Prac 9 42094353 Justin Mancinelli

Prep

Task 1 — Stopwatch

① Write an ASM and C program for the AVR 8515 for a stopwatch which has a HEX0, HEX1 for displaying the current time and a PB0 for starting and stopping.

- When the system is reset, HEX0 = HEX1 = 0.
- When the PB0 is pressed, the stopwatch starts counting up once every 10th second in decimal.
- When the PB0 is pressed again, the stopwatch pauses at its current value.
- When HEX0 = HEX1 = 9, the timer stops and can only be reset.

② Simulate ASM

- Note, simulation Auto Step is very slow so set Timer 1 such that clock is not divided (CS[12,11,10] = [0,0,1])
- Note, INT1 = PORTD[3]
- Testing:
  - ☐ Step 0 — Initialized HEX0=0, HEX1=0, time=0x00, run=0
  - ☐ Step 1 — Run without triggering INT1
    HEX0 = HEX1 = run = time = 0
  - ☐ Step 2 — Trigger INT1
    run = 1, time counts by one and if it reaches 0x0A gets reset to 0x10
    HEX[1,0] takes valid time values.
  - ☐ Step 3 — Trigger INT1
    run = 0, time and HEX0 and HEX1 stop counting
  - ☐ Step 4 — Trigger INT1, let run to 99
    When HEX1 = HEX0 = 9, the timer stops
  - ☐ Step 5 — Trigger INT1 multiple times
    HEX1 = HEX0 = 9 no matter how many times INT1 is triggered.

```
; Prac 9: prep.asm
.include "8515def.inc"

; Interrupt vector table
rjmp    RESET           ; IRQ0Reset handler
reti                            ; IRQ1 handler - not used
rjmp    EXT_INT1        ; IRQ2 handler (external interrupt)
reti                            ; IRQ3 handler - not used
rjmp    TIMER1_OCA      ; IRQ4 handler (Timer1 Compare Match A)


; Initialization
.def temp = r16
.def run = r17          ; if run = 1, timer is running
.def time = r18         ; current time

RESET:
        ;initialize stack pointer
        ldi     temp, low(RAMEND)
        out     SPL, temp
        ldi             temp, high(RAMEND)
        out             SPH, temp

        ; Timer is stopped initially
        clr             run
        ; Timer starts at 0
        clr             time

        ;set PORTC to be output
        ser             temp
        out DDRC, temp

        ; Set up timer 1 so we get an interrupt approximately
        ; every second. Set Output Compare value to 3906 (We
        ; will divide clock by 1024 to count it so this gives
        ; us about 1 sec delay for a 4MHz clock)
        .equ COUNT = 1;3906 1 sec, 390 1/10 sec

        ldi     temp, high(COUNT)
        out     OCR1AH, temp
        ldi     temp; low(COUNT)
        out             OCR1AL, temp
        ; Enable timer interrupt on output compare match
        ldi     temp, (1<<OCIE1A)
        out             TIMSK, temp
        ; Clear the counter on compare match
        ldi     temp, (1<<CTC1)
        out             TCCR1B, temp

        ; Setup External Interrupt 1
        ; Set INT1 for falling edge

        ldi     temp, (1<<ISC11)
        out             MCUCR, temp
        ; Clear INT1 flag
        ldi             temp, (1<<INTF1)
        out             GIFR, temp
        ; Enable INT1
        ldi             temp, (1<<INT1)
        out             GIMSK, temp

        ; Enable CPU interrupts
        sei

mainloop:
        ; The hardware initializes the counter as stopped at 00
        ; The external interrupt will alter values to start
        ; counting so there is nothing needed in the mainloop.
        rjmp    mainloop

EXT_INT1:
        ; External interrupt handler (service routine)
        ; - toggle whether we're running or not
```

```
        push    temp
        in              temp, SREG
        push    temp

        ; Are we running?
        tst             run
        brne    stop

start:
        ; Check whether we've reached 99. If we have, return.
        cpi             time, 0x99
        breq    end_ext_intl

        ; Start Timer 1 and configure it so clock is divided by 1024
        ; (CS12=1,CS11=0,CS10=0) and so that we clear the counter when we reach the
        ; OC value (CTC1=1)
        ;ldi    temp, (1<<CTC1)|(1<<CS12)|(0<<CS11)|(1<<CS10);
        ldi     temp, (1<<CTC1)|(0<<CS12)|(0<<CS11)|(1<<CS10) ← debugging
        out             TCCR1B, temp

        ; Set run flag and return
        inc             run
        rjmp    end_ext_intl

stop:
        ; Stop Timer 1
        clr             temp
        out             TCCR1B, temp

        ; Clear run flag and return
        clr             run

end_ext_intl:
        pop     temp
        out             SREG, temp
        pop             temp
        reti

TIMER1_OCA:
        ; Timer interrupt service routine. If we're running, we increment the time.
        ; If we reach 99, we stop running. All registers are preserved other than
        ; time and run.

        push    temp
        in              temp, SREG
        push    temp

        ; If we're not running
        ; Do nothing.
        cpi             run, 0
        breq    restore_sreg

        ; We are running;
        ; Check whether we've reached 99. If not, keep counting.
        cpi             time, 0x99
        brne    increment

        ; We reached 99
        ; Stop running and stop Timer 1
        clr             run
        clr             temp
        out             TCCR1B, temp

        ; restore registers
        rjmp restore_sreg

increment:
        ; Increment the time
        inc             time

        ; Check if right digit is 10, if not, display the time
        mov             temp, time
        andi    temp, 0x0F
        cpi             temp, 0x0A
        brne    display

        ; The right digit is 10.
        ; Clear the right digit and increment the left digit
        andi    time, 0xF0
        ldi     temp, 0x10
        add             time, temp

display:
        out             PORTC, time

restore_sreg:
        pop             temp
        out             SREG, temp
        pop             temp

        reti
```

```c
/*
 * FILE: prepC.c
 *
 */

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

volatile unsigned char run;      /* If run = 1, timer is running */
int      time;   /* the current time */

int main(void)
{
        /* Initialize variables */
        run = 0;
        time = 0;

        /* Initalize AVR8515 */
        /* PORTC will be output for hex display */
        DDRC = 0xFF;

        /* Set up timer 1 so we get an interrupt approximately
           every second. Set Output Compare value to 3906 (We
           will divide clock by 1024 to count it so this gives
           us about 1 sec delay for a 4MHz clock)
        */

        //OCR1A = 1                          //used for debugging
        OCR1A = 390;                 //tenth of a second delay

        TIMSK = (1<<OCIE1A);     //interrupt on output compare match
        TCCR1B = (1<<CTC1);      //clear the counter on output compare match

        MCUCR = (1<<ISC11);      //INT1 triggered on falling edge
        GIFR = (1<<INTF1);       //Clear the INT1 flag
        GIMSK = (1<<INT1);       //Enable INT1

        /* Enable interrupts */
        sei();

        /* Sit back and let it happen - this will loop forever */
        for (;;) {
        }

        return 0;
}

/* Save SREG Not necessary */

void stopTimer(void){
        TCCR1B = 0;
        run = 0;
}

ISR(INT1_vect){
        if(run == 0 && time != 0x99){    //not running and have more numbers to go
                        TCCR1B = (1<<CTC1)|(0<<CS12)|(0<<CS11)|(1<<CS10);
                        run = 1;

        }
        else
                stopTimer();
}

ISR(TIMER1_COMPA_vect){
        if(run == 1 && time != 0x99){    //running and have more numbers to go
                if( (time & 0x0F) >= 0x09)
                        time = (time & 0xF0) + 0x10; //clear right digit, carry to left
                else
                        time++; //add one to units

        }
        else    //either we're not supposed to be running or we've reached the end
                stopTimer();

        PORTC = time; //display the new time
}
```

½+½ Func ~~~ | Prep ~~~
| wkBk ~~~

Prac 7 4209435J 4/4 Justin Mancinelli

③ Simulate C

Simulation hangs when Enabling Interrupts "sei()"

④ Compare ASM with C lss file.

- The C program sets up a complete vector table. Note that only the interrupts we want are defined. Others say "<__bad_interrupt>"
- Global variables are initialized in memory
  run = 0 ⇒ sts 0x0062, r1
  Note r1 is constant 0
- 16 bit values are automatically split by the compiler
- The compiler automatically pushes and pops registers to and from the stack.

□ ASM: rjmp RESET ; reti ; rjmp EXT_INT1 ; ...
  C : rjmp .+24 ; rjmp .+54 ; rjmp .+106 ; ...

□ ASM: .def run = r17 ; clr run
  C: sts 0x0062, r1

□ ASM: ldi temp, high(390) ; ldi temp low(390)
  C: ~~#~~OCR1A = 390 → ldi r24, 0x86 ; ldi r25, 0x01

□ ASM: push temp ; in temp, SREG ; push temp
  C: push r1; push r0; in r0, 0x3f ; push r0

Procedure

Task 1 — Stopwatch

① Test both ASM and C programs on the project board.

- ASM and C tested and performed the same.
  1) Program board — initialized at 00, stopped
  2) Push button — counter starts, increasing every tenth sec
  3) Push button — counter stops, current time is preserved
  4) Push button — counter starts, increasing every tenth sec
  5) Let run — counter stops at 99
  6) Push button — counter stays stopped, display preserved at 99
  7) Reset — display is at 00, counter is stopped

Prac 9 42094353 Justin Mancinelli

Task 2 — Interrupt Based Serial Input/Output

① Modify the given skeleton code to create an interrupt-based C program to read characters from the AVR UART (serial port) and echo them back to the serial port except if a number (0-9) is recieved as input. In this case, a word description is output ("zero" to "nine") instead.

• Notes:

A) UCR is the UART Control Register
it sets how the UART will be used

B) UDR is The UART ~~I/O~~ Data Register
It's actually two seperate registers TX and RX
The hardware determines which to use depending on whether the UART is sending or recieving data.

② Configure Terminal and program the board
• Terminal configured to connect via COM1 at 19200 baud accepting data 8 bits long, and 1 stop bit
• program loaded successfully

③ Testing
Step 1: Start connection — Nothing displayed on screen
Step 2: Type Alpha characters — Echoed to the screen
Step 3: Type Numbers — Correct string echoed to screen
Step 4: For fun, try backspace — carriage goes back one but character not erased (this requires further programming in software).

④ Conclusion
The AVR 8515 provides a simple interface for transmitting an recieving data over a Serial Port (UART). By taking advantage of interrupts defined on the hardware, we can efficiently use the CPU's resources. That is, the CPU does not have to wait to recieve data or wait for data to complete transfer. It can work on whatever it needs to until interrupted. It can then process the signal, perform as necessary, then get back to other matters.

```c
/*
 * FILE: prac9-1.c
 *
 * Replace the "<-YOUR CODE HERE->" comments with your code.
 */

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


/* Global variables */
/*
** Circular buffer to hold outgoing characters. The insert_pos variable
** keeps track of the position (0 to BUFFER_SIZE-1) that the next
** outgoing character should be written to. bytes_in_buffer keeps
** count of the number of characters currently stored in the buffer
** (ranging from 0 to BUFFER_SIZE). This number of bytes immediately
** prior to the current insert_pos are the bytes waiting to be output.
** If the insert_pos reaches the end of the buffer it will wrap around
** to the beginning (assuming those bytes have been output).
*/
#define BUFFER_SIZE 64
volatile char buffer[BUFFER_SIZE];
volatile unsigned char insert_pos;
volatile unsigned char bytes_in_buffer;

/*
** Text to be output in place of digits
*/
char* numbers[10] = {"zero", "one", "two", "three", "four",
        "five", "six", "seven", "eight", "nine"};

/*
 * main -- Main program.
 */
int main(void)
{
    /*
    ** Initialise our buffer
    */
    insert_pos = 0;
    bytes_in_buffer = 0;

    /* Set the baud rate to 19200 - see page 58 of the datasheet
    ** to see what value to write to the UBRR register when we
    ** have a 4MHz clock
    */
    UBRR = 12;

    /*
    ** Enable transmission and receiving via UART and also
    ** enable the Receive Complete Interrupt and the Data Register
    ** Empty interrupt. This ensures that we get an interrupt
    ** when the UART receives a character and when it is ready
    ** to accept a new character for transmission.
```

```c
    ** HINT: Look at the RXEN, TXEN, RXCIE, UDRIE bits of UCR
    ** (see page 56 of the datasheet)
    */

    /* UCR = UART Controll Register
       RXEN = Reciever Enable, TXEN, Transmitter Enable
       RXCIE = Recieve Complete Interrupt Enable
       ~~TXCIE = Transmit Complete Interrupt Enable~~ UDRIE - Data register
                                                       empty IE.
    */                                      UDRIE
    UCR = (1<<RXEN)|(1<<TXEN)|(1<<RXCIE)|(1<<~~TXCIE)~~;

    /* Enable interrupts */
    sei();

    /* Sit back and let it happen - this will loop forever */
    for (;;) {
    }
}

/*
** output_char
**
** Procedure to output a character (by adding it to the outgoing buffer)
** (The characters will get consumed by an interrupt handler (see below).)
*/
void output_char(char c) {
    /* Add the character to the buffer for transmission if there
    ** is space to do so. We advance the insert_pos to the next
    ** character position. If this is beyond the end of the buffer
    ** we wrap around back to the beginning of the buffer */
    /* NOTE: this only gets executed within an interrupt handler
    ** so we can be guaranteed uninterrupted access to the buffer.
    */
    if(bytes_in_buffer < BUFFER_SIZE) {
        /* We have room to add this byte */
        buffer[insert_pos++] = c;
        bytes_in_buffer++;
        if(insert_pos == BUFFER_SIZE) {
            /* Wrap around buffer pointer if necessary */
            insert_pos = 0;
        }
    }
    /* else, we have no room to add the byte - just discard it */
}

/*
** output_string
**
** Procedure to output a string (by adding it to the outgoing buffer
** character by character). We iterate over all characters in the
** string. (Remember, strings are null-terminated.)
*/
void output_string(char* str) {
    unsigned char i;    /* index into the string */
    for(i=0; str[i] != 0; i++) {
        output_char(str[i]);
    }
}

/*
 * Define the interrupt handler for UART Data Register Empty (i.e.
 * another character can be taken from our buffer and written out)
 */

ISR(UART_UDRE_vect) {
    /* Check if we have data in our buffer */
    if(bytes_in_buffer > 0) {
        /* Yes we do - remove the pending byte and output it
        ** via the UART. The pending byte (character) is the
        ** one which is "bytes_in_buffer" characters before the
        ** insert_pos (taking into account that we may
        ** need to wrap around to the end of the buffer).
        */
        char c;
        if(insert_pos - bytes_in_buffer < 0) {
            /* Need to wrap around */
            c = buffer[insert_pos - bytes_in_buffer
                + BUFFER_SIZE];
        } else {
            c = buffer[insert_pos - bytes_in_buffer];
        }
        /* Decrement our count of the number of bytes in the
        ** buffer
        */
        bytes_in_buffer--;

        /* Output the character via the UART */
        UDR = c;
    }
    /* else, no data in buffer - do nothing. This will cause this
    ** interrupt to trigger again immediately (unless there is an
    ** interrupt of higher priority (e.g. Receive complete)). Our
    ** program has nothing else to do so this is not a problem.
    */
}


/*
 * Define the interrupt handler for UART Receive Complete - i.e. a new
 * character has arrived in the UART Data Register (UDR).
 */
ISR(UART_RX_vect) {
    /* A character has been received - if it is not a number, then
    ** put it into the transmit buffer to echo it back. If it is
    ** a number than put the text equivalent into the transmit
    ** buffer
    */
    char input;

    /* Extract character from UART Data register and place in input
```

```c
    ** variable
    */

    /*When writing to the register, the UART Transmit Data register is written.
      When reading from UDR, the UART Receive Data register is read.*/
    input = UDR;

    if(input >= '0' && input <= '9') {
        /* Character is a number - output number string */
        /* We can use input-'0' (i.e. input - ASCII 48) as our
        ** index into the numbers array above */
        output_string(numbers[~~input~~]);
                            (input - 48)
    } else {
        /* Output just the character received */
        output_char(input);
    }
}
```

Task 2 - Conclusion Continued

- Using a higher baud rate introduces a lot of error. The Terminal is expecting abits at a certain rate and because of the stop bit it can handle some line noise. However The AVR 8515 cannot output to the UART at higher baud rates without significant error (the actual speed is not close enough to the expected speed). The wrong values are recognized (a → ±, u → §, 5 ⇒ Â).

Tutor Task - #2

① Write an AVR C program to allow the PC to control a speaker over the UART. When the system is reset, Timer 1 is set to clear on compare match (0x1000), the OC1A pin is set to toggle on compare match). The timer compare (OCR1A) can be updated by transmitting 2 8-bit chars over the UART. OCR1AH is set to the first character, OCR1AL is set to the second character.

② Test with the given C program, run in CMD.
- The first few tests failed because I hadn't initialized certain variables. When programming in C, you cannot assume that they will be initially given the value you want.
- After initializing my variables, the program worked as expected.

③ Conclusion.
Interrupts are very powerful tools. They give control over specific events and because they save the state of the program, they can do completely different behaviour from the main program.