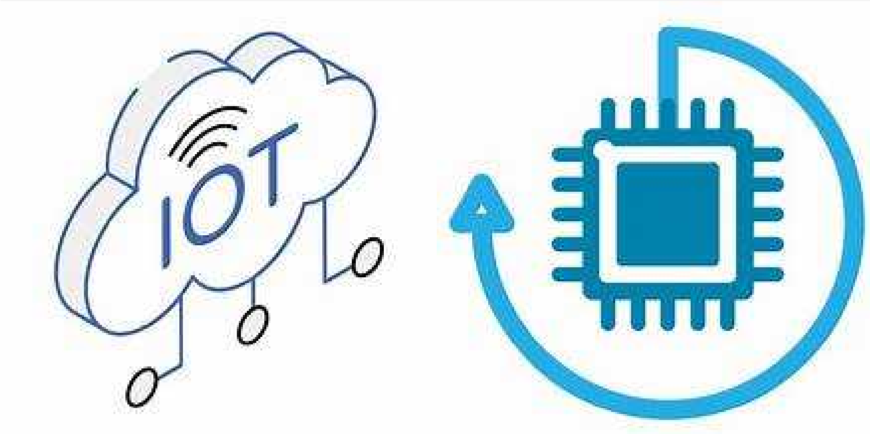
한국외국어대학교
HANKUK UNIVERSITY OF FOREIGN STUDIES

Internet of Things (IoT) Systems



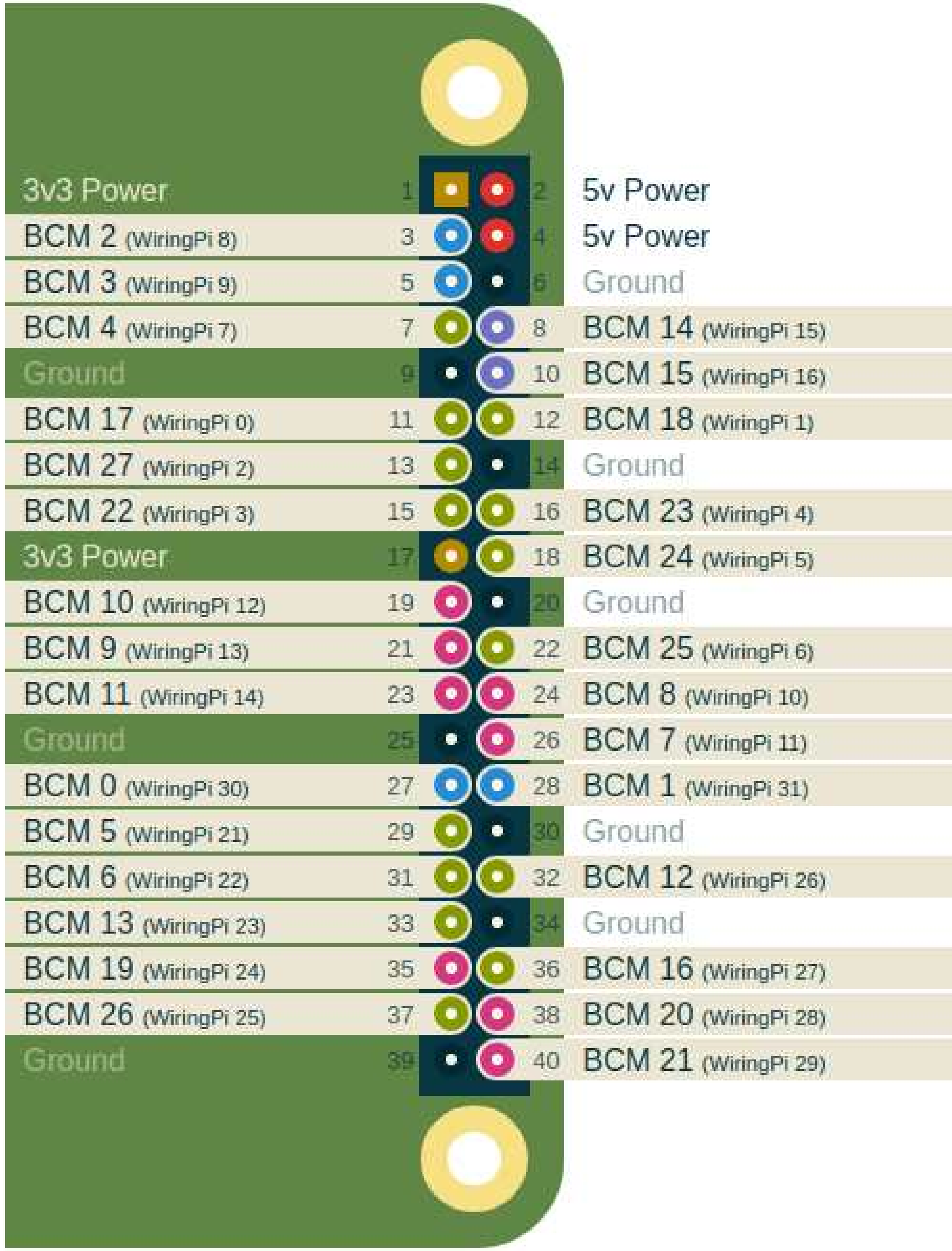
Week 5

Raspberry Pi Programming

Ikram Syed, Ph.D.
Associate Professor
Department of Information and Communication Engineering
Hankuk University of Foreign Studies (HUFS)

Spring – 2025

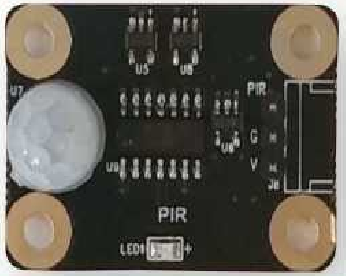
- **GPIO Pin Numbering Schemes**
  - Physical
    - The actual pin numbers on 40-pin connector
  - BCM
    - Broadcom pin numbers often called GPIO numbers
    - This is the most common method of naming the GPIO pins
  - WiringPi
    - Pin numbers used in WiringPi library



Task

- Combine the **PIR sensor** and **LED** projects so that the **LED** turns on when the **PIR sensor** detects motion (i.e., a human is detected). The LED will turn off once the motion is no longer detected.

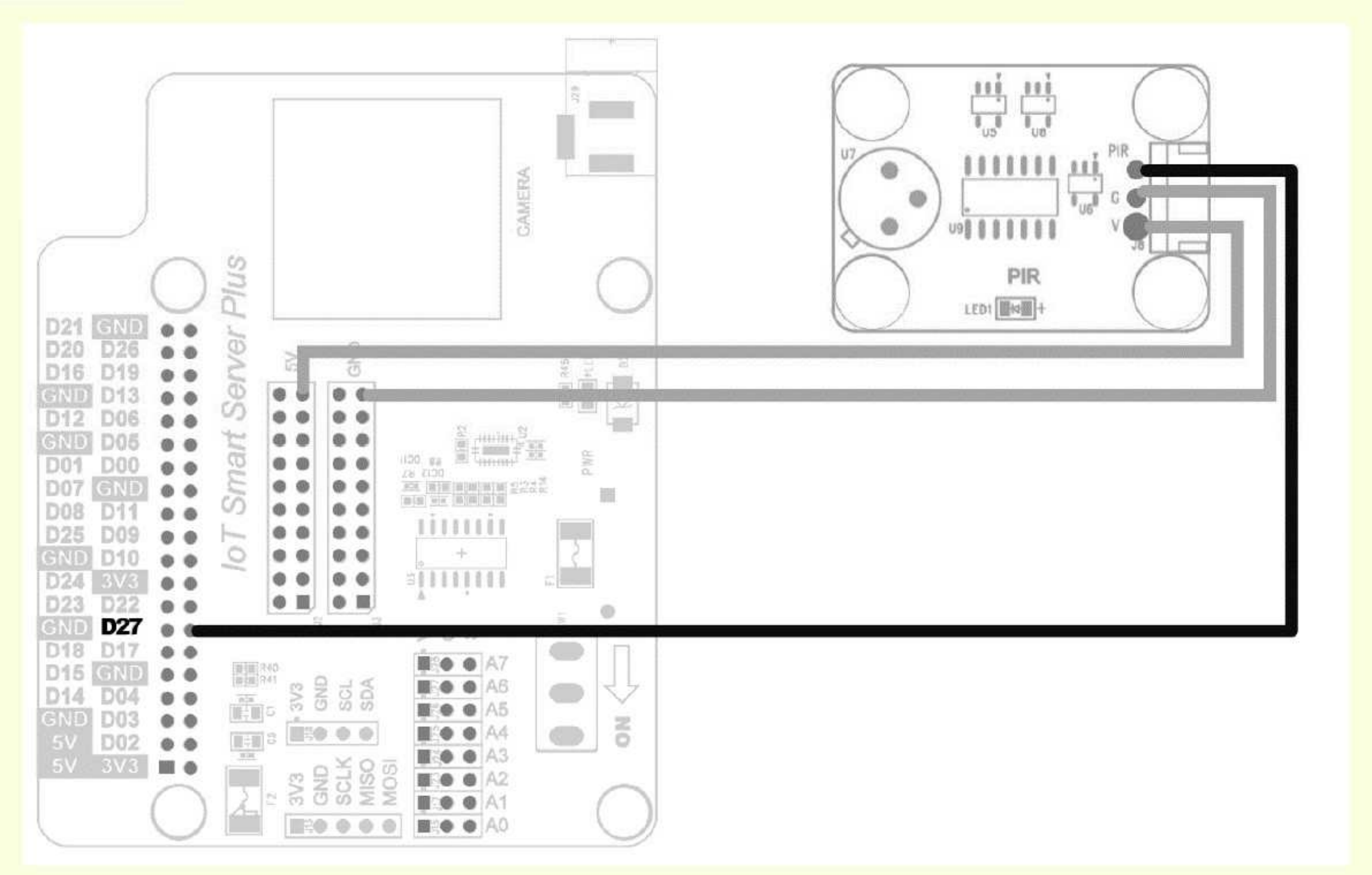
〈Table 3-3〉 Specification of human detection sensor

Shape of human detection sensor	Category	Description
	Infrared Sensor	RE200B
	Detection Range	110 degree
	Operating Voltage	3.3V
	I/O Pin	1 unit of 3-pin header (2.54mm pitch)

〈Table 3-4〉 Pin connection for Raspberry Pi and PIR Sensor

GPIO	Wiring Pi Pin No.	Pin Info.	PIR Sensor Pin No.
27	2	GPIO	PIR

Connect PIR sensor module without applying power to RPi



[Figure 3-9] Pin Connections for PIR Sensor

Check Input Pin States

- Ways to check input pin states
  - Polling
  - Interrupts
- Suppose you are waiting for an important email and want to open it as soon as it arrives
  - Poll check email every x time units
  - Interrupt activate a notification bell so you get a popup on your screen as soon as the email arrives
- An interrupt is a signal that temporarily halts the execution of the main program to execute a special function, called an Interrupt Service Routine (ISR).

Interrupts

- Interrupts will be triggered when a signal's state changes
- There are two kind of interrupts
  - RISING when the state goes from LOW to HIGH
  - FALLING when the state goes from HIGH to LOW

  - In your program if you set up a FALLING interrupt, then your program will be notified as soon as the signal goes from HIGH to LOW

Interrupts

```
int wiringPiISR (int pin, int edgeType,  void (*function)(void)) ;
```

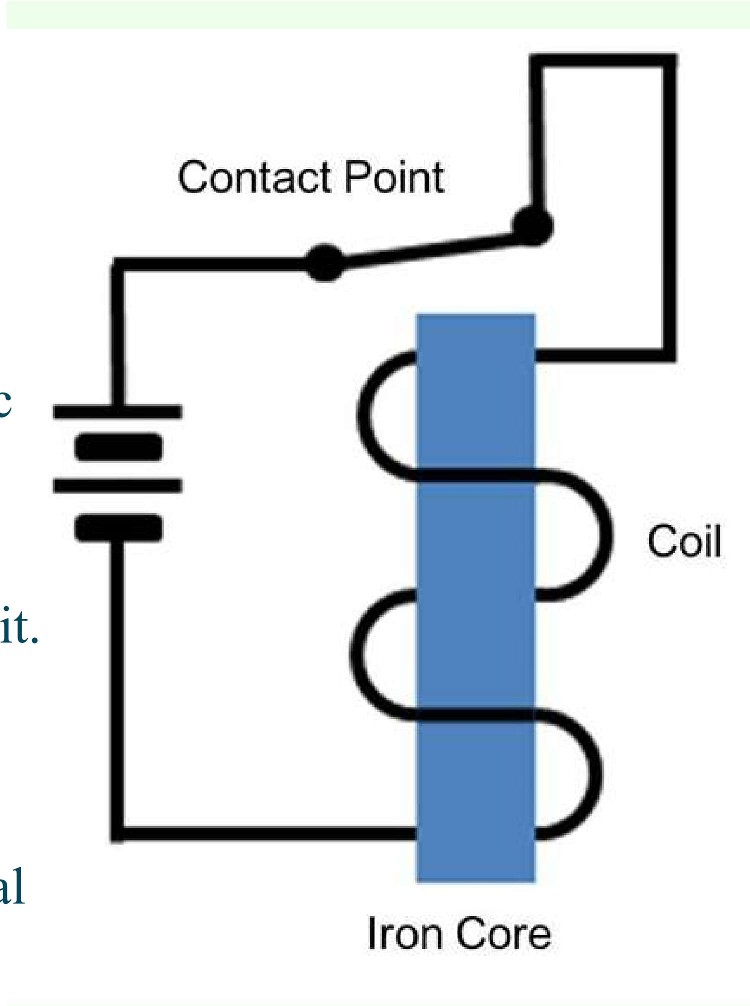
- This function registers a function to received interrupts on the specified pin
- The edgeType parameter is either
  - INT_EDGE_FALLING
  - INT_EDGE_RISING
  - INT_EDGE_BOTH

```c
#include <wiringPi.h>
#include <stdio.h>

#define PIN 2
#define CONTROL_PIN 7

void edge_rise(void);
void edge_fall(void);

int main(void){

    if(wiringPiSetup() == -1) return 1;
    pinMode(PIN,INPUT);

    wiringPiISR(PIN, INT_EDGE_RISING, edge_rise);

    delay(10000);
}


void edge_rise(void){
    pinMode(CONTROL_PIN,OUTPUT);
    digitalWrite(CONTROL_PIN,HIGH);
    wiringPiISR(PIN,INT_EDGE_FALLING, edge_fall);
    printf("Edge_Rised.\n");
}

void edge_fall(void){
    pinMode(CONTROL_PIN,OUTPUT);
    digitalWrite(CONTROL_PIN,LOW);
    wiringPiISR(PIN,INT_EDGE_RISING, edge_rise);
    printf("Edge_Falled.\n");
}

```

When movement is detected around the sensor, the LED is turned ON. If no movement is detected, the LED is turned OFF

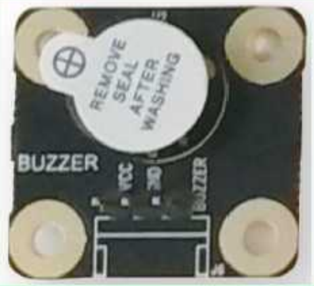
Buzzer control

- Buzzer control
  - HIGHT input to buzzer -> sound
  - LOW input to buzzer -> no sound
- Initial State: When the circuit is closed, current flows.
- Electromagnet Formation: The coil generates a magnetic field.
- Contact Point Displacement: The magnetic field attracts the contact point, causing it to move and break the circuit.
- Loss of Magnetism: Once the circuit is broken, current stops flowing, and the iron core loses its magnetism.
- Contact Restores: The contact point returns to its original position, completing the circuit again.
- Repetition: This cycle repeats rapidly, creating a continuous vibrating motion that produces a buzzing sound.



Buzzer control

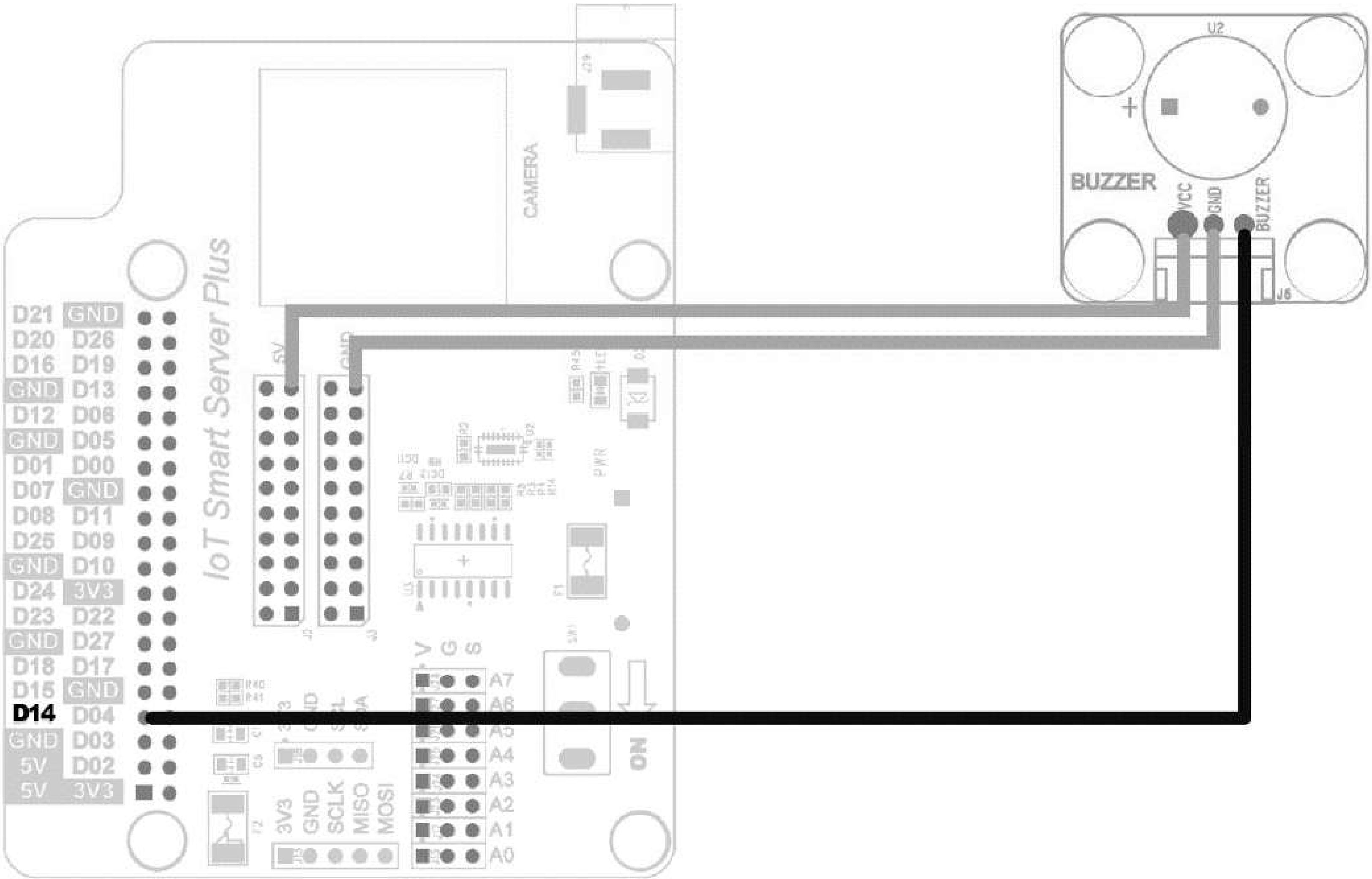
〈Table 3-8〉 The specification of buzzer module

Shape	Category	Description
	Operating Voltage	5V
	Sound Level	88dB

〈Table 3-9〉 Pin Connection Information for Raspberry Pi and Buzzer

GPIO	Wiring Pi Pin No.	Pin Info.	Buzzer Module Pin No.
14	15	GPIO	BUZZER

Connect module without applying power to RPi



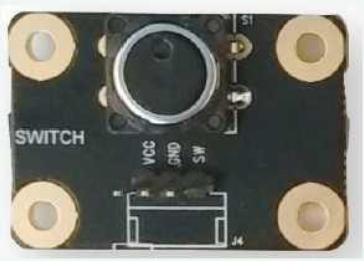
```c
#include <wiringPi.h>

#define PIN 15

int main(void){
        if(wiringPiSetup() == -1) return 1;

        pinMode(PIN,OUTPUT);

        digitalWrite(PIN,HIGH);
        delay(500);
        digitalWrite(PIN,LOW);
}
```

```
pi@raspberrypi:~ $ gcc -o SMART_BUZZER SMART_BUZZER.c -lwiringPi
pi@raspberrypi:~ $ sudo ./SMART_BUZZER
```

Switch Module

- **Switch Module**
  - It is a push button
  - PIN value LOW when button is pressed

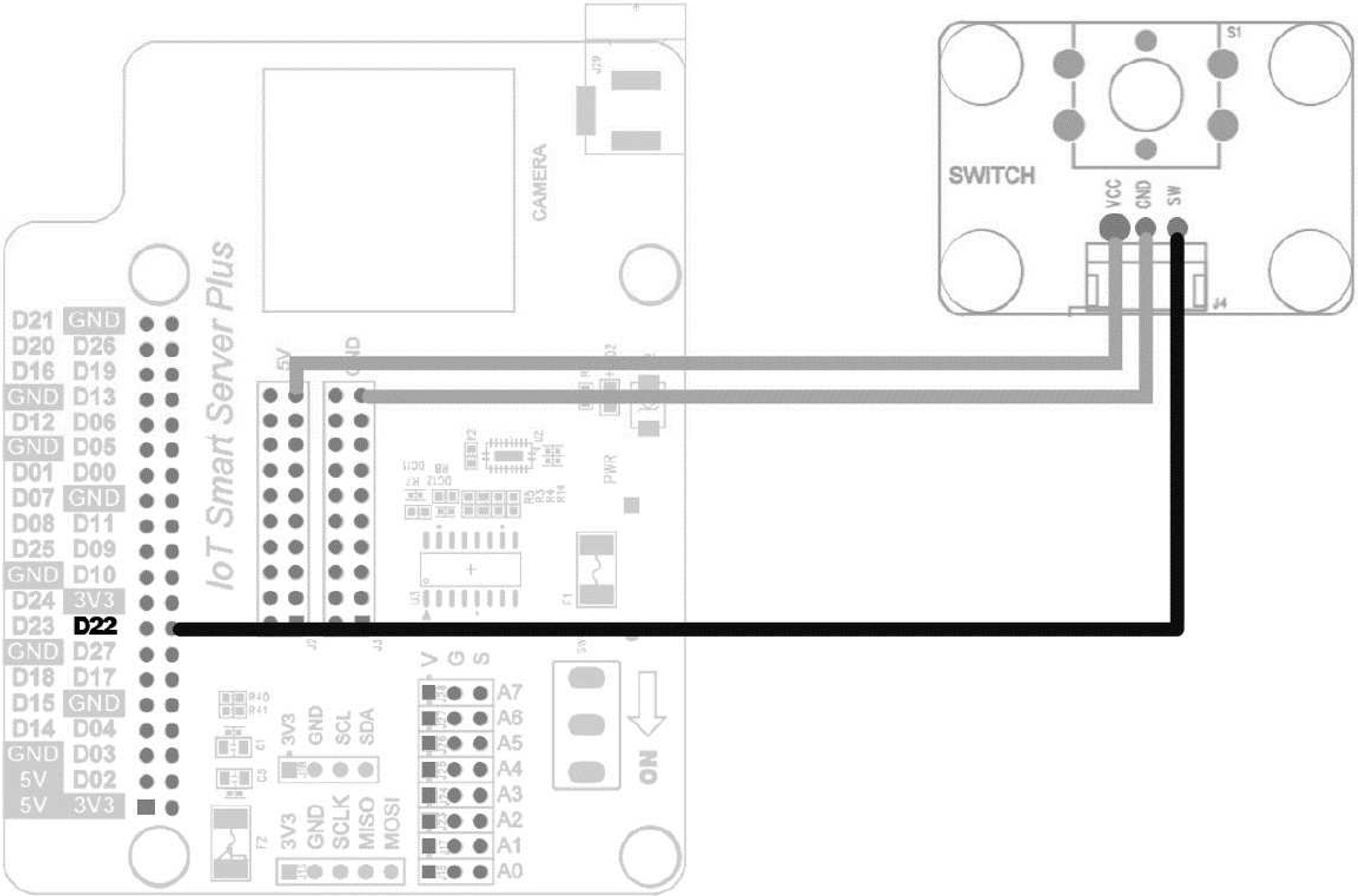
<Table 3-15> Specifications of Switch Module

Shape	Category	Description
	Sensor	Button Switch
	Interface	1pin Digital OUTPUT
	Operating Voltage	3.3V~5V

<Table 3-16> Pin Connection Information of Raspberry Pi and Switch Module

GPIO	Wiring Pi Pin No.	Pin Info.	Switch Pin No.
22	3	GPIO	SW

Connect module without applying power to RPi



```c
#include <wiringPi.h>
#include <stdio.h>

#define PIN 3

int main(void){
		int sw, i;

		if(wiringPiSetup() == -1) return 1;
		pinMode(PIN,INPUT);

		for(i=0; i<20; i++){
			sw = digitalRead(PIN);
		printf("%d\n",sw);
		delay(100);
	}
}
```

```
pi@raspberrypi:~ $ gcc -o SMART_SWITCH SMART_SWITCH.c -lwiringPi
pi@raspberrypi:~ $ sudo ./SMART_SWITCH
1
1
1
1
1
0
0
0
```

Change this code for Turn on LED when Switch is pressed

```c
#include <wiringPi.h>
#include <stdio.h>

#define PIN 3
#define CONTROL_PIN 7

void edge_rise(void);
void edge_fall(void);

int main(void){

    if(wiringPiSetup() == -1) return 1;
    pinMode(PIN,INPUT);

    wiringPiISR(PIN, INT_EDGE_RISING, edge_rise);

    delay(10000);
}


void edge_rise(void){
    pinMode(CONTROL_PIN,OUTPUT);
    digitalWrite(CONTROL_PIN,HIGH);
    wiringPiISR(PIN,INT_EDGE_FALLING, edge_fall);
    printf("Edge_Rised.\n");
}

void edge_fall(void){
    pinMode(CONTROL_PIN,OUTPUT);
    digitalWrite(CONTROL_PIN,LOW);
    wiringPiISR(PIN,INT_EDGE_RISING, edge_rise);
    printf("Edge_Falled.\n");
}
```

DC Motor

- DC Motor
  - A direct current (DC) motor is a type of electric machine that converts electrical energy into mechanical energy
  - It take electrical power through direct current, and convert this energy into mechanical rotation
  - Applications of DC motors
    - Fans
    - Toys
    - Electric cars
    - Robots
    - Electric bikes

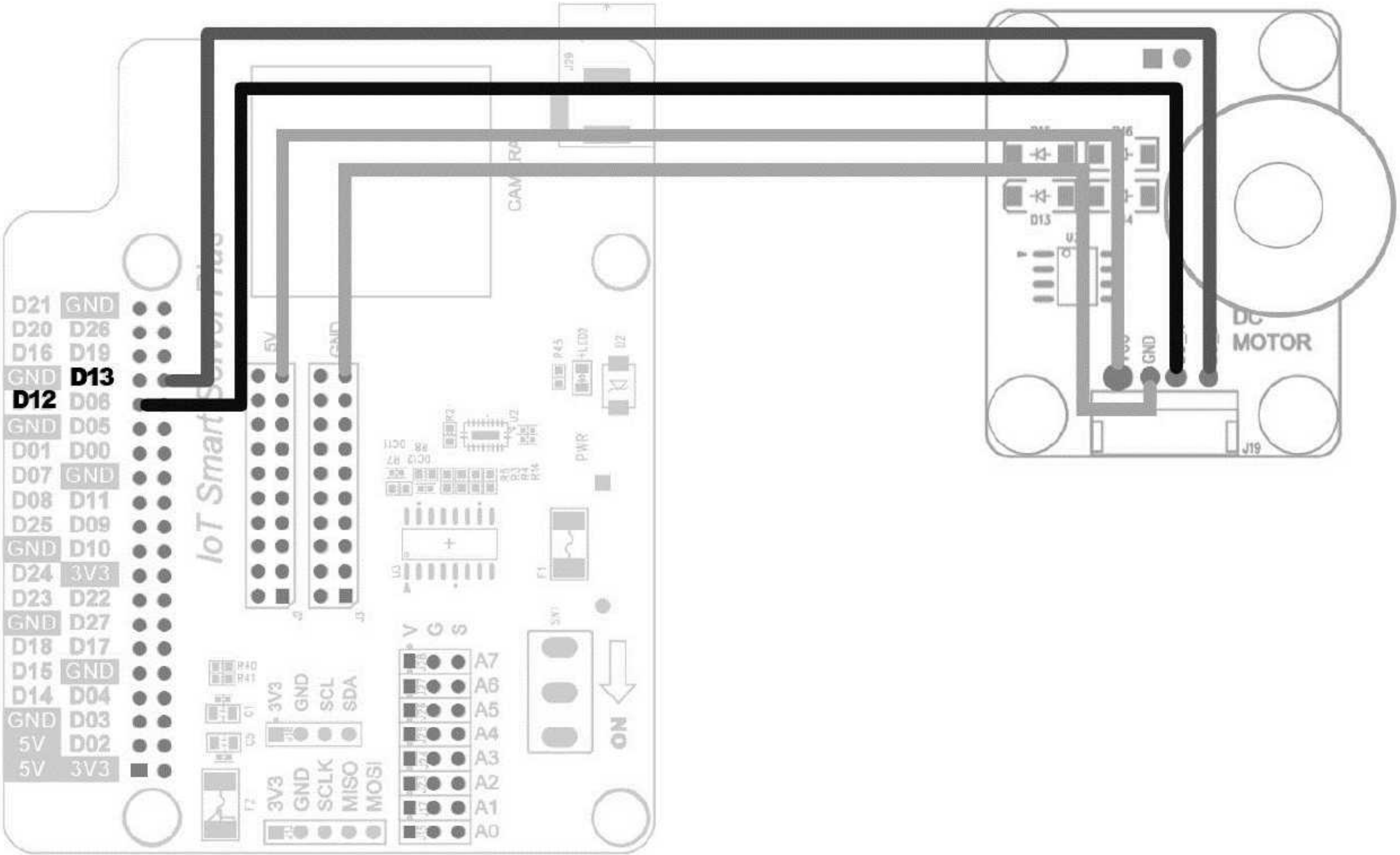
<Table 3-10> Specifications of DC Motor Module

Shape	Category	Description
	Motor Driver	BA6208F
	Moter	Micro Type DC Motor
	Operating Voltage	5V
DC Motor Module as Actuator		

<Table 3-11> Pin Connection Information for Raspberry Pi and DC Motor

GPIO	Wiring Pi Pin No.	Pin Info.	DC Motor Pin No.
12	26	GPIO	INA
13	23	GPIO	INB

Connect module without applying power to RPi

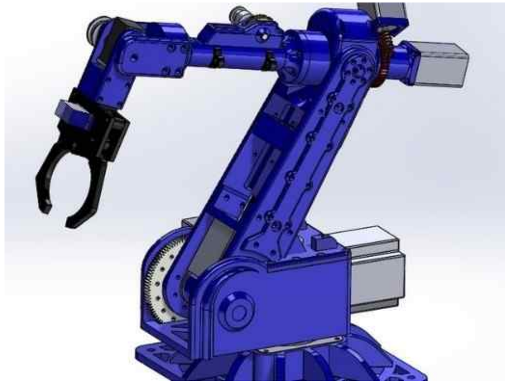


```c
#include <wiringPi.h>

#define PIN_INA 26
#define PIN_INB 23

int main(void){
	if(wiringPiSetup() == -1) return 1;

	pinMode(PIN_INA,OUTPUT);
	pinMode(PIN_INB,OUTPUT);

	digitalWrite(PIN_INA,HIGH);
	delay(2000);
	digitalWrite(PIN_INA,LOW);
	digitalWrite(PIN_INB,HIGH);
	delay(2000);
	digitalWrite(PIN_INB,LOW);
}
```

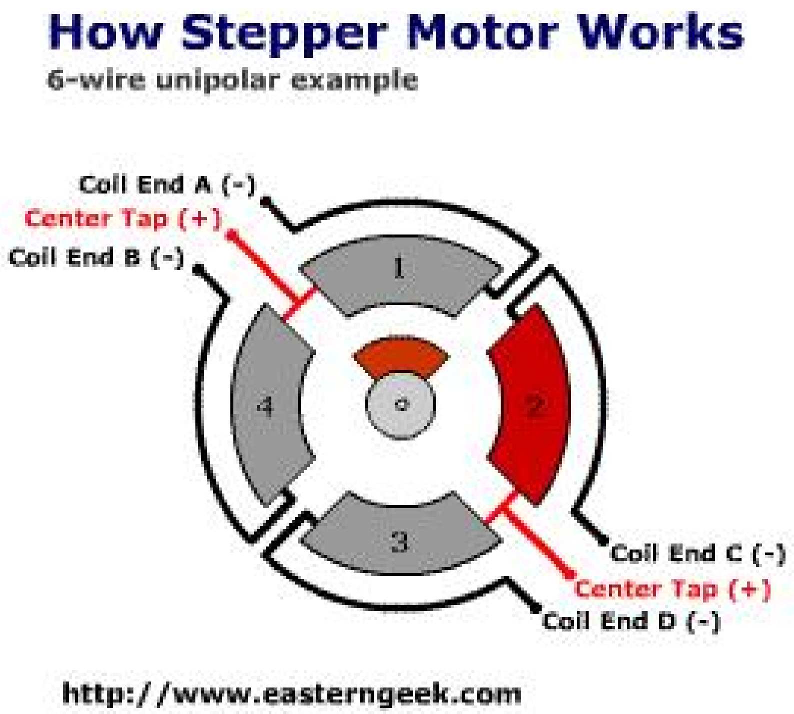
Rotate forward for 2 seconds and then rotate reverse for 2 seconds

```
pi@raspberrypi:~ $ gcc -o SMART_DCMOTOR SMART_DCMOTOR.c -lwiringPi
pi@raspberrypi:~ $ sudo ./SMART_DCMOTOR
```

Step Motor



- Step Motor
  - Step motors are DC motors that move in discrete steps
  - Precision motion control and positioning systems



Imagine the there are 4 coils in a stepper motor. If any one of the coil is energized, the motor will make one step, then stays in that place. In order for the motor to complete one full revolution, it needed to make multiple steps. The coils need to be energized in the proper sequence to achieve this.

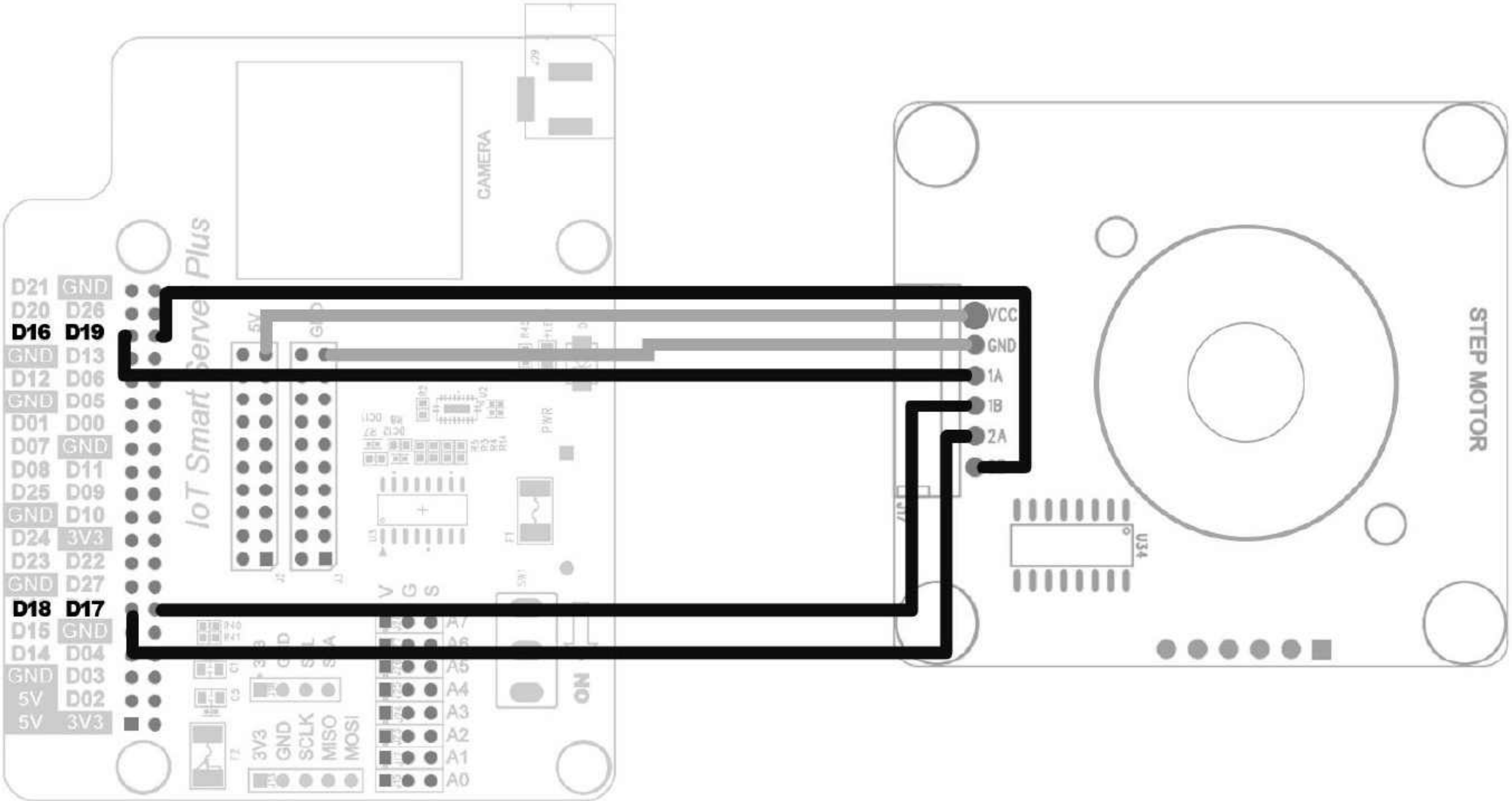
〈Table 3-13〉 The specification of step motor

Shape	Category	Description
	Motor	Step Motor (PF25-48)
	Step Angle	7.5°
	Operating Voltage	5V
Step Motor Module as Actuator		

〈Table 3-14〉 Pin Connection Information for Raspberry Pi and Step Motor

GPIO	Wiring Pi Pin No.	Pin Info.	Step Motor Pin No.
16	27	GPIO	1A
17	0	GPIO	1B
18	1	GPIO	2A
19	24	GPIO	2B

Connect module without applying power to RPi



```c
#include <wiringPi.h>
#define PIN_1A 27
#define PIN_1B 0
#define PIN_2A 1
#define PIN_2B 24
int main(void){
        int i=0;
        if(wiringPiSetup() == -1) return 1;
        pinMode(PIN_1A,OUTPUT);
        pinMode(PIN_1B,OUTPUT);
        pinMode(PIN_2A,OUTPUT);
        pinMode(PIN_2B,OUTPUT);
        for(i=0; i<500; i++){
                digitalWrite(PIN_1A,HIGH);
                digitalWrite(PIN_1B,LOW);
                digitalWrite(PIN_2A,LOW);
                digitalWrite(PIN_2B,LOW);
                usleep(2000);
                digitalWrite(PIN_1A,LOW);
                digitalWrite(PIN_1B,HIGH);
                digitalWrite(PIN_2A,LOW);
                digitalWrite(PIN_2B,LOW);
                usleep(2000);
                digitalWrite(PIN_1A,LOW);
                digitalWrite(PIN_1B,LOW);
                digitalWrite(PIN_2A,HIGH);
                digitalWrite(PIN_2B,LOW);
                usleep(2000);
                digitalWrite(PIN_1A,LOW);
                digitalWrite(PIN_1B,LOW);
                digitalWrite(PIN_2A,LOW);
                digitalWrite(PIN_2B,HIGH);
                usleep(2000);
        }
}
```

Step motor rotates for *x* seconds

The function *usleep* suspends the current process for the number of microseconds passed to it.

<unistd.h>

```
pi@raspberrypi:~ $ gcc -o SMART_STEPMOTOR SMART_STEPMOTOR.c -lwiringPi
pi@raspberrypi:~ $ sudo ./SMART_STEPMOTOR
```

Access and Control of IoT Devices

- Three most common protocols
  - Serial Peripheral Interface (SPI)
  - Inter-Integrated Circuit (I2C)
  - Universal Asynchronous Receiver Transmitter (UART)

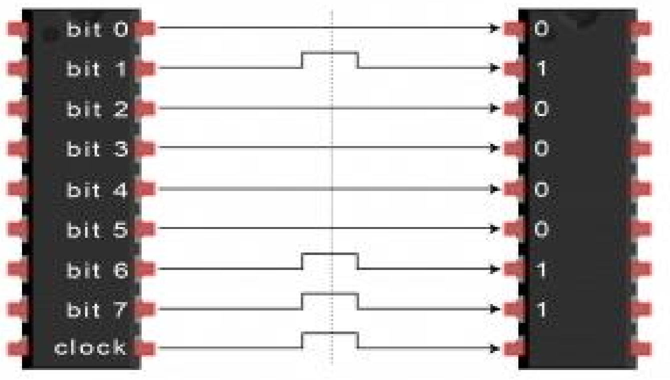
These protocols are ideal for communication between microcontrollers and between microcontrollers and sensors, especially when:

- SPI is preferred for high-speed data transfer
- I2C is efficient for multi-device communication with fewer wires.
- UART is commonly used for serial communication with peripherals

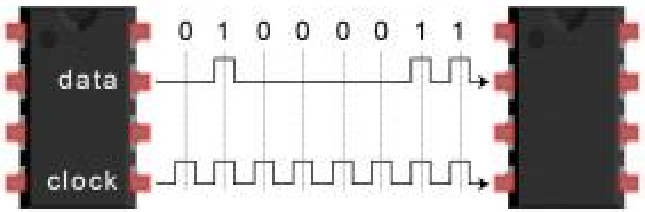
source: www.circuitbasics.com

Access and Control of IoT Devices

- In parallel communication the bits of data are sent all at the same time, each through a separate wire



- In serial communication the bits are sent one by one through a single wire



source: www.circuitbasics.com

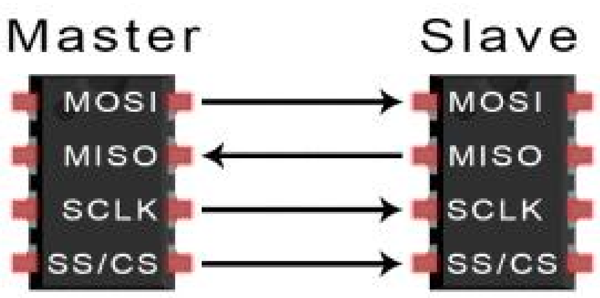
Serial Peripheral Interface (SPI)

- SPI is a common communication protocol used for two-way communication between two devices
  - SD card modules, RFID card reader modules, and 2.4 GHz wireless transmitter and receivers all use SPI to communicate with microcontrollers
- Data can be transferred without interruption
  - Any number of bits can be sent or received in a continuous stream

source: www.circuitbasics.com

Serial Peripheral Interface (SPI)

- Master-slave relationship
  - The master is the controlling device
  - The slave takes instruction from the master
  - The simplest configuration of SPI is a single master, single slave system, but one master can control more than one slave
    - Master Output Slave Input (MOSI) Line for master to send data to slave
    - Master Input Slave Output (MISO) Line for slave to send data to master
    - Slave Select / Chip Select (SS/CS) Line for master to select which slave to send data to
    - Clock (SCLK) Line for clock signal



source: www.circuitbasics.com

SPI Protocol	
Wires used	4
Maximum Speed	Up to 10 Mbps
Synchronous or Asynchronous	Synchronous
Serial or Parallel	Serial
Max Masters	1
Max Slaves	Theoretically unlimited

source: www.circuitbasics.com

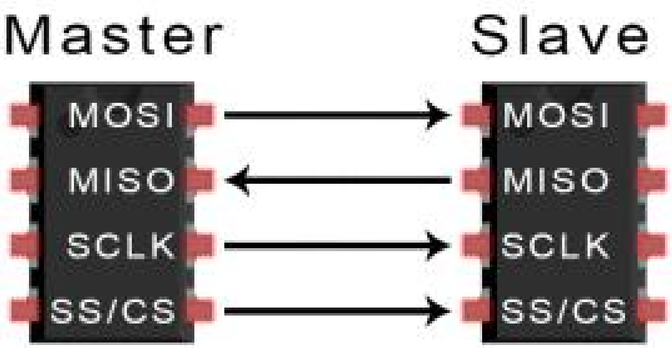
How SPI Works

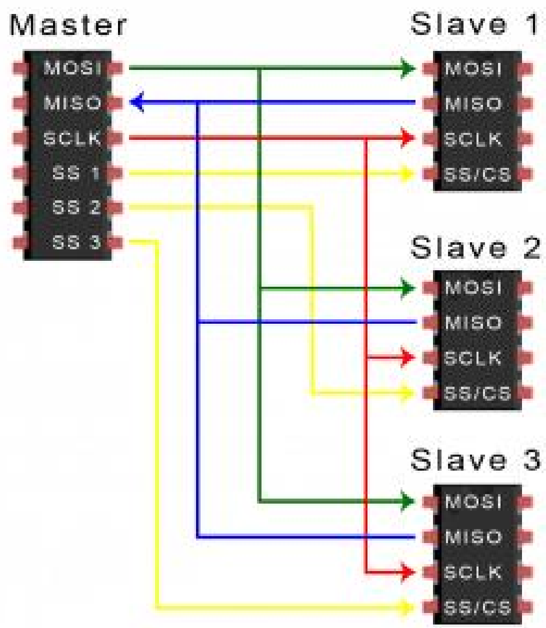
- The Clock
  - SPI is a synchronous communication protocol
    - Any protocol where devices share a clock signal is known as synchronous
  - Communication is always initiated by master
    - Master configures and generates clock signal
  - One bit of data is transferred in each clock cycle
    - The speed of data transfer is determined by frequency of clock signal

source: www.circuitbasics.com

How SPI Works

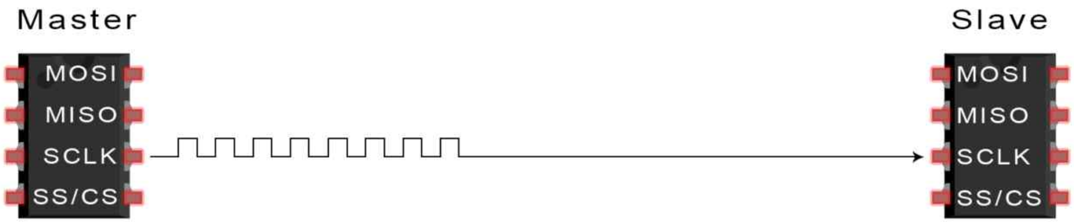
- Slave Select
  - The master can choose which slave it wants to talk to by setting the slave's CS/SS line to a low voltage level
    - In idle state the slave select line is kept at a high voltage level
- Multiple Slaves
  - A single master and a single slave
  - A single master and multiple slaves



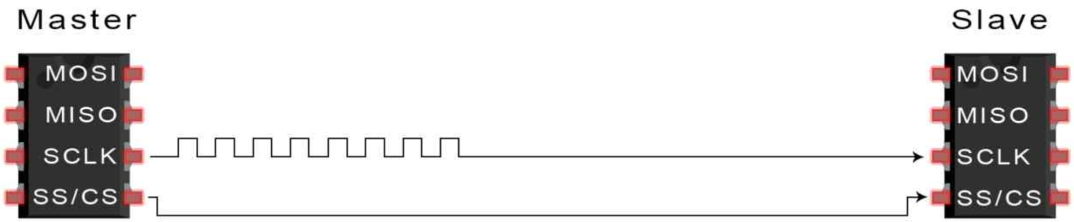


source: www.circuitbasics.com

- Data Transmission
  - The master outputs the clock signal



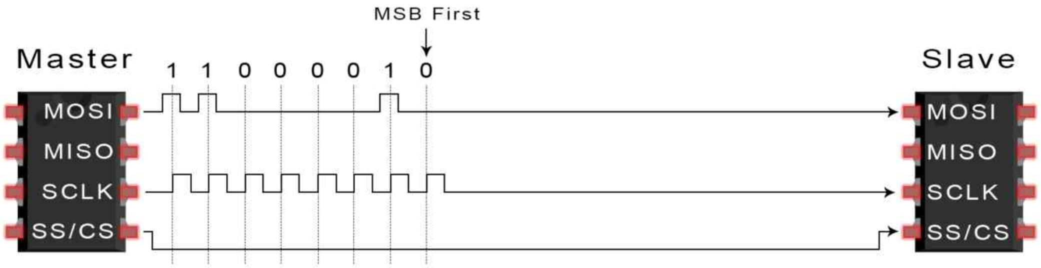
  - The master switches SS/CS pin to a low voltage state, which activates slave



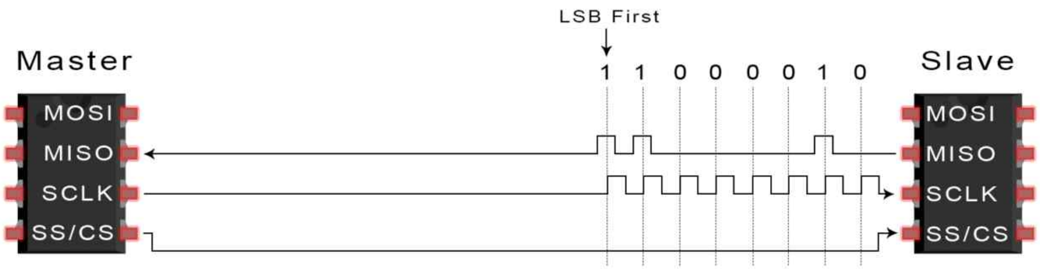
source: www.circuitbasics.com

SPI

- Data Transmission
  - The master sends data one bit at a time to slave via MOSI line



  - If a response is needed, slave returns data one bit at a time to master via MISO line



There is no way for a slave to opt-out of sending data when the master makes a transfer, however, devices will send dummy bytes usually all 1's or all 0's when communication should be one way.

source: www.circuitbasics.com

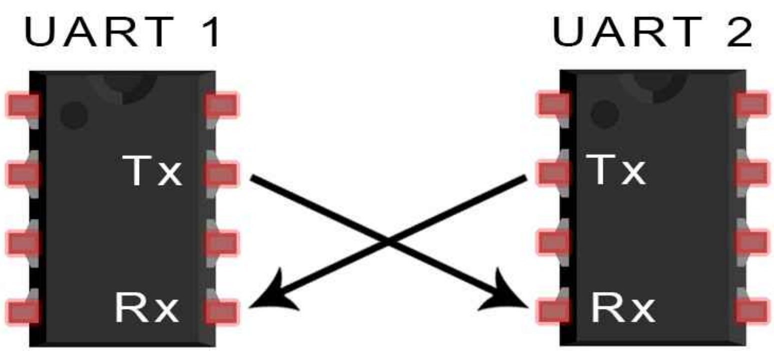
SPI

- Advantages
  - No start and stop bits, so data can be streamed continuously without interruption
  - No complicated slave addressing system
  - Higher data transfer rate
  - Separate MISO and MOSI lines, so data can be sent and received at the same time
- Disadvantages
  - Uses four wires
  - No acknowledgement
  - No form of error checking
  - Allows for a single master

source: www.circuitbasics.com

UART

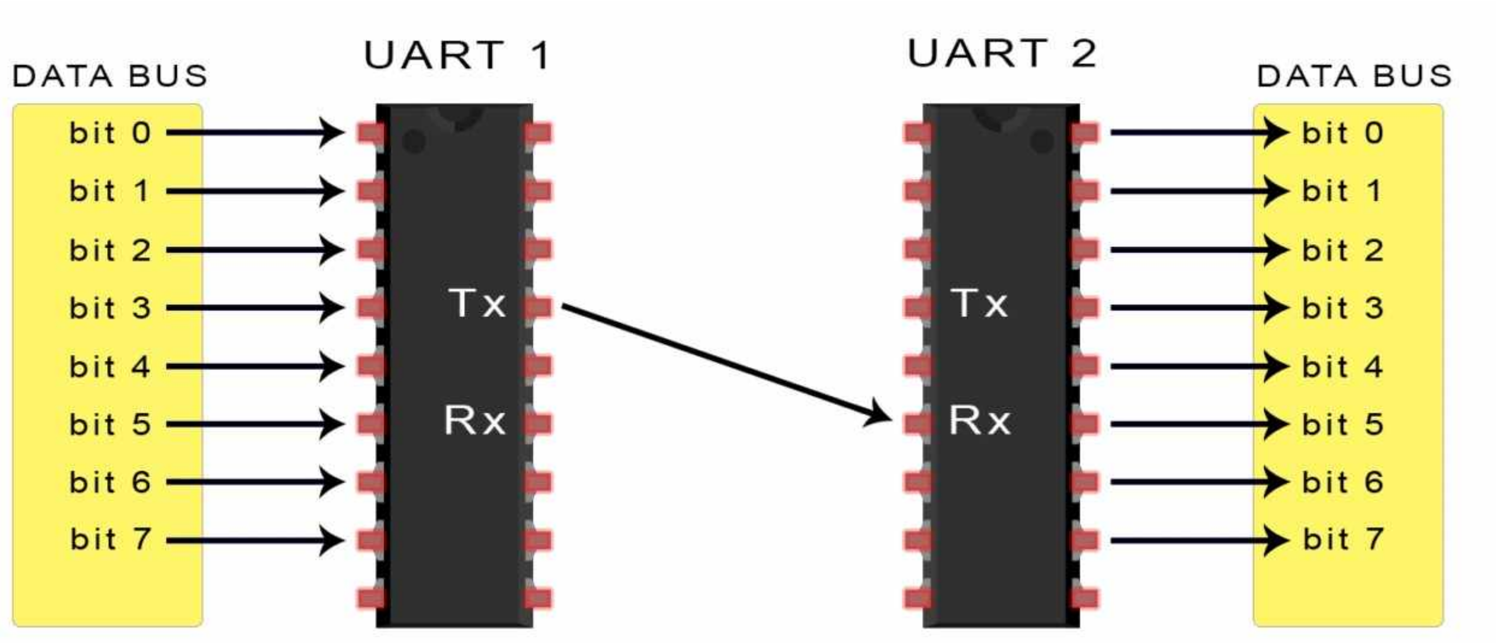
- Universal Asynchronous Receiver Transmitter (UART)
  - Two UARTs communicate directly with each other
  - Only two wires are needed to transmit data between two UARTs
  - Data flows from Tx pin of transmitting UART to Rx pin of receiving UART



source: www.circuitbasics.com

UART

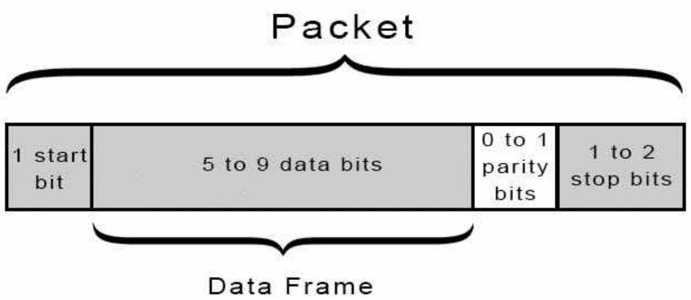
- UART Communication
  - The transmitting UART converts parallel data from a controlling device like a CPU into serial form, transmits it in serial to the receiving UART, which then converts the serial data back into parallel data for the receiving device.

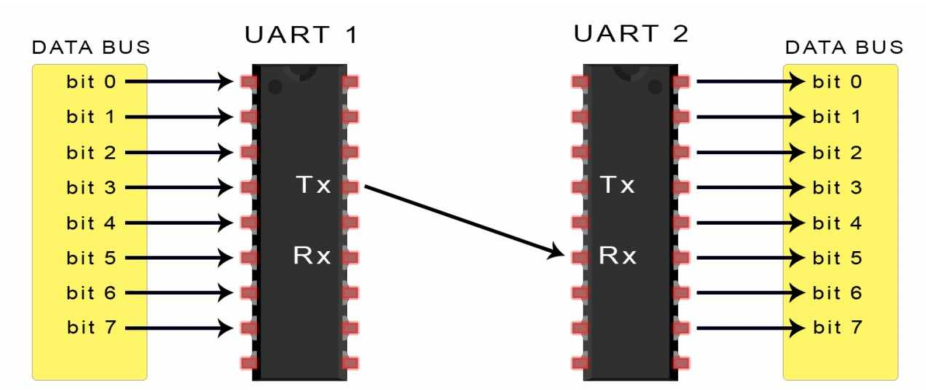


source: www.circuitbasics.com

UART

- UART Communication
  - The UART that is going to transmit data receives the data from a data bus
    - The data bus is used to send data to UART by another device such as CPU
  - Data is transferred from data bus to transmitting UART in parallel form
  - Transmitting UART creates a packet and adds a start bit, a parity bit and a stop bit
  - The data packet is output serially at Tx pin

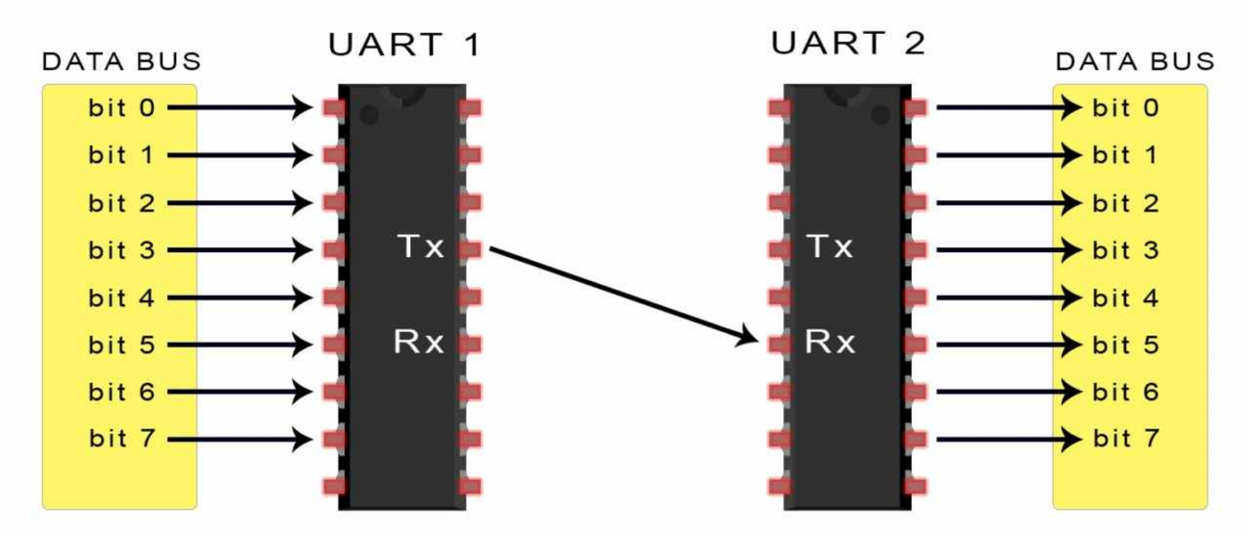




source: www.circuitbasics.com

UART

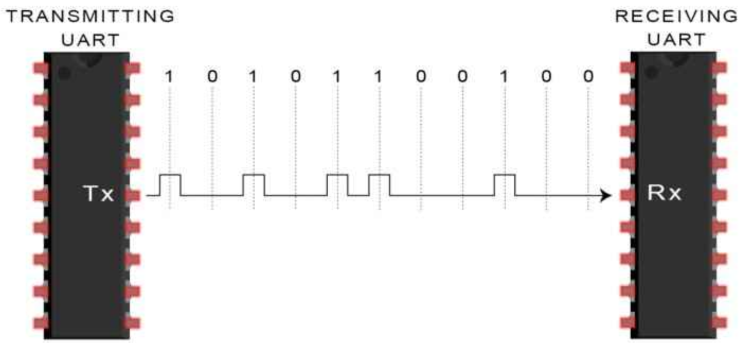
- How UART Works
  - The receiving UART reads data packet bit by bit at its Rx pin
  - The receiving UART then converts the data back into parallel form and removes the start bit, parity bit, and stop bits.
  - Finally, the receiving UART transfers the data packet in parallel to the data bus on the receiving end



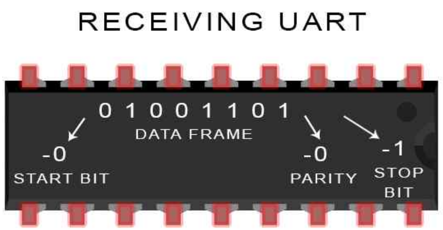
source: www.circuitbasics.com

UART

- UART Transmission
  - The entire packet is sent serially from transmitting UART to receiving UART
    - The receiving UART samples the data line at pre-configured baud rate



  - The receiving UART discards the start bit, parity bit, and stop bit



source: www.circuitbasics.com

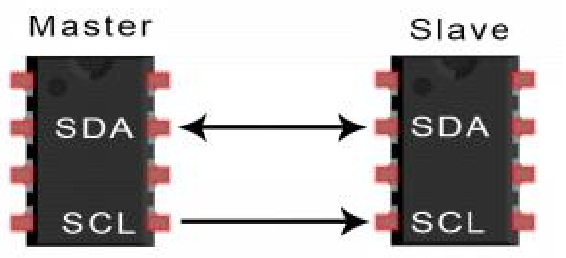
UART

- Advantages
  - Uses two wires
  - Clock signal is not required
  - Parity bit for error checking
  - The structure of data packet can be changed as long as both sides are set up for it
  - Well documented and widely used method
- Disadvantages
  - The size of data frame is limited to a maximum of 9 bits
  - Does not support multiple slave or multiple master systems
  - The baud rates of each UART must be within 10% of each other

source: www.circuitbasics.com

I2C Communication Protocol

- Multiple slaves to a single master
- Multiple masters and a single or multiple slaves
- Uses two wires to transmit data between devices



- Serial Data (SDA)
    - The line for the master and slave to send and receive data
- Serial Clock (SCL)
    - The line that carries the clock signal

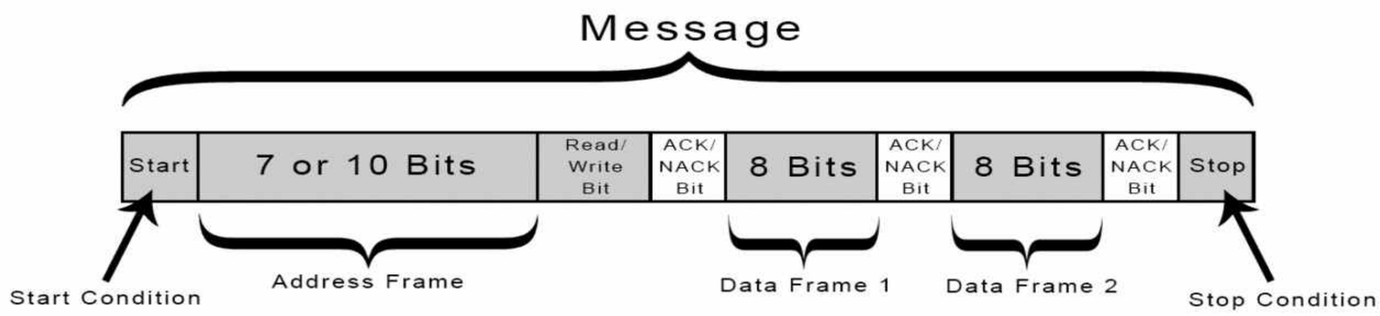
source: www.circuitbasics.com

I2C Communication Protocol	
Wires used	2
Maximum Speed	Standard mode=100 kbps Fast mode=400 kbps High speed mode= 3.4 Mbps Ultra fast mode=5 Mbps
Synchronous or Asynchronous	Synchronous
Serial or Parallel	Serial
Max Masters	Unlimited
Max Slaves	1008

source: www.circuitbasics.com

How I2C Works

Data is transferred in messages and messages are broken up into frames of data



Start Condition SDA line switches from a high voltage level to a low voltage level before SCL line switches from high to low

Stop Condition SDA line switches from a low voltage level to a high voltage level after SCL line switches from low to high

Address Frame 7 or 10 bit unique sequence identifies slave when master wants to talk to it

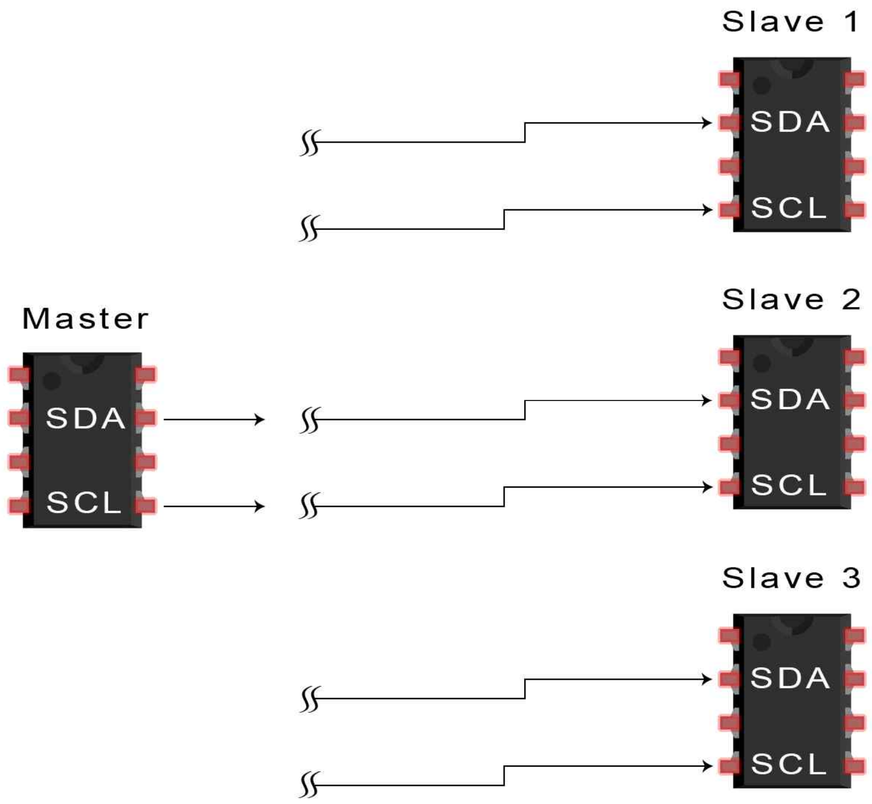
Read or Write bit A single bit specifying whether the master is sending data to the slave or requesting data from it

ACK or NACK bit Each frame in a message is followed by an acknowledge or no-acknowledge bit

source: www.circuitbasics.com

I2C Communication Protocol

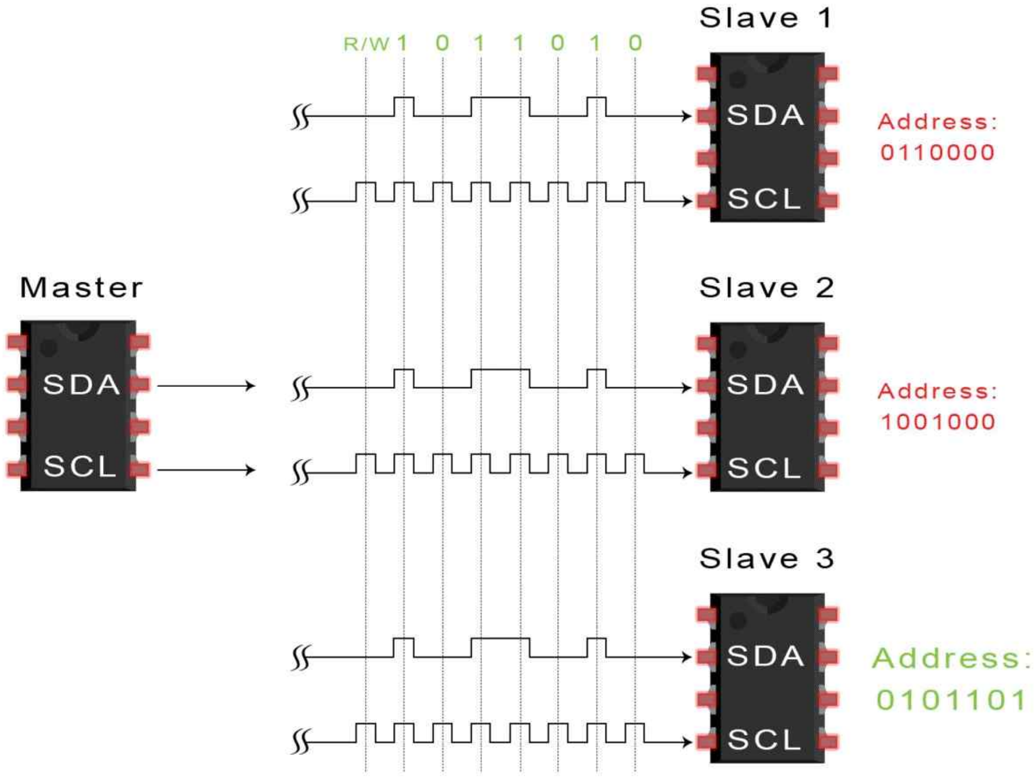
- Data Transmission
  - The master sends the start condition to every connected slave by switching the SDA line from a high voltage level to a low voltage level before switching the SCL line from high to low



source: www.circuitbasics.com

I2C Communication Protocol

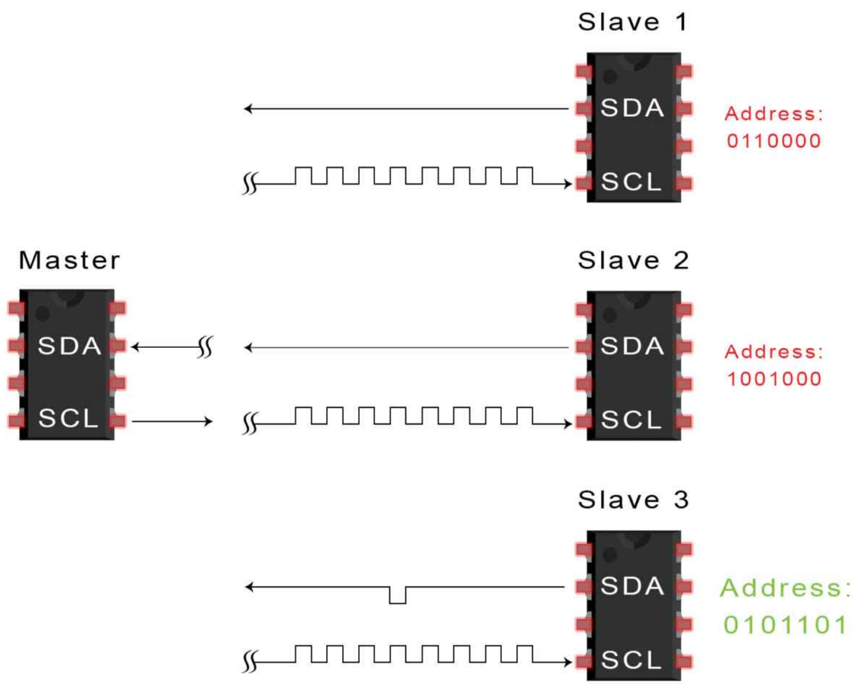
- Data Transmission
  - The master sends each slave the 7 or 10 bit address of the slave it wants to communicate with, along with the read-write bit



source: www.circuitbasics.com

I2C Communication Protocol

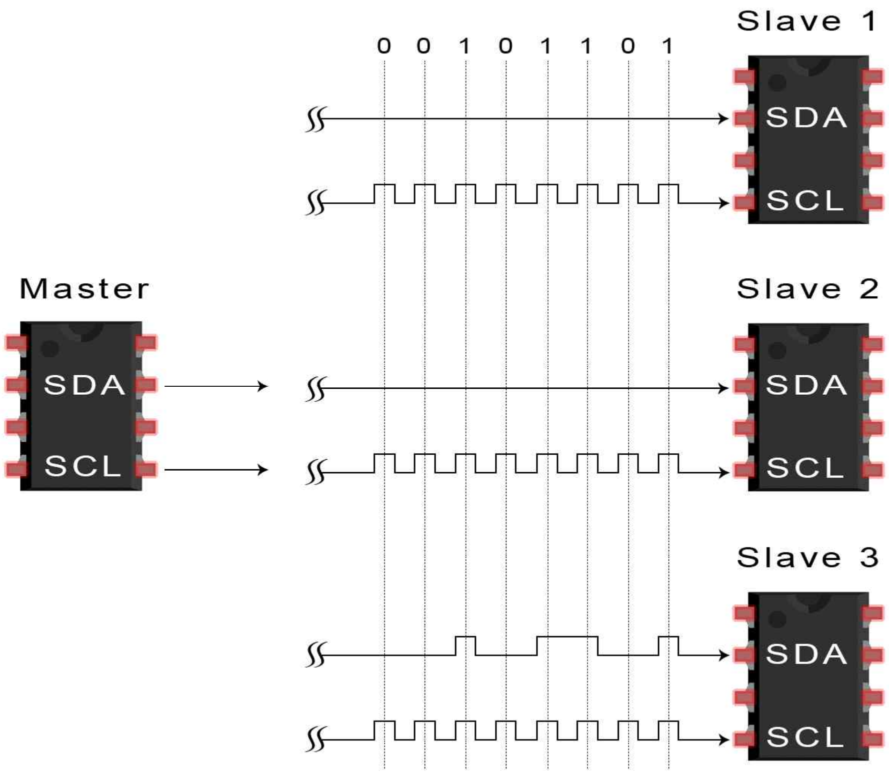
- Data Transmission
  - Each slave compares the address sent from the master to its own address. If the address matches, the slave returns an ACK bit by pulling the SDA line low for one bit. If the address from the master does not match the slave's own address, the slave leaves the SDA line high.



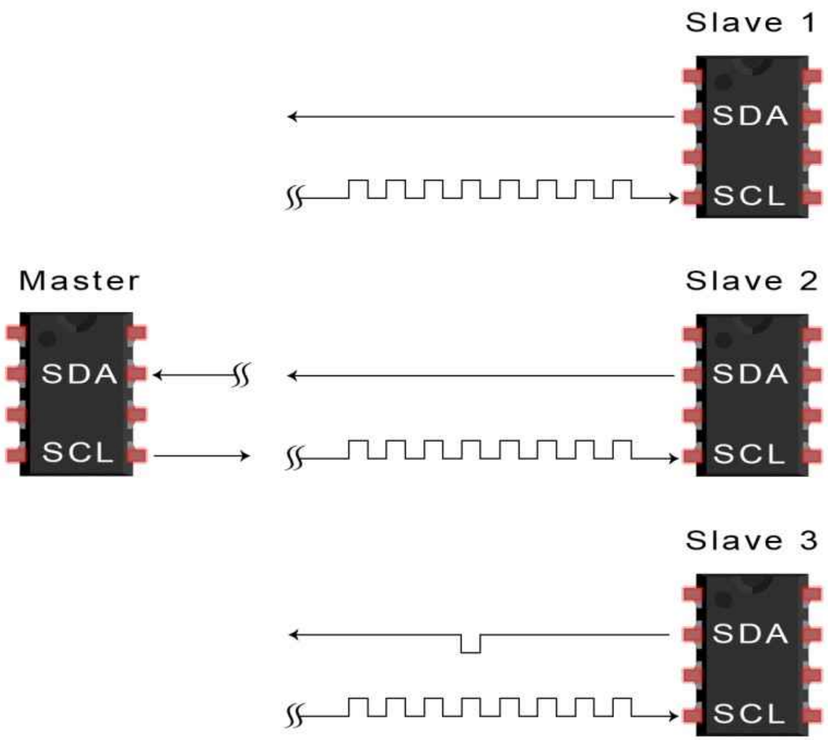
source: www.circuitb

I2C Communication Protocol

- Data Transmission
  - The master sends or receives the data frame
  - After each data frame has been transferred, the receiving device returns another ACK bit to the sender to acknowledge successful receipt of the frame

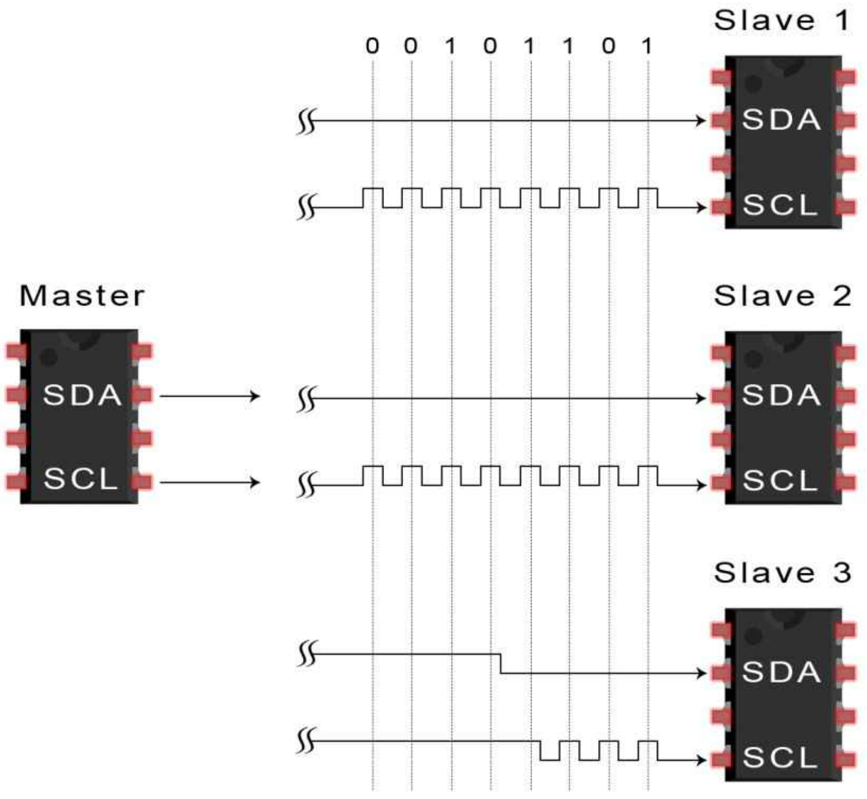


w.circuitbasics.c



I2C Communication Protocol

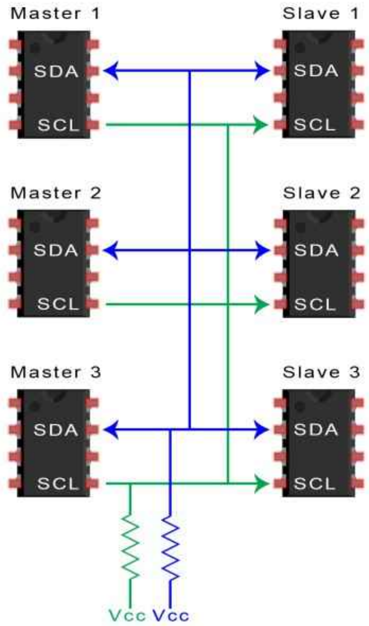
- Data Transmission
  - To stop the data transmission, the master sends a stop condition to the slave by switching SCL high before switching SDA high



source: www.circuitbasics.com

I2C Communication Protocol

- Advantages
  - Uses two wires
  - Supports multiple masters and multiple slaves
  - ACK/NACK bit gives confirmation that each frame is transferred successfully
- Disadvantages
  - Slower data transfer rate than SPI
  - The size of the data frame is limited to 8 bits
  - More complicated hardware needed to implement than SPI



source: www.circuitbasics.com

Any Questions!