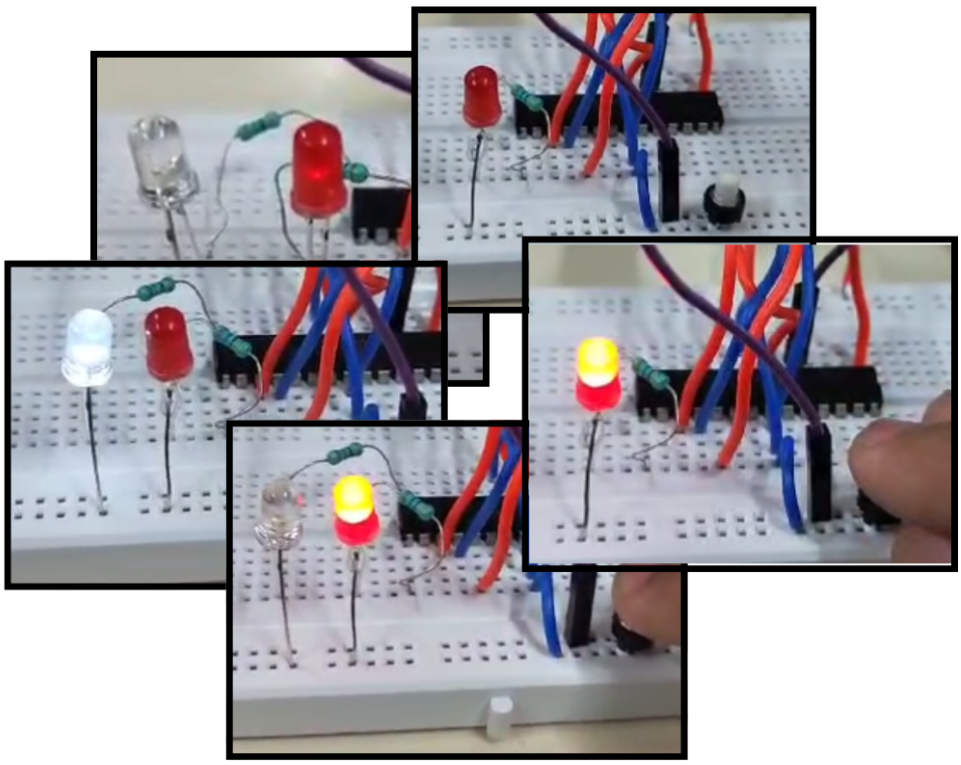

EE2016: Assignment-6

By Siva(EE23B151),
Bhavesh(EE23B017)
and Naveen(EE23B113)

Group 28

Date: 27/9/2024

Indian Institute of Technology Madras

Contents

1 Introduction

In this experiment, we will implement **interrupt-driven** control for LEDs using both C and AVR assembly code. The goal is to utilize hardware interrupts to manage the state of two LEDs—one white and one red—connected through appropriate resistors on a breadboard. A press-button switch will trigger an interrupt, allowing the LEDs to either turn ON or blink, depending on the **interrupt service routine** (ISR) defined in the code.

1.1 Interrupt Service Routine (ISR)

Interrupts are signals that cause the microcontroller to stop its current execution and immediately jump to a specific block of code called the Interrupt Service Routine (ISR).

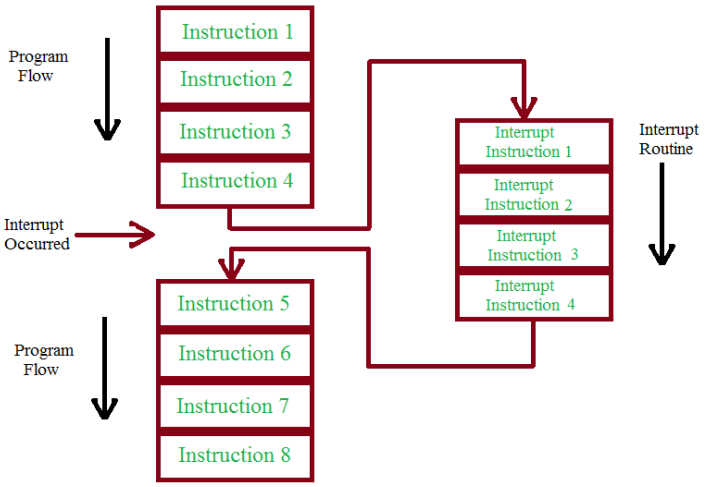

Figure 1: Interrupt workflow

Unlike function calls, interrupts are triggered by hardware or software events rather than **explicitly invoked** by the code.

1.2 Syntax of Interrupt in C

In C, interrupts are handled through special keywords and attributes that indicate the code snippet belongs to an ISR. Here's a basic example using AVR libraries:

```
#include <avr/interrupt.h>
#include <avr/io.h>

ISR(INT0_vect) {
    // Code to execute when interrupt occurs (e.g., toggle LED)
}
```

In this syntax:

- `ISR(INT0_vect)` is the ISR associated with the external interrupt `INT0` (which is `PD2` in Atmega8).
- The `INT0_vect` is the **interrupt vector**[1] for the `INT0` interrupt line.

[1] It is a variable which stores the address of the interrupt pin. It is named like that to indicate that a vectorised interrupt system is used. See link.

- Inside the ISR, you can define what actions need to be taken when the interrupt is triggered.

Syntax of Interrupt in AVR Assembly

In AVR assembly, you typically define an ISR by jumping to the interrupt vector table:

```
.org 0x0000       ; Reset vector
rjmp RESET

.org INT0_vect    ; Interrupt vector for INT0
rjmp ISR_INT0     ; Jump to ISR

ISR_INT0:         ; Interrupt service routine
    ; Code to handle the interrupt
    reti           ; Return from interrupt
```

The `reti` instruction **marks the end** of the ISR and restores the microcontroller's previous state, allowing the program to resume where it left off.

1.3 Interrupt vs. Function Call

- **Triggering:** Interrupts are triggered by hardware events (e.g., a button press) or software interrupts (e.g., division by zero). Function calls, on the other hand, are explicitly invoked within the program's flow by the user.
- **Control Flow:** In a function call, control is transferred sequentially, and once the function finishes, it returns to the calling location. With interrupts, the program can be interrupted at any point, and control is transferred to the ISR. Once the ISR completes (using the `reti` instruction in AVR), the program resumes from where it was interrupted.
- **Response Time:** Interrupts provide real-time response since they don't wait for the program to reach a specific point; they interrupt the flow as soon as the triggering condition occurs. Function calls only execute when the program reaches them.

1.4 Software events which trigger Interrupts

These interrupts are generated by the program itself or internal events rather than external physical actions (like pressing a button), which are referred to as hardware interrupts. These include:

- Directly invoked software interrupts via specific instructions (like `SWI`).
- Internal events like **timer overflows**[2], **watchdog timers**[3], or **system calls**[4].
- Exceptions or faults like **division by zero** or **memory access violations**.

[2]Occurs when a timer or counter in a microcontroller exceeds its maximum count and resets to zero.

[3]It is a hardware timer that resets the microcontroller if the main program fails to reset it within a specified time frame. If the program enters an infinite loop or crashes (and doesn't reset the watchdog), the timer will expire, causing the microcontroller to reset.

[4]They are functions provided by an operating system that allow a program to request services from the kernel.

2 Tasks

2.1 Task 1: C code to implement interrupt which Turns OFF White LED and Turns ON Red LED

Connections

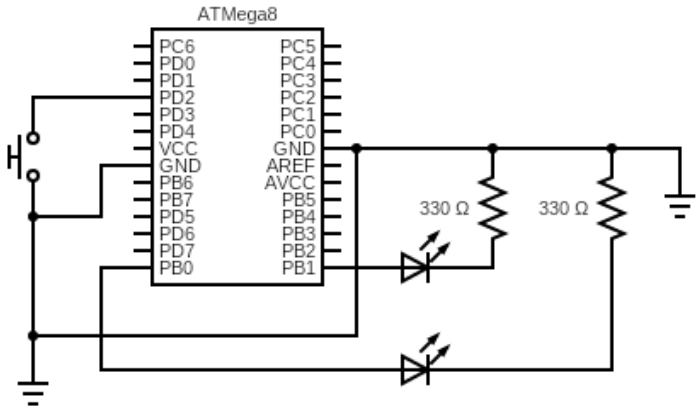

Figure 2: White LED at PB0, Red LED at PB1

Code explanation

In this program, a white LED connected to `PB0` is initially turned on, while a red LED connected to `PB1` starts blinking only upon a key press. The external interrupt is configured on the `INT0` pin (`PD2`, with an internal **pull-up resistor**[5]), which is connected to a button. By default, when the button is not pressed, the input at `PD2` remains logic **HIGH**. However, pressing the button triggers a **falling edge**, causing the interrupt service routine (ISR) to execute. This ISR transfers control to the red LED, making it blink, while the white LED remains off during the interrupt.

The use of `sei()`, `cli()`, `GICR` and `SREG` are used to activate and deactivate interrupts as per requirement. All of these is explained in the comments of the code.

C code, Hex file and Output

You can find the C code using the following link: **C Code**.
Watch the video of the output here: **Watch Video**.

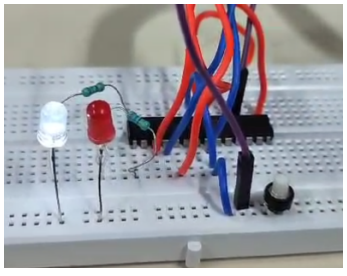

Figure 3: Button not pressed

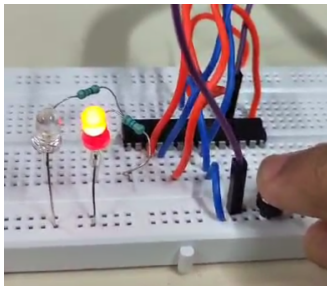

Figure 4: Button pressed

[5]They are used to prevent **floating** pins. Here, we are using it to make the pin be at **HIGH** state if kept open.

2.2 Task 2: AVR assembly code for blinking a Red LED while it recieves an interrupt

Connections

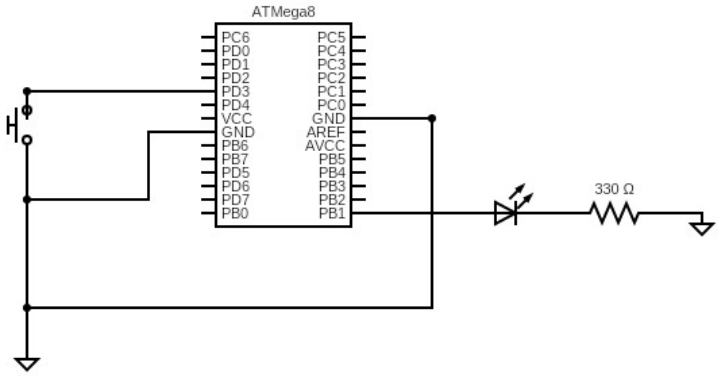

Figure 5: Red LED at PB1

Code explanation

This AVR assembly code is designed to blink a red LED connected to pin `PB1` 10 times whenever an external interrupt (`INT1`/`PD3`) is triggered. The program starts by setting up the stack pointer and configuring `PB1` as an **output** pin and `PD3` as an input with a **pull-up resistor** to detect a button press. The interrupt is configured to trigger on a **falling edge** of the signal on `PD3`.

When the button is pressed, `INT1` triggers the `int1_ISR`, which disables further interrupts and saves the current status register. Inside the ISR, the LED blinks 10 times, with delays between turning the LED ON and OFF. After the blinking sequence, the original status register is restored, interrupts are re-enabled, and the program returns to the main loop, which waits for another interrupt indefinitely. The `main` loop itself does nothing but hold the program in an infinite loop, while the ISR handles the LED blinking.

AVR code, Hex file and Output

You can find the C code in the `.asm` file: **.asm file**.
Watch the video of the output here: **Watch Video**.

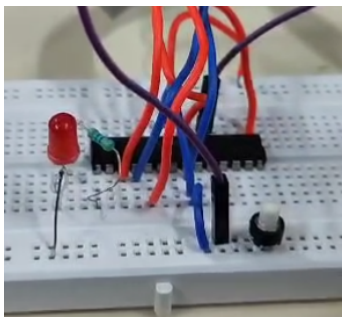

Figure 6: Button not pressed

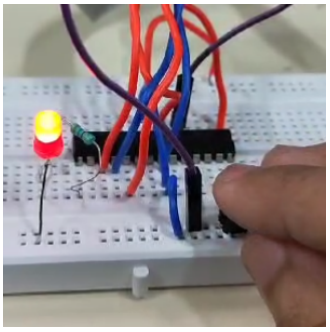

Figure 7: Button pressed, blinks 10 times

2.3 Task 3: Modification of Task 2 to include a White LED which glows after interrupt statement ends

Connections

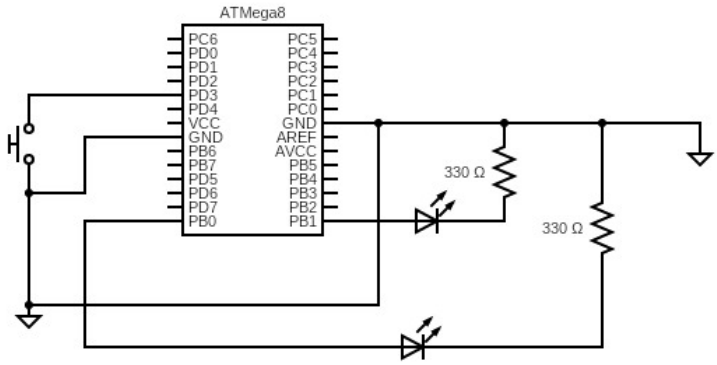

Figure 8: White LED at PB0, Red LED at PB1

Code explanation

This AVR assembly code is almost similar to the previous one, with a key difference that the red LED connected to pin `PB1` is made to blink 10 times only upon external interrupt and an additional white LED connected to pin `PB0` stays OFF upon external interrupt and ON when there is no interrupt. The addition in previous code is explained here.

Initially, the white LED is turned ON. During the interrupt, the red LED blinks for 10 times, during which the white LED stays OFF. During the ISR, after blinking of the red LED, the white LED is turned back on. The original status register is retrieved that turns ON the white LED back, interrupts re-enabled, and goes back to the main loop waiting for an another interrupt during which the red LED stays OFF.

AVR code, Hex file and Output

You can find the C code in the `.asm` file: **.asm file**.
Watch the video of the output here: **Watch Video**.

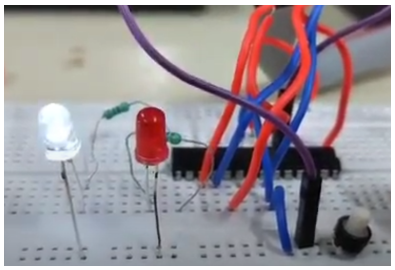

Figure 9: Button not pressed

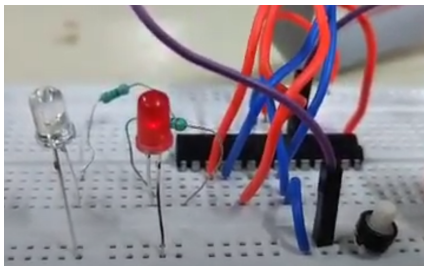

Figure 10: Button pressed, blinks 10 times

The red LED does blink, but with dim light. We think it is due to loose connections. Or it could be due to weak power supply from USBASP cable.