Lab 7: Timers and Counters: Making a Clock

CompEng 3151: Digital Engineering Lab II

Last revised: July 16, 2019 (BJZ, RJS)

Description/Overview

Timers are standard features available in almost all microcontrollers. The AVR microcontroller has powerful and multifunction timers. Its timers can be used to generate a delay, count events, and generate PWM waveforms, among their other uses.

During this lab, you will use your knowledge of AVR timers to create a working clock display. The time will be displayed on the LCD display you learned about during the last lab and will be changeable by a few of the switches on the AVR Simon Board.

Objectives

Students will:

- Configure the registers for a timer/counter on the AVR microcontroller
- Generate a time delay using the AVR timers
- Create an interrupt routine to handle a counter operation

Materials Required

- Atmel Studio 7 Installed on PC (ECE CLC Computers will have this)
- AVR Simon Board with a USB cable
- 2x16 Character LCD Module
- Connection Wires

Preparation

In order to be successful with this lab, students should review Chapter 9 in the Mazidi textbook.

Background *(excerpted from electroschematics.com - used by permission of author T.K. Hareendran)*

As mentioned earlier, the AVR has timers that can serve a number of purposes. But what exactly is a timer?

A timer is a register of 8- or 16-bit size. This size determines how large it can count or time and is therefore also known as the resolution of the timer (i.e. 8-bit timer, 16-bit timer).

A timer is a simple counter. The input clock of microcontroller and operation of the timer is independent of the program execution. All the Atmel microcontrollers have timers as built-in peripherals. The AVR microcontrollers we will be working with come with two 8-bit (TIMER0 & TIMER2) and three 16-bit timers

(TIMER1, TIMER3, & TIMER4). This means that there are 5 sets of timers, each with the ability to count at different rates.

In an 8-bit timer, the register is 8 bits long and thus can store a number from 0 to 255 or count to 255. Likewise, a 16-bit timer can hold a value between 0 to 65535 or count to 65535. This register has a property of increasing or decreasing its value without any intervention by CPU at a rate frequently defined by the user. This frequency (at which the timer register increases/decreases) is known as the Timer Frequency. Note that Timer Frequency can be less than or equal to the CPU clock frequency.

Timers can run asynchronously to the main AVR core hence timers are totally independent of CPU. A timer is usually specified by the maximum value to which it can count, beyond which it overflows and resets to zero.

The speed of the counting can be controlled by varying the speed of clock input to it. As we know, all the components of a microcontroller unit (MCU) are someway connected to the central processing unit (CPU) and some registers, to share information between them.

TIMER0 – 8-bit Timer

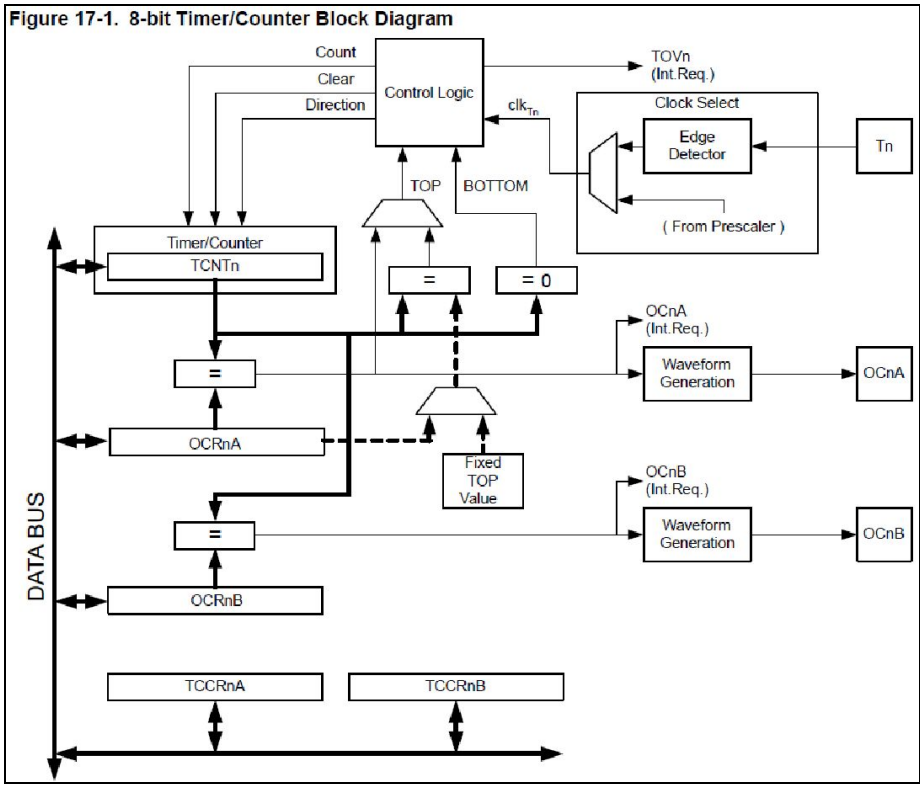


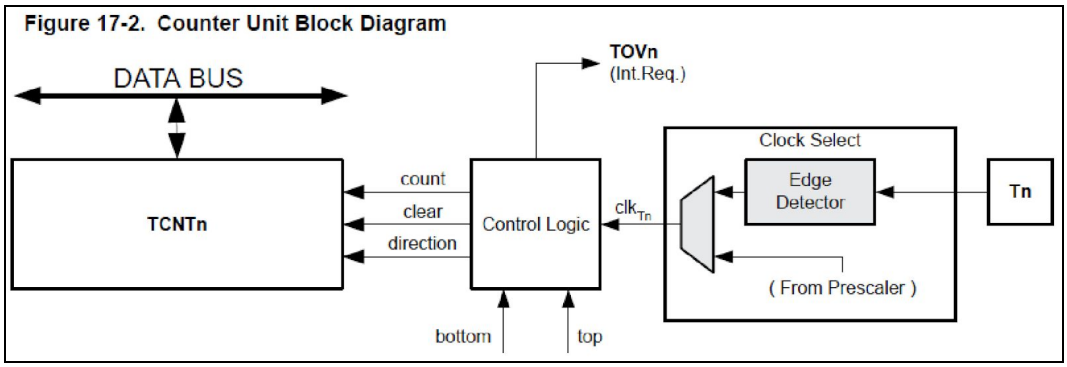
Figure 17-1. 8-bit Timer/Counter Block Diagram

The ATmega324PB has 5 different timers, of which the simplest one is TIMER0, with an 8 bit (0-255) resolution. The ATmega controllers provide hardware counters. Those counters are registers that are incremented normally by a signal from the oscillator which also drives the ATmega. The oscillator is not connected directly to the counter but is instead connected via a variable prescaler than can be used to run the timer/counter slower. To every counter, there is at least one compare register. When the counter value equals the value of the compare register, a certain action can be triggered. Actions can also be triggered when the timer value reaches an overflow.

Prescaler: We already remarked that the Timer frequency can be less than or equal to the CPU clock frequency. OK, then how we decide or set the timer frequency? Here the "Prescaler" comes in! It is the method of generating the clock frequency for the TIMER from the CPU clock by dividing it with a suitable number.

Overflow: The timer in some conditions automatically take an action or informs the CPU to suspend the current execution, by an Interrupt signal. One example of this is the Timer Overflow (i.e. an 8-bit timer has counted up to its maximum value (255) and reverted to 0). Here, the timer sends a signal or interrupt to the CPU to break its current execution and execute the ISR (interrupt service routine). ISR is a function that the CPU should execute whenever an interrupt occurs. The programmer must write into the ISR to handle the interrupt.

The main part of the 8-bit Timer/Counter is the programmable counter unit. The Timer/Counter can be clocked by an internal or an external clock source. The clock source is selected by the clock select logic which is controlled by the clock select bits located in the Timer/Counter Control Register.



The Timer0 in ATmega8 has some registers:

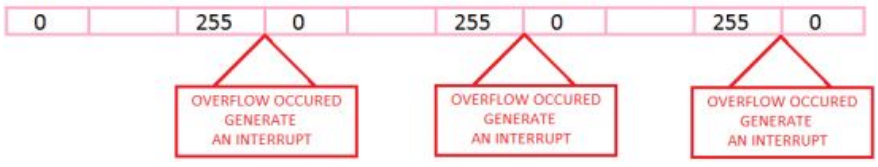
- **TCCR0A/B:** Timer Counter Control Registers are used to configure the timer. Each of them contains 8-bits, although not all of their bits are used. The TCCRs are used to set the timer mode, configure a waveform generator (PWM – we will not use) and set up the prescaler.

- **TCNT0:** Timer Counter0 register is the "real" counter in the TIMER0 counter. The timer clock counts this register as 1 (i.e. the timer clock increases the value of this 8-bit register by 1 with every timer clock pulse). The timer clock can be defined by the TCCRO register.

- **TIMSK0:** Timer Interrupt Mask Register – used to activate/deactivate interrupts related to the timer. It controls the interrupts of all three timers. BIT0 (the first bit from the right) controls the overflow interrupts of TIMER0.

- **TOIE0:** This bit, when sets to "1" as a part of TIMSK0, enables the OVERFLOW interrupt.

An ISR (Interrupt Service Routine) is a function that the CPU should execute whenever an interrupt occurs/is triggered. In the case of Timer0, when the timer rolls from 255 to 0, the TOV0 flag is set. This flag can be monitored and used to trigger an interrupt routine.



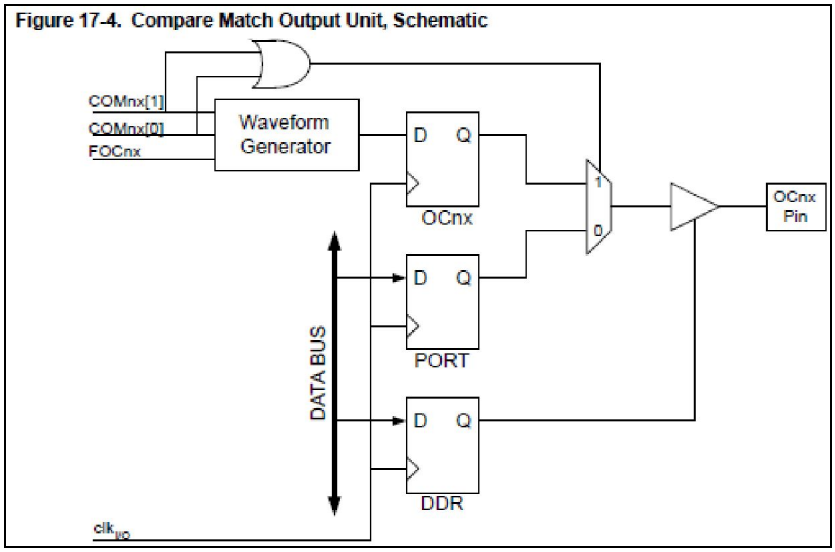
Timer Modes

Timers are usually used in Normal Mode and CTC mode. In Normal Mode, when the prescaler receives a pulse from a clock cycle and passes it onto the Control Logic, the Control Logic increments the TCNTn register by 1. When TCNTn hits the TOP (0xFF in the 8-bit timers and 0xFFFF in the 16-bit timer) it overflows to 0 and sets the TOVn bit in the TIFR register. The problem with this Normal Mode is that it is very hard to use for an exact interval.

CTC stands for "Clear Timer on Compare". In CTC Mode, when the prescaler receives a pulse from the clock cycle and passes it onto the Control Logic, the Control Logic increments the TCTn register by 1. The TCNTn register is compared to the OCRn register when a compare match occurs the TOVn bit is set in the TIFR register. What is interesting is that Timer/Counter1 can run in Normal mode (0xFFFF) and in two CTC modes. The difference between the two CTC modes is that mode 4 uses the OCR1A register for its compare value, and mode 12 uses the ICR1 register. In normal mode, TOV1 can generate an Overflow interrupt, while in CTC (mode 4) mode OCIF1A can generate an interrupt when it detects a compare match, and in CTC (mode 12) mode TICIE1 can generate an interrupt when it detects a compare match. The 3rd, Timer/Counter2 (8-Bit) is the preferred one for short time delays. It can run in Normal mode or CTC mode.

Output Compare Mode

In some applications, we need a very specific time period, or occurrence of an event before the register overflows. This time we want the CPU to execute ISR when TCNT0 register reaches a particular value. Output Compare Mode is a practical solution for this task. The registers used to set the value being compared are known as the Output Compare Registers (OCR0A & OCR0B). When a value is compared to either of the OCR registers and matches, it will throw an interrupt (OCIE0A or OCIE0A), depending on the OCR used.



Wrap-up

We have discussed various features and modes of Timer0 on the ATmega324PB. Handling all these require plentiful knowledge of datasheet for the AVR, so always remember to keep handy. Your textbook also has good information on how to work with these timers and set them into the various modes they support. Be aware that some of the generic information contained in your textbook is different from the specific information this chip presents. Example: Timer0 has two control registers on the ATMega324PB, which is different than how your book presents it.

Procedure

1) For the first part of this lab, you are going to modify the *toggleProject* program from the tutorial in Lab 4. In the tutorial, you used the *_delay_ms* function to create a delay of 1000 ms that switched the AVR Simon Board's LEDs on and off. You will modify this program by removing the delay_ms function and instead creating your own routine that utilizes one of the AVR's timers to create a 1.5-second delay between switching the LEDs on and off.

2) For the second part of this lab, you will create a working clock that keeps hours, minutes, and seconds. You'll need to configure one of the AVR timers to trigger an ISR routine every time 1 second has passed so that the time can be correctly incremented. The time should be displayed on an LCD module, so you'll need to utilize the LCD class you created in Labs 5 and 6 to help you. The time should configurable by the user so that hours and minutes can be set by the use of push buttons on the AVR Simon Board. It would also be useful to reset the seconds back to zero if the minutes are changed so that the clock can be synced by the user to some clock source.

Deliverables

Both blocks of code you created in the lab will be submitted as a part of a lab report. Your solution to the clock problem should be thoroughly documented and demonstrated to your TA. Make sure your code also includes a documented configuration of how the hardware is connected (i.e. what pins are connected to what devices and what they are used for). A picture of the hardware setup/wiring would also be helpful. In the lab report, the questions/observations below should be included in the results or conclusions section of your report.

Questions/Observations

1. What is the difference in resolution between TIMER0 and TIMER1?
2. Describe the two flags bits (TOV and OCF) and what they can be utilized for?
3. What would you have to change in your clock program in order for the AVR to utilize an external clock source?

References

- Muhammad Ali Mazidi, Sarmad Naimi, and Sepehr Naimi. 2010. *AVR Microcontroller and Embedded Systems: Using Assembly and C (1st ed.)*. Prentice Hall Press, Upper Saddle River, NJ, USA.
- ATmega324PB datasheet
- LCD Module datasheet

- Hareendran, T.K. *ATmega8 Advanced Guide: AVR 8/16 Bit Timers/Counters – Tutorial #10 & Tutorial #11.* Accessed from https://www.electroschematics.com/9846/avr-8-16-bit-timers-counters/ Accessed on January 18, 2019.