Technical report

Assignment 5: Interrupts

Course title: Embedded Systems Semester 2
Teacher name: Heumen, Hans J.B.H.M. van
Date: 14.06.2024
Authors/Students: Simeon Stoilov, Veselin Atanasov

Abstract:

This technical report presents the development and implementation of real-time I/O handling mechanisms using interrupts and timers on the Arduino platform. The assignment is divided into three parts focused on low-level programming techniques to manage hardware resources effectively. In the first part we had to make the foundation for the next steps, we had to use register manipulation to handle button presses and turn on LEDs accordingly. Part B extends the application by incorporating interrupt service routines (IRS) to handle the button presses, enhancing the system's responsiveness and efficiency. In the last part, a binary counter had to be implemented displayed by the LEDs, this was accomplished by using system timers, interrupts, and a counter. Each section of the report details the design, implementation, and performance analysis of the solutions, highlighting the benefits and drawbacks of using interrupts and timers in embedded systems. This report is supplemented with a hardware schematic, code walkthrough, and video evidence to show the demonstration of each part of the assignment.

Table of Contents

Introduction:

In the world of embedded systems and microcontroller programming, mastering the mechanisms of interrupts and timers is essential. It is crucial to comprehend how to effectively use these low-level hardware functionalities to optimize program execution and resource utilization. This practicum assignment provides a practical opportunity to delve deep into the intricacies of interrupts and timers using the Arduino Uno platform.

Throughout this assignment, the fundamental concepts of using interrupts and timers have been explored within microcontroller registers to control various input/output devices. By leveraging these powerful features, the operation of the microcontroller is optimized.

Divided into three assignments, this practicum challenges not only the comprehension and theoretical understanding of interrupts and timers but also the application of this knowledge in practical scenarios. From handling simple button presses to implementing complex timed tasks, the assignments build a robust foundation in real-time system design.

Moreover, emphasis is placed on fostering a deep understanding of the underlying hardware architecture, as we explore the datasheets of the ATmega328 microcontroller to decipher the correct use of registers and associated functionalities. This approach ensures a comprehensive grasp of how interrupts and timers enhance the performance and efficiency of embedded systems.

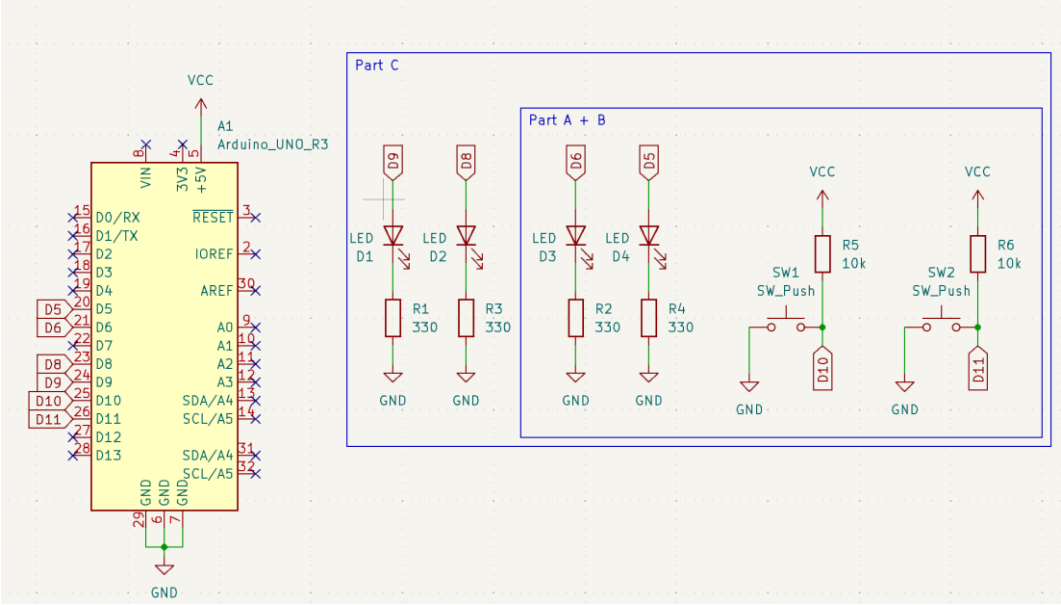
Procedure:

Equipment needed:

An Arduino Uno, 4x LEDs, 4x 330Ω resistors, 2x Buttons, 2x 10kΩ resistors.

Connections:

In the picture below the schematic for the assignment can be seen. Two buttons are used, each of which has a pull-up resistor to pull it high when the button is not pressed. The 4 LEDs are connected as usual to pins D5, D6, D8, D9.



Assignment A:

- When the user presses only button D10 then LED D5 turns on (and turns off when the button is released).
- When the user presses only button D11 then LED D6 turns on (and turns off when the button is released). During the press, the string "Hello World!\n" is repeatedly sent over the serial line and stops when the button is released.
- When both buttons D10 and D11 are pressed then both LEDs are blinking with alternating pattern repeatedly every 0.5 seconds.
- Use register manipulation instead of predefined functions like: pinMode(), digitalRead(), or digitalWrite().
- Implement a debounce solution using millis().
- Add comments for clarity.

Assignment B:

Refactor part A to include the following requirements:

- Each button must be handled by the correct interrupt service routine and make sure that the LEDs change state when the button is pressed on the right edge only.
- Do not use “expensive functions” in your interrupt service routines, i.e. an interrupt service routine (isr) must be quick and short at all times.
- As you know, buttons are nasty things because they cause a bouncing effect. You will need to debounce your buttons by yourself.

Assignment C:

Refactor part B to include the following requirements:

- Add two more LEDs
- Timer 1 runs in CTC mode.
- Timer 1 has a prescaler of 64.
- The Compare Match A interrupt routine displays a counter (4-bits binary number), by the four LEDs (0 = off, 1 = on). The speed by which the number is displayed is default approx. 4 times per second.
- The LEDs are connected as follows: Bit3: D9, Bit2: D8, Bit 1: D6, Bit 0: D5. With four bits we can count from 0x0 till 0xF Hex (0 till 15 decimal).
- Pressing button D10 yields an interrupt; the interrupt routine slows the update speed of the binary numbers down.
- Pressing button D11 yields an interrupt; the interrupt routine speeds up the update speed of the binary numbers.
- The timer 1 interrupt routine must also turn on and turn off the built-in LED at a rate of approx. 1x per second.
- Things that run at specific frequencies should not be influenced by delays in the loop().

Documentation:

- Write a technical report detailing the design and implementation.
- Record videos showing functionality.
- Submit the report, videos, and source code on Canvas.

Testing:

Thoroughly test solutions for functionality and adherence to criteria.

Submission:

Compile deliverables and submit on Canvas by the deadline.

Assignment A:

The first step, is to set up the input and output pins. To set up the buttons, on pins 10 and 11 the DDRB register is used, then to enable the pull-up resistors, a “1” has to be written to the respective button pins on the PORTB register. The DDRD register is used to set the LEDs as outputs. To control LEDs using the buttons, a debouncer was used that uses millis and returns Boolean with the value of the button. After getting the value a few checks using if statements are done to see if only the D10 button is pressed, if so it turns on the D5 LED. If the button D11 is pressed it turns on the second LED and prints out a “Hello World”, and if both of the buttons are pressed both of the LEDs blink with alternating pattern repeatedly every 0.5 seconds.

Assignment B:

In this part of the assignment falling edge interrupts need to be implemented for the button presses. To accomplish this, first interrupts have to be enabled and set up accordingly, to enable a pin change interrupt for the whole pin group the PCICR register is used in the following manner: PCICR |= (1 << PCIE0) the PCIE0 is the bit on that register that is responsible for edge interrupts in pin group 0 and then to enable the pin change interrupts for each pin PCMSK0 |= (1 << PCINT2) | (1 << PCINT3), the last thing to do in the setup is to enable the global interrupts using sei(). In the interrupt service routine, the PCINT0 vector is set and the only thing the ISR is doing is to read the values of the buttons and store them as a Boolean value. Then in the loop the debouncer solution is applied to the buttons. The debounce algorithm ensures that the output signal only changes when the input signal is stable for a predefined duration, effectively filtering out noise and transient spikes. The integrator, acting as a counter, smooths the input signal by incrementing or decrementing based on the input state, and the output only changes state when the integrator reaches its bounds, thus providing a stable and noise-free signal. And apart from that everything else is the same as in part A, the buttons handle the LEDs as required.

Assignment C:

This goal is to create a binary counter with four leds and blink the internal led at 1Hz. The update speed of the binary counter depends whether no buttons are pressed or which button is pressed. The setup of the falling edge interrupts is the same as in assignment B. To use timer interrupts they also need to be set up. To do this, the proper bits on the proper registers to achieve desired settings. For this assignment timer 1 on the Arduino is used because it will not interfere with the normal operation of the microcontroller and because the compare register has 16 bits so it offers higher values. For precaution also TCCR1B and TCCR1A registers are cleared before setting any new

settings. To make things more automatic, first the WGM12 bit on the TCCR1B register. This enables clear timer on compare match settings which resets the internal counter when it reaches the compare value. Then the compare value register for timer 1 OCR1A is set to 62499. This is the desired value to reach 4Hz frequency for interrupt events using the 64 prescaler according to the formula in the datasheet. To set the 64 prescaler, CS10 and CS11 bits on the TCCR1B register. Other combinations of the CS bits on this register lead to different prescaler settings. The prescaler basically divides the frequency of the chip by the prescaler to achieve less frequent timer events. Finally the compare event is set by setting the OCIE1A bit on the TIMSK1 register. After all the setting up is done, the code for the interrupt routine is in the TIMER1_COMPA_vect. To speed up the update of the leds when D11 is pressed the compare register value is set to 31249 so the interrupt event happens at 8Hz frequency. To make it slower when D10 is pressed the compare register is still with the default value in the setup, but a slowDownCounter variable is introduced, this variable increments every time an timer interrupt event happens, and increments the binary counter variable when whenever the slowDownCounter variable reaches 2. After it reaches it, the variable is reset to 0. This makes the counter update when two interrupt events happen, which as a result slows down the update of the leds. To maintain the internal led frequency, a ledCounter variable is used. At the default value of the compare value, the ledCounter variable needs to reach 2 for the led to flip states. When D11 is pressed, the ledCounter variable needs to reach 4 for the led to flip, because in this case the interrupt events happens twice faster.

Conclusion:

This assignment provides a rich experience with low-level programming of interrupts and timers on the Arduino Uno. The first part of the assignment revolves around the usage of buttons to control LEDs, this part only uses register manipulation for the buttons and LEDs, while part B implements a interrupt service routine that detects the falling edge of the button press and reads the values and in the loop we give these values to the debounce function to check for false signals and get the correct output data. Using interrupts proves to improve the accuracy and responsiveness of the application. Assignment C helps with understanding why timer interrupts are important and how to properly implement them to achieve better productivity and more optimized code.

Bibliography

I²C-bus specification and user manual. (n.d.). Retrieved 5 24, 2024, from NXP: http://www.nxp.com/documents/user_manual/UM10204.pdf

Instruments, T. (2022). *A Basic Guide to I2C.* Retrieved from https://www.ti.com/lit/an/sbaa565/sbaa565.pdf?ts=1716524445830&ref_url=https%253A%252F%252Fwww.google.com%252F

Sensortec., B. (2024, February). *BME280 Combined humidity and pressure sensor.* Retrieved from https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bme280-ds002.pdf