <= "00";
        else
          Q <= "000";
          i_outs <= "00";
          i := "00000000";
          v <= "00000000";
        end if;
      elsif (D = "001") then
        if (i = SSSS) then
          Q <= "111"; -- done
          i_outs <= "01";
          i_v <= v;
        else
          Q <= "011"; -- get more
          i_outs <= "00";
        end if;
      elsif (D = "011") then
        if (receive = '1') then
          Q <= "001";
          -- wait for do receive
          i_outs <= "00";
        else
          Q <= "000";
          i_outs <= "00";
          i := "00000000";
          v <= "00000000";
        end if;
      end if;
      -- end receive
    end if;
  end process;
if ((nextbit_ack = '0') and (nextbit_err = '0'))
    then
        Q <= "010";
        i_outs <= "10";
    elsif (nextbit_err = '1') then
        Q <= "111";
        i_outs <= "01";
    else
        Q <= "011";
end if;
ready
i_outs <= "00";
else if (D = "010") then
    if ((nextbit_ack = '1') and (nextbit_err = '0'))
        then
            Q <= "110";
            i_outs <= "00";
    elsif (nextbit_err = '1') then
            Q <= "111";
            i_outs <= "01";
    else
            Q <= "010";
end if;
ack
i_outs <= "10";
elsif (D = "110") then
    if (nextbit_ack = '0')
        then
            Q <= "100";
    elsif (D = "100") then
        -- go to check
        i_outs <= "00";
    elsif (D = "111") then
        if (receive = '0')
            then
                Q <= "000";
                i_outs <= "00";
        else
                Q <= "111";
                i_outs <= "01";
        end if;
    else
        Q <= "000";
        i_outs <= "00";
    end if;
end if;
end process receive_process;

D <= Q;
receive_ack <= i_outs(0);
nextbit_req <= i_outs(1);
Vout <= i_V;
end receive_arch;

COMPONENT: extend DESCRIPTION: (architecture)

-- library and use clauses
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164_extensions.all;

library my_packages;
use my_packages.package_1.all;

entity extend is
    port (-- inputs
        clock : in std_ulogic;
        reset : in std_ulogic;
        extend : in std_ulogic;
        extend_ack : out std_ulogic;
        V : in std_ulogic_vector(7 downto 0);
        T : in std_ulogic_vector(7 downto 0);
        RetV : out std_ulogic_vector(7 downto 0));
    end;

COMPONENT: extend DESCRIPTION: (architecture)

-- Author : Douglas A. Carpenter
-- Created : 02-JUN-1994
-- Revised :

-- library and use clauses
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164_extensions.all;

library my_packages;
use my_packages.package_1.all;

architecture extend_arch of extend is
    signal D : std_ulogic_vector(2 downto 0);
    signal Q : std_ulogic_vector(2 downto 0);
    signal i_V : std_ulogic_vector(0 downto 0);
    signal V0 : std_logic_vector_CHAN(7 downto 0);
    signal V1 : integer range -1024 to 1024;
    signal V2 : std_logic_vector_CHAN(7 downto 0);
begin

    extend_process : process(reset, clock)
    variable two : std_logic_vector(7 downto 0);
    begin
        if (reset = '0') then
            Q <= "000";
            i_outs <= "00";
            two := "00000010";
        elsif (((clock'event) and (clock = '1') and
            (clock'last_value = '0')) then
            if (D = "000") then
                if (extend = '0') then
                    Q <= "000";
                    -- wait for do extend
                    i_outs <= "00";
                    two := "00000010";
                else
                    Q <= "001";
                    i_outs <= "0";
                    end if;
            elsif (D = "001") then
                if (T = "00000001") then
                    Vt <= "00000010";
                elsif (T = "00000100") then
                    Vt <= "00000010";
                end if;
        end if;
    end if;
end process receive_process;

D <= Q;
receive_ack <= i_outs(0);
nexbit_req <= i_outs(1);
Vout <= i_V;
end receive_arch;
elsif (T = "0000111") then
    Vt <= "00000100";
elsif (T = "0000100") then
    Vt <= "000001000";
elsif (T = "0000101") then
    Vt <= "000100000";
elsif (T = "0000110") then
    Vt <= "001000000";
elsif (T = "0000111") then
    Vt <= "010000000";
end if;
elsif (D = "011") then
    if (V < Vt) then
        if (T = "000000000") then
            Vt2 <= "1111111111";
        elsif (T = "000000001") then
            Vt2 <= "1111111111";
        else
            Vt2 <= "1111111111";
        end if;
        elsif (D = "010") then
            Vt1 <= To_Integer(Vt2) + 1;
            i_outs <= "0";
            Q <= "110";
        elsif (D = "110") then
            Q <= "100";
            i_outs <= "0";
            Vtemp <= To_Integer(V) + Vt1;
        elsif (D = "111") then
            Q <= "101";
            i_outs <= "1";
            i_V <= To_StdULogicVector(Vtemp, 8);
        elsif (D = "111") then
            if (extrem_D "0") then
                Q <= "000";
                i_outs <= "0";
            else
                Q <= "111";
                i_outs <= "1";
            end if;
        else
            Q <= "000";
            i_outs <= "0";
        end if;
    end if;
end process extend_process;
D <= Q;

extend_ack <= i_outs(0);

RetV <= i_V;
end extend_arch;

**DESCRIPTION:**

-- Purpose: nextbit module
-- Author: Douglas A. Carpenter
-- Created: 02-JUN-1994
-- Revised: 

architecture nextbit_arch of nextbit is
    signal D : std_logic_vector(3 downto 0);
    signal Q : std_logic_vector(3 downto 0);
    signal i_outs : std_logic_vector(2 downto 0);
    signal i_data : std_logic;
    signal next_counter : std_logic_vector(3 downto 0);
    signal next_byte : std_logic_vector(7 downto 0);
    signal next_byte2 : std_logic_vector(7 downto 0);
begin

nextbit_process : process(reset, clock)
begin
    if (reset = '0') then
        Q <= "0000";
        i_outs <= "0000";
    else
        next_counter <= "0000";
        elsif ([clock'event] and (clock = '1') and
                    (clock'last_value = '0')) then
            if (D = "0000") then
                if (nextbit_req = '1') then
                    Q <= "0001";
                    -- got nextbit request
            i_outs <= "000";
            else
                i_outs <= "0000";
            end if;
        wait
    end process nextbit_process;
end nextbit;
end if;
elsif (D = "0001") then
  if (next_counter = "0000") then
    Q <= "0011";  -- get another byte
  else
    Q <= "0100";  -- still have data in byte, go to send back next bit
    i_outs <= "000";
    end if;
  elsif (D = "0011") then
    if (next_ack = '0') and (err = '0') then
      Q <= "0010";  -- ready to get next byte, there is an error
      i_outs <= "100";
    else
      Q <= "0011";
      i_outs <= "000";  -- not ready for next byte req, wait.
    end if;
  elsif (D = "0010") then
    if ((next_ack = '1') and (err = '0')) then
      Q <= "0100";  -- ready to get next byte, there is an error
      i_outs <= "100";
    else
      Q <= "0010";  -- wait for next_ack
      i_outs <= "010";
    end if;
  elsif (D = "0110") then
    if (next_ack = '0') then
      Q <= "0100";  -- next_ack dropped, ok to latch data
      i_outs <= "000";
      next_byte <= data;
    else
      Q <= "0110";  -- wait for next_ack to drop
      i_outs <= "000";
    end if;
  elsif (D = "0100") then
    if (next_byte = "1111111") then
      Q <= "0101";  -- go to get another byte
      i_outs <= "000";
    else
      Q <= "1010";  -- nextbyte was not FF, go to return a bit
      i_outs <= "000";
    end if;
  elsif (D = "0101") then
    if (next_ack = '0') and (err = '0') then
      Q <= "0111";  -- ready to get next byte, there is an error
      i_outs <= "100";
    else
      Q <= "0101";
      i_outs <= "000";  -- not ready for next byte req, wait.
    end if;
  elsif (D = "0111") then
    if ((next_ack = '1') and (err = '0')) then
      Q <= "1111";  -- got next_ack, with no err, go to wait till ack drops
      i_outs <= "000";
    elsif (err = '1') then
      Q <= "1100";  -- err with nextbyte
      i_outs <= "100";
    else
      Q <= "0111";  -- wait for next_ack
      i_outs <= "010";
    end if;
  elsif (D = "1111") then
    if (next_ack = '0') then
      Q <= "1101";  -- next_ack dropped
      i_outs <= "000";
      next_byte2 <= data;
    else
      Q <= "1111";  -- wait for next_ack to drop
      i_outs <= "000";
    end if;
  elsif (D = "1101") then
    if (next_byte2 = "00000000") then
      Q <= "1010";  -- was equal to zero, ok
      i_outs <= "000";
    else
      Q <= "1100";
      i_outs <= "100";
    end if;
  elsif (D = "1010") then
    i_data <= next_byte(7);
    Q <= "1000";
    i_outs <= "000";
  elsif (D = "1000") then
    i_data <= next_byte(6 downto 0) & '0';
    next_counter <= next_counter - "0001";
    Q <= "1011";
    i_outs <= "001";
  elsif (D = "1011") then
    if (nextbit_req = '0') then
      Q <= "0000";
      i_outs <= "000";
    else
      Q <= "1011";
      i_outs <= "001";
    end if;
  elsif (D = "1100") then
    if (nextbit_req = '0') then
      Q <= "0000";
      i_outs <= "000";
    else
      Q <= "1100";
      i_outs <= "100";
    end if;
  elsif (D = "1110") then
    if (nextbit_req = '0') then
      Q <= "0000";
      i_outs <= "000";
    else
      Q <= "1110";
      i_outs <= "100";
    end if;
  end process nextbit_process;

D <= Q;
nextbit_ack <= i_outs(0);
next_request <= i_outs(1);
nextbit_err <= i_outs(2);
nextbit_data <= 1_data;
end nextbit_arch;

COMPONENT: huff_acdcsel16
DESCRIPTION: (entity)

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_1164_extensions.all;

-- Written by LL_to_VHDL at Tue May 3 10:07:38 1994
-- Parameterized Generator Specification to VHDL Code
--
-- LogicLib generator called: MULTIPLEXER
-- Passed Parameters are:
-- tinst name = huff_acdcsel
-- parameters are:
-- type = SIMPLE
-- W = 16
-- numin = 2
-- SW = 1
-- bus_mask = 0
-- comp_out = NO
--
-- huff_acdcsel Entity Description
entity huff_acdcsel6 is
port (IN0 : in std_logic_vector(15 downto 0);
INI : in std_logic_vector(15 downto 0);
SEL : in std_logic_vector(0 downto 0);
DOUT : out std_logic_vector(15 downto 0))
end huff_acdcsel6;

COMPONENT: huff_acdcsel6
DESCRIPTION: (architecture)

begin
huff_acdcsel6_Process: process(IN0,INI,SEL)
variable iaddress : integer range 0 to 1;
variable state : std_logic_vector(15 downto 0);
begin
iaddress := to_Integer('0' & SEL,0);
case iaddress is
when 0 =>
  state := IN0;
when 1 =>
  state := IN1;
when others =>
  state := IN1;
end case;

-- Assign outputs
DOUT <= state;
end process huff_acdcsel6_Process;
end rtl;

}
### Appendix B State Tables and Espresso Results

**Controller State Table**

<table>
<thead>
<tr>
<th>Q3</th>
<th>Q2</th>
<th>Q0</th>
<th>start_decode</th>
<th>req_ack</th>
<th>complete_ack</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
<th>work_req(2)</th>
<th>work_req(1)</th>
<th>work_req(0)</th>
<th>decoding</th>
<th>complete</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>0</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>0</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>0</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>0</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>0</td>
<td>X</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
# 2-bit by 2-bit binary adder
.i 8
.o 10
.type fr
.phase 1111111111

0000 0--- 0000 000000
0000 1-1- 0000 000000
0000 1-0- 0001 001100
0001 -0- 0001 001100
0001 -0- 0011 000100
0001 -1-- 1000 000011
0011 --1-- 0011 000100
0011 --0-- 0010 010100
0010 --0-- 0010 010100
0010 --0- 0110 000100
0010 --1-- 1000 000011
0110 --1- 0110 000100
0110 --0- 0100 011100
0100 --0- 0100 011100
0100 --0- 0101 000100
0100 --1-- 1000 000011
0101 --1- 0101 000100
0101 --0- 0111 100100
0111 --0- 0111 100100
0111 --0- 0111 000100
0111 --1-- 1000 000011
1111 --1-- 1111 000100
1111 --0- 1110 110100
1110 --0- 1110 110100
1110 --0- 1110 000100
1110 --1-- 1000 000011
1100 --1- 1110 000100
1100 --0- 1101 000100
1101 --0 1101 000010
1101 --1 1001 000000
1001 --1 1001 000000
1001 --0 0000 000000
1000 --0 1000 100000
1000 --1 1001 000000
1010 ---- 0000 000000
1011 ---- 0000 000000
.end
Controller Espresso Results

# jpeg decoder controller
.i 8
.o 10
#.phase 1111111111
.p 30
00001-0- 0001001100  #A0
0001-00- 0001001100  #A1
1101----0 0100000010  #A2
0010-1-- 1000000011  #A3
0010-1-- 1000000011  #A4
1110-1-- 1000000011  #A5
1110--00- 0010110000  #A6
100-----1 1001000000  #A7
0100-1-- 1000000011  #A8
0101--0- 0010100000  #A9
1100--1- 0100000010  #A10
0111-1-- 1000000011  #A11
001--00- 0010010100  #A12
1100--0- 0101000010  #A13
00-1-01- 0011000100  #A14
0011--0- 0010010100  #A15
0-10-01- 0110000100  #A16
0011--1- 0011000100  #A17
1-00---- 1000000000  #A18
1101---- 1001000000  #A19
01-0-00- 0100011100  #A20
010--01- 0101000100  #A21
0110--1- 0110000100  #A22
0110--0- 0100011100  #A23
111--0-- 1100000100  #A24
01-1-00- 0111000100  #A25
-111-01- 1111000100  #A26
1111--1- 1111000100  #A27
1111--0- 1110111010  #A28
0101---- 0101000100  #A29
.e
<table>
<thead>
<tr>
<th>Q2</th>
<th>Q1</th>
<th>Q0</th>
<th>next_req</th>
<th>incoming_data_ack</th>
<th>reg_err</th>
<th>Q2</th>
<th>D1</th>
<th>D0</th>
<th>next_ack</th>
<th>err</th>
<th>next_byte_req</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Wait for next request</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Got next request, send out to memory a request</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>wait for inc_ack</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>err on memory req</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>got inc_ack, no err</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>wait for inc_ack to return to 0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>0</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>inc_ack returned to 0, send next_ack</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>wait for next_req to return to 0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>next_req returned to 0, go to 000</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>err state, wait for next_req to return to 0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>err state, next_req returned to 0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Not Used</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Not Used</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Not Used</td>
</tr>
</tbody>
</table>

# 2-bit by 2-bit binary adder
.i 6 
.o 6 
.type fr 
.phase 111111 
000 0-- 000 000 
000 1-- 001 001 
001 -00 001 001 
001 001 111 010 
001 -10 011 000 
011 -1- 011 000 
011 -0- 010 100 
010 1-- 010 100 
010 0-- 000 000 
111 1-- 111 010 
111 0-- 000 000 
.end
NextByte Espresso Results

# NextByte Module
.i 6
.o 6
#phase 111111
.p 7
-01--1 111010  #A0
 0-1--1- 011000  #A1
-01-00 001001  #A2
 1--1-- 111010  #A3
-101-- 010100  #A4
-001-- 001001  #A5
 011-0- 010100  #A6
.e

FindSOI Espresso Input

.i 18
.o 16
.type fr
#phase 111111111111111
#phase 111111111111111
0000 --0- -------- --- 0000 0000 00000000
0000 -1- -------- --- 0000 0000 00000000
0000 1-- -------- --- 0000 0000 00000000
0000 001 -------- 10- 0000 0000 00000000
0000 001 -------- -1- 1001 0001 00000001
0000 001 -------- 00- 0001 0010 00000000
0001 --- -------- 00- 0001 0010 00000000
0001 --- -------- -1- 1001 0001 00000001
0001 --- -------- 10- 0011 0000 00000000
0011 --- 1111111 1-- 0011 0000 00000000
0011 --- 1111111 0-- 0010 0010 00000000
0011 --- -------- 0 1-- 0011 0000 00000000
0011 --- -------- 0 0-- 0001 0010 00000000
0011 --- -------- 0- 1-- 0011 0000 00000000
0011 --- -------- 0- 0-- 0001 0010 00000000
0011 --- -------- 0-- 1-- 0011 0000 00000000
0011 --- -------- 0-- 0-- 0001 0010 00000000
0011 --- -------- 0-- 0-- 0001 0010 00000000
0011 --- -------- 1-- 0011 0000 00000000
0011 --- -------- 1-- 0-- 0001 0010 00000000
0011 --- -------- 1-- 0-- 0001 0010 00000000
0011 --- -------- 0-- 1-- 0011 0000 00000000
0011 --- -------- 0-- 0-- 0001 0010 00000000
0011 --- -------- 0-- 0-- 0001 0010 00000000
0011 --- -------- 0-- 0-- 0001 0010 00000000
0011 --- -------- 0-- 0-- 0001 0010 00000000
0011 --- 0-------- 1-- 0011 0000 00000000
0011 --- 0-------- 1-- 0-- 0001 0010 00000000
0011 --- 0-------- 0-- 1-- 0011 0000 00000000
0011 --- 0-------- 0-- 0-- 0001 0010 00000000
0011 --- 0-------- 0-- 0-- 0001 0010 00000000
0100 --- -------- 00- 0010 0010 00000000
0100 --- -------- -1- 1001 0001 00000001
0100 --- -------- 10- 0110 0000 00000000
0110 --- 11011000 1-- 0110 0000 00000000
0110 --- 11011000 0-- 1110 0100 00000000
<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0110---</td>
<td>---1</td>
</tr>
<tr>
<td>0110---</td>
<td>---1</td>
</tr>
<tr>
<td>0110---</td>
<td>---1-</td>
</tr>
<tr>
<td>0110---</td>
<td>---1-</td>
</tr>
<tr>
<td>0110---</td>
<td>---1-</td>
</tr>
<tr>
<td>0110---</td>
<td>---1-</td>
</tr>
<tr>
<td>0110---</td>
<td>---0---</td>
</tr>
<tr>
<td>0110---</td>
<td>---0---</td>
</tr>
<tr>
<td>0110---</td>
<td>---0---</td>
</tr>
<tr>
<td>0110---</td>
<td>---0---</td>
</tr>
<tr>
<td>0110---</td>
<td>---1---</td>
</tr>
<tr>
<td>0110---</td>
<td>---1---</td>
</tr>
<tr>
<td>0110---</td>
<td>---1---</td>
</tr>
<tr>
<td>0110---</td>
<td>---0---</td>
</tr>
<tr>
<td>0110---</td>
<td>---0---</td>
</tr>
<tr>
<td>0110---</td>
<td>---0---</td>
</tr>
<tr>
<td>0110---</td>
<td>---0---</td>
</tr>
<tr>
<td>0110---</td>
<td>---0---</td>
</tr>
<tr>
<td>0110---</td>
<td>---0---</td>
</tr>
<tr>
<td>0110---</td>
<td>---0---</td>
</tr>
<tr>
<td>0110---</td>
<td>---0---</td>
</tr>
<tr>
<td>0110---</td>
<td>---0---</td>
</tr>
<tr>
<td>1110 000</td>
<td>---</td>
</tr>
<tr>
<td>1110 --1</td>
<td>---</td>
</tr>
<tr>
<td>1110 --1</td>
<td>---</td>
</tr>
<tr>
<td>1110 1--</td>
<td>---</td>
</tr>
<tr>
<td>1110 1--</td>
<td>---</td>
</tr>
<tr>
<td>1111 000</td>
<td>---</td>
</tr>
<tr>
<td>1111 --1</td>
<td>---</td>
</tr>
<tr>
<td>1111 --1</td>
<td>---</td>
</tr>
<tr>
<td>1111 1--</td>
<td>---</td>
</tr>
<tr>
<td>1111 1--</td>
<td>---</td>
</tr>
<tr>
<td>0100---</td>
<td>---</td>
</tr>
<tr>
<td>0101---</td>
<td>---</td>
</tr>
<tr>
<td>0111---</td>
<td>---</td>
</tr>
<tr>
<td>1001---</td>
<td>---</td>
</tr>
<tr>
<td>1010---</td>
<td>---</td>
</tr>
<tr>
<td>1100---</td>
<td>---</td>
</tr>
<tr>
<td>1101---</td>
<td>---</td>
</tr>
</tbody>
</table>

===FindSOI Espresso Results===

```
.i 18
.o 16
*.phase 111111111111111
.p 36
0110------1--0-- 0001001000000000
0011------111111110-- 0010001000000000
0011----0-------0-- 0001001000000000
000-001-------1-- 10010001000000001
000-001-------00-- 00010010000000000
0011----0-------0-- 00010010000000000
0011------0-------0-- 00010010000000000
0011------0-------0-- 00010010000000000
0011------0-------0-- 00010010000000000
000------0--0-- 00010010000000000
0010------0--0-- 00010010000000000
```
0011--------0--0-- 0001001000000000
0001--------00- 0001001000000000
0011--------0-0-- 0001001000000000
0011--------00-- 0001001000000000
0110--110110--00-- 1110010000000000
0110--------1-- 0110000000000000
-011--------1-- 0011000000000000
00-1--------10- 0011000000000000
1001--------00--0 1001000100000001
10-1--------1-- 1011000000000000
0110--------10-- 0001001000000000
0110--------00--0-- 0001001000000000
0110--------00--0-- 0001001000000000
0110----1----0-- 0001001000000000
0110----0-----0-- 0001001000000000
0110----0-----0-- 0001001000000000
0-10--------10- 0110000000000000
0001--------1- 1001000100000001
1110----1------ 1110010000000000
11101-------- 1110010000000000
1110----1------ 1110010000000000
1111----1------ 1111000000000000
11111-------- 1111000000000000
11111-------- 1111000000000000
1011--------0 1111000000000000
0010--------1- 1001000100000001

FindSOF Espresso Input

.i 21
.o 11
.type fr
.phase 11111111111

0000 --1 -------- ----- 0000 0000 0 00
0000 -0- -------- ----- 0000 0000 0 00
0000 1- -------- ----- 0000 0000 0 00
0000 010 -------- 10---- 1000 0000 0 00
0000 010 -------- -1---- 1001 0001 1 00
0000 010 -------- 00---- 0001 0010 0 00
0001 --- -------- 00---- 0001 0010 0 00
0001 --- -------- -1---- 1001 0001 1 00
0001 --- -------- 10---- 0011 0000 0 00
0011 --- 11111111 1----- 0011 0000 0 00
0011 --- 11111111 0----- 0010 0010 0 00
0011 --- --------0 1----- 0011 0000 0 00
0011 --- --------0 0----- 0001 0010 0 00
0011 --- --------0- 1----- 0011 0000 0 00
0011 --- --------0- 0----- 0001 0010 0 00
0011 --- --------0- 1----- 0011 0000 0 00
0011 --- --------0- 0----- 0001 0010 0 00
0011 --- --------0- 1----- 0011 0000 0 00
0011 --- --------0- 0----- 0001 0010 0 00
0011 --- --------0- 1----- 0011 0000 0 00
0011 --- --------0- 0----- 0001 0010 0 00
0011 --- --------0- 1----- 0011 0000 0 00
0011 --- --------0- 0----- 0001 0010 0 00
FindSOF Espresso Results

.i 21
.o 11
#.phase 1111111111
.p 61
0011---11111110----- 00100010000
0110---110110110----- 00110000010
010---110001000----- 11000000001
1010---110000000----- 10110100000
-000010---------1----- 10000000000
000-010---------00----- 00010010000
000-010---------1----- 10010001100
1100---------------0110000000001
0-10---------10----- 01100000000
0010---------------00001001000
11-0---------0------11010000000
1010---------1--0------ 00010001000
1010---------1------0----- 00010001000
1010---------0-------0----- 00010001000
1010---------0-------0----- 00010001000
1010---------10------ 00010001000
1010-------1-0----- 0001001000
1010-------1--0---- 0001001000
1010-------1---0---- 0001001000
0110--------1----- 0010000000
1000--------0----- 0101000000
1001--------0----- 0001000110
0011--------00----- 0001001000
0011--------0-0---- 0001001000
0011--------0--0--- 0001001000
0011-------0---0--- 0001001000
0011-------0--0--- 0001001000
0011-------0--0--- 0001001000
010--------1----- 0100000000
00-1-------10----- 0011000000
1110--------0----- 0001000110
010--------1----- 1010000000
010------0------ 1010000000
010------0------- 1010000000
1111--------0----- 0001000000
010--------10----- 1010000000
010--------1-0----- 1010000000
010--------0-0----- 1010000000
010--------1-0----- 1010000000
010--------1-0----- 1010000000
1010--------1----- 1010000000
01-1-------1-0---- 0101100000
1110--------1-1---- 1110000000
1100--------1-1---- 1001000110
0-01--------000---- 0001001000
0110--------1----- 0100000000
0011--------1----- 0011000000
01-1-------1-0---- 1110000000
0010-------1-1----- 1001100110
1011-------1-1----- 1011010000
1011------1-1----- 1011010000
1011------1-1----- 1011010000
1111--------1-1---- 1110000000
-101-------1-1----- 1011000000
-101-------1-1----- 1011000000
-101-------1-1----- 1011000000
0-01-------1-1----- 1001000110
100--------0-0----- 1000000000
01-1-------1-1----- 1001000110
.e

Fr_Header Espresso Input

.i 20
.o 15
.type fr
.phase 1111111111111

# 0
000000  --0  --  -------  -  000000  000000000
000000  -0-  --  -------  -  000000  000000000
000000  1--  --  -------  -  000000  000000000
000000  011  10  -------  -  000000  000000000
<table>
<thead>
<tr>
<th>#</th>
<th>1</th>
<th>000000</th>
<th>011</th>
<th>-1</th>
<th>000000</th>
<th>100000</th>
<th>00110000</th>
<th>000001</th>
<th>10000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>000001</td>
<td>---</td>
<td>00</td>
<td>---</td>
<td>000001</td>
<td>100000</td>
<td>00110000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#2</td>
<td>000001</td>
<td>---</td>
<td>-1</td>
<td>---</td>
<td>100000</td>
<td>00110000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#3</td>
<td>000001</td>
<td>---</td>
<td>10</td>
<td>---</td>
<td>000011</td>
<td>000000</td>
<td>00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#4</td>
<td>000100</td>
<td>---</td>
<td>1-</td>
<td>---</td>
<td>000100</td>
<td>000000</td>
<td>00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#5</td>
<td>000100</td>
<td>---</td>
<td>0-</td>
<td>---</td>
<td>000100</td>
<td>000000</td>
<td>00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#6</td>
<td>000110</td>
<td>---</td>
<td>00</td>
<td>---</td>
<td>000110</td>
<td>100000</td>
<td>00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#7</td>
<td>000110</td>
<td>---</td>
<td>-1</td>
<td>---</td>
<td>100000</td>
<td>00110000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#8</td>
<td>000110</td>
<td>---</td>
<td>10</td>
<td>---</td>
<td>000100</td>
<td>000000</td>
<td>00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#9</td>
<td>000110</td>
<td>---</td>
<td>1-</td>
<td>---</td>
<td>000110</td>
<td>000000</td>
<td>00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#10</td>
<td>000110</td>
<td>---</td>
<td>0-</td>
<td>---</td>
<td>000110</td>
<td>000000</td>
<td>00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#11</td>
<td>000110</td>
<td>---</td>
<td>1-</td>
<td>---</td>
<td>000110</td>
<td>000000</td>
<td>00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#12</td>
<td>000110</td>
<td>---</td>
<td>0-</td>
<td>---</td>
<td>000110</td>
<td>000000</td>
<td>00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#</td>
<td>Instruction</td>
<td>Z</td>
<td>X</td>
<td>Y</td>
<td>A</td>
<td>R</td>
<td>S</td>
<td>W</td>
<td>V</td>
</tr>
<tr>
<td>----</td>
<td>-------------</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>12</td>
<td>001100</td>
<td>00</td>
<td>-1</td>
<td>0</td>
<td>001100</td>
<td>10000000</td>
<td>001100000</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>001100</td>
<td>00</td>
<td>1</td>
<td>0</td>
<td>001100</td>
<td>10000000</td>
<td>001100000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>001110</td>
<td>01</td>
<td>-1</td>
<td>0</td>
<td>001110</td>
<td>00000000</td>
<td>000000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>001110</td>
<td>00</td>
<td>0</td>
<td>0</td>
<td>001010</td>
<td>00000000</td>
<td>000000001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>001010</td>
<td>0--</td>
<td>0</td>
<td>0</td>
<td>001011</td>
<td>00000000</td>
<td>000000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>001011</td>
<td>0--</td>
<td>0</td>
<td>0</td>
<td>001001</td>
<td>10000000</td>
<td>000000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>001001</td>
<td>00</td>
<td>-1</td>
<td>0</td>
<td>001001</td>
<td>10000000</td>
<td>000000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>001001</td>
<td>00</td>
<td>1</td>
<td>0</td>
<td>001000</td>
<td>10000000</td>
<td>000000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>001000</td>
<td>01</td>
<td>-1</td>
<td>0</td>
<td>001000</td>
<td>00000000</td>
<td>000000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>001000</td>
<td>00</td>
<td>0</td>
<td>0</td>
<td>011000</td>
<td>00000000</td>
<td>000000010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>011000</td>
<td>0--</td>
<td>0</td>
<td>0</td>
<td>010000</td>
<td>00000000</td>
<td>000000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>010000</td>
<td>0--</td>
<td>0</td>
<td>0</td>
<td>010001</td>
<td>00000000</td>
<td>000000100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>010001</td>
<td>0--</td>
<td>0</td>
<td>0</td>
<td>010011</td>
<td>00001000</td>
<td>000000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>010011</td>
<td>0--</td>
<td>0</td>
<td>0</td>
<td>010010</td>
<td>10000000</td>
<td>000000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>010010</td>
<td>00</td>
<td>-1</td>
<td>0</td>
<td>010010</td>
<td>10000000</td>
<td>000000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>010010</td>
<td>00</td>
<td>1</td>
<td>0</td>
<td>100000</td>
<td>00110000</td>
<td>000000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>010010</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>010110</td>
<td>00000000</td>
<td>000000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>010110</td>
<td>01</td>
<td>-1</td>
<td>0</td>
<td>010110</td>
<td>10000000</td>
<td>000000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>010110</td>
<td>00</td>
<td>0</td>
<td>0</td>
<td>010100</td>
<td>00000000</td>
<td>000000001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>010100</td>
<td>0--</td>
<td>0</td>
<td>0</td>
<td>010101</td>
<td>00000000</td>
<td>000000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>010101</td>
<td>0--</td>
<td>0</td>
<td>0</td>
<td>010111</td>
<td>10000000</td>
<td>000000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>010111</td>
<td>00</td>
<td>-1</td>
<td>0</td>
<td>010111</td>
<td>10000000</td>
<td>000000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>010111</td>
<td>00</td>
<td>1</td>
<td>0</td>
<td>100000</td>
<td>00110000</td>
<td>000000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>010111</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>011111</td>
<td>00000000</td>
<td>000000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
011111 --- 1- 10000000 011111 00000000
011111 --- 0- 10000000 011101 00000010

#29
011101 --- -- 10000000 011100 00000000

#28
011100 --- -- 00000000 011110 00000100

#30
011110 --- -- 00000000 011010 00000100

#26
011010 --- -- 00000000 011011 10000000

#27
011011 --- 00 00000000 011011 10000000
011011 --- -1 00000000 100000 00110000
011011 --- 10 00000000 011001 00000000

#25
011001 --- 1- 00000000 011001 00000000
011001 --- 0- 00000000 111001 00000000

#57
111001 --- -- 00000000 111001 10000000
111001 --- -- 00000000 100000 00110000
111001 --- -- 10000000 100000 00110000
111001 --- -- -10000000 100000 00110000
111001 --- -- -10000000 100000 00110000
111001 --- -- -10000000 100000 00110000
111001 --- -- -10000000 100000 00110000
111001 --- -- -10000000 100000 00110000

#59
111011 --- 00 00000000 111011 10000000
111011 --- -1 00000000 100000 00110000
111011 --- 10 00000000 111010 00000000

#58
111010 --- 1- 00000000 111010 00000000
111010 --- 0- 00000000 111110 00000000

#62
111110 --- -- 00000000 111110 10000000
111110 --- -- 00000000 100000 00110000
111110 --- -- 10000000 100000 00110000
111110 --- -- -10000000 100000 00110000
111110 --- -- -10000000 100000 00110000
111110 --- -- -10000000 100000 00110000
111110 --- -- -10000000 100000 00110000

#60
111100 --- 00 00000000 111100 10000000
111100 --- -1 00000000 100000 00110000
111100 --- 10 00000000 111101 00000000
| #61 | 111101 | -- | 1- | -------- | - | 111101 | 000000000 |
|     | 111101 | -- | 0- | -------- | - | 111111 | 000000000 |
| #63 | 111111 | -- | -- | 0010001 | - | 101111 | 100000000 |
|     | 111111 | -- | -- | --------0 | - | 100000 | 001100000 |
|     | 111111 | -- | -- | --------1- | - | 100000 | 001100000 |
|     | 111111 | -- | -- | ----1-- | - | 100000 | 001100000 |
|     | 111111 | -- | -- | ----1-- | - | 100000 | 001100000 |
|     | 111111 | -- | -- | --0-- | - | 100000 | 001100000 |
|     | 111111 | -- | -- | --1-- | - | 100000 | 001100000 |
|     | 111111 | -- | -- | 1------ | - | 100000 | 001100000 |
|     | 111111 | -- | -- | 1------ | - | 100000 | 001100000 |
| #47 | 101111 | -- | 00 | -------- | - | 101111 | 100000000 |
|     | 101111 | -- | -1 | -------- | - | 100000 | 001100000 |
|     | 101111 | -- | 10 | -------- | - | 101101 | 000000000 |
| #45 | 101101 | -- | 1- | -------- | - | 101101 | 000000000 |
|     | 101101 | -- | 0- | -------- | - | 101100 | 000000000 |
| #44 | 101100 | -- | -- | 00000000 | - | 101110 | 010000000 |
|     | 101100 | -- | -- | --------1 | - | 100000 | 001100000 |
|     | 101100 | -- | -- | --------1 | - | 100000 | 001100000 |
|     | 101100 | -- | -- | ----1-- | - | 100000 | 001100000 |
|     | 101100 | -- | -- | ----1-- | - | 100000 | 001100000 |
|     | 101100 | -- | -- | --1-- | - | 100000 | 001100000 |
|     | 101100 | -- | -- | --1-- | - | 100000 | 001100000 |
|     | 101100 | -- | -- | 1------ | - | 100000 | 001100000 |
|     | 101100 | -- | -- | 1------ | - | 100000 | 001100000 |
| #46 | 101110 | 000 | -- | -------- | - | 000000 | 000000000 |
|     | 101110 | -1 | -- | -------- | - | 101110 | 010000000 |
|     | 101110 | -1 | -- | -------- | - | 101110 | 010000000 |
|     | 101110 | 1-- | -- | -------- | - | 101110 | 010000000 |
| #32 | 100000 | -- | -- | -------- | 0 | 100000 | 001100000 |
|     | 100000 | -- | -- | -------- | 1 | 100001 | 000000000 |
| #33 | 100001 | -- | -- | -------- | 1 | 100001 | 000000000 |
|     | 100001 | -- | -- | -------- | 0 | 100011 | 000010000 |
| #35 | 100011 | 000 | -- | -------- | - | 000000 | 000000000 |
|     | 100011 | -1 | -- | -------- | - | 100011 | 000010000 |
|     | 100011 | -1 | -- | -------- | - | 100011 | 000010000 |
|     | 100011 | 1-- | -- | -------- | - | 100011 | 000010000 |

.end

Fr_Header Espresso Results
1-01------- 1101000
1-1--1----- 1110100
1-1-1------ 1110100
1-11------- 1110100
01--------1 1110100
.e

FindEOI Espresso Input

.i 18
.o 16
.type fr
.phase 1111111111111111

0000 --1 -------- --- 0000 0000 00000000
0000 -0- -------- --- 0000 0000 00000000
0000 0-- -------- --- 0000 0000 00000000
0000 110 -------- 10- 0000 0000 00000000
0000 110 -------- -1- 1001 0001 00000111
0000 110 -------- 00- 0001 0010 00000000

0001 --- -------- 00- 0001 0010 00000000
0001 --- -------- -1- 1001 0001 00000111
0001 --- -------- 10- 0011 0000 00000000

0011 --- 11111111 1-- 0011 0000 00000000
0011 --- 11111111 0-- 0010 0010 00000000
0011 --- -------0 1-- 0011 0000 00000000
0011 --- -------0 0-- 0001 0010 00000000
0011 --- -------0 1-- 0011 0000 00000000
0011 --- -------0 0-- 0001 0010 00000000
0011 --- -------0 1-- 0011 0000 00000000
0011 --- -------0 0-- 0001 0010 00000000
0011 --- -------0 1-- 0011 0000 00000000
0011 --- -------0 0-- 0001 0010 00000000
0011 --- -------0 1-- 0011 0000 00000000
0011 --- -------0 0-- 0001 0010 00000000
0011 --- -------0 1-- 0011 0000 00000000
0011 --- -------0 0-- 0001 0010 00000000
0011 --- -------0 1-- 0011 0000 00000000
0011 --- -------0 0-- 0001 0010 00000000
0011 --- -------0 1-- 0011 0000 00000000
0011 --- -------0 0-- 0001 0010 00000000
0011 --- -------0 1-- 0011 0000 00000000
0011 --- -------0 0-- 0001 0010 00000000
0011 --- -------0 1-- 0011 0000 00000000
0011 --- -------0 0-- 0001 0010 00000000

0010 --- -------- 00- 0010 0010 00000000
0010 --- -------- -1- 1001 0001 00000111
0010 --- -------- 10- 0110 0000 00000000

0110 --- 11011001 1-- 0110 0000 00000000
0110 --- 11011001 0-- 1110 0100 00000000
0110 --- -------0 1-- 0110 0000 00000000
0110 --- -------0 0-- 0001 0010 00000000
0110 --- -------1 1-- 0110 0000 00000000
0110 --- -------1 1-- 0110 0000 00000000
0110 --- -------1 0-- 0001 0010 00000000
0110 --- -------1 0-- 0001 0010 00000000
0110 --- -------0 1-- 0110 0000 00000000
0110 --- -------0 0-- 0001 0010 00000000
0110 --- -------0 1-- 0110 0000 00000000
0110 --- -------0 0-- 0001 0010 00000000
0110  ---  --1------  1-- 0110  0000  00000000
0110  ---  --1------  0-- 0001  0010  00000000
0110  ---  --0--------  1-- 0110  0000  00000000
0110  ---  --0--------  0-- 0001  0010  00000000
0110  ---  --0--------  1-- 0110  0000  00000000
0110  ---  --0--------  0-- 0001  0010  00000000

1110  000  --------  ---  0000  0000  00000000
1110  --1  --------  ---  1110  0100  00000000
1110  --1  --------  ---  1110  0100  00000000
1110  --1  --------  ---  1110  0100  00000000

1001  ---  --------  --0  1001  0001  00001111
1001  ---  --------  --1  1011  0000  00000000
1011  ---  --------  --1  1011  0000  00000000
1011  ---  --------  --0  1111  1000  00000000

1111  000  --------  ---  0000  0000  00000000
1111  --1  --------  ---  1111  1000  00000000
1111  --1  --------  ---  1111  1000  00000000
1111  --1  --------  ---  1111  1000  00000000

0100  ---  --------  ---  0000  0000  00000000
0101  ---  --------  ---  0000  0000  00000000
0111  ---  --------  ---  0000  0000  00000000
1000  ---  --------  ---  0000  0000  00000000
1010  ---  --------  ---  0000  0000  00000000
1100  ---  --------  ---  0000  0000  00000000
1101  ---  --------  ---  0000  0000  00000000

.end

FindEOI Espresso Results

.i 18
.o 16
#.phase 11111111111111
.p 36
0110-------1--0-- 00100100000000
0011----111111110-- 001000100000000
000-110------1- 1001000100000001
0011----0-------0-- 0010010000000000
000-110------00- 0001001000000000
0011----0-------0-- 00010010000000000
0011----0-------0-- 00010010000000000
0011----0-------0-- 00010010000000000
0011----0-------0-- 00010010000000000
0011----0-------0-- 00010010000000000
0110-------1--0-- 1110010000000000
0110-------1--0-- 0110000000000000
-011-------1-- 0011000000000000
00-1-------10- 0011000000000000
1001-------0 100100010000111
10-1-------1 1011000000000000
FindSOS Espresso Input

# find start of scan
.i 19
.o 11
.type fr
.phase 1111111111

0000  0  -------  -------  0000  0000000
0000  1  -------  10-----  0000  0000000
0000  1  -------  -1-----  1001  0001100
0000  1  -------  00-----  0001  0010000

0001  -  -------  00-----  0001  0010000
0001  -  -------  -1-----  1001  0001100
0001  -  -------  10-----  0011  0000000

0011  -  11111111  1-----  0011  0000000
0011  -  11111111  0-----  0010  0010000
0011  -  -------0  1-----  0011  0000000
0011  -  -------0  0-----  0001  0010000
0011  -  -------0  1-----  0011  0000000
0011  -  -------0  0-----  0001  0010000
0011  -  -------0  0-----  0001  0010000
0011  -  -------0  1-----  0011  0000000
0011  -  -------0  0-----  0001  0010000
0011  -  -------0  1-----  0011  0000000
0011  -  -------0  0-----  0001  0010000
0011  -  -------0  1-----  0011  0000000
0011  -  -------0  0-----  0001  0010000
0011  -  -------0  1-----  0011  0000000
0011  -  -------0  0-----  0001  0010000
0011  -  -------0  1-----  0011  0000000
0011  -  -------0  0-----  0001  0010000
0011  -  -------0  1-----  0011  0000000
0011  -  -------0  0-----  0001  0010000
0011  -  -------0  1-----  0011  0000000
0011  -  -------0  0-----  0001  0010000
0011  -  -------0  1-----  0011  0000000
0011  -  -------0  0-----  0001  0010000
0011  -  -------0  1-----  0011  0000000
0011  -  -------0  0-----  0001  0010000
0011  -  -------0  1-----  0011  0000000
0011  -  -------0  0-----  0001  0010000
0011  -  -------0  1-----  0011  0000000
0011  -  -------0  0-----  0001  0010000
0011  -  -------0  1-----  0011  0000000
0011  -  -------0  0-----  0001  0010000
0011  -  -------0  1-----  0011  0000000
0011  -  -------0  0-----  0001  0010000
0010  -  -------0  0-----  0010  0010000
0010  -  -------0  -1-----  1001  0001100
0010  -  -------0  10-----  0110  0000000

0110  -  11011011  1-----  0110  0000000
| 0110 | 11011011 | 0----- | 0111 | 0000010 |
| 0110 | ------0 | 1----- | 0110 | 0000000 |
| 0110 | ------0 | 0----- | 0100 | 0000000 |
| 0110 | ------0 | 1----- | 0110 | 0000000 |
| 0110 | ------0 | 0----- | 0100 | 0000000 |
| 0110 | ------1 | 1----- | 0110 | 0000000 |
| 0110 | ------1 | 0----- | 0100 | 0000000 |
| 0110 | ------0 | 1----- | 0110 | 0000000 |
| 0110 | ------0 | 0----- | 0100 | 0000000 |
| 0110 | ------1 | 1----- | 0110 | 0000000 |
| 0110 | ------1 | 0----- | 0100 | 0000000 |
| 0110 | ------0 | 1----- | 0110 | 0000000 |
| 0110 | ------0 | 0----- | 0100 | 0000000 |
| 0100 | 11000100 | 1----- | 0100 | 0000000 |
| 0100 | 11000100 | 0----- | 1100 | 0000001 |
| 0100 | ------1 | 1----- | 0100 | 0000000 |
| 0100 | ------1 | 0----- | 1010 | 0000000 |
| 0100 | ------1 | 1----- | 0100 | 0000000 |
| 0100 | ------1 | 0----- | 1010 | 0000000 |
| 0100 | ------0 | 1----- | 0100 | 0000000 |
| 0100 | ------0 | 0----- | 1010 | 0000000 |
| 0100 | ------1 | 1----- | 0100 | 0000000 |
| 0100 | ------1 | 0----- | 1010 | 0000000 |
| 0100 | ------0 | 1----- | 0100 | 0000000 |
| 0100 | ------0 | 0----- | 1010 | 0000000 |
| 0111 | ------0 | 0----- | 0111 | 0000010 |
| 0111 | ------1 | 0----- | 1111 | 0000000 |
| 0111 | ------1 | 1----- | 1001 | 0001100 |
| 1111 | ------1 | 0----- | 1111 | 0000000 |
| 1111 | ------0 | 0----- | 0001 | 0010000 |
| 1100 | ------0 | 0----- | 1100 | 0000001 |
| 1100 | ------1 | 1----- | 1110 | 0000000 |
| 1100 | ------1 | 0----- | 1001 | 0001100 |
| 1110 | ------1 | 1----- | 1110 | 0000000 |
| 1110 | ------0 | 0----- | 0001 | 0010000 |
| 1010 | 11011010 | 1----- | 1010 | 0000000 |
| 1010 | 11011010 | 0----- | 1011 | 1000000 |
| 1010 | ------1 | 1----- | 1010 | 0000000 |
| 1010 | ------1 | 0----- | 0001 | 0010000 |
| 1010 | ------0 | 1----- | 1010 | 0000000 |
| 1010 | ------0 | 0----- | 0001 | 0010000 |
| 1010 | ------1 | 1----- | 1010 | 0000000 |
| 1010 | ------1 | 0----- | 0001 | 0010000 |
| 1010 | ------0 | 1----- | 1010 | 0000000 |
1010  -  -----0---  0-----  0001  0010000
1010  -  ---0---  1-----  1010  0000000
1010  -  ---0---  0-----  0001  0010000
1010  -  --1-----  1-----  1010  0000000
1010  -  --1-----  0-----  0001  0010000
1010  -  -0------  1-----  1010  0000000
1010  -  -0------  0-----  0001  0010000
1010  -  0--------  1-----  1010  0000000
1010  -  0--------  0-----  0001  0010000
1011  0  --------  ------  0000  0000000
1011  1  --------  ------  1011  1000000
1001  -  --------  --0---  1001  0001100
1001  -  --------  --1---  1000  0000000
1000  -  --------  --1---  1000  0000000
1000  -  --------  --0---  1101  0100000
1101  0  --------  ------  0000  0000000
1101  1  --------  ------  1101  0100000

.end

FindSOS Espresso Results

# find start of scan
.i 19 .o 11 .#phase 1111111111 .p 56
0011--111111110----- 00100010000
0110-111001000----- 11000000001
0110-1110110110----- 00110000010
1010-1110100100----- 10111000000
1100-------------00 11000000001
000-10-------00--- 00100010000
11-0----------10 11100000000
0-10----------10--- 01100000000
1010--------0-0----- 00100010000
1010--------1-0----- 00101001000
1010--------0-0----- 00101001000
1010--------0-0----- 00100010000
1010--------1-0----- 00101001000
1010--0------0----- 00101001000
1010-0-------0----- 00101001000
0110----------1----- 00100000000
1010--------10----- 00100000000
0110---------00--- 00100010000
1001----------00--- 00100001100
1000----------0--- 01010100000
0011--------00--- 00100010000
0011--------0-0----- 00100010000
0011--------0-0----- 00100010000
0011--------0-0----- 00100010000
0011--------0-0----- 00100010000
0011--------0-0----- 00100010000
0011--------0-0----- 00100010000
0011--------0-0----- 00100010000
0011--------0-0----- 00100010000
0011--------0-0----- 00100010000
0011--------0-0----- 00100010000
0011--------0-0----- 00100010000
0011--------0-0----- 00100010000
0011--------0-0----- 00100010000
loadq Espresso Input

# load q table controller
.i 24
.o 12
.type fr
.phase 11111111

00000 0 ------ ---- 1---- 00000 000000
00000 1 ------ -1-0 ---- 00000 000000
00000 1 ------ -1 1-0 ---- 11111 100000
00000 1 ------ -0-0 ---- 00001 000010
00001 - ------ -0-0 ---- 00001 000010
00001 - ------ -1-- ---- 11111 100000
00001 - ------ -1-0 ---- 00011 000000
00011 - ------ -1-- ---- 00011 000000
00011 - ------ -0-- ---- 00010 000000
00010 - ------ ---- 00000000 00110 0000010
00010 - ------ ---- -1---- 11111 100000
00010 - ------ ---- ----- -1- 11111 100000
00010 - ------ ---- ----- -1-- 11111 100000
00010 - ------ ---- ----- -1---- 11111 100000
00010 - ------ ---- -1-- ---- 11111 100000
00010 - ------ ---- -1----- 11111 100000
00010 - ------ ---- -1------ 11111 100000
00010 - ------ ---- 1------ 11111 100000

B-23
.end

loadq Espresso Results

# load q table controller
.i 24
.o 12
#.phase 111111111111
.p 60
-1101------------00000000 000000100000
-0010------------00000000 001000000100
-0101------------01000000011 001000000010
01100-000000000000 010000000100
110--000000000000 100000100000
0000-1--0-0-------- 000010000100
01000---------0-0-------- 010000000100
-0001---------0-0-------- 000010000100
-011---------0-0-------- 000100000100
01100-----1--------- 011000010000
01100-----1--------- 011000100000
01100-----1--------- 011001000000
01100-----1--------- 011001000000
01100-----1--------- 011001100000
0-111-------1--------- 010100000100
-10-1-------1--------- 010010000000
-010---------0-------- 000010000000
0-111---------0-------- 000100000000
0100--------1--------- 010000000000
01110--------0--------- 111100100000
01-10--------1--------- 011100000000
-101--------0--------- 010000000011
110-------1--------- 010000000100
110-------1--------- 010000000100
110-------1--------- 010000000100
110-------1--------- 010000000100
110-------1--------- 010000000100
110-------1--------- 010000000100
10----1--------- 10000100000
-1001--------0--------- 010110000100
-1101---------110111000000
huff lenloader Espresso Input

# huffman data length and type loader
.i 33
.o 13
.type fr
.phase 111111111111

00000 0-- -------- 111111000000 - 00000 00000000
00000 110 -------- 000110000000 - 00000 00000000
00000 1-1 -------- 110111000000 - 11111 01000000
00000 100 -------- 110111000000 - 00001 00000001

00001 -00 -------- 111111000000 - 00001 00000001
00001 --1 -------- 110111000000 - 11111 01000000
00001 -10 -------- 110111000000 - 00011 00000000

00011 -1- -------- 111111000000 - 00011 00000000
00011 -0- -------- 111111000000 - 00010 00100000

00010 --- -------- 0000000000000000 - 00110 00001000
00110 --- -------- 0000000000000000 - 00100 00010000

00010 --- -------- 0000000000000000 - 11111 01000000
00100 --- -------- 0001000000000000 - 00101 00010000
00100 --- -------- 0001100000000000 - 00101 00010000
00100 --- -------- 0001100000000000 - 00101 00010000

huff_lenloader Espresso Output

# huffman data length and type loader
.i 33
.o 13
#.phase 111111111111
.p 92
01110 -1-        --------------                       - 01110 00000000
01110 -0-        0000----                       - 01101 00000100
01110 -0-        ___1--                       - 11111 01000000
01110 -0-        ___1--                       - 11111 01000000
01110 -0-        ___1--                       - 11111 01000000
01110 -0-        1--1--                       - 11111 01000000
01010 ---        --------------                       - 01011 00000010
01011 ---        --------------                       - 01001 00010010
01001 ---        --------------                       - 10001 00010010
01001 ---        00000000000000000 0 11111 01000000
01001 ---        --------------                       - 01000 00000000
01001 ---        --------------                       - 01000 00000000
01001 ---        --------------                       - 01000 00000000
01001 ---        --------------                       - 01000 00000000
01001 ---        --------------                       - 01000 00000000
01001 ---        --------------                       - 01000 00000000
01001 ---        --------------                       - 01000 00000000
01001 ---        --------------                       - 01000 00000000
01001 ---        --------------                       - 01000 00000000
01001 ---        --------------                       - 01000 00000000
01001 ---        --------------                       - 01000 00000000
01001 ---        --------------                       - 01000 00000000
01001 ---        --------------                       - 01000 00000000
01000 1--        --------------                       - 01000 10000000
01000 0--        --------------                       - 00000 00000000
11111 1--        --------------                       - 11111 01000000
11111 0--        --------------                       - 00000 00000000

.end
.e

# read in huffman data
.i 8
.o 9
.type fr
.phase 1111111

0000 0--- 0000 00100
0000 11-0 0000 00100
0000 1--1 1111 10100
0000 10-0 0001 00000

0001 ---- 0011 00000
0011 ---- 0010 00110

0010 --0- 0010 00110
0010 ---- 1111 10100
0010 --1- 0110 00100

0110 --1- 0110 00100
0110 --0- 0111 00101

0111 --00 0111 00101
0111 ---- 1111 10100
0111 --10 0101 00100

0101 --1- 0101 00100
0101 --0- 1101 01100

1101 1--- 1101 01100
1101 0--- 0000 00100

1111 1--- 1111 10100
1111 0--- 0000 00100

1000 ---- 0000 00100
1001 ---- 0000 00100
1010 ---- 0000 00100
1011 ---- 0000 00100
1100 ---- 0000 00100
1110 ---- 0000 00100
0100 ---- 0000 00100
.end

hread Espresso Output

# read in huffman data
.i 8
.o 9
#.phase 111111111
.p 18
0111--00 001000001
000-10-- 00010000
0101--0- 100010000
11011--- 110101000
001--0-0 001000110
0111---1 101010000
0-10-1-- 011000000
0110-0-- 011100001
0001---- 001100000
11111---- 111110000
---00--- 000000100
---0-1-- 000000100
0010----1 111110000
0011---- 001000110
00-01--1 111110100
01-1---- 010100000
1-------- 000000100
-1------- 000000100
.e

hread Espresso Input

# read in huffman data
.i 9
.o 9
.type fr
.phase 111111111

0000 0---- 0000 00000
0000 11--0 0000 00000
0000 1----1 1111 01000
0000 10--0 0001 00100

0001 -0--0 0001 00100
0001 ----1 1111 01000
0001 -1--0 0011 00000

0011 -1---- 0011 00000
0011 -0--- 0010 00010

0010 --0-0 0010 00010
0010 ----1 1111 01000
0010 --1-0 0110 00000
0110 --1-- 0110 0000
0110 --0-- 0100 0001
0100 ---00 0100 0001
0100 ----1 1111 0100
0100 ---10 0101 0000
0101 ---1- 0101 0000
0101 ---0- 0111 1000
0111 1---- 0111 1000
0111 0---- 0000 0000
1111 1---- 1111 0100
1111 0---- 0000 0000

.end

hreader Espresso Output

# read in huffman data
.i 8
.o 9
#.phase 111111111
.p 18
0111--00 00100001
000--10-- 00010000
0101--0- 10001000
11011---- 11010100
001--0-0 00100010
0111----1 10101000
0-10-1-- 01100000
0110-0-- 01110001
0001---- 00110000
11111--- 11110000
---00--- 000000100
---0-1-- 000000100
0010----1 11111000
0011---- 00100110
00-01--1 11110100
01-1---- 01010000
1------- 00000100
-1------- 00000100
.e

qaddr Espresso Input

# quant address incremeneter
.i 3
.o 4
.type fr
.phase 1111
00 0 0 0
00 1 01 00
01 - 11 01
11 - 10 10
10 1 10 10
10 0 00 00
.end
qaddr Espresso Output

# quant address incrementer
.i 3
.o 4
#.phase 1111
.p 4
0-1 0100
1-1 1010
01- 1101
11- 1010
.e

qmem Espresso Input

# load q table
.i 5
.o 7
.type fr
.phase 1111111
.phase 1111111
 000 00 000 0000
 000 11 000 0000
 000 01 001 0001
 000 10 100 0010
 001 -- 011 1000
 011 -1 011 0100
 011 -0 000 0000
 100 -- 110 1000
 110 1- 110 1000
 110 0- 000 0000
 101 -- 000 0000
111 -- 000 0000
010 -- 000 0000
.e

qmem Espresso Output

# load q table
.i 5
.o 7
#.phase 1111111
.p 6
00010 1000010
00001 0010001
011-1 0110100
1-01- 1101000
100-- 1101000
001-- 0111000
.e


Appendix C - Support Code

dec.c

/**
** dec.c
** behavioral model of jpeg baseline decoder

#include <stdio.h>
#include <unistd.h>
#include <unistd.h>
#include <math.h>
#include <sys/types.h>
#include <fcntl.h>
#include <sys/time.h>
#include <values.h>

/----------------- Local Definitions -----------------*/

#define ON 1
#define OFF 0
#define DEBUG 0
#define HDEBUG
#define DDEBUG
#define S0F1
#define JPG 0xC8
#define DHP
#define SOS

/*Start of frame, non-differential Huffman coding*/
#define SOFO 0xC0 /* Start of frame, baseline DCT */
#define S0F1 0xC1 /* Start of frame, extended sequential DCT */
#define SOF2 0xC2 /* Start of frame, progressive DCT */
#define SOF3 0xC3 /* Start of frame, spatial (sequential) lossless */
#define SOF5 0xC5 /* Start of frame, differential sequential DCT */
#define SOF6 0xC6 /* Start of frame, differential progressive DCT */
#define SOF7 0xC7 /* Start of frame, differential spatial */
/* Start of frame, non-differential arithmetic coding */
#define SOFO8 0xC8 /* Reserved for JPEG extensions */
#define SOFO9 0xC9 /* Extended sequential DCT */

#define DHT 0xC4 /* Define huffman table(s) */
#define DDC 0xCC /* Define arithmetic coding conditions */
#define DQT 0xDB /* Define quantization table(s) */
#define DRI 0xDD /* Define restart interval */
#define APP0 0xE0 /* application segment 0 */
#define APP1 0xE1 /* application segment 1 */
#define APP2 0xE2 /* application segment 2 */
#define APP3 0xE3 /* application segment 3 */
#define APP4 0xE4 /* application segment 4 */
#define APP5 0xE5 /* application segment 5 */
#define APP6 0xE6 /* application segment 6 */
#define APP7 0xE7 /* application segment 7 */
#define APP8 0xE8 /* application segment 8 */
#define APP9 0xE9 /* application segment 9 */
#define APPA 0xEA /* application segment A */
#define APPB 0xEB /* application segment B */
#define APPC 0xEC /* application segment C */
#define APPD 0xED /* application segment D */
#define APPE 0xEE /* application segment E */
#define APPF 0xEF /* application segment F */
#define COM 0xFE /* Comment */
#define TEM 0x01 /* Temporary */
#define RESL 0x02 /* reserved, lower end */
#define RSHU 0xBF /* reserved, upper end */
#define RSTL 0x0D /* Restart with modulo count "m" lower end */
#define RSTU 0x0E /* Restart with modulo count "m" upper end */
#define DNL 0x1C /* Define number lines */
#define SOS 0x0A /* Start of Scan */
#define EOI 0x0D /* End of Image */
#define DHP 0x0E /* Define hierarchical progression */
#define EXP 0x0F /* Expand reference component(s) */

C-1
#define DC 0
#define AC 1

/* -------------------------------------------------------- local external variables ----------------------------------------*/

short QTables[4][64];
int NumLinesImage;
int NumSamplesImage;
int NumImgCompFrame;
int NumImgCompScan;
int Compldent[256];
int HorzSampFactor[256];
int VertSampFactor[256];
int QuantTableSelector[256];
int ScanCompIdent[256];
int ScanDCTableSel[256];
int ScanACTableSel[256];

short precision;
int StartSpectral;
int EndSpectral;
short SuccAppxBitHigh;
short SuccAppxBitLow;
int RestartInterval;
int byte_counter;

int file;       /* file desc */
int outfile;

/* huffman coding stuff */
int BITS[17];
int HUFFVALS[256];

struct huffstruct
{
    int lastk;
    int BITS[17];
    int HUFFCODE[256];
    int HUFFSIZE[256];
    int HUFFVALS[256];
    int VALPTR[17];
    int MINCODE[17];
    int MAXCODE[17];
};

    /* first index */
    /* 0 is table 0, 1 is table 1 */
    /* second index */
    /* 0 is DC table, 1 is AC table */
struct huffstruct huffman_tables[2][2];

    /* huffman decoder */
short PRED[10];
short DIFF;
short T;
short SS[8];
short ZZ[64];
short ss[64][8];
short zigzag[64] = { 0, 1, 5, 6,14,15,27,28, 2, 4, 7,13,16,26,29,42, 3, 8,12,17,25,30,41,43, 9,11,18,24,31,45,44,53, 10,19,23,32,39,45,52,54, 20,22,33,38,46,51,55,60, 21,34,37,47,59,58,59,61, 22,38,37,47,59,58,59,61, 35,36,48,49,57,58,62,63};

struct scan_struct
{
    short line[1024][1024];
    };;

struct scan_struct scans[3];

    /*---------------------------------------------------------- Local Global Functions -----------------------------*/

int ReadAByte(short *d_byte);

main()
{
    char filename[80];
    void FindStartOfImage();
    void GetFileName(char *filename);
    void FindStartOfFrame();
    void DecodeFrame();
    RestartInterval 0;
    byte_counter 0;
}
GetFileName(filename);
printf("Reading %s JPEG file \n",filename);
file = open(filename,O_RDONLY);
printf("File is open\n");
printf("Reading the file, %s\n",filename);
FindStartOfFile();
FindStartOfFrame();
DecodeFrame();
close(file);
close(outfile);
printf("File closed\n");
printf("Done\n");
return (1);
}
/**
 * FindStartOfFile
 */

void
FindStartOfFile()
{
  int res;
  short data_byte;
  int marker;
  printf("Searching for start of image\n");
  res = 1;
  while (res != 0)
  {
    res = ReadAByte(&data_byte);
    if (data_byte == 0xFF)
    {
      marker = ON;
    }
    else if (marker == ON)
    {
      if (data_byte == 0x01)
      {
        printf("Start of image found FF%OX\n",data_byte);
        return;
      }
      else
      {
        marker = OFF;
      }
    }
    else
    {
      marker = OFF;
    }
    close(file);
    printf("ERROR!!! Start of Image NOT found\n");
    printf("Aborting\n");
    exit(1);
  }
}

/**
 * FindStartOfFrame
 */

void
FindStartOfFrame()
{
  int res;
  short data_byte;
  int marker;
  void InterpretComment();
  void InterpretAPP(int type);
  void DefineQuantTables();
  void DefineHuffmanTables();
  void DefineArithmeticTables();
  void DefineRestartInterval();
  printf("Searching for start of Frame\n");
  res = 1;
}
while (res != 0)
{
    res = ReadAByte(data_byte);
    if (data_byte == 0xFF)
    {
        marker = ON;
    }
    else if (marker == ON)
    {
        switch (data_byte)
        {
            case DHT:
                printf("Define Huffman tables \xFF\02X\n",data_byte);
                DefineHuffmanTables();
                break;
            case DAC:
                printf("Define Arithmetic Conditioning \xFF\02X\n",data_byte);
                DefineArithmeticTables();
                break;
            case DQT:
                printf("Define Quantization Tables \xFF\02X\n",data_byte);
                DefineQuantTables();
                break;
            case DRI:
                printf("Define Restart Interval \xFF\02X\n",data_byte);
                DefineRestartInterval();
                break;
            case APP0:
                printf("Application defined marker 0 : \xFF\0X\n",data_byte);
                InterpretAPP(0);
                break;
            case APP1:
                printf("Application defined marker 1 \xFF\0X\n",data_byte);
                InterpretAPP(1);
                break;
            case APP2:
                printf("Application defined marker 2 \xFF\0X\n",data_byte);
                InterpretAPP(2);
                break;
            case APP3:
                printf("Application defined marker 3 \xFF\0X\n",data_byte);
                InterpretAPP(3);
                break;
            case APP4:
                printf("Application defined marker 4 \xFF\0X\n",data_byte);
                InterpretAPP(4);
                break;
            case APP5:
                printf("Application defined marker 5 : \xFF\0X\n",data_byte);
                InterpretAPP(5);
                break;
            case APP6:
                printf("Application defined marker 6 : \xFF\0X\n",data_byte);
                InterpretAPP(6);
                break;
            case APP7:
                printf("Application defined marker 7 \xFF\0X\n",data_byte);
                InterpretAPP(7);
                break;
            case APP8:
                printf("Application defined marker 8 : \xFF\0X\n",data_byte);
                InterpretAPP(8);
                break;
            case APP9:
                printf("Application defined marker 9 \xFF\0X\n",data_byte);
                InterpretAPP(9);
                break;
            case APPA:
                printf("Application defined marker A : \xFF\0X\n",data_byte);
                InterpretAPP(10);
                break;
            case AP PB:
                printf("Application defined marker B \xFF\0X\n",data_byte);
                InterpretAPP(11);
                break;
            case APPC:
                printf("Application defined marker C \xFF\0X\n",data_byte);
                InterpretAPP(12);
                break;
            case APPD:
                printf("Application defined marker D \xFF\0X\n",data_byte);
                InterpretAPP(13);
                break;
            case APPE:
                printf("Application defined marker E \xFF\0X\n",data_byte);
                InterpretAPP(14);
                break;
            case APPF:
                printf("Application defined marker F \xFF\0X\n",data_byte);
                InterpretAPP(15);
                break;
        }
    }
}
case COM:
    printf("Comment : FF%02X\n", data_byte);
    InterpretComment();
    break;

case SOFO:
    printf("Start of Frame, Baseline DCT  FF%0X\n", data_byte);
    return;

case SOF1:
    printf("Start of Frame, Extended sequential DCT\n");
    printf("Extended sequential DCT not implemented\n");
    close(file);
    exit(1);
    break;

case SOF2:
    printf("Start of Frame, Progressive DCT\n");
    printf("Progressive DCT not implemented\n");
    printf("ABORTING\n");
    close(file);
    exit(1);
    return;

case SOF3:
    printf("Start of Frame, Spatial (sequential) lossless\n");
    printf("Spatial (sequential) lossless not implemented\n");
    printf("ABORTING\n");
    close(file);
    exit(1);
    return;

case SOF5:
    printf("Start of Frame, Differential sequential DCT\n");
    printf("Differential sequential DCT not implemented\n");
    printf("ABORTING\n");
    close(file);
    exit(1);
    return;

case SOF6:
    printf("Start of Frame, Differential progressive DCT\n");
    printf("Differential progressive DCT not implemented\n");
    printf("ABORTING\n");
    close(file);
    exit(1);
    return;

case SOF7:
    printf("Start of Frame, Differential spatial\n");
    printf("Differential spatial not implemented\n");
    printf("ABORTING\n");
    close(file);
    exit(1);
    return;

case TEM:
    printf("Temp private use in arithmetic coding\n");
    break;

default:
    if ((data_byte >= RESL) && (data_byte <= RESU))
    {
        printf("Reserved marker used FF%02X\n", data_byte);
    }
    else
    {
        printf("Unknown marker\n");
        printf(" marker = FF%02X\n", data_byte);
    }
    break;

    marker = OFF;
}
else
{
    marker = OFF;
}
}

close(file);
printf("ERROR!! Start of Frame NOT found\n");
printf("Exiting\n\n");
exit(1);

GetFileName(char *filename)
{
    int fileptr;
    char lastfile[81];
    FILE *lastused;

    lastused = fopen("lastused","r");
    if (lastused == NULL)
printf("Error reading last used file\n ABORTING!\n\n");
fclose(lastused);
exit(1);
}

fscanf(lastused,"%s",lastfile);
printf("Enter file name [%s]: ",lastfile);
gets(filename);
if (filename[0] == 0)
    strcpy(filename,lastfile);
printf("file %s\n",filename);
fileptr = open(filename,O_RDONLY);
while (fileptr == -1) {
    printf("fileptr = %d\n",fileptr);
    printf("Invalid file\n");
    printf("Enter file name ");
    gets(filename);
    if (filename[0] == 'e') {
        close(fileptr);
        exit(0);
    }
    fileptr = open(filename,O_RDONLY);
}
fclose(lastused);
lastused = open("lastused","w");
fprintf(lastused,"%s",filename);
fclose(lastused);
close(fileptr);
}/**
******************************************************************************/
** DecodeFrame  
******************************************************************************/

void DecodeFrame()
{
    int res;
    char input[4];
    short data_byte;
    int marker,
    int counter;
    void DecodeScan();
    void DecodeFrameHeader();
    void DefineHuffmanTables();
    void DefineArithmeticTables();
    void DefineRestartInterval();
    void DefineQuantTables();
    void InterpretComment();
    counter  0;
    printf("Decoding Frame\n");
    DecodeFrameHeader();
    res  1;
    while (res != 0) {
        res = ReadAByte(&data_byte);
        if (data_byte == 0xFF) {
            marker = ON;
        } else if (marker == ON) {
            switch (data_byte) {
            case SOS:
                printf("Start of Scan FF%02X\n",data_byte);
                DecodeScan();
                break;
            case EOI:
                printf("End of Image FF%02X\n",data_byte);
                printf("Done Decoding Frame\n");
                return;
            case DHT:
                printf("Define Huffman tables FF%02X\n",data_byte);
                DefineHuffmanTables();
                break;
            case DAC:
                printf("Define Arithmetic Conditioning FF%02X\n",data_byte)
DefineArithmeticTables();
                break;
            case EMT:
                printf("Define Restart Interval FF%02X\n",data_byte);
                DefineRestartInterval();
                break;
            case END:
                printf("End of File FF%02X\n",data_byte);
                close(fileptr);
                break;
            default:
                printf("Unknown byte FF%02X\n",data_byte);
                printf("Invalid byte \n");
                break;
            }  
        }
    }

C-6
case DQT:
    printf("Define Quantization Tables\n\n\n",data_byte);
    DefineQuantTables();
    break;

case DRI:
    printf("Define Restart Interval FF%02X\n",data_byte);
    DefineRestartInterval();
    break;

case APP0:
    printf("Application defined marker 0 FF%02X\n",data_byte);
    break;

case APP1:
    printf("Application defined marker 1 FF%02X\n",data_byte);
    break;

case APP2:
    printf("Application defined marker 2 FF%02X\n",data_byte);
    break;

case APP3:
    printf("Application defined marker 3 FF%02X\n",data_byte);
    break;

case APP4:
    printf("Application defined marker 4 FF%02X\n",data_byte);
    break;

case APP5:
    printf("Application defined marker 5 FF%02X\n",data_byte);
    break;

case APP6:
    printf("Application defined marker 6 FF%02X\n",data_byte);
    break;

case APP7:
    printf("Application defined marker 7 FF%02X\n",data_byte);
    break;

case APP8:
    printf("Application defined marker 8 FF%02X\n",data_byte);
    break;

case APP9:
    printf("Application defined marker 9 FF%02X\n",data_byte);
    break;

case APPA:
    printf("Application defined marker A FF%02X\n",data_byte);
    break;

case APPB:
    printf("Application defined marker B FF%02X\n",data_byte);
    break;

case APPC:
    printf("Application defined marker C FF%02X\n",data_byte);
    break;

case APPD:
    printf("Application defined marker D FF%02X\n",data_byte);
    break;

case APPF:
    printf("Application defined marker E FF%02X\n",data_byte);
    break;

case TEM:
    printf("Temp private use in arithmetic coding\n\n\n",data_byte);
    break;

case COM:
    printf("Comment\n\n\n",data_byte);
    InterpretComment();
    break;

default:
    if (data_byte >= RESL && data_byte <= RESU)
    {
        printf("Reserved marker used FF%02X\n",data_byte);
    }
    else if (data_byte >= RSTL && data_byte <= RSTU)
    {
        printf("Restart marker FF%02X\n",data_byte);
    }
    else if (data_byte == 0 || (data_byte == 0xFF))
    {
        counter--;
        \/* do nothing ,stuffed bytes */
    }
    else
    {
        printf("Unknown marker\n\n\n",data_byte);
        printf(" marker = FF%02X\n",data_byte);
    }
    break;

    marker = OFF;
    counter++;
    }
else
    {
        marker = OFF;
    }

if (counter == 10)
printf("[RET] to continue :"");
 gets(input);
 if (input[0] == 'e') {
   printf("ABORTING\n");
   return;
 }

 printf("Error!
! No End of Image found!!
!\n\n");
 return;

/**
 * DecodeScan
 */
void DecodeScan()
{
  void OpenOutputFile();
  void WriteOutHeader();
  struct timeval {
    unsigned long tv_sec;
    long tv_usec;
  } first, second, lapsed;
  struct timezone {
    int tz_minuteswest;
    int tz_dsttime;
  } tzp;
  short DecodeMCU(short x, short y, short hmax, short vmax, int num_blocks);
  void DecodeScanHeader();
  void DecodeRestartInterval();
  void ConvertRGBA();
  int wid, height;
  int hmax, vmax;
  short i;
  short x,y;
  short cnt, m, n;
  char data[80];
  int num_blocks;
  short rst, res;
  DecodeScanHeader();
  printf("Decoding Scan\n");
  if (NumlmgCompScan == 1) {
    if (DEBUG) printf("Scan in non-interleaved\n");
  } else {
    if (DEBUG) printf("Scan is interleaved with %d scans", NumlmgCompScan);
  }
  if (RestartInterval == 0) {
    if (DEBUG) printf("Restart NOT Enabled\n");
    PRED[0] = 0;
    PRED[1] 0;
    PRED[2] 0;
    PRED[3] 0;
    OpenOutputFile();
    WriteOutHeader();
    if ((DEBUG) {
      printf("Beginning to decode Scan\n");
    }
    CNT 0;
    hmax = vmax = 1;
    for (i 0; i < NumlmgCompScan; i++) {
      if (HorzSampFactor[i] > hmax) hmax = HorzSampFactor[i];
      if (VertSampFactor[i] > vmax) vmax = VertSampFactor[i];
    }
    wid = (int)ceil((double)NumSamplesImage/8.0)/(double)hmax;
    height = (int)ceil((double)NumLineImage/8.0)/(double)vmax;
    printf("Number of MCU's to decode = %dx%d", wid*height);
    num_blocks = 0;
    for (i 0; i < NumlmgCompScan; i++) {
      num_blocks = num_blocks + HorzSampFactor[i]*VertSampFactor[i];
    }
}
if (DEBUG) printf("Number of 8 x 8 blocks is \d \n", num_blocks);

gmtimeofday(&first, &step);
x = y = 0;
for (i = 0; i < (width * height); i++)
{
  if (DEBUG) printf("MCU Id \n", i + 1);
  if (DEBUG) printf("Corner (x,y) : (%d,%d)\n", x, y);
  res = DecodeMCU(x, y, hmax, vmax, num_blocks);
  x = x + (8 * hmax);
  if (x >= NumSamplesImage)
  {
    x = 0;
    y = y + (8 * vmax);
  }
}
gmtimeofday(&second, &step);
if (first.tv_usec > second.tv_usec)
{
  second.tv_usec -= 1000000;
  second.tv_usec = second.tv_usec + 1;
}
lapsed.tv_usec = second.tv_usec - first.tv_usec;
lapsed.tv_sec = second.tv_sec - first.tv_sec;
printf("Elapsed time to decode = %u sec \n", lapsed.tv_sec, lapsed.tv_usec);
if (!DEBUG)
{
  printf("Done decoding\n");
}
if (NumImgCompScan > 1)
{
  if (DEBUG) printf("Converting to RGB\n");
  ConvertRGB();
  if (DEBUG) printf("Done converting to RGB\n");
  /* write data to file */
  if (DEBUG) printf("Writing image to file\n");
  cnt = 0;
  for (y = 0; y < NumLineImage; y++)
  {
    for (x = 0; x < NumSamplesImage; x++)
    {
      for (i = 0; i < NumImgCompScan; i++)
      {
        sprintf(data, "$d \n", scans[i].line[x][y]);
        write(outfile, data, 4);
        cnt++;
      }
      if (cnt >= 17)
      {
        cnt = 0;
        sprintf(data, "$n\n");
        write(outfile, data, strlen(data));
      }
    }
  }
}
else
{
  printf("Restart Enabled\n");
  OpenOutputFile();
  WriteOutHeader();
  num_blocks = 0;
  for (i = 0; i < NumImgCompScan; i++)
  {
    num_blocks = num_blocks + HorzSampFactor[i] * VertSampFactor[i];
  }
  if (DEBUG) printf("Number of 8 x 8 blocks is \d \n", num_blocks);
  CNT = 0;
  hmax = vmax = 1;
  for (i = 0; i < NumImgCompScan; i++)
  {
    if (HorzSampFactor[i] > hmax) hmax = HorzSampFactor[i];
    if (VertSampFactor[i] > vmax) vmax = VertSampFactor[i];
    wid = (int)ceil((double)NumSamplesImage / (8.0) / (double)hmax);
    height = (int)ceil((double)NumLineImage / (8.0) / (double)vmax);
    printf("Number of MCU's to decode = %d\n", wid * height);
    num_blocks = 0;
    for (i = 0; i < NumImgCompScan; i++)
    {
      num_blocks = num_blocks + HorzSampFactor[i] * VertSampFactor[i];
    }
    if (DEBUG) printf("Number of 8 x 8 blocks is \d \n", num_blocks);
    i = 0;
  }
while (i < (wid*height)) & (rst < RestartInterval))

while (i < (wid*height))
{
    if (DEBUG) printf(" MCU \d \n",i+1);
    if (DEBUG) printf("Corner (x,y) : \d\d\n",x,y);
    res = DecodeMCU(x,y,hmax,vmax,num_blocks);
    if (res == (short)NORESTART)
    {
        x = x + (8*hmax);
        if (x >= NumSamplesImage)
        {
            x = 0;
            y = y + (8*vmax);
        }
        i++;
    }
    else
    {
        if (DEBUG) printf("Restart interval \d\n",rst+i);
        PRED[0] = 0;
        PRED[1] = 0;
        PRED[2] = 0;
        PRED[3] = 0;
        rst++;
    }
}

if (DEBUG)
{
    printf("Done decoding\n");
}

if (NumImgCompScan > 1)
{
    if (DEBUG) printf("Converting to RGB\n");
    ConvertRGB();
    if (DEBUG) printf("Done converting to RGB\n");
    /* write data to file */
    if (DEBUG) printf("Writing image to file\n");
    cnt = 0;
    for (y = 0; y < NumLineImage; y++)
    {
        for (x = 0; x < NumSamplesImage; x++)
        {
            for (i = 0; i < NumImgCompScan; i++)
            {
                sprintf(data,"%d",scans[i].line(x)(y));
                write(outfile,data,i);
                cnt++;
            }
            if (cnt >= 17)
            {
                cnt = 0;
                sprintf(data,"\n");
                write(outfile,data,strlen(data));
            }
        }
    }
    printf(" Done Decoding Scan\n");
    return;
}

/*******************************************************************
 ** DecodeScanHeader
 *********************************************************************/

void DecodeScanHeader()
{
    char data[];
    short data_byte;
    short data_byte2;
    int i;
    short counter;
    int Cs;
    short Td, Ta;
    int sum;
    printf(" Decode Scan Header\n");
    ReadAByte(data_byte);
    ReadAByte(data_byte2);
    if (DEBUG) printf(" Scan Header len = \dX\dX \n",data_byte,data_byte2);
    if (DEBUG) printf("Scan Header len = \d\n",(data_byte << 8) + data_byte2)
ReadAByte(_data_byte);
NumImgCompScan = _data_byte;
if (DEBUG) printf(" Number of Image components in Scan = \n",NumImgCompScan);
if (NumImgCompScan > 1) {
  if (DEBUG) printf(" Checking for Ns greater than 1 here!
");
  sum = 0;
  for (i = 0; i < NumImgCompScan; i++) {
    sum = sum + HorizSampFactor[i] + VertSampFactor[i];
  }
  if (sum > 10) {
    printf(" ERROR, sum > 10 in scan header check!
");
    printf(" ABORTING!
");
    close(file);
    exit(1);
  } else {
    if (DEBUG) printf(" Ns > 1 checks good! sum = %dn",sum);
  }
}

for (i = 0; i < NumImgCompScan; i++) {
  ReadAByte(_data_byte);
  Cs = _data_byte;
  ScanCompIdent[i] = Cs;
  ReadAByte(_data_byte2);
  Td = (_data_byte2 >> 4) & 0x0F;
  Ta = _data_byte2 & 0x0F;
  if (DEBUG) printf(" Scan Comp Selector \n",Cs);
  if (DEBUG) printf(" DC table = $d AC table = $d, Td, Ta",

  ScanDCTableSel[i] = Td;
  ScanACTableSel[i] = Ta;
}

for (i = 0; i < NumImgCompScan; i++) {
  if (DEBUG) printf(" Scan Frame \n",ScanCompIdent[i],CompIdent[i]);
}

ReadAByte(_data_byte);
StartSpectral = _data_byte;
if (DEBUG) printf(" Start Spectral = \n",StartSpectral);
StartSpectral = 0; /* FOR DCT modes */

ReadAByte(_data_byte);
EndSpectral = _data_byte;
if (DEBUG) printf(" End Spectral = \n",EndSpectral);
EndSpectral = 63; /* FOR DCT modes */

ReadAByte(_data_byte);
SuccAppxBitHigh = (_data_byte >> 4) & 0x0F;
SuccAppxBitLow = _data_byte & 0x0F;
if (DEBUG) printf(" Successive Appx Bit High = %d Low = %d
",SuccAppxBitHigh,SuccAppxBitLow);
  SuccAppxBitHigh = SuccAppxBitLow = 0; /* DCT modes of operation */
if (DEBUG) printf(" Done Decoding Scan Header!\n");

/*--------------------------------------------------------------------------------*/
**
** InterpretComment
**
--------------------------------------------------------------------------------*/

void InterpretComment()
{
  char data[1];
  short data_byte;
  short data_byte2;
  int numbytes;
  int i;
  short counter;
  ReadAByte(_data_byte);
  ReadAByte(_data_byte2);
  printf(" COM length = $02X$02X \n",data_byte,data_byte2);
  numbytes = data_byte2 << 8 + data_byte;
  printf("Comment length $d\n",numbytes);
  printf(" COMMENT FOLLOWS \n\n");
  counter = 0;
  for (i = 0; i < numbytes-2; i++)
  {
    counter++;

C-11
ReadAByte(&data_byte);
if ((data_byte > 'A') && (data_byte < 'Z'))
    printf("%c",data_byte);
else if ((data_byte == 'a') && (data_byte <= 'z'))
    printf("%c",data_byte);
else if ((data_byte == '0') && (data_byte <= '9'))
    printf("%c",data_byte);
else
    printf(".");
if (counter == 30)
{
    counter = 0;
    printf("\n\n");
}
printf("\n");

void
InterpretAPP(int type)
{
    char data[1];
    short data_byte;
    short data_byte2;
    int numbytes;
    int i;
    short counter;
    int res;

    ReadAByte(&data_byte);
    ReadAByte(&data_byte2);
    printf(" APP length = %02X%02X",data_byte,data_byte2);
    numbytes = (data_byte << 8) + data_byte2;
    printf(" APP length = %dn",numbytes);

    printf(" Application id data :\n\tVt",type);
    counter = 0;
    for (i = 0; i < numbytes-2; i++)
    {
        counter++;
        ReadAByte(&data_byte);
        if ((data_byte > 'A') && (data_byte < 'Z'))
            printf("%c",data_byte);
        else if ((data_byte == 'a') && (data_byte <= 'z'))
            printf("%c",data_byte);
        else if ((data_byte == '0') && (data_byte <= '9'))
            printf("%c",data_byte);
        else
            printf(".");
    }
    if (counter == 30)
    {
        counter = 0;
        printf("\n\n");
    }
    printf("\n");

    //******************************************************************************
    // InterpretAPP
    //******************************************************************************

    void
DefineQuantTables()
{
    char data[1];
    short data_byte;
    short data_byte2;
    int i, j;
    int numbytes;
    short P, T;
    short num_tables;
    short x,y;

    ReadAByte(&data_byte);
    ReadAByte(&data_byte2);
    if (DEBUG) printf(" Quant length = %02X%02X",data_byte,data_byte2);
    numbytes = (data_byte << 8) + data_byte2;
    if (DEBUG) printf(" Quant length = %dn",numbytes);
    num_tables = (numbytes-2)/65;
if (DEBUG) printf(" Number of Tables = %d\n", num_tables);
for (i = 0; i < num_tables; i++)
{
    ReadAByte(&data_byte);
    P = (data_byte >> 4) & 0x0F;
    T = data_byte & 0x0F;
    if (DEBUG) printf(" P = %02X T = %12X,P,T);
    if (P == 0)
    {
        if (DEBUG) printf(" Quantization tables are 8 bit\n");
    }
    else
    {
        printf("Quantization tables are 16 bit\n");
        printf("NOT supported\n");
        close(file);
        exit(1);
    }
}

if (DEBUG) printf(" Loading Quantization Table #d\n ",T);
for (j = 0; j < 64; j++)
{
    ReadAByte(&data_byte);
    QTables[T][j] = data_byte;
    if (DEBUG) printf("%2d
",QTables[T][j]);
}
if (DEBUG) printf("\n");

if (DEBUG) printf(" Unzipped Quantization Table #d\n ",T);
for (y = 0; y < 8; y++)
{
    for (x = 0; x < 8; x++)
    {
        if (DEBUG) printf("%d
",QTables[T][zigzag(y*8)+x]);
    }
    if (DEBUG) printf("\n");
}

if (DEBUG) printf("\n");

/***************************************************************************/
** DecodeFrameHeader
***************************************************************************/
void DecodeFrameHeader()
{
    char data[1];
    short data_byte;
    short data_byte2;
    short data_byte3;
    int i;
    ReadAByte(&data_byte);
    ReadAByte(&data_byte2);
    if (DEBUG) printf(" Frame Header len = %02X&02X",data_byte,data_byte2);
    if (DEBUG) printf(" Frame Header len = %d\n",(data_byte << 8) + data_byte2);
    ReadAByte(&data_byte);
    precision = data_byte;
    if (DEBUG) printf(" Precision = %d\n",precision);
    ReadAByte(&data_byte);
    ReadAByte(&data_byte3);
    NumLineImage = (data_byte << 8) + data_byte3;
    ReadAByte(&data_byte);
    ReadAByte(&data_byte2);
    NumSamplesImage = (data_byte << 8) + data_byte2;
    printf(" Image size = (%4d x %4d)\n",NumSamplesImage,NumLineImage);
    ReadAByte(&data_byte);
    NumImgCompFrame = data_byte;
    if (DEBUG) printf(" Number of Image components in frame = %d\n",NumImgCompFrame);
    for (i = 0; i < NumImgCompFrame; i++)
    {
        ReadAByte(&data_byte);
        ReadAByte(&data_byte2);
        ReadAByte(&data_byte3);
    }
CompIdent[i] data_byte;
HorizSampFactor[i] = (data_byte2 >> 4) & 0x0F;
VertSampFactor[i] = data_byte2 & 0x0F;
QuantTableSelector[i] = data_byte;

if (DEBUG)
  printf(" Comp %d Quant Table
", CompIdent[i], QuantTableSelector[i]);
if (DEBUG)
  printf(" Horz Factor %d Vert Factor = %d
", HorizSampFactor[i], VertSampFactor[i]);

/******************************************************************************
** DefineHuffmanTables
*******************************************************************************/

void DefineHuffmanTables()
{
  void LoadAHuffTable(short Htable, short Htype);
  short data_byte;
  short data_byte2;
  int numbytes;
  int i, j, k;
  short Tc, Th;
  ReadAByte(data_byte);
  ReadAByte(&data_byte2);

  if (DEBUG) printf(" Huffman length - %dX802X", data_byte, data_byte2);
  numbytes = (data_byte8) + data_byte2;
  if (DEBUG) printf(" Huffman length - %d
", numbytes);
  i = 0;
  while (i < (numbytes-2))
  {
    ReadAByte(sdata_byte);
    i++;
    Tc = (data_byte >> 4) & 0x0F;
    Th = (data_byte & 0x0F);
    if (DEBUG) printf(" Loading Huffman Table %d, type is ", Th);
    if (Tc == 0)
      if (DEBUG) printf("DC
")
    else if (Tc == 1)
      if (DEBUG) printf("AC
")
    else
      printf("Error with Huffman table type\n");
      printf(" ABORTING\n");
      close(file);
      exit(1);
  }

  /* read in a BITS array */
  for (j = 1; j < 17; j++)
  {
    ReadByte(data_byte);
    BITS[j] = data_byte;
    huffman_tables[Th][Tc].BITS[j] = data_byte;
    i++;
    if (DEBUG)
      printf("BITS[%d] %d %x\n", j, huffman_tables[Th][Tc].BITS[j],
               huffman_tables[Th][Tc].BITS[j]);
  }

  /* read in a HUFFVAL array */
  1 = 0;
  for (j = 1; j < 17; j++)
  {
    for (k = 0; k < BITS[j]; k++)
    {
      i++;
      ReadByte(data_byte);
      HUFFVALS[j] = data_byte;
      huffman_tables[Th][Tc].HUFFVALS[j] = data_byte;
      if (DEBUG)
        printf("HUFFVALS[%d] %d %x\n", 1, huffman_tables[Th][Tc].HUFFVALS[j],
                 huffman_tables[Th][Tc].HUFFVALS[j]);
    }
  }

C-14
LoadAHuffTable(Th,Tc);
if (i != (numbytes-2))
printf("i != numbytes-2");

void LoadAHuffTable(short Htable, short Htype)
{
    void GenHuffTableSize(short Htable, short Htype, int *HUFFSIZE, int *lastk);
    void GenHuffCodeTables(short Htable, short Htype, int *HUFFSIZE, int *HUFFCODE);
    void HuffGen(int Htable, int Htype);
    int lastk;
    int HUFFSIZE[256];
    int HUFFCODE[256];
    /* Generate Table Size */
    GenHuffTableSize(Htable, Htype, HUFFSIZE, lastk);
    /* Generate Code Table */
    GenHuffCodeTables(Htable, Htype, HUFFSIZE, HUFFCODE);
    HuffGen(Htable, Htype);
}

void HuffGen(int Htable, int Htype)
{
    int I, j;
    i = j = 0;
    while (i <= 16)
    {
        if (huffman_tables[Htable][Htype].BITS[i] != 0)
        {
            huffman_tables[Htable][Htype].VALPTR[i] = j;
            huffman_tables[Htable][Htype].MINCODE[i] =
            huffman_tables[Htable][Htype].HUFFCODE[j];
            j = j + huffman_tables[Htable][Htype].BITS[i] + 1;
            huffman_tables[Htable][Htype].MAXCODE[i] =
            huffman_tables[Htable][Htype].HUFFCODE[j];
            j++;
        }
        else
        {
            huffman_tables[Htable][Htype].MAXCODE[i] = -1;
        }
    }
    for (i = 0; i < 17; i++)
    {
        if (HDEBUG)
            printf("MINCODE[83d]: 84d 84x MAXCODE[83d] 84d 84x VALPTR[83d] 84d 84x
", i, huffman_tables[Htable][Htype].MINCODE[i],
            huffman_tables[Htable][Htype].MINCODE[i],
            huffman_tables[Htable][Htype].MAXCODE[i],
            huffman_tables[Htable][Htype].VALPTR[i],
            huffman_tables[Htable][Htype].VALPTR[i]);
    }
}

void GenHuffCodeTables(short Htable, short Htype, int *HUFFSIZE, int *HUFFCODE)
{
    int k, CODE, SI;
    int DONE;
    DONE = 0;
```c
k = 0;
CODE = 0;
SI = HUFFSIZE[0];

while (DONE -- 0)
{
    HUFFCODE[k] = CODE;
huffman_tables[Htable][Htype].HUFFCODE[k] = CODE;
    if (HDEBUG)
        printf("HUFFCODE[%d] = %d\n", k, huffman_tables[Htable][Htype].HUFFCODE[k]);
    CODE++;
    k++;
    if (HUFFSIZE[k] != SI)
    {
        if (HUFFSIZE[k] == 0)
            return;
        else
        {
            SI++;
            CODE CODE <<= 1;
            while (HUFFSIZE[k] != SI)
            {
                SI++;
                CODE = CODE << 1;
            }
        }
    }
}

/**
 * GenHuffTableSize
 *
 * @brief Generates Huffman table size
 *
 * @param Htable Huffman table
 * @param Htype Huffman type
 * @param HUFFSIZE Huffman size table
 * @param lastk Last k value
 */
void GenHuffTableSize(short Htable, short Htype, int *HUFFSIZE, int *lastk)
{
    int i, j, k;
    k = 0;
    i j 1;

    while (i <= 16)
    {
        while (j <= BITS[i])
        {
            HUFFSIZE[k] = i;
            huffman_tables[Htable][Htype].HUFFSIZE[k] = i;
            if (HDEBUG)
                printf("HUFFSIZE[%d] = %d\n", k, huffman_tables[Htable][Htype].HUFFSIZE[k]);
            CODE++;
            j++;
        }
        j = 1;
        i++;
    }
    HUFFSIZE[k] = 0;
    huffman_tables[Htable][Htype].lastk = k;
    *lastk = k;
}

/**
 * DefineArithmeticTables
 *
 * @brief Defines arithmetic tables
 */
void DefineArithmeticTables()
{
    printf(" Arithmetic Tables not implemented\n");
    printf(" ABORTING\n");
    close(file);
    exit(1);
}

/**
 * DefineRestartInterval
 *
 * @brief Defines restart interval
 */
void DefineRestartInterval()
{
    char data[1];
    short data_byte;
    short data_byte2;
    int numbytes;
```
ReadAByte(\*data_byte);
ReadAByte(\*data_byte);

printf(" Restart int len = %02X\%02X", data_byte, data_byte2);
numbytes = (data_byte << 8) + data_byte2;
printf(" Restart int len %dn",numbytes);

ReadAByte(\*data_byte);
ReadAByte(\*data_byte2);
RestartInterval = (data_byte << 8) + data_byte2;

printf(" Restart Interval %dn", RestartInterval);

/*******************************************************************************/

**
** ReadAByte
**
*******************************************************************************/

int ReadAByte(short *d_byte)
{
    char data[1];
    int res;
    res = read(file, data, 1);
    *d_byte = data[0];
    *d_byte = *d_byte & 0x00FF;
    if (DEBUG) printf("Byte 1.2x\n".*d_byte);
    byte_counter++;
    return res;
}

/*******************************************************************************/

**
** DecodeRestartInterval
**
*******************************************************************************/

void DecodeRestartInterval()
{
}

/*******************************************************************************/

**** DecodeMCU
** For sequential baseline, a MCU is an 8x8 block;
**
*******************************************************************************/

short DecodeMCU(short startX, short startY, short hmax, short vmax, int num_blocks)
{
    short DecodeDCCoeff(short comp, int DCsel);
    short DecodeACCoeff(int ACsel);
    void CalcInverse(int QTable);
    short res;
    short x, y;
    short dx, dy;
    short sx, sy;
    short lx, ly;
    short wid;
    short i, j, n, m, k, l;
    char input[3];
    for (i = 0; i < NumImgCompScan; i++)
    {
        wid = Horz Samp Factor[i] * 8;
        x = y = 0;
        for (j = 0; j < (Horz Samp Factor[i] * Vert Samp Factor[i]); j++)
        {
            res = DecodeDCCoeff(i, ScanDCTableSel[i]);
            if (res == (short)RESTART)
            {
                if (DEBUG) printf("Returning Restart\n");
                return (short)RESTART;
            }
            res = DecodeACCoeff(ScanACTableSel[i]);
            if (res == (short)RESTART)
                return (short)RESTART;
            CalcInverse(QuantTableSelector[i]);
            m = 0;
        }
    }
}
sy = startY + y;
for (k = startY; k < (startY+y+8); k++)
{
    n = 0;
    sx = startX + x;
    for (l = startX+x; l < (startX+x+8); l++)
    {  
        if ((ss[m][n] < 0) || (ss[m][n] > 255))
        {
            if (ss[m][n] < 0)
                ss[m][n] = 0;
            else
                ss[m][n] = 255;
        }
        if ((hmax == HorzSampFactor[i]) && (vmax == VertSampFactor[i]))
        {
            scans[i].line[l][k] = ss[m][n];
        }
        else
        {
            for (dy 0; dy < vmax/VertSampFactor[i]; dy++)
            {
                for (dx 0; dx < hmax/HorzSampFactor[i]; dx++)
                {
                    lx = sx + dx;
                    ly = sy + dy;
                    scans[i].line[lx][ly] = ss[m][n];
                }
            }
            n++;
        }
    }
    n++;
    sx = sx + hmax/HorzSampFactor[i];
    m++;
    sy = sy + vmax/HorzSampFactor[i];
    x = x + 8;
    if (k == wid)
    {
        x 0;
        y = y + 8;
    }
}
return (short)NORESTART;

/**************************************************************************
 ** CalcInverse
 ** ************************************************************************/

void CalcInverse(int QTable)
{
    void DeQuantize(int QTable);
    void IDCT();

    DeQuantize(QTable);
    IDCT(QTable);
}

/**************************************************************************
 ** DecodeACCoeff
 ** ************************************************************************/

short DecodeACCoeff(int ACsel)
{
    short i,k;
    short RS;
    short DECODE(int ACsel, int Type, short *rst);
    short SSSS;
    short R;
    void Decode_ZZ(short k, short SSSS, short *rst);
    short i;
    for (i = 1; i < 64; i++)
    {  
        ZZ[i] = 0;
        while (i)
        {  
            if (k > 63)
            {
                printf("ERROR! k > 63\n ABORTING\n")
                printf("%d\n",k);
                close(file);
                exit(1);
RS = DECODE(ACsel, AC, &rst);
if (ADSBug) printf("DecodeAC RS = %.2X\n", RS);
if (rst == (short)RESTART)
{
    if (DEBUG) printf("DecodeAC returning restart\n");
    return (short)RESTART;
}

SSSS = RS & 0x0F;
R = (RS & 0xF0) >> 4;
if (SSSS == 0)
{
    if (R != 15)
        return;
    else
        k = k + 16;
}
else
{
    k = k + R;
    Decode_ZZ(k, SSSS, &rst);
    if (rst == (short)RESTART)
        return (short)RESTART;
    if (k == 63)
        return;
    else
        k++;
}
return (short)NORESTART;

/*********************************************************
** ** Decode_ZZ
** ** pg. F-24, fig F.14
**********************************************************/
void Decode_ZZ(short k, short SSSS, short *rst)
{
    short RECEIVE(int SSSS, short *rst);
    short EXTEND(int DIFF, int SSSS);
    ZZ[k] = RECEIVE(SSSS, rst);
    if (*rst == (short)RESTART)
        return;
    ZZ[k] = EXTEND(ZZ[k], SSSS);
}

/***********************************************************/
** ** DecodeDCCoeff
** **
/***********************************************************/
short DecodeDCCoeff(short comp, int DCsel)
{
    short DECODE(int Htable, int type, short *rst);
    short RECEIVE(int SSSS, short *rst);
    short EXTEND(int DIFF, int T);
    short T, short rst;

    T = DECODE(DCsel, DC, &rst);
    if (ADSBug) printf("DC decode return %.2X\n", T);
    if (rst == (short)RESTART)
    {
        if (DEBUG) printf("DecodeDCCoeff (1) returning restart\n");
        return (short)RESTART;
    }
    DIFF = RECEIVE(T, &rst);
    if (rst == (short)RESTART)
        return rst;
    DIFF = EXTEND(DIFF, T);
    ZZ[0] = PRSD[comp] + DIFF;
    PRSD[comp] = ZZ[0];
    return rst;
}

/***********************************************************/
** ** RECEIVE
** **
RECEIVE(int SSSS, short *rst)
{
    short NEXTBIT(short *rst);
    short i, v;
    short tmp;
    i = v = 0;
    while (i != SSSS)
    {
        tmp = NEXTBIT(rst);
        if (*rst == (short)RESTART)
        {
            if (DEBUG) printf("RECEIVE returning restart\n");
            return;
        }
        v = (v << 1) + tmp;
        i++;
    }
    return v;
}

DECODE(int Htable, int Htype, short *rst)
{
    short NEXTBIT(short *rst);
    short i, j;
    short VALUE, CODE;
    i = 1;
    CODE = NEXTBIT(rst);
    if (*rst == (short)RESTART)
    {
        if (DEBUG) printf("DECODE (1) returning Restart\n");
        return;
    }
    while (CODE > huffman_tables[Htable][Htype].MAXCODE(i))
    {
        if (ADEBUG) printf("CODE = %.2X\n", CODE);
        if (ADEBUG) printf("i = %d\n", i);
        CODE = (CODE << 1) + NEXTBIT(rst);
        if (*rst == (short)RESTART)
        {
            if (DEBUG) printf("DECODE (2) returning restart\n");
            return;
        }
        i++;
    }
    if (ADEBUG) printf("CODE = %.2X\n", CODE);
    if (ADEBUG) printf("i = %d\n", i);
    j = huffman_tables[Htable][Htype].VALPTR[i];
    if (ADEBUG) printf("j = %d\n", j);
    j = j + CODE - huffman_tables[Htable][Htype].MINCODE(i);
    if (ADEBUG) printf("j = %d\n", j);
    VALUE = huffman_tables[Htable][Htype].HUFFVALS[j];
    if (ADEBUG) printf("VALUE = %.4X\n", VALUE);
    return VALUE;
}

NEXTBIT(short *rst)
{
    static short B;
    static short B2;
    short mask;
    short BIT;
}
rst  (short)NORESTART;

if (CNT == 0)
{
    ReadADbyte(4B);
    CNT = 8;
    if (B == OxFF)
    {
        ReadADbyte(4B2);
        if (B2 != 0)
        {
            if (B2 == DNL)
            {
                printf("DNL marker found in scan, not programmed to handle\n");
                printf("ABORTING\n");
                close(file);
                exit(1);
            }
        }
        else
        {
            if (B2 >= RSTL && B2 <= RSTU)
            {
                if (DEBUG) printf("RST marker found FF\O2X\n",B2);
                *rst = (short)RESTART;
                CNT = 0;
                return;
            }
            else
            {
                printf("Unknown marker found FF\O2X\n",B2);
                printf("ABORTING\n");
                close(file);
                close(outfile);
                exit(1);
            }
        }
    }
}

mask  i < (CNT-1);
BIT  = B & mask;
BIT  = BIT >> (CNT-1);
CNT--;

return BIT;

/***************************************************************************/
**
** EXTEND
**
** F-22
**
******************************************************************************/

short EXTEND(int V, int T)
{
    short Vt;

    Vt = (int)pow(2.0,(double)(T-1));
    if (V < Vt)
    {
        Vt = (-1 << T) + 1;
        V = V + Vt;
    }

    return V;
}

/*******************************************************************************/
**
** DeQuantize
**
******************************************************************************/

void DeQuantize(int QTable)
{
    short i,j;

    if (DEBUG) printf("\n Unloaded\n\t");
    for (i = 0; i < 8; i++)
    {
        for (j = 0; j < 8; j++)
        {
            if (DEBUG) printf(" ,ZZ[(i*8)+j]);
        }
    }
}
if (DDEBUG) printf("\n\t");
}  
if (DDEBUG) printf("\n");
if (DDEBUG) printf("\n\t Dequantized and unZigZagged\n\t");
for (i = 0; i < 8; i++)
  for (j = 0; j < 8; j++)
    { 
      SS[i][j] = SS[sigzag[(i*8)+j]] * QTables[QTable[sigzag[(i*8)+j]]];
      if (DDEBUG) printf("% 4d ",SS[i][j]);
    }  
if (DDEBUG) printf("\n\t Deguantized
\tand
\tunZigZagged
\t");
for (i = 0; i < 8; i++)
  for (j = 0; j < 8; j++)
    { 
      S[I][SS[i][j]] = ZZ[sigzag[(i*8)+j]] * QTables[QTable[sigzag[(i*8)+j]]];
      if (DDEBUG) printf("% 4d ",SS[i][j]);
    }  
if (DDEBUG) printf("\n\t");
if (DDEBUG) printf("\n");

IDCT

Performs an Inverse Discrete Cosine Transform on an 8x8 array.
The method of using two 1-D transforms is performed.
Also the level shift operation is performed at the end of the
second 1-D transform. It is hard coded for eight bit data.

*******************************************************************************/

void IDCT()
{
  short a,y;
  short u,v;
  double a,b,Cv,Cu;
  double mid;
  double sum;
  short i,j;
  int num_blocks;
  short sss[8][8];

  if (DDEBUG) printf("Begin IDCT\n\t");
  for (y = 0; y < 8; y++)
    { 
      for (x = 0; x < 8; x++)
        { 
          sum = 0.0;
          for (u = 0; u < 8; u++)
            { 
              a = \( \cos\left(\frac{\pi \times (2u+1) \times \times \times)}{16}\right); 
              if (u == 0)
                Cu = M_SQRT1_2;
              else 
                Cu = 1.0;
              sum = sum + SS[y][u] \times Cu \times a ;
            } 
          sss[y][x] = (int)sum;
        } 
    } 
  for (x = 0; x < 8; x++)
    { 
      for (y = 0; y < 8; y++)
        { 
          sum = 0.0;
          for (v = 0; v < 8; v++)
            { 
              b = \( \cos\left(\frac{\pi \times (2v+1) \times \times \times)}{16}\right); 
              if (v == 0)
                Cv = M_SQRT1_2;
              else 
                Cv = 1.0;
              sum = sum + sss[y][x] \times Cv \times b ;
            } 
          sss[y][x] = (int)((sum/4.0) + 128);
        } 
    } 
  if (DDEBUG) printf("\n\n");
  for (y = 0; y < 8; y++)
    { 
      for (x = 0; x < 8; x++)
        { 
          if (DDEBUG) printf("% 4d ",ss[y][x]);
        } 
    } 
  if (DDEBUG) printf("\n\n");
  if (DDEBUG) printf("\n");
}
printf("\n");
if (IDBG)
printf("Finished IDCT\n");

/***********************************************************/
**
** OpenOutputFile
**
** open the output file for a PPM image
***********************************************************/

void
OpenOutputFile()
{
    outfile = open("out.ppm", O_RDWR | O_CREAT);
    if (outfile == -1)
    {
        printf("Error opening output file!!\n");
        printf("Aborting\n
");
        close(outfile);
        exit(1);
    }
}

void
WriteOutHeader()
{
    char data[80];
    if (NumImgCompScan == 1)
        write(outfile,"P2\n",3);
    else
        write(outfile,"P3\n",3);
    write(outfile,"# Doug's code\n",14);
    sprintf(data,"Idld
",NumSamplesImage,NumLineImage);
    write(outfile,data,strlen(data));
    write(outfile,"255\n",4);
}

/***********************************************************/
**
** ConvertRGB
**
** Convert YCrCb to RGB
**
/***********************************************************/

void
ConvertRGB()
{
    short x,y,i;
    float Y,Cb,Cr;
    char input[2];
    for (y = 0; y < NumLineImage; y++)
    {
        for (x = 0; x < NumSamplesImage; x++)
        {
            Y = (float)scans[0].line[x][y];
            Cb = (float)scans[1].line[x][y];
            Cr = (float)scans[2].line[x][y];
            scans[0].line[x][y] = (short)(Y + (1.402*(Cr-128.0)));
            scans[1].line[x][y] = (short)(Y - 0.34414*(Cb-128.0) - 0.71414*(Cr-128.0));
            scans[2].line[x][y] = (short)(Y + 1.772*(Cb-128.0));
            if (scans[0].line[x][y] < 0) scans[0].line[x][y] = 0;
            if (scans[1].line[x][y] < 0) scans[1].line[x][y] = 0;
            if (scans[2].line[x][y] < 0) scans[2].line[x][y] = 0;
            if (scans[0].line[x][y] > 255) scans[0].line[x][y] = 255;
            if (scans[1].line[x][y] > 255) scans[1].line[x][y] = 255;
            if (scans[2].line[x][y] > 255) scans[2].line[x][y] = 255;
        }
    }
}
** usage : conv input_file_name
**
*****************************************************************************************/

#define _INCLUDE_POSIX_SOURCE

#include <stdio.h>
#include <fcntl.h>
#include <sys/types.h>
#include <unistd.h>

/*------------------------------------------------------------------------*/
/* local global variables */
int infile;
int outfile;

main(int argc, char *argv[])
{
    void OpenInFile(char *infilename);
    void GetInFileName(char *infilename);
    void OpenOutFile();
    char infilename[80];
    char data[1];
    unsigned short d_byte;
    int counter;
    char str[3];

    if (argc == 1)
    {
        GetInFileName(infilename);
    }
    else if (argc == 2)
    {
        strcpy(infilename,argv[1]);
    }
    else
    {
        printf("Usage : conv [infile]\n");
        exit(1);
    }

    OpenInFile(infilename);
    OpenOutFile();

    counter = 0;
    while (read(infile,data,1) != NULL)
    {
        counter++;
        d_byte = data[0];
        d_byte = d_byte & 0x00FF;
        sprintf(str,"%02X",d_byte);
        write(outfile,str,2);
        if (counter == 30)
        {
            sprintf(str,"
");
            write(outfile,str,strlen(str));
            counter = 0;
        }
    }

    printf("Done!\n");
    close (infile);
}

/***************************************************************************/
/* OpenInFile */
/************************** Attempt to open the input file. */
/* If unable to open, abort. */
/***************************************************************************/

void
OpenInFile(char *infilename)
{
    infile = open(infilename, O_RDONLY);
    if (infile == -1)
    {
        printf("Error opening input file : %s\n", infilename);
        printf("Try Again   Aborting conversion\n\n");
        exit(1);
    }
}

/* *********************************************/
** GetInFileName
** Get the input file name, including path
** *********************************************/

void
GetInFileName(char *filename)
{
    int fileptr;
    char lastfile[80];
    FILE *lastused;

    lastused = fopen("lastused","r");
    if (lastused == NULL)
    {
        printf("Error reading last used file\n ABORTING!!\n\n");
        fclose(lastused);
        exit(1);
    }

    fscanf(lastused,"%s",lastfile);
    printf("Enter file name [%s]: ", lastfile);
    gets(filename);
    if (filename[0] == 0)
        strcpy(filename,lastfile);
    fileptr = open(filename,O_RDONLY);
    while (fileptr == -1)
    {
        printf("fileptr = %d\n", fileptr);
        printf("Invalid file\n");
        printf("Enter file name : ");
        gets(filename);
        if (filename[0] == 'e')
        {
            close(fileptr);
            exit(0);
        }
        fileptr = open(filename,O_RDONLY);
    }

    fclose(lastused);
    lastused = fopen("lastused","w");
    fprintf(lastused,"%s",filename);
    fclose(lastused);
    close(fileptr);
}

/* *********************************************/
** OpenOutFile
** Open the output file, hard coded name : vhdl.dat
** open to write over existing file
** *********************************************/

void
OpenOutFile()
{
outfile = open("vhdl.dat", O_RDWR | O_CREAT);
if (outfile == -1)
{
    printf("\nError opening output file : %d\n", outfile);
    printf(" Aborting\n
");
    close(infile);
    exit(1);
}

**display.c**

/*display.c

** Convert the output from the decoder, which has data as 8x8 blocks,
** into proper display format. I.E line by line. Write a ppm b/w file.
**
** usage
display
**
** input file name vhdl.ppm
**
** output file name display.ppm
**
*******************************************************************************/

#include <stdio.h>
#include <fcntl.h>
#include <sys/types.h>
#include <unistd.h>

/* local global variables */
FILE *infile;
FILE *outfile;

main()
{
    void OpenInFile();
    void OpenOutFile();
    char data[80];
    int line_block[64][1024];
    int xlen = 0;
    int ylen = 0;
    int line_cntr = 0;
    int vert_cntr = 0;
    int pix_cntr = 0;
    int x, y, xx, yy;
    int j, k;
    int i0;
    int i1;
    int i2;
    int i3;
    int i4;
    int i5;
    int i6;
    int i7;
    int i8;
    int i9;
    int i10;
    int i11;
    int i12;
    int i13;
    int i14;
    int i15;
    int i16;
    int i17;
    char input[9];
    printf("\nDisplay program in work\n");
    OpenInFile();
    OpenOutFile();

    /* next two lines are for test only to read in comments */
    fgets(data, 80, infile);
    fprintf(outfile, data);
    fgets(data, 80, infile);
    fprintf(outfile, data);
    if (fgets(data, 80, infile) == NULL)
    {
        close(infile);
        close(outfile);
        printf("Error reading X,Y dimension\n\n");
        exit(1);
    }
else

(
sscanf

(data,"Sd

printf

("X len

fprintf

%d", ixlen, sylen) ;
%d Y len
8d\n",xlen,ylen)

=

=

(outfile, data)

;

;

)
if

(fgets (data,

80, infile)

==

NULL)

(
fclose (infile) ;
fclose (outfile)

;

("Error reading

printf

num

colors\n\n") ;

exit(l);

)
else

fprintf

(outfile, data)

0;
y
line_cntr

x

;

=

0;
0;

vert_cntr
pix_cntr

0;

=

(fgets(data, 80, infile)

while

!=

NULL)

1
(data,"%d %d Sd Sd Sd Sd Sd Sd Sd Sd Sd Sd Sd Sd Sd Sd Sd ",

sscanf

Si0,sil,i2,si3,5i4,si5,si6,i7,si8,si9,sil0,sill,sil2,sil3,sil4,sil5,sil6);
(data) ;
iO;
il;
i2;
i3;
i4;
i5;
i6;
i7;
i8;
i9;
ilO;

printf

i[0]
ill]
1[2]
i[3]
i[4]
i[5]
i[6]
i[7]
i[8]
i[9]
i[10]
i[ll]
i[12]
i[13]
i[14]
i[15]
i[16]

=

=

=

=

=

=

-

ill
il2
il3

=

il4
=

il5

-

il6

=

0;

j

< 17)
(
i[j);
line_block[vert_cntr+y] [line_cntr+x]
printf ("Sd Sd
Sd\n", line_cntr+x,vert_cntr+y,i [j ] )

(j

while

=

=

;

x++;
pix_cntr++;

if

(x

8)

==

0;

=

x

y++;

if

8)

(y

(
0;

=

y

line_cntr+=

if

8;
>=

(line_cntr

xlen)

I
line_cntr =0;
vert_cntr+=

8;

)
)

1
k

=

for

0;

(y

0; y

=

<

ylen;

y++)

{
for

(x

0;

-

x

^

x++)

xlen;

[
fprintf (outfile, "83d

k++;
if (k

==

", line_block[y] [x] )

;

17)

(
k

=

0;

fprintf

(outfile, "\n")

;

1
1
)
fclose (infile) ;
fclose (outfile) ;
printf

("Display

program

complete

view

display. ppm\n\n") ;

OpenlnFile

C-27


void OpenInFile()
{
    infile = fopen("vhdl.ppm", "r");
    if (infile == NULL)
    {
        printf("Error opening input file vhdl.ppm\n");
        printf("Try Again  Aborting conversion\n\n");
        exit(1);
    }
}

void OpenOutFile()
{
    outfile = fopen("display.ppm", "w");
    if (outfile == NULL)
    {
        printf("\nError opening output file %d\n", outfile);
        printf(" Aborting\n");
        fclose(infile);
        exit(1);
    }
}

main(int argc, char *argv[])
{
    if (argc == 1)
    { GetInFileName(infilename); }
    else if (argc == 2)
    { strcpy(infilename, argv[1]); }
else
{
    printf("Usage compare [infile]\n"); exit(1); }

OpenInFile(infile); OpenOutFile();

printf("\nReading in headers...\n");
/* number of colors, must be P2 */
fgets(disdata,80,disfile);
fgets(indata,80,infile);
printf(".\n");
sscanf(disdata,"%c",distype);
sscanf(indata,"%c",intype);
if (strcmp(distype,intype) != 0)
{
    printf("FAIL\n");
    printf("\nThe PPM types did not match!! ABORTING!!!\n");
    fclose(infile);
    fclose(disfile);
    exit(1);
}

printf(".\n"); /* comment line */
fgets(disdata,80,disfile);
fgets(indata,80,infile);
printf(".\n"); /* x,y dimensions, must be the same */
fgets(disdata,80,disfile);
fgets(indata,80,infile);
sscanf(indata,"%d %d",inx,iny);
sscanf(disdata,"%d %d",disx,disy);
if (inx != disx)
{
    printf("FAIL\n");
    printf("\nX dimensions did not match!! ABORTING!!!\n");
    fclose(infile);
    fclose(disfile);
    exit(1);
}
else if(iny != disy)
{
    printf("FAIL\n");
    printf("\nY dimensions did not match!! ABORTING!!!\n");
    fclose(infile);
    fclose(disfile);
    exit(1);
}
printf(".\n"); /* number of possible colors; bit size */
fgets(disdata,80,disfile);
fgets(indata,80,infile);
printf(".\n");
sscanf(indata,"%d",inpix);
sscanf(disdata,"%d",dispix);
if (inpix != dispix)
{
    printf("FAIL\n");
    printf("\nBits/pixel count did not match!! ABORTING!!!\n");
    fclose(infile);
    fclose(disfile);
    exit(1);
}
printf(".\n");
printf("...DONE\n\n"); printf("Checking data...\n");
x = 0; y = 0; num_pix = hits = 0;
while((fgets(disdata,80,disfile) != NULL) && (fgets(indata,80,infile) != NULL))
{
    for (i = 0; i < 17; i++)
    {
       in_pix[i] -= 1;
       dis_pix[i] -= 1;
    }

    sscanf(indata,"%d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d",
for (i = 0; i < 17; i++)
{
    if (((in_pix[i] - dis_pix[i]) > 1) || ((in_pix[i] - dis_pix[i]) < -1))
    {
        hits++;
        printf("Diff > 1 (\%3d,\%3d) = in:\%3d dis:\%3d diff:\%3d
",
            x,y,in_pix[i],dis_pix[i],(in_pix[i]-dis_pix[i]));
        x++;
        num_pix++;
        if (x == inx)
        {
            x = 0;
            y++;
        }
    }
}
printf(". Done! \n");
printf("Number of pixels analyzed \t %d
Number of hits \t %d
\n",num_pix,hits);
fclose (infile);
fclose (disfile);

void OpenInFile(char *infilename)
{
    infile = fopen(infilename, "r");
    if (infile == NULL)
    {
        printf("Error opening input file \n",infilename);
        printf("Try Again - Aborting compare\n\n");
        exit(1);
    }
}

void GetInFileName()
{
    FILE *fileptr;
    char lastfile[80];
    printf("Enter file name (\%s): ",lastfile);
    gets(filename);
    if (filename[0] == 0)
        strcpy(filename,lastfile);
    fileptr = fopen(filename,"r");
    while (fileptr == NULL)
    {
        printf("Fileptr = \%d\n",fileptr);
        printf("Invalid file\n");
        printf("Enter file name : ");
        gets(filename);
        if (filename[0] == 'e')
        {
            fclose(fileptr);
            exit(0);
        }
        fileptr = fopen(filename,"r");
    }
}
fclose(fileptr);

/******************************************************
 * OpenOutFile
 * Open the output file, hard coded name vhdl.dat
 * open to write over existing file
 ******************************************************/

void OpenOutFile()
{

disfile = fopen("out.ppm", "w");
printf("Need to change to display.ppm here!!!\n");
if (disfile == NULL)
{
printf("\nError opening display file display.ppm\n");
printf(" Aborting\n");
fclose(infile);
exit(1);
}
}