Exercise 4: Timers, Input Capture and PWM

Overview

Microcontrollers usually have at least one timer. Timers are typically counters that are either incremented or decremented at a fixed frequency, which is often configurable, and that interrupt the processor when reaching zero. More sophisticated timers have a comparison logic to compare the counter against a specific value set by software. Although the number of hardware timers is limited, we can use a single hardware timer to implement an extensible set of software timers.

Timers in ATmega8515

ATmega8515 integrates two timers, namely Timer0 and Timer1. Timer0 is an 8 bit general purpose timer whereas Timer1 is 16 bit wide. Each timer has a counter register, a value which increases in every timer clock cycle. The timers can directly be clocked by the system clock, which provides the fastest operation. Alternatively, the timer clock can be scaled down by using the *prescaler*. In ATmega8515, the two timers share the same prescaler module, which includes four outputs running at 8, 64, 256, 1024 times slower frequency respectively. The timers can choose one of the four outputs freely. During this session we will only use Timer1, since it provides higher accuracy.

Output Compare Unit

Timer1 incorporates an output compare unit, which continuously compares the current timer counter (`TCNT1`) with the Output Compare Registers (`OCR1A` and `OCR1B`). When `TCNT1` equals `OCR1x`, the comparator signals a match, which will set the Output Compare Flag (`OCF1x`) and generates an interrupt if enabled. Note that `OCR1A` and `OCR1B` work independently and a separate ISR can be provided for them.

Exercise 4.1

a) Which types of interrupts can be generated by Timer1? How can they be enabled/disabled? What is the name of the ISR for output match interrupt A and B?

b) The timer in ATmega8515 can be configured to have several Waveform Generation Modes (WGM). In which registers are the Waveform Generation Mode (WGM) bits? For Exercise 4.1 to 4.4, we will use the WGM 4.

c) For WGM mode 4, which register controls the *TOP* (maximum) value of timer/counter? What will occur in case the timer counter reaches TOP?

d) Assume we choose the timer clock to be SysClock/256, where SysClock is 8 MHz. What value should we store in `OCR1A` in order to generate a sequence of timer interrupts with an interval of 1 second?

Exercise 4.2

a) Implement a `timer_init()` function that initializes the timer and generates an interrupt for a given real time interval (e.g., 1 ms). Use the output compare unit for the implementation.

b) Recall that in previous sessions, we used an imprecise `wait()` function. Develop an accurate `wait()` function with the same function signature using Timer1, which will let the program “sleep” for the given interval. Demonstrate your implementation by letting an LED blink.

c) Develop an application which implements 3 (or an extensible number of) software timers with different frequencies using the same hardware timer. The respective counter of the software timers should be updated within the ISR of the hardware timer. Demonstrate the different frequencies using LEDs.

Input Capture Unit

Timer1 incorporates an input capture unit, which can capture an external event and give it a time-stamp indicating the time of occurrence. This feature then allows calculation of frequency and duty cycle of external signals. The pin used to capture external events is called *Input Capture Pin* (`ICP1`). When an event is detected at `ICP1`, the current 16-bit value in the timer counter register (`TCNT1`) is copied to *Input Capture Register* (`ICR1`). This value then represents the time-stamp of the event. Note that `ICR1` value will be overwritten once another event happens. When enabled, an interrupt can be generated upon capturing an event. Find the following information from the manual.

Exercise 4.3

a) Which pin is used as `ICP1` in ATmega8515?

b) Input capture is edge sensitive. How to set it to be rising edge or falling edge sensitive?

Exercise 4.4

a) Implement an application to measure the time interval between a button is pressed and released. Display the measured result in the debug console.

b) Implement an application that will wait a random (see hints section) amount of time between 2 and 10 seconds and switch an LED on after the interval has passed. Then, measure the time until the user presses a button (i.e., the reaction time). Be sure to precisely measure the time by using the input capture unit. Output the reaction time in milliseconds to the debug console.

Hints

- When the time interval to be measured is large, a timer overflow may occur between two successive input capture events. Hence, to calculate the interval, you need to know not only the timer counter values at the beginning/end of the interval, but also how many timer overflows occured in between.
- Assuming the default setting says that input capture is falling-edge (button pressed) sensitive, you can change it to rising edge sensitive inside the ISR in order to detect the next rising edge (button release).
- It is actually hard to generate numbers that are somewhat random on a microcontroller. See `http://www.rn-wissen.de/index.php/Zufallszahlen_mit_avr-gcc` (German) for starting points. You might also want to have Timer0 continuously running and use its current value as random seed.

Pulse Width Modulation (PWM)

Pulse Width Modulation (PWM) is an efficient way to control analog circuits with digital outputs of microcontrollers. It is deployed for a wide variety of applications, among them motor drivers. As its name suggests, the control information is modulated in the width of the pulse. Figure 1 depicts three example PWM signals. Signal 1 a) is a PWM signal of 25% *duty cycle*, that is, the signal is *on* for one quarter of the time in one *period* and is *off* for the other three quarters. Figure 1 b) and 1 c) show respectively PWM signals of 50% and 75% duty cycle.

In the next exercise we will develop an application to control the brightness of LEDs. The idea is to use a PWM signal with different duty cycles to determine the amount of power conveyed to the LEDs. The brightness of the LEDs is therefore controlled. Naturally, the higher the duty cycle is, the higher the brightness is. The microcontroller provides several possibilities to generate PWM signals. For simplicity, we will use the *Fast PWM Mode*.

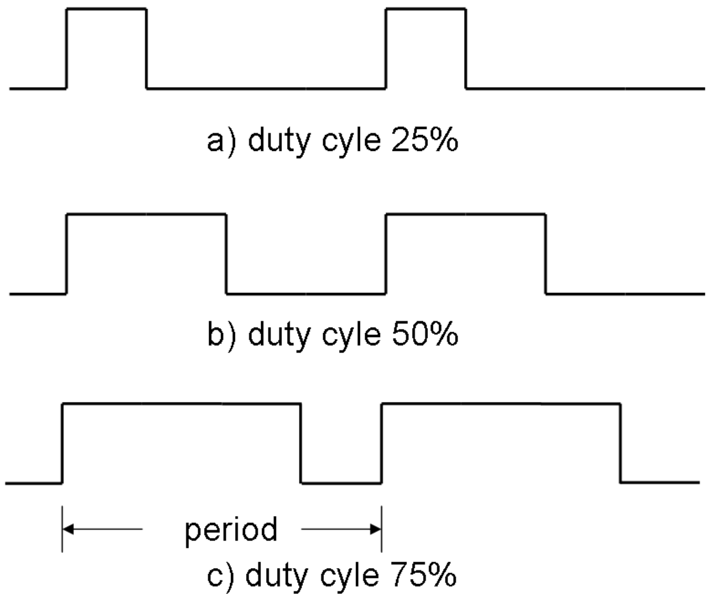

Table 1: Example PWM Signals

Exercise 4.5

a) Read the manual and make yourself clear how the fast PWM mode works. Find out the following information:

- Which pins can be used for PWM output? What needs to be done to enable PWM output on those pins (e.g., do we need to configure them as output pins)?
- If we use the WGM 15, how does the PWM generator work? Which register controls the period? How is the the duty cycle be controlled?
- What are so called inverted and non-inverted PWM signals? How can configure the microcontroller to generate such signals?

b) Develop a `pwm_init()` function that takes the period and duty cycle as input parameters and initialize the timer correspondingly to generate the PWM signal on PE2. Use the waveform generation mode mentioned above (WGM 15). You might want to use the LEDs to visualize the PWM output (see hints).

c) Develop a `pwm_set()` function that can be used to change the duty cycle of the PWM signal. It takes the new duty cycle value (between 0 and 100 percent) as input parameter. The period is not changed by this function.

d) Develop an application that waits for a command on the serial port and adjusts the duty cycle of PWM signal correspondingly. For example, the duty cycle of PWM signal is increased upon receiving a “`+`” character, and decreased upon receiving a “`-`”.

Hints

- When using LEDs for visualization, use a rather large period value (e.g., 1 second) for better visibility. Note that you need to set the timer prescaler to a high value in order to obtain such a large period.

Exercise 4.6

a) As you have seen in the previous steps, the blinking of the LEDs is visible when the period is large. Now try to reduce the period of the PWM signal to find out at which value (frequency and time interval) the blinking is no longer visible.

b) Set the period to a sufficient low value such that blinking of LEDs is not visible. Control the brightness of LEDs by configuring the duty cycle.

c) Optional: Do some research to find out why the brightness of LEDs is commonly controlled by using pulse width modulation instead of analog signals.