1 Page

Prac 8 4209453 Justin Mankinelli

Prep

Task 1 — Software Delay

① Write an ASM program for the AVR8515 running at 4MHz to toggle PORTB[0] at 1KHz

② Simulation

A) Set breakpoint at line where bit is toggled

B) Check Stop Watch

i) With CYCLES = 10, Period is 19 µs

ii) $\frac{10}{19} = \frac{x}{500} \Rightarrow \frac{5000}{19} = x \approx 263$, Period is ~400 µs

iii) CYCLES = 330, Period is ~500 µs

iv) add two 'nop' instructions before toggle ⇒ T = 500 µs

Task 2 — Output Compare

① Write an ASM program for the AVR8515 running at 4MHz to toggle the OC1A pin at 1KHz using OCR1

② Simulation

(A) Note: The mainloop is entered after 5.75 µs

(B) I expect OC1A (PORTD[5]) to toggle at 505.75 µs

i) This is precisely what happens

C) Set Auto Step to watch toggling behaviour

Notes: The software delay is not a very good system. Firstly, when in the delay loop, the processor cannot do anything else. If it did the timing would be wrong. Second, depending on the number of instructions in the delay loop and how many cycles each instruction takes, a certain number of 'nop' instructions are necessary to improve accuracy.

task2.asm

```
; FILE: prac8-2.asm
;
; Replace the lines ";<-YOUR CODE HERE->" with your code.
;
;*****************************************************
.include "8515def.inc"
        rjmp    RESET           ; Reset Handler
;*****************************************************

.def    temp    = r16

RESET:
        ; Initialise Stack Pointer
        ldi     temp,low(RAMEND)
        out     SPL,temp
        ldi     temp,high(RAMEND)
        out     SPH,temp

        ; Set PORT D Pins 5 and 4 direction to output

        ;<-YOUR CODE HERE->
        ldi temp, 0x30
        out DDRD, temp

        ; Set PORT D Pin 4 output to zero

        ;<-YOUR CODE HERE->
        clr temp
        out PORTD, temp         ; this will actually set both pin 4 and 5 outputs to 0


        ; Set OC1A pin to toggle
        ; (See description of TCCR1A register on page 36 of the 8515
        ; datasheet. In particular look at bits COM1A1 and COM1A0)

        ;<-YOUR CODE HERE->
        ldi temp, 0x40          ; we need COM1A1 = 0, COM1A0 = 1
        out TCCR1A, temp        ; set TCCR1A to toggle OC1A

        ; Set Output Compare 1 value (i.e. registers OCR1AH and OCR1AL)
        ; to be 1999 (i.e. when the timer reaches this value, we want the
        ; output compare to trigger). This will mean the OC1A output will
        ; toggle every 2000 clock cycles.
        ; (Note (from page 39 of the datasheet) - the high byte of OCR1A must
        ; be written before the low byte.)
        ; Hint: use the low() and high() macros as in the stack pointer initialisation

        ;<-YOUR CODE HERE->
        .equ    CYCLES = 1999
        ldi temp, high(CYCLES)
        out OCR1AH, temp
        ldi temp, low(CYCLES)
        out OCR1AL, temp

        ; Set Timer 1 to clear on compare match (i.e. when the timer reaches
        ; 1999 we want it to start counting from 0 again)
        ; HINT: Look at the CTC1 bit of the TCCR1B register - see page 37
        ; of the datasheet.

        ;<-YOUR CODE HERE->
        ldi temp, 0x08          ; we need CTC1 = 1
        out TCCR1B, temp        ; set TCCR1B to clear on compare match

        ; Start Timer 1, so that it counts by 1 on every clock cycle.
        ; HINT: Look at the CS12:CS11:CS10 bits of the TCCR1B register -
        ; see page 37 of the datasheet
        ; (Remember that this is the same register that holds the CTC1
        ; bit - don't undo what you just did above - or maybe combine
```

```
                                                task2.asm
        ; the two lines of code that modify TCCR1B together.)

        ;<-YOUR CODE HERE->
        in temp, TCCR1B         ; load current status into temp
        ori temp, 0x01          ; modify only the bit 0 to 1
        out TCCR1B, temp        ; set TCCR1B to count every clock cycle


mainloop:
        ; Sit back and let it happen - the hardware takes care of it all
        rjmp    mainloop
```

```
; Replace the lines ";<-YOUR CODE HERE->" with your code.
;
;*****************************************************
.include "8515def.inc"
        rjmp    RESET           ; Reset Handler
;*****************************************************

.def    temp    = r16

RESET:
        ; Initialise Stack Pointer
        ldi     temp,low(RAMEND)
        out     SPL,temp
        ldi     temp,high(RAMEND)
        out     SPH,temp

        ; Set PORT B direction to output
        ;<-YOUR CODE HERE->
        clr temp
        out DDRB, temp

        ; Set PORT B output to zero
        ;<-YOUR CODE HERE->
        out PORTB, temp

mainloop:
        ; Call procedure which delays for 500uS
        rcall   delay

        ; Toggle (i.e. flip) bit 0 of PORT B
        ; HINT: Consider using the eor (Exclusive OR) instruction,
        ; but remember that it doesn't operate on I/O registers,
        ; only on general purpose registers;
        ;<-YOUR CODE HERE->
        in temp, PORTB
        ldi r17, 0x01
        eor temp, r17
        nop
        nop
        out PORTB, temp

        ; Run forever
        rjmp    mainloop

;*****************************************************

; define a symbolic constant that holds how many times we'll iterate through
; the loop
; <- CHANGE THIS NUMBER ->
.equ    CYCLES = 330

delay:
        ; Procedure to generate 500uS (i.e. 2000 clock cycle) delay
        ; We'll use the 16-bit X register (i.e. r27:r26) to count the
        ; number of times we'll iterate through the loop

        ; load XH with the high byte of our loop counter
        ldi XH, high(CYCLES)

        ; load XL with the low byte of our loop counter
        ;<-YOUR CODE HERE->
        ldi XL, low(CYCLES)

loop:
        ; decrement X by 1 (i.e. subtract 1 from X)
        ; (Remember, X is a 16-bit quantity)

        ;<-YOUR CODE HERE->
        ldi temp, 0xFF
        add XL, temp    ; add -1 to the low byte
        adc XH, temp    ; add -1 to the high byte including the carry from the low byte

        ; if we haven't reached 0, return to loop
        ;<-YOUR CODE HERE->
        cpi XL, 0x00    ; is the low byte 0?
        brne loop               ; no? loop again
        cpi XH, 0x00    ; is the high byte 0?
        brne loop               ; no? loop again

        ; We have reached 0 (i.e. have completed our loop)
        ; Return from procedure
        ret
```

Prac 8 4209353

Procedure

Task 1 – Software delay

① Load program to the Project Board
② Switch off power supply
③ Connect speaker to PORTB[0-1]
④ Switch on power supply

Notes: Parter's and my code is practical for the AVR board. We need to be sure to debug in AVR software and debug hardware connections.

Task 2 – Output Compare

① Load program to the Project Board
② Switch off power supply
③ Connect speaker to PORTD[4-5]
④ Switch on power supply

Notes: We noticed the speaker had a bad connexion. But, when that was fixed, the output was as expected.

Task 3 – Tone Generator in C

① Write a C program for Task 2
② Test – Simulator won't show the infinite loop. (Hardware clock)
③ Load program to project board
④ Switch off, connect speaker, Switch on power supply

Notes: It is much easier to write this program in C than ASM because the C compiler chooses registers and manipulates values (such as 16 bit registers) automatically. The final output sound is the same so the hardware was programed the same way. C programming at this low level make it more fun than ASM.

Prac 8

task3.c

```
/*
 * FILE: task3.c
 *
 * Replace the "<-YOUR CODE HERE->" comment lines with your code.
 */

#include <avr/io.h>

/*
 * main -- Main program.
 */
int main(void)
{
        /* Set PORT D Pins 5 and 4 direction to output */
        /* NOTE that DDD5 is equal to the value 5 and
        ** DDD4 is equal to the value 4. (These constants
        ** are defined in a header file included by avr/io.h.)s
        ** The names are the same as those given in the
        ** datasheet for each bit position.
        */
        DDRD = (1<<DDD5) | (1<<DDD4);

        /* Set PORT D Pin 4 output to zero */
        /*<-YOUR CODE HERE->*/
        PORTD = 0; /* both 4 and 5 will go to 0 but OK */

        /* Set OC1A pin to toggle */
        /*<-YOUR CODE HERE->*/
        TCCR1A = 0x40;

        /* Set Output Compare 1 value */
    OCR1A = 1999;
        /* An alternative would be to write:
        *       OCR1AH = 1999/256;
        *       OCR1AL = 1999 & 0xFF;
        *  but C allows us to treat OCR1A as a single 16 bit register
        */

        /* Set Timer 1 to clear on compare match */
        /*<-YOUR CODE HERE->*/
        TCCR1B = 0x08;


        /* Start Timer 1 */
        /*<-YOUR CODE HERE->*/
        TCCR1B |= 0x01;

        /* Sit back and let it happen - the hardware takes care of it all */
        while(1){
        }
}
```

1 Demo
1 WkBk
1 Tutor

Prac 8 42094353

Task 4 - Analysis of Compiled C Programs

① Set PORTD[4-5] to output
C) ldi r24, 0x30 ; out 0x11, r24
ASM) ldi temp, 0x30 ; out DDRD, temp
Note) C uses a temp register 'r24' then outputs to IO

② Set PORTD output to 0
C) out 0x12, r1
ASM) clr temp ; out PORTD, temp
Note) C sets r1 to 0 earlier in the program

③ Set OC1A pin to toggle
C) ldi r24, 0x40 ; out 0x2F, r24
ASM) ldi temp, 0x40 ; out TCCR1A, temp
Note) Difference is in temp register

④ Set Output Compare 1 to 1999
C) ldi r24, 0xCF ; ldi r25, 0x07
out 0x2B, r25 ; out 0x2A, r24
ASM) ldi Temp, high(CYCLES) ; out OCR1AH, temp
ldi temp, low(CYCLES) ; out OCR1AL, temp
Note) C uses two registers for temp values

⑤ Set timer to clear on compare match
C) ldi r24, 0x08 ; out 0x2e, r24
ASM) ldi temp, 0x08 ; out TCCR1B, temp
Note) Explicit registers, different temp register

⑥ Set Timer 1
C) in r24, 0x2e ; ori r24, 0x01 ; out 0x2e, r24
ASM) in temp, TCCR1B ; ori temp, 0x01 ; out TCCR1B, temp
Note) Just different temp registers.

Prac 8 4209 4353

Task 4 Conclusion

The compiler uses more optimized registers and refers directly to I/O registers. The I/O registers are defined in the header file as macros so the aliases are replaced within the code before compile. The compile options give the compiler the information it needs to optimise register usage (eg. using a seperate register to store a constant 0 instead of clearing a temp register

Tutor Task

Set up push buttons to alter the tone. It should play Mary Had a Little Lamb

① PB0 → 440.00 Hz
PB1 → 493.88 Hz
PB2 → 554.37 Hz
PB3 → 659.26 Hz

② This is implemented in C using if statements in the infinite loop.

③ We were able to compile and load the program successfully. We needed to be sure that with no buttons pushed, the timer/counter would be disabled. This was very simple in C: 'If' statements check the state of input pins and execute the correct expressions to change the output compare values.

④ Compiling with different optimizations:
The '-Os' optimization uses registers efficiently as if a person with full knowledge of the AVR hardware were writing it by hand in ASM. The '-O0' optimization uses many more registers and actually loads and stores values to memory. Memory instructions are very slow in terms of latency and processor cycles. '-Os' is much better than '-O0'
'-Os' optimizes for size. '-O0' is no optimization.

Prac 8

Tutor Task – Comparison of optimizations

-O0:
```
ldi r30, 0x4F
ldi r31, 0x00
ldi r24, 0x40
st  Z, r24
```
} loads an address into Z, then loads the value into temp, then stores the value at memory at the address stored in Z.

-Os:
```
ldi r24, 0x40
out 0x2F, r24
```
} loads the value into temp; outputs value directly to I/O register