Silesian University of Technology
Faculty of Automatic Control,
Electronics and Computer Science

Embedded Systems Laboratory

Exercise 23

AVR Microcontrollers – part III

Timer interrupts

Mgr inż. Jarosław Paduch

Exercise description:

Requirements:

- Hardware: One computer PC

  Arduino MEGA 2560 board /there are two different versions !/

- Software: ATMEL Studio version 7.0 or above
- Knowledge of topics from the previous laboratory, how to create, compile and test program.

During the exercise, students are to:

Become familiar with the circuit diagram of ATMEGA 2560 Add-on board,

- Create the project in ATMEL Studio version 7.0 or above,

- Write the program in C (topic given by the supervisor),

- Compile and link the program,

- Observe program execution in the simulator, step by step.

- If necessary, correct the code.

- If the program in the simulator works correctly, then connect board Arduino MEGA256 to the serial port of PC.

- Prepare and upload the application to the device. (as in laboratory instruction part II)

- Check the correctness of the program on real hardware.

Goal

✓ The purpose of the exercise is to learn about ATMEGA 2560 microcontroller interrupts and timer/counters. Student during the laboratory should write a program in C language using 16-bit Timer1, program the device and test program behaviour. After the laboratory, each subsection should send a report.

Report

The report should contain:

- Proper Title page

- The topic of the program

- The source code of the program with comments!!!

-Two screenshots from the simulator that are confirming the duration of the timer interrupt cycle.

- Description and results of program testing

- Conclusions!!!

Report template is given on our website: http:\\www.zmitac.aei.polsl.pl

under sheet "Course MS" (microprocessor systems) / Report and project registration card templates.

BASICS

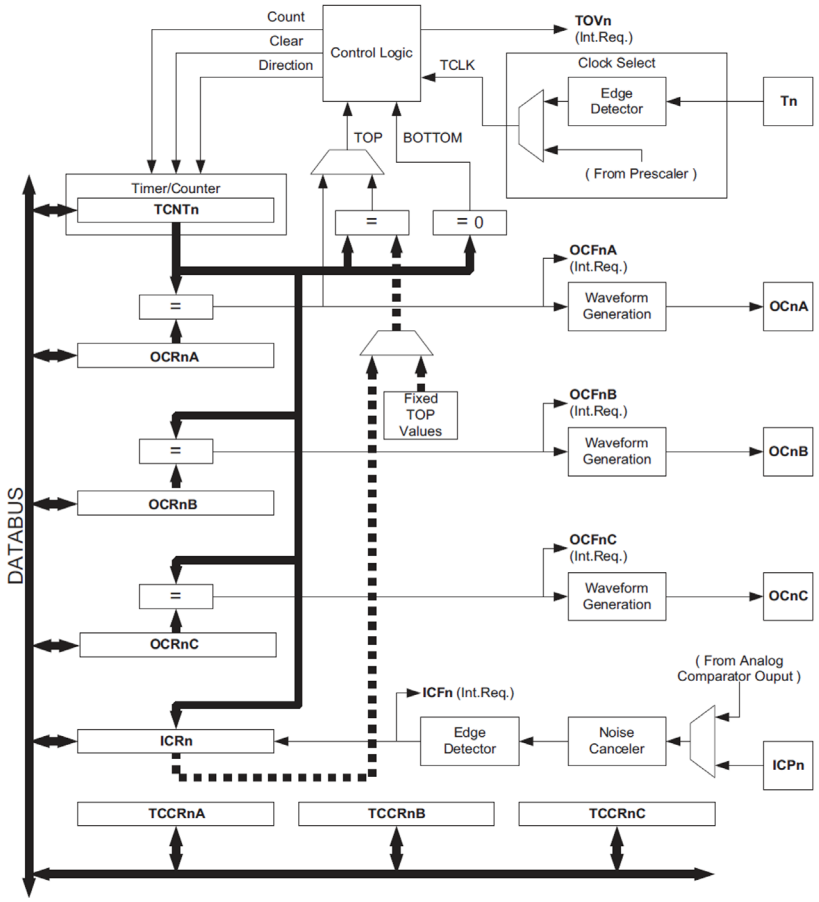
Timer/Counter 1

The 16-bit Timer/Counter unit allows accurate program execution timing (event management), wave generation, and signal timing measurement.

Features

- **True 16-bit Design (that is, allows 16-bit PWM)**
- **Three independent Output Compare Units**
- **Double Buffered Output Compare Registers**
- **One Input Capture Unit**
- **Input Capture Noise Canceler**
- **Clear Timer on Compare Match (Auto Reload)**
- **Glitch-free, Phase Correct Pulse Width Modulator (PWM)**
- **Variable PWM Period**
- **Frequency Generator**
- **External Event Counter**
- **Independent interrupt sources (TOV1, OCF1A, OCF1B, OCF1C, ICF1)**

Block diagram



Registers

The Timer/Counter (TCNTn), Output Compare Registers (OCRnA/B/C), and Input Capture Register (ICRn) are all 16-bit registers. Special procedures must be followed when accessing the 16-bit registers. The Timer/Counter Control Registers (TCCRnA/B/C) are 8-bit registers and have no CPU access restrictions. Interrupt requests (shorten as Int.Req.) signals are all visible in the Timer Interrupt Flag Register (TIFRn). All interrupts are individually masked with the Timer Interrupt Mask Register (TIMSKn). TIFRn and TIMSKn are not shown in the figure since these registers are shared by other timer units.

The Timer/Counter can be clocked internally, via the prescaler, or by an external clock source on the Tn pin. The Clock Select logic block controls which clock source and edge the Timer/Counter uses to increment (or decrement) its value. The Timer/Counter is inactive when no clock source is selected. The output from the clock select logic is referred to as the timer clock (clk_{Tn}).

The double buffered Output Compare Registers (OCRnA/B/C) are compared with the Timer/Counter value at all time. The result of the compare can be used by the Waveform Generator to generate a PWM or variable frequency output on the Output Compare pin (OCnA/B/C). The compare

match event will also set the Compare Match Flag (OCFnA/B/C) which can be used to generate an Output Compare interrupt request.

The Input Capture Register can capture the Timer/Counter value at a given external (edge triggered) event on either the Input Capture pin (ICPn) or the Analog Comparator pins. The Input Capture unit includes a digital filtering unit (Noise Canceler) for reducing the chance of capturing noise spikes.

The TOP value, or maximum Timer/Counter value, can in some modes of operation be defined by either the OCRnA Register, the ICRn Register, or by a set of fixed values. When using OCRnA as TOP value in a PWM mode, the OCRnA Register can not be used for generating a PWM output. However, the TOP value will, in this case, be double buffered, allowing the TOP value to be changed in run time. If a fixed TOP value is required, the ICRn Register can be used as an alternative, freeing the OCRnA to be used as PWM output.

Accessing 16-bit Registers

The TCNTn, OCRnA/B/C, and ICRn are 16-bit registers that can be accessed by the AVR CPU via the 8-bit data bus. The 16-bit register must be byte accessed using two reads or write operations. Each 16-bit timer has a single 8- bit register for temporary storing of the high byte of the 16-bit access. The same Temporary Register is shared between all 16-bit registers within each 16-bit timer. Accessing the low byte triggers the 16-bit read or write operation. When the low byte of a 16-bit register is written by the CPU, the high byte stored in the Temporary Register, and the low byte written are both copied into the 16-bit register in the same clock cycle. When the low byte of a 16- bit register is read by the CPU, the high byte of the 16-bit register is copied into the Temporary Register in the same clock cycle as the low byte is read.

Not all 16-bit accesses uses the Temporary Register for the high byte. Reading the OCRnA/B/C 16-bit registers does not involve using the Temporary Register.

To do a 16-bit write, the high byte must be written before the low byte. For a 16-bit read, the low byte must be read before the high byte.

The following code examples show how to access the 16-bit timer registers assuming that no interrupts updates the temporary register. The same principle can be used directly for accessing the OCRnA/B/C and ICRn Registers. Note that when using “C”, the compiler handles the 16-bit access. It is important to notice that accessing 16-bit registers are atomic operations. If an interrupt occurs between the two instructions accessing the 16-bit register, and the interrupt code updates the temporary register by accessing the same or any other of the 16-bit Timer Registers, then the result of the access outside the interrupt will be corrupted. Therefore, when both the main code and the interrupt code update the temporary register, the main code must disable the interrupts during the 16-bit access.

```
unsigned char sreg;
unsigned int i;
/* Save global interrupt flag */
sreg = SREG;
/* Disable interrupts */
__disable_interrupt();
/* Read TCNTn into i */
i = TCNT1;
/* or */
/* write I into  TCNTn  */
TCNT1 = i;
/* Restore global interrupt flag */
SREG = sreg;
```

Timer 1 Interrupts

Timer interrupts are an excellent way of having your AVR do something at a given interval. They can fire off and interrupt whatever else the AVR is doing making for very precise timing. They are one of the best ways to implement custom waveforms for things such as positioning robot servos, dimming LED's, and driving speakers at different frequencies. In our laboratory, we concentrate on using Timer 1 only.

For this example, assume that you have your PORTA jumpered to LEDs,

The Interrupt Header

To use the built-in interrupt features in C, you need to include the interrupt header like this:

#include <avr/interrupt.h>

The ISR keyword - C compiler uses the keyword ISR to denote an Interrupt Service Routine. We need to define *the ISR for timer1 overflow. You do it like this:*

// timer1 overflow

ISR(TIMER1_OVF_vect) {

// process the timer1 overflow here

}

There are also other interrupt vectors available for timer 1:

TIMER1 CAPT	Timer/Counter1 Capture Event
TIMER1 COMPA	Timer/Counter1 Compare Match A

TIMER1 COMPB	Timer/Counter1 Compare Match B
TIMER1 COMPC	Timer/Counter1 Compare Match C
TIMER1 OVF	Timer/Counter1 Overflow

Turning on the Timer Interrupt

For the interrupt to fire, you must enable it in the TIMSK register. For timer1 interrupts, use the following code in main:

// enable timer overflow interrupt for both Timer1

TIMSK= (1<<TOIE1);

Setup Your Timer1

```
// lets turn on 16 bit timer1 with prescaler  /1024
TCCR1B |= (1 << CS10) | (1 << CS12);
```

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (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 T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

$$Timer\ Count = \frac{Required\ Delay}{Clock\ Time\ Period} - 1$$

Enable Interrupts

This step is easy, call sei(); to turn on the global interrupt enable flag.

A Simple Example

Here is a simple example that turns on timer1. It accomplishes the following:

Sets up timer1 in a divide by 1024 mode, counting from 0 to 65,535

On timer1 interrupt, toggles PORTA bit 1.

You will see PORTA bit 1 blinking on and off every 8.3 seconds.

```
/*****************************************************************************
Includes
*****************************************************************************/
#include <avr/io.h>
#include <stdbool.h>
#include <avr/interrupt.h>
/*****************************************************************************
Interrupt Routines
*****************************************************************************/
// timer1 overflow
ISR(TIMER1_OVF_vect) {
   // XOR PORTA with 0x02 to toggle the LSB
   PORTA=PORTA ^ 0x02;
}
/*****************************************************************************
Main
*****************************************************************************/
int main( void ) {
   // Configure PORTA as output
   DDRA = 0xFF;
   PORTA = 0xFF;

   // enable timer overflow interrupt for Timer1
   TIMSK= (1<<TOIE1);
   // lets turn on 16 bit timer1 also with /1024
   TCCR1B |= (1 << CS10) | (1 << CS12);
   // enable interrupts
   sei();
   while(true) {
   }
}
```

Timer 1 Registers

TCNT1 Register

The **Timer/Counter1** – TCNT1 Register is as follows. It is 16 bits wide since the TIMER1 is a 16-bit register. **TCNT1H** represents the HIGH byte, whereas **TCNT1L** represents the LOW byte. The timer/counter value is stored in these bytes.

Bit	7	6	5	4	3	2	1	0	
	TCNT1[15:8]								TCNT1H
	TCNT1[7:0]								TCNT1L
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TCCR1A Register

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	COM1C1	COM1C0	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:6 – COMnA1:0: Compare Output Mode for Channel A**
- **Bit 5:4 – COMnB1:0: Compare Output Mode for Channel B**
- **Bit 3:2 – COMnC1:0: Compare Output Mode for Channel C**

Compare Output Mode, non-PWM

COMnA1 COMnB1 COMnC1	COMnA0 COMnB0 COMnC0	Description
0	0	Normal port operation, OCnA/OCnB/OCnC disconnected
0	1	Toggle OCnA/OCnB/OCnC on compare match
1	0	Clear OCnA/OCnB/OCnC on compare match (set output to low level)
1	1	Set OCnA/OCnB/OCnC on compare match (set output to high level)

Compare Output Mode, Fast PWM

COMnA1 COMnB1 COMnC1	COMnA0 COMnB0 COMnC0	Description
0	0	Normal port operation, OCnA/OCnB/OCnC disconnected
0	1	WGM13:0 = 14 or 15: Toggle OC1A on Compare Match, OC1B and OC1C disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B/OC1C disconnected
1	0	Clear OCnA/OCnB/OCnC on compare match, set OCnA/OCnB/OCnC at BOTTOM (non-inverting mode)
1	1	Set OCnA/OCnB/OCnC on compare match, clear OCnA/OCnB/OCnC at BOTTOM (inverting mode)

Compare Output Mode, Phase Correct and Phase and Frequency Correct PWM

COMnA1 COMnB1 COMnC1	COMnA0 COMnB0 COMnC0	Description
0	0	Normal port operation, OCnA/OCnB/OCnC disconnected
0	1	WGM13:0 =9 or 11: Toggle OC1A on Compare Match, OC1B and OC1C disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B/OC1C disconnected
1	0	Clear OCnA/OCnB/OCnC on compare match when up-counting Set OCnA/OCnB/OCnC on compare match when downcounting
1	1	Set OCnA/OCnB/OCnC on compare match when up-counting Clear OCnA/OCnB/OCnC on compare match when downcounting

TCCR1B Register

The **Timer/Counter1 Control Register B**– TCCR1B Register is as follows.

Bit	7	6	5	4	3	2	1	0	
	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Right now, only the highlighted bits concern us. The **bit 2:0 – CS12:10** are the **Clock Select Bits** of TIMER1. Their selection is as follows.

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (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 T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

TCCR1C Register

Bit	7	6	5	4	3	2	1	0	
(0x82)	FOC1A	FOC1B	FOC1C	–	–	–	–	–	TCCR1C
Read/Write	W	W	W	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – FOCnA: Force Output Compare for Channel A**
- **Bit 6 – FOCnB: Force Output Compare for Channel B**
- **Bit 5 – FOCnC: Force Output Compare for Channel C**

TIMSK1 Register

The **Timer/Counter Interrupt Mask Register** – TIMSK Register is as follows.

TIMSK1 – Timer/Counter 1 Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x6F)	–	–	ICIE1	–	OCIE1C	OCIE1B	OCIE1A	TOIE1	TIMSK1
Read/Write	R	R	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The bits associated with other timers are greyed out. **Bits not greyed out** correspond to TIMER1. **Bit 0 – TOIE1 – Timer/Counter1 Overflow Interrupt Enable** bit enables the overflow interrupt of TIMER1.

TIFR Register

The **Timer/Counter Interrupt Flag Register** – TIFR is as follows.

TIFR1 – Timer/Counter1 Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x16 (0x36)	–	–	ICF1	–	OCF1C	OCF1B	OCF1A	TOV1	TIFR1
Read/Write	R	R	R/W	R	R/W	R/W	R/W	R/W	

Initial Value	0	0	0	0	0	0	0	0

The greyed out bits correspond to different timers. Bits not greyed out are related to TIMER1. Of these, we are interested in **Bit 0 – TOV1 – Timer/Counter1 Overflow Flag**. This bit is set to '1' whenever the timer overflows. It is cleared (to zero) automatically as soon as the corresponding Interrupt Service Routine (ISR) is executed. Alternatively, if there is no ISR to execute, we can clear it by writing '1' to it.

OCR1AH and OCR1AL Register

Bit	7	6	5	4	3	2	1	0	
(0x89)	OCR1A[15:8]								OCR1AH
(0x88)	OCR1A[7:0]								OCR1AL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

ICR3H and ICR3L – Input Capture Register 3

Bit	7	6	5	4	3	2	1	0	
(0x97)	ICR3[15:8]								ICR3H
(0x96)	ICR3[7:0]								ICR3L
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Another sample with interrupts:

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

// global variable to count the number of overflows
volatile uint8_t tot_overflow;

// initialize timer, interrupt and variable
void timer1_init()
{
    // set up timer with prescaler = 8
    TCCR1B |= (1 << CS11);

    // initialize counter
    TCNT1 = 0;

    // enable overflow interrupt
    TIMSK |= (1 << TOIE1);

    // enable global interrupts
    sei();

    // initialize overflow counter variable
    tot_overflow = 0;
}

// TIMER1 overflow interrupt service routine
// called whenever TCNT1 overflows
ISR(TIMER1_OVF_vect)
{
    // keep track of the number of overflows
    tot_overflow++;

    // check for number of overflows here itself
    // 61 overflows = 2 seconds delay (approx.)
    if (tot_overflow >= 61) // NOTE: '>=' used instead of '=='
    {
        PORTC ^= (1 << 0);  // toggles the led
        // no timer reset required here as the timer
        // is reset every time it overflows

        tot_overflow = 0;   // reset overflow counter
    }
}

int main(void)
{
    // connect led to pin PC0
    DDRC |= (1 << 0);

    // initialize timer
    timer1_init();
```

```
    // loop forever
    while(1)
    {
        // do nothing
        // comparison is done in the ISR itself
    }
}
```

TODO:

ALL Program assumptions:

- The frequency of processor ATMEGA 2560 clocking is equal to 12 or 16 MHz

- Port A has eight buttons connected to ground with or without pull-up resistor,

- Port B bits 4 and 0 has two buttons connected to ground with a pull-up resistor,

- Port C has eight diodes LED connected to ground or power supply,

- Port L has eight diodes LED connected to ground,

- Port B[7-5] has three diodes LED connected to ground

- Somewhere in the program memory, there is a "ROMTAB" table with unspecified length, having word "0000" in the end

- Somewhere in the data memory, there is a "RAMTAB" table – with a length exactly 256 bytes

- All delays are obtained with the use of interrupts from the timers 1

a. Write a program which endlessly copies data from ROMTAB to RAMTAB. Coping should occur only if at least one button is pressed. If the program runs into the end of one of the tables, it should begin from the beginning of the table. There is a 200 millisecond delay between each byte. After each cycle, LED on PORTB.6 must switch on for 1 second.

b. Write a program which copies data from ROMTAB to RAMTAB. Coping should occur only if at least one button is pressed. If the program runs into the end of one of the tables, it should begin from the beginning of the table. There is a 500-millisecond delay between each byte. After each cycle, LED on PORTB.5 must switch on for 1 second.

c. Write a program which endlessly copies data from ROMTAB to PORT C (displaying data on LED's). Start copying after pressing any key on Port A. There is a 0.250-second delay between each byte and a 2-second delay between each full cycle.

d. Write a program which copies data endlessly from ROMTAB to PORT C (displaying data on LED's). Start copying after pressing any key on Port A. There is a 0.5-second delay between each byte and 4-second delay after each full cycle.

e. Write a program that gives an SOS signal on LED connected to PORTB.7 with 0.25s tic break.

BIBLIOGRAPHY

1. Jarosław Doliński - "Mikrokontrolery AVR w praktyce"
2. Andrzej Pawluczuk - "Sztuka programowania mikrokontrolerów. AVR - podstawy"
3. Piotr Górecki - "Mikrokontrolery dla początkujących"
4. http://www.itee.uq.edu.au/~cse/_atmel/AVR_Studio_Tutorial/
5. http://winavr.scienceprog.com/avr-gcc-tutorial/
6. http://imakeprojects.com/Projects/avr-tutorial/
7. http://www.atmel.com/products/avr/
8. http://www.avrfreaks.net/
9. http://pl.wikipedia.org/wiki/Atmel_AVR
10. http://www.elportal.pl/ea/asm_avr.html
11. Jakub Jankowski, Marcin Kania, Mariusz Macheta, Łukasz Strzelecki – Opracowanie na temat mikrokontrolery AVR”
12. http://maxembedded.com/2011/06/avr-timers-timer1/

SCHEMATICS