Timers as counters

for Basics of Microprocessor technology

Alexey Tukalo,
EFA12SF,
Information Technology,
Savonia University of Applied Sciences

May 6, 2015

Contents

1 The main features of counter circuits

There are two types of counters in ATmega boards.

1.1 8-bit Timer/Counter

- Two Independent Output Compare Units
- Double Buffered Output Compare Registers
- Clear Timer on Compare Match (Auto Reload)
- Glitch Free, Phase Correct Pulse Width Modulator (PWM)
- Variable PWM Period
- Frequency Generator
- Three Independent Interrupt Sources (TOV0, OCF0A, and OCF0

8-bit Timer/Counter module is used for accurate event management and wave generation. It has two independent Output Compare Units, and with Phase Correct Pulse Width Modulator support.

1.2 16-bit Timer/Counter

- 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
- Twenty independent interrupt sources (TOV1, OCF1A, OCF1B, OCF1C, ICF1, TOV3, OCF3A, OCF3B, OCF3C, ICF3, TOV4, OCF4A, OCF4B, OCF4C, ICF4, TOV5, OCF5A, OCF5B, OCF5C and ICF5)

16-bit Timer/Counter module is used for very accurate event management, signal timing measurement and wave generation.

1.3 Difference

The main difference is that 16-bit counter is much more accurate, but the precision is not always required that's why 8-bit counter is also used widely. For example it is better to use 16-bit timer to measure time and 8-bit is better for some event which have to be repeated after same period of time, but the period of time should not be as accurate as watch.

2 The operating principle

Timers work by incrementing a counter variable, also known as a counter register. The counter register can count to a certain value, depending on its size. The timer increments this counter one step at a time until it reaches its maximum value, at which point the counter overflows, and resets back to zero. The timer normally sets a flag bit to let you know an overflow has occurred. You can check this flag manually, or you can also have the timer trigger an interrupt as soon as the flag is set. Like any other interrupt, you can specify an Interrupt Service Routine (ISR) to run code of your choice when the timer overflows. The ISR will reset the overflow flag behind the scenes, so using interrupts is usually your best option for simplicity and speed.

In order to increment the counter value at regular intervals, the timer must have access to a clock source. The clock source generates a consistent repeating signal. Every time the timer detects this signal, it increases its counter by one.

Timers can run asynchronous to the main AVR core it means timers are totally independent of CPU. A timer is usually specified by the maximum value to which it can count called MAX beyond which it overflows and resets to zero is called BOTTOM. The speed of counting can be controlled by varying the speed of clock input to it.

Table 1: BOTTOM, MAX, TOP values explanation

BOTTOM	counter is equal to 0x00
MAX	counter is equal to 0xFF
TOP	the value is dependent on the mode operation. It can be equal to MAX or to the value stored OCRnA Register.

Let us first have a look at the basics of how a timer works. There are basically two types of timers, 8 bit(counter 0) and 16 bit(counter 1) timers. The only major difference between them is that they have different maximum values up to which they can count. As you may already know, an 8 bit binary number can have a maximum value of 255 and a 16 bit timer can go up to 65535. The 16 bit register stores the value of the count using two 8 bit registers. The registers are explained on the table 2.

Table 2: Timer0 and Timer1 registers

Timer0	Description	Timer1	Description
TCCR0A	Timer/Counter Control Register A	TCNT1	16-bit counter register
TCCR0B	Timer/Counter Control Register B	TCCR1A	Mode of operation and other settings
TCNT0	Timer/Counter Register	TCCR1B	Mode of operation, prescaler and other settings
OCR0A	Output Compare Register A	OCR1A	16-bit Compare Register A
OCR0B	Output Compare Register B	OCR1B	16 bit Compare Register B
TIMSK0	Timer/Counter Interrupt Mask Register	TIMSK	Interrupt Mask Register
TIFR0	Timer/Counter Interrupt Flag Register	TIFR0	Timer/Counter Interrupt Flag Register

2.1 Timer Modes

Timers are usually used in one of the following modes:

- Normal - As we know a timer is an 8 or 16 bit register that keeps on increasing its value, so one of the basic condition is when timer register overflows, means it counts reaches to its max value (255 for 8bit and 65535 for 16bit timers) and gets reset back to 0. At this situation timer can issue an interrupt.
- Clear on Timer Capture - Instead of counting until an overflow occurs, the timer compares its count to a value that was previously stored in a register. When the count matches that value, the timer can either set a flag or trigger an interrupt, just like the overflow case.
- Fast Pulse Width Modulator
- Phase correct Pulse Width Modulator

2.2 Prescaler

A technique to derive a lower frequency from F_CPU, without effecting actual F_CPU to run timer is called prescaler. In other words it is a mechanism for generating clock for timer by F_CPU clock. Atmega series of microcontrollers are available in several frequencies such as 1MHz, 8MHz, and 12MHz etc.

Prescaler allows us to divide up the incoming clock signal by power of 2. It reduces the resolution which means that the accuracy has decreased but giving us the longer timer range.

Prescalar can be set to produce the following clocks:

- No Clock Timer Stop
- No prescaling Clock frequency = F_CPU
- $F_CPU/8$
- $F_CPU/64$
- $F_CPU/256$
- $F_CPU/1024$

2.3 Output compare unit

Output compare unit is used to perform output compare matches. When a output compare match interrupt occurs, the OCFxy flag will be set in the interrupt flag register TIFRx . When the output compare interrupt enable bit OCIExy in the interrupt mask register TIMSKx is set, the output compare match interrupt service ISR(TIMERx_COMPy_vect) routine will be called.

2.4 Registers

There are four Registers: TCNTn, OCRn, TCCRn,

- TCNTn(timer/counter register):
  - the counter itself
  - holds the present value of count
- OCRn(output compare register):
  - this register is always compared against TCNTn
  - The double buffered Output Compare Registers (OCRnA and OCRnB)
- TCCRn(timer/counter n control register):
  - determines the mode of operation

3 The measurement accuracy and frequencies

Accuracy of timer is dependent from frequency, amount of bits and prescaler. The timer became more accurate with higher frequency and lower prescaler, of course higher amount of bits makes the counter more precise also.

4 Real time clock

The most accurate way to make real time clock for AVR is 16-bit counter. The time between interrupts must be calculated in an according with prescaler and CPU speed, the time should be around several millisecond. Low prescaler will make timer more accurate. The interrupt handler have to plus the amount of milliseconds to the correspondent variable and if the amount inside became more than 999, it should plus one second and set the milliseconds to zero, we also need to check that minutes are not more than 59, set it to zero too and plus one hour if it so, and of course our hours should be less than 24 or we have to force it to zero.

5 Measuring the speed of a program loop

The easiest way to measure the program loop speed is:

1. Start real time clock timer right before the loop
2. In the end of the loop check the value of the timer, and set it to zero.

It is possible to make the loop run several times and sum the values, after that you can get more accurate result by dividing of the number by amount of loops. Some times program loop can be even smaller than resolution of your real clock, in this case you can make the timer run several times and only after that check the duration of the time, divide it by number of loops.

6 Measuring the signal frequency

Frequencies can be measured via using of internal and external counters together. You have to set an internal counter as a real time clock, and use it to measure some period of time for example one second, an external counter can be used to calculate the number of pulses in incoming during that time. It is possible to calculate frequency if we know how many pulses were during some determined period of time.

7 Measuring the length of a pulse

The pulse can be measured in this way:

1. The program have to wait until the start of the pulse
2. After that it must start timer, save the value of counter as reference or simply set counter to zero value
3. Wait until the end of the pulse
4. Analyse the value of counter in according with step 2

8 Programming Tasks

8.1 The first task

Listing 1: L5_12.c

```
#include "L4_123.h"
#include "L5_12.h"

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

```
unsigned long long int time=0;

unsigned int hours  = 0;
unsigned int minutes= 0;
unsigned int seconds= 0;

ISR(TIMER1_OVF_vect){
        time+=8;
        }

void BeginTimer(){
        TCCR1A = 0;          // set entire TCCR1A register to 0
        TIMSK1=1<<TOIE1;
        TCNT1=0;
        TCCR1B=1;
        sei();

}
unsigned RunnedTime(){
        return time;
}
```

Listing 2: L5_12.h

```
#ifndef L5_12_H_
#define L5_12_H_

void BeginTimer();
unsigned RunnedTime();

#endif
```

The program loop contains nothing that's why it was smaller than resolution of my timer. I solved the problem by measuring of the duration of 1000 of loop and dividing the time by 1000. I also repeated the experiment with bigger amount of loops(10000, 100000 and so on), the values change correspondent.

8	72	728	7336
8	72	736	7336
8	72	728	7336
0	72	736	7336
8	72	736	7336
8	72	736	7336
8	72	736	7344
8	72	728	7336
8	72	736	7336
8	72	728	7336
8	72	736	7336
8	72	736	7336
8	72	736	7336
8	72	736	7344
8	72	728	7336
8	72	736	7336
8	72	728	7336
0	72	736	7336
8	72	736	7336
8	72	736	7336
8	72	736	7336
8	72	728	7336
0	72	736	7336

Figure 1: Duration of program loop form 1000 to 1000000 repeats

The duration of one loop is $8ms/1000 = 8\mu s$.

Listing 3: main.c

```
//std libs
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <stdbool.h>

//my libs
#include "L4_123.h"
#include "L5_12.h"

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


#define number_of_times 100000

int main(void){
                char string[10];
                unsigned long long int i,prevTime;

        SetLineParameters( 0, 9600, EVEN_PARITY, BIT_FORMAT_8bit, STOP_BIT_2bit );

        BeginTimer();

        //program loop
        while(true){
                i++;
                if(i>number_of_times){
                        sprintf(string,"\n\r%d",RunnedTime()-prevTime);
// show time from previous n loops
                        WriteString(0,string);
                        prevTime=RunnedTime();
                        i=0;
                }
        }
}
```

8.2 The second task

I have modified my code for the second task:

Listing 4: L5_12.c

```
#include "L4_123.h"
#include "L5_12.h"

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

unsigned long long int time=0;
```

```
unsigned int hours  = 0;
unsigned int minutes= 0;
unsigned int seconds= 0;

void watch(void){
        static char string[10];
        sprintf(string,"\r%d:%d:%d",hours,minutes,seconds);
        WriteString(0,string);
        if(time>999){
                seconds++;
                time=0;
                if(seconds>59){
                        minutes++;
                        seconds=0;
                        if(minutes>59){
                                hours++;
                                minutes=0;
                                WriteString(0,"\r          ");
                        }
                        if(hours>23)
                        hours=0;
                        WriteString(0,"\r          ");
                        }
}
}
ISR(TIMER1_OVF_vect){
        time+=8;
        watch();
}
void BeginTimer(){
        TCCR1A = 0;         // set entire TCCR1A register to 0
        TIMSK1=1<<TOIE1;
        TCNT1=0;
        TCCR1B=1;
        sei();

}
unsigned RunnedTime(){
        return time;
}

//time setters
void set_minutes(int m){
        if(m>0 && m<60)
                minutes=m;
        else
                minutes=0;
}
void set_hours(int h){
        if(h>0 && h<24)
        hours=h;
        else
        hours=0;
```

```
}
```

Listing 5: L5_12.h

```
#ifndef L5_12_H_
#define L5_12_H_

void BeginTimer();
unsigned RunnedTime();
void set_minutes(int m);
void set_hours(int h);
void watch(void);

#endif
```

Listing 6: main.c

```
//std libs
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <stdbool.h>
//my libs
#include "L4_123.h"
#include "L5_12.h"
//avr libs
#include <avr/io.h>
#include <avr/interrupt.h>

void main(void){
        char string[10];

        SetLineParameters( 0, 9600, EVEN_PARITY, BIT_FORMAT_8bit, STOP_BIT_2bit );
        WriteString(0,"\033[2J");
        WriteString(0,"\rType hours:");
        strcpy(string,ReadString(0));
        set_hours(atoi(string));
        WriteString(0,string);
        WriteString(0,"\n\rType minutes:");
        strcpy(string,ReadString(0));
        set_minutes(atoi(string));
        WriteString(0,string);
        WriteString(0,"\n");
        BeginTimer();
        watch();
}
```

8.3 The third task

Not done.

8.4 The fourth task

We used the original L5_12.c and L5_12.h files from listings 1 and 2 for this task.

Listing 7: L5_4.c

```
#include "L4_123.h"
#include "L5_12.h"
#include "L5_4.h"
#include <avr/io.h>
#include <avr/interrupt.h>

int GetPulse(unsigned switchNumber, unsigned seconds){
        unsigned int bCounter=1;
        int i,switchPin=1;

        for(i=1;i<switchNumber;i++)
                switchPin*=2;

        while(PINA & switchPin);
        while(!(PINA & switchPin));

        BeginTimer();
        while(RunnedTime()<seconds*1000)

        if(!(PINA & switchPin)) // a key is pressed
        {
                bCounter++;
                while(!(PINA & switchPin)); // wait until released
        }
        cli();
        return ~bCounter;
}
```

8.5 The firth task

We used the original L5_12.c and L5_12.h files from listings 1 and 2 for this task.

Listing 8: L5_5.c

```
#include "L4_123.h"
#include "L5_12.h"
#include "L5_5.h"
#include <avr/io.h>
#include <avr/interrupt.h>

long ElapsedTime(unsigned switchNumber){
        int i,switchPin=1;
        for(i=1;i<switchNumber;i++)
        switchPin*=2;
        while(PINA & switchPin);
        BeginTimer();
        while(!(PINA & switchPin)); // wait until released
        i=RunnedTime();
        cli();

        return i;
}
```