Al-Mustaqbal University
Department of Medical Instrumentation Techniques Engineering
Class: four
Subject: Advanced logic design
Lecturer: Dr. Zahraa hashim kareem

Second term- Lecture- 2,3: Arduino Hardware Interrupt+ Receiving Serial Data in Arduino

Arduino Hardware Interrupt

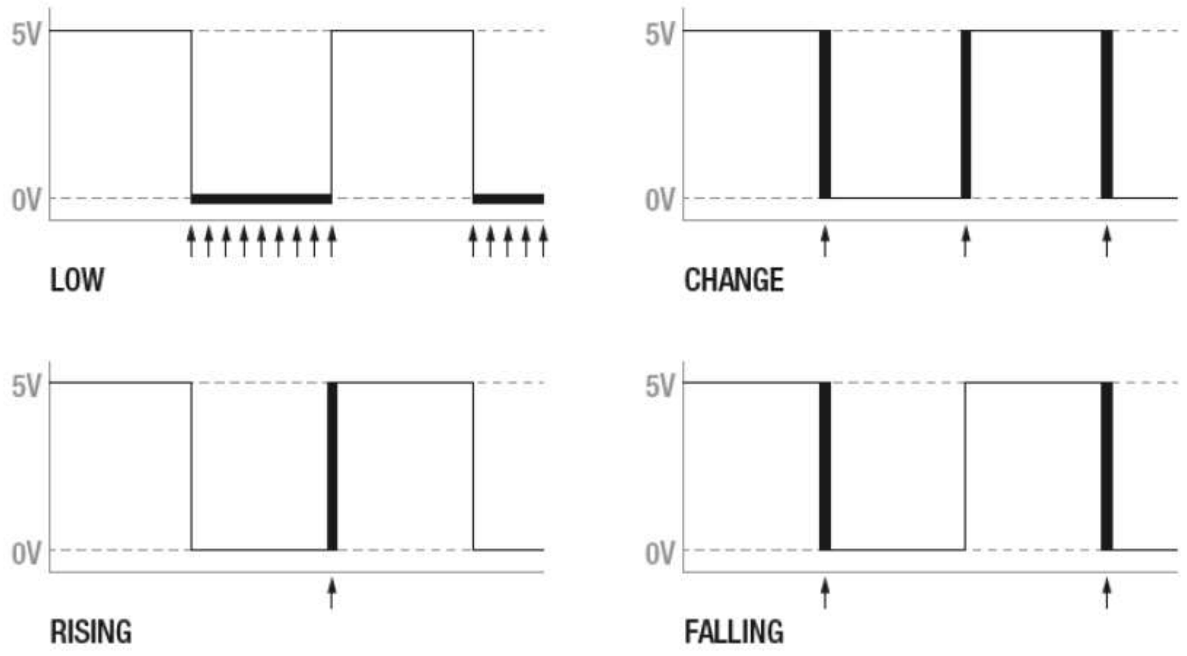
With an idea as to what functions can do for us, we can move on to a different kind of function—one that is driven by specific hardware in the microcontroller.
The reasons for using interrupts are many; maybe we have a lot of code in our **loop()** function and sitting there waiting for a button press would slow things down too much or maybe we might even miss the button press all together.
Or instead we might be using a photo sensor or interrupter that triggers when something gets close and it is important to stop a motor right at that exact time.
These things are possible with a hardware interrupt that can be configured on one of two digital pins to trigger a unique kind of advanced function called an **interrupt service routine (ISR)**. When the interrupt is triggered, the ISR will be called regardless of whatever the program is in the middle of. Like using one of the internal timers, monitoring of the hardware interrupt pin happens in the background and the jump to the ISR can happen within four instructions, so fairly quickly. After the ISR has been executed, program flow returns back to where it left off before the interrupt and, if done correctly, with little effect on the rest of the code. An ISR looks a lot like a regular function but there are a few rules that we need to stick to. The ISR should be kept as short as possible, often only three to four lines of code at the maximum, so as to not overly disrupt the program flow and prevent the interrupt from being triggered again while executing the ISR.
If we have a longer ISR than that, we need to disable the interrupt briefly while the ISR is running. Likewise, timing related functions would not work properly within the ISR because they use the same hardware timer. For example, millis() will not increment and delay() will not work at all. Other than that, an ISR is declared just like a normal function, but in order to use a hardware interrupt we need to first attach it in our code.

Function name: any function

mode: defines when the interrupt should be triggered. Four constants are predefined as valid values:

- **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, and
- **FALLING:** for when the pin goes from high to low.

The Due board allows also: **HIGH** to trigger the interrupt whenever the pin is high. *(Arduino Due, Zero, MKR1000 only).*



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	
Due, Zero, MKR1000, 101	interrupt number = pin number					

Example

const byte ledPin = 13;
const byte interruptPin = 0;
volatile byte state = LOW; // Because **state** is changed in the function that is called when the interrupt happens, it needs to be declared as volatile when it is created. This tells the compiler that it could change at any time; don't optimize the code by assuming it won't have changed, as the interrupt can

happen at any time.

```
void setup() {
pinMode(ledPin, OUTPUT);
pinMode(interruptPin, INPUT_PULLUP);
attachInterrupt(digitalPinToInterrupt(interruptPin), blink, CHANGE);
}
void loop() {
digitalWrite(ledPin, state);
}
void blink() {
state = !state;
}
```

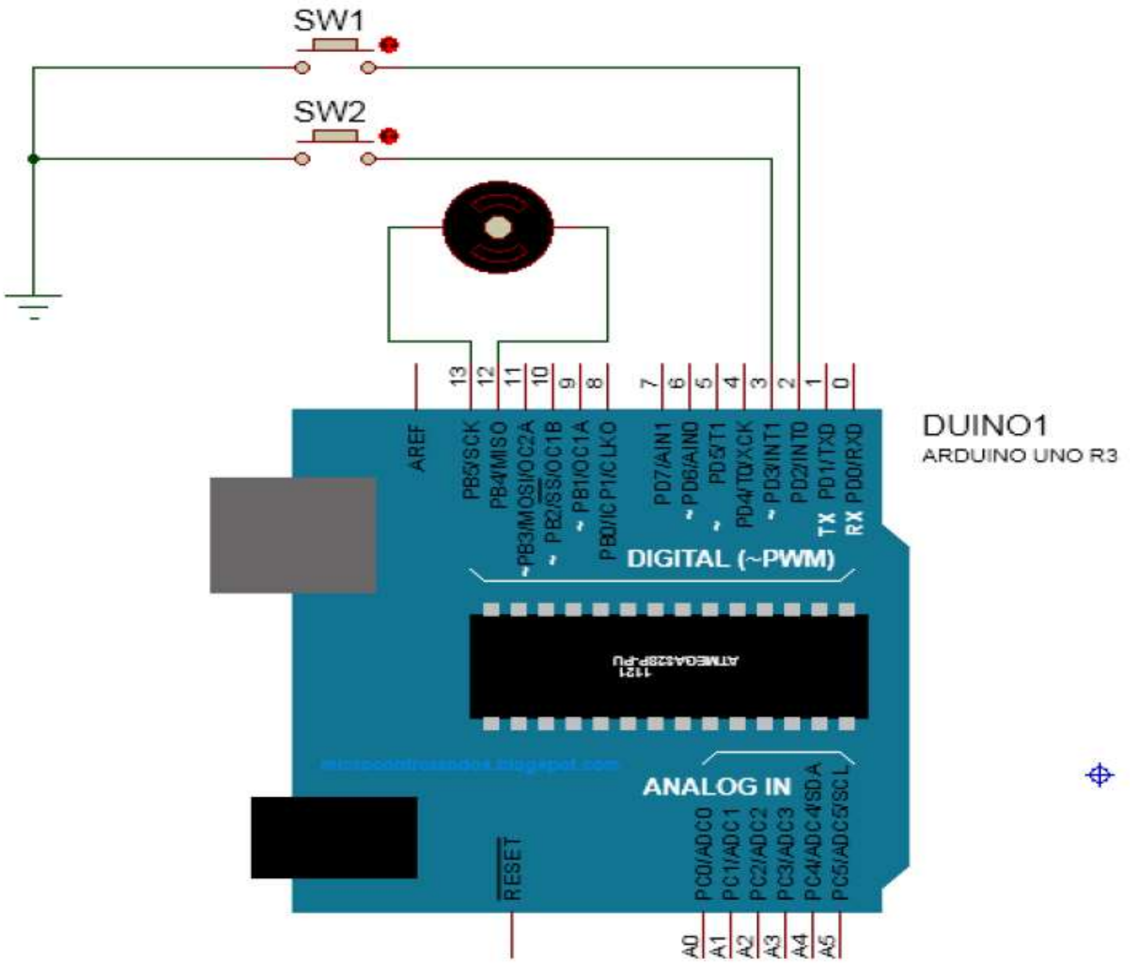
Example (H.W. this example for training)

We want to write a code when SW1 closed, the Motor run with clockwise and when SW2 close, the motor run anticlockwise.

```
const byte ledPin = 13;
const byte ledPin1 = 12;
const byte interruptPin = 0;
const byte interruptPin1 = 1;
void setup() {
pinMode(ledPin, OUTPUT);
pinMode(ledPin1, OUTPUT);
pinMode(interruptPin, HIGH);
pinMode(interruptPin1, HIGH);
}
void loop() {
attachInterrupt(interruptPin, blink,FALLING);
attachInterrupt(interruptPin1, blink1,FALLING);
}

void blink() { // with clockwise
digitalWrite(ledPin, LOW);
digitalWrite(ledPin1, HIGH);
}
void blink1() { // anticlockwise
digitalWrite(ledPin, HIGH);
digitalWrite(ledPin1, LOW);
```

}



detachInterrupt()

Description

Turns off the given interrupt

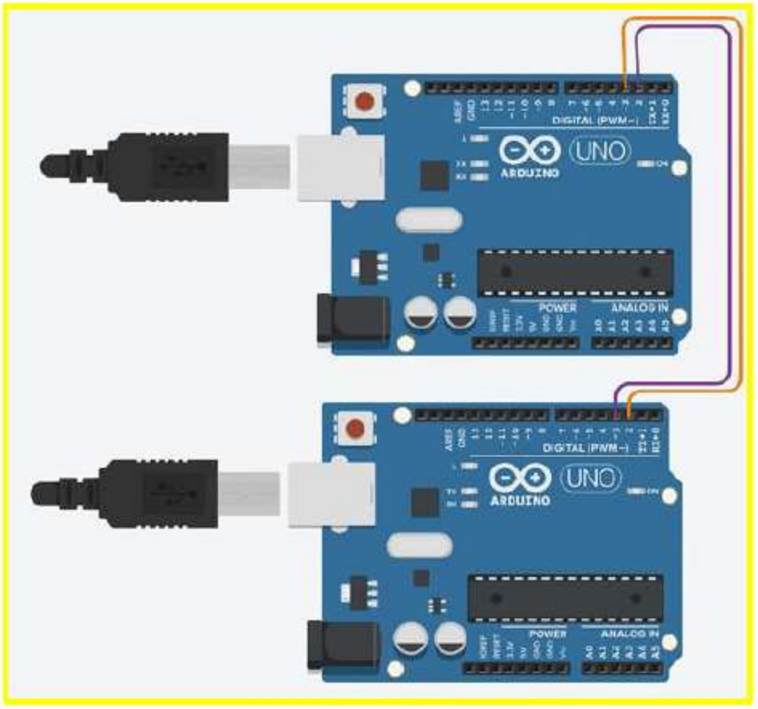
Syntax

detachInterrupt(*interrupt*)
detachInterrupt(digitalPinToInterrupt(pin));
detachInterrupt(*pin*) (*Arduino Due, Zero only*)

Parameters

- *interrupt*: the number of the interrupt to disable.
- *pin*: the pin number of the interrupt to disable (*Arduino Due only*)

Receiving Serial Data in Arduino



Receiving Serial Data in Arduino

Problem

You want to receive data on Arduino from a computer or another serial device; for example, to have Arduino react to commands or data sent from your computer.

Solution

It's easy to receive 8-bit values (chars and bytes), because the Serial functions use 8- bit values. This sketch receives a digit (single characters 0 through 9) and blinks the LED on pin 13 at a rate proportional to the received digit value:

```
/*
* SerialReceive sketch
* Blink the LED at a rate proportional to the received digit value
*/
const int ledPin = 13; // pin the LED is connected to
int blinkRate=0; // blink rate stored in this variable
void setup()
```

```
{
Serial.begin(9600); // Initialize serial port to send and receive at 9600
//baud
pinMode(ledPin, OUTPUT); // set this pin as output
}
void loop()
{
if ( Serial.available()) // Check to see if at least one character is available
{
char channel = Serial.read();
if(channel >= '0' && channel <= '9') // is this an ASCII digit between 0 and
//9
{
blinkRate = (channel - '0'); // ASCII value converted to numeric value
blinkRate = blinkRate * 100; // actual blinkrate is 100 mS times received
//digit
}
}
}


// blink the LED with the on and off times determined by blinkRate
void blink()
{
digitalWrite(ledPin,HIGH);
delay(blinkRate); // delay depends on blinkrate value
digitalWrite(ledPin,LOW);
delay(blinkRate);
}
```

Discussion

Converting the received ASCII characters to numeric values may not be obvious if you are not familiar with the way ASCII represents characters. The following converts the character channel to its numeric value:

blinkRate = (channel - '0'); // ASCII value converted to numeric value . This is done by subtracting 48, because 48 is the ASCII value of the digit 0. For example, if channel is representing the character 1, its ASCII value is 49. The expression 49- '0' is the same as 49-48. This equals 1, which is the numeric value of the character 1. In other words, the expression (channel - '0') is the same as (channel - 48); this converts the ASCII value of the variable channel to a numeric value.
84