# Week 7 - Interrupt

## Polling

- Repeatedly checking the status of a device
- Done inside the main loop of the program
- **Disadvantage:**
  - Performance can suffer if there are many thing to perform

## Interrupt

- The peripheral device signals the CPU.
- Main loop is free of polling.
- CPU can sleep, and only jump to a Interrupt Service Routine (ISR) when there is an interrupt signal.
  - CPU stops current task, does ISR, returns to previous task.

### Context Management

- Context: state of the CPU and registers, program counter etc.

- Context should be saved when an interrupt is triggered so that the task can again be resumed later on.
- Complex.

### Interrupt Vector Table

Holds list of interrupts along with the address of the interrupt handlers.

ATmega328p

| Pri. | Address | Interrupt Source | ISR C Function Name | Description |
|---|---|---|---|---|
| 1 | 0x0000 | RESET | | System reset (power-on) |
| 2 | 0x0002 | INT0 | INT0_vect | External Interrupt Request 0 |
| 3 | 0x0004 | INT1 | INT1_vect | External Interrupt Request 1 |
| 4 | 0x0006 | PCINT0 | PCINT0_vect | Pin Change Interrupt Request 0 |
| 5 | 0x0008 | PCINT1 | PCINT1_vect | Pin Change Interrupt Request 1 |
| 6 | 0x000A | PCINT2 | PCINT2_vect | Pin Change Interrupt Request 2 |
| 7 | 0x000C | WDT | WDT_vect | Watchdog Time-out Interrupt |
| 8 | 0x000E | TIMER2 COMPA | TIMER2_COMPA_vect | Timer/Counter2 Compare Match A |
| 9 | 0x0010 | TIMER2 COMPB | TIMER2_COMPB_vect | Timer/Counter2 Compare Match B |
| 10 | 0x0012 | TIMER2 OVF | TIMER2_OVF_vect | Timer/Counter2 Overflow |
| 11 | 0x0014 | TIMER1 CAPT | TIMER1_CAPT_vect | Timer/Counter1 Capture Event |
| 12 | 0x0016 | TIMER1 COMPA | TIMER1_COMPA_vect | Timer/Counter1 Compare Match A |
| 13 | 0x0018 | TIMER1 COMPB | TIMER1_COMPB_vect | Timer/Counter1 Compare Match B |

**Examples**

```
ISR(INT1_vect)
{
    //Code here
}
```

```
ISR(ADC_vect)
{
    //Code here
}
```

- The MSB of the SREG (status register) is used to enable or disable global interrupts.

## External Interrupts

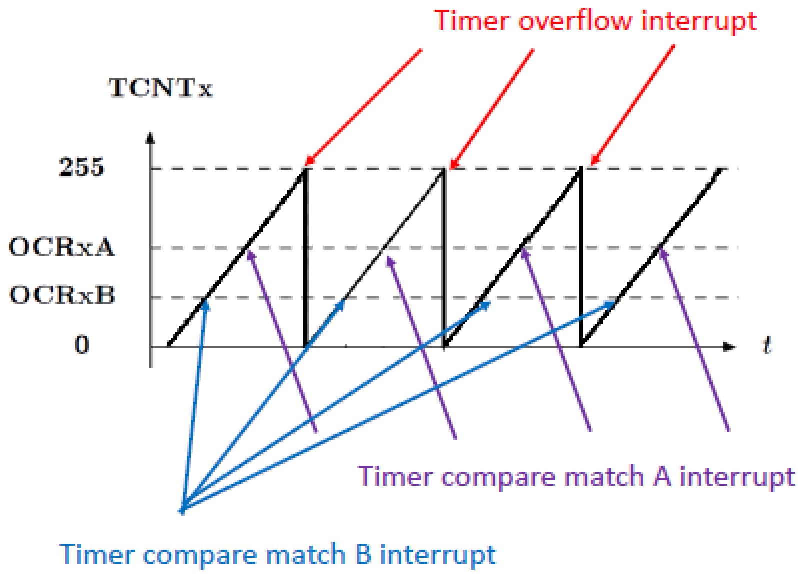
- INT0 and INT1
- 4 Modes:
  - Low level
  - any transition
  - high to low
  - low to high
- Flags are used to indicate if a specific interrupt has occured.

## Pin Change Interrupt

- Interrupt for a group of pins. If there is a change in logic level for any of the pins in the group, the interrupt can be activated.
- For example, in a security system, multiple sensors can be connected to the same group of pins so that if there is any intruded detected, the interrupt can be activated.

## Timer Interrupts

- Overflow or Matching to a OCR interrupt can be generated.




# Raspberry Pi 3

- The Linux kernel handles the low-level interrupts and provides a way for the user-program to interact with the interrupt.
- Unlike ATmega, the user doesn’t directly interact with the registers. The kernel handles the low level work. The user just programs using libraries.

# Exercise - Pushbutton Interrupt

```
#!/usr/bin/env python3

# Import the RPi.GPIO library to control the GPIO pins
import RPi.GPIO as GPIO
# Import the time library to allow the main loop to sleep
import time

# --- Configuration ---

# Define the GPIO pin number that the LED is connected to.
```

```
# We are using the BCM (Broadcom) numbering scheme, where 18 refers to GPIC
# On the physical header, this is pin 12.
LED_PIN = 18

# Define the GPIO pin number that the Button is connected to.
# We are using the BCM numbering scheme, where 23 refers to GPIO23.
# On the physical header, this is pin 16.
BUTTON_PIN = 23

# Define a variable to help with debouncing the button press.
# Buttons aren't perfect switches; they can 'bounce' when pressed or released,
# causing multiple rapid high/low transitions.
# bouncetime is specified in milliseconds (ms). 300ms is a reasonable value.
BUTTON_BOUNCETIME = 300 # milliseconds

# --- Callback Function (Interrupt Service Routine equivalent in user space) ---

# This function will be called by the RPi.GPIO library when the event
# (button press) is detected on the specified pin.
# It takes one argument, 'channel', which is the pin number that triggered the eve
def button_callback(channel):
    """
    Callback function executed when the button is pressed (falling edge detected)
    Toggles the state of the LED.
    """
    print(f"--- Button pressed on channel {channel}! ---") # Verbose print stateme

    # Read the current state of the LED pin.
    current_led_state = GPIO.input(LED_PIN)

    # Determine the new state for the LED.
    # If the LED is currently HIGH (ON), set it to LOW (OFF).
    # If the LED is currently LOW (OFF), set it to HIGH (ON).
    new_led_state = GPIO.LOW if current_led_state == GPIO.HIGH else GPIO.HIGH

    # Set the LED pin to the new state.
```

```
    GPIO.output(LED_PIN, new_led_state)

    # Add a print statement to confirm the LED state change.
    print(f"LED state toggled to { 'ON' if new_led_state == GPIO.HIGH else 'OFF' }.


# --- Main Program Execution ---

def main():
    """
    Main function to set up GPIO, register interrupt event, and run the main loop.
    """
    print("Setting up GPIO for button and LED...")

    # Set the GPIO mode.
    # GPIO.BCM: Refers to the Broadcom SOC channel names (GPIO numbers). Th
    # GPIO.BOARD: Refers to the physical pin numbers on the header (less recomm
    GPIO.setmode(GPIO.BCM)

    # --- Configure GPIO Pins ---

    # Set up the LED pin as an output.
    # We can specify an initial value (optional), here we ensure it's OFF initially.
    GPIO.setup(LED_PIN, GPIO.OUT, initial=GPIO.LOW)
    print(f"GPIO {LED_PIN} (LED) configured as output, initially LOW.")

    # Set up the Button pin as an input.
    # We configure an internal pull-up resistor because the button is connected to
    # This ensures the pin is HIGH when the button is not pressed.
    GPIO.setup(BUTTON_PIN, GPIO.IN, pull_up_down=GPIO.PUD_UP)
    print(f"GPIO {BUTTON_PIN} (Button) configured as input with internal pull-up.'

    # --- Register the Event Detection ---

    # This is the core of the "interrupt-based" approach in user space.
    # We tell the library to call our 'button_callback' function
```

```
# whenever a falling edge is detected on the BUTTON_PIN.
# GPIO.FALLING: Detects transition from HIGH to LOW (happens when button i
# GPIO.RISING: Detects transition from LOW to HIGH.
# GPIO.BOTH: Detects any change (rising or falling edge).
# callback=button_callback: Specifies the function to call when the event occu
# bouncetime=BUTTON_BOUNCETIME: Ignores further transitions for this dur
#                                helping to prevent multiple triggers from a single button pr
GPIO.add_event_detect(
    BUTTON_PIN,
    GPIO.FALLING,
    callback=button_callback,
    bouncetime=BUTTON_BOUNCETIME
)
print(f"Event detection added on GPIO {BUTTON_PIN} for FALLING edge with 
print("Ready! Press the button to toggle the LED. Press Ctrl+C to exit.")

# --- Main Program Loop ---

# The main loop simply keeps the script running.
# The actual work of responding to the button press is handled asynchronousl
# by the RPi.GPIO library calling the 'button_callback' function.
# We use a try/except block to catch a KeyboardInterrupt (Ctrl+C) to allow gra
try:
    # In a real application, this loop could be performing other tasks.
    # For this example, we just make it sleep to prevent it from
    # consuming excessive CPU while waiting for button presses.
    while True:
        # Sleep for a short duration. This loop is not doing any polling itself.
        time.sleep(1)
        # print("Main loop is running (doing other stuff or sleeping)...") # Optional

except KeyboardInterrupt:
    # This block is executed if the user presses Ctrl+C.
    print("\nCtrl+C detected. Exiting gracefully.")

finally:
```

```
        # This block is executed when the try or except block finishes.
        # It's crucial to clean up the GPIO settings to release the pins.
        # This prevents issues with future scripts using the same pins.
        GPIO.cleanup()
        print("GPIO cleaned up. Goodbye!")

# Check if the script is being run directly
if __name__ == "__main__":
    # Call the main function to start the program
    main()
```