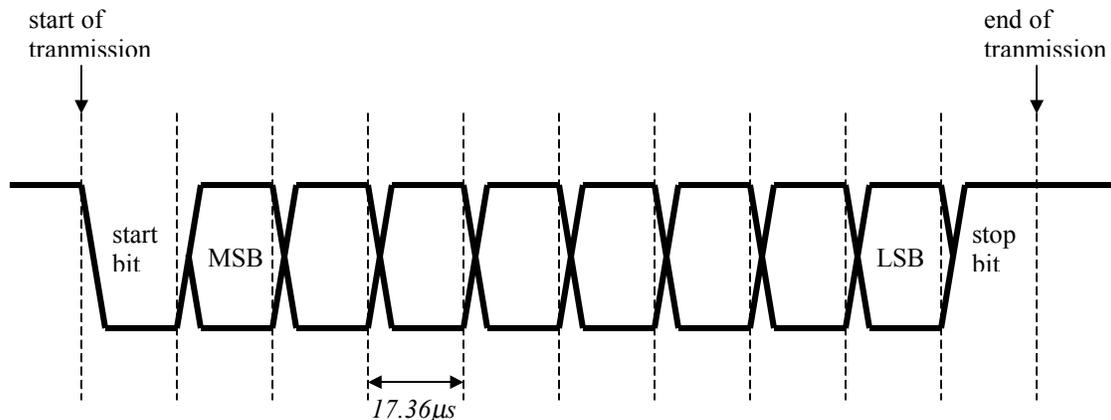


### Project Checkpoint 3 Video Interface

#### UART Overview

A UART is a universal asynchronous receiver transmitter. In this checkpoint we will be using a UART transmitter to transmit data onto the RS232 line that is ultimately received by the video display program.

The UART transmitter serially transmits one byte at a time, MSB first. Each transmission also has a start bit (logic 0) and a stop bit (logic 1). We will be transmitting at a baud rate of 57600 bps, which corresponds to a bit length of *approximately*  $17.36\mu\text{s}$ . The line is normally high (logic 1) when idle.



#### Video Protocol Overview

The DreamKatz video protocol allows for three types of screens: the game start screen, the game over screen, and the game play screen.

Since we are using the UART to send the video data to the computer, each piece of data is one byte. The MSB of each byte determines if that byte is a synchronization (sync) byte – a byte used to indicate the start of a new frame. The lower 7 bits are used to store the actual data.



A sync bit of 1 indicates that the byte is a sync byte. Each new video frame must begin with a sync byte so that the video display program knows when the data for each video frame begins. All other data bytes have a sync bit values of 0. Since there are three types of screens, there are three valid sync bytes, each with a different data segment (lower 7 bits of the byte).

The sync byte is then followed by the necessary data bytes to display that particular type of screen.

	<b>game start</b>	<b>game over</b>	<b>game play</b>
<b>data byte sequence</b> (sync byte always first)	0x81	0x80 <score hi> <score lo>	0x82 <block 0> <block 1> : : <block 127> <mana hi> <mana lo> <score hi> <score lo>

### Game Start Screen

The game start screen only requires the sync byte, there is no other data that needs to be sent. Once the video display program receives a game start sync byte, it will display a predetermined start screen.

### Game Over Screen

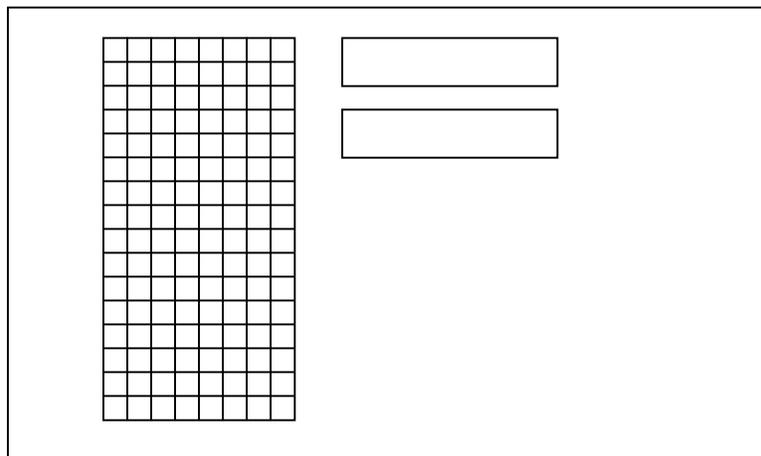
To display the game over screen, the final score needs to be sent so that it can be displayed, in addition to the sync byte. The score is sent in two bytes, where the first byte sent contains the higher seven bits (in the 7 bit data segment), and the second byte contains the lower seven bits. Thus, the score is a 14-bit number.

### Game Play Screen

The game play screen is divided into 128 blocks, arranged in 16 rows of 8 columns. The blocks are numbered from 0 – 127, starting with the top left, moving across the columns first, and then down the rows. Also, the game play screen shows the current score and number of mana points.

The block data is sent one block at a time, starting with block 0 and ending with block 127. After the block data has been sent, the mana points and score are sent, each being a 14-bit number divided into two 7-bit halves; the higher 7-bits are sent first.

### game play screen



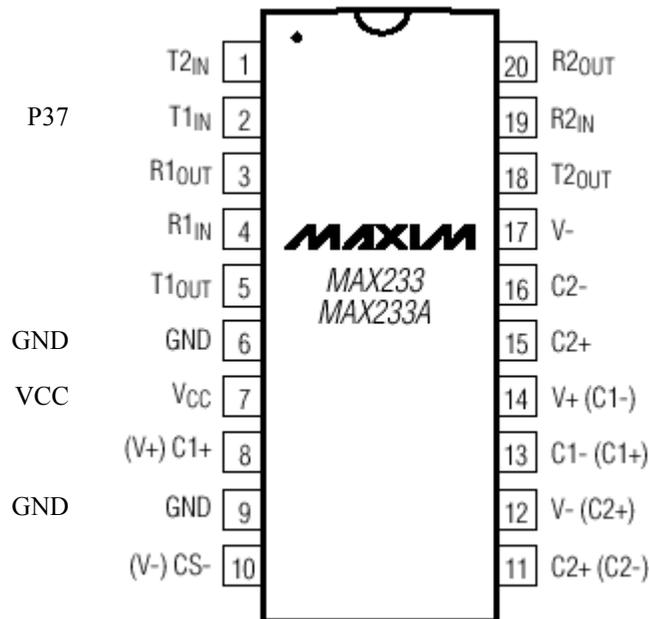
The 7-bit data segment of each block specifies a certain image to use to display that block. There are a total of 128 different blocks available for display. For example, if block 0 was sent as 0x01, then that would cause the top left block in the game play screen to be white.

7-bit data segment	display block
0	black
1	white
2	red
3	orange
4	yellow
5	green
6	blue
7	violet
8-127	???

### Prelab Preparations

This checkpoint only requires the MAX233 chip to be wire wrapped. The MAXIM chip is just a driver used to drive the RS232 serial line.

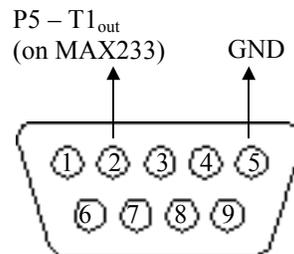
#### connection



#### Serial Port Connection

The output of the MAXIM chip will be used to connect to the serial port of the computer. We will only use two lines of the serial connector:

- 1) Pin 2 – RX – this is the receive line (relative to the computer), which will carry the data sent out by the UART transmitter. It connects to the output of the MAX233, which is pin 5 (T1<sub>out</sub>).
- 2) Pin 5 – GND – this is the reference ground line.



Notice that there is another connector on pin 3 of the serial cable. This is the TX line, used for receiving data from the computer. This line is not used, and can just be left unconnected.

### Video Display Program

The video display program is located in <U:\cs150\video>. It is recommended to copy the entire program (all folder contents) to your home directory.

The program will output errors if the protocol is not followed correctly. For example, if a sync byte is sent while in the middle of sending the 128 blocks' data, an error message will be displayed. However, the program will continue as if there were no error and receive the new sync byte as if it were the beginning of a new frame.

### High Level Specifications

Upon startup, your FSM should display a blank game play screen, with score and mana points of the initial values of 0. We will not be using the game start screen or the game over screen for this checkpoint.

A blank game play screen should consist of all 128 blocks being black, although you are free to choose any block you wish to be the blank block, such as white, green, etc.

When a button is pressed, a 1x1 block should appear at the top of the game play area and begin to fall downwards. Also, a sound should play and the score (or mana, or both) should be incremented by 1. You are free to choose any button to perform this action, and any one of the six sounds can be played. Also, you can choose at which column the block will appear.

The 1x1 block may fall at any speed, although it should not be too slow or too fast. A rate of about one row per second is probably best. As the block is falling, pressing the left and right directional buttons on the controller should move the block left or right by one block unit.

If the block gets moved left past the left side of the game play area, it should wrap around to the right side. Similarly, the block should wrap around to the left side if it is moved past the right side. When the block falls past the bottom of the game play area, it should just disappear.

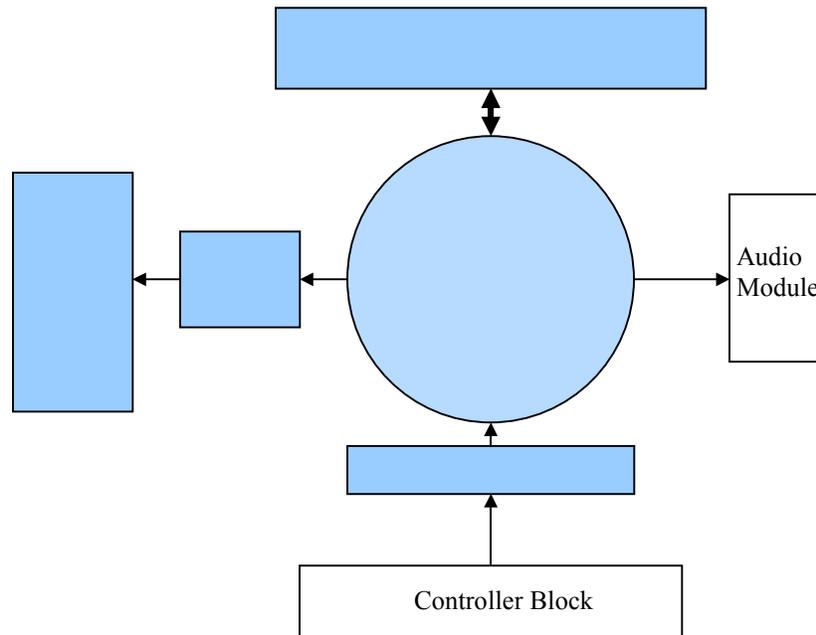
It is possible to have more than one block falling at a time, if the button is pressed while there is still another block falling. In this case, pressing left and right should move both of the blocks left and right.

### Frame Rate

The screen should be updated at 4 frames per second. So every 250ms, a sync byte, along with the necessary block, mana, and score data bytes should be sent to the video display program.

## Design Structure

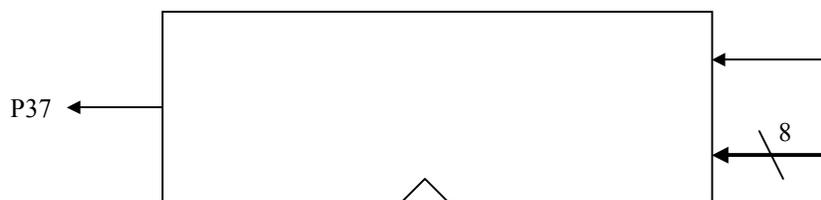
The goal of this checkpoint is to build a skeleton of the final project. All the components from previous checkpoints will be incorporated, along with some new components. New components are highlighted in blue:



## Design Specifications

### UART Transmitter

The UART transmitter should take a transmit signal (a one-clock-cycle pulse) and an 8-bit data byte. Upon receiving the transmit pulse, it should latch the byte into a register and begin shifting out the byte (along with the start and stop bits) at a speed of 57600 baud, in accordance with the UART specifications discussed earlier.



The serial output of the UART should be output to pin 37, where it will be connected to the MAX233 RS232 driver. Remember that the line is high when idle.

The speed of the clock for the UART should be fast, preferably 16mhz. This will allow for a better approximation to the 57600 baud rate.

If the UART receives another transmit request while it is busy sending a transmission, the new request should be ignored.

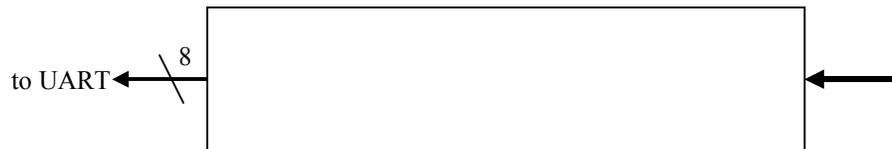
### Memory Module

The memory module stores the current game state. This includes the 128 blocks as well as the mana and score values. Normal 16-bit registers can be used to store the values for mana and score, while RAM (preferably RAM16X8S) can be used to store the game play state.

The memory module should interface with the rest of the system through read and write ports, address lines, write enable signals, etc.

### Video Encoder

The video encoder is responsible for generating the data byte that is to be sent out through the UART.

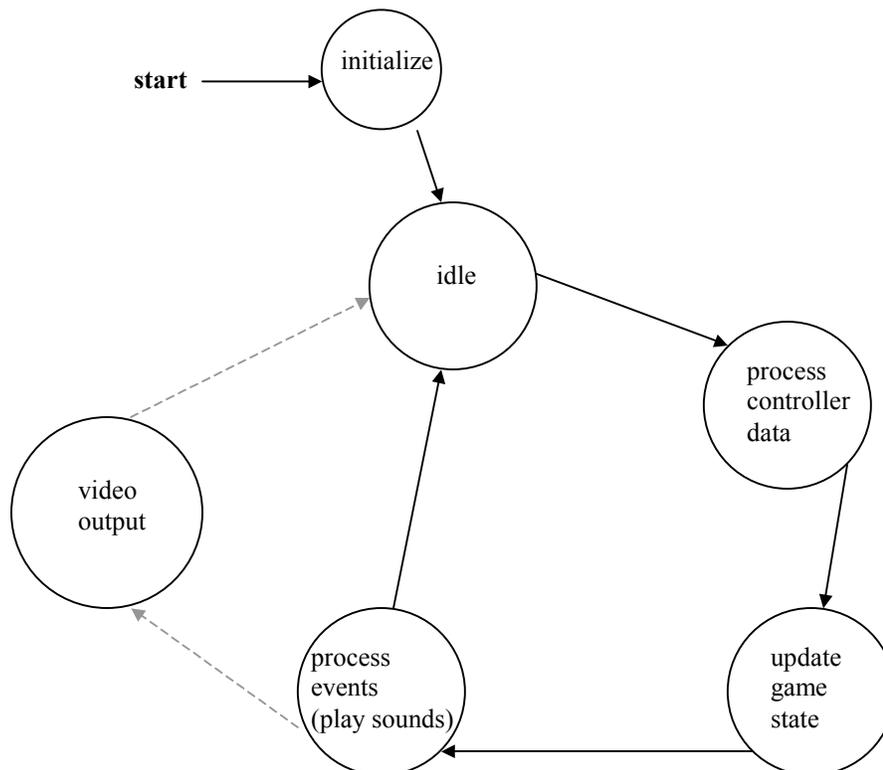


The data inputs to the video encoder should consist of signals that are needed to determine the data byte, such as sync bit, mana and score values, block state etc.

### Controller Interpreter and FSM

The controller interpreter should be somewhat similar the controller interpreter built in checkpoint 2. It should keep track of button presses and store them so that the FSM knows which buttons were pressed during a certain time period.

The FSM should have the following structure:



## **FSM Implementation**

There are two basic methods of implementing the FSM structure. One method is to create a state machine that cycles through all of the stages, including the video output stage, on each cycle. Thus, the cycle would have to run every 250ms to ensure that the video is output at 4 frames per second.

The drawback to this method is that it complicates the controller interpreter, because it would have to keep track of multiple button presses (and the order in which they were pressed) since the controller data is only processed once every 250ms. Also, updating the game state would be more complicated, since it would require that each button press be interpreted separately and in order.

A second method of implementing the FSM structure is to use two communicating state machines. One state machine processes controller data, updates the game state, and plays sounds. The other only handles video output. This allows the game state to be updated whenever is necessary, without having to wait for the cycle (as in the first method) to start.

The drawback to this method is that the timing of these state machines is much more complicated than in the first method. Both machines need to access the memory module, but the video output state machine **MUST** operate every 250ms, so the memory must be available when the video output machine needs it. Thus, these two state machines must be well timed and coordinated.

Of course, there are clever ways of getting around the drawbacks for each of these two methods. Also, these two methods are not the **ONLY** ways of implementing the FSM. Make sure to completely plan out all aspects of the FSM such as state transition diagram, timing behavior, etc. before you attempt to implement it.

## **Things To Do**

- Wire wrap the MAXIM chip
- Create the UART Transmitter
- Create the Memory Module
- Create the Video Encoder block
- Create the Controller Interpreter and FSM

Name \_\_\_\_\_

Name \_\_\_\_\_

### **Project Checkpoint 3 Checkoff Sheet**

#### **Design**

FSM state diagram (more detailed) \_\_\_\_\_

#### **Implementation / schematics**

UART Transmitter \_\_\_\_\_

Memory Module \_\_\_\_\_

Video Encoder block \_\_\_\_\_

Controller Interpreter \_\_\_\_\_

FSM \_\_\_\_\_

#### **Testing**

4 frames per second (oscilloscope) \_\_\_\_\_

Everything works \_\_\_\_\_

**finished 1<sup>st</sup> week** (extra credit... yay!) \_\_\_\_\_

**finished 2<sup>nd</sup> week** (normal credit) \_\_\_\_\_