**ABSTRACT**

The main points of lab4 was to extend a microprocessor based on the SRAM system in lab4 on our FPGA board using Qsys tools in order to further develop an asynchronous serial network communication between boards. Once we finished building the hardware subsystems and network interfaces on the board, eclipse console, oscilloscope and SignalTap were used to help test our results. Lastly, we confirmed our asynchronous communication system actually worked by comparing the sent and received results of connected boards displayed on LED and on eclipse console.

Lab5 was developed based on the asynchronous communication system built in lab4. It required us to build a game application that involves multiple players (two or more). The game would be played on their own system and communicated with opponents’. In addition, an extra SRAM was brought into this game application for storing statistics. We finally made three game applications as final project. The games that we made are “Hangman” and “Fight the Landlord” (2-player version/3-player version), which are traditional western countries word guessing game and traditional Chinese poker game respectively, based on what we learned from lab4. The detailed rules and specific implementations about the games will be discussed later.

**INTRODUCTION**

Lab 4 mainly required us to build an asynchronous communication system between two boards. More specifically, there will be one transmitter and one corresponding receiver at the same time. The purpose for building asynchronous communication system is to deal with irregular intervals between sending sets of data. We firstly implemented a microprocessor system that utilizes the Altera NIOS II core using Quartus II and Qsys tool. The microprocessor is able to send and receive parallel data bus. We then implemented the serial-to-parallel and parallel-to-serial network interfaces, which basically can transfer the parallel data bus to serial data for sending and transferred the serial data back to parallel data bus for receiving. The network interfaces were implemented with several FPGA functional modules including counting modules BSC (bit sample count) and BIC (bit identification count), shift registers, signal control modules and clock control modules. Lastly, we integrated the hardware microprocessor and network systems together to achieve the asynchronous communication goal. Each single board will have two pins assigned for communication: one for sending data out and one for receiving data in.

Based on what we had done on lab 4, we built the game applications that support communication between players. The game applications should be able to both communicate with microprocessor and with other players who play this game. For communicating with microprocessor, player would input some necessary information and microprocessor will provide some information back for the player. For example, for the game “Hangman”, microprocessor will ask players to input which role the player wants to be, what the length of the word is if the player would like to give the vocabulary, and what the player’s guess is if the player would like to guess the vocabulary. For the game “Fight the Landlord”, the communication between the player and microprocessor is also needed because microprocessor needs to know players’ score preferences, playing orders and show how many cards and what cards the players have in hand currently. For communicating with opponents, we connected first board’s output pin to second board’s input pin and second board’s output pin to first board’s input pin to accomplish the communication purpose. For 3-player version “Fight the Landlord”, the communication wire is built between each pair of boards. Thus, microprocessors are able to exchange game information and proceed the games. The rules for games mostly follow the original game rules, with slight modification. More details about game rules can be found in design specification and appendix.

**DISCUSSION OF THE LAB**

**Design Specification**

Lab4 Communication System:

* The communication system is able to transmit data out and receive data in with microprocessor and multiple components built in Quartus with Verilog on FPGA. It must be able to handle irregular intervals between data. At the transmitting state, the system needs to add a start bit and an end bit, add one parity bit at the end of the data for error checking, and change the parallel data form to serial data form. After completely sending out the data, the FPGA will send a signal to microprocessor to let it know that the data had been transmitted out. At the receiving state, the system needs to detect the start bit, change the serial data to parallel data and store it in microprocessor. After completely receiving in the data, the FPGA will send a signal to microprocessor to let it know that the data had been received. After the communication system is built, the user can use it to either transfer out or receive in any 7 bit numbers at one time.

Lab5 Game Application Hangman:

* Hangman is a two player word guessing game. One player will question the other player by typing the words and giving the number of guesses. The other player who guesses the word will aim to guess what the word is within the given number of guesses.

Lab5 Game Application Fight the Landlord (2 players version):

* Fight the Landlord (2 players version) is a two-player card game and a simplified version of three-player of version of Fight the Landlord. There will be one player being landlord and the other being farmer. Whoever gets rid of all its cards wins the game. Please refer to Fight the Landlord User Guide for more details.

Lab5 Game Application Fight the Landlord (3 players version):

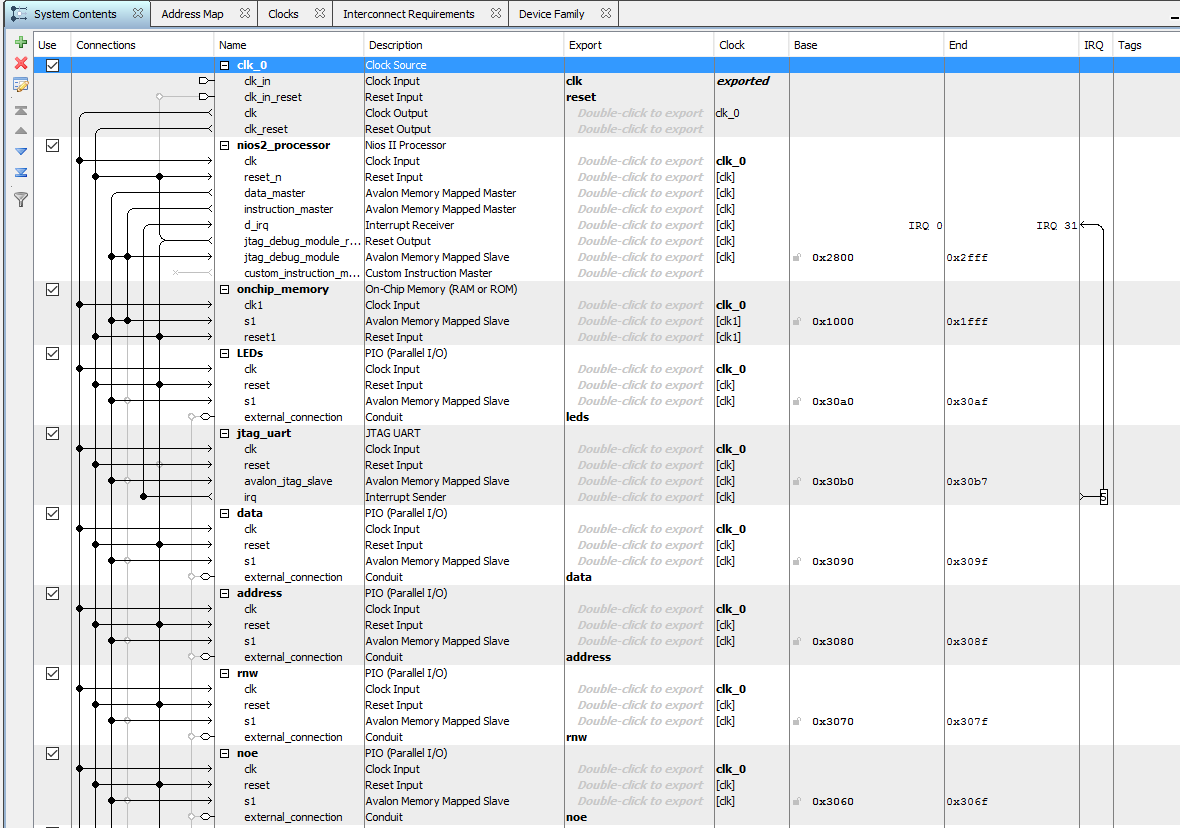
* Fight the Landlord (3 players version) is a traditional Chinese three-player card game. There will be one player being landlord against the other two players being farmer. Please refer to Fight the Landlord User Guide for more details.

**Design Procedure**

Lab4 Communication System:

We designed and implemented an asynchronous serial communication system on FPGA with Verilog, and then we wrote the driver for transmitting data and receiving data in C using Eclipse. A succinct procedure are shown below:

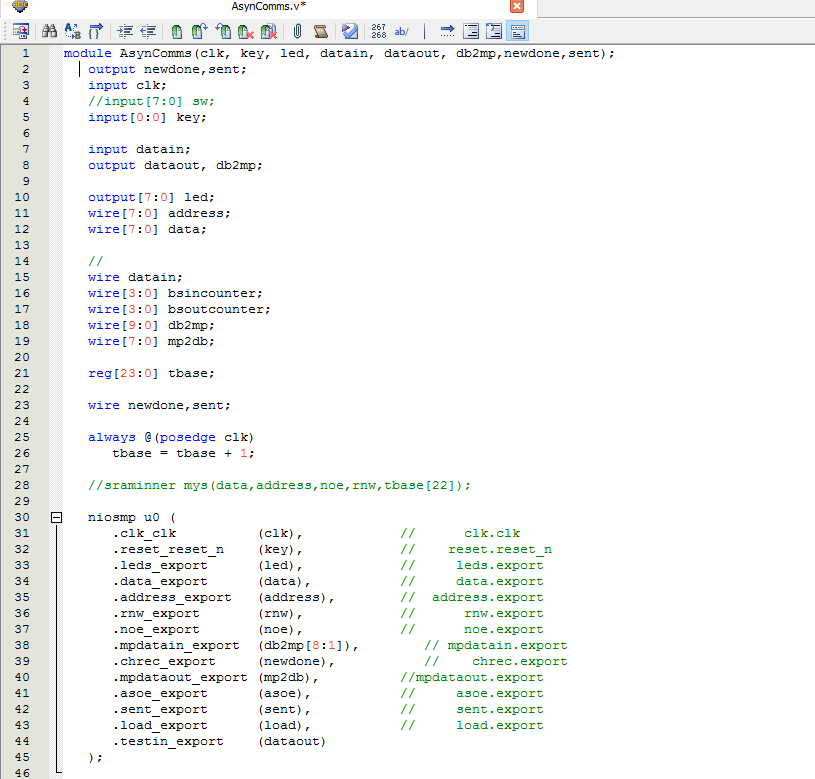
1. Set up the core microprocessor system. We followed the hardware setup tutorial for lab3 and add more appropriate inputs and outputs for the system.
2. Generate the microprocessor in Qsys and instantiate the microprocessor in the top level module.
3. Design and implement modules for communication network interface, including Bit Identification Count, Bit Sample Count, S/R Clock Control, Shift Register, and Start Bit Detect. The detailed information about how we specifically build network interface can be found on hardware implementation.
4. Integrate the FPGA network interface and microprocessor.
5. Do "smoke test". Specifically, connect the output of a system with its own input, and write driver using C in Eclipse to test if the transmitted data can be received by itself.
6. Connect two boards together and use appropriate C codes in Eclipse for driving purpose to test if the board can receive the data sent by the other.



**Figure 1,** the complete system of NIOS II microprocessor for Communication System in Qsys part I



**Figure 2,** the complete system of NIOS II microprocessor for Communication System in Qsys part II



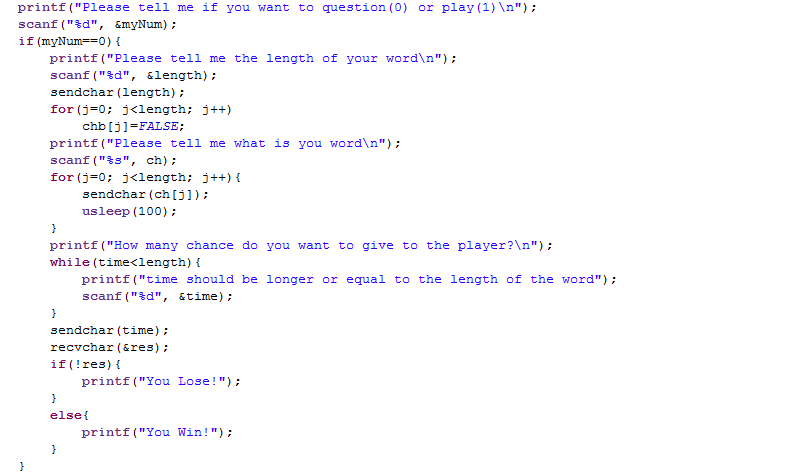
**Figure 3,** the top level module of Communication System



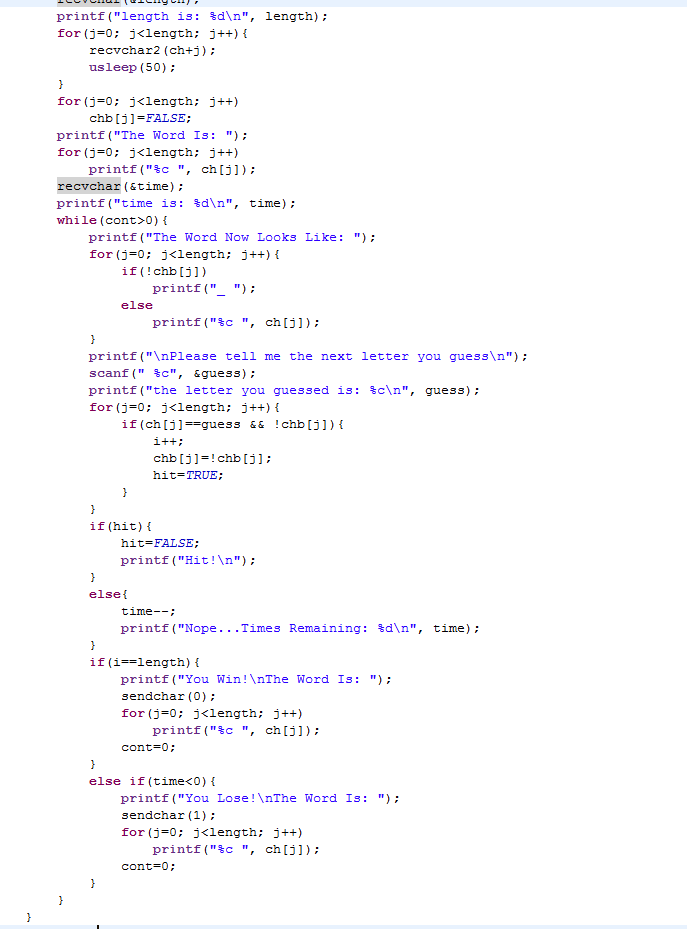
**Figure 4,** Communication system test driver in C language, sending out “Hello 1234”

Lab5 Game Application Hangman:

The Game Application Hangman was designed following the design specification. Firstly, the hardware part is exactly the same as the communication system discussed above. For the software part, we firstly identified that this game involves two players. One player needs to type the length of the word, the characters in the word, and the number of chances for the other player to guess. So the players need to choose which role he/she wants to be before the game starts (i.e., to distinguish the board for questioner and the board for guesser). Then, the codes are divided into two parts by a condition statement. One part for questioning and another for guessing, are executed by the boards for questioner and for guesser, separately. The necessary information including length of the word, the characters in the word, and the number of chances will be transferred from the board for questioner to the board for guesser. Procedures including calculating the number of chances left, determining whether or not the player guesses correctly will be executed on the board for guesser. Since the questioner also should know the result, the board for guesser will send the result to the board for questioner after the player finishing the game by running out of chance or finding the correct word. More detailed information about the implementation using C language codes can be found in Software Implementation part.



**Figure 5,** Game Application Hangman, question part



**Figure 6,** Game Application Hangman, guessing part

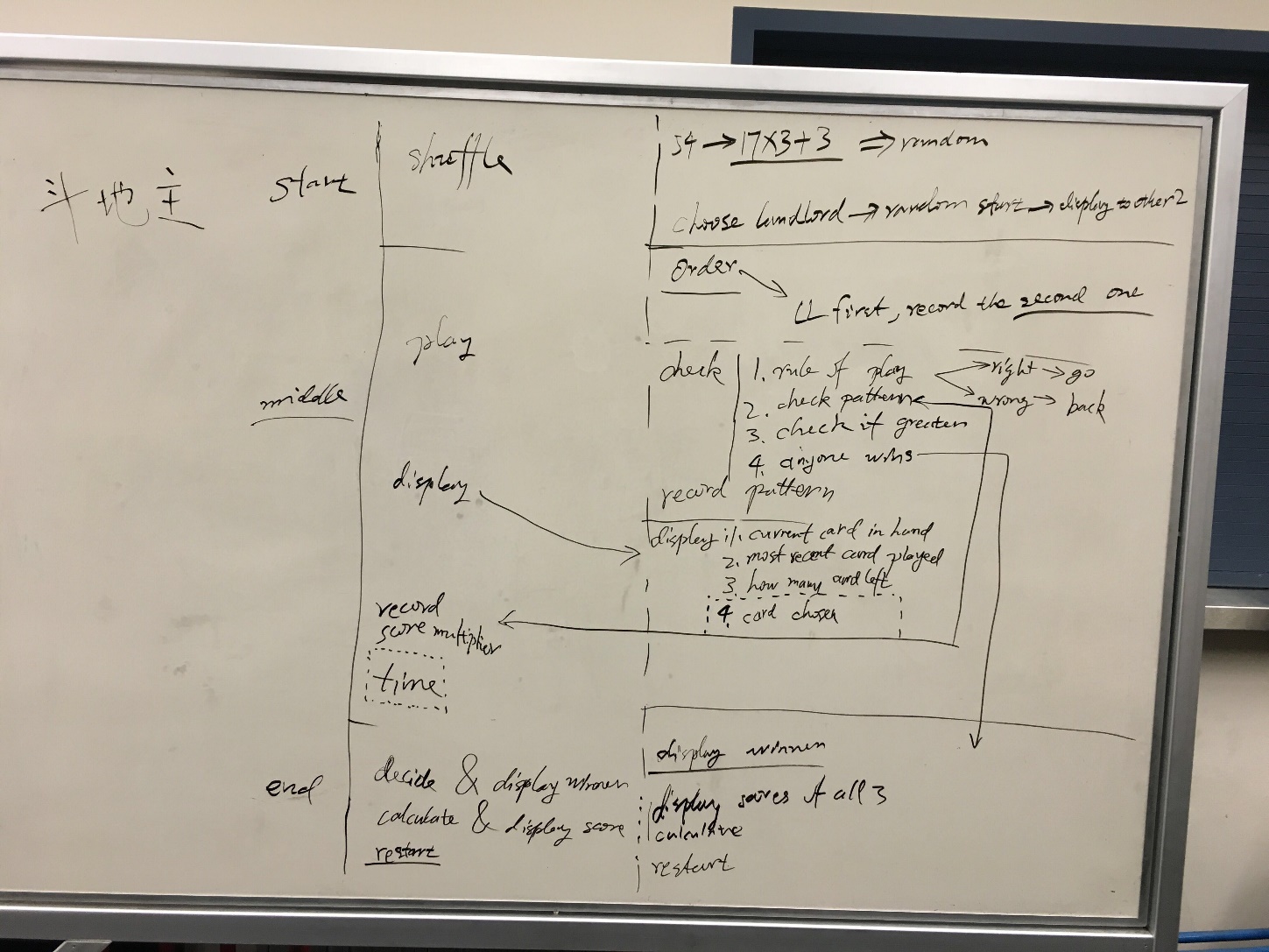
Lab5 Application Game Fight the Landlord (2-player version/3-player version):

The hardware part different versions of Game Application Fight the Landlord are also the same as communication system implemented in lab4. For software part, the 2 players version of this game is implemented because we could not resolve the time constraints of communication in 3 players version at the beginning. Thus 3 player version will be discussed firstly below (or the two versions are discussed as a whole in some sense), and then the modifying procedure due to the differences between the two versions will be discussed in detail.

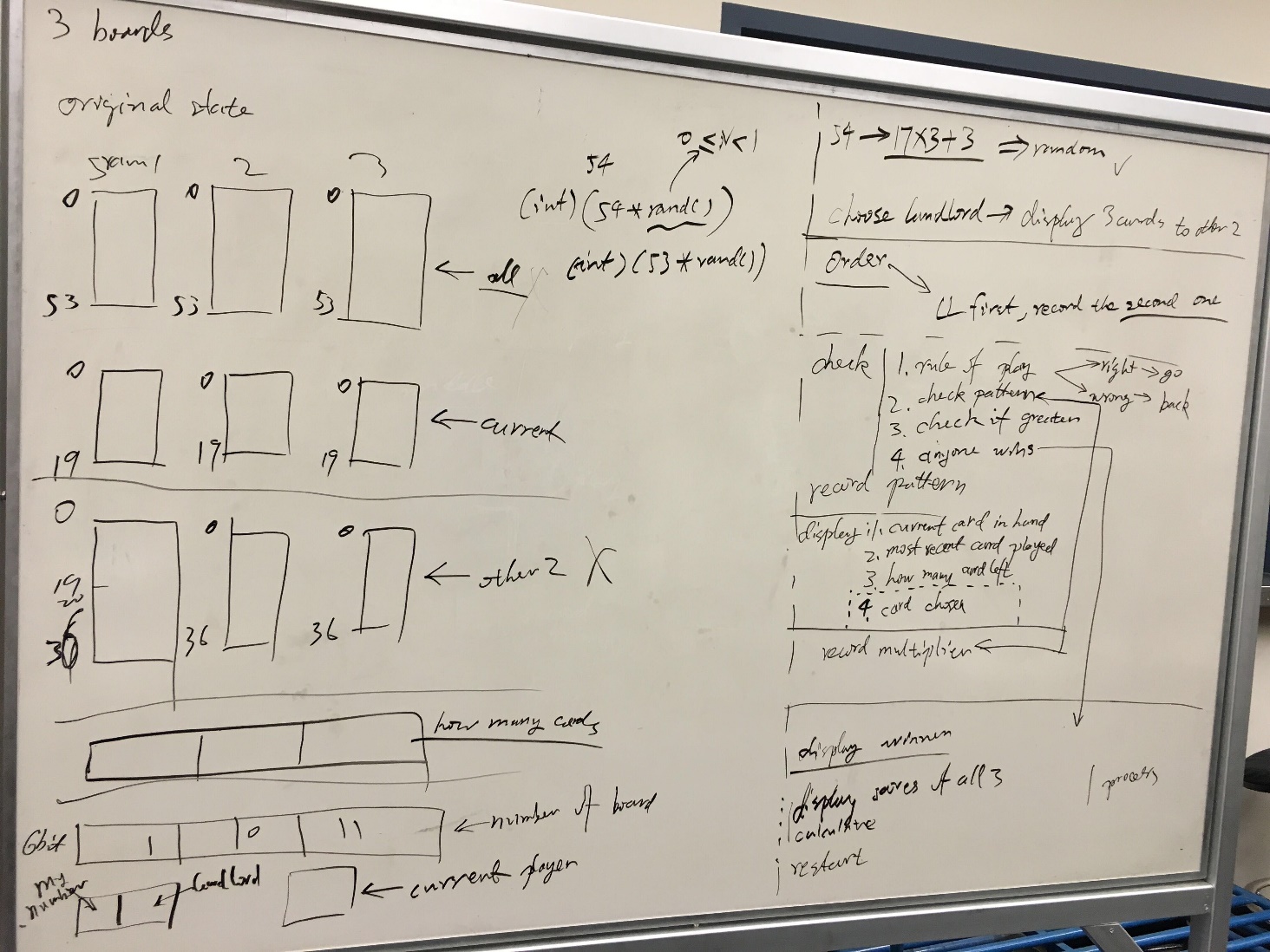
Since Fight the Landlord is a card game, we first decided how to represent different cards. Considering the rule of this game, specifically, 3 is the smallest card and 2 is the largest one except jokers, we decided to use 0 to represent card 3, 1 to represent card 4, and so on up to K. Then card A is represented by 11 and card 2 is represented by 12. Finally, 13 represents black joker and 14 represents red joker. This way makes it easier to compare cards in the program.

Then, the way of communication is designed. Since there are three players in this game and one board need to communicate with other two, the first board received a message need to forward the message to the next board. We decided to use the most significant bit of a 7-bit data can be transmitted in one time between two boards as a "forward bit". If a board receives a 7-bit data with the most significant bit as 1, then it will change the forward bit to 0 by and (&) with 0x3f, then forward this data to the next board and return the data to the process on the current board.

Then the full game process was analyzed from top to bottom, mainly about functions required in the game. The diagrams are shown below.



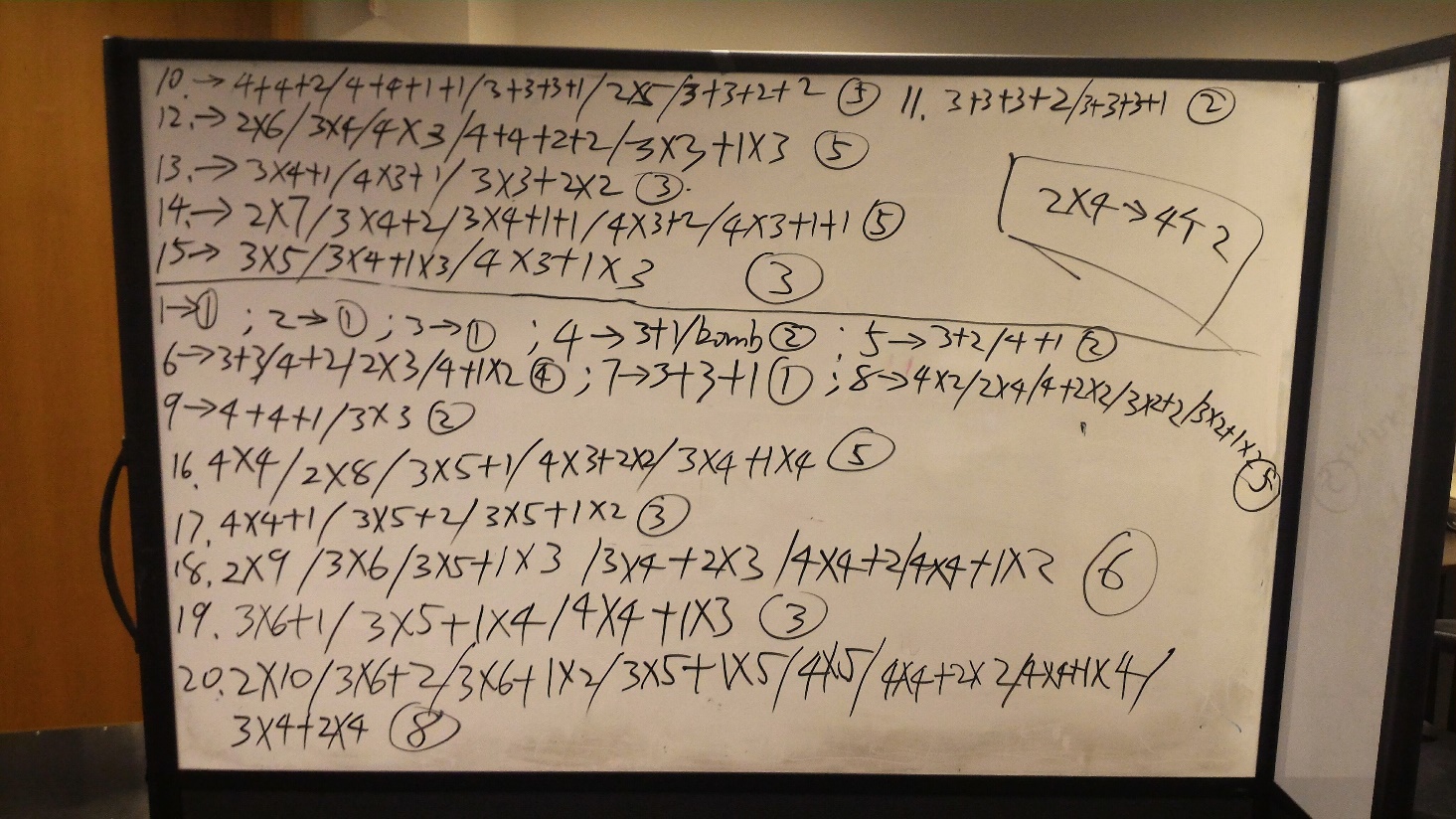
**Figure 7,**  Functions Required in Fight the Landlord, Part I



**Figure 8,** Functions Required in Fight the Landlord, Part II

Note that, these pictures just shows functions required in the game in order of the process of this game, but not strict procedure. Also, the functions in the graph may be slightly different from those in the final design. To avoid confusion, the following discussion about the process is mainly based on our final design rather than directly discussion of functions in the diagrams.

As shown in the pictures, the procedure is firstly divided into start, middle and end. The start part is mainly about shuffling and auction. The middle part is about playing cards and keeping track of score multiplier. The final part is find the winner and calculate the score. In out final implementation, the last 3 procedures, playing cards and The final part is find the winner and calculate the score. In out final implementation, concerning about simplicity, playing cards, keeping track of score multiplier, decide the winner and calculate the score are integrated to play part. Then, more detailed functions are designed. We decided to use SRAM in shuffle part to model the cards left in the shuffling process. In play part, we decided to store the variables related to the functioning of the game, such as score multiplier, current cards in hand, number of cards in three players hands, number of landlord and current player, in each board, and synchronize them between three boards after every possible modification. Also, since we need to check whether the cards played are legal, we designed pattern code for simplicity of communication. Different pattern are firstly written down, as shown below.



**Figure 9,** Patterns of Cards can be Played in Fight the Landlord  
(For Electronic Version, Please Refer to "pattern code.xlsx" Uploaded to Dropbox)

Then pattern codes are designed based on these patterns. Pattern code has 3 decimal digits. The most significant 2 digits is the number of cards played in this round. The least significant digit is the specific pattern number (0 reserved for sequence), in order of the patterns written down. With the pattern code, and variable keyValue, one can know the card played by the other player (though not very user friendly). And this way, we only need to transmit 4 bits for keyValue, and 8 bits for pattern code between boards. Since 8 bits is larger than the maximum length in one transmission of both versions of Fight the Landlord, it is split into two 6-bit (7-bit in 2 players version) data and send continuously, and then restored after received by receiver(s). For more detailed implementation of 3 player version of Fight the Landlord, please refer to the software implementation part.

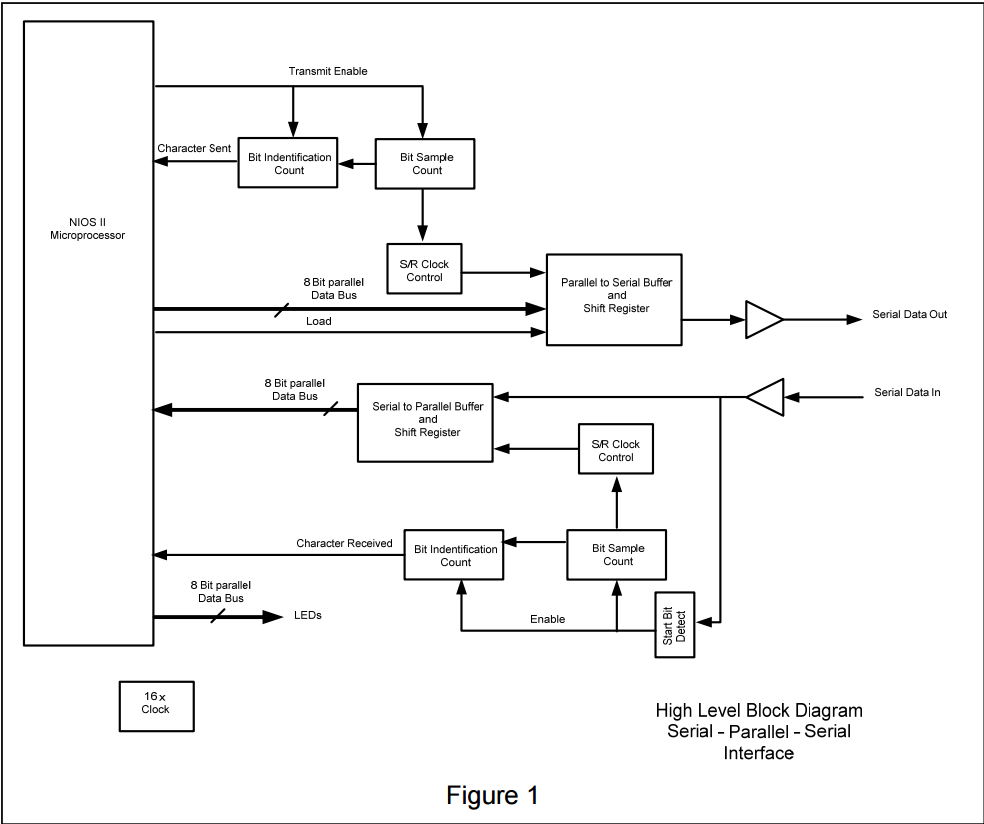
The main difference in game rules between 2 players version and 3 players version is auction part. While in 3 players version the numbers given by players are compared, in 2 players version we used a "rock paper scissor" game to decide which player is the landlord. In the "rock paper scissor" game, "rock", "paper", and "scissor" are represented by numbers 3, 2, and 1, separately. The programs first checks the player number. The first player will be asked to input the number first and send to the second board, and then wait to receive the number chosen by the second player. The procedure of the second player is reversed, i.e., that board will receive first, and then ask for input and send.e.

The implementation of other parts were all modified in a similar way, that is, in 3 players version for simplicity we need a "while-do-while" pattern (a while loop to receive data and wait for its turn, and do its work when the current player is using this board, and use another while loop to receive data and wait for other board to finish their work if necessary), but in 2 players version, an if-else statement similar to the one used in the "rock paper scissor" game is simple enough. For more detailed implementation of 2 player version of Fight the Landlord, please refer to the software implementation part.

**System Description**

Lab4 Communication System:

The communication system has two main functions: transmitting data and receiving data. For the system as a whole, it has an output GPIO pin and an input GPIO pin. For the microprocessor, it has inputs datain (8-bit bus), character received and character sent, and has outputs dataout (8-bit bus), transmit enable, load, and LED (8-bit bus). At the transmitting side, the output data is sent to the shift register directly by microprocessor in parallel form. The time control function blocks will convert the parallel data to serial for communication, after the data are output, a character sent signal will be sent to the microprocessor. At the receiving side, the input serial data is converted back to serial form using the same time control function blocks as transmitting side. After the data are received, a character sent signal will be sent to the microprocessor. The communication system is built according to following lab4 spec graph.



**Figure 10,** High Level Block Diagram for communication system, from our Lab spec

There are several time constraints for this communication system. Firstly, since the highest data transfer rate on a wire is about 9600 bps, we need to choose an appropriate time base. In our system, we used tbase[9] as the clock for Bit Identification Count and Bit Sample Count, and thus the data transfer rate is 50M/2^14 = 3051 bps. We chose our own time bases here since we are implementing our own system. However, when we communicate with other group’s board, we must agree on the same time base in order to communicate successfully. Finally, the data transfer rate of output of the system should be consistent with the input. This time constraint is guaranteed by using the different instances of the common module with the same time base for the transmitting side and the receiving side. There are no side effect in this system.

Hangman:

Hangman is implemented as one of our final projects. It is a two-player game. The NIOS console in Eclipse is used as the user interface. One player will give the length of the word, the characters in the word, and number of chance to guess as inputs to computer. The other player will input the letter for guessing and the computer will tell that player if the guess is correct and number of chance left. At the end of the game, the computer will tell the players if they win or lose. There will be one player win and one player lose. In this game, the length of the word, the characters in the word, the chances the guesser has is transmitted from the board for questioner to the board for guesser. And the result of the game is transmitted from the board for guesser to the board for questioner. There is no side effects. A time constraint must me met in this game is that, player 1 (the guesser) need to type in player number, 1, before player 0 input anything except the player number 0. I.e., player numbers must be typed in before the game start. Another time constraint is that, there should be a certain amount of delay between receives of data to avoid repeated reception. For detailed discussion and analysis, please refer to Analysis of Any Errors part. The last time constraint is that, one board must begin waiting to receive data before data is sent from another board. This time constraint is met by a certain time of delay.

Fight the Landlord (2-player version)

The 2-player version of Fight the Landlord is implemented as one of our final projects. It is a two-player game. The NIOS console in Eclipse is used as the user interface. Players need to type in their player number (0 or 1), then the shuffle procedure starts automatically. In this process, the cards players get are transmitted between the boards and being tracked by SRAM in the boards (each board need to keep track of all cards in the two players' hands to avoid repeated cards, though this information is not shown to players). Then a "rock, paper, scissor" game is played to decide the landlord. In this process, the number each player inputs to the console is transmitted. For the play procedure, each player should type in the number of cards and the cards he/she wants to play. The program can translate these cards into pattern code and key value using a switch-cases statement, which distinguishes different cases of different card numbers played, and in which each case nests a series of if-else statements for comparison of card patterns. A sort function is written for simplicity of comparison of card patterns. This is a very simple sort algorithm with time complexity of O(n2), but is good enough for this project. The program is responsible for guarantee the cards played conform to the rule. In play procedure, the pattern code (at least 8 bits) and key value (at least 4 bits) are transmitted. Since the length of pattern code is larger than the maximum data length the communication system can transmit in one time, it is split into 1 bit and 7 bits, and is transmitted in two times. The receiver restores the pattern code. Finally, each of the boards check if anyone wins by checking if the number of cards in anyone's hand currently is 0. All the procedures, including shuffle, auction, and play are written in a if-else style. Specifically, if a player is the current player, then the board will ask for user input, send data and wait for reception of data, else the board will first wait for reception of data and then ask for user input and send data.

There are several time constraints in this game. Firstly, player 1 should input his/her player number before player 0, since the system shuffles cards automatically after player 0 input play number. The rest times constraints are the same as the last two of Hangman, i.e., there should be a certain amount of delay between receives of data to avoid repeated reception, and one board must begin waiting to receive data before data is sent from another board. There are no side effect.

Fight the Landlord (3-player version):

The 3-player version of Fight the Landlord is implemented as one of our final projects. It is a three-player game. There is a message forward system in 3-player version of Fight the Landlord since one board need to communicate with other two. In this system, the first board received a message need to forward the message to the next board. The most significant bit of a 7-bit data can be transmitted in one time between two boards as a "forward bit". If a board receives a 7-bit data with the most significant bit as 1, then it will change the forward bit to 0 by and (&) with 0x3f, then forward this data to the next board and return the data to the process on the current board.

Same as the 2-player version of this game, the NIOS console in Eclipse is used as the user interface. Players need to type in their player number (1, 2, or 3), then every player should input 4 arbitrary integers for the program to generate random numbers before the shuffle procedure starts automatically. The shuffle process has a similar mechanism as the 2-player version of Fight the Landlord, the difference is 3-player version program uses "while-do-while" style rather than "if-else". This difference will be discussed later in this section. The auction part is completely different from the 2-player version. In this procedure, three players input a number from 1 to 3, and the player who gives the largest number will be the landlord (the earliest player if two or more players give the same number). The play procedure also has similar mechanism as the 2-player version, and the "while-do-while" style will be discussed later in this section. Also a little difference with the 2-player version in communication in play procedure is that the maximum data length within one transmission is 6 bits due to the forward bit, but this difference nearly does not effect the transmission process, except the 8-bit pattern code is split into 2 + 6 bits. Finally, each of the boards check if anyone wins by checking if the number of cards in anyone's hand currently is 0. All the procedures, including shuffle, auction, and play are written in a "while-do-while" style. Specifically, a board first go into a while loop to receive data and wait for its turn, and do its work when the current player is using this board, and use another while loop to receive data and wait for other board to finish their work if necessary.

The time constraints are the same as those in 2-player version, except for 3-player version, player 2 and player 3 should finish inputting their random numbers before player 1 finishes. There are no side effect.

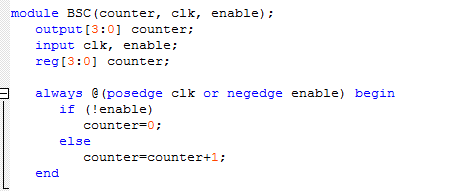
**Software Implementation:**

Lab4 Communication System:

The communication system is built based on the block diagram provided in lab spec (figure 7 in this report). Each function block or signal in the block diagram is discussed in this section. Also, the difference between our implementation and the diagram is also discussed below.

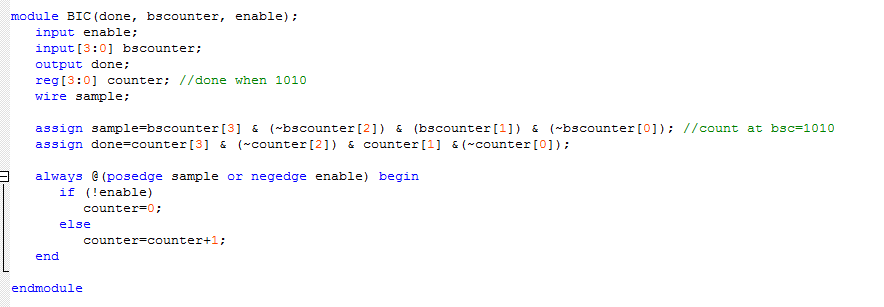
In the top level module, all the function blocks are connected according to the diagram, as shown in figure 3.

The signal “transmit enable” is sent directly by microprocessor, which is the control signal for both BSC (Bit Sample Count) and BIC (Bit Identification Count).



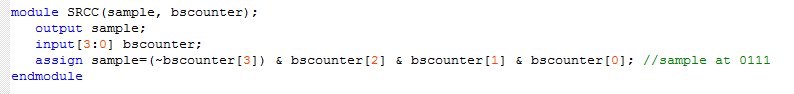
**Figure 11,** Bit Sample Count implemented using Verilog code

As Figure 11 indicated, Bit Sample Count has two inputs: a time base signal and a control signal. When the control signal is a digital one, the four bits output reg bus “counter” increments at the positive edge of the time base signal. Basically, counter counts from 0 to 15 repeatedly when it is enabled, and this module finally produces a 16xClock.



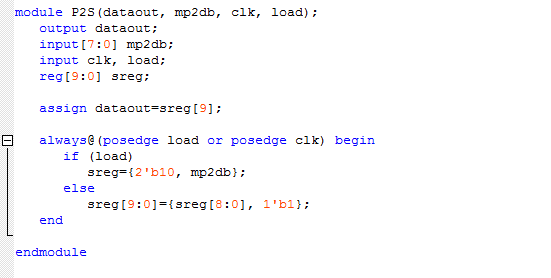
**Figure 12,** Bit Identification Count implemented using Verilog code

As figure 12 indicated, Bit Identification Count takes two inputs: a time control signal and a counter. The input counter is also the output counter generated by Bit Sample Count. The wire sample will be digital one when the input counter is 1010 in binary (10 in decimal), since there are 10 bits for every data transmitted or received. At the positive edge of sample, the reg bus counter will increment. The output signal “done” will be a digital one when the bus counter is 1010 in binary to indicate that the system has completely sent a data out or received a data in.



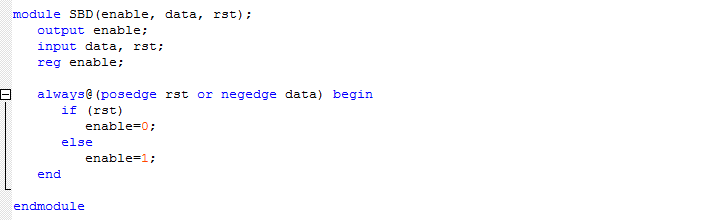
**Figure 13,** S/R Clock Control implemented using Verilog code

As figure 13 indicated, S/R Clock Control is a very simple function. It takes the output counter from Bit Sample Count as input. The output signal will become a digital one when the counter is 0111 in binary because 0111 indicate the middle. The output will be used as input clock signal for P2S (parallel to serial) to shift data out of the shift register, or as input clock signal for S2P (serial to parallel) to shift data into the register, i.e., to sample the bit currently on the wire.



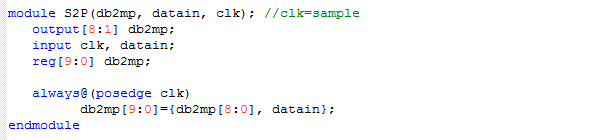
**Figure 14,** Parallel to Serial implemented using Verilog code

As Figure 14 indicated, Parallel to Serial (P2S) has three inputs. Load signal will load the data in and save it as right 8 bits of sreg. The left 2 bits of sreg will be initially 10 in binary. The clock signal is the output of S/R Clock Control. Mp2db is the data generated directly by microprocessor in parallel form. When the load is 0, sreg will do left shift at the positive edge of clock signal. The output “dataout” is the output data in serial form. It is assigned to the most important bit of sreg. So it would initially be 1, then 0, then the data in serial form from most significant bit to least significant bit. The most significant two bits 10 of sreg is used for detecting start bit on receiving side.



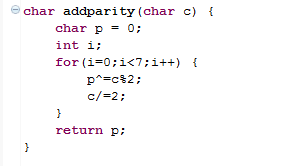
**Figure 15**, Start bit detect implemented using Verilog code

As figure 15 indicated, Start Bit Detect will detect the start bit of the incoming serial data. The reset is controlled by character received. Character received signal will become digital one when the system completely received the incoming data. The output enable signal will be the control signal of Bit Identification Count (receive side) and Bit Sample Count (receive side).



**Figure 16**, Serial to Parallel implemented using Verilog code

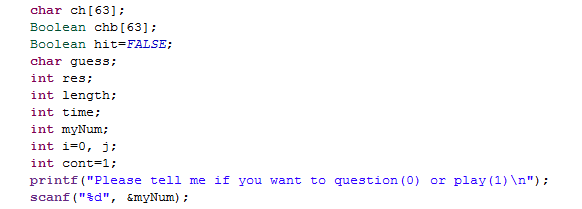
As figure 16 indicated, Serial to Parallel has two inputs signal and will transfer the serial data to parallel data and store it at microprocessor. The way to store the data is using left shift. Each incoming data bit will be stored to the least significant bit of the dp2mp.



**Figure 17**, Add parity function implemented using C code

As figure 17 indicated, this function adds an extra bit as parity bit to data. It will append either a "0" or "1" to the end of the data to make the number of bit "1" in data to be even.

Hangman:



**Figure 18**, hangman codes for introduction

Figure 18 is the introduction part of the hangman. It first defines variables and then asks user for input as questioner or guesser. The rest of codes can be divided into two parts for players depending on the player's choice at this place.

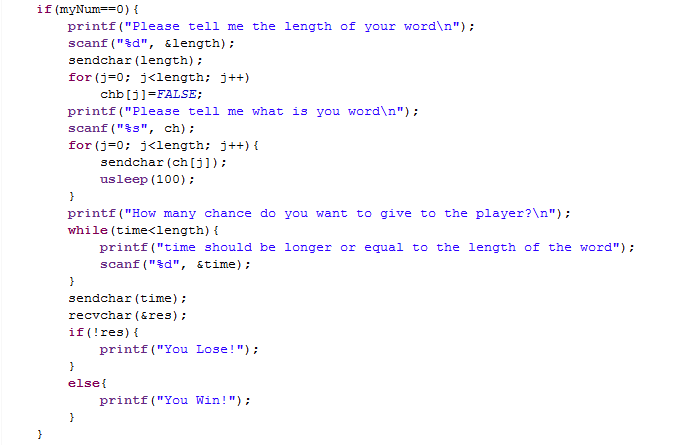
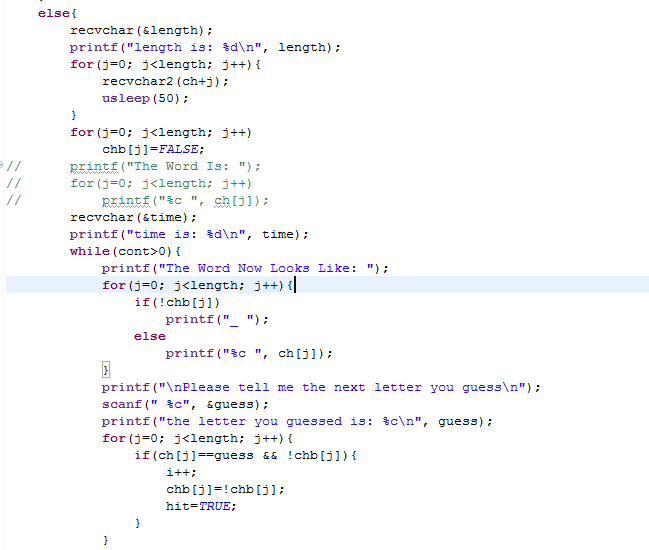
**Figure 19**, hangman codes for giving question

Figure 19 shows the codes for giving the questions. It firstly asks the player who just chose "0" the length of the word. It then transmits the length to other board. Then it asks the player to input the word. Next, it transmits the word to the other board. The other board who will guess the word actually already knew the vocabulary but it will not display it to the player for the game purpose. Lastly, it asks the player to input the number of chance for guessing and transmit it to the other board, and then wait to receive the signal from the other board to determine either win or lose.



**Figure 20**, hangman codes for guessing part I

Figure 20 indicates the codes for guessing part. It initially waited for the signal from the questioning board. So a time constraint here is that the player who would guess the word must input "1" before the other player starts to type the length of the word. After receiving the length and the number of chance, it will display them to the player. Then, it goes into a while loop of guessing. In the while loop, it will firstly display how the vocabulary look like. Initially the word is filled with "\_" because the player hasn't guessed anything. Then, it asks the user for input as next guessing letter. It will compare the guessing letter with the vocabulary. If the vocabulary contains the guessing letter, the variable "hit" will be TRUE. Note that if the guessing letter has already been guessed, it will take it as wrong even the vocabulary contains that guessing letter.

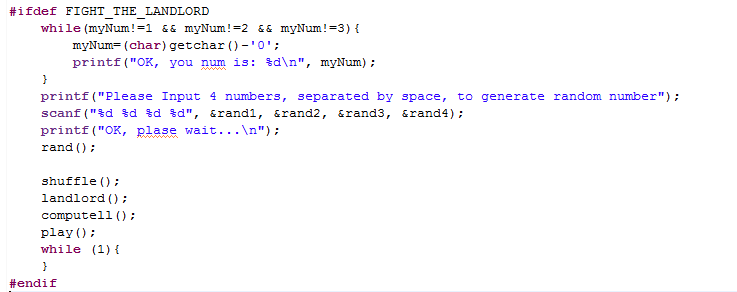


**Figure 21**, hangman codes for guessing part II

Figure 21 indicates the second part of codes for guessing part. If the player guesses a word correctly, it will display "Hit". Otherwise, it will display "Nope" and the number of chances left after decrementing 1. Then it will check if the player wins or loses. If the total correct guessed times equal to the length of the vocabulary, it will display the player wins. If the number of chance left is smaller than 1, it means the player has used up all his chances and it will display that the player loses. At the same time, it will send the signal to the other board. So if this board loses, the other board wins and if this board wins, the other board loses.

Fight the Landlord:

The 2-player version and 3-player version will be discussed together here but the difference will be explained if necessary. Also, please refer to the header file and other C codes like player control and play control of our uploaded file to catalyst for more details. They have detailed comment for explaining. Due to the considerable length of our codes, I would only introduce the major function blocks of the games.

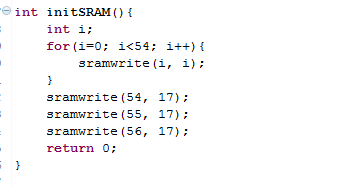


**Figure 22**, fight the landlord codes in main

Figure 22 indicates the codes of "Fight the Landlord" in main. In 3-player version, it firstly asks the three players for an input of a player number from "1", "2" and "3", and unique. Then it asks for four arbitrary numbers from each player to generate random numbers. Due to the time constraint issue, the player 1 should finish inputting the 4 arbitrary number after the other two players have finished. In 2-player version, the program asks the two players to choose a player number from "0" and "1", and unique. Due to the time constraint issue, the player who intends to input "0" must input the player number after the other one or two players. The reason will be explained the later. Player numbers input here are used for the order of playing. It determine who will play first at the auction part and the order of playing card. Then, it asks the player to input four numbers used for shuffle the cards in function "rand".



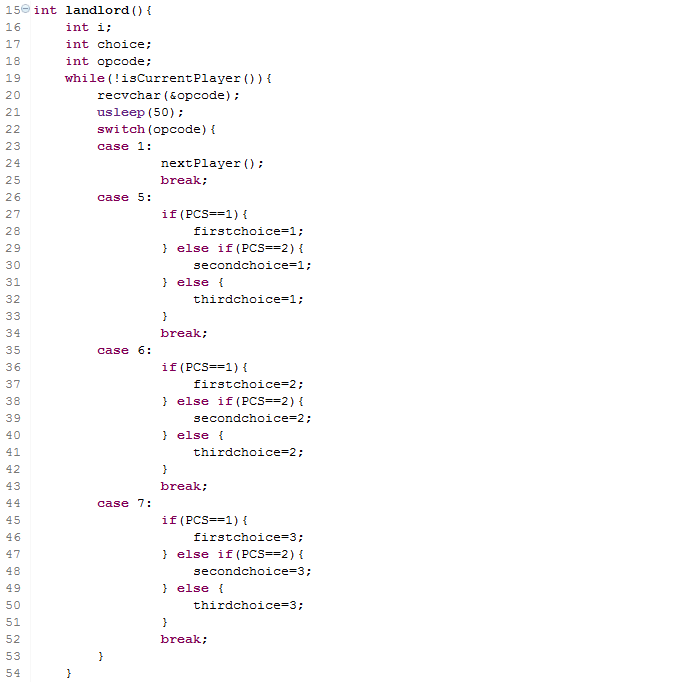
**Figure 23**, fight the landlord codes for shuffle



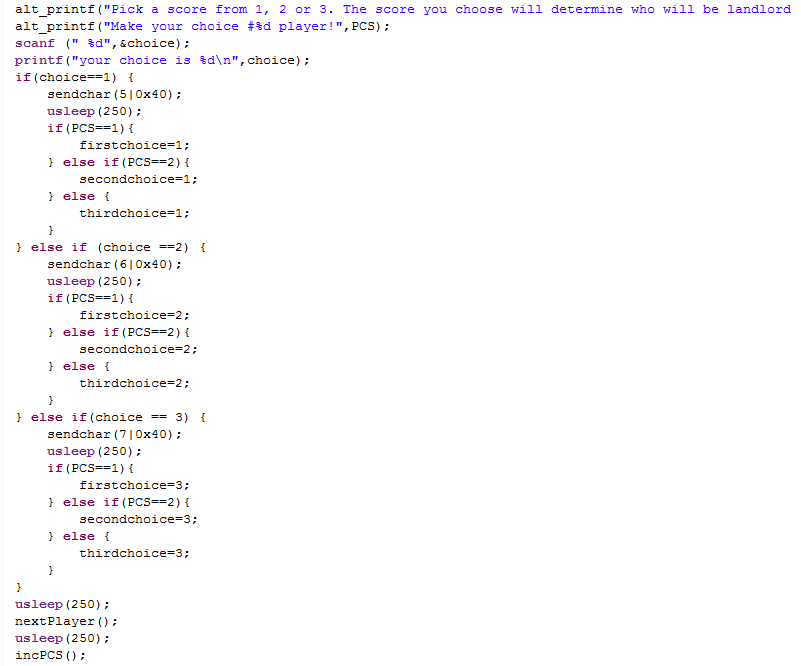
**Figure 24**, fight the landlord codes for initializing SRAM

Figure 23 indicates the codes for shuffling. In this game, SRAM is used for keeping track of cards left when shuffling (every board should keep track of cards other boards have at present when shuffling to avoid repeated cards, though there is no need to store which player has what cards, and no need to give any information to the player in this process), and the number of cards left in each player's hand throughout the entire game. Before going into the while loop, each player will initialize their SRAM as figure 21 indicated, setting the address 54, 55 and 56 to 17 because those three spots are used for storing the number of cards of each player and each of the player will obtain 17 cards before choosing the landlord. Then, the #2 player and #3 player will go into the while loop and wait for commands sent from the #1 player (board), which is the reason why the player who intends to choose "1" at main block must finish inputting the arbitrary numbers after the other one or two players.

The function "rand" shuffles cards in the SRAM randomly. In the for loop, it will choose a card currently in the SRAM and store it at char array "cch" (Current Cards in Hand). Then, it will delete that card in the SRAM and synchronize this deletion by first sending out opcode "3" and then the generated random number (address). After getting17 cards, it will increment PCS (Player in Current State) and send out opcode "1" to tell other board to increment PCS (Player in Current State), so #2 player will get out of the while loop and #1 player will go into the second while loop and wait for receiving signals. The whole process also applies to #3 player. Finally, three players will all get their own cards, the PCS will become 1 again, and all of the three players (boards) will get out of the second while loop. After the shuffle function, each player will has their own distinct 17 cards stored in "cch" (Current Cards in Hand).



**Figure 25**, fight the landlord codes for choosing the landlord part I

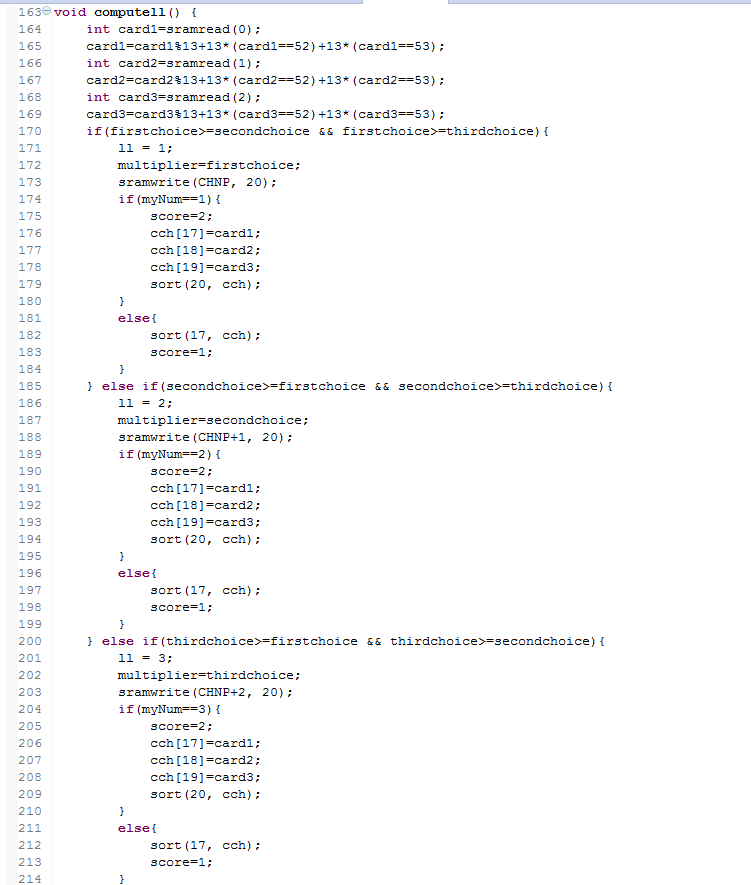


**Figure 26**, fight the landlord codes for choosing the landlord part II



**Figure 27**, fight the landlord codes for choosing the landlord part III

Figure 25, 26 and 27 indicates the codes of function "landlord". The function "landlord" will ask each player to input a number from "1", "2", or "3" to represent their score starting from #1 player. The basic concept of this function is similar as shuffle: two players will be in while loop waiting to receive signals and the other player will execute certain codes (choose score for auction in this function). After the player chooses a score, the score value will be transmitted to other two boards.

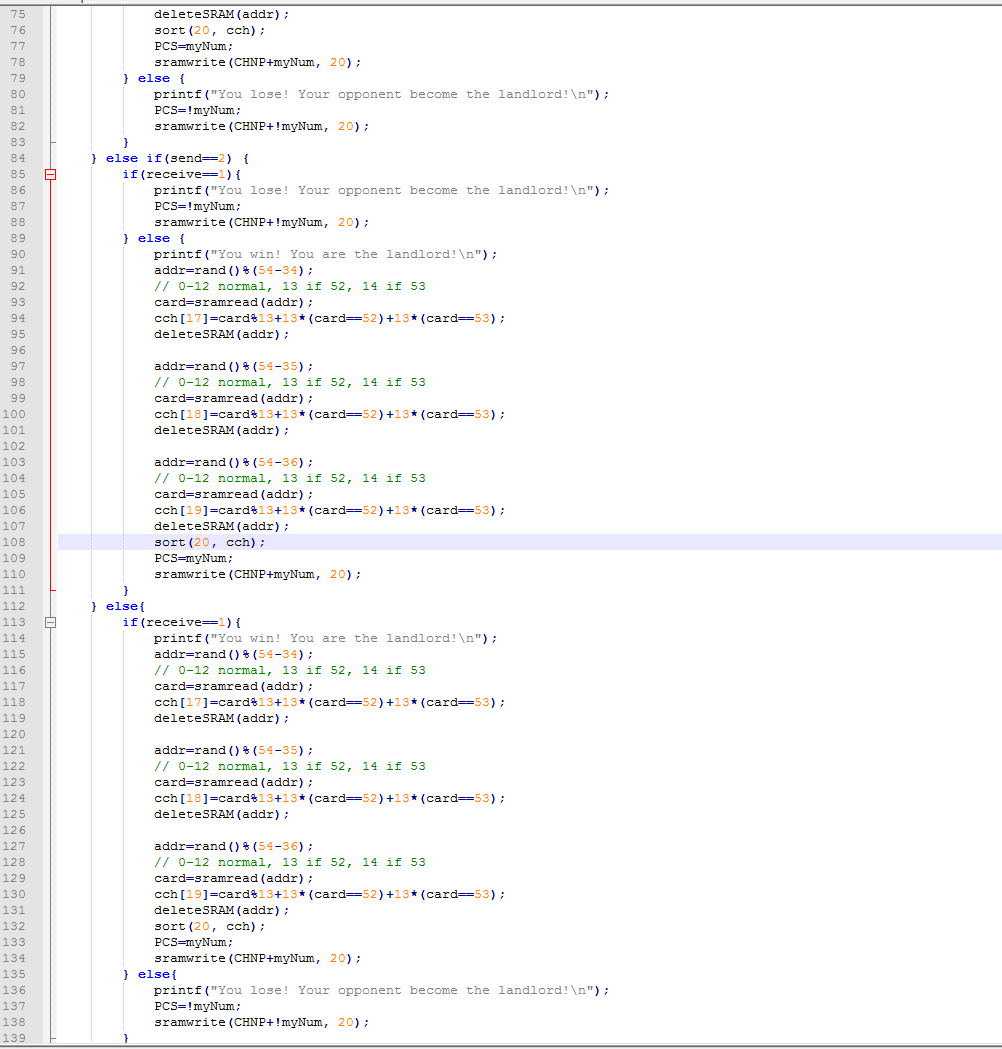


**Figure 28**, fight the landlord codes for computing the landlord

Now, the microprocessor contains each player's score. It will store the extra three cards into each player's cch (Current Cards in Hand) although only landlord has the access to them. Then, it will compute who is the landlord by checking which player chooses the highest score. It will also store the highest score into variable "score", which will be used as a multiplier when computing the scores of the game. The order number of the landlord will be saved into variable "ll" (landlord). Note that if there are more than one player choosing the highest score, the player who chooses the score first will be the landlord.

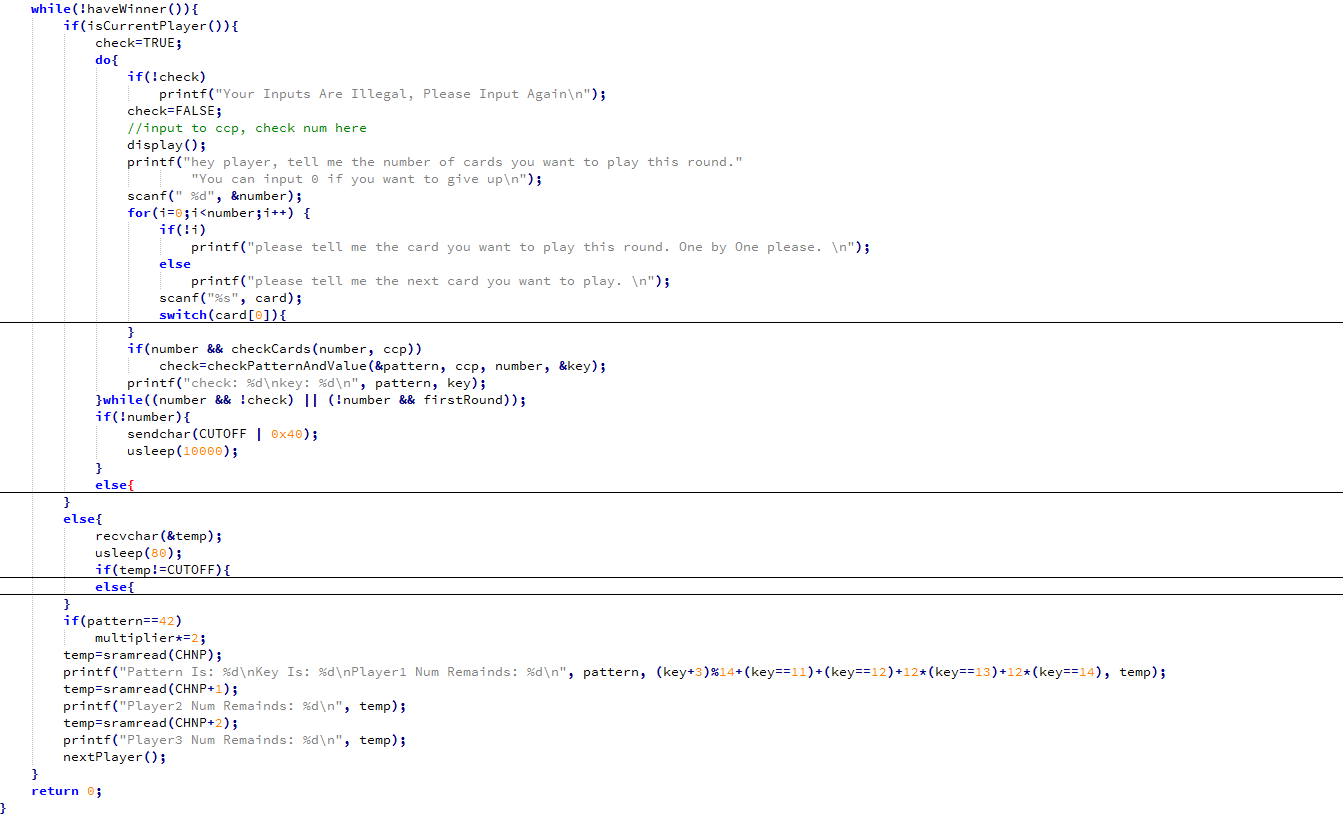


**Figure 29**, fight the landlord codes for computing the landlord (2-player version) part1



**Figure 30**, fight the landlord codes for computing the landlord (2-player version) part2

The main difference between 2-player version and 3-player version is that they have a different auction system (choosing the landlord). For 2-player version, the landlord will be chose by a game "Rock Paper Scissor". In order to simplify this game, we use number "1", "2" and "3" instead. The basic rules is that "1" can beat "2", "2" can beat "3", "3" can beat "1". The microprocessor will ask #1 player to input a number first and then #2 player. The microprocessor will compare the result. If the two players input a same number, then it goes into a tie, which will lead more rounds until one side wins.

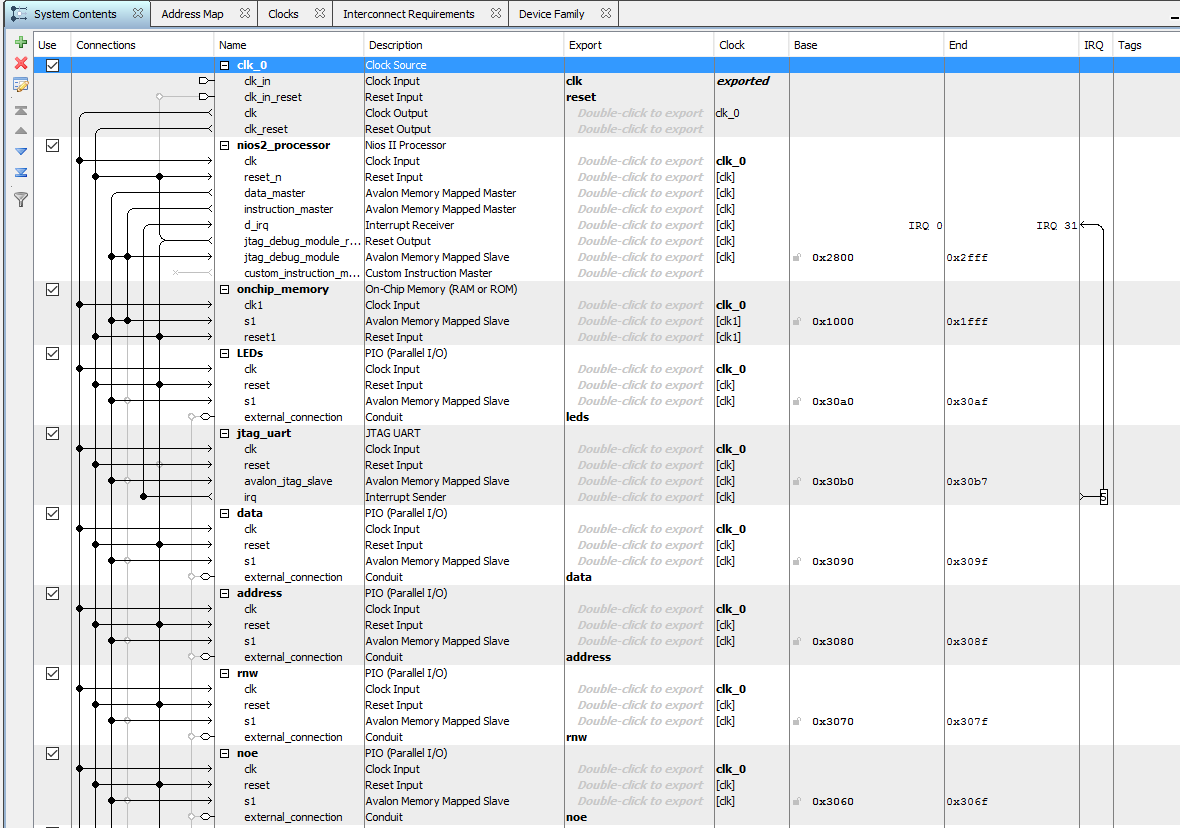


**Figure 31**, the while loop in play function

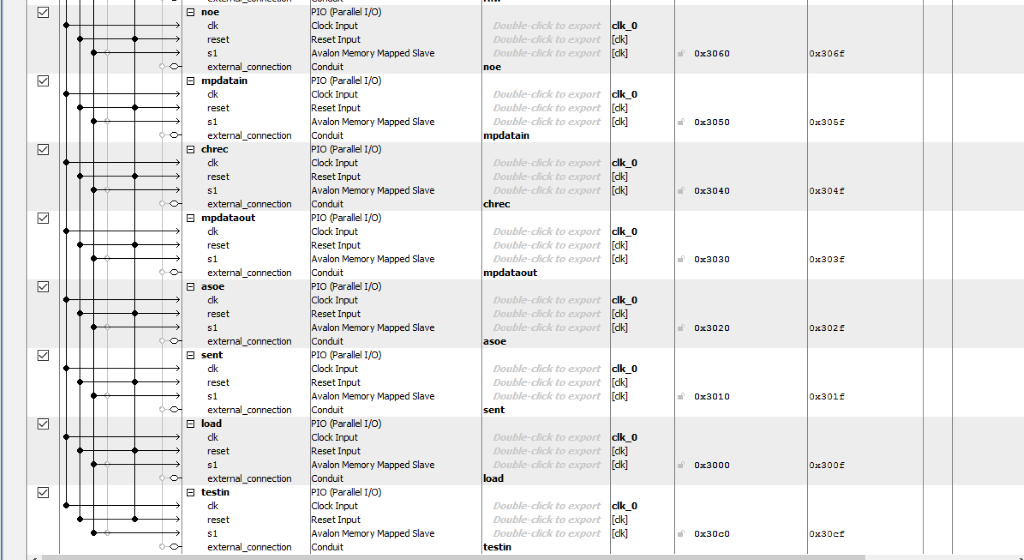
The play part is implemented in a different way with shuffle and landlord. Since players play (or pass) one by one in order and in loop, thus there is no need to worry about going through the two or three players in and only in one time, or random values. So, only a while loop nesting an if-else statement is needed, no matter in 2-player version or 3-player version of this game. If the player is the current player, then the board goes to the if block and ask the player to play cards, check the number and pattern of cards played (pattern check will be discussed later in this section) and send necessary information, including pattern code and keyValue, to other boards if the cards the play intends to play passes pattern check. For simplicity, if a play want to play certain cards, this player need to input the number of cards he or she want to play, and then input the cards one by one. If the player is not the current player, then the board will wait for information from other board. After this process, all three boards will increment PCS.

Codes for card pattern detection and check are all in patternDetection.c file. Due to its considerable size, pictures or codes will not be shown here, but only the implementation. In pattern detection process, the function detectPattern detects the pattern of ccp (current cards played). In this detect function, the cards are firstly sorted for simplicity in actual detection. Then a switch-cases statement is used to distinguish different cases as different card numbers played, and each case nests a series of if-else statements for comparison of card patterns. All 69 patterns allowed in this game can be detected. The detection can extract both the pattern code and keyValue form ccp (current cards played). If the cards played are illegal, i.e., does not belong to any of the patterns, -1 will be returned. If the returned value is -1 or inconsistent with the current pattern code if the pattern code is not 0, the top level function in pattern detection, checkPatternAndValue will return FALSE to indicate that the cards played are illegal. If the pattern of cards played is consistent with the current pattern code, and the keyValue is larger, then the keyValue will be set to the new (i.e., larger) one, and TRUE will be returned from the top level function to indicate a legal play. If the pattern code is 0, which means this player is the first one in this round, and any legal cards pattern is a legal play, and keyValue and pattern code will be set, and the top level function will return TRUE to indicate a legal play. Note that, pattern code 42 means bomb (for more detailed rules of Fight the Landlord, please refer to the user guide), is always legal except if the pattern code is already 42 and the keyValue is larger.

**Hardware Implementation**

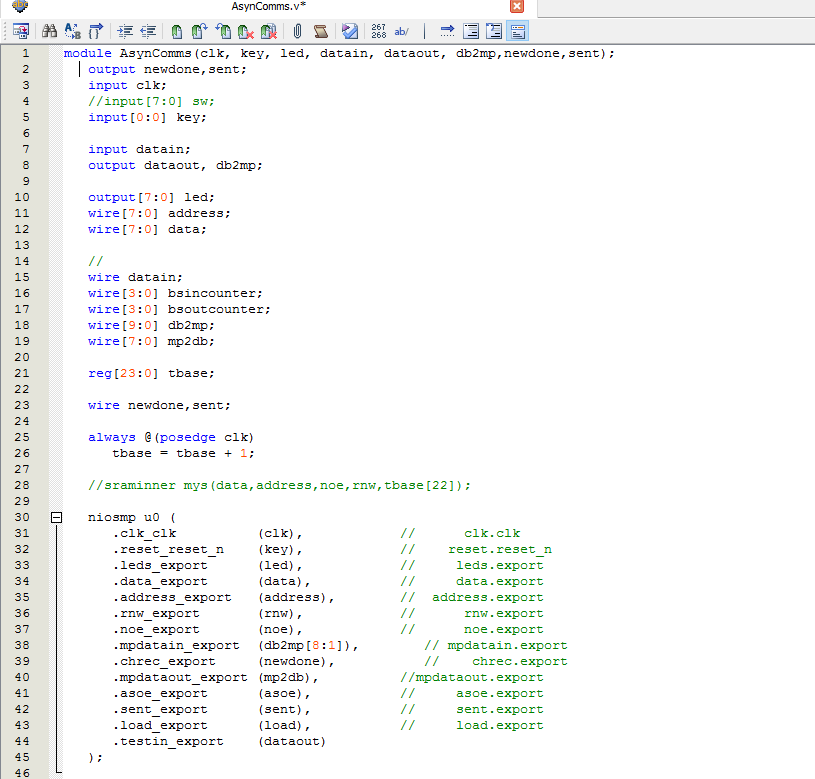


**Figure 32**, the complete system of NIOS II microprocessor for Communication System in Qsys part I



**Figure 33**, the complete system of NIOS II microprocessor for Communication System in Qsys part II

Firstly, different parts of a microprocessor are connected in Qsys. The target FPGA and clock are also set. Then, other basic parts of a microprocessor, including the NIOS II core, on-chip memory, JTAG-UART, PIOs, and interval timer are connected to the microprocessor system as Figure X indicated above. Then base addresses are also automatically specified.



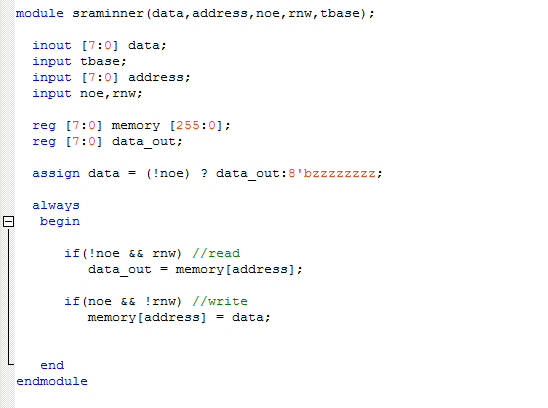
**Figure 34**, the top level module of Communication System

Then, the microprocessor is instantiated in the top level module. Although we actually used tbase[9] for inputs of Bit Sample Count and Bit Identification Count, the 50 MHz clock is used as the clock for the microprocessor, in order to obtain faster processing speed.



**Figure 35**, the pin assignment of the communication system

The dataout and datain are assigned to AG17 and AF16 respectively in order to transmit data out and receive data in. We use wires to connect a board's output pin to other board's input pin to accomplish communication.

  
 **Figure 36**, the SRAM written in Verilog code

The three game we implemented later based on the communication system uses the same hardware except the SRAM, which is brought in for storing some game data in final projects.

TEST Plan

Lab4 Communication System:

Firstly we test it using only one board with its output pin connecting with its own input pin. At this time, we only test it by sending single letter or number. We can read the output through either LED or printing it on Eclipse console. Once it is proved to work, we will test it using two boards with each output pin connecting to the other's input pin. One board will be on transmitting mode and the other board will be on receiving mode. We also begin with single letter or number. Once it is proved to work, we will send "Hello 1234" in sequence to see if the received side can completely and correctly receive the strings.

Hangman:

There are several things need to be tested. Firstly, we want to test if the microprocessor responds correctly to the guessing letter. For example, whether or not the microprocessor can tell the player that he/she guesses wrong when the vocabulary does not contain his/her guessing letter. Then, we will test if the microprocessor can tell which player is the winner correctly. For example, when the player guesses all letters, whether or not the microprocessor can tell the player that he/she wins. When the player uses up all his/her chances, whether or not the microprocessor can tell the player that he/she loses. The win/lose situation should be opposite on the other board.

Fight the Landlord (2-player version):

There are also several things need to be tested. Firstly, we need to test if three players have distinct cards after the shuffling the card. Secondly, we need to test if the microprocessor can compute the correct landlord according to the result of game "rock, paper, scissor". Thirdly, we need to test if the playing procedure works, which means the program should be able to accept legal cards played and can reject illegal cards played. Lastly, we need to test whether the program can tell if there is anyone wins.

Fight the Landlord (3-player version):

There are several things rather similar to those discussed in 2-player version part need to be tested. Firstly, we need to test if three players have distinct cards after the shuffle part. Secondly, we need to test if the microprocessor can compute the correct landlord according to the scores players specified. Thirdly, we need to test if the playing procedure works, which means the program should be able to accept legal cards played and can reject illegal cards played. Lastly, we need to test whether the program can tell if there is anyone wins and can calculate correct scores for different players at the end of the game.

**TEST Specification**

Lab4 Communication System:

* Inputs: The output of the board itself in "smoke test", and the output of the other board.
* Outputs: The output data is specified by users in NIOS console in Eclipse. The input data of the system are displayed both as LED lights and in console of Eclipse.
* Constraints: The board must be ready to receive data in before the data is actually sending out.

Hangman:

* Inputs of questioner (board):
  + From player:
    - The length of the word
    - The characters in the word
    - The number of chances the questioner want to give to the guesser
  + Sent by the guesser (board):
    - The result of the game
* Inputs of guesser (board):
  + From player
    - The letter the player wants to guess
  + Sent by the questioner (board)
    - The length of the word
    - The characters in the word
    - The number of chances the questioner want to give to the guesser
* Outputs of questioner (board):
  + Display either “win” or “lose” in console. Note this should be opposite on the other player’s computer.
* Outputs of guesser (board):
  + At the beginning:
    - The length of the word
    - The characters in the word
    - The number of chances the questioner want to give to the guesser
  + Throughout the game:
    - The letter that the player just guessed is correct or not.
    - Number of chances left
    - What does the word currently look like (display the correctly guessed letter)
* Constraints: The player who wants to guess the word must input “1” before the other player inputs the length of the word.

Fight the Landlord:

For simplicity, this game is divided into 4 parts: initialization, shuffle, auction, and play, and each part is formatted in the same way.

Initialization

* Inputs:
  + From the player:
    - The player number
    - The arbitrary numbers used to generate random numbers (only in 3-player version)
  + From the other board(s): None
* Outputs: None
* Constraints: In 2-player version, player 1 should input the player number "1" before player 0 does. In 3-player version, player 2 and 3 should finish inputting the arbitrary numbers before player 1 finishes doing so.

Shuffle

* Inputs:
  + From the player: None
  + From the other board(s):
    - The cards need to be deleted in the SRAM
* Outputs:
  + To the player:
    - The result of shuffling, i.e., the number of cards the player has
    - For test purpose only, the content of the SRAM after shuffling
  + To the other board(s):
    - The cards need to be deleted in the SRAM
* Constraints: There is no significant constraint.

Auction

* Inputs:
  + From the player:
    - The score the player intended to choose for the auction (3-player version) OR the number input by each player for game “Rock Paper Scissor” (2-player version)
  + From the other board(s):
    - The scores chosen by other players (3-player version) OR the input by the opponent in the game "Rock Paper Scissor" (2-player version)
* Outputs:
  + To the player:
    - The score chosen by each player
    - The player number of the landlord
    - The three extra cards given to the landlord
    - The card in each player's own hands
  + To the other board(s):
    - The scores chosen by other players (3-player version) OR the input by the opponent in the game "Rock Paper Scissor" (2-player version)
* Constraints: There is no significant constraint.

Play

* Inputs:
  + From the player:
    - Number of cards the player wants to play
    - The cards the player wants to play
  + From the other board(s):
    - The pattern code of cards played by the other player
    - The keyValue of cards played by the other player
* Outputs:
  + To the player:
    - Pattern code of cards played by other players
    - keyValue of cards played by other players
    - Whether or not cards played by this player in this round is legal
    - The winner and the score (only 3-player version) at the end of the game
  + To the other board(s):
    - The pattern code of cards played by this player
    - The keyValue of cards played by this player
* Constraints: There is no significant constraint.

**TEST Cases**

Note that, for simplicity, in each test case, the order of outputs need to be checked is the same as the order of inputs.

Lab4 Communication System Case 1:

* Test Setup: Synthesize the microprocessor on board, and run the C program using NIOS II SBT for Eclipse. The board’s output pin connects to its own input pin.
* Input: Single letter "U"
* Outputs:
  + On Board: It displays “01010101” as LED light
  + In Console: It prints “U”.
* Pass/fail criterion: the results should produce the same output as expected.

Lab4 Communication System Case 2:

* Test Setup: Synthesize the microprocessors on boards, and run the C program using NIOS II SBT for Eclipse. The first board’s output pin connects to second board’s input pin.
* Inputs: Letter “U” and then string “Hello 1234”
* Outputs:
  + On Board: It displays “01010101” as LED light and then the Ascii code of “Hello 1234” in binary in sequence (from “H” to “4”)
  + In Console: It prints “U” and then “Hello 1234”
* Pass/fail criterion: the results should produce the same output as expected.

Hangman Test Case 1:

* Test Setup: Synthesize the microprocessor on board, and run the C program using NIOS II SBT for Eclipse. The first board’s output pin connects to the second board’s input pin. The second board’s output pin connects to the first board input pin.
* Inputs from questioner:
  + Then length of the word: 4
  + The characters in the word: “test”
  + The number of chances: 5
* Inputs from guesser:
  + “e”, “s”, “a”, “t” each time
* Outputs for questioner:
  + It prints “You lose” on the console
* Outputs for guesser:
  + For letter “e” “s” “t”, it displays “Hit!” and for “a”, it displays “wrong!”
  + The number of chances left decrements each time
  + It also displays the current state of vocabulary correct
  + It displays “You win” at the end
* Pass/fail criterion: Whether the results are the same as expected.

Hangman Test Case 2:

* Test Setup: Synthesize the microprocessor on board, and run the C program using NIOS II SBT for Eclipse. The first board’s output pin connects to the second board’s input pin. The second board’s output pin connects to the first board input pin.
* Inputs from questioner:
  + Then length of the word: 4
  + The characters in the word: “beer”
  + The number of chances: 5
* Inputs from guesser:
  + “a”, “c”, “d”, “f” “z” each time
* Outputs for questioner:
  + It prints "You win" on the console
* Outputs for guesser:
  + For each letter, it displays "wrong!"
  + The number of chances left decrement each time
  + It also displays the current state of the vocabulary correct (all "\_' in this case since the player does not guess any letter correct)
  + It displays "You lose" on the console
* Pass/fail criterion: Whether the results are the same as expected.

Fight the Landlord:

Though this game is tested as a whole, for simplicity of discussion, the test cases are separated into 5 parts below, including initialization, shuffle, auction, play and score computation.

Fight the Landlord Test for initialization:

 Test Setup: Synthesize the microprocessors on boards, and run the C program using NIOS II SBT for Eclipse. The first board’s output pin connects to the second board’s input pin. The second board’s output pin connects to the third board’s input pin. The third board’s output pin connects to the first board’s input pin.

 Inputs from players: After each player chooses their order number, they each input four different numbers. Each player firstly input “1,2,3,4” “5,6,7,8” “1,2,3,4” respectively, then input “10,20,30.40” “1,2,3,4” “4,3,2,1” respectively.

 Outputs to console: For different two sets of data, each player will get different cards

 Pass/fail criterion: Whether the results are the same as expected.

Fight the Landlord Test for shuffling:

 Test Setup: Synthesize the microprocessors on boards, and run the C program using NIOS II SBT for Eclipse. The first board’s output pin connects to the second board’s input pin. The second board’s output pin connects to the third board’s input pin. The third board’s output pin connects to the first board’s input pin.

 Inputs: Using the same data as initialization test

 Outputs: Each player's SRAM contents agree with each other, each player has 17 cards, all three players cards with the first 3 cards in SRAM adds up to a set of cards.

 Pass/fail criterion: Whether the results are the same as expected.

Fight the Landlord Test for auction (3-player version):

 Test Setup: Synthesize the microprocessors on boards, and run the C program using NIOS II SBT for Eclipse. The first board’s output pin connects to the second board’s input pin. The second board’s output pin connects to the third board’s input pin. The third board’s output pin connects to the first board’s input pin.

 Inputs: At the auction parts, each player inputs “3”,”3”,”3” respectively, then inputs “1”,”1”,”2” respectively.

 Outputs:

* + The score chosen by each player "3","3","3" and "1", "1", "2"
  + The player number of the landlord: #1 and #3
  + The three extra cards given to the landlord: "4" "10" "K" for both cases
  + The cards in each player's hand are legal and reasonable

 Pass/fail criterion: Whether the results are the same as expected.

Fight the Landlord Test for auction (2-player version):

 Test Setup: Synthesize the microprocessors on boards, and run the C program using NIOS II SBT for Eclipse. The first board’s output pin connects to the second board’s input pin. The second board’s output pin connects to the first board’s input pin.

 Inputs: At the auction parts, the first player inputs “1” and the second player inputs “1”. Then the first player inputs “2” and the second player inputs “3”

 Outputs: It will firstly displays “tie” then displays that the first player is the landlord

 Pass/fail criterion: Whether the results are the same as expected.

Fight the Landlord Test for playing (3-player version):

 Test Setup: Synthesize the microprocessors on boards, and run the C program using NIOS II SBT for Eclipse. The first board’s output pin connects to the second board’s input pin. The second board’s output pin connects to the third board’s input pin. The third board’s output pin connects to the first board’s input pin.

 Inputs of #1 player:

* + Number of cards the player wants to play: 2
  + The cards the player wants to play: "3", "3"

 Inputs of #2 player:

* + Number of cards the player wants to play: 2
  + The cards the player wants to play: "5", "5"

 Inputs of #3 player:

* + Number of cards the player wants to play: 3
  + The cards the player wants to play: "6", "6", "6"

 Outputs to #2 player:

* + Pattern code: 21 (According to pattern code, 21 means pair)
  + keyValue of cards: "3"
  + The cards played by this player is: legal

 Outputs to #3 player:

* + Pattern code: 21 (According to pattern code, 21 means pair)
  + keyValue of cards: "5"
  + The cards played by this player is: not legal

 Pass/fail criterion: Whether the results are the same as expected.

Fight the Landlord Test for score computation(3-player version):

 Test Setup: Synthesize the microprocessors on boards, and run the C program using NIOS II SBT for Eclipse. The first board’s output pin connects to the second board’s input pin. The second board’s output pin connects to the third board’s input pin. The third board’s output pin connects to the first board’s input pin.

 Inputs: For the first round, the base multiplier is “3”, and the landlord wins. For the second round, the base multiplier is “2”, and the farmers win. For the third round, the base multiplier is “2”, the landlord wins and there is a set of bomb used.

 Outputs: For the first round, the landlord gets 6 points and farmers get -3 points. For the second round, the landlord gets -4 points and the farmers get 2 points. For the third round, the landlord gets 8 points and the farmers get -4 points.

 Pass/fail criterion: Whether the results are the same as expected.

**PRESENTATION, DISCUSSION, AND ANALYSIS OF THE RESULTS**

Hangman:

Hangman game developed in right way, so microprocessor and network interfaces were implemented to allow two boards to communicate each other. To proceed communication system, NIOS II microprocessor integrated on Quartus II board. For Hangman game, two microprocessors need to connect together using the pins on the board as the first board's output pin to the second board input pin, and the second board output pin to the first board input pin. In this way, microprocessors receive and transmit the data. After done with the implementation of the communication system, players play Hangman game with their own board using eclipse program. And when players play Hangman game, the process and even result such as what did guesser guess so far, chance left, and the answer (only if guesser didn't make it), display on the eclipse program. So in this way, we could finalize that the communication system between two microprocessors work properly. Plays described below:

Either of players choose 0 or 1, player with 0 : executioner & player with 1 : guesser

1. Executioner chooses a word (let's say executioner typed christmas), and provides a number of chance to guess (6 chances)
2. Then on the screen, it displays dashes as long as length of a word (\_ \_ \_ \_ \_ \_ \_ \_ \_in this case), and number of chance that left (6 chances left)
3. Now, guesser guess an alphabet for the word, if guesser guessed right alphabet (c in this case) it will be displayed as c \_ \_ \_ \_ \_ \_ \_ \_, otherwise (u in this case), it will display chances that left (5 chances left).
4. So, if guesser got the word within chances left, guesser win (executioner lost), otherwise executioner win (guesser lost)! All the results and processes will be displayed on the eclipse program.

2-players Fight the Landlord:

2-players mode Fight the Landlord was expected to allow two players play the card game using their own board as they communicate each other. To proceed communication system, NIOS II microprocessor integrated on Quartus II board, and same as Hangman game, two microprocessors need to connect together using the pins on the board as the first board's output pin to the second board input pin, and the second board output pin to the first board input pin. In this way, microprocessors receive and transmit the data. After done with the implementation of the communication system, we setup the SRAM and wrote C language coding to implement Fight the Landlord game. After all of steps are done, players play Fight the Landlord. When they play Fight the Landlord, all the processes and results display on Eclipse program (which verify that the microprocessor communicate each other). Plays of Fight the Landlord (2-players version) described below:

1. Each of players chooses player number 0 or 1. This number should be unique. In this way, the system determine which player play first.
2. After determination of order, players do 'Rock, Paper Scissors' to choose the landlord, this 'Rock, Paper, Scissors' implemented with 3, 2, 1 as in order
3. After choosing the landlord, players play the card game by order with the cards that randomly shuffled
4. So, player 0 chooses number of cards, card number and combination (all combinations are written on game user guide) of cards that player wants to hand in. On player 0 and 1 programs, they display combination code and card number of player 0's, so player 1 can play against or pass, for example, if first player type 2, 3 and 3, this means first player like to hand in 2 cards of three (in this case, it's double) then next player has to hand in the card that is higher than 3 of double combination or bomb, otherwise second player can pass
5. So, the system lets player 0 and 1 to play back and forth until one of players has no cards left to hand in (the winner)

3-players Fight the Landlord:

3-players mode Fight the Landlord was expected to allow three players play the card game using their own board as they communicate each other. To proceed communication system, NIOS II microprocessor integrated on Quartus II board, three microprocessors need to connect together using the pins on the board, but unlike 2-players version Fight the Landlord, all three microprocessors don't connect altogether. For three-players version, the microprocessor connected to their right side, so first microprocessor connected to second, second to third, and third to the first. In this way, microprocessors receive and transmit the data. After done with the implementation of the communication system, we setup the SRAM and wrote C language coding to implement Fight the Landlord game. After all of steps are done, players play Fight the Landlord. When they play Fight the Landlord, all the processes and results display on Eclipse program (which verify that the microprocessor communicate each other). Plays of Fight the Landlord (3-players version) described below:

1. Each of players chooses player number 1, 2, or 3. This number should be unique. In this way, the system determine which player play first
2. After determination of order, players choose 1, 2, or 3 to determine the landlord and players' initial score. Biggest number chooser become a landlord. All players can type same number either 1 or 2 or 3, in this case, player which typed biggest at first become a landlord
3. After choosing the landlord, players play the card game by order with the cards that randomly shuffled
4. So, first player chooses number of cards, card number and combination (all combinations are written on game user guide) of cards that player wants to hand in. On all players' programs, they display combination code and card number of player 1's, so player 2 can play against or pass, after that it is player 3 turn. For example, if first player type 2, 3 and 3, this means first player like to hand in 2 cards of three (in this case, it's double) then next player has to hand in the card that is higher than 3 of double combination or bomb, otherwise second player can pass
5. So, the system lets player 1, 2 and 3 to play back and forth until one of players has no cards to hand in (the winner)

**ANALYSIS OF ANY ERRORS**

N/A

**ANALYSIS OF WHY THE PROJECT MAY NOT OF WORKED AND WHAT EFFORTS WERE MADE TO IDENTIFY THE ROOT CAUSE OF ANY PROBLEMS**

Communication:

For the 3-player version Fight the Landlord, we initially connected the boards in a wrong way. We connected the output pin of a board to other two boards' input pins. We later found out that one output pin can only connect to one input pin, otherwise it may lead to undeterminable situation. We then discussed and came up with a new way of connecting.

Assume three boards are named as "A", "B" and "C". The output pin of A connects to the input pin of B. The output pin of B connects to the input pin of C. The output pin of C connects to the input pin of A.

If A wants to talk to C, A needs to talk to B first and then B will forward the signal (message) to C. In order to accomplish the "forward" function, we set the seventh bit of data (counting from least significant to most significant) as "forward bit". If the forward bit is set, we will forward the signal (message) to the next board. The forwarding signal's seventh bit will be set to zero to avoid further forwarding.

Time constraints:

When testing send and receive for 3-board communication, we used the first board to send out continuous data with the most significant bit set to 1 (64 to 127 in decimal), so that the second board should forward the data received to the third board. However, we found that the received data by the second board could not receive all the data sent by the first board, but the third board can receive all the data received by the second board. We discussed and analyzed this error, and concluded that the error is caused by the forwarding data operation. That is, sending data after the normal receive operation leads to certain delay, and this delay may be long enough to effect the next receive since the data sent is continuous.

To solve this problem, used a certain amount of delay after every send and receive operation.

Wired reception pattern:

When we were doing "smoke test", and sending out continuous data (usually infinite continuous sending operation with data as 0-127), though 0 was just sent one time, but some times it was received several times. We solved this problem by using functions in library "altera\_avalon\_pio\_regs.h" rather than simply assign value to or read value from base addresses. Specifically, function IOWR\_ALTERA\_AVALON\_PIO\_DATA(base\_address, data) was used to output from the microprocessor, and IORD\_ALTERA\_AVALON\_PIO\_DATA(base\_address) is used to read the input data to the microprocessor. We do not know the specific reason why using the two functions solves the problem. Our guess this that the two functions may be able to deal with some potential errors caused by direct accessing to base addresses.

**SUMMARY AND CONCLUSION**

**Summary**

Overall lab focused on an asynchronous communication system among the boards (two and three boards in this lab) and the applications using the communication system. Firstly, we built an asynchronous communication system using Quartus II and Qsys tools. After setup the NIOS II microprocessor that is available to send and receive the data, we implemented the serial-to-parallel and parallel-to-serial network interfaces. This network interfaces were implemented with functional modules such as BSC, BIC, shift registers, signal control modules and clock control modules that were provided on the lab specification. In last, we integrated the hardware microprocessor and network systems on the Quartus II board as we setup one pin for receiving and the other one for transmitting. After done with developing asynchronous communication system, we advanced to the applications using a communication system. For these applications, we designed and developed the games called ‘Hangman,’ and ‘fight the Landlord.’ These games fundamentally used NIOS II microprocessor and network interfaces to let players play altogether.

**Conclusion**

The overall labs worked properly, NIOS II microprocessor from Lab 3 was implemented and an asynchronous communication system transmits and receives the right data from one to another as the serial-to-parallel and parallel-to-serial network interfaces setup. And the games were developed and played as Design Specification proposed using the communication system and microprocessors on Quartus II board. But, the lab 4 specification was little confusing, so it might be better if there are more explanations for each functional modules. As a result, after finishing lab 4 and 5, all group members now understand the fundamentals of the communication system, and how to implement the system using tools such as Qsys and Eclipse.