Timer/Counter Basics

Introduction

All AVR parts feature one or more general purpose Timer/Counters. All AVRs have at
least one 8-bit Counter. Some have more, and some have 16-bit Counters as well. All of
these can be used as a Timer with an internal clock base, or as a Counter with an exter-
nal pin connection which triggers the counting. Some devices allows the Timer to use an
external 32.768 Hz crystal as clock base.

This document is intended to enable the reader to set up and use the Timer/Counters of
the AT90S8515 device. It's easy to expand the example to another AVR with a little help
from that device’s datasheet.

Overview

The AT90S8515 is a very common all-round AVR. This part features one 8-bit
Timer/Counter (T/C) called Timer/Counter0, and one 16-bit T/C called Timer/Counter1,
each with separate prescalers. The 16-bit T/C furthermore features Compare and Cap-
ture Modes.

The following might clarify some questions about the description so far:
• The prescaler permits the clock source for the Timer/Counter to be a fraction (1/1,
  1/8, 1/64, 1/256, or 1/1024) of the internal clock, enabling the timers to be set to a
  number of modes. When the prescaler is set, the T/C is in Timer mode.
• The external input pin can trig the Counter on rising or falling edge. When enabled,
  the T/C is in Counter mode.

The Timer/Counters are affected by the following registers:
• The Timer/Counter Interrupt Mask Register (TIMSK): In this register, the various
  interrupts of both Timers are enabled or disabled.
• The Timer/Counter Interrupt Flag Register (TIFR): This register holds the Interrupt
  Flags for the Timer/Counter1 interrupts. Each bit in the TIFR has a corresponding bit
  in TIMSK
• The Timer/Counter Control Register(s) (TCCR0 / TCCR1A/B): These registers
  contain the prescaler value, external pin edge detection, Output Compare and Input
  Capture modes (if available) as well as PWM (Pulse Width Modulation) modes.
  PWM is not within the scope of this document.
• The Output Compare Registers (OCR1A/B – Timer/Counter1 only) contain the data
to be continuously compared with Timer/Counter1 when in Compare mode. A
  Compare Match will occur if Timer/Counter1 counts to the OCR value. This will
  invoke a compare interrupt if enabled.
The Input Capture Register (ICR1 – Timer/Counter1 only): When the rising or falling edge (according to settings) of the signal at the Input Capture Pin – ICP – is detected, the current value of the Timer/Counter1 is transferred to this register. At the same time, the Input Capture Flag – ICF1 – is set (one). This will invoke a capture interrupt if enabled.

Examples

The following source code is an example, created to enable the reader to perform normal setup of the Timer/Counters, as well as basic operations. Both the 8-bit and the 16-bit timers of the AT90S8515 are used, as well as the Timer0 Overflow Interrupt and the Timer1 Compare A Interrupt with Timer Reset. The result is a (poor due to inadequate oscillating frequency) clock which flashes the four inner LEDs of an STK500 every second, toggles the two second-to-outer LEDs every half minute, and the outer LEDs every minute.

The frequency is based on the STK500's on-board driver Oscillator of 3.69 MHz to make the example more available. This frequency is prescaled with a 256 ratio, software scaled to a ratio of 100 and then converted to hex format, losing accuracy. The result is rather good on the minute accuracy, but poor on seconds. To make an actual clock, a watch crystal should be used.

The following approach has been used:

\[
\begin{align*}
3690000 \text{ Hz} / 256 &= 14414,0625 \\
14414,0625 \times 60 &= 864843,75 / 100 = 8648,4375 \sim 0x21C8 \text{ (minute)} \\
14414,0625 / 100 &= 144,140625 \sim 0x90 \text{ (second)}
\end{align*}
\]

Please read the comments in the C code, as they are applicable to the assembly as well.

C Code

```c
//Compiler: IAR EW A90 1.51b
//include definitions for our part
#include <io8515.h>

//include intrinsic commands
#include <ina90.h>

#define COUNT 100
#define COUNT_HALF 50
int minutecounter, secondcounter;

interrupt [TIMER1_COMPA_vect] void min(void)
{
    minutecounter--;
    if(minutecounter == 0){
        PORTB = PORTB ^ 0x81;//toggle outside LEDs
        minutecounter = COUNT;
    }
    else if(minutecounter == COUNT_HALF){
        PORTB = PORTB ^ 0x42;//toggle next LEDs
    }
}
```
interrupt [TIMER0_OVF0_vect] void second(void)
{
  secondcounter--;  
  if(secondcounter == 0){
    PORTB = ~PORTB ^ 0xc3;/*toggle inside LEDs while keeping outside*/
    TCNT0 = 0x100-0x90; //reload timer 0
    secondcounter = COUNT_HALF; //twice a sec
  }
}

void initialize(void)
{
  secondcounter = COUNT_HALF;
  minutecounter = COUNT;

  DDRB = 0xff;        //port B all outputs

  //set the timer0 prescaler to CK/256
  TCCR0 |= (1<<CS02);

  /*load the nearest-to-one-second value into the timer0*/
  TCNT0 = 0x100-0x90;

  /*clear timer/counter1 on compare matchA and set the prescaler to CK/256*/
  TCCR1B = (1<<CTC1)|(1<<CS12);

  //Set the compare register to "one minute"
  OCR1A = 0x21c8;

  /*enable the compare match1 interrupt and the timer/counter0 overflow interrupt*/
  TIMSK |= (1<<OCIE1A)|(1<<TOIE0);

  _SEI();       //global interrupt enable
}

void main(void)
{
  initialize();
  while(1)
  ;            //eternal loop
Assembly Code

; Assembler: AVR Studio 3.53
; include bit definitions for the AT90S8515
.include "8515def.inc"

.def temp = r16 ; temporary data1
.def temp2 = r17 ; temporary data2
.def minutecounter = r18
.def secondcounter = r19
.equ COUNT = 100
.equ COUNT_HALF = 50

.org $0000
   rjmp start ; reset handler

.org OC1Aaddr ; definitions in the
   rjmp minute ; 8515 include file

.org OVF0addr
   rjmp second

; Output Compare1A Interrupt (minute)
minute:
   dec minutecounter
   cpi minutecounter, 0
   breq toggle_outside
   cpi minutecounter, COUNT_HALF
   breq toggle_next
   rjmp minute_return

toggle_outside:
   ldi temp, 0x81
   in temp2, PORTB
   eor temp2, temp
   out PORTB, temp2
   ldi minutecounter, COUNT
   rjmp minute_return

toggle_next:
   ldi temp, 0x42
   in temp2, PORTB
   eor temp2, temp
   out PORTB, temp2

minute_return:
   reti

; Timer0 Overflow Interrupt (second)
second:
  dec secondcounter
  cpi secondcounter, 0
  brne return_second
  ldi temp, 0xff
  in temp2, PORTB
  eor temp2, temp
  ldi temp, 0xc3
  eor temp2, temp
  out PORTB, temp2
  ldi secondcounter, COUNT_HALF

return_second:
  reti

initialize:
  ldi secondcounter, COUNT_HALF
  ldi minutecounter, COUNT
  ldi temp, 0xff
  out DDRB, temp
  ldi temp, (1<<CS02)
  out TCCR0, temp
  ldi temp, 0x100
  subi temp, 0x90
  out TCNT0, temp
  ldi temp, (1<<CTC1)|(1<<CS12)
  out TCCR1B, temp
  ldi temp, 0x21
  ldi temp2, 0xc8
  out OCR1AH, temp
  out OCR1AL, temp2
  ldi temp, (1<<OCIE1A)|(1<<TOIE0)
  out TIMSK, temp
  sei
  ret

start:
  ldi temp,low(RAMEND)
  out SPL,temp
  ldi temp,high(RAMEND)
  out SPH,temp ;init Stack Pointer
  rcall initialize
  forever: rjmp forever ;eternal loop