Maximilien Malderle
ID: 26562906
SOEN 422
LAB 2

Introduction:

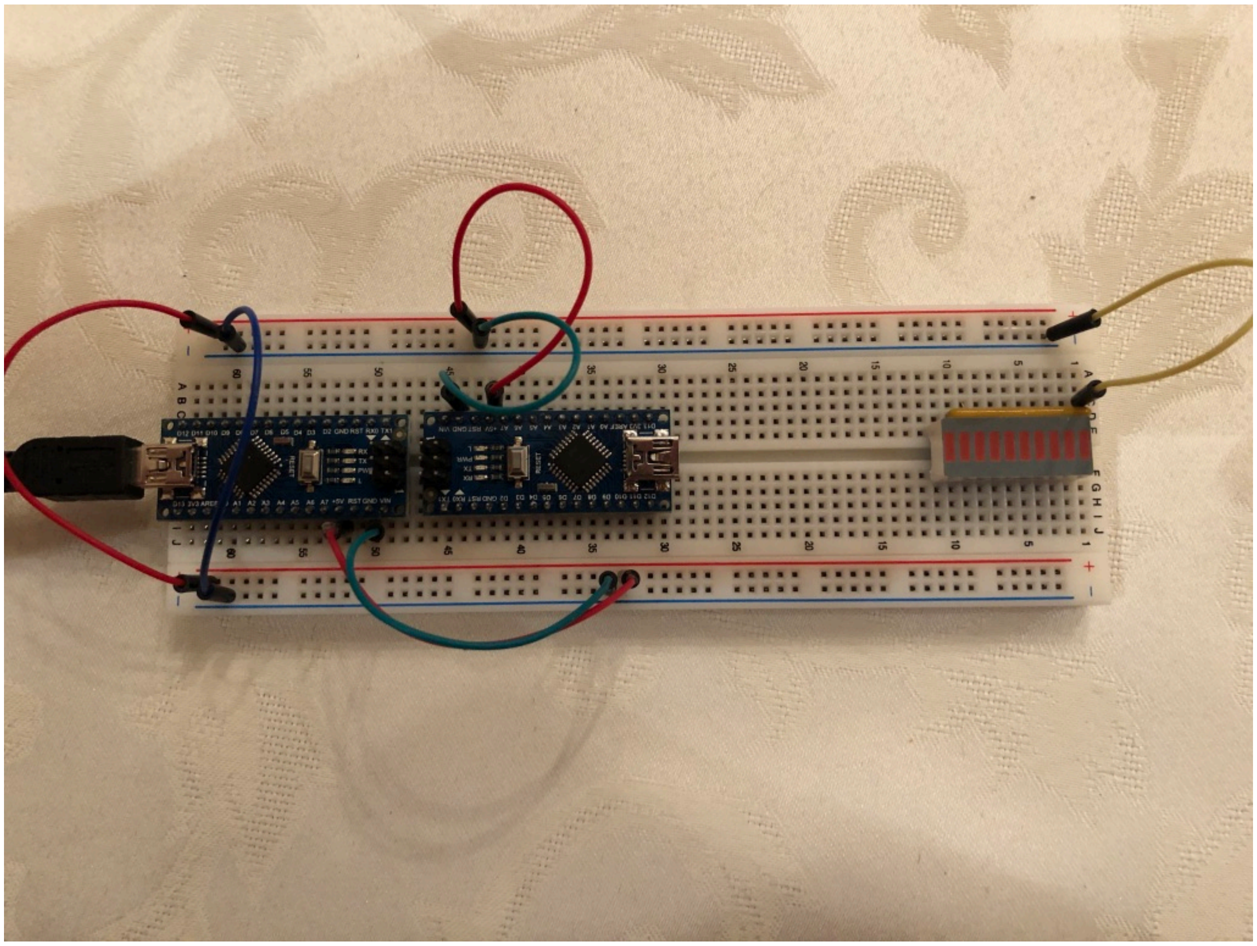
The purpose of this lab is to use the Arduino nano board to set up three different methods of communication. The first is called UART and utilizes a USB connection to communicate information from the nano to a computer or vice versa. The second method of communication is called SPI, which utilizes 4 connection points between two nano chips. SPI is a form of duplex communication i.e. it can send and receive data at the same time using a serial protocol. The last method is called I2C which utilizes a bus communication pattern. Both SPI and I2C require the creation of a master and one or multiple slaves. The first question of the lab requires a UART connection between the computer and the nano, information such as a name, will be sent to the board, where a greeting, which incorporates the received name, shall be created and sent back to the computer. The second question requires a UART connection between the computer and a nano, with an additional connection between the two Nanos. The master nano will be connected to the computer and to the slave nano, which will be connected to an IR sensor. Data from the sensor will be communicated to the master nano through SPI and the master will communicate this information to the computer through a UART connection. The third question requires the same set-up and goal as the second, but the SPI connection is replaced by a I2C connection.

Physical Resources:

For all of the questions the Arduino nano is used, for parts 2 and 3 two are used. In questions 2 and 3 there is also a capacitor and IR sensor which are required.

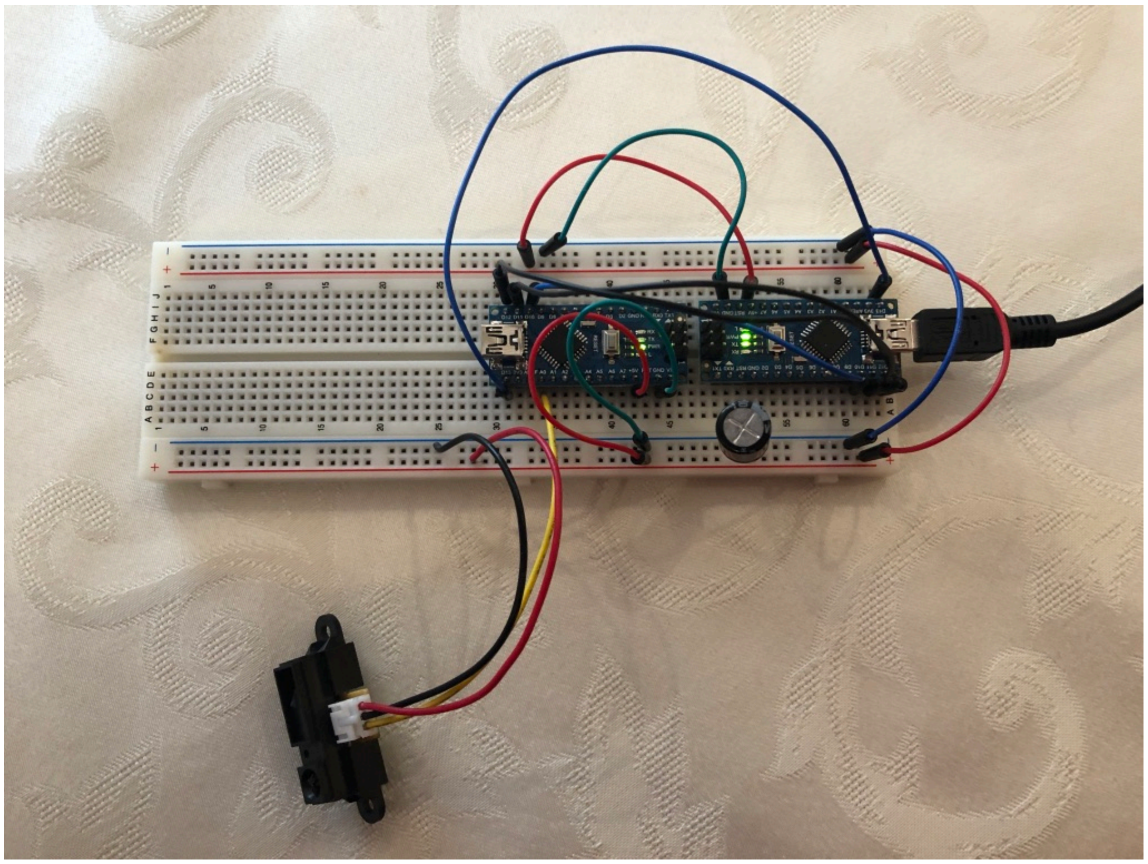
Question 1:

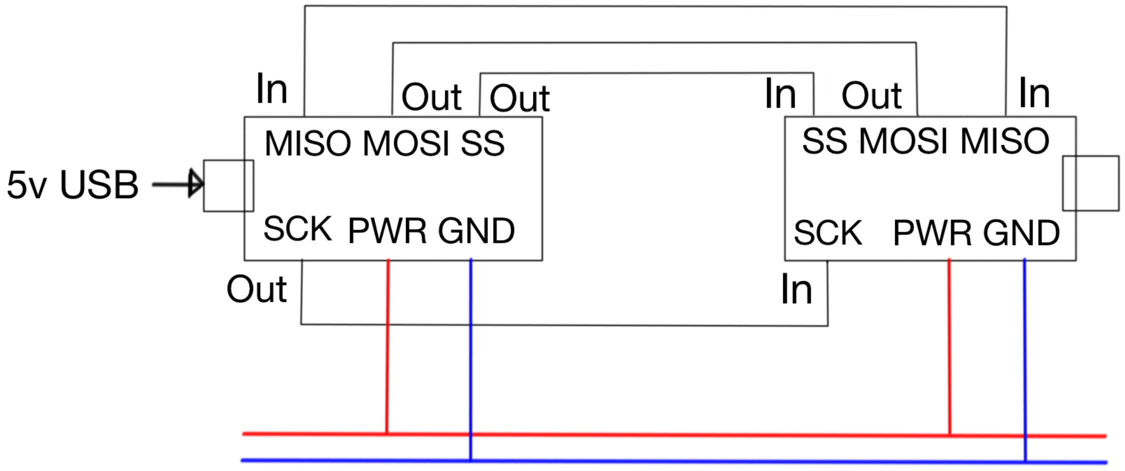
Power to the Arduino is provided by the USB port, which is connected to the computer. A ground and power is connected to the bread board. On the picture two wires are used to connect the bus with the nano's power and ground to the other side to provide power to a second nano, which will not be used in this case. There is also an LED array with a ground, which has no use other than to test the output of ports in questions 2 and 3.



Question 2:

The setup on this board is exactly the same as for question one, except for the interconnections between the boards and the IR sensor on the slave. The master has a 4 connections to the slave, each port is mapped to the same port on the other board. The master sets pins D10{PB2, SS}, D11{PB3, MOSI} and D13{PB5, SCK} as outputs and pin D12{PB4,MISO} as an input. The slave sets the pins as the opposite as the master. MOSI stands for master out slave in, MISO stands for master in slave out, SCK is for the clock used for communication and SS stands for slave select. The slave also has an IR sensor connected to pin A0{PC0} as an input. Between the rail connection and the nano connection there is a capacitor to make the signal smoother for the IR sensor.

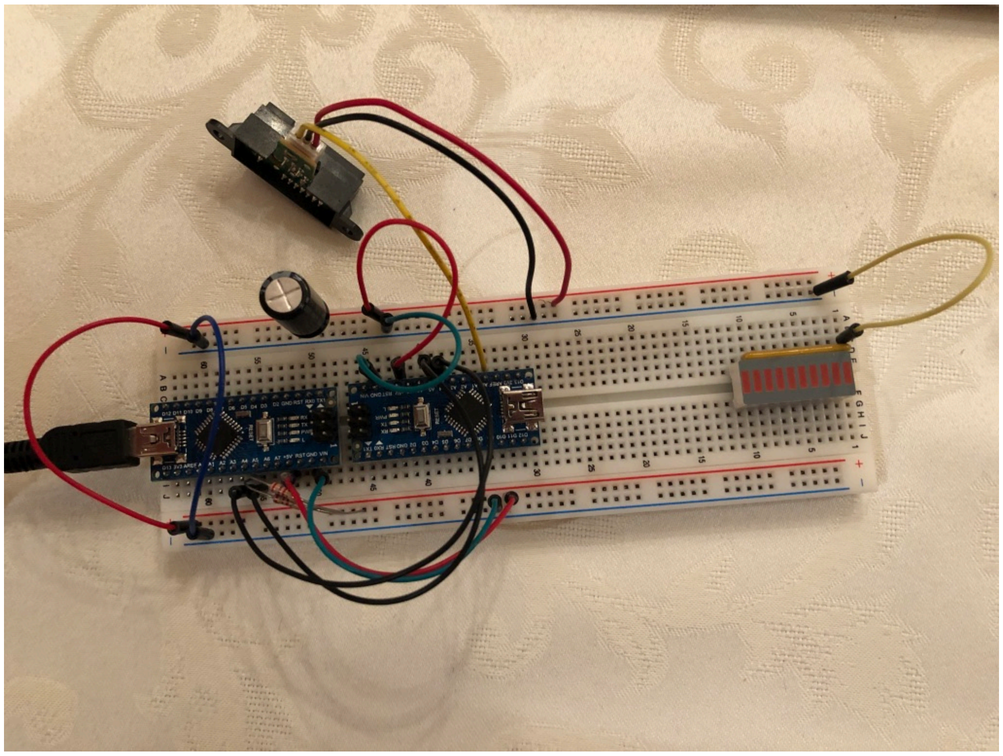


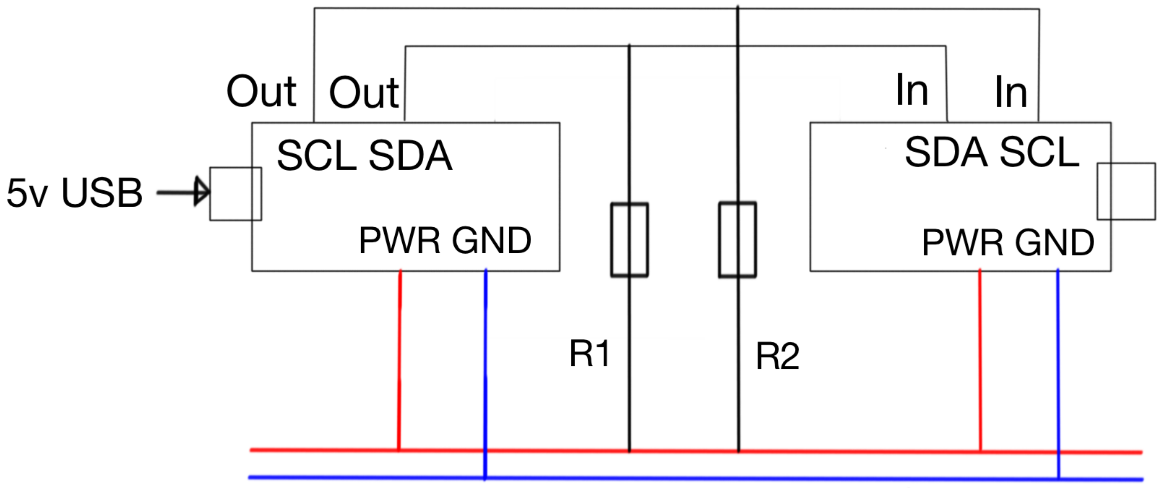


IR Sensor is Connected to Slave Pin A0 and is connected to the powerline with a capacitor between it and the source.

Question 3:

The IR sensor is setup is the same as in Question 2. Here there are two 4.7 ohm resistors inserted into the master's A5{PC5,SCL} and A4{PC4, SDA} pins, which are connected to the positive rail, these resistors act as pull-up resistors for I2C communication. There are also two connections from these pins to the corresponding pins on the slave.





IR Sensor is Connected to Slave Pin A0 and is connected to the powerline with a capacitor between it and the source.

Software Resources:

To complete this lab, only pure C code was utilized, it was compiled through the Arduino IDE, yet it is now possible to compile and load the software through the terminal. The programming was done in “Bare Metal” requiring C libraries, to allow for bit wise manipulation of the hardware registers. The functions are declared before the main in the program file, with a main which runs the majority of the functions in a indefinite loop. The CPU frequency and the BAUD rate are defined as global variables at the beginning of every program.

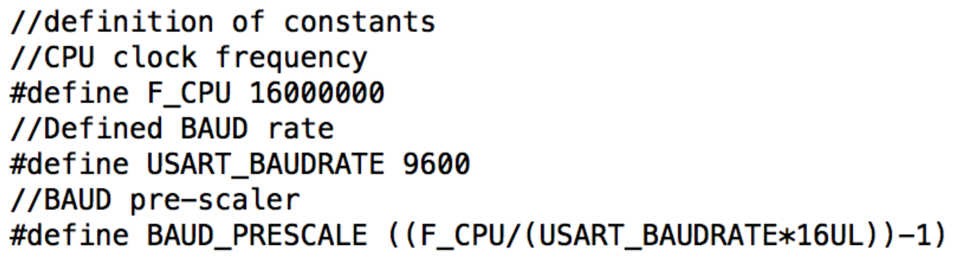
Program Snippets:

In this section snippets of code will be presented to show how certain necessary parts to the problems are solved.

Problem 1 (UART Communication):

Definitions:

These definitions are used to set key information relating to the hardware, so that the communication between the nano and the pc are in sync.



```
//definition of constants
//CPU clock frequency
#define F_CPU 16000000
//Defined BAUD rate
#define USART_BAUDRATE 9600
//BAUD pre-scaler
#define BAUD_PRESCALE ((F_CPU/(USART_BAUDRATE*16UL))-1)
```

Initiation Protocol:

This function is created to set the BAUD rate in the appropriate register, while activating the sending and receiving capabilities of the nano.

```
void init()
{
  //Set up BAUD rate in the 16 bit UBRR0 register
  //separated into two 8-bit portions UBRR0H and UBRR0L
  UBRR0H = (BAUD_PRESCALE >> 8);
  UBRR0L = BAUD_PRESCALE;

  //enable Send (RXEN0 set to 1) and Receive (TXEN0 set to 1)
  UCSR0B |= (1<<RXEN0) | (1<< TXEN0);

}
```

Receive Character Function:

Used to communicate a character from the serial Monitor to the nano.

```
//Receive Character Function
char reChar()
{
  //Wait until the message has been received (RXC0 will be 0)
  while((UCSR0A & (1<<RXC0))==0){}

  //Return the byte in the data register
  return UDR0;
}
```

Send Character Function:

Used to communicate a character to the Serial Monitor from the nano.

```
//Send Character Function
void seChar(char c)
{
  //Wait until data register is empty (UDRE0 will be 0)
  while((UCSR0A & (1<<UDRE0))==0){}

  //fill the data register with the character fed into the function
  UDR0 = c;

}
```

Send Message Function:

Used to send a string of characters to the Serial Monitor from the nano.

```
//Send Message (array of characters) Function
//accepts char pointer (point to the beginning of the array)
//accepts int to describe the length of the character array
void seMess(char* msg, int size)
{
  //loops through each char in the passed array
  for(int i=0; i<size; i++)
  {
    //waits for the data register is empty
    while((UCSR0A & (1<<UDRE0))==0){}

    //pass the specified character into the data register
    UDR0 = msg[i];
  }
}
```

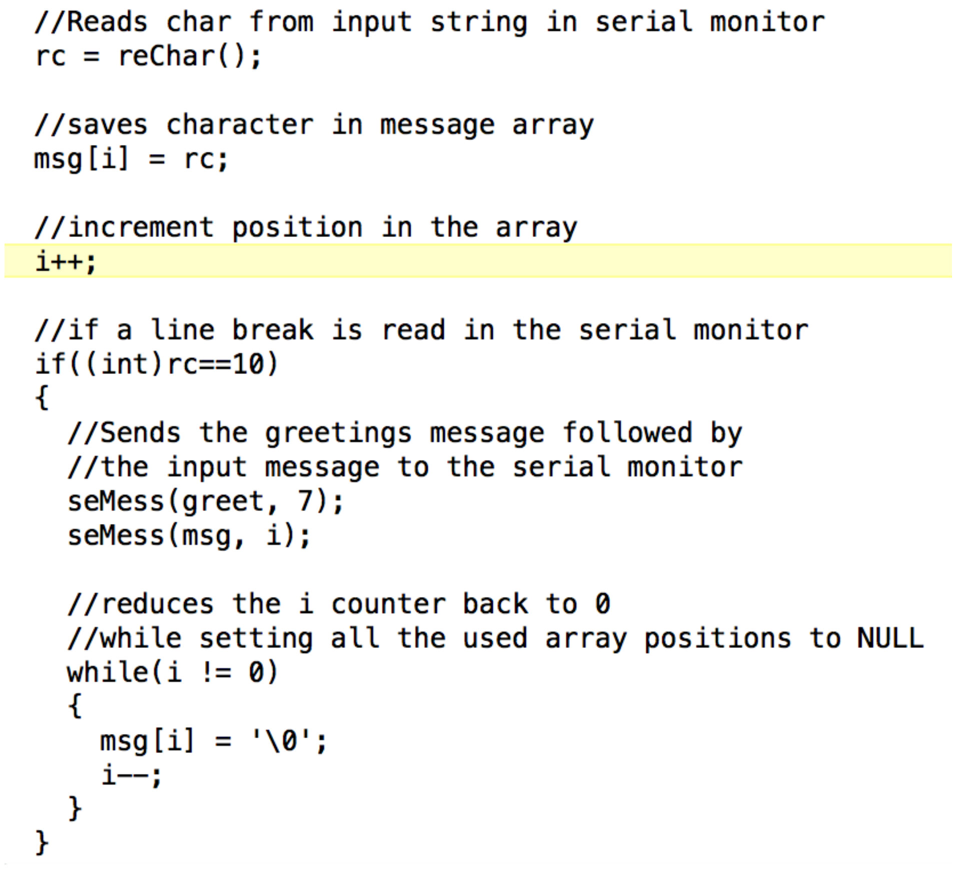
Main Function:

Variable Declarations:

```
//char set to hold received character
char rc;

//char array with capacity of 255 characters
//set to hold serial monitor inputs
char msg [255];

//set to hold greetings string
// to manipulate input string
char greet[7] = "Hello ";

//integer used to hold position
//of the current character in the input string
int i = 0;

//calls initiation function to activate Send/Receiving
//BAUD rate
init();
```

Main Operations:

The logic here is that a character is read from the serial monitor and stored into a char array named msg, after this the index is incremented. This happens until the serial monitor reads a line break character, then the accrued characters in the msg array and the saved greetings message are written to the serial monitor. Afterwards the msg array is cleared and the index is set back to 0.

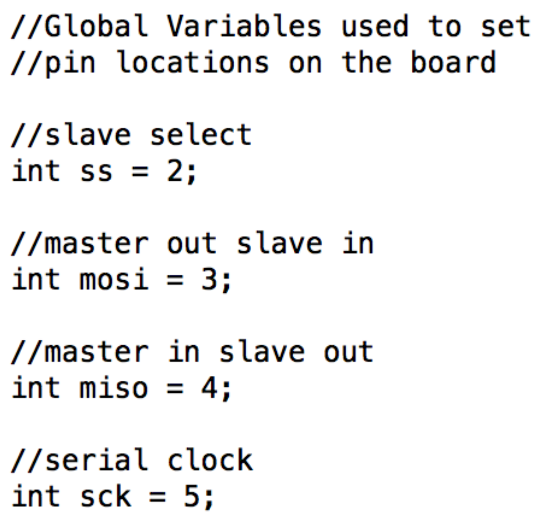


```
//Reads char from input string in serial monitor
rc = reChar();

//saves character in message array
msg[i] = rc;

//increment position in the array
i++;

//if a line break is read in the serial monitor
if((int)rc==10)
{
  //Sends the greetings message followed by
  //the input message to the serial monitor
  seMess(greet, 7);
  seMess(msg, i);

  //reduces the i counter back to 0
  //while setting all the used array positions to NULL
  while(i != 0)
  {
    msg[i] = '\0';
    i--;
  }
}
```

Problem 2 (SPI Communication):

Master:

Global Variables:

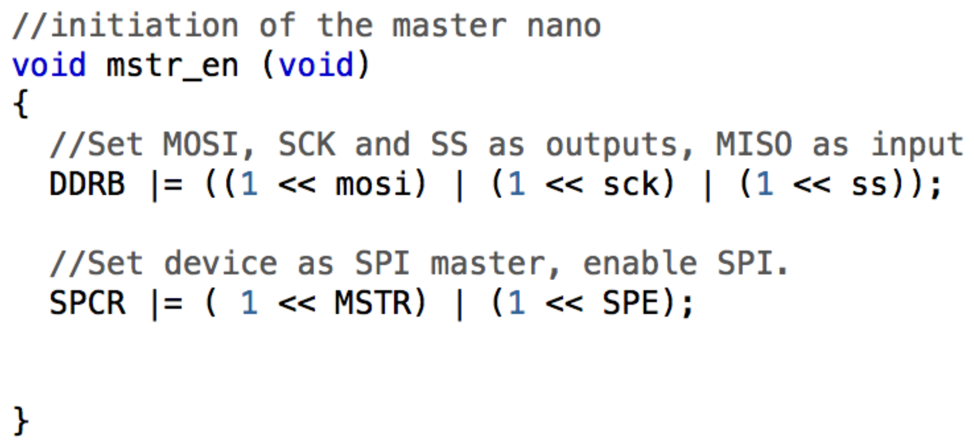


```
//Global Variables used to set
//pin locations on the board

//slave select
int ss = 2;

//master out slave in
int mosi = 3;

//master in slave out
int miso = 4;

//serial clock
int sck = 5;
```

UART Communication Initiation and Send/Receive functions from problem 1

Initialization of Master Nano:

This is used to set the master pins and the registers to a master configuration.



```
//initiation of the master nano
void mstr_en (void)
{
  //Set MOSI, SCK and SS as outputs, MISO as input
  DDRB |= ((1 << mosi) | (1 << sck) | (1 << ss));

  //Set device as SPI master, enable SPI.
  SPCR |= ( 1 << MSTR) | (1 << SPE);


}
```

Transceiver Function:

Accepts 8-bit data to send and returns 8-bit data, which was received.

```
//Transmitting and receiving function
uint8_t transceiver (uint8_t data)
{
  //Load byte of data into data transfer register
  SPDR = data;

  // Wait until interrupt flag is asserted.
  //Is asserted when transmission is complete
  while (! (SPSR & (1 << SPIF))){}

  //returns the contents of the data register
  return (SPDR);
}
```

IR_Data Array Reset Function:

The point here is to set the character array that holds the IR data back to NULL, so that no value is reused if not overwritten in the next cycle.

```
//Reset the array segments of the IR char array to NULL
void reset(uint8_t* c)
{
  c[0] = '\0';
  c[1] = '\0';
  c[2] = '\0';

}
```

IR Receiver function

Variables:

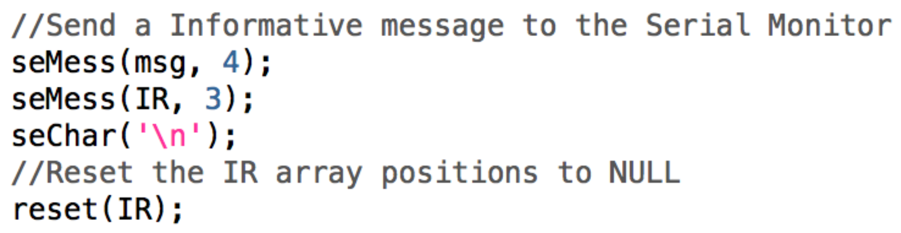
```
//Send variable
//To hold Send 8 bit char
uint8_t sd;

//Receive variable
//To hold Receive 8 bit char
uint8_t rc;

//3 8 bit char array to hold IR sensor data
uint8_t IR[3];

//4 8 bit char array to hold message intended for Serial Monitor output
uint8_t msg[4] = "IR: ";

//int to hold count position for IR array
int i = 0;
```

Work Portion:

```
while(i<3)
{
  //Master receiving IR data from Slave
  //Master sending empty 8 bit message
  rc = transceiver(sd);

  //if the received data is a digit in the range of 0-9
  if((int)rc>47 && (int)rc<58)
  {
    //Place received data into the appropriate array segment
    IR[i] = rc;
    //increment the index
    i++;
  }

}
```

The logic here is that the Transceiver function receives bytes of data from the slave and stores it in a variable, if the value is a number in the range of 0-9 it will be stored in the char array, this is done three times (Corresponding to the format of the IR data from the sensor).

Serial Monitor Message presentation:

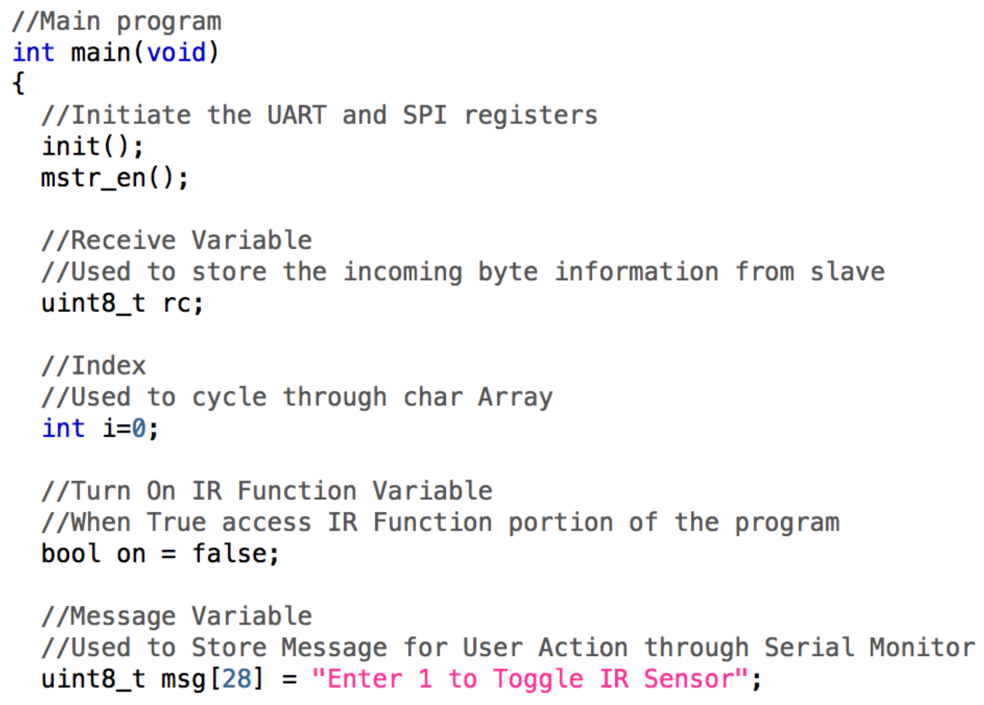


```
//Send a Informative message to the Serial Monitor
seMess(msg, 4);
seMess(IR, 3);
seChar('\n');
//Reset the IR array positions to NULL
reset(IR);
```

The logic here, is that there are 3 pieces of data that are received from the IR sensor. Once the index reaches 3 the accumulated data from the sensor is presented with an additional message to add context. Afterwards the array containing the IR data is reset alongside the index.

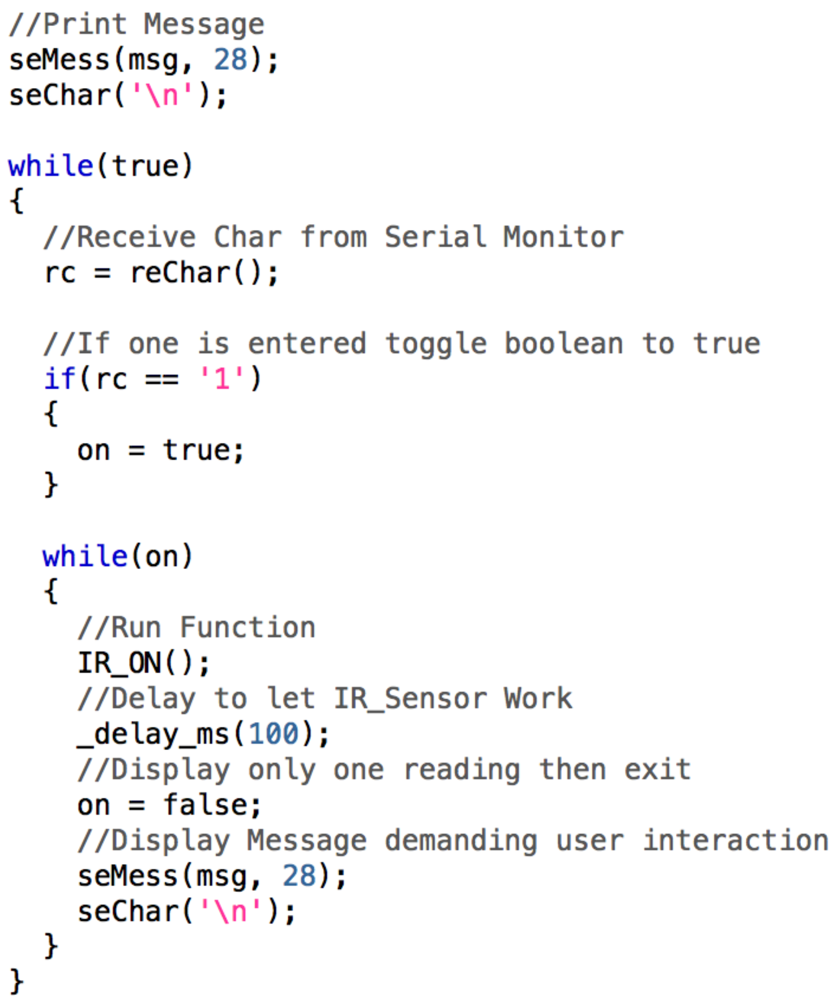
Main Program:

Initialization and Variables:



```
//Main program
int main(void)
{
  //Initiate the UART and SPI registers
  init();
  mstr_en();

  //Receive Variable
  //Used to store the incoming byte information from slave
  uint8_t rc;

  //Index
  //Used to cycle through char Array
  int i=0;

  //Turn On IR Function Variable
  //When True access IR Function portion of the program
  bool on = false;

  //Message Variable
  //Used to Store Message for User Action through Serial Monitor
  uint8_t msg[28] = "Enter 1 to Toggle IR Sensor";
```

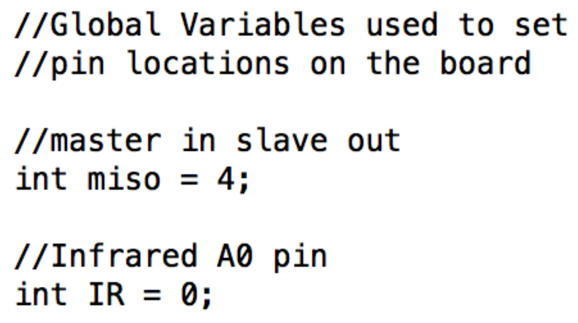
Working Portion:



```
//Print Message
seMess(msg, 28);
seChar('\n');

while(true)
{
  //Receive Char from Serial Monitor
  rc = reChar();

  //If one is entered toggle boolean to true
  if(rc == '1')
  {
    on = true;
  }

  while(on)
  {
    //Run Function
    IR_ON();
    //Delay to let IR_Sensor Work
    _delay_ms(100);
    //Display only one reading then exit
    on = false;
    //Display Message demanding user interaction
    seMess(msg, 28);
    seChar('\n');
  }
}
```

The logic here is that a message is displayed to the user through the Serial Monitor asking for the user to input a ‘1’ if they wish to have one value from the IR Sensor to be presented. If one is read the IR_ON function is run once and the message is displayed again, continuing the process.

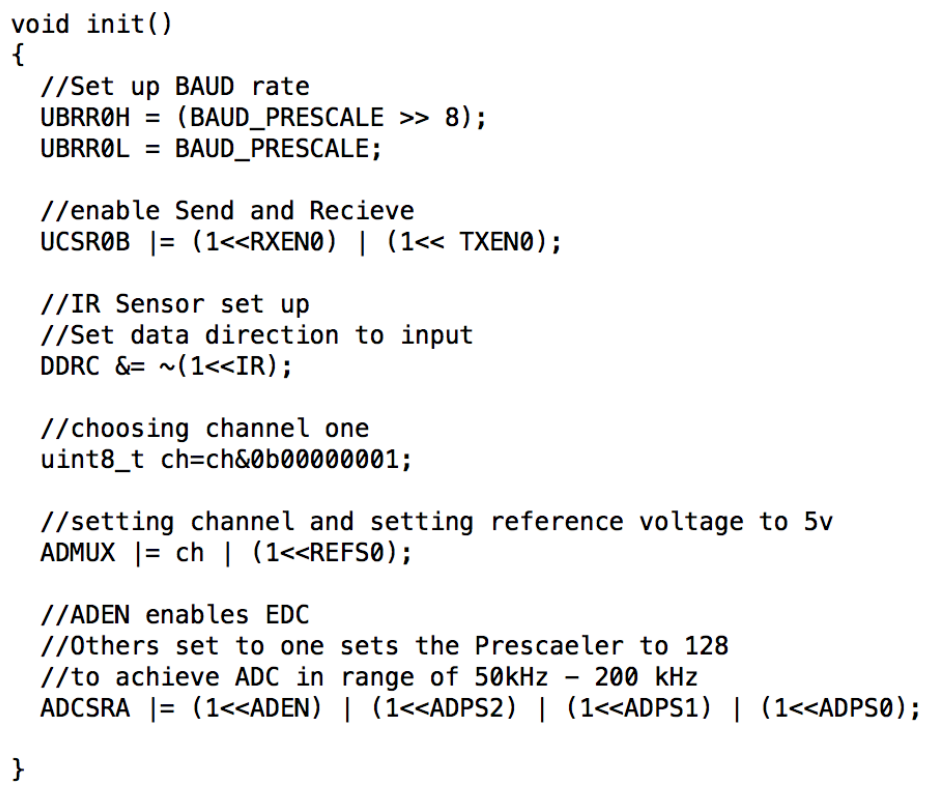
Slave:
Global Variables:



```
//Global Variables used to set
//pin locations on the board

//master in slave out
int miso = 4;

//Infrared A0 pin
int IR = 0;
```

Baud rate, Send/Receive, IR_Sensor Initialization:

The BAUD rate and the Send/Receive activation are for UART communication, which is not utilized by the slave. The initialization of the IR sensor is the same as in lab 1b.



```
void init()
{
  //Set up BAUD rate
  UBRR0H = (BAUD_PRESCALE >> 8);
  UBRR0L = BAUD_PRESCALE;

  //enable Send and Recieve
  UCSR0B |= (1<<RXEN0) | (1<< TXEN0);

  //IR Sensor set up
  //Set data direction to input
  DDRC &= ~(1<<IR);

  //choosing channel one
  uint8_t ch=ch&0b00000001;

  //setting channel and setting reference voltage to 5v
  ADMUX |= ch | (1<<REFS0);

  //ADEN enables EDC
  //Others set to one sets the Prescaeler to 128
  //to achieve ADC in range of 50kHz - 200 kHz
  ADCSRA |= (1<<ADEN) | (1<<ADPS2) | (1<<ADPS1) | (1<<ADPS0);

}
```

Initialization of Slave Nano:

```
//Initialization of a Slave nano
void slv_en (void)
{
  // Set MISO as output. MOSI, SCK and SS are set to inputs
  DDRB |= (1 << miso);

  // Set device as SPI slave just by enabling SPI
  SPCR |= (1 << SPE);
}
```

Transceiver Function:

Same as the Master function.

```
//Transmitting and receiving function
uint8_t transceiver (uint8_t data)
{
  //Load byte of data into data transfer register
  SPDR = data;


  // Wait until interrupt flag is asserted
  //Is asserted when transmission is complete
  while (! (SPSR & (1 << SPIF))){}

  //returns the contents of the data register
  return (SPDR);
}
```

IR Sensor Function:

IR setup is very similar to that in Lab 1b. The only difference is that an integer cannot be sent out in an 8-bit register. Therefor the integer is broken into 3 portions the 100, 10 and 1 and is converted into a char which is stored in a 3-index array.

```
//IR_Sensor function accepts a pointer to a 8-bit char
void IR_Sensor(uint8_t* msg)
{
  //ADC setting up conversion
  ADCSRA|=(1<<ADSC);

  //waiting for conversion to complete
  while(!(ADCSRA & (1<<ADIF))){};

  //conversion is complete
  ADCSRA|=(1<<ADIF);

  //Separates the 100, 10 and 1
  //into three categories
  int hun = (int)ADC/100;
  int ten = (int)(ADC/10)%10;
  int one = (int)ADC%10;

  //store the int as an 8-bit char into
  //the passed array
  msg[0] = hun + '0';
  msg[1] = ten + '0';
  msg[2] = one + '0';

}
```

Main Program:

Initialization of Variables:

```
//Enables UART, IR and Slave
init();
slv_en();

//Send Variable
//Used to store Send infromation
uint8_t sd;

//Receive Variable
//Used to store Receive information
uint8_t rc;

//Infra-Red Data
//Used to store IR_Sensor data
uint8_t IR_data[3];
```

Working Section:

The logic here is that the IR_Sensor function accepts the 3-char array and fills it with IR data. Afterwards the program cycles through the array and sends the char one by one from the slave to the master.

```
while(true)
{
  //Receives Data from IR_Sensor and breaks it down
  //and stores it in the 3-char array
  IR_Sensor(IR_data);

  //Delay to let IR Sensor work
  _delay_ms(100);

//Cycles through the three positions
//of the array to send to Master
  for(int i=0; i<3;i++)
  {
    rc = transceiver(IR_data[i]);
    _delay_ms(100);
  }

  //Let the IR Sensor work
  _delay_ms(100);
}
```

Problem 3 (I2C Communication):

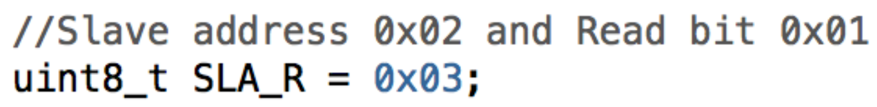
MASTER:

Does not work properly, will present current configuration.

Definitions:

```
//definitions
#define F_CPU 16000000
#define USART_BAUDRATE 9600
#define BAUD_PRESCALE ((F_CPU/(USART_BAUDRATE*16UL))-1)
```

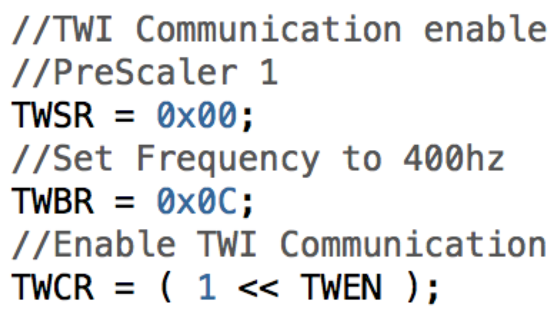
Global Variables:



```
//Slave address 0x02 and Read bit 0x01
uint8_t SLA_R = 0x03;
```

Stores the Slaves address plus read bit as a 8-bit variable.

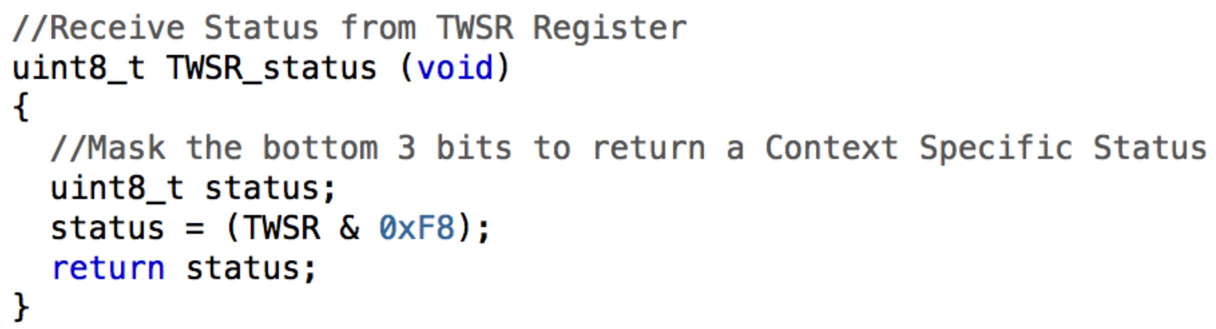
Initialization protocol:



```
//TWI Communication enable
//PreScaler 1
TWSR = 0x00;
//Set Frequency to 400hz
TWBR = 0x0C;
//Enable TWI Communication
TWCR = ( 1 << TWEN );
```

Baud and Send/Receive are for UART communication. The pre-scaler is initialized to 1 by setting the TWSR register. The frequency of 400 hz corresponds to the hex variable 0C which represents 12, which is stored in the TBWR register. TWI communication is enabled by setting the TWEN bit in the TWCR register.

TWI Status Function:



```
//Receive Status from TWSR Register
uint8_t TWSR_status (void)
{
  //Mask the bottom 3 bits to return a Context Specific Status
  uint8_t status;
  status = (TWSR & 0xF8);
  return status;
}
```

TWSR register is where statuses are written when certain conditions have been fulfilled. The (TWSR & 0xF8) makes sure to mask the three least significant bits, which are used for another purpose.

Start Condition:

This is used to signal the beginning of a communication.

```
//Sends a start Condition over the bus to the slave
void START()
{
  //TWINT Set the Interrupt flag
  //TWSTA Set the Start condition
  //TWEN set TWI enable
  TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

  //Wait for TWI Interrupt Flag to be set
  //indicating the end of a process
  while((TWCR & (1<<TWINT))==0) {}
}
```

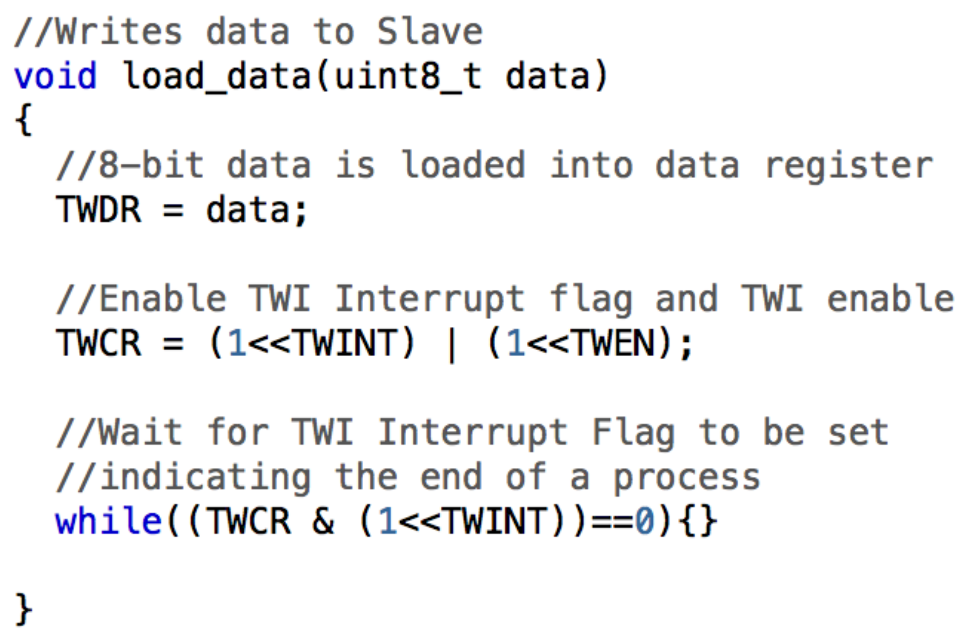
Load Address:

This is used to load the Slave address into the data register while setting the proper bits.

```
//Used to deliver the address and read bit to the slave
void load_addr()
{
  //load slave address
  TWDR = SLA_R;

  //Enable TWI Interrupt flag and TWI enable
  TWCR = (1<<TWINT) | (1<<TWEN);

  //Wait for TWI Interrupt Flag to be set
  //indicating the end of a process
  while((TWCR & (1<<TWINT))==0){}
}
```

Load Data:

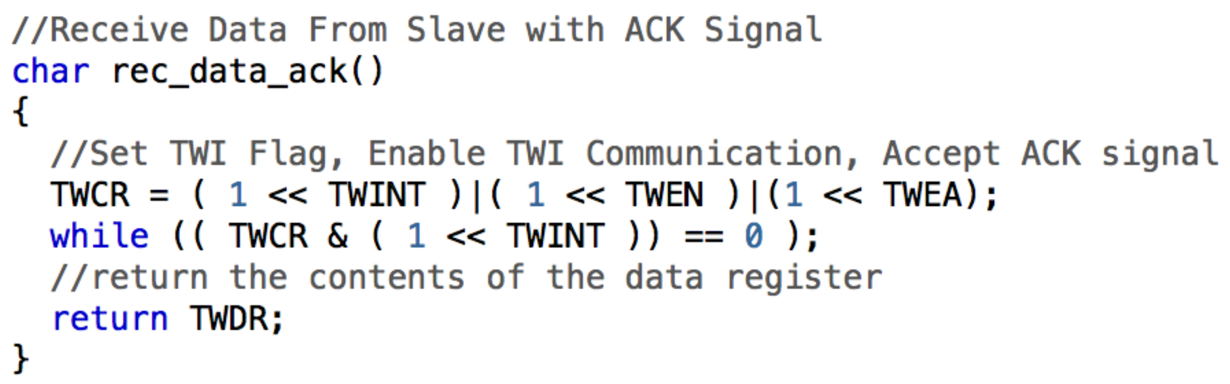
This is used to load the data into the data register while setting the Interrupt flag and enabling TWI communication, then waiting for the TWINT flag to be set to 0, indicating a that the transfer is complete.



```
//Writes data to Slave
void load_data(uint8_t data)
{
  //8-bit data is loaded into data register
  TWDR = data;

  //Enable TWI Interrupt flag and TWI enable
  TWCR = (1<<TWINT) | (1<<TWEN);

  //Wait for TWI Interrupt Flag to be set
  //indicating the end of a process
  while((TWCR & (1<<TWINT))==0){}

}
```

Receive Data ACK:

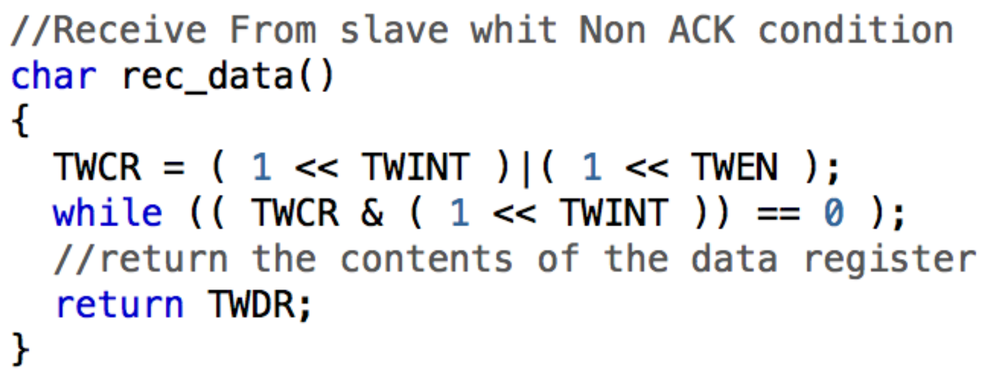
This is used to confirm that the data was received and returns the contents of the data register. The TWCR register has the TWINT, TWEN and TWEA bits are set to one, to enable TWI communication, set the Interrupt Flag and to acknowledge the ACK signal.



```
//Receive Data From Slave with ACK Signal
char rec_data_ack()
{
  //Set TWI Flag, Enable TWI Communication, Accept ACK signal
  TWCR = ( 1 << TWINT )|( 1 << TWEN )|(1 << TWEA);
  while (( TWCR & ( 1 << TWINT )) == 0 );
  //return the contents of the data register
  return TWDR;
}
```

Receive Data Non-ACK:

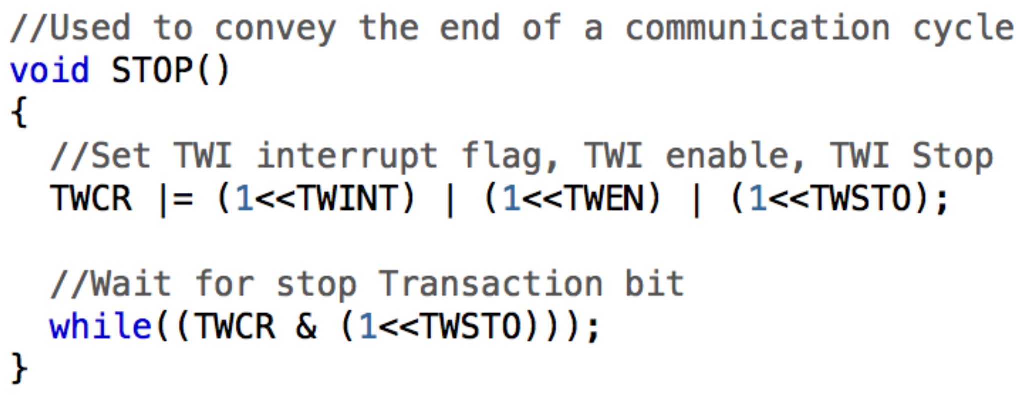
This is used to receive a message from the slave in an NACK condition, signaling the end of a transmission.



```
//Receive From slave whit Non ACK condition
char rec_data()
{
  TWCR = ( 1 << TWINT )|( 1 << TWEN );
  while (( TWCR & ( 1 << TWINT )) == 0 );
  //return the contents of the data register
  return TWDR;
}
```

Stop Conditions:

This is used to convey the end of a communication cycle, by changing the TWCR register bits and by changing the hardware electrical flow.



```
//Used to convey the end of a communication cycle
void STOP()
{
  //Set TWI interrupt flag, TWI enable, TWI Stop
  TWCR |= (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);

  //Wait for stop Transaction bit
  while((TWCR & (1<<TWSTO)));
}
```

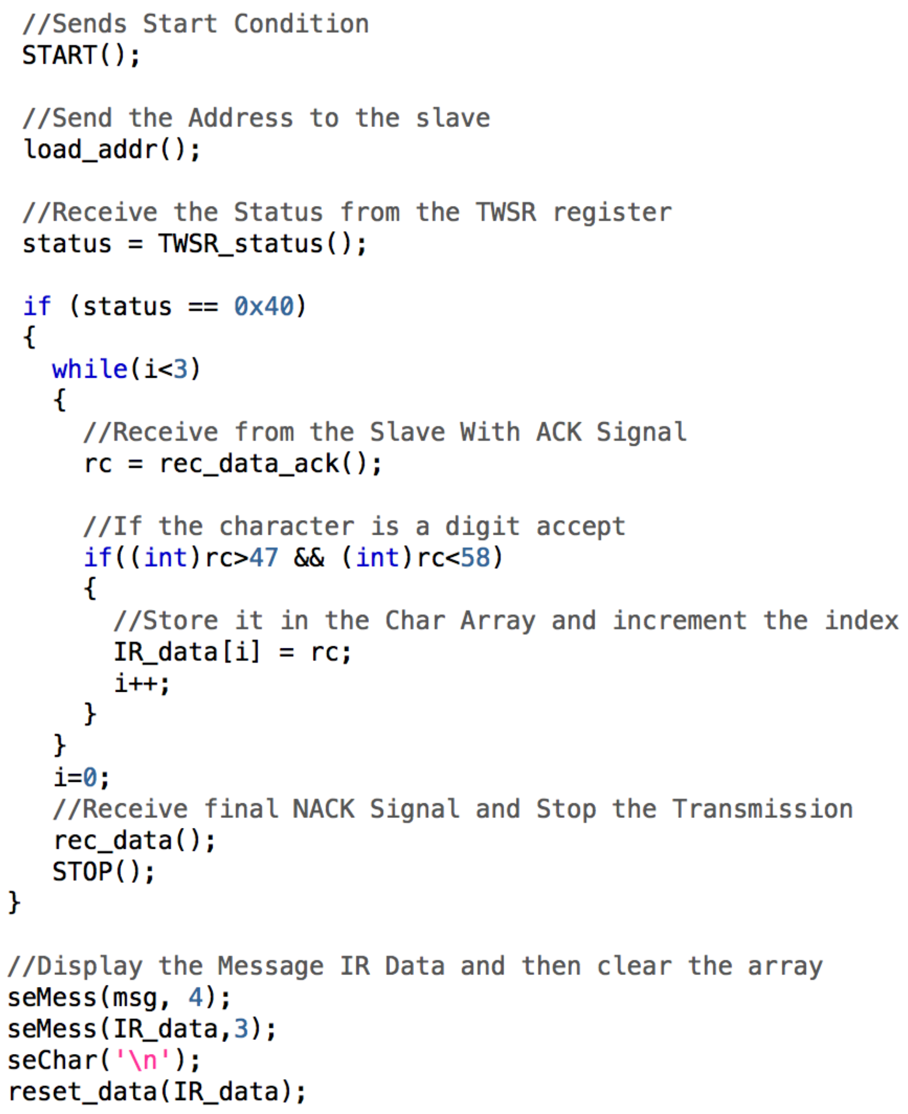
IR Function:

This function is used to create a communication link between the master and the slave, to then send the master Infra-Red data.

Variables:

```
//IR Sensor Function creates communication with the Slave
//Return the value of the Sensor
void IR_ON()
{
  //used for the while loop to fill the char array for IR data
  int i=0;

  //Receive variable Holds the contents of the Slave Transmission
  uint8_t rc;

  //Variable to hold status from TWSR Register
  uint8_t status;

  //Data Array to hold the IR Sensor Data
  uint8_t IR_data[3];

  //Message to display to Serial Monitor
  uint8_t msg[4] = "IR: ";
```

Working Portion:



```
//Sends Start Condition
START();

//Send the Address to the slave
load_addr();

//Receive the Status from the TWSR register
status = TWSR_status();

if (status == 0x40)
{
  while(i<3)
  {
    //Receive from the Slave With ACK Signal
    rc = rec_data_ack();

    //If the character is a digit accept
    if((int)rc>47 && (int)rc<58)
    {
      //Store it in the Char Array and increment the index
      IR_data[i] = rc;
      i++;
    }
  }
  i=0;
  //Receive final NACK Signal and Stop the Transmission
  rec_data();
  STOP();
}

//Display the Message IR Data and then clear the array
seMess(msg, 4);
seMess(IR_data,3);
seChar('\n');
reset_data(IR_data);
```

The logic here is that the START is sent to the slave, to then load the Slave Address and Read bit to the slave, which then returns an ACK which sets the TWSR result to 0x40. Afterwards the received byte is stored in a variable, tested for correctness and then stored in a char array. After the last byte is received, the NACK is read and the STOP condition is enacted. Afterwards the Information is presented to the user through UART communication.

Main Program:

Variables and Initialization:

```
//Main function to Get IR Data On Demand
int main(void)
{
  //Initialize UART and TWI
  init();

  //Receive Character Hold Character From Serial Monitor
  uint8_t rc;

  //Boolean to See if the IR data should be displayed
  bool on = false;

  //Holds Message to prompt user
  uint8_t msg[28] = "Enter 1 to Toggle IR Sensor";
```

Working portion

The logic here is exactly the same as for the SPI main program.

```
//Writes the message to the Serial Monitor
seMess(msg, 28);
seChar('\n');

while(1)
{
  //Recieves the Character From the Serial Monitor
  rc = reChar();

  if(rc == '1')
  {
    //Sets the IR Bool variable to true
    on = true;
  }

  //Returns IR data on Command
  while(on)
  {
    //Creates the communication to Slave and returns
    IR_ON();
    //Let the IR Sensor do Work
    _delay_ms(100);
    //Sets bool to false to display only one IR value
    on = false;
    //Asks user for prompt again
    seMess(msg, 28);
    seChar('\n');

  }
}
```

SLAVE:

Everything is the same other than the Initialization of the salve, the loading of data and the Main program.

Global Variables:

This sets the Slave address to 1 with an additional 0 padded on the right.

```
//Slave Address with a leftward shift to fill 7 most significant bits
uint8_t SLA_A = 0x02;
```

Initialization:

Here the pre-scaler is set to 1 by storing 0x00 in the TWSR Register. The frequency is set to 400 hz by loading 0x0C in the TWBR register. The slave address is saved in the TWAR address register. The enable and ACK bits are set to 1 in the TWCR register.

```
//Slave Initialization with Prescaler 1
TWSR = 0x00;
//Frequency of 400 hz
TWBR = 0x0C;
//Load the address into the address register
TWAR = SLA_A ;
//Enable ACK and TWI communication
TWCR = ( 1 << TWEA ) | ( 1 << TWEN );
```

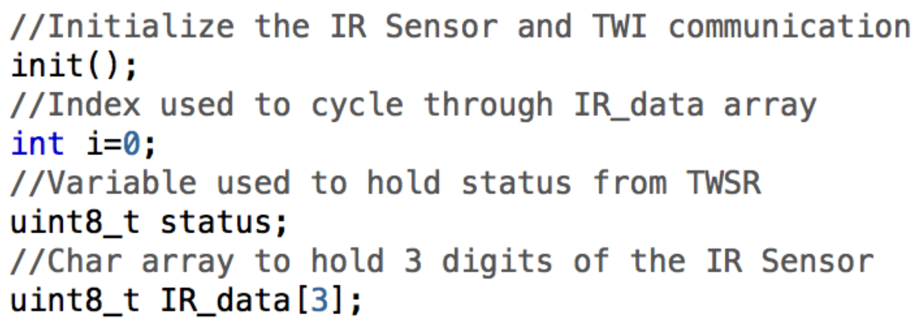
Load Data:

The function is the same as in the Master other than the TWEA bit is set to 1 in the TWCR register.

```
//Function used to transmit data to master
void load_data ( uint8_t data)
{
  //load data into Data Register
  TWDR = data;
  //Enable Interrupt flag, TWI communication and ACK bit
  TWCR = ( 1 << TWINT ) | ( 1 << TWEN ) | ( 1 << TWEA );
  //Wait for the Interrupt flag to set
  while (( TWCR & ( 1 << TWINT )) == 0 );
}
```

Main Program:

Variables and Initiation:



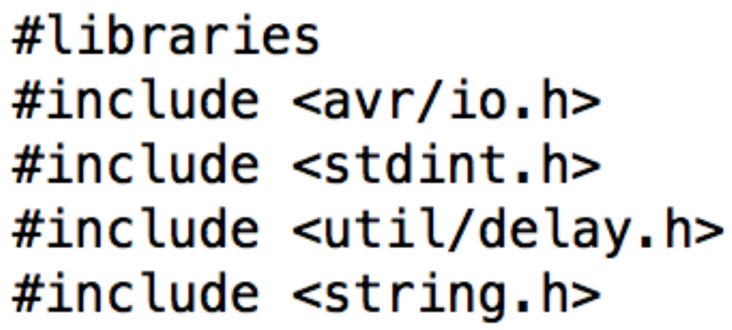
```
//Initialize the IR Sensor and TWI communication
init();
//Index used to cycle through IR_data array
int i=0;
//Variable used to hold status from TWSR
uint8_t status;
//Char array to hold 3 digits of the IR Sensor
uint8_t IR_data[3];
```

Working Portion:

The logic here is that the ACK bit is set to one, making sure that when the address and read bit are received that the Slave can signal an ACK to the master. Thereafter the IR data is stored in a char array. After the data is sent one byte at a time the ACK is set to 0 to make sure the master receives the NACK condition. The continue is useless, but the program does not work without it for some reason.

```
//Enable ACK Transmission to trigger 0xA8 condition when the
//address is received from master
TWCR |= ( 1 << TWEA );

//Fill the IR_data array with Sensor data
IR_Sensor(IR_data);
//Wait to let Sensor work
_delay_ms(100);

//Receive status form TWSR
//Address has been received transmission is enabled
status = TWSR_status();

if (status == 0xA8)
{
  //cycle through the IR data array
  while(i<3)
  {
    //Transmit one of the bytes at a time
    load_data(IR_data[i]);
    _delay_ms(100);
    //increment index
    i+=1;
  }
  //reset index
  i=0;

  //Send NACK signal to master signifying end of transmission
  TWCR &= ~( 1 << TWEA );
}
else
{
  //NO IDEA DOES NOT WORK WITHOUT ???
  continue;
}
//Delay to let the IR Sensor work
_delay_ms(100);
```

Reference Code Common:



```
#libraries
#include <avr/io.h>
#include <stdint.h>
#include <util/delay.h>
#include <string.h>
```

Avr/io.h is used so that the C language can set all the register bits.

Stdint.h allows for the bool data type.

Util/delay.h is included to give the user the ablity to call _*delay*_ms() functions, creating a waiting condition in the program.

String.h is used to utilize the string variable. Most of the time char* var is used in the place of a string variable.

Conclusion/Discussion:

This lab is focused on communication, but also builds on the previous lab. UART communication is used to replace the Serial library that was used for pc nano USB communication. The second goal of this lab is to create communication channels between two nanos, through either SPI or I2C protocols. SPI utilizes 4 pin connections to establish the channel between the slave and master nano. I2C utilizes a bus communication channel by connecting the master and slave together through SCL and SDA pins. Communication speed in I2C exceed those available through SPI. All this can be done without any software packages, everything is done through bare metal programming.