Exercise 2: Microcontroller programming in C

Purpose: The purpose of this exercise is to give an introduction to C programming of micro controllers. The exercise will combine high level programming with low-level manipulations of registers and handling of interrupts.

It is recommended that all the programs are checked in the simulator before downloading to the target microcontroller as it is very difficult to debug the programs in the target itself.

Literature: Atmel ATmega32 data sheet manual and instruction set manual is needed (hand-out). The C library documentation for the Atmel processors is available at c:\WinAVR\doc\avr-libc\avr-libc-user-manual\index.html.

This exercise is written for AVR Studio version 4.1 and GCC C compiler used from Programmers Notepad version 2.0. If another version is used the same facilities should be available, but may not be found as described.
Programmers Notepad is a part of a WinAvr package that is free and can be downloaded from http://www.avrfreaks.net (or http://sourceforge.net/projects/winavr) The WinAvr package is available from Windows at C:\WinAvr.

Exercise a.:

The first exercise is to get a sample code running to try the tools.

Make a short-cut on your desktop to Programmers Notepad (if it is not there already) from the location C:\WinAVR\pn\pn.exe. And a short-cut to the C library documentation at the position described above.

Fetch the program blink3.c from Campusnet.

blink3.c.:

```
/* filename */
#include <avr/io.h>
#include <avr/interrupt.h> /* interrupt flags e.g. TIMSK */
/* Make boolean more readable */
#define true 1
void initialize (void);
void changeState(void);

unsigned char delay;
unsigned char state;

int main(int argn, char * argv[])
{ /* initialize system and variables */
  initialize();
  while (true)
    ; /* loop forever */
  return 0;
};

ISR (TIMER0_OVF_vect)
{ /* timer 0 interrupt */
    changeState();
}

void changeState(void)
{ /* Blink led 0 */
  switch (state)
  {
    case 0: /* lamp is off */
      if (delay >= 40) /* long off */
      { /* turn led 0 on */
        PORTB &= 0xfe;
        delay = 0; /* no initial delay */
        state = 1;
      }
      break;
    case 1: /* lamp is on */
      if (delay >= 3) /* short on */
      { /* turn led 0 off */
        PORTB |= 0x01;
        delay = 0;
        state = 2;
      }
      break;
    case 2: /* lamp is off */
      if (delay >= 5) /* short off */
      { /* turn led 0 off */
        PORTB &= 0xfe; /* turn led 0 on */
        delay = 0;
        state = 3;
      }
      break;
    case 3: /* lamp is on */
      if (delay >= 9) /* long on */
      { /* turn led 0 off */
        PORTB |= 0x01;
        delay = 0;
        state = 0; /* back to start */
      }
      break;
    default:
      state = 0;
      break;
  }
  delay++; /* advance time */
}

void initialize (void)
{ /* Configuration setup */
  delay = 0;
  state = 1; /* start at state 1*/
  DDRB = 0xff; /* Set direction of port B */
  PORTB = 0xfe; /* turn led 0 on initially */
  TCCR0 = _BV (CS02) | _BV (CS00); /* timer 0 prescaler 1024 */
  TIMSK = _BV (TOIE0); /* enable timer 0 interrupts */
  sei (); /* enable interrupts */
}
```

The specific processor type to compile the code for, is defined in the makefile. Copy the sample makefile from `C:\WinAVR\sample\Makefile` to your directory and open it in Programmers Notepad. There is a few variables that need to be set:

- Change the TARGET variable to your source file name without extension, e.g.: `TARGET = blink3` - if the source file is blink3.c. If there is more files, then these can be added to the `SRC` variable.
- The MCU variable must be set to the processor type used, i.e.: `MCU = atmega32` - in lower case letters as shown.
- The optimizer should not be used initially - but may be switched on later, so: `OPT = 0` should be used in place of the default value `OPT=s`. This makes it easier to debug using AVRStudio.
- AVRDUDE_PROGRAMMER = stk500v2
- Save the Makefile

Open blink3.c in programmers notepad. The code should now be ready to compile. Do a "make all" from the Tools menu.

When compiled correctly start AVRStudio and open the file with the .elf extension, this should bring you in simulation mode after selecting the right processor type. Try to step through the code and watch the increments of the variables by adding a watch on the `delay` and `state` variables (Select the variable in the source - right click and select "Add to watch: xxx", and make the watch window visible). It may be a good idea to change the prescaler value for timer 0 to 1 in place of 1024, to make the simulation run faster.

1. Look at a disassembled version of the code (View Disassembler),
   Where is the variable `state` stored?
   Interrupt from timer 0 goes to address 0x16 in program memory, at this position is a jump to another address which? - Show the calculation,
   To use calls the stackpointer needs to be loaded, where is that done?
   How many bytes do the program occupy?

2. From the C-library documentation find the following.
   Is floating point operation supported - e.g.: `sin(x)`?
   How big (bits or bytes) is an integer - e.g. specified as: `int n;`?

3. Download the code to the Atmel STK500 board and verify that the LED blinks (one short and one long (an A in Morse-code)).

Exercise b.:

In this exercise a C-program must be constructed that can communicate using the serial RS232 connection with a PC. It must be possible to set 2 values from the PC and to read the values back. The values are 4-digit codes (no zeros) that are to be used in a PIN code lock in exercise c.

4. Make a C program that uses the RS232 connection (as in the similar assembler exer-

cise) that accepts the following commands from the PC (HyperTerminal):

```
set a [xxxx](return)
set b [yyyy](return)
get a(return)
get b(return)
```

The [xxxx] and [yyyy] is an optional 4 digit number like 2352. If no code is supplied then the code is reset to 0000. Reply to the terminal either "Code is loaded" "Code is cleared" (if no code) or "Bad: xxxxx" where xxxxx is the bad command. The `get` command shall reply with the current code a or code b.

5. Test the application with the following commands:
```
set a 2345 (should be accepted)
set b 0099 (should be rejected - zeros not allowed)
set b 123 (should be rejected - too few digits)
set b (should be accepted)
get a (should reply 2345)
get b (should reply 0000)
```
   Printout of the communication should be included in the exercise log.

<u>Exercise c.:</u>

Make a PIN code lock using the codes set in exercise b. The function should be keyed in using one of the keys on the Atmel board. When a key is pressed a number of times with intervals of less than (e.g. 0.5 second), then this should be interpreted as a digit, a pause of more than this time should mark a change to digit 2 - and so on. When 4 digits is received the code should be matched with either code a or code b. If the code is correct the lock is either locked or unlocked. The state of the lock must be visible on the board LED and be written to the terminal.

6. Write C code that reads the state of one of the switches on the board at regular intervals, e.g. 50 times each second. The switches may bounce - i.e. switch between 0 and 1 several times when the switch is pressed or released - therefore a change of state must only be accepted if stable over e.g. 3 consecutive reads. When the switch has been pressed (>=3 reads) and released (>= 3 reads) then the key-press should be accepted.

7. Handling of key presses and key press timing should be implemented as a state machine. States could be digit number (e.g. 0 for no typing, 1 to 4 for digit being received), and time since last key press.
   When codes are being typed this should be indicated with one of the LEDs. If a too long break (timeout) between key-presses are detected (e.g. 3 seconds) or all 4 digits are received the code receiver should go back to state 0 (waiting for digit 1).

8. When a full code is received this should be validated with either code a or b, and the

result of this validation (good or bad, and if good, the new state - locked or unlocked) must be visible on both LEDs and on the terminal.

9. The selected values for time between key presses for one code, between codes and for timeout should be determined with experiments and reported on the home page. Printout from the terminal from a lock-unlock sequence must be reported on the home page.

Journal: The home page must (at least) include answers to questions and (link) to the final C code.

Note: **Strings in C library**: String handling is supported, but string constants (e.g.: the line `char s[] = "abc"`) will fail in simulator (in the tested version of AVRStudio 4.0). If the Atmel processor is loaded from simulator memory it will fail here too, but if loaded from .hex-file the string library works OK.