Digital Poker
6.111 Final Project- Fall 2005

Amy Daitch
Elizabeth Murnane

Abstract:
We implemented a game of five card draw poker for one person to play against the computer. All the parameters of the game (cards, chips in pot, scores, etc.) are displayed on the screen for the user to see. To simulate the experience of playing poker in person, we have the user enter a "poker face" every time he/she is dealt cards by placing one of four pre-defined face cards (smile, frown, angry, straight face) in front of a video camera at the appropriate times. The computer decides on its moves based on an algorithm accounting for various states in the game such as its current hand of cards, the user's poker face, etc. The computer also must decide on a poker face whenever it is dealt cards, which is also displayed on the monitor.

Table of Contents

List of Figures and Tables

Introduction

We implemented a game of five card draw poker for a single player (against the computer). To summarize the rules of the game, at the beginning of each round, each player is dealt five cards. A hand of five cards is ranked based on the standard poker point system (exact point system unimportant for now). After five cards are dealt to each player, the non-dealer makes his/her bet after which the dealer must either match the bet or fold (drop out of the round). Players can continue to increase bets, and a round of betting is over either when each player has bet the same amount and neither wishes to raise, or if a player decides to fold. Then, each player can trade any of his/her current cards. After the new cards are dealt, one more round of betting takes place, after which the players must reveal their cards. The player with the highest ranked hand wins the round and keeps all the chips in the pot. The game continues until one of the players is out of chips.

Since much of the fun of poker is in the psychology of the game (e.g. psyching out one's opponent, maintaining a "poker face", etc.) we decided to incorporate these aspects into the project, giving the computer some "human" qualities. The computer will be able to "see" the expression on the player's face (smile vs. straight face vs. frown) by analyzing the input from a video camera which is monitoring the player's facial expression (in actuality, the player will simply chose one of a set of pre-made face cards with easily detectible facial features) and may use this information to strategize its next move. Furthermore, the computer will generate a facial expression after each set of cards is dealt which is somehow related to the computer's current hand and other variables of the game (e.g. opponent's facial expression). The player may then use this information to strategize his/her next move. To make the game even more interesting, the player may choose which of the computer's "personas" to play against. Each persona will have a different strategy which determines which cards it will trade, how much it will bet for each round, and what facial expression it displays, all as a function of its current hand of cards and its opponents facial expression. For example, one persona might simply always bet high, never trade any cards, disregard the facial expression of its opponent, display a smiley face when it has a bad hand of cards, and a frown when it has a good hand of cards.

To simplify the design of this game, we divided it into several subsystems which could be designed and tested individually before being integrated into the final product. The main blocks of our project are the Video Controller, Game Controller, and Facial Feature Detector (Figure 1), each divided further into smaller subsystems. Amy worked on the Video Controller and Elizabeth worked on the Game Controller. These blocks are described in more detail below.

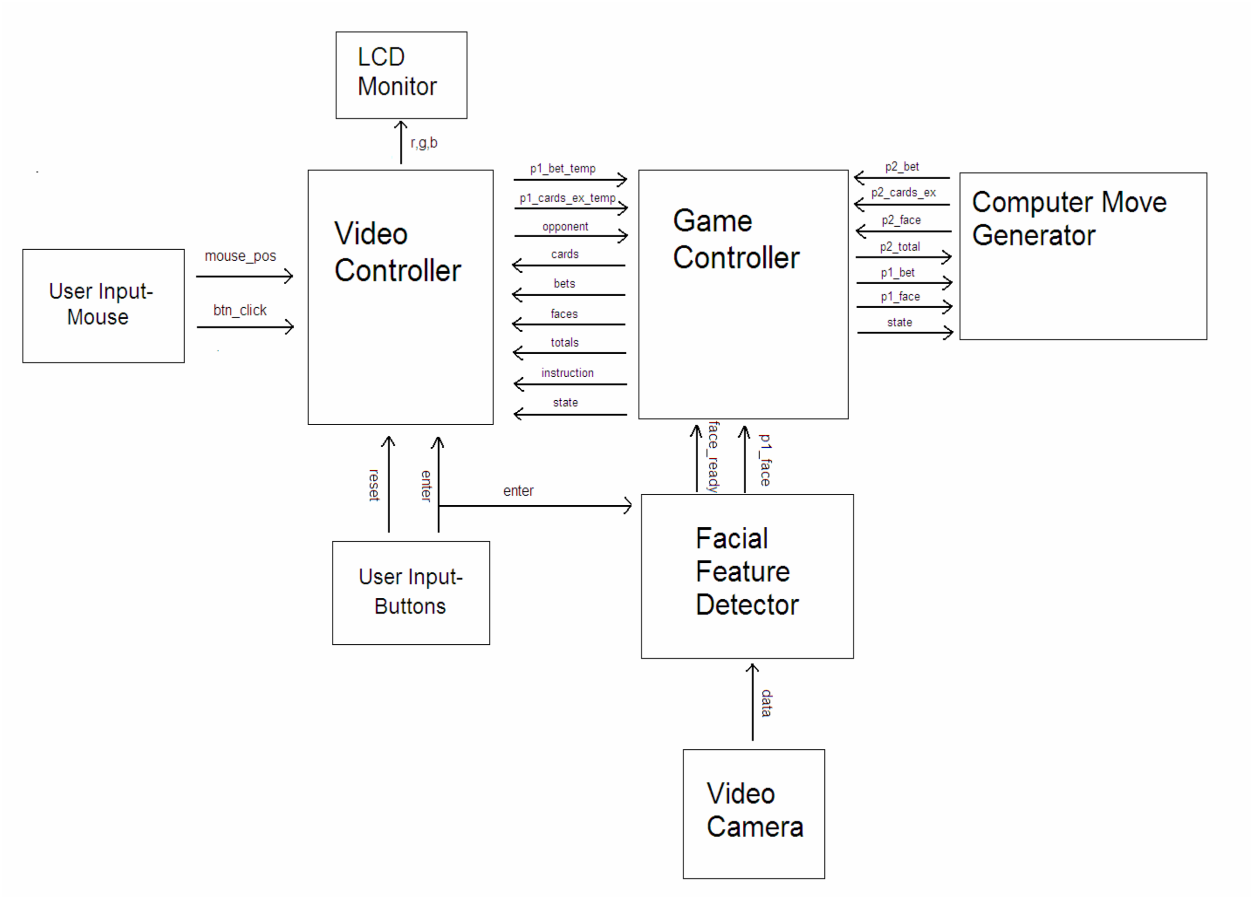

Figure 1. Top-Level Block Diagram

Detailed Design and Implementation

Video Controller

The Video Controller is the top level module of this project. It contains all of the labkit components and inputs from external sources, such as the video camera and mouse. Within the Video Controller, the XVGA module creates the appropriate timing signals (hcount, vcount, etc.) for all the modules creating images, so the appropriate pixels are generated at the correct time to produce the desired image on the monitor. The Video Controller then sends the pixel data to the digital to analog converter which sends the appropriate signals to the monitor, generating the desired image.

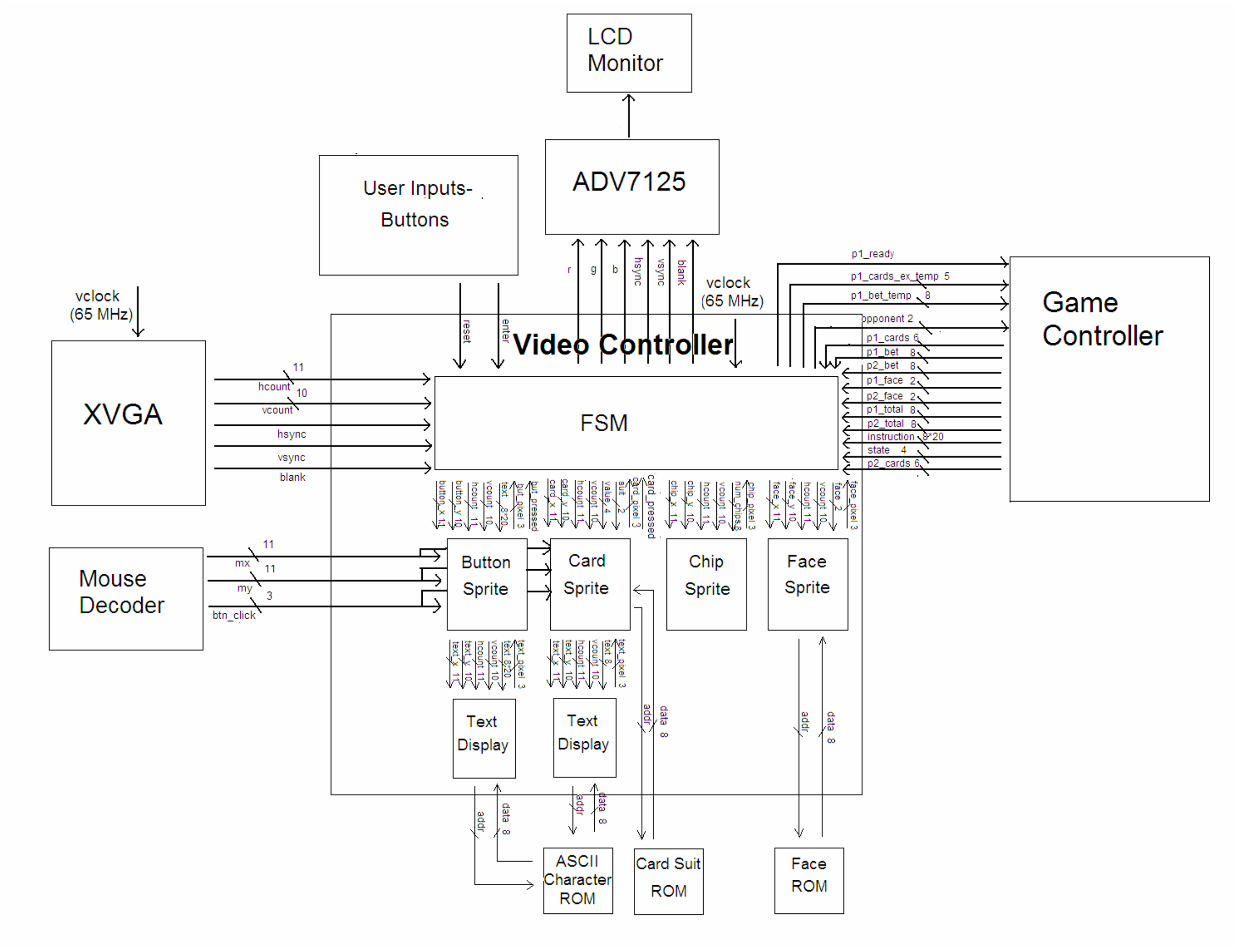

Figure 2. Detailed Diagram of Video Controller

Poker Table

The Poker Table module controls which images (which components of the poker game) are sent to the monitor at any particular time, creating a visual interface for the user. The Poker Table module takes as inputs various states of the game (not to be confused with the states of the FSM); the current hand of each player, each player's current score (chip count), current total bet for the round, current number of chips in the pot, players' poker faces, and the current stage within a round (Figure 3). It will output on the computer monitor an image of a poker table with some combination of: the players' cards, scores, number of chips bet, facial expressions, and instructions for the player or a declaration of the winner at the end of the round, depending on the current state (Table 1). An example layout of the poker table image is below (Figure 4). The monitor display will update at the start of every stage within a round according to the states of the inputs, so the Poker Table module can essentially be modeled as an FSM. The FSM waits for a particular user input to change states. For example, at the betting state, this module waits for the user to enter a bet (by clicking on various "betting buttons" on the screen with the mouse, then pressing the labkit's enter button), before transitioning to the next state. The next state is determined by the Game Controller.

Each image on the screen (cards, chips, etc.) will be created using replicable sprite modules, which create an image at a desired location on the screen by generating the correct pixels at the appropriate times. These sprite modules are described below:

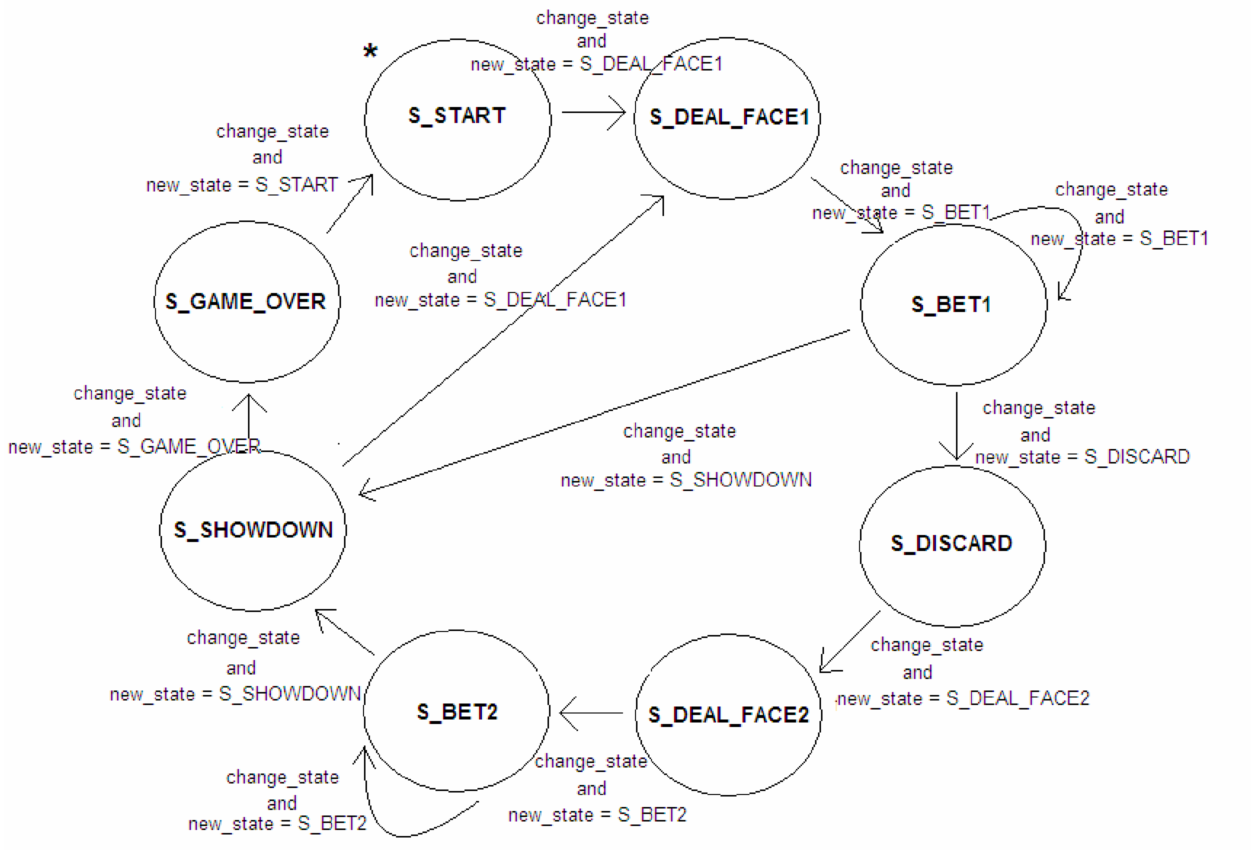
*
S_START
S_DEAL_FACE1
S_BET1
S_DISCARD
S_DEAL_FACE2
S_BET2
S_SHOWDOWN
S_GAME_OVER
change_state and new_state = S_DEAL_FACE1
change_state and new_state = S_BET1
change_state and new_state = S_BET1
change_state and new_state = S_DISCARD
change_state and new_state = S_DEAL_FACE2
change_state and new_state = S_BET2
change_state and new_state = S_BET2
change_state and new_state = S_SHOWDOWN
change_state and new_state = S_SHOWDOWN
change_state and new_state = S_DEAL_FACE1
change_state and new_state = S_GAME_OVER
change_state and new_state = S_START

Figure 3. Finite State Machine for Poker Table

Button

The button module creates an image of button to the screen (a colored rectangle) with the appropriate text on it. It also takes input from a mouse to determine whether it has been clicked. In our game, the buttons were used during the betting stage, with possible betting amounts on each button. The user clicks on any series of these buttons to place his/her bet. The buttons also change color when being clicked, so the user knows his/her actions are being registered by the game.

Card

The card module takes as input a card's value (Ace, 2, 3, etc.) and suit and generates an image of the card with these parameters. The character representing the value is read from a Font ROM and the suit images are taken from four Suit ROMs. This module also takes input from the mouse, so that during the card exchange state the user can click on the cards he/she wants to exchange. The card changes color when the user clicks on it to distinguish the cards that have been chosen.

Character String Display

The Character String Display module takes as input a string of characters and outputs the characters to the monitor at a desired location. It is used to display text to the screen, such as instructions for the user, each player's score, etc. This module gets the pixels for a particular character from the Font ROM.

Chip

The chip module takes as input a number of chips and generates an image of a pile with the appropriate number of chips. This module is used to display how much each user has bet thus far in the current round (i.e. the pot).

Face Display

The Face Display module takes as input one of four possible face parameters (angry, smile, frown, and straight face) and outputs an image of a face corresponding to the given facial expression. These faces images are stored in four Face ROMs.

Mouse Cursor

The mouse cursor module displays a small square to the screen representing the mouse so the user knows its current location. The input to this module comes from the ps2_mouse module which calculates the absolute position of the mouse (on the monitor) based on the signals coming from the mouse.

Number Display

This module outputs numbers to the screen as characters. To do this, it converts a binary value to that number's ascii code (done in the bin_to_ascii module), and the ascii code is sent to the character string display module to generate the image of that number.

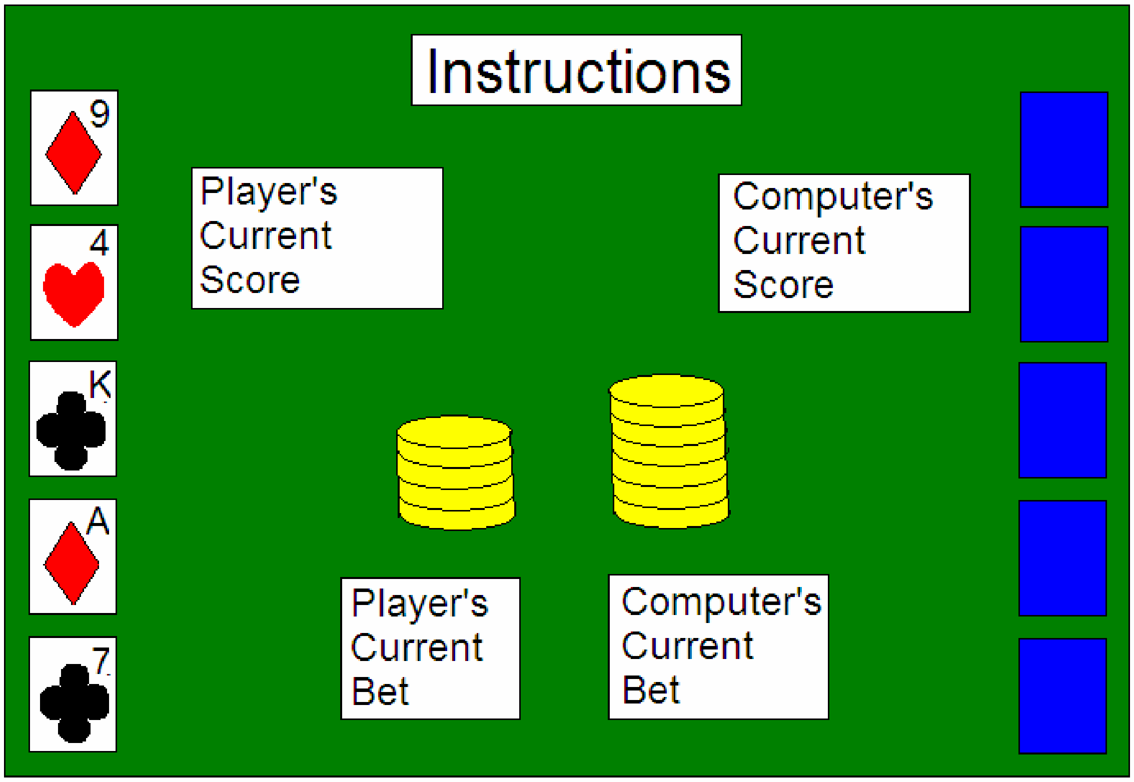

Figure 4. Example layout of the poker table

Table 1. Table of images displayed during each state of poker game

S_START	S_DEAL_FACE1	S_BET1	S_DISCARD	S_DEAL_FACE2	S_BET2	S_SHOWDOWN	S_GAME_OVER
instruction	instruction	instruction	instruction	instruction	instruction	instruction	instruction
mouse curser	cards	mouse curser	mouse curser	cards	mouse curser	cards	scores
opponent buttons	scores	cards	cards	scores	cards	(p2_cards face up)	
		scores	scores	chips in pot	scores	scores	
		bet buttons	chips in pot	faces	bet buttons	chips in pot	
		chips in pot	faces		chips in pot		
		faces			faces		

Game Controller

Game play is directed by a large game controller. It tells the video controller what to show on the screen and when to show it. From the video controller module, it takes user controlled signals as inputs and outputs to it the computer's face, bet, and discard choices. From the sorting module, it inputs sorted cards and outputs unsorted cards. Figure 5 is a block diagram for the game controller module.

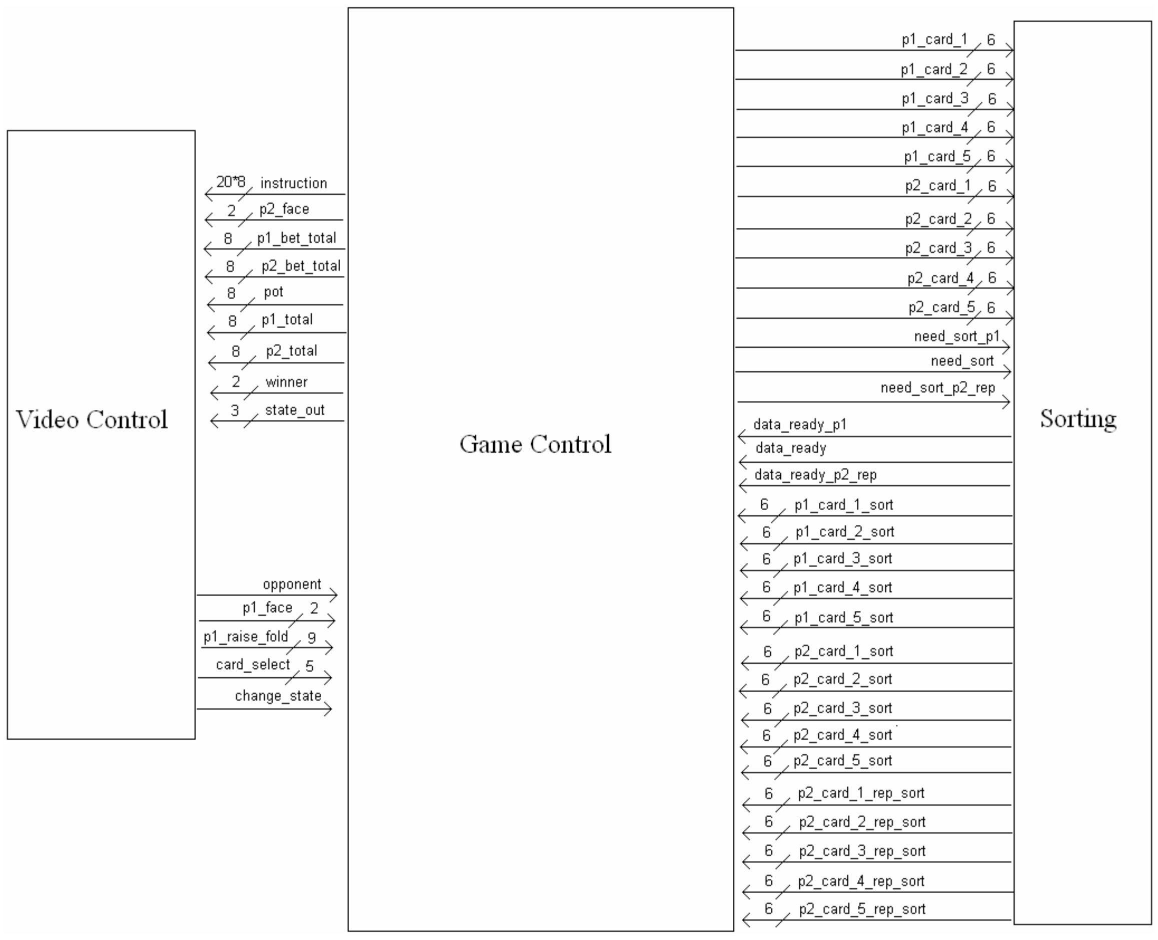

Figure 5: Game Control Block Diagram

The Game Controller uses a finite state machine (FSM) to cycle through the various phases of game play. Figure 6 is the state transition diagram, which demonstrates how the states of the FSM pass from one to the next.

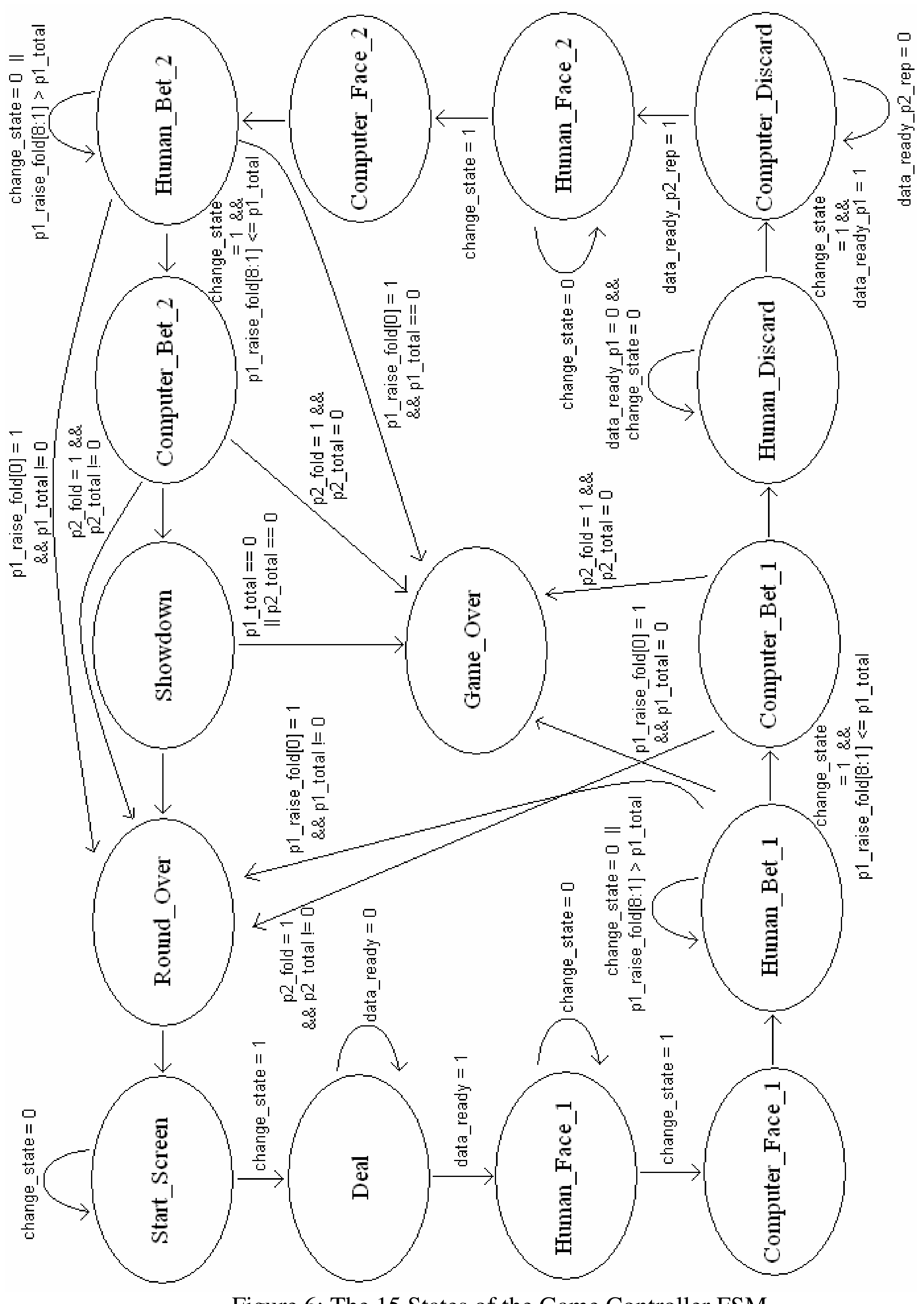
Start_Screen
Deal
Human_Face_1
Computer_Face_1
Round_Over
Human_Bet_1
Showdown
Computer_Bet_1
Game_Over
Computer_Bet_2
Human_Discard
Human_Bet_2
Computer_Face_2
Human_Face_2
Computer_Discard
change_state = 0
change_state = 1
data_ready = 0
data_ready = 1
change_state = 0
change_state = 1
change_state = 0 || p1_raise_fold[8:1] > p1_total
change_state = 1 && p1_raise_fold[8:1] <= p1_total
p2_fold = 1 && p2_total != 0
p1_raise_fold[0] = 1 && p1_total != 0
p1_raise_fold[0] = 1 && p1_total = 0
p2_fold = 1 && p2_total = 0
p1_total == 0 || p2_total == 0
p2_fold = 1 && p2_total = 0
p2_fold = 1 && p2_total != 0
p1_raise_fold[0] = 1 && p1_total != 0
p1_raise_fold[0] = 1 && p1_total == 0
change_state = 0 || p1_raise_fold[8:1] > p1_total
change_state = 1 && p1_raise_fold[8:1] <= p1_total
data_ready_p1 = 0 && change_state = 0
change_state = 1 && data_ready_p1 = 1
data_ready_p2_rep = 0
data_ready_p2_rep = 1
change_state = 0
change_state = 1

Figure 6: The 15 States of the Game Controller FSM

Start_Screen is the first state in the game. During Start_Screen, the user sees the opening screen on the video display, where he is asked to choose an opponent. The computer player has a personality in order to better simulate a live opponent. The personalities are Nasty and The Bluffer. Nasty is an aggressive player, and The Bluffer always bluffs. The default personality is called Eye for Eye and simply does whatever the user does. Start_Screen tells the video controller to output the instruction "Choose your opponent", and we remain in the Start_Screen state until the user enters this choice. When the user selects an opponent, the change state signal goes high, and we move to the state Deal.

During Deal, the cards are dealt. User and computer each have a hand made up of five cards. We represent a card as a six bit binary number. Bits 0 and 1 are the suit (00 = Clubs, 01 = Diamonds, 10 = Hearts, 11 = Spades), and the other four bits are the value of the card. Values can range from 2 to 14; Jack is given a value of 11, Queen 12, King 13, and Ace 14. Thus, each card will be one of the 52 cards in a deck. A linear feedback shift register can be used to generate a pseudo-random sequence, so we use it to generate the 10 cards making up the user and computer's original hands. In addition, we generate 10 additional "replacement cards" during the deal state, which will be used later to replace cards discarded in the Discard state.

Also, the cards are sorted in the deal state. During the deal state, we output a signal to the sorter indicating that a sort is needed. The sorting module takes as input five randomly generated 6 bit cards and outputs the same five cards but in sorted order. The sorter uses a sorting algorithm called the odd-even transposition sort, which is a parallel version of bubble-sort. Since it takes more than one clock cycle to complete sorting, we wait until we receive a data ready signal from the sorter. Once it is received, we transition to the next state, Human_Face_1.

It is necessary to sort the cards because of the way our Hand Evaluator works. The type of hand a player has needs to be determined so that hands can be compared to find a winner and so that the computer can make decisions appropriate for its hand rank. It is much easier to evaluate the hand if the cards are sorted. A series of registers indicate whether a player's cards are all the same suit (last two bits equal), are in consecutive increasing order, contain exactly three cards of the same value, contain two pairs, or contain one pair. We can then use the values of these indicators to determine the type of hand we have. For example, if all the cards are the same suit and are also in consecutive increasing order, the hand type is a Straight Flush. Or, if the hand contains both three cards of the same value and two more cards of a different value, the hand type is Full House.

The hand evaluator also finds the best cards for the computer to discard. There are a series of discard signals. Each is a 5 bit number, where each bit represents a card in the hand. If a bit is equal to 1, the corresponding card should be discarded, and if it is 0, the card should be kept. For example, when the computer has a Three of a Kind, the discard indicator, Trip_discards, will have 3 bits equal to 0 - representing the three cards in the triple, and 2 bits equal to 1 - representing the unmatched cards that should be thrown away.

After dealing, the next state is Human_Face. In the Human_Face_1 state, we output the instruction, "Choose poker face", and remain in the state until the user makes this choice. Once the choice is made and the change state signal goes high, we move to the state, Computer_Face_1. Here the computer gets his turn to choose a face. Each personality considers his hand, his total money, the user's total money, and the user's face to pick his face. The Nasty personality always frowns, unless he has a good hand (better than three of a kind) and the user is frowning, in which case he makes a neutral face. The personality called The Bluffer frowns if he has a good hand or more money than the user and smiles if he has a bad hand. Both players now have on their poker faces, and we transition to the next state, Human_Bet_1, which is part of the first betting stage.

We first output the instruction "Place a bet" and wait for the user to select the amount of money he wants to bet. This amount is taken as an input and is 9 bits long. The least significant bit represents whether or not the user is folding. If it is equal to 1, it means the user folds. The computer is declared as the winner, and the pot is deposited into its bank account. If the user has no money left when he folds, we transition to the state Game_Over since one of the players is bankrupt. If he still has money left, we transition to the state Round_Over. If the least significant bit of the player's bet is equal to 0, it means he is staying in the game, and we need to test that his bet amount, (the other 8 bits) is a valid one. If the bet is greater than the user's total amount of money, it is an invalid bet. We output the instruction "Bet too high" and remain in the Human_Bet_1 state, waiting for the user to enter an allowable bet. Otherwise, the user is making a valid bet. We increase the pot size by the user's bet, increase the user's total bet amount by the new bet size, and subtract the bet from the user's bank account. If the user's bet is equal to the computer's, the betting stage is over and we transition to the state Human_Discard. If not, it means the user has made a raise, and we transition to Computer_Bet_1 to give the computer a chance to match the user's bet or re-raise.

In the state Computer_Bet_1, the computer decides what bet to make based on its personality, its hand, its total money, the user's face, and the user's bet. Regardless of personality, if the computer has the hand Royal Flush, the best hand possible, he bets all the money he has. If this amount is equal to the user's last bet, we transition to the next state Human_Discard. Otherwise, we return to the state Human_Bet_1 to give the user a chance to match the computer's raise. If the computer does not have a Royal Flush, it will either match the user's bet, raise, or fold. After a match or raise, the bet amount is added to the pot, added to the computer's total bet amount, and subtracted from the computer's bank account. The state transitions to Human_Discard after a match, and it transitions back to Human_Bet_1 after a raise. After a fold, the user is declared as the winner and the pot is deposited in the user's bank account. We transition to Game_Over if the computer has gone bankrupt and to Round_Over if it still has money left.

In the Human_Discard state, the instruction "Discard" is output, and we wait for the user to confirm his discard choice. The user's discard choice is sent to the game controller as a 5 bit binary number. Each of the numbers represents a card. If the number is 0, the corresponding card has not been selected for discard; if it is 1, the user wants to throw away that card. Each card selected for discard is replaced by one of the replacement cards that was randomly generated in the Deal state. The need_sort signal goes high to prompt sorting to begin, and we wait for the sorter's data ready signal before transitioning to the state Computer_Discard.

In the Computer_Discard state, we use the discard signals described earlier. For each bit, we randomly replace the corresponding card if the bit is equal to 1 and keep the card if the bit is equal to 0. We output a high need sort signal to the sorting module, wait for a high data ready signal, and then transition to Human_Face_2.

The next four states - Human_Face_2, Computer_Face_2, Human_Bet_2, and Computer_Bet_2 give the user and computer the chance to select new faces and make new bets based on their new set of cards. They perform as the states Human_Face_1, Computer_Face_1, Human_Bet_1, and Computer_Bet_1 do.

Next we move to the Showdown state, where the winner is determined. Since each hand has a rank, we compare the rank of the user's hand with the rank of the computer's hand. Whoever has the highest ranking hand is the winner and receives the money in the pot. If the two players have hands of equal rank, the game is a tie, and the pot is split. If the two players both still have money left, we move to Round_Over. After a high change state signal from the video controller, we move back to the Start_Screen state and begin a new round. If either of the two players are bankrupt, we finally transition to Game_Over, where the game must be reset in order to start a new game.

Facial Expression Detector

The Facial Feature Detector will detect the facial expression of the player (smile, straight face, frown, or angry). Since it will be too difficult to detect such features on a human face, we created four simple "face cards" each with one of the above expressions easily detected (Figure 7).

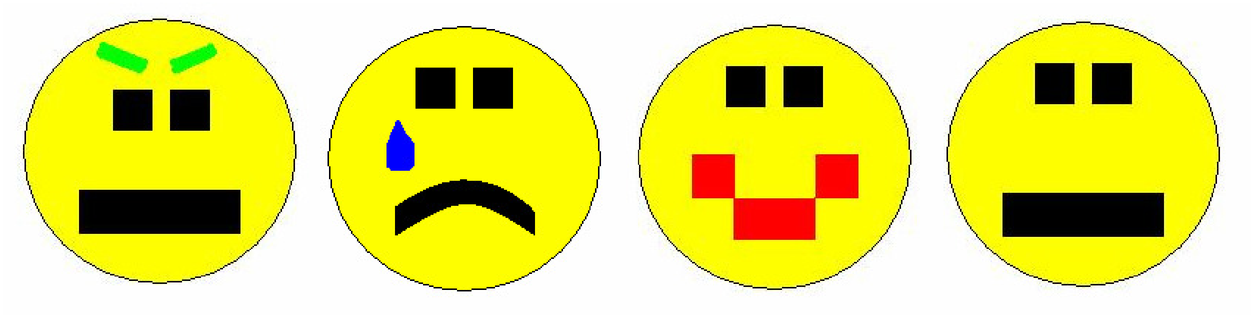

Figure 7. Predefined "Poker Faces"

The input to this module will come from a video camera which monitors the player. Instead of interpreting a constant feed of video data, each player will enter his/her face at specific times during a round (as specified in the game controller FSM), by pressing the labkit's enter button to register the current video input. This module will output one of four possible values (representing the four different facial expressions) to the game controller, which will modify the computer's strategy accordingly. The Poker Table module will also display the user's face on the screen.

After the user registers a face (by pressing enter), the Facial Expression Detector will scan through the next frame of data. Since each of the predefined poker faces has a distinctive color feature ("smile" has a red mouth, "frown" has a blue tear, "angry" has green eyebrows, and "straight" is simply a yellow face), this module will simply keep a counter of the number of red, blue, and green pixels in the appropriate frame and will determine which face the user entered by seeing which color was most represented in the frame. If neither red, blue, nor green was significantly represented in the frame, the facial feature detector will assume the user entered a straight face. After detecting a face, this module will send a face_ready signal to the Poker Table, at which point Poker Table will display the face on the monitor.

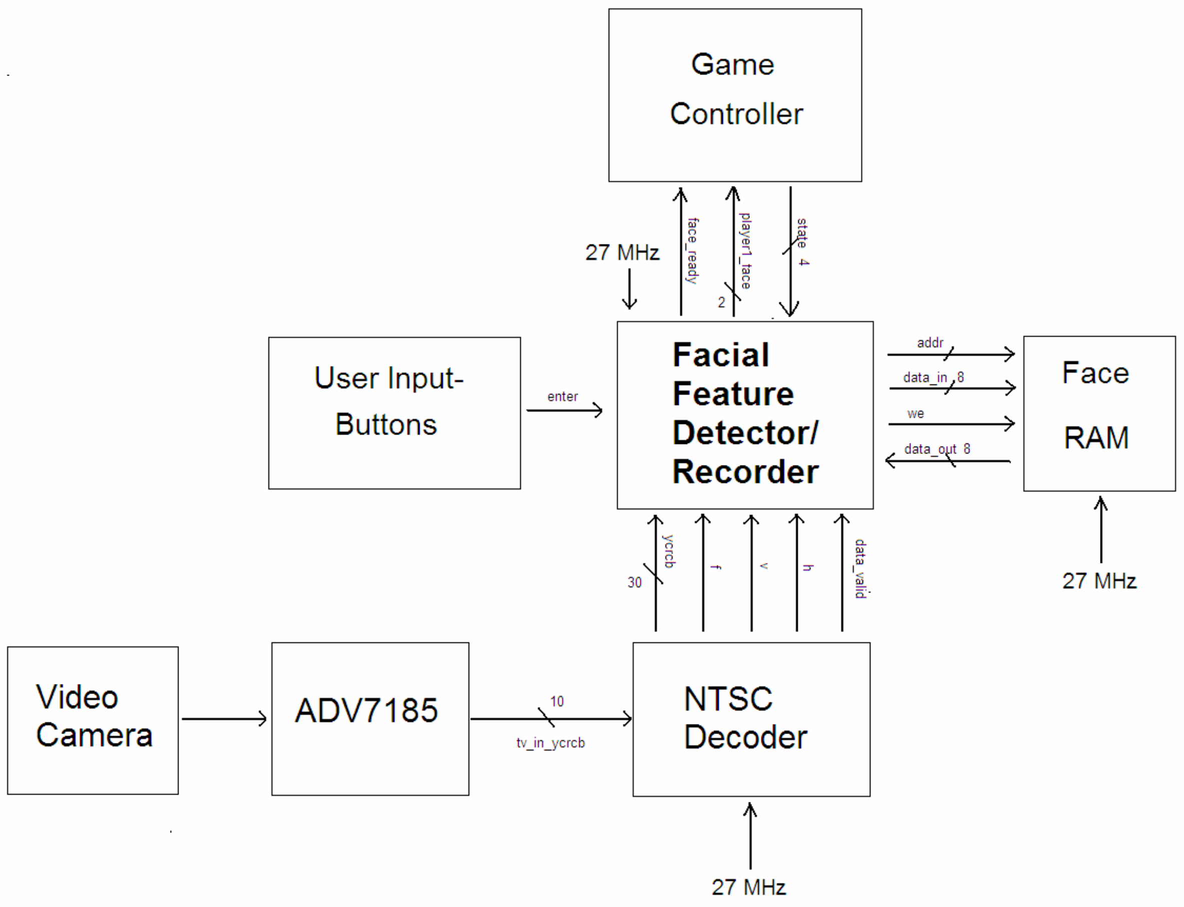

Figure 8. Detailed Block Diagram for Facial Expression Detector

Testing and Debugging

Video Controller:

Fortunately, the Video Controller was straightforward to test, since all of the outputs were displayed to the screen. One problem I encountered, though, was the lack of synchronization between signals. For example, when I didn't synchronize the user inputs, such as the button presses, the video display didn't update correctly. Another problem I encountered was "fuzzy" looking sprites. Unfortunately, I didn't have time to fix this problem, but I imagine the problem results from the fact that the logic to combine all the sprites (by "or-ing" them together) took longer than the clock period. If I had time, I would have pipelined the sprites so that the chains of or-gates would be shorter between registers and their total propagation delay would be shorter than the clock period.

Game Controller:

The modularity of the design made it easy to test and debug the system piece by piece. The necessary processes for each state were written and tested one at a time. After giving a section simulated user inputs, the products from the section were output on the LEDs and displayed on the hex display. Each piece functioned as expected. Dependent pieces were then gradually put together; all components functioned properly, and game play cycled correctly. The most common problem was a delay in output. This was fixed by removing elements from the clock dependent always blocks and putting them in different always blocks, which were dependent on the correct signals. Finally, the game controller was connected to the video controller. This integration caused some problems. While both game controller and video controller functioned properly separately, when put together, the states of the Game FSM did not always run in the correct order. To try to debug this, we output various important signals on the hex display and used the LEDs; but we were unable to find what caused the problem. We have not been able to solve the problem and achieve full functionality to the entire system.

System Integration:

The system was hardest to debug once we combined our blocks together. First of all, we had different ideas of which signals were to be generated by which block, so first we had to go through our blocks to make sure our inputs and outputs were compatible. After making these changes, though, we still had problems synchronizing some of our signals. The Poker Table states weren't changing correctly with the user inputs. If we had a few more days, I think we could have worked through this problem.

Conclusion

We tried to implement this poker game in a straightforward, testable, and modular form. This made debugging simple. It also means that it is easier for another person to understand the code and its purpose.

Integrating the major controllers was the most difficult and posed the most problems. Most of the testing and debugging was devoted to this process. While the game controller and video controller functioned separately, there were problems with game behavior once they were combined. Most of the problems were resolved, but we have been unable to correct some. While a user is still able to play a game of poker against the computer, he sometimes has to go through too many bet or discarding states than he should; and he is unable to choose his opponent since the first state is skipped. Also, we have not been able to fix the problems that we had with the facial feature detector, and it does not work. For this reason, we have the user select their poker face using the labkit switches instead of by showing a picture to the camera.

The investigation, analysis, and experimentation that went into planning and implementing the project educate us about the challenges that engineers face when designing similar systems. Having to make design choices for an original project help to cement concepts learned in class since we had to put them into practice on our own.