Internet of Things Programming

Week 4 Lecture

Anas Dawod

adawod@swin.edu.au

Semester 1 2023

Last week...

We discussed about

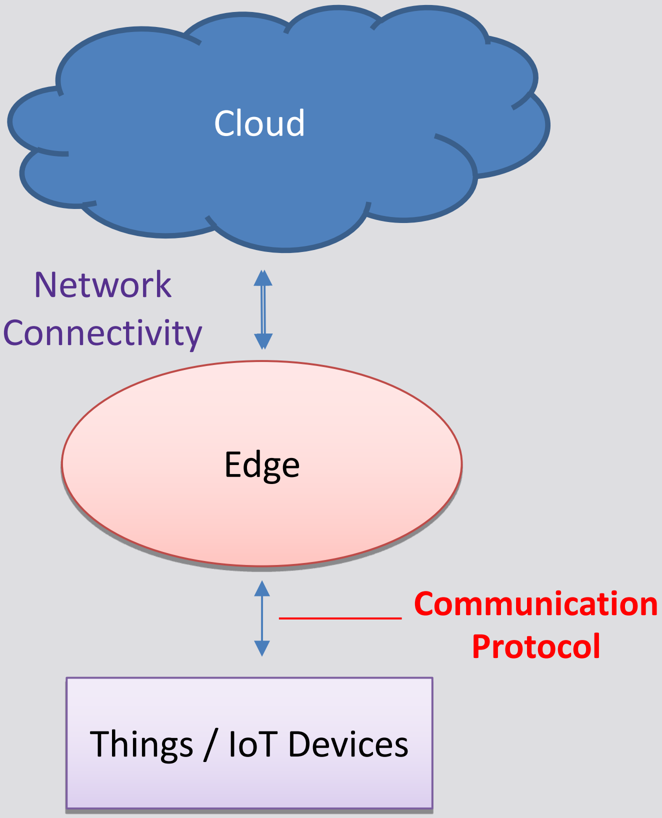
- Edge computing
- Raspberry Pi Architecture
- Introduction to Raspberry Pi OS
- Programming edge device

This week

We are going to work on:

- Communication Protocols
- Parallel Communication
- Serial Communication
  - How to connect different peripherals (Sensors-Arduino, Arduino-RPi, etc.)
- Programming time consuming tasks in edge devices (microcontrollers)
  - Actuators (e.g., closing a door, opening a valve, etc.)
  - Counting time
  - To do so, we are going to use
    - Timers
    - Interrupts

IoT Architecture



An IoT application involves connecting sensors, processors, edge devices etc.

For these devices to exchange information, they need to share a common communication protocol.

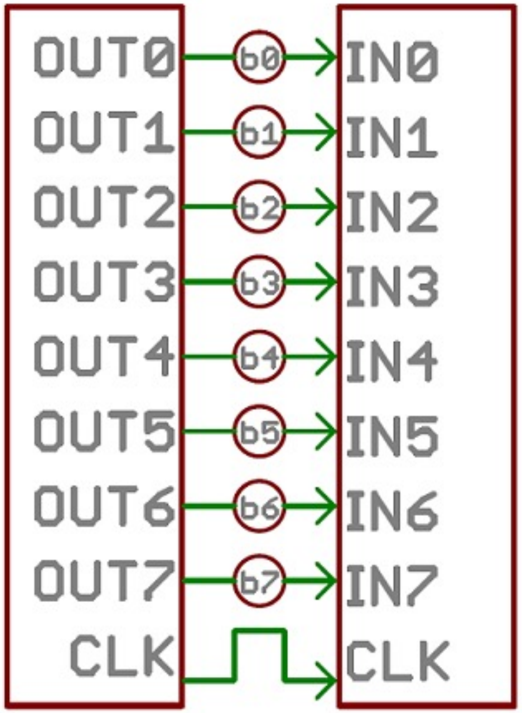
Several communication protocols exist for data exchange.

Broadly they can be divided into two categories:

- parallel
- serial.

Parallel Communication

Parallel Communication



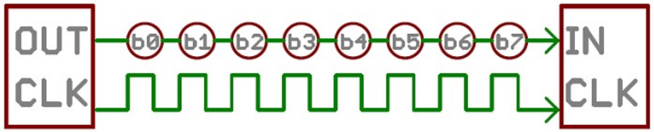
Parallel communication is a method of transmitting/sending multiple binary bits simultaneously over multiple channels (wires/buses/frequencies etc.).

- High Speed
- Expensive
- Susceptible to cross-talk
- Used for short distances
- E.g., Computer mother board and hard disk.

https://learn.sparkfun.com/tutorials/serial-communication/all

Serial Communication

Serial Communication

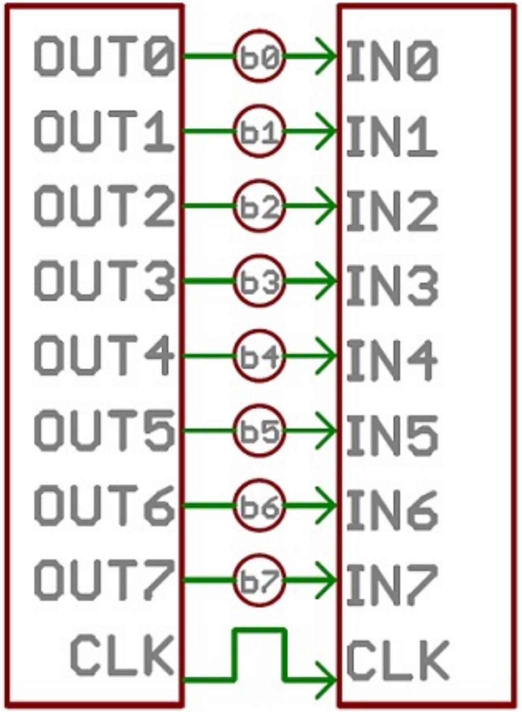


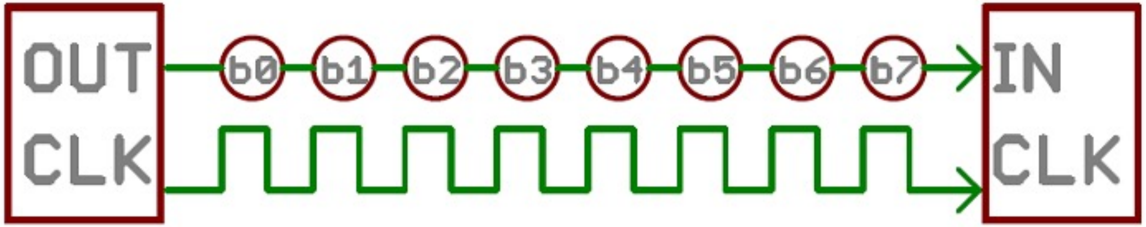
Serial communication is a method of transmitting/sending one bit at a time over a single channel (wire/bus/frequency etc.).

- Low speed
- Less expensive
- Less susceptible to cross-talk
- Used for long distances
- E.g., Sensor and processor OR IoT device and Edge device.

https://learn.sparkfun.com/tutorials/serial-communication/all

Parallel vs Serial Communication



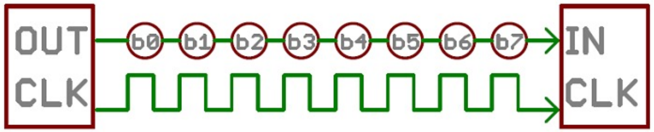


https://learn.sparkfun.com/tutorials/serial-communication/all

Serial Communication

Two types of serial communication:

- Asynchronous serial communication
- Synchronous serial communication



https://learn.sparkfun.com/tutorials/serial-communication/all

Asynchronous Serial Communication

Asynchronous serial communication is a communication interface in which the transmitter and receiver endpoints are not continuously synchronised to each other using a common clock signal.

- No external clock signal
- Ideal for minimising wires and I/O pins
- Serial protocols help to reliably transfer and receive data during asynchronous communication.

Asynchronous Serial Communication

Asynchronous serial protocol helps to achieve ensure robust and error-free data transfers.
Asynchronous serial protocol consists of a number of built-in rules – mechanisms to help communicate with the receiver.

Built-in rules:

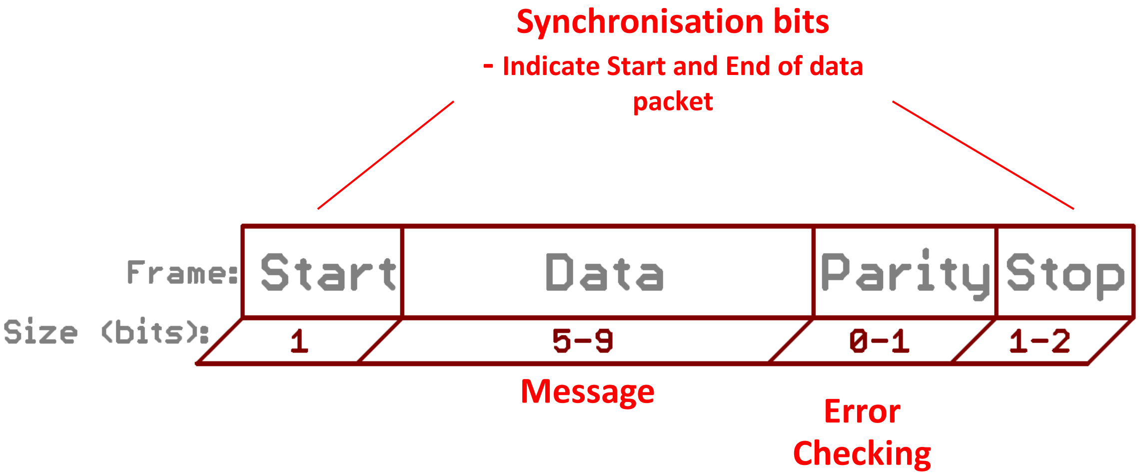
Data Frame

- **Data bits**
- **Synchronisation bits**
- **Parity bits**
- **and Baud rate**

https://learn.sparkfun.com/tutorials/serial-communication/all

Asynchronous Serial Communication

- Data Frame



- Example: 8 data bits, no parity, and 1 stop bit

https://learn.sparkfun.com/tutorials/serial-communication/all

Asynchronous Serial Communication

Asynchronous (No clock)

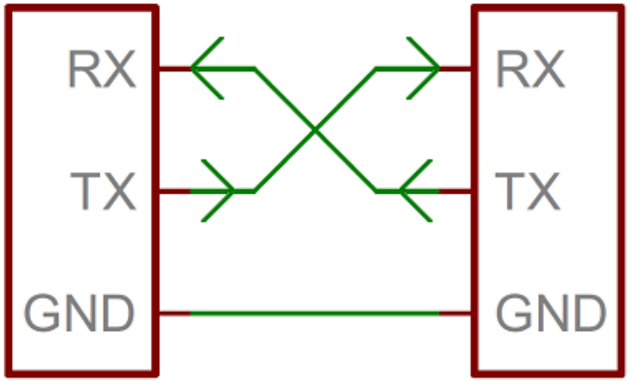
WITHOUT CLOCK, HOW DOES RECEIVER KNOW WHEN TO RECEIVE DATA FROM TRANSMITTER?

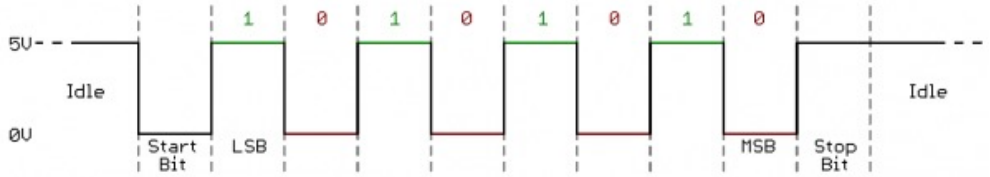
- **BAUD Rate:1200, 2400, 4800, 9600, 19200, 115200**
  - How fast the data is sent over the serial line and baud rate is defined in bits per second.
  - Data is lost if transmitter and receiver are not set to same baudrate.
- **9600 8N1 - 9600 baud, 8 data bits, no parity, and 1 stop bit - is one of the more commonly used serial protocols**

An example: 9600 baud

https://learn.sparkfun.com/tutorials/serial-communication/all

Serial interface





- Physical interface (circuitry) that enables Serial Communication
- RX/TX from the device point of view
- Full-duplex vs Half-duplex
- Signal voltages match up

https://learn.sparkfun.com/tutorials/serial-communication/all

Serial Interface - UART

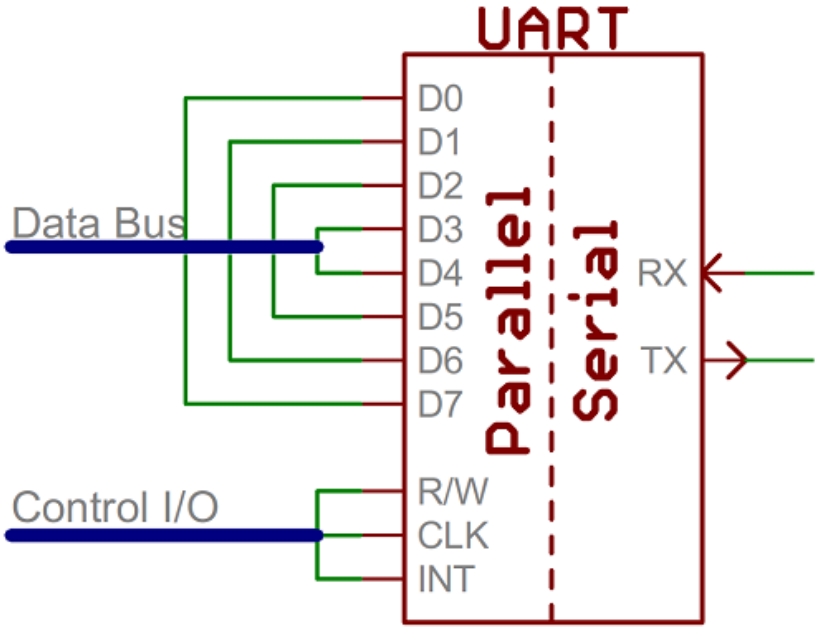
Universal Asynchronous Reception and Transmission (block of circuitry responsible for implementing serial communication).

- Hardware component that interfaces parallel and serial communication

Simplest serial form of bidirectional and asynchronous communication between two devices.

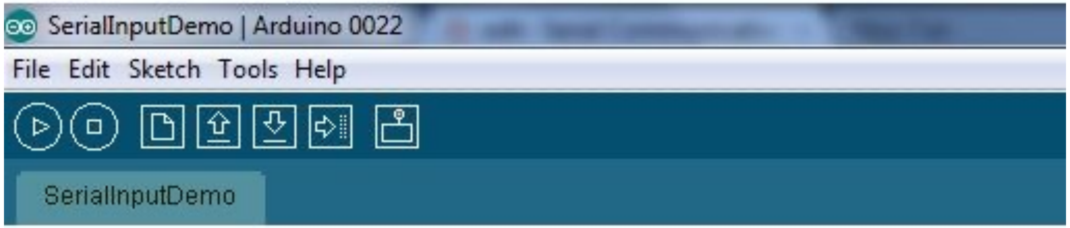
Two data lines: Transmit (Tx) and Receive (Rx) which is used to communicate through digital pin 0 and 1.

No clock.



Serial transmission

```
void setup (){
  Serial.begin(9600);
}

void loop(){
  Serial.print("The analog value is: ");
  Serial.println(analogRead(A0));
  delay(100);
}
```



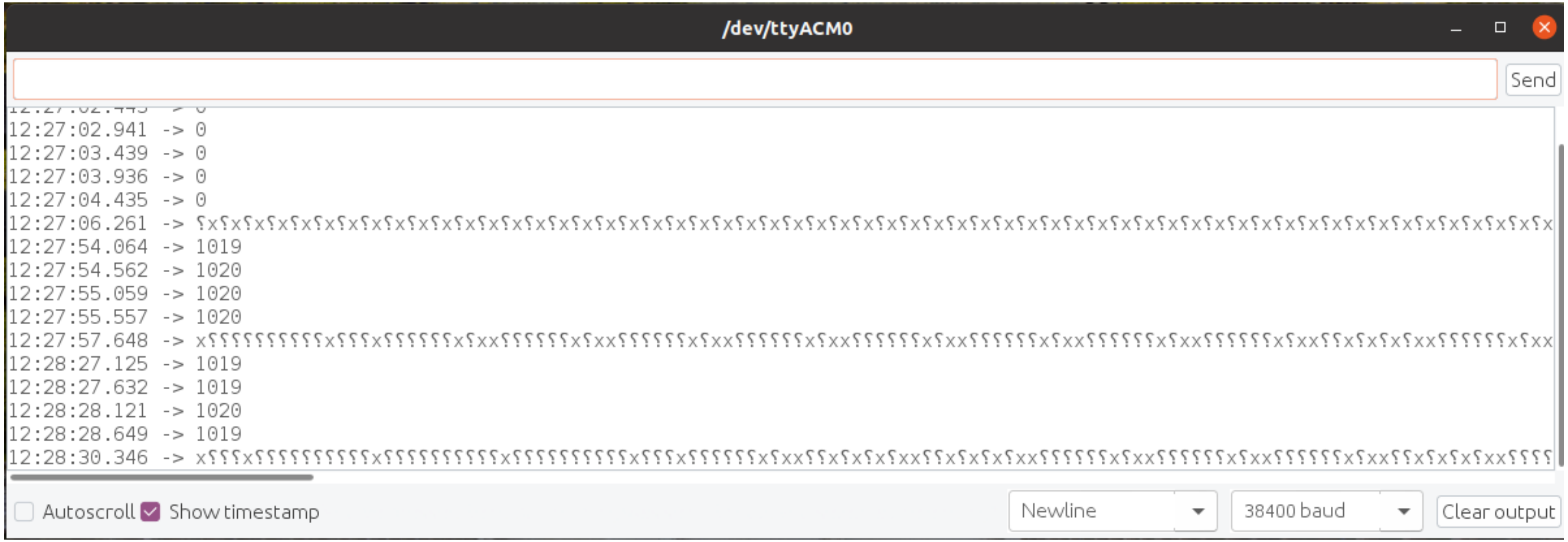

```
//Serial Demo
//Led connected to Pin 13
int inByte;

void setup() {
  Serial.begin(9600);
  pinMode(13, OUTPUT);
  Serial.println("Ready");
}

void loop() {
  if(Serial.available() > 0) {
    inByte = Serial.read();
    if(inByte == 'a') {
      digitalWrite(13, HIGH);
      Serial.println("LED - On");
    }
    else {
      digitalWrite(13, LOW);
      Serial.println("LED - off");
    }
  }
}
```

Program to turn an LED ON/OFF when a byte is read

https://www.instructables.com/id/Serial-Communications-with-Arduino/

Common Pitfalls in Asynchronous communication

- Baud rate mismatch



Data transmitted at baud rate 9600 and received at 38400 baud rate

https://learn.sparkfun.com/tutorials/serial-communication/all

Common Pitfalls in Asynchronous communication

- TX to TX and RX to RX

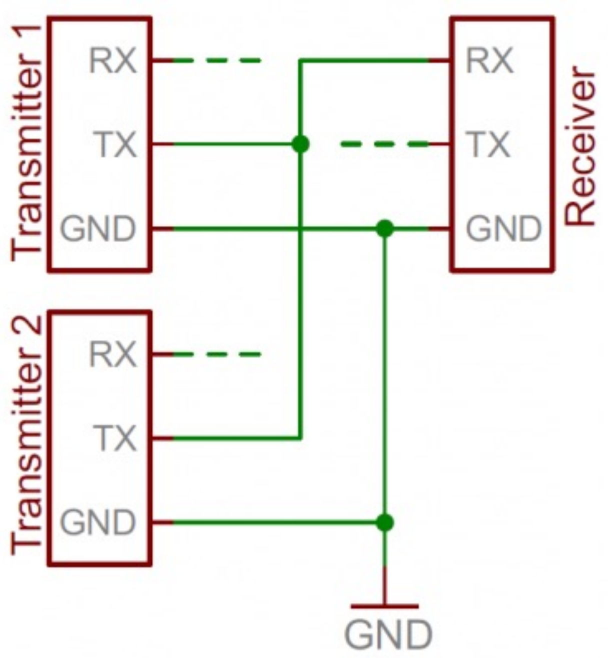
FT232RL FTDI USB to Serial
UART Adaptor Module

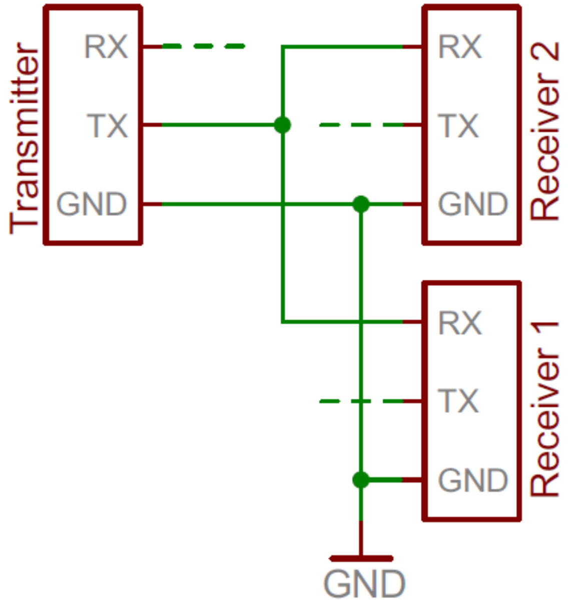
Arduino Mini Pro

https://learn.sparkfun.com/tutorials/serial-communication/all

Common Pitfalls in Asynchronous communication

Bus Contention





Synchronous serial communication

I2C (Inter-Integrated Circuit)

Originally developed by Philips in 1982.

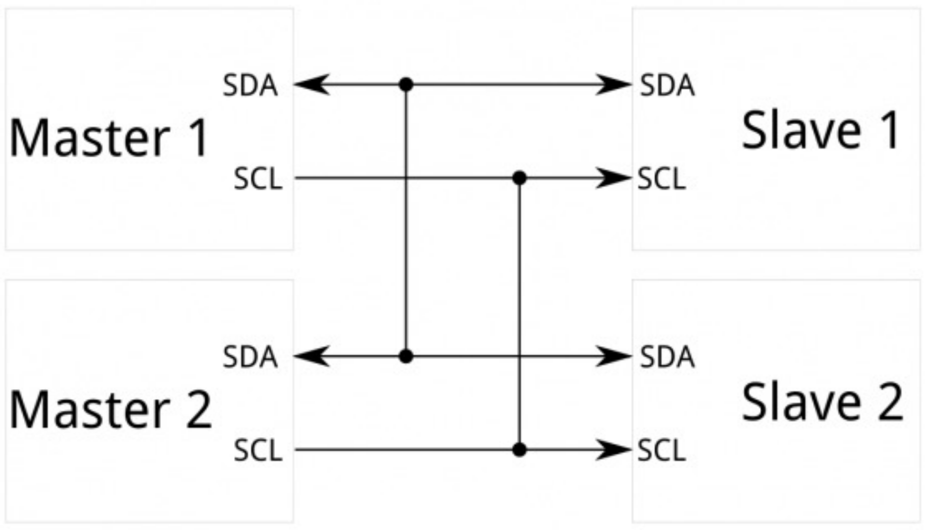
I2C consists of two signal lines (SCL, and SDA)

- SCL: Clock Signal.
- SDA: Data Signal

Differences of I2C compared to UART and SPI:

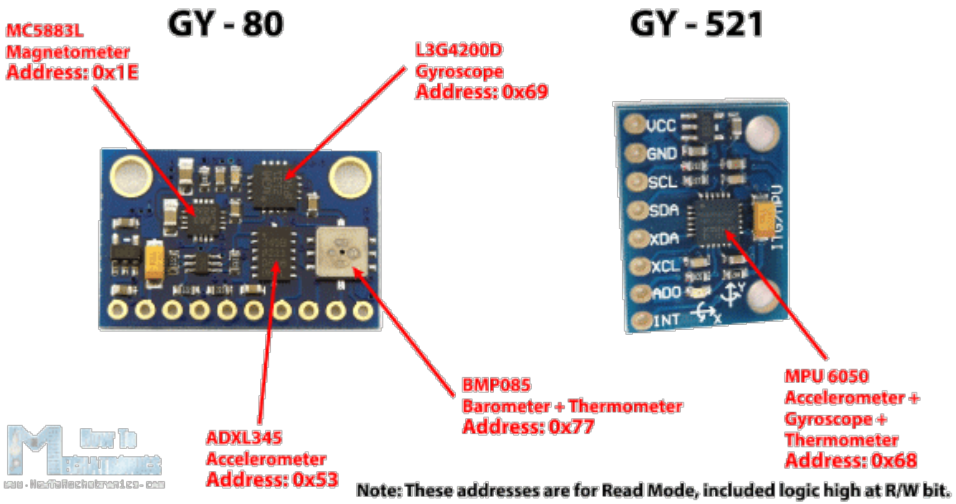
- Synchronous communication protocol
- Only two wires
- Each device on the bus is independently addressable
- Capability to handle multiple masters and multiple slaves
- Open-drain topology (holds the signal line high until a device pulls the line low)
- Half-duplex

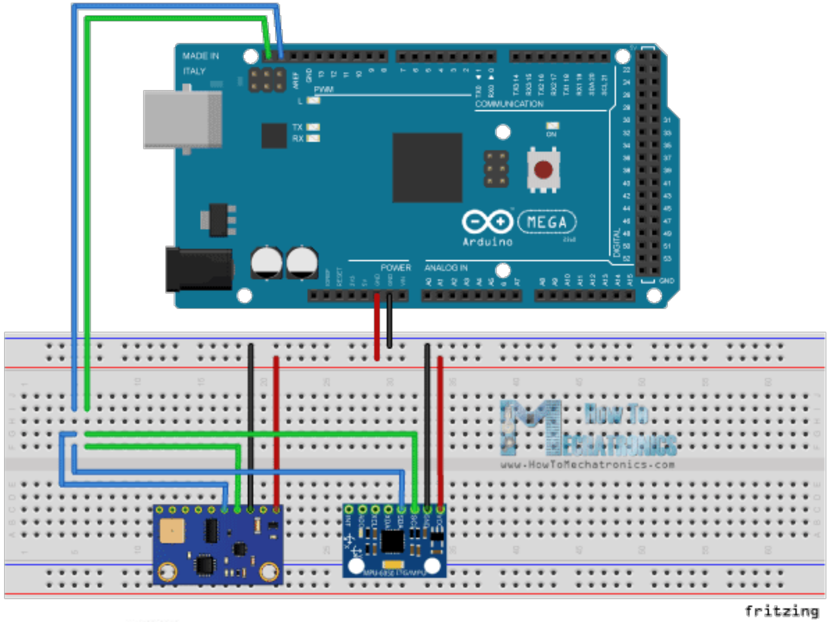
I2C

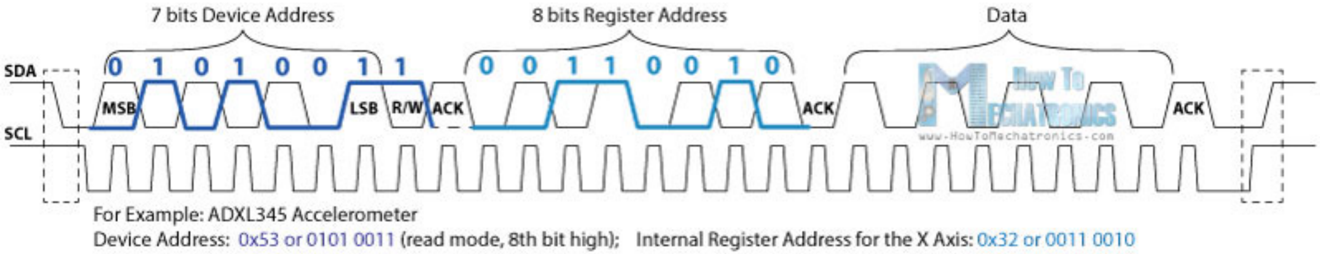


https://learn.sparkfun.com/tutorials/i2c/all/

Sensors using I2C interface

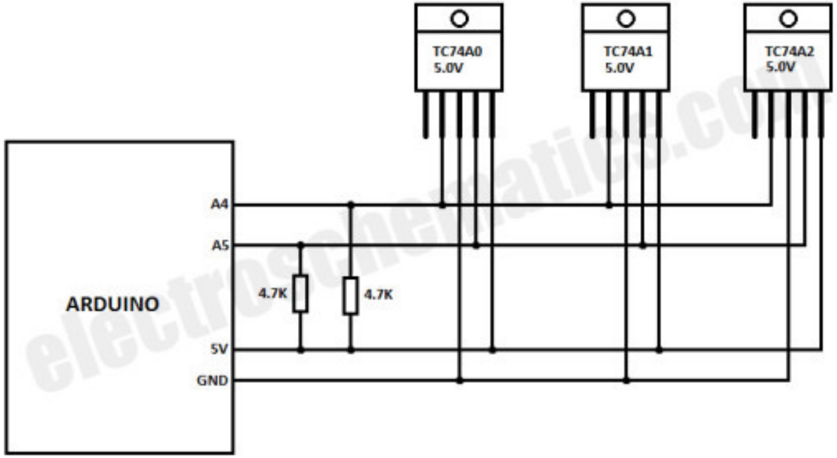






https://howtomechatronics.com/tutorials/arduino/how-i2c-communication-works-and-how-to-use-it-with-arduino/

Connecting three I2C temperature sensors to an Arduino board



SOT-23 Package Marking Codes

SOT-23 (V)	Address	Code	SOT-23 (V)	Address	Code
TC74A0-3.3VCT	1001 000	V0	TC74A0-5.0VCT	1001 000	U0
TC74A1-3.3VCT	1001 001	V1	TC74A1-5.0VCT	1001 001	U1
TC74A2-3.3VCT	1001 010	V2	TC74A2-5.0VCT	1001 010	U2
TC74A3-3.3VCT	1001 011	V3	TC74A3-5.0VCT	1001 011	U3
TC74A4-3.3VCT	1001 100	V4	TC74A4-5.0VCT	1001 100	U4
TC74A5-3.3VCT	1001 101*	V5	TC74A5-5.0VCT	1001 101*	U5
TC74A6-3.3VCT	1001 110	V6	TC74A6-5.0VCT	1001 110	U6
TC74A7-3.3VCT	1001 111	V7	TC74A7-5.0VCT	1001 111	U7

Note: * Default Address

https://www.electroschematics.com/reading-temperatures-i2c-arduino/

SPI (Serial Peripheral Interface)

- Synchronous serial data protocol.
- Always one master and different slaves (as peripheral devices)
- Full-duplex
- SPI interface has four lines:
  - 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 pulse which synchronise data transmission generated by the master
  - SS (Slave Select) – The pin which master can use to select a slave device.

SPI important parameters

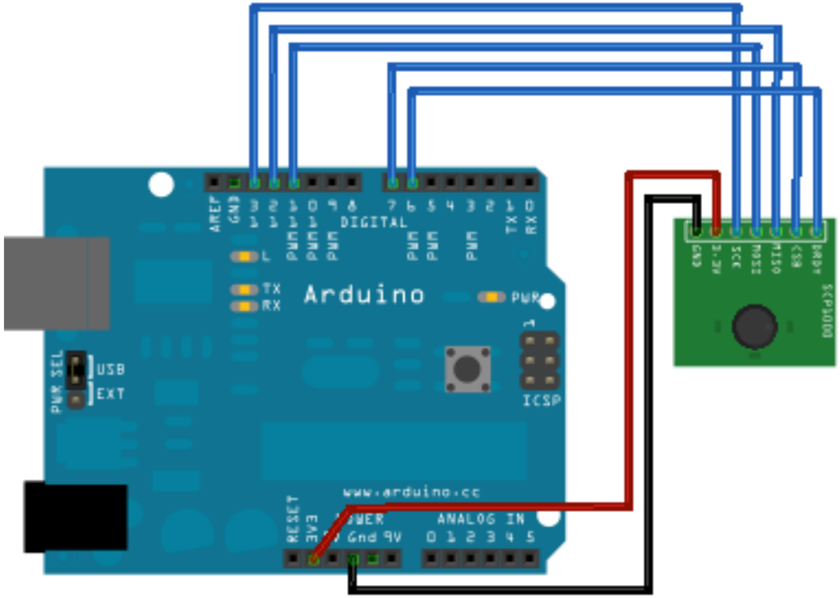
First parameter: Maximum SPI speed (For example 15 MHz, 15000000)

Second Parameter: MSBFIRST or LSBFIRST; whether you send MSB first or LSB first. In most SPI setting it is MSBFIRST.

Third parameter: SPI_Mode- there are four modes

Clock phase (CPHA) and Clock polarity (CPOL)

SPI



https://www.arduino.cc/en/reference/SPI

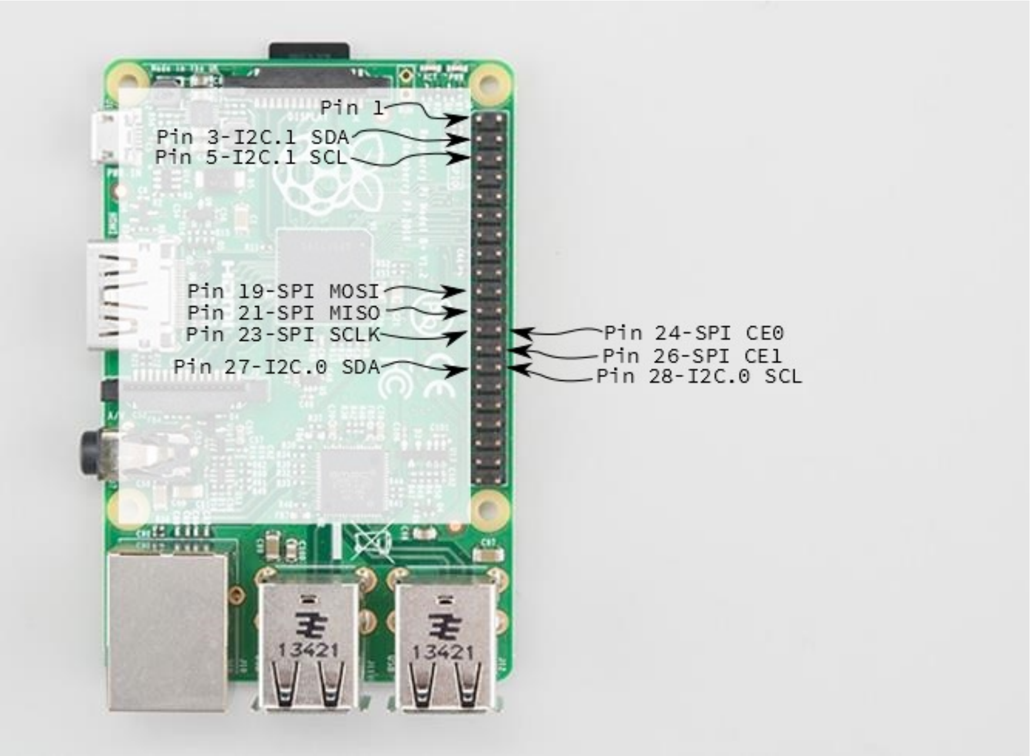
SPI VS I2C

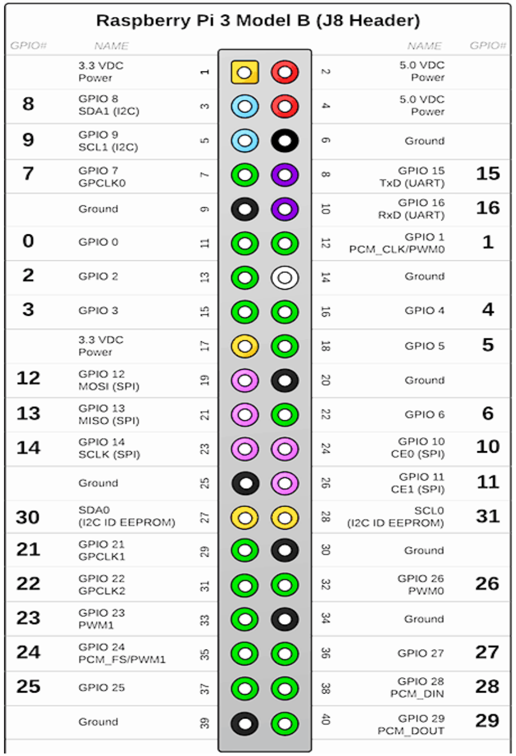
Both SPI and I2C are synchronous serial communication standards.

I2C uses only two lines for synchronous communication of master with multiple slaves

SPI on the other hand uses a separate slave select(SS) line for each slave. This may be challenging for master if numerous slave devices connect to a master using SPI.

I2C AND SPI Pins on Raspberry Pi





Questions?

Timer and Interrupts

Timers and Interrupts

Goal:

- To program tasks in Arduino that should be executed periodically
- or take too long and blocks our code (e.g., delay() function)

IoT Scenario

Environment Protection Authority Victoria (EPA) is concerned about the traffic noise impact on Victorians' wellbeing. They want to start monitoring the noise produced when vehicles go through certain roads. To do so, they are trying to deploy the following edge IoT system:

- Car detector sensor: Digital sensor car detected (1); no car (0).
- Noise detector: Analog sensor.
- Sending noise level via serial bus.
- Status LED to check the performance of the system:
  - Blinks every 2s if doing nothing.
  - Blinks every 0.5s if a car is detected, when measuring noise, and sending data.

Let’s code

First of all:

- Car detection
- LED will turn on if a car is detected
- Modification of the Example Button code

```
#define carDetectedPin 2
#define ledPin 3

// variables will change:
int carDetectedState = 0;

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the carDetected pin as an input:
  pinMode(carDetectedPin, INPUT);
}

void loop() {
  // read the state of the sensor to detect the car:
  carDetectedState = digitalRead(carDetectedPin);

  // check if the sensor detects a car.
  if (carDetectedState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  }
  else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```

Let’s code

Adding status LED:

- Every 2s: 200ms ON, 800ms OFF
- Every 0.5s: 200ms ON, 300ms OFF

```
#define carDetectedPin 2
#define ledPin 3

// variables will change:
int carDetectedState = 0;

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the carDetected pin as an input:
  pinMode(carDetectedPin, INPUT);
}

void loop() {
  // read the state of the sensor to detect the car:
  carDetectedState = digitalRead(carDetectedPin);

  // check if the sensor detects a car.
  if (buttonState == HIGH) {
    digitalWrite(ledPin, HIGH);
    delay(200);
    digitalWrite(ledPin, LOW);
    delay(300);
  }
  else {
    digitalWrite(ledPin, HIGH);
    delay(200);
    digitalWrite(ledPin, LOW);
    delay(1800);
  }
}
```

Let’s code

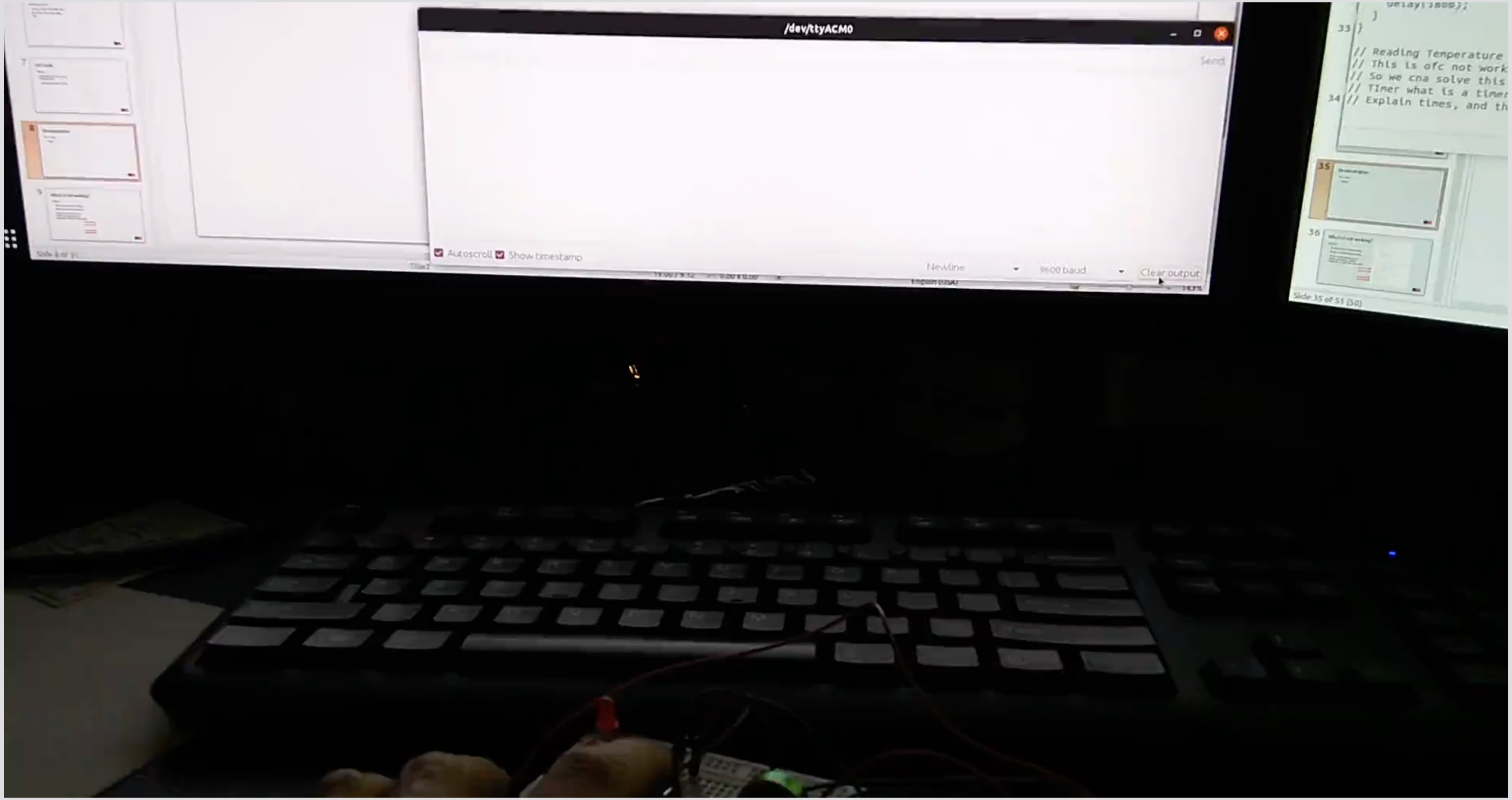
Finally:

- Reading from noise sensor (analog input).
- Sending data via serial bus.

```
#define carDetectedPin 2
#define ledPin 3
#define noiseSensorPin A0

// variables will change:
int carDetectedState = 0;

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the carDetected pin as an input:
  pinMode(carDetectedPin, INPUT);
  // Serial data at 9600 baud (bits per second)
  Serial.begin(9600);
}

void loop() {
  // read the state of the sensor to detect the car:
  carDetectedState = digitalRead(carDetectedPin);

  // check if the sensor detects a car.
  if (carDetectedState == HIGH) {
    digitalWrite(ledPin, HIGH);
    delay(200);
    digitalWrite(ledPin, LOW);
    delay(300);
    int noise = analogRead(noiseSensorPin);
    Serial.println(noise);
  }
  else {
    digitalWrite(ledPin, HIGH);
    delay(200);
    digitalWrite(ledPin, LOW);
    delay(1800);
  }
}
```


SWINBURNE UNIVERSITY OF TECHNOLOGY

Demonstration



Why is it not working?

delay();

- Arduino is not multitasking
- Delay is a blocking function.
- If the sensor detects a car while executing a delay function, it won't be executed.

```
#define carDetectedPin 2
#define ledPin 3
#define noiseSensorPin A0

// variables will change:
int carDetectedState = 0;

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the carDetected pin as an input:
  pinMode(carDetectedPin, INPUT);
  // Serial data at 9600 baud (bits per second)
  Serial.begin(9600);
}

void loop() {
  // read the state of the sensor to detect the car:
  carDetectedState = digitalRead(carDetectedPin);

  // check if the sensor detects a car.
  if (carDetectedState == HIGH) {
    digitalWrite(ledPin, HIGH);
    delay(200);
    digitalWrite(ledPin, LOW);
    delay(300);
    int noise = analogRead(noiseSensorPin);
    Serial.println(noise);
  } else {
    digitalWrite(ledPin, HIGH);
    delay(200);
    digitalWrite(ledPin, LOW);
    delay(1800);
  }
}
```

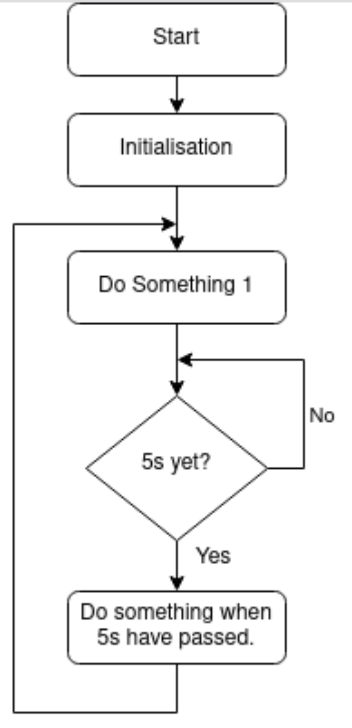
How can we fix it?

With Timers and Interrupts

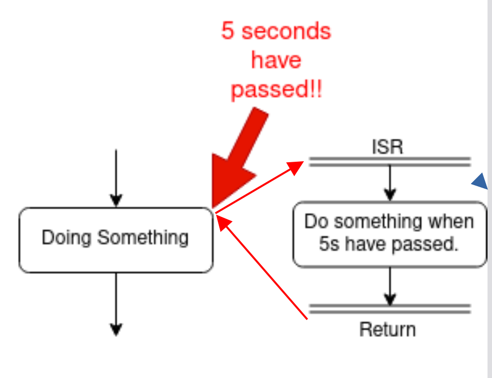
- Timers can trigger events after a certain period of time.
- The current process in the CPU will be halted to execute the interrupt.
- Once the interrupt finish, the program will resume.
- Timers and Interrupts are basic functionalities of a microcontroller

Interrupts vs Polling (Time)

Polling: constantly checking if something has happened

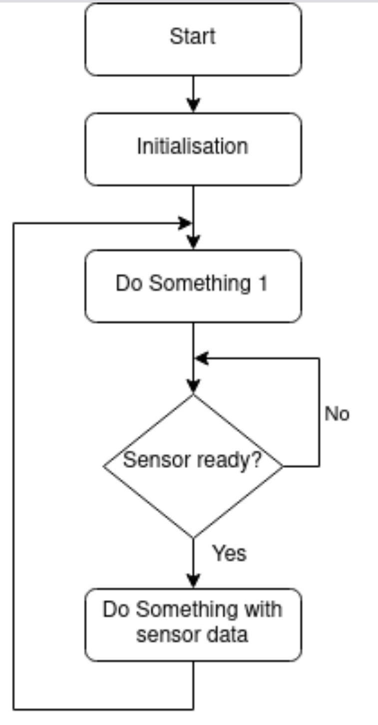


Interrupt: notification to the CPU that something has happened and needs attention

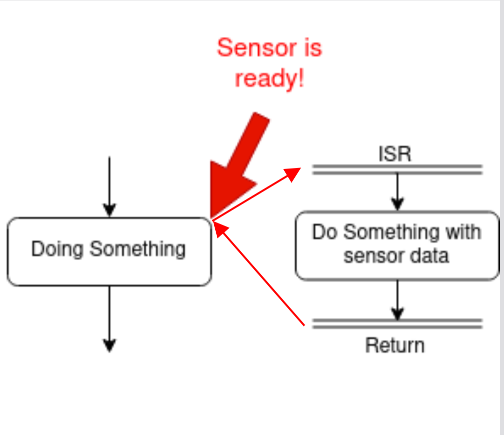


Interrupts vs Polling (Sensor)

Polling: constantly checking if something has happened

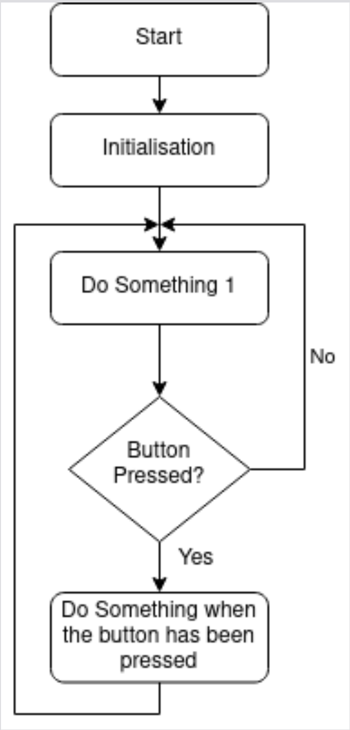


Interrupt: notification to the CPU that something has happened and needs attention

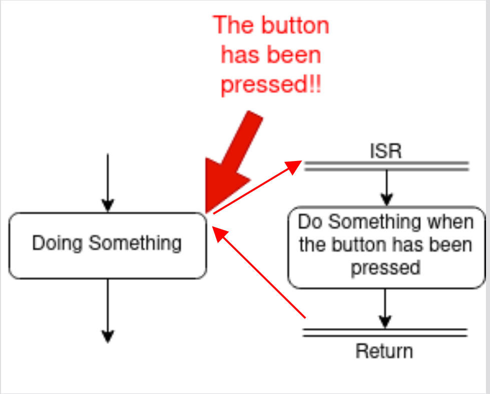


Interrupts vs Polling (Sensor)

Polling: constantly checking if something has happened

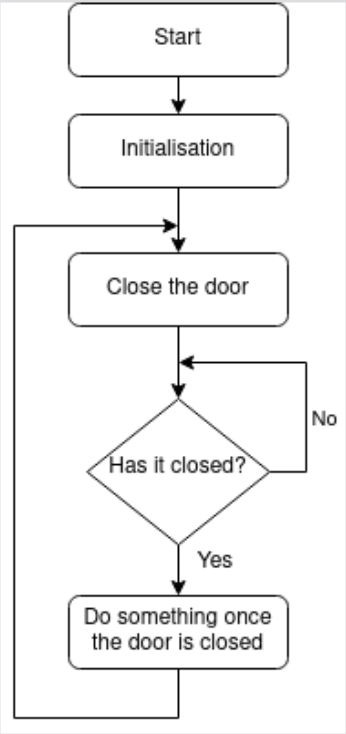


Interrupt: notification to the CPU that something has happened and needs attention

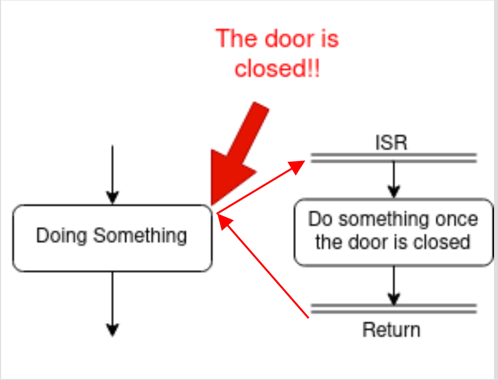


Interrupts vs Polling (Actuator)

Polling: constantly checking if something has happened



Interrupt: notification to the CPU that something has happened and needs attention



Interrupts vs Polling

- Interrupts allows devices (e.g., sensors, actuators, timers) to signal the CPU and to force execution of a particular piece of code at any time.
- Interrupt priorities allow the CPU to recognize some interrupts as more important than others
- Interrupt vectors allow the interrupting device to specify its handler.
- Polling checks the state of the device in a pre-defined sequence
- Polling wastes a lot of time checking a device which is doing nothing
- If the device freezes, the microcontroller most likely will get stuck

Timers

- **Timers are hardware components of the microcontroller that count clock pulses**
  - Normally working with the clock (CLK) of the microcontroller. However, external clocks can be used (we won't do that).
  - Clock used to measure timed events
  - Can be programmed using specialized timer registers.
  - Clock speed of Atmega328P microcontroller: **16 MHz.**
  - 16MHz = 16 Million cycles per second (not instructions per second!)
- **Timers can be configured to throw interrupts**
- **ATmega328P microcontroller (Arduino Uno) has 3 timers:**
  - Timer 0: 8-bit timer (0 - 255); already in use for other functionalities (e.g., *delay()*, *millis()* function). Thus, we won't use it!
  - Timer 1: 16-bit timer (0 – 65535); used for *analogWrite() pins 9, 10.*
  - Timer 2: 8-bit timer (0 – 255); used for *tone()* and *analogWrite() pins 3,11*.
- **All the information can be found in the datasheet of the microcontroller:**

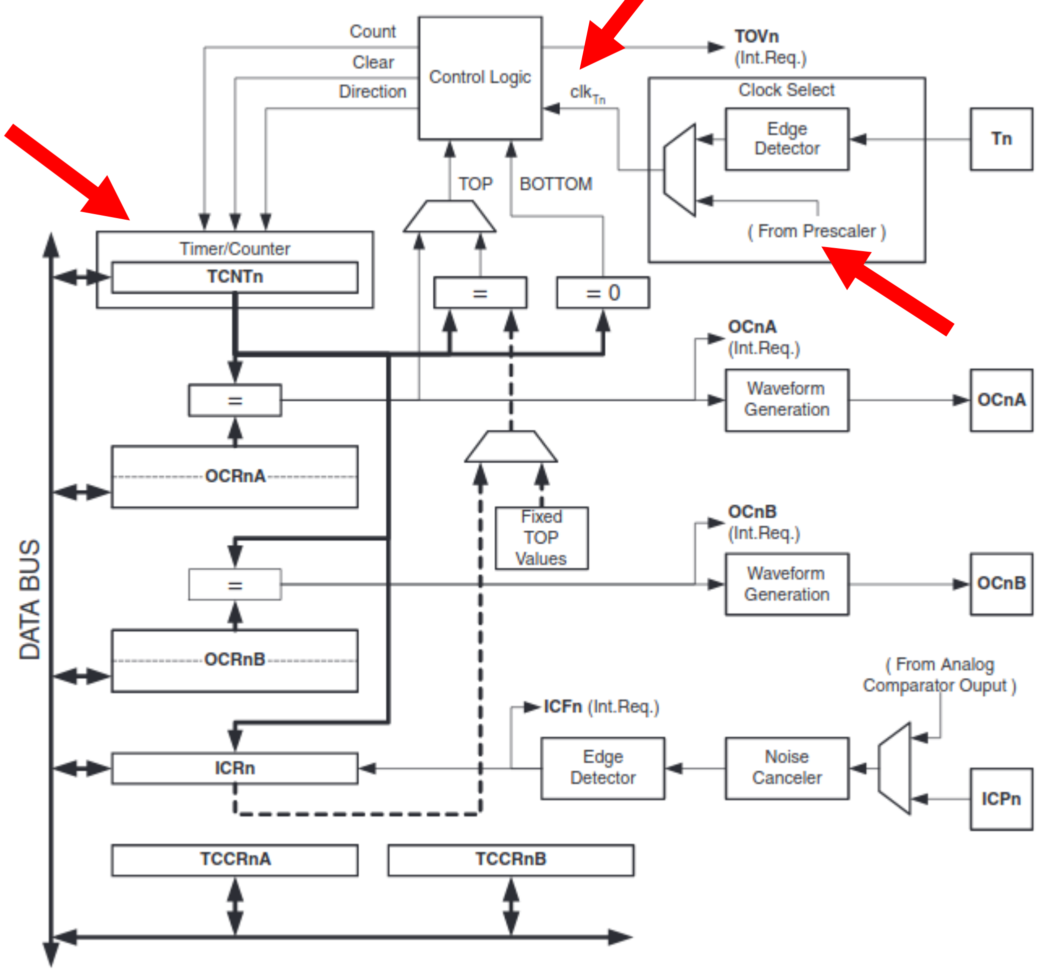
https://ww1.microchip.com/downloads/en/DeviceDoc/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061B.pdf

Timers

How to program a Timer?

- Timer block for Timer 0 and Timer 1. Timer 2 is very similar.
- Timer block extracted from datasheet (p.121)
- Timer/Counter (TCNTn) is incremented by clock pulses of clk_Tn
- clk_Tn: Clock timer N comes from Prescaler

Figure 16-1. 16-bit Timer/Counter Block Diagram[1]



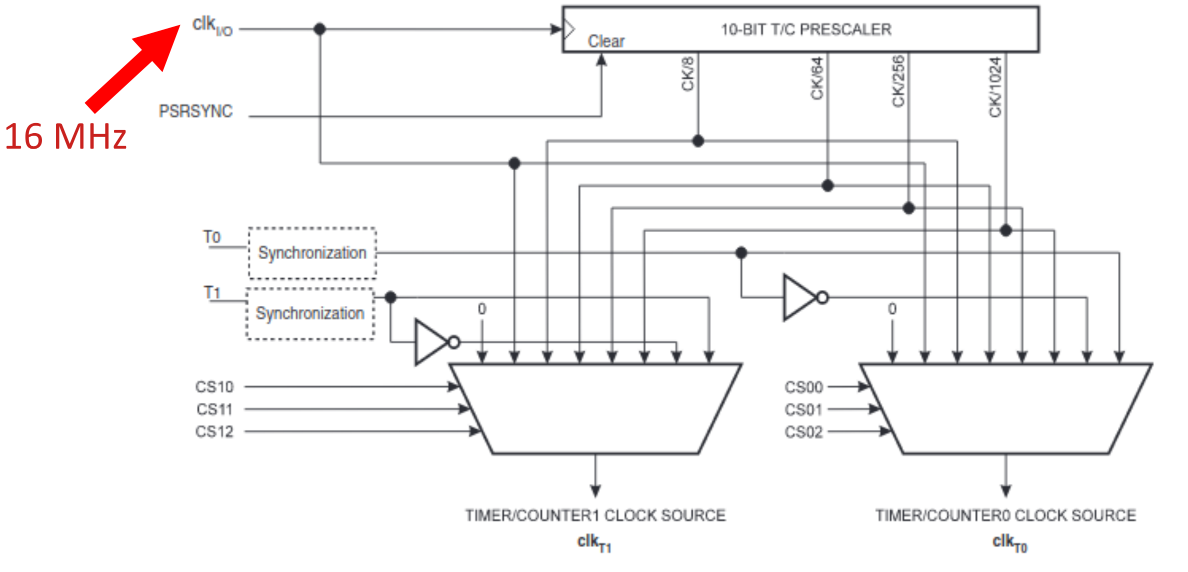
Timers

$$\frac{16}{8} MHz; \text{ or } \frac{16}{64} MHz; \text{ or } \frac{16}{256} MHz; \text{ or } \frac{16}{1024} MHz$$

What is a Prescaler?

- Prescaler block extracted from datasheet (p.148).
- The prescaler can divide the system clock by the selected value (only 4 options).
- E.g., Clk/1024 (Prescaler 1024) will count every 0.064 ms (15625 Hz).
- Timer 1 and 2 share the same prescaler module, but they can have different prescaler settings.

Figure 17-2. Prescaler for Timer/Counter0 and Timer/Counter1[1]



Timers

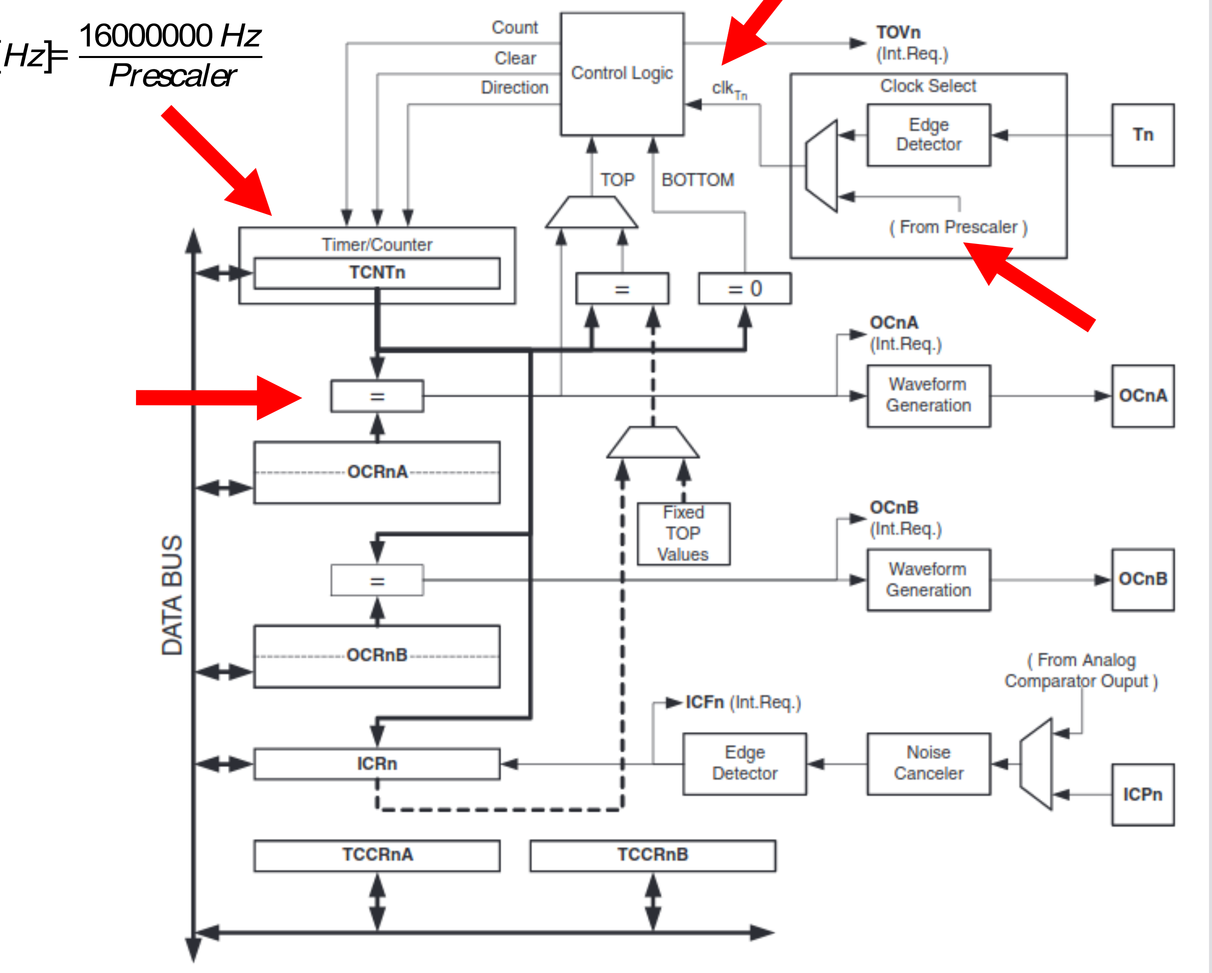
$$Timer/CounterSpeed[Hz] = \frac{16000000\ Hz}{Prescaler}$$

How to program a Timer?

$$InterruptFreq.[Hz] = \frac{16000000\ Hz}{Prescaler * (1 + OCRnA)}$$

- **Clk_Tn increments TCNTn counter every clock pulse**
- **TCNTn is compared to OCRnA register**
  - OCR1A is a 16 bit register (Timer1)
  - OCR2A is an 8 bit register (Timer2)
- **If it matches, throws an interrupt**

Figure 16-1. 16-bit Timer/Counter Block Diagram[1]



Timers

Let's find out the register value configuration

- Timer 1, so N=1
- Prescaler = 256
  - CS10 = 0; CS11 = 0; CS12 = 1
- OCR1A = 6249
- WGM12 = 1; Clear Timer on Compare match (CTC) mode
- OCIE1A = 1; Enable interrupt when OCR1A match the counter (TCNT1)

16.11.1 TCCR1A – Timer/Counter1 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

16.11.2 TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

16.11.8 TIMSK1 – Timer/Counter1 Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x6F)	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1	TIMSK1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 16-5. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}/1$ (No prescaling)
0	1	0	$clk_{I/O}/8$ (From prescaler)
0	1	1	$clk_{I/O}/64$ (From prescaler)
1	0	0	$clk_{I/O}/256$ (From prescaler)
1	0	1	$clk_{I/O}/1024$ (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

Timers

Let's Code! Initialisation

- Timer configuration goes at setup()
- First of all we stop the interrupts.
- Then we reset the configuration registers.
- So we can setup our configuration.
- Finally, we enable interrupts again.
- We will need to code a timer interrupt function. Variables used in that function must be declared "volatile"

```
#define carDetectedPin 2
#define ledPin 3
#define noiseSensorPin A0

// variables will change:
int carDetectedState = 0;
volatile int timer1InterruptCounter = 0;

void setup() {
  // TIMER 1 Configuration for 100ms
  cli();  // Stop interrupts
  TCCR1A = 0;     // Reset TCCR1A register
  TCCR1B = 0;     // Reset TCCR1B register
  TCNT1  = 0;     // Counter = 0
  OCR1A = 6249;   // Compare match register for 100ms
  TCCR1B |= (1 << WGM12);   // Clear Timer on Compare (CTC) on
  TCCR1B |= (1 << CS12) | (0 << CS11) | (0 << CS10);   // Prescaler 256
  TIMSK1 |= (1 << OCIE1A); // Enable interrupt when TCNT1 == OCR1A
  sei();  // Allow again interrupts

  // Initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // Initialize the carDetected pin as an input:
  pinMode(carDetectedPin, INPUT);
  // Serial data at 9600 baud (bits per second)
  Serial.begin(9600);

}
```

Timers

Let's Code! Interrupt function

- **Interrupt functions should be SHORT and FAST! NEVER put a delay() there;**
- **ISR: Interrupt Service Routine**
- **With an interrupt counter variable (every 100ms) we can make the LEDs blink at the same pattern as before**

200ms

300ms

200ms

1800ms

```
ISR(TIMER1_COMPA_vect){
  // Timer 1 Interrupt every 100ms (Comparing with OCR1A)
  timer1InterruptCounter++;
  timer1InterruptCounter %= 20;
}

void loop() {
  // read the state of the sensor to detect the car:
  carDetectedState = digitalRead(carDetectedPin);

  // Check if the sensor detects a car.
  if (carDetectedState == HIGH) {
    if (timer1InterruptCounter%5 < 2) {
      digitalWrite(ledPin, HIGH);
    }
    else {
      digitalWrite(ledPin, LOW);
    }
    int noise = analogRead(noiseSensorPin);
    Serial.println(noise);
  }
  else {
    if (timer1InterruptCounter < 2) {
      digitalWrite(ledPin, HIGH);
    }
    else {
      digitalWrite(ledPin, LOW);
    }
  }
 }
```

Interrupt

How to code an external interrupt?

- **Arduino UNO has two external interrupt pins: Digital pin 2 and Digital pin 3**
- **External interrupts can be programmed using Arduino**
  - https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/
- **Interrupts can be triggered with the following modes:**
  - **LOW:** to trigger the interrupt whenever the pin is low,
  - **CHANGE:** to trigger the interrupt whenever the pin changes value
  - **RISING:** to trigger when the pin goes from low to high,
  - **FALLING:** for when the pin goes from high to low.
- **WARNING: programming push buttons with interrupts is not the best idea.**

Interrupt

Let's Code! Car sensor via interrupt.

- **Make variable volatile**
- **Attach interrupt in setup()**
- **Code ISR (Interrupt Service Routine)**
- **Remove from loop():**
  - // read the state of the sensor to detect the car:
    carDetectedState = digitalRead(carDetectedPin);

```
#define carDetectedPin 2
#define ledPin 3
#define noiseSensorPin A0
// read the state of the sensor to detect the car:
// variables will change:
volatile bool carDetectedState = false;
volatile int timer1InterruptCounter = 0;

void setup() {
  // INTERRUPT Car detection sensor
  attachInterrupt(digitalPinToInterrupt(carDetectedPin), car_detected, CHANGE);
```

```
void car_detected() {
  carDetectedState = !carDetectedState;
}
```

```
void loop() {

  // Check if the sensor detects a car.
  if (carDetectedState == HIGH) {
    if (timer1InterruptCounter%5 < 2) {
      digitalWrite(ledPin, HIGH);
    }
    else {
      digitalWrite(ledPin, LOW);
    }
    int noise = analogRead(noiseSensorPin);
    Serial.println(noise);
  }
  else {
    if (timer1InterruptCounter < 2) {
      digitalWrite(ledPin, HIGH);
    }
    else {
      digitalWrite(ledPin, LOW);
    }
  }
}
```

Interrupt

Vector

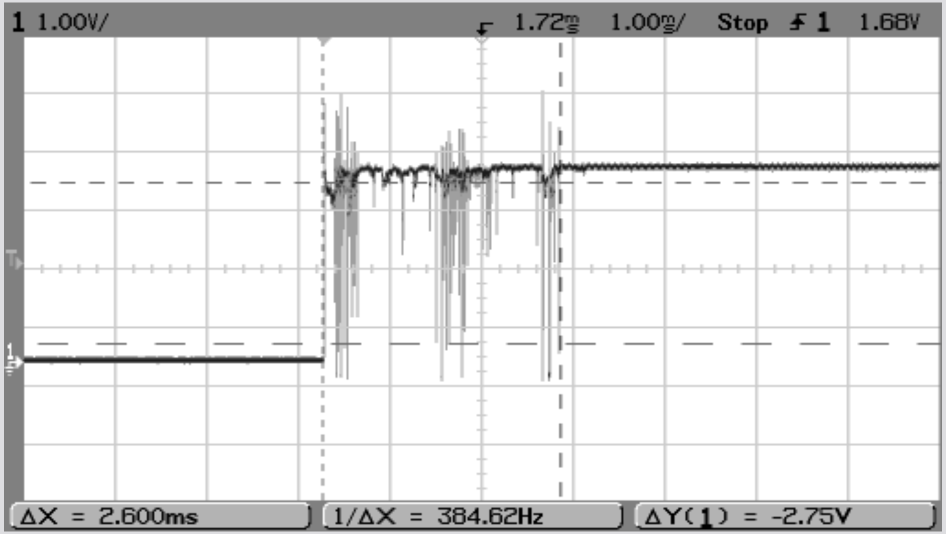
- Smaller the vector number, higher the priority
- External Interrupts used for real-time systems (critical operations)
- Push buttons used for interrupts in a microcontroller is not a good idea.
- Use interruptions wisely

Table 11-1. Reset and Interrupt Vectors in ATmega328P

Vector No.	Program Address	Source	Interrupt Definition
1	0x0000	RESET	External pin, power-on reset, brown-out reset and watchdog system reset
2	0x002	INT0	External interrupt request 0
3	0x0004	INT1	External interrupt request 1
4	0x0006	PCINT0	Pin change interrupt request 0
5	0x0008	PCINT1	Pin change interrupt request 1
6	0x000A	PCINT2	Pin change interrupt request 2
7	0x000C	WDT	Watchdog time-out interrupt
8	0x000E	TIMER2 COMPA	Timer/Counter2 compare match A
9	0x0010	TIMER2 COMPB	Timer/Counter2 compare match B
10	0x0012	TIMER2 OVF	Timer/Counter2 overflow
11	0x0014	TIMER1 CAPT	Timer/Counter1 capture event
12	0x0016	TIMER1 COMPA	Timer/Counter1 compare match A
13	0x0018	TIMER1 COMPB	Timer/Counter1 compare match B
14	0x001A	TIMER1 OVF	Timer/Counter1 overflow
15	0x001C	TIMER0 COMPA	Timer/Counter0 compare match A
16	0x001E	TIMER0 COMPB	Timer/Counter0 compare match B
17	0x0020	TIMER0 OVF	Timer/Counter0 overflow
18	0x0022	SPI, STC	SPI serial transfer complete
19	0x0024	USART, RX	USART Rx complete
20	0x0026	USART, UDRE	USART, data register empty
21	0x0028	USART, TX	USART, Tx complete
22	0x002A	ADC	ADC conversion complete
23	0x002C	EE READY	EEPROM ready
24	0x002E	ANALOG COMP	Analog comparator
25	0x0030	TWI	2-wire serial interface
26	0x0032	SPM READY	Store program memory ready

Button Issue

- **Contact bounce**
- **Can be debounced via:**
  - Software
  - Hardware



Questions?

Timers

A bit of maths:

$$InterruptFrequency[Hz] = \frac{16000000\ Hz}{Prescaler * (1 + OCRnA)}$$

$$InterruptInterval[s] = \frac{Prescaler * (1 + OCRnA)}{16000000\ Hz}$$

$$OCRnA = \frac{InterruptInterval[s] * 16000000\ Hz}{Prescaler} - 1$$

If I want a timer interrupt every 100ms:

$$OCRnA = \frac{0.1 * 16000000\ Hz}{Prescaler} - 1$$

- **Prescaler = 1**

  $$OCRnA = \frac{0.1 * 16000000\ Hz}{1} - 1 = 1599999$$

- **Prescaler = 8**

  $$OCRnA = \frac{0.1 * 16000000\ Hz}{8} - 1 = 199999$$

- **Prescaler = 64**

  $$OCRnA = \frac{0.1 * 16000000\ Hz}{64} - 1 = 24999$$

- **Prescaler = 256**

  $$OCRnA = \frac{0.1 * 16000000\ Hz}{256} - 1 = 6249$$

- **Prescaler = 1024**

  $$OCRnA = \frac{0.1 * 16000000\ Hz}{1024} - 1 = 1561.5$$