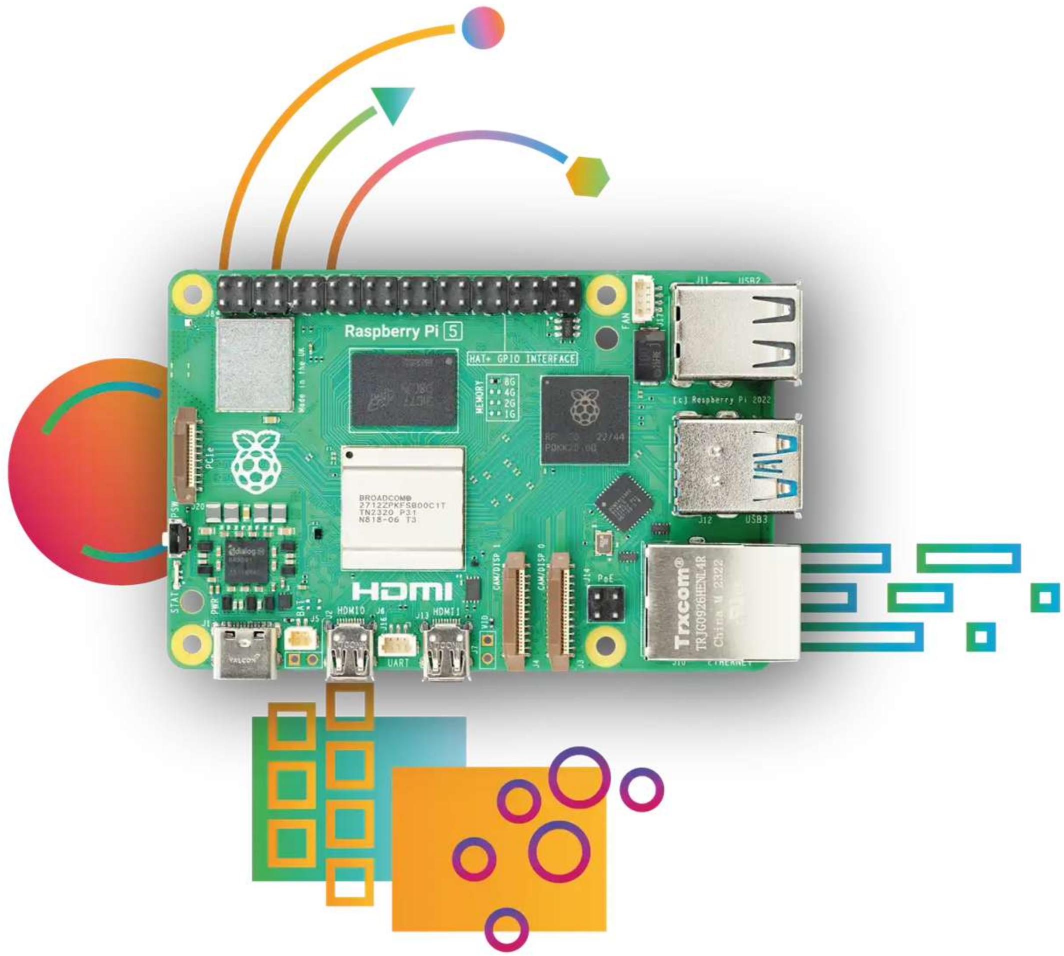
RaspberryPi



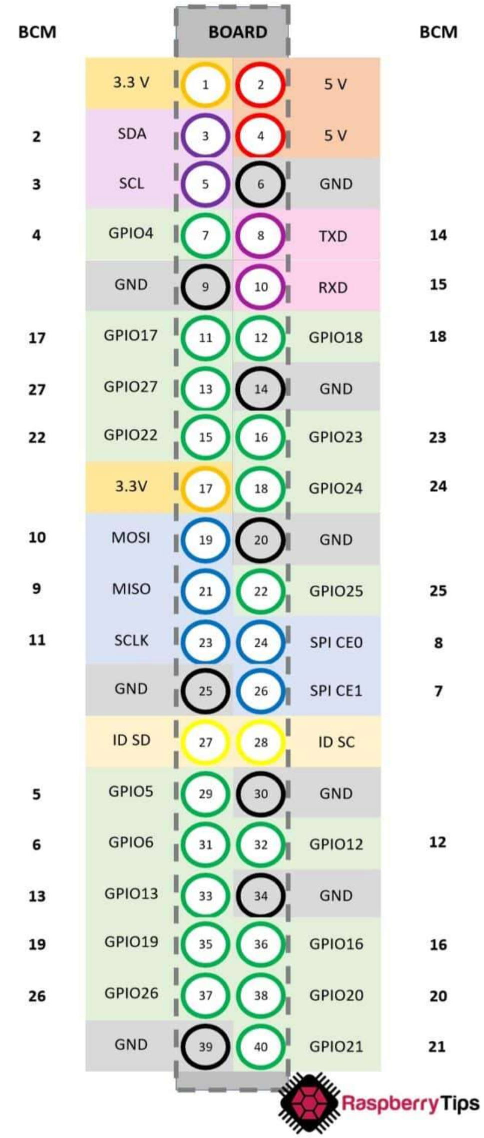
GPIO

pip install gpiozero

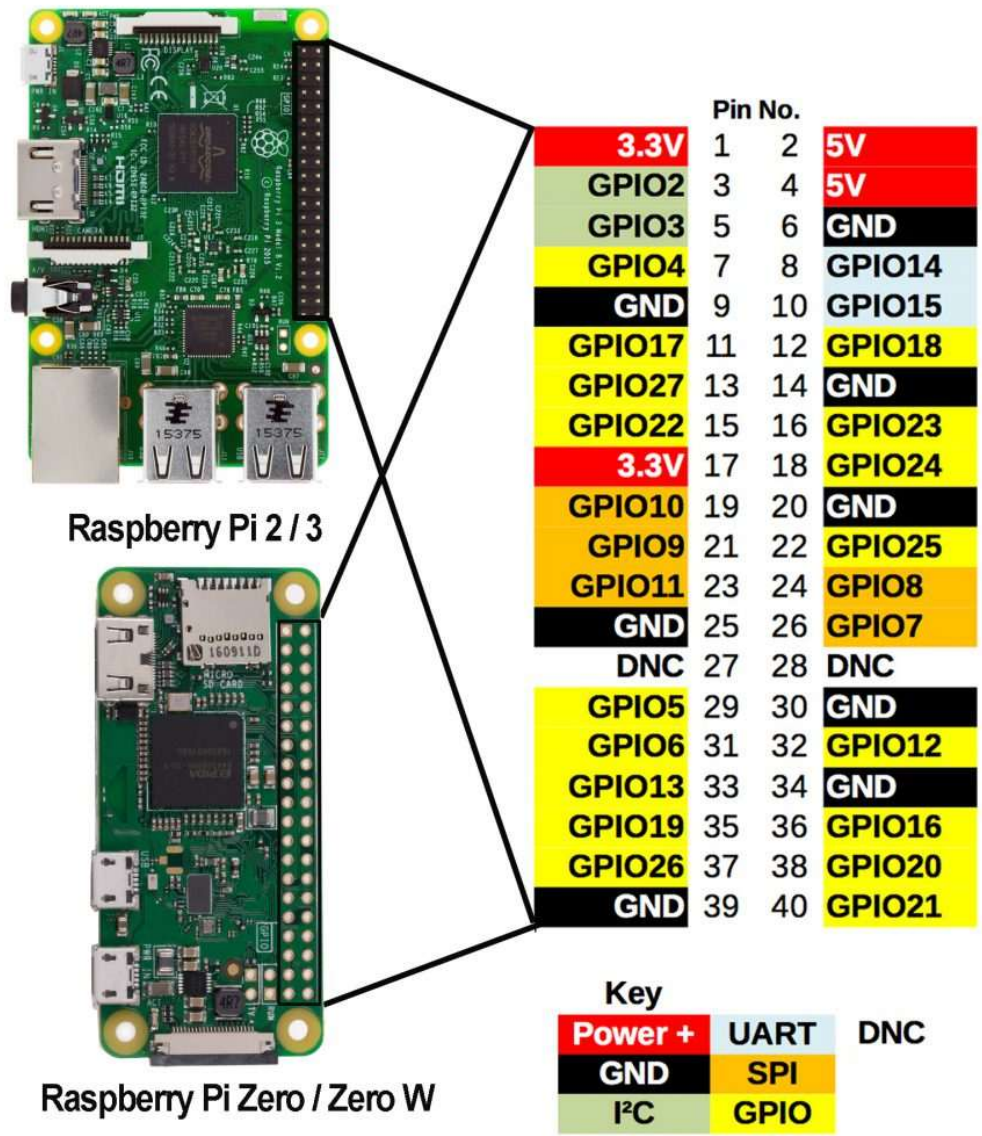
The gpiozero library is a Python module designed to simplify working with GPIO pins on Raspberry Pi. It provides an intuitive interface for controlling LEDs, buttons, motors, sensors, and other electronic components.

Key Features:

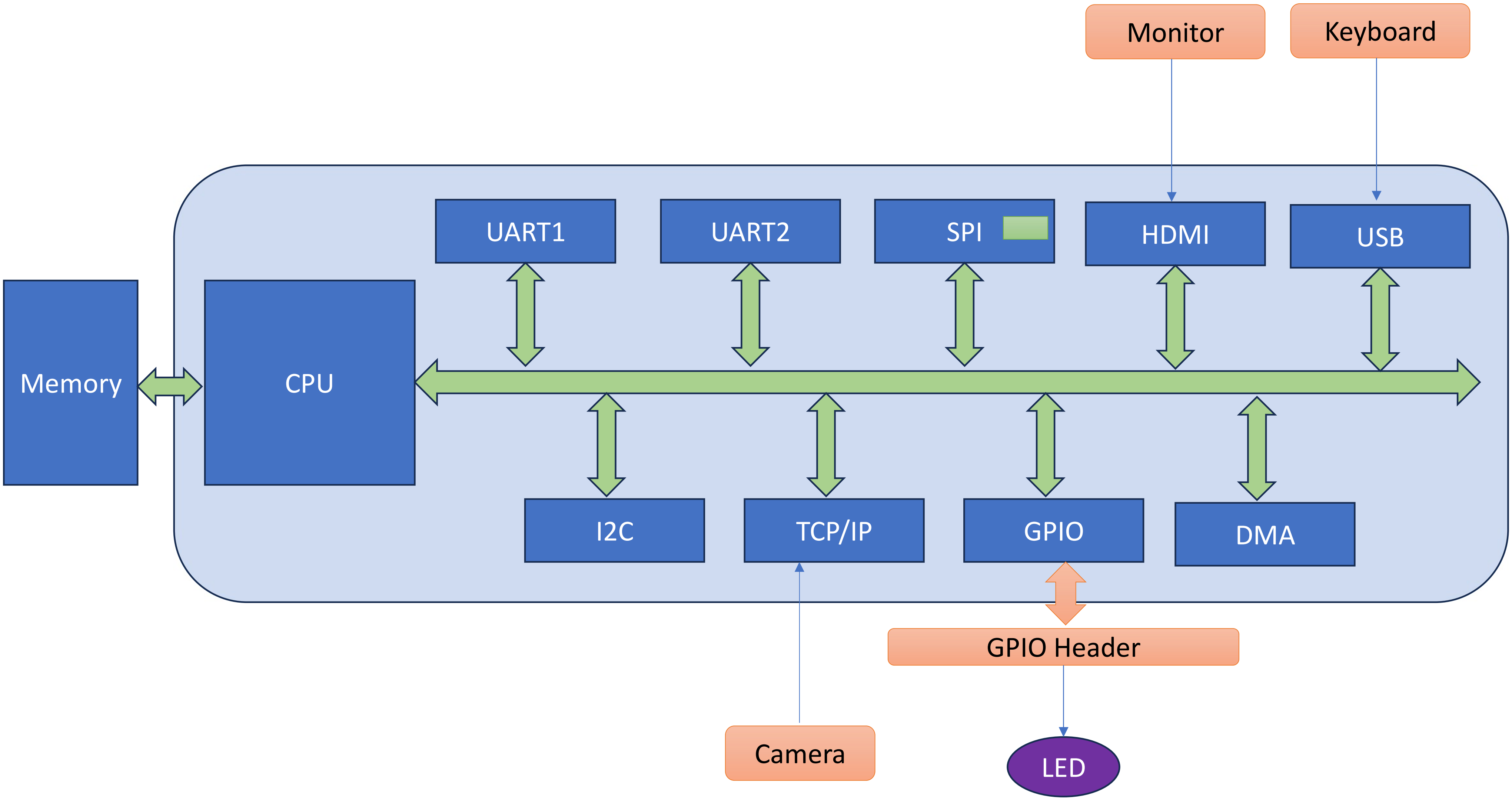
- **Easy-to-use API**: Simplifies GPIO programming with high-level abstractions.
- **Supports multiple pin libraries**: Works with RPi.GPIO, pigpio, and lgpio.
- **Event-driven programming**: Allows handling button presses and sensor readings efficiently.
- **Built-in device classes**: Includes predefined classes for LEDs, buttons, motors, and more.

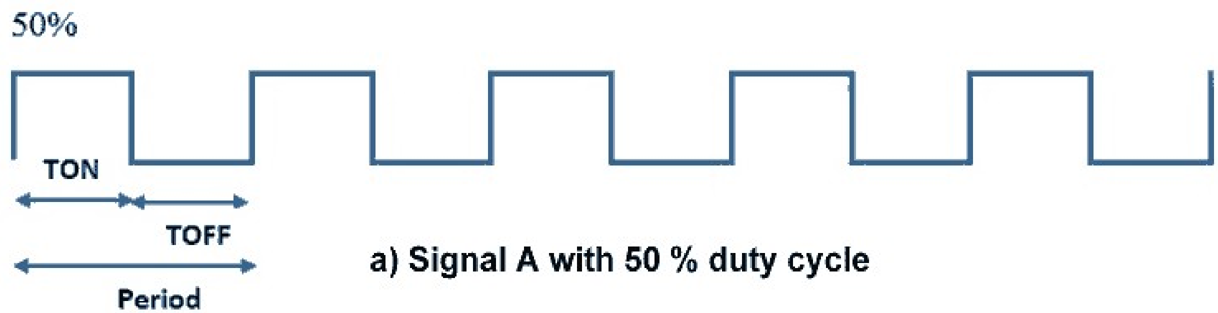


GPIO

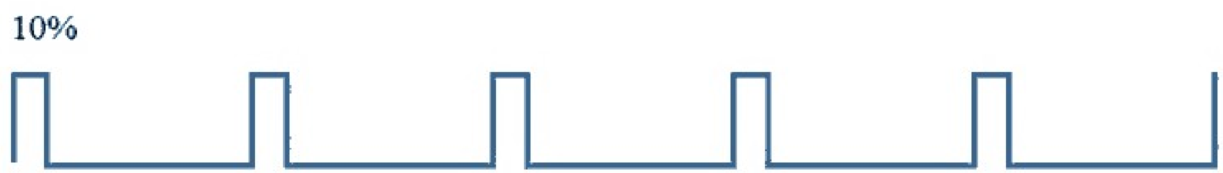


Raspberry Pi 2 / 3
Raspberry Pi Zero / Zero W
Pin No.
3.3V 1 2 5V
GPIO2 3 4 5V
GPIO3 5 6 GND
GPIO4 7 8 GPIO14
GND 9 10 GPIO15
GPIO17 11 12 GPIO18
GPIO27 13 14 GND
GPIO22 15 16 GPIO23
3.3V 17 18 GPIO24
GPIO10 19 20 GND
GPIO9 21 22 GPIO25
GPIO11 23 24 GPIO8
GND 25 26 GPIO7
DNC 27 28 DNC
GPIO5 29 30 GND
GPIO6 31 32 GPIO12
GPIO13 33 34 GND
GPIO19 35 36 GPIO16
GPIO26 37 38 GPIO20
GND 39 40 GPIO21
Key
Power +
UART
DNC
GND
SPI
I²C
GPIO


Monitor
Keyboard
UART1
UART2
SPI
HDMI
USB
Memory
CPU
I2C
TCP/IP
GPIO
DMA
GPIO Header
Camera
LED



a) Signal A with 50 % duty cycle



b) Signal B with 10% duty cycle

c) Signal C with 30% duty cycle

d) Signal D with 70% duty cycle

What is UART?

UART is a **hardware communication protocol** that uses **asynchronous serial communication** with configurable **speed** (baud rate). It is used for **short-distance, low-cost, low-speed data exchange** between devices.

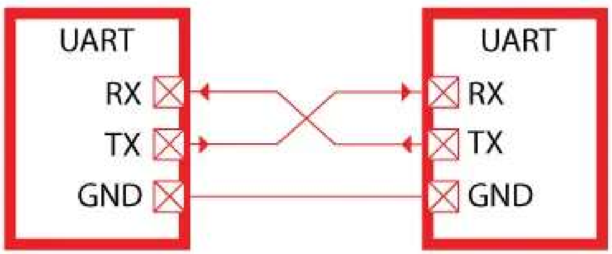
Key Features:

- **Asynchronous**: No clock signal is shared between sender and receiver.
- **Serial**: Bits are sent one after another on a single wire.
- **Full-duplex**: Separate lines for transmission and reception

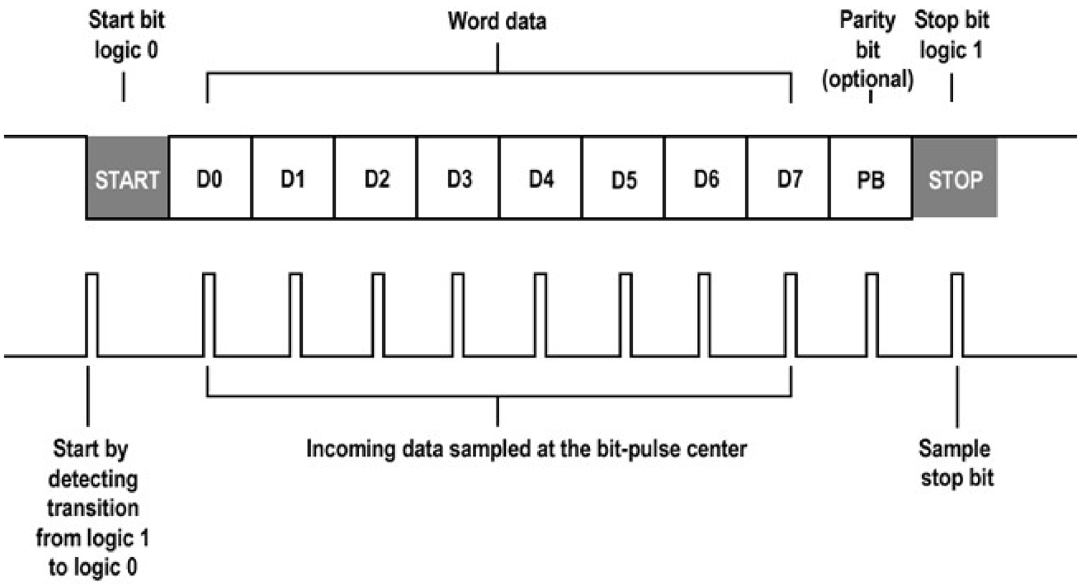
Components of UART Communication

UART Pin	Description
TX	Transmit pin
RX	Receive pin
GND	Common ground between devices

UART



UART
RX
TX
GND
UART
RX
TX
GND



Start bit logic 0
Word data
Parity bit (optional)
Stop bit logic 1
START
D0
D1
D2
D3
D4
D5
D6
D7
PB
STOP
Start by detecting transition from logic 1 to logic 0
Incoming data sampled at the bit-pulse center
Sample stop bit

Baud Rate

- Baud Rate: Number of bits transmitted per second.
- Common baud rates: 9600, 19200, 38400, 57600, 115200
- Both sender and receiver must agree on the baud rate.

How UART Works?

Transmitter Side:

1. Prepares the frame with start, data, optional parity, and stop bits.
2. Converts parallel data to serial data.
3. Sends one bit at a time through TX pin.

Receiver Side:

1. Detects start bit (0).
2. Samples data bits at expected intervals.
3. Checks parity (if used).
4. Verifies stop bit(s).
5. Converts serial data back to parallel format.

Applications of UART

- Serial consoles in embedded systems
- Communication with GPS, GSM, Bluetooth modules
- Debugging output from microcontrollers
- Serial programming (e.g., Arduino)
- Industrial devices (PLCs, sensors, etc.)

FTDI

- The FTDI chip acts as a bridge, converting USB communication to serial communication

I2C

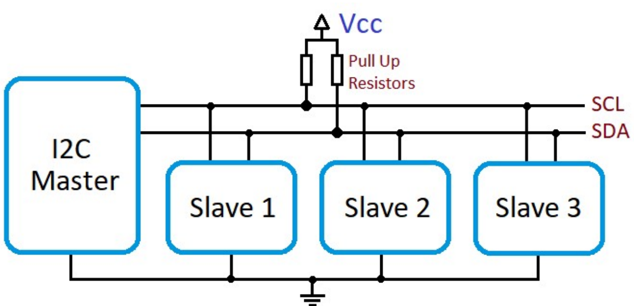
- **I2C (Inter-Integrated Circuit)** is a **synchronous, multi-master, multi-slave, serial communication protocol** commonly used for short-distance communication between microcontrollers and peripherals (like sensors, EEPROMs, RTCs).

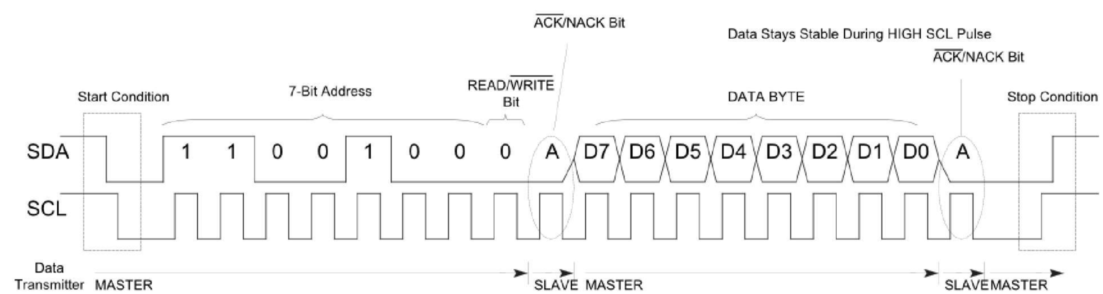
Key Features

- **Two-wire interface:** Only two lines are required — **SDA (data)** and **SCL (clock)**.
- **Multi-master and multi-slave:** Multiple devices can share the bus; multiple masters can initiate communication.
- **Addressing:** Each device has a unique 7-bit or 10-bit address.
- **Half-duplex:** Data is transferred one direction at a time.
- **Synchronous:** Clock generated by master synchronizes data transfer.

Hardware Lines

Signal Name	Description
SDA	Serial Data Line (bi-directional)
SCL	Serial Clock Line (generated by master)





I2C Working

1. **Bus Idle:** Both SDA and SCL lines are HIGH (pulled-up).
2. **Start Condition:**
   Master pulls SDA **low** while SCL is **high** to signal start.
3. **Address Frame:**
   Master sends 7-bit slave address + 1-bit Read/Write flag (R/W) on SDA, synchronized by clock on SCL.
4. **ACK/NACK:**
   The addressed slave acknowledges (ACK) by pulling SDA low during the next clock pulse. If no device responds, master gets NACK.
5. **Data Transfer:**
   1. Data bytes are transferred sequentially, each followed by an ACK bit.
   2. Master controls the clock.
   3. Data is valid on the rising or falling clock edge (depending on device).
6. **Stop Condition:**
   Master releases SDA to HIGH while SCL is HIGH, signaling end of communication.

Why Use I2C?

- Minimal wiring (only two lines)
- Supports multiple devices on the same bus
- Widely supported by sensors, RTCs, EEPROMs, displays
- Easy to add/remove devices without changing wiring (just unique address)

Limitations

- Slower than SPI (standard mode 100 kHz, fast mode 400 kHz, some go higher)
- Half-duplex (one direction at a time)
- Requires pull-up resistors on SDA and SCL lines

I2C Devices on Pioneer600

Device	I2C Address	Description
PCF8591	0x48	8-bit ADC/DAC converter (4-channel analog input, 1 analog output)
AT24C08	0x50	EEPROM (1 KByte, organized as 8 pages of 128 bytes)
BMP280	0x76	Digital barometric pressure and temperature sensor
TCS34725	0x29	RGB color sensor with IR filter and white LED
ADS7830	0x48	8-channel 8-bit ADC (sometimes used on variants)

What is SPI?

SPI is a **synchronous**, **full-duplex**, **master-slave** communication protocol used primarily for **short-distance** communication in embedded systems.

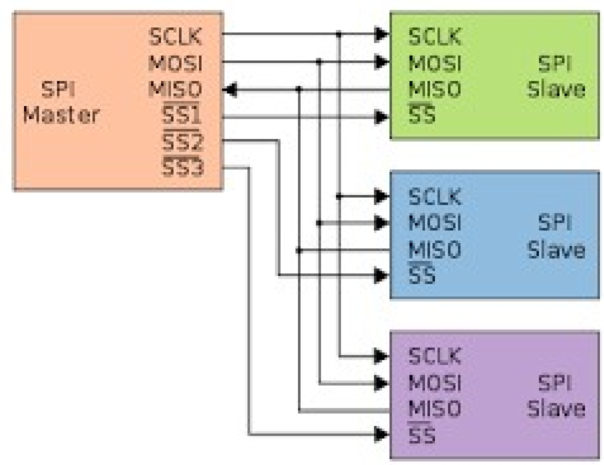
➕ **Key Features:**

- Synchronous (uses a clock line)
- High-speed (much faster than UART or I2C)
- Full-duplex (simultaneous send/receive)
- Multi-slave capable using chip-select lines

Basic SPI Architecture

SPI typically uses **four wires**:

- **MOSI (Master Out Slave In)**: Data sent from master to slave.
- **MISO (Master In Slave Out)**: Data sent from slave to master.
- **SCLK (Serial Clock)**: Clock signal generated by the master to sync data.
- **SS/CS (Slave Select or Chip Select)**: Used by the master to activate a specific slave.



SPI Roles

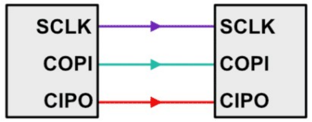
- **Master**: Controls the clock and initiates communication.
- **Slave**: Responds to master's signals.
- Only one master at a time; multiple slaves can be supported.

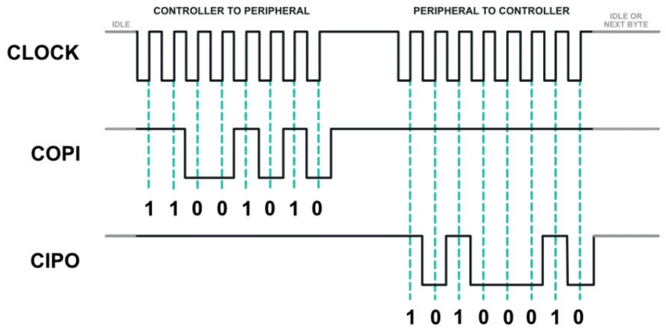
SPI Communication Process

1. **Master Initiates Transfer:** The master device starts the communication by selecting a slave device (using a Chip Select/Slave Select line) and then synchronizing data transfer with the clock line (SCK).
2. **Command on MOSI:** The master sends a command instruction, typically a byte or a few bytes, over the MOSI line.
3. **Slave Receives and Processes:** The slave device receives the command on MOSI and interprets it.
4. **Slave Response on MISO:** Based on the received command, the slave performs an action, and then sends a response (data, acknowledgment, etc.) back to the master via MISO.
5. **Master Receives:** The master reads the data or acknowledgment received on the MISO line

Example

- **Master sends a "read register" command:**
  - The master might send a command byte that indicates it wants to read a specific register address.
- **Slave sends register value:**
  - The slave device then sends the contents of the requested register on the MISO line.
- **Master reads and processes:**
  - The master device receives the data on MISO, interprets it (e.g., converts it to a value), and then uses the data.



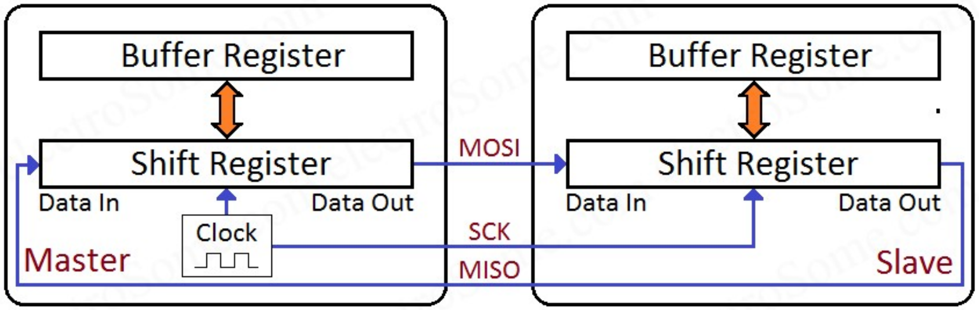


SPI Clock Configuration

Clock settings define data timing:

- **CPOL (Clock Polarity)**:
  - 0: Clock idle low
  - 1: Clock idle high
- **CPHA (Clock Phase)**:
  - 0: Data sampled on leading edge
  - 1: Data sampled on trailing edge

Mode	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1



SPI Configuration Parameters

- **Clock speed (baud rate)**: Defined by the master; can reach tens of MHz.
- **Bit order**: Usually MSB (Most Significant Bit) first; some devices use LSB first.
- **Data word length**: Typically 8-bit, but configurable to 16, 32, etc.

SPI Advantages

- High-speed data transfer.
- Simple hardware implementation.
- Full-duplex communication.
- Supports multiple slaves with individual SS lines.

SPI Disadvantages

- No acknowledgment mechanism (unlike I2C).
- Requires more pins than I2C (4 vs 2).
- No formal standard for multi-master systems.
- Limited distance (short range).

Common SPI Devices

- Displays (e.g., OLED, TFT)
- Flash memory (e.g., EEPROM, NOR/NAND flash)
- ADCs and DACs
- Sensors (e.g., accelerometers, temperature sensors)
- SD cards

Typical SPI Use Case (Raspberry Pi Example)

- Connect SPI OLED display:
  - MOSI → DIN
  - MISO → (not connected if unused)
  - SCLK → CLK
  - SS → CS
- Use spidev library in Python to communicate with the device.
- Send data/image buffers directly over the SPI interface.