EE/CSE 371 Lab 5 Report

Lingli Zeng
Jichun Li
Zicheng He
Zewei Du

Abstract (5)

In this lab, we chose the existing track for our final project. The goal of this lab is to prototype a portable game specific console that runs some classic games like Space Invaders, Battleship or Pacman etc. The actual task of this project is developing a system that integrates the hardware and software and is capable of communicating with other system with the same setup. In our system, instead of a single classic game, we actually built three subgames: Reversi, Gomoku and Tic-Tac-Toe, so we named our project Board Game Collection. The hardware of the system contains two parts: a NIOS II based system and serial communication protocol written in SystemVerilog. The hardware system is built to tackle the serial communication between two systems and the communication between software and hardware. The software of the system contains includes several submodules: the user interface, the display and the game logic.

The whole system was built in a 'bottom to top' manner. That is, once we had a overall idea of what we want to build, we will start from building the smallest submodules, and when the submodules function as we expected, we will move on and connect the submodules together, and eventually complete the whole system.

The lab specification required us to provide seven deliverables, and in our case, these seven deliverables are: the Reversi subgame, the Gomoku subgame, the Tic-Tac-Toe subgame, the serial communication protocol, the user interface, graphical display, and the NIOS II system and a functional toplevel system. Due to the limit of time, the graphical display was not implemented using the LCD screen driven by Arduino microcontroller, we choose to display the chess board on the console. The rest of the deliverables we managed to achieve.

1. Introduction (5)

a. Purpose of this project

Our team chose the existing track of the final project, which is prototyping a portable game specific console. Throughout the quarter we have learned a lot on digital

circuits and systems. The purpose of the last project is to give us a chance to apply our knowledge and develop a system that integrates hardware and software. To be more specific,we applied the knowledge of SystemVerilog Programming, C programming, test and verification of digital systems, communication and building NIOS II systems during the process of designing and implementing the system.

Besides applying the knowledge of software and hardware we learned from the class, the project also helped us review the flow of design and implementation. For a relatively large project, dividing the work is always hard. With the experience we obtained from previous labs of this class, this issue no longer bothers us. Most of our projects were developed in 'bottom to top' way: which is implementing the modules with simplest logic and most basic functions first, and then test these submodules with testbenches. If the submodules works as we expected, we would assemble the submodules to build the modules with higher hierarchy level. But before the implementation, we still need a 'top to bottom' step, which is dividing functions and modules that will be used in the top level modules into submodules. The smaller and simpler a submodule is, the less time it will take to debug. When are dividing the top level to submodules, one thing to be noticed is that the function and logic are not the only thing that will be specified in this process, the data flow should also be considered in this step, so that when we attempt to assemble submodules, the ports will match.

b. Relevant background information

In this project we are building a game collection of three games. Here are the brief introduction of them:

- Reversi: Reversi is a strategy board game for two players, played on an 8×8 uncheckered board. There are sixty-four identical game pieces called disks (often spelled "discs"), which are light on one side and dark on the other. Players take turns placing disks on the board with their assigned color facing up. During a play, any disks of the opponent's color that are in a straight line and bounded by the disk just placed and another disk of the current player's color are turned over to the current player's color. The object of the game is to have the majority of disks turned to display your color when the last playable empty square is filled.
- Gomoku: Also called Gobang or Five in a Row, is an abstract strategy board game. It is traditionally played with Go pieces (black and white stones) on a Go board, using 15×15 of the 19×19 grid intersections.[1] Because pieces are not moved or removed from the board, Gomoku may also be played as a paper and pencil game. The game is known in several countries under different names. Players alternate turns placing a stone of their color on an empty intersection. The winner is the first player to form an unbroken chain of five stones horizontally, vertically, or diagonally.
- Tic-Tac-Toe: Tic-tac-toe (also known as noughts and crosses or Xs and Os) is a paper-and-pencil game for two players, X and O, who take turns marking the spaces in a 3×3 grid. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row wins the game.

2. Requirements Documentation (25 Total)

a. System Description (10)

i. Overall description of your hardware and software modules. What is the scenario you are developing?

For lab 4 and our final project, we implemented SystemVerilog modules, C program, and a simple nios processor to accomplish our goals. The deliverables for lab 4 is predetermined on the lab handout. For our final project, as specified in the proposal, we will design a classical two-player board game collections which includes tic tac toe(board size: 3*3), Gomoku(also known as five-in-a-row, board size:8*8), and Reversi(board size: 8*8). The game rules will follow the standards as for original games.

Starting with nios processor, we basically reuse the NIOS II microprocessor built in lab 3. We changed the on-chip memory sizes as we have a big c program comparing to lab 3, the original memory size was not big enough to hold what we implemented and caused overflow when ran the c program. Then we added in several PIOs to facilitilize the connection between hardware modules and software programs. The newly added PIOs includes busInput(8 bits), busOutput(8 bits), loadi(1 bit output), transEn(1 bit output), charSent(1 bit input), charRec(1 bit input). Those PIOs are matched to our systemVerilog modules to perform the communication task.

Second, we designed in total 12 systemVerilog modules including the top module, mainly focusing on the transmission/communication functionality. The detailed description of each module will include in the hardware section of this report.

Lastly the Eclipse C programs, it includes our mandatory lab 4 deliverables which are “Smoke Test” and “Hello 1234”. And also our final project programs: *Tic-Tac-Toe*, *Reversi*, *Gomoku* and a linker file. The detailed functionalities of our C program will include in the software section.

ii. What are the inputs and output to/from system? What interfaces with what?

For both lab 4 and lab 5, we mainly take the input from the Nios console, and a key on the FPGA board as our reset button. The input program needs is basically the data user wants to transmit. User can freely type in numbers or letters one at a time to transmit, and the led lights will light up as corresponding ascii binary values to what user typed.

Our transmission system is exactly based on the block diagram provided in the lab handout and we separate it into transmit and receive two parts. User can input datas in the console, and by c pointer arithmetics and with PIOs base address, we implemented c program to write user input into the 8 bits parallel data bus. Then hardware module can decided when to transmit the data and signaling the char sent & transmit enable signals. Next data will go through parallel to serial buffer and finally transfer out of the system to another serial data in port. Above largely describes the transmit process, after this, as the data transferred to serial data in port, our receive function is activated.

The data will first processed by start_detect, so the start bit would be transferred to be the enable_i signal, which would be sent to start_detect module simultaneously with the 8 data bits. After processing/sampling the data, we can transfer the data through 8 bit parallel bus into nios processor. Then using the c program command, we can read the data by pointer arithmetic from the address of businput. This basically describes our communication/transmission function and how it interact with nios processor and software.

iii. What are some design constraints?

Except building up a complete communication system between FPGA boards, we want to produce a classical board game collections including *Tic-Tac-Toe*, *Reversi* and *Gomoku* as described in the proposal. In other words, the design constraints for software basically are the rules of each games.

Hardware part we need to detect the valid data(receive the eligible data), sample/process the coming in data, read or write at proper sample time, and transfer data out when eligible.

Software part is separate into three games. As the *Tic-Tac-Toe* and *Gomoku(five-in-a-row)* has similar rules differing in game board size, these two game should have same logic and small modifications. Reversi will be a separate program. Two player will play on a same board, and their chess will be distinguished by “O” and “X”. Player will place one chess piece at a time, other player cannot place the chess until the chess location that placed by other is received. Player cannot place the chess piece at pivot that is already occupied by other player. If player enters a non-valid pivot, system will automatically ask player to re-enter another pivot. For Reversi only, player can only enter the legal move pivot which provides by the system through console. Once one of the player reaches the goal, the game will stop and print out the winner.

b. Hardware Description (10)

i. What is your hardware design supposed to do?

The verilog of this project is mainly used for the communication between two boards. The system can process a parallel input signal to a serial signal that follows RS-232 standard. And it can also receive a serial signal that follows RS-232 standard and store it in a parallel output. A RS-232 serial bus is one kind of serial data transfer protocol that can be used on UART devices. The format of the bus is a start bit of low in front, and 8 data bits follows it. When the bus is idle it stays at high. So everytime data is send to a RS-232 bus the value will first change to low, when a receiver detects the start bit of low, it will start reading in data.

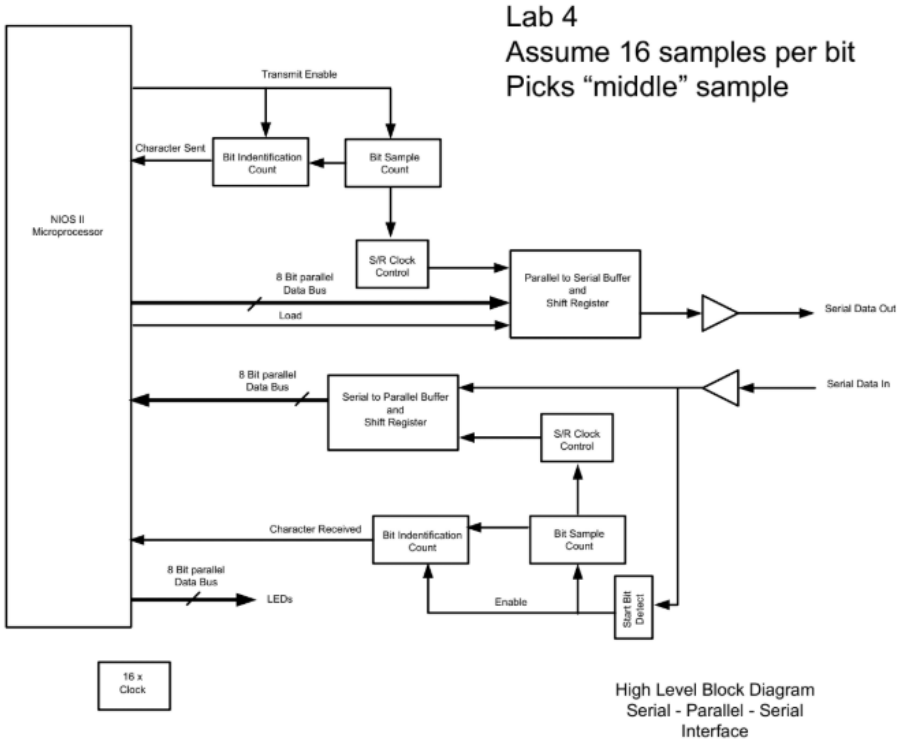
When the user operate with the system, they want to connect an 8 bit parallel input as well as an 8 bit parallel output to it, these are the data storages so the connecting port can be anything that can store data. Serial data in/out should be

two 1 bit pin, they behave as buses so they do not store data, so the port should be a wire that connects one SDI to a SDO. The system will also take in two other inputs: transmit enable and load. Transmit enable port enables the counters so their clocks moves, load port will trigger the PISO buffer to load the current parallel input into SDO. Both port should be operated as switches. For our design, the transmit enable signal must be triggered first so that the counters can create the SR clock that PISO buffer runs on. And then the load signal can be triggered. The input data must be present before load signal so that the system will not read wrong data in. There are also two other outputs to this system, char_sent and char_received. They turn to high for one cycle when a full process of transmit/receive is over. These output should be used as indicators for the operator to know that the system is not ready for another transmission until the signal turns high once.

In this project, the basic function of this Verilog system is that when it is transmitting it will take data and instruction from the NIOS processor, and transfer it to a GPIO port as the serial bus. For its' receiver functionality, it takes input from a GPIO port and write the output to NIOS.

ii. What does each of your Verilog modules do?

- **DE1_SoC**: This is the top module of our design, it connects the NIOS processor to the Verilog design, by connecting the ports of the NIOS processor, the modules can read and write data to NIOS memory for C program to operate with
- **serial_parallel_serial_interface**: The interface for the whole transmission system. It implements the system same as the block diagram we have in class:



High Level Block Diagram
Serial - Parallel - Serial Interface

- **clock_divider**: This clock divider uses a counter to divide the input clock to a 16 time slower clock and return it as the output.
- **clk16**: This divider divides the 50MHz original clock of FPGA board into a 153200Hz clock, which is 16 time faster than 9600Hz. After passing through the clock_divider, we will have two clocks of 9600Hz and 153200Hz. The reason we need to do this is because C programs must work with a 9600Hz clock. The 153200Hz is our sampling frequency for the sample counter.
- **transmit/receive**: these are the interfaces for the transmission part and the receiver part, the structure are not exactly the same so we can use basic modules repeatedly but we need different interfaces. The transmit interface takes in an enable and a load signal, it also output a char_sent signal when the process is complete. Receive interface use a start detect module to enable the design so it does not need any enable signals, it will have a output char_recv signal when the process is complete.
- **bit_identification_counter**: This counter operates on 9600Hz clock, it starts working when the enable is triggered, it will wait for a initiate signal and start counting for 16 cycles. For the transmission the initiate signal is load, for the receiver the initiate signal is the same as enable signal, which is the output for start detector. 16 cycles indicates the time interval that the system is transmitting/receiving data. After it finishes counting it will send a output signal indicates the system is idle now (char_sent and char_recv).
- **bit_sample_counter**: This counter operates on 153200Hz clock, it starts working when the enable is triggered. It simply counts on 153200Hz and send an 4 bit output of the current count.
- **sr_clock_control**: This module receives the 4 bit count from the sampling counter. It returns the first bit of the counter as the output sr clock. This bit starts at 0 and turn 1 on the 8th count of the sampling clock. So if there are no any delay caused by hardware fault, the output should be a clock of 9600Hz that had the exact reverted value of the original 9600Hz clock.
- **start_detect**: This module is used on the receiver side, it takes in the SDI and return an enable signal. When SDI turns zero, which indicates the following 8 bits are data, its output turns to high for 16 cycles so that the receiver will start working and take in serial data.
- **io_buffer**: This is a buffer that is used on both SDI and SDO, this buffer delays the data bus for 1 cycle to avoid meta-stability problems.
- **piso**: First of all this module works on the sr clock. This module operates the action of reading 8 bits parallel data into a 9 cycle output serial data (which includes the start bit). Like the method we learnt in 271, it use 9 flip-flops, or to say one 9-bit flip-flop, as a data buffer, and push out one bit at a time. The first bit is the padding of the start bit which is 0. It needs a load input signal to know that it can load in the parallel data and start sending. When all the data bits are pushed out to the output serial port, output goes back to idle and output 1. In the actual implementation, I used a 10 bit data buffer, the structure is 0-data bits-1, the data buffer shifts left left every cycle and add a 0 to the LSB, so when the 9

bits we needs go out, the buffer will always be 10'b1000000000, and this formation cannot occur at any time before the 9th cycle. The 10th bit here is working as a parity bit indicating the end of the transfer.

- **sipo**: First of all this module works on the sr clock. This module operates the action of reading a serial data bus and write the bits into a 8 bit parallel data storage. It has a 8 bit flip-flop as a data buffer. It needs a enable signal to know that it can start reading in serial data. Then it will record the data on the serial data bus for 8 cycles, ignoring the start bit padding. After 8 cycles the data buffer is filled, and it will load the 8 bit data into parallel output, the module goes back to idle.

c. Software Description (5)

i. What is your piece of software supposed to do?

The software for lab 4 mainly implements the basic communication system used in lab 5. The lab implements two basic functions, namely smakeTest and Hello 1234. The Two functions all send a character to the output port and read the data again from the input port.

The software part of lab 5 mainly implements an user interface, game logics, and hardware to software connection. First of all, at the beginning of the game, the software should print some introductions to the game collection and prompt the user to input necessary informations to play. More specifically, the software first introduces that the system is consisted of three games, namely, tic-tac-toe, gomoku, and reversi. The system prompt the user to input which game he or she wants to play using the code 01 for tic-tac-toe, 02 for gomoku, and 03 for reversi. Since the games designed for multiplayer, the software should also ask each user to input which player s/he defines his or her computer as. Since NIOS II has limited memory resulting in limited library, the user input has to be a character. Since it is more convenient in our game logic to treat user input as integers, we designed a special function in our software to translate the user input from characters to integer.

The next and most important part of the system's implementation is the game logic. As we mentioned before, there are three games in this systems. The logic for tic-tac-toe and gomoku should be roughly the same, except that they have different game capacities. Tic-tac-toe is a two-player board game played on a 3X3 grid. The first player's pieces are marked by the letter 'X' and the second player is marked by the letter 'O'. During the game, the first player who managed to place three of his or her piece in a vertical, horizontal, or diagonal line wins the game. The game gomoku is played in a similar fashion except that the playground becomes 8X8 and the first player who places five pieces in a line wins. There is scenarios in which nobody wins the game while the board is filled. In such a scenario, the game ends up with a tie between two players. The software that implements these two games should first prints the situation of the board, and ask the players the location that they want to place their piece at one

at a time. In our design, we choose to let player 1 choose first, and the alternate players in each round. After each player chooses in their turn, the software should first print out the most up-to-date piece layout and then check if there is a win by scanning all the lines on the board. For tic-tac-toe, if the program detects a line of three ‘X’s or three ‘O’s, the game is terminated, and the winner’s name will be shown. In the game Gomoku, the algorithm is the same, except that the winning magic number becomes 5.

The third game in this game collection is Reversi. Reversi has a completely different set of rule, so it has to be implemented separately. Reversi is a strategy board game for two players, played on an 8×8 uncheckered board. Players take turns placing disks on the board with their assigned color facing up. During a play, any disks of the opponent's color that are in a straight line and bounded by the disk just placed and another disk of the current player's color are turned over to the current player's color. Unlike tic-tac-toe and gomoku, not every empty spot on the board is a valid move for a certain player. A player’s move has to flip one or more of his or her opponent’s disk in order to be considered valid. The end of this game is marked by either side of the player having no valid move to take. When the game is over, the winner is the one that has the most disks marked on the board. The code that implements this game should first initialize a board that marks two of each player’s disk in the middle of the board and begin the game with player one’s turn. The software should should first detect all the valid turns for each player and print all the valid locations on the console. Once a player chooses a location, the program should check which disk to flip and print the most up-to-date board layout on the console. After each turn is over, the program should print out the current score (i.e. the amount of disks of each player) on the console, and count the valid moves of each player. Once either player’s valid move becomes 0, the game should terminate and the player with higher score becomes the winner.

The last important part of the software system is the communication between two players. The communication system is divided into two scenarios, namely receiving and transmitting. In the beginning of this game, each computer defines itself as either player 1 or player 2. The current player, on the other hand, switches between 1 and 2 in each turn. When the current player is equal to the computer defined player, the program takes in the player’s input location and write the data to the output port in order to transmit the data to the other player. On the contrary, if the current player is not equal to computer defined player in this turn, the program read the data form the input port and update its own board as well in the receiving turn. While the software defines when to transmit and receive, the main communication system is implemented on the hardware by verilog. Since there are some protection variables in the hardware implementation, the software has to address those variables in the appropriate

manner while communicating. The details of the protection variable manipulations are discussed in the later sections.

ii. What do your C code functions do?

- SmokeTest:
  This function smokeTest should simply transmit one character out and then read that same character again and display it.

- Hello1234
  Hello1234 should initiate the exchange by sending the character sequence Hello 1234 one character at a time. Each data byte to be transmitted must converted into an ASCII character. The receiver should accept that then display each incoming character to the eclipse console.

- Main
  The main function is the core of the whole system. It not only initializes the game, but also control the flow of the whole system. More specifically, it should first introduces the game and also prompt the user to input the game choice and player definition of the users' computer. From the user input, the main function should initialize the appropriate game capacity, game flow, and win numbers. During the game play, the main function also controls when to display the playboard again on the screen, when to update the current game scenario, and detect if the game is over by checking if the board is full or there is already a winner. After the game is over, the main function should also print out the winner. If nobody wins in the game, the main function should also report the game situation to the user accordingly.

- ticInit
  ticInit is a function that initialize the whole game setup. Since we define the game starts with player one, the ticInit function should start the game by setting the current player to one and defining the current mark as 'X'. What's more, when we design the game, we decide to mark each grid on the board with increasing numbers from left to right and up to down in order to let the player distinguish each grid. Therefore, in the ticInit function, we filled each grid with a number marking.

- drawTicBoard
  Since we decide to display our chess board on the console, we wrote this drawTicBoard function to print the most up-to-date board on the console. More specifically, if a certain grid is already marked by player one or player two, we print the content of the grid as character 'X' or 'O'. Otherwise, if a grid is unmarked, we just print the number that the grid is assigned when we first initialize the game.

- changeTicPlayer
  changeTicPlayer is a simple function that handles the player switching during the game. For the game convenience, we also kept the current player mark (‘X’ or ‘O’) as a global variable. Therefore, in addition to changing the current player number, the changeTicPlayer function should also update the player mark as well.

- ticUpdate
  ticUpdate is one of the core functions in this system that controls the main events in the game. It is called every time a player makes a move on the board. First of all, ticUpdate should check if it is the computer’s turn to make a move. If it is, then the function call the send() function and store the user input as the next move. If it is the computer’s opponent’s turn to play, the ticUpdate function calls the receive() function and record the other player’s move as the next step. Next, the ticUpdate function constantly checks if the next move is valid. A move is valid if the number is inside the game capacity and no player has ever marked that certain position before. If the next move passes the test, the ticUpdate function will then mark the next move position. Last but not the least, if the player chooses to player Reversi, the ticUpdate function will also call the flipBoard function to flip any piece that is surrounded by the opponent’s pieces.

- ticCheckOver
  ticCheckOver is the special function that check if the game is over only for tic-tac-toe and gomoku. The function simply check if all the grid on the game board has been marked. If all the content on the board are either ‘X’ or ‘O’, the ticCheckOver function returns 1. Otherwise, it returns 0.

- ticCheckWin
  The ticCheckWin function checks if there is winner for game tic-tac-toe and gomoku. According to the game rule of tic-tac-toe and gomoku, the winner is the first player who puts a certain amount of pieces in a straight line. Therefore, the ticCheckWin function checks a straight line of one mard horizontally, vertically, and diagonally. The magic winning number for tic-tac-toe is 3, and the one for gomoku is 5. If there is a winner detected, the function should return 1. Otherwise, it returns 0.

- reverInit
  reverInit is a special function for the game reversi. Since the game reversi starts with two of each player’s pieces in the middle of the board, reverInit function should first call the regular ticInit function and put the four pieces in the middle of the board.

- calculateValidMoves
  This function is one of the key functions lays in the game logic reversi. It is top function for validMove. This function loops through the entire game board and checks if there contains the mark of current player's opponent on the board in the intended direction

by calling the validMove function on all dirrections. If there is one direction that satisfy the requirement, the current grid that the function is looking is a validMove. If the computer is the current player, this function will print out the current location as a valid move. Since calculateValidMoves looks through all the positions on the board, all the possible valid moves will be printed within this function

- validMove
  As mentioned in the previous function, validMove is a function in Reversi's game logic that checks the current position contains the mark of current player's opponent on the board in the intended direction. It should first eliminate all the out-of-bound scenarios and then call the function checkLineMatch to see if there the current position is a valid move.

- checkLineMatch
  chekcLineMatch is a recursive function in the reversi function that checks if there is a current player's mark at the current position or anywhere further in the specified direction. The recursion will terminate once a current player's piece is found or when the check exceed the boundary of the game board.

- flipBoard & flipLine
  The functions flipBoard and flipLine are the two functions that perform the flip in the reversi game. Similar to validMove and flipLine, the function flipLine is the recursive method that perform the flip and the flipBoard method looks at one grid at a time and calls flipLine on all the directions around the current grid. The function flipBoard is further called in the ticUpdate function, so that the program looks at all the grid on the board to check potential flips.

- checkReversiOver
  checkReversiOver function is the function that detect if the reversi game is over by check if there is any valid moves for both players. At the same time, the function loops through all the grid in the gameboard to keep track of how many point each player has in the current state. If the game is not over yet, the function also prints out the current score of the two players.

- Stupid
  The purpose of this function is to accommodate the fact that NIOS II software development does not support scanf. Therefore, we implemented this function to eventually translate consecutive character inputs to integers. However, since we have to later transmit the resulting user inputs through a 8-bit bus, we cast our final return type to character.

- receive

receive is the software interface of the communication system. When the character is ready to be received, this function dereference a pointer pointing to the receiving port and return the value read.

- send
  send is an other software interface for the communication system. However, different from the receive function, the send function first prompt the user to input the intended move, and then write to the transmission port. After the transmission is over, the send function close down the sending procedure and return the user input as a character.

3. Design Documentation (35 Total)

a. System Description (10)

The biggest issue, we worries about the most for both lab 4 and final project, is how to get communication system to work properly including communication between two boards and interface between hardware and software. And when implementing the system, the clock is also tricky to use as we require two different paced clock. The faster one is used to sampling the data, whereas the slower one is used to take in the proper data. Although we can simply use clock divider and assign different dividing factor, which gives different clock speed, it is hard to debug using either signal tap or modelsim as components were running on different speed.

The c programs were initially written as “pure” software programs. In other words, we only implemented game logics in the program without considering other component such as hardwares. And it was tested using linux terminal.

Nios processor is implemented as predetermined to suit the needs of storing data and other instructional signals.

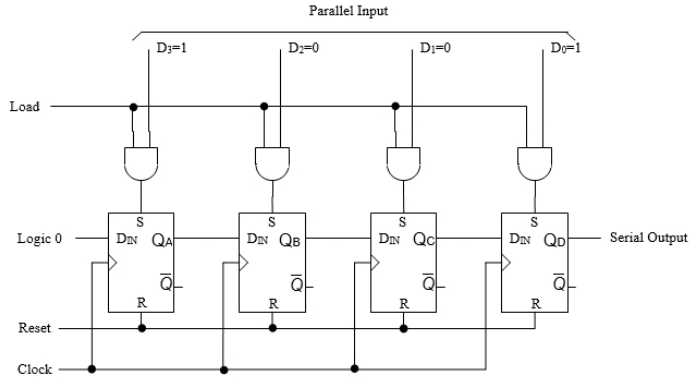
Above three parts of the system is tested separately to ensure the proper function solely. We combined it and modified the c program to use pointer and take the informations it needs from specific addresses(provided in Nios microprocessor).

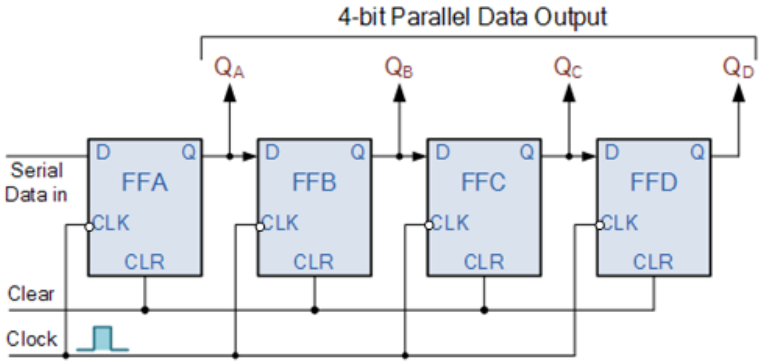
b. Hardware Description (15)

i. How did you design your digital circuit?

The Design of the Verilog transmission system is basically taken from the lecture. But we added our understanding and modifications to it.

The system can work with both sending and receiving signal. The parallel input/output are both 8-bit parallel data storage pins. The serial input/output are both 1-bit serial data bus line pins, both of them follows the RS-232 serial data bus protocol. It takes in two different enable signals and output two indicator signals. What we want from the Verilog system is that it can transfer between two kinds of signals, because we don’t want 8 buses to transfer the data, we want an 1-bit data bus to take care of all the communication. The basic form of this kind of transmission is to use flip-flops as data buffers, and connect them together so that it can behave one bit of data transmission on every cycle. This form of data structure is called registers, so in this system we will be creating a parallel-in-serial-out register and a serial-in-parallel-out register.





The first picture above is a sample for a a parallel-in-serial-out register and the second is a sample for a serial-in-parallel-out register. These structures are already pretty similar to the block diagram we have in class. But we still need to add other helper modules to make it a complete transmission system. First to implement a RS-232 type data bus we need a start bit of 0 and the serial data stays at 1 when it is idle. So firstly the serial output of transmitter must be constantly zero, then the register for transmitter must also add one flip-flop for the start bit. We do not need another register for the receiver because it can just discard the start bit. Then for reading data from the bus, the receiver must have some means to detect the start bit, so it knows when to start counting, then we will add a start bit detector to the receiver. To avoid meta-stability problems which would occur when you are reading data near a changing edge, we will use an asynchronous sampling counter to create another clock for the system that has it's rising edge at the middle of a data bit. We do not want the system to be constantly running, so we will add an identification counter to enable the other modules, so when the time duration of the identification ends, the transmission system stops running, this will also ensure that exactly one sending action or one reading action occurs in each transmitting/receiving process.

ii. Why does your Verilog work?

In this project, this Verilog system is connected to a C gaming logic system, as the tool to send data from one board to another board. Parallel data in/out, transmit enable and load signals will be send by C via NIOS. And serial data in/out will be bounded to GPIO[28] and GPIO[29], to connect the data bus, inputs

on each board will connect to outputs on each board using jump wires. For the C programs, they can write to addresses on NIOS to change the input ports for the Verilog system. To follow the requirements of the Verilog system, it will first change the enable system to 1, write the data and then load. A basic code template will be like this:

```
//transmission
*(paraIn) = alt_getchar();; //write input data
*(transEn) = 1; //enable
usleep(1000); //wait to avoid unstable edges
*(load) = 1; //load
while(!charSent){} //wait until charSent is true
usleep(2000);
*(load) = 0; //disable load
*(transEn) = 0; //disable enable
//receiving
while(!charReceived){} //wait until charReceived is true
char output = *(paraOut); //read output data
*(LED) = *(paraOut); // show on LED, debugging use
```

A 9600Hz clock has a period of 104ms, 16 cycles is 1666ms. When the enable signal is not 1, the piso register will not read from parallel data in. So if the enable and load signal are triggered at the same time, the data that is put into the register will not be the correct data we want. So we have to wait for at least one cycle after enable for the register to load in parallel data, then we turn on load signal to send it. After this action, we want to wait until the transmission process is over to do another transmission. So we have to at least wait for 1666ms which is 16 cycle. The method in the template is wait for charSent to be true, which is also correct, but after that I still wait for another 2000ms to make sure.
This template of reading/writing to the board avoids any meta-stability problems and rarely cause any fault when we were testing, so we will be using it as the communication standard for the Verilog system.

iii. What is the theory behind your Verilog modules?

- Counters: What both receiver and transmitter have in common is a counter that counts for the operation time which accounts for the time duration of the system to actively work, and another counter that counts for sampling which will be using the faster clock. The sampling counter will choose the middle of every bit on the data bus for the system to sample the result, so that it can avoid meta-stability problems. The output of the identification counter indicates the end of the process, when it turns 1, it is safe to do another transmission without causing fault.
- Clock Control: The sr clock control module is basically a expansion of sample counter, it only choose the first bit of the counter buffer as the output. Because the counter counts from 0000 to 1111, the first bit of the counter will be 0 for 8 sampling cycle and be 1 for 8 sampling cycle. This is the reason that the sr clock output by this module is half a cycle delayed compared to the original slow clock.

- SIPO: This system is basically implementing a SIPO register. When the register is fully filled with data, discarding the start bit, it will read the data into the parallel output. As the serial data bus progresses it read in the data each cycle. This module has a counter in it, the counter counts up to 16. When the counter reaches 9, which indicates that all the 8 data bits are now in the register, the module will push the data in the register out to parallel output. When the counter reaches 15, the module will know the loading process is complete and it will go to idle state, waiting for another cycle.
- PISO: This system is basically implementing a PISO register. This register will read from the parallel data input, and push out one at a time. We also need to add a start bit to the serial data to follow the RS-232 protocol. So we want one more digit for the register. Then the implementation is pretty simple, once load is triggered, it will load “start bit+data bits” into serial bus one at a time.
- Detector: The receiver system contains a start bit detector, it constantly reads from the serial data in bus, because of the RS-232 protocol, it stays at 1 when it is idle. For every valid data it definitely will have a start padding bit of zero followed by the actual 8-bit data. So when the detector get a zero for the first time, it knows it is a new data bit to be read, and it will send enable to the other modules.

c. Software Description (10)

i. How did you design your C code?

In lab 4, when we implement the software system, in addition to sending data to the output pio and read it back, we also created some variables that corporate with the hardware implementation. In this way, we make sure that the data is transmitted and received after the hardware communication system is ready.

In lab 5, the c code was initially designed as pure software implemented with code block. The player automatically switches between player one and player two. Without the hardware communication system, one computer has to play as two players, and the game logic was also tested with a single computer. With that being explained, the first few functions we implements are main, ticInit, drawTicBoard, changeTicPlayer, ticUpdate, ticCheckOver, ticCheckWin, reverInit, calculateValidMoves, validMove, checkLineMatch, flipBoard, flipLine, and checkReversiOver. We also used a header file linker.h to connect all the functions and variables. The implementation details are described below.
In the main function, we first introduced the game and also prompt the user to input the game choice and player definition of the users’ computer. From the user input, the main function initializes the appropriate game capacity, game flow, and win numbers. If the user enters an invalid input, the main function will immediately notice the user that the input is invalid and immediately exit the game. At this point, the user has to press the reset key on the FPGA board to reset the game. We made this design decision to protect the system from

entering some random state. Since at this point, it is still early in the game progress, we consider direct exit as our best choice. During the game play, we chose to use a while loop to create a continuous game flow. The exit condition is that a winner is a winner is either a winner is detected, or the game currently playing is over. After the while loop is over, since the game report for tic-tac-toe and gomoku is not integrated into the game logic, we choose to used a if-else statement to print out the game report.

ticInit is a function that initialize the whole game setup. More specifically, we decide to initialize three things here: current player, current player's mark, and the game board. In order to design a game system in which the user can easily address each location on the game board, we decided to mark the game board with different numbers. Since we decide to let the game startswith player one, the ticInit function first sets the current player to one and defining the current mark as 'X'. In order to mark the game board with appropriate numbers, we chose to use a counter and a nested for loop to efficiently complete the task.

Since we decide to display our chess board on the console, we wrote this drawTicBoard function to print the most up-to-date board on the console. More specifically, if a certain grid is already marked by player one or player two, we print the content of the grid as character 'X' or 'O'. Otherwise, if a grid is unmarked, we just print the number that the grid is assigned when we first initialize the game. During the design and development, we found out that some grid is marked with two characters (for example 13) while others are only marked with one. In order to perfectly align the game board, we carefully printed an extra space after each grid with only one character.

In order to automatically switch players during the game, we designed the function changeTicPlayer to handle the player switching. Since we need to check the player marks at many different locations in the code, we decided to set the current player mark as a global variable. Therefore, in addition to changing the current player number, the changeTicPlayer function should also update the player mark as well.

After each player take a move, the game logic has to update the current game status. Since the game logic of tic-tac-toe and gomoku is completely different from that of reversi, we chose to write the game update logic of both tic-tac-toe and gomoku in the same function and then have a separate function for reversi. The function ticUpdate is the update logic for gomoku and tic-tac-toe. First of all, ticUpdate checks if it is the computer's turn to make a move. If it is, then the function call the send() function and store the user input as the next move. If it is the computer's opponent's turn to play, the ticUpdate function calls the receive() function and record the other player's move as the next step.Throughout the progress, the function constantly check if the move is valid. Since at this point,

the user could have been playing this game for a long time, we don't want to directly exit the game every time a user takes an invalid move. Instead, our function will keep asking the client for another input until the move is considered valid. A move is valid if the number is inside the game capacity and no player has ever marked that certain position before. If the next move passes the test, the ticUpdate function will then mark the next move position. Last but not the least, if the player chooses to player Reversi, the ticUpdate function will also call the flipBoard function to flip any piece that is surrounded by the opponent's pieces.

During each round of the game tic-tac-toe and gomoku, we have to check the game status to decide if we continue running the game. ticCheckOver is the special function that check if the game is over only for tic-tac-toe and gomoku. The function uses a nested for loop to check if all the grid on the game board has been marked. If all the content on the board are either 'X' or 'O', the ticCheckOver function returns 1. Otherwise, it returns 0.

Besides checking whether the game is over, we also have to detect if there is already a winner. The ticCheckWin function checks if there is winner for game tic-tac-toe and gomoku. According to the game rule of tic-tac-toe and gomoku, the winner is the first player who puts a certain amount of pieces in a straight line. Therefore, the ticCheckWin function checks a straight line of one mark horizontally, vertically, and diagonally. In order to most efficiently implement the winner checking method, we choose to use for loops so that there is no redundant code. However, for the diagonal case, since checking from top left to bottom right and from top right to bottom left are completely different, we choose to use two sets of nested for loops to accomplish that. In this way, the code is also more redable and easier to understand.

After we finished all the design and implementation for gomoku and tic-tac-tow, we began to tackle the game reversi. From the game rule, we know that reversi is a game that needs a combination of breadth first search and depth first search. In otder to quickly implement the game, we also consulted some past implementations of reversi in javascript, and then integrated the algorithm into our system. The implementation of this game starts with the initialization in function reverInit. reverInit is a special function for the game reversi. Since the game reversi starts with two of each player's pieces in the middle of the board, reverInit function should first call the regular ticInit function and put the four pieces in the middle of the board.

After we successfully initialized the game reversi, we move on to calculate all the valid moves in each step. The algorithm is implemented in the function calculateValidMoves. Inside the calculateValidMoves function, we use nested for loops to iterate through the entire game board and checks if there contains the mark of current player's opponent on the board in the intended direction by

calling the validMove function on all dirrections. If there is one direction that satisfy the requirement, the current grid that the function is looking is a validMove. If the computer is the current player, this function will print out the current location as a valid move. Since calculateValidMoves looks through all the positions on the board, all the possible valid moves will be printed within this function

The function validMove is a function in Reversi's game logic that checks the current position contains the mark of current player's opponent on the board in the intended direction. In order to prevent the program from accessing data out of bound, we first eliminate all the out-of-bound scenarios and then call the function checkLineMatch to see if there the current position is a valid move.

chekcLineMatch is a recursive function in the reversi function that checks if there is a current player's mark at the current position or anywhere further in the specified direction. The recursion will terminate once a current player's piece is found or when the check exceed the boundary of the game board.

After checking the valid moves, we also has to flip the disks who are surrounded by the opponents' disks in a straight line. The functions flipBoard and flipLine are the two functions that perform the flip in the reversi game. Similar to validMove and flipLine, the function flipLine is the recursive method that perform the flip and the flipBoard method looks at one grid at a time and calls flipLine on all the directions around the current grid. The function flipBoard is further called in the ticUpdate function, so that the program looks at all the grid on the board to check potential flips.

The last step to implement the reversi game is to check whether the game is already ended. The reversi game ends when either side of the player has no legal move. checkReversiOver function is the function that detect if the reversi game is over by check if there is any valid moves for both players. At the same time, the function loops through all the grid in the gameboard to keep track of how many point each player has in the current state. If the game is not over yet, the function also prints out the current score of the two players.

After we implement and tested all the algorithms described above, the next step we took is to connect our software to NIOS II. However, the first thing we noticed is that NIOS II software development does not support scanf. Therefore, we implemented this function to eventually translate consecutive character inputs to integers. However, since we have to later transmit the resulting user inputs through a 8-bit bus, we cast our final return type to character.

Once we successfully connected our software to nios II, we moved on to communication. In the hardware implementation, once the variable charRec

becomes 1, the data is ready to receive. In order to accommodate this hardware feature, we dereferenced the pointer pointing to the charRec port. Once the value becomes 1, this function dereferences a pointer pointing to the receiving port and return the value read. The function send is an other software interface for the communication system. However, different from the receive function, the send function first prompt the user to input the intended move, and then write to the transmission port. After the transmission is over, the send function close down the sending procedure and return the user input as a character.

The theory behind the software system is that the how programmable NIOS II processor is connected to the software and software. With the software game logic, each player can has his or her own game running on the individual computer. With the connection to the hardware communication interface, two players can communicate their steps making this multiplayer game possible.

4. Presentation of Results (10)

For lab 4 c program part, we developed HELLO1234 together with the smoke Test, as we repeatedly read from user if there is a valid input. We connected our own output to input for test purpose as we don't have another FPGA board available at the moment. "Data is xxx" will print and the led light will lights up when character is received from the serial input, which proofs that our communication system is valid.

```
Receive or Send? Please enter the initial letter of your choice

s
please write

f
data is f
please write

g
data is g
please write
```

For the final project c program part, the display will be on the console. Below is the snips for game 1 which is tic-tac-toe. Player 1 will have the priority to enter first. And while player 2 is playing, player 1 will be waiting. The grid players choose will be

represent by different characters.

```
which player are you? (1 or 2)
0

1
me is 1
which game do you want to play? 1:tic-tac-toe 2:Gomoku 3:Reversi

0

1
Player 1 (X)  -  Player 2 (O)


 _____ _____ _____
|  0  |  1  |  2  |
|_____|_____|_____|
|  3  |  4  |  5  |
|_____|_____|_____|
|  6  |  7  |  8  |
|_____|_____|_____|
player 1 playing now
Which grid do you want to choose?
Now please enter the grid

0

5
Player 1 (X)  -  Player 2 (O)


 _____ _____ _____
|  0  |  1  |  2  |
|_____|_____|_____|
|  3  |  4  |  X  |
|_____|_____|_____|
|  6  |  7  |  8  |
|_____|_____|_____|
player 2 playing now
Player 1 (X)  -  Player 2 (O)


 _____ _____ _____
|  0  |  1  |  2  |
|_____|_____|_____|
|  3  |  O  |  X  |
|_____|_____|_____|
|  6  |  7  |  8  |
|_____|_____|_____|
player 1 playing now
Which grid do you want to choose?
Now please enter the grid
```

Then next snip shows once the player wins, the game will be terminated.

```
 _____ _____ _____
|  0  |  1  |  X  |
|_____|_____|_____|
|  O  |  O  |  X  |
|_____|_____|_____|
|  6  |  7  |  8  |
|_____|_____|_____|
player 1 playing now
Which grid do you want to choose?
Now please enter the grid

0

8
Player 1 (X)  -  Player 2 (O)



 _____ _____ _____
|  0  |  1  |  X  |
|_____|_____|_____|
|  O  |  O  |  X  |
|_____|_____|_____|
|  6  |  7  |  X  |
|_____|_____|_____|
Congratulations! player1 win!
```

Next, the below snip shows the game 2 which is Gomoku(five-in-a-row). We now have a 8*8 board size and as usual, player 1 will play first.

```
which player are you? (1 or 2)
0

1
me is 1
which game do you want to play? 1:tic-tac-toe 2:Gomoku 3:Reversi

0

2
Player 1 (X)  -  Player 2 (O)


 _____ _____ _____ _____ _____ _____ _____ _____
|  0  |  1  |  2  |  3  |  4  |  5  |  6  |  7  |
|_____|_____|_____|_____|_____|_____|_____|_____|
|  8  |  9  |  10 |  11 |  12 |  13 |  14 |  15 |
|_____|_____|_____|_____|_____|_____|_____|_____|
|  16 |  17 |  18 |  19 |  20 |  21 |  22 |  23 |
|_____|_____|_____|_____|_____|_____|_____|_____|
|  24 |  25 |  26 |  27 |  28 |  29 |  30 |  31 |
|_____|_____|_____|_____|_____|_____|_____|_____|
|  32 |  33 |  34 |  35 |  36 |  37 |  38 |  39 |
|_____|_____|_____|_____|_____|_____|_____|_____|
|  40 |  41 |  42 |  43 |  44 |  45 |  46 |  47 |
|_____|_____|_____|_____|_____|_____|_____|_____|
|  48 |  49 |  50 |  51 |  52 |  53 |  54 |  55 |
|_____|_____|_____|_____|_____|_____|_____|_____|
|  56 |  57 |  58 |  59 |  60 |  61 |  62 |  63 |
|_____|_____|_____|_____|_____|_____|_____|_____|
player 1 playing now
Which grid do you want to choose?
Now please enter the grid

0

2
```

Then we can see that the transmitted grid location is displayed on the board.

```
player 2 playing now
Player 1 (X)  -  Player 2 (O)


 _____ _____ _____ _____ _____ _____ _____ _____
|  0  |  1  |  X  |  3  |  4  |  5  |  6  |  7  |
|_____|_____|_____|_____|_____|_____|_____|_____|
|  8  |  O  |  10 |  11 |  12 |  13 |  14 |  15 |
|_____|_____|_____|_____|_____|_____|_____|_____|
|  16 |  17 |  18 |  19 |  20 |  21 |  22 |  23 |
|_____|_____|_____|_____|_____|_____|_____|_____|
|  24 |  25 |  26 |  27 |  28 |  29 |  30 |  31 |
|_____|_____|_____|_____|_____|_____|_____|_____|
|  32 |  33 |  34 |  35 |  36 |  37 |  38 |  39 |
|_____|_____|_____|_____|_____|_____|_____|_____|
|  40 |  41 |  42 |  43 |  44 |  45 |  46 |  47 |
|_____|_____|_____|_____|_____|_____|_____|_____|
|  48 |  49 |  50 |  51 |  52 |  53 |  54 |  55 |
|_____|_____|_____|_____|_____|_____|_____|_____|
|  56 |  57 |  58 |  59 |  60 |  61 |  62 |  63 |
|_____|_____|_____|_____|_____|_____|_____|_____|
player 1 playing now
Which grid do you want to choose?
Now please enter the grid

1

0
Player 1 (X)  -  Player 2 (O)


 _____ _____ _____ _____ _____ _____ _____ _____
|  0  |  1  |  X  |  3  |  4  |  5  |  6  |  7  |
|_____|_____|_____|_____|_____|_____|_____|_____|
|  8  |  O  |  X  |  11 |  12 |  13 |  14 |  15 |
|_____|_____|_____|_____|_____|_____|_____|_____|
|  16 |  17 |  18 |  19 |  20 |  21 |  22 |  23 |
|_____|_____|_____|_____|_____|_____|_____|_____|
|  24 |  25 |  26 |  27 |  28 |  29 |  30 |  31 |
|_____|_____|_____|_____|_____|_____|_____|_____|
|  32 |  33 |  34 |  35 |  36 |  37 |  38 |  39 |
|_____|_____|_____|_____|_____|_____|_____|_____|
|  40 |  41 |  42 |  43 |  44 |  45 |  46 |  47 |
|_____|_____|_____|_____|_____|_____|_____|_____|
|  48 |  49 |  50 |  51 |  52 |  53 |  54 |  55 |
|_____|_____|_____|_____|_____|_____|_____|_____|
|  56 |  57 |  58 |  59 |  60 |  61 |  62 |  63 |
|_____|_____|_____|_____|_____|_____|_____|_____|
player 2 playing now
.
```

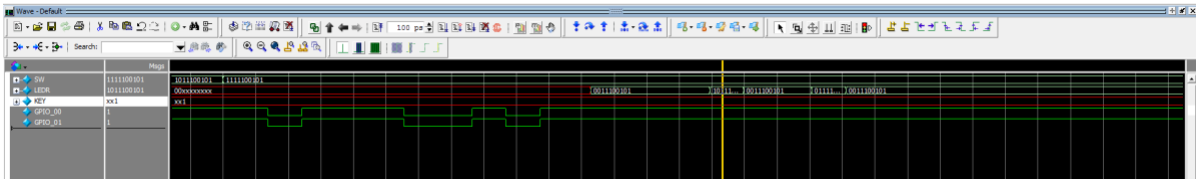
Finally, the below snip shows the game 3 which is the Reversi. Reversi will provide the legal move for user to choose from and user can only place chess pieces with legal move suggestions.

```
which player are you? (1 or 2)
0

1
me is 1
which game do you want to play? 1:tic-tac-toe 2:Gomoku 3:Reversi

0

3
Please only use the legal moves
Player 1 (X)  -  Player 2 (O)


 _____ _____ _____ _____ _____ _____ _____ _____
|  0  |  1  |  2  |  3  |  4  |  5  |  6  |  7  |
|_____|_____|_____|_____|_____|_____|_____|_____|
|  8  |  9  |  10 |  11 |  12 |  13 |  14 |  15 |
|_____|_____|_____|_____|_____|_____|_____|_____|
|  16 |  17 |  18 |  19 |  20 |  21 |  22 |  23 |
|_____|_____|_____|_____|_____|_____|_____|_____|
|  24 |  25 |  26 |  X  |  O  |  29 |  30 |  31 |
|_____|_____|_____|_____|_____|_____|_____|_____|
|  32 |  33 |  34 |  O  |  X  |  37 |  38 |  39 |
|_____|_____|_____|_____|_____|_____|_____|_____|
|  40 |  41 |  42 |  43 |  44 |  45 |  46 |  47 |
|_____|_____|_____|_____|_____|_____|_____|_____|
|  48 |  49 |  50 |  51 |  52 |  53 |  54 |  55 |
|_____|_____|_____|_____|_____|_____|_____|_____|
|  56 |  57 |  58 |  59 |  60 |  61 |  62 |  63 |
|_____|_____|_____|_____|_____|_____|_____|_____|
legal move position detected: 20
legal move position detected: 29
legal move position detected: 34
legal move position detected: 43
player 1 playing now
Which grid do you want to choose?
Now please enter the grid

2

0
Player 1 (X)  -  Player 2 (O)
```

The chess pieces will automatically flip according to game rules.

```
 _____ _____ _____ _____ _____ _____ _____ _____
|  0  |  1  |  2  |  3  |  4  |  5  |  6  |  7  |
|_____|_____|_____|_____|_____|_____|_____|_____|
|  8  |  9  |  10 |  11 |  12 |  13 |  14 |  15 |
|_____|_____|_____|_____|_____|_____|_____|_____|
|  16 |  17 |  18 |  19 |  X  |  21 |  22 |  23 |
|_____|_____|_____|_____|_____|_____|_____|_____|
|  24 |  25 |  26 |  X  |  X  |  29 |  30 |  31 |
|_____|_____|_____|_____|_____|_____|_____|_____|
|  32 |  33 |  34 |  O  |  X  |  37 |  38 |  39 |
|_____|_____|_____|_____|_____|_____|_____|_____|
|  40 |  41 |  42 |  43 |  44 |  45 |  46 |  47 |
|_____|_____|_____|_____|_____|_____|_____|_____|
|  48 |  49 |  50 |  51 |  52 |  53 |  54 |  55 |
|_____|_____|_____|_____|_____|_____|_____|_____|
|  56 |  57 |  58 |  59 |  60 |  61 |  62 |  63 |
|_____|_____|_____|_____|_____|_____|_____|_____|
current score: player1 4    player2 1
player 2 playing now
Player 1 (X)  -  Player 2 (O)


 _____ _____ _____ _____ _____ _____ _____ _____
|  0  |  1  |  2  |  3  |  4  |  5  |  6  |  7  |
|_____|_____|_____|_____|_____|_____|_____|_____|
|  8  |  9  |  10 |  11 |  12 |  13 |  14 |  15 |
|_____|_____|_____|_____|_____|_____|_____|_____|
|  16 |  17 |  18 |  O  |  X  |  21 |  22 |  23 |
|_____|_____|_____|_____|_____|_____|_____|_____|
|  24 |  25 |  26 |  O  |  X  |  29 |  30 |  31 |
|_____|_____|_____|_____|_____|_____|_____|_____|
|  32 |  33 |  34 |  O  |  X  |  37 |  38 |  39 |
|_____|_____|_____|_____|_____|_____|_____|_____|
|  40 |  41 |  42 |  43 |  44 |  45 |  46 |  47 |
|_____|_____|_____|_____|_____|_____|_____|_____|
|  48 |  49 |  50 |  51 |  52 |  53 |  54 |  55 |
|_____|_____|_____|_____|_____|_____|_____|_____|
|  56 |  57 |  58 |  59 |  60 |  61 |  62 |  63 |
|_____|_____|_____|_____|_____|_____|_____|_____|
current score: player1 3    player2 3
legal move position detected: 10
legal move position detected: 18
legal move position detected: 26
legal move position detected: 34
legal move position detected: 42
player 1 playing now
Which grid do you want to choose?
Now please enter the grid
```

The following is the testbench result of the communication system. It shows a sample transmission.



5. Error / Failure Analysis (5)

a. Document any errors you ran into in this lab project

- One bug we encountered frequently is failure report of invalid ELF file. ELF file represents the connection between C and quartus project, if it is missed the C project cannot be downloaded to the FPGA board. But encountering this error is kind of random, most frequent reason for it to happen is the ELF file is not stored in the right folder and eclipse cannot auto find it. So when we encountered it we would choose to first check if the ELF file is under the software folder and go to properties to manually choose the ELF file. In most situations this solves the problem.
- During the implementation of the serial communication part, we have encountered a problem with the receive system. The signal we received always contains an extra bit and miss the first bit of the original data. This is caused by the state machine updating logic. The system should start to receive when the start bit changes, which reflects in the state machine is that when the start bit changes, the state register will change to a different value, but originally the count would only start when the state register value changed. This is fixed by adding extra logic in the state machine so that the count would start without one cycle delay.
- When we are connecting all the submodules into the top level module, we mistakenly invert the input of the reset signal, which is a key on the DE1_SoC board. For this board, the signal sent out by the key is high when it is not pressed. However, in the real connection we inverted it once more, so when we are debug the system there is nothing shown because the system is always at the reset mode.

b. Discuss potential failures with your design

- The serial communication system we designed only suits to send and receive on character at a time. During the chess play, the users will enter the index of location where they want to place a piece. Since we the maximum board size is 64, so the user have to enter 2 numbers, which will be converted to 2 ASCII characters essentially. Two numbers should be entered separately. If the user entered the wrong number or entered both number at the same time, the system would stop responding. Since the serial communication system will only send the next character when the first one is already sent.

6. Summary and Conclusion (5)

As finishing up the lab4 and final project, we successfully implemented a working NIOS II microprocessor, a efficient hardware transmission/communication system, and a board game collection as we promised in the proposal. We met all our deliverables including:

1. A serial communication interface that can transfer the user input from one FPGA board to another
2. A text-base based user interface that can allow the user to select which game to play, and show the rules of the games, and displays messages recording the actions taken by each user.
3. A graphical display interface that can reflect the user actions
4. A NIOS II processor based system
5. A functional *Tic-Tac-Toe* game
6. A functional *Reversi* game
7. A functional *Gomoku* game
8. A top level system that combines the sub-games

Through this lab, we combines all our knowledge about c programming, verilog programming, and NIOS processor building together to achieve the result we want. We learned how each part should be connected and how to implement the system to as "robust" as it can be. The difficulty of final project is much higher than any other lab we did before, but we did deepen our understanding about hardware and software interface throughout the design process.

7. Individual Contribution

Name:	Contribution:
Zewei Du	Report writing, Software Implementation
Lingli Zeng	Report writing, Software Implementation
Zicheng He	Report writing, Hardware Implementation
Jichun Li	Report writing, Hardware Implementation