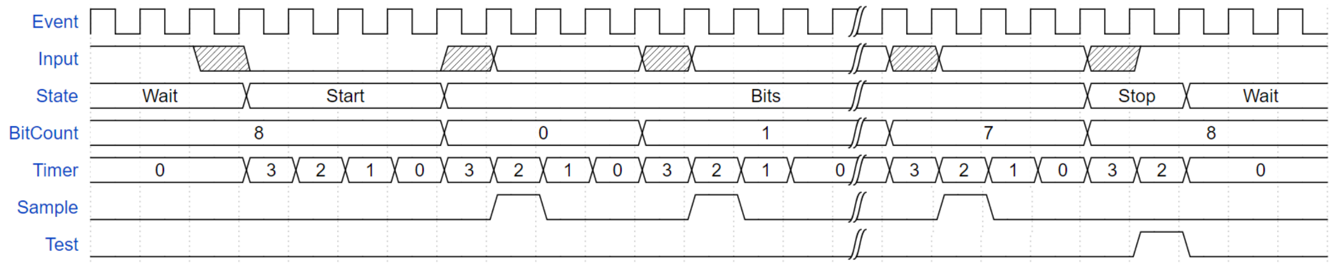
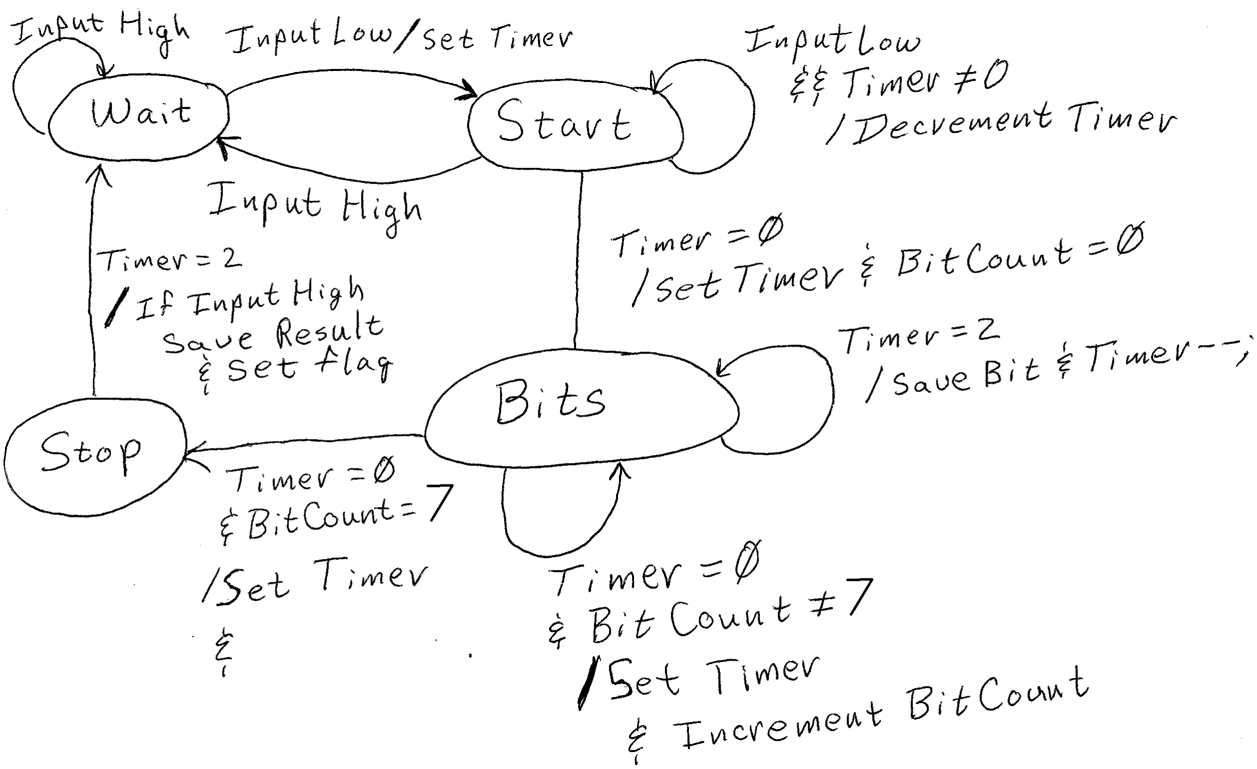
This document will describe the state machine design that was done to build a software based serial port on the Arduino.

First we will look at the timing that is involved in doing 4 times oversampling.



A description of the timing that is involved here will be described in class.

This timing can also be described in a State Transition Diagram (STD) as shown here.



Finally the whole thing can be represented in the following code. First the header file, holding all the special functions written for the Serial port

```
#ifndef SW_Serial_Out_H
#define SW_Serial_Out_H 1

#define OVER_SAMPLE 4
// Define buffer for outgoing data.
char SW_Serial_Out_Buffer;
int SW_Serial_Out_Flag = 0;
int SW_Serial_Out_Pin; // Output Pin for Serial port.
// Set up transmitter state machine
enum SW_Serial_Out_States {
	SW_Serial_Out_Idle, // State waiting for character
	SW_Serial_Out_Start, // Start Bit Tranmit
	SW_Serial_Out_Data
}; // Send data
SW_Serial_Out_States SW_Serial_Out_State = SW_Serial_Out_Idle;
int SW_Serial_Out_BitCount = 0;
int SW_Serial_OutTimer = OVER_SAMPLE - 1;
char SW_Serial_Out_Hold = 0; // Holds data being transmitted.

//////////////////////////////////////////////////////////////
// Define buffer for outgoing data.
char SW_Serial_In_Buffer;
int SW_Serial_In_Flag = 0;
int SW_Serial_In_Pin; // Output Pin for Serial port.
// Set up transmitter state machine
enum SW_Serial_In_States {
	SW_Serial_In_Wait, // State waiting for character
	SW_Serial_In_Start, // Start Bit wait
	SW_Serial_In_Bits,  // Send Bits
	SW_Serial_In_Stop,    // Stop Bit 0
}; // Input decoding states

SW_Serial_In_States SW_Serial_In_State = SW_Serial_In_Wait;
int SW_Serial_In_BitCount = 0;
int SW_Serial_InTimer = OVER_SAMPLE - 1;
char SW_Serial_In_Hold = 0; // Holds data being transmitted.

/////////////////////////////////////////////////////////////
// Interrupt Service Routine that sends characters serially
// over a pin, using a state machine.
void SW_Serial_ISR(void)
{
	// Check for four ISR, since output only
	// changes at one fourth the sampling rate.
	if (SW_Serial_OutTimer)
	{
		SW_Serial_OutTimer--;
	}
	else // Once we have four intervals.
	{
		SW_Serial_OutTimer = OVER_SAMPLE - 1;
		// Based on state,
		switch (SW_Serial_Out_State)
		{
		default:
		case SW_Serial_Out_Idle: // idling waiting for data in buffer
			if (SW_Serial_Out_Flag)
			{
				// Data in buffer, so Pull out data, and save for later.
				SW_Serial_Out_Hold = SW_Serial_Out_Buffer;
				SW_Serial_Out_Hold &= 0x7f; // Mask off top bit
				// Could add parity bit?
				// Advance OUT pointer
				SW_Serial_Out_Flag = 0; // Reset Flag
				digitalWrite(SW_Serial_Out_Pin, LOW); // Clear output (start bit).
				SW_Serial_Out_State = SW_Serial_Out_Start; // Move to start state.
			} // End of character in if
			break;
		case SW_Serial_Out_Start:
			// Send out LSB.
```

```
                digitalWrite(SW_Serial_Out_Pin, bitRead(SW_Serial_Out_Hold, 0)); // Set output.
                SW_Serial_Out_BitCount = 1; // Next bit to send.
                SW_Serial_Out_State = SW_Serial_Out_Data; // Move to data sending state.
                break;
        case SW_Serial_Out_Data:
                if (SW_Serial_Out_BitCount < 8) // if not past all bits.
                {
                        // Send next bit and increment bit count.
                        digitalWrite(SW_Serial_Out_Pin,
                                bitRead(SW_Serial_Out_Hold, SW_Serial_Out_BitCount++));
                }
                else // past all bits, Send Stop bit.
                {
                        digitalWrite(SW_Serial_Out_Pin, HIGH); // Set for stop bit.
                        SW_Serial_Out_State = SW_Serial_Out_Idle; // Next, look for another char.
                } // End of bit count if
                break;
        } // End of state switch

}
// This section checks for input.

switch (SW_Serial_In_State)
{
case SW_Serial_In_Wait: // State waiting for character
        // If start of start bit detected.
        if (digitalRead(SW_Serial_In_Pin) == LOW)
        {
                // Init variables
                SW_Serial_In_Hold = 0;
                SW_Serial_In_Flag = 0;  // Set to false.
                // Move to next State.
                SW_Serial_InTimer = OVER_SAMPLE - 1;
                SW_Serial_In_State = SW_Serial_In_Start;
        }
        break;
case SW_Serial_In_Start: // Start Bit wait
        if (digitalRead(SW_Serial_In_Pin) == HIGH)
        {
                SW_Serial_In_State = SW_Serial_In_Wait;
        }
        else if (SW_Serial_InTimer == 0) // end of start,
        {
                // Move to next State.
                SW_Serial_InTimer = OVER_SAMPLE;
                SW_Serial_In_State = SW_Serial_In_Bits;
                SW_Serial_In_BitCount = 0;
        }
        SW_Serial_InTimer--;
        break;
case SW_Serial_In_Bits:    // Read Bit SW_Serial_In_BitCount
        bitClear(PORTB, 2); // Debug
        if (SW_Serial_InTimer == 2) // Sample
        {
                bitSet(PORTB, 2); //Debug
                // Move bits up one
                SW_Serial_In_Hold = (SW_Serial_In_Hold >> 1) & 0x7f;
                // Save off incoming bit.
                if (digitalRead(SW_Serial_In_Pin) == HIGH)
                        SW_Serial_In_Hold |= 0x80; // set top bit
        }
        else if (SW_Serial_InTimer == 0)
        {
                SW_Serial_InTimer = OVER_SAMPLE;
                SW_Serial_In_BitCount++;
                if (SW_Serial_In_BitCount == 8)
                        SW_Serial_In_State = SW_Serial_In_Stop;
        }
        SW_Serial_InTimer--;
        break;
case SW_Serial_In_Stop:    // Stop Bit
        bitClear(PORTB, 2); // Debug
        if (SW_Serial_InTimer == 2) // Sample
        {
```

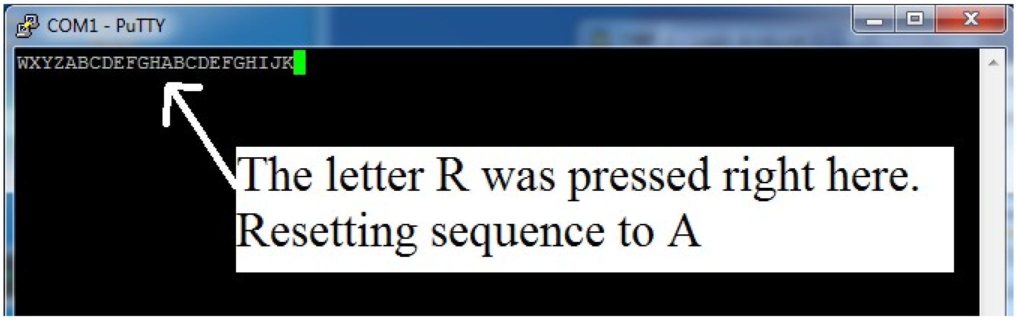
```
                    // Test for Stop bit.
                    if (digitalRead(SW_Serial_In_Pin) == HIGH)
                    {
                           bitSet(PORTB, 2); //Debug
                           // Valid stop so save data read in
                           SW_Serial_In_Buffer = SW_Serial_In_Hold;
                           // and flag that data is in.
                           SW_Serial_In_Flag = 1;  // Set to true.

                    }
                    SW_Serial_In_State = SW_Serial_In_Wait;
                    SW_Serial_InTimer = 1;
             }
             SW_Serial_InTimer--;
             break;
       }

}// End of SW_Serial_Out_ISR

// Code to set up the interrupts and data for Serial data
void SW_Serial_Initialize(int BaudRate, int pin, int In_pin)
{
       int BitTime_ms = 1000000 / (4 * BaudRate); // Compute sample time in microseconds.
       // Output pin setup.
       SW_Serial_Out_Pin = pin; // Save pin number for later use.
       pinMode(SW_Serial_Out_Pin, OUTPUT); // Set pin to output
       digitalWrite(SW_Serial_Out_Pin, HIGH); // and high or RS232-idle state.
       SW_Serial_OutTimer = 3;

       // Input pin setup.
       SW_Serial_In_Pin = In_pin; // Save pin number for later use.
       pinMode(SW_Serial_In_Pin, INPUT); // Set pin to input
       SW_Serial_InTimer = 3;

       // Set up timer to run ISR at bit time.
       Timer1.initialize(BitTime_ms);
       Timer1.attachInterrupt(SW_Serial_ISR, BitTime_ms);
       // Initialize buffer.
       SW_Serial_Out_Flag = 0;
       SW_Serial_In_Flag = 0;
}
// This function will place a character into the circular buffer.
void SW_Serial_Transmit(char ch)
{
       // Insert ch at current in pointer.
       SW_Serial_Out_Buffer = ch;
       // Advance in pointer
       SW_Serial_Out_Flag = 1;
}

int SW_Serial_Receive(char *ch)
{
       // Check to see if data has come in.
       if (SW_Serial_In_Flag)
       {
             // if it has come in, send to ch
             *ch = SW_Serial_In_Hold;
             SW_Serial_In_Flag = false;
             return 1; // returns a true for data in.
       }
       return 0; // returns a false in no data.
}

#endif
```

This is the Arduino code.

```
#include <TimerOne.h>

// Simple timer to flash LED.
#define LED_INTERVAL 500
unsigned long LedTimer;

#include "SW_Serial.h" // Header for serial code.

// Run once, To set up system
void setup()
{
	// Initialize LED timer, LED pin and clock.
	LedTimer = millis();
	pinMode(10, OUTPUT); // Debug line

	// Serial Com.
	SW_Serial_Initialize(4800, 12, 13);

} // End of setup

char SendChar = 'A';

// Code that is run continuously.
void loop()
{
	char ch;

	// LED Flashing Timer.
	if (millis() - LedTimer >= LED_INTERVAL)
	{
		// Transmit Character
		SW_Serial_Transmit(SendChar);
		// Move to next character
		SendChar++;
		// Wrap if it goes past 'Z'
		if (SendChar > 'Z')
		{
			SendChar = 'A';
		}

		LedTimer += LED_INTERVAL; // Update Timer.

	} // End of Led Timer.

	// Check for incoming serial data.
	if (SW_Serial_Receive(&ch))
	{
		// Use incoming character to reset SendChar
		if (ch == 'R')
			SendChar = 'A';

	} // End of Serial available if


} // End of loop
```

We first test the port by hooking it to a RS-232 translator, and then to the serial port on my desktop computer. It looks like the following image.

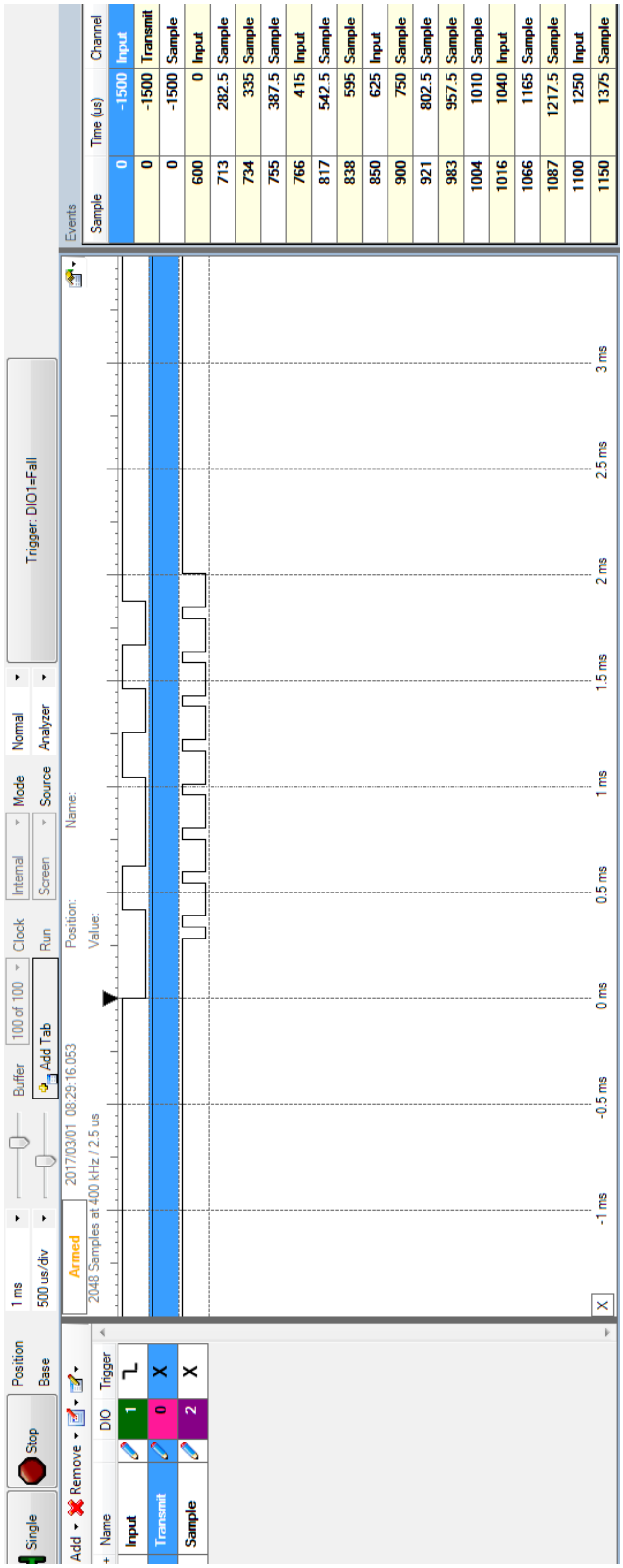
The serial cable, in the upper left corner, was attached to my desktop computer and a program called PuTTY was used to communicate to the Arduino at 4800 Baud. The results are shown here.



It should be noted that in order to validate the system, and all the counting an such, a single output pin was set each time a bit was sampled on each time the input is sampled and then cleared on the next pass through the program.

```
bitClear(PORTB, 2); // Debug
if (SW_Serial_InTimer == 2) // Sample
{
        bitSet(PORTB, 2); //Debug
        // Move bits up one
        SW_Serial_In_Hold = (SW_Serial_In_Hold >> 1) & 0x7f;
        // Save off incoming bit.
        if (digitalRead(SW_Serial_In_Pin) == HIGH)
                SW_Serial_In_Hold |= 0x80; // set top bit
```

This was tested by capturing the serial data and this extra pin (Pin 10) with the Analog Discovery. It is often hard to see the output since it happens rarely and is very short, thus a trigger was used to capture the event.



We should also verify that the output is also working so a capture of the output stream was done. The time need to send the 9 bits that we can resolve are shown in the events and required 1870 microseconds, or 207.778 microseconds. This translates to 4812.8 Baud. This is off by 0.27%, well with in tolerance.