Judah Ben-Eliezer

112352727
11/11/2020

Lab 10:

Timer/Counter TCA0 Overflow Interrupt

Theory:

To enable use of the TCA0 timer/counter for counting, first an origin statement needs to be made to connect TCA0_OVF_vect, the overflow vector for TCA0, to an interrupt service routine (ISR).
Next, in the configuration of the program, select normal mode for the timer/counter by writing TCA_SINGLE_WGMODE_gc, or group configuration for normal mode, to TCA0_SINGLE_CTRLB, or the register that controls the counter mode. This will make the counter run as a 16 bit binary counter.
Two enable overflow interrupt, set the TCA0_SINGLE_INTCTRL, or the interrupt control in single operation mode, to 0x01.
Next the period for the counter has to be set. For this, write the low byte of the desired period to TCA0_SINGLE_PER and the high byte to TCA0_SINGLE_PER + 1. The value in the period register determines when the counter will hit TOP, ie. when it will create an overflow interrupt.
Next the clock divisor needs to be set. The divisor is the bits 4 to 1 of the register TCA0_SINGLE_CTRLA. Set this to the desired divisor.
While working with TCA0_SINGLE_CTRLA, set bit 0 to 1 to start timer
Then, enable global interrupts.
Inside the ISR, first make sure to push to the stack any registers you plan on using in the ISR, as well as the status register.
Perform any desired periodic operations, and then clear the overflow interrupt flags
To clear the flags, write a 1 to bit 0 of the TCA0_INTFLAGS register
Make sure to pop off the stack any registers that were stored there, as well as restore the status register.

To sum up, the TCA0_SINGLE_CTRLB register determines the operation mode, the TCA0_SINGLE_INTCTRL register determines the interrupts to be used, the TCA0_SINGLE_PER register determines the period of the counter, and the TCA0_SINGLE_CTRLA register determines the clock divisor, as well as enables the timer.
The origin statement directs the program to the ISR whenever an interrupt occurs, and the ISR then clears the flag once it has performed the desired periodic operation.

Using the example of a simple periodic waveform, the ISR toggles an output by xoring the value off the pin with a bitmask with a 1 in the position of the output pin. The value written into the PER register should be half of the desired period, as the program toggles the pin. The CTRLA register can be used to slow the clock if needed.

```

AVRASM ver. 2.2.7  E:\ESE_280\$MyDocuments$\Atmel Studio\7.0\lab_10\intr_driven_bit_toggle\intr_driven_bit_toggle\main.asm Tue Nov 10 18:21:32 2020

E:\ESE_280\$MyDocuments$\Atmel Studio\7.0\lab_10\intr_driven_bit_toggle\intr_driven_bit_toggle\main.asm(9): Including file 'C:/Program Files (x86)\Atmel\Studio\7.0\Packs\atmel\ATmega_DFP\1.3.300\avrasm\inc\m4809def.inc'
E:\ESE_280\$MyDocuments$\Atmel Studio\7.0\lab_10\intr_driven_bit_toggle\intr_driven_bit_toggle\main.asm(9): Including file 'C:/Program Files (x86)\Atmel\Studio\7.0\Packs\atmel\ATmega_DFP\1.3.300\avrasm\inc\m4809def.inc'


                                 ; intr_driven_bit_toggle.asm
                                 ;
                                 ; Created: 11/10/2020 4:50:58 PM
                                 ; Author : Judah Ben-Eliezer
                                 ;

                                 .list


                                 .equ PERIOD_EXAMPLE_VALUE = 325        ;40.06 Hz

                                 reset:
000000 940c 0010                    jmp start

                                 .org TCA0_OVF_vect
00000e 940c 0023                    jmp toggle_pin_ISR

                                 start:
000010 9a81                          sbi VPORTE_DIR, 1

                                    ;configure TCA0
000011 e000                         ldi r16, TCA_SINGLE_WGMODE_NORMAL_gc      ;WGMODE normal
000012 9300 0a01                    sts TCA0_SINGLE_CTRLB, r16

                                    ;enable overflow interrupt
000014 e001                         ldi r16, TCA_SINGLE_OVF_bm
000015 9300 0a0a                    sts TCA0_SINGLE_INTCTRL, r16

                                    ;load period low byte then high byte
000017 e405                         ldi r16, LOW(PERIOD_EXAMPLE_VALUE)
000018 9300 0a26                    sts TCA0_SINGLE_PER, r16
00001a e001                         ldi r16, HIGH(PERIOD_EXAMPLE_VALUE)
00001b 9300 0a27                    sts TCA0_SINGLE_PER + 1, r16

```

```
                                    ;set clock and start timer
00001d e00d                         ldi r16, TCA_SINGLE_CLKSEL_DIV256_gc | TCA_SINGLE_ENABLE_bm
00001e 9300 0a00                    sts TCA0_SINGLE_CTRLA, r16

000020 9478                         sei     ;enable global interrupts

                                 main_loop:
000021 0000                         nop
000022 cffe                         rjmp main_loop

                                 toggle_pin_ISR:
000023 930f                         push r16
000024 b70f                         in r16, CPU_SREG
000025 930f                         push r16
000026 931f                         push r17

000027 e012                         ldi r17, $02    ;toggle PE1
000028 b301                         in r16, VPORTE_OUT
000029 2701                         eor r16, r17
00002a bb01                         out VPORTE_OUT, r16

00002b e001                         ldi r16, TCA_SINGLE_OVF_bm  ;clear OVF flag
00002c 9300 0a0b                    sts TCA0_SINGLE_INTFLAGS, r16

00002e 911f                         pop r17
00002f 910f                         pop r16
000030 bf0f                         out CPU_SREG, r16
000031 910f                         pop r16



RESOURCE USE INFORMATION
------------------------

Notice:
The register and instruction counts are symbol table hit counts,
and hence implicitly used resources are not counted, eg, the
'lpm' instruction without operands implicitly uses r0 and z,
none of which are counted.

x,y,z are separate entities in the symbol table and are
counted separately from r26..r31 here.

.dseg memory usage only counts static data declared with .byte

"ATmega4809" register use summary:
x  :   0 y  :   0 z  :   0 r0 :   0 r1 :   0 r2 :   0 r3 :   0 r4 :   0
r5 :   0 r6 :   0 r7 :   0 r8 :   0 r9 :   0 r10:   0 r11:   0 r12:   0
```

```
r13:   0 r14:   0 r15:   0 r16:  21 r17:   4 r18:   0 r19:   0 r20:   0
r21:   0 r22:   0 r23:   0 r24:   0 r25:   0 r26:   0 r27:   0 r28:   0
r29:   0 r30:   0 r31:   0
Registers used: 2 out of 35 (5.7%)

"ATmega4809" instruction use summary:
.lds  :   0 .sts  :   0 adc   :   0 add   :   0 adiw  :   0 and   :   0
andi  :   0 asr   :   0 bclr  :   0 bld   :   0 brbc  :   0 brbs  :   0
brcc  :   0 brcs  :   0 break :   0 breq  :   0 brge  :   0 brhc  :   0
brhs  :   0 brid  :   0 brie  :   0 brlo  :   0 brlt  :   0 brmi  :   0
brne  :   0 brpl  :   0 brsh  :   0 brtc  :   0 brts  :   0 brvc  :   0
brvs  :   0 bset  :   0 bst   :   0 call  :   0 cbi   :   0 cbr   :   0
clc   :   0 clh   :   0 cli   :   0 cln   :   0 clr   :   0 cls   :   0
clt   :   0 clv   :   0 clz   :   0 com   :   0 cp    :   0 cpc   :   0
cpi   :   0 cpse  :   0 dec   :   0 des   :   0 eor   :   1 fmul  :   0
fmuls :   0 fmulsu:   0 icall :   0 ijmp  :   0 in    :   2 inc   :   0
jmp   :   2 ld    :   0 ldd   :   0 ldi   :   7 lds   :   0 lpm   :   0
lsl   :   0 lsr   :   0 mov   :   0 movw  :   0 mul   :   0 muls  :   0
mulsu :   0 neg   :   0 nop   :   1 or    :   0 ori   :   0 out   :   2
pop   :   3 push  :   3 rcall :   0 ret   :   0 reti  :   1 rjmp  :   1
rol   :   0 ror   :   0 sbc   :   0 sbci  :   0 sbi   :   1 sbic  :   0
sbis  :   0 sbiw  :   0 sbr   :   0 sbrc  :   0 sbrs  :   0 sec   :   0
seh   :   0 sei   :   1 sen   :   0 ser   :   0 ses   :   0 set   :   0
sev   :   0 sez   :   0 sleep :   0 spm   :   0 st    :   0 std   :   0
sts   :   6 sub   :   0 subi  :   0 swap  :   0 tst   :   0 wdr   :   0

Instructions used: 13 out of 114 (11.4%)

"ATmega4809" memory use summary [bytes]:
Segment   Begin    End      Code   Data   Used    Size   Use%
---------------------------------------------------------------
[.cseg] 0x000000 0x000066     78      0     78   49152   0.2%
[.dseg] 0x002800 0x002800      0      0      0    6144   0.0%
[.eseg] 0x000000 0x000000      0      0      0     256   0.0%

Assembly complete, 0 errors, 0 warnings

```