****

**Department of Systems Design Engineering**

**Biomedical Engineering**

**Digital Systems Laboratory   
BME 393L Lab manual**

**Lab 5 & 6 Final Project: Background Information on Timers and Interrupts**

**Systems Design Engineering**

**University of Waterloo**

**Waterloo, Ontario N2L 3G1, Canada**

1. **Intended Learning Outcomes**

After reading this manual, students will be able to:

* Use Arduino’s ATMega’s built-in timers/counters in a digital design.
* Use Arduino’s ATMega’s interrupts in a digital design.
* Understand best practises for coding microcontroller based timers and interrupts.

1. **Background** 
   1. **Arduino Uno**

The Arduino is a multipurpose microcontroller board based on the ATmega328. A microcontroller (in this case an ATmega328) is a small computer on a single integrated circuit, see figure 2.1. In this integrated circuit there is a processor core, memory, and input/output peripherals. The full datasheet of the microcontroller can be found on LEARN. An overview of the more important microcontroller specifications can be found in Table 2.1.

This LED will be on if there is power

A close-up of a computer chip

Description automatically generated with low confidence

LED connected to pin 13

Reset Button

Power Inputs

Microcontroller (ATmega328)

Figure 2.1: Arduino Uno Microcontroller Board (front). Picture from arduino.cc/en/Main/

Table 2.1 Important ATmega328 specifications

|  |  |
| --- | --- |
| **Microcontroller** | **ATmega328** |
| Operating Voltage | 5V |
| Input Voltage | 7-12V |
| Max Input Voltage | 6-20V |
| Digital I/O (input/output) pins | 14 (6 of which are PWM output) |
| Flash Memory  (Used to store program) | 32kB (0.5 kB used by bootloader\*) |
| SRAM  (Memory for variables in your program) | 2kB |
| EEPROM  (Memory used for long term storage) | 1kB |
| Clock Speed  (Determines how fast a program will run) | 16MHz |

\*A bootloader is a program that loads up an operating system

The Arduino can be powered by either an external power supply or power can be supplied through the USB port of a computer. There is also a reset button and a power indicating LED (see figure 2.1)

* 1. **Using Arduino Digital I/O, Analog read, and Power Pins**

Refer to Figure 2.2 for pin locations.

Digital Input Output Pins: often referred to as Digital I/O pins, are used to read in digital voltages (High or Low), or to apply 5 V or ground, to a pin. See <https://docs.arduino.cc/learn/microcontrollers/digital-pins>.

To use digital I/0 pins: in the setup function, use pinMode(pin#, mode) to tell the Arduino if the specified pin will be an INPUT, OUTPUT, or an INPUT\_PULLUP. e.g. pinMode(2, OUTPUT) will define pin 2 as an output pin which can be set as 5 V or ground. See <https://docs.arduino.cc/language-reference/en/functions/digital-io/pinMode/>.

Analog Read Pins: will digitize (10 bits) the voltage seen at an analog pin and return an integer value between 0 (for ground) to 1023 (4.9951 volts). See <https://docs.arduino.cc/language-reference/en/functions/analog-io/analogRead/>.

To digitize a voltage: Connect the voltage an analog in pin and your code call analogRead(pin#).

Power and Ground Pins:

Ground: The Arduino has three pins connected to ground which can be used as a ground when connected to a breadboard.

Voltage Pins: You can use the 5 V pin on the Uno to power a circuit (400 mA limit). You can also use the 3.3 V pin to supply power to a circuit (150 mA limit). Do not use the Vin power pin to power your circuits.

A close-up of a computer chip

Description automatically generated with medium confidence

Digital I/O Pins

Analog In Pins

Power and Ground Pins: available to provide 5V 3.3V and ground

Figure 2.2: Uno Board Front. Picture from arduino.cc/en/Main/

* 1. **Arduino Hardware Timers**

Hardware Timers use counter circuits, and these circuits count clock cycles (which are very precise, fixed amounts of time). Therefore, the timer is a special case of the counter. An n-bit binary counter is made up of n flip-flops with circuitry that counts in binary from 0 to 2n – 1. The Arduino Uno has three timers: Timer0 and Timer2 are 8bit timers, and Timer1 is a 16bit timer. Therefore Timer0 and Timer2 can count up to 256, and Timer1 can count up to 65,536. Notice that the number counted has no units. Each timer has its own timer frequency. The frequency value is determined by the system clock (16 MHz for the Uno) which can be divided to a lower value by the timer’s prescaler. If for example the prescaler is set to 8, the timer frequency would be 16 MHz / 8 = 2 MHz.

We use timers to determine the lapsed time between events. To calculate the lapsed time we multiply the timer’s clock frequency by the number of counts between events. For example, if we wanted to know the time it took to push a button twice, using Timer0, what would we do?

1. Set Timer0’s frequency by setting a prescaler value. To keep the example simple let’s assume we set the prescaler value to 2 million. The timer’s clock frequency would be 16 MHz (the Uno’s system clock frequency) divided by 2 million, or 16 MHz / 2M = 16 Hz. Therefore Timer0 would be counting at a rate of 16 counts per second or 62.5 ms per count.
2. When the first button press is registered, check Timer0’s count value, let’s say it returned a value of 10.
3. Wait until the button is pressed again.
4. When the second button press is registered, check Timer0’s count value, let’s say it now returns a value of 30.
5. Determine the time between button presses by
   1. Subtracting the second button count value of 30 from the first button count value of 10. 30 counts – 10 counts = 20 counts.
   2. Knowing that Timer0’s clock frequency is 16 Hz, we would multiply the 20 counts \* 62.5 ms/count = 1.25 seconds.

We can now say it took 1.25 seconds to push a button twice.

Note: for longer periods of time the timers will “roll over,” meaning the timer has counted from 0 to 2n – 1 and started back at 0. We must keep track of how many times the timer has rolled over and take this into account.

* 1. **Arduino Hardware Timer Register**

Microcontrollers use Registers to store values and set options (such as prescalers values). Registers use 2n number of flip flops wired together, to store bits of information, 8 bits registers are common. These bits are be written or read by the microcontroller hardware and any program it is running.

Diagram, schematic

Description automatically generated

Figure 2.3: Example circuit of an 8-bit register, with inputs at the top, and stored values at the bottom. Picture from ee.bradley.edu

To access these bits, in a program, you must address the register then read or write the registers values. Registers are not addressed in binary; they are addressed as HEX values. When addressing a specific register, to read or write values, you use HEX numbers. If you are unfamiliar with HEX numbers the following reference may help: <http://en.wikipedia.org/wiki/Hexadecimal>. However, when using the Arduino you can address registers by their name, such as Timer0 register TCCR0B instead of address it as 0x25, see below).

The behaviour of the hardware timers can be changed through timer registers. Looking at Timer0, its important registers are:

* TCCR0B (Timer/Counter Control Register, address 0x25), Timer0’s prescaler can be configured here. See pages 109/110 of the ATMega328 Datasheet for more details for Timer0.
* TCNT0 (Timer/Counter Register, address 0x26), holds the value, in counts, of the timer.

The ‘0’ in these register names can be 0, 1, or 2 depending on which timer (Timer0, Timer1, or Timer2) you are using.

There are other timer related registers, but these are the two we are most concerned with.

**Bit Bashing with \_BV:**

The \_BV() macro is available to you so you can change the bit values in individual registers. Posted to Learn is a document titled BV Macro Explained. Below is an example code snipit to demonstrate how \_BV is used to change the default speed of Timer0

TCCR0B |= \_BV(CS02); // This will Set bit CS02 of register TCCROB to one, it’s the most signifigant bit of Timer0's prescaler. See page 110 of the datasheet.

TCCR0B &= ~\_BV(CS01); // This will Clear bit CS01 of register TCCROB to zero, See page 110 of the datasheet.

* 1. **Hardware Interrupts**

Interrupts interrupt, or halt the normal execution of a program. Hardware interrupts are circuits that react to an external condition (i.e. state of a pin), when such an event occurs the interrupt will immediately stop the current program execution and instead execute an interrupt handler routine. Once that subroutine (interrupt handler routine) has finished executing it will return to the main program where it left off and resume normal execution order.

In many programs, it is left to the program to continuously check for a change of state at a pin by explicitly reading it (using DigitalRead). Often the program checked the state inside control structures (if/while statements). This is called polling. Interrupts on the other hand will actively stop the normal execution of a program. Note: only pins 2 and 3 on the Uno are capable of generating interrupts.

* 1. **Using Interrupt Subroutines**

The interrupt subroutine (often called a handler) is the function that executes when an interrupt is triggered.

To use interrupts with the Uno board, in the setup loop call:

attachInterrrupt (digitalPinToInterrupt(pin), handler/subroutine, trigger type)

For example:

attachInterrupt(digitalPinToInterrupt(2), interrupt, RISING);

Which would have a function called:

void interrupt()

{

Some short fast code

}

See <https://docs.arduino.cc/language-reference/en/functions/external-interrupts/attachInterrupt/>. Be sure to read the Note in the link.

* + 1. **Volatile Variables and ‘Rules’ for Interrupt Subroutines**

As was mentioned in the note section of the link above, any variable in the interrupt function must have the volatile qualifier attached. The volatile qualifier ensures that any variable is only stored in RAM. Without this qualifier a variable can be stored in a register which is not guaranteed to be accurate when an interrupt is generated. For more details see <https://docs.arduino.cc/language-reference/en/variables/variable-scope-qualifiers/volatile/>

While an interrupt subroutine is executing, the main code will be paused and other interrupts will not be executed. Because of this it is important for interrupt subroutines to take as little time as possible, if a subroutine takes a long time it can cause other built-in routines to fail. For example while the Uno is executing an interrupt, subroutine dalay() won’t work, and millis() won’t update.

‘Rules’ for what can and can’t be done in an interrupt subroutine:

Do’s - Digital reading and writing of pins.

- Read and write variables (i.e. setting a flag to indicate that a subroutine has been executed).

- Addition and subtraction.

- Simple ‘If’ and ‘Case’ statements.

Don’ts - Serial prints.

- Division.

- ‘While’ and ‘For’ loops.

- Analog Reads.

* 1. **Using buttons with a microcontroller**

**Bouncing:**

When a button is activated, they ‘bounce,’ which is the physical effect of a button making and braking contact (due to the switches’ metal elasticity). See figure 2.4.

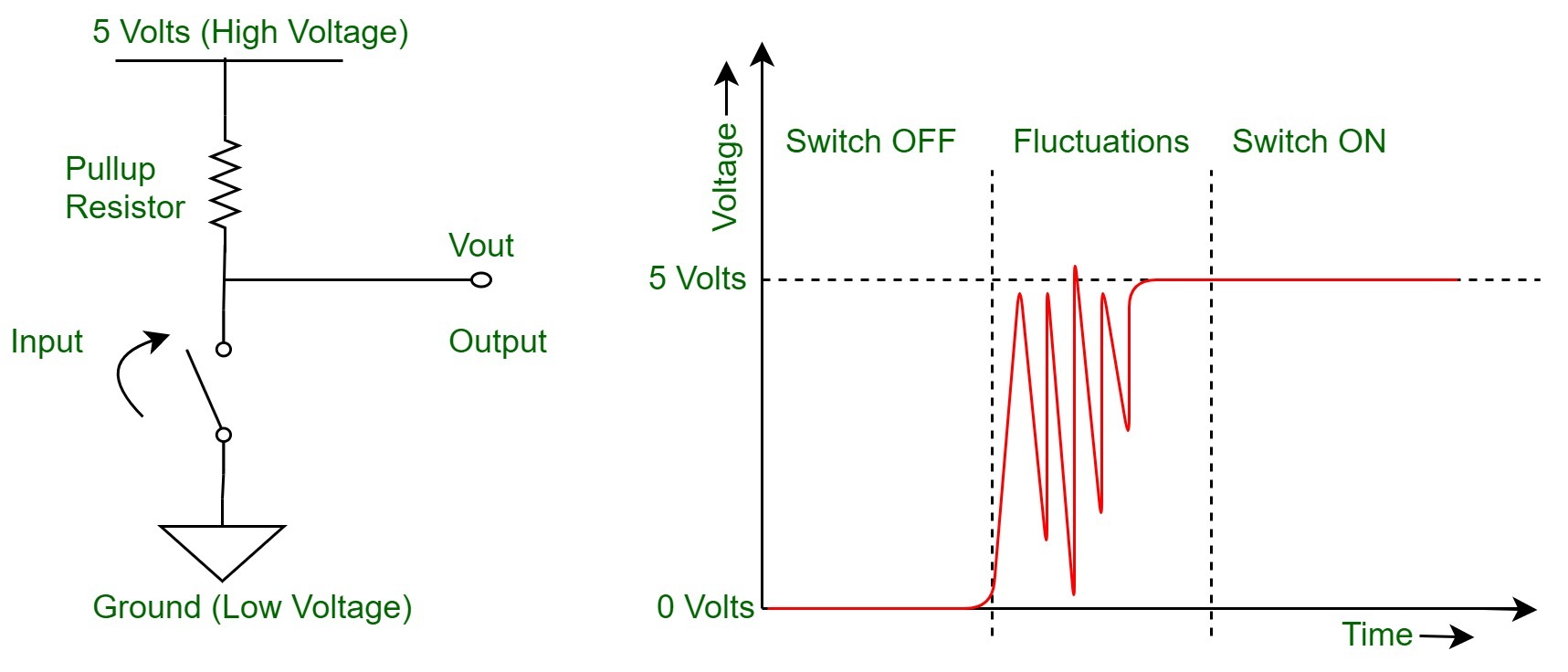


Figure 2.4 Button Bouncing, from geeksforgeeks.org

When a button bounces the system may register it as multiple button presses, however there are many hardware and software solutions to bouncing buttons. The simplest solution is to code an algorithm that, once a first press is registered, will ignore further button presses for a short time – Just don’t do it in an interrupt subroutine.

**Button Pulldown:**

If a pin is left to float it’s value can be read as High, Low, or oscillate between the two values. To prevent unpredictable readings at a pin, all inputs should be pulled high or low. For the project, groups should use hardware pulldown circuits for their buttons. See figure 2.5, when a pulldown circuit is used the digital value of a button will be low when it’s not activated. This is because when the button is open there is no current flow. And without current flow there will be no voltage drop across R1.

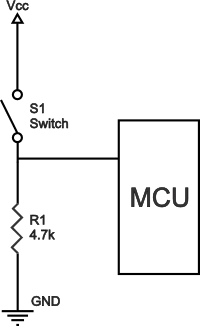


Figure 2.5 Pulldown Circuit

End.