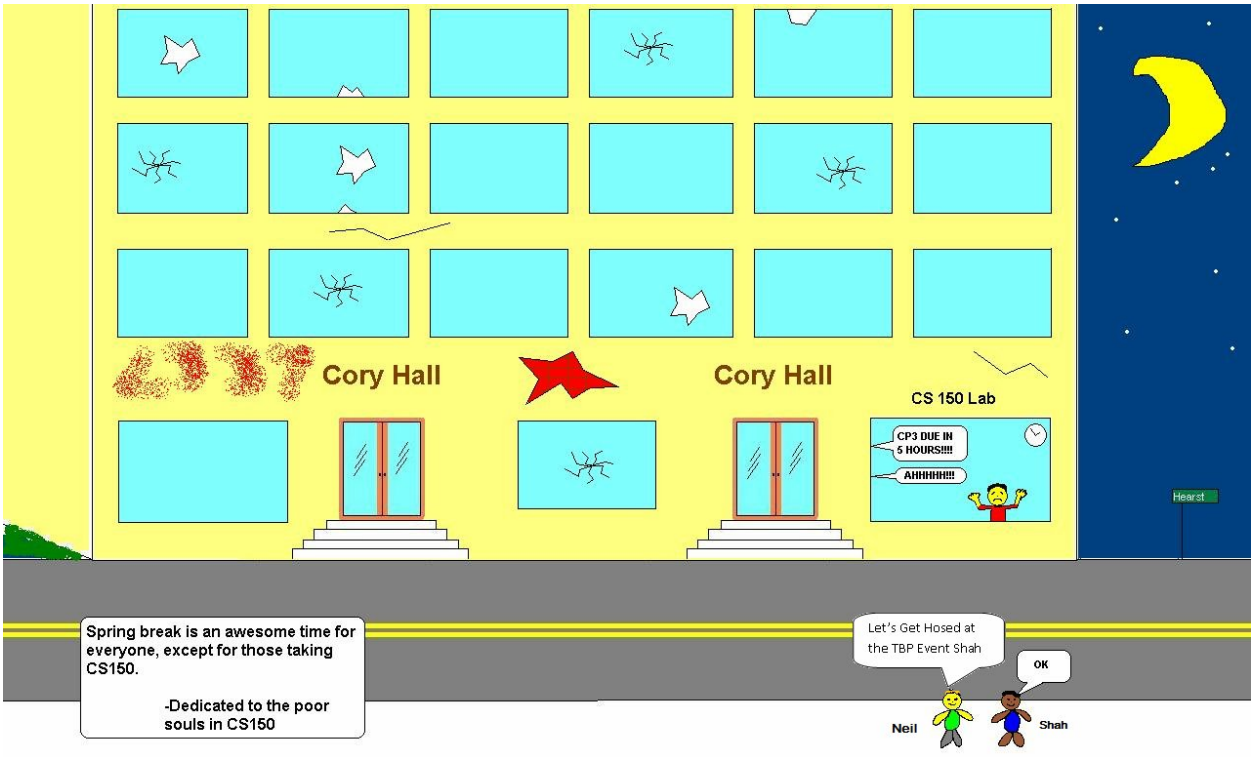
University of California, Berkeley

CS150 Final Report

Video Conferencing System



By Team Dumassmans
4/25/2007

Table of Contents

The Big Picture

Motivation

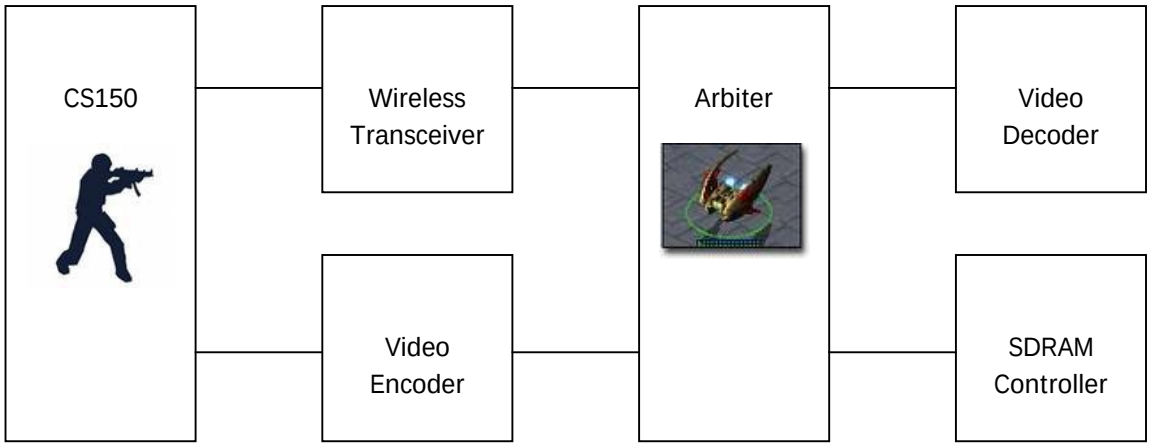
Communication is the foundation of civilization. History has taught us that those who learned to wield its endless power will always prevail. The Great Roman Empire was built and maintained on a solid foundation centered on roads. The great cross continental railroad that connected the east and the west deeply influenced the rapid growth of the US economy. The internet single handedly revolutionized the world and brought upon us the age of globalization where time and space has become increasingly obsolete. The power of communication rests in its ability to bring people together and we believe that our innovation represents a small step for mankind in this endless journey.

Abstract

The name of the game is video conferencing except it comes with a few kicks. Unlike most conventional video streaming systems out there, our project uses wireless technology to allow maximum mobility for our users. The hassle of hooking up Ethernet cables, dealing with broken Cisco switches and handling Microsoft's so called "networking interface" is all a thing of the past. All you have to do is switch to the correct channel, choose the appropriate mode[1] and hope Shah[2] or Neil[3] won't fail miserably. If a connection has been successfully established, the end users should be able to see each other at an astounding approximate rate of 1 frame / 1.5 seconds. Can you believe it?

Layout

- Checkpoint 0: Constructing an interface to the SDRAM to store and access video data
- Checkpoint 1: Interface with the Video Encoder to display video
- Checkpoint 2: Allow video data coming from the decoder to be written to the SDRAM and then displayed locally
- Checkpoint 3: Wireless Transceiver to allow wireless board to board communication
- Checkpoint 4: Compression and transmission of active video data
- CS150: Counter Strike 150 – Shah's Rebellion©



[1] When two video conferencing systems communicate with each other, one must be in Neil (Master) Mode and one must be in Shah (Slave) Mode. For more information, please refer to the Checkpoint 4 section.

[2] Sometimes, while handshaking, Shah the slave may not work properly due to his disgruntled state.

[3] Handshaking is a tricky process, even for master Neil since he is usually too busy stealing patents from Team Dumassmans

Checkpoint 0 - SDRAM Controller

Objective

Because of the large amounts of data involved in video conferencing, the SRAM available to us on the Xilinx FPGAs is insufficient in providing the necessary data storage. This justified the move to SDRAM, which provided virtually unlimited storage space in the scope of this project. Because of the complex timing involved in interfacing with SDRAM, a separate SDRAM controller was built to abstract away the details of interfacing with the SDRAM.

Design Methodology

32-bits using 16-bit RAM

The first challenge was to determine how to get 32 bits of data from 16-bit RAM chips. Since there were two RAM modules on the FGPA, every read and write would access both the modules in parallel; one RAM module would store the least significant 16 bits and the other would store the most significant 16 bits. Every output of the RAM controller would be fed into both the RAM modules.

SDRAM Timing

Because the clock was running at a mere 27MHz, most timing constraints on the SDRAM were trivially satisfied. The only major thing was the CAS latency of 2 cycles that must be obeyed during a read.

SDRAM Properties

A video conferencing system would need a lot of data to and from RAM. However, reads and writes are VERY sequential and follow in an extremely predictable pattern. For this reason, we decided to have the SDRAM work in bursts of 8. Auto-precharge was enabled to simplify the design and auto-refresh was unnecessary given how often the SDRAM was read / written to. To further simplify our design, our SDRAM controller would reject any read or write requests while a memory access was in progress, meaning that each memory access is guaranteed its full burst of 8.

State Machine Design

Our SDRAM controller consisted of three functional parts: initialization, read, and write, all implemented in one state machine. Initialization was run only once and consisted of a long wait time followed by a sequence of hard-coded steps to perform. The properties of the SDRAM (burst length, auto-precharge, etc.) were set during this stage. After initialization, the state machine moves to an idle state and waits for read or write requests. Read and writes were similar; both consisted of an active state to latch the row address, another state to latch the column address, followed by several wait stages as the data is fed into or delivered from the SDRAM. (see Appendix for State Machine Diagrams)

Design Tradeoffs

- Uninterruptable burst of 8 reads and writes means less flexible SDRAM usage.
- Not much consideration was given to obeying certain timing constraints of the SDRAM. Could potentially become a problem if a doubling (or quadrupling) of the SDRAM Clock frequency becomes necessary in the future.
- Lack of time dedicated to auto-refresh means a possible decay of stored data that is left unread for a long time.

Checkpoint 1 – Video Encoder

Objective

The video encoder is the second of the four major components of the video conferencing system. This module handled all outputs to the video encoder chip on the FPGA board and is responsible for correctly displaying video data on the screen.

Design Methodology

Row and Column Counters

Two separate counters were assigned with the keeping the current state. The column counter incremented every cycle and the row counter incremented once the column counter has reached 1716 (2 x 858 pixels). The column counter resets back to 0 at 1716 and the row counter resets at 525. Note that both these counters start at 1 and both counters are started after the I2C chip has finished initializing.

InRequest Signals

On the assumption that data fed into the encoder becomes valid the cycle after InRequest is asserted, InRequest must be asserted exactly two cycles before the requested data is fed into the video encoder chip. 1 cycle is needed for active video data to become valid and another cycle for the incoming data to be latched properly. Bear in mind that InRequest should only be pulsed once every four cycles in the active video range. A dedicated module that takes in the row and column counts handles all of this logic.

EAV and SAV signals

Another separate module generates the appropriate EAV and SAV signals given the column and row count.

Output Selection

There are three different types of data that could be fed into the video encoder chip: EAV/SAV, blanking data, or active video data. Based on the column count and row count, one of the three types of data is selected to be the output. Additionally, this module muxes between each 8-bit portion of active video data before sending it off to the video encoder chip, as the data is fed into the module 32-bits at a time while the chip accepts only 10-bits at a time.

State Machine Design

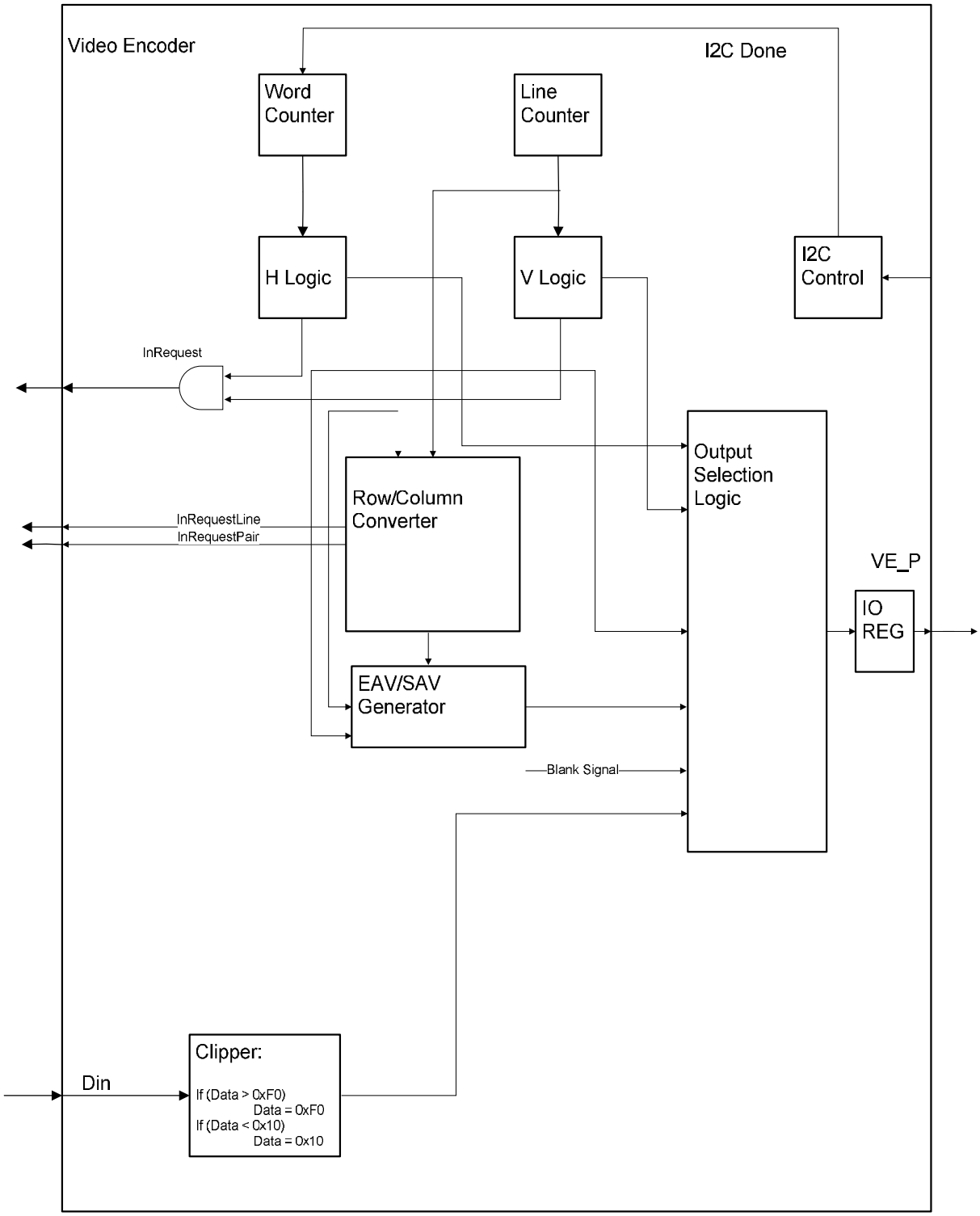
The row and column counters keep the state in the entire system. The design methodology of making dedicated combinational logic modules each doing a specialized task allows for easier testing and debugging. All functionality in this circuit is combinational logic based on the row and column counters.

Design Tradeoffs

- The presence of modules within modules made it difficult to connect an extra wire to a module buried within another module.
- Multi-level combinational logic allows less of an increase in clock frequency to be achieved.

Diagrams

Block Diagram



Checkpoint 2 – Local Video:

Objective

With the SDRAM and video encoder in place, a system can be set up to deliver video captured from the camera to the local television screen. Such a system must incorporate an SDRAM arbiter, to manage what is talking to the SDRAM, along with dedicated RAM Processors, for managing the SDRAM read and write processes of the video encoder and decoder.

Design Methodology

SDRAM Arbiter

The central backbone of the entire project, the SDRAM arbiter's job is to determine whose turn is it to use the SDRAM and to direct data to and from the SDRAM to the one that requested the read or write. Our arbiter implementation did not follow the recommended poll priority requests scheme. Instead, it polls each individual module, one at a time, to see whether they need the SDRAM to do something. If a request is needed, the arbiter directs the appropriate RAM address (and data) to the SDRAM Controller and moves to a wait state. In this state it will either request the data from or assert data valid for the process it is currently serving. When the request is complete, it will proceed to poll the next process. This design gives equal priority to all requests.

Video Decoder RAM Processor

This module handles the writing of data from the video camera into SDRAM. Data from the video decoder module is fed directly into an async FIFO. When the FIFO data count exceeds 8 (the minimum number needed for a full burst into RAM), the VDWriteRequest signal is asserted. RAM write address is kept by an address counter (counting between lines and pixel pairs of active video and increments by 1 every cycle it is enabled). When the Arbiter is serving a write from this module, it will assert VDDataRequest. This is fed directly in as the FIFO read signal and is delayed by 1 cycle before being fed as the enable signal of the address counter. This signal is asserted for 8 cycles by the Arbiter, effectively increasing the address counter by 8 and getting 8 pieces of data from the FIFO. This module does not keep the status of the current RAM request.

Video Encoder RAM Processor

This RAM Processor is in charge of delivering video data from the SDRAM to the video encoder. It works very similarly to the VDProcessor, containing a FIFO whose data is fed into the video encoder. The RAM read address is kept by an instance of an address counter identical to that used in the VDProcessor and VEReadRequest is asserted as soon as the FIFO is less than ¾ full. When the Arbiter is serving a read from this module, it will assert VEDataValid (which is directly tied into FIFO write) for 8 cycles. This will increase the RAM read address by 8 and write 8 valid pieces of active video data into the FIFO. InRequest is used as the FIFO read signal. This module also does not keep the RAM request status.

Text Overlay

The display of informational text is essential to our project as well as being a powerful debugging tool. Several designs were thought up for a text implementation but only one was able to provide the flexibility and reliability needed in the final extra credit design. The first step was to partition the screen into 8 pixel pair x 16 line blocks. Each block was mapped to an address in a 2048 x 8-bit wide SRAM dedicated to ASCII code storage and another 2048 x 8-bit wide SRAM dedicated to text color storage. Data coming

into the video encoder is first fed into a TextOverlay system. This module sequentially reads both SRAMs, does a translation into a character mask, and replaces active video data with the text color data according to the mask before passing it to the chooser.

State Machine Design

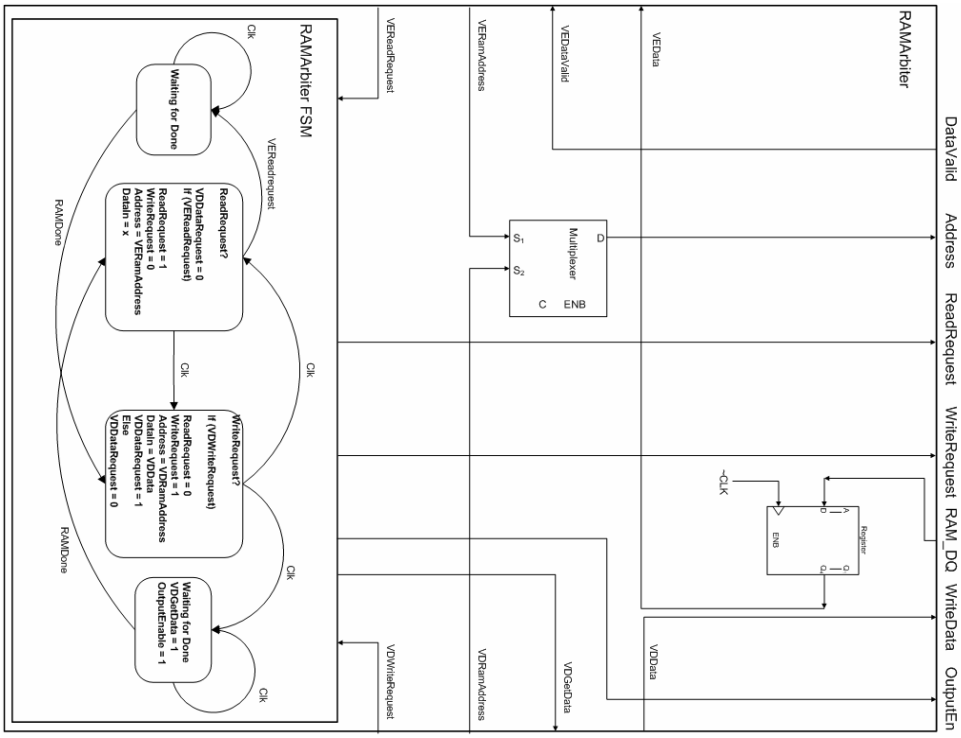
The Arbiter is a single state machine that transitions between different polling states. If it enters a state and the processor corresponding to that particular state has the request line enabled, it proceeds to serve that request. This FSM runs in an infinite loop and alternates between the VEProcessor state and the VDProcessor state. The only state elements in VDProcessor and VEProcessors are the address counters and the FIFOs. The RAM requests are generated off of combinational logic stemming from FIFO data counts. The address counter maintains the next RAM address to write/read active video data to/from. (See Appendix for State Machine Diagrams)

Design Tradeoffs

- Round-robin style SDRAM arbiter could potentially limit SDRAM bandwidth.
- Less sequential logic in both the processors simplify design but compromise flexibility in the SDRAM arbiter implementation
- A RAM Address counter incrementing by 1 could make off-by-1 errors much more difficult to debug.
- Asychronous FIFO increases LUT utilization by 20% and synthesis time by 5 minutes
- Usage of SRAM for text storage prevents the writing of multiple characters within the same cycle. As a result, writing text becomes a multiple cycle process and is much more difficult.
- Access of SRAMs and ASCII ROM is slow, so TextOverlay system becomes a multiple cycle process.

Diagrams

Arbiter



Checkpoint 3 – Wireless Transceiver

Objective

The Video Conferencing system requires a medium of communication in order for video data to be transmitted and received by the two end systems. The goal of this checkpoint is simply to allow the transmission and receipt of data using wireless technology by interfacing with a CC2420 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver.

Design Methodology

General Design Patterns

The general design pattern for this checkpoint is quite similar to all the others. We used a Moore machine to manage the outputs. The code was written using 3 general blocks. One block was used to take care of state transitions at every positive edge of the clock. Another block was used to generate the logic which would determine the next state. Finally the last block was used to generate the appropriate outputs.

The Serial Parallel Interface (SPI Module)

Originally, the CC2420 provides a serial interface with which all register accesses must be performed with. To simplify this task, a *SPI module* was provided so that we can interface with the CC2420 using a parallel interface.

Handshaking

In order to make sure that commands are issued properly and data is fetched correctly, we need to obey certain standards. Before issuing a new command, we need to make sure that the *SPI Module* had effectively understood and executed our previous request. After making sure that the *SPI Module* is in fact ready for a new command, we need to drive the command wire with the appropriate signals and then assert that a new command needs to be processed. Handling outputs from the *SPI Module* revolves around the same concept. Except this time, the *SPI Module* must assert that new data is ready after it drives valid data on its output.

State Machine Design

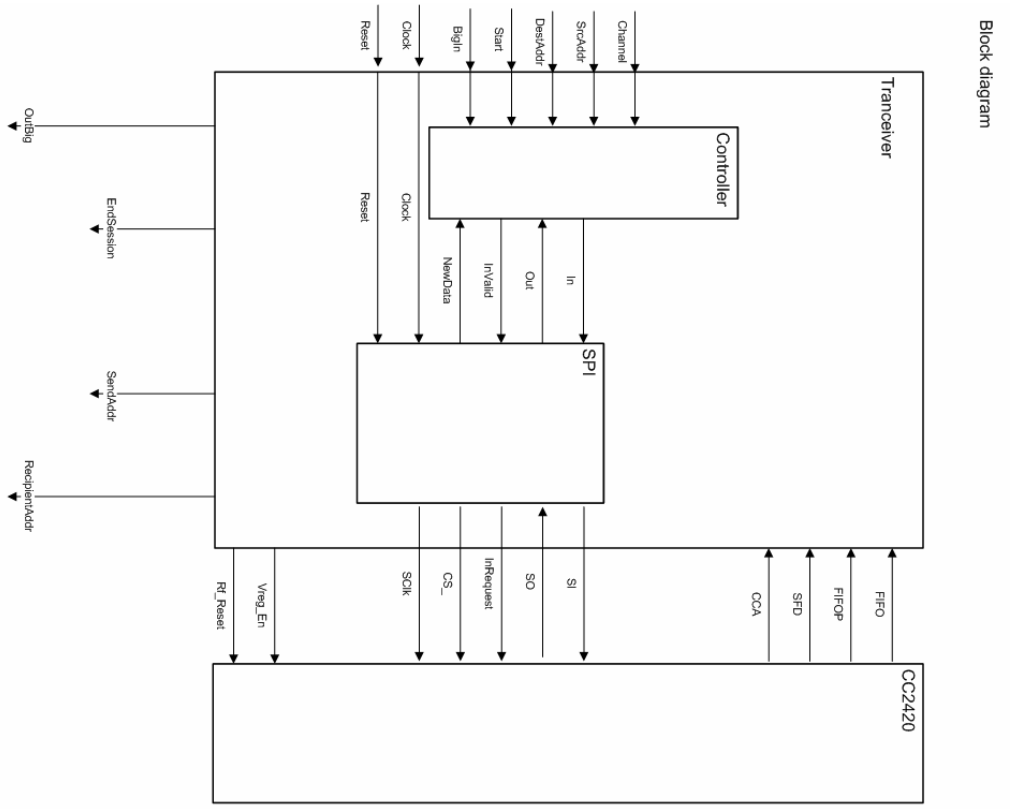
Four separate state machines were designed to work together to make this checkpoint work. The first one is the main / high level state machine implemented via time state and simply toggles between *Initialize*, *Idle, Transmit* and *Receive* state machines. Each individual task (*Initialize*, *Transmit* and *Receive*) has their own state machine to handle the specific details of each task. The main state machine decides which task to perform based on the requests received from the video conferencing system and signals from the CC2420 Transceiver.

Design Tradeoffs

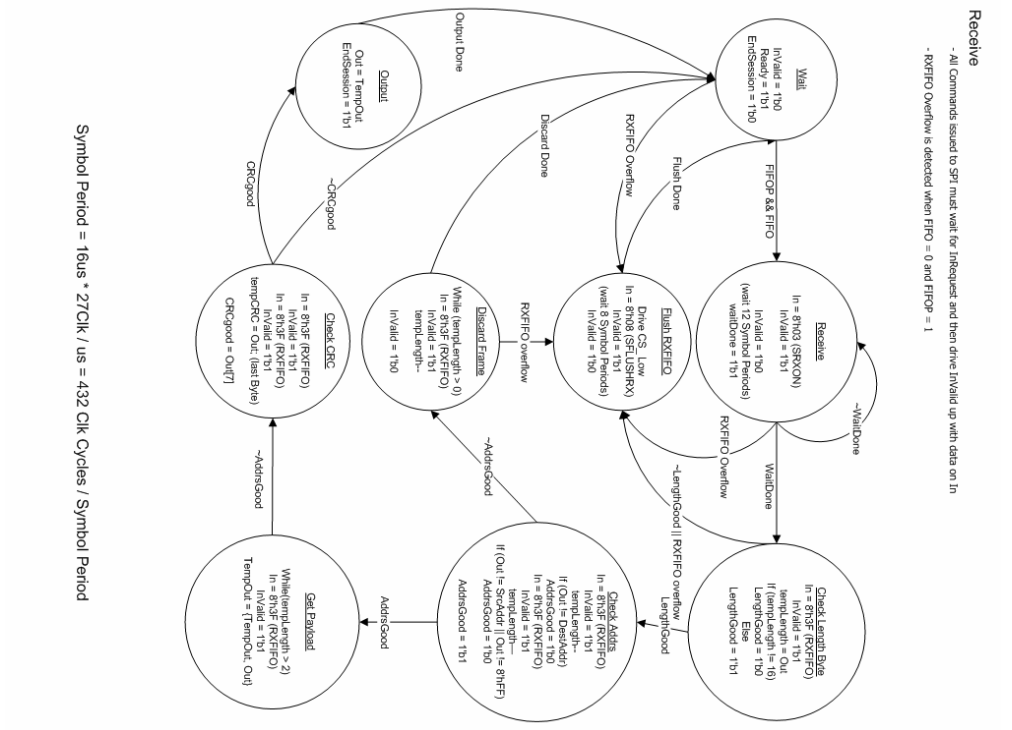
- Additional wires were needed to allow Receive to be performed while Transmit state machine is waiting for CCA.
- The 96bit payload limits our bandwidth significantly but a larger packet size would most likely induce a higher rate of garbled packets.
- The mandatory 12 symbol periods wait (~6000 clock cycles) after each transmission is a major overhead and could be partially eliminated by detecting if the next state is transmit or receive since the 12 symbol periods wait is only necessary when we're switching from transmit to receive.

Diagrams

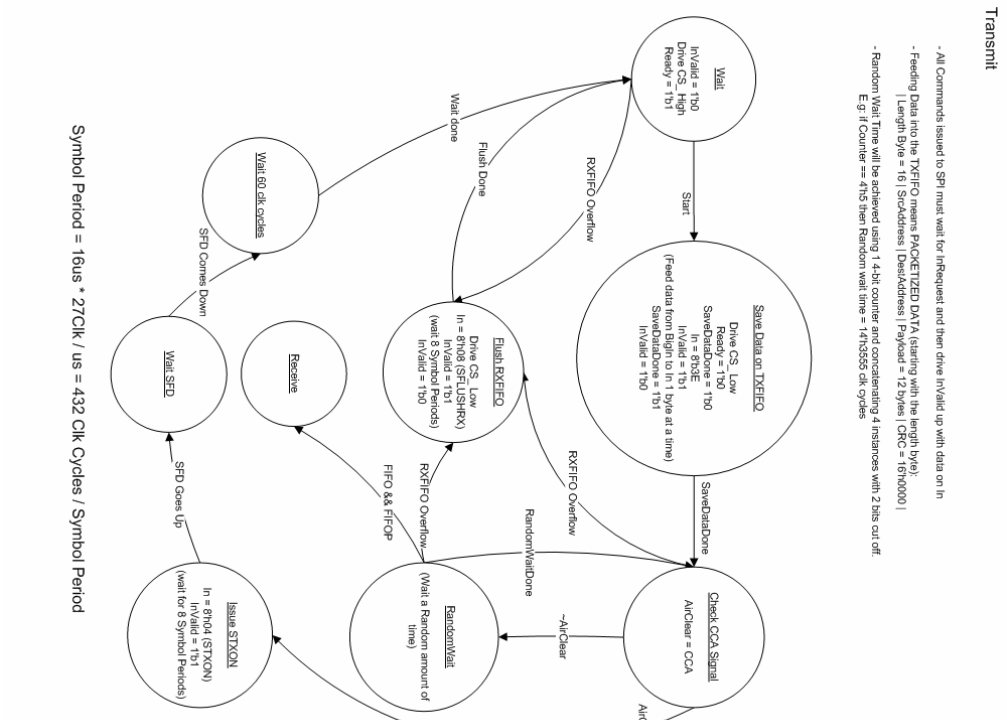
Overall Transceiver



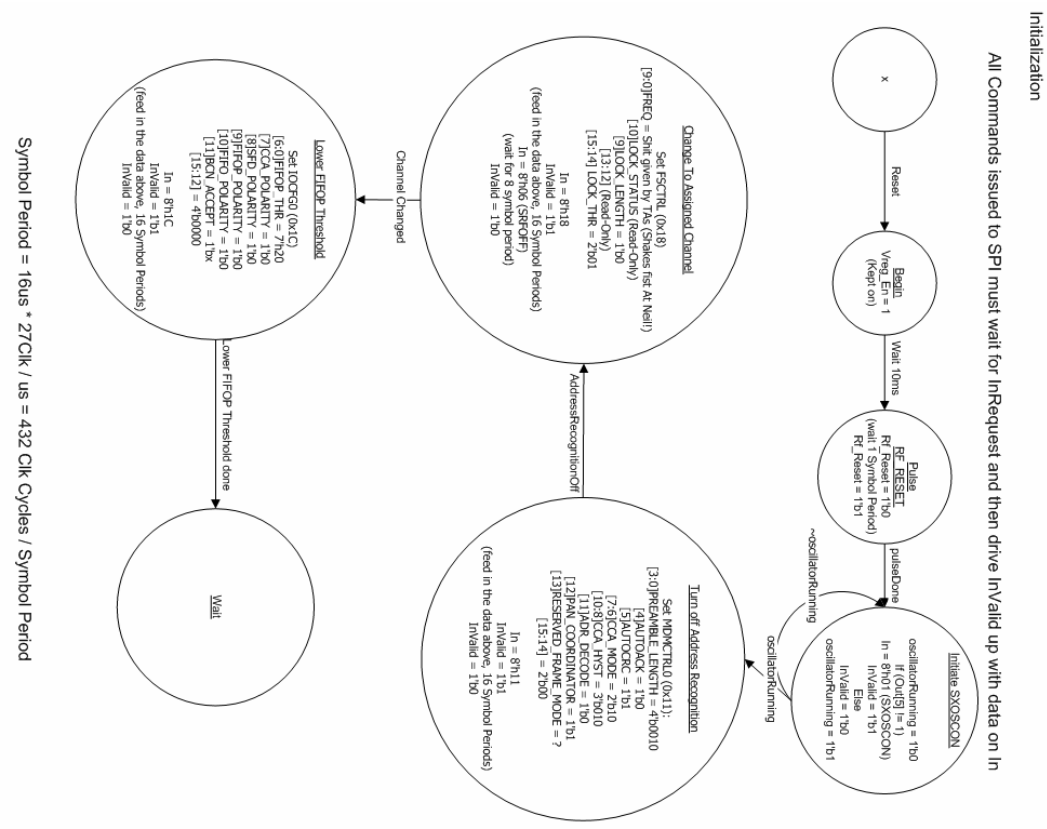
Receiver



Transmitter



Initializer



Checkpoint 4 – Video Conferencing

Objective

Create a video conferencing system by combining the components created in all the previous checkpoints. A Master-Slave paradigm must be put in place for handshaking. Compression and decompression must be put in place when video data is being sent or received.

Design Methodology

Communications & Handshaking

Handshaking is necessary to establish the connection between a master and a slave. Only after a successful handshake can we start video streaming. A simple state machine was designed for both sides. Even though master and slave shared certain behaviors, they differed drastically in others. The master must try to reestablish connection after a timeout if packet loss or garbling happened, while the slave only needed to stay ready to receive the next packet. The two communication timeouts were implemented using simple counters.

Compression

Local video must undergo a very rigorous and lossy compression scheme before being sent over wireless. This was implemented by putting a module that taps the data wire between VDProcessor and SDRAM. Since the address counter in VDProcessor kept track of where the current piece of data was being written to, it was used to determine when to sample. The sampled pixel pairs, compressed into 5-bit luminance values, were further processed before being sent to a FIFO. Since each word in SDRAM was 32 bits, SDRAM usage was optimized by storing 4 sampled pixel pairs per 32-bit word, which take up the lower 20 bits of the word. To simplify the wireless packetizing process, the top 5 bits contained the header and the next 7 bits contained the row number of the sampled value, for a combined total of 32 bits. All of this is fed into a FIFO. This module comes with its own address counter and data is read sequentially from the FIFO and written into a different bank. Note that this module will stop sampling after finishing a frame. It will begin sampling again at the start of the next frame if a wireless ready signal is asserted by the packetizer.

Decompression

Decompression is essentially just a simple reverse of the compression algorithm. A module put in parallel with VEProcessor was put to this task. Another module, PIPControl, controlled whether it was VEProcessor or the decompressor that is supposed to give out active video data based on InRequestLine and InRequestPair signals from the encoder. This module is very similar to VEProcessor with an added decompression stage and a different address counter. It puts the next piece of active video data on the line when the PIPControl asserts InRequest. Two exact instances of this are created, one for local compressed video and one for wirelessly received video.

Packetizer

Because of the compression technique, this process was extremely simple. It requests read requests to the arbiter and puts the data in a FIFO similar to other modules. Because of how the data is compressed,

the header and line number of the packet is constructed from the top 12 bits of a piece of a word. 4 32-bit words are needed to fill up the payload with 16 samples. A counter acting as a timer takes 4 pieces of data out of the FIFO every 2.4 milliseconds and holds the value of the packet until the next 4 pieces of data are taken out. This constructed packet is directly connected to the BigIn port of the transceiver. Based on the address counter, this module also asserts a wireless ready signal when the entire frame had its turn at BigIn.

Wirelessly Received Video

Wirelessly received packets are directly connected to a register and to a FIFO. A write request is generated whenever a full packet is received and the address to write to in RAM is generated from the header and line bits of the packet.

Text Writer

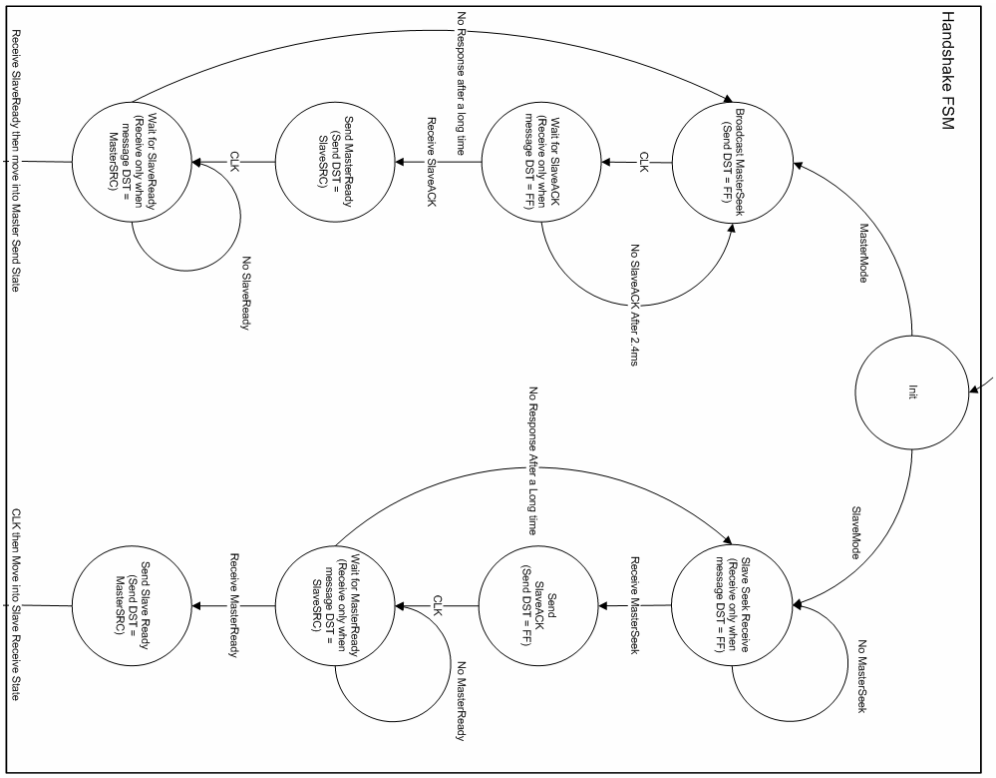
Because the previous text overlay implementation could only write one character per cycle, a layer of abstraction was needed to simplify the text writing process. This module will initialize the text overlay to some initial value, assert a ready signal, and enter an idle state. When ready, the inputs are a string of characters, the length of the string, the color of the string, where to write the string, and a write request signal. When write request is asserted, it will turn off the ready signal and begin to write, one character at a time, the string and color into the SRAM cells found in the text overlay. When done, it asserts ready again. Another module was made to use the text writer to update the video conferencing statuses.

Design Tradeoffs

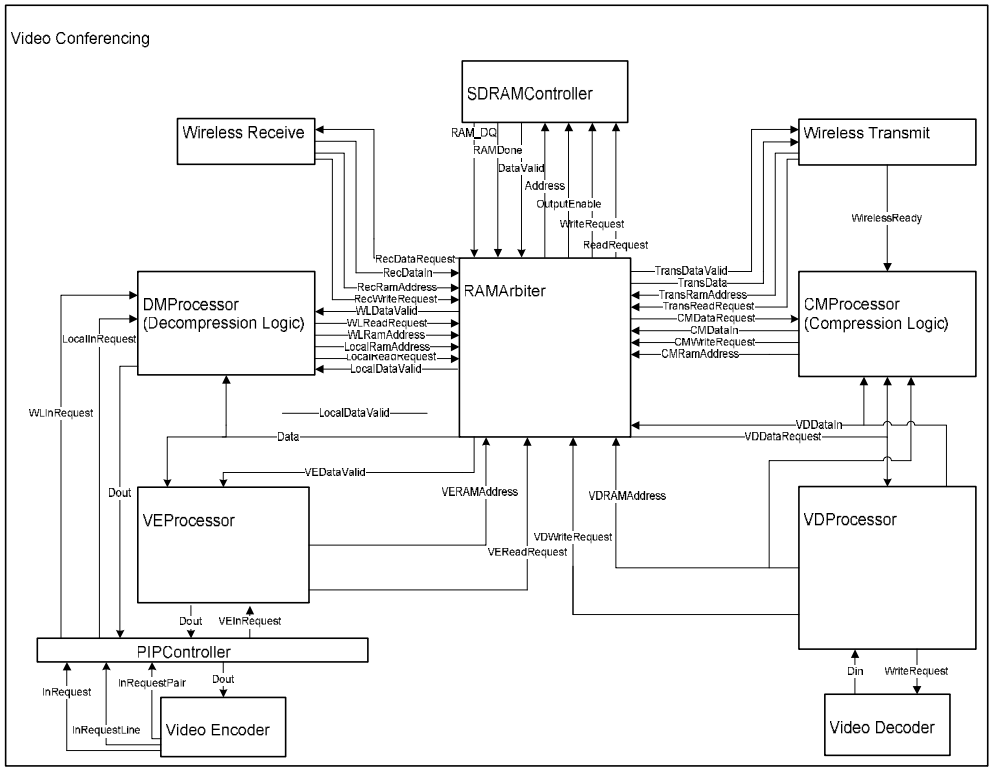
- Complex compression may increase SDRAM bandwidth and packetizing process, but also complicate design.
- Replacing the data to send every 2.4 ms increases frames per second, but will create extra loss when conferencing with another system that sends much slower.
- Several extra ports added to the SDRAM arbiter could create more overhead in its round-robin implementation.

Diagrams

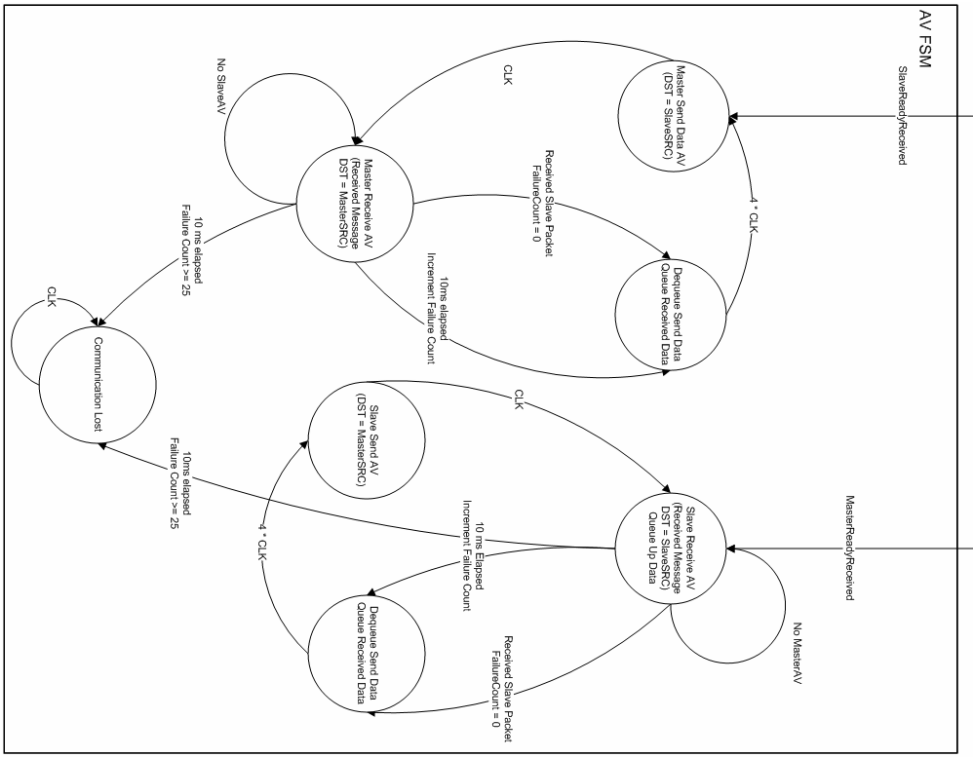
Handshake



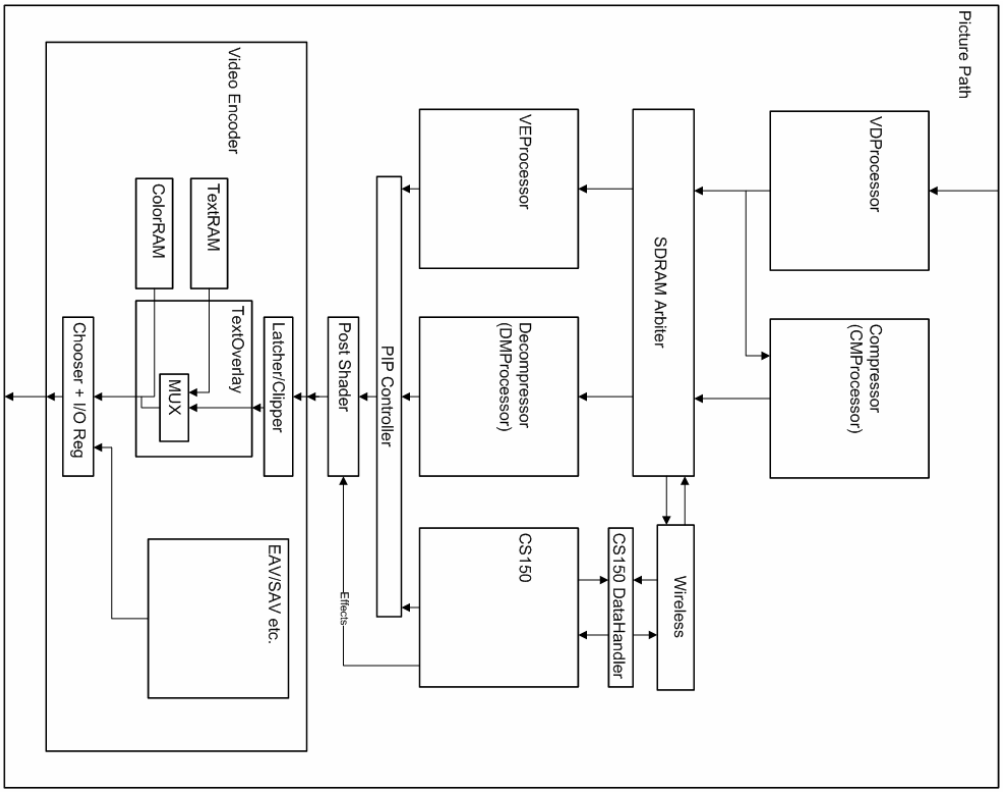
Overall Block Diagram



Active Video FSM



Picture Path



Design Metrics

FPGA Usage

LUT Usage	Number (with CS150 implemented)	% Utilization
4 Input LUTs	9389	24 %
Used as Logic	7904	
Used as route through	468	
Used as dual port RAM	512	
Used as shift registers	505	

Debugging and Design Time Estimates (In combined man hours)

Checkpoint 0
Design Time: 3 hours
Coding / Debugging time: 15 hours
Total: 18 hours

Checkpoint 1
Design Time: 5 hours
Coding / Debugging time: 8 hours
Total: 13 hours

Checkpoint 2
Design Time: 5 hours
Coding / Debugging time: 20 hours
Total: 25 hours

Checkpoint 3
Design Time: 12 hours
Coding / Debugging time: 40 hours
Total: 52 hours

Checkpoint 4 (w/o CS150)
Design Time: 5 hours
Coding / Debugging time: 60 hours
Total: 65 hours

CS150 Implementation
Design Time: 3 hours
Coding / Debugging time: 35 hours
Total: 38 hours

Total Time: 211 hours

Conclusion:

This semester's project was not only fun but also very insightful in the world of high level digital design. The video conferencing system not only tested our technical abilities, but also our teamwork skills and creativity.

Our video conferencing system not only featured standard two-way video, but also the game *Counter-Strike 150: Shah's Rebellion*. It is a standard 1v1 game where the objective of each player was to kill the other player. An N64 controller was used to provide the controls needed for each player and the communication between the two players was done via wireless. Each player moved around using the D-pad on the controller, aiming of the crosshair was done using the analog stick, and the "Z" button was used to fire a bullet. In addition to the standard "aim, then shoot" formula, several engine improvements were implemented to increase the difficulty.

The first of which was the limited bullet capacity; each player was given only a 30-bullet clip before having to hit "L" for reload. The second feature was dynamic recoil. Note that each bullet the player fired did not land exactly where the crosshair was. Each bullet fired increased a "recoil factor" that degraded with time. As this factor increased, bullets would begin to land farther and farther away from the crosshair in an unpredictable pattern. This was done to simulate the realism of firing a gun and to discourage bullet "spraying". But in addition to these engine improvements, several things were done on the user interface level to improve the overall experience.

The incredibly flexible text interface proved invaluable in our ability to improve visual interface. This text overlay system allowed us to display text in a dynamic fashion. Visual enhancements such as blinking characters (while reloading), visual counters (next round count down) and scrolling text (taunts) were all made possible. A wide variety of colors was available and changed dynamically with the text. All those great capabilities translated into a user-friendly environment for the video conferencing user as well as the counter strike player. The status of the game is very clearly displayed. Whether it is "Counter-Terrorist Wins" or "SHAH Mode" or "NEIL *headshot* SHAH" that needed to displayed, this text overlay system could do it with both graphical precision and timing accuracy. Last but not least, the taunting system was made possible so that users can spare themselves from deadly boredom by sending their "TRUE FEELINGS" to their opponents.

One of the design challenges we faced was during the wireless receiving process. Because of our compression scheme, each packet corresponded to only half a burst rather than two bursts. This created a problem; since memory writes were not necessarily sequential due to lost packets, we cannot assume that the next packet received belongs in the same burst as the previous packet. This meant that we cannot wait for two packets to be received before writing a burst. Our solution took advantage of the DQM signal fed into the SDRAM. Each write request made by the wireless receive process would assert DQM for the second half of the burst, effectively writing only one packet at a time. This solution comes with a cost in performance, as the arbiter will still wait eight cycles for the entire burst to be finished while in reality there are only four cycles that are actually doing something.

While interfacing with the CC2420 wireless transceiver module, it was extremely difficult to debug the initialization process. The only indications available were "it works" or "it doesn't work". There is no

mid-way indication which could be used for debugging parts of the design. Certain parts of the design we thought were wrong turned out to be right. Eventually we initialized properly, but it was extremely tedious due to the lack of indications.

Another challenge we faced was implementing an edge detector. What was wrong with it? It never worked. Why couldn't we get it to work? It's because Shah was too busy playing guitar hero like a guy who owns a cheap banjo instead of helping us or other groups. Here is a quote from Shah: "Just delete your project, make a new one. I don't know why, but it should work!" (See Figure 1)

There are several things we should have done differently. One of which was the SDRAM arbiter implementation. Instead of asking each module for requests one at a time, we should have used a rotating priority. In other words, each state corresponds to the module with top priority in that cycle. So, rather than doing nothing if that module does not have a request, the arbiter will instead ask everyone else for a request.

Another thing we should have done differently was the compression. Its implementation was justified by the potential savings in SDRAM requests. But since SDRAM bandwidth was not the limiting factor in the performance of our project, all it did was complicate our design. We should have just written 32-bit chunks into SDRAM, each containing 1 pixel sample.

Figure 1:

Notice how Shah is completely ignoring Sylvain who is asking him an important question very nicely and also notice that Sylvain's partner (Michael Le) is crying at the printer due to the lack of help. (He even dressed up just to get Shah's attention)

Yes Shah, we HAVE PROOF! Take that Team Awesimo! Team NEIL PWNS JOO!

Suggestions

- Make specs more clear and centralize the important information, having 151 pages in a datasheet where only 3 pages are useful does not help
- Somehow I have the feeling that we should have used the ETHERNET so we can actually get a decent video conference going, this is why our extra credit did not have anything to do with video conferencing because it sucked to begin with
- Leave more room for extra credit, I think that we should have a shorter required part and a much longer self innovative part. Perhaps make it required instead of extra credit even?
- Last Chaos Pwns. Pease
- Make our project a game. A game is much more fun and interactive than just a video conferencing project.
- Looser requirements about what it is supposed to do don't make us follow some sort of design (i.e. 96 bits per packet) that was not exactly justified in the beginning. Cross compatibility is also an unnecessary addition.
- Do not give us async FIFOs that are 1000 times larger than what it needs to be and add 5 minutes to synthesis time.
- Design reviews should be graded on an effort basis. The purpose of it is to make sure our design works. If the only way to get 100% on a design review was to have a perfect design, there would be no point in having it in the first place.
- Lab instructions are clearly outdated. Please update them!
- Sometimes homework and test questions (i.e. 6 variable k-map) are extremely tedious and require no intellectual ability.
- Make Neil more awesome! And make Shah more awesome too!
- Explain some physical layer and link layer networking concepts as we do checkpoint 3. Concepts such as CSMA/CA and/or Error detection / correction (fountain code) can help us understand / appreciate the specifications better.
- Find some way to prevent people to lock 50 computers forever which stops other people from using it.
- Don't admit 100 people into the class, reject the second years!
- TAs need to be more specific during design reviews and also tell us more about corner cases such that we can address them sooner rather than later (which usually translates into 30 hours of debugging)
- *Bring down the lab noise! We can't concentrate! That dumass and idiot are really ANNOYING!*

NEIL!!!

NEIL!!!

SHAH!!