CS 150 CHECKPOINT 3

Video Interface
Jeffery Tsai
Sammy Sy
And pretty much everything else too....

The Project so far
- Controller Interface
- Audio Interface
- Video Interface
- Game Engine

Checkpoint 3 Overview
- UART
- Serial Connection
- Learn Sammy’s GUI
- Memory Module (16x8 RAM)
- Simplified Game Engine
- Integration with previous checkpoints

Overall State Machine
- Poll the controller like normal and update the game state.
- Update the video 4 times each second.

Serial Cable Protocol
- Regular Serial Cable
- Only need 2 pins since we only care about sending data to the computer.
  - Pin 2 is to transmit (to the computer)
  - Pin 5 is ground
  - Pin 3 is there, but NOT USED

RS232 Waveforms
- 8 bits of data, 1 start bit, 1 stop bit
- Wire is normally high, and this time, you MUST keep it high.
- Transmission rate = 57600bps
57,600bps?!?!

- You can’t get a 57.6kHz clock with a conventional clock divider. (16,000,000/57,600 = 278)
- Use the 16MHz clock, and pulse the output every 278 cycles.
- By the way.....remember to use BUFG’s for ALL your clocks!

UART

- Universal Asynchronous Receiver Transmitter.
- Since we don’t receive messages from the computer, only the transmitter is needed.
- Need a READY signal since it is critical that you try to send at the maximum rate.

Maxim Transceiver

- Actual serial connection is inverted and is 12V.
- Maxim transceiver takes care of everything.
- BUT BE CAREFUL!!! DO NOT PLUG THE 12 VOLT END TO YOUR XILINX CHIP!

Video Interpreter

- This “DreamKatz” program draws the game screen on the computer monitor.
- <demonstration>

Protocol (1)

- Types of messages (all 1 byte wide)
  - SYNC
    - Game start (GS)
    - Game over (GO)
    - Game play (GP)
  - DATA
    - Data byte (D)

Protocol (2)

- Only 3 byte sequences, all preceded by a SYNC byte.
- Each sequence can be followed by a not her valid sequence.
  - Display game start screen
    - GS
  - Display game over screen with final score
    - GO, D, D
  - Display a new frame (grid + mana = score)
    - GP, DD, D1, D2, .. D127, D, D, D

Reminder:
- GS = game start
- GO = game over
- GP = game play
- D = data byte
Protocol (3)

- Physical representation in bits
  - Game start = 1000 0001
  - Game over = 1000 0000
  - Game play = 1000 0010
  - Data = 0XXX XXXX

- Hints:
  - 0 <= Data <= 127
  - 1XXX XXXX is always illegal

Protocol (4)

- Display game start screen
  - easy... just send GS byte
- Display game over screen and final score
  1. GO: 1000 0000
  2. D: 0, score [13:7]
  3. D: 0, score [6:0]

Protocol (5)

- Display a frame
  - GP: 1000 0010
  - D0: 0000 0001 (block0 = white)
  - D1: 0000 0001 (block1 = white)
  - ...
  - D127: 0000 0001 (block127 = white)
  - D: 0000 0000 (manaHi = 000 0000)
  - D: 0000 0111 (manaLo = 000 0111)
  - D: 0000 0001 (scoreHi = 000 0001)
  - D: 0000 0000 (scoreLo = 000 0000)

Protocol Example

- Let’s display a frame with all white blocks (white= 000 0001), mana=7, score=128

<table>
<thead>
<tr>
<th>GP</th>
<th>D0</th>
<th>D1</th>
<th>..</th>
<th>D127</th>
<th>D</th>
<th>D</th>
<th>..</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000 0010</td>
<td>0000 0001</td>
<td>0000 0001</td>
<td>0000 0001</td>
<td>0000 0001</td>
<td>0000 0001</td>
<td>0000 0001</td>
<td>0000 0001</td>
</tr>
</tbody>
</table>

Killer checkpoint => Hints

- The master FSM is complicated, but fortunately runs sequentially.
- Make sure you design it with Control and Datapath in mind.
- You might find pseudocode helpful for laying out the FSM’s and thus the control signals for the datapath. If pseudocode is good, datapath falls out naturally from it.

Pseudocode (1)

```c
Memory<module GameState; // insn ram

// Rotate all bits of GameState to left/right once

void HandleLeftRight(bool left, bool right) {
    8bitRegister tmp;
    for (I = 0; I < 16; I++) {
        if (left) tmp = UniversalShiftRegister(tmp, left, right);
        GameState[I] = tmp;
    }
}

void HandleDown() {
    8bitRegister tmp;
    for (I = 15; I >= 1; I--) {
        // process backwards!
        tmp = GameState[I-1];
        GameState[I] = tmp;
    }
    GameState[0] = 0000 0000;
}
```
/* encode GameState into frame update message sequence */
HandleEncode() {
    8bitShiftRegister tmp;
    UART_Send(1000 0010);  // sync game play byte
    for (I = 0; I < 16; I++) {
        tmp = GameState[I];
        for (J = 0; J < 8; J++) {
            UART_SEND(0000 000,tmp[J]);  // either white or black
            tmp = tmp >> 1;
        }
    }
    UART_SEND(0, MANA[13:7]);
    UART_SEND(0, MANA[6:0]);
    UART_SEND(0, SCORE[13:7]);
    UART_SEND(0, SCORE[6:0]);
}

/* Main loop
* ENCODE_LIMIT = period of video refresh
* FALL_LIMIT = period of blocks falling */
HandleMain(bool left, bool right) {
    count = 0;
    while (1) {
        HandleLeftRight(left, right);  // process left/right
        if (0 == (count % FALL_LIMIT)) {  // process gravity
            HandleDown();
        }
        if (0 == (count % ENCODE_LIMIT)) {  // process video
            HandleEncode();
        }
        count ++;
    }
}