

Interrupts

Embedded Systems

Group II

Group Members

Raymond NTUMWA (S24B23/101)

Faith AINOMUGISHA (S24B23/087)

Isaac RUBANGANGEYO (S24B23/061)

Nathaniel MUGENYI (M24B23/027)

Abubaker SSENDAGIRE (S24B23/012)

What is an Interrupt?

An interrupt is a signal that makes the microcontroller pause its current task and quickly execute a special function called an Interrupt Service Routine (ISR), before returning to what it was doing.

Analogy: Someone taps your shoulder while reading → you stop → respond → continue reading.

Why Interrupts Are Important

Embedded systems handle real-time events:

- Button presses
- Timer overflows
- Sensor triggers
- Communication (UART, SPI, I2C)
- Motor feedback

Without interrupts → Polling wastes CPU time.

Interrupts make systems:

- Fast
- Efficient
- Real-time
- Low-power

Types of Interrupts

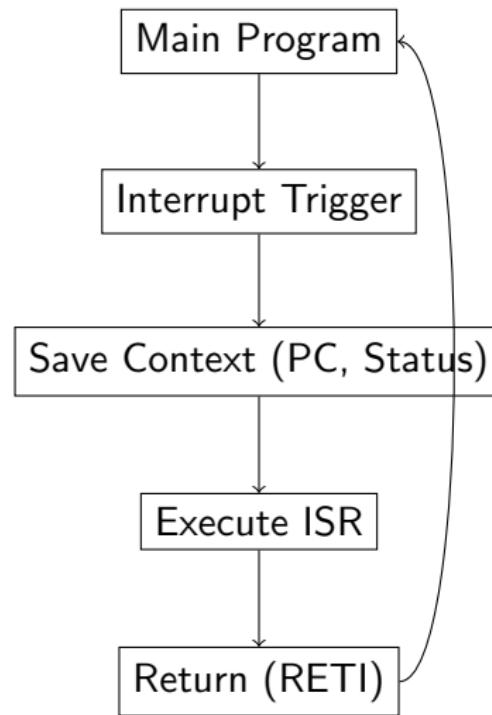
A. Hardware Interrupts

- Button press
- Timer overflow
- ADC conversion complete
- UART receive complete

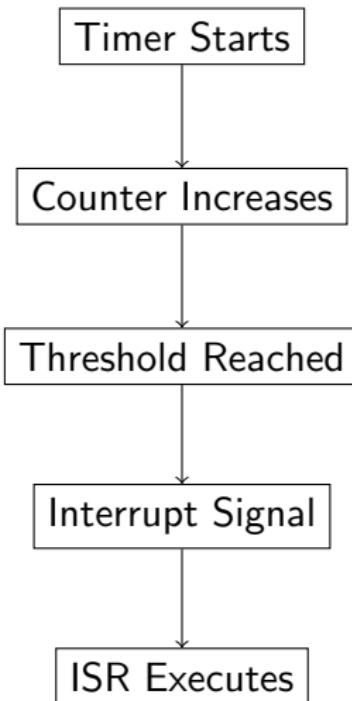
B. Software Interrupts

- System calls
- Context switching (RTOS)
- Firmware-triggered events

Interrupt Flow Diagram



How a Timer Interrupt Works



Timer interrupts ensure precise timing for real-time control.

Inside the Timer Peripheral

A timer contains:

- Counter register
- Compare register

When Counter = Compare value → Interrupt triggered.

Features:

- High precision
- Low overhead
- Independent of CPU loop

Insert Timer Block Diagram Here

The Vector Table

The Vector Table stores addresses of all ISRs.

- Located at start of memory
- Each interrupt has an IRQ number
- CPU jumps to ISR address automatically

Address Formula:

$$\text{Address} = \text{Base} + (\text{IRQ} \times \text{Vector Size})$$

Insert Vector Table Memory Layout
Diagram

Critical Performance Concepts

Interrupt Latency

- Time from signal → first ISR instruction

Masking

- Disable specific interrupts

NVIC (ARM)

- Manages priority
- Allows preemption

Tail-Chaining

- Switch between ISRs efficiently

Register-Level Code Example (AVR)

```
int main(void)  DDRB |= (1 << DDB5); // LED as output
EICRA |= (1 << ISC01); // Falling edge EICRA = (1 << ISC00);
EIMSK |= (1 << INT0); // Enable INT0 sei(); // Global enable
while(1)
ISR(INT0_vect) PORTB=(1 << PORTB5);
```

Real Uses of Timer Interrupts

- LED blinking (non-blocking)
- Motor control (PWM)
- Communication timeouts
- Real-time clocks
- OS task scheduling

Smart System Example

Imagine a Smart Home System:

- Motion sensor → turns lights ON
- Touch panel → user input interrupt
- Timer → logs data every minute

All handled efficiently through interrupts.

Insert Smart Home Illustration Here

Conclusion & Key Takeaways

- ① Interrupts are essential for real-time systems.
- ② Hardware and Software interrupts manage different events.
- ③ Timer interrupts enable precise scheduling.
- ④ Interrupts improve efficiency and power usage.

Mastering interrupts is key to building reliable embedded systems.