Workshop VII

Interrupts, Timers, and TM1637

SECTION I

Interrupts

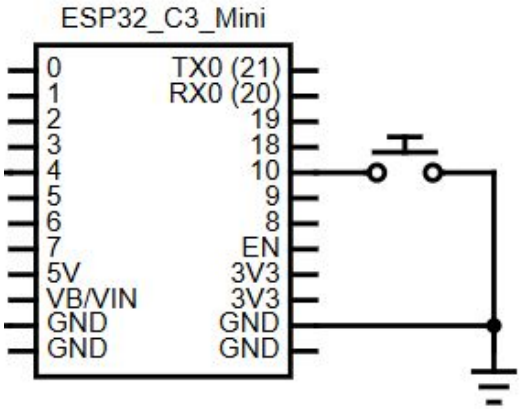
Interrupts

- An **interrupt** is a request for the CPU to **halt the currently executing code** when an event occurs
- In other words, the current code is paused, and a different block of code runs
  - The CPU suspends the current program to *handle* the event by executing a function called an **interrupt handler** or **interrupt service routine** (**ISR**)
  - When the interrupt handler finishes execution, the CPU returns to the old program

Interrupts

- Interrupts on ESP32 are triggered by the signal at a pin

- We can use pull-up resistors for this!
  - When we press the button, the voltage at the pin changes from HIGH to LOW. This would be considered a *falling edge*

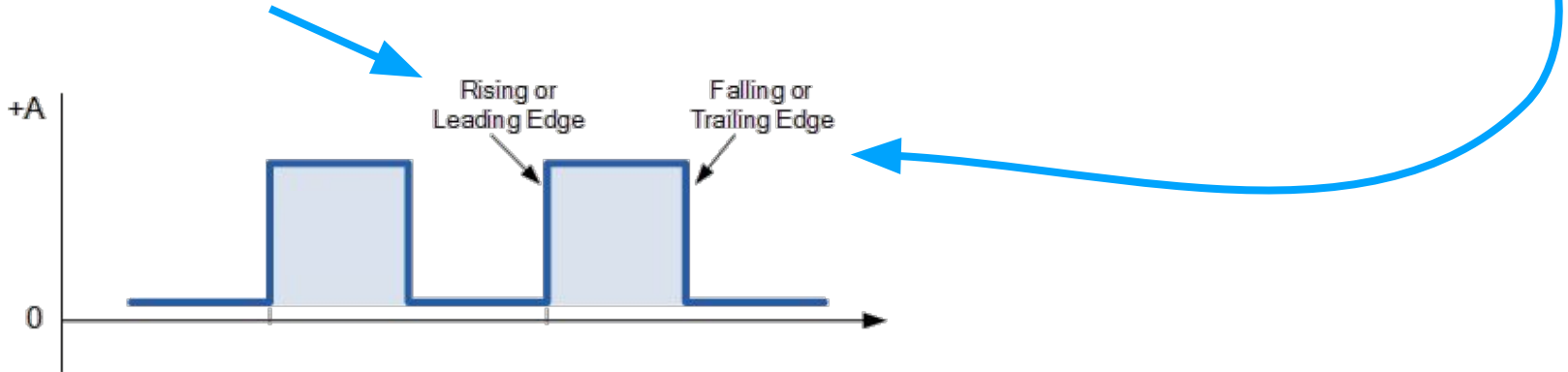


Setting Interrupts

- Interrupts are configured in the `void setup()` block
- `attachInterrupt(interrupt, ISR, mode)`
  - Initializes an interrupt
  - *`interrupt`* - the interrupt number (**not a pin number**)
  - *`ISR`* - the name of the function to call as the interrupt handler
    - This function must be defined beforehand
  - *`mode`* - configures the timing of the IRQ
    - You will set the mode to one of the following constants…

Setting Interrupts (Cont'd)

- The options for ***mode*** are…
  - **`LOW`** - when the pin is a LOW voltage
  - **`CHANGE`** - when the pin changes value
  - **`FALLING`** - when the pin voltage changes from HIGH to LOW
  - **`RISING`** - when the pin voltage changes from LOW to HIGH



Setting Interrupts (Cont'd)

- `digitalPinToInterrupt(pin)`
  - This function converts a digital pin number to its corresponding interrupt number
- It is **recommended that you call `attachInterrupt` as follows:**
  - `attachInterrupt(digitalPinToInterrupt(pin), ISR, mode)`
  - This approach avoids confusion involving digital pin and interrupt numbers

Onboard LED Exercise

Write a program that satisfies the following requirements:

- You must breadboard a circuit in which **one pushbutton turns on the ESP32's onboard LED and the other pushbutton turns it off**.
- The LED should only turn on **when you let go** of the ON button (it should stay on afterwards as well!).
- The LED should only turn off **immediately** when you press the OFF button.
- The ESP32 should **print a message** whenever a button is pressed.
- Onboard LED is pin 8 (built-in)
    - **Reverse enabled**: turned on by writing LOW and off by writing HIGH

Onboard LED Exercise Code

```
const int on_btn = 3;
const int off_btn = 4;
const int led = 8;

void turnOn() {
  digitalWrite(led, LOW);
  Serial.println("ON");
}

void turnOff() {
  digitalWrite(led, HIGH);
  Serial.println("OFF");
}

void setup() {
  Serial.begin(115200);
  pinMode(on_btn, INPUT_PULLUP);
  pinMode(off_btn, INPUT_PULLUP);
  pinMode(led, OUTPUT);
  attachInterrupt(digitalPinToInterrupt(on_btn), turnOn, RISING);
  attachInterrupt(digitalPinToInterrupt(off_btn), turnOff, FALLING);
}

void loop() {

}
```

SECTION II

Timers

Arduino Timer Library

- We will use one of the Arduino's internal timers to **measure time**
  - To do so, we will enlist the help of the Timer library, which will need to be downloaded and installed to the Arduino IDE
  - Make sure to use `#include <Timer.h>`

Arduino Timer Library (Cont'd)

- First, instantiate a `Timer` object as `Timer timer;`
  - Time is **measured in milliseconds**
  - A timer has 3 states: `RUNNING`, `PAUSED`, or `STOPPED`
    - `STOPPED` is default state
- `timer.start()` starts the timer and puts it in the `RUNNING` state
- `timer.pause()` pauses the timer and puts it in the `PAUSED` state
- `timer.resume()` resumes the timer from the `PAUSED` state, putting it back in the `RUNNING` state

Arduino Timer Library (Cont'd)

- `timer.stop()` stops the timer and put it in the `STOPPED` state
  - The time recorded since the last `timer.start()` is **reset** the next time the `timer.start()` is called
- `timer.read()` returns the time recorded since the last `timer.start()`
- `timer.state()` returns a byte value indicating the timer's current state
  - `0 - STOPPED`
  - `1 - RUNNING`
  - `2 - PAUSED`

Timer Exercise

Write a program that satisfies the following requirements:

- You must write a program in which the ESP32 **continuously runs a timer and converts the current timer value to minutes and seconds,** printing all three values to the serial monitor
- Example output:
  - Time (ms): 301000
  - Minutes: 5
  - Seconds: 1

Timer Exercise Code

```
#include <Timer.h>

Timer timer;

void setup() {
  Serial.begin(115200);
  timer.start();
}

void loop() {
  int time = timer.read();
  int min = time / 60000;
  int sec = time % 60000 / 1000;

  Serial.print("Time (ms):\t");
  Serial.print(time);
  Serial.println();

  Serial.print("Minutes:\t");
  Serial.print(min);
  Serial.println();
  Serial.print("Seconds:\t");
  Serial.print(sec);
  Serial.println();

  delay(207);
}
```

SECTION III

TM1637

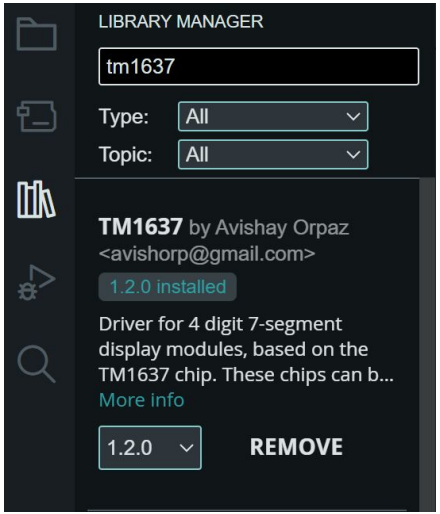
TM1637 Display

- The TM1637 is a 7-segment display
  - Shows 4 digits, each with 7 LED segments

- Pinout
  - CLK - I^{2}C SCL
  - DIO - I^{2}C SDA
  - GND - ground
  - VCC - 5V power

TM1637 Display Library

- Install the TM1637 library via the Arduino IDE library manager
  - Author is Avishay Orpaz



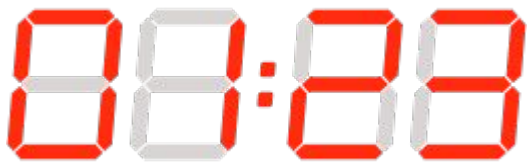
- Include this library with `#include <TM1637Display.h>`

TM1637 Library Functions

- `TM1637Display display(int CLK, int DIO);`
  - Initializes a display object using two ESP32 pin numbers for I^2C
- `display.clear()` clears the display
- `display.setBrightness(int brightness)`
  - sets segment brightness from a value 0 to 7 (7 being the brightest)
- `display.showNumberDecEx(int num, 0x40, bool leading_zero)`
  - `num` - exact number to display
  - `0x40` - enables colon in middle of display
  - `leading_zero` - turns on a leading 0 for numbers less than 4 digits if true

Example

```
display.showNumberDecEx(123, 0x40, true);
```



- Number displayed is 123
- Colon is enabled
- Last parameter turns on a 0 before the number

Display Exercise

Add to the previous timer exercise code:

- You must write a program in which the ESP32 **continuously runs a timer and displays the current time in minutes and seconds to the TM1637**

- Hint: You will need to do some math to convert the time in minutes and seconds to a number to display on the TM1637

Display Exercise Code

I/A

```ino
#include <Timer.h>
#include <TM1637Display.h>

const int CLK = 4;
const int DIO = 6;

Timer timer;
TM1637Display display(CLK, DIO);

void setup() {
  Serial.begin(115200);
  timer.start();

  display.clear();
  display.setBrightness(7);
}

void loop() {
  int time = timer.read();
  int min = time / 60000;
  int sec = time % 60000 / 1000;

  Serial.print("Time (ms):\t");
  Serial.print(time);
  Serial.println();

  Serial.print("Minutes:\t");
  Serial.print(min);
  Serial.println();
  Serial.print("Seconds:\t");
  Serial.print(sec);
  Serial.println();

  int result = min * 100 + sec;

  display.showNumberDecEx(result, 0x40, true);

  delay(207);
}
```

FAIR USE DISCLAIMER

Copyright Disclaimer under section 107 of the Copyright Act 1976, allowance is made for “fair use” for purposes such as criticism, comment, news reporting, teaching, scholarship, education and research.

Fair use is a use permitted by copyright statute that might otherwise be infringing.

Non-profit, educational or personal use tips the balance in favor of fair use.

CC BY-NC-SA 4.0