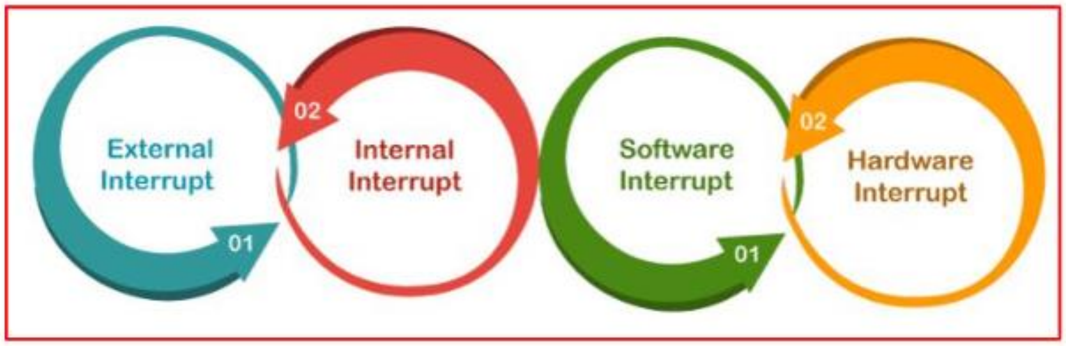
Al-Mustaqbal University
Department of Medical Instrumentation Techniques Engineering
Class: four
Subject: Advanced logic design
Lecturer: Dr. Zahraa hashim kareem
Lecture- 2: INTERRUPT ON ARDUINO

Arduino Hardware Interrupt

Students of fourth class



1. Introduction

Interrupts are a fundamental concept in embedded systems and microcontroller programming. They allow your Arduino to react immediately to specific events—such as a button press or sensor signal—without having to constantly check for them in your main program loop. This makes your code more efficient and responsive.

1. What Are Interrupts?

Definition:
An interrupt is a signal that temporarily halts the normal flow of execution in a program, allowing the microcontroller to execute a special routine called an Interrupt Service Routine (ISR) before returning to the main code. Interrupts are particularly useful when you need to respond quickly to time-sensitive events.

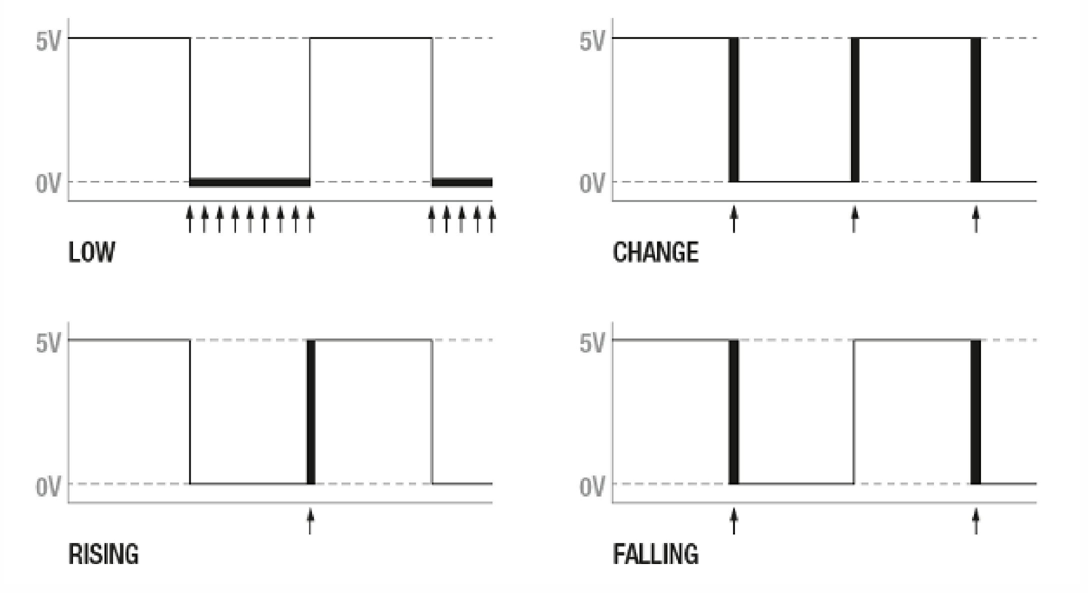
Key Benefits:

- **Immediate Response:** Interrupts allow the Arduino to respond instantly to external events.
- **Efficient CPU Usage:** Instead of continuously polling for a change, the processor can perform other tasks or enter a low-power mode until an interrupt occurs.
- **Improved Code Structure:** By isolating event-driven code within ISRs, you can keep your main program cleaner and more efficient.

2. Types of Interrupts in Arduino

a. External Interrupts

- **Definition:** External interrupts are triggered by changes on specific pins (e.g., pins 2 and 3 on an Arduino Uno).
- **Modes:** They can be configured to trigger on:
  - **LOW:** When the pin is at a low voltage level.
  - **CHANGE:** When the pin's state changes (from LOW to HIGH or vice versa).
  - **RISING:** When the pin transitions from LOW to HIGH.
  - **FALLING:** When the pin transitions from HIGH to LOW.



b. Pin Change Interrupts

- **Definition:** Unlike external interrupts, pin change interrupts can be triggered by a state change on any digital pin, though they are less specific.
- **Usage:** They are particularly useful when you need to monitor multiple pins simultaneously.

c. Timer Interrupts

- **Definition:** Timer interrupts are generated by the microcontroller's internal timers. They are used for tasks that require precise timing or periodic actions.

- **Usage:** Common in applications like generating PWM signals or scheduling regular tasks without blocking the main loop.

3. Implementing Interrupts in Arduino

Using attachInterrupt()

The Arduino function attachInterrupt() binds an ISR to an interrupt-capable pin. Its syntax is as follows:

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

- **pin:** The Arduino digital pin number that triggers the interrupt.
- **ISR:** The function to be executed when the interrupt occurs. This function should not accept parameters nor return any value.
- **mode:** The condition that triggers the interrupt (e.g., LOW, CHANGE, RISING, FALLING).

Example Code: External Interrupt on a Button Press

Below is a simple example where a button press on pin 2 triggers an interrupt that increments a counter.

```
volatile int counter = 0;  // Variable modified by ISR must be declared as volatile
void setup() {
  Serial.begin(9600);
  pinMode(2, INPUT_PULLUP);  // Set pin 2 as input with an internal pull-up resistor
  attachInterrupt(digitalPinToInterrupt(2), incrementCounter, FALLING);
}
void loop() {
  Serial.println(counter);
  delay(500);  // Delay for readability
}
void incrementCounter() {
  counter++;  // ISR: increment the counter each time a falling edge is detected on pin 2
}
```

4. Best Practices for Using Interrupts

- **Keep ISRs Short:** Avoid lengthy computations, delays, or serial communications inside an ISR. The ISR should execute quickly and return control to the main program.
- **Minimal Resource Usage:** Use minimal resources within an ISR to avoid interfering with the main loop or other interrupts.
- **Avoid Global Variables Without volatile:** Always declare variables shared between your ISR and your main code as volatile to ensure proper updates.
- **Debounce Mechanism:** When using mechanical switches, incorporate hardware or software debouncing to prevent multiple rapid triggers due to contact bounce.

5. Common Pitfalls

- **Overusing Interrupts:** Too many or poorly designed interrupts can lead to complex, hard-to-debug issues.
- **Interrupt Latency:** Long ISRs may block other interrupts from being serviced, leading to potential missed events.
- **Concurrency Issues:** Improper handling of shared resources can lead to race conditions. Consider using flags and careful variable management.

6. Applications of Interrupts in Arduino Projects

- **Real-Time Monitoring:** Quick response to sensor inputs, such as motion detection or environmental monitoring.
- **Interactive Devices:** Implementing responsive interfaces for buttons, encoders, and other input devices.
- **Power Management:** Utilizing sleep modes combined with interrupts to wake up the microcontroller on an event, thereby saving power.
- **Time-Critical Tasks:** Managing precise timing events, like generating periodic signals or triggering actions at specific intervals.

Conclusion

Interrupts are a powerful tool in Arduino programming that enable your projects to react quickly and efficiently to external events. By understanding and properly implementing interrupts, you can design systems that are both responsive and energy-efficient. Adhering to best practices—such as keeping ISRs short, using the volatile keyword, and ensuring proper debouncing—will help you avoid common pitfalls and create robust, high-performance Arduino applications.