A Hardware Platform for JPEG Compression/Decompression

Evan Broder and C. Christopher Post
Massachusetts Institute of Technology
E-mail: broder@mit.edu, ccpost@mit.edu

Introductory Digital Systems Laboratory

Abstract

This project describes a hardware platform for encoding and decoding JPEG image data on an FPGA. The encoding module takes image data and processes it using JPEG compression. The resulting compressed data is then framed for serial transmission and sent to the decoder with a checksum to ensure data integrity. This allows the JPEG compression and decompression units to communicate over a low-bandwidth serial communication line. The receiving module checks the incoming packets for integrity and passes the JPEG encoded data along to the decoder, where decompression is performed to produce a resulting image.
# Contents

1 Overview 1

2 Modules 1

2.1 Encoding 1

2.1.1 2-D DCT 1

2.1.2 1-D DCT 2

2.1.3 Matrix Transpose 3

2.1.4 Coordinate Delay 4

2.1.5 Quantizer 4

2.1.6 Entropy Coder 5

2.1.7 Huffman Coder 6

2.1.8 Huffman Serializer 7

2.2 Transmission/Reception 7

2.2.1 Packer 7

2.2.2 Packet Wrapper 8

2.2.3 Serial Asynchronous Transmitter 8

2.2.4 Unpacker 8

2.2.5 Packet Unwrapper 9

2.2.6 Serial Asynchronous Receiver 9

2.3 Decoding 10

2.3.1 Huffman Decoder 10

2.3.2 Entropy Decoder 11

2.3.3 Dequantizer 12

2.3.4 1-D iDCT 12

2.3.5 2-D iDCT 13

3 Conclusions 13

References 15

A Derivation of 1-D DCT Algorithm 16

B Quantization Tables 18

C Huffman Tables 19

C.1 Luma DC Coefficients 19

C.2 Chroma DC Coefficients 19

C.3 Luma AC Coefficients 20

C.4 Chroma AC Coefficients 24
D Source Code

D.1 array_shrinker.v ................................................................. 28
D.2 array_sign_extender.v .......................................................... 28
D.3 clock_divider.v ................................................................. 29
D.4 coordinate_delay.v ............................................................ 30
D.5 crc.v .................................................................................. 31
D.6 data_spew.v ........................................................................ 32
D.7 dct_1d.v ................................................................................ 34
D.8 dct_2d.v ................................................................................ 36
D.9 debounce.v ........................................................................... 38
D.10 decoder.v ............................................................................ 39
D.11 delay.v ................................................................................ 40
D.12 dequantizer.v ...................................................................... 41
D.13 display_16hex.v ................................................................. 43
D.14 encoder.v ............................................................................ 47
D.15 entropy.v ............................................................................ 48
D.16 entropy_huffman.v .............................................................. 51
D.17 huffman.v ........................................................................... 53
D.18 huffman_categorizer.v .......................................................... 56
D.19 huffman_serializer.v ............................................................. 57
D.20 idct_1d.v ............................................................................. 59
D.21 idct_2d.v ............................................................................. 61
D.22 labkit.v ................................................................................. 63
D.23 level_shifter.v ...................................................................... 71
D.24 level_to_pulse.v ................................................................. 72
D.25 matrix_transpose.v .............................................................. 73
D.26 packer.v ............................................................................. 74
D.27 packer_fifo.v ...................................................................... 81
D.28 packer_sat.v ...................................................................... 83
D.29 packer_unwrapper.v ............................................................. 85
D.30 quantizer.v ........................................................................... 87
D.31 sign_extender.v ................................................................. 89
D.32 unentropy.v ......................................................................... 90
D.33 unentropy_huffman.v ............................................................. 93
D.34 unhuffman.v ........................................................................ 94
D.35 unpacker.v ........................................................................... 101
D.36 unpacker_fifo.v ................................................................. 105
D.37 unpacker_sar.v ................................................................. 107
D.38 unpacker_unwrapper.v .......................................................... 109
List of Figures

1  Block diagram for JPEG compression/decompression .................. 1
2  Data transfer in the Matrix Transpose module ..................... 4
3  Entropy coder zig-zag ordering [1] .................................. 5
4  Packet format ............................................................... 8
5  Timing diagram for start bit sampling [2] ............................ 10
6  Typical operation of the Huffman Decoder on a small block matrix .... 10
List of Tables

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1-D DCT algorithm [3]</td>
<td>..........................................................</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1-D iDCT algorithm [4]</td>
<td>..........................................................</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>Luminance quantization table [5]</td>
<td>..........................................................</td>
<td>18</td>
</tr>
<tr>
<td>4</td>
<td>Chrominance quantization table [5]</td>
<td>..........................................................</td>
<td>18</td>
</tr>
</tbody>
</table>
1 Overview

In this project, we developed a hardware JPEG block encoder and decoder for an FPGA with the goal of transmitting and replaying real-time video from a video camera.

The JPEG compression algorithm operates on $8 \times 8$ pixel blocks of data, which are treated as matrices for much of the process. First a 2-D discrete cosine transform is applied to the values. This organizes them by frequency data as opposed to spacial data. The matrix is then quantized, which lowers the accuracy with which certain frequency components are represented. Finally, the data is Entropy/Huffman encoded, which reduces the space needed to store long runs of zeros.

For this project, the resulting Huffman-encoded stream is then transmitted at a relatively low baud rate along a low-quality serial line using a customized packet format which includes verification that the contents of the packet were successfully transmitted. The receiving end then decodes the packet format, reverses the Entropy/Huffman-encoding, dequantizes the matrix, and performs an inverse DCT, resulting in a matrix that is similar to the original. Note that because JPEG is a lossy compression algorithm, the values may not be identical.

![Block diagram for JPEG compression/decompression](image)

Because this project’s goal was to interact with high frame-rate video, one of the key design principles used was that of graceful failure in the case when blocks are arriving too quickly. Several of the modules have non-constant latencies to process blocks, so it was important to ensure that faster modules could not overload the slower modules with too much data. The graceful failure mechanism throughout is that a module should finish processing old blocks before it accepts new ones. If this encoding/decoding system were used for processing video, the effect would essentially be a lowered frame-rate.

2 Modules

2.1 Encoding

2.1.1 2-D DCT (broder)

Like the 1-D DCT, the 2-D DCT transforms spacial information into the spacial frequency domain. However, instead of operating on a vector, the 2-D DCT operates on a matrix. The definition of the 2-D DCT is
\[ y_{kl} = \frac{c(k)c(l)}{4} \sum_{i=0}^{7} \sum_{j=0}^{7} x_{ij} \cos \left( \frac{(2i + 1)k\pi}{16} \right) \cos \left( \frac{(2j + 1)l\pi}{16} \right) \]

The 2-D DCT, however, is separable, which means that it can be broken up into 2 applications of the 1-D DCT. In practice, this means that if \( \tilde{y} = T\tilde{x} \) represents the 1-D DCT applied to the vector \( \tilde{x} \), then the 2-D DCT of a matrix \( X \) can be calculated as \( Y = TXT^T \).

This operation is equivalent to applying the 1-D DCT first to each of the rows of the input matrix, and then applying it to each of the columns resulting from that operation.

There are several additional steps that are required for valid output. First, input rows, which range from 0 to 255, must be adjusted to the range of \([-128, 127]\), which can be accomplished simply by inverting the most significant bit of each value.

Next, the width of each input value must be widened. Each result value from an unscaled 1-D DCT is 4 bits wider than the input. Therefore, the second 1-D DCT needed to take 12 bit inputs. To conserve area and multipliers, only a single 1-D DCT instance was used in each 2-D DCT, so it was necessary for it to take 12 bit inputs. Since the inputs to the module are each 8 bits wide, they are sign extended to 12 bits.

The input matrix is then passed to the 1-D DCT one row at a time, and the output is shifted into the Matrix Transpose module. Once the first 1-D DCT has completed, the data is read out one column at a time and inserted back into the 1-D DCT. The matrix is output in columns instead of rows to avoid the latency and area usage of another transposition matrix. The output from this second run is the output of the module, and it is accompanied by a valid_out signal which perfectly frames the output.

This module uses approximately 800 slices of logic and a total of 5 multipliers.

### 2.1.2 1-D DCT (broder)

The one-dimensional discrete cosine transform changes spatial information to spacial frequency information by decomposing the input vector into a sum of purely sinusoidal waves [6]. The DCT is expressed as

\[ y_k = \frac{c(k)}{2} \sum_{i=0}^{7} x_i \cos \left( \frac{(2i + 1)k\pi}{16} \right) , \]

where \( k \) is the spacial frequency, \( \tilde{x} \) is the input vector, \( \tilde{y} \) is the output vector, and

\[
c(k) = \begin{cases} 
\frac{1}{\sqrt{2}}, & \text{if } k = 0 \\
1, & \text{otherwise} 
\end{cases}
\]

The 1-D DCT module used for this project is a fairly straightforward implementation of the algorithm outlined in [3], reproduced as Table 1 (see Appendix A for a more detailed derivation of this particular algorithm). The values for the constants are

\[ m_1 = \cos(4\pi/16) \]
\[ m_3 = \cos(2\pi/16) - \cos(6\pi/16) \]
\[ m_2 = \cos(6\pi/16) \]
\[ m_4 = \cos(2\pi/16) + \cos(6\pi/16). \]

Table 1: 1-D DCT algorithm [3]

<table>
<thead>
<tr>
<th>Step 1</th>
<th>Step 2</th>
<th>Step 3</th>
<th>Step 4</th>
<th>Step 5</th>
<th>Step 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>( b_0 = a_0 + a_7 )</td>
<td>( c_0 = b_0 + b_5 )</td>
<td>( d_0 = c_0 + c_3 )</td>
<td>( e_0 = d_0 )</td>
<td>( f_0 = e_0 )</td>
<td>( S_0 = f_0 )</td>
</tr>
<tr>
<td>( b_1 = a_1 + a_6 )</td>
<td>( c_1 = b_1 - b_4 )</td>
<td>( d_1 = c_0 - c_3 )</td>
<td>( e_1 = d_1 )</td>
<td>( f_1 = e_1 )</td>
<td>( S_1 = f_4 + f_7 )</td>
</tr>
<tr>
<td>( b_2 = a_3 - a_4 )</td>
<td>( c_2 = b_2 + b_6 )</td>
<td>( d_2 = c_2 )</td>
<td>( e_2 = m_3 \times d_2 )</td>
<td>( f_2 = e_5 + e_6 )</td>
<td>( S_2 = f_2 )</td>
</tr>
<tr>
<td>( b_3 = a_1 - a_6 )</td>
<td>( c_3 = b_1 + b_4 )</td>
<td>( d_3 = c_1 + c_4 )</td>
<td>( e_3 = m_1 \times d_7 )</td>
<td>( f_3 = e_5 - e_6 )</td>
<td>( S_3 = f_5 - f_6 )</td>
</tr>
<tr>
<td>( b_4 = a_2 + a_5 )</td>
<td>( c_4 = b_0 - b_5 )</td>
<td>( d_4 = c_2 - c_5 )</td>
<td>( e_4 = m_4 \times d_6 )</td>
<td>( f_4 = e_3 + e_8 )</td>
<td>( S_4 = f_1 )</td>
</tr>
<tr>
<td>( b_5 = a_3 + a_4 )</td>
<td>( c_5 = b_3 + b_7 )</td>
<td>( d_5 = c_4 )</td>
<td>( e_5 = d_5 )</td>
<td>( f_5 = e_8 - e_3 )</td>
<td>( S_5 = f_5 + f_6 )</td>
</tr>
<tr>
<td>( b_6 = a_2 - a_5 )</td>
<td>( c_6 = b_3 + b_6 )</td>
<td>( d_6 = c_5 )</td>
<td>( e_6 = m_1 \times d_3 )</td>
<td>( f_6 = e_2 + e_7 )</td>
<td>( S_6 = f_3 )</td>
</tr>
<tr>
<td>( b_7 = a_0 - a_7 )</td>
<td>( c_7 = b_7 )</td>
<td>( d_7 = c_6 )</td>
<td>( e_7 = m_2 \times d_4 )</td>
<td>( f_7 = e_4 + e_7 )</td>
<td>( S_7 = f_4 - f_7 )</td>
</tr>
<tr>
<td>( d_8 = c_7 )</td>
<td>( e_8 = d_8 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Because Verilog does not allow arrays as ports, the inputs and outputs from the module are each single concatenated vectors. For this project, each of the 8 values in a row or column is 12 bits long, so the input vector is 96 bits long. Since the module increases the width of each value by 4 bits, the output vector is 128 bits long.

The algorithm from [3] is a 6-stage pipeline, so the latency is 6 clock cycles. The implementation uses 5 multipliers (one for each multiplication operation in step 4) and approximately 425 slices.

2.1.3 Matrix Transpose (broder)

The Matrix Transpose module is used in the 2-D DCT and 2-D iDCT between the first and second times that data is run through the 1-D DCT (or 1-D iDCT). It takes for input a row array, which is concatenated into a single vector, and outputs a column array, also a single vector of the same length. There are additionally two control signals, shift_row and shift_column.

In order to transpose the matrix from a series of rows to a series of columns, it is necessary to store a representation of the entire matrix internally, which the module does in register memory. When shift_row is asserted, the module shifts each row upwards and sets the bottom row to the input. When shift_column is asserted, each column is shifted to the left. The output of the module is set to be the left column of the internal representation. Figure 2 shows the direction that the data moves depending on which control signal is asserted.

Because the matrices that must be transformed for the 2-D DCT and 2-D iDCT are of different widths, the width of each value in the matrix is parametrized.

When instantiated with a width of 8 bits per value, this module uses approximately 300 slices. When instantiated with a width of 12 bits per value, the module uses approximately 450 slices.
2.1.4 Coordinate Delay (broder)

This module was created to separate out a common pattern from the encoding and decoding modules. All of the encoding and decoding modules keep track of the coordinates for the block they are currently working on and pass that information along to the next module in the encoding or decoding pipeline. They are expected to latch the incoming coordinates on the positive edge of valid\_in and therefore setup the outgoing coordinates on the clock cycle when valid\_out is asserted.

A level-to-pulse converter is used for both the input and output sides of the modules. On the input side, the incoming values are simply latched on the next clock cycle. For the output, since the next clock cycle would be too late, a “recursive mux” structure is used.

This small module requires less than 10 slices to synthesize.

2.1.5 Quantizer (broder)

The quantization stage performs an element-wise division on the output of the DCT. The quantization values used for each element of a block can vary from image to image, but for this project, the standard quantization matrix recommended by [5] was used (reproduced in Appendix B).

The quantization tables also factored in the scaling factors necessitated by the scaled DCT algorithm used (see Appendix A).

The module uses a single Xilinx IP Core full divider. As columns come in, they are stored into a memory array. The module then inserts one value at a time along with the corresponding quantization factor (which is loaded from a BRAM).

Because the quantization factors are frequently very high relative to the range of input values, it is important to round, not truncate, the results of the division step, so the IP
Core divider was configured to output a fractional response. The most significant bit of this fractional component is added to the result of the division to round the output appropriately.

The latency of the module is 32 clock cycles from the first input to the first output, and it uses approximately 1200 slices. The large area is due to a combination of the register memory buffer of the full matrix and the use of a full divider module. Additionally, it uses one BRAM module to store the quantization table.

### 2.1.6 Entropy Coder (ccpost)

The matrix of values received from the Quantizer contains mostly zeros in the lower-right corner of the block matrix, and the majority of the nonzero values are concentrated in the upper-left corner. Thus, the matrix values are entropy coded into a linear stream in a zig-zag ordering pattern as shown in subsubsection 2.1.6. This ensures that the majority of the zero values in the block matrix are placed at the end of the entropy-coded stream, so that the Huffman Coder can optimally code the stream by replacing all trailing zeros with a special end of block (EOB) character.

In order to accomplish the entropy coding, the Entropy Coder must take in 12-bit values one at a time from the quantizer, starting with the upper-left corner, going top to bottom in each column, and left to right across the columns. These values are stored in register memory to avoid latency during the entropy coding. Once the entire block matrix has been stored, the Entropy Coder asserts the `ent_rdy` signal so that the Huffman Coder can read values in entropy order from the Entropy Coder. The `ent_read_en` signal from the Huffman Coder controls the reading of these values so that while the Huffman Coder is actively busy sending
Huffman codes, it can pause reading values in entropy order from the Entropy Coder.

When reading the values in entropy order from the block matrix, the Entropy Coder uses hard-coded logic to determine the coordinates of the next value to read from the block matrix based on the current coordinates. While this implementation can be inefficient in terms of area, it allows the Huffman Coder to pause and resume reading values in entropy order easily, without any latency that would be introduced using other types of memory.

The previous modules in the encoding pipe (the 2-D DCT and the Quantizer) operate with a fixed processing time. The Huffman Coder and Huffman Serializer do not. Thus, the Entropy Coder will refuse new data unless the Huffman Coder and Huffman Serializer modules are done processing a block matrix. This ensures that in a condition where new data comes down the encoding pipe too quickly for the Huffman Coder and Huffman Serializer modules to process it, some blocks will make it through the pipe, while others will be dropped. The end effect of this kind of failure will be a drop in the rate at which blocks are updated, so it is impossible for a block that takes a proportionally long time to encode to cause the entire encoding pipe to fail.

The Entropy Coder module uses approximately 675 slices.

2.1.7 Huffman Coder (ccpost)

In the JPEG Huffman coding scheme, each value in the entropy coded stream is transmitted along with a Huffman code that represents the size of the value to be encoded, and the length of the run of zeros that precedes the value. The first value in the stream (the DC value from the DCT) is always coded explicitly with it’s own code. In a normal JPEG implementation, the Huffman coding tables can be computed to be ideal for that particular image and transmitted along with the image to the decoder. In a hardware implementation, however, this would be very unwieldy, so this implementation uses the JPEG standard’s preset typical Huffman coding tables from [5], which are reproduced in Appendix C.

First, the incoming values are converted to one’s complement binary notation so that they are symmetrical around zero. This is accomplished by subtracting one from the value if it is negative. Then, the values are sent through a categorizing unit which determines the bit length of the values. If the value is negative, all the leading ones will be dropped from the value before transmission; if the value is positive, all the leading zeros will be dropped from the value before transmission.

Then, the size of the incoming value is determined. Originally, the determination of the size of the incoming value was done completely combinatorially, which introduced a very long critical path in the logic. The determination of size inherently has a very long critical path, since it uses priority logic to determine the size of the incoming value. Thus, the final implementation is pipelined in order to avoid having a long critical path in a single clock cycle.

The Huffman Coder uses this size to lookup the appropriate Huffman code in a BRAM for the DC value. For the subsequent AC values, the Huffman Coder uses internal counters that keep track of the number of zeros encountered in the entropy steam. When it encounters a nonzero value, it then uses this count of preceding zeros and the size of the nonzero value
to lookup the appropriate Huffman code in the BRAM from the AC code section. If it encounters a run of 16 zeros, it uses a zero run length (ZRL) code to indicate this condition.

Once the remaining values in the entropy stream are all zero, however, there is a special end of block (EOB) code that is used to signify this condition. Thus, no codes for ZRL sections are output unless the ZRL codes come before a nonzero value in the entropy stream. This is accomplished using another counter to keep track of how many ZRL sections have been seen since the last nonzero value in the entropy stream. The Huffman Coder will output the appropriate number of ZRL codes before outputting the code for the nonzero value. This is one reason that the Huffman Coder must be able to control reading the values in entropy order from the Entropy Coder.

Also, the Huffman Coder will not process values from the entropy coded stream unless the `serial_rdy` signal from the Huffman Serializer is asserted, since the Huffman Serializer needs multiple clock cycles to output the coded bitstream serially to the Packer. This is another reason why the Huffman Coder must be able to pause reading the entropy coded stream from the Entropy Coder.

The Huffman Coder uses approximately 75 slices and one 20x272 BRAM.

### 2.1.8 Huffman Serializer (ccpost)

The Packer needs to take Huffman codes and values in a serial bitsream format, while the Huffman Coder outputs a Huffman code, Huffman code size, the associated value, and its size simultaneously. Thus, the Huffman Serializer takes the Huffman code, code size, value, and value size as inputs, and outputs a serial stream for the Packer.

In order to accomplish this, the Huffman Serializer stores the input values in register memory, then steps through all the bits in the correct order, using the size values to ensure that only the correct number of bits is placed into the serial stream. For each Huffman code in the stream, except for an EOB code or where the value size is zero, the value will follow the Huffman code serially.

To ensure that the Huffman Coder does not send values to the Huffman Serializer when it is currently in the process of sending a code serially, it deasserts the `serial_rdy` and does not reassert it until it is done sending the code, preventing data collisions.

The Huffman Serializer uses approximately 25 slices.

### 2.2 Transmission/Reception

#### 2.2.1 Packer (broder)

The Packer is responsible for processing the output of all of the separate encoding pipelines. Each pipeline is connected to a BRAM which acts as a FIFO. Each FIFO keeps track of when it has been filled, and refuses to accept new data until the old data has been read out. This is accomplished by having both a write enable input and a stream enable input. While the write enable is only asserted when the incoming data is valid, the stream enable input frames an entire block’s worth of data. When the stream enable has been deasserted and
the Packer has not yet read the data in the FIFO, then subsequently asserting the stream enable is ignored by the FIFO.

The Packer looks at each FIFO in turn. If it contains a full block of data, then it connects the relevant ports to the Packet Wrapper by way of a large, bi-directional mux.

This module uses approximately 475 logic slices

### 2.2.2 Packet Wrapper (broder)

The Packet Wrapper receives from the Packer several pieces of information about a packet of information. Using that information, it sends a packet over the serial transmission line according to the protocol created for this project.

The Packet Wrapper takes as inputs the length of a memory buffer, the x- and y-coordinates of the block, the channel of the block, and an interface for accessing the contents of the memory buffer. It then outputs a packet with the format outlined in Figure 4 by passing each byte to the Serial Asynchronous Transmitter in turn (note that audio packets do not include the coordinate information). After the actual data has been transmitted, the Packet Wrapper computes a CRC-8 checksum of just the data component and appends that to the end.

![Figure 4: Packet format](image)

This module uses approximately 75 slices.

### 2.2.3 Serial Asynchronous Transmitter (broder)

The Serial Asynchronous Transmitter is responsible for taking individual bytes from the Packet Wrapper and transmitting them across the serial line.

The spacing of each bit is timed by an instance of the Clock Divider module, which asserts an `enable` signal approximately 250,000 times per second. This signal is used to enable the other logic in the module, which first outputs a single space (0) bit, then outputs the byte, least-significant byte first. Finally, it requires there to be at least two bits-worth of mark before allowing for another byte to be sent.

The module uses a few more than 25 slices.

### 2.2.4 Unpacker (broder)

The Unpacker is responsible for receiving packets and determining which of the decoding pipelines should receive the data. When the Packet Unwrapper asserts that it has a packet coming in, the Unpacker uses the channel information to connect the output of the Packet Unwrapper to the appropriate channel. The Unpacker balances between the four luma
decoding pipelines by outputting incoming luma packets to each one in series. Finally, if the Packet Unwrapper determines that a packet is malformed (i.e. the CRC does not match the one calculated by the Packet Unwrapper), it can assert a clearing signal which clears the contents of the FIFO.

The FIFOs for the Unpacker, like the FIFOs for the Packer refuse to accept new input until the old input has been read out.

This module uses approximately 425 slices and 7 BRAMs.

### 2.2.5 Packet Unwrapper (broder)

The Packet Unwrapper is responsible for decoding the packet format. It idles until it receives the start byte (0xFF), and then goes through a series of states extracting the relevant metadata included with each packet and latching the data to a series of ports that the ?? can pass to the decoders.

As the actual data component is being processed, the Packet Unwrapper computes the CRC of the data. If the CRC transmitted with the packet does not match the one computed by the Packet Unwrapper, than the module asserts a signal to clear the memory.

The Packet Unwrapper uses approximately 125 slices.

### 2.2.6 Serial Asynchronous Receiver (broder)

The Serial Asynchronous Receiver (SAR) is responsible for taking the data from the serial transmission line and decoding it into bytes, which are then processed by the Packet Unwrapper.

The input is first debounced to try and remove any glitches. The debouncer is taken from [7]. However, while the Lab 3 debouncer was removing mechanical glitches on the order of milliseconds, the purpose here is to remove transmission faults on the order of microseconds, so the period for which a signal must remain constant has been reduced to 5 clock cycles.

The actual frame syncing decoding algorithm is derived from [2, pp. 128-130]. This algorithm attempts to sample the middle of each bit in a frame, using a majority voting mechanism.

The SAR remains in the idle state until it detects a negative edge in the serial line. At that point, it reads the 7th, 8th, and 9th samples (marked in Figure 5). If the majority of those samples is a space (0), the module considers itself to be synced with the frame. For each subsequent bit, it uses the majority vote of those same 7th, 8th, and 9th samples to determine whether the bit is a mark or a space, setting the output for each bit in turn.

Once the module has finished reading all of the bits in a frame, it asserts the rx signal for one clock cycle to inform the Packet Unwrapper that the output is valid.

This module uses approximately 50 slices.
2.3 Decoding

2.3.1 Huffman Decoder (ccpost)

After the Unpacker removes the packet information and checks the CRC, it sends the serial stream encoded by the ?? to the Huffman Decoder. The Huffman Decoder constantly shifts this data into a buffer large enough to hold the longest code plus the longest possible value at any given time (and a few clock cycles after), allowing the longest possible code/value combination to be read directly from the buffer once it is recognized as a valid code. There is also a no_out flag, controlled by the no_out_count variable, that disables code output if the data in the code recognition position in the buffer does not currently contain valid data. These variables are set when a new serial stream begins to shift in, and also when a valid code is recognized.

As valid data is shifted in, it is also accompanied by a corresponding buffer that contains 1s in corresponding positions where valid data exists the input buffer. This allows the Huffman Decoder to determine if data at any given point in the buffer is valid. Also, the Huffman Decoder will stop processing the incoming data stream when the entire mask buffer is 0s, indicating that there is no valid data left to process. When a valid code is recognized, the buffer and mask buffer are cleared where the code and its corresponding value were so that there are no collisions with the recognition of new codes. Since the code recognition section of the buffer is not at the beginning of the buffer, there is a latency from when the first bit of serial data is shifted in and when the first code is recognized, as shown in subsubsection 2.3.1. This is not a problem, however, since the serial stream does not need to be paused because the buffer can be read at any point along the buffer.

Looking up the Huffman codes for DC values is hard coded directly, since there are only
There are 160 possible AC Huffman codes, however, making a arcaded lookup severely inefficient. Looking these codes up directly in BRAM using the code as an address would also be extremely inefficient, since the maximum code size is 16 bits, meaning an exhaustive lookup table would use 65,536 memory locations, where only 160 locations would contain valid codes.

In this implementation, however, the codes are divided into three lookup bins based on their size. This is a variation of the method presented in [8]. The Huffman codes are placed into groups according to size. Codes of 8 bits or less are placed in group A, codes 9 to 12 bits long are placed in group B, and codes of 13 bits or longer are placed in group C. The JPEG Huffman coding tables are designed in such a way that codes of 9 bits or longer always have at least 5 leading 1s, and codes of 13 bits or longer always have at least 9 leading ones. Thus, the remaining bits are used as address values for lookup of these codes. Group A uses 8-bit addressing, while groups B and C use 7-bit addressing. Consequently, only 512 memory locations are used in this implementation.

In order to prevent code collisions, the size of the code is stored in memory along with the decoded values. This allows the code data to only be output when the predicted size of the code based on the mask buffer matches the size returned from memory.

Inherently, there is a 2 clock cycle latency in the code recognition process because the memory address must be calculated based on the group and remaining bits, then the data must be returned from memory. This latency does not cause any overall throughput issues, however, because the buffer continues shifting the serial data to the left on every cycle, such that the code and its associated value can easily be cleared from the buffer after it has been recognized.

Once a code is recognized, the run length of zeros, the size of the value, and the value are output to the Entropy Decoder.

The Huffman Decoder uses approximately 175 slices and one 12x512 BRAM.

2.3.2 Entropy Decoder (ccpost)

The Entropy Decoder reverses the entropy ordering performed by the Entropy Coder. In the Entropy Coder, the zigzag ordering is performed via hard coded logic. In this module, however, performing the reverse zigzag ordering would not be straightforward in hard coded logic. It would also be expensive in area. Thus, the entropy coded stream is treated as a vector of values, and the output stream to the Dequantizer is treated as a vector. Thus two vectors are associated by a lookup table programmed into a BRAM, where the address is the order in the stream output to the Dequantizer and the value in the memory is the order in the entropy coded stream.

When codes are received from the Huffman Decoder, the value is stored in a BRAM with an address corresponding to its order in the entropy coded stream. This address is calculated using the order of the last value stored and the run of preceding zeros. Since it would waste valuable clock cycles to reset every location to 0 that does not have a value, there is a 64-bit mask vector used to mark when a location in the vector contains a valid value. This mask vector is reset at the beginning of processing every new block.
When all the codes from the Huffman Decoder have been read into the BRAM, the Entropy Decoder begins reading them out to the Dequantizer in normal (non-entropy) order. To do this, a two-port BRAM is used. The Entropy Decoder looks up the non-entropy order from the BRAM, and uses the value to look up the value from the entropy coded vector on the other port of the BRAM. This produces some latency, but with a two-port BRAM, the values can be read out with a throughput of 1 value per clock cycle. In order to correct for locations that should have a value of 0, a value of 0 is output when the mask vector indicates that there was no valid data written to the corresponding memory location.

This implementation could also be used in the Entropy Coder in future cases. In addition, it allows for any permutation of two vectors of the same size, since the permutation can easily be changed by changing the contents of the BRAM.

The Entropy Decoder uses approximately 150 slices and 1 two-port 12x128 BRAM.

### 2.3.3 Dequantizer (broder)

The Dequantizer reverses the quantization stage of the compression by multiplying each element of the output from the Entropy Decoder by a fixed dequantization factor.

Additionally, the dequantization table accounted for the scaling factors needed to accurately perform the iDCT.

As it receives values from the Entropy Decoder, the Dequantizer passes each incoming value to a Xilinx IP Core multiplier (used for symmetry with the Quantizer) along with the corresponding dequantization factor (loaded from a BRAM). The output is then stored in the appropriate location in a full matrix-sized register memory buffer. Once this is completed, the buffer is read out one row at a time, framed by the valid\_out signal.

A single instantiation of this module uses approximately 700 slices, one BRAM, and one multiplier.

### 2.3.4 1-D iDCT (broder)

The iDCT, or inverse discrete cosine transform, reverses the operation of the 1-D DCT. The calculation is very similar to the 1-D DCT, and can be expressed as

\[
y_k = \sum_{i=0}^{7} x_i c(k) \cos \left( \frac{(2k + 1)i\pi}{16} \right),
\]

where, as before, \(k\) is the spacial frequency, \(\vec{x}\) is the input vector, \(\vec{y}\) is the output vector, and

\[
c(k) = \begin{cases} 
\frac{1}{\sqrt{2}}, & \text{if } k = 0 \\
1, & \text{otherwise} 
\end{cases}
\]

Because some of the steps from the forward DCT do not reverse cleanly, three of the pipeline calculations involve three instead of two operands. However, the increased logic
does not seem to severely impact the propagation speed of the circuit. The algorithm for the iDCT is listed in Table 2.

<table>
<thead>
<tr>
<th>Step 1</th>
<th>Step 2</th>
<th>Step 3</th>
<th>Step 4</th>
<th>Step 5</th>
<th>Step 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b_0 = a_0$</td>
<td>$c_0 = b_0$</td>
<td>$d_0 = c_0$</td>
<td>$e_0 = d_0 + d_1$</td>
<td>$f_0 = e_0 + e_3$</td>
<td>$S_0 = f_0 + f_7$</td>
</tr>
<tr>
<td>$b_1 = a_5$</td>
<td>$c_1 = b_1$</td>
<td>$d_1 = c_1$</td>
<td>$e_1 = d_0 - d_1$</td>
<td>$f_1 = e_1 + e_2$</td>
<td>$S_1 = f_1 + f_6$</td>
</tr>
<tr>
<td>$b_2 = a_2$</td>
<td>$c_2 = b_2 - b_3$</td>
<td>$d_2 = n_1 \times c_2$</td>
<td>$e_2 = d_2 - d_3$</td>
<td>$f_2 = e_1 - e_2$</td>
<td>$S_2 = f_2 + f_5$</td>
</tr>
<tr>
<td>$b_3 = a_6$</td>
<td>$c_3 = b_2 + b_3$</td>
<td>$d_3 = c_3$</td>
<td>$e_3 = d_3$</td>
<td>$f_3 = e_0 - e_3$</td>
<td>$S_3 = f_3 - f_4 - f_5$</td>
</tr>
<tr>
<td>$b_4 = a_5 - a_3$</td>
<td>$c_4 = b_4$</td>
<td>$d_4 = n_2 \times c_4$</td>
<td>$e_4 = d_8 - d_4$</td>
<td>$f_4 = e_4$</td>
<td>$S_4 = f_3 + f_4 + f_5$</td>
</tr>
<tr>
<td>$b_5 = a_1 + a_7$</td>
<td>$c_5 = b_5 - b_7$</td>
<td>$d_5 = n_1 \times c_5$</td>
<td>$e_5 = d_5$</td>
<td>$f_5 = e_5 - e_6 + e_7$</td>
<td>$S_5 = f_2 - f_5$</td>
</tr>
<tr>
<td>$b_6 = a_1 - a_7$</td>
<td>$c_6 = b_6$</td>
<td>$d_6 = c_3 \times c_6$</td>
<td>$e_6 = d_6 - d_8$</td>
<td>$f_6 = e_6 - e_7$</td>
<td>$S_6 = f_1 - f_6$</td>
</tr>
<tr>
<td>$b_7 = a_3 + a_5$</td>
<td>$c_7 = b_5 + b_7$</td>
<td>$d_7 = c_7$</td>
<td>$e_7 = d_7$</td>
<td>$f_7 = e_7$</td>
<td>$S_7 = f_0 - f_7$</td>
</tr>
<tr>
<td>$c_8 = b_4 - b_6$</td>
<td>$d_8 = n_4 \times c_8$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The implementation of the 1-D iDCT used in this project uses 5 multipliers and approximately 250 slices.

2.3.5 2-D iDCT (broder)

This module is the decoding counterpart of the 2-D DCT. It receives data in columns and passes that into the 1-D iDCT. The output is then stored in an instance of the Matrix Transpose module, which outputs rows. These rows are multiplexed into the 1-D iDCT again, and this forms the output of the module.

The 2-D iDCT can be explicitly expressed by

$$y_{kl} = \frac{c(k)c(l)}{4} \sum_{i=0}^{7} \sum_{j=0}^{7} x_{ij} \cos \left( \frac{(2k + 1)i\pi}{16} \right) \cos \left( \frac{(2l + 1)j\pi}{16} \right)$$

This module uses approximately 525 logic slices and 5 multipliers.

3 Conclusions

Originally, this project was intended to receive video from the NTSC decoder on the 6.111 labkit, encode the image data using the JPEG compression modules, then transmit it along a serial line. On the receiving end, the intention was to receive this data serially, process it using the JPEG decoding modules, and display it on the VGA output.

The video capture and display components of this project, however, were not completed by the project deadline. The complexity involved in synchronizing different clock domains for NTSC decoding, image processing, and VGA display was greatly underestimated. Also, using two separate frame buffers for capture and playback of image data was much more
challenging than originally predicted. Consequently, the video capture and display aspects of this project were dropped from the final product.

During implementation, many problems were encountered when modules were integrated together, especially when the modules had different designers. Thus, it is very important to make sure well-defined contracts exist between modules so that if any module outputs unexpected data, subsequent modules will not interpret this data in an incorrect manner.

Despite these problems, the final product does successfully encode image data using a JPEG compression algorithm and transmit it along a low-bandwidth serial line. It also receives the packets from the serial line, check for errors, and decompresses the received data in order to output a resulting image. When tested on the labkit, the project successfully produces image data that is suitably similar to the input data given the level of compression used. Also, two labkits that were connected together using a long, noisy wire were able to successfully transmit and receive image data while correcting for errors.
References


A Derivation of 1-D DCT Algorithm

Because each element of the result of the one-dimensional DCT can be expressed as a linear combination of the inputs, the one-dimensional DCT can be expressed as a linear transformation \( T : \mathbb{R}^8 \rightarrow \mathbb{R}^8 \), with an approximate value of

\[
T = \begin{bmatrix}
0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 \\
0.4904 & 0.4157 & 0.2778 & 0.0975 & -0.0975 & -0.2778 & -0.4157 & -0.4904 \\
0.4619 & 0.1913 & -0.1913 & -0.4619 & -0.4619 & -0.1913 & 0.1913 & 0.4619 \\
0.4157 & -0.0975 & -0.4904 & -0.2778 & 0.2778 & 0.4904 & 0.0975 & -0.4157 \\
0.3536 & -0.3536 & 0.3536 & 0.3536 & -0.3536 & -0.3536 & 0.3536 & 0.3536 \\
0.2778 & -0.4904 & 0.0975 & 0.4157 & -0.4157 & -0.0975 & 0.4904 & -0.2778 \\
0.1913 & -0.4619 & 0.4619 & -0.1913 & -0.1913 & 0.4619 & -0.4619 & 0.1913 \\
0.0975 & -0.2778 & 0.4157 & -0.4904 & 0.4904 & -0.4157 & 0.2778 & -0.0975 \\
\end{bmatrix}
\]

Directly computing the DCT of a vector \( \vec{x} \) simply requires computing the matrix multiply \( T \vec{x} \). This naïve method, however, requires 64 multiply operations and 64 addition operations. In order to reduce the number of operations required, most algorithms factor the matrix \( T \) into a series of matrices \( T_1 \) through \( T_k \) such that \( T_k \ldots T_2 T_1 = T \) [6]. Typically, matrices are found such that most of the elements are 0, and the rest are either 1 or \(-1\). For example, in the algorithm used in this project, the first transformation step can be expressed by

\[
T_1 = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\
\end{bmatrix}
\]

Evaluating \( T_1 \vec{x} \) only requires 4 additions and 4 subtractions; the steps for calculating \( \vec{b} = T_1 \vec{x} \) can be expressed as

\[
\begin{align*}
b_0 &= x_0 + x_7; \\
b_1 &= x_1 + x_6; \\
b_2 &= x_3 - x_4; \\
b_3 &= x_1 - x_6; \\
b_4 &= x_2 + x_5; \\
b_5 &= x_3 + x_4; \\
b_6 &= x_2 - x_5; \\
b_7 &= x_0 - x_7;
\end{align*}
\]

Finally, to further reduce the number of required multiplications, the last matrix \( T_k \) is found such that it can be expressed as \( H \hat{T}_k \), where \( H \) is a diagonal matrix and \( \hat{T}_k \) is in a form similar to the other \( T_i \) matrices. Let the quantization of \( Y \) with elements \( y_{ij} \) yield \( Z \) with elements

\[
z_{ij} = \frac{y_{ij}}{q_{ij}},
\]

16
where $q_{ij}$ is the quantization factor. The matrix $H$ can be applied in the quantization stage instead of the DCT stage by using the altered quantization factors

$$
\hat{q}_{ij} = \frac{q_{ij}}{h_{ii}h_{jj}}.
$$

[9] proposed and [3] corrected the algorithm used in this project. The steps to compute the algorithm were listed in Table 1, and the quantization scale factors are

$$
\begin{bmatrix}
0.176776695 \\
0.127448894 \\
0.135299025 \\
0.150336221 \\
0.176776695 \\
0.224994055 \\
0.326640741 \\
0.640728861
\end{bmatrix}.
$$
B Quantization Tables

The following tables were determined based on psychovisual thresholding and are included in [5]. They are the default tables for this project.

Table 3: Luminance quantization table [5]

```
[16 11 10 16 24 40 51 61]
[12 12 14 19 26 58 60 55]
[14 13 16 24 40 57 69 56]
[14 17 22 29 51 87 80 62]
[18 22 37 56 68 109 103 77]
[24 35 55 64 81 104 113 92]
[49 64 78 87 103 121 120 77]
[72 92 95 98 112 100 103 99]
```

Table 4: Chrominance quantization table [5]

```
[17 18 24 47 99 99 99 99]
[18 21 26 66 99 99 99 99]
[24 26 56 99 99 99 99 99]
[47 66 99 99 99 99 99 99]
[99 99 99 99 99 99 99 99]
[99 99 99 99 99 99 99 99]
[99 99 99 99 99 99 99 99]
[99 99 99 99 99 99 99 99]
```
# Huffman Tables

## C.1 Luma DC Coefficients

<table>
<thead>
<tr>
<th>Category (Length)</th>
<th>Code size</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>00</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>010</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>011</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>100</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>101</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>110</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>1110</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>11110</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>111110</td>
</tr>
<tr>
<td>9</td>
<td>7</td>
<td>1111110</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
<td>11111110</td>
</tr>
<tr>
<td>11</td>
<td>9</td>
<td>11111110</td>
</tr>
</tbody>
</table>

## C.2 Chroma DC Coefficients

<table>
<thead>
<tr>
<th>Category (Length)</th>
<th>Code size</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>00</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>01</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>110</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>1110</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>11110</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>111110</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>1111110</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>11111110</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>111111110</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>1111111110</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>1111111110</td>
</tr>
</tbody>
</table>
### C.3 Luma AC Coefficients

<table>
<thead>
<tr>
<th>Run/Length</th>
<th>Code size</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0/0 (EOB)</td>
<td>3</td>
<td>1010</td>
</tr>
<tr>
<td>0/1</td>
<td>1</td>
<td>00</td>
</tr>
<tr>
<td>0/2</td>
<td>1</td>
<td>01</td>
</tr>
<tr>
<td>0/3</td>
<td>2</td>
<td>100</td>
</tr>
<tr>
<td>0/4</td>
<td>3</td>
<td>1011</td>
</tr>
<tr>
<td>0/5</td>
<td>4</td>
<td>11010</td>
</tr>
<tr>
<td>0/6</td>
<td>6</td>
<td>1111000</td>
</tr>
<tr>
<td>0/7</td>
<td>7</td>
<td>11111000</td>
</tr>
<tr>
<td>0/8</td>
<td>9</td>
<td>1111110110</td>
</tr>
<tr>
<td>0/9</td>
<td>15</td>
<td>1111111110000010</td>
</tr>
<tr>
<td>0/A</td>
<td>15</td>
<td>1111111110000011</td>
</tr>
<tr>
<td>1/1</td>
<td>3</td>
<td>1100</td>
</tr>
<tr>
<td>1/2</td>
<td>4</td>
<td>11011</td>
</tr>
<tr>
<td>1/3</td>
<td>6</td>
<td>1111001</td>
</tr>
<tr>
<td>1/4</td>
<td>8</td>
<td>111110110</td>
</tr>
<tr>
<td>1/5</td>
<td>10</td>
<td>11111110110</td>
</tr>
<tr>
<td>1/6</td>
<td>15</td>
<td>1111111110000100</td>
</tr>
<tr>
<td>1/7</td>
<td>15</td>
<td>1111111110000101</td>
</tr>
<tr>
<td>1/8</td>
<td>15</td>
<td>1111111110000110</td>
</tr>
<tr>
<td>1/9</td>
<td>15</td>
<td>1111111110000111</td>
</tr>
<tr>
<td>1/A</td>
<td>15</td>
<td>1111111110001000</td>
</tr>
<tr>
<td>2/1</td>
<td>4</td>
<td>1111100</td>
</tr>
<tr>
<td>2/2</td>
<td>7</td>
<td>11111001</td>
</tr>
<tr>
<td>2/3</td>
<td>9</td>
<td>1111110111</td>
</tr>
<tr>
<td>2/4</td>
<td>11</td>
<td>111111110100</td>
</tr>
<tr>
<td>2/5</td>
<td>15</td>
<td>1111111110001001</td>
</tr>
<tr>
<td>2/6</td>
<td>15</td>
<td>1111111110001010</td>
</tr>
<tr>
<td>2/7</td>
<td>15</td>
<td>1111111110001011</td>
</tr>
<tr>
<td>2/8</td>
<td>15</td>
<td>1111111110001100</td>
</tr>
<tr>
<td>2/9</td>
<td>15</td>
<td>1111111110001101</td>
</tr>
<tr>
<td>2/A</td>
<td>15</td>
<td>1111111110001110</td>
</tr>
<tr>
<td>3/1</td>
<td>5</td>
<td>111010</td>
</tr>
<tr>
<td>3/2</td>
<td>8</td>
<td>111110111</td>
</tr>
<tr>
<td>3/3</td>
<td>11</td>
<td>111111110101</td>
</tr>
<tr>
<td>3/4</td>
<td>15</td>
<td>1111111110001111</td>
</tr>
<tr>
<td>3/5</td>
<td>15</td>
<td>1111111110010000</td>
</tr>
<tr>
<td>3/6</td>
<td>15</td>
<td>1111111110010001</td>
</tr>
<tr>
<td>3/7</td>
<td>15</td>
<td>1111111110010010</td>
</tr>
<tr>
<td>3/8</td>
<td>15</td>
<td>1111111110010011</td>
</tr>
</tbody>
</table>

20
<table>
<thead>
<tr>
<th>Run/Length</th>
<th>Code size</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>3/9</td>
<td>15</td>
<td>1111111110010100</td>
</tr>
<tr>
<td>3/A</td>
<td>15</td>
<td>1111111110010101</td>
</tr>
<tr>
<td>4/1</td>
<td>5</td>
<td>111011</td>
</tr>
<tr>
<td>4/2</td>
<td>9</td>
<td>1111111000</td>
</tr>
<tr>
<td>4/3</td>
<td>15</td>
<td>1111111110010110</td>
</tr>
<tr>
<td>4/4</td>
<td>15</td>
<td>1111111110010111</td>
</tr>
<tr>
<td>4/5</td>
<td>15</td>
<td>1111111110011000</td>
</tr>
<tr>
<td>4/6</td>
<td>15</td>
<td>1111111110011001</td>
</tr>
<tr>
<td>4/7</td>
<td>15</td>
<td>1111111110011010</td>
</tr>
<tr>
<td>4/8</td>
<td>15</td>
<td>1111111110011011</td>
</tr>
<tr>
<td>4/9</td>
<td>15</td>
<td>1111111110011100</td>
</tr>
<tr>
<td>4/A</td>
<td>15</td>
<td>1111111110011101</td>
</tr>
<tr>
<td>5/1</td>
<td>6</td>
<td>1111010</td>
</tr>
<tr>
<td>5/2</td>
<td>10</td>
<td>11111110111</td>
</tr>
<tr>
<td>5/3</td>
<td>15</td>
<td>1111111110011110</td>
</tr>
<tr>
<td>5/4</td>
<td>15</td>
<td>1111111110011111</td>
</tr>
<tr>
<td>5/5</td>
<td>15</td>
<td>1111111111010000</td>
</tr>
<tr>
<td>5/6</td>
<td>15</td>
<td>1111111111010001</td>
</tr>
<tr>
<td>5/7</td>
<td>15</td>
<td>1111111111010010</td>
</tr>
<tr>
<td>5/8</td>
<td>15</td>
<td>1111111111010011</td>
</tr>
<tr>
<td>5/9</td>
<td>15</td>
<td>1111111111010010</td>
</tr>
<tr>
<td>5/A</td>
<td>15</td>
<td>1111111111010011</td>
</tr>
<tr>
<td>6/1</td>
<td>6</td>
<td>1111011</td>
</tr>
<tr>
<td>6/2</td>
<td>11</td>
<td>111111110110</td>
</tr>
<tr>
<td>6/3</td>
<td>15</td>
<td>1111111110100110</td>
</tr>
<tr>
<td>6/4</td>
<td>15</td>
<td>1111111110100111</td>
</tr>
<tr>
<td>6/5</td>
<td>15</td>
<td>1111111111010010</td>
</tr>
<tr>
<td>6/6</td>
<td>15</td>
<td>1111111111010101</td>
</tr>
<tr>
<td>6/7</td>
<td>15</td>
<td>1111111111010101</td>
</tr>
<tr>
<td>6/8</td>
<td>15</td>
<td>1111111111010111</td>
</tr>
<tr>
<td>6/9</td>
<td>15</td>
<td>1111111111010110</td>
</tr>
<tr>
<td>6/A</td>
<td>15</td>
<td>1111111111010111</td>
</tr>
<tr>
<td>7/1</td>
<td>7</td>
<td>11111010</td>
</tr>
<tr>
<td>7/2</td>
<td>11</td>
<td>111111110111</td>
</tr>
<tr>
<td>7/3</td>
<td>15</td>
<td>1111111110101110</td>
</tr>
<tr>
<td>7/4</td>
<td>15</td>
<td>1111111111010111</td>
</tr>
<tr>
<td>7/5</td>
<td>15</td>
<td>1111111111011000</td>
</tr>
<tr>
<td>7/6</td>
<td>15</td>
<td>11111111110110001</td>
</tr>
<tr>
<td>7/7</td>
<td>15</td>
<td>1111111111011001</td>
</tr>
<tr>
<td>7/8</td>
<td>15</td>
<td>1111111111011011</td>
</tr>
<tr>
<td>7/9</td>
<td>15</td>
<td>1111111111011010</td>
</tr>
<tr>
<td></td>
<td></td>
<td>21</td>
</tr>
<tr>
<td>Run/Length</td>
<td>Code size</td>
<td>Code</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>7/A</td>
<td>15</td>
<td>1111111101101011</td>
</tr>
<tr>
<td>8/1</td>
<td>8</td>
<td>1111111000</td>
</tr>
<tr>
<td>8/2</td>
<td>14</td>
<td>1111111110000000</td>
</tr>
<tr>
<td>8/3</td>
<td>15</td>
<td>1111111110110110</td>
</tr>
<tr>
<td>8/4</td>
<td>15</td>
<td>1111111110110111</td>
</tr>
<tr>
<td>8/5</td>
<td>15</td>
<td>1111111110111000</td>
</tr>
<tr>
<td>8/6</td>
<td>15</td>
<td>1111111110111001</td>
</tr>
<tr>
<td>8/7</td>
<td>15</td>
<td>1111111110111010</td>
</tr>
<tr>
<td>8/8</td>
<td>15</td>
<td>1111111110111011</td>
</tr>
<tr>
<td>8/9</td>
<td>15</td>
<td>1111111110111000</td>
</tr>
<tr>
<td>8/A</td>
<td>15</td>
<td>1111111110111011</td>
</tr>
<tr>
<td>9/1</td>
<td>8</td>
<td>111111001</td>
</tr>
<tr>
<td>9/2</td>
<td>15</td>
<td>1111111110111110</td>
</tr>
<tr>
<td>9/3</td>
<td>15</td>
<td>1111111110111111</td>
</tr>
<tr>
<td>9/4</td>
<td>15</td>
<td>1111111111000000</td>
</tr>
<tr>
<td>9/5</td>
<td>15</td>
<td>1111111111000001</td>
</tr>
<tr>
<td>9/6</td>
<td>15</td>
<td>1111111111000010</td>
</tr>
<tr>
<td>9/7</td>
<td>15</td>
<td>1111111111000011</td>
</tr>
<tr>
<td>9/8</td>
<td>15</td>
<td>1111111111000100</td>
</tr>
<tr>
<td>9/9</td>
<td>15</td>
<td>1111111111000101</td>
</tr>
<tr>
<td>9/A</td>
<td>15</td>
<td>1111111111000110</td>
</tr>
<tr>
<td>A/1</td>
<td>8</td>
<td>111111010</td>
</tr>
<tr>
<td>A/2</td>
<td>15</td>
<td>1111111111000111</td>
</tr>
<tr>
<td>A/3</td>
<td>15</td>
<td>1111111111010000</td>
</tr>
<tr>
<td>A/4</td>
<td>15</td>
<td>1111111111010001</td>
</tr>
<tr>
<td>A/5</td>
<td>15</td>
<td>1111111111001000</td>
</tr>
<tr>
<td>A/6</td>
<td>15</td>
<td>1111111111001001</td>
</tr>
<tr>
<td>A/7</td>
<td>15</td>
<td>1111111111001010</td>
</tr>
<tr>
<td>A/8</td>
<td>15</td>
<td>1111111111001100</td>
</tr>
<tr>
<td>A/9</td>
<td>15</td>
<td>1111111111001110</td>
</tr>
<tr>
<td>A/A</td>
<td>15</td>
<td>1111111111001111</td>
</tr>
<tr>
<td>B/1</td>
<td>9</td>
<td>111111001</td>
</tr>
<tr>
<td>B/2</td>
<td>15</td>
<td>1111111111010000</td>
</tr>
<tr>
<td>B/3</td>
<td>15</td>
<td>1111111111010001</td>
</tr>
<tr>
<td>B/4</td>
<td>15</td>
<td>1111111111010010</td>
</tr>
<tr>
<td>B/5</td>
<td>15</td>
<td>1111111111010011</td>
</tr>
<tr>
<td>B/6</td>
<td>15</td>
<td>1111111111010100</td>
</tr>
<tr>
<td>B/7</td>
<td>15</td>
<td>1111111111010101</td>
</tr>
<tr>
<td>B/8</td>
<td>15</td>
<td>1111111111010110</td>
</tr>
<tr>
<td>B/9</td>
<td>15</td>
<td>1111111111010111</td>
</tr>
<tr>
<td>B/A</td>
<td>15</td>
<td>1111111111011000</td>
</tr>
<tr>
<td>Run/Length</td>
<td>Code size</td>
<td>Code</td>
</tr>
<tr>
<td>------------</td>
<td>-----------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>C/1</td>
<td>9</td>
<td>11111111010</td>
</tr>
<tr>
<td>C/2</td>
<td>15</td>
<td>1111111111101101</td>
</tr>
<tr>
<td>C/3</td>
<td>15</td>
<td>1111111111011010</td>
</tr>
<tr>
<td>C/4</td>
<td>15</td>
<td>1111111111011111</td>
</tr>
<tr>
<td>C/5</td>
<td>15</td>
<td>1111111111011100</td>
</tr>
<tr>
<td>C/6</td>
<td>15</td>
<td>1111111111011101</td>
</tr>
<tr>
<td>C/7</td>
<td>15</td>
<td>1111111111011110</td>
</tr>
<tr>
<td>C/8</td>
<td>15</td>
<td>1111111111011111</td>
</tr>
<tr>
<td>C/9</td>
<td>15</td>
<td>1111111111100000</td>
</tr>
<tr>
<td>C/A</td>
<td>15</td>
<td>1111111111100001</td>
</tr>
<tr>
<td>D/1</td>
<td>10</td>
<td>11111111000</td>
</tr>
<tr>
<td>D/2</td>
<td>15</td>
<td>1111111111100010</td>
</tr>
<tr>
<td>D/3</td>
<td>15</td>
<td>1111111111100001</td>
</tr>
<tr>
<td>D/4</td>
<td>15</td>
<td>1111111111100001</td>
</tr>
<tr>
<td>D/5</td>
<td>15</td>
<td>1111111111100010</td>
</tr>
<tr>
<td>D/6</td>
<td>15</td>
<td>1111111111100011</td>
</tr>
<tr>
<td>D/7</td>
<td>15</td>
<td>1111111111100011</td>
</tr>
<tr>
<td>D/8</td>
<td>15</td>
<td>1111111111100010</td>
</tr>
<tr>
<td>D/9</td>
<td>15</td>
<td>1111111111100011</td>
</tr>
<tr>
<td>D/A</td>
<td>15</td>
<td>1111111111101010</td>
</tr>
<tr>
<td>E/1</td>
<td>15</td>
<td>111111111101011</td>
</tr>
<tr>
<td>E/2</td>
<td>15</td>
<td>111111111101100</td>
</tr>
<tr>
<td>E/3</td>
<td>15</td>
<td>111111111101101</td>
</tr>
<tr>
<td>E/4</td>
<td>15</td>
<td>111111111101110</td>
</tr>
<tr>
<td>E/5</td>
<td>15</td>
<td>111111111101111</td>
</tr>
<tr>
<td>E/6</td>
<td>15</td>
<td>111111111110000</td>
</tr>
<tr>
<td>E/7</td>
<td>15</td>
<td>111111111110001</td>
</tr>
<tr>
<td>E/8</td>
<td>15</td>
<td>111111111110010</td>
</tr>
<tr>
<td>E/9</td>
<td>15</td>
<td>111111111110100</td>
</tr>
<tr>
<td>E/A</td>
<td>15</td>
<td>111111111110100</td>
</tr>
<tr>
<td>F/0 (ZRL)</td>
<td>10</td>
<td>11111111001</td>
</tr>
<tr>
<td>F/1</td>
<td>15</td>
<td>111111111110101</td>
</tr>
<tr>
<td>F/2</td>
<td>15</td>
<td>111111111110110</td>
</tr>
<tr>
<td>F/3</td>
<td>15</td>
<td>111111111110111</td>
</tr>
<tr>
<td>F/4</td>
<td>15</td>
<td>111111111110100</td>
</tr>
<tr>
<td>F/5</td>
<td>15</td>
<td>111111111110110</td>
</tr>
<tr>
<td>F/6</td>
<td>15</td>
<td>111111111110111</td>
</tr>
<tr>
<td>F/7</td>
<td>15</td>
<td>111111111110111</td>
</tr>
<tr>
<td>F/8</td>
<td>15</td>
<td>111111111111000</td>
</tr>
<tr>
<td>F/9</td>
<td>15</td>
<td>111111111111010</td>
</tr>
<tr>
<td>F/A</td>
<td>15</td>
<td>111111111111110</td>
</tr>
</tbody>
</table>
### C.4 Chroma AC Coefficients

<table>
<thead>
<tr>
<th>Run/Length</th>
<th>Code size</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0/0 (EOB)</td>
<td>3</td>
<td>1010</td>
</tr>
<tr>
<td>0/1</td>
<td>1</td>
<td>00</td>
</tr>
<tr>
<td>0/2</td>
<td>1</td>
<td>01</td>
</tr>
<tr>
<td>0/3</td>
<td>2</td>
<td>100</td>
</tr>
<tr>
<td>0/4</td>
<td>3</td>
<td>1011</td>
</tr>
<tr>
<td>0/5</td>
<td>4</td>
<td>11010</td>
</tr>
<tr>
<td>0/6</td>
<td>6</td>
<td>1111000</td>
</tr>
<tr>
<td>0/7</td>
<td>7</td>
<td>11111000</td>
</tr>
<tr>
<td>0/8</td>
<td>9</td>
<td>1111110110</td>
</tr>
<tr>
<td>0/9</td>
<td>15</td>
<td>111111110000010</td>
</tr>
<tr>
<td>0/A</td>
<td>15</td>
<td>1111111110000011</td>
</tr>
<tr>
<td>1/1</td>
<td>3</td>
<td>1100</td>
</tr>
<tr>
<td>1/2</td>
<td>4</td>
<td>11011</td>
</tr>
<tr>
<td>1/3</td>
<td>6</td>
<td>1111001</td>
</tr>
<tr>
<td>1/4</td>
<td>8</td>
<td>111110110</td>
</tr>
<tr>
<td>1/5</td>
<td>10</td>
<td>11111110110</td>
</tr>
<tr>
<td>1/6</td>
<td>15</td>
<td>111111111000010</td>
</tr>
<tr>
<td>1/7</td>
<td>15</td>
<td>1111111110000101</td>
</tr>
<tr>
<td>1/8</td>
<td>15</td>
<td>1111111110000110</td>
</tr>
<tr>
<td>1/9</td>
<td>15</td>
<td>1111111110000011</td>
</tr>
<tr>
<td>1/A</td>
<td>15</td>
<td>1111111110001000</td>
</tr>
<tr>
<td>2/1</td>
<td>4</td>
<td>11100</td>
</tr>
<tr>
<td>2/2</td>
<td>7</td>
<td>11111001</td>
</tr>
<tr>
<td>2/3</td>
<td>9</td>
<td>111110111</td>
</tr>
<tr>
<td>2/4</td>
<td>11</td>
<td>11111110100</td>
</tr>
<tr>
<td>2/5</td>
<td>15</td>
<td>1111111110001001</td>
</tr>
<tr>
<td>2/6</td>
<td>15</td>
<td>1111111110001010</td>
</tr>
<tr>
<td>2/7</td>
<td>15</td>
<td>1111111110001011</td>
</tr>
<tr>
<td>2/8</td>
<td>15</td>
<td>1111111110001100</td>
</tr>
<tr>
<td>2/9</td>
<td>15</td>
<td>1111111110001101</td>
</tr>
<tr>
<td>2/A</td>
<td>15</td>
<td>1111111110001110</td>
</tr>
<tr>
<td>3/1</td>
<td>5</td>
<td>111010</td>
</tr>
<tr>
<td>3/2</td>
<td>8</td>
<td>111110111</td>
</tr>
<tr>
<td>3/3</td>
<td>11</td>
<td>11111110101</td>
</tr>
<tr>
<td>3/4</td>
<td>15</td>
<td>1111111110001111</td>
</tr>
<tr>
<td>3/5</td>
<td>15</td>
<td>1111111110010000</td>
</tr>
<tr>
<td>3/6</td>
<td>15</td>
<td>1111111110010001</td>
</tr>
<tr>
<td>3/7</td>
<td>15</td>
<td>1111111110010010</td>
</tr>
<tr>
<td>3/8</td>
<td>15</td>
<td>1111111110010011</td>
</tr>
<tr>
<td>Run/Length</td>
<td>Code size</td>
<td>Code</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>3/9</td>
<td>15</td>
<td>1111111110010100</td>
</tr>
<tr>
<td>3/A</td>
<td>15</td>
<td>11111111110010101</td>
</tr>
<tr>
<td>4/1</td>
<td>5</td>
<td>111011</td>
</tr>
<tr>
<td>4/2</td>
<td>9</td>
<td>1111111000</td>
</tr>
<tr>
<td>4/3</td>
<td>15</td>
<td>1111111110010110</td>
</tr>
<tr>
<td>4/4</td>
<td>15</td>
<td>1111111110011011</td>
</tr>
<tr>
<td>4/5</td>
<td>15</td>
<td>1111111110011100</td>
</tr>
<tr>
<td>4/6</td>
<td>15</td>
<td>1111111110011101</td>
</tr>
<tr>
<td>4/7</td>
<td>15</td>
<td>1111111110011010</td>
</tr>
<tr>
<td>4/8</td>
<td>15</td>
<td>1111111110011111</td>
</tr>
<tr>
<td>4/9</td>
<td>15</td>
<td>1111111110011110</td>
</tr>
<tr>
<td>4/A</td>
<td>15</td>
<td>1111111110011110</td>
</tr>
<tr>
<td>5/1</td>
<td>6</td>
<td>1111010</td>
</tr>
<tr>
<td>5/2</td>
<td>10</td>
<td>11111110111</td>
</tr>
<tr>
<td>5/3</td>
<td>15</td>
<td>111111110011110</td>
</tr>
<tr>
<td>5/4</td>
<td>15</td>
<td>1111111110011111</td>
</tr>
<tr>
<td>5/5</td>
<td>15</td>
<td>1111111110100000</td>
</tr>
<tr>
<td>5/6</td>
<td>15</td>
<td>1111111110100001</td>
</tr>
<tr>
<td>5/7</td>
<td>15</td>
<td>1111111110100011</td>
</tr>
<tr>
<td>5/8</td>
<td>15</td>
<td>1111111110100100</td>
</tr>
<tr>
<td>5/9</td>
<td>15</td>
<td>1111111110100101</td>
</tr>
<tr>
<td>5/A</td>
<td>15</td>
<td>1111111110100101</td>
</tr>
<tr>
<td>6/1</td>
<td>6</td>
<td>1111011</td>
</tr>
<tr>
<td>6/2</td>
<td>11</td>
<td>11111110110</td>
</tr>
<tr>
<td>6/3</td>
<td>15</td>
<td>1111111110100110</td>
</tr>
<tr>
<td>6/4</td>
<td>15</td>
<td>1111111110100111</td>
</tr>
<tr>
<td>6/5</td>
<td>15</td>
<td>1111111110101000</td>
</tr>
<tr>
<td>6/6</td>
<td>15</td>
<td>1111111110101001</td>
</tr>
<tr>
<td>6/7</td>
<td>15</td>
<td>1111111110101010</td>
</tr>
<tr>
<td>6/8</td>
<td>15</td>
<td>1111111110101011</td>
</tr>
<tr>
<td>6/9</td>
<td>15</td>
<td>1111111110101100</td>
</tr>
<tr>
<td>6/A</td>
<td>15</td>
<td>1111111110101101</td>
</tr>
<tr>
<td>7/1</td>
<td>7</td>
<td>11111010</td>
</tr>
<tr>
<td>7/2</td>
<td>11</td>
<td>11111110111</td>
</tr>
<tr>
<td>7/3</td>
<td>15</td>
<td>1111111110101110</td>
</tr>
<tr>
<td>7/4</td>
<td>15</td>
<td>1111111110101111</td>
</tr>
<tr>
<td>7/5</td>
<td>15</td>
<td>1111111110110000</td>
</tr>
<tr>
<td>7/6</td>
<td>15</td>
<td>1111111110110001</td>
</tr>
<tr>
<td>7/7</td>
<td>15</td>
<td>1111111110110010</td>
</tr>
<tr>
<td>7/8</td>
<td>15</td>
<td>1111111110110011</td>
</tr>
<tr>
<td>7/9</td>
<td>15</td>
<td>1111111110110100</td>
</tr>
<tr>
<td>Run/Length</td>
<td>Code size</td>
<td>Code</td>
</tr>
<tr>
<td>------------</td>
<td>-----------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>7/A</td>
<td>15</td>
<td>1111111111011010</td>
</tr>
<tr>
<td>8/1</td>
<td>8</td>
<td>111111000</td>
</tr>
<tr>
<td>8/2</td>
<td>14</td>
<td>111111111000000</td>
</tr>
<tr>
<td>8/3</td>
<td>15</td>
<td>11111111110110111</td>
</tr>
<tr>
<td>8/4</td>
<td>15</td>
<td>1111111111011011</td>
</tr>
<tr>
<td>8/5</td>
<td>15</td>
<td>1111111111011100</td>
</tr>
<tr>
<td>8/6</td>
<td>15</td>
<td>11111111110111001</td>
</tr>
<tr>
<td>8/7</td>
<td>15</td>
<td>11111111110111010</td>
</tr>
<tr>
<td>8/8</td>
<td>15</td>
<td>1111111110111011</td>
</tr>
<tr>
<td>8/9</td>
<td>15</td>
<td>11111111110111000</td>
</tr>
<tr>
<td>8/A</td>
<td>15</td>
<td>1111111111011101</td>
</tr>
<tr>
<td>9/1</td>
<td>8</td>
<td>111111001</td>
</tr>
<tr>
<td>9/2</td>
<td>15</td>
<td>1111111111011111</td>
</tr>
<tr>
<td>9/3</td>
<td>15</td>
<td>111111111101111</td>
</tr>
<tr>
<td>9/4</td>
<td>15</td>
<td>1111111111000000</td>
</tr>
<tr>
<td>9/5</td>
<td>15</td>
<td>1111111111000001</td>
</tr>
<tr>
<td>9/6</td>
<td>15</td>
<td>1111111111000010</td>
</tr>
<tr>
<td>9/7</td>
<td>15</td>
<td>1111111111000011</td>
</tr>
<tr>
<td>9/8</td>
<td>15</td>
<td>1111111111000100</td>
</tr>
<tr>
<td>9/9</td>
<td>15</td>
<td>1111111111000101</td>
</tr>
<tr>
<td>9/A</td>
<td>15</td>
<td>1111111111000110</td>
</tr>
<tr>
<td>A/1</td>
<td>8</td>
<td>111111010</td>
</tr>
<tr>
<td>A/2</td>
<td>15</td>
<td>1111111111000011</td>
</tr>
<tr>
<td>A/3</td>
<td>15</td>
<td>1111111111001000</td>
</tr>
<tr>
<td>A/4</td>
<td>15</td>
<td>1111111111001001</td>
</tr>
<tr>
<td>A/5</td>
<td>15</td>
<td>1111111111001010</td>
</tr>
<tr>
<td>A/6</td>
<td>15</td>
<td>1111111111001011</td>
</tr>
<tr>
<td>A/7</td>
<td>15</td>
<td>1111111111001100</td>
</tr>
<tr>
<td>A/8</td>
<td>15</td>
<td>1111111111000110</td>
</tr>
<tr>
<td>A/9</td>
<td>15</td>
<td>1111111111001110</td>
</tr>
<tr>
<td>A/A</td>
<td>15</td>
<td>1111111111001111</td>
</tr>
<tr>
<td>B/1</td>
<td>9</td>
<td>111111001</td>
</tr>
<tr>
<td>B/2</td>
<td>15</td>
<td>1111111111010000</td>
</tr>
<tr>
<td>B/3</td>
<td>15</td>
<td>1111111111010001</td>
</tr>
<tr>
<td>B/4</td>
<td>15</td>
<td>1111111111010001</td>
</tr>
<tr>
<td>B/5</td>
<td>15</td>
<td>1111111111010011</td>
</tr>
<tr>
<td>B/6</td>
<td>15</td>
<td>1111111111010100</td>
</tr>
<tr>
<td>B/7</td>
<td>15</td>
<td>1111111111010101</td>
</tr>
<tr>
<td>B/8</td>
<td>15</td>
<td>1111111111011010</td>
</tr>
<tr>
<td>B/9</td>
<td>15</td>
<td>1111111111011011</td>
</tr>
<tr>
<td>B/A</td>
<td>15</td>
<td>1111111111011000</td>
</tr>
<tr>
<td>Run/Length</td>
<td>Code size</td>
<td>Code</td>
</tr>
<tr>
<td>------------</td>
<td>-----------</td>
<td>--------------------</td>
</tr>
<tr>
<td>C/1</td>
<td>9</td>
<td>11111111010</td>
</tr>
<tr>
<td>C/2</td>
<td>15</td>
<td>1111111111011001</td>
</tr>
<tr>
<td>C/3</td>
<td>15</td>
<td>1111111111010111</td>
</tr>
<tr>
<td>C/4</td>
<td>15</td>
<td>1111111111011111</td>
</tr>
<tr>
<td>C/5</td>
<td>15</td>
<td>1111111111011100</td>
</tr>
<tr>
<td>C/6</td>
<td>15</td>
<td>1111111111011101</td>
</tr>
<tr>
<td>C/7</td>
<td>15</td>
<td>1111111111011110</td>
</tr>
<tr>
<td>C/8</td>
<td>15</td>
<td>1111111111011111</td>
</tr>
<tr>
<td>C/9</td>
<td>15</td>
<td>1111111111000000</td>
</tr>
<tr>
<td>C/A</td>
<td>15</td>
<td>1111111111000001</td>
</tr>
<tr>
<td>D/1</td>
<td>10</td>
<td>11111111000</td>
</tr>
<tr>
<td>D/2</td>
<td>15</td>
<td>1111111111100010</td>
</tr>
<tr>
<td>D/3</td>
<td>15</td>
<td>1111111111100011</td>
</tr>
<tr>
<td>D/4</td>
<td>15</td>
<td>1111111111100100</td>
</tr>
<tr>
<td>D/5</td>
<td>15</td>
<td>1111111111100101</td>
</tr>
<tr>
<td>D/6</td>
<td>15</td>
<td>1111111111100110</td>
</tr>
<tr>
<td>D/7</td>
<td>15</td>
<td>1111111111100111</td>
</tr>
<tr>
<td>D/8</td>
<td>15</td>
<td>1111111111101000</td>
</tr>
<tr>
<td>D/9</td>
<td>15</td>
<td>1111111111101001</td>
</tr>
<tr>
<td>D/A</td>
<td>15</td>
<td>1111111111101010</td>
</tr>
<tr>
<td>E/1</td>
<td>15</td>
<td>1111111111101011</td>
</tr>
<tr>
<td>E/2</td>
<td>15</td>
<td>1111111111101100</td>
</tr>
<tr>
<td>E/3</td>
<td>15</td>
<td>1111111111101111</td>
</tr>
<tr>
<td>E/4</td>
<td>15</td>
<td>1111111111101110</td>
</tr>
<tr>
<td>E/5</td>
<td>15</td>
<td>1111111111101111</td>
</tr>
<tr>
<td>E/6</td>
<td>15</td>
<td>1111111111100000</td>
</tr>
<tr>
<td>E/7</td>
<td>15</td>
<td>1111111111100001</td>
</tr>
<tr>
<td>E/8</td>
<td>15</td>
<td>1111111111100100</td>
</tr>
<tr>
<td>E/9</td>
<td>15</td>
<td>1111111111100111</td>
</tr>
<tr>
<td>E/A</td>
<td>15</td>
<td>1111111111101001</td>
</tr>
<tr>
<td>F/0 (ZRL)</td>
<td>10</td>
<td>11111111001</td>
</tr>
<tr>
<td>F/1</td>
<td>15</td>
<td>1111111111101011</td>
</tr>
<tr>
<td>F/2</td>
<td>15</td>
<td>1111111111101100</td>
</tr>
<tr>
<td>F/3</td>
<td>15</td>
<td>1111111111101111</td>
</tr>
<tr>
<td>F/4</td>
<td>15</td>
<td>1111111111101100</td>
</tr>
<tr>
<td>F/5</td>
<td>15</td>
<td>1111111111101101</td>
</tr>
<tr>
<td>F/6</td>
<td>15</td>
<td>1111111111110000</td>
</tr>
<tr>
<td>F/7</td>
<td>15</td>
<td>1111111111110001</td>
</tr>
<tr>
<td>F/8</td>
<td>15</td>
<td>1111111111111001</td>
</tr>
<tr>
<td>F/9</td>
<td>15</td>
<td>1111111111111010</td>
</tr>
<tr>
<td>F/A</td>
<td>15</td>
<td>1111111111111110</td>
</tr>
</tbody>
</table>
D Source Code

D.1 array_shrinker.v

```vhd
'timescale 1ns / 1ps

// //////////////////////////////////////////////////////////////////////////
// Engineer: Evan Broder
// // Create Date: 12:08:03 11/18/2007
// Module Name: array_shrinker
// Project Name: Video-Conferencing System
// Description: Truncates the most-significant bytes from the input for each
// of 8 elements in a concatenated vector
// //
// Dependencies:
// // Revision:
// $Id: array_shrinker.v 91 2007-11-19 07:30:47Z evan $
// // Additional Comments:
// //
// //////////////////////////////////////////////////////////////////////////

module array_shrinker(reset, clock, big, little);

parameter BIG_WIDTH = 16;
parameter LITTLE_WIDTH = 12;
input reset;
input clock;
input [(BIG_WIDTH * 8) - 1:0] big;
output [(LITTLE_WIDTH * 8) - 1:0] little;

genvar i;
generate for (i = 0; i < 8; i = i + 1)
begin
    // Take the full width of the little vector alloted to this particular
    // element
    assign little[(LITTLE_WIDTH * (i + 1)) - 1 : LITTLE_WIDTH * i] =
        // And assign it to the <LITTLE_WIDTH> least-significant bits of
        // big
        big[(BIG_WIDTH * i) + LITTLE_WIDTH - 1 : BIG_WIDTH * i];
end
generate

endmodule
```

D.2 array_sign_extender.v

```vhd
'timescale 1ns / 1ps

// //////////////////////////////////////////////////////////////////////////
// Engineer: Evan Broder
// // Create Date: 12:26:26 11/17/2007
// Module Name: array_sign_extender
// //
```
module array_sign_extender(reset, clock, little, big);
  parameter LITTLE_WIDTH = 8;
  parameter BIG_WIDTH = 12;
  input reset;
  input clock;
  input [(LITTLE_WIDTH * 8) - 1:0] little;
  output [(BIG_WIDTH * 8) - 1:0] big;

  genvar i,j;
  generate for (i = 0; i < 8; i = i + 1)
    // Assigns the parts that are the same length
    assign big[(BIG_WIDTH * i) * LITTLE_WIDTH - 1 : BIG_WIDTH * i] =
      little[(LITTLE_WIDTH * (i + 1)) - 1 : LITTLE_WIDTH * i];
    // Handles the sign-extension
    for (j = LITTLE_WIDTH; j < BIG_WIDTH; j = j + 1)
      begin: sign
        assign big[(BIG_WIDTH * i) * j] =
          little[(LITTLE_WIDTH * (i + 1)) - 1];
      end
  endgenerate
endmodule

D.3 clock_divider.v

'timescale 1ns / 1ps

// Engineer: Evan Broder

// Create Date: 16:27:14 12/01/2007
// Module Name: clock_divider
// Project Name: Video-Conferencing System
// Description: Taken from code developed for 6.111 Lab 3, this module outputs
// an enable signal every D clock cycles

// Dependencies:

// Revision:
module clock_divider(reset, clock, enable);

// 27000000 / 250000 = 108
// (27 megahertz clock, 250 kilohertz divider speed)
parameter D = 108;

input clock, reset;
output enable;
reg [24:0] count;

always @(posedge clock)
begin
  if (count == (D - 1) || reset) count <= 0;
  else count <= count + 1;
end
assign enable = (count == (D - 1));
endmodule

D.4 coordinate_delay.v

module coordinate_delay(reset, clock, valid_in, x_in, y_in, valid_out, x_out, y_out);
input reset;
input clock;
input valid_in;

// $Id: coordinate_delay.v 291 2007-12-15 16:27:42Z evan $
input [5:0] x_in;
input [4:0] y_in;
input valid_out;
output [5:0] x_out;
output [4:0] y_out;

reg [5:0] x_cache;
reg [4:0] y_cache;

wire valid_in_pulse, valid_out_pulse;

level_to_pulse valid_in_level(.reset(reset), .clock(clock),
  .level(valid_in), .pulse(valid_in_pulse));
level_to_pulse valid_out_level(.reset(reset), .clock(clock),
  .level(valid_out), .pulse(valid_out_pulse));

// Grab x_in and y_in on the positive edge of valid_in
always @(posedge clock)
begin
  if (valid_in_pulse)
  begin
    x_cache <= x_in;
    y_cache <= y_in;
  end
end

// Latch the old values until the instant that valid_out is asserted -
// we don't want a one cycle delay here, which is why this isn't being done
// sequentially
assign x_out = valid_out_pulse ? x_cache : x_out;
assign y_out = valid_out_pulse ? y_cache : y_out;
endmodule

D.5  crc.v

`timescale 1ns / 1ps

///////////////////////////////////////////////////////////////////////////////////
// Engineer:   Evan Broder
///////////////////////////////////////////////////////////////////////////////////
// Create Date: 12:08:03 11/18/2007
// Module Name: crc
// Project Name: Video-Conferencing System
// Description: Incorporates a new byte on data into the current value of the
// CRC.
// Function generated by www.easics.com
// Info: tools@easics.be
// http://www.easics.com
//
// Dependencies:
//
module crc(reset, clock, data, crc, en);
  input reset;
  input clock;
  input [7:0] data;
  output reg [7:0] crc;
  input en;

  // polynomial: (0 1 2 8)
  // data width: 8
  // convention: the first serial data bit is D[7]

  function [7:0] nextCRC8_D8;
    input [7:0] Data;
    input [7:0] CRC;
    reg [7:0] D;
    reg [7:0] C;
    reg [7:0] NewCRC;

    begin
      D = Data;
      C = CRC;
      nextCRC8_D8 = NewCRC;
    end
  endfunction

always @(posedge clock)
  begin
    if (reset) crc <= 0;
    else if (en) crc <= nextCRC8_D8(data, crc);
  end
endmodule

D.6 data_spew.v

.timescale ins / ips
module data_spew(reset, clock, data_set, start, row, valid_out);

  input reset;
  input clock;
  input [2:0] data_set;
  input start;
  output [63:0] row;
  output valid_out;

  parameter S_IDLE = 0;
  parameter S_SPEW = 1;

  wire [6:0] addr;
  reg state;
  reg [2:0] count;

  // This is how the BRAM is addressed
  assign addr = {data_set, count};

  data_spew_mem mem(.clk(clock), .addr(addr), .dout(row));

  assign valid_out = (state == S_SPEW);

  always @(posedge clock)
  begin
    if (reset)
      begin
        state <= S_IDLE;
        count <= 0;
      end
    else
      case (state)
      S_IDE: begin
      end
  end
```verilog
if (start) state <= S_SPEW;
end
S_SPEW:
begin
if (count == 7) state <= S_IDLE;
count <= count + 1;
end
endcase
endmodule

D.7 dct_1d.v

`timescale ins / ips

/////////////////////////////////////////////////////////////////////////////
// Engineer: Evan Broder, Chris Post
// Create Date: 19:23:10 11/05/2007
// Module Name: dct_1d
// Project Name: Video-Conferencing System
// Description: Finds a one-dimensional DCT based on the algorithm outlined
// in Agostini:2001 and corrected in Agostini:2005
//
module dct_1d(reset, clock, dct_in, dct_out);
  parameter WIDTH = 12;
  parameter m1 = 23170;
  parameter m2 = 12540;
  parameter m3 = 17734;
  parameter m4 = 42813;
  input reset;
  input clock;
  // Verilog doesn’t seem to like using arrays as inputs and outputs, so
  // we’ll take in the data as one massive vector
  input [WIDTH * 8 - 1:0] dct_in;
  // ...and output it as one even /more/ massive vector
  output [((WIDTH + 4) * 8) - 1:0] dct_out;
```
Once we get the data in, though, the array notation makes the implementation very clean, so we’ll use that. The increasing widths used to store the data are taken from the Agostini paper.

```verilog
wire signed [WIDTH - 1:0] a[0:7];
reg signed [WIDTH:0] b[0:7];
reg signed [WIDTH + 1:0] c[0:7];
reg signed [WIDTH + 2:0] d[0:8];
reg signed [WIDTH + 2:0] e[0:8];
reg signed [WIDTH + 3:0] f[0:7];
reg signed [WIDTH + 3:0] S[0:7];
```

Split up the really wide bus into an array of 8 8-bit values.

```verilog
genvar i;
generate for (i = 0; i < 8; i = i + 1)
begin:
    //assign a[i] = row[WIDTH * (7 - i) +: WIDTH];
    assign a[i] = dct_in[(WIDTH * (8 - i)) - 1 : WIDTH * (7 - i)];
end
generate
```

Take the final answer and concatenate it back into a giant array.

```verilog
assign dct_out = {S[0], S[1], S[2], S[3], S[4], S[5], S[6], S[7]};
```

always @(posedge clock)

```verilog
begin
    // Each variable is a different stage in the pipeline
    b[0] <= a[0] + a[7];
    b[1] <= a[1] + a[6];
    // This next line was wrong in the original Agostini paper, but 
    // corrected later
    b[7] <= a[0] - a[7];
    c[0] <= b[0] + b[5];
    c[4] <= b[0] - b[5];
    c[7] <= b[7];
    d[0] <= c[0] + c[3];
    d[1] <= c[0] - c[3];
    d[2] <= c[2];
```
D.8 dct_2d.v

`timescale ins / 1ps

// In the multiplier stage, the constant multiplicands were bit shifted
// to get integer numbers. Once the multiplication is done, we bit
// shift back the same amount

e[0] <= d[0];
e[1] <= d[1];
e[2] <= (m3 * d[2]) >> 15;
e[3] <= (m1 * d[7]) >> 15;
e[4] <= (m4 * d[6]) >> 15;
e[5] <= d[5];
e[6] <= (m1 * d[3]) >> 15;
e[7] <= (m2 * d[4]) >> 15;
e[8] <= d[8];
f[0] <= e[0];
f[1] <= e[1];
S[0] <= f[0];
S[2] <= f[2];
S[4] <= f[1];
S[6] <= f[3];
end
endmodule
module dct_2d(reset, clock, row, valid_in, column, valid_out, x_in, y_in, 
x_out, y_out);

input reset;
input clock;
input [63:0] row;
input valid_in;
output [127:0] column;
output valid_out;

input [5:0] x_in;
input [4:0] y_in;
output [5:0] x_out;
output [4:0] y_out;

assign column = dct_out;

assign dct_in = shift_column ? dct_column_in : row_extended;

// shift_row should happen 6 clock signals after data first starts coming 
// in because that's the latency of the DCT pipeline
delay shift_row_delay(.clock(clock), .undelayed(valid_in), .delayed(shift_row));
defparam shift_row_delay.DELAY = 6;
// 8 clock cycles after data starts row shifting into the matrix, it should
// all be there, so start shifting it out in columns
// Note that, for right now, I’m assuming that the data is coming in a
// single chunk without interruptions
delay shift_column_delay(.clock(clock), .undelayed(shift_row),
 .delayed(shift_column));
defparam shift_column_delay.DELAY = 8;

// And then, 6 cycles after we start looking at column-wise data, it should
// actually start coming out the other end
delay valid_out_delay(.clock(clock), .undelayed(shift_column),
 .delayed(valid_out));
defparam valid_out_delay.DELAY = 6;

coordinate_delay coord_delay(.reset(reset), .clock(clock),
 .valid_in(valid_in), .x_in(x_in), .y_in(y_in),
 .valid_out(valid_out), .x_out(x_out), .y_out(y_out));

level_shifter shifter(.reset(reset), .clock(clock), .row(row),
 .row_shifted(row_shifted));

// Since the DCT needs to be wide enough for the 2nd round, it needs to be
// wider than the incoming data (which has now been signed), so sign-extend
// each of the 8 values up to 12 bits
array_sign_extender sign_extender(.reset(reset), .clock(clock),
 .little(row_shifted), .big(row_extended));
defparam sign_extender.LITTLE_WIDTH = 8;
defparam sign_extender.BIG_WIDTH = 12;

dct_1d dct_1d(.reset(reset), .clock(clock), .dct_in(dct_in),
 .dct_out(dct_out));
defparam dct_1d.WIDTH = 12;

// The first time the data comes out, it’ll be 16 bits, but there are only
// 12 bits of information, and we need to shorten it so it’ll fit back into
// the DCT, so chop off the 4 MSBs for each value
array_shrinker shrinker(.reset(reset), .clock(clock), .big(dct_out),
 .little(dct_out_shrunk));
defparam shrinker.BIG_WIDTH = 16;
defparam shrinker.LITTLE_WIDTH = 12;

matrix_transpose transpose(.reset(reset), .clock(clock), .row(dct_out_shrunk),
 .column(dct_column_in), .shift_row(shift_row),
 .shift_column(shift_column));
defparam transpose.WIDTH = 12;
endmodule
// use your system clock for the clock input
// to produce a synchronous, debounced output
module debounce (reset, clock, noisy, clean);

parameter DELAY = 270000; // .01 sec with a 27Mhz clock
input reset, clock, noisy;
output clean;

reg [18:0] count;
reg new, clean;

always @(posedge clock)
  if (reset)
    begin
      count <= 0;
      new <= noisy;
      clean <= noisy;
    end
  else if (noisy != new)
    begin
      new <= noisy;
      count <= 0;
    end
  else if (count == DELAY)
    clean <= new;
  else
    count <= count+1;
endmodule

D.10 decoder.v

' timescale 1ns / 1ps

// ///////////////////////////////////////////////////////////////////////////
// Engineer: Evan Broder
//
// Create Date: 15:37:11 12/09/2007
// Module Name: decoder
// Project Name: Video-Conferencing System
// Description: Feeds incoming serial data through each stage of the decoding
// process
//
// Dependencies:
//
// Revision:
// $Id: decoder.v 291 2007-12-15 16:27:42Z evan $
//
// Additional Comments:
//
// ///////////////////////////////////////////////////////////////////////////
module decoder(reset, clock, stream, valid_in, ready, x_in, y_in, row_out,
   valid_out, x_out, y_out, eh_value_out, eh_valid_out, deq_column_out,
D.11  delay.v

1 'timescale 1ns / 1ps
2
3 module delay(clock, undelayed, delayed);
4   parameter DELAY = 1;
5   input clock, undelayed;
6   output delayed;
7
8   reg [DELAY-2:0] buffer;
9   reg delayed;
10
11   always @(posedge clock)
if (DELAY == 1) delayed <= undelayed;
else {delayed, buffer} <= {buffer, undelayed};
endmodule

D.12 dequantizer.v

`timescale 1ns / 1ps

_SHARED

Engineer: Evan Broder

Create Date: 15:00:49 11/29/2007
Module Name: dequantizer
Project Name: Video-Conferencing System
Description: Dequantizes each value from the DCT by a separate quantization factor.

Dependencies:
Revision:
$Id: dequantizer.v 224 2007-12-11 01:38:03Z evan $

Additional Comments:

module dequantizer(reset, clock, value_in, valid_in, column_out, valid_out, x_in, x_out, y_in, y_out);
    parameter S_IN = 0;
    parameter S_OUT = 1;

    input reset;
    input clock;
    input [11:0] value_in;
    input valid_in;
    output [95:0] column_out;
    output valid_out;
    input [5:0] x_in;
    input [4:0] y_in;
    output [5:0] x_out;
    output [4:0] y_out;

    reg [11:0] buffer [0:7][0:7];
    reg [5:0] addr;
    reg [5:0] buffer_addr;
    reg [11:0] multiplicand;
    wire [15:0] q_factor;
    wire [27:0] product;
    wire posedge_valid_in, nedge_valid_in;
    wire buffer_store;
    reg [2:0] out_counter;

    reg state;
level_to_pulse pos_valid_in(.reset(reset), .clock(clock),
   .level(valid_in), .pulse(posedge_valid_in));
level_to_pulse neg_valid_in(.reset(reset), .clock(clock),
   .level(~valid_in), .pulse(negedge_valid_in));
delay buffer_store_delay(.clock(clock), .undelayed(valid_in),
   .delayed(buffer_store));
defparam buffer_store_delay.DELAY = 3;
delay buffer_store_delay(.clock(clock), .undelayed(valid_in),
   .delayed(buffer_store));
defparam buffer_store_delay.DELAY = 3;
coordinate_delay coord_delay(.reset(reset), .clock(clock),
   .valid_in(valid_in), .x_in(x_in), .y_in(y_in),
   .valid_out(valid_out), .x_out(x_out), .y_out(y_out));
luma_dequantizer_table q_factors(.clk(clock), .addr(addr), .dout(q_factor));
dequantizer_mult mult(.clk(clock), .a(multiplicand), .b(q_factor),
   .q(product));
always @(posedge clock)
begin
  if (reset)
  begin
    addr <= 0;
    buffer_addr <= 0;
    state <= 0;
  end
  else
    case (state)
      S_IN:
      begin
        if (buffer_addr == 63)
        begin
          out_counter <= 0;
          state <= S_OUT;
        end
        if (valid_in)
        begin
          addr <= addr + 1;
          multiplicand <= value_in;
        end
        if (buffer_store)
        begin
          buffer_addr <= buffer_addr + 1;
          buffer[buffer_addr[5:3]][buffer_addr[2:0]] <=
            {product[27], product[22:12]} + product[11];
        end
      end
      S_OUT:
      begin
        // code
      end
    endcase
  end
{'timescale 1ns / 1ps

// 6.111 FPGA Labkit -- Hex display driver

// File: display_16hex.v
// Date: 24-Sep-05
// Created: April 27, 2004
// Author: Nathan Ickes

// 24-Sep-05 Ike: updated to use new reset-once state machine, remove clear
// 28-Nov-06 CJT: fixed race condition between CE and RS (thanks Javier!)
// This verilog module drives the labkit hex dot matrix displays, and puts
// up 16 hexadecimal digits (8 bytes). These are passed to the module
// through a 64 bit wire ("data"), asynchronously.

module display_16hex (reset, clock_27mhz, data,
    disp_blank, disp_clock, disp_rs, disp_ce_b,
    disp_reset_b, disp_data_out);

input reset, clock_27mhz; // clock and reset (active high reset)
input [63:0] data; // 16 hex nibbles to display

output disp_blank, disp_clock, disp_data_out, disp_rs, disp_ce_b,
    disp_reset_b;
```verilog
define disp_data_out, disp_rs, disp_ce_b, disp_reset_b;

 Verilog Code

-- Display Clock
-- Generate a 500kHz clock for driving the displays.

-- Verilog Code

always @(posedge clock_27mhz)
begin
    if (reset)
        begin
            count = 0;
            clock = 0;
        end
    else if (count == 26)
        begin
            clock = ~clock;
            count = 5'h00;
        end
    else
        count = count+1;
end

always @(posedge clock_27mhz)
if (reset)
    reset_count <= 100;
else
    reset_count <= (reset_count==0) ? 0 : reset_count-1;

assign dreset = (reset_count != 0);
assign disp_clock = ~clock;

-- Verilog Code

-- Display State Machine

-- Verilog Code

reg [7:0] state; // FSM state
reg [9:0] dot_index; // index to current dot being clocked out
reg [31:0] control; // control register
reg [3:0] char_index; // index of current character
```
reg [39:0] dots;       // dots for a single digit
reg [3:0] nibble;  // hex nibble of current character

assign disp_blank = 1'b0; // low <= not blanked

always @(posedge clock)
  if (dreset)
    begin
      state <= 0;
      dot_index <= 0;
      control <= 32'h7F7F7F7F;
    end
  else
    casex (state)
    8'h00: begin
      // Reset displays
      disp_data_out <= 1'b0;
      disp_rs <= 1'b0; // dot register
      disp_ce_b <= 1'b1;
      disp_reset_b <= 1'b0;
      dot_index <= 0;
      state <= state+1;
    end
  8'h01: begin
    // End reset
    disp_reset_b <= 1'b1;
    state <= state+1;
  end
  8'h02: begin
    // Initialize dot register (set all dots to zero)
    disp_ce_b <= 1'b0;
    disp_data_out <= 1'b0; // dot_index[0];
    if (dot_index == 639)
      state <= state+1;
    else
      dot_index <= dot_index+1;
  end
  8'h03: begin
    // Latch dot data
    disp_ce_b <= 1'b1;
    dot_index <= 31; // re-purpose to init ctrl reg
    disp_rs <= 1'b1; // Select the control register
    state <= state+1;
  end
begin
  // Setup the control register
  disp_ce_b <= 1'b0;
  disp_data_out <= control[31];
  control <= {control[30:0], 1'b0}; // shift left
  if (dot_index == 0)
    state <= state+1;
  else
    dot_index <= dot_index-1;
end

begin
  // Latch the control register data / dot data
  disp_ce_b <= 1'b1;
  dot_index <= 39; // init for single char
  char_index <= 15; // start with MS char
  state <= state+1;
  disp_rs <= 1'b0; // Select the dot register
end

begin
  // Load the user's dot data into the dot reg, char by char
  disp_ce_b <= 1'b0;
  disp_data_out <= dots[dot_index]; // dot data from msb
  if (dot_index == 0)
    if (char_index == 0)
      state <= 5; // all done, latch data
  else
    begin
      char_index <= char_index - 1; // goto next char
      dot_index <= 39;
    end
  else
    dot_index <= dot_index-1; // else loop thru all dots
end
always @ (data or char_index)
  case (char_index)
    4'h0:  nibble <= data[3:0];
    4'h1:  nibble <= data[7:4];
    4'h2:  nibble <= data[11:8];
    4'h3:  nibble <= data[15:12];
    4'h4:  nibble <= data[19:16];
    4'h5:  nibble <= data[23:20];
    4'h6:  nibble <= data[27:24];
always @(nibble)
  case (nibble)
    4'h0: dots <= 40'b00111110_01010001_01001001_01000101_00111110;
    4'h1: dots <= 40'b00000000_01000010_01111111_01000000_00000000;
    4'h2: dots <= 40'b01111111_00000101_00000101_01001001_01001001;
    4'h3: dots <= 40'b01111111_00001001_00001001_01001001_01001001;
    4'h4: dots <= 40'b01111111_00000101_00000101_01001001_01001001;
    4'h5: dots <= 40'b00111111_00001001_00001001_01001001_01001001;
    4'h6: dots <= 40'b01111111_00001001_00001001_01001001_01001001;
    4'h7: dots <= 40'b01111111_00001001_00001001_01001001_01001001;
    4'h8: dots <= 40'b01111111_00001001_00001001_01001001_01001001;
    4'h9: dots <= 40'b01111111_00001001_00001001_01001001_01001001;
    4'hA: dots <= 40'b01111111_00001001_00001001_01001001_01001001;
    4'hB: dots <= 40'b01111111_00001001_00001001_01001001_01001001;
    4'hC: dots <= 40'b01111111_00001001_00001001_01001001_01001001;
    4'hD: dots <= 40'b01111111_00001001_00001001_01001001_01001001;
    4'hE: dots <= 40'b01111111_00001001_00001001_01001001_01001001;
    4'hF: dots <= 40'b01111111_00001001_00001001_01001001_01001001;
  endcase
endmodule

D.14 encoder.v

`timescale 1ns / 1ps

// Engineer: Evan Broder
//
// Create Date: 21:11:58 12/05/2007
// Module Name: encoder
// Project Name: Video-Conferencing System
// Description: Perform the entire encoding stage on a block of a single channel
//
// Dependencies:
//
// Revision:
// $Id: encoder.v 266 2007-12-15 01:29:37Z evan $
//
// Additional Comments:
module encoder(reset, clock, row, valid_in, x_in, y_in, stream_out, stream_out_we, stream_en, x_out, y_out, dct_out, dct_valid_out, quantizer_out, quantizer_valid_out);

// CHANNEL == 0 => luma
// CHANNEL == 1 => chroma
parameter CHANNEL = 0;

input reset;
input clock;
input [63:0] row;
input valid_in;
input [5:0] x_in;
input [4:0] y_in;
output stream_out;
output stream_out_we;
output stream_en;
output [5:0] x_out;
output [4:0] y_out;
output [127:0] dct_out;
output dct_valid_out;
wire [5:0] dct_x_out;
wire [4:0] dct_y_out;
output [11:0] quantizer_out;
output quantizer_valid_out;
wire [5:0] quantizer_x_out;
wire [4:0] quantizer_y_out;
dct_2d dct_2d(.reset(reset), .clock(clock), .row(row), .valid_in(valid_in), .column(dct_out), .valid_out(dct_valid_out), .x_in(x_in), .y_in(y_in), .x_out(dct_x_out), .y_out(dct_y_out));
quantizer quantizer(.reset(reset), .clock(clock), .column_in(dct_out), .valid_in(dct_valid_out), .value_out(quantizer_out), .valid_out(quantizer_valid_out), .x_in(dct_x_out), .y_in(dct_y_out), .x_out(quantizer_x_out), .y_out(quantizer_y_out));
defparam quantizer.CHANNEL = CHANNEL;
entropy_huffman entropy_huffman(.reset(reset), .clock(clock), .value(quantizer_out), .valid_in(quantizer_valid_out), .stream(stream_out), .we(stream_out_we), .stream_en(stream_en), .x_in(quantizer_x_out), .y_in(quantizer_y_out), .x_out(x_out), .y_out(y_out));
defparam entropy_huffman.CHANNEL = CHANNEL;
endmodule

D.15 entropy.v
module entropy(reset, clock, valid_in, in_value, read_en, out_value, ready);
parameter S_STORE = 0;
parameter S_READ = 1;
parameter S_STANDBY = 2;

input reset;
input clock;
input valid_in;
input [11:0] in_value;
input read_en;
output [11:0] out_value;
output ready;

reg [11:0] block [0:7][0:7];
reg [2:0] x;
reg [2:0] y;
reg direction;
wire valid_in_pulse;
reg [1:0] state;
parameter DOWNLEFT = 0;
parameter UPRIGHT = 1;

reg valid_in_delay;
reg [11:0] in_value_delay;

level_to_pulse valid_in_level(.reset(reset), .clock(clock),
.in_level(in_value), .pulse(valid_in_pulse));

always @(posedge clock) begin
  valid_in_delay <= valid_in;
in_value_delay <= in_value;
  if (reset) begin
    x <= 0;
  end
y <= 0;
state <= S_STANDBY;
direction <= DOWNLEFT;

end

else if (valid_in_pulse & (state == S_STANDBY)) begin
x <= 0;
y <= 0;
state <= S_STORE;
end

else if (valid_in_delay & (state == S_STORE)) begin
    block [x][y] <= in_value_delay;
    if ((y == 7) & (x == 7)) begin
        x <= 0;
y <= 0;
state <= S_READ;
end
else if (y == 7) begin
    x <= x + 1;
y <= 0;
end
else y <= y + 1;
end

if ((state == S_READ) & (x == 7) & (y == 7)) begin
    state <= S_STANDBY;
end

// Huhm, this can probably be optimized for readability...
else if (read_en & (state == S_READ)) begin
    if (y == 7) begin
        if ((x == 0) | (x == 2) | (x == 4) | (x == 6)) begin
            x <= x + 1;
end
        else if ((x == 1) | (x == 3) | (x == 5)) begin
            direction <= UPRIGHT;
x <= x + 1;
y <= y - 1;
end
end
else if (x == 7) begin
    if ((y == 1) | (y == 3) | (y == 5)) begin
        y <= y + 1;
end
    if ((y == 0) | (y == 2) | (y == 4) | (y == 6)) begin
        direction <= DOWNLEFT;
x <= x - 1;
y <= y + 1;
end
end
```verilog
103        end
104        else if (y == 0) begin
105            if ((x == 0) | (x == 2) | (x == 4) | (x == 6)) begin
106                x <= x + 1;
107            end
108        else if ((x == 1) | (x == 3) | (x == 5) | (x == 7)) begin
109                direction <= DOWNLEFT;
110                x <= x - 1;
111                y <= y + 1;
112            end
113        end
114        else if (x == 0) begin
115            if ((y == 1) | (y == 3) | (y == 5)) begin
116                y <= y + 1;
117            end
118        else if ((y == 2) | (y == 4) | (y == 6)) begin
119                direction <= UPRIGHT;
120                x <= x + 1;
121                y <= y - 1;
122        end
123        end
124        else if (direction == DOWNLEFT) begin
125            x <= x - 1;
126            y <= y + 1;
127        end
128        else
129            begin
130                x <= x + 1;
131                y <= y - 1;
132            end
133        end
134    end
135
136    assign out_value = block [x][y];
137    assign ready = (state == S_READ);
138 endmodule

D.16 entropy_huffman.v
```

```verilog
1 `timescale ins / 1ps
2
3 // Engineer: Chris Post
4 // Create Date: 19:44:53 11/30/2007
5 // Module Name: entropy_huffman
6 // Project Name: Video-Conferencing System
7 // Description:
8 // Dependencies:
9 //
10 //
11 //
```
module entropy_huffman(reset, clock, valid_in, x_in, y_in, value, x_out, y_out, stream, we, stream_en);

parameter CHANNEL = 0;

input reset;
input clock;
input valid_in;
input [5:0] x_in;
input [4:0] y_in;
inpu[11:0] value;
output [5:0] x_out;
output [4:0] y_out;
output stream;
output we;
output stream_en;

wire ent_read_en;
wire [11:0] ent_out_value;
wire ent_rdy;
wire serial_rdy;
wire serial_en;
wire [15:0] code;
wire [4:0] code_size;
wire [11:0] huff_out_value;
wire [3:0] huff_out_value_size;
wire huff_done;

cordinate_delay coord_delay(.reset(reset), .clock(clock),
   .valid_in(valid_in), .x_in(x_in), .y_in(y_in),
   .valid_out(stream_en), .x_out(x_out), .y_out(y_out));

entropy entropy_coder(.reset(reset), .clock(clock), .valid_in(valid_in),
   .in_value(value), .read_en(ent_read_en), .out_value(ent_out_value),
   .ready(ent_rdy));

huffman huffman_coder(.reset(reset), .clock(clock), .value_in(ent_out_value),
   .ent_rdy(ent_rdy), .ent_read(ent_read_en), .serial_rdy(serial_rdy),
   .out_en(serial_en), .code(code), .code_size(code_size),
   .value_out(huff_out_value), .value_size(huff_out_value_size),
   .huff_done(huff_done));
defparam huffman_coder.CHANNEL = CHANNEL;

huffman_serializer serializer(.reset(reset), .clock(clock),
   .enable(serial_en), .ready(serial_rdy), .code(code),
63   .code_size(code_size), .value(huff_out_value),
64   .value_size(huff_out_value_size), .stream(stream));

66   assign we = ~serial_rdy;
67   assign stream_en = ~huff_done | ~serial_rdy;
69
70 endmodule

D.17  huffman.v

1 'timescale 1ns / 1ps
2 //]+$Id: huffman.v 202 2007-12-09 04:07:27Z evan $

3 // Engineer:  Chris Post
4 //
5 // Create Date: 21:34:00 12/03/2007
6 // Module Name: huffman
7 // Project Name: Video-Conferencing System
8 // Description:
9 //
10 // Dependencies:
11 //
12 // Revision:
13 // $Id: huffman.v 202 2007-12-09 04:07:27Z evan $
14 // Additional Comments:
15 //
16 //+$Id: huffman.v 202 2007-12-09 04:07:27Z evan $
17
18 module huffman(reset, clock, value_in, ent_rdy, ent_read, serial_rdy, out_en,
19       code, code_size, value_out, value_size, huff_done);
20
21   parameter CHANNEL = 0;
22
23   input reset;
24   input clock;
25   input signed [11:0] value_in;
26   input ent_rdy;
27   output reg ent_read;
28   input serial_rdy;
29   output reg out_en;
30   output [15:0] code;
31   output [4:0] code_size;
32   output reg [11:0] value_out;
33   output reg [3:0] value_size;
34   output reg huff_done;
35
36   // Placeholders for delays
37   reg [11:0] value_out_PD;
38   reg [3:0] value_size_PD;
39   reg out_en_PD;
40   reg [6:0] cur_val;
reg ent_read_D;
reg ent_read_DD;
reg huff_done_D;

reg [3:0] run;
reg [3:0] size;
reg [1:0] zrl;

reg [8:0] code_addr;
wire [19:0] code_data;
assign code_size = code_data[19:16] + 1;
assign code = code_data[15:0];

reg [11:0] value_in_latch;
reg [11:0] value_cor;
wire [3:0] category;

parameter DC_OFFSET = 0;
parameter AC_OFFSET = 16;

generate
  if (CHANNEL == 0) begin: luma
    luma_huffman_code huffman_code(.addr(code_addr), .clk(clock),
      .dout(code_data));
  end
  else begin: chroma
    chroma_huffman_code huffman_code(.addr(code_addr), .clk(clock),
      .dout(code_data));
  end
endgenerate

huffman_categorizer categorizer(.reset(reset), .clock(clock),
  .value(value_cor), .category(category));

always @(posedge clock) begin
if (reset) begin
  ent_read <= 0;
  out_en <= 0;
  out_en_PD <= 0;
  value_out <= 0;
  value_out_PD <= 0;
  cur_val <= 0;
  run <= 0;
  size <= 0;
  zrl <= 0;
  huff_done <= 1;
  huff_done_D <= 1;
  code_addr <= 0;
  ent_read_D <= 0;
  ent_read_DD <= 0;
  value_in_latch <= 0;
end
value_cor <= 0;

end

else begin
    // Do the delays
    value_out <= value_out_PD;
    value_size <= value_size_PD;
    out_en <= out_en_PD;
    ent_read_D <= ent_read;
    ent_read_DD <= ent_read_D;
    value_in_latch <= value_in;
    huff_done_D <= huff_done;

    if (ent_rdy & huff_done) begin
        cur_val <= 0;
        run <= 0;
        size <= 0;
        zrl <= 0;
        huff_done <= 0;
        ent_read <= 0;
        out_en_PD <= 0;
    end

    else if (~huff_done & ~huff_done_D & serial_rdy & ~out_en_PD & ~out_en & ~ent_read & ~ent_read_D & ~ent_read_DD) begin
        // Handle the DC component
        if (cur_val == 0) begin
            code_addr <= DC_OFFSET + {4'h0, category[3:0]};
            value_out_PD <= value_cor;
            value_size_PD <= category;
            out_en_PD <= 1;
            ent_read <= 1;
            cur_val <= cur_val + 1;
        end

        else if (cur_val == 64) huff_done <= 1;
    end

    // Handle AC components
    else begin
        if (category == 0) begin
            cur_val <= cur_val + 1;
        end

        if (cur_val == 63) begin
            code_addr <= AC_OFFSET + 8'h00;
            value_out_PD <= 0;
            value_size_PD <= 0;
            out_en_PD <= 1;
        end
    end

else begin
out_en_PD <= 0;
ent_read <= 1;

if (run == 15) begin
  zrl <= zrl + 1;
  run <= 0;
end
else run <= run + 1;
end

else begin
  if (zrl == 0) begin
    cur_val <= cur_val + 1;
    code_addr <= AC_OFFSET + {run[3:0], category[3:0]};
    value_out_PD <= value_cor;
    value_size_PD <= category;
    out_en_PD <= 1;
    ent_read <= 1;
  end
end

else begin
  code_addr <= AC_OFFSET + 8'hF0;
  value_out_PD <= 0;
  value_size_PD <= 0;
  out_en_PD <= 1;
  ent_read <= 0;
  zrl <= zrl - 1;
end

end
end

else begin
  ent_read <= 0;
  out_en_PD <= 0;
end

dendmodule

D.18  huffman_categorizer.v

`timescale ins / 1ps

// *******************************************************************************
// Engineer:     Chris Post
// Module Name: huffman_categorizer
// Project Name: Video-Conferencing System

56
// Description: Given an input, finds the "category" - i.e. the length - of
the input
// Dependencies:
// Revision:
// $Id: huffman_categorizer.v 181 2007-12-07 11:11:01Z evan $
// Additional Comments:

module huffman_categorizer(reset, clock, value, category);

input reset;
input clock;
input signed [11:0] value;
output reg [3:0] category;
wire a0, a1, a2, a3, a4, a5, a6, a7, a8, a9, a10;

assign a0 = value[11] ^ value[10];
assign a1 = value[11] ^ value[9];
assign a2 = value[11] ^ value[8];
assign a3 = value[11] ^ value[7];
assign a4 = value[11] ^ value[6];
assign a5 = value[11] ^ value[5];
assign a6 = value[11] ^ value[4];
assign a7 = value[11] ^ value[3];
assign a8 = value[11] ^ value[2];
assign a9 = value[11] ^ value[1];
assign a10 = value[11] ^ value[0];

always @*
begina
  if (a0) category = 11;
else if (a1) category = 10;
else if (a2) category = 9;
else if (a3) category = 8;
else if (a4) category = 7;
else if (a5) category = 6;
else if (a6) category = 5;
else if (a7) category = 4;
else if (a8) category = 3;
else if (a9) category = 2;
else if (a10) category = 1;
else category = 0;
end
endmodule

D.19  huffman_serializer.v

'timescale ins / 1ps
module huffman_serializer(reset, clock, enable, ready, code, code_size, value, value_size, stream);

parameter S_WAIT = 0;
parameter S_CODE = 1;
parameter S_VALUE = 2;

input reset;
input clock;
input enable;
output ready;
input [15:0] code;
input [4:0] code_size;
input [11:0] value;
input [3:0] value_size;
output stream;

reg [4:0] code_pos;
reg [3:0] value_pos;
reg [1:0] state;

always @(posedge clock) begin
  if (reset) begin
    state <= S_WAIT;
  end
  else begin
    case (state)
      S_WAIT: begin
        if (enable) begin
          state <= S_CODE;
          code_pos <= code_size - 1;
        end
      end
      default: begin
      end
    endcase
  end
S_CODE: begin
    if (code_pos == 0)
        begin
            if (value_size == 0) state <= S_WAIT;
            else begin
                state <= S_VALUE;
                value_pos <= value_size - 1;
            end
        end
    code_pos <= code_pos - 1;
end
S_VALUE: begin
    if (value_pos == 0) state <= S_WAIT;
    value_pos <= value_pos - 1;
endcase
end

assign stream = (state == S_WAIT) ? 0 : (state == S_CODE) ? code[code_pos] : value[value_pos];
assign ready = (state == S_WAIT);
endmodule

D.20 idct_1d.v

'timescale ins / 1ps

// Engineer: Evan Broder

// Create Date: 12:38:13 11/24/2007
// Module Name: idct_1d
// Project Name: Video-Conferencing System
// Description: Finds a one-dimensional idCT based on the algorithm outlined in Arai:1988
// Dependencies:
// Revision:
// $Id: idct_1d.v 113 2007-12-01 07:45:32Z evan $
// Additional Comments:

module idct_1d(reset, clock, idct_in, idct_out);
    parameter WIDTH = 8;
    // (1 / cos(4 * pi / 16)) << 15
    parameter n1 = 46341;
    // (1 / cos(6 * pi / 16)) << 15
    parameter n2 = 85627;
26    // (1 / \cos(2 \cdot \pi / 16)) << 15
27    parameter n3 = 35468;
28    // (1 / (\cos(2 \cdot \pi / 16) + \cos(6 \cdot \pi / 16))) << 15
29    parameter n4 = 25080;
30
31    input reset;
32    input clock;
33    input [(WIDTH + 4) * 8 - 1:0] idct_in;
34    output [WIDTH * 8 - 1:0] idct_out;
35
36    wire signed [WIDTH + 3:0] a[0:7];
37    reg signed [WIDTH + 2:0] b[0:7];
38    reg signed [WIDTH + 1:0] c[0:8];
39    reg signed [WIDTH + 1:0] d[0:8];
40    reg signed [WIDTH:0] e[0:7];
41    reg signed [WIDTH - 1:0] f[0:7];
42    reg signed [WIDTH - 1:0] S[0:7];
43
44    // Split up the really wide bus into an array of 8 12-bit values
45    genvar i;
46    generate for (i = 0; i < 8; i = i + 1)
47        begin:A
48            //assign a[i] = row[WIDTH * (7 - i) +: WIDTH];
49            assign a[i] =
50                idct_in[((WIDTH + 4) * (8 - i)) - 1 : (WIDTH + 4) * (7 - i)];
51        end
52    endgenerate
53
54    // Take the final answer and concatenate it back into a giant array
55    assign idct_out = {S[0], S[1], S[2], S[3], S[4], S[5], S[6], S[7]};
56
57    always @(posedge clock)
58    begin
59        b[0] <= a[0];
60        b[1] <= a[4];
61        b[2] <= a[2];
62        b[3] <= a[6];
64        b[5] <= a[1] + a[7];
67
68        c[0] <= b[0];
69        c[1] <= b[1];
72        c[4] <= b[4];
74        c[6] <= b[6];
D.21 idct_2d.v

`timescale 1ns / 1ps

// ///////////////////////////////////////////////////////////////////////////////
// // Engineer:    Evan Broder
// // Create Date: 01:31:00 11/25/2007
// // Module Name: idct_2d
// // Project Name: Video-Conferencing System
// // Description: Finds a two-dimensional iDCT by taking a 1-D iDCT on each
// // column, transposing it, then taking the 1-D iDCT on each row
module idct_2d(reset, clock, column, valid_in, row, valid_out, x_in, y_in, x_out, y_out);
  input reset;
  input clock;
  input [95:0] column;
  input valid_in;
  output [63:0] row;
  output valid_out;
  input [5:0] x_in;
  input [4:0] y_in;
  output [5:0] x_out;
  output [4:0] y_out;

  wire [95:0] idct_in;
  wire [63:0] idct_out;
  wire [63:0] transpose_out;
  wire [95:0] transpose_extended;

  wire shift_row;
  wire shift_column;

  assign idct_in = shift_column ? transpose_extended : column;

  delay shift_row_delay(.clock(clock), .undelayed(valid_in),
    .delayed(shift_row));
  defparam shift_row_delay.DELAY = 6;

  delay shift_column_delay(.clock(clock), .undelayed(shift_row),
    .delayed(shift_column));
  defparam shift_column_delay.DELAY = 8;

  delay valid_out_delay(.clock(clock), .undelayed(shift_column),
    .delayed(valid_out));
  defparam valid_out_delay.DELAY = 6;

  coordinate_delay coord_delay(.reset(reset), .clock(clock),
    .valid_in(valid_in), .x_in(x_in), .y_in(y_in),
    .valid_out(valid_out), .x_out(x_out), .y_out(y_out));

  // idCT takes an array of 12-bit values and outputs 8-bit values
idct_1d idct_1d(.reset(reset), .clock(clock), .idct_in(idct_in),
    .idct_out(idct_out));

matrix_transpose transpose(.reset(reset), .clock(clock), .row(idct_out),
    .column(transpose_out), .shift_row(shift_row),
    .shift_column(shift_column));
defparam transpose.WIDTH = 8;

// We have to make the output of the first round longer so it fits back in
// to the iDCT for the second go
array_sign_extender sign_extender(.reset(reset), .clock(clock),
    .little(transpose_out), .big(transpose_extended));
defparam sign_extender.LITTLE_WIDTH = 8;
defparam sign_extender.BIG_WIDTH = 12;

// And last but not least, add 128 to put the data back in [0,255]
level_shifter shifter(.reset(reset), .clock(clock), .row(idct_out),
    .row_shifted(row));

D.22  labkit.v

`default_nettype none

`
// 2) Renamed "mousedata" to "mouse_data"
// 3) Renamed some ZBT memory signals. Parity bits are now incorporated into
// the data bus, and the byte write enables have been combined into the
// 4-bit ram#_bwe_b bus.
// 4) Removed the "systemace_clock" net, since the SystemACE clock is now
// hardwired on the PCB to the oscillator.

//=== Complete change history (including bug fixes) ===
// 2006-Mar-08: Corrected default assignments to "vga_out_red", "vga_out_green"
// and "vga_out_blue". (Was 10'h0, now 8'h0.)
// 2005-Sep-09: Added missing default assignments to "ac97_sdata_out",
// "disp_data_out", "analyzer[2-3]_clock" and
// "analyzer[2-3]_data".
// 2005-Jan-23: Reduced flash address bus to 24 bits, to match 128Mb devices
// actually populated on the boards. (The boards support up to
// 256Mb devices, with 25 address lines.)
// 2004-Oct-31: Adapted to new revision 004 board.
// 2004-May-01: Changed "disp_data_in" to be an output, and gave it a default
// value. (Previous versions of this file declared this port to
// be an input.)
// 2004-Apr-29: Reduced SRAM address busses to 19 bits, to match 18Mb devices
// actually populated on the boards. (The boards support up to
// 72Mb devices, with 21 address lines.)
// 2004-Apr-29: Change history started

module labkit (beep, audio_reset_b, ac97_sdata_out, ac97_sdata_in, ac97_synch,
  ac97_bit_clock,
  vga_out_red, vga_out_green, vga_out_blue, vga_out_sync_b,
  vga_out_blank_b, vga_out_pixel_clock, vga_out_hsync,
  vga_out_vsync,
  tv_out_ycrcb, tv_out_reset_b, tv_out_clock, tv_out_i2c_clock,
  tv_out_i2c_data, tv_out_pal_ntsc, tv_out_hsync_b,
  tv_out_v sync_b, tv_out_blank_b, tv_out_subcar_reset,
  tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1,
  tv_in_line_clock2, tv_in_aef, tv_in_hff, tv_in_aff,
  tv_in_i2c_clock, tv_in_i2c_data, tv_in_fifo_read,
  tv_in_fifo_clock, tv_in_iso, tv_in_reset_b, tv_in_clock,
ram0_data, ram0_address, ram0_adv_ld, ram0_clk, ram0_cen_b, 
ram0_ce_b, ram0_oe_b, ram0_we_b, ram0_bwe_b,

ram1_data, ram1_address, ram1_adv_ld, ram1_clk, ram1_cen_b, 
ram1_ce_b, ram1_oe_b, ram1_we_b, ram1_bwe_b,
clock_feedback_out, clock_feedback_in,
flash_data, flash_address, flash_ce_b, flash_oe_b, flash_we_b, 
flash_reset_b, flash sts, flash_byte_b,
rs232_txd, rs232_rxd, rs232_rts, rs232_cts,
mouse_clock, mouse_data, keyboard_clock, keyboard_data,
clock_27mhz, clock1, clock2,
disp_blank, disp_data_out, disp_clock, disp_rs, disp_ce_b, 
disp_reset_b, disp_data_in,
button0, button1, button2, button3, button_enter, button_right, 
button_left, button_down, button_up,
switch,
led,
user1, user2, user3, user4,
daughtercard,
systemace_data, systemace_address, systemace_ce_b, 
systemace_we_b, systemace_oe_b, systemace_irq, systemace_mpbrdy,
analyzer1_data, analyzer1_clock, 
analyzer2_data, analyzer2_clock, 
analyzer3_data, analyzer3_clock, 
analyzer4_data, analyzer4_clock);

output beep, audio_reset_b, ac97_synch, ac97_sdata_out; 
input ac97_bit_clock, ac97_sdata_in;

output [7:0] vga_out_red, vga_out_green, vga_out_blue; 
output vga_out_sync_b, vga_out_blank_b, vga_out_pixel_clock, 
vga_out_hsync, vga_out_vsync;

output [9:0] tv_out_ycrcb; 
output tv_out_reset_b, tv_out_clock, tv_out_i2c_clock, tv_out_i2c_data, 
tv_out_pal_ntsc, tv_out_hsync_b, tv_out_vsync_b, tv_out_blank_b, 
tv_out_subcar_reset;
input [19:0] tv_in_ycrcb;
input tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2, tv_in_aef,
tv_in_hff, tv_in_aff;
output tv_in_i2c_clock, tv_in_fifo_read, tv_in_fifo_clock, tv_in_iso,
tv_in_reset_b, tv_in_clock;
inout tv_in_i2c_data;
inout [35:0] ram0_data;
output [18:0] ram0_address;
output ram0_adv_ld, ram0_clk, ram0_cen_b, ram0_ce_b, ram0_oe_b, ram0_we_b;
output [3:0] ram0_bwe_b;
inout [35:0] ram1_data;
output [18:0] ram1_address;
output ram1_adv_ld, ram1_clk, ram1_cen_b, ram1_ce_b, ram1_oe_b, ram1_we_b;
output [3:0] ram1_bwe_b;
inout clock_feedback_in;
output clock_feedback_out;
inout [15:0] flash_data;
output [23:0] flash_address;
output flash_ce_b, flash_oe_b, flash_we_b, flash_reset_b, flash_byte_b;
input flash_sts;
output rs232_txd, rs232_rts;
inout rs232_rxd, rs232_csr;
inout mouse_clock, mouse_data, keyboard_clock, keyboard_data;
inout clock_27mhz, clock1, clock2;
output disp_blank, disp_clock, disp_rs, disp_ce_b, disp_reset_b;
input disp_data_in;
output disp_data_out;
inout button0, button1, button2, button3, button_enter, button_right,
button_left, button_down, button_up;
inout [7:0] switch;
output [7:0] led;
inout [31:0] user1, user2, user3, user4;
inout [43:0] daughtercard;
inout [15:0] systemace_data;
output [6:0] systemace_address;
output systemace_ce_b, systemace_we_b, systemace_oe_b;
inout systemace_irq, systemace_mpbrdy;
output [15:0] analyzer1_data, analyzer2_data, analyzer3_data,
analyzer4_data;
output analyzer1_clock, analyzer2_clock, analyzer3_clock, analyzer4_clock;

////////////////////////////////////////////////////////////////////////
//
// I/O Assignments
//
////////////////////////////////////////////////////////////////////////

// Audio Input and Output
assign beep = 1'b0;
assign audio_reset_b = 1'b0;
assign ac97_synch = 1'b0;
assign ac97_sdata_out = 1'b0;
// ac97_sdata_in is an input

// VGA Output
assign vga_out_red = 8'h0;
assign vga_out_green = 8'h0;
assign vga_out_blue = 8'h0;
assign vga_out_sync_b = 1'b1;
assign vga_out_blank_b = 1'b1;
assign vga_out_pixel_clock = 1'b0;
assign vga_out_hsync = 1'b0;
assign vga_out_vsync = 1'b0;

// Video Output
assign tv_out_ycrcb = 10'h0;
assign tv_out_reset_b = 1'b0;
assign tv_out_clock = 1'b0;
assign tv_out_i2c_clock = 1'b0;
assign tv_out_i2c_data = 1'b0;
assign tv_out_pal_ntsc = 1'b0;
assign tv_out_hsync_b = 1'b1;
assign tv_out_vsync_b = 1'b1;
assign tv_out_blank_b = 1'b1;
assign tv_out_subcar_reset = 1'b0;

// Video Input
assign tv_in_i2c_clock = 1'b0;
assign tv_in_fifo_read = 1'b0;
assign tv_in_fifo_clock = 1'b0;
assign tv_in_iso = 1'b0;
assign tv_in_reset_b = 1'b0;
assign tv_in_clock = 1'b0;
assign tv_in_i2c_data = 1'bZ;
// tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2,
// tv_in_aef, tv_in_hff, and tv_in_aff are inputs

// SRAMs
assign ram0_data = 36'hZ;
assign ram0_address = 19'h0;
assign ram0_adv_ld = 1'b0;
assign ram0_clk = 1'b0;
assign ram0_cen_b = 1'b1;
assign ram0_ce_b = 1'b1;
assign ram0_oe_b = 1'b1;
assign ram0_we_b = 1'b1;
assign ram0_bwe_b = 4'hF;
assign ram1_data = 36'hZ;
assign ram1_address = 19'h0;
assign ram1_adv_ld = 1'b0;
assign ram1_clk = 1'b0;
assign ram1_cen_b = 1'b1;
assign ram1_ce_b = 1'b1;
assign ram1_oe_b = 1'b1;
assign ram1_we_b = 1'b1;
assign ram1_bwe_b = 4'hF;
assign clock_feedback_out = 1'b0;

// clock_feedback_in is an input

assign flash_data = 16'hZ;
assign flash_address = 24'h0;
assign flash_ce_b = 1'b1;
assign flash_oe_b = 1'b1;
assign flash_we_b = 1'b1;
assign flash_reset_b = 1'b0;
assign flash_byte_b = 1'b1;

// flash_sts is an input

assign rs232_txd = 1'b1;
assign rs232_rts = 1'b1;

// rs232_rxd and rs232_cts are inputs

assign user1 = 32'hZ;
assign user2 = 32'hZ;
assign user3 = 32'hZ;
assign user4 = 32'hZ;

// Daughtercard Connectors
assign daughtercard = 44'hZ;

// SystemACE Microprocessor Port
assign systemace_data = 16'hZ;
assign systemace_address = 7'h0;
assign systemace_ce_b = 1'b1;
assign systemace_we_b = 1'b1;
assign systemace_oe_b = 1'b1;
// systemace_irq and systemace_mpbrdy are inputs

wire [63:0] row_sprite;
wire [63:0] row_spew;
wire [5:0] x_in;
wire [4:0] y_in;
wire [127:0] dct_out;
wire dct_valid_out;
wire [11:0] quantizer_out;
wire quantizer_valid_out;
wire [11:0] row_out;
wire valid_out;
wire [5:0] x_out;
wire [4:0] y_out;
wire de_stream_out;
wire de_stream_valid;
wire de_eh_ready;
wire [5:0] de_x_in;
wire [4:0] de_y_in;
wire [63:0] row_out;
wire valid_out;
wire [5:0] de_x_out;
wire [4:0] de_y_out;
wire eh_valid_out;
wire [11:0] eh_value_out;
wire [95:0] deq_column_out;
wire deq_valid_out;
wire [63:0] transpose_out;
wire sdo;
wire sdi;
assign user1[0] = sdo;
assign led = {sdi, sdo};

// power-on reset generation
wire power_on_reset; // remain high for first 16 clocks
SRL16 reset_sr (.D(1'b0), .CLK(clock_27mhz), .Q(power_on_reset), .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));
defparam reset_sr.INIT = 16'hFFFF;

// UP button is user reset
wire reset, user_reset;
debounce db1(power_on_reset, clock_27mhz, ~button_up, user_reset);
assign reset = user_reset | power_on_reset;

// UP and DOWN buttons for pong paddle
wire send, shift, send_p, shift_pulse;
debounce db9(reset, clock_27mhz, ~button_enter, send);
debounce db10(reset, clock_27mhz, ~button3, shift);
level_to_pulse send_ltp(reset, clock_27mhz, send, send_p);
level_to_pulse shift_level(reset, clock_27mhz, shift, shift_pulse);
wire [63:0] row_in = row_spew;
wire valid_in = valid_in_spew;
data_spew spew(.reset(reset), .clock(clock_27mhz), .data_set(switch[2:0]), .start(send_p), .row(row_in), .valid_out(valid_in));
encoder encoder(.reset(reset), .clock(clock_27mhz), .row(row_in), .valid_in(valid_in), .x_in(x_in), .y_in(y_in), .stream_out(stream_out), .stream_out_we(stream_out_we), .stream_en(stream_en), .x_out(x_out), .y_out(y_out), .dct_out(dct_out), .dct_valid_out(dct_valid_out), .quantizer_out(quantizer_out), .quantizer_valid_out(quantizer_valid_out));
packer packer(.reset(reset), .clock(clock_27mhz), .cr_data(stream_out), .cr_we(stream_out_we), .cr_stren(stream_en), .cr_x(x_out), .cr_y(y_out), .sdo(sdo));
unpacker unpacker(.reset(reset), .clock(clock_27mhz), .sdi(sdi), .cr_data(de_stream_out), .cr_valid(de_stream_valid), .cr_ready(de_eh_ready), .cr_x(de_x_in), .cr_y(de_y_in));
decoder decoder(.reset(reset), .clock(clock_27mhz), .stream(de_stream_out), .valid_in(de_stream_valid), .ready(de_eh_ready), .x_in(de_x_in), .y_in(de_y_in), .row_out(row_out), .valid_out(valid_out), .x_out(de_x_out), .y_out(de_y_out), .eh_valid_out(eh_valid_out), .eh_value_out(eh_value_out), .deq_column_out(deq_column_out), .deq_valid_out(deq_valid_out));
matrix_transpose transpose(.reset(reset), .clock(clock_27mhz), .row(row_out), .column(transpose_out), .shift_row(valid_out), .shift_column(shift_pulse));
defparam transpose.WIDTH = 8;
display_16hex display(reset, clock_27mhz, transpose_out, disp_blank, disp_clock, disp_rs, disp_ce_b, disp_reset_b, disp_data_out);
assign analyzer1_clock = clock_27mhz;
reg [15:0] analyzer1_data, analyzer2_data, analyzer3_data, analyzer4_data;
reg analyzer2_clock;
assign {analyzer3_clock, analyzer4_clock} = 0;
always @* begin
case (switch[7:5])
  0: begin
    {analyzer1_data, analyzer2_data, analyzer3_data, analyzer4_data} = row_in;
    analyzer2_clock = valid_in;
  end
  1: begin
    analyzer1_data = {dct_out[127:120], dct_out[111:104]};
    analyzer2_data = {dct_out[95:88], dct_out[79:72]};
    analyzer3_data = {dct_out[63:56], dct_out[47:40]};
    analyzer4_data = {dct_out[31:24], dct_out[15:8]};
    analyzer2_clock = dct_valid_out;
  end
  2: begin
    analyzer1_data = {de_stream_out, de_stream_valid, stream_out, stream_out_we, sdo, sdi};
    analyzer2_clock = stream_en;
  end
  3: begin
    analyzer1_data = eh_value_out;
    analyzer2_clock = eh_valid_out;
  end
  4: begin
    analyzer1_data = {deq_column_out[95:88], deq_column_out[83:76]};
    analyzer2_data = {deq_column_out[71:64], deq_column_out[59:52]};
    analyzer3_data = {deq_column_out[47:40], deq_column_out[35:28]};
    analyzer4_data = {deq_column_out[23:16], deq_column_out[11:4]};
    analyzer2_clock = deq_valid_out;
  end
  5: begin
    {analyzer1_data, analyzer2_data, analyzer3_data, analyzer4_data} = row_out;
    analyzer2_clock = valid_out;
  end
endcase
end

D.23  level_shifter.v
\begin{verbatim}
module level_shifter(reset, clock, row, row_shifted);
    parameter WIDTH = 8;

    input reset;
    input clock;
    input [WIDTH * 8 - 1:0] row;
    output [WIDTH * 8 - 1:0] row_shifted;

    genvar i;
    generate for (i = 0; i < WIDTH * 8; i = i + 1)
        begin
            bit_pos
            assign row_shifted[i] = ((i + 1) % WIDTH == 0) ? ~row[i] : row[i];
        end
    endgenerate
endmodule
\end{verbatim}
module level_to_pulse(reset, clock, level, pulse);
  input reset;
  input clock;
  input level;
  output pulse;
  reg r;
  always @(posedge clock)
    if (reset) r <= 1;
    else r <= level;
  assign pulse = level & ~r;
endmodule

D.25 matrix_transpose.v

module matrix_transpose(reset, clock, row, column, shift_row, shift_column);
  parameter WIDTH = 8;
  input reset;
  input clock;
  input [(WIDTH * 8) - 1:0] row;
  output [(WIDTH * 8) - 1:0] column;
  input shift_row, shift_column;
  reg [WIDTH - 1:0] row_matrix [0:7][0:7];
  // The output should be the right column of the matrix
assign column = {row_matrix[0][0], row_matrix[1][0], row_matrix[2][0],
    row_matrix[3][0], row_matrix[4][0], row_matrix[5][0], row_matrix[6][0],
    row_matrix[7][0]};

genvar i, j;

// The bottom row is a special case because when shifting in data, it
// shifts from the external source
// When shifting columns, it should shift to the right
generate for (i = 0; i < 8; i = i + 1)
begin:row_0
    always @(posedge clock)
    if (shift_row)
        row_matrix[7][i] <= row[WIDTH * (7 - i) +: WIDTH];
    else if (shift_column)
        row_matrix[7][i] <= row_matrix[7][(i == 7) ? i : i + 1];
end
dengenerate

// When shift_row is asserted, rows are shifted "up" the matrix
// When shift_column is asserted, columns are shifted right
generate for (i = 0; i < 7; i = i + 1)
begin:other_row
    for (j = 0; j < 8; j = j + 1) begin:column
        always @(posedge clock)
        if (shift_row)
            row_matrix[i][j] <= row_matrix[i + 1][j];
        else if (shift_column)
            row_matrix[i][j] <= row_matrix[i][(j == 7) ? j : j + 1];
    end
end
dengenerate
endmodule

D.26 packer.v

`timescale 1ns / 1ps

// Engineer: Evan Broder
// Create Date: 23:34:13 12/01/2007
// Module Name: packer
// Project Name: Video-Conferencing System
// Description: Checks each stream in turn to see if it's been loaded up. If
// it has, connect it up to the packet_wrapper and send out the
// data
// Dependencies:
// Revision:
// $Id: packer.v 124 2007-12-03 01:19:47Z evan $
module packer(reset, clock, y_a_data, y_a_we, y_a_stren, y_a_x, y_a_y, y_b_data,
y_b_we, y_b_stren, y_b_x, y_b_y, y_c_data, y_c_we, y_c_stren, y_c_x,
y_c_y, y_d_data, y_d_we, y_d_stren, y_d_x, y_d_y, cr_data, cr_we,

  cr_stren, cr_x, cr_y, cb_data, cb_we, cb_stren, cb_x, cb_y, audio_data,

  audio_we, audio_stren, sdo);

  parameter S_Y_A_CHECK = 0;
  parameter S_Y_B_CHECK = 1;
  parameter S_Y_C_CHECK = 2;
  parameter S_Y_D_CHECK = 3;
  parameter S_CR_CHECK = 4;
  parameter S_CB_CHECK = 5;
  parameter S_AUDIO_CHECK = 6;
  parameter S_Y_A_SEND = 7;
  parameter S_Y_B_SEND = 8;
  parameter S_Y_C_SEND = 9;
  parameter S_Y_D_SEND = 10;
  parameter S_CR_SEND = 11;
  parameter S_CB_SEND = 12;
  parameter S_AUDIO_SEND = 13;

  parameter CHAN_Y = 0;
  parameter CHAN_CR = 1;
  parameter CHAN_CB = 2;
  parameter CHAN_AUDIO = 3;

  input reset;
  input clock;
  input y_a_data;
  input y_a_we;
  input y_a_stren;
  input [5:0] y_a_x;
  input [4:0] y_a_y;
  input y_b_data;
  input y_b_we;
  input y_b_stren;
  input [5:0] y_b_x;
  input [4:0] y_b_y;
  input y_c_data;
  input y_c_we;
  input y_c_stren;
  input [5:0] y_c_x;
  input [4:0] y_c_y;
  input y_d_data;
  input y_d_we;
  input y_d_stren;
  input [5:0] y_d_x;
  input [4:0] y_d_y;
input cr_data;
input cr_we;
input cr_stren;
input [5:0] cr_x;
input [4:0] cr_y;
input cb_data;
input cb_we;
input cb_stren;
input [5:0] cb_x;
input [4:0] cb_y;
input audio_data;
input audio_we;
input audio_stren;
output sdo;
reg [3:0] state;
reg start;
wire done;
reg [7:0] mem_data;
wire read;
reg read_ready;
reg [10:0] len;
reg [5:0] x;
reg [4:0] y;
reg [1:0] channel;
reg [5:0] y_a_x_cache, y_b_x_cache, y_c_x_cache, y_d_x_cache, cr_x_cache,
        cb_x_cache;
reg [4:0] y_a_y_cache, y_b_y_cache, y_c_y_cache, y_d_y_cache, cr_y_cache,
        cb_y_cache;
wire [7:0] y_a_out, y_b_out, y_c_out, y_d_out, cr_out, cb_out, audio_out;
wire [10:0] y_a_len, y_b_len, y_c_len, y_d_len, cr_len, cb_len, audio_len;
wire y_a_ready, y_b_ready, y_c_ready, y_d_ready, cr_ready, cb_ready,
        audio_ready;
reg y_a_read, y_b_read, y_c_read, y_d_read, cr_read, cb_read,
        audio_read;
wire y_a_stren_pulse, y_b_stren_pulse, y_c_stren_pulse, y_d_stren_pulse,
        cr_stren_pulse, cb_stren_pulse;

    // Used to latch the coordinates
    level_to_pulse y_a_stren_level(.reset(reset), .clock(clock),
                                .level(y_a_stren), .pulse(y_a_stren_pulse));
    level_to_pulse y_b_stren_level(.reset(reset), .clock(clock),
                                .level(y_b_stren), .pulse(y_b_stren_pulse));
    level_to_pulse y_c_stren_level(.reset(reset), .clock(clock),
                                .level(y_c_stren), .pulse(y_c_stren_pulse));
level_to_pulse y_d_stren_level(.reset(reset), .clock(clock),
    .level(y_d_stren), .pulse(y_d_stren_pulse));
level_to_pulse cr_stren_level(.reset(reset), .clock(clock),
    .level(cr_stren), .pulse(cr_stren_pulse));
level_to_pulse cb_stren_level(.reset(reset), .clock(clock),
    .level(cb_stren), .pulse(cb_stren_pulse));

// Each channel gets recorded into its own FIFO
packer_fifo y_a_fifo(.reset(reset), .clock(clock), .din(y_a_data),
    .we(y_a_we), .stren(y_a_stren), .dout(y_a_out), .read_ready(y_a_ready),
    .len(y_a_len), .read(y_a_read));
packer_fifo y_b_fifo(.reset(reset), .clock(clock), .din(y_b_data),
    .we(y_b_we), .stren(y_b_stren), .dout(y_b_out), .read_ready(y_b_ready),
    .len(y_b_len), .read(y_b_read));
packer_fifo y_c_fifo(.reset(reset), .clock(clock), .din(y_c_data),
    .we(y_c_we), .stren(y_c_stren), .dout(y_c_out), .read_ready(y_c_ready),
    .len(y_c_len), .read(y_c_read));
packer_fifo y_d_fifo(.reset(reset), .clock(clock), .din(y_d_data),
    .we(y_d_we), .stren(y_d_stren), .dout(y_d_out), .read_ready(y_d_ready),
    .len(y_d_len), .read(y_d_read));
packer_fifo cr_fifo(.reset(reset), .clock(clock), .din(cr_data),
    .we(cr_we), .stren(cr_stren), .dout(cr_out), .read_ready(cr_ready),
    .len(cr_len), .read(cr_read));
packer_fifo cb_fifo(.reset(reset), .clock(clock), .din(cb_data),
    .we(cb_we), .stren(cb_stren), .dout(cb_out), .read_ready(cb_ready),
    .len(cb_len), .read(cb_read));
packer_fifo audio_fifo(.reset(reset), .clock(clock), .din(audio_data),
    .we(audio_we), .stren(audio_stren), .dout(audio_out),
    .len(audio_len), .read_ready(audio_ready), .read(audio_read));
packer_wrapper wrapper(.reset(reset), .clock(clock), .start(start),
    .done(done), .mem_data(mem_data), .read(read), .read_ready(read_ready),
    .len(len), .x(x), .y(y), .channel(channel), .sdo(sdo));

always @(posedge clock)
begin
    if (reset)
        begin
            state <= S_Y_A_CHECK;
            start <= 0;
        end
    else
        case (state)
        S_Y_A_CHECK:
            begin
                if (y_a_ready) state <= S_Y_A_SEND;
                else state <= S_Y_B_CHECK;
                start <= 0;
            end
        S_Y_B_CHECK:
            begin


if (y_b_ready) state <= S_Y_B_SEND;
else state <= S_Y_C_CHECK;
start <= 0;
end

S_Y_C_CHECK:
begin
if (y_c_ready) state <= S_Y_C_SEND;
else state <= S_Y_D_CHECK;
start <= 0;
end

S_Y_D_CHECK:
begin
if (y_d_ready) state <= S_Y_D_SEND;
else state <= S_CR_CHECK;
start <= 0;
end

S_CR_CHECK:
begin
if (cr_ready) state <= S_CR_SEND;
else state <= S_CB_CHECK;
start <= 0;
end

S_CB_CHECK:
begin
if (cb_ready) state <= S_CB_SEND;
else state <= S_AUDIO_CHECK;
start <= 0;
end

S_AUDIO_CHECK:
begin
if (audio_ready) state <= S_AUDIO_SEND;
else state <= S_Y_A_CHECK;
start <= 0;
end

S_Y_A_SEND:
begin
if (done & start) state <= S_Y_B_CHECK;
channel <= CHAN_Y;
start <= 1;
end

S_Y_B_SEND:
begin
if (done & start) state <= S_Y_C_CHECK;
channel <= CHAN_Y;
start <= 1;
end

S_Y_C_SEND:
begin
if (done & start) state <= S_Y_D_CHECK;
channel <= CHAN_Y;
start <= 1;
end
S_Y_D_SEND:
begin
  if (done & start) state <= S_CR_CHECK;
  channel <= CHAN_Y;
  start <= 1;
end
S_CR_SEND:
begin
  if (done & start) state <= S_CB_CHECK;
  channel <= CHAN_CR;
  start <= 1;
end
S_CB_SEND:
begin
  if (done & start) state <= S_AUDIO_CHECK;
  channel <= CHAN_CB;
  start <= 1;
end
S_AUDIO_SEND:
begin
  if (done & start) state <= S_Y_A_CHECK;
  channel <= CHAN_AUDIO;
  start <= 1;
endcase
if (y_a_stren_pulse & ~y_a_ready)
begin
  y_a_x_cache <= y_a_x;
  y_a_y_cache <= y_a_y;
end
if (y_b_stren_pulse & ~y_b_ready)
begin
  y_b_x_cache <= y_b_x;
  y_b_y_cache <= y_b_y;
end
if (y_c_stren_pulse & ~y_c_ready)
begin
  y_c_x_cache <= y_c_x;
  y_c_y_cache <= y_c_y;
end
if (y_d_stren_pulse & ~y_d_ready)
begin
  y_d_x_cache <= y_d_x;
  y_d_y_cache <= y_d_y;
end
if (cr_stren_pulse & ~cr_ready)
begin
271     cr_x_cache <= cr_x;  
272     cr_y_cache <= cr_y;  
273 end  
274 if (cb_stren_pulse & ~cb_ready)  
275 begin  
276     cb_x_cache <= cb_x;  
277     cb_y_cache <= cb_y;  
278 end  
279 end  
280  
281 // This is a huge, really ugly, multi-channel, bi-directional MUX.  
282 // But basically, it involves hooking up the ports on the packer_wrapper to  
283 // whichever channel’s ports have data.  
284 always @*  
285 begin  
286     case (state)  
287         S_Y_A_SEND:  
288             begin  
289                 mem_data = y_a_out;  
290                 read_ready = y_a_ready;  
291                 len = y_a_len;  
292                 x = y_a_x_cache;  
293                 y = y_a_y_cache;  
294                 y_a_read = read;  
295             end  
296         S_Y_B_SEND:  
297             begin  
298                 mem_data = y_b_out;  
299                 read_ready = y_b_ready;  
300                 len = y_b_len;  
301                 x = y_b_x_cache;  
302                 y = y_b_y_cache;  
303                 y_b_read = read;  
304             end  
305         S_Y_C_SEND:  
306             begin  
307                 mem_data = y_c_out;  
308                 read_ready = y_c_ready;  
309                 len = y_c_len;  
310                 x = y_c_x_cache;  
311                 y = y_c_y_cache;  
312                 y_c_read = read;  
313             end  
314         S_Y_D_SEND:  
315             begin  
316                 mem_data = y_d_out;  
317                 read_ready = y_d_ready;  
318                 len = y_d_len;  
319                 x = y_d_x_cache;  
320                 y = y_d_y_cache;  
321                 y_d_read = read;  
322             end  
323     end  
324 end  
325
end
S_CR_SEND:

begin
    mem_data = cr_out;
    read_ready = cr_ready;
    len = cr_len;
    x = cr_x_cache;
    y = cr_y_cache;
    cr_read = read;
end

S_CB_SEND:

begin
    mem_data = cb_out;
    read_ready = cb_ready;
    len = cb_len;
    x = cb_x_cache;
    y = cb_y_cache;
    cb_read = read;
end

S_AUDIO_SEND:

begin
    mem_data = audio_out;
    read_ready = audio_ready;
    len = audio_len;
    audio_read = read;
end
endcase
end
endmodule
module packer_fifo(reset, clock, din, stren, we, dout, read_ready, len, read);

parameter S_WRITE = 0;
parameter S_READ = 1;

input reset;
input clock;
input din;
input stren;
input we;
output [7:0] dout;
output read_ready;
output [10:0] len;
input read;

reg [13:0] wptr;
wire [13:0] wptr_inc;
reg [10:0] rptr;
wire [10:0] rptr_inc;

reg state;
wire bram_we;
wire negedge_stren;
wire empty;
wire full;

level_to_pulse we_level(.reset(reset), .clock(clock), .level(~stren), .pulse(negedge_stren));

packer_fifo_memory memory(.clka(clock), .clkb(clock), .dina(din),
.doutb(dout), .wea(bram_we), .addra(wptr), .addrb(rptr));

assign wptr_inc = wptr + 1;
assign rptr_inc = rptr + 1;
assign bram_we = we & (state == S_WRITE);
assign empty = (wptr[13:3] == rptr);
assign full = (wptr_inc == rptr << 3);
assign read_ready = state == S_READ;
assign len = wptr[13:3] - rptr;

always @(posedge clock)
begin
  if (reset)
    begin
      wptr <= 0;
    end
  else
    begin
      if (we)
        begin
          if (state == S_WRITE && wptr[13:3] != rptr)
            begin
              wptr <= wptr + 1;
            end
          end
        end
      else
        begin
          if (state == S_READ && wptr[13:3] <= rptr)
            begin
              wptr <= wptr - 1;
            end
        end
    end
end

assign bram_we = we & (state == S_WRITE);
assign empty = (wptr[13:3] == rptr);
assign full = (wptr_inc == rptr << 3);
assign read_ready = state == S_READ;
assign len = wptr[13:3] - rptr;
module packer_sat(reset, clock, data, tx, txready, sdo);
  parameter DATA_LEN = 8;
  parameter S_IDLE = 0;
  parameter S_SEND = 1;
  parameter S_STOP_1 = 2;
  parameter S_STOP_2 = 3;
endmodule

D.28  packer_sat.v

`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Engineer:   Evan Broder
// Create Date: 14:05:31 12/01/2007
// Module Name: packer_sat
// Project Name: Video-Conferencing System
// Description: SAT: Serial Asynchronous Transmitter - the part of a USART
// that’s implemented by this module
// Dependencies:
// Revision:
// $Id: packer_sat.v 125 2007-12-03 04:17:56Z evan $
// Additional Comments:
////////////////////////////////////////////////////////////////////////////////
input reset;
input clock;
input [DATA_LEN - 1:0] data;
inout tx;
output txready;
output sdo;

reg [1:0] state;
reg [3:0] pos;
reg [DATA_LEN - 1:0] data_cache;
wire clk_enable;

clock_divider divider(.reset(state == S_IDLE), .clock(clock),
.enable(clk_enable));
defparam divider.D = 112;

always @(posedge clock)
begin
  if (reset)
  begin
    pos <= 0;
    state <= S_IDLE;
  end
  else
  begin
    case (state)
      S_IDLE:
      begin
        pos <= 0;
        if (tx)
        begin
          data_cache <= data;
          state <= S_SEND;
        end
      end
      S_SEND:
      if (clk_enable)
      begin
        if (pos == DATA_LEN) state <= S_STOP_1;
        else pos <= pos + 1;
      end
      S_STOP_1: if (clk_enable) state <= S_STOP_2;
      S_STOP_2: if (clk_enable) state <= S_IDLE;
    endcase
  end

assign sdo = (state != S_SEND) ? 1 :
  (pos == 0) ? 0 : data_cache[pos - 1];
assign txready = (state == S_IDLE) & ~tx;
D.29 packer_wrapper.v

module packer_wrapper(reset, clock, start, done, mem_data, read, read_ready, len, x, y, channel, sdo);

parameter S_WAIT = 0;
parameter S_START = 1;
parameter S_CHANNEL = 2;
parameter S_LEN_1 = 3;
parameter S_LEN_2 = 4;
parameter S_COORD_X = 5;
parameter S_COORD_Y = 6;
parameter S_DATA = 7;
parameter S_CRC = 8;

parameter CHAN_Y = 0;
parameter CHAN_CR = 1;
parameter CHAN_CB = 2;
parameter CHAN_AUDIO = 3;

input reset;
input clock;
input start;
output done;
input [7:0] mem_data;
input [10:0] len;
input [5:0] x;
input [4:0] y;
input [1:0] channel;
output reg read;
input read_ready;
output sdo;
reg [3:0] state;
reg tx;
wire txready;
reg [7:0] sat_data;
wire [7:0] crc;
wire start_pulse;

level_to_pulse start_level(.reset(reset), .clock(clock), .level(start),
  .pulse(start_pulse));
packer_sat sat(.reset(reset), .clock(clock), .data(sat_data), .tx(tx),
  .txready(txready), .sdo(sdo));
crc calc_crc(.reset(reset | (state == S_WAIT)), .clock(clock),
  .data(sat_data), .crc(crc), .en(tx & (state == S_DATA)));
assign done = (state == S_WAIT) & ~start_pulse;

always @(posedge clock)
begin
  if (reset)
    begin
      state <= S_WAIT;
      tx <= 0;
    end
  else if (state == S_WAIT & start_pulse) state <= S_START;
    // Every so often, the SAT is ready to transmit again. We should only be
    // moving forward when it is.
  else if (txready & state != S_WAIT)
    begin
      tx <= 1;
      case (state)
        // Most of these states represent one byte in the packet
        S_START:
          begin
            sat_data <= 8'hff;
            state <= S_CHANNEL;
          end
        S_CHANNEL:
          begin
            sat_data <= channel;
            state <= S_LEN_1;
          end
        // Break the length into two bytes because it theoretically
        // could be, and this should support whatever might come out of
        // the FIFO
        S_LEN_1:
          begin
            sat_data <= len >> 8;
            state <= S_LEN_2;
          end
      endcase
    end
  end
end
end
S_LEN_2:
begin
sat_data <= len;
if (channel == CHAN_AUDIO) state <= S_DATA;
else state <= S_COORD_X;
end
S_COORD_X:
begin
sat_data <= x;
state <= S_COORD_Y;
end
S_COORD_Y:
begin
sat_data <= y;
state <= S_DATA;
end
S_DATA:
begin
// If there's still data, spit it out
if (read_ready)
begin
sat_data <= mem_data;
read <= 1;
end
// If there's not, we need to respond before the state
// change to make sure that the CRC gets sent out properly
else
begin
state <= S_WAIT;
sat_data <= crc;
end
endcase
end
else
begin
tx <= 0;
read <= 0;
end
end
endmodule

D.30 quantizer.v

1 'timescale 1ns / 1ps
2 ///////////////////////////////////////////////////////////////////////////
3 // Engineer: Evan Broder
4 //
5 // Create Date: 17:53:46 11/27/2007
6 // Module Name: quantizer

87
module quantizer(reset, clock, column_in, valid_in, value_out, valid_out, x_in, y_in, x_out, y_out);
   parameter CHANNEL = 0;
   input reset;
   input clock;
   input [127:0] column_in;
   input valid_in;
   output reg [11:0] value_out;
   output valid_out;
   input [5:0] x_in;
   input [4:0] y_in;
   output [5:0] x_out;
   output [4:0] y_out;
   wire negedge_valid_in;
   reg [2:0] column;
   reg calculating;
   reg [5:0] addr;
   reg [15:0] dividend;
   wire [11:0] divisor;
   wire [15:0] quotient;
   wire [1:0] fraction;
   reg [15:0] buffer [0:7][0:7];
   level_to_pulse neg_valid_in(.reset(reset), .clock(clock), .level(~valid_in), .pulse(negedge_valid_in));
   delay valid_out_delay(.clock(clock), .undelayed(calculating), .delayed(valid_out));
   defparam valid_out_delay.DELAY = 24;
   coordinate_delay coord_delay(.reset(reset), .clock(clock), .valid_in(valid_in), .x_in(x_in), .y_in(y_in),
      .valid_out(valid_out), .x_out(x_out), .y_out(y_out));
generate if (CHANNEL == 0)
begin: luma
  luma_quantizer_table q_table(.clk(clock), .addr(addr), .dout(divisor));
end
else
begin: chroma
  chroma_quantizer_table q_table(.clk(clock), .addr(addr),
  .dout(divisor));
end
dendgenerate
quantizer_divider divider(.clk(clock), .dividend(dividend),
  .divisor(divisor), .quotient(quotient), .remainder(fraction));
always @(posedge clock)
begin
  if (reset)
  begin
    column <= 0;
    addr <= 0;
    calculating <= 0;
  end
  else if (calculating)
  begin
    addr <= addr + 1;
    dividend <= buffer[addr[5:3]][addr[2:0]];
    if (addr == 63) calculating <= 0;
  end
  else if (valid_in)
  begin
    {buffer[0][column], buffer[1][column], buffer[2][column],
      buffer[3][column], buffer[4][column], buffer[5][column],
      buffer[6][column], buffer[7][column]} <= column_in;
    column <= column + 1;
  end
  else if (negedge_valid_in)
  begin
    calculating <= 1;
  end
  end
  value_out <= quotient[11:0] + fraction[1];
endmodule

D.31  sign_extender.v

'timescale ins / ips

// Engineeer:  Chris Post

//
module sign_extender(reset, clock, value, size, extended_value);
  input reset;
  input clock;
  input [11:0] value;
  input [3:0] size;
  output reg [11:0] extended_value;
  wire [11:0] value_tc;
  assign value_tc = value[size - 1] ? value : value + 1;
  always @*
  case (size)
    0: extended_value = 0;
    1: extended_value = {{11{~value_tc[0]}}, value_tc[0]};
    2: extended_value = {{10{~value_tc[1]}}, value_tc[1:0]};
    3: extended_value = {{9{~value_tc[2]}}, value_tc[2:0]};
    4: extended_value = {{8{~value_tc[3]}}, value_tc[3:0]};
    5: extended_value = {{7{~value_tc[4]}}, value_tc[4:0]};
    6: extended_value = {{6{~value_tc[5]}}, value_tc[5:0]};
    7: extended_value = {{5{~value_tc[6]}}, value_tc[6:0]};
    8: extended_value = {{4{~value_tc[7]}}, value_tc[7:0]};
    9: extended_value = {{3{~value_tc[8]}}, value_tc[8:0]};
    10: extended_value = {{2{~value_tc[9]}}, value_tc[9:0]};
    11: extended_value = {{1{~value_tc[10]}}, value_tc[10:0]};
  endcase
endmodule

D.32 unentropy.v

'timescale 1ns / 1ps

module unentropy(reset, clock, value);
  input reset;
  input clock;
  input [22:0] value;

  always @*
  case (reset, clock)
    1:
      if (reset)
        value[12:0] = 0;
      else
        value = 434713144;
    2: value = 238314231;
  endcase
endmodule
// Dependencies:

// Revision:
// $Id: unentropy.v 224 2007-12-11 01:38:03Z evan $

// Additional Comments:

module unentropy(reset, clock, huffman_proc, valid_in, run, size, value_in,
    valid_out, value_out, processing);

  parameter S_IDLE = 0;
  parameter S_WRITE = 1;
  parameter S_READ = 2;

  input reset;
  input clock;
  input huffman_proc;
  input valid_in;
  input [3:0] run;
  input [3:0] size;
  input [11:0] value_in;
  output valid_out;
  output reg [11:0] value_out;
  output processing;

  reg huffman_proc_D;
  reg [1:0] state;
  reg [63:0] mask;
  reg [6:0] cur_val;

  wire mem_val_WE;
  reg [6:0] mem_val_addr;
  reg [11:0] mem_val_in;
  wire [11:0] mem_val_out;
  reg [6:0] mem_perm_addr;
  wire [11:0] mem_perm_out;
  reg [11:0] perm_addr_D, perm_addr_DD;
  wire [11:0] extended_value;

  wire valid_out_PD;

  delay valid_out_delay (.clock(clock), .undelayed(valid_out_PD),
    .delayed(valid_out));
  defparam valid_out_delay.DELAY = 5;

  delay mem_WE_delay (.clock(clock), .undelayed((state == S_WRITE) & valid_in),
    .delayed(mem_val_WE));
  defparam mem_WE_delay.DELAY = 1;
sign_extender extender(.reset(reset), .clock(clock), .value(value_in),
    .size(size), .extended_value(extended_value));

parameter VALUE_OFFSET = 0;
parameter PERM_OFFSET = 64;

unentropy_mem unentropy_mem(
    .addra(mem_val_addr),
    .addrb(mem_perm_addr),
    .clka(clock),
    .clkb(clock),
    .dina(mem_val_in),
    .douta(mem_val_out),
    .doutb(mem_perm_out),
    .wea(mem_val_WE));

assign valid_out_PD = (state == S_READ);
assign processing = ~(state == S_IDLE);

always @(posedge clock) begin
    huffman_proc_D <= huffman_proc;
    if (reset) begin
        huffman_proc_D <= 0;
        state <= S_IDLE;
        mask <= 0;
        cur_val <= 0;
        mem_val_addr <= 0;
        mem_val_in <= 0;
        mem_perm_addr <= 0;
        value_out <= 0;
    end
    else if (state == S_IDLE) begin
        if (huffman_proc) state <= S_WRITE;
        mask <= 0;
        cur_val <= 0;
    end
    else if (state == S_WRITE) begin
        if (~huffman_proc_D) begin
            state <= S_READ;
            cur_val <= 0;
        end
        else if (valid_in) begin
            if ((cur_val == 64) | ((run == 0) & (size == 0))) begin
                state <= S_READ;
                cur_val <= 0;
            end
        end
    end
module unentropy_huffman(reset, clock, rdy, valid_in, stream_in, x_in, y_in,
   valid_out, value_out, x_out, y_out, entropy_proc);

   input reset;
   input clock;

   else begin
       cur_val <= cur_val + run + 1;
       mask[cur_val + run] <= 1;
       mem_val_addr <= VALUE_OFFSET + cur_val + run;
       if (size == 0) mem_val_in <= 0;
       else mem_val_in <= extended_value;
   end

   else if (state == S_READ) begin
       if (cur_val == 68) state <= S_IDLE;
       cur_val <= cur_val + 1;
       mem_perm_addr <= PERM_OFFSET + cur_val;
       mem_val_addr <= VALUE_OFFSET + mem_perm_out;
       value_out <= mem_val_out;
       value_out <= mask[perm_addr_DD] ? mem_val_out : 0;
       end

       perm_addr_D <= mem_perm_out;
       perm_addr_DD <= perm_addr_D;
       end

   endmodule

D.33 unentropy_huffman.v
output rdy;
input valid_in;
input stream_in;
input [5:0] x_in;
input [4:0] y_in;
output valid_out;
output [11:0] value_out;
output [5:0] x_out;
output [4:0] y_out;
wire huffman_proc;
wire huffman_valid_out;
wire [11:0] huffman_value_out;
output entropy_proc;
reg busy;
wire [3:0] run;
wire [3:0] size;
coordinate_delay coord_delay(.reset(reset), .clock(clock),
        .valid_in(valid_in), .x_in(x_in), .y_in(y_in),
        .valid_out(valid_out), .x_out(x_out), .y_out(y_out));
unhuffman unhuffman(.reset(reset), .clock(clock), .serial_in(stream_in),
        .serial_valid(valid_in), .processing(huffman_proc),
        .valid_out(huffman_valid_out), .run(run), .size(size),
        .value_out(huffman_value_out));
unentropy unentropy(.reset(reset), .clock(clock),
        .huffman_proc(huffman_proc), .valid_in(huffman_valid_out), .run(run),
        .size(size), .value_in(huffman_value_out), .valid_out(valid_out),
        .value_out(value_out), .processing(entropy_proc));
always @(posedge clock) begin
    if (reset) busy <= 0;
    else if (valid_in) busy <= 1;
    else if (~entropy_proc) busy <= 0;
end
assign rdy = ~busy;
endmodule

D.34 unhuffman.v

'timescale 1ns / 1ps
///////////////////////////////////////////////////////////////////////////////////////////////
// Engineer: Chris Post
///////////////////////////////////////////////////////////////////////////////////////////////
// Create Date: 22:26:00 12/05/2007
// Module Name: unhuffman
module unhuffman(reset, clock, serial_in, serial_valid, processing, valid_out, run, size, value_out);

parameter CHANNEL = 0;

input reset;
input clock;
input serial_in;
input serial_valid;
output processing;
output reg valid_out;
output reg [3:0] run;
output reg [3:0] size;
output reg [11:0] value_out;

reg working_internal;
reg DC_proc;
reg no_out;
reg [3:0] no_out_count;
reg [26:0] buffer;
wire [10:0] DC_code;
wire [8:0] high_code_buff;
wire [7:0] low_code_buff;
wire [10:0] VC_value; // Value in the buffer at the valid code checking stage
wire [10:0] DC_value; // Value in the buffer at the DC valid checking stage
reg [26:0] mask_buffer;

assign DC_code = buffer[21:11];
assign high_code_buff = buffer[26:18];
assign low_code_buff = buffer[18:11];
assign VC_value = buffer[12:2];
assign DC_value = buffer[11:1];
wire reset_buf;
reg [3:0] DC_size;
reg [3:0] DC_codesize;
reg [3:0] DC_predict_codesize;
reg [8:0] mem_addr;
wire [11:0] mem_dout;
wire [3:0] mem_run;
wire [3:0] mem_size;
wire [3:0] mem_codesize;

reg [3:0] predict_size;
reg group_a;
reg [3:0] predict_size_D;
reg group_a_D;

assign mem_run = mem_dout[11:8];
assign mem_size = mem_dout[7:4];
assign mem_codesize = mem_dout[3:0];

parameter GROUP_A = 0;
parameter GROUP_B = 256;
parameter GROUP_C = 256 + 128;

// Instantiate a LTP for posedge serial_valid -> reset_buf
level_to_pulse posedge_serial_valid(.reset(reset), .clock(clock),
    .level(serial_valid), .pulse(reset_buf));

// Instantiate the proper memory element
generate
    if (CHANNEL == 0) begin: luma_AC_decode
        luma_unhuffman_code unhuffman_code(.addr(mem_addr), .clk(clock),
            .dout(mem_dout));
    end

    else begin: chroma_AC_decode
        chroma_unhuffman_code unhuffman_code(.addr(mem_addr), .clk(clock),
            .dout(mem_dout));
    end
endgenerate

// Processing goes high when serial data starts coming in, and goes low
// after last code in serial stream is output.
assign processing = serial_valid | working_internal;

always @ (posedge clock) begin
    // Make sure output isn’t valid if we aren’t working
    if (~working_internal) valid_out <= 0;

    // Do some delays
    predict_size_D <= predict_size;
    group_a_D <= group_a;

    if (reset) begin
        working_internal <= 0;
    end
end
DC_proc <= 0;
no_out <= 1;
no_out_count <= 0;
buffer <= 0;
mask_buffer <= 0;
DC_predict_codesize <= 12;
mem_addr <= GROUP_C + 8'hFF; // Code here never valid
predict_size <= 0;
group_a <= 0;
predict_size_D <= 0;
group_a_D <= 0;
valid_out <= 0;
run <= 0;
size <= 0;
value_out <= 0;
end

else if (reset_buf | working_internal) begin
// Wait for the last of the serial data to process, then we're done
if (working_internal & (mask_buffer == 0)) working_internal <= 0;

// Reset the buffer and don't recognize codes until data is aligned
if (reset_buf) begin
buffer <= 0;
buffer[0] <= serial_in;
mask_buffer <= 1;
no_out_count <= 13; // 12, code in position; 2, code out of memory
no_out <= 1;
working_internal <= 1;
DC_proc <= 1;
predict_size <= 0;
group_a <= 0;
end

else begin
// Shift the buffer
buffer <= buffer << 1;
buffer[0] <= (serial_valid) ? serial_in : 0;
mask_buffer <= mask_buffer << 1;
mask_buffer[0] <= serial_valid;

if (DC_proc & (no_out_count == 3)) DC_predict_codesize <= 0;
if (DC_predict_codesize != 12)
DC_predict_codesize <= DC_predict_codesize + 1;

// Always block for DC code lookup generated below
// Lookup the buffer's current potential AC code
case (high_code_buff)
9'b000000000: begin
mem_addr <= GROUP_A + low_code_buff[6:0];
predict_size <= 1;
group_a <= 1;
end
9'b000000001: begin
mem_addr <= GROUP_A + low_code_buff[7:0];
predict_size <= 8 - 1;
group_a <= 1;
end
9'b000000011: begin
mem_addr <= GROUP_B + low_code_buff[3:0];
predict_size <= 9 - 1;
group_a <= 0;
end
9'b000000111: begin
mem_addr <= GROUP_B + low_code_buff[4:0];
predict_size <= 10 - 1;
group_a <= 0;
end
9'b000001111: begin
mem_addr <= GROUP_B + low_code_buff[5:0];
predict_size <= 11 - 1;
group_a <= 0;
end
9'b000011111: begin
mem_addr <= GROUP_B + low_code_buff[6:0];
predict_size <= 12 - 1;
group_a <= 0;
end
9'b000111111: begin
mem_addr <= GROUP_C + low_code_buff[3:0];
predict_size <= 13 - 1;
group_a <= 0;
end
9'b001111111: begin
mem_addr <= GROUP_C + low_code_buff[4:0];
predict_size <= 14 - 1;
group_a <= 0;
end
9'b011111111: begin
mem_addr <= GROUP_C + low_code_buff[5:0];
predict_size <= 15 - 1;
group_a <= 0;
end
9'b111111111: begin
mem_addr <= GROUP_C + low_code_buff[6:0];
predict_size <= 16 - 1;
group_a <= 0;
end
default : begin
mem_addr <= GROUP_C + 8'hFF;
predict_size <= 1;
group_a <= 0;
end
case
// Test for valid code, output if valid
if (~no_out & ((DC_proc & (DC_codesize == DC_predict_codesize)) |
    (~DC_proc & (mem_codesize == predict_size_D)) |
    (~DC_proc & group_a_D & (mem_codesize != 0)))) begin
valid_out <= 1;
DC_proc <= 0;
run <= DC_proc ? 0 : mem_run;
size <= DC_proc ? DC_size : mem_size;
if (DC_proc) begin
  value_out <= DC_value >> (11 - DC_size);
  no_out_count <= DC_size + 2;
  no_out <= 1;
  buffer[26:12] <= 0;
  mask_buffer[26:12] <= 0;
  buffer[12:2] <= buffer[11:1] & (11'b11111111111 >> DC_size);
end
else begin
  value_out <= VC_value >> (11 - mem_size);
  no_out_count <= mem_size + 2;
  no_out <= (mem_size == 0) ? 0 : 1;
  buffer[26:14] <= 0;
  mask_buffer[26:14] <= 0;
  buffer[13:3] <= buffer[12:2] & (11'b11111111111 >> mem_size);
  mask_buffer[13:3] <= mask_buffer[12:2] & (11'b11111111111 >> mem_size);
end
end
// Disable output otherwise
else begin
  // Code invalid, output nothing, run output disable counter
  valid_out <= 0;
  // Run the counter for output disable
  if ((no_out_count == 1) | (no_out_count == 0)) begin
    no_out_count <= 0;
    no_out <= 0;
  end
  else no_out_count <= no_out_count - 1;
end
// Lookup the buffer's current potential DC code

if (CHANNEL == 0) begin: luma_DC_decode
    always @(posedge clock) begin: luma_DC_always
        if (reset) begin
            DC_size <= 0;
            DC_codesize <= 0;
        end
    end
else begin
    case (DC_code)
        11'b00000000000: begin DC_size <= 0; DC_codesize <= 2; end
        11'b00000000001: begin DC_size <= 1; DC_codesize <= 3; end
        11'b00000000010: begin DC_size <= 2; DC_codesize <= 3; end
        11'b00000000100: begin DC_size <= 3; DC_codesize <= 3; end
        11'b00000000101: begin DC_size <= 4; DC_codesize <= 3; end
        11'b00000001110: begin DC_size <= 5; DC_codesize <= 3; end
        11'b00000011110: begin DC_size <= 6; DC_codesize <= 4; end
        11'b00000111110: begin DC_size <= 7; DC_codesize <= 4; end
        11'b00001111110: begin DC_size <= 8; DC_codesize <= 5; end
        11'b00011111110: begin DC_size <= 9; DC_codesize <= 5; end
        11'b00111111110: begin DC_size <= 10; DC_codesize <= 6; end
        11'b01111111110: begin DC_size <= 11; DC_codesize <= 6; end
        default : begin DC_size <= 0; DC_codesize <= 0; end
    endcase
end

else begin: chroma_DC_decode
    always @(posedge clock) begin: chroma_DC_always
        if (reset) begin
            DC_size <= 0;
            DC_codesize <= 0;
        end
    end
else begin
    case (DC_code)
        11'b00000000000: begin DC_size <= 0; DC_codesize <= 2; end
        11'b00000000001: begin DC_size <= 1; DC_codesize <= 2; end
        11'b000000000010: begin DC_size <= 2; DC_codesize <= 2; end
        11'b000000000011: begin DC_size <= 3; DC_codesize <= 3; end
        11'b0000000000110: begin DC_size <= 4; DC_codesize <= 3; end
        11'b0000000000111: begin DC_size <= 5; DC_codesize <= 3; end
        11'b0000000001110: begin DC_size <= 6; DC_codesize <= 4; end
        11'b0000000001111: begin DC_size <= 7; DC_codesize <= 4; end
        11'b0000000011110: begin DC_size <= 8; DC_codesize <= 5; end
        11'b0000000011111: begin DC_size <= 9; DC_codesize <= 5; end
        11'b0000000111110: begin DC_size <= 10; DC_codesize <= 6; end
        11'b0000000111111: begin DC_size <= 11; DC_codesize <= 6; end
        default : begin DC_size <= 0; DC_codesize <= 0; end
    endcase
end
D.35 unpacker.v

`timescale ins / 1ps

///////////////////////////////////////////////////////////////////////////////
//  Engineer:    Evan Broder
//  Create Date:  16:26:31  12/04/2007
//  Module Name:  unpacker
//  Project Name: Video-Conferencing System
//  Description: Receives packets, decodes the packet format, checks the CRC,
//                and passes the data on to one of the FIFOs, which are hooked
//                up to the Huffman decoders
//  Dependencies:
//  Revision:
// $Id: unpacker.v 186 2007-12-07 23:41:55Z evan $
// Additional Comments:
///////////////////////////////////////////////////////////////////////////////

module unpacker(reset, clock, sdi, y_a_data, y_a_valid, y_a_x, y_a_y, y_a_ready,
            y_b_data, y_b_valid, y_b_x, y_b_y, y_b_ready, y_c_data, y_c_valid,
            y_c_x, y_c_y, y_c_ready, y_d_data, y_d_valid, y_d_x, y_d_y, y_d_ready,
            cr_data, cr_valid, cr_x, cr_y, cr_ready, cb_data, cb_valid, cb_x, cb_y,
            cb_ready, audio_data, audio_valid, audio_ready);

  parameter S_WAIT = 0;
  parameter S_STORE = 1;

  parameter CHAN_Y = 0;
  parameter CHAN_CR = 1;
  parameter CHAN_CB = 2;
  parameter CHAN_AUDIO = 3;

  input  reset;
  input  clock;
  input  sdi;
  output y_a_data;
  output y_a_valid;
  output reg [5:0] y_a_x;
  output reg [4:0] y_a_y;
  input  y_a_ready;
  output y_b_data;
output y_b_valid;
output reg [5:0] y_b_x;
output reg [4:0] y_b_y;
input y_b_ready;
output y_c_data;
output y_c_valid;
output reg [5:0] y_c_x;
output reg [4:0] y_c_y;
input y_c_ready;
output y_d_data;
output y_d_valid;
output reg [5:0] y_d_x;
output reg [4:0] y_d_y;
input y_d_ready;
output cr_data;
output cr_valid;
output reg [5:0] cr_x;
output reg [4:0] cr_y;
input cr_ready;
output cb_data;
output cb_valid;
output reg [5:0] cb_x;
output reg [4:0] cb_y;
input cb_ready;
output audio_data;
output audio_valid;
input audio_ready;

reg [7:0] y_a_din, y_b_din, y_c_din, y_d_din, cr_din, cb_din, audio_din;
reg y_a_we, y_b_we, y_c_we, y_d_we, cr_we, cb_we, audio_we;
reg y_a_stren, y_b_stren, y_c_stren, y_d_stren, cr_stren, cb_stren, audio_stren;
reg y_a_clear, y_b_clear, y_c_clear, y_d_clear, cr_clear, cb_clear, audio_clear;

wire [1:0] channel;
wire [5:0] x;
wire [4:0] y;
wire [7:0] data;
wire we;
wire stren;
wire clear;

reg state;
reg [1:0] y_counter;

unpacker_fifo y_a_fifo(.reset(reset | y_a_clear), .clock(clock),
    .din(y_a_din), .we(y_a_we), .stren(y_a_stren), .dout(y_a_data),
    .valid_out(y_a_valid), .ready(y_a_ready));
unpacker_fifo y_b_fifo(.reset(reset | y_b_clear), .clock(clock),
    .din(y_b_din), .we(y_b_we), .stren(y_b_stren), .dout(y_b_data),
    .valid_out(y_b_valid), .ready(y_b_ready));
unpacker_fifo y_c_fifo(.reset(reset | y_c_clear), .clock(clock),
    .din(y_c_din), .we(y_c_we), .stren(y_c_stren), .dout(y_c_data),
    .valid_out(y_c_valid), .ready(y_c_ready));

unpacker_fifo y_d_fifo(.reset(reset | y_d_clear), .clock(clock),
    .din(y_d_din), .we(y_d_we), .stren(y_d_stren), .dout(y_d_data),
    .valid_out(y_d_valid), .ready(y_d_ready));

unpacker_fifo cr_fifo(.reset(reset | cr_clear), .clock(clock),
    .din(cr_din), .we(cr_we), .stren(cr_stren), .dout(cr_data),
    .valid_out(cr_valid), .ready(cr_ready));

unpacker_fifo cb_fifo(.reset(reset | cb_clear), .clock(clock),
    .din(cb_din), .we(cb_we), .stren(cb_stren), .dout(cb_data),
    .valid_out(cb_valid), .ready(cb_ready));

unpacker_fifo audio_fifo(.reset(reset | audio_clear), .clock(clock),
    .din(audio_din), .we(audio_we), .stren(audio_stren), .dout(audio_data),
    .valid_out(audio_valid), .ready(audio_ready));

unpacker_unwrapper unwrapper(.reset(reset), .clock(clock), .sdi(sdi),
    .channel(channel), .x(x), .y(y), .data(data), .we(we), .stren(stren),
    .clear_mem(clear));

always @(posedge clock)
begin
    if (reset)
    begin
        state <= S_WAIT;
        y_counter <= 0;
    end
    else
        case (state)
            S_WAIT: if (stren)
            begin
                state <= S_STORE;
            end
            case (channel)
                CHAN_Y:
                case (y_counter)
                    0:
                    begin
                        y_a_x <= x;
                        y_a_y <= y;
                    end
                    1:
                    begin
                        y_b_x <= x;
                        y_b_y <= y;
                    end
                    2:
                    begin
                        y_c_x <= x;
                        y_c_y <= y;
                    end
        endcase
end
end
endcase
CHAN_CR:
begin
    cr_x <= x;
    cr_y <= y;
end
CHAN_CB:
begin
    cb_x <= x;
    cb_y <= y;
end
endcase
end
S_STORE:
if (~stren)
begin
    state <= S_WAIT;
    if (channel == CHAN_Y) y_counter <= y_counter + 1;
end
endcase
end
always @*
begin
    if (state == S_STORE)
begin
        case (channel)
CHAN_Y:
        case (y_counter)
0:
begin
    y_a_din = data;
    y_a_we = we;
    y_a_stren = stren;
    y_a_clear = clear;
end
1:
begin
    y_b_din = data;
    y_b_we = we;
    y_b_stren = stren;
    y_b_clear = clear;
end
2:
begin
D.36 unpacker_fifo.v

```verilog

`timescale 1ns / 1ps

// _______________________________\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\n```
// There is a one clock cycle delay between when read is asserted
// and when the data comes out dout.

// Dependencies:

// Revision:
$Id: unpacker_fifo.v 144 2007-12-05 00:40:30Z evan $

// Additional Comments:

module unpacker_fifo(reset, clock, din, we, stren, dout, valid_out, ready);

parameter S_WRITE = 0;
parameter S_READ = 1;
parameter S_READ_WAIT = 2;

input reset;
input clock;
input [7:0] din;
input we;
input stren;
output dout;
output valid_out;
input ready;

reg [1:0] state;

reg [10:0] wptr;
wire [10:0] wptr_inc;
reg [13:0] rptr;
wire [13:0] rptr_inc;
wire bram_we;
wire negedge_stren;
wire empty;
wire full;

assign bram_we = we & (state == S_WRITE);
assign wptr_inc = wptr + 1;
assign rptr_inc = rptr + 1;
assign empty = (wptr == rptr[13:3]);
assign full = (wptr_inc == rptr[13:3]);
assign valid_out = state == S_READ;

level_to_pulse we_level(.reset(reset), .clock(clock), .level(~stren),
.pulse(negedge_stren));

unpacker_fifo_memory memory(.clka(clock), .clkb(clock), .dinb(din),

assign bram_we = we & (state == S_WRITE);
assign wptr_inc = wptr + 1;
assign rptr_inc = rptr + 1;
assign empty = (wptr == rptr[13:3]);
assign full = (wptr_inc == rptr[13:3]);
assign valid_out = state == S_READ;

level_to_pulse we_level(.reset(reset), .clock(clock), .level(~stren),
.pulse(negedge_stren));

unpacker_fifo_memory memory(.clka(clock), .clkb(clock), .dinb(din),
always @(posedge clock)
begin
if (reset)
begin
wptr <= 0;
rpbr <= 0;
state <= S_WRITE;
end
else
begin
  case (state)
  S_WRITE:
  if (negedge_stren) state <= S_READ_WAIT;
  else if (we & ~full) wptr <= wptr_inc;
  S_READ_WAIT:
  if (ready)
  begin
    state <= S_READ;
    rptr <= rptr_inc;
  end
  S_READ:
  if (empty) state <= S_WRITE;
  else rptr <= rptr_inc;
  endcase
end
endmodule

D.37 unpacker_sar.v

 timescale ins / 1ps
 // Engineer: Evan Broder
 // Create Date: 20:32:16 12/02/2007
 // Module Name: unpacker_sar
 // Project Name: Video-Conferencing System
 // Description: Asynchronously aligns the clock with the signal and receives
 // serial data.
 // Framing algorithm based on explanation of USART module in AVR
 // ATTiny2313
 // (SAR stands for Serial Asynchronous Receiver)
 // Dependencies:
 // Revision:
 // $Id: unpacker_sar.v 193 2007-12-08 07:50:51Z evan $
module unpacker_sar(reset, clock, sdi, rx, data);

    parameter DATA_LEN = 8;

    parameter S_IDLE = 0;
    parameter S_START = 1;
    parameter S_DATA = 2;

    input reset;
    input clock;
    input sdi;
    output rx;
    output reg [7:0] data;

    wire sdi_debounce;
    wire clk_enable;
    wire rx_level;

    reg [3:0] sample_count;
    reg [4:0] bit_count;
    reg [1:0] state;

    reg [1:0] votes;

    debounce debounce(.reset(reset), .clock(clock), .noisy(sdi), .clean(sdi_debounce));
    defparam debounce.DELAY = 5;

    clock_divider divider(.reset(reset), .clock(clock), .enable(clk_enable));
    defparam divider.D = 7;

    level_to_pulse rx_l2p(.reset(reset), .clock(clock), .level(rx_level), .pulse(rx));

always @(posedge clock)
begin
    if (reset)
    begin
        state <= S_IDLE;
        votes <= 0;
        data <= 0;
    end
    else if (clk_enable)
    begin
        sample_count <= sample_count + 1;

        if (sample_count == 0) votes <= 0;
        else if (sample_count == 6 || sample_count == 7 || sample_count == 8) votes <= votes + sdi;

        case (state)
          S_IDLE:
begin
  if (~sdi_debounce) state <= S_START;
  sample_count <= 1;
  bit_count <= 0;
end
S_START:
begin
if (sample_count == 0)
begin
  if (votes[1] == 0) state <= S_DATA;
  else state <= S_IDLE;
end
endcase
S_DATA:
begin
if (sample_count == 10) data[bit_count] <= votes[1];
else if (sample_count == 0) bit_count <= bit_count + 1;
if (bit_count == DATA_LEN) state <= S_IDLE;
end
end
assign rx_level = (state == S_IDLE);
endmodule

D.38 unpacker_unwrapper.v

`timescale 1ns / 1ps

/////////////////////////////////////////////////////////////////////////////
// Engineer: Evan Broder
// Create Date: 12:08:03 11/18/2007
// Module Name: unpacker_unwrapper
// Project Name: Video-Conferencing System
// Description: Takes a serial incoming line and outputs all of the relevant
data. If the CRC check fails, the clear_mem flag gets asserted
// just before stren is deasserted.
// Dependencies:
// Revision:
// $Id: unpacker_unwrapper.v 218 2007-12-10 21:40:37Z evan $
// Additional Comments:
//
module unpacker_unwrapper(reset, clock, sdi, channel, x, y, data, we, stren,
clear_mem);
parameter S_WAIT = 0;
parameter S_CHAN = 1;
parameter S_LEN_1 = 2;
parameter S_LEN_2 = 3;
parameter S_COORD_X = 4;
parameter S_COORD_Y = 5;
parameter S_DATA = 6;
parameter S_CRC = 7;
parameter CHAN_Y = 0;
parameter CHAN_CR = 1;
parameter CHAN_CB = 2;
parameter CHAN_AUDIO = 3;

input reset;
input clock;
input sdi;
output reg [1:0] channel;
output reg [5:0] x;
output reg [4:0] y;
output [7:0] data;
output reg we;
output stren;
output reg clear_mem;
wire rx;
wire [7:0] sat_data;
wire [7:0] crc;
reg [10:0] len;
reg [2:0] state;
wire stren_undelayed;

unpacker_sar sar(.reset(reset), .clock(clock), .sdi(sdi), .rx(rx),
   .data(sat_data));
crc calc_crc(.reset(reset | (state == S_WAIT)), .clock(clock),
   .data(sat_data), .crc(crc), .en(rx & (state == S_DATA) & (len != 0)));
delay stren_delay(.clock(clock), .undelayed(stren_undelayed),
   .delayed(stren));
defparam stren_delay.DELAY = 1;
always @(posedge clock)
begin
  if (reset) state <= S_WAIT;
  else if ((state == S_WAIT) & rx & (sat_data == 8'hff)) state <= S_CHAN;
  else if (state == S_CRC)
    begin
      state <= S_WAIT;
      clear_mem <= (sat_data != crc);
    end
  else if (rx)
case (state):
  S_CHAN:
    begin
      channel <= sat_data;
      state <= S_LEN_1;
    end
  S_LEN_1:
    begin
      len[10:8] <= sat_data;
      state <= S_LEN_2;
    end
  S_LEN_2:
    begin
      len[7:0] <= sat_data;
      if (channel == CHAN_AUDIO) state <= S_DATA;
      else state <= S_COORD_X;
    end
  S_COORD_X:
    begin
      x <= sat_data;
      state <= S_COORD_Y;
    end
  S_COORD_Y:
    begin
      y <= sat_data;
      state <= S_DATA;
    end
  S_DATA:
    begin
      if (len == 0) state <= S_CRC;
      else
        begin
          we <= 1;
          len <= len - 1;
        end
    endcase
  else
    begin
      we <= 0;
      clear_mem <= 0;
    end
end
assign stren_undelayed = (state == S_DATA) | (state == S_CRC);
assign data = sat_data;
endmodule