I2C stands for - Inter-Integrated Circuit
I2C is called TWI (Two Wire Interface) in avrdude.

I2C supports upper to 5 MBaud for serial communication.

When an 8-bit ASCII character is transmitted over the I2C it follows the following series of operations.

– send a start bit,
– send the 7-bit slave address it wishes to talk to,
– send a direction bit (write(0), read(1))
– wait for/send an acknowledge bit
– send/receive data byte (8 bits)
– wait for/send an acknowledge bit
– send the stop bit

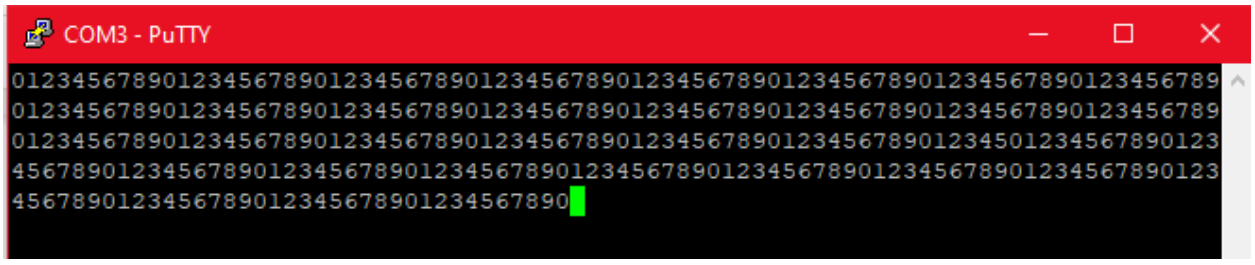
Meaning to send 8 bits of information 18 bits are sent and 2 acknowledgement bits are received.

I2C can support up to 128 simultaneous devices, these can be any number of master or slave devices.

**18/11/20**

I modified my code from the first part of the C6 Lab, so that it would also print the value to the terminal. This allowed me to see what state the program was in.

```
22  const uint8_t segments[10] = {0xFC, 0x60, 0xDA, 0xF2, 0x66, 0xB6, 0xBE, 0xE0, 0xFE, 0xE6};
23  uint8_t count = 0;
24
25  int main(void)
26  {
27      DDRA = 0xFF; //Sets portA as outputs
28      init_debug_uart0();
29
30      for (;;)
31      {
32          int display = count % 10;
33          printf("%i", display);
34          PORTA = segments[display];
35          count++;
36          _delay_ms(1000); //Delay by 1 sec
37      }
38  }
```



I then made it so that it printed a message every time there was a count overflow and then entered a new line.

```
22  const uint8_t segments[10] = {0xFC, 0x60, 0xDA, 0xF2, 0x66, 0xB6, 0xBE, 0xE0, 0xFE, 0xE6};
23  uint8_t count = 0;
24
25  int main(void)
26  {
27      DDRA = 0xFF; //Sets portA as outputs
28      init_debug_uart0();
29
30      for (;;)
31      {
32          int display = count % 10;
33          printf("%i", display);
34          while (display == 0)
35          {
36              printf("\n");
37              fprintf(stderr, "Count overflow\n\r");
38              break;
39          }
40          PORTA = segments[display];
41          count++;
42          _delay_ms(1000); //Delay by 1 sec
43      }
44  }
```

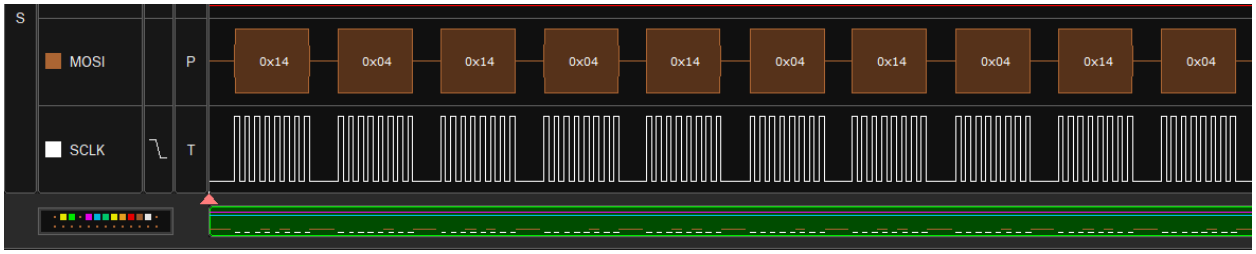
I then made it so that it would wait until an input was made from the terminal before it began counting again.

```c
const uint8_t segments[10] = {0xFC, 0x60, 0xDA, 0xF2, 0x66, 0xB6, 0xBE, 0xE0, 0xFE, 0xE6};
uint8_t count = 0;

int main(void)
{
    DDRA = 0xFF; //Sets portA as outputs
    init_debug_uart0();

    for (;;)
    {
        int display = count % 10;
        printf("%i", display);
        while (display == 9)
        {
            char str1[1];
            printf("\n");
            fprintf(stderr, "Count overflow\n\r");
            while (fscanf(stdin, "%s", str1) == 0)
            {
            }
            break;
        }
        PORTA = segments[display];
        count++;
        _delay_ms(1000); //Delay by 1 sec
    }
}
```

```c
#include <avr/io.h>
#include "spi.h"
#include "digitalPot.h"

int main(void)
{
    uint8_t i;
    init_pot();

    for(;;)
    {
        setXpot(0x00);
        setYpot(0x00);

        for(i=0; i <= 255; i++)
        {
            incXpot();
            incYpot();
        }
    }
}
```

```c
#include "spi.h"

void init_pot(void)
{
    /* TODO: Code to initiliase interface */
    init_spi_master();
}

void decXpot(void)
{
    /* TODO: Code to decrement first pot by 1 */
    uint8_t decx = 0x08;    //0000 1000
    spi_tx(decx);
    spi_rx();
}

void incXpot(void) ···

void decYpot(void) ···

void incYpot(void)
{
    /* TODO: Code to increment second pot by 1 */
    uint8_t incy = 0x14;    //0001 0100
    spi_tx(incy);
    spi_rx();
}

void setXpot(uint8_t x)
{
    /* TODO: Code to set first pot to a value between 0 and 255 */
    uint8_t setx = 0x00;    //0000 00nn nnnn nnnn
    spi_tx(setx);
    spi_rx();
    spi_tx(x);
    spi_rx();
}
```

To test the digital potentiometer I connected my logic analyse to the clock and MOSI wires and read the data packages being sent. These are what I expected as they are the same values as incXpot() and incYpot().




My connected digital potentiometer.