University of California: Berkeley, Electrical Engineering & Computer Science

Wireless Video Conferencing

EECS150: Components and Design Techniques for Digital Systems

Farzad Fatollahi-Fard & Eric Zhou

08

Table of Contents

I General Overview

1 Abstract

The EECS150 project for the Spring of 2008 consisted of designing and implementing a *Wireless Video Conferencing System* on the CaLinx 2 board using Verilog. The final design of the project involves two CaLinx 2 boards connected to a camera and a television communicating over wireless, displaying local video as well as compressed video from the other board. Due to the low bandwidth of the wireless transceiver, the remote video is low resolution and grayscale, and has an extremely low frame rate. However, if implemented correctly, you can achieve a bidirectional video conferencing system, as this document will provide.

2 Overview

Our project was a bidirectional wireless video conferencing system. Each node has a video camera as well as a television. Both nodes show the same thing: the upper-left corner shows local sub-sampled grayscale video; the upper-right corner displays remote compressed video. As a result of this duality of seeing local as well as remote video, we can call this a bidirectional video conferencing system.

Figure 1 shows a high-level diagram of the designed system. As the diagram shows, there are three basic blocks: the transceiver module, the graphics module, and the SDRAM module.

The SDRAM module contains the SDRAM controller, which communicates with the SDRAM, and the SDRAM Arbiter. The SDRAM Arbiter has four data request ports: Video Encoder/Picture-In-Picture, Video Decoder/Sub-sampler, Wireless Transmission/Video Compression, and Wireless Reception/Video Decompression. The Arbiter controls which requests to the SDRAM get fulfilled by assigning priority to each request whilst making sure the modules don't overflow or underflow with data.

The wireless module contains the Transceiver controller, which communicates with the wireless card via a SPI interface, and the Protocol module. The Protocol determines the communication between the two nodes. One node must be set to "Master" and the other must be set to "Slave". Once the Protocol module establishes a connection with another node, the video is continuously streamed between the two nodes until one of the nodes times out (either from a board reset or bad wireless connection).

The graphics module contains the modules that communicate with the television and the camera, and also contains the video compression modules. The Video Decoder/Sub-sampler takes data from the camera, subsamples it, converts it to grayscale, and sends a write request to the SDRAM Arbiter. The Wireless Transmission/Video Compression module reads data from the SDRAM and compresses it using a Discrete Cosine Transform (DCT) and a Huffman encoder, after which it is sent to the Wireless module. The Wireless Reception/Video Decompression receives data coming from the Wireless module and decompresses it with a Huffman decoder and an Inverse DCT, after which the data is sent to the SDRAM Arbiter for writing to the SDRAM. Finally, the Video Encoder/Picture-In-Picture module reads the sub-sampled local video and the remote video from the SDRAM and displays it to the screen.

The connection of these three top-level modules is the culmination of the EECS150 project.

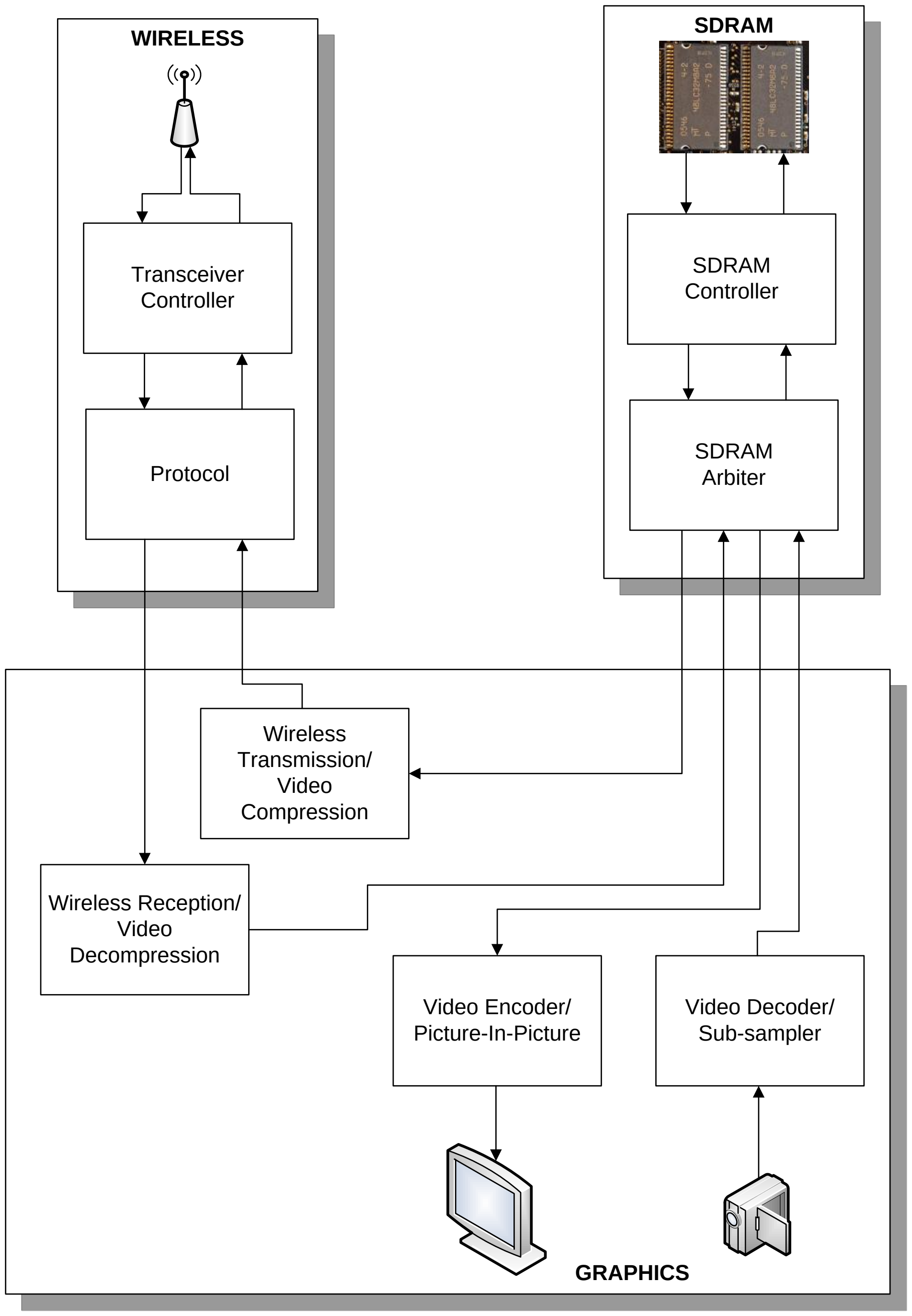
WIRELESS
Transceiver Controller
Protocol
SDRAM
SDRAM Controller
SDRAM Arbiter
Wireless Transmission/ Video Compression
Wireless Reception/ Video Decompression
Video Encoder/ Picture-In-Picture
Video Decoder/ Sub-sampler
GRAPHICS

Figure 1 - General Overview

II System Description

1 SDRAM Module

The SDRAM on the CaLinx 2 is a Micron Technologies SDRAM Chip (MT48LC16M16A2TG-7E). As mentioned in the overview, the SDRAM is comprised of two components: the SDRAM controller and the arbiter. The purpose of the controller is that it provides an interface between the user and the chip, while the arbiter manages the signals going into the controller.

1.1 SDRAM Controller

The purpose of designing an SDRAM controller, as mentioned above, is to provide a reliable interface to the SDRAM chip. This was the first, and probably one of the most key, modules that was written for this project, as the SDRAM was where the video information was going to be stored. Using the Micron datasheet, we designed a state machine to handle any reading or writing that could be given to the SDRAM (See Figure 2).

When designing the controller, we opted for a sequential design as opposed to a pipelined design. The reason is that a sequential design is much simpler to implement than a pipelined design. Furthermore, a pipelined controller seemed to be too much for a simple project as this.

Another design consideration was refreshing memory. Since the chip is a Synchronous DRAM chip, we would be susceptible to data loss due to the volatility of the chip. Therefore, to insure integrity of the data saved in the SDRAM, we need to periodically refresh the SDRAM. However, on every read to the SDRAM, the data being read is refreshed. Since we will be frequently reading from the SDRAM to maintain the image on the screen, the problem date integrity is solved.

1.2 SDRAM Arbiter

The arbiter was probably one of the most key modules in the project. Since the SDRAM controller can only handle one read/write request at a time, only one read/write request can be issued to the controller at a time and it has to be ensured that the request is complete before the next command is issued. The arbiter must also assign priorities to different requests to make sure that their buffers don't overflow or underflow.

Figure 3 is the state diagram for our SDRAM Arbiter. As the diagram points out, when a command is issued, the arbiter waits until it receives a "MemDone" signal from the controller before issuing any other commands. This ensures that only one request is fulfilled at a time.

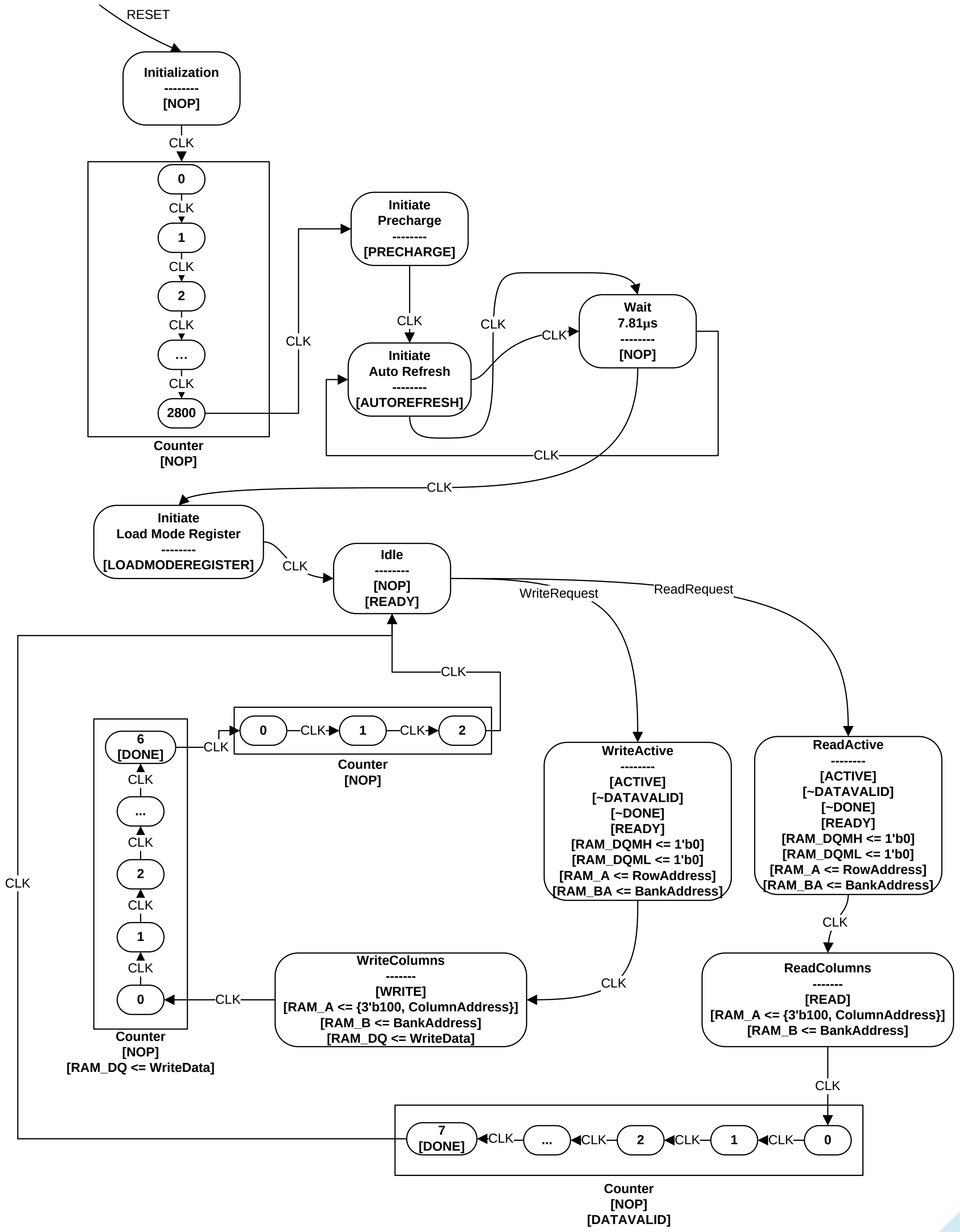
RESET
Initialization -------- [NOP]
CLK
0
1
2
…
2800
Counter [NOP]
Initiate Precharge -------- [PRECHARGE]
Initiate Auto Refresh -------- [AUTOREFRESH]
Wait 7.81µs -------- [NOP]
Initiate Load Mode Register -------- [LOADMODEREGISTER]
Idle -------- [NOP] [READY]
WriteRequest
ReadRequest
WriteActive -------- [ACTIVE] [~DATAVALID] [~DONE] [READY] [RAM_DQMH <= 1'b0] [RAM_DQML <= 1'b0] [RAM_A <= RowAddress] [RAM_BA <= BankAddress]
ReadActive -------- [ACTIVE] [~DATAVALID] [~DONE] [READY] [RAM_DQMH <= 1'b0] [RAM_DQML <= 1'b0] [RAM_A <= RowAddress] [RAM_BA <= BankAddress]
WriteColumns ------- [WRITE] [RAM_A <= {3'b100, ColumnAddress}] [RAM_B <= BankAddress] [RAM_DQ <= WriteData]
ReadColumns ------- [READ] [RAM_A <= {3'b100, ColumnAddress}] [RAM_B <= BankAddress]
6 [DONE]
Counter [NOP] [RAM_DQ <= WriteData]
Counter [NOP]
7 [DONE]
Counter [NOP] [DATAVALID]

Figure 2 - SDRAM Controller FSM

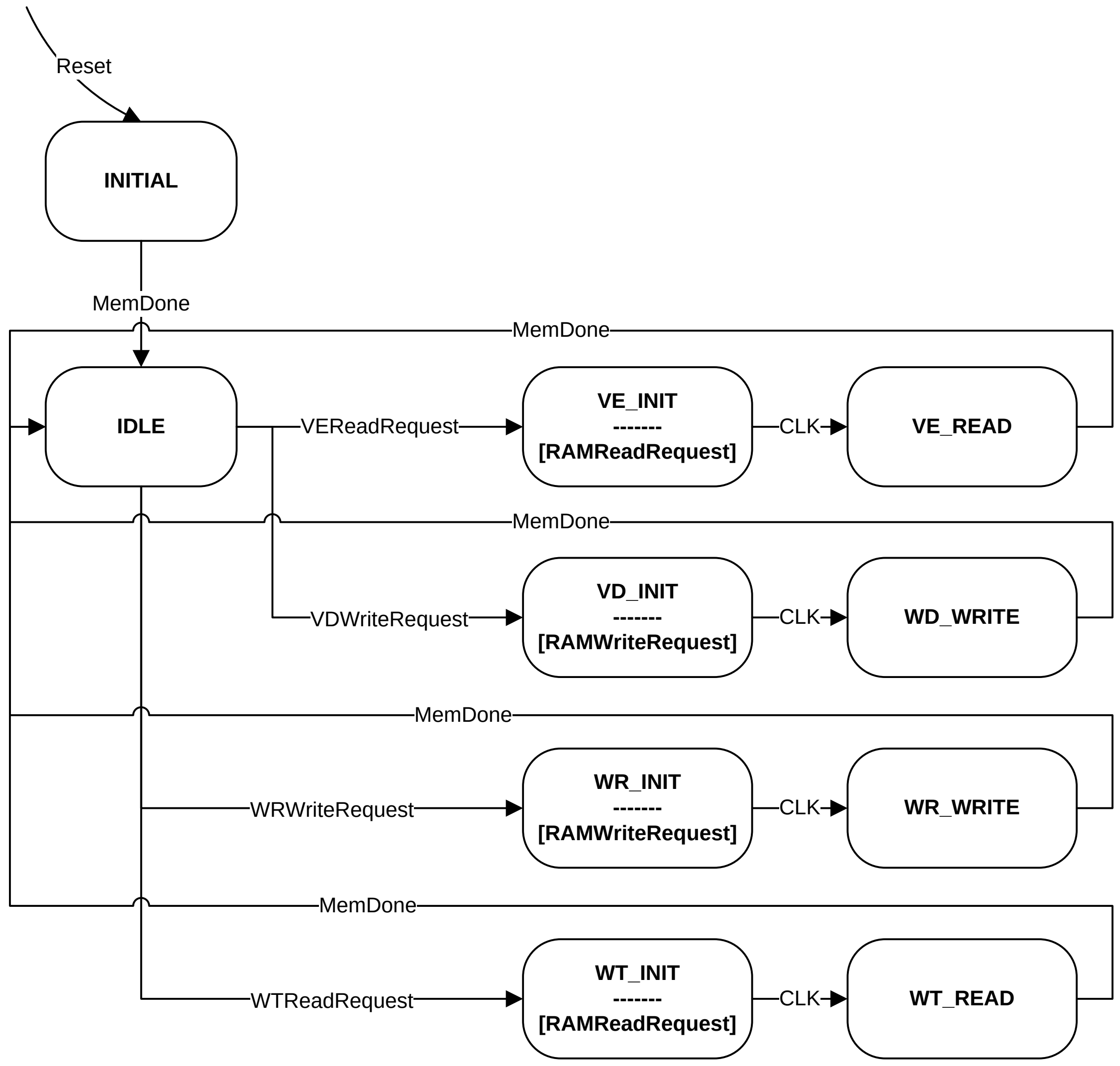

Figure 3 - SDRAM Arbiter FSM

The priorities were given in the following order: Video Encoder, Video Decoder, Wireless Receive, and Wireless Transmit. The Video Encoder was given the highest priority because the Video Encoder is in constant need of data to update the television screen. The Video Decoder was second in priority because the Video Decoder is constantly sending data. The reason it was given put in second priority is because there are a few cycles when the Video Decoder doesn't send data, at which time the Sub-sampler is idle (the following section will elaborate on the reason). The Wireless Receiver was put in the third position because the Transceiver system is extremely slow as compared to the Video Encoder/Decoder. The reason it was given higher priority than the Wireless Transmitter is because we want to ensure that we save every received piece of data sent.

With this arbitration scheme and design, we managed to maintain FIFO stability and managed not to overwhelm the SDRAM Controller.

2 Wireless Module

The Transceiver chip on the CaLinx 2 is a Chipcon RF Transceiver CC2420. As mentioned in the overview, the Wireless module is comprised of two components: the Transceiver controller and the Protocol. The purpose of the controller is that it provides an interface between the user and the chip, while the Protocol maintains a standard protocol for communication between two nodes. See Figure 4 for a general overview of the Wireless module.

2.1 Transceiver Controller

The purpose of designing a Transceiver controller, as mentioned above, is to provide a reliable interface to the Transceiver chip. The Transceiver chip uses a Serial Peripheral Interface (SPI) to initialize, configure, and send/receive data. To communicate through the SPI, we have dedicated SPI states, which communicate directly with the SPI.

The Transceiver controller has five high-level states: Initialization, Channel Changing, Transmission, and Receiving. With the exception of the Receiving states, they all use the SPI states to send commands and data to the Transceiver chip.

For a general picture of, see Figure 5.

2.2 Protocol

The purpose of Protocol module is to create a standard of communication between the two communicating nodes. It breaks down the 256-bit input to the Transceiver into the following way: 5-bits for Packet Type, 3-bits for Data count, and 248-bits for Payload. There are three different packet types: SYN (for initiating connections), SYNACK (for acknowledging connection initiations), and DATA. The Data count is incremented every time a packet is sent and a new packet is received. The payload is the actual data received from the Graphics module, which is sent when the node is sending DATA packets. If there isn't any data ready to send, the Protocol waits until there is data.

The Protocol can be broken down into two parts, a "Master" and a "Slave". When the Protocol is in Master mode, it sends SYN packets until it receives a SYNACK packet, at which point it starts sending DATA packets. When the Protocol is in Slave mode, it listens on the channel for SYN packets. When it receives one, it replies with a SYNACK packet, and waits for a DATA packet, after which it sends its own DATA packet. See Figure 6 for the Protocol State Machine.

The Protocol waits 10 milliseconds before resending the DATA packet. After a 20 resends (200 milliseconds), the Protocol State Machine times out and the connection is closed.

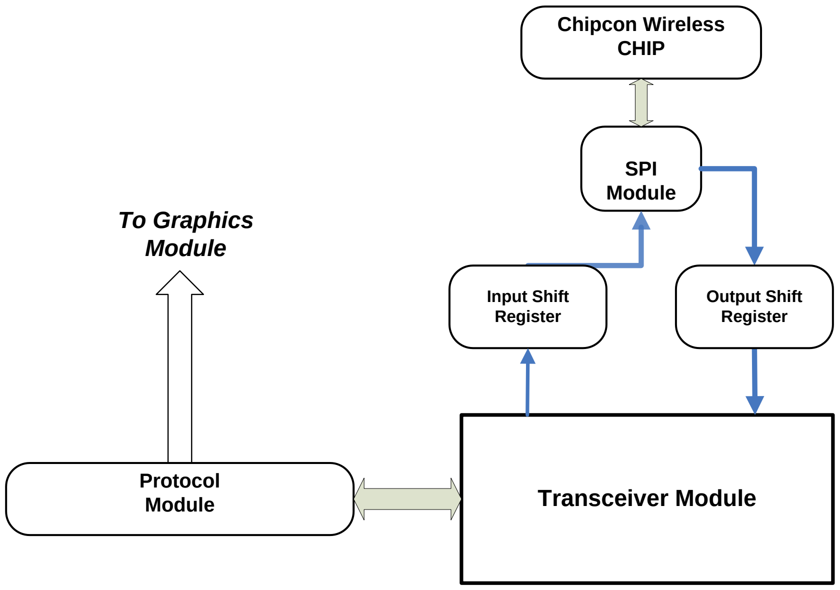

Figure 4 - Wireless Block Diagram

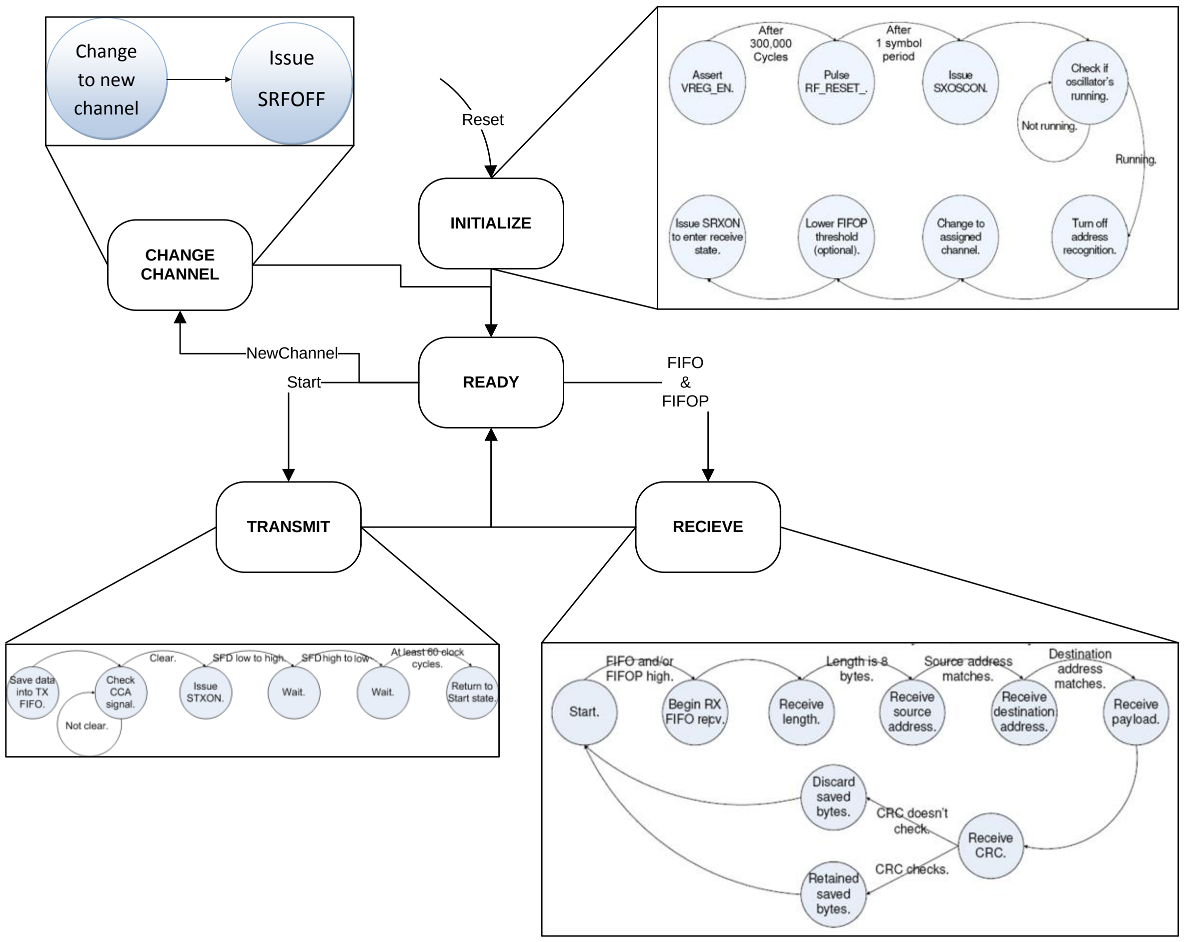

Figure 5 - Transceiver Controller FSM

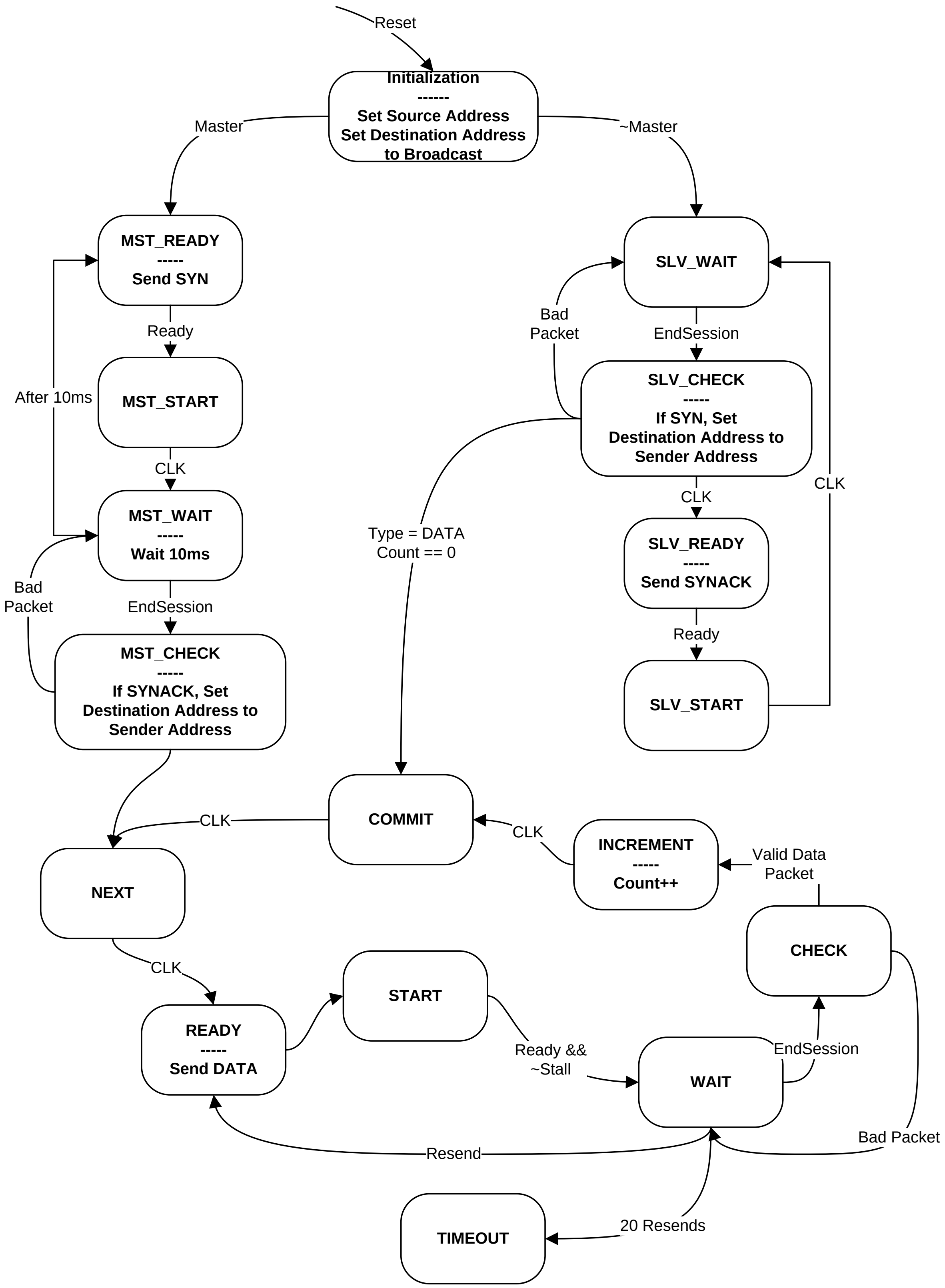

Figure 6 - Protocol Module FSM

3 Graphics Module

This module is broken down into four sub-modules: Video Decoder/Sub-sampler, Video Encoder/Picture-In-Picture, Wireless Transmission/Video Compression, and Wireless Reception/Video Decompression. Figure 8 has block diagram of these modules as well as other handshaking/buffering modules.

3.1 Video Decoder/Sub-sampler

The Video Decoder/Sub-sampler module is responsible for reading data from the camera and initiating write requests to store that data into the SDRAM. The module contains an address counter to determine where to write to RAM, and a FIFO for buffering. The FIFO saves data whenever the Video Decoder sends in a signal, and writes to RAM when the FIFO data count is greater than one. The address counter runs through the address one entire row at a time. The Sub-sampler outputs a pulse whenever it finishes a full frame. The Sub-sampler only samples the Video Decoder signal on every fourth line and every fourth pair. Due to the standard of the camera, it gives the odd lines of video then returns the even lines of data after a short delay. We had to compensate for this delay by keeping the Sub-sampler in idle mode while the Video Decoder was in this range. This was reason we gave this module lower priority in the SDRAM Arbiter.

Writing into the Sub-sampler handshakes with the Wireless Transmission/Video Compression module; this will be described below in the Wireless Transmission/Video Compression section.

3.2 Video Encoder/Picture-In-Picture

The Video Encoder/Picture-In-Picture reads data from SDRAM and outputs it to the television. Our Graphics module has two Video Encoder/Picture-In-Picture modules, one to read the local video data in the SDRAM, and one to read the remote video data from the SDRAM, both of which are stored in different SDRAM banks. The module contains an address counter and a FIFO, which is written into whenever its data count is less than or equal to two, and is read from whenever the Video Encoder demands it. The Video Encoder/Picture-In-Picture that is read to screen is determined by the “InRequestPair” and “InRequestLine” generated by the Video Encoder, denoting current position on the screen.

The Video Encoder/Picture-In-Picture that reads the remote video uses the double-buffering generated by the Wireless Reception/Video Decompression module (see the Wireless Reception/Video Decompression module description to learn more about double buffering).

3.3 Wireless Transmission/Video Compression

The Wireless Transmission/Video Compression module is responsible for reading data from SDRAM and sending it to the Wireless module. This module consists of several components: an address counter to determine where it reads the data from, and a FIFO for buffering purposes.

When the FIFO data count is less or equal to two, a ReadRequest signal is sent from the Wireless Transmission/Video Compression to the SDRAM Arbiter, to be fulfilled according to the Arbiter's arbitration scheme. Data is sent to the Discrete Cosine Transform (DCT) module by activating the FIFO read enable signal when the following conditions are true: the FIFO data count is greater or equal to one, the data count of the FIFO located between the DCT and Huffman Encoder module is less than or equal to one.

The address counter in the Wireless Transmission/Video Compression module is different from the one found in the Sub-sampler and the Picture-In-Picture modules. Instead of incrementing the address line by line, the address counter increments in 8×8 blocks.

In order to prevent the Sub-sampler from overwriting the data in SDRAM many times before the Wireless module can process a full frame, the Wireless Transmission/Video Compression and the Sub-sampler will alternate turning on and off using a hand-shake, which is pulsed once the Sub-sampler has processed a full frame, as the signal. Wireless Transmission/Video Compression will process a frame; stop as Sub-sampler processes a frame, before turning on again and processing another frame, and so on.

3.4 Wireless Reception/Video Decompression

The Wireless Reception/Video Decompression module is responsible for writing data received from wireless into the SDRAM. It consists of an address counter identical to the one used in Wireless Transmission/Video Compression, as well as a FIFO for buffering.

Wireless Reception/Video Decompression sends a write request to the SDRAM arbiter whenever the data count of its FIFO is equal or greater than two. However, the write enable signal of its FIFO does not take into account the data count, writing into the FIFO whenever the Inverse DCT (IDCT) module outputs a value. However, given the slow rate of wireless data transmission, as well as the fact that the Wireless Reception/Video Decompression FIFO data count is taken into account for data going into the IDCT, this is not a problem.

This module implements double buffering along with the Picture-In-Picture module. Every time the Wireless Reception/Video Decompression finishes reading in a frame, it switches banks in the SDRAM. In the meantime, the Picture-In-Picture module is switched to the bank that the Wireless Reception/Video Decompression is not on.

3.5 Huffman Encoder/Huffman Decoder

The Huffman Encoder is designed to convert the incoming data from the DCT to a prefix-free encoding. The Huffman Encoder module will also decrease the number of bits needed to be transmitted over wireless since the bit length for each symbol is inversely proportional to the frequency of that symbol outputted by the DCT module. For example, the most common symbol, zero, takes only one bit when put through the Huffman Encoder. The Huffman Encoder uses the following scheme: for zero, output one; for all nonzero numbers, output a number of zeroes equal to the minimum number of bits required to represent that number, followed by that number represented in that number of bits, and lastly one bit indicating the sign of the number. The Huffman Decoder module does the same but in reverse. See Figure 7 for state machine of each.

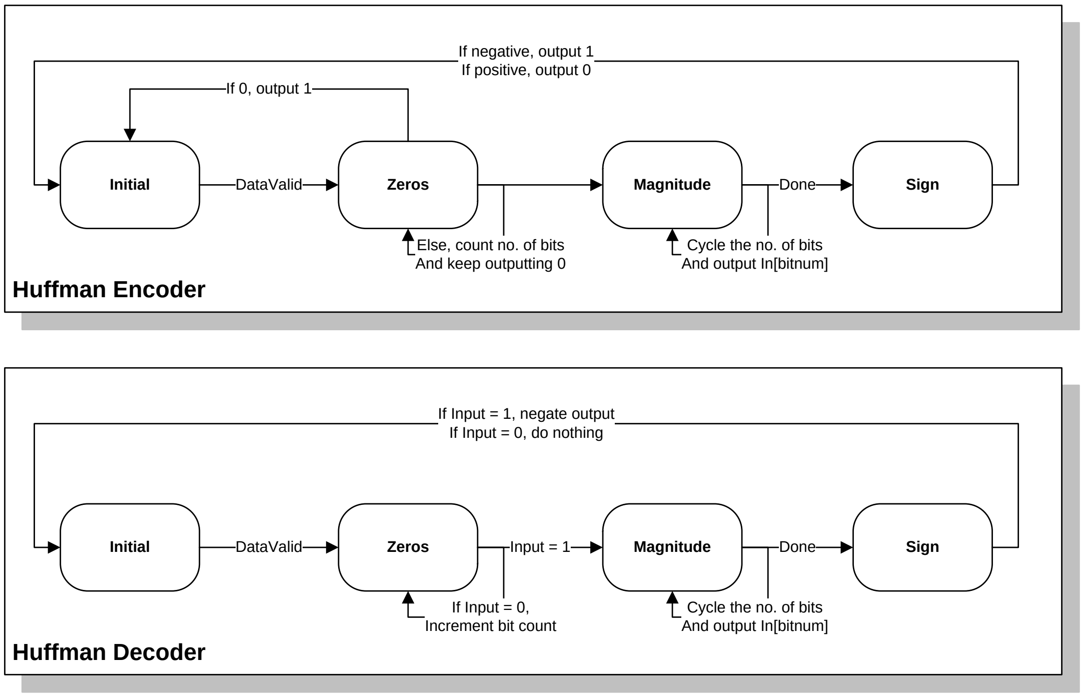

Figure 7 - Huffman Encoder/Decoder FSMs

3.6 Packetizer/De-Packetizer

The Packetizer takes in the 1-bit data stream from the Huffman encoder, and with a shift register, groups it into packets of 248 bits to be sent to the Wireless module. The De-Packetizer does the opposite, taking in packets of 248 bits and breaking it into a 1-bit stream.

3.7 Buffering FIFOs

The Graphics module contains three other FIFOs: one between the DCT and Huffman Encoder modules, one between the Packetizer and Wireless module input, and one between the Huffman Decoder and IDCT modules.

The FIFO between the DCT module and Huffman Encoder follows the following handshaking conditions:

- Is written into when whenever the DCT is ready to output values.
- Is read from when data count is greater than or equal to two, the Huffman Encoder is ready to receive a new input, and when the FIFO between the Packetizer and Transceiver output has a data count of less than six.

The FIFO between the Packetizer and Wireless module input obeys the following handshaking conditions:

- Is written into whenever the Packetizer is ready to output a packet.
- Is read from whenever the Transceiver is ready for an input and when the data count is greater than 5.

The FIFO between the Huffman Decoder module and IDCT module obeys the following handshaking conditions:

- Is written into whenever the inverse Huffman outputs data.
- Is read from whenever IDCT is ready to receive data, the FIFO data count is greater or equal to 2, and the FIFO data count in the Wireless Reception/Video Decompression module is less than 2.

These handshaking conditions allow data to travel through the Graphics module without causing any unnecessary data loss or FIFO overflows.

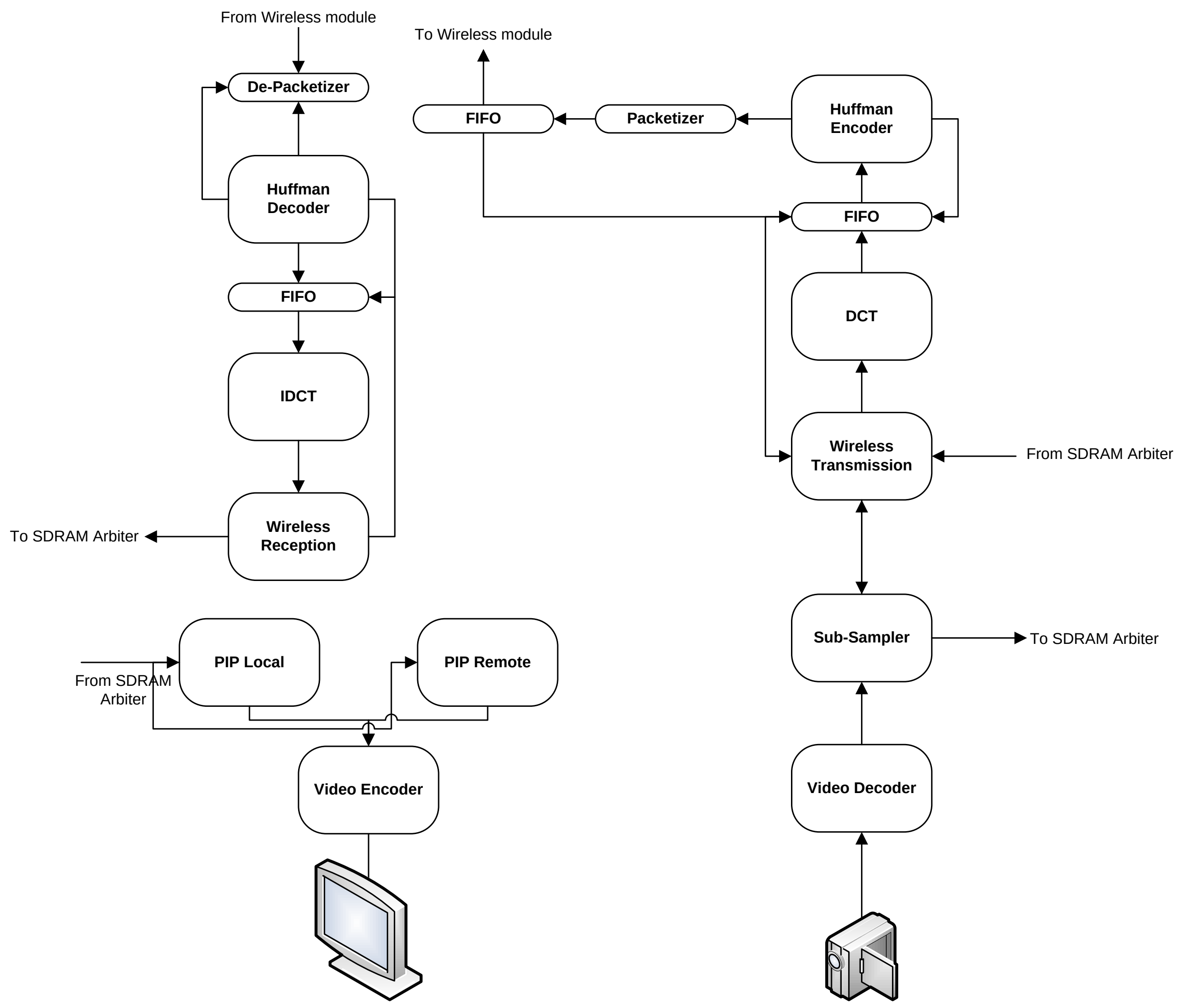

Figure 8 - Block Diagram of Graphics Module

III Conclusion

1 Design Metrics

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops	12,151	38,400	31%
Number of 4 input LUTs	9,492	38,400	24%
Logic Distribution			
Number of occupied Slices	7,855	19,200	40%
Number of Slices containing only related logic	7,855	7,855	100%
Number of Slices containing unrelated logic	0	7,855	0%
Total Number 4 input LUTs	10,091	38,400	26%
Number used as logic	9,492		
Number used as a route-thru	383		
Number used as 16x1 ROMs	23		
Number used as Shift registers	193		
Number of bonded IOBs	210	512	41%
IOB Flip Flops	14		
Number of Tbufs	4	19,520	1%
Number of Block RAMs	31	160	19%
Number of GCLKs	2	4	50%
Number of GCLKIOBs	1	4	25%
Total equivalent gate count for design	707,628		
Additional JTAG gate count for IOBs	10,128		

Table 1 - Device Utilization Summary

Timing Constraints				
Clocks	**Requested**	**Actual**	**Logic Levels**	**Absolute Slack**
System Clock (Y5_CLK)	37.037ns	36.245ns	1	0.792ns
Video Decoder Clock (VD_LLC_1_)	37.037ns	17.667ns	5	19.370ns
PS/2 Clock (PS2_CLK)	37.037ns	7.679ns	0	29.358ns

Table 2 - Timing Constraint Summary

Checkpoint	Design	Coding	Debugging	Total
1	2 hrs	3 hrs	6 hrs	11 hrs
2	2 hrs	3 hrs	6 hrs	11 hrs
2.5	2 hrs	5 hrs	20 hrs	27 hrs
3	4 hrs	10 hrs	15 hrs	29 hrs
4	6 hrs	15 hrs	20 hrs	41 hrs
Total	16 hrs	36 hrs	67 hrs	119 hrs

Table 3 - Estimated Hours Spent on Project

2 Acknowledgements

Firstly, we would like to give thanks to our fellow student/comrades who helped us through the project and through the course. It really created a helpful and friendly atmosphere that really helped seed a helpful atmosphere.

We would also like to acknowledge Greg Gibeling for providing some data sheets as well as many of the Verilog files that we used throughout the project.

3 Conclusion

Over the course of this semester, we were able to implement a basic working version of a two-way video conferencing system using the CaLinx 2 boards provided in lab. Through working on this project, we were able to use and apply all of the knowledge and techniques that we learned during class and the early labs. The project allowed us to truly appreciate the intricacies and complexities of accomplishing a seemingly simple task like displaying video. Through the various checkpoints, we learned to interface with SDRAM, subsample data from a camera and display it on the monitor, and use handshaking to send and receive packets of information across a not always reliable wireless connection.

Major bumps and bugs we encountered occurred during Checkpoints 3, where we had to get the timing just right in order to successfully send and receive data packets through the wireless transceiver. More roadblocks occurred during Checkpoint 4, where the seemingly simple task of hooking up a series of modules to transmit data turned into a major handshaking nightmare.

Our final product, following the project specs, is able to display a sub-sampled grayscale local video as well as a compressed video of another system located roughly ten feet away. However, the video, due to slow rate of transmission, has a very slow frame rate, sometimes reaching less than one frame a second. Also, the system is prone to timeout after a certain period of time.

As an extra feature, we were able to program the CaLinx 2 to accept keyboard inputs through the PS/2 port and to display inputted text on the LCD screen found on the CaLinx 2. Had we more time, we would have wanted to display this text to the computer monitor and then transmit it along with the video to form a simple instant-messaging interface.

As for what we would do differently next time, knowing what the project entails, we would certainly want to finish the first two checkpoints, not on time, but ahead of time, in order to give more time for the challenge that is the later two checkpoints. Also, for certain parts in the project we would, in an attempt to get things done quicker, dive right in before completely understanding all the complexities of a certain task. Occurrences like these were apt to cause much grief as we had to go back, revise, and make messy changes.

4 Suggestions

We were assured at the beginning of the semester that signing up for a particular lab section will not have any effect on the time we get to work on a certain checkpoint, however it was utterly false. In principal, if a checkpoint specs were posted on time all lab sections would have had a week or more to work on a checkpoint. Unfortunately for students in early lab sections, the specs were posted days after they were promised. This is more of grievance but I feel it belongs here so that future semesters do not suffer the same fate.

The design specs posted had more than a few inconsistencies and bugs. Even more hurting to our work was the fact even the Teaching Assistants were off sync. We received different answers from different TAs for the same question, only to hear an indifferent apology from one later on.

One of the most disappointing facets to the course was that the final project was not decided until after Checkpoint 2. We could be wrong regarding the exact date when the project was finalized; but as far as the response we got from TAs when we inquired about the final project, we started getting a consistent reply only after Checkpoint 2. We had heard good things about management of the class from previous semesters. Most of those semesters had introduction to the final project during the first two lectures usually.

We were warned about the project being heavily back-loaded, but we think it could have been avoided. This again ties back to me last suggestion of having a concrete idea about the project by the time the semester begins.

Another factor that was slightly disconcerting was that, during the first lecture, the Professor gave us the notion that this was going to be a course dedicated to glue logic. However, as the semester went on, it seemed like the checkpoints' focus was more on state machines than on actual glue logic. It would be nice to have that glue logic experience for course on glue logic.

A big step towards improving the project would be to actually pay attention to the suggestion that students make. We were provided with sample final reports from a year ago. One particular by Chen (his TA was Shah) has a few suggestions that still apply to our batch. These suggestions seem to be just ignored and are another formality. In particular suggestion about Ethernet, more room for extra credit creativity and more input from TAs during design review are something we was inclined to add on my own. However, it seems like they've already been ignored.