CHECKPOINT 4
Wireless Video Conferencing

1.0 Introduction
In this checkpoint you will complete the video conferencing system. The system works on a master/slave paradigm. Your final solution should be 1 bit file that can be toggled to operate as either the master or the slave. You may choose to implement your extra credit portions of this checkpoint in any manner you wish, however, you must also produce a solution that is compatible with the TA solution.

2.0 Prelab
1. Complete checkpoints 1, 2 and 3.
2. Begin your design as soon as possible; this will be a challenging checkpoint.
3. Complete your design review; we will still be holding design reviews for this checkpoint.

3.0 Lab Procedure

Our Advice
Beyond conforming to our communications protocol and the size/compression of the video, you may use whatever method you deem necessary to complete the checkpoint. However, you should be aware that the advice we offer in this document is likely to be extremely helpful if followed.

Division of Labor
It is strongly recommended that one partner works on the communications engine while the other works on the graphics engine and that the parts be connected when completed. While this is not required, it will help if one partner becomes an “expert” in the operation of one of the two modules.

Single Bit File
Just to head-off any questions on this topic, you must implement both the master and the slave sides of this project using only a single bit file that can toggle between operating as a master or a slave. You may not use 2 different bit files for each master and slave.
4.0 Checkpoint Overview
The overall block layout for the checkpoint will resemble the diagram below. Note that there may be additional buffering required between the DCT and Huffman Coder.

5.0 Communications Engine

Cross Compatibility
Your solution should be able to communicate with any other student’s solutions and the TA’s solution. Your solution should be able to communicate with solutions operating at different speeds relative to yours as well. This means that your solution must should event driven, and not operate based solely on counters. The wireless communication medium does not grant a consistent speed of operation at the nanosecond level, so your solution must be able to adapt and cope with changing conditions.

Packet-loss Robustness
Because of our compression technique, we must implement a lossless protocol. As a result, your solution should have a relatively low loss rate and should retransmit to ensure that all packets are received in order.

Handshaking
In order to initially establish a communication link and set the correct source and destination addresses, a handshaking methodology is needed. The one we will use proceeds as follows:
- At the beginning both the master and slave begin with their destination addresses being 0xFF and their source addresses being unique.
These packets are sent by the TA solution with delays of approximately 100ms and both the master and slave sides.

1. Master broadcasts SYN (Master connection request) packets once every 100ms (until a timeout occurs after a reasonable amount of time -- ~1s)
2. When a slave receives a SYN packet it sends a SYNACK (Slave request ack) with a destination address set to the source address of the master’s SYN packet.
3. When the master receives the SYNACK packet it changes its destination address to the source address of the slave and then transmits a DATA packet with the new destination address and sequence number 0.
4. When the slave receives the first DATA packet the slave sends a DATA packet with sequence number 0.
5. When the master gets the valid (i.e. correct sequence number) slave DATA packet, it should increment the sequence number by one and transmit the next DATA packet.
6. This alternates back and forth where DATA packets are resent if a response DATA packet with the correct sequence number is not received. The next DATA packet is loaded and sent with the next sequence number upon receipt of a valid DATA packet.
7. If no data packet at all is received after 200ms, the connection is “timed out” and both sides should stop transmitting.

NOTE: It is not necessarily for both the master and slave to retransmit if a data packet is not received; only one side has to retransmit.

Communication Protocol
The table below describes how the 256-bits of packet payload should be allocated to transmit each type of data.

- The 5-bit type designates the type of packet.
- The 3-bit sequence number increments for each new DATA packet sent and wraps
- The 248-bit Data Payload contains the compressed video data as a bitstream for DATA packets and no data for all other types of packets.
- The first data packet sent by the Master and the Slave should start with the sequence number 3'b0.

<table>
<thead>
<tr>
<th>Type</th>
<th>5-Bit Type</th>
<th>3-Bit Seq. Number</th>
<th>248-Bit Data Payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYN</td>
<td>8'd1</td>
<td>3'bX</td>
<td>248'bX</td>
</tr>
<tr>
<td>SYNACK</td>
<td>8'd2</td>
<td>3'bX</td>
<td>248'bX</td>
</tr>
<tr>
<td>DATA</td>
<td>8'd3</td>
<td>Counter</td>
<td>248 bits of the video bitstream</td>
</tr>
</tbody>
</table>
6.0 Video Compression/Decompression

SDRAM Arbiter Modifications and Display Processor
See Checkpoint 2.5 documentation. You must complete all requirements of Checkpoint 2.5 for Checkpoint 4.

DCT
A blackboxed DCT will be provided to you. The interface with the blackbox is outlined on the table below:

<table>
<thead>
<tr>
<th>Signal</th>
<th>Width</th>
<th>Dir.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ND</td>
<td>1</td>
<td>I</td>
<td>Pulse to indicate that the next piece of data on DIN is valid</td>
</tr>
<tr>
<td>RDY</td>
<td>1</td>
<td>O</td>
<td>Indicates the output on DOUT is valid</td>
</tr>
<tr>
<td>RFD</td>
<td>1</td>
<td>O</td>
<td>Indicates that the DCT is ready for data on DIN</td>
</tr>
<tr>
<td>CLK</td>
<td>1</td>
<td>I</td>
<td>Clock</td>
</tr>
<tr>
<td>RST</td>
<td>1</td>
<td>I</td>
<td>Resets the DCT</td>
</tr>
<tr>
<td>DIN</td>
<td>8</td>
<td>I</td>
<td><strong>Unsigned input, put all 8 bits of your Luma value into here.</strong></td>
</tr>
<tr>
<td>DOUT</td>
<td>16</td>
<td>O</td>
<td>DCT coefficients as 2’s complement signed values</td>
</tr>
</tbody>
</table>

Note that the inputs will arrive in block order. It is up to the user of the DCT to know when a full 8x8 block has been sent into the DCT. This is accounted for in your CP2.5 address counter.

IDCT
A blackboxed IDCT will also be provided to you. The interface with the blackbox is outlined on the table below:

<table>
<thead>
<tr>
<th>Signal</th>
<th>Width</th>
<th>Dir.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ND</td>
<td>1</td>
<td>I</td>
<td>Pulse to indicate that the next piece of data on DIN is valid</td>
</tr>
<tr>
<td>RDY</td>
<td>1</td>
<td>O</td>
<td>Indicates the output on DOUT is valid</td>
</tr>
<tr>
<td>RFD</td>
<td>1</td>
<td>O</td>
<td>Indicates that the DCT is ready for data on DIN</td>
</tr>
<tr>
<td>CLK</td>
<td>1</td>
<td>I</td>
<td>Clock</td>
</tr>
<tr>
<td>RST</td>
<td>1</td>
<td>I</td>
<td>Resets the DCT</td>
</tr>
<tr>
<td>DIN</td>
<td>16</td>
<td>I</td>
<td>DCT coefficients as 2’s complement signed values</td>
</tr>
<tr>
<td>DOUT</td>
<td>52</td>
<td>O</td>
<td>Resulting image data. This is also as a 2’s complement value, but since we don’t want negative values you can treat negative values as 0’s and extract bits [34:27] of any positive values to get an 8-bit value that can be used as your luma value. Check bit 51 to determine the sign of the output.</td>
</tr>
</tbody>
</table>

Quantization
Since our image is not very sensitive to slight changes in the frequency domain, we can quantize the output of the DCT by decreasing the precision of the DCT output. We can also exploit the fact that the values output by the DCT for our inputs (which are all positive) will never exceed a certain maximum. As a result, the output of the DCT that goes into the Huffman Coder will only use the bits [11:3], effectively dividing the
coefficients by 8 and limiting their range. Note that bit 11 is only used to determine the sign of the 2’s complement value, so it is equivalent to using \( \{\text{dctout}[16], \text{dctout}[10:3]\} \). It may also be useful to convert this value to a sign-and-magnitude number to simplify the computation done by the Huffman Coder. **Any transformation made to the original DCT coefficients must be undone on the other end of the wireless channel (except for restoring the bottom bits, which is lossy but tolerable).**

**Huffman Coder**

The Huffman Coder module shown in the block diagram above implements a coding scheme that very closely resembles true Huffman Coding based on the frequency of values in the DCT coefficients of most image data. The coding scheme uses a signed unary binary code and should work as follows:

1. If the input is zero, output a single bit 1 and ignore the remaining steps.
2. If the input is nonzero, generate N bits of 0’s where N is the number of bits required to implement the magnitude of the input. This is one less than the number of bits required to represent the signed number.
3. Generate the magnitude of the input. This is the unsigned version of |Input|. Only use as many bits as is necessary to represent the number, so this must start with a 1 in its most significant bit.
4. Generate 1’b1 if the input is negative and 1’b0 if the input is positive.

Note this should take multiple clock cycles as the input is a parallel 9-bit value and the output is up to 17 bits in serial. Most of the inputs will be 0, so expect many 1 bit outputs.

It is recommended that you do this using a standard handshaking protocol such as the InRequest/InValid or Ready/Start handshaking from Checkpoint 3.

A table corresponding to the steps above is shown below:

<table>
<thead>
<tr>
<th>Number</th>
<th>Zeros</th>
<th>Magnitude</th>
<th>Sign</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>00</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>-2</td>
<td>00</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>000</td>
<td>101</td>
<td>0</td>
</tr>
<tr>
<td>-20</td>
<td>00000</td>
<td>10100</td>
<td>1</td>
</tr>
</tbody>
</table>

**Buffering**

Since the DCT will output up to 64 values continuously, it is necessary to buffer the output of the DCT before sending it into the Huffman Coder. Similarly, your FIFO’s connecting to your Checkpoint 2.5 and Checkpoint 3 should also be buffered. You should ensure that these FIFOs never overflow or underflow by regulating the rate at which full 8x8 subimages are sent into your DCT.
7.0 Hints and Tips

1. Start early! This is by far the hardest checkpoint of the project. If you're already
done with checkpoint 3, start designing for this checkpoint immediately. Note
that we are not giving you any code for this checkpoint besides the DCT and
IDCT blackboxes. You will need to write all the Verilog on your own!

2. Split up the project into communications and graphics. These two parts are fairly
independent and come together via the DCT and Huffman Coder so you and your
partner can work in parallel.

3. Work incrementally. Do not write thousands of lines of code and test it all at
once. The following is a rough list of testable milestones to accomplish on the
video side. Note that you can implement all of these independently of the
communications module:
   a. Checkpoint 2.5
   b. Test compression at each stage by looping back via a FIFO. You can test
      just the DCT looped through the IDCT, DCT+Huffman looped back
      through inverse Huffman+IDCT, etc.
   c. Test the sender and receiver FSMs by sending local “packets” to yourself.