CSE211s:Introduction to Embedded Systems

Lect. #6: Communication Systems

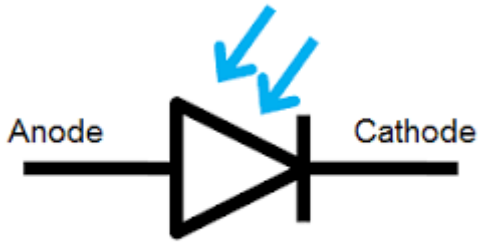
Ahmed M. Zaki

IR Communication (IR LED)

- An Infrared light-emitting diode (IR LED) is a special purpose LED emitting infrared rays ranging from 700 nm to 1 mm wavelength.
- Different IR LEDs may produce infrared light of differing wavelengths, just like different LEDs produce light of different colors.
- The appearance of IR LED is same as a common LED. Since the human eye cannot see the infrared radiations, it is not possible for a person to identify if an IR LED is working.
- A camera on a cell phone camera solves this problem. The IR rays from the IR LED in the circuit are shown in the camera.

IR Communication (IR Sensor/ Photodiode)

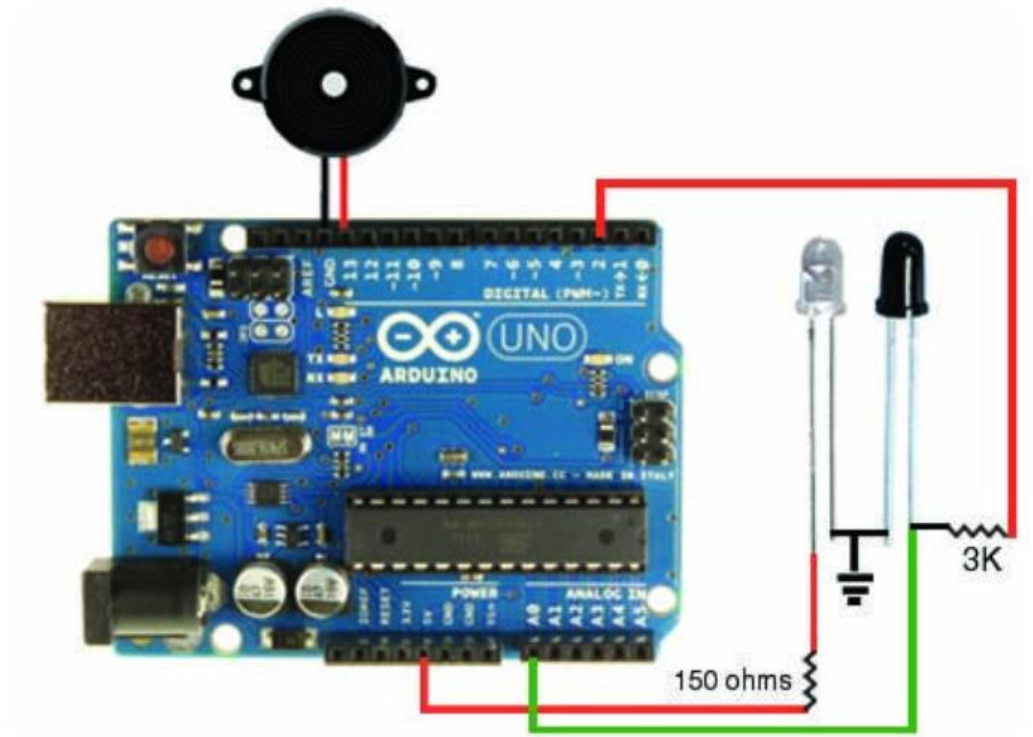
- An IR sensor is an electronic device that detects IR radiation falling on it. Proximity sensors (used in touchscreen phones and edge avoiding robots), contrast sensors (used in line following robots) and obstruction counters/sensors (used for counting goods and in burglar alarms) are some applications involving IR sensors.



Photodiode symbol

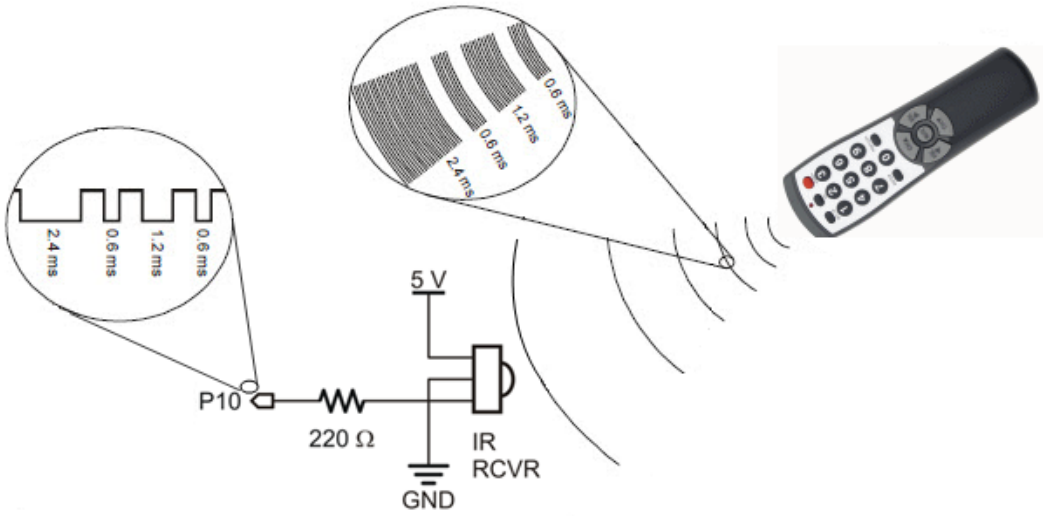
IR Communication

- IR LED and Photodiode can be used to detect obstacles

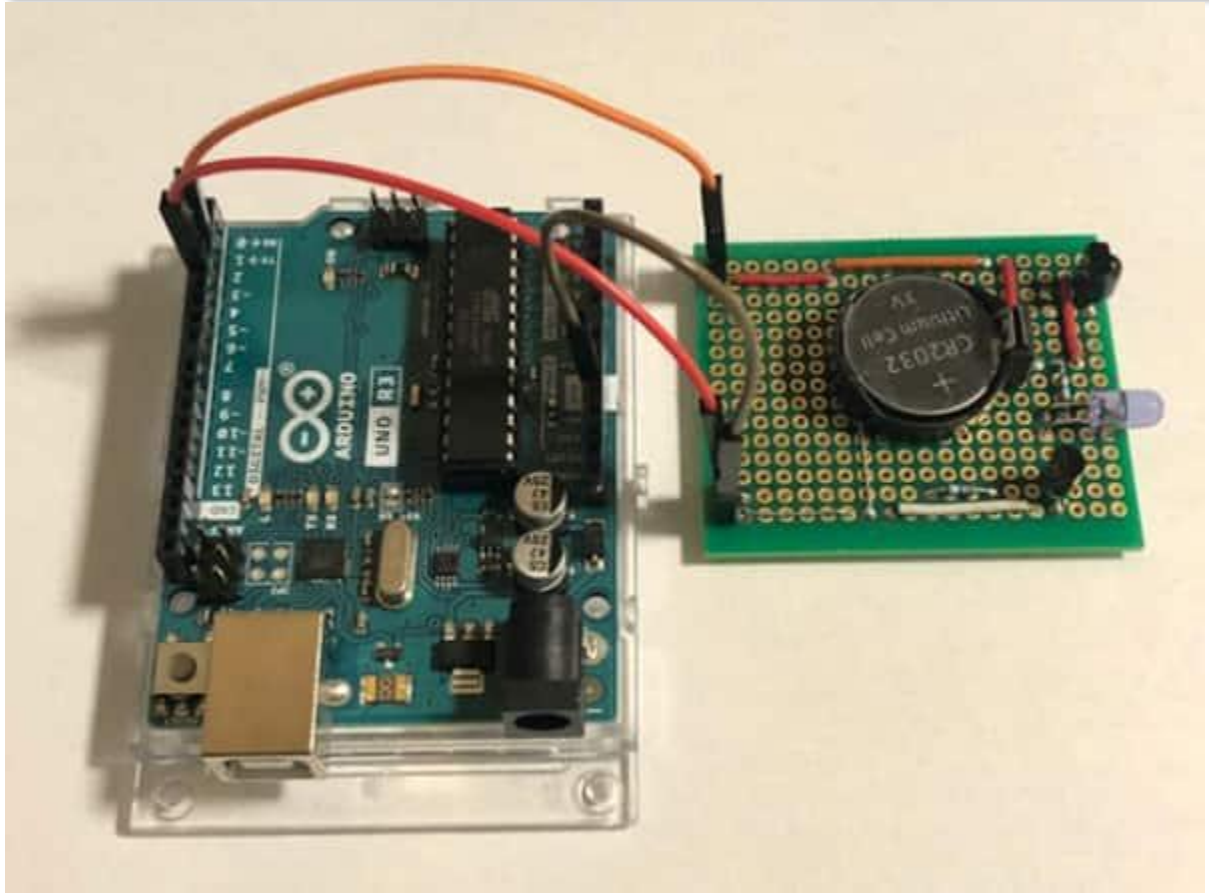


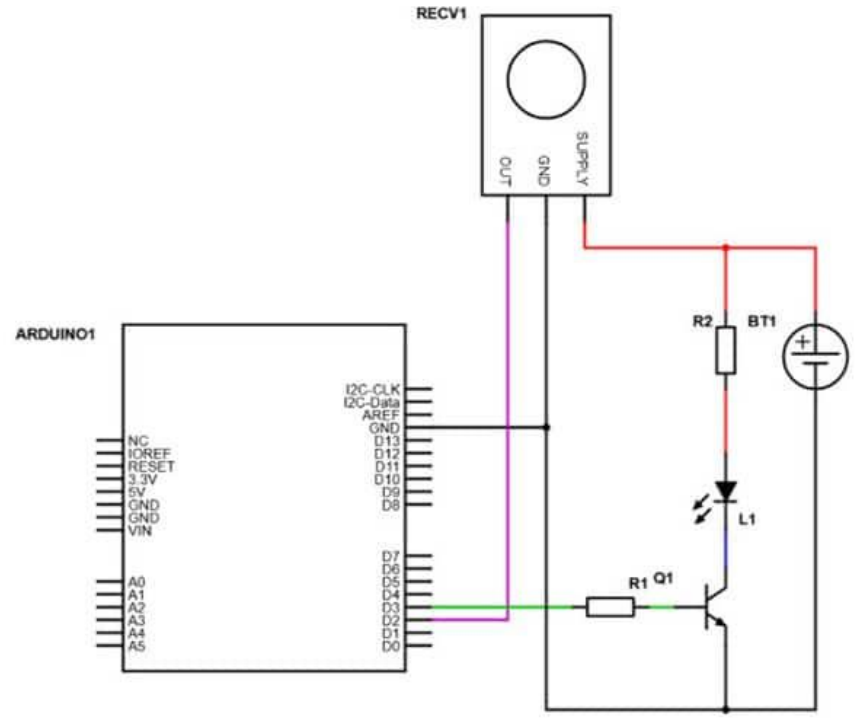
IR Remote controller

- **IR Remote Signals**
  - The remote flashes its IR LED on/off very rapidly—at 38.5 kHz—for certain periods of time, with periods of off-time in between.
  - The combinations of these pulses are received by the microcontroller



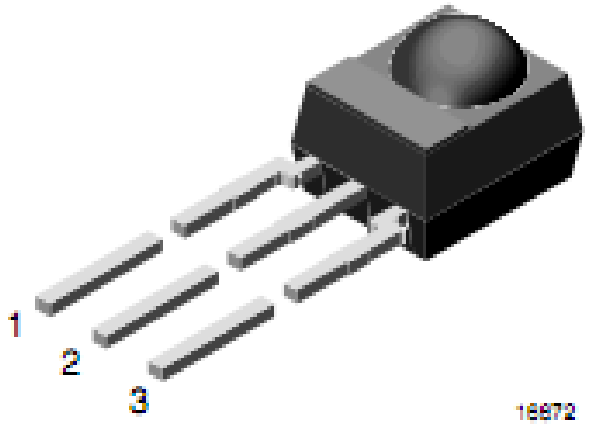
IR Remote controller (connection)





IR Remote controller (IR Receiver)

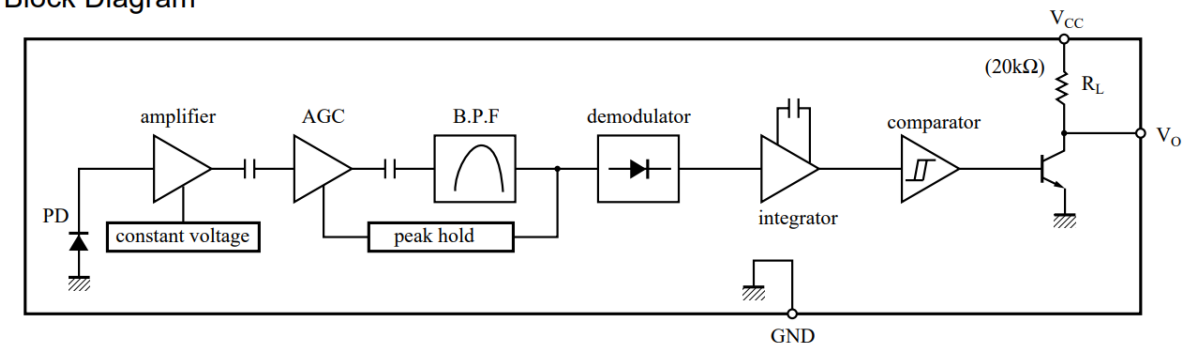
- Converts IR signal to digital pulses
- Models
  - TSOP4838 and TSOP2438 from www.mouser.com
  - PNA4602 by Panasonic



IR Remote controller
(Panasonic Transmitter Specification)

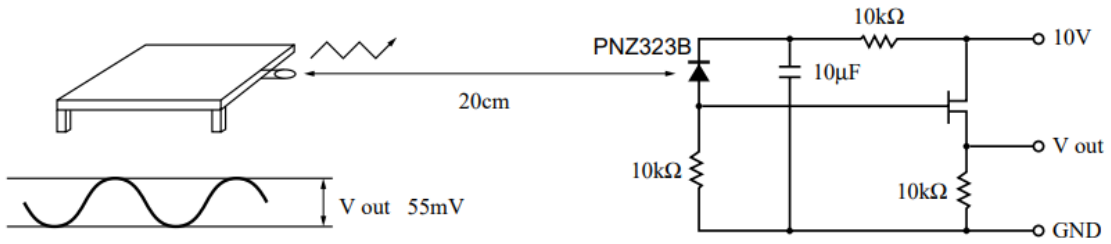
- The light output of the LED transmission unit is adjusted so that the transmission output (V out) of the standard reception unit will be 55 mV when the transmission waveform **(duty = 50%)** is output from the LED transmission unit.
- Here, infrared sensitivity (SIR) of PNZ323B is 0.53 µA when emission illuminance (H) is 12.45 µW/ cm2.

Block Diagram



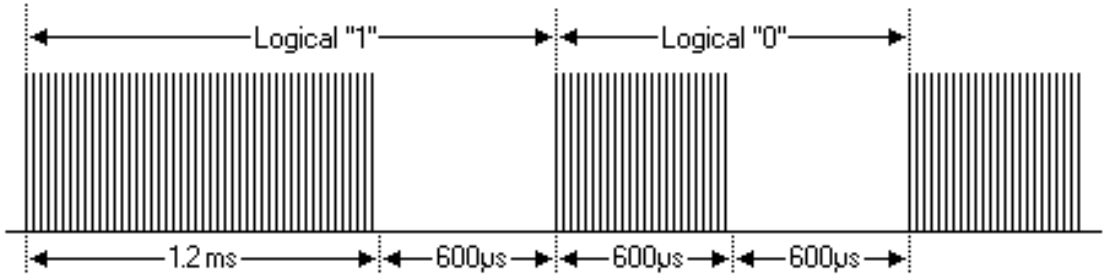
LED Transmission unit

Standard reception unit



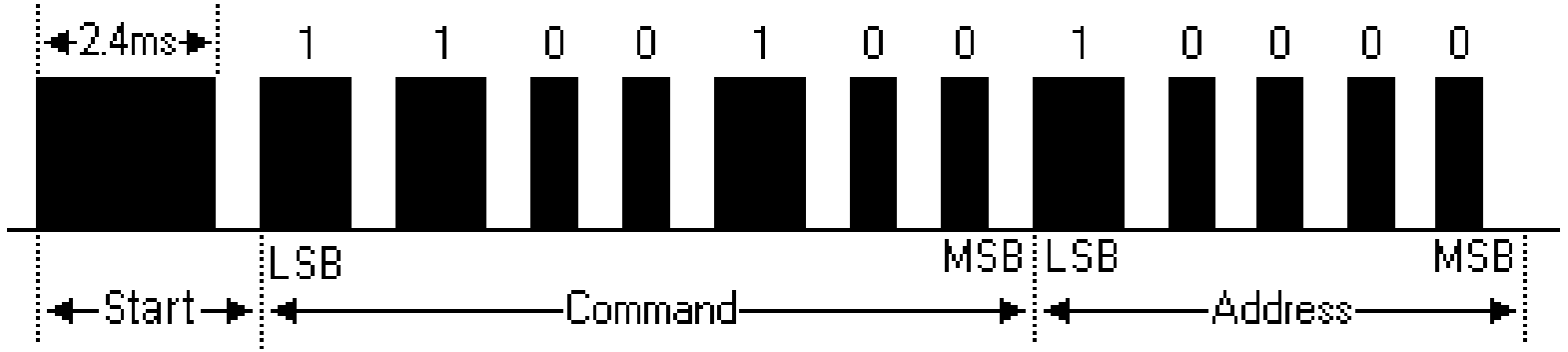
IR Remote controller
Sony SIRC Protocol

- **Features**
  - 12-bit version, 7 command bits, 5 address bits.
  - 15-bit version, 7 command bits, 8 address bits.
  - 20-bit version, 7 command bits, 5 address bits, 8 extended bits.
  - Pulse width modulation.
  - Carrier frequency of 40kHz.
  - Bit time of 1.2ms or 0.6ms
- **Modulation**
  - The SIRC protocol uses pulse width encoding of the bits. The pulse representing a logical "1" is a 1.2ms long burst of the 40kHz carrier, while the burst width for a logical "0" is 0.6ms long. All bursts are separated by a 0.6ms long space interval. The recommended carrier duty-cycle is 1/4 or 1/3.



IR Remote controller
Sony SIRC Protocol- (cont.)

- **Protocol**
  - The following picture shows a typical pulse train of the 12-bit SIRC protocol. With this protocol the LSB is transmitted first. The start burst is always 2.4ms wide, followed by a standard space of 0.6ms. Apart from signaling the start of a SIRC message this start burst is also used to adjust the gain of the IR receiver. Then the 7-bit Command is transmitted, followed by the 5-bit Device address.
  - In this case Address 1 and Command 19 is transmitted.
  - Commands are repeated every 45ms(measured from start to start) for as long as the key on the remote control is held down.



IR Remote controller
Sony SIRC Protocol- (cont.)

Example Commands

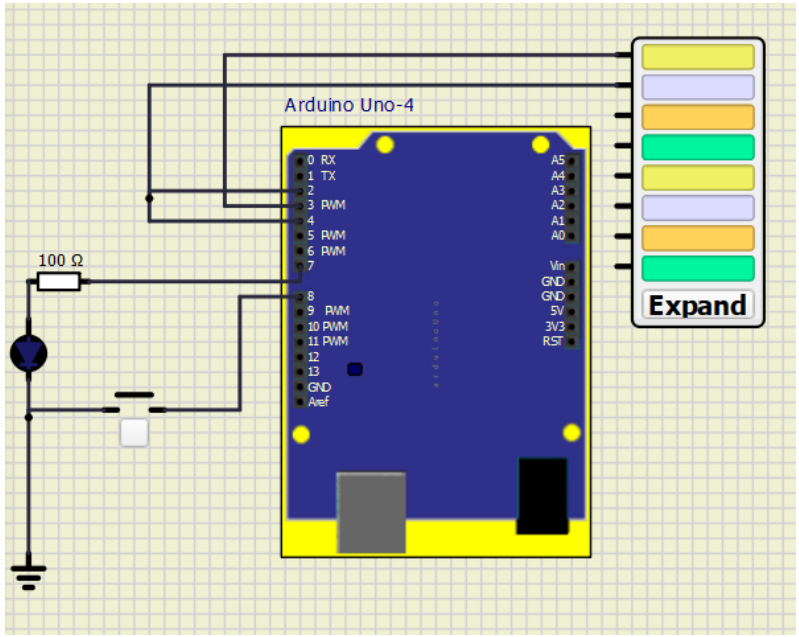
The table below lists some messages sent by Sony remote controls in the 12-bit protocol.

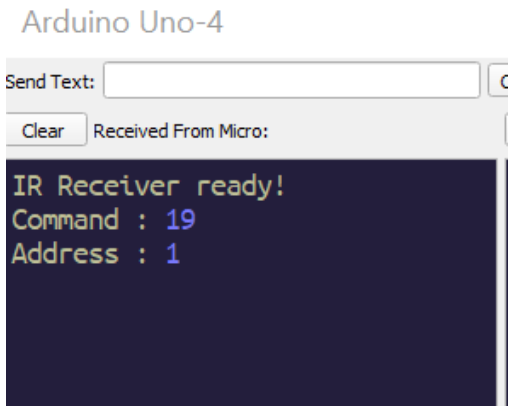
Address	Device
1	TV
2	VCR 1
3	VCR 2
6	Laser Disk
12	Surround Sound
16	Cassette Deck / Tuner
17	CD Player
18	Equalizer

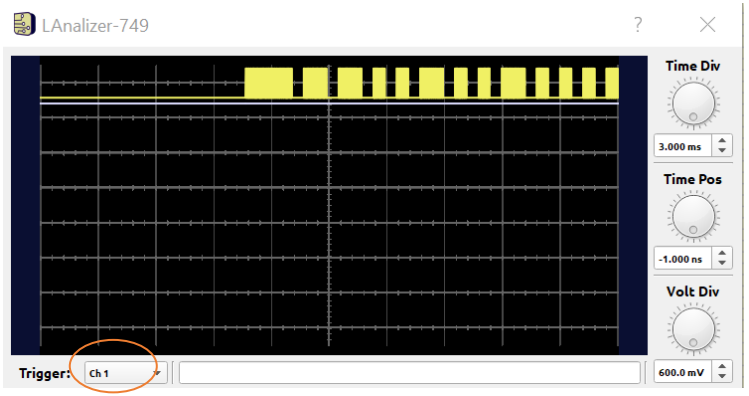
Command	Function
0	Digit key 0
1	Digit key 1
2	Digit key 2
3	Digit key 3
4	Digit key 4
5	Digit key 5
6	Digit key 6
7	Digit key 7
8	Digit key 8
9	Digit key 9
16	Channel +
17	Channel -
18	Volume +
19	Volume -
20	Mute
21	Power

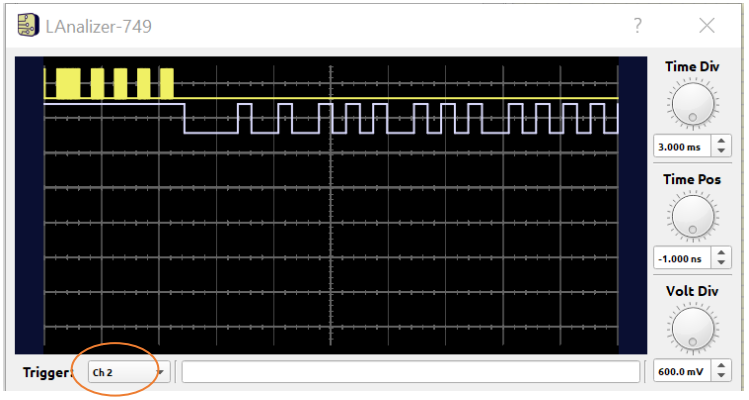
Command	Function
22	Reset
23	Audio Mode
24	Contrast +
25	Contrast -
26	Color +
27	Color -
30	Brightness +
31	Brightness -
38	Balance Left
39	Balance Right
47	Standby

IR Send/Receive Example



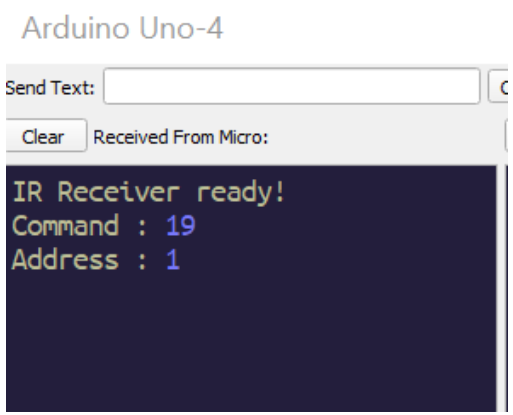


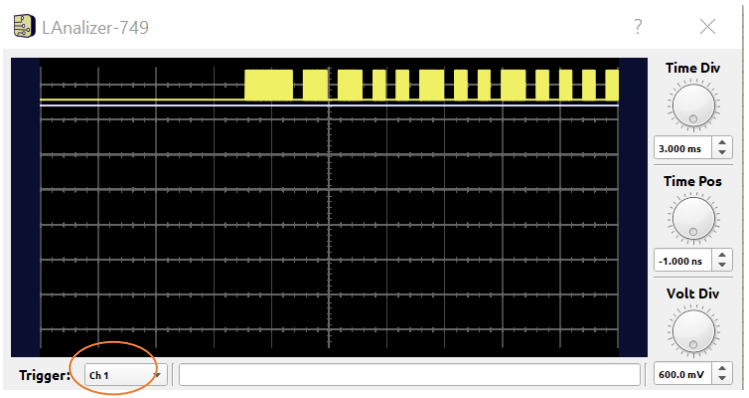


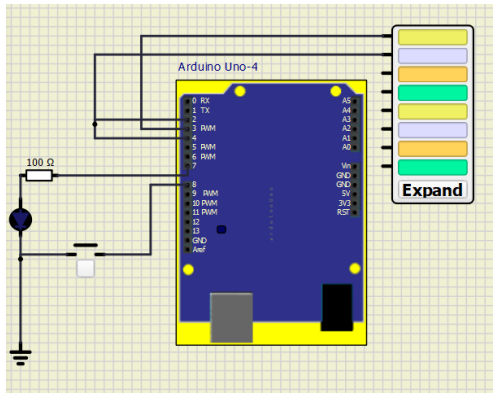


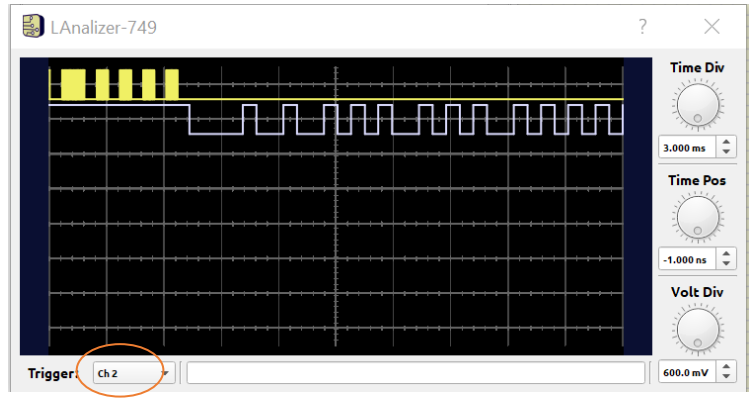
IR Send/Receive Example (cont.)

```
#define IR_SEND_PIN    3
#define IR_RECEIVE_PIN 2
#define LED_PIN 7
#include <IRremote.h>
#define SIM_IR_OUT 4
#define SW_INP 8
IRrecv receiver(IR_RECEIVE_PIN);
IRsend sender;
void setup(){
  pinMode(SW_INP, INPUT_PULLUP);
  pinMode(LED_PIN, OUTPUT);
  pinMode(SIM_IR_OUT, OUTPUT);
  Serial.begin(9600);
  receiver.enableIRIn();
  Serial.println("IR Receiver ready!");
}
```



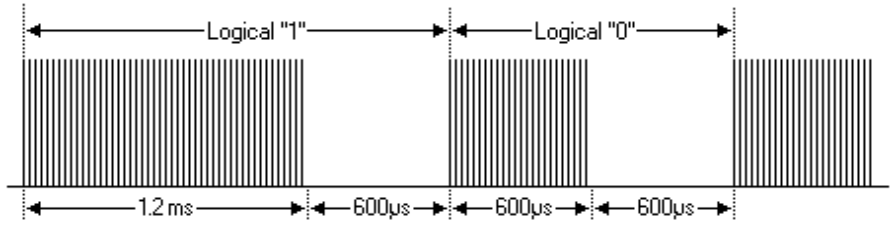


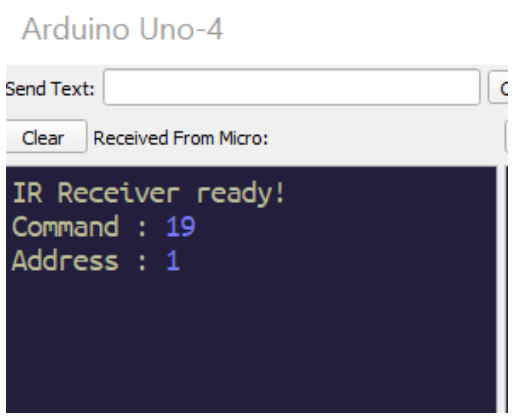


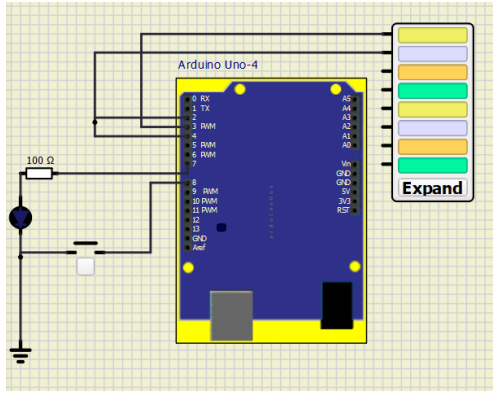


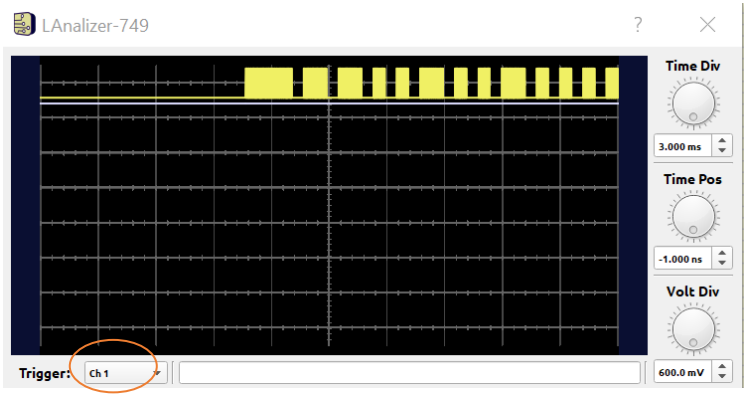
IR Send/Receive Example (cont.)

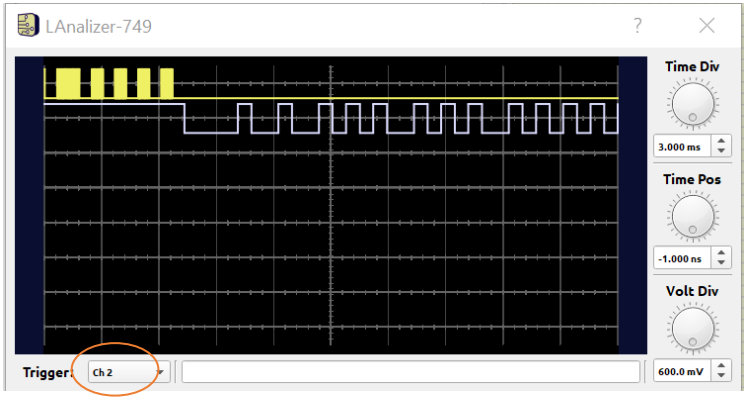
```
17 bool state=0; bool old_state=0;
18 void sendIR(bool v){
19       digitalWrite(SIM_IR_OUT,0);
20       delayMicroseconds((v)?1200:600);
21       digitalWrite(SIM_IR_OUT,1);
22       delayMicroseconds(600);
23 }
```





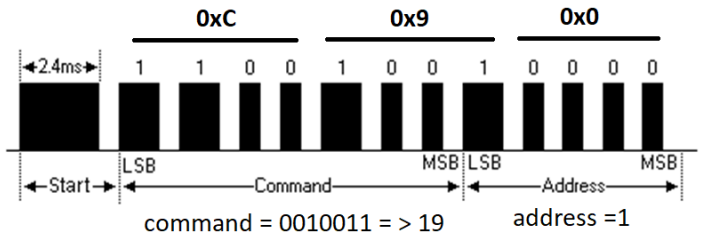






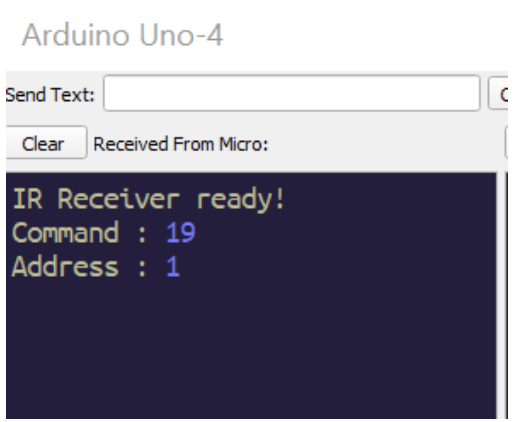
IR Send/Receive Example (cont.)

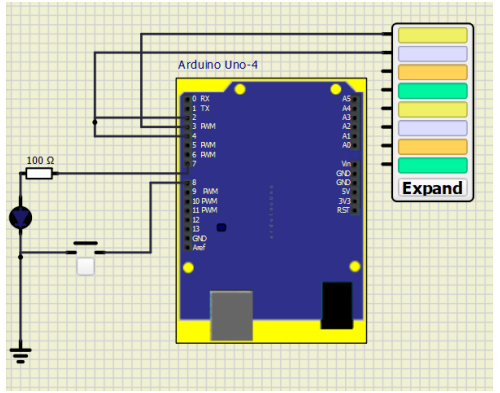
```
24 void loop() {
25 state=digitalRead(SW_INP);
26    if (state && !old_state ){
27       sender.sendSony(0xC90,12);
28       // for simulation only
29       digitalWrite(SIM_IR_OUT,0); delayMicroseconds(2400);
30       digitalWrite(SIM_IR_OUT,1); delayMicroseconds(600);
31       sendIR(1);    sendIR(1);      sendIR(0);      sendIR(0); //0xc
32       sendIR(1);    sendIR(0);      sendIR(0);      sendIR(1); //0x9
33       sendIR(0);    sendIR(0);      sendIR(0);      sendIR(0); //0x0
34     }
35    old_state=state;
36   // decode returns 1 if something was received
37   // otherwise it returns 0
38   if (receiver.decode())
39   {
40    if(receiver.decodedIRData.command == 19)
41          digitalWrite(LED_PIN, !digitalRead(LED_PIN));
42    Serial.print("Command : ");
43    Serial.println(receiver.decodedIRData.command);
44    Serial.print("Address : ");
45    Serial.println(receiver.decodedIRData.address, HEX);
46    receiver.resume(); // Receive the next value
47   }
48 }
```

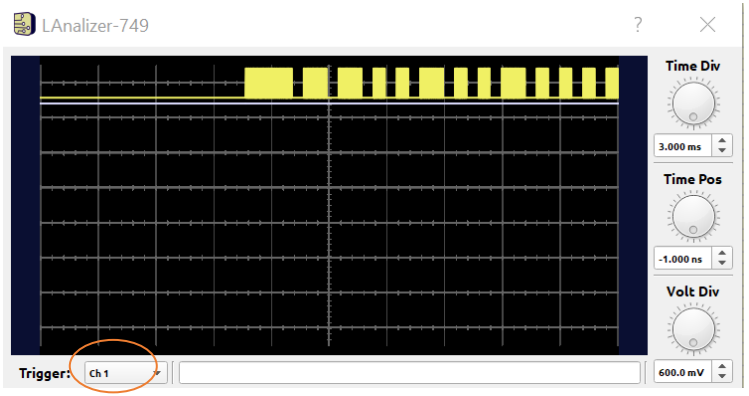


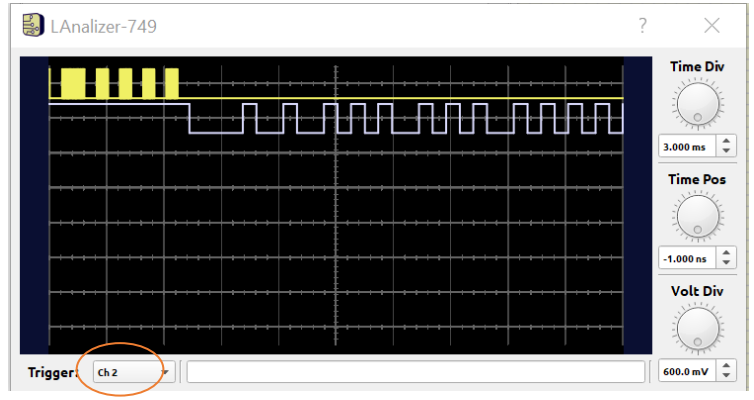
command = 0010011 = > 19

address =1





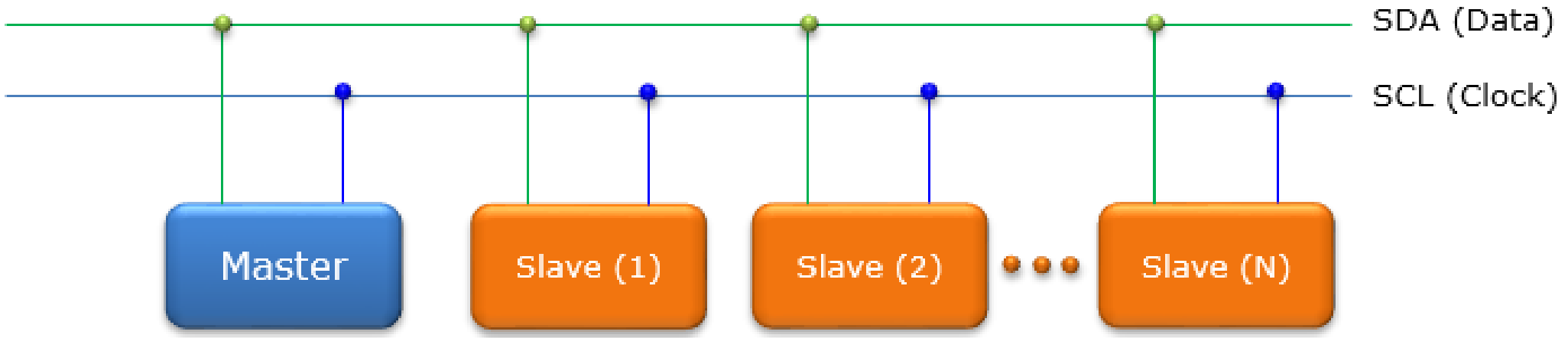




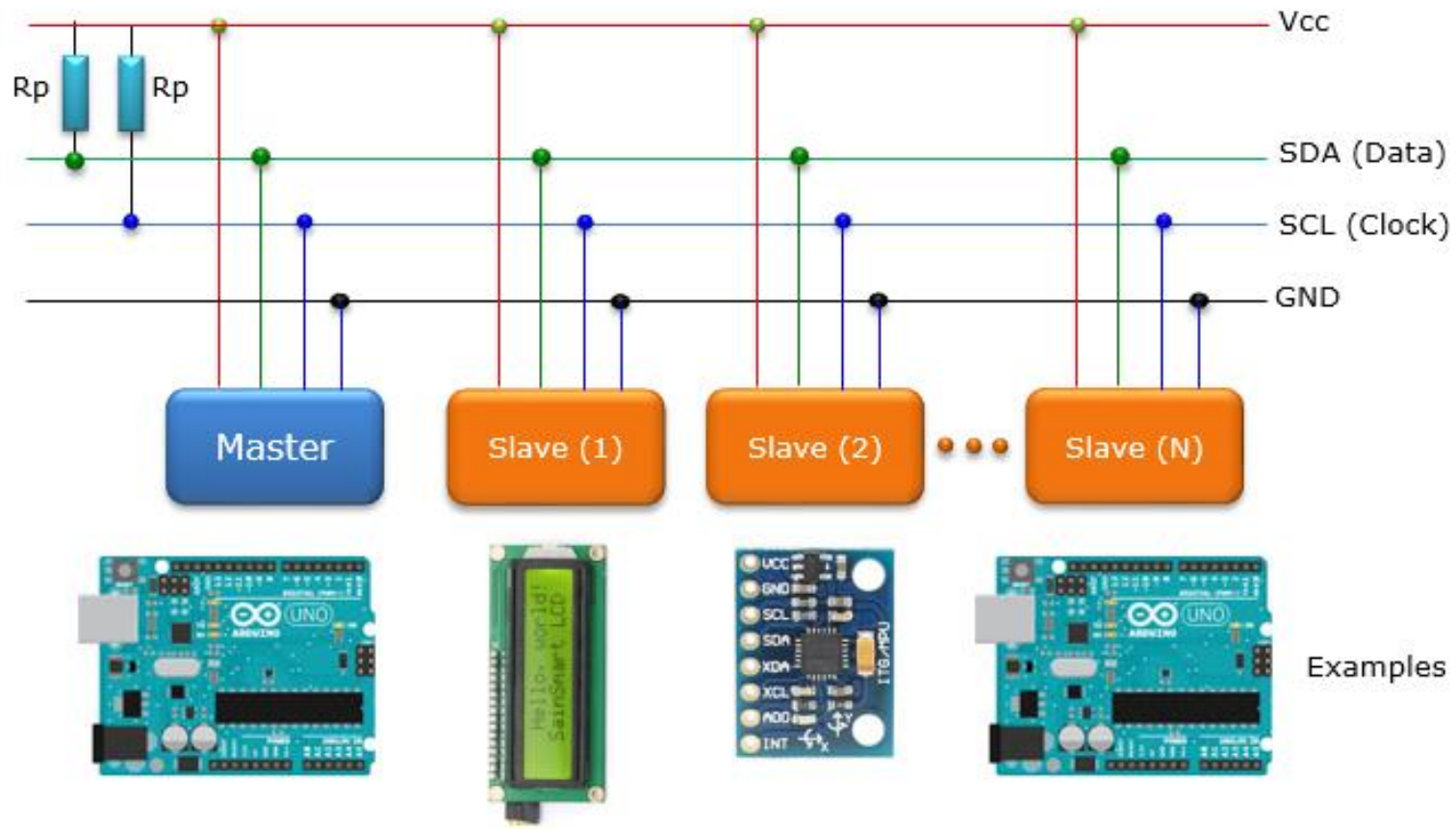
(I^2C) Inter Integrated Circuit

- I2C or I^2C came from the term Inter Integrated Circuit. You see two 'I's and one 'C'
- I2C is implemented by only Two Wires (SDA, SCL), it is also called TWI (Two Wire Interface).
- It is a kind of serial communication technology which is originally designed for exchanging data between multiple IC chips.
- This is very simple communication mechanism, and It is very smartly designed. It requires only two lines as. One of the line is used for sending Clock signal and the other line is to send/receive data.
- Also, these two lines can be shared by multiple devices. This communication works as Master-Slave mode. One Master can control / communicate multiple Slaves.

(I^2C) Inter Integrated Circuit (Cont.)



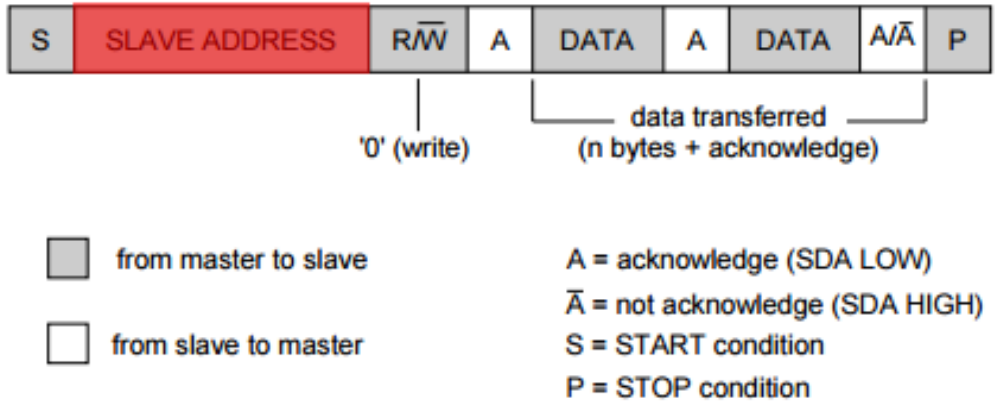
(I^2C) Inter Integrated Circuit (Cont.)



(I^2C) Inter Integrated Circuit (Cont.)

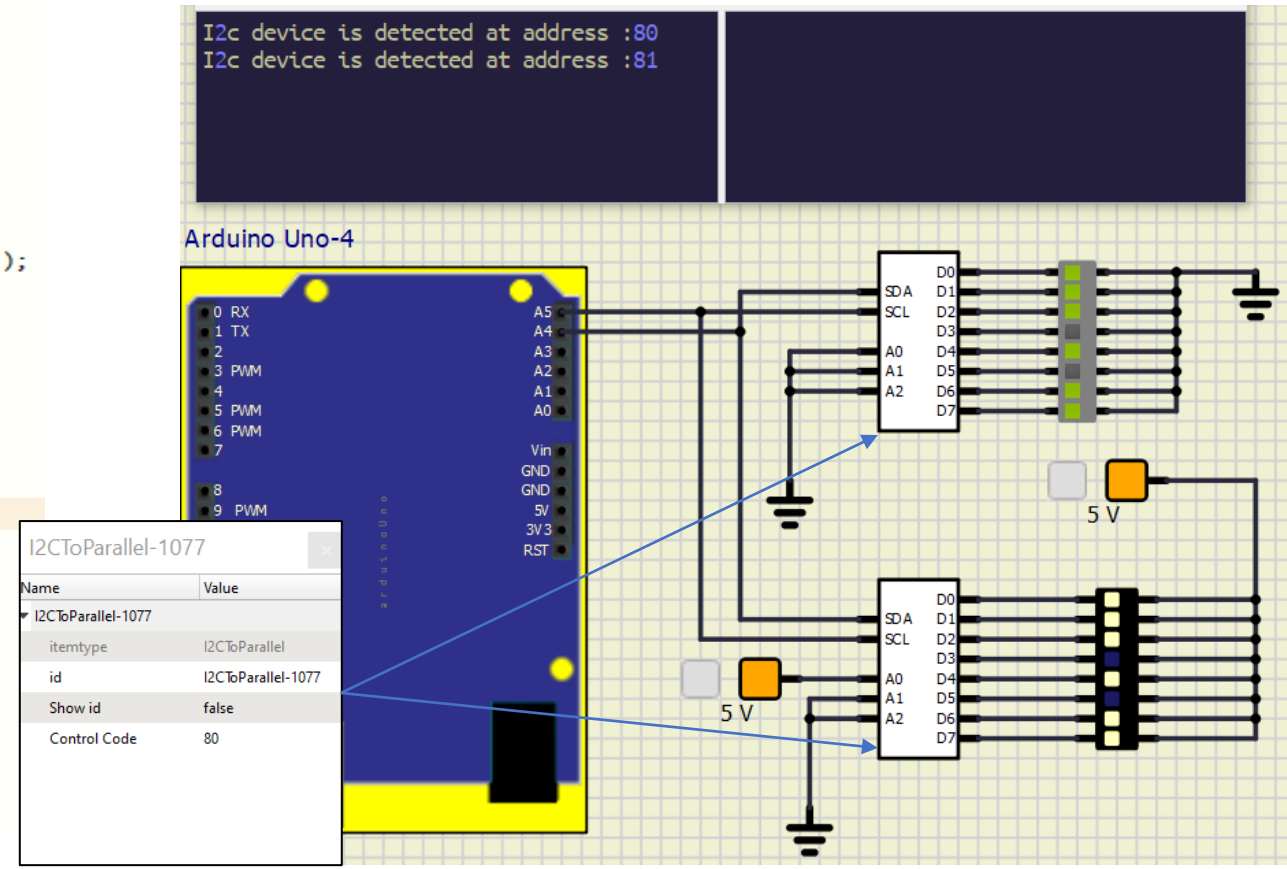
How can a Master send data to a specific Slave ?

Step	Direction	Message
1	Master -> Slave	Start
2	Master -> Slave	Slave Address
3	Master <- Slave	Ack
4	Master -> Slave	Data
5	Master <- Slave	Ack
6	Master -> Slave	Stop



(I^2C) Example(Port Expander)

```
#include <Wire.h>
#define IN_ADDRESS 80
#define OUT_ADDRESS 81
void I2C_scan(void){
    for (byte address=0;address<127;address++){
        Wire.beginTransmission(address);
        byte error=Wire.endTransmission();
        if (!error){
              Serial.print("I2c device is detected at address :");
              Serial.println(address);
        }
    }
}
void setup(){
    Wire.begin();
    Serial.begin(9600);
    I2C_scan();
}
void loop(){
        Wire.requestFrom(IN_ADDRESS, 1);
        byte read_val = Wire.read();
        Wire.beginTransmission(OUT_ADDRESS);
        Wire.write(read_val);
        Wire.endTransmission();
        delay(100);
}
```



I²C Externa EEPROM (Example)

- EEPROM: Electrically Erasable Programmable ROM
- Uses I2C for communication.
- Size: 128 Kbit
- Default Address: 0x50
- Model:
  - 24LC128 from www.mouser.com

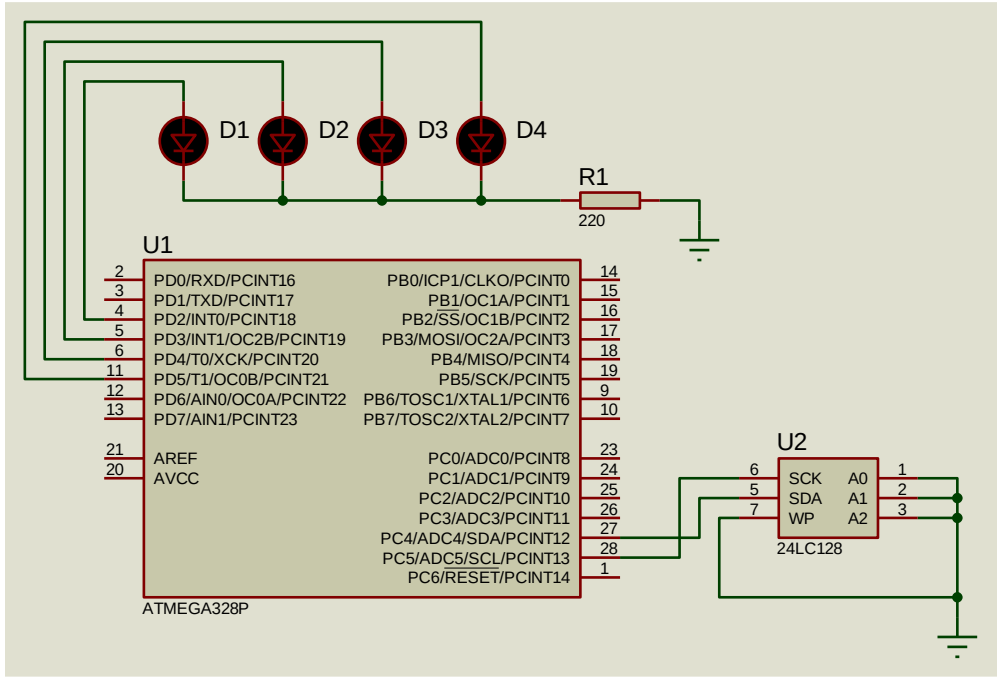
A0	A1	A2	Address
Gnd	Gnd	Gnd	0x50
+5V	Gnd	Gnd	0x51
Gnd	+5V	Gnd	0x52
+5V	+5V	Gnd	0x53
Gnd	Gnd	+5V	0x54
+5V	Gnd	+5V	0x55
+5V	+5V	Gnd	0x56
+5V	+5V	+5V	0x57

I²C Externa EEPROM (Example) Cont.

```
#include <Wire.h>
const byte EEPROMAddress = 0x50;
char text[] = "234543223234545234523454325432\n";
void I2CEEPROM_Write(unsigned int address, byte data) {
     Wire.beginTransmission(EEPROMAddress);
     Wire.send((int)highByte(address) );
     Wire.send((int)lowByte(address) );
     Wire.send(data);
     Wire.endTransmission();
     delay(5);
     // wait for the I2C EEPROM to complete the write cycle
}
byte I2CEEPROM_Read(unsigned int address) {
     byte data;
     Wire.beginTransmission(EEPROMAddress);
     Wire.send((int)highByte(address) );
     Wire.send((int)lowByte(address) );
     Wire.endTransmission();
     Wire.requestFrom(EEPROMAddress,(byte)1);
     while(Wire.available() == 0); // wait for data
     data = Wire.receive();
     return data;
}
void setup() {
     Wire.begin();
     for(int i=2;i<=5;i++) pinMode(i, OUTPUT);
     for(int i=0; i < sizeof(text); i++)
          I2CEEPROM_Write(i, text[i]);
}
void loop() {
     for(int i=0; i < sizeof(text); i++){
          char c = I2CEEPROM_Read(i);
          digitalWrite(c-'0', HIGH);
          delay(100);
          digitalWrite(c-'0', LOW);
     }
}
```

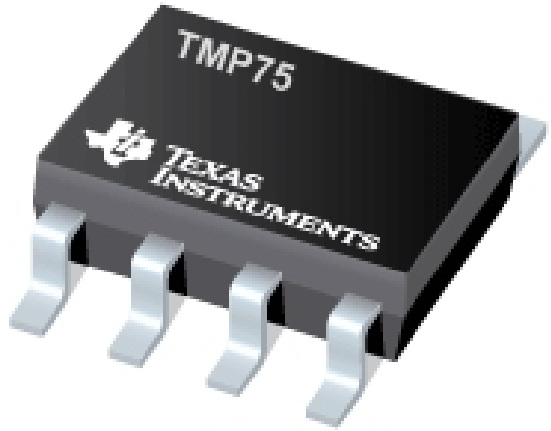
Wire.write(data)

Wire.read()



Reading Temperature with a Digital Thermometer

- Digital Thermometer Module: Measure Temperature
- Uses I2C for communication.
- Range: -40 →125 Celsius degree
- Default Address: 0x49
- Model:
  - TMP75 from www.mouser.com

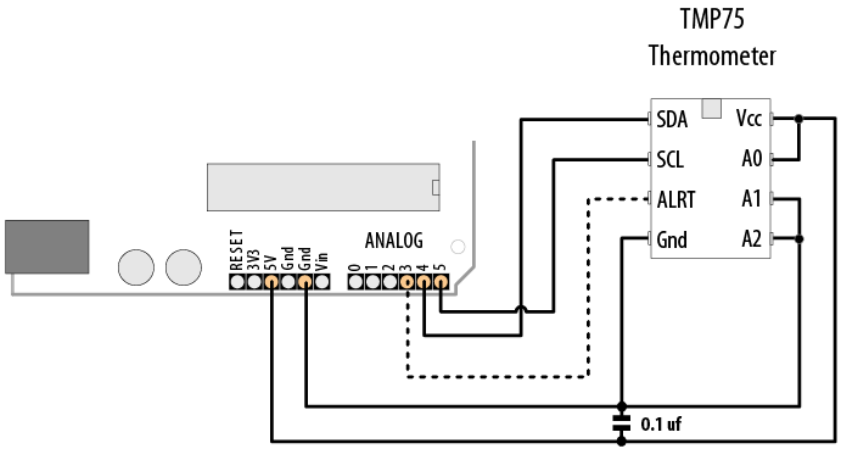


Reading Temperature with a Digital Thermometer (Example)

```
#include <Wire.h>
const byte TMP75Address = 0x49;
void setup() {
      Serial.begin(9600);
      //Configure and Initialize the Module
      Wire.begin();
      Wire.beginTransmission(TMP75Address);
      Wire.send(1);
      Wire.send(0);
      Wire.endTransmission();
      //Choose 12 Bit Temprature
      Wire.beginTransmission(TMP75Address);
      Wire.send(0);
      Wire.endTransmission();
}
void loop(){
      Wire.requestFrom(TMP75Address, (byte)2);
      //Request 2 fields
      byte tempHighByte = Wire.receive();
      byte tempLowByte = Wire.receive();
      Serial.print("Integer temperature is ");
      float temperature = word( tempHighByte, tempLowByte) / 256.0;
      //Convert the value to a float
      Serial.println(temperature);
      delay(1000);
}
```

Wire.write(data)

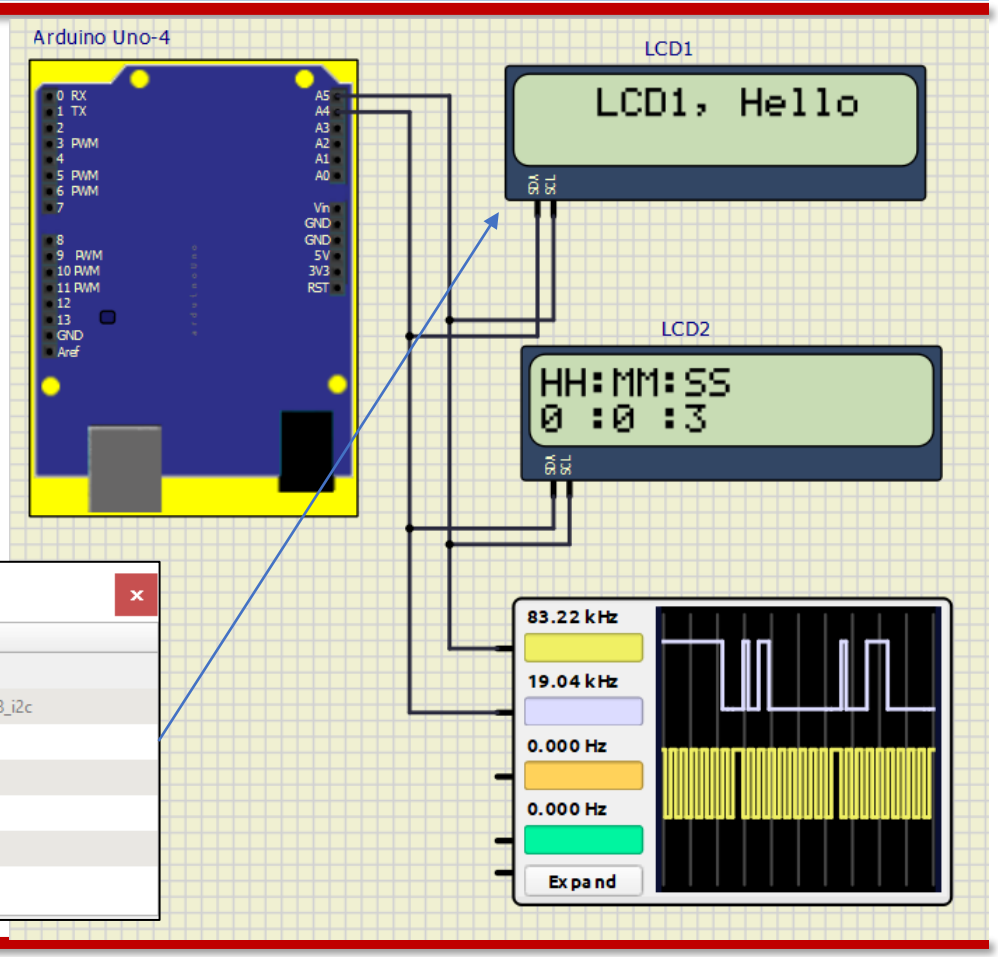
Wire.read()



I²C LCD

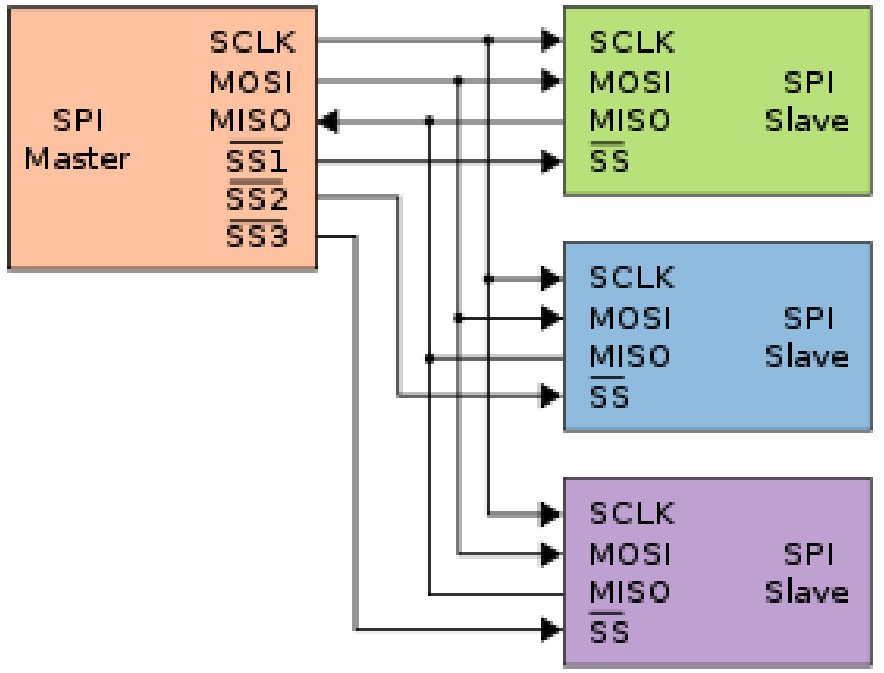
```
#include <LiquidCrystal_AIP31068_I2C.h>
// set the LCD address to 0x3E for a 20 chars and 4 line display
LiquidCrystal_AIP31068_I2C lcd1(0x3E,20,2);
 // set the LCD address to 0x3E for a 20 chars and 4 line display
LiquidCrystal_AIP31068_I2C lcd2(0x3F,20,2);

void setup()
{
  lcd1.init();                      // initialize the lcd 1
  lcd1.setCursor(3,0);
  lcd1.print("LCD1, Hello");
  lcd2.init();                      // initialize the lcd  2
  lcd2.setCursor(0,0);
  lcd2.print("HH:MM:SS");
  lcd2.setCursor(0,1);
  lcd2.print("  :  :  ");
}
int s=0;
int m=0;
int h=0;
void loop()
{
     h= (h<59)?  ((m==59)?h+1:h)  :0;
     m= (m<59)?  ((s==59)?m+1:m)  :0;
     s= (s<59)?s+1:0;
     lcd2.setCursor(6,1);
     lcd2.print(s);
     lcd2.setCursor(3,1);
     lcd2.print(m);
     lcd2.setCursor(0,1);
     lcd2.print(h);
     delay(1000);
}
```

$(62)_{10} = (3E)_{16}$



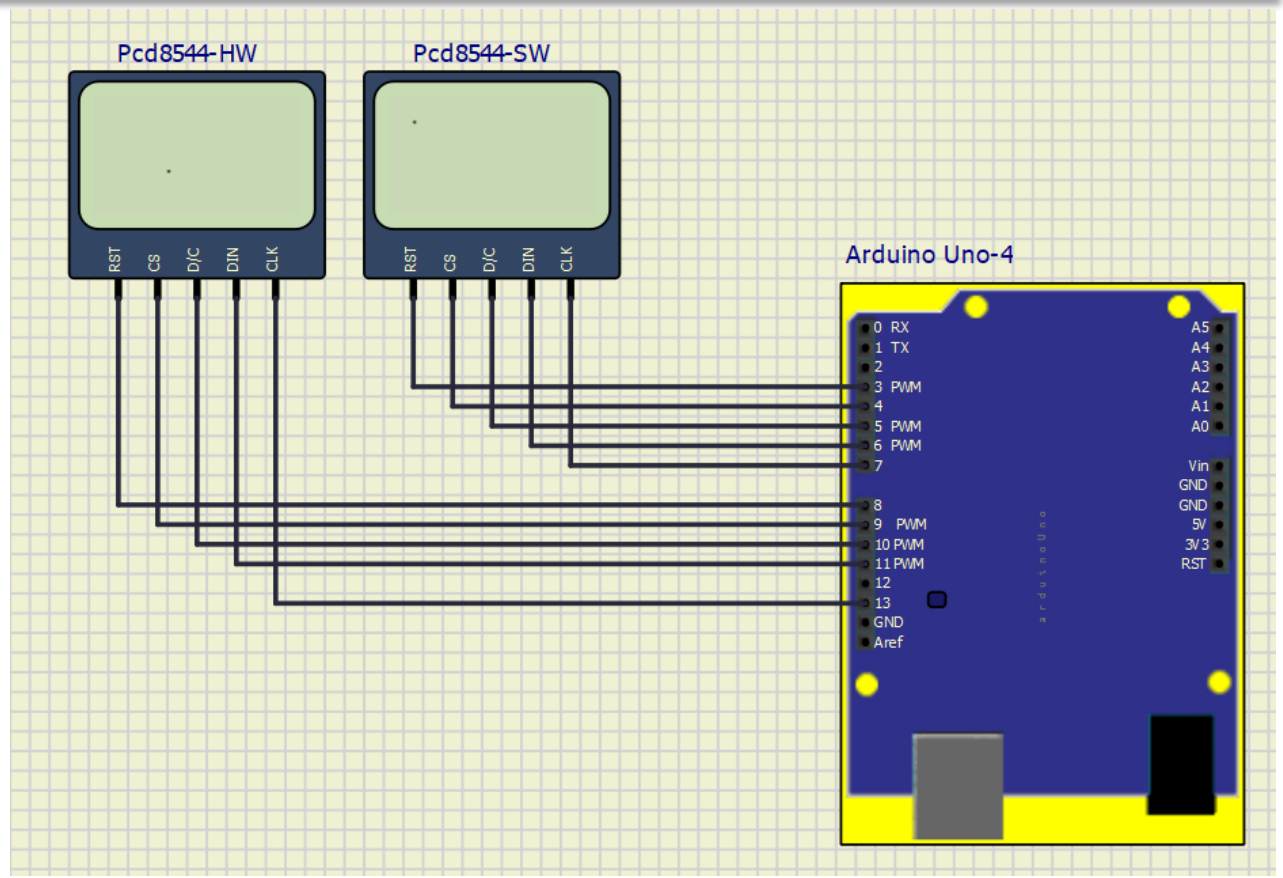
(SPI) Serial Peripheral Interface

- Serial Peripheral Interface (SPI) is a synchronous serial data protocol used by microcontrollers for communicating with one or more peripheral devices quickly over short distances. It can also be used for communication between two microcontrollers.
- With an SPI connection there is always one master device (usually a microcontroller) which controls the peripheral devices. Typically, there are three lines common to all the devices:
  - MISO (Master In Slave Out) - The Slave line for sending data to the master,
  - MOSI (Master Out Slave In) - The Master line for sending data to the peripherals,
  - SCK (Serial Clock) - The clock pulses which synchronize data transmission generated by the master and one line specific for every device:
  - SS (Slave Select) - the pin on each device that the master can use to enable and disable specific devices.
- When a device's Slave Select pin is low, it communicates with the master. When it's high, it ignores the master. This allows you to have multiple SPI devices sharing the same MISO, MOSI, and CLK lines.
- Arduino UNO (SCLCK : 13, MISO : 12, MOSI0 : 11, SS : 10)



(SPI) Example (LCD SW/HW)

```
15 #include <SPI.h>
16 #include <Adafruit_GFX.h>
17 #include <Adafruit_PCD8544.h>
18
19 // Software SPI (slower updates, more flexible pin options):
20 // pin 7 - Serial clock out (SCLK)
21 // pin 6 - Serial data out (DIN)
22 // pin 5 - Data/Command select (D/C)
23 // pin 4 - LCD chip select (CS)
24 // pin 3 - LCD reset (RST)
25 Adafruit_PCD8544 sw_display = Adafruit_PCD8544(7, 6, 5, 4, 3);
26
27 // Hardware SPI (faster, but must use certain hardware pins):
28 // SCK is LCD serial clock (SCLK) - this is pin 13 on Arduino Uno
29 // MOSI is LCD DIN - this is pin 11 on an Arduino Uno
30 // pin 10 - Data/Command select (D/C)
31 // pin 9 - LCD chip select (CS)
32 // pin 8 - LCD reset (RST)
33 Adafruit_PCD8544 hw_display = Adafruit_PCD8544(10, 9, 8);
34 // Note with hardware SPI MISO and SS pins aren't used but will still be read
35 // and written to during SPI transfer.  Be careful sharing these pins!
36
37 void setup()   {
38   sw_display.begin();
39   sw_display.clearDisplay();   // clears the screen and buffer
40   sw_display.drawPixel(10, 10, BLACK);
41   sw_display.display();
42
43   hw_display.begin();
44   hw_display.clearDisplay();   // clears the screen and buffer
45   hw_display.drawPixel(30, 30, BLACK);
46   hw_display.display();
47 }
48
49 void loop() {
50 }
```



Difference Between I2C vs SPI

What is I2C Protocol?
I2C stands for Inter-Integrated Circuit. I2C is a simple two-wire serial protocol used to communicate between two devices or chips in an embedded system. I2C has two lines SCL and SDA, SCL is used for clock and SDA is used for data.

What is SPI Protocol?
SPI stands for Serial Peripheral interface. SPI is a four-wire serial communication protocol. SPI follows master-slave architecture. The four lines of SPI are MOSI, MISO, SCL and SS. SCL is a serial clock that is used for entire data communication. Slave Select(SS) is used to select the slave. Master out Slave In (MOSI) is the output data line from the master and Master in Slave out(MISO) is the input data line for the Master.

Advantage I2C:
- Easy device add on : It is easy to add new slave devices into the bus. Just add the new device without adding a new slave select unlike SPI.

Advantage SPI :
- Speed : SPI can support up to 10MB/s however I^2C in the ultra-fast mode can support only 5MB/s.

References

1) https://www.electronicsforu.com/technology-trends/learn-electronics/ir-led-infrared-sensor-basics
2) https://www.pinterest.co.uk/pin/485403666063432329/
3) https://www.digikey.com/en/maker/blogs/2021/how-to-send-and-receive-data-over-ir-signals-with-an-arduino
4) https://cdn-learn.adafruit.com/assets/assets/000/010/139/original/PNA4602.pdf
5) https://www.youtube.com/watch?v=BUvFGTxZBG8
6) https://www.sbprojects.net/knowledge/ir/sirc.php
7) http://www.sharetechnote.com/html/EmbeddedSystem_I2C.html
8) https://www.arduino.cc/en/reference/SPI
9) https://es.wikipedia.org/wiki/Serial_Peripheral_Interface
10) https://prodigytechno.com/i2c-vs-spi/