Fall 2023

School of Mechanical Engineering
College of Engineering
University of Tehran

Mechatronics Lab Manual

PREPARED BY

Mahdi Nozari
Graduate Teaching Assistant
nozarim@ut.ac.ir

Arvin Rezvani
Graduate Teaching Assistant
arvin.rezvani11@ut.ac.ir

Ali Sadighi, PhD
Assistant Professor
asadighi@ut.ac.ir

SESSION #3 Timer and Interrupts

Mechatronics

Lab Manual

Session #3 - Timer and Interrupts

School of Mechanical Engineering
College of Engineering
University of Tehran
Fall 2023

Session Objectives

- Introduction to hardware and software interrupt in Arduino.

Table of Contents

Introduction to TA

Mahdi Nozari, nozarim@ut.ac.ir
Arvin Rezvani, arvin.rezvani11@ut.ac.ir

Thanks to:

Amin Mirzaee, Mahdi Robati

1. Interrupt

An interrupt is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention. Whenever an interrupt occurs, the controller completes the execution of the current instruction and starts the execution of an Interrupt Service Routine (ISR) or Interrupt Handler.

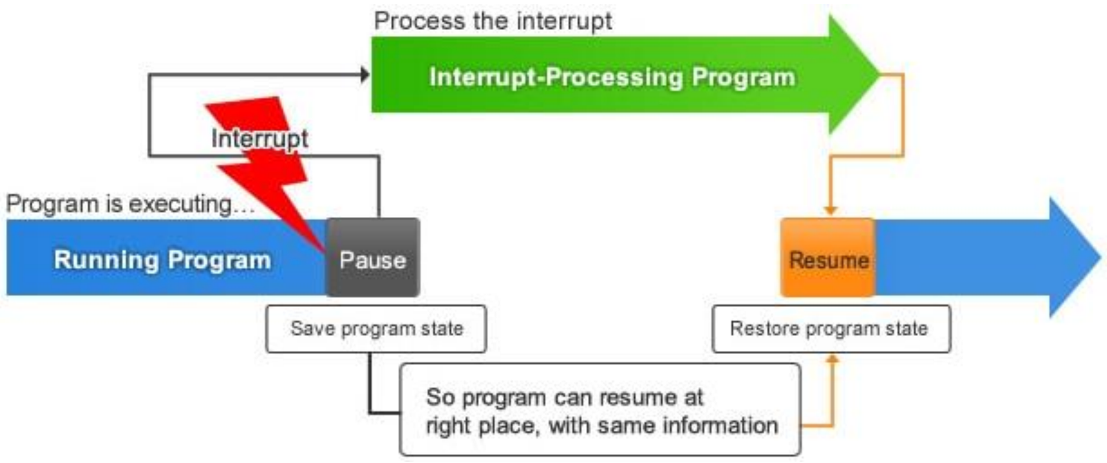

Figure. 1. Interrupt process.

1.1. Hardware Interrupt

A hardware interrupt is an electronic alerting signal sent to the processor from an external device, like an external peripheral. For example, a push button triggers a hardware interrupt that will cause the processor to read the push button state or do something.

1.1.1. Triggering methods

Each interrupt is designed to be triggered by either a logic signal level or a particular signal edge (level transition). Level-sensitive inputs continuously request processor service so long as a specific (HIGH or LOW) logic level is applied to the input. Edge-sensitive inputs react to signal edges: a particular (rising or falling) edge will cause a service request to be latched; the processor resets the latch when the interrupt handler executes.

1.1.2. Arduino Uno Hardware Interrupts

On the Arduino Uno, pins 2 and 3 can generate interrupts. The following function is used to declare a hardware interrupt:

Syntax: `attachInterrupt(digitalPinToInterrupt(pin), ISR, mode)`

Parameters:

- `pin`: the Arduino pin number.
- `ISR`: the ISR to call when the interrupt occurs; this function must take no parameters and return nothing. This function is sometimes referred to as an interrupt service routine.
- `mode:` defines when the interrupt should be triggered. Four constants are predefined as valid values: LOW, CHANGE, RISING, and FALLING.

1.2. Software Interrupt

A software interrupt is requested by the processor itself upon executing particular instructions or when certain conditions are met. Every software interrupt signal is associated with a particular interrupt handler. A software interrupt may be intentionally caused by executing a special instruction that, by design, invokes an interrupt when executed.

1.2.1. Clock Signal

In electronics and especially synchronous digital circuits, a clock signal oscillates between a high and a low state which is used to synchronize the operations of the circuit. Although more complex arrangements are used, the most common clock signal is in the form of a square wave with a 50% duty cycle, usually with a fixed, constant frequency.

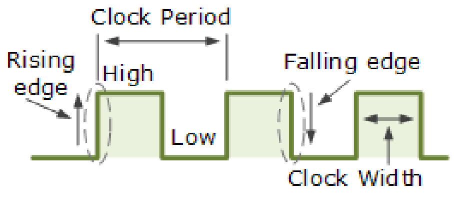

Figure 1: Clock signal.

1.2.2. Counters

In digital logic and computing, a counter is a device that stores the number of times a particular event or process has occurred, often in relationship to a clock. The most common type is a sequential digital logic circuit with an input line called the clock and multiple output lines. The values on the output lines represent a number in the binary or BCD number system. Each pulse applied to the clock input increments or decrements the number in the counter. Fig. 3 shows a typical 4-bit down counter circuit diagram.

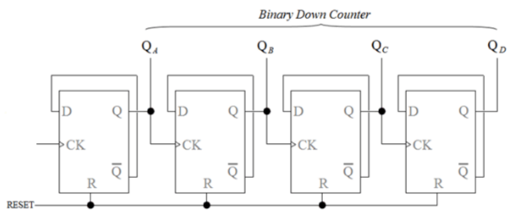

Figure 2: 4-bit down counter circuit diagram.

1.2.3. Timers

A timer is a clock that controls an event sequence at a fixed amount of time. Timers are used for precise delay generation, to trigger an activity before and after a predetermined time, and to measure the time elapsed between two successive events. The timer inside a microcontroller is a free-running binary counter along with some registers and circuits. The counter increments for each pulse that is applied to it. It counts continuously from 0 to $2^n - 1$ where n is the number of bits. The main advantage of timers is that it works independently of the microcontroller CPU, and the timer values can be read whenever needed. Fig. 4 shows the block diagram of a timer.

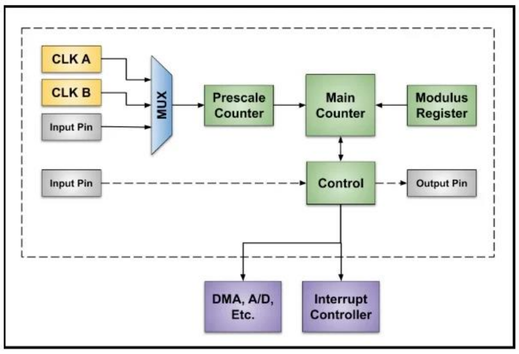

Figure 3: Block diagram of a timer.

Every timer needs a clock source or timebase. There are often several possible clock sources, and one is selected with a switch or multiplexer. Sometimes an external clock source is an option. To increase the count range, the selected clock goes to a "Prescaler," which divides the clock before it goes to the main counter. The output of the Prescaler goes to the main counter. The count range of the main counter is set by a "modulus" value stored in a register. Fig. 5 shows a count-up timer.

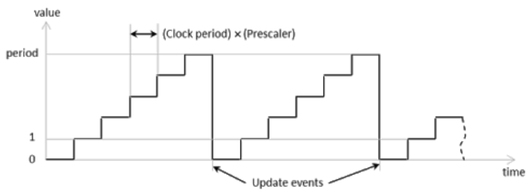

Figure 4: Count-up timer.

The main program and the timer are asynchronous, which means the timer operates independently of the program flow. The program can poll the timer to get timer information. Polling a timer is periodically reading status registers to detect timer events or the current value of a counter. This type of coordination can use much processing time, or the response time can vary significantly with a complicated program. The solution for these problems is using interrupts. Fig. 12 shows a typical program flow using a periodic interrupt.

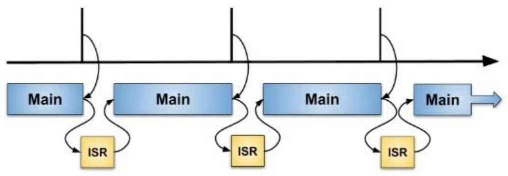

Figure 5: Program flow using a periodic interrupt.

"Main" is the regular program running on the processor. A timer event occurs and triggers an interrupt. For example, an interrupt occurs when a down-counting timer reaches 0 and reloads the counter value to the modulus in the main counter. The timer sends a hardware signal to an "interrupt controller," which suspends the execution of the main program and makes the processor jump to a software function called an ISR. When the ISR is done, the main program resumes. Notice how the main program does not spend time checking for a timer event. The response to the timer event can be swift and predictable.

1.2.4. Arduino Uno Timers

ATmega328P, the main microcontroller of Arduino Uno R3, has three timers called Timer0, Timer1, and Timer2. Timer0 and Timer2 are 8-bit timers, meaning they can store a maximum counter value of 255. Timer1 is a 16-bit timer, meaning it can hold a maximum counter value of 65535. Once a counter reaches its maximum, it will tick back to zero. Prescaler is set to 64 for all three timers by default. In the Arduino world, Timer0 is used for timer functions like delay(), millis(), and micros(). Keep in mind that changing Timer0 registers may influence the Arduino timer function.

1.2.5. TimerOne Library

This library is a collection of routines for configuring the 16-bit hardware timer called Timer1 on the ATmega328. The development of this library began with the need for a way to quickly and easily set the PWM period or frequency but has grown to include timer overflow interrupt handling and other features. It could easily be expanded upon or ported to work with the other timers. The following functions that are included in this library will help set up a timer-based interrupt:

Syntax: `Timer1.initialize(microseconds)`
Description: Sets the period in microseconds. The minimum period this library supports is 1 microsecond. The maximum period is 8388480 microseconds or about 8.3 seconds.

- `microseconds`: the period in microseconds.

Syntax: `Timer1.attachInterrupt(function)`
Description: Calls a function at the specified interval. The time it takes to complete the tasks in this function should not exceed the interrupt time interval. Otherwise, the CPU may never enter the main loop, and the program will "lock up."

- `function`: the function name to be called.

Syntax: `Timer1. detachInterrupt()`
Description: Disables the attached interrupt.

2. Reminder: Pushbutton

There are generally 2 ways to know when a button is pushed.

1. Pulldown: In this circuit, when the button is pushed the digital input will be 1 (5V) and when not pushed it's 0 (0V).
2. Pullup: The input will be 0 when the button is pushed and 1 when not pushed.

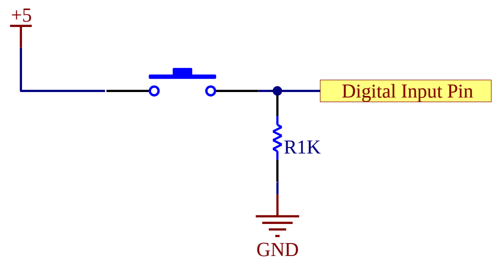

Figure 7:Pull-down resistor circuit diagram

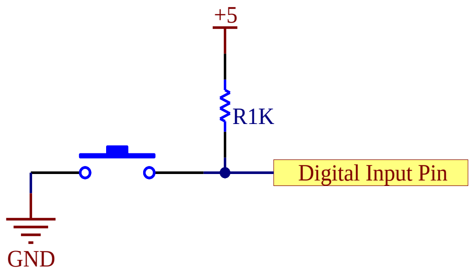

Figure 6: Pull-up resistor circuit diagram

Arduino has internal pullup resistors connected to its digital pins. In order to use these resistors, the pin modes have to be set to INPUT_PULLUP.

3. Debouncing

Bouncing is a phenomenon occurred when a physical switch is pressed and the signal bounces between 0 and 1 before it settles on a state. Although the time is very short, the microcontroller can pick up these signals and consider them as if the switch is pressed a few times instead of one. This becomes specially important if hardware interrupt is used. Overcoming this challenge is called debouncing, and can be managed with two general approaches.

1. Hardware Debouncing
   If an external circuit or device is used to eliminate bouncing, it is called hardware debouncing. This can be managed by using RC filters and SR latches. There are also dedicated ICs such as MAX6816 available for this purpose.
2. Software Debouncing
   If an external circuit is not available, the problem can also be taken care of with a little bit of coding. The idea behind software debounce is to measure the time between two switches and if it is short enough, it is considered to be a bounce.

Task 1: Hardware Interrupt

Make a simple chronometer.

- Single press → Start or stop
- Double press → Reset
- Software debouncing → To restrict the calling of a function frequently

Task 2: Software Interrupt

Write a program that reads the voltage from a potentiometer with a frequency of 2Hz and prints it in the serial monitor.