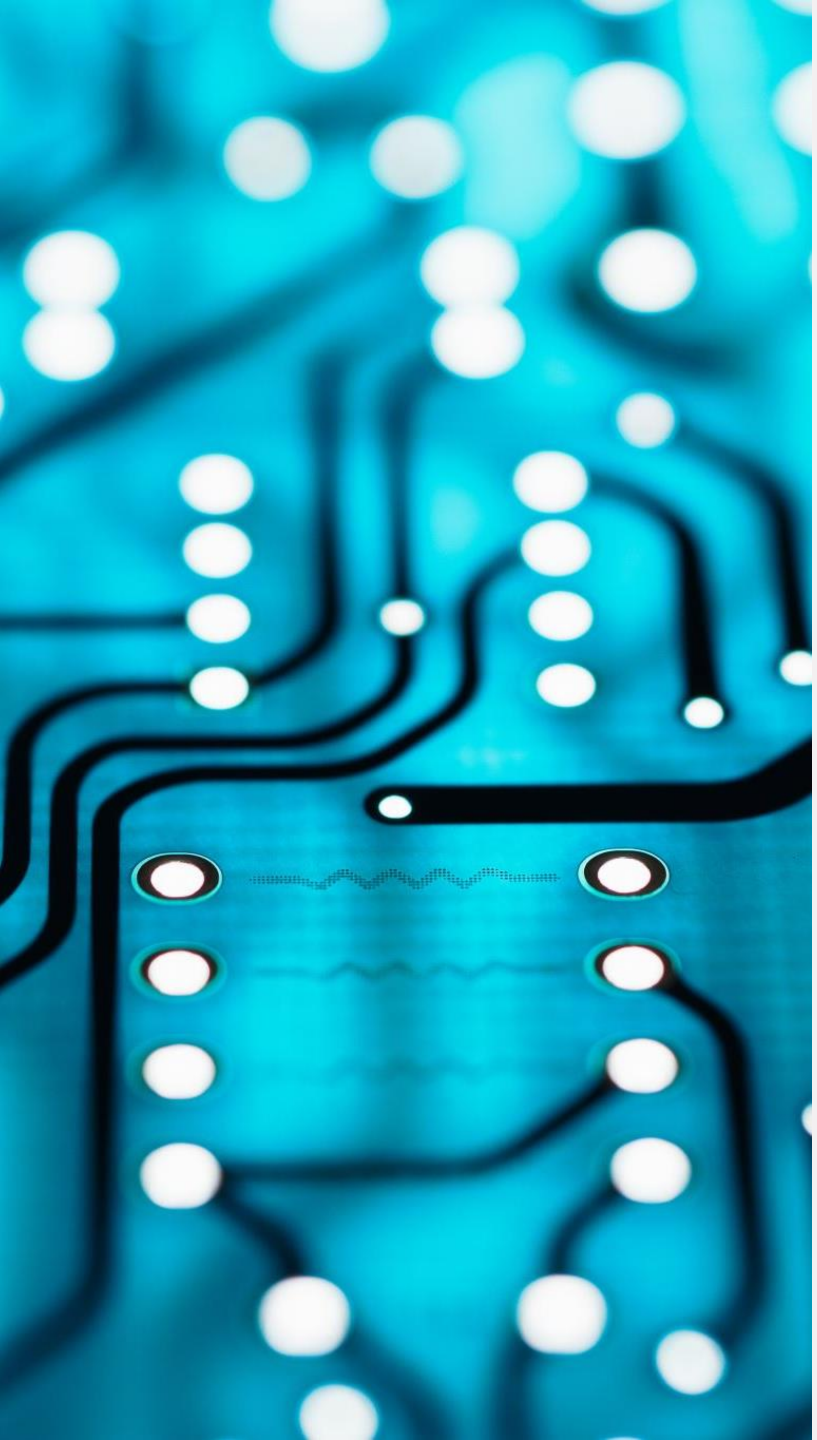
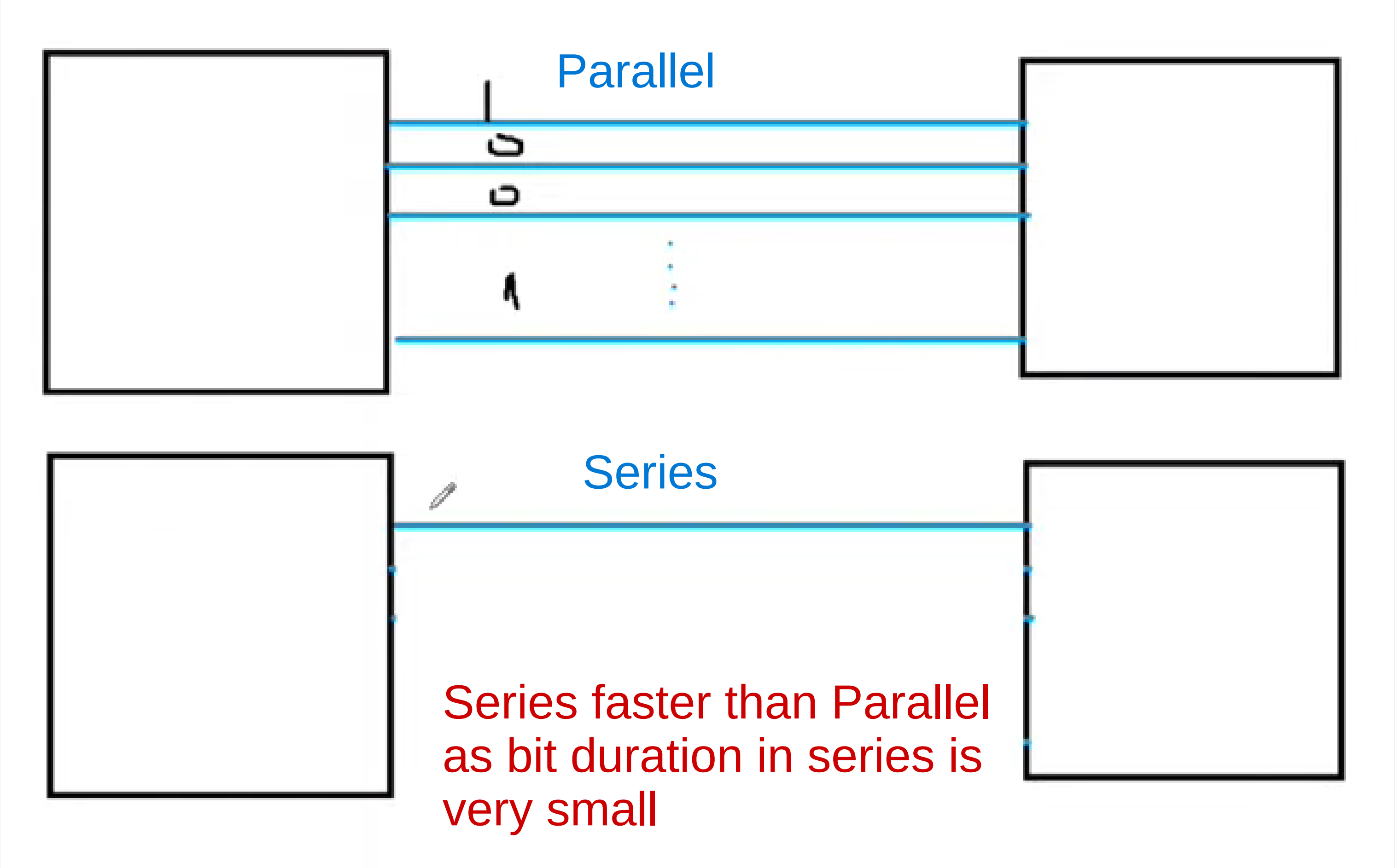


Embedded Systems (EPM)

Lecture (8) Summary

Parallel VS Series Data Sending:



Series faster than Parallel as bit duration in series is very small

IR Communication:

(IR LED):

- is a LED emitting infrared rays ranging from 700 nm to 1 mm wavelength.
- Different IR LEDs may produce infrared light of differing wavelengths, human eye cannot see the infrared radiations.

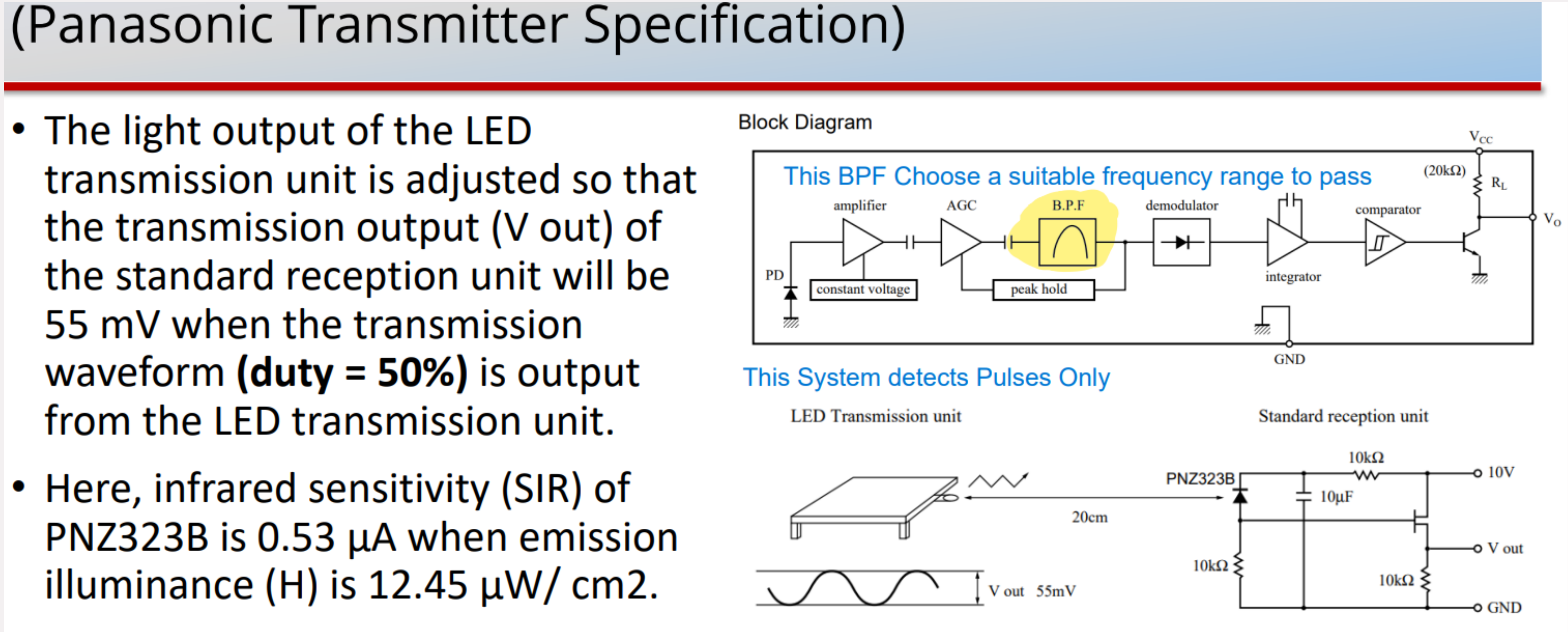
(IR Sensor/ Photodiode):

- is an electronic device that detects IR radiation falling on it.
- used in touchscreen phones , in line following robots, for counting goods and in burglar alarms.
- IR LED and Photodiode can be used to detect obstacles

(IR Remote controller):

IR Remote Signals: flashes its IR LED on/off very rapidly—at 38.5 kHz—for certain periods of time and these pulses are received by the microcontroller.

IR Receiver: Converts IR signal to digital pulses



(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

This BPF Choose a suitable frequency range to pass

This System detects Pulses Only

IR Remote controller

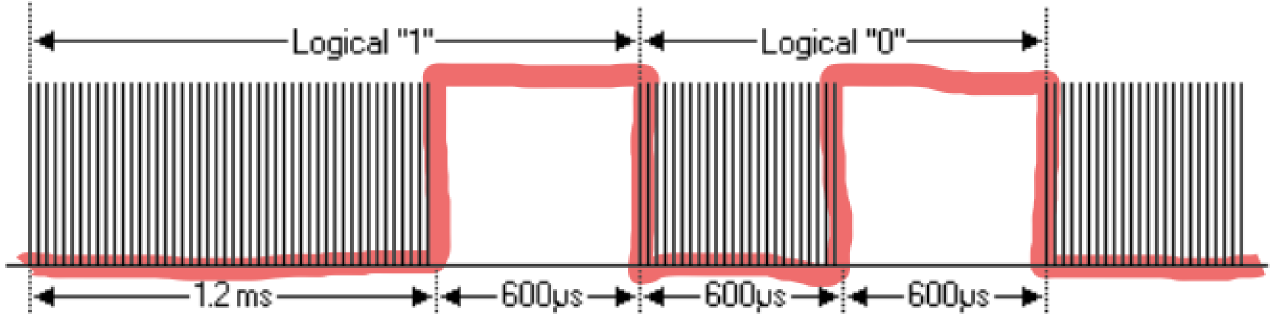
Sony SIRC Protocol

Every Remote Control has its own 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.

This Signal Sended Modulated to Avoid Noises

The Receiver Recives The Signal with inverted Logic

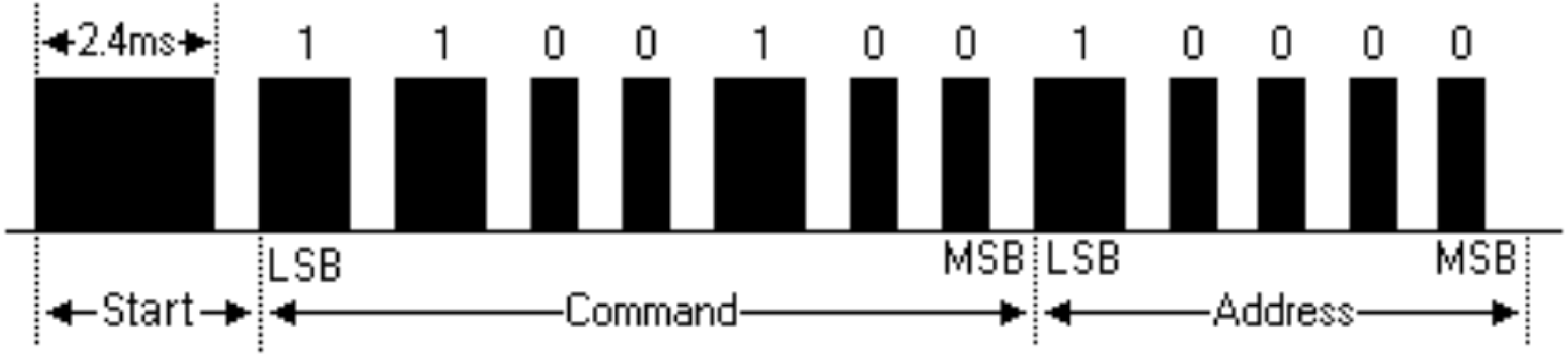


IR Remote controller

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.

Inverted Signal to be Recieved Correctly

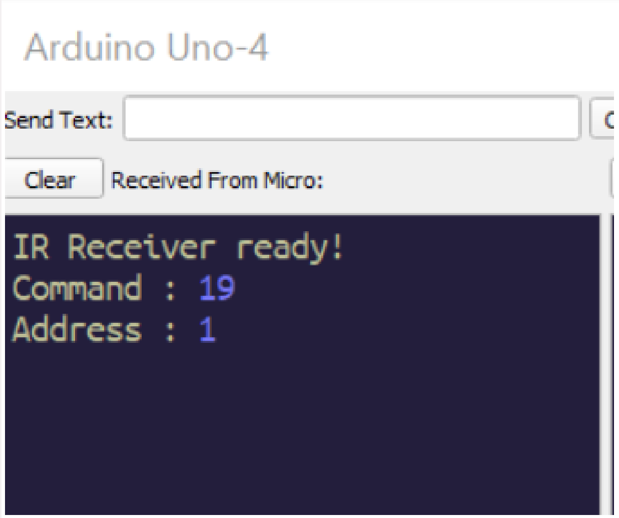


The Received bits : Command , Address

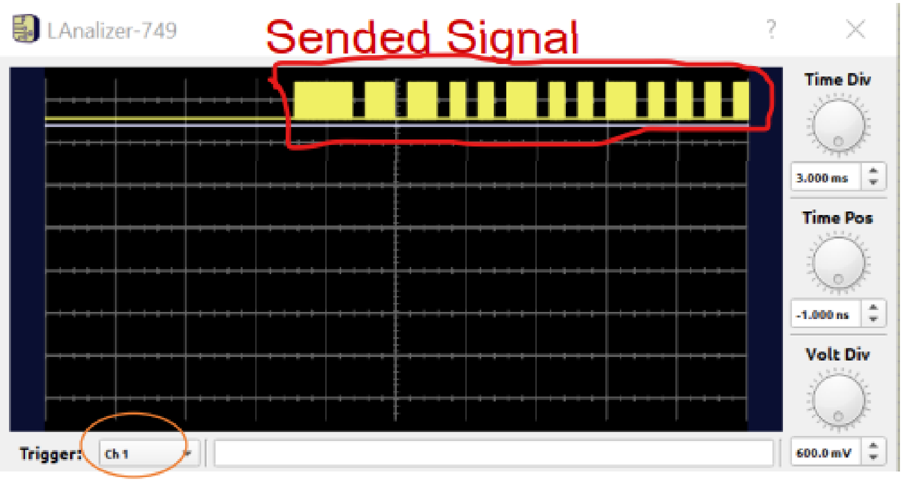
0010011 , 00001

19 , 1

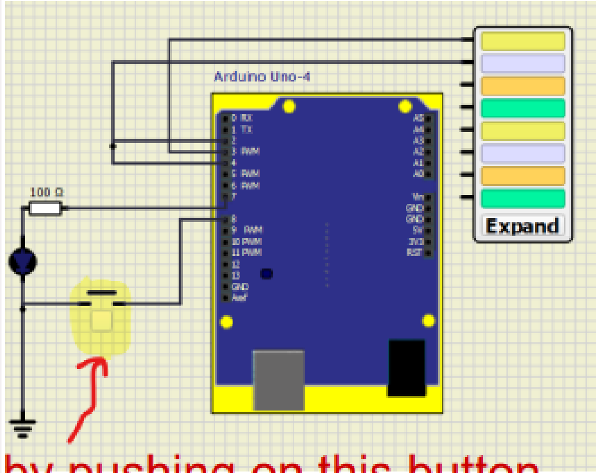
IR Send/Receive Example



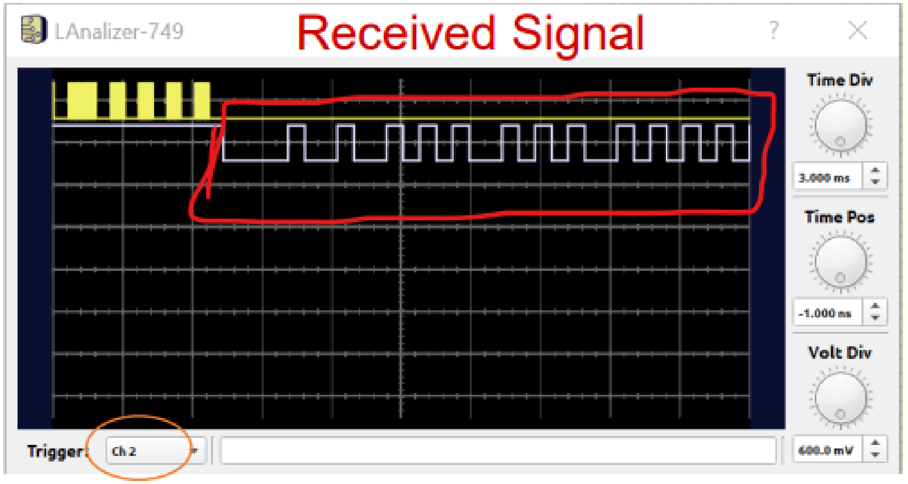
```
IR Receiver ready!
Command : 19
Address : 1
```



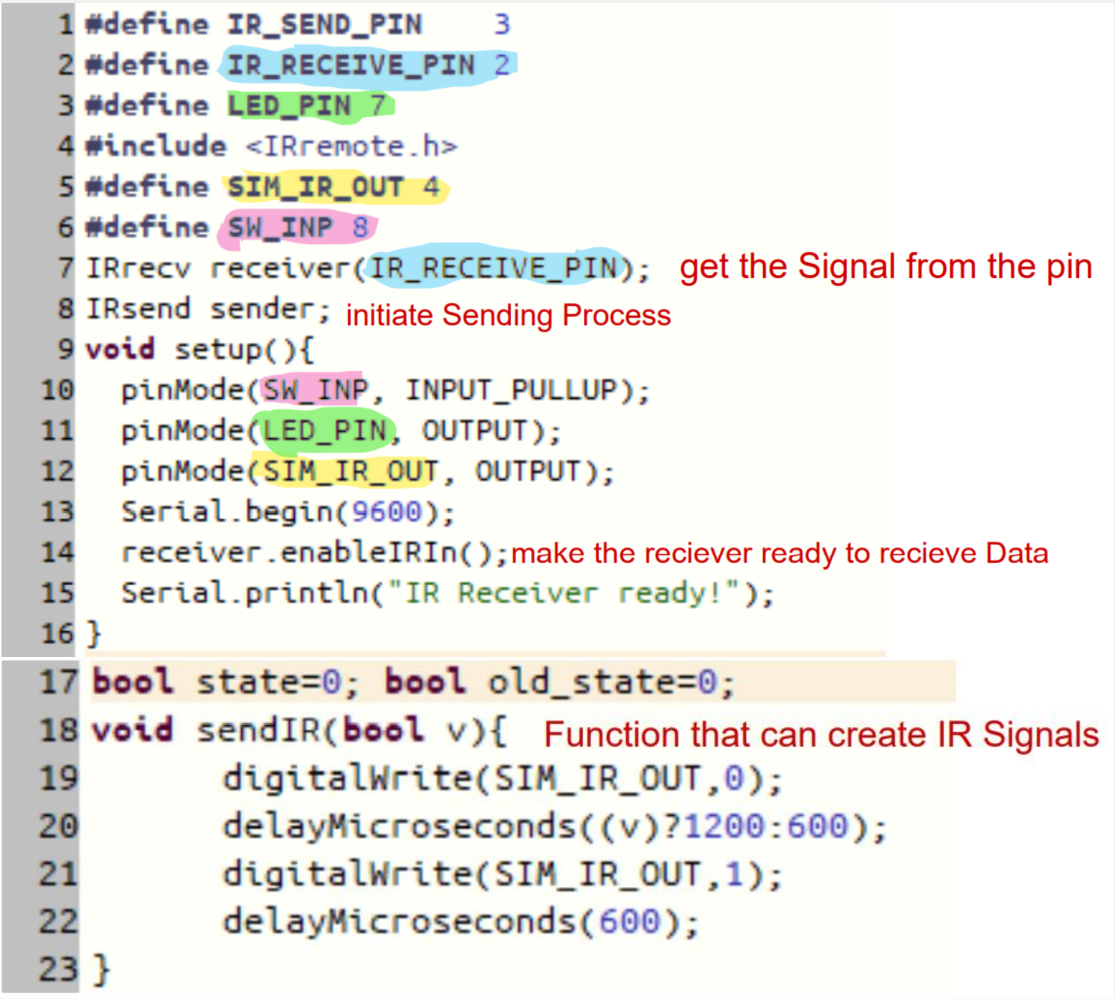
Sended Signal



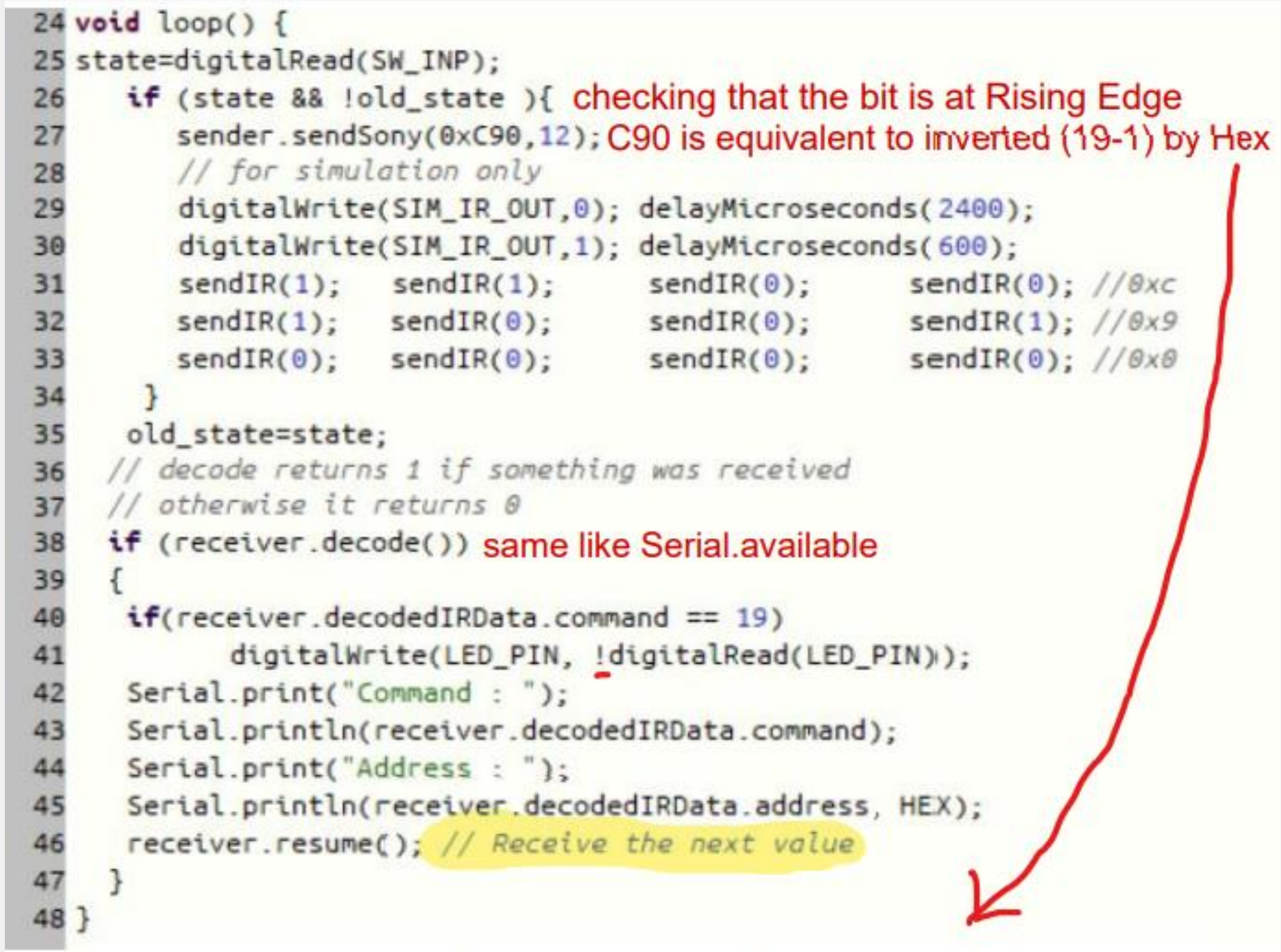
by pushing on this button
The IR signal is being sended



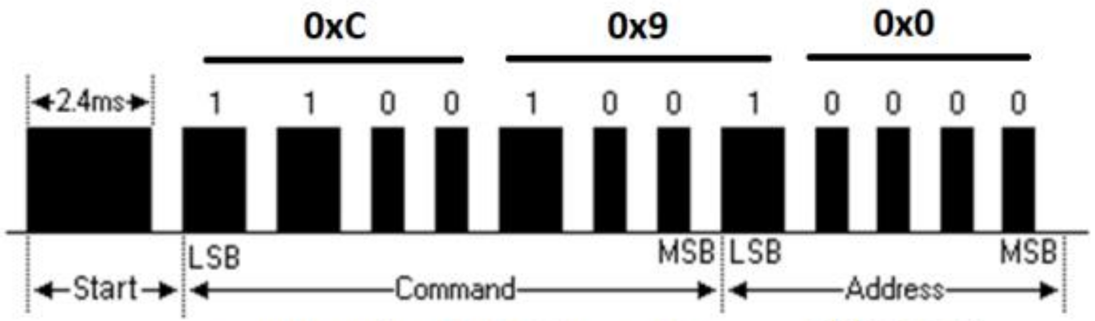
Received Signal



```
#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);  get the Signal from the pin
IRsend sender;  initiate Sending Process
void setup(){
  pinMode(SW_INP, INPUT_PULLUP);
  pinMode(LED_PIN, OUTPUT);
  pinMode(SIM_IR_OUT, OUTPUT);
  Serial.begin(9600);
  receiver.enableIRIn();make the reciever ready to recieve Data
  Serial.println("IR Receiver ready!");
}
bool state=0; bool old_state=0;
void sendIR(bool v){  Function that can create IR Signals
        digitalWrite(SIM_IR_OUT,0);
        delayMicroseconds((v)?1200:600);
        digitalWrite(SIM_IR_OUT,1);
        delayMicroseconds(600);
}
```

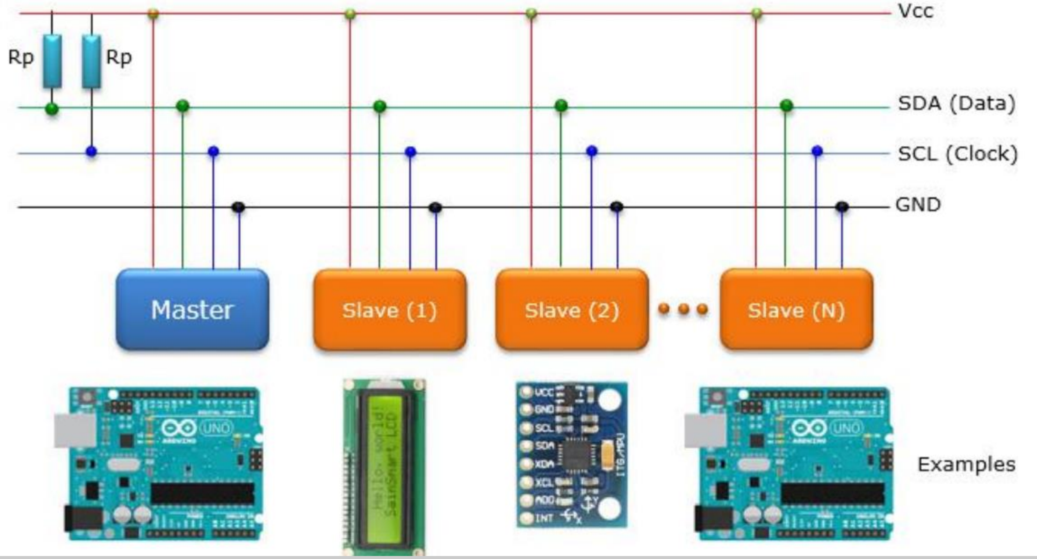
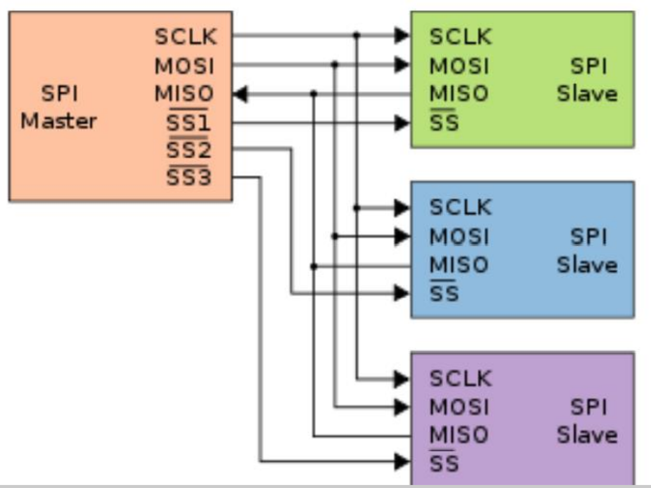


```
void loop() {
state=digitalRead(SW_INP);
   if (state && !old_state ){  checking that the bit is at Rising Edge
      sender.sendSony(0xC90,12); C90 is equivalent to inverted (19-1) by Hex
      // for simulation only
      digitalWrite(SIM_IR_OUT,0); delayMicroseconds(2400);
      digitalWrite(SIM_IR_OUT,1); delayMicroseconds(600);
      sendIR(1);  sendIR(1);    sendIR(0);    sendIR(0); //0xc
      sendIR(1);  sendIR(0);    sendIR(0);    sendIR(1); //0x9
      sendIR(0);  sendIR(0);    sendIR(0);    sendIR(0); //0x0
   }
   old_state=state;
  // decode returns 1 if something was received
  // otherwise it returns 0
  if (receiver.decode()) same like Serial.available
  {
   if(receiver.decodedIRData.command == 19)
         digitalWrite(LED_PIN, !digitalRead(LED_PIN));
   Serial.print("Command : ");
   Serial.println(receiver.decodedIRData.command);
   Serial.print("Address : ");
   Serial.println(receiver.decodedIRData.address, HEX);
   receiver.resume(); // Receive the next value
  }
}
```



command = 0010011 = > 19

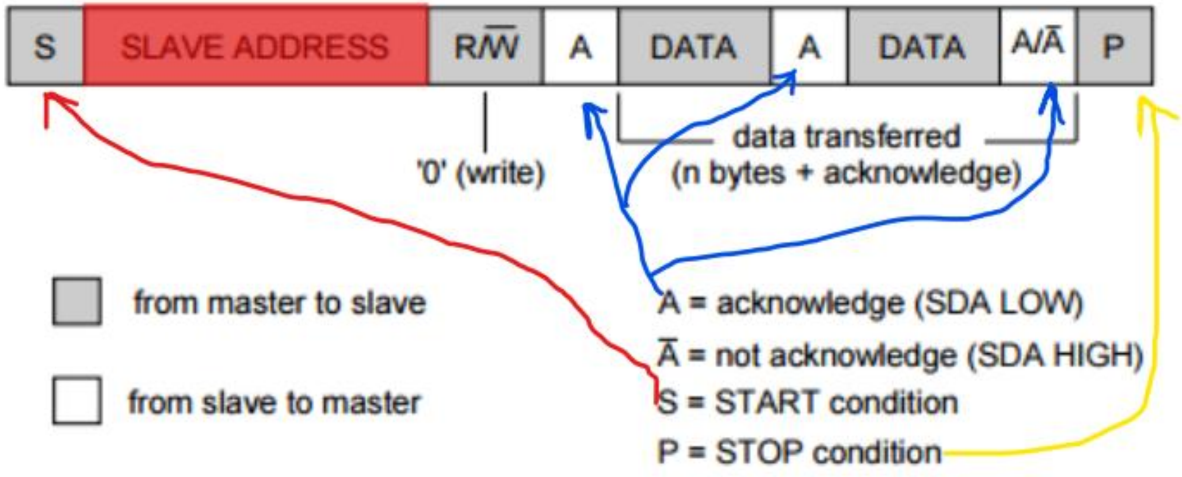
address =1

I2C	SPI
Inter-**I**ntegrated **C**ircuit is a simple two-wire serial protocol used to communicate between two devices or chips. it has two lines SCL and SDA, SCL is used for clock and SDA is used for data. It is easy to add new slave devices into the bus. Just add the new device without adding a new slave select unlike SPI  Rp Rp Vcc SDA (Data) SCL (Clock) GND Master Slave (1) Slave (2) Slave (N) Examples	Serial Peripheral interface is a four-wire serial communication protocol, it follows master-slave architecture. The four lines are MOSI, MISO, SCL and SS. (SCLK) 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 Master in Slave out(MISO) is the input data line for the Master. SPI can support up to 10MB/s I2C in the ultra-fast Mode can support only 5MB/s  SPI Master SCLK MOSI MISO SS1 SS2 SS3 SCLK MOSI MISO SS SPI Slave SCLK MOSI MISO SS SPI Slave SCLK MOSI MISO SS SPI Slave

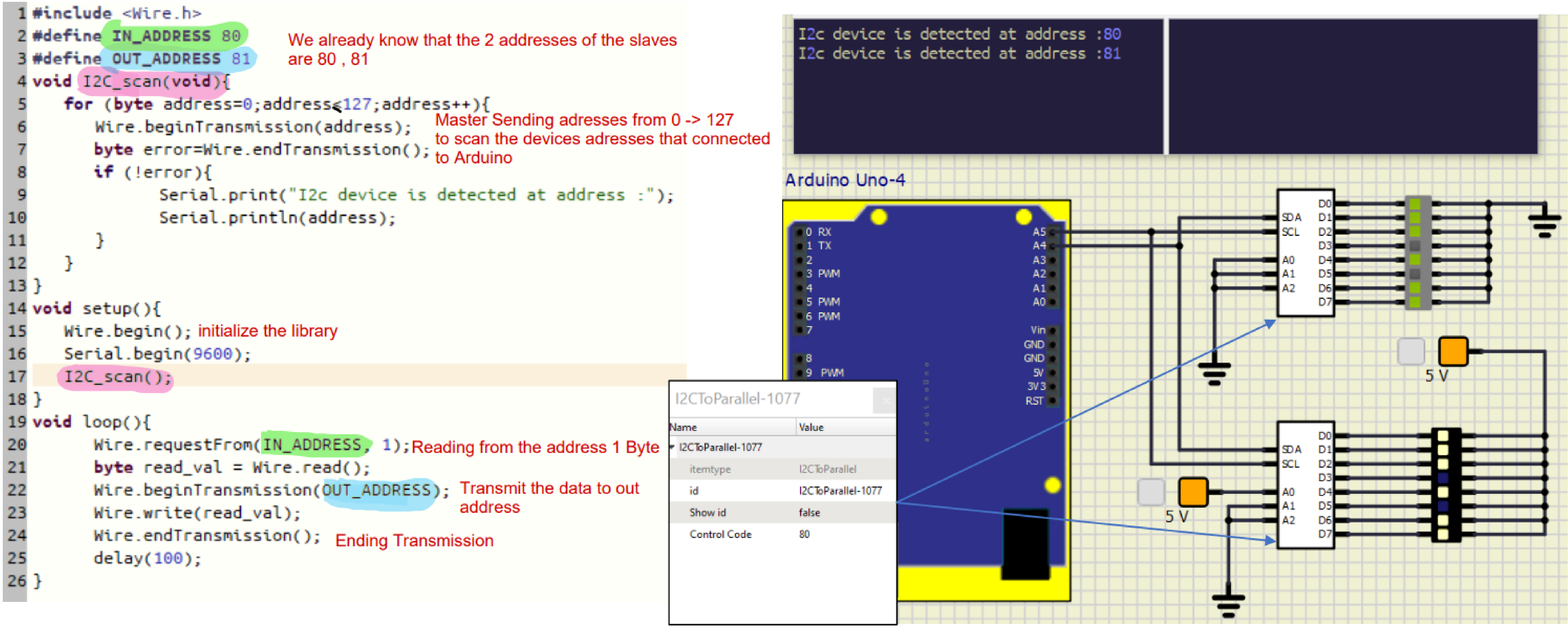
(I^{2}C) Inter Integrated Circuit (Cont.)

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);
}
```

We already know that the 2 addresses of the slaves are 80 , 81

Master Sending adresses from 0 -> 127 to scan the devices adresses that connected to Arduino

initialize the library

Reading from the address 1 Byte

Transmit the data to out address

Ending Transmission

```
I2c device is detected at address :80
I2c device is detected at address :81
```

I2CToParallel-1077

Name	Value
I2CToParallel-1077	
itemtype	I2CToParallel
id	I2CToParallel-1077
Show id	false
Control Code	80

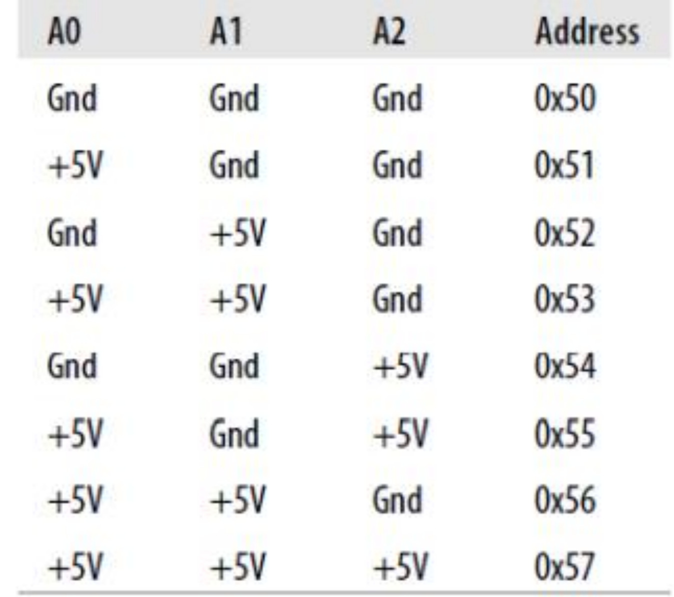
I2C Externa EEPROM:

- EEPROM: Electrically Erasable Programmable ROM

- Uses I2C for communication.

- Size: 128 Kbit

- Default Address: 0x50

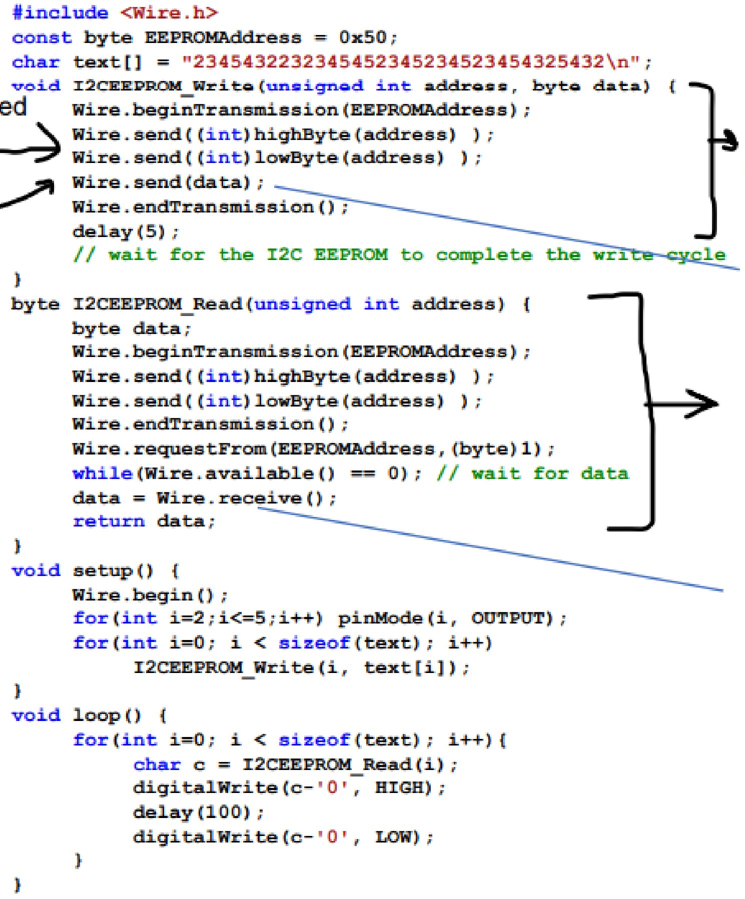


A0	A1	A2	Address
Gnd	Gnd	Gnd	0x50
+5V	Gnd	Gnd	0x51
Gnd	+5V	Gnd	0x52
+5V	+5V	Gnd	0x53
Gnd	Gnd	+5V	0x54
+5V	Gnd	+5V	0x55
+5V	+5V	Gnd	0x56
+5V	+5V	+5V	0x57

I^{2}C Externa EEPROM (Example) Cont.

here we divide sended address to 2 bytes

we send data to memory



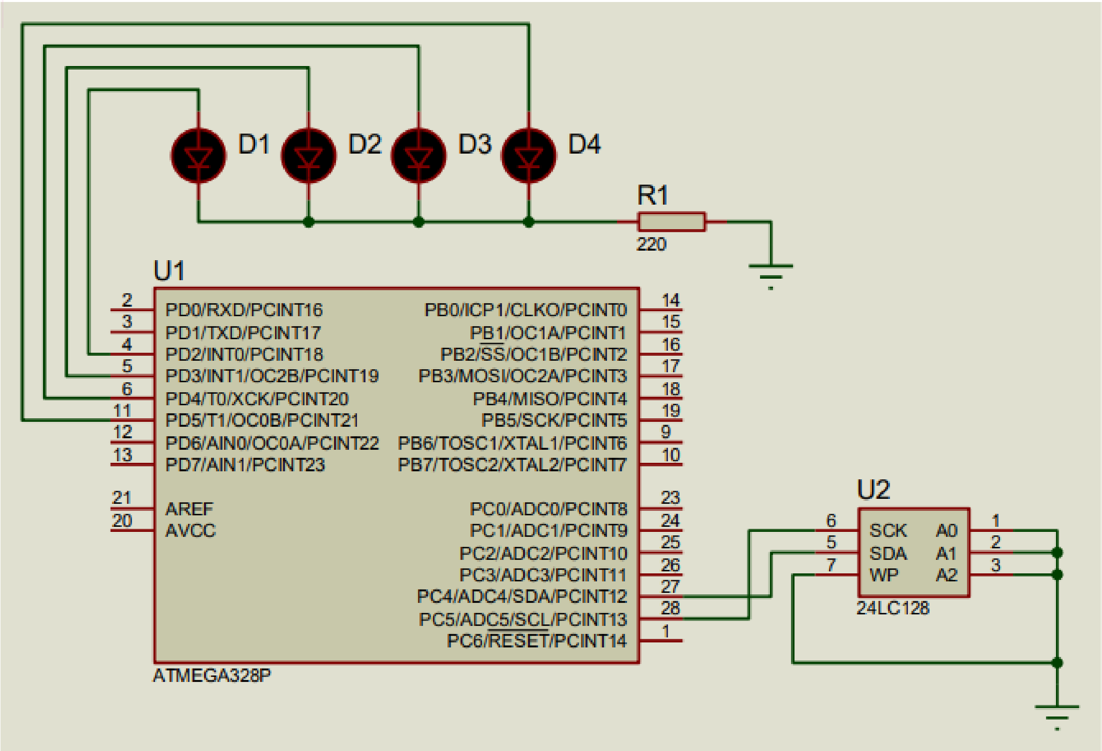
```
#include <Wire.h>
const byte EEPROMAddress = 0x50;
char text[] = "23454322323454523452345234543254 32\n";
void I2CEEPROM_Write(unsigned int address, byte data) {
      Wire.beginTransmission(EEPROMAddress);
      Wire.send((int)highByte(address) );
      Wire.send((int)lowByte(address) );
      Wire.send(data);
      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);
      }
}
```

Function to write an address and data in memory

Wire.write(data)

Function Receives data from Memory

Wire.read()



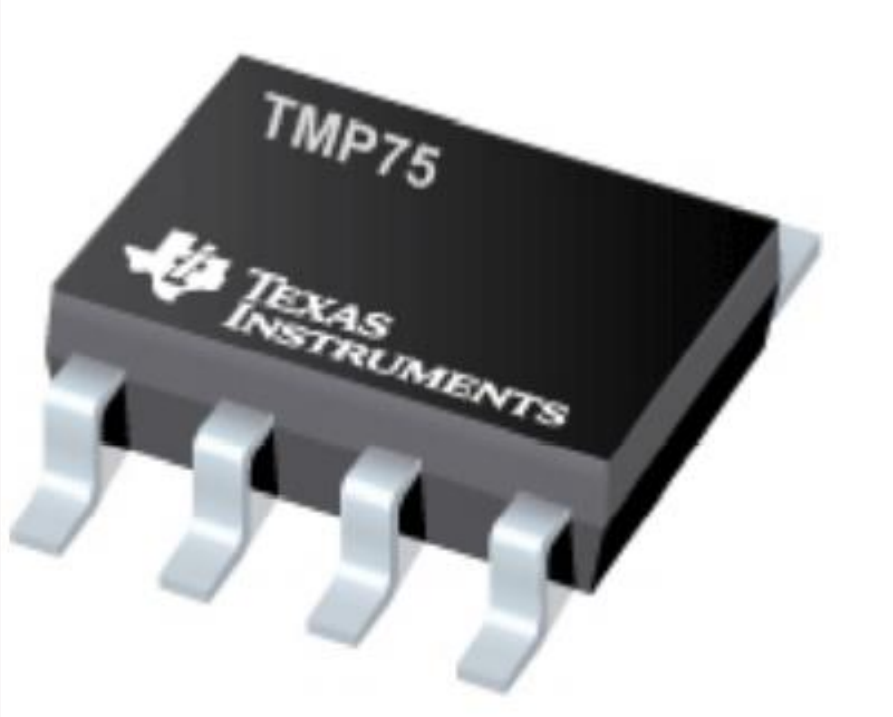
Digital Thermometer Module:

- Measure Temperature

- Uses I2C for communication.

- Range: -40 →125 Celsius degree

- Default Address: 0x49



Reading Temperature with a Digital Thermometer (Example)

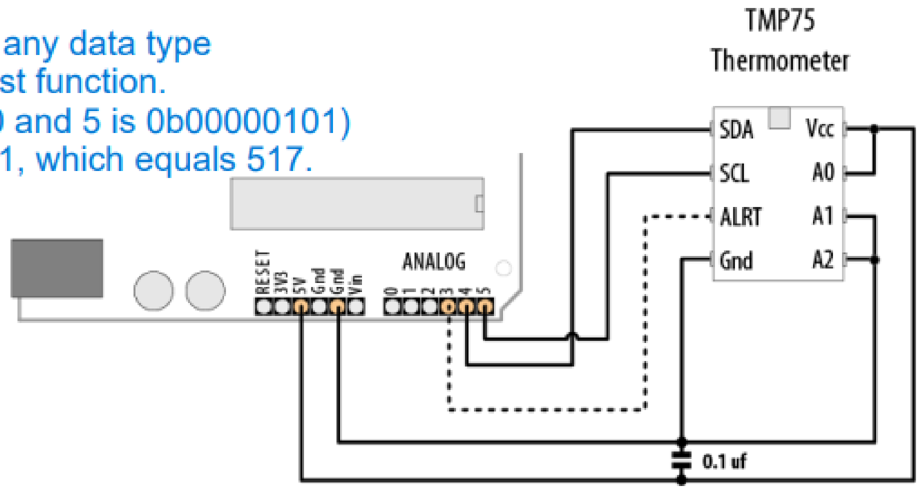
```
#include <Wire.h>
const byte TMP75Address = 0x49;
void setup() {
      Serial.begin(9600);
      //Configure and Initialize the Module
      Wire.begin();
      Wire.beginTransmission(TMP75Address);
      Wire.send(1);
      Wire.send(0);
      Wire.endTransmission();
      //Choose 12 Bit Temprature
      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);
}
```

from Data Sheet

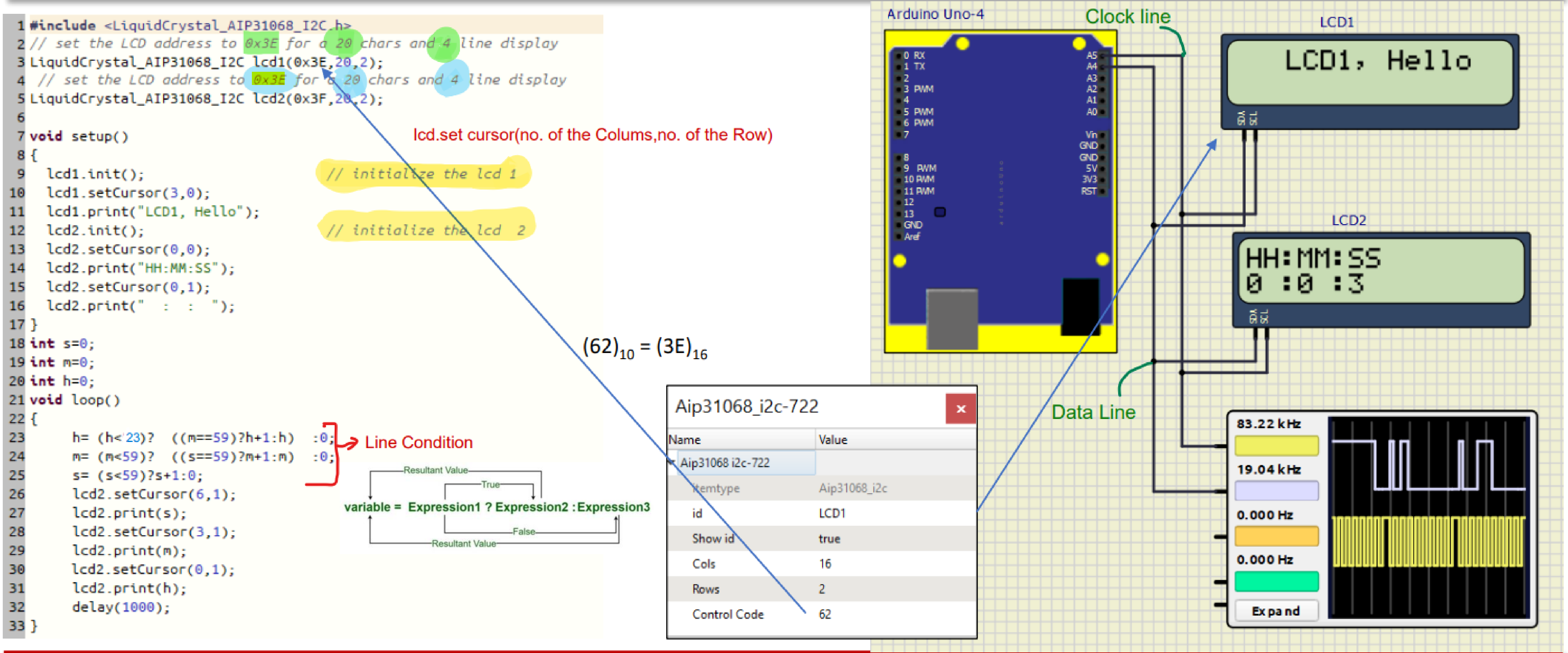
Wire.write(data)

The word() function converts a variable of any data type to the word data type. It is essentially a cast function.
word(2,5) will return 517 (2 is 0b00000010 and 5 is 0b00000101)
word(2,5) will return 0b0000001000000101, which equals 517.

Wire.read()



I^{2}C 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();
  lcd1.setCursor(3,0);
  lcd1.print("LCD1, Hello");
  lcd2.init();
  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<23)?  ((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);
}
```

lcd.set cursor(no. of the Colums,no. of the Row)

// initialize the lcd 1

// initialize the lcd 2

Line Condition

variable = Expression1 ? Expression2 : Expression3

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

Aip31068_i2c-722

Name	Value
Aip31068 i2c-722	
Itemtype	Aip31068_i2c
id	LCD1
Show id	true
Cols	16
Rows	2
Control Code	62

Software SPI VS Hardware SPI:

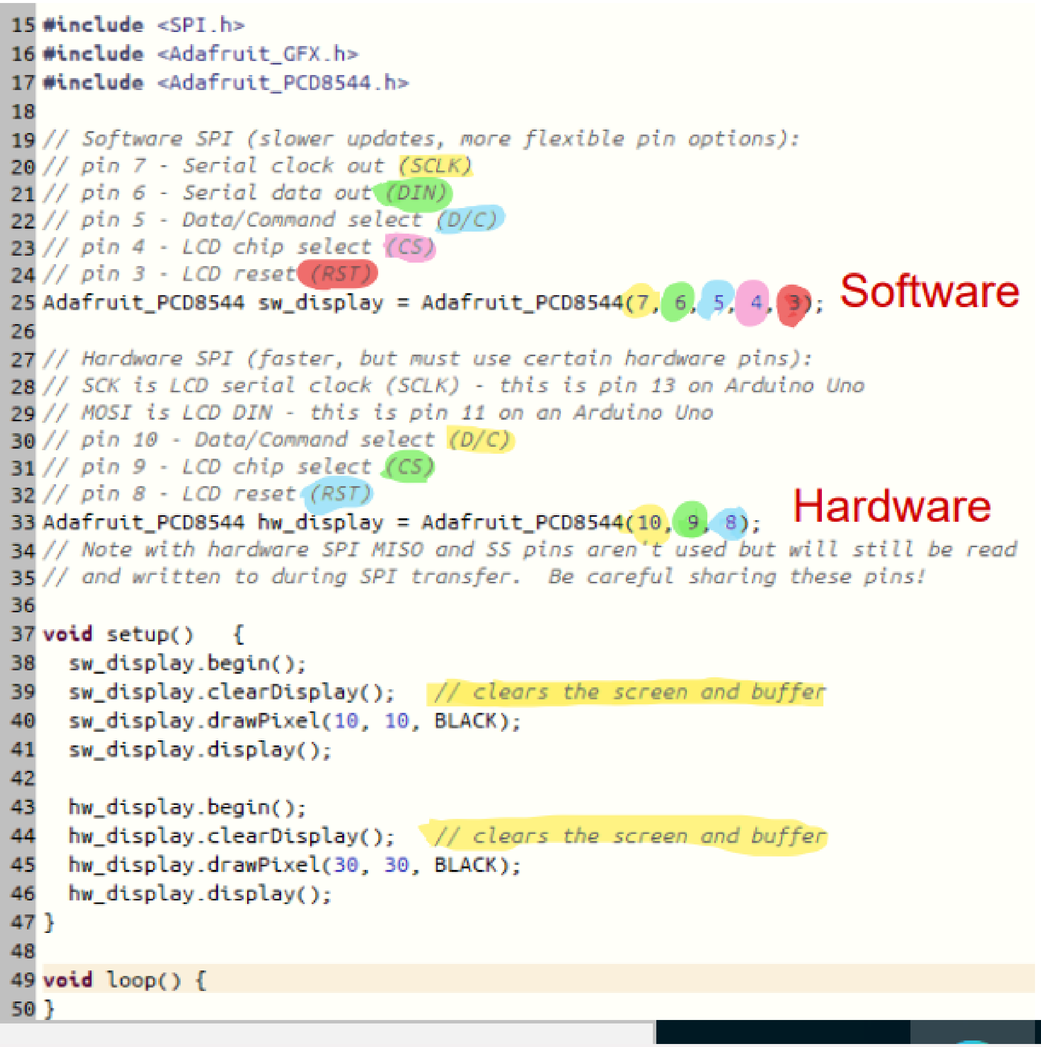
Software SPI:

Slower Updates , more flexible pins action

Hardware SPI:

faster but must use certain Hardware pins

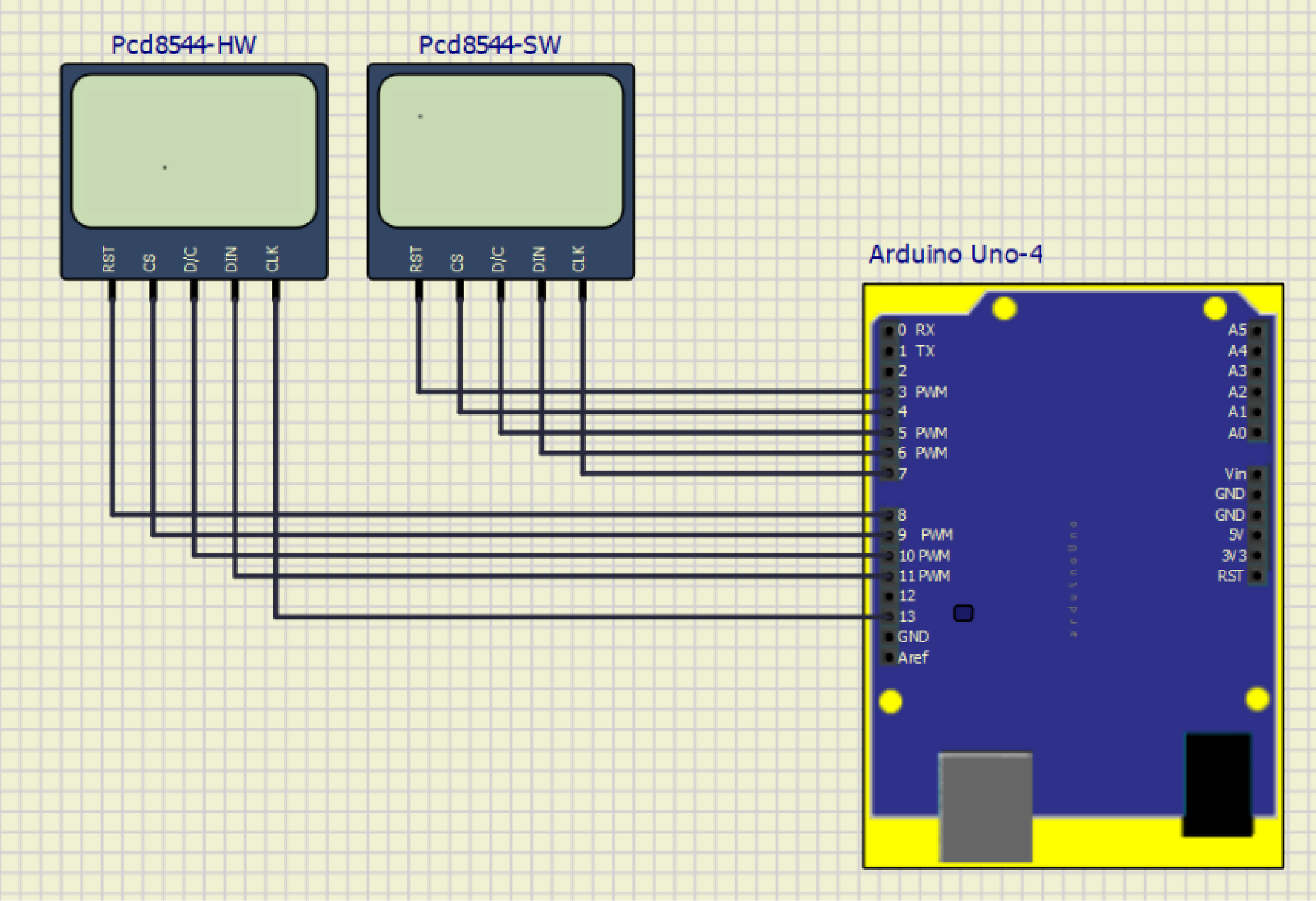
(SPI) Example (LCD SW/HW)



```
#include <SPI.h>
#include <Adafruit_GFX.h>
#include <Adafruit_PCD8544.h>

// Software SPI (slower updates, more flexible pin options):
// pin 7 - Serial clock out (SCLK)
// pin 6 - Serial data out (DIN)
// pin 5 - Data/Command select (D/C)
// pin 4 - LCD chip select (CS)
// pin 3 - LCD reset (RST)
Adafruit_PCD8544 sw_display = Adafruit_PCD8544(7, 6, 5, 4, 3);

// Hardware SPI (faster, but must use certain hardware pins):
// SCK is LCD serial clock (SCLK) - this is pin 13 on Arduino Uno
// MOSI is LCD DIN - this is pin 11 on an Arduino Uno
// pin 10 - Data/Command select (D/C)
// pin 9 - LCD chip select (CS)
// pin 8 - LCD reset (RST)
Adafruit_PCD8544 hw_display = Adafruit_PCD8544(10, 9, 8);
// Note with hardware SPI MISO and SS pins aren't used but will still be read
// and written to during SPI transfer.  Be careful sharing these pins!

void setup()   {
  sw_display.begin();
  sw_display.clearDisplay();   // clears the screen and buffer
  sw_display.drawPixel(10, 10, BLACK);
  sw_display.display();

  hw_display.begin();
  hw_display.clearDisplay();   // clears the screen and buffer
  hw_display.drawPixel(30, 30, BLACK);
  hw_display.display();
}

void loop() {
}
```

Software

Hardware


Pcd8544-HW
RST
CS
D/C
DIN
CLK
Pcd8544-SW
RST
CS
D/C
DIN
CLK
Arduino Uno-4
0 RX
1 TX
2
3 PWM
4
5 PWM
6 PWM
7
8
9 PWM
10 PWM
11 PWM
12
13
GND
Aref
A5
A4
A3
A2
A1
A0
Vin
GND
GND
5V
3V3
RST