ECE272 Final Design

Controller Input and VGA

Drew Ortega
Cole Swanson
Jonathan Alexander

November 27, 2018

Contents

1 Introduction **2**

2 Top Level Design **2**

3 Controllers **2**
- 3.1 NES/SNES Controller . . . 2
- 3.2 IR Controller . . . 2
  - 3.2.1 Self-Clocking Signal . . . 2
  - 3.2.2 IR Signal & RC 5 Code . . . 3

4 HDL Modules **4**
- 4.1 NES/SNES input decoder . . . 4
- 4.2 IR Input Decoder . . . 5
  - 4.2.1 IR Clocks . . . 6
- 4.3 IR Simplifier . . . 7
  - 4.3.1 IR Parser . . . 8
- 4.4 PS/2 Input Decoder . . . 8
- 4.5 Input Decoder/Multiplexer . . . 8
- 4.6 Game Logic Controller . . . 9
- 4.7 Player Top Module . . . 9
- 4.8 Pixel Counter . . . 10
- 4.9 Renderer . . . 10

5 Putting it Together **10**

6 Extra Implementation **10**

7 Appendix **10**

1 Introduction

TODO

2 Top Level Design

Inputs: TODO

Outputs: TODO

Simulation: TODO

Description: TODO

3 Controllers

3.1 NES/SNES Controller

The NES and SNES controllers utilize three lines to transfer data, along with a ground and 5V power supply. Communication wit the controllers uses a data latch line, clock line and data line. The clock and data latch lines are supplied to the controller, while the data line is generated. When the latch line is driven high, the controllers log the pressed buttons within an active low shift register. The first bit of this register is automatically output on the data line. Whenever the clock input is cycled, the shift register moves along by one bit. The NES controller has eight buttons, and an eight bit shift register, while the SNES has sixteen buttons and a sixteen bit shift register. The data latch can then be cycled to obtain a new set of inputs. For this design, the up, down, left, and right inputs were of interest. Within the NES controller, the zeroth bit of the shift register corresponds to the right input, the first to the left input, the second to the down input, and the fourth to the up input. The final four inputs were ignored. Within the SNES controller, the up input corresponds to the fifth bit in the shift register, the down input to the sixth bit, the left input to the seventh bit, and the right input to the eight bit.

To communicate with the NES or SNES controllers, the host must do the following:
1. Drive the data latch input high to populate the shift register
2. The first data bit can now be read
3. The clock input should be driven high, then low
4. The next data bit can now be read
5. Steps 3-4 should be repeated 7 times for the NES, 15 times for the SNES(as the first bit of the register is already present, one less clock cycle is needed)

3.2 IR Controller

3.2.1 Self-Clocking Signal

Figure 1: Manchester Code/Bi-Phase Coding: from Data to Output

The IR Controller uses a self-clocking signal to send data to the receiver. For a signal to be self-clocking, the source must have a clock of a specified speed and a serial data output. The source combines the clock and serial output in to a single serial output of a designated encoding. For the IR Controller, the IEEE Manchester Code specification, also known as "Bi-Phase Coding" (BPC), is used. For BPC, the source checks the data it wishes to output on every rising edge of its clock. If the data to be transmitted is a 1, than the serial output is driven to mimic the clock for that cycle. Essentially, it rises on a rising edge of a clock, and falls on the falling edge of the clock. If the data to be transmitted is a 0, than the serial output begins to be the inverted signal of the clock. It falls on the rising edfe of the clock, and rises on the falling edge of the clock. What a Bi-Phase Coding might look like can be found in Figure 1.

3.2.2 IR Signal & RC 5 Code

Figure 2: RC 5 Code - Chunk and delay

This implementation of IR reception assumes that the IR signal received will be in the form of BPC, as explained in 3.2.1. There are several standards for IR data transmission. The that is decoded for this design is the RC 5 Code (RC5). RC5 sends data in chunks, separated by a static amount of clock cycles at 36 kHz. Each chunk is has four sections, totalling to 14 bits.

- Section 1 - Start Bits (2 bits): Help orient the reciever to understand source clock timing.
- Section 2 - Toggle Bit (1 bit): Defines whether or not this chunk is a new key press or a held key already transmitted.
- Section 3 - Address (5 bits): Defines the address of the receiver. If this does not match the expected address, the data is ignored.
- Section 4 - Data (6 bits): Defines the actual data of the chunk, up to 6 bits. A single combination represents an individual button.

After the full chunk has been transmitted, the serial data signal is driven to 0 for 50 clock cycles before data is transmitted again. After the 50 clock cycles, if a button on the controller is still being pressed, the transmitter on the controller will send the same data chunk again, except the toggle bit will be 0 instead of 1. If after 50 cycles no data is read, there is no button being pressed or the receiver is not receiving data.[1] The structure and timing of RC 5 Code can be found in Figure 2.

4 HDL Modules

4.1 NES/SNES input decoder

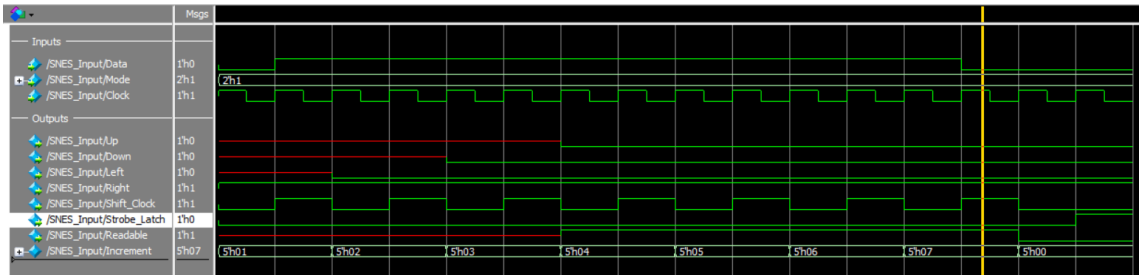

Figure 3: ModelSim of the NES/SNES controller decoder in NES mode.

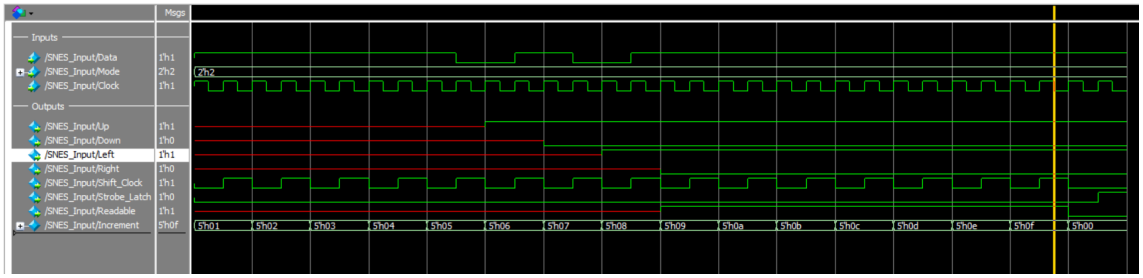

Figure 4: ModelSim of the NES/SNES controller decoder in SNES mode

Inputs: One bit data representing the current bit from the controller shift register, one bit clock, and two bit mode used to determine which controller is in use(NES or SNES)

Outputs: One bit up, down, left, and right representing the decoded values from the controller, and one bit readable output which is driven high once the current set of values have been decoded. Also present are the one bit strobe latch output, which functions as the data latch for the controller to be driven high then low to log the inputs to the controller, and a one bit shift clock, which is used to shift the values in the controller shift register.

Simulations:

NES:
For this simulation, the data bit for right was driven low(representing an input of 'right'). The data_latch is initially driven high. As expected, the decoder logs the right input as pressed, and continues through the inputs by providing a shift_clock to the controller. Once the data has been cycled three times, all values of interest are accounted for, and the readable output is driven high. After seven cycles of the output, the data_latch is again driven high and readable is driven low; new inputs are ready to be read. The value of increment represents the current bit of the shift register.

SNES:
For this simulation, the data bits for up and left were driven low(representing inputs of 'left' and 'up'). The data_latch is initially driven high. As expected, the decoder logs the up and left inputs as pressed, and continues through the inputs by providing a shift_clock to the controller. Once the data has been cycled eight times, all values of interest are accounted for, and the readable output is driven high. After fifteen cycles of the output, the data_latch is again driven high and readable is driven low; new inputs are ready to be read. The value of increment represents the current bit of the shift register.

This module will use the system clock to drive its logic. On the first clock cycle, the data latch output will be driven high to log the controller inputs. The data latch will then be driven low, and the shift clock will be cycled seven times for the NES controller, or fifteen times for the SNES controller, with the bit corresponding to the inputs of interest recorded. Once all inputs of interest have been recorded, the readable output will be driven high, signaling all inputs are accounted for. Pressed buttons on the controllers will be signaled by a high value on the corresponding output from the decoder(up, down, left, or right).

4.2 IR Input Decoder

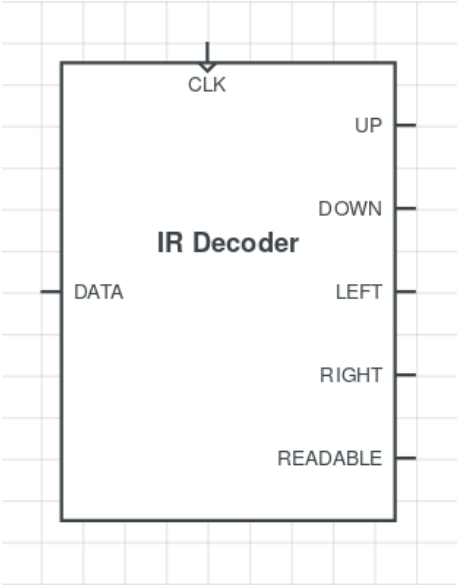

Figure 5: IR Top Module Diagram

Figure 6: IR Top Module Simulation

Input: One Serial Data stream 'Data', Clock signal.

Output: One output for each cardinal direction (Up, Down, Left, Right), and signal describing whether or not the directions are valid or not, 'Readable'.

Description: For the IR Decoder, it was noted that its output must match the output of any other input decoder, such as the SNES/NES and PS/2. Because of this, the Up, Down, Left, Right, and Readable streams were required to be used for this particular module. This particular model

Simulation: TODO

4.2.1 IR Clocks

Figure 7: IR Clock Slowing Block Diagram

Figure 8: IR Clock Simulation

Figure 9: IR data received vs. 72 kHz Clock

Inputs: TODO

Outputs: TODO

Simulation: TODO

Description: The IR Input reciever takes in the BPC serial data stream and a 50MHz Clock signal. A slowed clock is generated to slow it down to approximately 72 kHz. This is achieved by dividing the 50 MHz clock by 694. However, if the Clock was flipped every 694 50 MHz cycles, it

would have a clock cycle of 36 kHz. To account for this, we must double the resulting clock, by dividing the original clock by 347. With the 72 kHz clock, for each rising edge of this clock the IR signal will have either a rising edge or a falling edge. Since the IR signal is asynchronous, meaning its clock will not line up with our clock, the 72 kHz clock can check on each of its rising edges whether or not the Data input is a 0 or a 1. An example of what the async data vs our internal clock can be found in Figure 9.

4.3 IR Simplifier

Figure 10: IR Simplifier Block Diagram

Figure 11: IR Simplifier Simulation

Inputs: TODO

Outputs: TODO

Simulation: TODO

Description:

4.3.1 IR Parser

Figure 12: IR Decoder Simulation

Figure 13: IR Decoder Simulation

Inputs: TODO

Outputs: TODO

Simulation: TODO

Description: TODO

4.4 PS/2 Input Decoder

Inputs: TODO

Outputs: TODO

Simulation: TODO

Description: TODO

4.5 Input Decoder/Multiplexer

Inputs: TODO

Outputs: TODO

Simulation: TODO

Description: TODO

4.6 Game Logic Controller

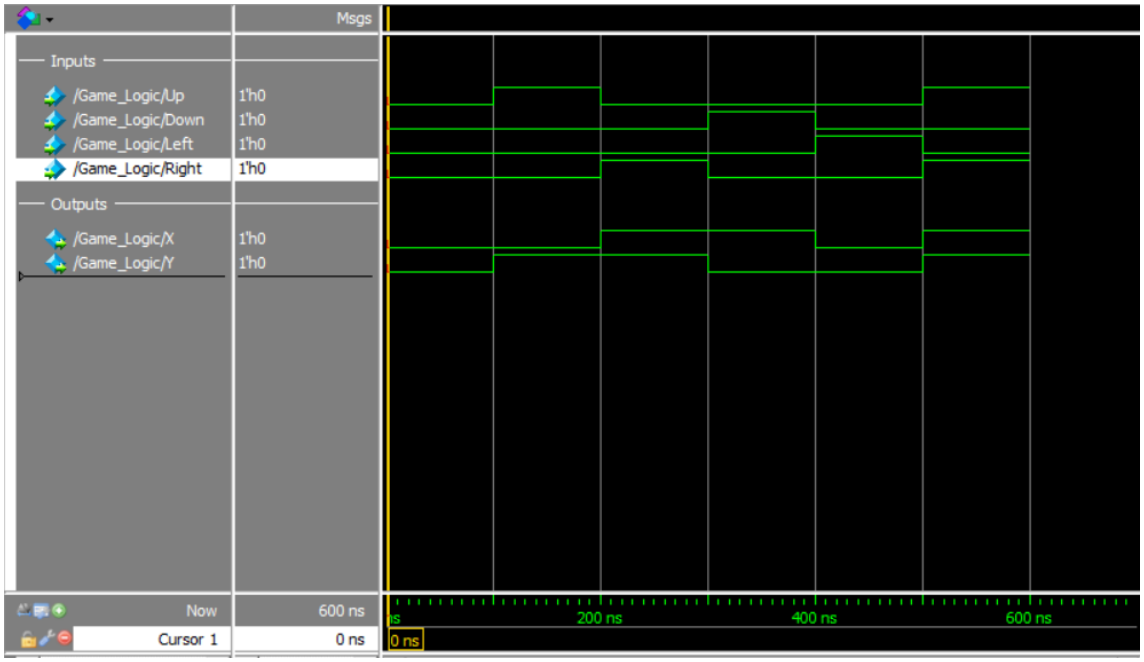

Figure 14: ModelSim of the Game Logic Controller.

Inputs: One bit inputs for up, down, left, and right

Outputs: One bit data inputs for X and Y, representing the position of the player object within the four by four grid where 0,0 is the lower left corner

Simulation:

Within this simulation, the initial position of the player, 0,0. The up input is driven high, and as expected the coordinate output for Y also changes to high. Next, a right input is provided, and again the coordinates change such that X is high. After this, down is driven high and then left is driven high, resulting in the coordinates shifting to 0,1 then 0,0. Finally, both up and right are driven high, resulting in a coordinate of 1,1.

This module will use combinational logic to determine the coordinate state of a player. The current position is stored within the module, and from the provided values of up, down, left, and right the next position is determined. An input that would result in the player going outside of the coordinate bounds(0,0 to 1,1) is ignored. Multiple inputs can be accepted at once, allowing for diagonal movement.

4.7 Player Top Module

Inputs: TODO

Outputs: TODO

Simulation: TODO

Description: TODO

4.8 Pixel Counter

Inputs: TODO

Outputs: TODO

Simulation: TODO

Description: TODO

4.9 Renderer

Inputs: TODO

Outputs: TODO

Simulation: TODO

Description: TODO

5 Putting it Together

TODO

6 Extra Implementation

TODO

7 Appendix

TODO

References

[1] Vishay. Data Formats for IR Remote Control.
`https://www.vishay.com/docs/80071/dataform.pdf`