Microprocessors Lab

4- External Interrupts

Instructor: Dr. Mohsen Raji

Graders:
- Mohammad Hossein Hashemi
- Reza Hesami
- Negin Shirvani

Shiraz University – Fall 2022

دانشگاه شیراز

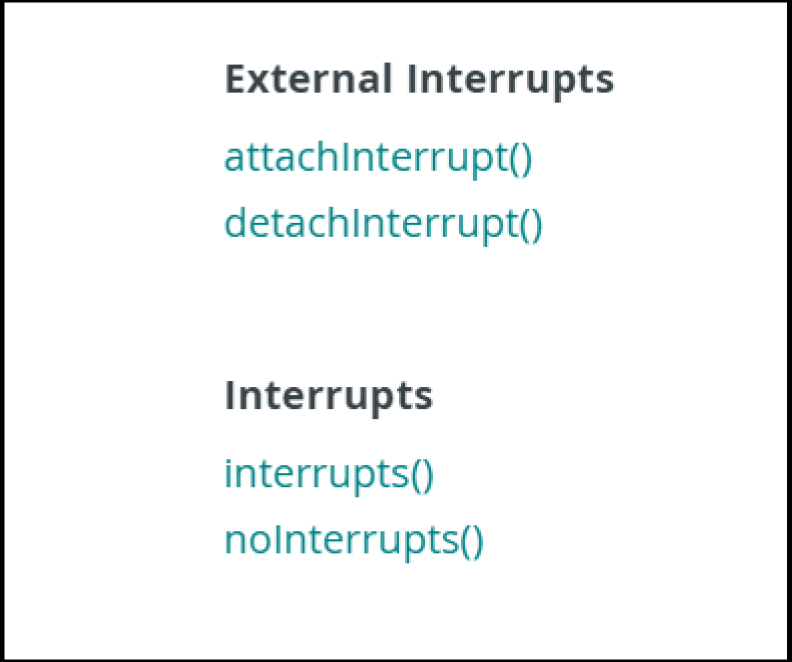
Shiraz University

Based on the slides by:
- Hossein Dehghanipour
- Mohammad Hossein Allah Akbari

1 - What is Interrupt?
2 – How to use it?
3 – Codes

- Microprocessors have two ways of fetching data from an external peripheral, Polling and Interrupt.
- What is Polling? Google it ☺ You must be pretty familiar with Polling in Operating System Course. If you are not, visit this Link.
- So, what is interrupt? Google it ☺ You must be pretty familiar with interrupts in Micro Processors Course. If you are not, visit this Link.
- How to use it?
  - There are some pre-built functions in Arduino in order to attach a pin as an interrupt pin or detach that pin from being an interrupt pin. Look at the next slide.

- According to Arduinos official website, there are 4 functions that must be remembered:
- Reference: https://www.arduino.cc/reference/en/



External Interrupts

attachInterrupt()

detachInterrupt()

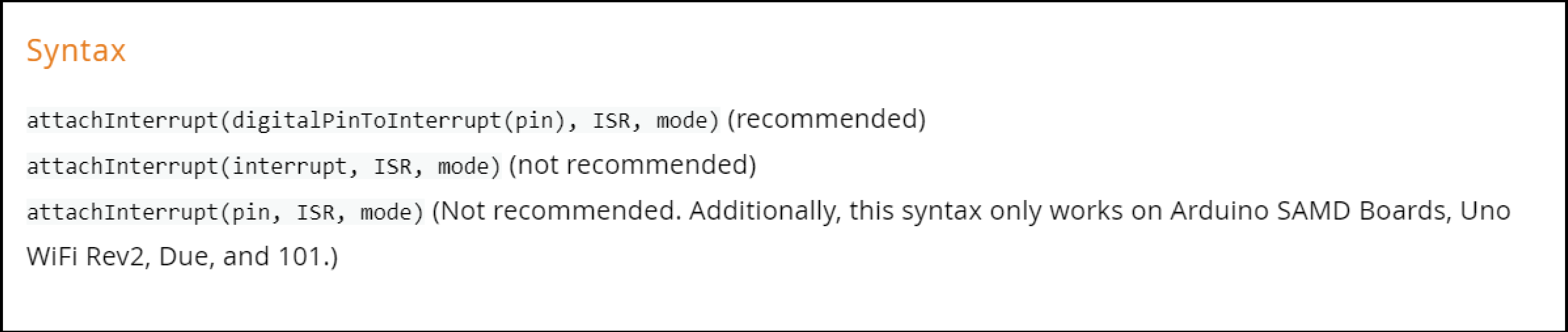
Interrupts

interrupts()

noInterrupts()

attachInterrupt()

- The first parameter to attachInterrupt() is an interrupt number. Normally you should use digitalPinToInterrupt(pin) to translate the actual digital pin to the specific interrupt number. For example, if you connect to pin 3, use digitalPinToInterrupt(3) as the first parameter to attachInterrupt().



Syntax

`attachInterrupt(digitalPinToInterrupt(pin), ISR, mode)` (recommended)

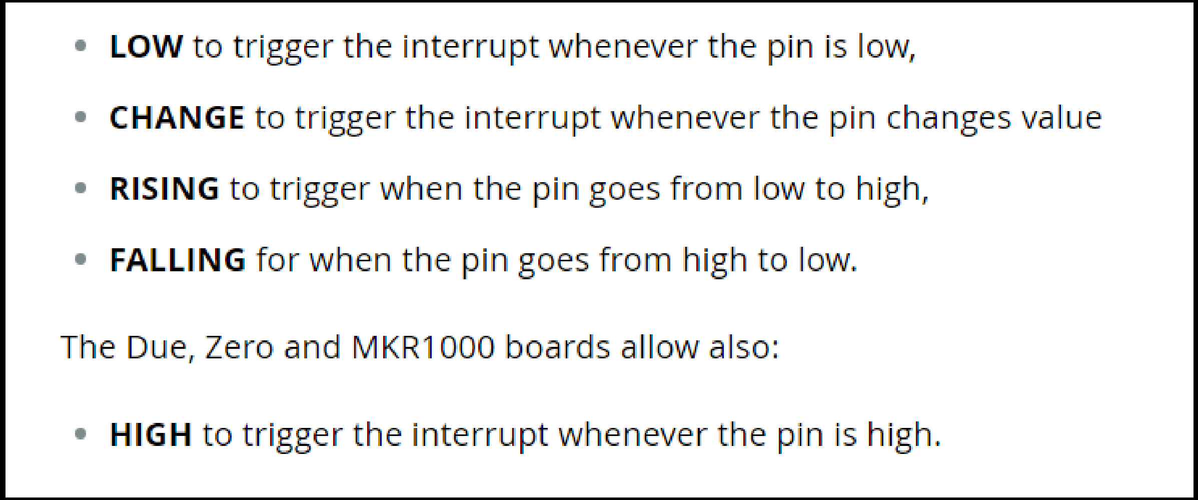
`attachInterrupt(interrupt, ISR, mode)` (not recommended)

`attachInterrupt(pin, ISR, mode)` (Not recommended. Additionally, this syntax only works on Arduino SAMD Boards, Uno WiFi Rev2, Due, and 101.)

Pin: The pin number that we want to act as an interrupt pin.
ISR: Interrupt Service Routine (A function that we write to be invoked when the interrupt happens).
Mode: Defines when the interrupt should be triggered(LOW, HIGH, RISING, FALLING, CHANGE)

Reference: https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/

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.

The Due, Zero and MKR1000 boards allow also:

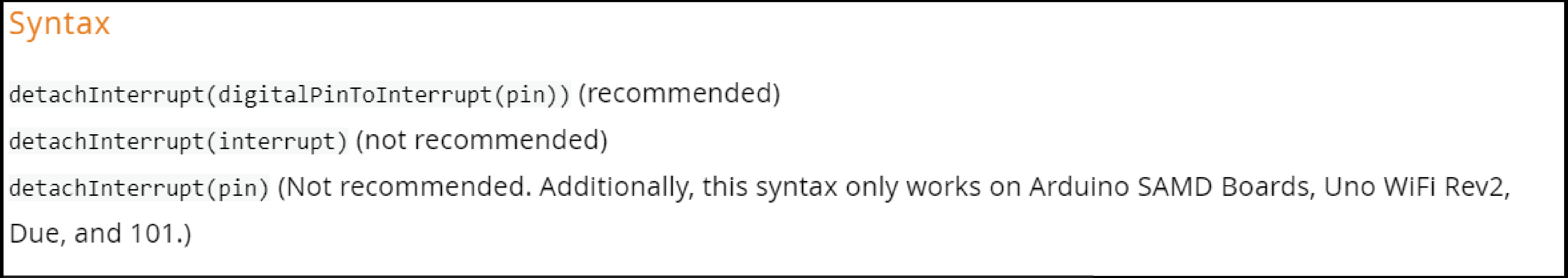
- **HIGH** to trigger the interrupt whenever the pin is high.

ISR

The ISR is called when the interrupt occurs. This function must take NO parameters and return NOTHING. This function is sometimes referred to as an interrupt service routine.

detachInterrupt()

- Turns off the given interrupt.

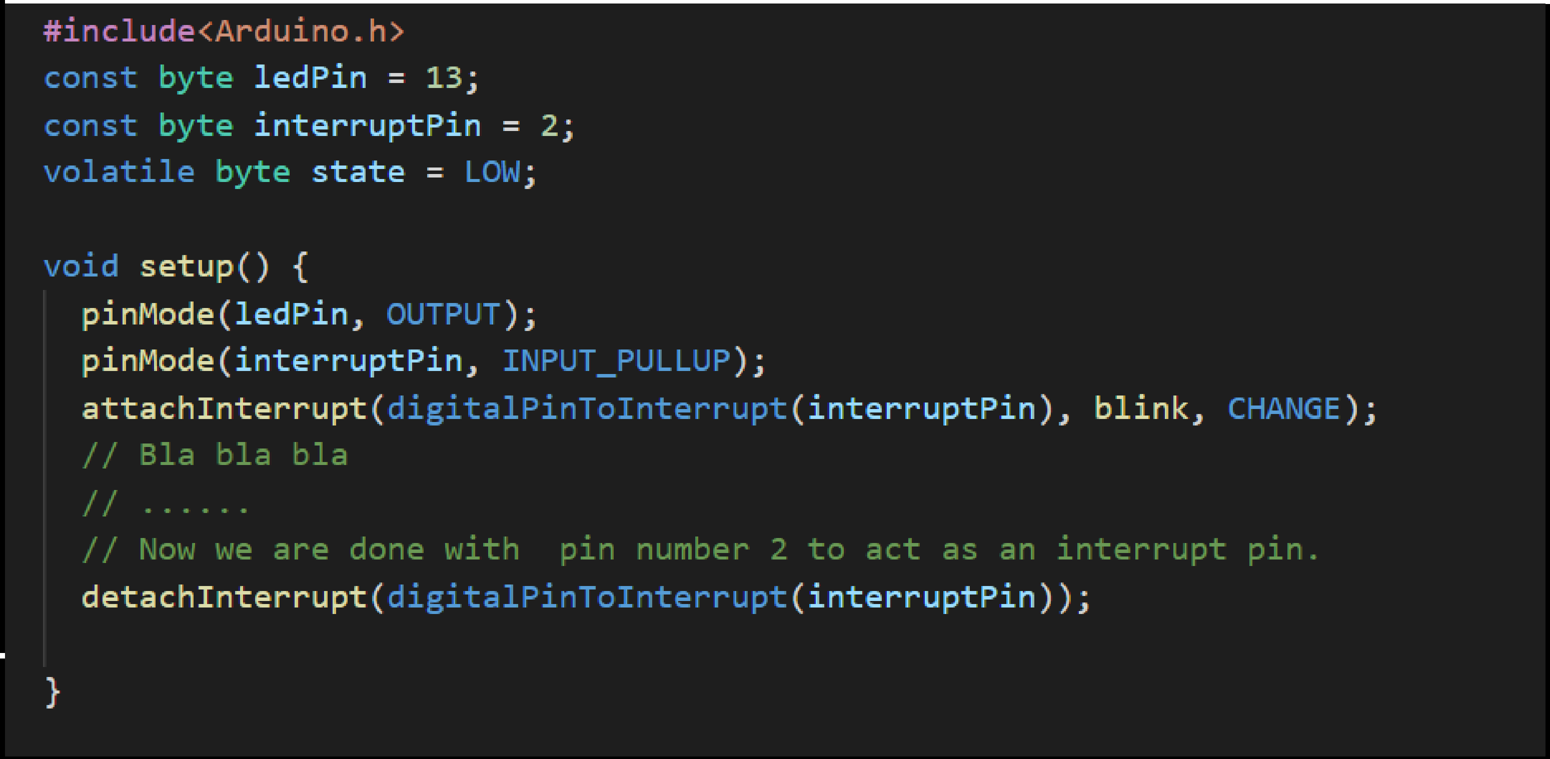


Syntax

`detachInterrupt(digitalPinToInterrupt(pin))` (recommended)

`detachInterrupt(interrupt)` (not recommended)

`detachInterrupt(pin)` (Not recommended. Additionally, this syntax only works on Arduino SAMD Boards, Uno WiFi Rev2, Due, and 101.)



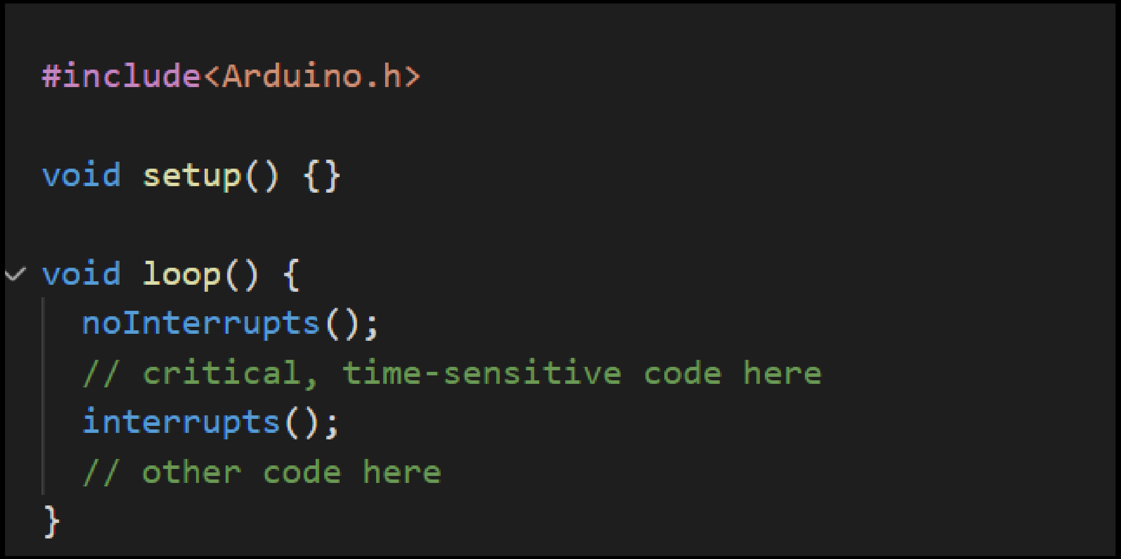
```
#include<Arduino.h>
const byte ledPin = 13;
const byte interruptPin = 2;
volatile byte state = LOW;

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(interruptPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(interruptPin), blink, CHANGE);
  // Bla bla bla
  // ......
  // Now we are done with  pin number 2 to act as an interrupt pin.
  detachInterrupt(digitalPinToInterrupt(interruptPin));

}
```

interrupts()

- Re-enables interrupts (after they've been disabled by noInterrupts(). Interrupts allow certain important tasks to happen in the background and are enabled by default. Some functions will not work while interrupts are disabled, and incoming communication may be ignored. Interrupts can slightly disrupt the timing of code, however, and may be disabled for particularly critical sections of code.

noInterrupts()

Disables interrupts (you can re-enable them with interrupts()).



```
#include<Arduino.h>

void setup() {}

void loop() {
  noInterrupts();
  // critical, time-sensitive code here
  interrupts();
  // other code here
}
```

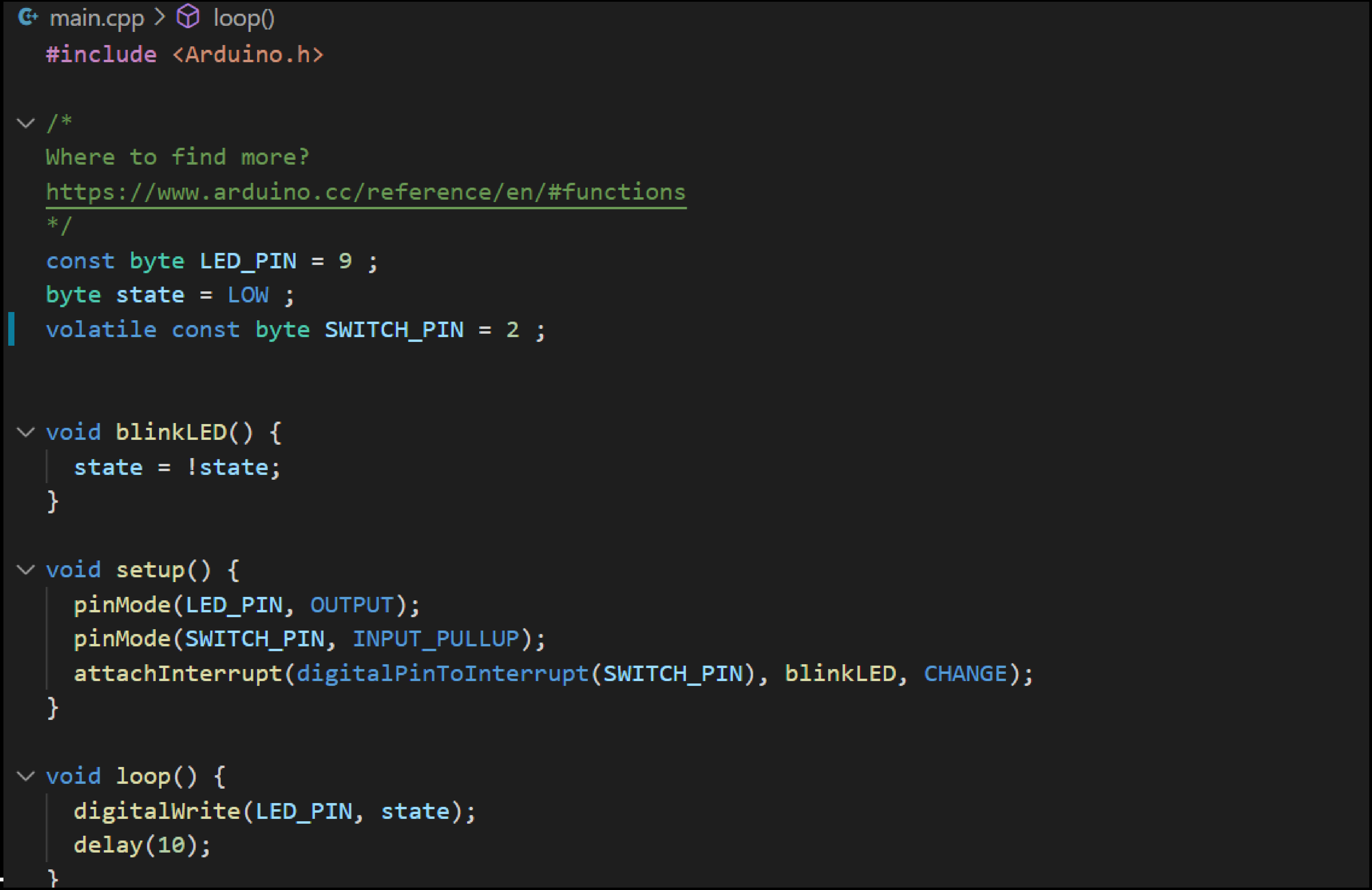
Why using Interrupts?

- Interrupts are useful for making things happen automatically in microcontroller programs and can help solve timing problems. Good tasks for using an interrupt may include reading a rotary encoder, or monitoring user input.
- If you wanted to ensure that a program always caught the pulses from a rotary encoder, so that it never misses a pulse, it would make it very tricky to write a program to do anything else, because the program would need to constantly poll the sensor lines for the encoder, in order to catch pulses when they occurred. Other sensors have a similar interface dynamic too, such as trying to read a sound sensor that is trying to catch a click, or an infrared slot sensor (photo-interrupter) trying to catch a coin drop. In all of these situations, using an interrupt can free the microcontroller to get some other work done while not missing the input.

More about ISR

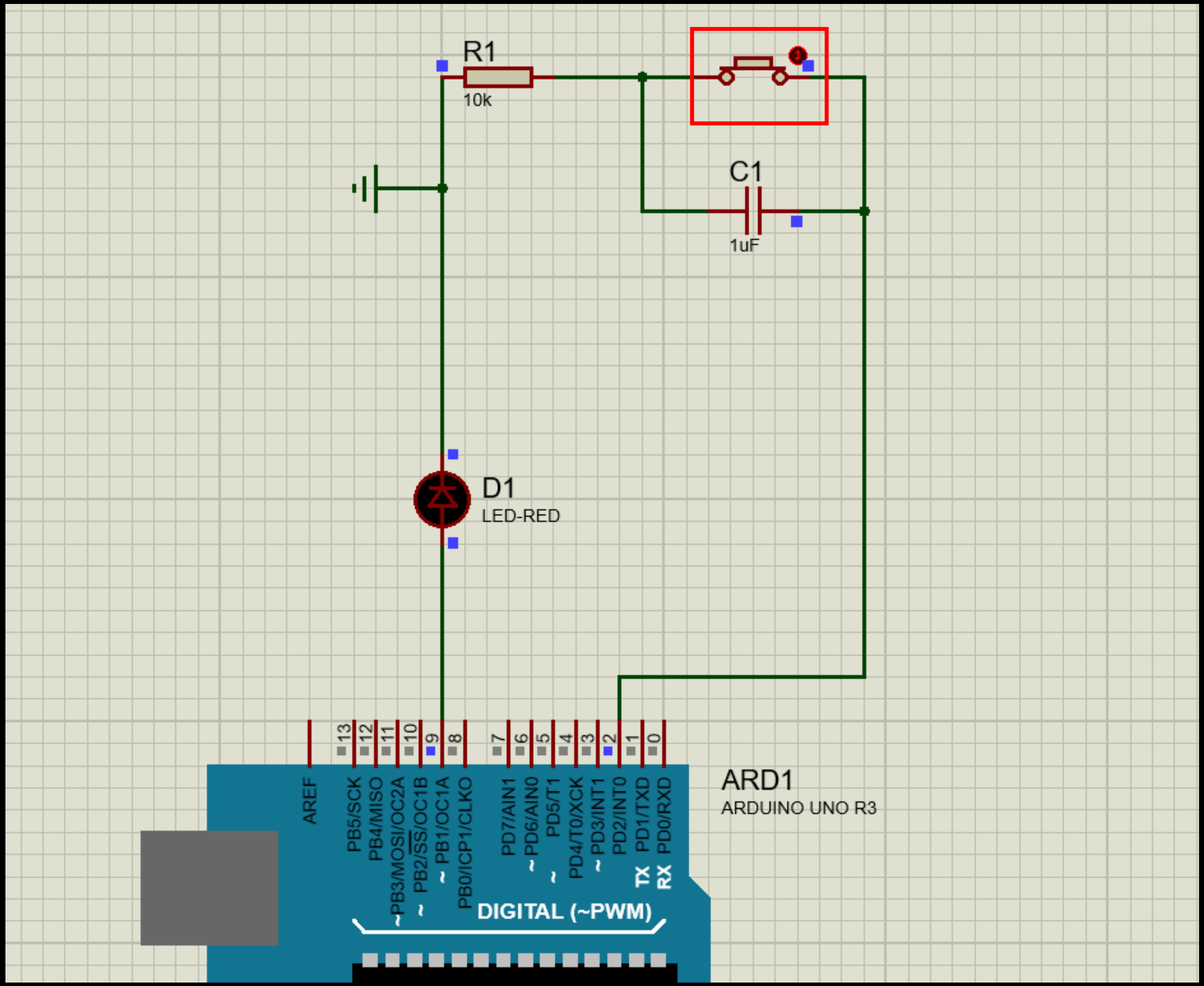
ISRs are special kinds of functions that have some unique limitations most other functions do not have. An ISR cannot have any parameters, and they shouldn’t return anything.
Generally, an ISR should be as short and fast as possible. If your sketch uses multiple ISRs, only one can run at a time, other interrupts will be executed after the current one finishes in an order that depends on the priority they have. millis() relies on interrupts to count, so it will never increment inside an ISR.

Since delay() requires interrupts to work, it will not work if called inside an ISR. micros() works initially but will start behaving erratically after 1-2 ms. delayMicroseconds() does not use any counter, so it will work as normal.
Typically global variables are used to pass data between an ISR and the main program. To make sure variables shared between an ISR and the main program are updated correctly, declare them as volatile.
For more information on interrupts, see Nick Gammon’s notes.

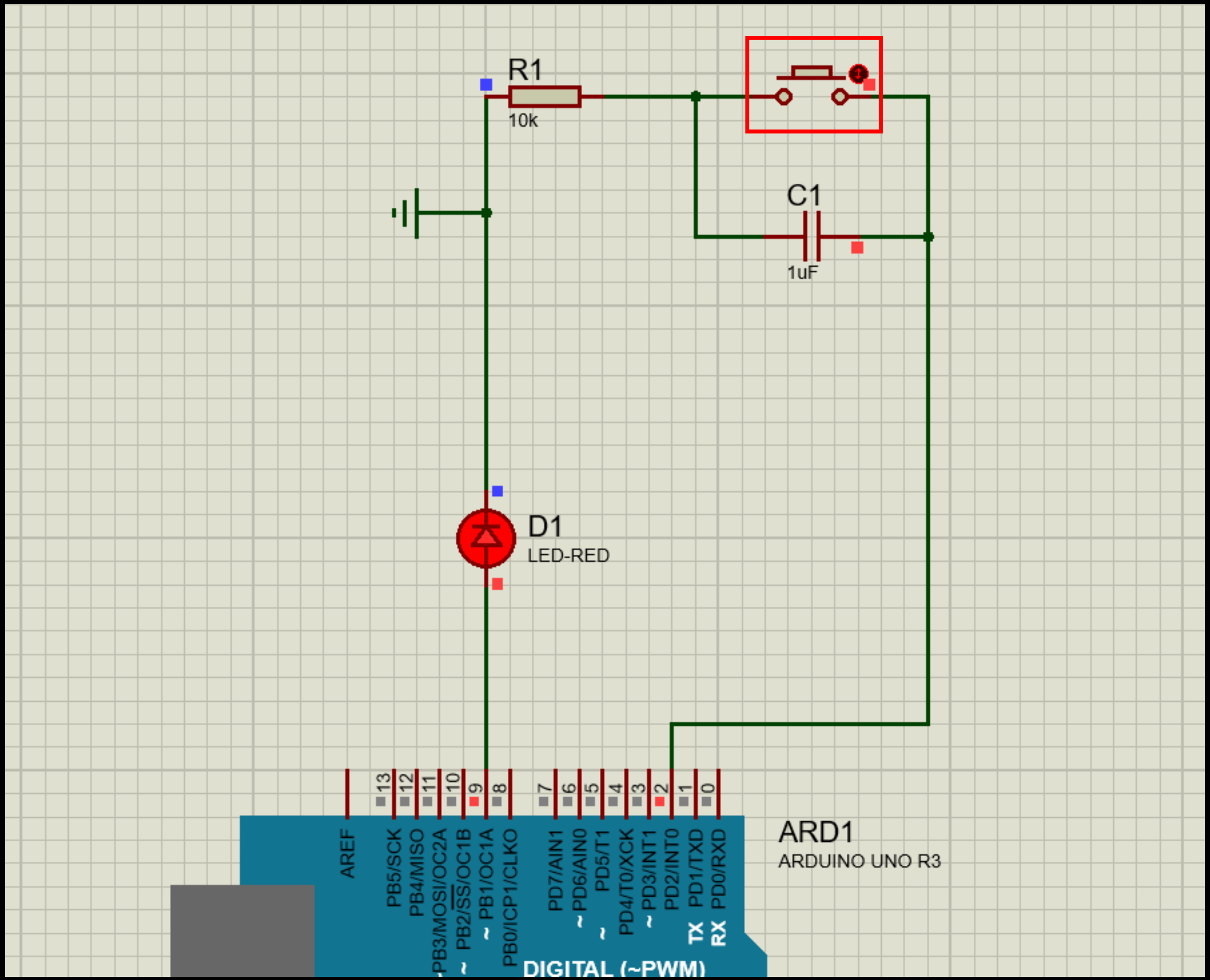


main.cpp > loop()

```cpp
#include <Arduino.h>

/*
Where to find more?
https://www.arduino.cc/reference/en/#functions
*/
const byte LED_PIN = 9 ;
byte state = LOW ;
volatile const byte SWITCH_PIN = 2 ;


void blinkLED() {
  state = !state;
}

void setup() {
  pinMode(LED_PIN, OUTPUT);
  pinMode(SWITCH_PIN, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(SWITCH_PIN), blinkLED, CHANGE);
}

void loop() {
  digitalWrite(LED_PIN, state);
  delay(10);
}
```



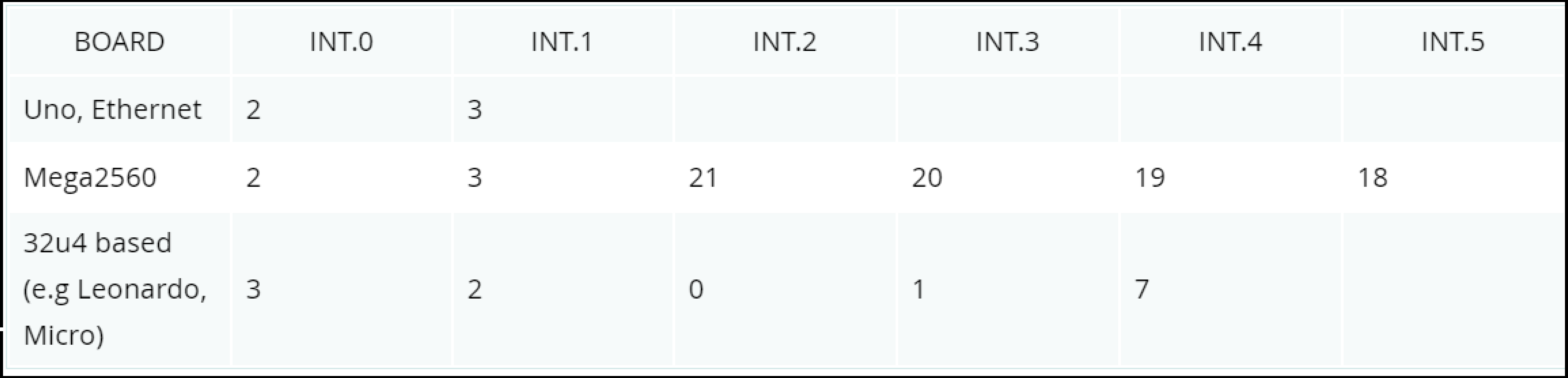
R1
10k
C1
1uF
D1
LED-RED
ARD1
ARDUINO UNO R3
AREF
PB5/SCK
PB4/MISO
PB3/MOSI/OC2A
PB2/SS/OC1B
PB1/OC1A
PB0/ICP1/CLKO
PD7/AIN1
PD6/AIN0
PD5/T1
PD4/T0/XCK
PD3/INT1
PD2/INT0
PD1/TXD
PD0/RXD
TX
RX
DIGITAL (~PWM)



R1
10k
C1
1uF
D1
LED-RED
ARD1
ARDUINO UNO R3
AREF
PB5/SCK
PB4/MISO
PB3/MOSI/OC2A
PB2/SS/OC1B
PB1/OC1A
PB0/ICP1/CLKO
PD7/AIN1
PD6/AIN0
PD5/T1
PD4/T0/XCK
PD3/INT1
PD2/INT0
PD1/TXD
PD0/RXD
TX
RX
DIGITAL (~PWM)

Interrupt Numbers

Normally you should use digitalPinToInterrupt(pin), rather than place an interrupt number directly into your sketch. The specific pins with interrupts and their mapping to interrupt number varies for each type of board. Direct use of interrupt numbers may seem simple, but it can cause compatibility trouble when your sketch runs on a different board.
However, older sketches often have direct interrupt numbers. Often number 0 (for digital pin 2) or number 1 (for digital pin 3) were used. The table below shows the available interrupt pins on various boards.
Note that in the table below, the interrupt numbers refer to the number to be passed to attachInterrupt(). For historical reasons, this numbering does not always correspond directly to the interrupt numbering on the ATmega chip (e.g. int.0 corresponds to INT4 on the ATmega2560 chip).



BOARD	INT.0	INT.1	INT.2	INT.3	INT.4	INT.5
Uno, Ethernet	2	3				
Mega2560	2	3	21	20	19	18
32u4 based (e.g Leonardo, Micro)	3	2	0	1	7	

If you had any questions:

1. Google is your friend! It always gives you updated data.
2. If you did not find your answer on Google, search it on YouTube. There is definitely an Indian fellow answering your questions.
3. After all, you can also ask your graders (:
4. Besides, all the content gathered in this document is **exactly** based on what is written on the Arduino website. (You can click on this link and compare it to the current document.)