Department of Computer Engineering
Faculty of Engineering, University of Peradeniya

CO321 Embedded Systems - 2019

Lab 3 - Timers

Using Timers in AVR

Introduction to timers/counters

Many applications need to count occurrences of an event or generate time delays. So, there are counter registers in microcontrollers for this purpose. When we want to *count events*, we connect the external event source to the clock pin of the counter register. Then, when an event occurs externally, the content of the counter is incremented; this way, the content of the counter represents *how many times the event has occurred*.

When we want to generate time delays, we *connect the oscillator* to the clock pin of the counter. So, when the oscillator 'ticks', the content of the counter is incremented. As a result, the content of the counter register represents how many 'tick' events have occurred from the time we have cleared the counter. Since the speed of the oscillator in a microcontroller is known, we can calculate the 'tick' period. Therefore, using the content of the counter register we can derive how much time has elapsed altogether.

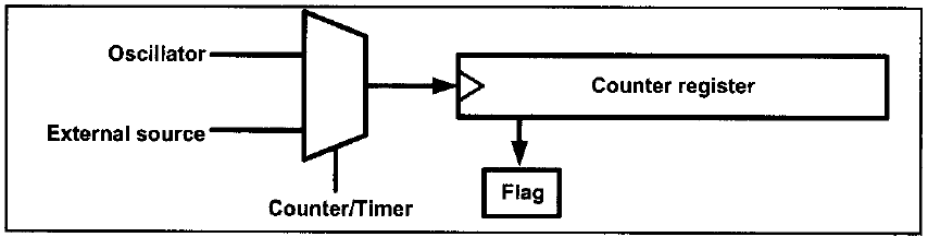

Figure 1: A high-level schematic of counters and timers in microcontrollers

Figure 1 shows the high-level schematic of a timer module. All the Atmel microcontrollers have Timers as an inbuilt peripheral. ATmega328P has three timers: *TIMER0, TIMER1,* and *TIMER2*. They also have a *Watchdog Timer*, which can be used as a safeguard or software reset mechanism.

Here are a few details about each timer:

TIMER0
TIMER0 is an 8-bit timer, meaning its counter register can record a maximum value of 255 (unsigned 8-bit). *TIMER0* is used by native Arduino timing functions such as `delay()`.

TIMER1
TIMER1 is a 16-bit timer, with a maximum counter value of 65535 (unsigned 16-bit integer). The Arduino Servo library uses this timer.

TIMER2
TIMER2 is an 8-bit timer that is very similar to *TIMER0*. It is utilized by the Arduino `tone()` function.

Using *TIMER0*

To use a timer, we need to set it up, and then get it to start counting clock 'ticks'. To do this, we'll use built-in registers on the AVR chip that store timer settings. Each timer involves a number of registers that do various things and the registers corresponding to *TIMER0* are described below. When programming, we can access these timer registers directly using their names.

- **TCCR0A and TCCR0B**

These two 8-bit registers hold setup values for the *TIMER0*. TCCR stands for *Timer/Counter Control Register*.
Each register holds 8 bits, and each bit stores a configuration value as shown in Figure 2.

TCCR0A – Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0	
0x24 (0x44)	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	TCCR0A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TCCR0B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figure 2: Timer/Counter0 control registers A & B

To start using our timer, the most important settings are the last three bits in TCCR0B: CS12, CS11, and CS10. These dictate the timer clock signal (external or oscillator) and prescaling. By setting these bits in various combinations, we can tell the timer to count at different scales of the oscillator clock speed (×1, $\times\frac{1}{8}$, $\times\frac{1}{64}$, etc.). In other words, we can slow down the timer's clock compared to the oscillator clock. Figure 3 shows the relevant table from the data sheet.

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

Figure 3: Clock select bit description

- **TCNT0**

This is the *Timer/Counter Register* where the 8-bit counter value for *TIMER0* resides. The value of the counter is stored here and increases/decreases automatically. Data can be both read/written from this register.

- **TIFR0**

The *Timer/Counter Interrupt Flag Register* is a register shared by all the timers. Bits 1 and 0 are allotted for *TIMER0*. At present we are interested in the zeroth bit named as TOV0 bit. This bit is set ('1') whenever *TIMER0*'s counter *overflows* (exceeds the maximum value). This bit is cleared ('0') whenever the corresponding Interrupt Service Routine (ISR) is executed. If there is no ISR in the program to execute, we should clear it manually by writing logic '1' to it.

For complete information about the registers, refer to section 15.9 in the ATmega328P datasheet.

Modes of operation for *TIMER0*

TIMER0 has several modes of operation, that is, different behaviors for the Timer/Counter. They are namely, Normal Mode, Clear Timer on Compare Match (CTC) Mode, Fast PWM Mode and Phase Correct PWM Mode. The mode is defined by the combination of the Waveform Generation mode (WGM02:0) and Compare Output mode (COM0x1:0) bits. In this class, we are only focusing on the normal mode.

Normal mode of *TIMER0*

In this mode, the value of the timer/counter increments with the clock. It counts up until it reaches its max of `0xFF`. When it rolls over from `0xFF` to `0x00`, it sets the flag bit called TOV0 (Time Overflow) in TIFR0 Register. This time flag can be monitored.

For complete information about the modes of operation, refer to section 15.7 in the ATmega328P datasheet.

Steps to program *TIMER0* in normal mode

To generate a time delay using *TIMER0* in Normal mode, the following steps can be taken. (note that this description uses a polling method instead of interrupts).

1. Load the TCNT0 register with the *initial count value*.
2. Load the values into the TCCR0A and TCCR0B register, indicating which mode is to be used and the prescaler option (you will have to refer to the datasheet to identify the bits that should be set for the corresponding mode). When you select the clock source, the timer/counter starts to count, and each tick causes the content of the timer/counter to increment by 1.
3. Keep monitoring the Timer Overflow flag (TOV0) to see if it is raised. Get-out of the loop when TOV0 becomes high.
4. Stop the timer by disconnecting the clock source.
5. Clear the TOV0 flag for the next round.
6. Go back to step 1 to load TCNT0 again.

Example:

Write a C program to toggle all the bits of PORTB continuously with a fixed delay. Use *TIMER0*, in normal mode, and no pre-scalar options to generate the delay. (go to next page for the sample code)

Code :

```
#include <avr/io.h>
void delay_timer0(){

    TCNT0 = 0x00;    /*Load timer counter register with 0*/

    TCCR0A = 0x00;   /*Set the Timer0 under normal mode with no prescaler*/
    TCCR0B = 0x01;

    while((TIFR0&0x01)==0); /*Wait till timer overflow bit (T0V0) is set*/

    TCCR0A = 0x00;      /*clear timer settings (this stops the timer)*/
    TCCR0B = 0x00;

    TIFR0 = 0x01;   /*Clear the timer overflow bit (T0V0) for next round*/
    /*strange thing about this flag is that in order to clear it we should
    write 1 to it This rule applies to all flags of AVR chip*/

}

int main (void)
{

 DDRB = DDRB | (1<<5);         /* configure pin 5 of PORTB for output*/

 while(1) {

  PORTB = PORTB | (1<<5);    /* set pin 5 high to turn led on */
  delay_timer0();

  PORTB = PORTB & ~(1<<5);   /* set pin 5 low to turn led off */
  delay_timer0();

 }
}
```

Calculating delay length for timers

The delay length for TIMER0 in normal mode depends primarily on two factors:

a) The crystal oscillator's frequency
b) The pre-scalar factor

A third factor in the delay size is the C compiler because various C compilers generate different hex code sizes, and amount of overhead due to the instructions varies by compiler.

Exercise 1:

Write a C program to toggle only bit 5 in PORTB register (PB5) every 70µs. Use *TIMER0*, normal mode, and 1:8 prescaler to create the delay. Assume XTAL=16MHz.

Derivation of initial counter value:

XTAL = 16MHz → $T_{xtal\_clock} = \frac{1}{16}$ µs

Prescaler = 1:8 → $T_{counter\_clock} = 8 \times \frac{1}{16}$ µs = 0.5 µs

Counter increments needed = 70µs / 0.5µs = 140 increments

Initial counter value = 1+255 – 140 = 116

Code :

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

void delay_timer0(){

    TCNT0 = 116;        /*Load timer counter register*/

    TCCR0A = 0x00;      /*Set the Timer0 under normal mode with 1:8 prescaler*/
    TCCR0B = 0x02;

    while((TIFR0&0x01)==0); /*Wait till timer overflow bit (T0V0) is set*/

    TCCR0A = 0x00;      /*clear timer settings (this stops the timer)*/
    TCCR0B = 0x00;

    TIFR0 = 0x01;       /*Clear the timer overflow bit (T0V0) for next round*/
    /*strange thing about this flag is that inorder to clear it we should write 1 to it.
    This rule applies to all flags of AVR chip*/
}

int main (void)
{

    DDRB = DDRB | (1<<5);        /* configure pin 5 of PORTB for output*/

    while(1) {
        PORTB = PORTB ^ (1<<5);    /* toggle pin 5 */
        delay_timer0();
    }
}
```

Exercise 2 :

Write a C program to toggle an LED connected to pin 5 of PORTB register (PB5) every 2ms. Use *TIMER0*, normal mode, and a suitable pre-scalar to create the delay. Assume XTAL= 16 MHz. What is the selected prescaler? What is the initial counter value? Explain the output. What is the reason for it?

Exercise 3 :

Try to increase the delay in Exercise 2 to 500ms. Talk to an instructor and explain how this can be done, or why this cannot be done.

Exercise 4 :

Find out the highest possible countable time interval using *TIMER0*, normal mode, and a suitable pre-scalar. Assume XTAL= 16MHz. What is the selected prescaler? What is the highest countable time interval?

Exercise 5:

Write a program that toggles an LED connected to PB5 pin every second using TIMER1, while at the same time operating a knight rider circuit of 4 LEDs (from Lab 1). For the LED toggle you should use the timer overflow interrupts. For the knight rider circuit you are allowed use either timer interrupts, polling, software delays or any other method.

(You must finish and show exercise 1-5 on the first day of Lab3. Remaining exercises can be finished before coming to the second day of Lab3)

Exercise 6:

You found out the maximum interval you can implement using *TIMER0* in exercise 4. However, by using an 8-bit counter variable and interrupts, you can make this interval much longer. Develop a program to blink an LED with a 100ms interval using TIMER0 .

Exercise 7:

Implement a program to blink two LEDs. One led should blink with an interval of 50 ms and the other should blink with an interval of 500ms. You must use two timers, and timer overflow interrupts.