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 communicaiton 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

List of Figures

List of Tables

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 8×8 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.

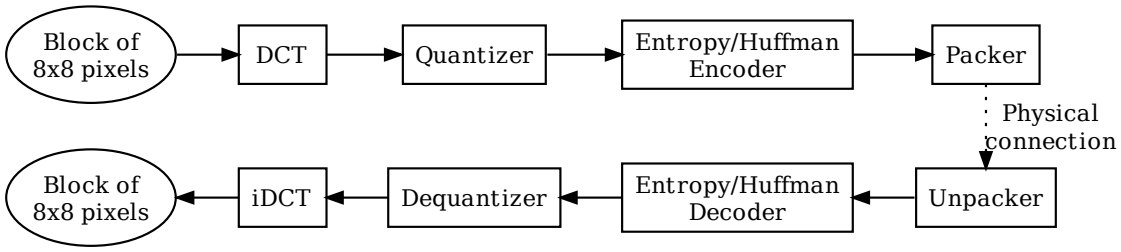


Figure 1: Block diagram for JPEG compression/decompression

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 can be broken up into 2 applications of the 1-D DCT. In practice, this means that if $\vec{y} = T\vec{x}$ represents the 1-D DCT applied to the vector $\vec{x}$, 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, $\vec{x}$ is the input vector, $\vec{y}$ is the output vector, and

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

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]

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

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.

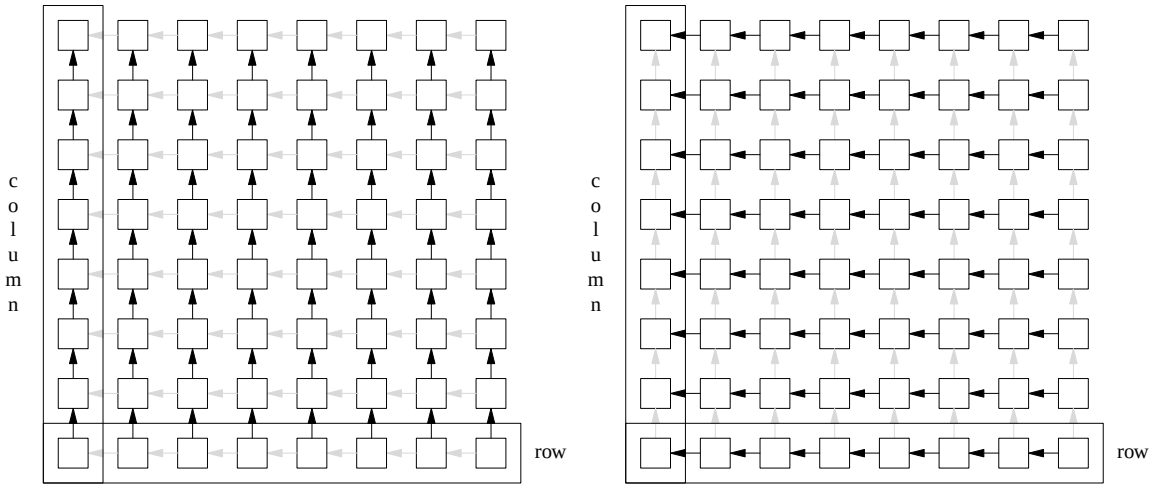
2.1.3 Matrix Transpose (broder)

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.



(a) `shift_row` asserted (b) `shift_column` asserted

Figure 2: Data transfer in the Matrix Transpose module

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

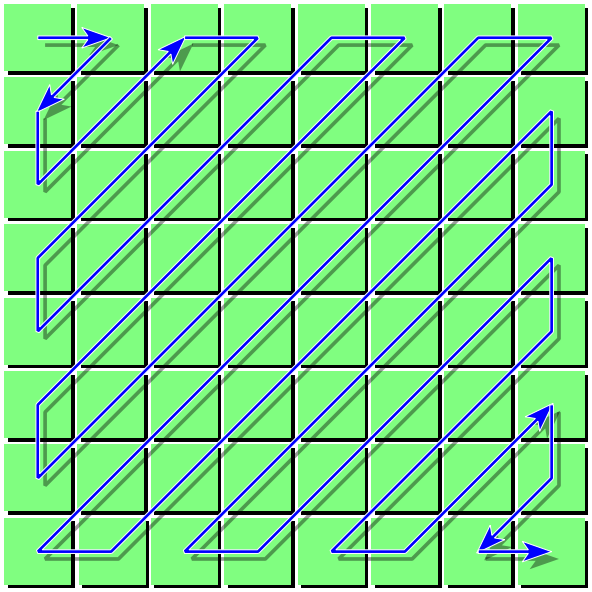


Figure 3: Entropy coder zig-zag ordering [1]

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 quanization 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 combinationally, 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.

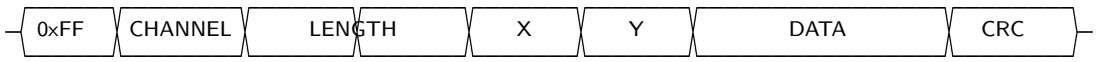


Figure 4: Packet format

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 hs 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.

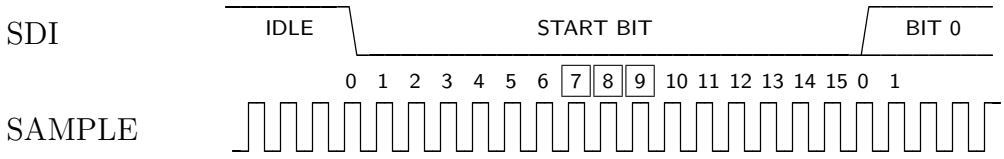


Figure 5: Timing diagram for start bit sampling [2]

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.

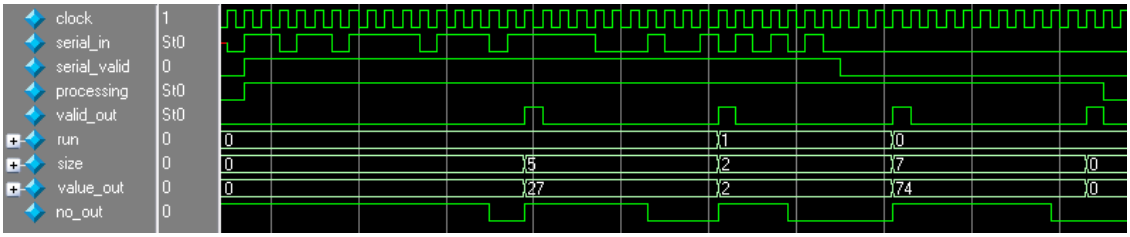


Figure 6: Typical operation of the Huffman Decoder on a small block matrix

Looking up the Huffman codes for DC values is hard coded directly, since there are only

12 codes. 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 rest 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

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

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 propogation speed of the circuit. The algorithm for the iDCT is listed in Table 2

Table 2: 1-D iDCT algorithm [4]

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

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

[1] "JPEG ZigZag.svg — Wikimedia Commons," 2007, accessed 1-November-2007. [Online]. Available: http://commons.wikimedia.org/w/index.php?title=Image:JPEG_ZigZag.svg&oldid=5483305

[2] *ATtiny2313/V: 8-bit AVR Microcontroller with 2K Bytes In-System Programmable Flash*, Atmel Corporation, 2514I-04/06 2006. [Online]. Available: http://www.atmel.com/dyn/resources/prod_documents/doc2543.pdf

[3] L. V. Agostini, R. C. Porto, S. Bampi, and I. S. Silva, "A FPGA based design of a multiplierless and fully pipelined JPEG compressor," *Proceedings of the 8th Euromicro conference on Digital System Design*, pp. 210–213, 2005. [Online]. Available: http://ieeexplore.ieee.org/iel5/10433/33127/01559802.pdf

[4] Y. Arai, T. Agui, and M. Nakajima, "A fast DCT-SQ scheme for images," vol. E71, no. 11, Nov. 1988, pp. 1095–1097.

[5] The International Telegraph and Telephone Consultative Committee (CCITT), "Digital compression and coding of continuous-tone still images," Recommendation T.81, September 1992. [Online]. Available: http://www.w3.org/Graphics/JPEG/itu-t81.pdf

[6] V. Bhaskaran and K. Konstantinides, *Image and Video Compression Standards: Algorithms and Architectures*, 2nd ed. Boston: Kluwer Academic Publishers, 1997.

[7] 6.111 lab #3. [Online]. Available: http://web.mit.edu/6.111/www/f2007/handouts/labs/lab3.html

[8] R. J. D'Ortenzio, "US patent 5,825,312: DX JPEG huffman decoder," Xerox Corporation, Oct 1998.

[9] M. Kovac and N. Ranganathan, "JAGUAR: A fully pipelined VLSI architecture for JPEG image compression standard," *Proceedings of the IEEE*, vol. 83, no. 2, pp. 247–258, Feb 1995. [Online]. Available: http://ieeexplore.ieee.org/iel1/5/8350/00364464.pdf

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 $T : \mathbb{R}^8 \to \mathbb{R}^8$, 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 $T_k \dots T_2 T_1 = T$ [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

$$\begin{array}{llll} b_0 = x_0 + x_7; & b_1 = x_1 + x_6; & b_2 = x_3 - x_4; & b_3 = x_1 - x_6; \\ b_4 = x_2 + x_5; & b_5 = x_3 + x_4; & b_6 = x_2 - x_5; & b_7 = x_0 - x_7; \end{array}$$

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

$$z_{ij} = \frac{y_{ij}}{q_{ij}},$$

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]

$$\begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

Table 4: Chromanance quantization table [5]

$$\begin{bmatrix} 17 & 18 & 24 & 47 & 99 & 99 & 99 & 99 \\ 18 & 21 & 26 & 66 & 99 & 99 & 99 & 99 \\ 24 & 26 & 56 & 99 & 99 & 99 & 99 & 99 \\ 47 & 66 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \end{bmatrix}$$

C Huffman Tables

C.1 Luma DC Coefficients

Category (Length)	Code size	Code
0	2	00
1	3	010
2	3	011
3	3	100
4	3	101
5	3	110
6	4	1110
7	5	11110
8	6	111110
9	7	1111110
10	8	11111110
11	9	111111110

C.2 Chroma DC Coefficients

Category (Length)	Code size	Code
0	2	00
1	2	01
2	2	10
3	3	110
4	4	1110
5	5	11110
6	6	111110
7	7	1111110
8	8	11111110
9	9	111111110
10	10	1111111110
11	11	11111111110

C.3 Luma AC Coefficients

Run/Length	Code size	Code
0/0 (EOB)	3	1010
0/1	1	00
0/2	1	01
0/3	2	100
0/4	3	1011
0/5	4	11010
0/6	6	1111000
0/7	7	11111000
0/8	9	1111110110
0/9	15	1111111110000010
0/A	15	1111111110000011
1/1	3	1100
1/2	4	11011
1/3	6	1111001
1/4	8	111110110
1/5	10	11111110110
1/6	15	1111111110000100
1/7	15	1111111110000101
1/8	15	1111111110000110
1/9	15	1111111110000111
1/A	15	1111111110001000
2/1	4	11100
2/2	7	11111001
2/3	9	1111110111
2/4	11	111111110100
2/5	15	1111111110001001
2/6	15	1111111110001010
2/7	15	1111111110001011
2/8	15	1111111110001100
2/9	15	1111111110001101
2/A	15	1111111110001110
3/1	5	111010
3/2	8	111110111
3/3	11	111111110101
3/4	15	1111111110001111
3/5	15	1111111110010000
3/6	15	1111111110010001
3/7	15	1111111110010010
3/8	15	1111111110010011

Run/Length	Code size	Code
3/9	15	1111111110010100
3/A	15	1111111110010101
4/1	5	111011
4/2	9	1111111000
4/3	15	1111111110010110
4/4	15	1111111110010111
4/5	15	1111111110011000
4/6	15	1111111110011001
4/7	15	1111111110011010
4/8	15	1111111110011011
4/9	15	1111111110011100
4/A	15	1111111110011101
5/1	6	1111010
5/2	10	11111110111
5/3	15	1111111110011110
5/4	15	1111111110011111
5/5	15	1111111110100000
5/6	15	1111111110100001
5/7	15	1111111110100010
5/8	15	1111111110100011
5/9	15	1111111110100100
5/A	15	1111111110100101
6/1	6	1111011
6/2	11	11111110110
6/3	15	1111111110100110
6/4	15	1111111110100111
6/5	15	1111111110101000
6/6	15	1111111110101001
6/7	15	1111111110101010
6/8	15	1111111110101011
6/9	15	1111111110101100
6/A	15	1111111110101101
7/1	7	11111010
7/2	11	11111110111
7/3	15	1111111110101110
7/4	15	1111111110101111
7/5	15	1111111110110000
7/6	15	1111111110110001
7/7	15	1111111110110010
7/8	15	1111111110110011
7/9	15	1111111110110100

Run/Length	Code size	Code
7/A	15	1111111110110101
8/1	8	111111000
8/2	14	111111111000000
8/3	15	1111111110110110
8/4	15	1111111110110111
8/5	15	1111111110111000
8/6	15	1111111110111001
8/7	15	1111111110111010
8/8	15	1111111110111011
8/9	15	1111111110111100
8/A	15	1111111110111101
9/1	8	111111001
9/2	15	1111111110111110
9/3	15	1111111110111111
9/4	15	1111111111000000
9/5	15	1111111111000001
9/6	15	1111111111000010
9/7	15	1111111111000011
9/8	15	1111111111000100
9/9	15	1111111111000101
9/A	15	1111111111000110
A/1	8	111111010
A/2	15	1111111111000111
A/3	15	1111111111001000
A/4	15	1111111111001001
A/5	15	1111111111001010
A/6	15	1111111111001011
A/7	15	1111111111001100
A/8	15	1111111111001101
A/9	15	1111111111001110
A/A	15	1111111111001111
B/1	9	1111111001
B/2	15	1111111111010000
B/3	15	1111111111010001
B/4	15	1111111111010010
B/5	15	1111111111010011
B/6	15	1111111111010100
B/7	15	1111111111010101
B/8	15	1111111111010110
B/9	15	1111111111010111
B/A	15	1111111111011000

Run/Length	Code size	Code
C/1	9	1111111010
C/2	15	1111111111011001
C/3	15	1111111111011010
C/4	15	1111111111011011
C/5	15	1111111111011100
C/6	15	1111111111011101
C/7	15	1111111111011110
C/8	15	1111111111011111
C/9	15	1111111111100000
C/A	15	1111111111100001
D/1	10	11111111000
D/2	15	1111111111100010
D/3	15	1111111111100011
D/4	15	1111111111100100
D/5	15	1111111111100101
D/6	15	1111111111100110
D/7	15	1111111111100111
D/8	15	1111111111101000
D/9	15	1111111111101001
D/A	15	1111111111101010
E/1	15	1111111111101011
E/2	15	1111111111101100
E/3	15	1111111111101101
E/4	15	1111111111101110
E/5	15	1111111111101111
E/6	15	1111111111110000
E/7	15	1111111111110001
E/8	15	1111111111110010
E/9	15	1111111111110011
E/A	15	1111111111110100
F/0 (ZRL)	10	11111111001
F/1	15	1111111111110101
F/2	15	1111111111110110
F/3	15	1111111111110111
F/4	15	1111111111111000
F/5	15	1111111111111001
F/6	15	1111111111111010
F/7	15	1111111111111011
F/8	15	1111111111111100
F/9	15	1111111111111101
F/A	15	1111111111111110

C.4 Chroma AC Coefficients

Run/Length	Code size	Code
0/0 (EOB)	3	1010
0/1	1	00
0/2	1	01
0/3	2	100
0/4	3	1011
0/5	4	11010
0/6	6	1111000
0/7	7	11111000
0/8	9	1111110110
0/9	15	1111111110000010
0/A	15	1111111110000011
1/1	3	1100
1/2	4	11011
1/3	6	1111001
1/4	8	111110110
1/5	10	11111110110
1/6	15	1111111110000100
1/7	15	1111111110000101
1/8	15	1111111110000110
1/9	15	1111111110000111
1/A	15	1111111110001000
2/1	4	11100
2/2	7	11111001
2/3	9	1111110111
2/4	11	111111110100
2/5	15	1111111110001001
2/6	15	1111111110001010
2/7	15	1111111110001011
2/8	15	1111111110001100
2/9	15	1111111110001101
2/A	15	1111111110001110
3/1	5	111010
3/2	8	111110111
3/3	11	111111110101
3/4	15	1111111110001111
3/5	15	1111111110010000
3/6	15	1111111110010001
3/7	15	1111111110010010
3/8	15	1111111110010011

Run/Length	Code size	Code
3/9	15	1111111110010100
3/A	15	1111111110010101
4/1	5	111011
4/2	9	1111111000
4/3	15	1111111110010110
4/4	15	1111111110010111
4/5	15	1111111110011000
4/6	15	1111111110011001
4/7	15	1111111110011010
4/8	15	1111111110011011
4/9	15	1111111110011100
4/A	15	1111111110011101
5/1	6	1111010
5/2	10	11111110111
5/3	15	1111111110011110
5/4	15	1111111110011111
5/5	15	1111111110100000
5/6	15	1111111110100001
5/7	15	1111111110100010
5/8	15	1111111110100011
5/9	15	1111111110100100
5/A	15	1111111110100101
6/1	6	1111011
6/2	11	11111110110
6/3	15	1111111110100110
6/4	15	1111111110100111
6/5	15	1111111110101000
6/6	15	1111111110101001
6/7	15	1111111110101010
6/8	15	1111111110101011
6/9	15	1111111110101100
6/A	15	1111111110101101
7/1	7	11111010
7/2	11	11111110111
7/3	15	1111111110101110
7/4	15	1111111110101111
7/5	15	1111111110110000
7/6	15	1111111110110001
7/7	15	1111111110110010
7/8	15	1111111110110011
7/9	15	1111111110110100

Run/Length	Code size	Code
7/A	15	1111111110110101
8/1	8	111111000
8/2	14	111111111000000
8/3	15	1111111110110110
8/4	15	1111111110110111
8/5	15	1111111110111000
8/6	15	1111111110111001
8/7	15	1111111110111010
8/8	15	1111111110111011
8/9	15	1111111110111100
8/A	15	1111111110111101
9/1	8	111111001
9/2	15	1111111110111110
9/3	15	1111111110111111
9/4	15	1111111111000000
9/5	15	1111111111000001
9/6	15	1111111111000010
9/7	15	1111111111000011
9/8	15	1111111111000100
9/9	15	1111111111000101
9/A	15	1111111111000110
A/1	8	111111010
A/2	15	1111111111000111
A/3	15	1111111111001000
A/4	15	1111111111001001
A/5	15	1111111111001010
A/6	15	1111111111001011
A/7	15	1111111111001100
A/8	15	1111111111001101
A/9	15	1111111111001110
A/A	15	1111111111001111
B/1	9	1111111001
B/2	15	1111111111010000
B/3	15	1111111111010001
B/4	15	1111111111010010
B/5	15	1111111111010011
B/6	15	1111111111010100
B/7	15	1111111111010101
B/8	15	1111111111010110
B/9	15	1111111111010111
B/A	15	1111111111011000

Run/Length	Code size	Code
C/1	9	1111111010
C/2	15	1111111111011001
C/3	15	1111111111011010
C/4	15	1111111111011011
C/5	15	1111111111011100
C/6	15	1111111111011101
C/7	15	1111111111011110
C/8	15	1111111111011111
C/9	15	1111111111100000
C/A	15	1111111111100001
D/1	10	11111111000
D/2	15	1111111111100010
D/3	15	1111111111100011
D/4	15	1111111111100100
D/5	15	1111111111100101
D/6	15	1111111111100110
D/7	15	1111111111100111
D/8	15	1111111111101000
D/9	15	1111111111101001
D/A	15	1111111111101010
E/1	15	1111111111101011
E/2	15	1111111111101100
E/3	15	1111111111101101
E/4	15	1111111111101110
E/5	15	1111111111101111
E/6	15	1111111111110000
E/7	15	1111111111110001
E/8	15	1111111111110010
E/9	15	1111111111110011
E/A	15	1111111111110100
F/0 (ZRL)	10	11111111001
F/1	15	1111111111110101
F/2	15	1111111111110110
F/3	15	1111111111110111
F/4	15	1111111111111000
F/5	15	1111111111111001
F/6	15	1111111111111010
F/7	15	1111111111111011
F/8	15	1111111111111100
F/9	15	1111111111111101
F/A	15	1111111111111110

D Source Code

D.1 array_shrinker.v

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Engineer:    Evan Broder
4  //
5  // Create Date: 12:08:03 11/18/2007
6  // Module Name: array_shrinker
7  // Project Name: Video-Conferencing System
8  // Description: Truncates the most-significant bytes from the input for each
9  //              of 8 elements in a concatenated vector
10 //
11 // Dependencies:
12 //
13 // Revision:
14 // $Id: array_shrinker.v 91 2007-11-19 07:30:47Z evan $
15 // Additional Comments:
16 //
17 //////////////////////////////////////////////////////////////////////////////////
18
19 module array_shrinker(reset, clock, big, little);
20     parameter BIG_WIDTH = 16;
21     parameter LITTLE_WIDTH = 12;
22     input reset;
23     input clock;
24     input [(BIG_WIDTH * 8) - 1:0] big;
25     output [(LITTLE_WIDTH * 8) - 1:0] little;
26
27     genvar i;
28     generate for (i = 0; i < 8; i = i + 1)
29         begin:elt
30             // Take the full width of the little vector alloted to this particular
31             // element
32             assign little[(LITTLE_WIDTH * (i + 1)) - 1 : LITTLE_WIDTH * i] =
33                 // And assign it to the <LITTLE_WIDTH> least-significant bits of
34                 // big
35                 big[(BIG_WIDTH * i) + LITTLE_WIDTH - 1 : BIG_WIDTH * i];
36         end
37     endgenerate
38 endmodule
```

D.2 array_sign_extender.v

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Engineer:    Evan Broder
4  //
5  // Create Date: 12:26:26 11/17/2007
6  // Module Name: array_sign_extender
```

```
// Project Name: Video-Conferencing System
// Description: Takes a concatenated array of 8 signed values and sign
//              extend each of them
//
// Dependencies:
//
// Revision:
// $Id: array_sign_extender.v 291 2007-12-15 16:27:42Z evan $
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////

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)
        begin:elt
            // 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
        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:
```

```
// $Id: clock_divider.v 116 2007-12-01 21:54:45Z evan $
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

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

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Engineer:    Evan Broder
//
// Create Date: 20:18:24 11/27/2007
// Module Name: coordinate_delay
// Project Name: Video-Conferencing System
// Description: Because I was using the same code over and over again to grab
//              the input coordinates on posedge valid_in and then setting
//              them on posedge valid_out, it seemed like it would make sense
//              to pull the functionality into a separate module
//
// Dependencies:
//
// Revision:
// $Id: coordinate_delay.v 291 2007-12-15 16:27:42Z evan $
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

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

```
26      input [5:0] x_in;
27      input [4:0] y_in;
28      input valid_out;
29      output [5:0] x_out;
30      output [4:0] y_out;
31
32      reg [5:0] x_cache;
33      reg [4:0] y_cache;
34
35      wire valid_in_pulse, valid_out_pulse;
36
37      level_to_pulse valid_in_level(.reset(reset), .clock(clock),
38          .level(valid_in), .pulse(valid_in_pulse));
39
40      level_to_pulse valid_out_level(.reset(reset), .clock(clock),
41          .level(valid_out), .pulse(valid_out_pulse));
42
43      // Grab x_in and y_in on the positive edge of valid_in
44      always @(posedge clock)
45      begin
46          if (valid_in_pulse)
47          begin
48              x_cache <= x_in;
49              y_cache <= y_in;
50          end
51      end
52
53      // Latch the old values until the instant that valid_out is asserted -
54      // we don't want a one cycle delay here, which is why this isn't being done
55      // sequentially
56      assign x_out = valid_out_pulse ? x_cache : x_out;
57      assign y_out = valid_out_pulse ? y_cache : y_out;
58  endmodule
```

D.5 crc.v

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3   // Engineer:    Evan Broder
4   //
5   // Create Date: 12:08:03 11/18/2007
6   // Module Name: crc
7   // Project Name: Video-Conferencing System
8   // Description: Incorporates a new byte on data into the current value of the
9   //              CRC.
10  //              Function generated by www.easics.com
11  //              Info: tools@easics.be
12  //                    http://www.easics.com
13  //
14  // Dependencies:
15  //
```

```
16 // Revision:
17 // $Id: crc.v 166 2007-12-06 01:38:10Z evan $
18 // Additional Comments:
19 //
20 //////////////////////////////////////////////////////////////////////////////////
21
22 module crc(reset, clock, data, crc, en);
23     input reset;
24     input clock;
25     input [7:0] data;
26     output reg [7:0] crc;
27     input en;
28
29     // polynomial: (0 1 2 8)
30     // data width: 8
31     // convention: the first serial data bit is D[7]
32     function [7:0] nextCRC8_D8;
33         input [7:0] Data;
34         input [7:0] CRC;
35
36         reg [7:0] D;
37         reg [7:0] C;
38         reg [7:0] NewCRC;
39
40         begin
41             D = Data;
42             C = CRC;
43
44             NewCRC[0] = D[7] ^ D[6] ^ D[0] ^ C[0] ^ C[6] ^ C[7];
45             NewCRC[1] = D[6] ^ D[1] ^ D[0] ^ C[0] ^ C[1] ^ C[6];
46             NewCRC[2] = D[6] ^ D[2] ^ D[1] ^ D[0] ^ C[0] ^ C[1] ^ C[2] ^ C[6];
47             NewCRC[3] = D[7] ^ D[3] ^ D[2] ^ D[1] ^ C[1] ^ C[2] ^ C[3] ^ C[7];
48             NewCRC[4] = D[4] ^ D[3] ^ D[2] ^ C[2] ^ C[3] ^ C[4];
49             NewCRC[5] = D[5] ^ D[4] ^ D[3] ^ C[3] ^ C[4] ^ C[5];
50             NewCRC[6] = D[6] ^ D[5] ^ D[4] ^ C[4] ^ C[5] ^ C[6];
51             NewCRC[7] = D[7] ^ D[6] ^ D[5] ^ C[5] ^ C[6] ^ C[7];
52
53             nextCRC8_D8 = NewCRC;
54         end
55     endfunction
56
57     always @(posedge clock)
58     begin
59         if (reset) crc <= 0;
60         else if (en) crc <= nextCRC8_D8(data, crc);
61     end
62 endmodule
```

D.6 data_spew.v

```
1 `timescale 1ns / 1ps
```

```
2  //////////////////////////////////////////////////////////////////////////////
3  // Engineer:    Evan Broder, Chris Post
4  //
5  // Create Date: 19:23:10 11/05/2007
6  // Module Name: data_spew
7  // Project Name: Video-Conferencing System
8  // Description: Selects an example matrix based on the data_set input and
9  //              outputs it in the format expected by encoder when it gets
10 //              a positive edge on start
11 //
12 // Dependencies:
13 //
14 // Revision:
15 // $Id: data_spew.v 291 2007-12-15 16:27:42Z evan $
16 // Additional Comments:
17 //
18 //////////////////////////////////////////////////////////////////////////////
19
20 module data_spew(reset, clock, data_set, start, row, valid_out);
21     input reset;
22     input clock;
23     input [2:0] data_set;
24     input start;
25     output [63:0] row;
26     output valid_out;
27
28     parameter S_IDLE = 0;
29     parameter S_SPEW = 1;
30
31     wire [6:0] addr;
32     reg state;
33     reg [2:0] count;
34
35     // This is how the BRAM is addressed
36     assign addr = {data_set, count};
37
38     data_spew_mem mem(.clk(clock), .addr(addr), .dout(row));
39
40     assign valid_out = (state == S_SPEW);
41
42     always @(posedge clock)
43     begin
44         if (reset)
45         begin
46             state <= S_IDLE;
47             count <= 0;
48         end
49         else
50         case (state)
51             S_IDLE:
52             begin
```

```
53              if (start) state <= S_SPEW;
54          end
55          S_SPEW:
56          begin
57              if (count == 7) state <= S_IDLE;
58              count <= count + 1;
59          end
60      endcase
61  end
62  endmodule
```

D.7 dct_1d.v

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3   // Engineer:    Evan Broder, Chris Post
4   //
5   // Create Date: 19:23:10 11/05/2007
6   // Module Name: dct_1d
7   // Project Name: Video-Conferencing System
8   // Description: Finds a one-dimensional DCT based on the algorithm outlined
9   //              in Agostini:2001 and corrected in Agostini:2005
10  //
11  // Dependencies:
12  //
13  // Revision:
14  // $Id: dct_1d.v 291 2007-12-15 16:27:42Z evan $
15  // Additional Comments:
16  //
17  //////////////////////////////////////////////////////////////////////////////////
18
19  module dct_1d(reset, clock, dct_in, dct_out);
20      parameter WIDTH = 12;
21
22      // cos(4 pi / 16) << 15
23      parameter m1 = 23170;
24      // cos(6 pi / 16) << 15
25      parameter m2 = 12540;
26      // (cos(2 pi / 16) - cos(6 pi / 16)) << 15
27      parameter m3 = 17734;
28      // (cos(2 pi / 16) + cos(6 pi / 16)) << 15
29      parameter m4 = 42813;
30
31      input reset;
32      input clock;
33      // Verilog doesn't seem to like using arrays as inputs and outputs, so
34      // we'll take in the data as one massive vector
35      input [WIDTH * 8 - 1:0] dct_in;
36      // ...and output it as one even /more/ massive vector
37      output [((WIDTH + 4) * 8) - 1:0] dct_out;
38
```

```
39      // Once we get the data in, though, the array notation makes the
40      // implementation very clean, so we'll use that
41      // The increasing widths used to store the data are taken from the Agostini
42      // paper
43      wire signed [WIDTH - 1:0] a[0:7];
44      reg signed [WIDTH:0] b[0:7];
45      reg signed [WIDTH + 1:0] c[0:7];
46      reg signed [WIDTH + 2:0] d[0:8];
47      reg signed [WIDTH + 2:0] e[0:8];
48      reg signed [WIDTH + 3:0] f[0:7];
49      reg signed [WIDTH + 3:0] S[0:7];
50
51      // Split up the really wide bus into an array of 8 8-bit values
52      genvar i;
53      generate for (i = 0; i < 8; i = i + 1)
54          begin:A
55              //assign a[i] = row[WIDTH * (7 - i) +: WIDTH];
56              assign a[i] = dct_in[(WIDTH * (8 - i)) - 1 : WIDTH * (7 - i)];
57          end
58      endgenerate
59
60      // Take the final answer and concatenate it back into a giant array
61      assign dct_out = {S[0], S[1], S[2], S[3], S[4], S[5], S[6], S[7]};
62
63      always @(posedge clock)
64      begin
65          // Each variable is a different stage in the pipeline
66          b[0] <= a[0] + a[7];
67          b[1] <= a[1] + a[6];
68          // This next line was wrong in the original Agostini paper, but
69          // corrected later
70          b[2] <= a[3] - a[4];
71          b[3] <= a[1] - a[6];
72          b[4] <= a[2] + a[5];
73          b[5] <= a[3] + a[4];
74          b[6] <= a[2] - a[5];
75          b[7] <= a[0] - a[7];
76
77          c[0] <= b[0] + b[5];
78          c[1] <= b[1] - b[4];
79          c[2] <= b[2] + b[6];
80          c[3] <= b[1] + b[4];
81          c[4] <= b[0] - b[5];
82          c[5] <= b[3] + b[7];
83          c[6] <= b[3] + b[6];
84          c[7] <= b[7];
85
86          d[0] <= c[0] + c[3];
87          d[1] <= c[0] - c[3];
88          d[2] <= c[2];
89          d[3] <= c[1] + c[4];
```

```
90          d[4] <= c[2] - c[5];
91          d[5] <= c[4];
92          d[6] <= c[5];
93          d[7] <= c[6];
94          d[8] <= c[7];
95
96          // In the multiplier stage, the constant multiplicands were bit shifted
97          // to get integer numbers. Once the multiplication is done, we bit
98          // shift back the same amount
99          e[0] <= d[0];
100         e[1] <= d[1];
101         e[2] <= (m3 * d[2]) >> 15;
102         e[3] <= (m1 * d[7]) >> 15;
103         e[4] <= (m4 * d[6]) >> 15;
104         e[5] <= d[5];
105         e[6] <= (m1 * d[3]) >> 15;
106         e[7] <= (m2 * d[4]) >> 15;
107         e[8] <= d[8];
108
109         f[0] <= e[0];
110         f[1] <= e[1];
111         f[2] <= e[5] + e[6];
112         f[3] <= e[5] - e[6];
113         f[4] <= e[3] + e[8];
114         f[5] <= e[8] - e[3];
115         f[6] <= e[2] + e[7];
116         f[7] <= e[4] + e[7];
117
118         S[0] <= f[0];
119         S[1] <= f[4] + f[7];
120         S[2] <= f[2];
121         S[3] <= f[5] - f[6];
122         S[4] <= f[1];
123         S[5] <= f[5] + f[6];
124         S[6] <= f[3];
125         S[7] <= f[4] - f[7];
126     end
127 endmodule
```

D.8 dct_2d.v

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Engineer:    Evan Broder
4  //
5  // Create Date: 18:17:12 11/08/2007
6  // Module Name: dct_2d
7  // Project Name: Video-Conferencing System
8  // Description: This performs the 2-D DCT by connecting the level shifter to
9  //              the row-wise DCT, to the transpose module, then back to the
10 //              the first DCT
```

```
11 //
12 // Dependencies:
13 //
14 // Revision:
15 // $Id: dct_2d.v 106 2007-11-28 01:48:58Z evan $
16 // Additional Comments:
17 //
18 //////////////////////////////////////////////////////////////////////////////////
19
20 module dct_2d(reset, clock, row, valid_in, column, valid_out, x_in, y_in,
21        x_out, y_out);
22     input reset;
23     input clock;
24     input [63:0] row;
25     input valid_in;
26     output [127:0] column;
27     output valid_out;
28
29     input [5:0] x_in;
30     input [4:0] y_in;
31     output [5:0] x_out;
32     output [4:0] y_out;
33
34     // The row data once it's been shifted to [-128,127]
35     wire [63:0] row_shifted;
36     // Shifted row data that's been sign extended to 12 bits per value
37     wire [95:0] row_extended;
38     // What goes into the DCT (the output of the transposer muxed with the
39     // appropriately altered input)
40     wire [95:0] dct_in;
41     // The data coming out of the DCT
42     wire [127:0] dct_out;
43     // The same data, reduced to 12 bits per value by eliminating the 4 MSBs
44     wire [95:0] dct_out_shrunk;
45     // The data that goes into the column-wise DCT
46     wire [95:0] dct_column_in;
47
48     wire shift_row;
49     wire shift_column;
50
51     assign column = dct_out;
52     // When data is getting shifted out of the transposer, it should be going
53     // into the DCT for the second round1
54     assign dct_in = shift_column ? dct_column_in : row_extended;
55
56     // shift_row should happen 6 clock signals after data first starts coming
57     // in because that's the latency of the DCT pipeline
58     delay shift_row_delay(.clock(clock), .undelayed(valid_in),
59         .delayed(shift_row));
60     defparam shift_row_delay.DELAY = 6;
61
```

```
62      // 8 clock cycles after data starts row shifting into the matrix, it should
63      // all be there, so start shifting it out in columns
64      // Note that, for right now, I'm assuming that the data is coming in a
65      // single chunk without interruptions
66      delay shift_column_delay(.clock(clock), .undelayed(shift_row),
67          .delayed(shift_column));
68      defparam shift_column_delay.DELAY = 8;
69
70      // And then, 6 cycles after we start looking at column-wise data, it should
71      // actually start coming out the other end
72      delay valid_out_delay(.clock(clock), .undelayed(shift_column),
73          .delayed(valid_out));
74      defparam valid_out_delay.DELAY = 6;
75
76      coordinate_delay coord_delay(.reset(reset), .clock(clock),
77          .valid_in(valid_in), .x_in(x_in), .y_in(y_in),
78          .valid_out(valid_out), .x_out(x_out), .y_out(y_out));
79
80      level_shifter shifter(.reset(reset), .clock(clock), .row(row),
81          .row_shifted(row_shifted));
82
83      // Since the DCT needs to be wide enough for the 2nd round, it needs to be
84      // wider than the incoming data (which has now been signed), so sign-extend
85      // each of the 8 values up to 12 bits
86      array_sign_extender sign_extender(.reset(reset), .clock(clock),
87          .little(row_shifted), .big(row_extended));
88      defparam sign_extender.LITTLE_WIDTH = 8;
89      defparam sign_extender.BIG_WIDTH = 12;
90
91      dct_1d dct_1d(.reset(reset), .clock(clock), .dct_in(dct_in),
92          .dct_out(dct_out));
93      defparam dct_1d.WIDTH = 12;
94
95      // The first time the data comes out, it'll be 16 bits, but there are only
96      // 12 bits of information, and we need to shorten it so it'll fit back into
97      // the DCT, so chop off the 4 MSBs for each value
98      array_shrinker shrinker(.reset(reset), .clock(clock), .big(dct_out),
99          .little(dct_out_shrunk));
100     defparam shrinker.BIG_WIDTH = 16;
101     defparam shrinker.LITTLE_WIDTH = 12;
102
103     matrix_transpose transpose(.reset(reset), .clock(clock), .row(dct_out_shrunk),
104         .column(dct_column_in), .shift_row(shift_row),
105         .shift_column(shift_column));
106     defparam transpose.WIDTH = 12;
107 endmodule
```

D.9 debounce.v

```
1  `timescale 1ns / 1ps
2  // Switch Debounce Module
```

```
// 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,
```

```
21          deq_valid_out);
22      input reset;
23      input clock;
24      input stream;
25      input valid_in;
26      output ready;
27      input [5:0] x_in;
28      input [4:0] y_in;
29      output [63:0] row_out;
30      output valid_out;
31      output [5:0] x_out;
32      output [4:0] y_out;
33
34      output [11:0] eh_value_out;
35      output eh_valid_out;
36      wire [5:0] eh_x_out;
37      wire [4:0] eh_y_out;
38
39      output [95:0] deq_column_out;
40      output deq_valid_out;
41      wire [5:0] deq_x_out;
42      wire [4:0] deq_y_out;
43
44      unentropy_huffman unentropy_huffman(.reset(reset), .clock(clock),
45          .stream_in(stream), .valid_in(valid_in), .rdy(ready), .x_in(x_in),
46          .y_in(y_in), .value_out(eh_value_out), .valid_out(eh_valid_out),
47          .x_out(eh_x_out), .y_out(eh_y_out));
48
49      dequantizer dequantizer(.reset(reset), .clock(clock),
50          .value_in(eh_value_out), .valid_in(eh_valid_out), .x_in(eh_x_out),
51          .y_in(eh_y_out), .column_out(deq_column_out), .valid_out(deq_valid_out),
52          .x_out(deq_x_out), .y_out(deq_y_out));
53
54      idct_2d idct_2d(.reset(reset), .clock(clock), .column(deq_column_out),
55          .valid_in(deq_valid_out), .x_in(deq_x_out), .y_in(deq_y_out),
56          .row(row_out), .valid_out(valid_out), .x_out(x_out), .y_out(y_out));
57  endmodule
```

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)
```

```
12          if (DELAY == 1) delayed <= undelayed;
13          else {delayed, buffer} <= {buffer, undelayed};
14  endmodule
```

D.12 dequantizer.v

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////
3   // Engineer:    Evan Broder
4   //
5   // Create Date: 15:00:49 11/29/2007
6   // Module Name: dequantizer
7   // Project Name: Video-Conferencing System
8   // Description: Deuantizes each value from the DCT by a separate
9   //              quantization factor.
10  //
11  // Dependencies:
12  //
13  // Revision:
14  // $Id: dequantizer.v 224 2007-12-11 01:38:03Z evan $
15  // Additional Comments:
16  //
17  //////////////////////////////////////////////////////////////////////////////
18
19  module dequantizer(reset, clock, value_in, valid_in, column_out, valid_out,
20          x_in, x_out, y_in, y_out);
21      parameter S_IN = 0;
22      parameter S_OUT = 1;
23
24      input reset;
25      input clock;
26      input [11:0] value_in;
27      input valid_in;
28      output [95:0] column_out;
29      output valid_out;
30      input [5:0] x_in;
31      input [4:0] y_in;
32      output [5:0] x_out;
33      output [4:0] y_out;
34
35      reg [11:0] buffer [0:7][0:7];
36      reg [5:0] addr;
37      reg [5:0] buffer_addr;
38      reg [11:0] multiplicand;
39      wire [15:0] q_factor;
40      wire [27:0] product;
41      wire posedge_valid_in, negedge_valid_in;
42      wire buffer_store;
43      reg [2:0] out_counter;
44
45      reg state;
```

```
46
47      level_to_pulse pos_valid_in(.reset(reset), .clock(clock),
48          .level(valid_in), .pulse(posedge_valid_in));
49      level_to_pulse neg_valid_in(.reset(reset), .clock(clock),
50          .level(~valid_in), .pulse(negedge_valid_in));
51
52      delay buffer_store_delay(.clock(clock), .undelayed(valid_in),
53          .delayed(buffer_store));
54      defparam buffer_store_delay.DELAY = 3;
55
56      coordinate_delay coord_delay(.reset(reset), .clock(clock),
57          .valid_in(valid_in), .x_in(x_in), .y_in(y_in),
58          .valid_out(valid_out), .x_out(x_out), .y_out(y_out));
59
60      luma_dequantizer_table q_factors(.clk(clock), .addr(addr), .dout(q_factor));
61
62      dequantizer_mult mult(.clk(clock), .a(multiplicand), .b(q_factor),
63          .q(product));
64
65      always @(posedge clock)
66      begin
67          if (reset)
68          begin
69              addr <= 0;
70              buffer_addr <= 0;
71              state <= 0;
72          end
73          else
74          case (state)
75              S_IN:
76              begin
77                  if (buffer_addr == 63)
78                  begin
79                      out_counter <= 0;
80                      state <= S_OUT;
81                  end
82
83                  if (valid_in)
84                  begin
85                      addr <= addr + 1;
86                      multiplicand <= value_in;
87                  end
88
89                  if (buffer_store)
90                  begin
91                      buffer_addr <= buffer_addr + 1;
92                      buffer[buffer_addr[5:3]][buffer_addr[2:0]] <=
93                          {product[27], product[22:12]} + product[11];
94                  end
95              end
96              S_OUT:
```

```
97              begin
98                  out_counter <= out_counter + 1;
99                  if (out_counter == 7)
100                 begin
101                     state <= S_IN;
102                     buffer_addr <= 0;
103                     addr <= 0;
104                 end
105             end
106         endcase
107     end
108
109     assign valid_out = (state == S_OUT);
110     assign column_out = {buffer[out_counter][0], buffer[out_counter][1],
111         buffer[out_counter][2], buffer[out_counter][3], buffer[out_counter][4],
112         buffer[out_counter][5], buffer[out_counter][6], buffer[out_counter][7]};
113 endmodule
```

D.13 display_16hex.v

```
1  `timescale 1ns / 1ps
2
3  //////////////////////////////////////////////////////////////////////////////
4  //
5  // 6.111 FPGA Labkit -- Hex display driver
6  //
7  // File:  display_16hex.v
8  // Date:  24-Sep-05
9  //
10 // Created: April 27, 2004
11 // Author: Nathan Ickes
12 //
13 // 24-Sep-05 Ike: updated to use new reset-once state machine, remove clear
14 // 28-Nov-06 CJT: fixed race condition between CE and RS (thanks Javier!)
15 //
16 // This verilog module drives the labkit hex dot matrix displays, and puts
17 // up 16 hexadecimal digits (8 bytes). These are passed to the module
18 // through a 64 bit wire ("data"), asynchronously.
19 //
20 //////////////////////////////////////////////////////////////////////////////
21
22 module display_16hex (reset, clock_27mhz, data,
23        disp_blank, disp_clock, disp_rs, disp_ce_b,
24        disp_reset_b, disp_data_out);
25
26   input reset, clock_27mhz; // clock and reset (active high reset)
27   input [63:0] data;     // 16 hex nibbles to display
28
29   output disp_blank, disp_clock, disp_data_out, disp_rs, disp_ce_b,
30     disp_reset_b;
31
```

```
32    reg disp_data_out, disp_rs, disp_ce_b, disp_reset_b;
33
34    ////////////////////////////////////////////////////////////////////////
35    //
36    // Display Clock
37    //
38    // Generate a 500kHz clock for driving the displays.
39    //
40    ////////////////////////////////////////////////////////////////////////
41
42    reg [4:0] count;
43    reg [7:0] reset_count;
44    reg clock;
45    wire dreset;
46
47    always @(posedge clock_27mhz)
48      begin
49     if (reset)
50       begin
51          count = 0;
52          clock = 0;
53       end
54     else if (count == 26)
55       begin
56          clock = ~clock;
57          count = 5'h00;
58       end
59     else
60       count = count+1;
61      end
62
63    always @(posedge clock_27mhz)
64      if (reset)
65        reset_count <= 100;
66      else
67        reset_count <= (reset_count==0) ? 0 : reset_count-1;
68
69    assign dreset = (reset_count != 0);
70
71    assign disp_clock = ~clock;
72
73    ////////////////////////////////////////////////////////////////////////
74    //
75    // Display State Machine
76    //
77    ////////////////////////////////////////////////////////////////////////
78
79    reg [7:0] state;   // FSM state
80    reg [9:0] dot_index;   // index to current dot being clocked out
81    reg [31:0] control;    // control register
82    reg [3:0] char_index;  // index of current character
```

```
83    reg [39:0] dots;   // dots for a single digit
84    reg [3:0] nibble;      // hex nibble of current character
85
86    assign disp_blank = 1'b0; // low <= not blanked
87
88    always @(posedge clock)
89      if (dreset)
90        begin
91       state <= 0;
92       dot_index <= 0;
93       control <= 32'h7F7F7F7F;
94        end
95      else
96        casex (state)
97      8'h00:
98        begin
99           // Reset displays
100          disp_data_out <= 1'b0;
101          disp_rs <= 1'b0; // dot register
102          disp_ce_b <= 1'b1;
103          disp_reset_b <= 1'b0;
104          dot_index <= 0;
105          state <= state+1;
106       end
107
108     8'h01:
109       begin
110          // End reset
111          disp_reset_b <= 1'b1;
112          state <= state+1;
113       end
114
115     8'h02:
116       begin
117          // Initialize dot register (set all dots to zero)
118          disp_ce_b <= 1'b0;
119          disp_data_out <= 1'b0; // dot_index[0];
120          if (dot_index == 639)
121        state <= state+1;
122          else
123         dot_index <= dot_index+1;
124       end
125
126     8'h03:
127       begin
128          // Latch dot data
129          disp_ce_b <= 1'b1;
130          dot_index <= 31;    // re-purpose to init ctrl reg
131          disp_rs <= 1'b1; // Select the control register
132          state <= state+1;
133       end
```

```
134
135       8'h04:
136         begin
137           // Setup the control register
138           disp_ce_b <= 1'b0;
139           disp_data_out <= control[31];
140           control <= {control[30:0], 1'b0}; // shift left
141           if (dot_index == 0)
142          state <= state+1;
143           else
144          dot_index <= dot_index-1;
145         end
146
147       8'h05:
148         begin
149           // Latch the control register data / dot data
150           disp_ce_b <= 1'b1;
151           dot_index <= 39;    // init for single char
152           char_index <= 15;   // start with MS char
153           state <= state+1;
154           disp_rs <= 1'b0;    // Select the dot register
155         end
156
157       8'h06:
158         begin
159           // Load the user's dot data into the dot reg, char by char
160           disp_ce_b <= 1'b0;
161           disp_data_out <= dots[dot_index]; // dot data from msb
162           if (dot_index == 0)
163             if (char_index == 0)
164               state <= 5;         // all done, latch data
165          else
166          begin
167           char_index <= char_index - 1; // goto next char
168           dot_index <= 39;
169          end
170           else
171          dot_index <= dot_index-1; // else loop thru all dots
172         end
173
174         endcase
175
176     always @ (data or char_index)
177       case (char_index)
178         4'h0:       nibble <= data[3:0];
179         4'h1:       nibble <= data[7:4];
180         4'h2:       nibble <= data[11:8];
181         4'h3:       nibble <= data[15:12];
182         4'h4:       nibble <= data[19:16];
183         4'h5:       nibble <= data[23:20];
184         4'h6:       nibble <= data[27:24];
```

```
185         4'h7:       nibble <= data[31:28];
186         4'h8:       nibble <= data[35:32];
187         4'h9:       nibble <= data[39:36];
188         4'hA:       nibble <= data[43:40];
189         4'hB:       nibble <= data[47:44];
190         4'hC:       nibble <= data[51:48];
191         4'hD:       nibble <= data[55:52];
192         4'hE:       nibble <= data[59:56];
193         4'hF:       nibble <= data[63:60];
194       endcase
195
196     always @(nibble)
197       case (nibble)
198         4'h0: dots <= 40'b00111110_01010001_01001001_01000101_00111110;
199         4'h1: dots <= 40'b00000000_01000010_01111111_01000000_00000000;
200         4'h2: dots <= 40'b01100010_01010001_01001001_01001001_01000110;
201         4'h3: dots <= 40'b00100010_01000001_01001001_01001001_00110110;
202         4'h4: dots <= 40'b00011000_00010100_00010010_01111111_00010000;
203         4'h5: dots <= 40'b00100111_01000101_01000101_01000101_00111001;
204         4'h6: dots <= 40'b00111100_01001010_01001001_01001001_00110000;
205         4'h7: dots <= 40'b00000001_01110001_00001001_00000101_00000011;
206         4'h8: dots <= 40'b00110110_01001001_01001001_01001001_00110110;
207         4'h9: dots <= 40'b00000110_01001001_01001001_00101001_00011110;
208         4'hA: dots <= 40'b01111110_00001001_00001001_00001001_01111110;
209         4'hB: dots <= 40'b01111111_01001001_01001001_01001001_00110110;
210         4'hC: dots <= 40'b00111110_01000001_01000001_01000001_00100010;
211         4'hD: dots <= 40'b01111111_01000001_01000001_01000001_00111110;
212         4'hE: dots <= 40'b01111111_01001001_01001001_01001001_01000001;
213         4'hF: dots <= 40'b01111111_00001001_00001001_00001001_00000001;
214       endcase
215
216 endmodule
217
```

D.14 encoder.v

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Engineer:    Evan Broder
4  //
5  // Create Date: 21:11:58 12/05/2007
6  // Module Name: encoder
7  // Project Name: Video-Conferencing System
8  // Description: Perform the entire encoding stage on a block of a single
9  //              chanel
10 //
11 // Dependencies:
12 //
13 // Revision:
14 // $Id: encoder.v 266 2007-12-15 01:29:37Z evan $
15 // Additional Comments:
```

```
16 //
17 //////////////////////////////////////////////////////////////////////////////////
18
19 module encoder(reset, clock, row, valid_in, x_in, y_in, stream_out,
20         stream_out_we, stream_en, x_out, y_out, dct_out, dct_valid_out, quantizer_out, quantizer_valid_o
21     // CHANNEL == 0 => luma
22     // CHANNEL == 1 => chroma
23     parameter CHANNEL = 0;
24
25     input reset;
26     input clock;
27     input [63:0] row;
28     input valid_in;
29     input [5:0] x_in;
30     input [4:0] y_in;
31     output stream_out;
32     output stream_out_we;
33     output stream_en;
34     output [5:0] x_out;
35     output [4:0] y_out;
36
37     output [127:0] dct_out;
38     output dct_valid_out;
39     wire [5:0] dct_x_out;
40     wire [4:0] dct_y_out;
41
42     output [11:0] quantizer_out;
43     output quantizer_valid_out;
44     wire [5:0] quantizer_x_out;
45     wire [4:0] quantizer_y_out;
46
47     dct_2d dct_2d(.reset(reset), .clock(clock), .row(row), .valid_in(valid_in),
48         .column(dct_out), .valid_out(dct_valid_out), .x_in(x_in), .y_in(y_in),
49         .x_out(dct_x_out), .y_out(dct_y_out));
50
51     quantizer quantizer(.reset(reset), .clock(clock), .column_in(dct_out),
52         .valid_in(dct_valid_out), .value_out(quantizer_out),
53         .valid_out(quantizer_valid_out), .x_in(dct_x_out), .y_in(dct_y_out),
54         .x_out(quantizer_x_out), .y_out(quantizer_y_out));
55     defparam quantizer.CHANNEL = CHANNEL;
56
57     entropy_huffman entropy_huffman(.reset(reset), .clock(clock),
58         .value(quantizer_out), .valid_in(quantizer_valid_out),
59         .stream(stream_out), .we(stream_out_we), .stream_en(stream_en),
60         .x_in(quantizer_x_out), .y_in(quantizer_y_out), .x_out(x_out),
61         .y_out(y_out));
62     defparam entropy_huffman.CHANNEL = CHANNEL;
63 endmodule
```

D.15 entropy.v

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////
3  // Engineer:    Chris Post
4  //
5  // Create Date: 00:29:28 12/02/2007
6  // Module Name: entropy
7  // Project Name: Video-Conferencing System
8  // Description:
9  //
10 // Dependencies:
11 //
12 // Revision:
13 // $Id: entropy.v 169 2007-12-06 06:26:53Z evan $
14 // Additional Comments:
15 //
16 //////////////////////////////////////////////////////////////////////////////
17
18 module entropy(reset, clock, valid_in, in_value, read_en, out_value, ready);
19     parameter S_STORE = 0;
20     parameter S_READ = 1;
21     parameter S_STANDBY = 2;
22
23     input reset;
24     input clock;
25     input valid_in;
26     input [11:0] in_value;
27     input read_en;
28     output [11:0] out_value;
29     output ready;
30
31     reg [11:0] block [0:7][0:7];
32     reg [2:0] x;
33     reg [2:0] y;
34     reg direction;
35     wire valid_in_pulse;
36     reg [1:0] state;
37     parameter DOWNLEFT = 0;
38     parameter UPRIGHT = 1;
39
40     reg valid_in_delay;
41     reg [11:0] in_value_delay;
42
43     level_to_pulse valid_in_level(.reset(reset), .clock(clock),
44         .level(valid_in), .pulse(valid_in_pulse));
45
46     always @(posedge clock) begin
47         valid_in_delay <= valid_in;
48         in_value_delay <= in_value;
49
50         if (reset) begin
51             x <= 0;
```

```
52              y <= 0;
53              state <= S_STANDBY;
54              direction <= DOWNLEFT;
55          end
56
57          else if (valid_in_pulse & (state == S_STANDBY)) begin
58              x <= 0;
59              y <= 0;
60              state <= S_STORE;
61          end
62
63          else if (valid_in_delay & (state == S_STORE)) begin
64              block [x][y] <= in_value_delay;
65
66              if ((y == 7) & (x == 7)) begin
67                  x <= 0;
68                  y <= 0;
69                  state <= S_READ;
70              end
71              else if (y == 7) begin
72                  x <= x + 1;
73                  y <= 0;
74              end
75              else y <= y + 1;
76          end
77
78          if ((state == S_READ) & (x == 7) & (y == 7)) begin
79              state <= S_STANDBY;
80          end
81
82          // Huhm, this can probably be optimized for readability...
83          else if (read_en & (state == S_READ)) begin
84              if (y == 7) begin
85                  if ((x == 0) | (x == 2) | (x == 4) | (x == 6)) begin
86                      x <= x + 1;
87                  end
88                  else if ((x == 1) | (x == 3 ) | (x == 5)) begin
89                      direction <= UPRIGHT;
90                      x <= x + 1;
91                      y <= y - 1;
92                  end
93              end
94              else if (x == 7) begin
95                  if ((y == 1) | (y == 3) | (y == 5)) begin
96                      y <= y + 1;
97                  end
98                  if ((y == 0) | (y == 2) | (y == 4) | (y == 6)) begin
99                      direction <= DOWNLEFT;
100                     x <= x - 1;
101                     y <= y + 1;
102                 end
```

```
            end
            else if (y == 0) begin
                if ((x == 0) | (x == 2) | (x == 4) | (x == 6)) begin
                    x <= x + 1;
                end
                else if ((x == 1) | (x == 3) | (x == 5) | (x == 7)) begin
                    direction <= DOWNLEFT;
                    x <= x - 1;
                    y <= y + 1;
                end
            end
            else if (x == 0) begin
                if ((y == 1) | (y == 3) | (y == 5)) begin
                    y <= y + 1;
                end
                else if ((y == 2) | (y == 4) | (y == 6)) begin
                    direction <= UPRIGHT;
                    x <= x + 1;
                    y <= y - 1;
                end
            end
            else if (direction == DOWNLEFT)
            begin
                x <= x - 1;
                y <= y + 1;
            end
            else
            begin
                x <= x + 1;
                y <= y - 1;
            end
        end
    end

    assign out_value = block [x][y];
    assign ready = (state == S_READ);
endmodule
```

D.16 entropy_huffman.v

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Engineer:     Chris Post
4  //
5  // Create Date:  19:44:53 11/30/2007
6  // Module Name:  entropy_huffman
7  // Project Name: Video-Conferencing System
8  // Description:
9  //
10 // Dependencies:
11 //
```

```
12 // Revision:
13 // $Id: entropy_huffman.v 168 2007-12-06 05:02:06Z evan $
14 // Additional Comments:
15 //
16 ////////////////////////////////////////////////////////////////////////////////
17
18 module entropy_huffman(reset, clock, valid_in, x_in, y_in, value, x_out, y_out,
19         stream, we, stream_en);
20     parameter CHANNEL = 0;
21
22     input reset;
23     input clock;
24     input valid_in;
25     input [5:0] x_in;
26     input [4:0] y_in;
27     input [11:0] value;
28     output [5:0] x_out;
29     output [4:0] y_out;
30     output stream;
31     output we;
32     output stream_en;
33
34     wire ent_read_en;
35     wire [11:0] ent_out_value;
36     wire ent_rdy;
37
38     wire serial_rdy;
39     wire serial_en;
40     wire [15:0] code;
41     wire [4:0] code_size;
42     wire [11:0] huff_out_value;
43     wire [3:0] huff_out_value_size;
44     wire huff_done;
45
46     coordinate_delay coord_delay(.reset(reset), .clock(clock),
47         .valid_in(valid_in), .x_in(x_in), .y_in(y_in),
48         .valid_out(stream_en), .x_out(x_out), .y_out(y_out));
49
50     entropy entropy_coder(.reset(reset), .clock(clock), .valid_in(valid_in),
51         .in_value(value), .read_en(ent_read_en), .out_value(ent_out_value),
52         .ready(ent_rdy));
53
54     huffman huffman_coder(.reset(reset), .clock(clock), .value_in(ent_out_value),
55         .ent_rdy(ent_rdy), .ent_read(ent_read_en), .serial_rdy(serial_rdy),
56         .out_en(serial_en), .code(code), .code_size(code_size),
57         .value_out(huff_out_value), .value_size(huff_out_value_size),
58         .huff_done(huff_done));
59     defparam huffman_coder.CHANNEL = CHANNEL;
60
61     huffman_serializer serializer(.reset(reset), .clock(clock),
62         .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));
65
66      assign we = ~serial_rdy;
67
68      assign stream_en = ~huff_done | ~serial_rdy;
69
70  endmodule
```

D.17 huffman.v

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
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  //////////////////////////////////////////////////////////////////////////////////
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;
```

```
41      reg ent_read_D;
42      reg ent_read_DD;
43      reg huff_done_D;
44
45      reg [3:0] run;
46      reg [3:0] size;
47      reg [1:0] zrl;
48
49      reg [8:0] code_addr;
50      wire [19:0] code_data;
51      assign code_size = code_data[19:16] + 1;
52      assign code = code_data[15:0];
53
54      reg [11:0] value_in_latch;
55      reg [11:0] value_cor;
56      wire [3:0] category;
57
58      parameter DC_OFFSET = 0;
59      parameter AC_OFFSET = 16;
60
61      generate
62          if (CHANNEL == 0) begin:luma
63              luma_huffman_code huffman_code(.addr(code_addr), .clk(clock),
64                  .dout(code_data));
65          end
66          else begin:chroma
67              chroma_huffman_code huffman_code(.addr(code_addr), .clk(clock),
68                  .dout(code_data));
69          end
70      endgenerate
71
72      huffman_categorizer categorizer(.reset(reset), .clock(clock),
73          .value(value_cor), .category(category));
74
75      always @(posedge clock) begin
76          if (reset) begin
77              ent_read <= 0;
78              out_en <= 0;
79              out_en_PD <= 0;
80              value_out <= 0;
81              value_out_PD <= 0;
82              cur_val <= 0;
83              run <= 0;
84              size <= 0;
85              zrl <= 0;
86              huff_done <= 1;
87              huff_done_D <= 1;
88              code_addr <= 0;
89              ent_read_D <= 0;
90              ent_read_DD <= 0;
91              value_in_latch <= 0;
```

```
92              value_cor <= 0;
93          end
94
95          else begin
96              // Do the delays
97              value_out <= value_out_PD;
98              value_size <= value_size_PD;
99              out_en <= out_en_PD;
100             ent_read_D <= ent_read;
101             ent_read_DD <= ent_read_D;
102             value_in_latch <= value_in;
103             value_cor <= value_in_latch[11] ? (value_in_latch - 1) : value_in_latch;
104             huff_done_D <= huff_done;
105
106             if (ent_rdy & huff_done) begin
107                 cur_val <= 0;
108                 run <= 0;
109                 size <= 0;
110                 zrl <= 0;
111                 huff_done <= 0;
112                 ent_read <= 0;
113                 out_en_PD <= 0;
114             end
115
116             else if (~huff_done & ~huff_done_D & serial_rdy & ~out_en_PD &
117                 ~out_en & ~ent_read & ~ent_read_D & ~ent_read_DD) begin
118                 // Handle the DC component
119                 if (cur_val == 0) begin
120                     code_addr <= DC_OFFSET + {4'h0, category[3:0]};
121                     value_out_PD <= value_cor;
122                     value_size_PD <= category;
123                     out_en_PD <= 1;
124                     ent_read <= 1;
125                     cur_val <= cur_val + 1;
126                 end
127
128                 else if (cur_val == 64) huff_done <= 1;
129
130                 // Handle AC components
131                 else begin
132                     if (category == 0) begin
133                         cur_val <= cur_val + 1;
134
135                         if (cur_val == 63) begin
136                             code_addr <= AC_OFFSET + 8'h00;
137                             value_out_PD <= 0;
138                             value_size_PD <= 0;
139                             out_en_PD <= 1;
140                         end
141
142                         else begin
```

```
                        out_en_PD <= 0;
                        ent_read <= 1;

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

                    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
        end

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

endmodule
```

D.18 huffman_categorizer.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Engineer:    Chris Post
//
// Create Date: 22:48:22 12/04/2007
// Module Name: huffman_categorizer
// Project Name: Video-Conferencing System
```

```
8  // Description: Given an input, finds the "category" - i.e. the length - of
9  //              the input
10 //
11 // Dependencies:
12 //
13 // Revision:
14 // $Id: huffman_categorizer.v 181 2007-12-07 11:11:01Z evan $
15 // Additional Comments:
16 //
17 ////////////////////////////////////////////////////////////////////////////////
18
19 module huffman_categorizer(reset, clock, value, category);
20     input reset;
21     input clock;
22     input signed [11:0] value;
23     output reg [3:0] category;
24
25     wire a0, a1, a2, a3, a4, a5, a6, a7, a8, a9, a10;
26
27     assign a0 = value[11] ^ value[10];
28     assign a1 = value[11] ^ value[9];
29     assign a2 = value[11] ^ value[8];
30     assign a3 = value[11] ^ value[7];
31     assign a4 = value[11] ^ value[6];
32     assign a5 = value[11] ^ value[5];
33     assign a6 = value[11] ^ value[4];
34     assign a7 = value[11] ^ value[3];
35     assign a8 = value[11] ^ value[2];
36     assign a9 = value[11] ^ value[1];
37     assign a10 = value[11] ^ value[0];
38
39     always @*
40     begin
41         if (a0) category = 11;
42         else if (a1) category = 10;
43         else if (a2) category = 9;
44         else if (a3) category = 8;
45         else if (a4) category = 7;
46         else if (a5) category = 6;
47         else if (a6) category = 5;
48         else if (a7) category = 4;
49         else if (a8) category = 3;
50         else if (a9) category = 2;
51         else if (a10) category = 1;
52         else category = 0;
53     end
54 endmodule
```

D.19 huffman_serializer.v

```
1  `timescale 1ns / 1ps
```

```
2  //////////////////////////////////////////////////////////////////////////////
3  // Engineer:    Chris Post
4  //
5  // Create Date: 01:10:00 12/05/2007
6  // Module Name: huffman_serializer
7  // Project Name: Video-Conferencing System
8  // Description:
9  //
10 // Dependencies:
11 //
12 // Revision:
13 // $Id: huffman_serializer.v 202 2007-12-09 04:07:27Z evan $
14 // Additional Comments:
15 //
16 //////////////////////////////////////////////////////////////////////////////
17
18 module huffman_serializer(reset, clock, enable, ready, code, code_size, value,
19        value_size, stream);
20
21     parameter S_WAIT = 0;
22     parameter S_CODE = 1;
23     parameter S_VALUE = 2;
24
25     input reset;
26     input clock;
27     input enable;
28     output ready;
29     input [15:0] code;
30     input [4:0] code_size;
31     input [11:0] value;
32     input [3:0] value_size;
33     output stream;
34
35     reg [4:0] code_pos;
36     reg [3:0] value_pos;
37     reg [1:0] state;
38
39     always @(posedge clock) begin
40         if (reset) begin
41             state <= S_WAIT;
42         end
43
44         else begin
45             case (state)
46                 S_WAIT: begin
47                     if (enable)
48                     begin
49                         state <= S_CODE;
50                         code_pos <= code_size - 1;
51                     end
52                 end
```

```
53                S_CODE: begin
54                    if (code_pos == 0)
55                    begin
56                        if (value_size == 0) state <= S_WAIT;
57                        else begin
58                            state <= S_VALUE;
59                            value_pos <= value_size - 1;
60                        end
61                    end
62                    code_pos <= code_pos - 1;
63                end
64                S_VALUE: begin
65                    if (value_pos == 0) state <= S_WAIT;
66                    value_pos <= value_pos - 1;
67                end
68            endcase
69        end
70    end
71
72    assign stream = (state == S_WAIT) ? 0 :
73        (state == S_CODE) ? code[code_pos] : value[value_pos];
74    assign ready = (state == S_WAIT);
75 endmodule
```

D.20 idct_1d.v

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Engineer:    Evan Broder
4  //
5  // Create Date: 12:38:13 11/24/2007
6  // Module Name: idct_1d
7  // Project Name: Video-Conferencing System
8  // Description: Finds a one-dimensional iDCT based on the algorithm outlined
9  //              in Arai:1988
10 //
11 // Dependencies:
12 //
13 // Revision:
14 // $Id: idct_1d.v 113 2007-12-01 07:45:32Z evan $
15 // Additional Comments:
16 //
17 //////////////////////////////////////////////////////////////////////////////////
18
19 module idct_1d(reset, clock, idct_in, idct_out);
20    parameter WIDTH = 8;
21
22    // (1 / cos(4 * pi / 16)) << 15
23    parameter n1 = 46341;
24    // (1 / cos(6 * pi / 16)) << 15
25    parameter n2 = 85627;
```

```
26      // (1 / cos(2 * pi / 16)) << 15
27      parameter n3 = 35468;
28      // (1 / (cos(2 * pi / 16) + cos(6 * 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];
63          b[4] <= a[5] - a[3];
64          b[5] <= a[1] + a[7];
65          b[6] <= a[1] - a[7];
66          b[7] <= a[3] + a[5];
67
68          c[0] <= b[0];
69          c[1] <= b[1];
70          c[2] <= b[2] - b[3];
71          c[3] <= b[2] + b[3];
72          c[4] <= b[4];
73          c[5] <= b[5] - b[7];
74          c[6] <= b[6];
75          c[7] <= b[5] + b[7];
76          c[8] <= b[4] - b[6];
```

```
77
78          d[0] <= c[0];
79          d[1] <= c[1];
80          d[2] <= (n1 * c[2]) >> 15;
81          d[3] <= c[3];
82          d[4] <= (n2 * c[4]) >> 15;
83          d[5] <= (n1 * c[5]) >> 15;
84          d[6] <= (n3 * c[6]) >> 15;
85          d[7] <= c[7];
86          d[8] <= (n4 * c[8]) >> 15;
87
88          e[0] <= d[0] + d[1];
89          e[1] <= d[0] - d[1];
90          e[2] <= d[2] - d[3];
91          e[3] <= d[3];
92          e[4] <= d[8] - d[4];
93          e[5] <= d[5];
94          e[6] <= d[6] - d[8];
95          e[7] <= d[7];
96
97          f[0] <= e[0] + e[3];
98          f[1] <= e[1] + e[2];
99          f[2] <= e[1] - e[2];
100         f[3] <= e[0] - e[3];
101         f[4] <= e[4];
102         f[5] <= e[5] - e[6] + e[7];
103         f[6] <= e[6] - e[7];
104         f[7] <= e[7];
105
106         S[0] <= f[0] + f[7];
107         S[1] <= f[1] + f[6];
108         S[2] <= f[2] + f[5];
109         S[3] <= f[3] - f[4] - f[5];
110         S[4] <= f[3] + f[4] + f[5];
111         S[5] <= f[2] - f[5];
112         S[6] <= f[1] - f[6];
113         S[7] <= f[0] - f[7];
114     end
115 endmodule
```

D.21 idct_2d.v

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Engineer:    Evan Broder
4  //
5  // Create Date: 01:31:00 11/25/2007
6  // Module Name: idct_2d
7  // Project Name: Video-Conferencing System
8  // Description: Finds a two-dimensional iDCT by taking a 1-D iDCT on each
9  //              column, transposing it, then taking the 1-D iDCT on each row
```

```
10 //
11 // Dependencies:
12 //
13 // Revision:
14 // $Id: idct_2d.v 111 2007-11-30 22:52:44Z evan $
15 // Additional Comments:
16 //
17 //////////////////////////////////////////////////////////////////////////////////
18
19 module idct_2d(reset, clock, column, valid_in, row, valid_out, x_in, y_in,
20        x_out, y_out);
21     input reset;
22     input clock;
23     input [95:0] column;
24     input valid_in;
25     output [63:0] row;
26     output valid_out;
27
28     input [5:0] x_in;
29     input [4:0] y_in;
30     output [5:0] x_out;
31     output [4:0] y_out;
32
33     wire [95:0] idct_in;
34     wire [63:0] idct_out;
35
36     wire [63:0] transpose_out;
37     wire [95:0] transpose_extended;
38
39     wire shift_row;
40     wire shift_column;
41
42     assign idct_in = shift_column ? transpose_extended : column;
43
44     delay shift_row_delay(.clock(clock), .undelayed(valid_in),
45         .delayed(shift_row));
46     defparam shift_row_delay.DELAY = 6;
47
48     delay shift_column_delay(.clock(clock), .undelayed(shift_row),
49         .delayed(shift_column));
50     defparam shift_column_delay.DELAY = 8;
51
52     delay valid_out_delay(.clock(clock), .undelayed(shift_column),
53         .delayed(valid_out));
54     defparam valid_out_delay.DELAY = 6;
55
56     coordinate_delay coord_delay(.reset(reset), .clock(clock),
57         .valid_in(valid_in), .x_in(x_in), .y_in(y_in),
58         .valid_out(valid_out), .x_out(x_out), .y_out(y_out));
59
60     // iDCT takes in an array of 12-bit values and outputs 8-bit values
```

```
61      idct_1d idct_1d(.reset(reset), .clock(clock), .idct_in(idct_in),
62          .idct_out(idct_out));
63
64      matrix_transpose transpose(.reset(reset), .clock(clock), .row(idct_out),
65          .column(transpose_out), .shift_row(shift_row),
66          .shift_column(shift_column));
67      defparam transpose.WIDTH = 8;
68
69      // We have to make the output of the first round longer so it fits back in
70      // to the iDCT for the second go
71      array_sign_extender sign_extender(.reset(reset), .clock(clock),
72          .little(transpose_out), .big(transpose_extended));
73      defparam sign_extender.LITTLE_WIDTH = 8;
74      defparam sign_extender.BIG_WIDTH = 12;
75
76      // And last but not least, add 128 to put the data back in [0,255]
77      level_shifter shifter(.reset(reset), .clock(clock), .row(idct_out),
78          .row_shifted(row));
79  endmodule
```

D.22 labkit.v

```
1   `default_nettype none
2   ////////////////////////////////////////////////////////////////////////////////
3   //
4   // 6.111 FPGA Labkit -- Template Toplevel Module
5   //
6   // For Labkit Revision 004
7   //
8   //
9   // Created: October 31, 2004, from revision 003 file
10  // Author: Nathan Ickes
11  //
12  ////////////////////////////////////////////////////////////////////////////////
13  //
14  // CHANGES FOR BOARD REVISION 004
15  //
16  // 1) Added signals for logic analyzer pods 2-4.
17  // 2) Expanded "tv_in_ycrcb" to 20 bits.
18  // 3) Renamed "tv_out_data" to "tv_out_i2c_data" and "tv_out_sclk" to
19  //    "tv_out_i2c_clock".
20  // 4) Reversed disp_data_in and disp_data_out signals, so that "out" is an
21  //    output of the FPGA, and "in" is an input.
22  //
23  // CHANGES FOR BOARD REVISION 003
24  //
25  // 1) Combined flash chip enables into a single signal, flash_ce_b.
26  //
27  // CHANGES FOR BOARD REVISION 002
28  //
29  // 1) Added SRAM clock feedback path input and output
```

```
30 // 2) Renamed "mousedata" to "mouse_data"
31 // 3) Renamed some ZBT memory signals. Parity bits are now incorporated into
32 //    the data bus, and the byte write enables have been combined into the
33 //    4-bit ram#_bwe_b bus.
34 // 4) Removed the "systemace_clock" net, since the SystemACE clock is now
35 //    hardwired on the PCB to the oscillator.
36 //
37 //////////////////////////////////////////////////////////////////////////////
38 //
39 // Complete change history (including bug fixes)
40 //
41 // 2006-Mar-08: Corrected default assignments to "vga_out_red", "vga_out_green"
42 //              and "vga_out_blue". (Was 10'h0, now 8'h0.)
43 //
44 // 2005-Sep-09: Added missing default assignments to "ac97_sdata_out",
45 //              "disp_data_out", "analyzer[2-3]_clock" and
46 //              "analyzer[2-3]_data".
47 //
48 // 2005-Jan-23: Reduced flash address bus to 24 bits, to match 128Mb devices
49 //              actually populated on the boards. (The boards support up to
50 //              256Mb devices, with 25 address lines.)
51 //
52 // 2004-Oct-31: Adapted to new revision 004 board.
53 //
54 // 2004-May-01: Changed "disp_data_in" to be an output, and gave it a default
55 //              value. (Previous versions of this file declared this port to
56 //              be an input.)
57 //
58 // 2004-Apr-29: Reduced SRAM address busses to 19 bits, to match 18Mb devices
59 //              actually populated on the boards. (The boards support up to
60 //              72Mb devices, with 21 address lines.)
61 //
62 // 2004-Apr-29: Change history started
63 //
64 //////////////////////////////////////////////////////////////////////////////
65
66 module labkit (beep, audio_reset_b, ac97_sdata_out, ac97_sdata_in, ac97_synch,
67        ac97_bit_clock,
68
69        vga_out_red, vga_out_green, vga_out_blue, vga_out_sync_b,
70        vga_out_blank_b, vga_out_pixel_clock, vga_out_hsync,
71        vga_out_vsync,
72
73        tv_out_ycrcb, tv_out_reset_b, tv_out_clock, tv_out_i2c_clock,
74        tv_out_i2c_data, tv_out_pal_ntsc, tv_out_hsync_b,
75        tv_out_vsync_b, tv_out_blank_b, tv_out_subcar_reset,
76
77        tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1,
78        tv_in_line_clock2, tv_in_aef, tv_in_hff, tv_in_aff,
79        tv_in_i2c_clock, tv_in_i2c_data, tv_in_fifo_read,
80        tv_in_fifo_clock, tv_in_iso, tv_in_reset_b, tv_in_clock,
```

```
81
82          ram0_data, ram0_address, ram0_adv_ld, ram0_clk, ram0_cen_b,
83          ram0_ce_b, ram0_oe_b, ram0_we_b, ram0_bwe_b,
84
85          ram1_data, ram1_address, ram1_adv_ld, ram1_clk, ram1_cen_b,
86          ram1_ce_b, ram1_oe_b, ram1_we_b, ram1_bwe_b,
87
88          clock_feedback_out, clock_feedback_in,
89
90          flash_data, flash_address, flash_ce_b, flash_oe_b, flash_we_b,
91          flash_reset_b, flash_sts, flash_byte_b,
92
93          rs232_txd, rs232_rxd, rs232_rts, rs232_cts,
94
95          mouse_clock, mouse_data, keyboard_clock, keyboard_data,
96
97          clock_27mhz, clock1, clock2,
98
99          disp_blank, disp_data_out, disp_clock, disp_rs, disp_ce_b,
100         disp_reset_b, disp_data_in,
101
102         button0, button1, button2, button3, button_enter, button_right,
103         button_left, button_down, button_up,
104
105         switch,
106
107         led,
108
109         user1, user2, user3, user4,
110
111         daughtercard,
112
113         systemace_data, systemace_address, systemace_ce_b,
114         systemace_we_b, systemace_oe_b, systemace_irq, systemace_mpbrdy,
115
116         analyzer1_data, analyzer1_clock,
117         analyzer2_data, analyzer2_clock,
118         analyzer3_data, analyzer3_clock,
119         analyzer4_data, analyzer4_clock);
120
121     output beep, audio_reset_b, ac97_synch, ac97_sdata_out;
122     input ac97_bit_clock, ac97_sdata_in;
123
124     output [7:0] vga_out_red, vga_out_green, vga_out_blue;
125     output vga_out_sync_b, vga_out_blank_b, vga_out_pixel_clock,
126         vga_out_hsync, vga_out_vsync;
127
128     output [9:0] tv_out_ycrcb;
129     output tv_out_reset_b, tv_out_clock, tv_out_i2c_clock, tv_out_i2c_data,
130         tv_out_pal_ntsc, tv_out_hsync_b, tv_out_vsync_b, tv_out_blank_b,
131         tv_out_subcar_reset;
```

```
132
133     input [19:0] tv_in_ycrcb;
134     input tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2, tv_in_aef,
135         tv_in_hff, tv_in_aff;
136     output tv_in_i2c_clock, tv_in_fifo_read, tv_in_fifo_clock, tv_in_iso,
137         tv_in_reset_b, tv_in_clock;
138     inout tv_in_i2c_data;
139
140     inout [35:0] ram0_data;
141     output [18:0] ram0_address;
142     output ram0_adv_ld, ram0_clk, ram0_cen_b, ram0_ce_b, ram0_oe_b, ram0_we_b;
143     output [3:0] ram0_bwe_b;
144
145     inout [35:0] ram1_data;
146     output [18:0] ram1_address;
147     output ram1_adv_ld, ram1_clk, ram1_cen_b, ram1_ce_b, ram1_oe_b, ram1_we_b;
148     output [3:0] ram1_bwe_b;
149
150     input clock_feedback_in;
151     output clock_feedback_out;
152
153     inout [15:0] flash_data;
154     output [23:0] flash_address;
155     output flash_ce_b, flash_oe_b, flash_we_b, flash_reset_b, flash_byte_b;
156     input flash_sts;
157
158     output rs232_txd, rs232_rts;
159     input rs232_rxd, rs232_cts;
160
161     input mouse_clock, mouse_data, keyboard_clock, keyboard_data;
162
163     input clock_27mhz, clock1, clock2;
164
165     output disp_blank, disp_clock, disp_rs, disp_ce_b, disp_reset_b;
166     input disp_data_in;
167     output disp_data_out;
168
169     input button0, button1, button2, button3, button_enter, button_right,
170         button_left, button_down, button_up;
171     input [7:0] switch;
172     output [7:0] led;
173
174     inout [31:0] user1, user2, user3, user4;
175
176     inout [43:0] daughtercard;
177
178     inout [15:0] systemace_data;
179     output [6:0] systemace_address;
180     output systemace_ce_b, systemace_we_b, systemace_oe_b;
181     input systemace_irq, systemace_mpbrdy;
182
```

```
183     output [15:0] analyzer1_data, analyzer2_data, analyzer3_data,
184         analyzer4_data;
185     output analyzer1_clock, analyzer2_clock, analyzer3_clock, analyzer4_clock;
186
187     ////////////////////////////////////////////////////////////////////////
188     //
189     // I/O Assignments
190     //
191     ////////////////////////////////////////////////////////////////////////
192
193     // Audio Input and Output
194     assign beep= 1'b0;
195     assign audio_reset_b = 1'b0;
196     assign ac97_synch = 1'b0;
197     assign ac97_sdata_out = 1'b0;
198     // ac97_sdata_in is an input
199
200     // VGA Output
201     assign vga_out_red = 8'h0;
202     assign vga_out_green = 8'h0;
203     assign vga_out_blue = 8'h0;
204     assign vga_out_sync_b = 1'b1;
205     assign vga_out_blank_b = 1'b1;
206     assign vga_out_pixel_clock = 1'b0;
207     assign vga_out_hsync = 1'b0;
208     assign vga_out_vsync = 1'b0;
209
210     // Video Output
211     assign tv_out_ycrcb = 10'h0;
212     assign tv_out_reset_b = 1'b0;
213     assign tv_out_clock = 1'b0;
214     assign tv_out_i2c_clock = 1'b0;
215     assign tv_out_i2c_data = 1'b0;
216     assign tv_out_pal_ntsc = 1'b0;
217     assign tv_out_hsync_b = 1'b1;
218     assign tv_out_vsync_b = 1'b1;
219     assign tv_out_blank_b = 1'b1;
220     assign tv_out_subcar_reset = 1'b0;
221
222     // Video Input
223     assign tv_in_i2c_clock = 1'b0;
224     assign tv_in_fifo_read = 1'b0;
225     assign tv_in_fifo_clock = 1'b0;
226     assign tv_in_iso = 1'b0;
227     assign tv_in_reset_b = 1'b0;
228     assign tv_in_clock = 1'b0;
229     assign tv_in_i2c_data = 1'bZ;
230     // tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2,
231     // tv_in_aef, tv_in_hff, and tv_in_aff are inputs
232
233     // SRAMs
```

```
234     assign ram0_data = 36'hZ;
235     assign ram0_address = 19'h0;
236     assign ram0_adv_ld = 1'b0;
237     assign ram0_clk = 1'b0;
238     assign ram0_cen_b = 1'b1;
239     assign ram0_ce_b = 1'b1;
240     assign ram0_oe_b = 1'b1;
241     assign ram0_we_b = 1'b1;
242     assign ram0_bwe_b = 4'hF;
243     assign ram1_data = 36'hZ;
244     assign ram1_address = 19'h0;
245     assign ram1_adv_ld = 1'b0;
246     assign ram1_clk = 1'b0;
247     assign ram1_cen_b = 1'b1;
248     assign ram1_ce_b = 1'b1;
249     assign ram1_oe_b = 1'b1;
250     assign ram1_we_b = 1'b1;
251     assign ram1_bwe_b = 4'hF;
252     assign clock_feedback_out = 1'b0;
253     // clock_feedback_in is an input
254
255     // Flash ROM
256     assign flash_data = 16'hZ;
257     assign flash_address = 24'h0;
258     assign flash_ce_b = 1'b1;
259     assign flash_oe_b = 1'b1;
260     assign flash_we_b = 1'b1;
261     assign flash_reset_b = 1'b0;
262     assign flash_byte_b = 1'b1;
263     // flash_sts is an input
264
265     // RS-232 Interface
266     assign rs232_txd = 1'b1;
267     assign rs232_rts = 1'b1;
268     // rs232_rxd and rs232_cts are inputs
269
270     // PS/2 Ports
271     // mouse_clock, mouse_data, keyboard_clock, and keyboard_data are inputs
272
273     // Buttons, Switches, and Individual LEDs
274     //lab3 assign led = 8'hFF;
275     // button0, button1, button2, button3, button_enter, button_right,
276     // button_left, button_down, button_up, and switches are inputs
277
278     // User I/Os
279     assign user1 = 32'hZ;
280     assign user2 = 32'hZ;
281     assign user3 = 32'hZ;
282     assign user4 = 32'hZ;
283
284     // Daughtercard Connectors
```

```
285     assign daughtercard = 44'hZ;
286
287     // SystemACE Microprocessor Port
288     assign systemace_data = 16'hZ;
289     assign systemace_address = 7'h0;
290     assign systemace_ce_b = 1'b1;
291     assign systemace_we_b = 1'b1;
292     assign systemace_oe_b = 1'b1;
293     // systemace_irq and systemace_mpbrdy are inputs
294
295     wire [63:0] row_sprite;
296     wire [63:0] row_spew;
297     wire [5:0] x_in;
298     wire [4:0] y_in;
299
300     wire [127:0] dct_out;
301     wire dct_valid_out;
302     wire [11:0] quantizer_out;
303     wire quantizer_valid_out;
304
305     wire stream_out;
306     wire stream_out_we;
307     wire stream_en;
308     wire [5:0] x_out;
309     wire [4:0] y_out;
310
311     wire de_stream_out;
312     wire de_stream_valid;
313     wire de_eh_ready;
314     wire [5:0] de_x_in;
315     wire [4:0] de_y_in;
316
317     wire [63:0] row_out;
318     wire valid_out;
319     wire [5:0] de_x_out;
320     wire [4:0] de_y_out;
321
322     wire eh_valid_out;
323     wire [11:0] eh_value_out;
324
325     wire [95:0] deq_column_out;
326     wire deq_valid_out;
327
328     wire [63:0] transpose_out;
329
330     wire sdo;
331     wire sdi;
332
333     assign user1[0] = sdo;
334     assign sdi = switch[3] ? user1[1] : sdo;
335
```

```
336     assign led = {sdi, sdo};
337
338     // power-on reset generation
339     wire power_on_reset; // remain high for first 16 clocks
340     SRL16 reset_sr (.D(1'b0), .CLK(clock_27mhz), .Q(power_on_reset),
341         .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));
342     defparam reset_sr.INIT = 16'hFFFF;
343
344     // UP button is user reset
345     wire reset,user_reset;
346     debounce db1(power_on_reset, clock_27mhz, ~button_up, user_reset);
347     assign reset = user_reset | power_on_reset;
348
349     // UP and DOWN buttons for pong paddle
350     wire send, shift, send_p, shift_pulse;
351     debounce db9(reset, clock_27mhz, ~button_enter, send);
352     debounce db10(reset, clock_27mhz, ~button3, shift);
353     level_to_pulse send_ltp(reset, clock_27mhz, send, send_p);
354     level_to_pulse shift_level(reset, clock_27mhz, shift, shift_pulse);
355
356     wire [63:0] row_in = row_spew;
357     wire valid_in = valid_in_spew;
358     data_spew spew(.reset(reset), .clock(clock_27mhz), .data_set(switch[2:0]),
359         .start(send_p), .row(row_in), .valid_out(valid_in));
360
361     encoder encoder(.reset(reset), .clock(clock_27mhz), .row(row_in),
362         .valid_in(valid_in), .x_in(x_in), .y_in(y_in), .stream_out(stream_out),
363         .stream_out_we(stream_out_we), .stream_en(stream_en), .x_out(x_out),
364         .y_out(y_out), .dct_out(dct_out), .dct_valid_out(dct_valid_out),
365         .quantizer_out(quantizer_out), .quantizer_valid_out(quantizer_valid_out));
366     packer packer(.reset(reset), .clock(clock_27mhz), .cr_data(stream_out),
367         .cr_we(stream_out_we), .cr_stren(stream_en), .cr_x(x_out),
368         .cr_y(y_out), .sdo(sdo));
369     unpacker unpacker(.reset(reset), .clock(clock_27mhz), .sdi(sdi),
370         .cr_data(de_stream_out), .cr_valid(de_stream_valid),
371         .cr_ready(de_eh_ready), .cr_x(de_x_in), .cr_y(de_y_in));
372     decoder decoder(.reset(reset), .clock(clock_27mhz), .stream(de_stream_out),
373         .valid_in(de_stream_valid), .ready(de_eh_ready), .x_in(de_x_in),
374         .y_in(de_y_in), .row_out(row_out), .valid_out(valid_out),
375         .x_out(de_x_out), .y_out(de_y_out), .eh_valid_out(eh_valid_out), .eh_value_out(eh_value_out),
376         .deq_column_out(deq_column_out), .deq_valid_out(deq_valid_out));
377
378     matrix_transpose transpose(.reset(reset), .clock(clock_27mhz), .row(row_out),
379         .column(transpose_out), .shift_row(valid_out),
380         .shift_column(shift_pulse));
381     defparam transpose.WIDTH = 8;
382     display_16hex display(reset, clock_27mhz, transpose_out, disp_blank,
383        disp_clock, disp_rs, disp_ce_b, disp_reset_b, disp_data_out);
384
385     assign analyzer1_clock = clock_27mhz;
386     reg [15:0] analyzer1_data, analyzer2_data, analyzer3_data, analyzer4_data;
```

```
387     reg analyzer2_clock;
388     assign {analyzer3_clock, analyzer4_clock} = 0;
389     always @*
390     begin
391         case (switch[7:5])
392         0:
393         begin
394             {analyzer1_data, analyzer2_data, analyzer3_data, analyzer4_data} =
395                 row_in;
396             analyzer2_clock = valid_in;
397         end
398         1:
399         begin
400             analyzer1_data = {dct_out[127:120], dct_out[111:104]};
401             analyzer2_data = {dct_out[95:88], dct_out[79:72]};
402             analyzer3_data = {dct_out[63:56], dct_out[47:40]};
403             analyzer4_data = {dct_out[31:24], dct_out[15:8]};
404             analyzer2_clock = dct_valid_out;
405         end
406         2:
407         begin
408             analyzer1_data = {de_stream_out, de_stream_valid, stream_out, stream_out_we, sdo, sdi};
409             analyzer2_clock = stream_en;
410         end
411         3:
412         begin
413             analyzer1_data = eh_value_out;
414             analyzer2_clock = eh_valid_out;
415         end
416         4:
417         begin
418             analyzer1_data = {deq_column_out[95:88], deq_column_out[83:76]};
419             analyzer2_data = {deq_column_out[71:64], deq_column_out[59:52]};
420             analyzer3_data = {deq_column_out[47:40], deq_column_out[35:28]};
421             analyzer4_data = {deq_column_out[23:16], deq_column_out[11:4]};
422             analyzer2_clock = deq_valid_out;
423         end
424         5:
425         begin
426             {analyzer1_data, analyzer2_data, analyzer3_data, analyzer4_data} =
427                 row_out;
428             analyzer2_clock = valid_out;
429         end
430         endcase
431     end
432 endmodule
```

D.23 level_shifter.v

```
1 `timescale 1ns / 1ps
2 //////////////////////////////////////////////////////////////////////////////
```

```
// Engineer:    Evan Broder
//
// Create Date: 23:05:06 11/10/2007
// Module Name: level_shifter
// Project Name: Video-Conferencing System
// Description: The DCT algorithm depends on the input being in the range
//              [-128, 127], but the input actually comes in the range
//              [0, 255], so this subtracts 128 (actually, it just flips the
//              most significant bit of each value)
//
// Dependencies:
//
// Revision:
// $Id: level_shifter.v 91 2007-11-19 07:30:47Z evan $
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

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
```

D.24 level_to_pulse.v

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Engineer:    Evan Broder
//
// Create Date: 01:41:04 11/20/2007
// Module Name: level_to_pulse
// Project Name: Video-Conferencing System
// Description: A level-to-pulse converter, based on the lecture slides from
//              Lecture 7
//
// Dependencies:
//
// Revision:
// $Id: level_to_pulse.v 179 2007-12-07 10:38:08Z evan $
// Additional Comments:
```

```
16 //
17 //////////////////////////////////////////////////////////////////////////////
18
19 module level_to_pulse(reset, clock, level, pulse);
20     input reset;
21     input clock;
22     input level;
23     output pulse;
24
25     reg r;
26
27     always @(posedge clock)
28         if (reset) r <= 1;
29         else r <= level;
30
31     assign pulse = level & ~r;
32 endmodule
```

D.25 matrix_transpose.v

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////
3  // Engineer:    Evan Broder
4  //
5  // Create Date: 16:17:39 11/08/2007
6  // Module Name: matrix_transpose
7  // Project Name: Video-Conferencing System
8  // Description: Transposes a matrix by shifting it in in rows and shifting it
9  //              out in columns
10 //
11 // Dependencies:
12 //
13 // Revision:
14 // $Id: matrix_transpose.v 108 2007-11-29 04:16:12Z evan $
15 // Additional Comments:
16 //
17 //////////////////////////////////////////////////////////////////////////////
18
19 module matrix_transpose(reset, clock, row, column, shift_row, shift_column);
20     parameter WIDTH = 8;
21
22     input reset;
23     input clock;
24     input [(WIDTH * 8) - 1:0] row;
25     output [(WIDTH * 8) - 1:0] column;
26
27     input shift_row, shift_column;
28
29     reg [WIDTH - 1:0] row_matrix [0:7][0:7];
30
31     // The output should be the right column of the matrix
```

```
32      assign column = {row_matrix[0][0], row_matrix[1][0], row_matrix[2][0],
33          row_matrix[3][0], row_matrix[4][0], row_matrix[5][0], row_matrix[6][0],
34          row_matrix[7][0]};
35
36      genvar i, j;
37
38      // The bottom row is a special case because when shifting in data, it
39      // shifts from the external source
40      // When shifting columns, it should shift to the right
41      generate for (i = 0; i < 8; i = i + 1)
42          begin:row_0
43              always @(posedge clock)
44                  if (shift_row)
45                      row_matrix[7][i] <= row[WIDTH * (7 - i) +: WIDTH];
46                  else if (shift_column)
47                      row_matrix[7][i] <= row_matrix[7][(i == 7) ? i : i + 1];
48          end
49      endgenerate
50
51      // When shift_row is asserted, rows are shifted "up" the matrix
52      // When shift_column is asserted, columns are shifted right
53      generate for (i = 0; i < 7; i = i + 1)
54          begin:other_row
55              for (j = 0; j < 8; j = j + 1) begin:column
56                  always @(posedge clock)
57                      if (shift_row)
58                          row_matrix[i][j] <= row_matrix[i + 1][j];
59                      else if (shift_column)
60                          row_matrix[i][j] <= row_matrix[i][(j == 7) ? j : j + 1];
61              end
62          end
63      endgenerate
64  endmodule
```

D.26 packer.v

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3   // Engineer:    Evan Broder
4   //
5   // Create Date: 23:34:13 12/01/2007
6   // Module Name: packer
7   // Project Name: Video-Conferencing System
8   // Description: Checks each stream in turn to see if it's been loaded up. If
9   //              it has, connect it up to the packet_wrapper and send out the
10  //              data
11  //
12  // Dependencies:
13  //
14  // Revision:
15  // $Id: packer.v 124 2007-12-03 01:19:47Z evan $
```

```
16 // Additional Comments:
17 //
18 //////////////////////////////////////////////////////////////////////////////////
19
20 module packer(reset, clock, y_a_data, y_a_we, y_a_stren, y_a_x, y_a_y, y_b_data,
21        y_b_we, y_b_stren, y_b_x, y_b_y, y_c_data, y_c_we, y_c_stren, y_c_x,
22        y_c_y, y_d_data, y_d_we, y_d_stren, y_d_x, y_d_y, cr_data, cr_we,
23        cr_stren, cr_x, cr_y, cb_data, cb_we, cb_stren, cb_x, cb_y, audio_data,
24        audio_we, audio_stren, sdo);
25     parameter S_Y_A_CHECK = 0;
26     parameter S_Y_B_CHECK = 1;
27     parameter S_Y_C_CHECK = 2;
28     parameter S_Y_D_CHECK = 3;
29     parameter S_CR_CHECK = 4;
30     parameter S_CB_CHECK = 5;
31     parameter S_AUDIO_CHECK = 6;
32     parameter S_Y_A_SEND = 7;
33     parameter S_Y_B_SEND = 8;
34     parameter S_Y_C_SEND = 9;
35     parameter S_Y_D_SEND = 10;
36     parameter S_CR_SEND = 11;
37     parameter S_CB_SEND = 12;
38     parameter S_AUDIO_SEND = 13;
39
40     parameter CHAN_Y = 0;
41     parameter CHAN_CR = 1;
42     parameter CHAN_CB = 2;
43     parameter CHAN_AUDIO = 3;
44
45     input reset;
46     input clock;
47     input y_a_data;
48     input y_a_we;
49     input y_a_stren;
50     input [5:0] y_a_x;
51     input [4:0] y_a_y;
52     input y_b_data;
53     input y_b_we;
54     input y_b_stren;
55     input [5:0] y_b_x;
56     input [4:0] y_b_y;
57     input y_c_data;
58     input y_c_we;
59     input y_c_stren;
60     input [5:0] y_c_x;
61     input [4:0] y_c_y;
62     input y_d_data;
63     input y_d_we;
64     input y_d_stren;
65     input [5:0] y_d_x;
66     input [4:0] y_d_y;
```

```
67      input cr_data;
68      input cr_we;
69      input cr_stren;
70      input [5:0] cr_x;
71      input [4:0] cr_y;
72      input cb_data;
73      input cb_we;
74      input cb_stren;
75      input [5:0] cb_x;
76      input [4:0] cb_y;
77      input audio_data;
78      input audio_we;
79      input audio_stren;
80
81      output sdo;
82
83      reg [3:0] state;
84
85      reg start;
86      wire done;
87
88      reg [7:0] mem_data;
89      wire read;
90      reg read_ready;
91      reg [10:0] len;
92      reg [5:0] x;
93      reg [4:0] y;
94
95      reg [1:0] channel;
96
97      reg [5:0] y_a_x_cache, y_b_x_cache, y_c_x_cache, y_d_x_cache, cr_x_cache,
98          cb_x_cache;
99      reg [4:0] y_a_y_cache, y_b_y_cache, y_c_y_cache, y_d_y_cache, cr_y_cache,
100         cb_y_cache;
101
102     wire [7:0] y_a_out, y_b_out, y_c_out, y_d_out, cr_out, cb_out, audio_out;
103     wire [10:0] y_a_len, y_b_len, y_c_len, y_d_len, cr_len, cb_len, audio_len;
104     wire y_a_ready, y_b_ready, y_c_ready, y_d_ready, cr_ready, cb_ready,
105         audio_ready;
106     reg y_a_read, y_b_read, y_c_read, y_d_read, cr_read, cb_read,
107         audio_read;
108     wire y_a_stren_pulse, y_b_stren_pulse, y_c_stren_pulse, y_d_stren_pulse,
109         cr_stren_pulse, cb_stren_pulse;
110
111     // Used to latch the coordinates
112     level_to_pulse y_a_stren_level(.reset(reset), .clock(clock),
113         .level(y_a_stren), .pulse(y_a_stren_pulse));
114     level_to_pulse y_b_stren_level(.reset(reset), .clock(clock),
115         .level(y_b_stren), .pulse(y_b_stren_pulse));
116     level_to_pulse y_c_stren_level(.reset(reset), .clock(clock),
117         .level(y_c_stren), .pulse(y_c_stren_pulse));
```

```
118     level_to_pulse y_d_stren_level(.reset(reset), .clock(clock),
119         .level(y_d_stren), .pulse(y_d_stren_pulse));
120     level_to_pulse cr_stren_level(.reset(reset), .clock(clock),
121         .level(cr_stren), .pulse(cr_stren_pulse));
122     level_to_pulse cb_stren_level(.reset(reset), .clock(clock),
123         .level(cb_stren), .pulse(cb_stren_pulse));
124
125     // Each channel gets recorded into its own FIFO
126     packer_fifo y_a_fifo(.reset(reset), .clock(clock), .din(y_a_data),
127         .we(y_a_we), .stren(y_a_stren), .dout(y_a_out), .read_ready(y_a_ready),
128         .len(y_a_len), .read(y_a_read));
129     packer_fifo y_b_fifo(.reset(reset), .clock(clock), .din(y_b_data),
130         .we(y_b_we), .stren(y_b_stren), .dout(y_b_out), .read_ready(y_b_ready),
131         .len(y_b_len), .read(y_b_read));
132     packer_fifo y_c_fifo(.reset(reset), .clock(clock), .din(y_c_data),
133         .we(y_c_we), .stren(y_c_stren), .dout(y_c_out), .read_ready(y_c_ready),
134         .len(y_c_len), .read(y_c_read));
135     packer_fifo y_d_fifo(.reset(reset), .clock(clock), .din(y_d_data),
136         .we(y_d_we), .stren(y_d_stren), .dout(y_d_out), .read_ready(y_d_ready),
137         .len(y_d_len), .read(y_d_read));
138     packer_fifo cr_fifo(.reset(reset), .clock(clock), .din(cr_data),
139         .we(cr_we), .stren(cr_stren), .dout(cr_out), .read_ready(cr_ready),
140         .len(cr_len), .read(cr_read));
141     packer_fifo cb_fifo(.reset(reset), .clock(clock), .din(cb_data),
142         .we(cb_we), .stren(cb_stren), .dout(cb_out), .read_ready(cb_ready),
143         .len(cb_len), .read(cb_read));
144     packer_fifo audio_fifo(.reset(reset), .clock(clock), .din(audio_data),
145         .we(audio_we), .stren(audio_stren), .dout(audio_out),
146         .len(audio_len), .read_ready(audio_ready), .read(audio_read));
147
148     packer_wrapper wrapper(.reset(reset), .clock(clock), .start(start),
149         .done(done), .mem_data(mem_data), .read(read), .read_ready(read_ready),
150         .len(len), .x(x), .y(y), .channel(channel), .sdo(sdo));
151
152     always @(posedge clock)
153     begin
154         if (reset)
155         begin
156             state <= S_Y_A_CHECK;
157             start <= 0;
158         end
159         else
160         case (state)
161             S_Y_A_CHECK:
162             begin
163                 if (y_a_ready) state <= S_Y_A_SEND;
164                 else state <= S_Y_B_CHECK;
165                 start <= 0;
166             end
167             S_Y_B_CHECK:
168             begin
```

```
169             if (y_b_ready) state <= S_Y_B_SEND;
170             else state <= S_Y_C_CHECK;
171             start <= 0;
172         end
173         S_Y_C_CHECK:
174         begin
175             if (y_c_ready) state <= S_Y_C_SEND;
176             else state <= S_Y_D_CHECK;
177             start <= 0;
178         end
179         S_Y_D_CHECK:
180         begin
181             if (y_d_ready) state <= S_Y_D_SEND;
182             else state <= S_CR_CHECK;
183             start <= 0;
184         end
185         S_CR_CHECK:
186         begin
187             if (cr_ready) state <= S_CR_SEND;
188             else state <= S_CB_CHECK;
189             start <= 0;
190         end
191         S_CB_CHECK:
192         begin
193             if (cb_ready) state <= S_CB_SEND;
194             else state <= S_AUDIO_CHECK;
195             start <= 0;
196         end
197         S_AUDIO_CHECK:
198         begin
199             if (audio_ready) state <= S_AUDIO_SEND;
200             else state <= S_Y_A_CHECK;
201             start <= 0;
202         end
203         S_Y_A_SEND:
204         begin
205             // Wait until after the start signal has actually been
206             // asserted before deciding it's done
207             if (done & start) state <= S_Y_B_CHECK;
208             channel <= CHAN_Y;
209             start <= 1;
210         end
211         S_Y_B_SEND:
212         begin
213             if (done & start) state <= S_Y_C_CHECK;
214             channel <= CHAN_Y;
215             start <= 1;
216         end
217         S_Y_C_SEND:
218         begin
219             if (done & start) state <= S_Y_D_CHECK;
```

```
220             channel <= CHAN_Y;
221             start <= 1;
222         end
223         S_Y_D_SEND:
224         begin
225             if (done & start) state <= S_CR_CHECK;
226             channel <= CHAN_Y;
227             start <= 1;
228         end
229         S_CR_SEND:
230         begin
231             if (done & start) state <= S_CB_CHECK;
232             channel <= CHAN_CR;
233             start <= 1;
234         end
235         S_CB_SEND:
236         begin
237             if (done & start) state <= S_AUDIO_CHECK;
238             channel <= CHAN_CB;
239             start <= 1;
240         end
241         S_AUDIO_SEND:
242         begin
243             if (done & start) state <= S_Y_A_CHECK;
244             channel <= CHAN_AUDIO;
245             start <= 1;
246         end
247     endcase
248
249     if (y_a_stren_pulse & ~y_a_ready)
250     begin
251         y_a_x_cache <= y_a_x;
252         y_a_y_cache <= y_a_y;
253     end
254     if (y_b_stren_pulse & ~y_b_ready)
255     begin
256         y_b_x_cache <= y_b_x;
257         y_b_y_cache <= y_b_y;
258     end
259     if (y_c_stren_pulse & ~y_c_ready)
260     begin
261         y_c_x_cache <= y_c_x;
262         y_c_y_cache <= y_c_y;
263     end
264     if (y_d_stren_pulse & ~y_d_ready)
265     begin
266         y_d_x_cache <= y_d_x;
267         y_d_y_cache <= y_d_y;
268     end
269     if (cr_stren_pulse & ~cr_ready)
270     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             S_CR_SEND:
324             begin
325                 mem_data = cr_out;
326                 read_ready = cr_ready;
327                 len = cr_len;
328                 x = cr_x_cache;
329                 y = cr_y_cache;
330                 cr_read = read;
331             end
332             S_CB_SEND:
333             begin
334                 mem_data = cb_out;
335                 read_ready = cb_ready;
336                 len = cb_len;
337                 x = cb_x_cache;
338                 y = cb_y_cache;
339                 cb_read = read;
340             end
341             S_AUDIO_SEND:
342             begin
343                 mem_data = audio_out;
344                 read_ready = audio_ready;
345                 len = audio_len;
346                 audio_read = read;
347             end
348         endcase
349     end
350 endmodule
```

D.27 packer_fifo.v

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Engineer:    Evan Broder
4  //
5  // Create Date: 20:14:06 11/30/2007
6  // Module Name: packer_fifo
7  // Project Name: Video-Conferencing System
8  // Description: Stores incoming data if there's not data in the FIFO already
9  //              The first bit on din should be set at the same time that we is
10 //              asserted
11 //              There is a one clock cycle delay between when read is asserted
12 //              and when the data comes out dout.
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // $Id: packer_fifo.v 144 2007-12-05 00:40:30Z evan $
18 // Additional Comments:
19 //
```

```
20 //////////////////////////////////////////////////////////////////////////////
21
22 module packer_fifo(reset, clock, din, stren, we, dout, read_ready, len, read);
23     parameter S_WRITE = 0;
24     parameter S_READ = 1;
25
26     input reset;
27     input clock;
28     input din;
29     input stren;
30     input we;
31     output [7:0] dout;
32     output read_ready;
33     output [10:0] len;
34     input read;
35
36     reg [13:0] wptr;
37     wire [13:0] wptr_inc;
38     reg [10:0] rptr;
39     wire [10:0] rptr_inc;
40
41     reg state;
42
43     wire bram_we;
44     wire negedge_stren;
45     wire empty;
46     wire full;
47
48     level_to_pulse we_level(.reset(reset), .clock(clock), .level(~stren),
49         .pulse(negedge_stren));
50
51     packer_fifo_memory memory(.clka(clock), .clkb(clock), .dina(din),
52         .doutb(dout), .wea(bram_we), .addra(wptr), .addrb(rptr));
53
54     assign wptr_inc = wptr + 1;
55     assign rptr_inc = rptr + 1;
56
57     assign bram_we = we & (state == S_WRITE);
58
59     assign empty = (wptr[13:3] == rptr);
60     assign full = (wptr_inc == rptr << 3);
61
62     assign read_ready = state == S_READ;
63
64     assign len = wptr[13:3] - rptr;
65
66     always @(posedge clock)
67     begin
68         if (reset)
69         begin
70             wptr <= 0;
```

```
71              rptr <= 0;
72              state <= S_WRITE;
73          end
74          else
75          case (state)
76              S_WRITE:
77              begin
78                  if (negedge_stren)
79                  begin
80                      state <= S_READ;
81                      wptr <= (wptr[13:3] + 1) << 3;
82                  end
83                  else if (we & ~full) wptr <= wptr_inc;
84              end
85              S_READ:
86              begin
87                  if (empty) state <= S_WRITE;
88                  else if (read) rptr <= rptr_inc;
89              end
90          endcase
91      end
92  endmodule
```

D.28 packer_sat.v

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3   // Engineer:    Evan Broder
4   //
5   // Create Date: 14:05:31 12/01/2007
6   // Module Name: packer_sat
7   // Project Name: Video-Conferencing System
8   // Description: SAT: Serial Asynchronous Transmitter - the part of a USART
9   //              that's implemented by this module
10  //
11  // Dependencies:
12  //
13  // Revision:
14  // $Id: packer_sat.v 125 2007-12-03 04:17:56Z evan $
15  // Additional Comments:
16  //
17  //////////////////////////////////////////////////////////////////////////////////
18
19  module packer_sat(reset, clock, data, tx, txready, sdo);
20      parameter DATA_LEN = 8;
21
22      parameter S_IDLE = 0;
23      parameter S_SEND = 1;
24      parameter S_STOP_1 = 2;
25      parameter S_STOP_2 = 3;
26
```

```
27      input reset;
28      input clock;
29      input [DATA_LEN - 1:0] data;
30      input tx;
31      output txready;
32      output sdo;
33
34      reg [1:0] state;
35      reg [3:0] pos;
36      reg [DATA_LEN - 1:0] data_cache;
37
38      wire clk_enable;
39
40      clock_divider divider(.reset(state == S_IDLE), .clock(clock),
41          .enable(clk_enable));
42      defparam divider.D = 112;
43
44      always @(posedge clock)
45      begin
46          if (reset)
47          begin
48              pos <= 0;
49              state <= S_IDLE;
50          end
51          else
52          begin
53              case (state)
54                  S_IDLE:
55                  begin
56                      pos <= 0;
57                      if (tx)
58                      begin
59                          data_cache <= data;
60                          state <= S_SEND;
61                      end
62                  end
63                  S_SEND:
64                  if (clk_enable)
65                  begin
66                      if (pos == DATA_LEN) state <= S_STOP_1;
67                      else pos <= pos + 1;
68                  end
69                  S_STOP_1: if (clk_enable) state <= S_STOP_2;
70                  S_STOP_2: if (clk_enable) state <= S_IDLE;
71              endcase
72          end
73      end
74
75      assign sdo = (state != S_SEND) ? 1 :
76          (pos == 0) ? 0 : data_cache[pos - 1];
77      assign txready = (state == S_IDLE) & ~tx;
```

```
78 endmodule
```

D.29 packer_wrapper.v

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////
3  // Engineer:    Evan Broder
4  //
5  // Create Date: 10:50:30 12/02/2007
6  // Module Name: packet_wrapper
7  // Project Name: Video-Conferencing System
8  // Description: Given access to a BRAM FIFO and some other assorted info,
9  //              wraps up a packet in a nice bow and passes it to the SAT
10 //
11 // Dependencies:
12 //
13 // Revision:
14 // $Id: packer_wrapper.v 124 2007-12-03 01:19:47Z evan $
15 // Additional Comments:
16 //
17 //////////////////////////////////////////////////////////////////////////////
18
19 module packer_wrapper(reset, clock, start, done, mem_data, read, read_ready,
20        len, x, y, channel, sdo);
21    parameter S_WAIT = 0;
22    parameter S_START = 1;
23    parameter S_CHANNEL = 2;
24    parameter S_LEN_1 = 3;
25    parameter S_LEN_2 = 4;
26    parameter S_COORD_X = 5;
27    parameter S_COORD_Y = 6;
28    parameter S_DATA = 7;
29    parameter S_CRC = 8;
30
31    parameter CHAN_Y = 0;
32    parameter CHAN_CR = 1;
33    parameter CHAN_CB = 2;
34    parameter CHAN_AUDIO = 3;
35
36    input reset;
37    input clock;
38    input start;
39    output done;
40    input [7:0] mem_data;
41    input [10:0] len;
42    input [5:0] x;
43    input [4:0] y;
44    input [1:0] channel;
45    output reg read;
46    input read_ready;
47    output sdo;
```

```
48
49      reg [3:0] state;
50      reg tx;
51      wire txready;
52      reg [7:0] sat_data;
53      wire [7:0] crc;
54
55      wire start_pulse;
56
57      level_to_pulse start_level(.reset(reset), .clock(clock), .level(start),
58          .pulse(start_pulse));
59
60      packer_sat sat(.reset(reset), .clock(clock), .data(sat_data), .tx(tx),
61          .txready(txready), .sdo(sdo));
62      crc calc_crc(.reset(reset | (state == S_WAIT)), .clock(clock),
63          .data(sat_data), .crc(crc), .en(tx & (state == S_DATA)));
64
65      assign done = (state == S_WAIT) & ~start_pulse;
66
67      always @(posedge clock)
68      begin
69          if (reset)
70          begin
71              state <= S_WAIT;
72              tx <= 0;
73          end
74          else if (state == S_WAIT & start_pulse) state <= S_START;
75          // Every so often, the SAT is ready to transmit again. We should only be
76          // moving forward when it is.
77          else if (txready & state != S_WAIT)
78          begin
79              tx <= 1;
80              case (state)
81                  // Most of these states represent one byte in the packet
82                  S_START:
83                  begin
84                      sat_data <= 8'hff;
85                      state <= S_CHANNEL;
86                  end
87                  S_CHANNEL:
88                  begin
89                      sat_data <= channel;
90                      state <= S_LEN_1;
91                  end
92                  // Break the length into two bytes because it theoretically
93                  // could be, and this should support whatever might come out of
94                  // the FIFO
95                  S_LEN_1:
96                  begin
97                      sat_data <= len >> 8;
98                      state <= S_LEN_2;
```

```
                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
                end
            endcase
        end
        else
        begin
            tx <= 0;
            read <= 0;
        end
    end
endmodule
```

D.30 quantizer.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Engineer:    Evan Broder
//
// Create Date: 17:53:46 11/27/2007
// Module Name: quantizer
```

```
7  // Project Name: Video-Conferencing System
8  // Description: Quantizes each value from the DCT by a separate quantization
9  //              factor.
10 //
11 // Dependencies:
12 //
13 // Revision:
14 // $Id: quantizer.v 202 2007-12-09 04:07:27Z evan $
15 // Additional Comments:
16 //
17 ////////////////////////////////////////////////////////////////////////////////
18
19 module quantizer(reset, clock, column_in, valid_in, value_out, valid_out, x_in,
20        y_in, x_out, y_out);
21     parameter CHANNEL = 0;
22
23     input reset;
24     input clock;
25     input [127:0] column_in;
26     input valid_in;
27     output reg [11:0] value_out;
28     output valid_out;
29     input [5:0] x_in;
30     input [4:0] y_in;
31     output [5:0] x_out;
32     output [4:0] y_out;
33
34     wire negedge_valid_in;
35
36     reg [2:0] column;
37     reg calculating;
38     reg [5:0] addr;
39
40     reg [15:0] dividend;
41     wire [11:0] divisor;
42     wire [15:0] quotient;
43     wire [1:0] fraction;
44
45     reg [15:0] buffer [0:7][0:7];
46
47     level_to_pulse neg_valid_in(.reset(reset), .clock(clock),
48         .level(~valid_in), .pulse(negedge_valid_in));
49
50     delay valid_out_delay(.clock(clock), .undelayed(calculating),
51         .delayed(valid_out));
52     defparam valid_out_delay.DELAY = 24;
53
54     coordinate_delay coord_delay(.reset(reset), .clock(clock),
55         .valid_in(valid_in), .x_in(x_in), .y_in(y_in),
56         .valid_out(valid_out), .x_out(x_out), .y_out(y_out));
57
```

```
58      generate if (CHANNEL == 0)
59      begin: luma
60          luma_quantizer_table q_table(.clk(clock), .addr(addr), .dout(divisor));
61      end
62      else
63      begin: chroma
64          chroma_quantizer_table q_table(.clk(clock), .addr(addr),
65              .dout(divisor));
66      end
67      endgenerate
68
69      quantizer_divider divider(.clk(clock), .dividend(dividend),
70          .divisor(divisor), .quotient(quotient), .remainder(fraction));
71
72      always @(posedge clock)
73      begin
74          if (reset)
75          begin
76              column <= 0;
77              addr <= 0;
78              calculating <= 0;
79          end
80          else if (calculating)
81          begin
82              addr <= addr + 1;
83              dividend <= buffer[addr[5:3]][addr[2:0]];
84
85              if (addr == 63) calculating <= 0;
86          end
87          else if (valid_in)
88          begin
89              {buffer[0][column], buffer[1][column], buffer[2][column],
90                  buffer[3][column], buffer[4][column], buffer[5][column],
91                  buffer[6][column], buffer[7][column]} <= column_in;
92              column <= column + 1;
93          end
94          else if (negedge_valid_in)
95          begin
96              calculating <= 1;
97          end
98
99          value_out <= quotient[11:0] + fraction[1];
100     end
101 endmodule
```

D.31 sign_extender.v

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Engineer:     Chris Post
4  //
```

```
5  // Create Date:  12:52:16 12/10/2007
6  // Module Name:  sign_extender
7  // Project Name: Video-Conferencing System
8  // Description:  Sign-extends a shortened value based on its length
9  //
10 // Dependencies:
11 //
12 // Revision:
13 // $Id: sign_extender.v 291 2007-12-15 16:27:42Z evan $
14 // Additional Comments:
15 //
16 //////////////////////////////////////////////////////////////////////////////
17
18 module sign_extender(reset, clock, value, size, extended_value);
19     input reset;
20     input clock;
21     input [11:0] value;
22     input [3:0] size;
23     output reg [11:0] extended_value;
24
25     wire [11:0] value_tc;
26
27     assign value_tc = value[size - 1] ? value : value + 1;
28
29     always @*
30     case (size)
31         0: extended_value = 0;
32         1: extended_value = {{11{~value_tc[0]}}, value_tc[0]};
33         2: extended_value = {{10{~value_tc[1]}}, value_tc[1:0]};
34         3: extended_value = {{9{~value_tc[2]}}, value_tc[2:0]};
35         4: extended_value = {{8{~value_tc[3]}}, value_tc[3:0]};
36         5: extended_value = {{7{~value_tc[4]}}, value_tc[4:0]};
37         6: extended_value = {{6{~value_tc[5]}}, value_tc[5:0]};
38         7: extended_value = {{5{~value_tc[6]}}, value_tc[6:0]};
39         8: extended_value = {{4{~value_tc[7]}}, value_tc[7:0]};
40         9: extended_value = {{3{~value_tc[8]}}, value_tc[8:0]};
41         10: extended_value = {{2{~value_tc[9]}}, value_tc[9:0]};
42         11: extended_value = {{1{~value_tc[10]}}, value_tc[10:0]};
43     endcase
44 endmodule
```

D.32 unentropy.v

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////
3  // Engineer:    Chris Post
4  //
5  // Create Date: 08:23:00 12/08/2007
6  // Module Name: unentropy
7  // Project Name: Video-Conferencing System
8  // Description:
```

```
9  //
10 // Dependencies:
11 //
12 // Revision:
13 // $Id: unentropy.v 224 2007-12-11 01:38:03Z evan $
14 // Additional Comments:
15 //
16 //////////////////////////////////////////////////////////////////////////////////
17
18 module unentropy(reset, clock, huffman_proc, valid_in, run, size, value_in,
19        valid_out, value_out, processing);
20
21     parameter S_IDLE = 0;
22     parameter S_WRITE = 1;
23     parameter S_READ = 2;
24
25     input reset;
26     input clock;
27     input huffman_proc;
28     input valid_in;
29     input [3:0] run;
30     input [3:0] size;
31     input [11:0] value_in;
32     output valid_out;
33     output reg [11:0] value_out;
34     output processing;
35
36     reg huffman_proc_D;
37     reg [1:0] state;
38     reg [63:0] mask;
39     reg [6:0] cur_val;
40
41     wire mem_val_WE;
42     reg [6:0] mem_val_addr;
43     reg [11:0] mem_val_in;
44     wire [11:0] mem_val_out;
45     reg [6:0] mem_perm_addr;
46     wire [11:0] mem_perm_out;
47     reg [11:0] perm_addr_D, perm_addr_DD;
48     wire [11:0] extended_value;
49
50     wire valid_out_PD;
51
52     delay valid_out_delay(.clock(clock), .undelayed(valid_out_PD),
53         .delayed(valid_out));
54     defparam valid_out_delay.DELAY = 5;
55
56     delay mem_WE_delay(.clock(clock), .undelayed((state == S_WRITE) & valid_in),
57         .delayed(mem_val_WE));
58     defparam mem_WE_delay.DELAY = 1;
59
```

```
60      sign_extender extender(.reset(reset), .clock(clock), .value(value_in),
61          .size(size), .extended_value(extended_value));
62
63      parameter VALUE_OFFSET = 0;
64      parameter PERM_OFFSET = 64;
65
66      unentropy_mem unentropy_mem(
67          .addra(mem_val_addr),
68          .addrb(mem_perm_addr),
69          .clka(clock),
70          .clkb(clock),
71          .dina(mem_val_in),
72          .douta(mem_val_out),
73          .doutb(mem_perm_out),
74          .wea(mem_val_WE));
75
76      assign valid_out_PD = (state == S_READ);
77      assign processing = ~(state == S_IDLE);
78
79      always @ (posedge clock) begin
80          huffman_proc_D <= huffman_proc;
81
82          if (reset) begin
83              huffman_proc_D <= 0;
84              state <= S_IDLE;
85              mask <= 0;
86              cur_val <= 0;
87              mem_val_addr <= 0;
88              mem_val_in <= 0;
89              mem_perm_addr <= 0;
90              value_out <= 0;
91          end
92
93          else if (state == S_IDLE) begin
94              if (huffman_proc) state <= S_WRITE;
95              mask <= 0;
96              cur_val <= 0;
97          end
98
99          else if (state == S_WRITE) begin
100             if (~huffman_proc_D) begin
101                 state <= S_READ;
102                 cur_val <= 0;
103             end
104
105             else if (valid_in) begin
106                 if ((cur_val == 64) | ((run == 0) & (size == 0))) begin
107                     state <= S_READ;
108                     cur_val <= 0;
109                 end
110
```

```
111                 else begin
112                     cur_val <= cur_val + run + 1;
113                     mask[cur_val + run] <= 1;
114                     mem_val_addr <= VALUE_OFFSET + cur_val + run;
115                     if (size == 0) mem_val_in <= 0;
116                     else mem_val_in <= extended_value;
117                 end
118             end
119         end
120 
121         else if (state == S_READ) begin
122             if (cur_val == 68) state <= S_IDLE;
123 
124             cur_val <= cur_val + 1;
125 
126             mem_perm_addr <= PERM_OFFSET + cur_val;
127             mem_val_addr <= VALUE_OFFSET + mem_perm_out;
128             value_out <= mem_val_out;
129             value_out <= mask[perm_addr_DD] ? mem_val_out : 0;
130         end
131 
132         perm_addr_D <= mem_perm_out;
133         perm_addr_DD <= perm_addr_D;
134     end
135 
136 endmodule
```

D.33 unentropy_huffman.v

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Engineer:    Chris Post
4  //
5  // Create Date: 11:29:00 12/08/2007
6  // Module Name: unentropy_huffman
7  // Project Name: Video-Conferencing System
8  // Description:
9  //
10 // Dependencies:
11 //
12 // Revision:
13 // $Id: unentropy_huffman.v 217 2007-12-10 18:56:44Z evan $
14 // Additional Comments:
15 //
16 //////////////////////////////////////////////////////////////////////////////////
17 
18 module unentropy_huffman(reset, clock, rdy, valid_in, stream_in, x_in, y_in,
19         valid_out, value_out, x_out, y_out, entropy_proc);
20 
21     input reset;
22     input clock;
```

```
23      output rdy;
24      input valid_in;
25      input stream_in;
26      input [5:0] x_in;
27      input [4:0] y_in;
28      output valid_out;
29      output [11:0] value_out;
30      output [5:0] x_out;
31      output [4:0] y_out;
32
33      wire huffman_proc;
34      wire huffman_valid_out;
35      wire [11:0] huffman_value_out;
36      output entropy_proc;
37
38      reg busy;
39
40      wire [3:0] run;
41      wire [3:0] size;
42
43      coordinate_delay coord_delay(.reset(reset), .clock(clock),
44          .valid_in(valid_in), .x_in(x_in), .y_in(y_in),
45          .valid_out(valid_out), .x_out(x_out), .y_out(y_out));
46
47      unhuffman unhuffman(.reset(reset), .clock(clock), .serial_in(stream_in),
48          .serial_valid(valid_in), .processing(huffman_proc),
49          .valid_out(huffman_valid_out), .run(run), .size(size),
50          .value_out(huffman_value_out));
51
52      unentropy unentropy(.reset(reset), .clock(clock),
53          .huffman_proc(huffman_proc), .valid_in(huffman_valid_out), .run(run),
54          .size(size), .value_in(huffman_value_out), .valid_out(valid_out),
55          .value_out(value_out), .processing(entropy_proc));
56
57      always @ (posedge clock) begin
58          if (reset) busy <= 0;
59          else if (valid_in) busy <= 1;
60          else if (~entropy_proc) busy <= 0;
61      end
62
63      assign rdy = ~busy;
64  endmodule
```

D.34 unhuffman.v

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////
3   // Engineer:    Chris Post
4   //
5   // Create Date: 22:26:00 12/05/2007
6   // Module Name: unhuffman
```

```
7  // Project Name: Video-Conferencing System
8  // Description:
9  //
10 // Dependencies:
11 //
12 // Revision:
13 // $Id: unhuffman.v 201 2007-12-09 00:22:52Z ccpost $
14 // Additional Comments:
15 //
16 //////////////////////////////////////////////////////////////////////////////////
17
18 module unhuffman(reset, clock, serial_in, serial_valid, processing, valid_out,
19        run, size, value_out);
20
21     parameter CHANNEL = 0;
22
23     input reset;
24     input clock;
25     input serial_in;
26     input serial_valid;
27     output processing;
28     output reg valid_out;
29     output reg [3:0] run;
30     output reg [3:0] size;
31     output reg [11:0] value_out;
32
33     reg working_internal;
34     reg DC_proc;
35
36     reg no_out;
37     reg [3:0] no_out_count;
38
39     reg [26:0] buffer;
40     wire [10:0] DC_code;
41     wire [8:0] high_code_buff;
42     wire [7:0] low_code_buff;
43     wire [10:0] VC_value; // Value in the buffer at the valid code checking stage
44     wire [10:0] DC_value; // Value in the buffer at the DC valid checking stage
45     reg [26:0] mask_buffer;
46
47     assign DC_code = buffer[21:11];
48     assign high_code_buff = buffer[26:18];
49     assign low_code_buff = buffer[18:11];
50     assign VC_value = buffer[12:2];
51     assign DC_value = buffer[11:1];
52
53     wire reset_buf;
54
55     reg [3:0] DC_size;
56     reg [3:0] DC_codesize;
57     reg [3:0] DC_predict_codesize;
```

```
58
59      reg [8:0] mem_addr;
60      wire [11:0] mem_dout;
61      wire [3:0] mem_run;
62      wire [3:0] mem_size;
63      wire [3:0] mem_codesize;
64
65      reg [3:0] predict_size;
66      reg group_a;
67      reg [3:0] predict_size_D;
68      reg group_a_D;
69
70      assign mem_run = mem_dout[11:8];
71      assign mem_size = mem_dout[7:4];
72      assign mem_codesize = mem_dout[3:0];
73
74      parameter GROUP_A = 0;
75      parameter GROUP_B = 256;
76      parameter GROUP_C = 256 + 128;
77
78      // Instantiate a LTP for posedge serial_valid -> reset_buf
79      level_to_pulse posedge_serial_valid(.reset(reset), .clock(clock),
80          .level(serial_valid), .pulse(reset_buf));
81
82      // Instantiate the proper memory element
83      generate
84          if (CHANNEL == 0) begin: luma_AC_decode
85              luma_unhuffman_code unhuffman_code(.addr(mem_addr), .clk(clock),
86                  .dout(mem_dout));
87          end
88
89          else begin: chroma_AC_decode
90              chroma_unhuffman_code unhuffman_code(.addr(mem_addr), .clk(clock),
91                  .dout(mem_dout));
92          end
93      endgenerate
94
95      // Processing goes high when serial data starts coming in, and goes low
96      // after last code in serial stream is output.
97      assign processing = serial_valid | working_internal;
98
99      always @ (posedge clock) begin
100         // Make sure output isn't valid if we aren't working
101         if (~working_internal) valid_out <= 0;
102
103         // Do some delays
104         predict_size_D <= predict_size;
105         group_a_D <= group_a;
106
107         if (reset) begin
108             working_internal <= 0;
```

```
109             DC_proc <= 0;
110             no_out <= 1;
111             no_out_count <= 0;
112             buffer <= 0;
113             mask_buffer <= 0;
114             DC_predict_codesize <= 12;
115             mem_addr <= GROUP_C + 8'hFF; // Code here never valid
116             predict_size <= 0;
117             group_a <= 0;
118             predict_size_D <= 0;
119             group_a_D <= 0;
120             valid_out <= 0;
121             run <= 0;
122             size <= 0;
123             value_out <= 0;
124         end
125
126         else if (reset_buf | working_internal) begin
127             // Wait for the last of the serial data to process, then we're done
128             if (working_internal & (mask_buffer == 0)) working_internal <= 0;
129
130             // Reset the buffer and don't recognize codes until data is aligned
131             if (reset_buf) begin
132                 buffer <= 0;
133                 buffer[0] <= serial_in;
134                 mask_buffer <= 1;
135                 no_out_count <= 13; // 12, code in position; 2, code out of memory
136                 no_out <= 1;
137                 working_internal <= 1;
138                 DC_proc <= 1;
139                 predict_size <= 0;
140                 group_a <= 0;
141             end
142
143             else begin
144                 // Shift the buffer
145                 buffer <= buffer << 1;
146                 buffer[0] <= (serial_valid) ? serial_in : 0;
147                 mask_buffer <= mask_buffer << 1;
148                 mask_buffer[0] <= serial_valid;
149
150                 if (DC_proc & (no_out_count == 3)) DC_predict_codesize <= 0;
151
152                 if (DC_predict_codesize != 12)
153                     DC_predict_codesize <= DC_predict_codesize + 1;
154
155                 // Always block for DC code lookup generated below
156
157                 // Lookup the buffer's current potential AC code
158                 case (high_code_buff)
159                     9'b000000000: begin
```

```
160                     mem_addr <= GROUP_A + low_code_buff[6:0];
161                     predict_size <= 1;
162                     group_a <= 1;
163                 end
164                 9'b000000001: begin
165                     mem_addr <= GROUP_A + low_code_buff[7:0];
166                     predict_size <= 8 - 1;
167                     group_a <= 1;
168                 end
169                 9'b000000011: begin
170                     mem_addr <= GROUP_B + low_code_buff[3:0];
171                     predict_size <= 9 - 1;
172                     group_a <= 0;
173                 end
174                 9'b000000111: begin
175                     mem_addr <= GROUP_B + low_code_buff[4:0];
176                     predict_size <= 10 - 1;
177                     group_a <= 0;
178                 end
179                 9'b000001111: begin
180                     mem_addr <= GROUP_B + low_code_buff[5:0];
181                     predict_size <= 11 - 1;
182                     group_a <= 0;
183                 end
184                 9'b000011111: begin
185                     mem_addr <= GROUP_B + low_code_buff[6:0];
186                     predict_size <= 12 - 1;
187                     group_a <= 0;
188                 end
189                 9'b000111111: begin
190                     mem_addr <= GROUP_C + low_code_buff[3:0];
191                     predict_size <= 13 - 1;
192                     group_a <= 0;
193                 end
194                 9'b001111111: begin
195                     mem_addr <= GROUP_C + low_code_buff[4:0];
196                     predict_size <= 14 - 1;
197                     group_a <= 0;
198                 end
199                 9'b011111111: begin
200                     mem_addr <= GROUP_C + low_code_buff[5:0];
201                     predict_size <= 15 - 1;
202                     group_a <= 0;
203                 end
204                 9'b111111111: begin
205                     mem_addr <= GROUP_C + low_code_buff[6:0];
206                     predict_size <= 16 - 1;
207                     group_a <= 0;
208                 end
209                 default    : begin
210                     mem_addr <= GROUP_C + 8'hFF;
```

```
211                     predict_size <= 1;
212                     group_a <= 0;
213                 end
214             endcase
215
216             // Test for valid code, output if valid
217             if (~no_out & ((DC_proc & (DC_codesize == DC_predict_codesize)) |
218                     (~DC_proc & (mem_codesize == predict_size_D)) |
219                     (~DC_proc & group_a_D & (mem_codesize != 0)))) begin
220                 valid_out <= 1;
221
222                 DC_proc <= 0;
223
224                 run <= DC_proc ? 0 : mem_run;
225                 size <= DC_proc ? DC_size : mem_size;
226                 if (DC_proc) begin
227                     value_out <= DC_value >> (11 - DC_size);
228                     no_out_count <= DC_size + 2;
229                     no_out <= 1;
230                     buffer[26:12] <= 0;
231                     mask_buffer[26:12] <= 0;
232                     buffer[12:2] <= buffer[11:1] & (11'b11111111111 >> DC_size);
233                     mask_buffer[12:2] <= mask_buffer[11:1] & (11'b11111111111 >> DC_size);
234                 end
235                 else begin
236                     value_out <= VC_value >> (11 - mem_size);
237                     no_out_count <= mem_size + 2;
238                     no_out <= (mem_size == 0) ? 0 : 1;
239                     buffer[26:14] <= 0;
240                     mask_buffer[26:14] <= 0;
241                     buffer[13:3] <= buffer[12:2] & (11'b11111111111 >> mem_size);
242                     mask_buffer[13:3] <= mask_buffer[12:2] & (11'b11111111111 >> mem_size);
243                 end
244             end
245
246             // Disable output otherwise
247             else begin
248                 // Code invalid, output nothing, run output disable counter
249                 valid_out <= 0;
250
251                 // Run the counter for output disable
252                 if ((no_out_count == 1) | (no_out_count == 0)) begin
253                     no_out_count <= 0;
254                     no_out <= 0;
255                 end
256                 else no_out_count <= no_out_count - 1;
257             end
258         end
259     end
260 end
261
```

```
262     // Lookup the buffer's curent potential DC code
263     generate
264         if (CHANNEL == 0) begin: luma_DC_decode
265             always @ (posedge clock) begin: luma_DC_always
266                 if (reset) begin
267                     DC_size <= 0;
268                     DC_codesize <= 0;
269                 end
270 
271                 else begin
272                     case (DC_code)
273                         11'b00000000000: begin DC_size <= 0; DC_codesize <= 2; end
274                         11'b00000000010: begin DC_size <= 1; DC_codesize <= 3; end
275                         11'b00000000011: begin DC_size <= 2; DC_codesize <= 3; end
276                         11'b00000000100: begin DC_size <= 3; DC_codesize <= 3; end
277                         11'b00000000101: begin DC_size <= 4; DC_codesize <= 3; end
278                         11'b00000000110: begin DC_size <= 5; DC_codesize <= 3; end
279                         11'b00000001110: begin DC_size <= 6; DC_codesize <= 4; end
280                         11'b00000011110: begin DC_size <= 7; DC_codesize <= 5; end
281                         11'b00000111110: begin DC_size <= 8; DC_codesize <= 6; end
282                         11'b00001111110: begin DC_size <= 9; DC_codesize <= 7; end
283                         11'b00011111110: begin DC_size <= 10; DC_codesize <= 8; end
284                         11'b00111111110: begin DC_size <= 11; DC_codesize <= 9; end
285                         default       : begin DC_size <= 0; DC_codesize <= 0; end
286                     endcase
287                 end
288             end
289         end
290 
291         else begin: chroma_DC_decode
292             always @ (posedge clock) begin: chroma_DC_always
293                 if (reset) begin
294                     DC_size <= 0;
295                     DC_codesize <= 0;
296                 end
297 
298                 else begin
299                     case (DC_code)
300                         11'b00000000000: begin DC_size <= 0; DC_codesize <= 2; end
301                         11'b00000000001: begin DC_size <= 1; DC_codesize <= 2; end
302                         11'b00000000010: begin DC_size <= 2; DC_codesize <= 2; end
303                         11'b00000000110: begin DC_size <= 3; DC_codesize <= 3; end
304                         11'b00000001110: begin DC_size <= 4; DC_codesize <= 4; end
305                         11'b00000011110: begin DC_size <= 5; DC_codesize <= 5; end
306                         11'b00000111110: begin DC_size <= 6; DC_codesize <= 6; end
307                         11'b00001111110: begin DC_size <= 7; DC_codesize <= 7; end
308                         11'b00011111110: begin DC_size <= 8; DC_codesize <= 8; end
309                         11'b00111111110: begin DC_size <= 9; DC_codesize <= 9; end
310                         11'b01111111110: begin DC_size <= 10; DC_codesize <= 10; end
311                         11'b11111111110: begin DC_size <= 11; DC_codesize <= 11; end
312                         default       : begin DC_size <= 0; DC_codesize <= 0; end
```

```
313                     endcase
314                 end
315             end
316         end
317     endgenerate
318
319 endmodule
```

D.35 unpacker.v

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Engineer:    Evan Broder
4  //
5  // Create Date: 16:26:31 12/04/2007
6  // Module Name: unpacker
7  // Project Name: Video-Conferencing System
8  // Description: Receives packets, decodes the packet format, checks the CRC,
9  //              and passes the data on to one of the FIFOs, which are hooked
10 //              up to the Huffman decoders
11 //
12 // Dependencies:
13 //
14 // Revision:
15 // $Id: unpacker.v 186 2007-12-07 23:41:55Z evan $
16 // Additional Comments:
17 //
18 //////////////////////////////////////////////////////////////////////////////////
19
20 module unpacker(reset, clock, sdi, y_a_data, y_a_valid, y_a_x, y_a_y, y_a_ready,
21        y_b_data, y_b_valid, y_b_x, y_b_y, y_b_ready, y_c_data, y_c_valid,
22        y_c_x, y_c_y, y_c_ready, y_d_data, y_d_valid, y_d_x, y_d_y, y_d_ready,
23        cr_data, cr_valid, cr_x, cr_y, cr_ready, cb_data, cb_valid, cb_x, cb_y,
24        cb_ready, audio_data, audio_valid, audio_ready);
25     parameter S_WAIT = 0;
26     parameter S_STORE = 1;
27
28     parameter CHAN_Y = 0;
29     parameter CHAN_CR = 1;
30     parameter CHAN_CB = 2;
31     parameter CHAN_AUDIO = 3;
32
33     input reset;
34     input clock;
35     input sdi;
36     output y_a_data;
37     output y_a_valid;
38     output reg [5:0] y_a_x;
39     output reg [4:0] y_a_y;
40     input y_a_ready;
41     output y_b_data;
```

```
42      output y_b_valid;
43      output reg [5:0] y_b_x;
44      output reg [4:0] y_b_y;
45      input y_b_ready;
46      output y_c_data;
47      output y_c_valid;
48      output reg [5:0] y_c_x;
49      output reg [4:0] y_c_y;
50      input y_c_ready;
51      output y_d_data;
52      output y_d_valid;
53      output reg [5:0] y_d_x;
54      output reg [4:0] y_d_y;
55      input y_d_ready;
56      output cr_data;
57      output cr_valid;
58      output reg [5:0] cr_x;
59      output reg [4:0] cr_y;
60      input cr_ready;
61      output cb_data;
62      output cb_valid;
63      output reg [5:0] cb_x;
64      output reg [4:0] cb_y;
65      input cb_ready;
66      output audio_data;
67      output audio_valid;
68      input audio_ready;
69
70      reg [7:0] y_a_din, y_b_din, y_c_din, y_d_din, cr_din, cb_din, audio_din;
71      reg y_a_we, y_b_we, y_c_we, y_d_we, cr_we, cb_we, audio_we;
72      reg y_a_stren, y_b_stren, y_c_stren, y_d_stren, cr_stren, cb_stren,
73          audio_stren;
74      reg y_a_clear, y_b_clear, y_c_clear, y_d_clear, cr_clear, cb_clear,
75          audio_clear;
76
77      wire [1:0] channel;
78      wire [5:0] x;
79      wire [4:0] y;
80      wire [7:0] data;
81      wire we;
82      wire stren;
83      wire clear;
84
85      reg state;
86      reg [1:0] y_counter;
87
88      unpacker_fifo y_a_fifo(.reset(reset | y_a_clear), .clock(clock),
89          .din(y_a_din), .we(y_a_we), .stren(y_a_stren), .dout(y_a_data),
90          .valid_out(y_a_valid), .ready(y_a_ready));
91      unpacker_fifo y_b_fifo(.reset(reset | y_b_clear), .clock(clock),
92          .din(y_b_din), .we(y_b_we), .stren(y_b_stren), .dout(y_b_data),
```

```
93          .valid_out(y_b_valid), .ready(y_b_ready));
94      unpacker_fifo y_c_fifo(.reset(reset | y_c_clear), .clock(clock),
95          .din(y_c_din), .we(y_c_we), .stren(y_c_stren), .dout(y_c_data),
96          .valid_out(y_c_valid), .ready(y_c_ready));
97      unpacker_fifo y_d_fifo(.reset(reset | y_d_clear), .clock(clock),
98          .din(y_d_din), .we(y_d_we), .stren(y_d_stren), .dout(y_d_data),
99          .valid_out(y_d_valid), .ready(y_d_ready));
100     unpacker_fifo cr_fifo(.reset(reset | cr_clear), .clock(clock), .din(cr_din),
101         .we(cr_we), .stren(cr_stren), .dout(cr_data), .valid_out(cr_valid),
102         .ready(cr_ready));
103     unpacker_fifo cb_fifo(.reset(reset | cb_clear), .clock(clock), .din(cb_din),
104         .we(cb_we), .stren(cb_stren), .dout(cb_data), .valid_out(cb_valid),
105         .ready(cb_ready));
106     unpacker_fifo audio_fifo(.reset(reset | audio_clear), .clock(clock),
107         .din(audio_din), .we(audio_we), .stren(audio_stren), .dout(audio_data),
108         .valid_out(audio_valid), .ready(audio_ready));
109
110     unpacker_unwrapper unwrapper(.reset(reset), .clock(clock), .sdi(sdi),
111         .channel(channel), .x(x), .y(y), .data(data), .we(we), .stren(stren),
112         .clear_mem(clear));
113
114     always @(posedge clock)
115     begin
116         if (reset)
117         begin
118             state <= S_WAIT;
119             y_counter <= 0;
120         end
121         else
122         case (state)
123             S_WAIT: if (stren)
124             begin
125                 state <= S_STORE;
126
127                 case (channel)
128                     CHAN_Y:
129                     case (y_counter)
130                         0:
131                         begin
132                             y_a_x <= x;
133                             y_a_y <= y;
134                         end
135                         1:
136                         begin
137                             y_b_x <= x;
138                             y_b_y <= y;
139                         end
140                         2:
141                         begin
142                             y_c_x <= x;
143                             y_c_y <= y;
```

```
144                     end
145                     3:
146                     begin
147                         y_d_x <= x;
148                         y_d_y <= y;
149                     end
150                 endcase
151                 CHAN_CR:
152                 begin
153                     cr_x <= x;
154                     cr_y <= y;
155                 end
156                 CHAN_CB:
157                 begin
158                     cb_x <= x;
159                     cb_y <= y;
160                 end
161             endcase
162         end
163         S_STORE:
164         if (~stren)
165         begin
166             state <= S_WAIT;
167             if (channel == CHAN_Y) y_counter <= y_counter + 1;
168         end
169     endcase
170 end
171 
172 always @*
173 begin
174     if (state == S_STORE)
175     begin
176         case (channel)
177             CHAN_Y:
178             case (y_counter)
179                 0:
180                 begin
181                     y_a_din = data;
182                     y_a_we = we;
183                     y_a_stren = stren;
184                     y_a_clear = clear;
185                 end
186                 1:
187                 begin
188                     y_b_din = data;
189                     y_b_we = we;
190                     y_b_stren = stren;
191                     y_b_clear = clear;
192                 end
193                 2:
194                 begin
```

```
                    y_c_din = data;
                    y_c_we = we;
                    y_c_stren = stren;
                    y_c_clear = clear;
                end
                3:
                begin
                    y_d_din = data;
                    y_d_we = we;
                    y_d_stren = stren;
                    y_d_clear = clear;
                end
            endcase
            CHAN_CR:
            begin
                cr_din = data;
                cr_we = we;
                cr_stren = stren;
                cr_clear = clear;
            end
            CHAN_CB:
            begin
                cb_din = data;
                cb_we = we;
                cb_stren = stren;
                cb_clear = clear;
            end
            CHAN_AUDIO:
            begin
                audio_din = data;
                audio_we = we;
                audio_stren = stren;
                audio_clear = clear;
            end
        endcase
    end
  end
endmodule
```

D.36 unpacker_fifo.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Engineer:    Evan Broder
//
// Create Date: 02:15:47 12/04/2007
// Module Name: unpacker_fifo
// Project Name: Video-Conferencing System
// Description: Stores incoming data if there's not data in the FIFO already
//              The first bit on din should be set at the same time that we is
//              asserted
```

```
11 //              There is a one clock cycle delay between when read is asserted
12 //              and when the data comes out dout.
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // $Id: unpacker_fifo.v 144 2007-12-05 00:40:30Z evan $
18 // Additional Comments:
19 //
20 //////////////////////////////////////////////////////////////////////////////////
21
22 module unpacker_fifo(reset, clock, din, we, stren, dout, valid_out, ready);
23     parameter S_WRITE = 0;
24     parameter S_READ = 1;
25     parameter S_READ_WAIT = 2;
26
27     input reset;
28     input clock;
29     input [7:0] din;
30     input we;
31     input stren;
32     output dout;
33     output valid_out;
34     input ready;
35
36     reg [1:0] state;
37
38     reg [10:0] wptr;
39     wire [10:0] wptr_inc;
40     reg [13:0] rptr;
41     wire [13:0] rptr_inc;
42
43     wire bram_we;
44     wire negedge_stren;
45     wire empty;
46     wire full;
47
48     assign bram_we = we & (state == S_WRITE);
49
50     assign wptr_inc = wptr + 1;
51     assign rptr_inc = rptr + 1;
52
53     assign empty = (wptr == rptr[13:3]);
54     assign full = (wptr_inc == rptr[13:3]);
55
56     assign valid_out = state == S_READ;
57
58     level_to_pulse we_level(.reset(reset), .clock(clock), .level(~stren),
59         .pulse(negedge_stren));
60
61     unpacker_fifo_memory memory(.clka(clock), .clkb(clock), .dinb(din),
```

```
62          .douta(dout), .web(bram_we), .addrb(wptr), .addra(rptr));
63
64      always @(posedge clock)
65      begin
66          if (reset)
67          begin
68              wptr <= 0;
69              rptr <= 0;
70              state <= S_WRITE;
71          end
72          else
73          case (state)
74          S_WRITE:
75              if (negedge_stren) state <= S_READ_WAIT;
76              else if (we & ~full) wptr <= wptr_inc;
77          S_READ_WAIT:
78              if (ready)
79              begin
80                  state <= S_READ;
81                  rptr <= rptr_inc;
82              end
83          S_READ:
84              if (empty) state <= S_WRITE;
85              else rptr <= rptr_inc;
86          endcase
87      end
88  endmodule
```

D.37 unpacker_sar.v

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3   // Engineer:    Evan Broder
4   //
5   // Create Date: 20:32:16 12/02/2007
6   // Module Name: unpacker_sar
7   // Project Name: Video-Conferencing System
8   // Description: Asynchronously aligns the clock with the signal and receives
9   //              serial data.
10  //              Framing algorithm based on explanation of USART module in AVR
11  //              ATTiny2313
12  //              (SAR stands for Serial Asynchronous Receiver)
13  //
14  // Dependencies:
15  //
16  // Revision:
17  // $Id: unpacker_sar.v 193 2007-12-08 07:50:51Z evan $
18  // Additional Comments:
19  //
20  //////////////////////////////////////////////////////////////////////////////////
21
```

```
22 module unpacker_sar(reset, clock, sdi, rx, data);
23     parameter DATA_LEN = 8;
24 
25     parameter S_IDLE = 0;
26     parameter S_START = 1;
27     parameter S_DATA = 2;
28 
29     input reset;
30     input clock;
31     input sdi;
32     output rx;
33     output reg [7:0] data;
34 
35     wire sdi_debounce;
36     wire clk_enable;
37     wire rx_level;
38 
39     reg [3:0] sample_count;
40     reg [4:0] bit_count;
41     reg [1:0] state;
42 
43     reg [1:0] votes;
44 
45     debounce debounce(.reset(reset), .clock(clock), .noisy(sdi),
46         .clean(sdi_debounce));
47     defparam debounce.DELAY = 5;
48 
49     clock_divider divider(.reset(reset), .clock(clock), .enable(clk_enable));
50     defparam divider.D = 7;
51 
52     level_to_pulse rx_l2p(.reset(reset), .clock(clock), .level(rx_level),
53         .pulse(rx));
54 
55     always @(posedge clock)
56     begin
57         if (reset)
58         begin
59             state <= S_IDLE;
60             votes <= 0;
61             data <= 0;
62         end
63         else if (clk_enable)
64         begin
65             sample_count <= sample_count + 1;
66 
67             if (sample_count == 0) votes <= 0;
68             else if (sample_count == 6 || sample_count == 7 ||
69                 sample_count == 8) votes <= votes + sdi;
70 
71             case (state)
72                 S_IDLE:
```

```
73                begin
74                    if (~sdi_debounce) state <= S_START;
75                    sample_count <= 1;
76                    bit_count <= 0;
77                end
78                S_START:
79                begin
80                    if (sample_count == 0)
81                    begin
82                        if (votes[1] == 0) state <= S_DATA;
83                        else state <= S_IDLE;
84                    end
85                end
86                S_DATA:
87                begin
88                    if (sample_count == 10) data[bit_count] <= votes[1];
89                    else if (sample_count == 0) bit_count <= bit_count + 1;
90
91                    if (bit_count == DATA_LEN) state <= S_IDLE;
92                end
93            endcase
94        end
95    end
96
97    assign rx_level = (state == S_IDLE);
98 endmodule
```

D.38 unpacker_unwrapper.v

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Engineer:    Evan Broder
4  //
5  // Create Date: 12:08:03 11/18/2007
6  // Module Name: unpacker_unwrapper
7  // Project Name: Video-Conferencing System
8  // Description: Takes a serial incoming line and outputs all of the relevant
9  //              data. If the CRC check fails, the clear_mem flag gets asserted
10 //              just before stren is deasserted.
11 //
12 // Dependencies:
13 //
14 // Revision:
15 // $Id: unpacker_unwrapper.v 218 2007-12-10 21:40:37Z evan $
16 // Additional Comments:
17 //
18 //////////////////////////////////////////////////////////////////////////////////
19
20 module unpacker_unwrapper(reset, clock, sdi, channel, x, y, data, we, stren,
21        clear_mem);
22    parameter S_WAIT = 0;
```

```
23      parameter S_CHAN = 1;
24      parameter S_LEN_1 = 2;
25      parameter S_LEN_2 = 3;
26      parameter S_COORD_X = 4;
27      parameter S_COORD_Y = 5;
28      parameter S_DATA = 6;
29      parameter S_CRC = 7;
30
31      parameter CHAN_Y = 0;
32      parameter CHAN_CR = 1;
33      parameter CHAN_CB = 2;
34      parameter CHAN_AUDIO = 3;
35
36      input reset;
37      input clock;
38      input sdi;
39      output reg [1:0] channel;
40      output reg [5:0] x;
41      output reg [4:0] y;
42      output [7:0] data;
43      output reg we;
44      output stren;
45      output reg clear_mem;
46
47      wire rx;
48      wire [7:0] sat_data;
49      wire [7:0] crc;
50
51      reg [10:0] len;
52      reg [2:0] state;
53
54      wire stren_undelayed;
55
56      unpacker_sar sar(.reset(reset), .clock(clock), .sdi(sdi), .rx(rx),
57          .data(sat_data));
58      crc calc_crc(.reset(reset | (state == S_WAIT)), .clock(clock),
59          .data(sat_data), .crc(crc), .en(rx & (state == S_DATA) & (len != 0)));
60      delay stren_delay(.clock(clock), .undelayed(stren_undelayed),
61          .delayed(stren));
62      defparam stren_delay.DELAY = 1;
63
64      always @(posedge clock)
65      begin
66          if (reset) state <= S_WAIT;
67          else if ((state == S_WAIT) & rx & (sat_data == 8'hff)) state <= S_CHAN;
68          else if (state == S_CRC)
69          begin
70              state <= S_WAIT;
71              clear_mem <= (sat_data != crc);
72          end
73          else if (rx)
```

```
74          case (state)
75              S_CHAN:
76              begin
77                  channel <= sat_data;
78                  state <= S_LEN_1;
79              end
80              S_LEN_1:
81              begin
82                  len[10:8] <= sat_data;
83                  state <= S_LEN_2;
84              end
85              S_LEN_2:
86              begin
87                  len[7:0] <= sat_data;
88                  if (channel == CHAN_AUDIO) state <= S_DATA;
89                  else state <= S_COORD_X;
90              end
91              S_COORD_X:
92              begin
93                  x <= sat_data;
94                  state <= S_COORD_Y;
95              end
96              S_COORD_Y:
97              begin
98                  y <= sat_data;
99                  state <= S_DATA;
100             end
101             S_DATA:
102             begin
103                 if (len == 0) state <= S_CRC;
104                 else
105                 begin
106                     we <= 1;
107                     len <= len - 1;
108                 end
109             end
110         endcase
111         else
112         begin
113             we <= 0;
114             clear_mem <= 0;
115         end
116     end
117
118     assign stren_undelayed = (state == S_DATA) | (state == S_CRC);
119     assign data = sat_data;
120 endmodule
```