“REAL TIME CLOCK EMULATOR BASED SYSTEM ACTUATOR”

A Software Implementation Using AVR Atmega32 Microcontroller

A Project By:

Abhirag awasthi (200801160)

Ashwin Oke (200801164)

Darsh Shah (200801167)

Kartik Parekh (200801165)

Subodh Asthana (200801161)

Revision History:

Date	Version	Description
September,2010	1.0	Initial release

Roles:

ABHIRAG AWASTHI (200801160)[DOCU. + ALGO.]

ASHWIN OKE(200801164)[DOCU. + ALGO.+ CODE]

DARSH SHAH (200801167)[ALGO.+ CODE]

KARTIK PAREKH (200801165) [DOCU. + ALGO.]

SUBODH ASTHANA (200801161) [ALGO.+ CODE]

Table of Contents

Section 1-General Description

1.1-Introduction

We have basically developed a software emulation offering all the basic functionalities of any RTC chip. This can be used as a cost effective alternative to RTC chips in various embedded systems.

1.2- Overview

RTC emulator is a programmable software implementation of the basic functions of a RTC chip. It uses AVR Atmel Atmega32 microcontroller as the basic hardware platform and provides the utility of interrupts of any duration at any time of day. RTC emulator has been designed to be extremely portable. We have made this sure by developing it in "C" which is a high level programming language, thus enabling support for a variety of hardware platforms. But our prime focus has been on embedded systems using the Atmel AVR Atmega32 microcontroller. The software design is modular enabling easy integration of further developments and also making the code flexible and easy to manipulate. The most common applications which can make use of the RTC emulator functionality are:

- Handheld Data Loggers
- Security and Surveillance Systems
- Entertainment Systems
- Embedded Systems related to Agriculture
- Automotive Embedded Systems

The major characteristics of the RTC emulator are:

- Programmable Device
- Provides the feature to decide the time and duration of interrupts
- A very user-friendly LCD enabled user interface
- The user just has to input the current time along with other required inputs(specified step-by-step on LCD) to kick-start the device and does not have to worry about the implementation details of the system, thus abstraction level offered is high

1.3-AVR Atmel Atmega32 Microcontroller

The AVR Atmel Atmega32 has the following characteristics which are relevant to our system:

- Efficient "C" language support
- In-System FLASH programming
- Advanced RISC architecture
- 32 Programmable I/O lines

- Internal Calibrated Oscillator
- External and Internal interrupt sources
- Programmable Serial USART
- Two 8-bit Timer/Counters with Separate Prescalers and Compare modes
- One 16-bit Timer/Counter with Separate Prescaler and Compare Mode and Capture Mode
- 32K Bytes of In-System Self-programmable Flash Program Memory
- 32x8 General Purpose Working Registers
- Write/Erase Cycles: 10,000 Flash

1.4-Limitations

- Error accumulates at the rate of 50s/day during the day and is removed only at the end of the day
- The duration of interrupts cannot vary for different interrupt start times
- If input value for the interrupt to go-off is specified to be before the current time, the system will actuate only on the desired time the next day
- The time between the end of first interrupt and the start of second interrupt should be less than 24 hrs
- The deadlines implemented are soft in nature
- System is only designed to work on a daily basis and doesn't support working on alternate days etc.

1.5-Specifications

- UART interface to take the current time
- 16x2 LCD display for the display of system time and input values and input messages
- Atmel AVR Atmega32 as the basic hardware platform
- MAX 232 level converter helps in functioning of UART
- L293D for future extension of the system enabling interfacing with the motor
- DC motor(future extension)

Section 2- System Requirements

2.1-Introduction

The basic requirements so as to make RTC emulator practical, user-friendly, functionally robust are mentioned in this section. These requirements were kept in mind while designing the whole system and are the purpose and basis of the whole design of the system.

2.2-General requirements

The RTC emulator has the following general requirements:

- The user interface should must be as simple as possible
- The software architecture should be modular to enable reusability of code
- The design should include the possibility of being used for a variety of hardware platforms thus making the design portable
- The contact area of push buttons should be at-least 50 mm^2
- The power source can be 9V battery for standalone systems and for systems near to power supply a 9V DC adaptor can be used
- It must be able to provide specific (predefined) duration interrupts at specific (predefined) times of day

2.3-Functional requirements

The RTC emulator has the following functional requirements:

- It must be able to stop the errors arising due to not being able to produce exact one second interval from accumulating
- It must be able to perform the following functions:
  1. Keep track of the time of the day
  2. Provide specific duration interrupts at specific times
- It must be able to display the messages instructing the user to input particular values for the specific inputs

- It must also have the capability to display the input being entered by the user as he enters it

- It must have the capability to function continuously according to the provided inputs till new set of inputs are provided

2.4-Memory Requirements

RTC emulator uses FLASH memory as non-volatile read-only memory (between factory resets) and the requirement is that data should remain intact in the non-volatile memory for at-least one year even when the device is not being used.

The RTC emulator must store the following data in memory:

- The start time (hh:mm:ss) stored as part of initial configuration
- Desired time (hh:mm:ss) when the interrupt is supposed to go on for the first time
- Interval for which interrupt lasts

- Interval of time(calculated from the end of the first interrupt) after which next interrupt should go-off
- No. of times in a day such an interrupt is expected

2.5-Display Requirements

The RTC emulator has the following display requirements:

- Display must have at-least 8 digits
- It must have a display which enables the user to provide the inputs i.e. display with alphanumeric display capabilities
- The digits must be at-least .75 cm high

Section 3-Software Design

3.1.-Introduction

The whole theme of designing a real- time system for facilitating drip irrigation using moisture sensor involves implementing a Real Time Clock (RTC) using any microcontroller, which in this case is the Atmega32 AVR 8 bit microcontroller. The system is designed in such a way that it has to be first synchronized with the current time. After doing so, input is to be fed to the system through the user interface provided through the LCD display.

The LCD display first asks the user to input the time when the system is desired to operate/start. The input is to be given in the standard 24 hours time format. The input is provided by the user through the switches provided on the Atmega32 STK500 board. After inputting the time of start, the user is asked the duration for which the system is to be kept on. The next input to the system is the duration after which the system has to be started again after the previous system running is over. The last input that the user has to enter is the number of times the system has to run per day.

As an instance, if the series of inputs, the user provides to the system are as follows:

12:00:00, 00:15:00, 4:00:00, 2

The above series of inputs means that the system will first run at 12 noon. System, whenever it runs, will run for 15 minutes. It will restart every 4 hours and the 4 hours is counted after the previous running of the system is completed. The last input suggests that the system will run twice a day.

Hence, as an inference the above input will the start the system at 12 noon and at 1615hrs.

The output is depicted by one of the LEDs present on STK500 kit. The glowing LED signifies that the system is running and if it is not, the system is not running.

3.2.-Control Strategy

As discussed before, the primary portion of the system is to implement the RTC using the Atmega32 kit.

The RTC is implemented using the Timer 1 peripheral provided by the microcontroller. We chose the Timer 1 as it will help us reduce the errors. If we resolve to Timer0, the counter counts only till 255 and then raises an OVERFLOW0 interrupt. In this case, we will have to execute our ISR relatively greater number of times. Executing ISR greater number of times means wasting relatively more CPU cycles due to transition from main() to the ISR which can be reduced using the Timer 1 as it counts till 65535 (being a 16 bit counter) hence providing a greater scope.

An important issue in the design process is to minimize the error that is occurring due to transition of executing functions.

Parameters chosen for design:

Clock frequency: 8MHz

Prescalar ratio: 1 (timer frequency is the same as the clock frequency)

The calculation of the timer implementation as an RTC goes as follows:

We set the timer frequency equal to the system frequency at 8 MHz. It means that the timer increments the count value every 1/8 microseconds i.e. 0.125 us. Since we use the Timer1 peripheral, we even out the number of counts by making 50,000 counts and then raising the OVERFLOW1 interrupt of the Timer1.

Now, 50,000 counts will take 6250 us i.e 6.25 ms. Hence the interrupt will be generated every 6.25 ms. Now, for RTC, we need to tick our clock (in simplest terms observing the time by toggling a bit and displaying it on one of the ports) every 1 s.

So, for 1s, we need to execute the ISR 1000 ms/6.25ms = 160 times and then toggle the bit so that the time period of 1 s is observed. So after every 160^{th} execution of our ISR, we need to toggle the bit and show the tick of the clock on one of the LEDs. For this purposes, we set a global counter such that it increments its value every time the ISR is executed and after it reaches the value 160, we toggle one of the bits and reset the counter value to zero and implement the RTC logic.

As discussed before, a crucial point in this implementation is to minimize the error to as least value as possible.

We picked up certain values of prescalar ratios and system frequency and observed that the error was minimum when the timer frequency was maximum. Some error fractions are listed as below:

- CLK=8MHz, prescalar=8 (timer frequency=1MHz), the error is approximately: 0.05s/s which accumulates to approximately 72 mins/day

- CLK=4MHz, prescalar=8 (timer frequency=0.5MHz), the error is approximately: 0.1s/s which accumulates to 144 mins /day
- CLK= 8 MHz, prescalar=1 (timer frequency= 8 Mhz), the error is approximately: 0.006s/s which accumulates to approximately 518 sec/day (8.6 mins/day)

All these error calculations were calculated using the simulation mode provided by the Atmel AVR studio 4, software for programming the Atmel AVR microcontroller.

So we chose the 3rd case wherein the error is the minimum. We didn't go with other combinations of CLK frequency and prescalar ratios as we wanted to keep tidy relationships between number of counts and hence the executions of ISR that need to take place to observe 1s.

The error could further be reduced as it is known that some amount of time is lost due to the transition that the execution thread has to make from the ISR to main() and vice versa. It was a positive error which meant that time was getting lost and hence by decreasing the number of counts by calculated amount, the error could further be reduced.
Error:

- 6 ms for an interval of every 1s
- 6 ms for 160 executions of ISR
- 6/160ms (.0375 ms=37.5us) loss for every execution of ISR
- Clock cycles that are wasted = (37.5us/0.125us)=300 for every call to ISR or every other overhead involved.
- So if we reduce the number of counts by 300, we could get to the best approximation of the time period of 1 second.
- However, even after undergoing all these procedures, error was still prevalent. But the error now is reduced to 0.6ms/s (which is almost 10 times lesser than the previous observed error)

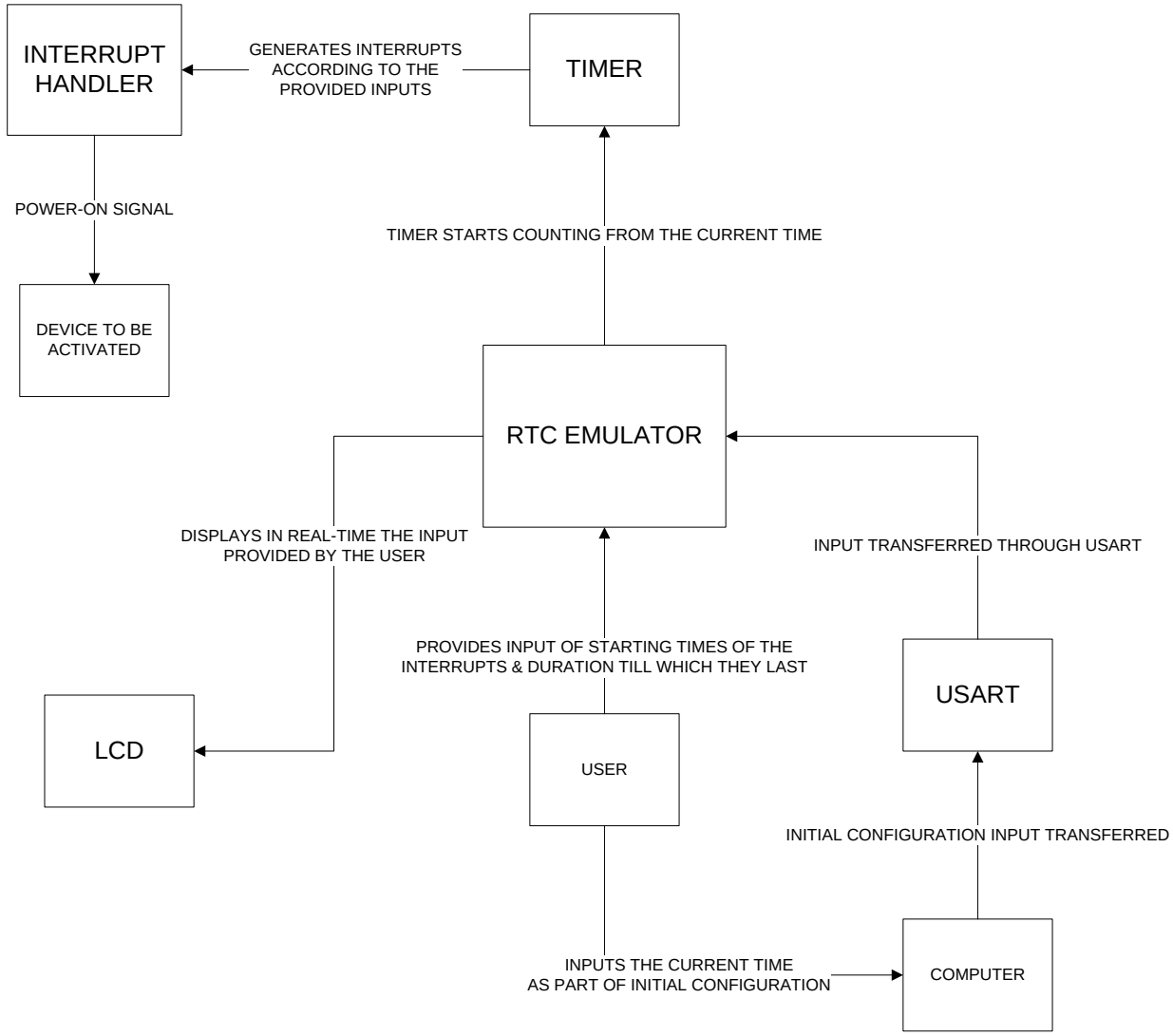
3.3-File Structure

FILE	DESCRIPTION
lcd.c	Defines the functions used for lcd interface
lcd.h	Header file containing the prototypes of the functions used for lcd interfacing.
rtc.c	Define the main() function and the ISRs

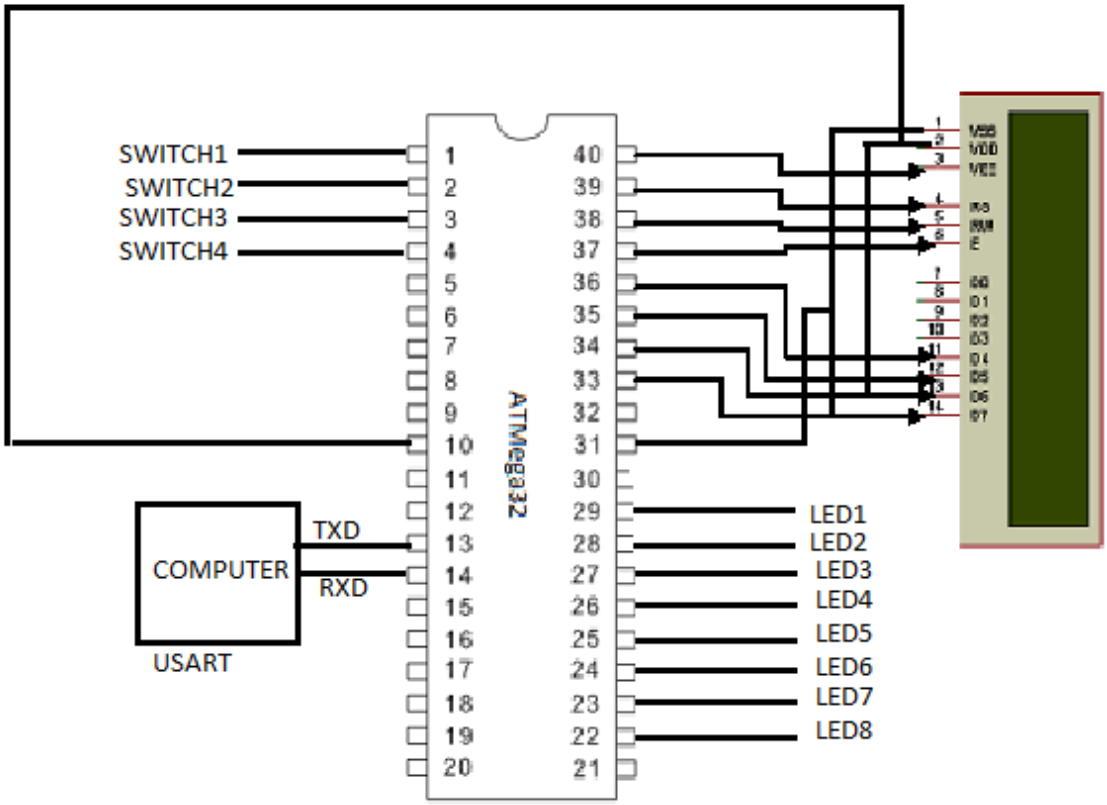
3.4-Summary of variables used

VARIABLE	DESCRIPTION
hr	The hour attribute of the current time
min	The minute attribute of the current time
sec	The second attribute of the current time
hr_d	The hour attribute of the desired time when the system has to start the first time
min_d	The minute attribute of the desired time when the system has to start the first time
sec_d	The second attribute of the desired time when the system has to start the first time
hr_dur	The hour attribute of the time when the system has to start after the previous running
min_dur	The minute attribute of the time when the system has to start after the previous running
sec_dur	The second attribute of the time when the system has to start after the previous running
turns	Number of times the system has to run per day
arr[j]	The array that shows any time in the format: hh:mm:ss

3.5-Block Diagram



3.6-Schematic Diagram



Section 4-Hardware Design

HARDWARE PLATFORM USED FOR THE SYSTEM:

For designing the system, the hardware board used is the **Atmel AVR STK500 kit**. The peripherals of this microcontroller used in the system include timers/counters, USART and interrupts.

The Atmel AVR STK500

The Atmel AVR STK500 is a starter kit and development system for Atmel's AVR Flash microcontrollers. The STK500 gives designers a quick start to develop code on the AVR, combined with features for developing prototypes and testing new designs. The STK500 interfaces AVR Studio, Atmel's Integrated Development Environment (IDE) for code writing and debugging.

Some salient features that the board provides are as follows:

- AVR Studio Operated
- Serial In-System Programming
- In-System Programming in External target Systems
- Parallel and Serial High-voltage Programming
- RS-232 Interface to PC
- Sockets for 8-, 20-, 28-, and 40-pin AVR Devices
- Flexible Clocking, Voltage and Reset System
- LEDs and Push Buttons for experimentation
- All AVR I/O Ports easily accessible through Pin Header Connectors
- Spare RS-232 Driver and Connector
- Upgrades can be done from AVR Studio, which offers diverse features that could help program the AVR microcontroller.
- Expansion Connectors for Plug-in- connector 0 and connector1

The STK500 is controlled from AVR Studio, version 3.2 and higher. AVR Studio is an Integrated Development Environment (IDE) for developing and debugging AVR applications. AVR Studio provides a project management tool, source file editor, simulator, in circuit Emulator interface and programming interface for STK500.

Section 5-PC Interface and Hardware Peripherals

Hardware peripherals used and their configurations so that they can be used with the AVR controller:

1. Timer1: The configuration of this peripheral is done using the several control registers which are listed as below:
    - TCNT1 (Timer Counter Register) – its value is set to be decimal 15836. It specifies the value from where we wish the counter to start counting from.
    - TCCR1A (Timer/Counter1 Control Register A) – Its value is set to be 0x00. This register mainly provides for features such as output compare, waveform generation mode, which are disabled
    - TCCR1B (Timer Counter Control Register B)- Its value is set to be decimal 1(0x01) which specifies that the timer frequency is the same as the CLK System frequency.
    - TIMSK (Timer Mask Register) – Its value is set to be 0x04. The bit corresponding to the TOIE1: Timer/Counter1, Overflow Interrupt Enable is written to 1.

2. USART (Universal Synchronous and Asynchronous Receiver and Transmitter): Even, this peripheral is configured using the Control registers provided by the microcontroller architecture. The registers configured are as below:
    - UCSRA (USART Control and Status Register A): Its value is set to 0x00.
    - UCSRB (USART Control and Status Register B): Its value is set such as the Receiver and the Transmitter is enabled. The interrupts are also enabled, Rx Complete Interrupt Enable and the Tx Complete Interrupt Enable.
      UCSRB= (1<<RXEN) | (1<<TXEN) |(1<<RXCIE)|(1<<TXCIE);
    - UBRR (USART Baud Rate Register) which consists of two registers UBRRH and UBRRL. The value of UBRRL is set to decimal value 207 which signifies the fact that we are using a Baud Rate of 2400 bits per sec at frequency of 8 MHz.

3. LCD Interface: The LCD is used as an output device. The LCD used in this project is JHD162A LCD module which is compatible with HD44780. Atmega 32 does not provide any inbuilt support for the LCD. Hence external libraries lcd.c and lcd.h are used. The LCD is used in 4-bit mode and hence only 10 pins are used. In 4 bit mode the upper nibble is sent first and then lower nibble. The pin configuration of LCD is given below:
   1-Vss
   2-Vcc
   3-Vee

4-RS
5-R/W
6-EN
7-DB0
8-DB1
9-DB2
10-DB3
11-DB4
12-DB5
13-DB6
14-DB7
15-LED+
16 LED-

DB0-DB7: Data Pins (use DB4-DB7 in 4 bit mode)

RS(Register Select) : 1 - Data is sent , 0 - Command is sent .

R/W(Read/Write) :1 - Read the LCD , 0 - Write to LCD. this pin is almost always low.

EN(Enable): to enable an operation . first make low(0) to send data and then set the other two control lines and when they are configured, bring EN high (1) and wait for the minimum amount of time required by the LCD and bring it low (0) again.

Vee: Contrast Adjust Pin. Connect to a potentiometer.

Now to use it in 8 bit mode we need 8 data pins DB0-DB7 and 3 control pins. So total 11-pins are required. In 4 bit mode, we need DB4-DB7 and 3 control pins. So total 7 pins and required. Note that the data pins DB0-DB3 are left open and are not shorted to ground. The Vee pin is connected to the potentiometer and is varied from Vcc and Gnd. Pin 15 and 16 are used for backlight of LCD and can be connected to Vcc and Gnd respectively.

We are using the library written by Peter Fleury - http://www.jump.to/fleury for the LCD interface. The library has excellent functions which can be directly used to make the lcd work.The LCD is initialized by function lcd_init(). To put anything on the LCD, put lcd_putc() or lcd_puts(). These functions can be seen from the header file lcd.h.

PC interface:

The design of the system facilitates it to be configured with a PC and can be used by interfacing the system with the Hyperterminal. Infact, in our system, the first input which is the current time given by the user is fed into the microcontroller through the Hyperterminal.

The Hyperterminal available as communicating modes on most of the PCs is configured with the AVR microcontroller using the USART, thus using the RS-232 medium and the physical connection between the two is made through the serial port. The interfacing between the two is done using the several USART control and configuration registers.

After interfacing, the microcontroller receives the current real time that the user inputs into the Hyperterminal in a PC. Using the USART peripheral of the microcontroller, we can program several other features of the system. Testing could also be done using the PC interfacing of the AVR microcontroller.

APPENDIX A:

Codes – lcd.h, lcd.c and rtc.c

lcd.h

```
#include <stdlib.h>
#include <avr/io.h>
#include <avr/pgmspace.h>
#include <compat/deprecated.h>                  //HEADER FILE FOR FUNCTIONS LIKE
SBI AND CBI
#include<util/delay.h>                                  //HEADER FILE FOR
DELAY


//                                                      MADE BY ROBOKITS INDIA.
//                                                      VISIT US AT
WWW.ROBOKITS.ORG
//                                                      DEFAULT PROGRAM SHOWING
SOME OF THE CAPABILITIES OF THIS BOAD
//                                                      WRITTEN FOR ROBOGRID

#ifndef LCD_H
#define LCD_H

#if (__GNUC__ * 100 + __GNUC_MINOR__) < 303
#error "This library requires AVR-GCC 3.3 or later, update to newer AVR-GCC
compiler !"
#endif

#include <inttypes.h>
#include <avr/pgmspace.h>

/**
 *  @name  Definitions for MCU Clock Frequency
 *  Adapt the MCU clock frequency in Hz to your target.
 */
#define XTAL 8000000            /**< clock frequency in Hz, used to calculate delay
timer */
```

```
/**
 * @name  Definition for LCD controller type
 * Use 0 for HD44780 controller, change to 1 for displays with KS0073 controller.
 */
#define LCD_CONTROLLER_KS0073 0  /**< Use 0 for HD44780 controller, 1 for
KS0073 controller */

/**
 *  @name  Definitions for Display Size
 *  Change these definitions to adapt setting to your display
 */
#define LCD_LINES           2     /**< number of visible lines of the display */
#define LCD_DISP_LENGTH     16     /**< visibles characters per line of the display */
#define LCD_LINE_LENGTH  0x40     /**< internal line length of the display    */
#define LCD_START_LINE1  0x00     /**< DDRAM address of first char of line 1 */
#define LCD_START_LINE2  0x40     /**< DDRAM address of first char of line 2 */
#define LCD_START_LINE3  0x14     /**< DDRAM address of first char of line 3 */
#define LCD_START_LINE4  0x54     /**< DDRAM address of first char of line 4 */
#define LCD_WRAP_LINES      0     /**< 0: no wrap, 1: wrap at end of visibile line */


#define LCD_IO_MODE      1         /**< 0: memory mapped mode, 1: IO port mode */
#if LCD_IO_MODE
/**
 *  @name Definitions for 4-bit IO mode
 *  Change LCD_PORT if you want to use a different port for the LCD pins.
 *
 *  The four LCD data lines and the three control lines RS, RW, E can be on the
 *  same port or on different ports.
 *  Change LCD_RS_PORT, LCD_RW_PORT, LCD_E_PORT if you want the control
lines on
 *  different ports.
 *
 *  Normally the four data lines should be mapped to bit 0..3 on one port, but it
 *  is possible to connect these data lines in different order or even on different
 *  ports by adapting the LCD_DATAx_PORT and LCD_DATAx_PIN definitions.
 *
 */
```

```
#define LCD_PORT         PORTA        /**< port for the LCD lines   */
#define LCD_DATA0_PORT   LCD_PORT     /**< port for 4bit data bit 0 */
#define LCD_DATA1_PORT   LCD_PORT     /**< port for 4bit data bit 1 */
#define LCD_DATA2_PORT   LCD_PORT     /**< port for 4bit data bit 2 */
#define LCD_DATA3_PORT   LCD_PORT     /**< port for 4bit data bit 3 */
#define LCD_DATA0_PIN    3            /**< pin for 4bit data bit 0  */
#define LCD_DATA1_PIN    2            /**< pin for 4bit data bit 1  */
#define LCD_DATA2_PIN    1            /**< pin for 4bit data bit 2  */
#define LCD_DATA3_PIN    0            /**< pin for 4bit data bit 3  */
#define LCD_RS_PORT      LCD_PORT     /**< port for RS line         */
#define LCD_RS_PIN       6            /**< pin  for RS line         */
#define LCD_RW_PORT      LCD_PORT     /**< port for RW line         */
#define LCD_RW_PIN       5            /**< pin  for RW line         */
#define LCD_E_PORT       LCD_PORT     /**< port for Enable line     */
#define LCD_E_PIN        4            /**< pin  for Enable line     */

#elif defined(__AVR_AT90S4414__) || defined(__AVR_AT90S8515__) ||
defined(__AVR_ATmega64__) || \
    defined(__AVR_ATmega8515__)|| defined(__AVR_ATmega103__) ||
defined(__AVR_ATmega128__) || \
    defined(__AVR_ATmega161__) || defined(__AVR_ATmega162__)
/*
 *  memory mapped mode is only supported when the device has an external data
memory interface
 */
#define LCD_IO_DATA      0xC000    /* A15=E=1, A14=RS=1                */
#define LCD_IO_FUNCTION  0x8000    /* A15=E=1, A14=RS=0                */
#define LCD_IO_READ      0x0100    /* A8 =R/W=1 (R/W: 1=Read, 0=Write   */
#else
#error "external data memory interface not available for this device, use 4-bit IO port
mode"

#endif

/* instruction register bit positions, see HD44780U data sheet */
#define LCD_CLR              0      /* DB0: clear display               */
#define LCD_HOME             1      /* DB1: return to home position     */
#define LCD_ENTRY_MODE       2      /* DB2: set entry mode              */
#define LCD_ENTRY_INC        1      /*   DB1: 1=increment, 0=decrement  */
#define LCD_ENTRY_SHIFT      0      /*   DB2: 1=display shift on        */
```

```
#define LCD_ON              3     /* DB3: turn lcd/cursor on           */
#define LCD_ON_DISPLAY      2     /*   DB2: turn display on            */
#define LCD_ON_CURSOR       1     /*   DB1: turn cursor on             */
#define LCD_ON_BLINK        0     /*     DB0: blinking cursor ?        */
#define LCD_MOVE            4     /* DB4: move cursor/display          */
#define LCD_MOVE_DISP       3     /*   DB3: move display (0-> cursor) ?  */
#define LCD_MOVE_RIGHT      2     /*   DB2: move right (0-> left) ?      */
#define LCD_FUNCTION        5     /* DB5: function set                 */
#define LCD_FUNCTION_8BIT   4     /*   DB4: set 8BIT mode (0->4BIT mode) */
#define LCD_FUNCTION_2LINES 3     /*   DB3: two lines (0->one line)      */
#define LCD_FUNCTION_10DOTS 2     /*   DB2: 5x10 font (0->5x7 font)      */
#define LCD_CGRAM           6     /* DB6: set CG RAM address           */
#define LCD_DDRAM           7     /* DB7: set DD RAM address           */
#define LCD_BUSY            7     /* DB7: LCD is busy                  */

/* set entry mode: display shift on/off, dec/inc cursor move direction */
#define LCD_ENTRY_DEC           0x04   /* display shift off, dec cursor move dir */
#define LCD_ENTRY_DEC_SHIFT     0x05   /* display shift on,  dec cursor move dir
*/
#define LCD_ENTRY_INC_          0x06   /* display shift off, inc cursor move dir */
#define LCD_ENTRY_INC_SHIFT     0x07   /* display shift on,  inc cursor move dir */

/* display on/off, cursor on/off, blinking char at cursor position */
#define LCD_DISP_OFF            0x08   /* display off                       */
#define LCD_DISP_ON             0x0C   /* display on, cursor off            */
#define LCD_DISP_ON_BLINK       0x0D   /* display on, cursor off, blink char    */
#define LCD_DISP_ON_CURSOR      0x0E   /* display on, cursor on             */
#define LCD_DISP_ON_CURSOR_BLINK 0x0F   /* display on, cursor on, blink char
*/

/* move cursor/shift display */
#define LCD_MOVE_CURSOR_LEFT    0x10   /* move cursor left  (decrement)
*/
#define LCD_MOVE_CURSOR_RIGHT   0x14   /* move cursor right (increment)
*/
#define LCD_MOVE_DISP_LEFT      0x18   /* shift display left                */
#define LCD_MOVE_DISP_RIGHT     0x1C   /* shift display right               */

/* function set: set interface data length and number of display lines */
```

```
#define LCD_FUNCTION_4BIT_1LINE  0x20   /* 4-bit interface, single line, 5x7 dots
*/
#define LCD_FUNCTION_4BIT_2LINES 0x28   /* 4-bit interface, dual line,   5x7 dots
*/
#define LCD_FUNCTION_8BIT_1LINE  0x30   /* 8-bit interface, single line, 5x7 dots
*/
#define LCD_FUNCTION_8BIT_2LINES 0x38   /* 8-bit interface, dual line,   5x7 dots
*/


#define LCD_MODE_DEFAULT     ((1<<LCD_ENTRY_MODE) |
(1<<LCD_ENTRY_INC) )


/**
 *  @name Functions
 */


/**
 @brief    Initialize display and select type of cursor
 @param    dispAttr \b LCD_DISP_OFF display off\n
               \b LCD_DISP_ON display on, cursor off\n
               \b LCD_DISP_ON_CURSOR display on, cursor on\n
               \b LCD_DISP_ON_CURSOR_BLINK display on, cursor on flashing
 @return  none
*/
extern void lcd_init(uint8_t dispAttr);


/**
 @brief    Clear display and set cursor to home position
 @param    void
 @return   none
*/
extern void lcd_clrscr(void);


/**
 @brief    Set cursor to home position
```

```
 @param    void
 @return   none
*/
extern void lcd_home(void);


/**
 @brief    Set cursor to specified position

 @param    x horizontal position\n (0: left most position)
 @param    y vertical position\n   (0: first line)
 @return   none
*/
extern void lcd_gotoxy(uint8_t x, uint8_t y);


/**
 @brief    Display character at current cursor position
 @param    c character to be displayed
 @return   none
*/
extern void lcd_putc(char c);


/**
 @brief    Display string without auto linefeed
 @param    s string to be displayed
 @return   none
*/
extern void lcd_puts(const char *s);


/**
 @brief    Display string from program memory without auto linefeed
 @param    s string from program memory be be displayed
 @return   none
 @see      lcd_puts_P
*/
extern void lcd_puts_p(const char *progmem_s);
```

```
/**
 @brief    Send LCD controller instruction command
 @param    cmd instruction to send to LCD controller, see HD44780 data sheet
 @return   none
*/
extern void lcd_command(uint8_t cmd);


/**
 @brief    Send data byte to LCD controller

 Similar to lcd_putc(), but without interpreting LF
 @param    data byte to send to LCD controller, see HD44780 data sheet
 @return   none
*/
extern void lcd_data(uint8_t data);


/**
 @brief macros for automatically storing string constant in program memory
*/
#define lcd_puts_P(__s)         lcd_puts_p(PSTR(__s))

/*@}*/
#endif //LCD_H
```

lcd.c

```
/*************************************************************************
*****
 Title                          :   HD44780U LCD library
 Author:    Peter Fleury <pfleury@gmx.ch>  http://jump.to/fleury
 File:              $Id: lcd.c,v 1.14.2.1 2006/01/29 12:16:41 peter Exp $
 Software:  AVR-GCC 3.3
 Target:    any AVR device, memory mapped mode only for AT90S4414/8515/Mega

 DESCRIPTION
```

Basic routines for interfacing a HD44780U-based text lcd display

Originally based on Volker Oth's lcd library,
changed lcd_init(), added additional constants for lcd_command(),
added 4-bit I/O mode, improved and optimized code.

Library can be operated in memory mapped mode (LCD_IO_MODE=0) or in
4-bit IO port mode (LCD_IO_MODE=1). 8-bit IO port mode not supported.

Memory mapped mode compatible with Kanda STK200, but supports also
generation of R/W signal through A8 address line.

USAGE

See the C include lcd.h file for a description of each function

```
**********************************************************************
*****/
#include <inttypes.h>
#include <avr/io.h>
#include <avr/pgmspace.h>
#include "lcd.h"



/*
** constants/macros
*/
#define DDR(x) (*(&x - 1))      /* address of data direction register of port x */
#if defined(__AVR_ATmega64__) || defined(__AVR_ATmega128__)
   /* on ATmega64/128 PINF is on port 0x00 and not 0x60 */
   #define PIN(x) ( &PORTF==&(x) ? _SFR_IO8(0x00) : (*(&x - 2)) )
#else
      #define PIN(x) (*(&x - 2))    /* address of input register of port x          */
#endif



#if LCD_IO_MODE
#define lcd_e_delay()   __asm__ __volatile__( "rjmp 1f\n 1:" );
#define lcd_e_high()    LCD_E_PORT  |=  _BV(LCD_E_PIN);
#define lcd_e_low()     LCD_E_PORT  &= ~_BV(LCD_E_PIN);
```

```
#define lcd_e_toggle()  toggle_e()
#define lcd_rw_high()   LCD_RW_PORT |=  _BV(LCD_RW_PIN)
#define lcd_rw_low()    LCD_RW_PORT &= ~_BV(LCD_RW_PIN)
#define lcd_rs_high()   LCD_RS_PORT |=  _BV(LCD_RS_PIN)
#define lcd_rs_low()    LCD_RS_PORT &= ~_BV(LCD_RS_PIN)
#endif

#if LCD_IO_MODE
#if LCD_LINES==1
#define LCD_FUNCTION_DEFAULT    LCD_FUNCTION_4BIT_1LINE
#else
#define LCD_FUNCTION_DEFAULT    LCD_FUNCTION_4BIT_2LINES
#endif
#else
#if LCD_LINES==1
#define LCD_FUNCTION_DEFAULT    LCD_FUNCTION_8BIT_1LINE
#else
#define LCD_FUNCTION_DEFAULT    LCD_FUNCTION_8BIT_2LINES
#endif
#endif

#if LCD_CONTROLLER_KS0073
#if LCD_LINES==4

#define KS0073_EXTENDED_FUNCTION_REGISTER_ON  0x24   /* |0|010|0100 4-
bit mode extension-bit RE = 1 */
#define KS0073_EXTENDED_FUNCTION_REGISTER_OFF 0x20   /* |0|000|1001 4
lines mode */
#define KS0073_4LINES_MODE                    0x09   /* |0|001|0000 4-bit mode,
extension-bit RE = 0 */

#endif
#endif

/*
** function prototypes
*/
#if LCD_IO_MODE
static void toggle_e(void);
#endif
```

```
/*
** local functions
*/



/*************************************************************************
**
 delay loop for small accurate delays: 16-bit counter, 4 cycles/loop
*************************************************************************
*/
static inline void _delayFourCycles(unsigned int __count)
{
   if ( __count == 0 )
      __asm__ __volatile__( "rjmp 1f\n 1:" );    // 2 cycles
   else
      __asm__ __volatile__ (
                                    "1: sbiw %0,1" "\n\t"
                        "brne 1b"                        // 4 cycles/loop
                                     : "=w" (__count)
                                      : "0" (__count)
                                              );
}


/*************************************************************************
**
delay for a minimum of <us> microseconds
the number of loops is calculated at compile-time from MCU clock frequency
*************************************************************************
*/
#define delay(us)  _delayFourCycles( ( ( 1*(XTAL/4000) )*us)/1000 )


#if LCD_IO_MODE
/* toggle Enable Pin to initiate write */
static void toggle_e(void)
{
   lcd_e_high();
```

```
   lcd_e_delay();
   lcd_e_low();
}
#endif


/*************************************************************************
**
Low-level function to write byte to LCD controller
Input:    data   byte to write to LCD
          rs     1: write data
                 0: write instruction
Returns:  none
*************************************************************************
*/
#if LCD_IO_MODE
static void lcd_write(uint8_t data,uint8_t rs)
{
   unsigned char dataBits ;


   if (rs) {   /* write data        (RS=1, RW=0) */
     lcd_rs_high();
   } else {    /* write instruction (RS=0, RW=0) */
     lcd_rs_low();
   }
   lcd_rw_low();

   if ( ( &LCD_DATA0_PORT == &LCD_DATA1_PORT) && (
&LCD_DATA1_PORT == &LCD_DATA2_PORT ) && ( &LCD_DATA2_PORT ==
&LCD_DATA3_PORT )
     && (LCD_DATA0_PIN == 0) && (LCD_DATA1_PIN == 1) &&
(LCD_DATA2_PIN == 2) && (LCD_DATA3_PIN == 3) )
   {
      /* configure data pins as output */
      DDR(LCD_DATA0_PORT) |= 0x0F;

      /* output high nibble first */
      dataBits = LCD_DATA0_PORT & 0xF0;
      LCD_DATA0_PORT = dataBits |((data>>4)&0x0F);
```

```
    lcd_e_toggle();

    /* output low nibble */
    LCD_DATA0_PORT = dataBits | (data&0x0F);
    lcd_e_toggle();

    /* all data pins high (inactive) */
    LCD_DATA0_PORT = dataBits | 0x0F;
}
else
{
    /* configure data pins as output */
    DDR(LCD_DATA0_PORT) |= _BV(LCD_DATA0_PIN);
    DDR(LCD_DATA1_PORT) |= _BV(LCD_DATA1_PIN);
    DDR(LCD_DATA2_PORT) |= _BV(LCD_DATA2_PIN);
    DDR(LCD_DATA3_PORT) |= _BV(LCD_DATA3_PIN);

    /* output high nibble first */
    LCD_DATA3_PORT &= ~_BV(LCD_DATA3_PIN);
    LCD_DATA2_PORT &= ~_BV(LCD_DATA2_PIN);
    LCD_DATA1_PORT &= ~_BV(LCD_DATA1_PIN);
    LCD_DATA0_PORT &= ~_BV(LCD_DATA0_PIN);
      if(data & 0x80) LCD_DATA3_PORT |= _BV(LCD_DATA3_PIN);
      if(data & 0x40) LCD_DATA2_PORT |= _BV(LCD_DATA2_PIN);
      if(data & 0x20) LCD_DATA1_PORT |= _BV(LCD_DATA1_PIN);
      if(data & 0x10) LCD_DATA0_PORT |= _BV(LCD_DATA0_PIN);
    lcd_e_toggle();

    /* output low nibble */
    LCD_DATA3_PORT &= ~_BV(LCD_DATA3_PIN);
    LCD_DATA2_PORT &= ~_BV(LCD_DATA2_PIN);
    LCD_DATA1_PORT &= ~_BV(LCD_DATA1_PIN);
    LCD_DATA0_PORT &= ~_BV(LCD_DATA0_PIN);
      if(data & 0x08) LCD_DATA3_PORT |= _BV(LCD_DATA3_PIN);
      if(data & 0x04) LCD_DATA2_PORT |= _BV(LCD_DATA2_PIN);
      if(data & 0x02) LCD_DATA1_PORT |= _BV(LCD_DATA1_PIN);
      if(data & 0x01) LCD_DATA0_PORT |= _BV(LCD_DATA0_PIN);
    lcd_e_toggle();

    /* all data pins high (inactive) */
```

```
      LCD_DATA0_PORT |= _BV(LCD_DATA0_PIN);
      LCD_DATA1_PORT |= _BV(LCD_DATA1_PIN);
      LCD_DATA2_PORT |= _BV(LCD_DATA2_PIN);
      LCD_DATA3_PORT |= _BV(LCD_DATA3_PIN);
   }
}
#else
#define lcd_write(d,rs) if (rs) *(volatile uint8_t*)(LCD_IO_DATA) = d; else *(volatile
uint8_t*)(LCD_IO_FUNCTION) = d;
/* rs==0 -> write instruction to LCD_IO_FUNCTION */
/* rs==1 -> write data to LCD_IO_DATA */
#endif


/*************************************************************************
**
Low-level function to read byte from LCD controller
Input:    rs     1: read data
               0: read busy flag / address counter
Returns:  byte read from LCD controller
*************************************************************************
*/
#if LCD_IO_MODE
static uint8_t lcd_read(uint8_t rs)
{
   uint8_t data;


   if (rs)
      lcd_rs_high();                  /* RS=1: read data      */
   else
      lcd_rs_low();                   /* RS=0: read busy flag */
   lcd_rw_high();                      /* RW=1  read mode      */

   if ( ( &LCD_DATA0_PORT == &LCD_DATA1_PORT) && (
&LCD_DATA1_PORT == &LCD_DATA2_PORT ) && ( &LCD_DATA2_PORT ==
&LCD_DATA3_PORT )
     && ( LCD_DATA0_PIN == 0 )&& (LCD_DATA1_PIN == 1) &&
(LCD_DATA2_PIN == 2) && (LCD_DATA3_PIN == 3) )
   {
```

```
    DDR(LCD_DATA0_PORT) &= 0xF0;        /* configure data pins as input */

    lcd_e_high();
    lcd_e_delay();
    data = PIN(LCD_DATA0_PORT) << 4;    /* read high nibble first */
    lcd_e_low();

    lcd_e_delay();                      /* Enable 500ns low       */

    lcd_e_high();
    lcd_e_delay();
    data |= PIN(LCD_DATA0_PORT)&0x0F;   /* read low nibble        */
    lcd_e_low();
}
else
{
    /* configure data pins as input */
    DDR(LCD_DATA0_PORT) &= ~_BV(LCD_DATA0_PIN);
    DDR(LCD_DATA1_PORT) &= ~_BV(LCD_DATA1_PIN);
    DDR(LCD_DATA2_PORT) &= ~_BV(LCD_DATA2_PIN);
    DDR(LCD_DATA3_PORT) &= ~_BV(LCD_DATA3_PIN);

    /* read high nibble first */
    lcd_e_high();
    lcd_e_delay();
    data = 0;
    if ( PIN(LCD_DATA0_PORT) & _BV(LCD_DATA0_PIN) ) data |= 0x10;
    if ( PIN(LCD_DATA1_PORT) & _BV(LCD_DATA1_PIN) ) data |= 0x20;
    if ( PIN(LCD_DATA2_PORT) & _BV(LCD_DATA2_PIN) ) data |= 0x40;
    if ( PIN(LCD_DATA3_PORT) & _BV(LCD_DATA3_PIN) ) data |= 0x80;
    lcd_e_low();

    lcd_e_delay();                      /* Enable 500ns low       */

    /* read low nibble */
    lcd_e_high();
    lcd_e_delay();
    if ( PIN(LCD_DATA0_PORT) & _BV(LCD_DATA0_PIN) ) data |= 0x01;
    if ( PIN(LCD_DATA1_PORT) & _BV(LCD_DATA1_PIN) ) data |= 0x02;
    if ( PIN(LCD_DATA2_PORT) & _BV(LCD_DATA2_PIN) ) data |= 0x04;
```

```
        if ( PIN(LCD_DATA3_PORT) & _BV(LCD_DATA3_PIN) ) data |= 0x08;
        lcd_e_low();
    }
    return data;
}
#else
#define lcd_read(rs) (rs) ? *(volatile uint8_t*)(LCD_IO_DATA+LCD_IO_READ) :
*(volatile uint8_t*)(LCD_IO_FUNCTION+LCD_IO_READ)
/* rs==0 -> read instruction from LCD_IO_FUNCTION */
/* rs==1 -> read data from LCD_IO_DATA */
#endif


/*************************************************************************
**
loops while lcd is busy, returns address counter
*************************************************************************
*/
static uint8_t lcd_waitbusy(void)

{
    register uint8_t c;

    /* wait until busy flag is cleared */
    while ( (c=lcd_read(0)) & (1<<LCD_BUSY)) {}

    /* the address counter is updated 4us after the busy flag is cleared */
    delay(2);

    /* now read the address counter */
    return (lcd_read(0));  // return address counter

}/* lcd_waitbusy */


/*************************************************************************
**
Move cursor to the start of next line or to the first line if the cursor
is already on the last line.
```

```
*************************************************************************
*/
static inline void lcd_newline(uint8_t pos)
{
   register uint8_t addressCounter;


#if LCD_LINES==1
   addressCounter = 0;
#endif
#if LCD_LINES==2
   if ( pos < (LCD_START_LINE2) )
      addressCounter = LCD_START_LINE2;
   else
      addressCounter = LCD_START_LINE1;
#endif
#if LCD_LINES==4
#if KS0073_4LINES_MODE
   if ( pos < LCD_START_LINE2 )
      addressCounter = LCD_START_LINE2;
   else if ( (pos >= LCD_START_LINE2) && (pos < LCD_START_LINE3) )
      addressCounter = LCD_START_LINE3;
   else if ( (pos >= LCD_START_LINE3) && (pos < LCD_START_LINE4) )
      addressCounter = LCD_START_LINE4;
   else
      addressCounter = LCD_START_LINE1;
#else
   if ( pos < LCD_START_LINE3 )
      addressCounter = LCD_START_LINE2;
   else if ( (pos >= LCD_START_LINE2) && (pos < LCD_START_LINE4) )
      addressCounter = LCD_START_LINE3;
   else if ( (pos >= LCD_START_LINE3) && (pos < LCD_START_LINE2) )
      addressCounter = LCD_START_LINE4;
   else
      addressCounter = LCD_START_LINE1;
#endif
#endif
   lcd_command((1<<LCD_DDRAM)+addressCounter);

}/* lcd_newline */
```

```
/*
** PUBLIC FUNCTIONS
*/

/*********************************************************************
**
Send LCD controller instruction command
Input:   instruction to send to LCD controller, see HD44780 data sheet
Returns: none
*********************************************************************
*/
void lcd_command(uint8_t cmd)
{
   lcd_waitbusy();
   lcd_write(cmd,0);
}


/*********************************************************************
**
Send data byte to LCD controller
Input:   data to send to LCD controller, see HD44780 data sheet
Returns: none
*********************************************************************
*/
void lcd_data(uint8_t data)
{
   lcd_waitbusy();
   lcd_write(data,1);
}



/*********************************************************************
**
Set cursor to specified position
Input:    x  horizontal position  (0: left most position)
          y  vertical position    (0: first line)
```

```
Returns:  none
*************************************************************************
*/
void lcd_gotoxy(uint8_t x, uint8_t y)
{
#if LCD_LINES==1
   lcd_command((1<<LCD_DDRAM)+LCD_START_LINE1+x);
#endif
#if LCD_LINES==2
   if ( y==0 )
      lcd_command((1<<LCD_DDRAM)+LCD_START_LINE1+x);
   else
      lcd_command((1<<LCD_DDRAM)+LCD_START_LINE2+x);
#endif
#if LCD_LINES==4
   if ( y==0 )
      lcd_command((1<<LCD_DDRAM)+LCD_START_LINE1+x);
   else if ( y==1)
      lcd_command((1<<LCD_DDRAM)+LCD_START_LINE2+x);
   else if ( y==2)
      lcd_command((1<<LCD_DDRAM)+LCD_START_LINE3+x);
   else /* y==3 */
      lcd_command((1<<LCD_DDRAM)+LCD_START_LINE4+x);
#endif

}/* lcd_gotoxy */


/*************************************************************************
**
*************************************************************************
*/
int lcd_getxy(void)
{
   return lcd_waitbusy();
}


/*************************************************************************
**
```

```
Clear display and set cursor to home position
*************************************************************************
*/
void lcd_clrscr(void)
{
   lcd_command(1<<LCD_CLR);
}


/*************************************************************************
**
Set cursor to home position
*************************************************************************
*/
void lcd_home(void)
{
   lcd_command(1<<LCD_HOME);
}


/*************************************************************************
**
Display character at current cursor position
Input:    character to be displayed
Returns:  none
*************************************************************************
*/
void lcd_putc(char c)
{
   uint8_t pos;


   pos = lcd_waitbusy();   // read busy-flag and address counter
   if (c=='\n')
   {
      lcd_newline(pos);
   }
   else
   {
#if LCD_WRAP_LINES==1
```

```
#if LCD_LINES==1
      if ( pos == LCD_START_LINE1+LCD_DISP_LENGTH ) {
         lcd_write((1<<LCD_DDRAM)+LCD_START_LINE1,0);
      }
#elif LCD_LINES==2
      if ( pos == LCD_START_LINE1+LCD_DISP_LENGTH ) {
         lcd_write((1<<LCD_DDRAM)+LCD_START_LINE2,0);
      }else if ( pos == LCD_START_LINE2+LCD_DISP_LENGTH ){
         lcd_write((1<<LCD_DDRAM)+LCD_START_LINE1,0);
      }
#elif LCD_LINES==4
      if ( pos == LCD_START_LINE1+LCD_DISP_LENGTH ) {
         lcd_write((1<<LCD_DDRAM)+LCD_START_LINE2,0);
      }else if ( pos == LCD_START_LINE2+LCD_DISP_LENGTH ) {
         lcd_write((1<<LCD_DDRAM)+LCD_START_LINE3,0);
      }else if ( pos == LCD_START_LINE3+LCD_DISP_LENGTH ) {
         lcd_write((1<<LCD_DDRAM)+LCD_START_LINE4,0);
      }else if ( pos == LCD_START_LINE4+LCD_DISP_LENGTH ) {
         lcd_write((1<<LCD_DDRAM)+LCD_START_LINE1,0);
      }
#endif
      lcd_waitbusy();
#endif
      lcd_write(c, 1);
   }

}/* lcd_putc */


/************************************************************************
**
Display string without auto linefeed
Input:    string to be displayed
Returns:  none
************************************************************************
*/
void lcd_puts(const char *s)
/* print string on lcd (no auto linefeed) */
{
   register char c;
```

```
   while ( (c = *s++) ) {
      lcd_putc(c);
   }

}/* lcd_puts */


/*************************************************************************
**
Display string from program memory without auto linefeed
Input:     string from program memory be be displayed
Returns:   none
*************************************************************************
*/
void lcd_puts_p(const char *progmem_s)
/* print string from program memory on lcd (no auto linefeed) */
{
   register char c;

   while ( (c = pgm_read_byte(progmem_s++)) ) {
      lcd_putc(c);
   }

}/* lcd_puts_p */


/*************************************************************************
**
Initialize display and select type of cursor
Input:    dispAttr LCD_DISP_OFF            display off
               LCD_DISP_ON             display on, cursor off
               LCD_DISP_ON_CURSOR      display on, cursor on
               LCD_DISP_CURSOR_BLINK   display on, cursor on flashing
Returns:  none
*************************************************************************
*/
void lcd_init(uint8_t dispAttr)
{
#if LCD_IO_MODE
```

```
   /*
    *  Initialize LCD to 4 bit I/O mode
    */

   if ( ( &LCD_DATA0_PORT == &LCD_DATA1_PORT) && (
&LCD_DATA1_PORT == &LCD_DATA2_PORT ) && ( &LCD_DATA2_PORT ==
&LCD_DATA3_PORT )
     && ( &LCD_RS_PORT == &LCD_DATA0_PORT) && ( &LCD_RW_PORT ==
&LCD_DATA0_PORT) && (&LCD_E_PORT == &LCD_DATA0_PORT)
     && (LCD_DATA0_PIN == 0 ) && (LCD_DATA1_PIN == 1) &&
(LCD_DATA2_PIN == 2) && (LCD_DATA3_PIN == 3)
     && (LCD_RS_PIN == 4 ) && (LCD_RW_PIN == 5) && (LCD_E_PIN == 6 ) )
   {
      /* configure all port bits as output (all LCD lines on same port) */
      DDR(LCD_DATA0_PORT) |= 0x7F;
   }
   else if ( ( &LCD_DATA0_PORT == &LCD_DATA1_PORT) && (
&LCD_DATA1_PORT == &LCD_DATA2_PORT ) && ( &LCD_DATA2_PORT ==
&LCD_DATA3_PORT )
        && (LCD_DATA0_PIN == 0 ) && (LCD_DATA1_PIN == 1) &&
(LCD_DATA2_PIN == 2) && (LCD_DATA3_PIN == 3) )
   {
      /* configure all port bits as output (all LCD data lines on same port, but control lines
on different ports) */
      DDR(LCD_DATA0_PORT) |= 0x0F;
      DDR(LCD_RS_PORT)    |= _BV(LCD_RS_PIN);
      DDR(LCD_RW_PORT)    |= _BV(LCD_RW_PIN);
      DDR(LCD_E_PORT)     |= _BV(LCD_E_PIN);
   }
   else
   {
      /* configure all port bits as output (LCD data and control lines on different ports */
      DDR(LCD_RS_PORT)    |= _BV(LCD_RS_PIN);
      DDR(LCD_RW_PORT)    |= _BV(LCD_RW_PIN);
      DDR(LCD_E_PORT)     |= _BV(LCD_E_PIN);
      DDR(LCD_DATA0_PORT) |= _BV(LCD_DATA0_PIN);
      DDR(LCD_DATA1_PORT) |= _BV(LCD_DATA1_PIN);
      DDR(LCD_DATA2_PORT) |= _BV(LCD_DATA2_PIN);
      DDR(LCD_DATA3_PORT) |= _BV(LCD_DATA3_PIN);
   }
```

```
    delay(16000);       /* wait 16ms or more after power-on       */

    /* initial write to lcd is 8bit */
    LCD_DATA1_PORT |= _BV(LCD_DATA1_PIN);  // _BV(LCD_FUNCTION)>>4;
    LCD_DATA0_PORT |= _BV(LCD_DATA0_PIN);  //
_BV(LCD_FUNCTION_8BIT)>>4;
    lcd_e_toggle();
    delay(4992);        /* delay, busy flag can't be checked here */

    /* repeat last command */
    lcd_e_toggle();
    delay(64);          /* delay, busy flag can't be checked here */

    /* repeat last command a third time */
    lcd_e_toggle();
    delay(64);          /* delay, busy flag can't be checked here */

    /* now configure for 4bit mode */
    LCD_DATA0_PORT &= ~_BV(LCD_DATA0_PIN);   //
LCD_FUNCTION_4BIT_1LINE>>4
    lcd_e_toggle();
    delay(64);          /* some displays need this additional delay */

    /* from now the LCD only accepts 4 bit I/O, we can use lcd_command() */
#else
    /*
     * Initialize LCD to 8 bit memory mapped mode
     */

    /* enable external SRAM (memory mapped lcd) and one wait state */
    MCUCR = _BV(SRE) | _BV(SRW);

    /* reset LCD */
    delay(16000);                       /* wait 16ms after power-on     */
    lcd_write(LCD_FUNCTION_8BIT_1LINE,0);   /* function set: 8bit interface */
    delay(4992);                        /* wait 5ms                     */
    lcd_write(LCD_FUNCTION_8BIT_1LINE,0);   /* function set: 8bit interface */
    delay(64);                          /* wait 64us                    */
    lcd_write(LCD_FUNCTION_8BIT_1LINE,0);   /* function set: 8bit interface */
    delay(64);                          /* wait 64us                    */
```

```
        #endif

        #if KS0073_4LINES_MODE
           /* Display with KS0073 controller requires special commands for enabling 4 line mode
        */
               lcd_command(KS0073_EXTENDED_FUNCTION_REGISTER_ON);
                           lcd_command(KS0073_4LINES_MODE);
              lcd_command(KS0073_EXTENDED_FUNCTION_REGISTER_OFF);
        #else
           lcd_command(LCD_FUNCTION_DEFAULT);      /* function set: display lines  */
        #endif
           lcd_command(LCD_DISP_OFF);              /* display off                  */
           lcd_clrscr();                           /* display clear                */
           lcd_command(LCD_MODE_DEFAULT);          /* set entry mode               */
           lcd_command(dispAttr);                  /* display/cursor control       */

}/* lcd_init */
```

rtc.c

```
        #include "lcd.h"


        #define STRING_LENGTH  28
        #define LCD_MAX      15
        #include<avr/io.h>
        #include<inttypes.h>
        #include<avr/interrupt.h>
        #define F_CPU 8000000UL

        int sec=0, min=0, hr=0;
        char buf[8];



        int count=0;

        SIGNAL(SIG_OVERFLOW1)
        {
```

```
                                        TCNT1=15836;
                                        if(count==160)
                                                {
                                                    //PORTD= PORTD ^ 0xF1;
                                                    count=0;
                                                    sec++;
                                                    if(sec==60)
                                                    {       min++;
                                                            sec=0;
                                                            if(min==60)
                                                            {       hr++;
                                                                    min=0;
                                                                    if(hr==24)
                                                                    {       hr=0;
                                                                    }
                                                            }
                                                    }


                                                }

                                               else
                                                    count++;
}


void uart_init()
{

                                        UCSRA= 0x00;
            UCSRB=(1<<RXEN) | (1<<TXEN) |(1<<RXCIE)|(1<<TXCIE);
                    UCSRC=(1<<URSEL)|(1<<UCSZ1)|(1<<UCSZ0);
                                        UBRRH=0x00;
                                         UBRRL=207;                     //setting baud rate as
2400 at Fosc=8 MHz

                                            sei();
}

char recv_byte()
```

```
{
                        while(!(UCSRA & (1<<RXC)));
                                        return UDR;
}

void transmit_byte(unsigned char data)
{
                        while(!(UCSRA & (1<<TXC)));
                                        UDR=data;
}



void lcdfunc()
{
char array[STRING_LENGTH] = { "  Give no. of times per day " };
int i,j;
int start;
int end;

lcd_init(LCD_DISP_ON_CURSOR);
  lcd_clrscr();

{
for ( j = 0; j < STRING_LENGTH; j++ )
{
   if ( j >= LCD_MAX )
      lcd_gotoxy( 0, 0 );
   else
      lcd_gotoxy( LCD_MAX - j, 0 );


                start = (j <= LCD_MAX) ? 0 : (j - LCD_MAX);

               end   = (j <= LCD_MAX) ? j : (start + LCD_MAX);

   if ( end >= STRING_LENGTH )
      end = STRING_LENGTH - 1;

   for ( i = start; i <= end; i++ )
```

```
      {lcd_putc( array[i] );
                                                  _delay_ms(20);
                                                  }
   _delay_ms(100);
}

}
}

void showTime()
{

                                        itoa(hr/10,buf,10);
                                      itoa(hr%10,buf+1,10);
                                      itoa(min/10,buf+2,10);
                                     itoa(min%10,buf+3,10);
                                      itoa(sec/10,buf+4,10);
                                     itoa(sec%10,buf+5,10);

                                           buf[7]=          buf[5];
                                       buf[6]= buf[4];
                                          buf[5]= ':';
                                       buf[4]= buf[3];
                                       buf[3]= buf[2];
                                          buf[2] = ':';
                                        lcd_clrscr();
                                       lcd_puts(buf);
                                      _delay_ms(50);
}


void extract(char input[], int *h,int *min,int *sec)
{
                                         *h = atoi(input+0);
                                       *min = atoi(input+3);
                                       *sec = atoi(input+6);
}

int main(void)
                                              {
```

```
char buffer[1];
char s[8];
int z=0;
int x = 0;
int j= 0;
int pos=0;
int y=0;
int cnt =0;
int m=0;
int arr[19];
int highT = 60000;
                                    int lowT = 26400;
                                      int temp = 0;
                                int sec_d=0, min_d=0, hr_d=0;              //system's desired
running time
                              int sec_dur=0, min_dur=0, hr_dur=0;//duration for which system
runs every time
                               int sec_int=0, min_int=0, hr_int=0;  // system restart interval;
                                     unsigned int turns=0;

                                     DDRD = 0xFF;
                                     PORTD=0xFF;
                                     DDRB = 0x00;
                                     PORTB=0xFF;
                                      DDRA=255;
                                      PORTA = 0;
                                     DDRC=0xFF;
                                     PORTC=0xFF;
                                         sei();

                        // first take current time from hyperterminal using usart
                                       uart_init();

   while(z<8)
   {    s[z]=recv_byte();
      _delay_ms(250);
      _delay_ms(250);
      transmit_byte(s[z]);
      PORTA = s[z];
      _delay_ms(250);
```

```
    _delay_ms(250);
                                                z++;
  }

            // put extract time here. data from uart is stored in s[z].
                        extract(s,&hr,&min,&sec);


                                    // start timer

                                TCCR1A= 0x00;          //disabling output compare
and PWM features
                                TCCR1B= 0x01;          //disabling noise capture
feature and setting
                                                                    //timer
frequency= CLK/8

                                 TIMSK= 0x04;          //unmasking timer 1 overflow
interrupt

                                TCNT1= 15836;


                        // start lcd - input sequence from user
                    lcd_init(LCD_DISP_ON_CURSOR_BLINK);


label:
                                        lcd_clrscr();
                                if (m == 0)
                                        lcd_puts("Put time:\n");
                                if (m == 1)
                                        lcd_puts("Put duration:\n");
                                if (m == 2)
                                        lcd_puts("Put interval:\n");
                                 if (m==3)
                                   lcdfunc();
  if (m==4)
                                {lcd_puts("byee !!!\n");
    _delay_ms(200);
```

```
goto over;
}

_delay_ms(250);


lcd_gotoxy(0,1);

while(1)
{
        x = PINB & 0x01;
        _delay_ms(50);

if (x ==0)
{
        y++;
        if (m==3)
        {}
        else{

        // Putting input digits within the
bounds


        if(j==0 || j==6 || j==12)
                y = y%3;
        else if (j==1 || j==3 || j==5 || j==7 ||
j==9 || j==11 || j==13 || j==15 || j==17)

                y = y%10;
        else if (j==2 || j==4 || j==8 || j==10 ||
j==14 || j==16)

                y = y%6;
        }
        itoa(y,buffer,10);
        lcd_puts(buffer);


lcd_command(LCD_MOVE_CURSOR_LEFT);
        _delay_ms(50);
}
```

```
                x = PINB & 0x02;
                _delay_ms(50);

            if (x ==0 )
            {

                if (y>0)
                {
                    y--;
                    if(j==0 || j==6 || j==12)
                        y = y%3;
                    else if (j==1 || j==3 || j==5 ||
j==7 || j==9 || j==11 || j==13 || j==15 || j==17)

                        y = y%10;
                    else if (j==2 || j==4 || j==8 ||
j==10 || j==14 || j==16)
                        y = y%6;
                }
                else
                y=9;


                itoa(y,buffer,10);
                lcd_puts(buffer);


        lcd_command(LCD_MOVE_CURSOR_LEFT);
                _delay_ms(50);
            }

            x = PINB & 0x04;
            _delay_ms(50);
            if (x ==0 )
            {
                arr[j] = y;
                j++;

                pos++;
```

```
                                        itoa(arr[j-1],buffer,10);   // Printing
the selected digit on the lcd
                                        lcd_puts(buffer);
                                        y=0;

                lcd_command(LCD_MOVE_CURSOR_RIGHT);
                                        _delay_ms(50);

                                    if( m == 3 )
                                    {   m++;
                                        break;
                                    }
                                    if ( pos >1)
                                    {       pos=0;
                                        cnt++;
                                        lcd_putc(':');

                lcd_command(LCD_MOVE_CURSOR_RIGHT);
                                        _delay_ms(50);
                                    }
                                    }


                                    x = PINB & 0x08; //backspace
                                    _delay_ms(50);
                                    if (x ==0 )
                                    {
                                      pos=0;
                                      cnt=0;
                                      y=0;

                                      if (j <6)
                                      j=0;
                                      if ( j>5 && j< 12)
                                      j=6;
                                      if(j>12 && j<19)
                                      j=12;

                                      goto label;
```

```
}


if (cnt>2)
{       lcd_clrscr();
        _delay_ms(250);
        m++;
        y=0;
        cnt=0;
        pos=0;

        break;
}

}


        goto label;


        over:   // this completes the input
```
taking process from the user.

// Tokenising the array arr[] and extracting the value of different inputs in correct format

```
hr_d = arr[0]*10 + arr[1];
min_d = arr[2]*10 + arr[3];
sec_d = arr[4]*10 + arr[5];


hr_dur = arr[6]*10 + arr[7];
min_dur = arr[8]*10 + arr[9];
sec_dur = arr[10]*10 + arr[11];

hr_int = arr[12]*10 + arr[13];
min_int = arr[14]*10 + arr[15];
sec_int = arr[16]*10 + arr[17];

turns = arr[18];
```

```
  //Checking if there is ovelap of the recurrent system with the next day start time

temp = turns * ((hr_dur*3600 + min_dur*60 +sec_dur) + (hr_int*3600 + min_int*60
+sec_int));

                        if(temp > highT + lowT)
                                {
                                    lcd_puts("Invalid Input");
                                    goto label;
                                }

                    // logic to check when to actuate the system
                                  while(1)
                                    {
                                                showTime();
                                while(turns)
                                    {
                                    while(!(hr==hr_d && min==min_d &&
sec==sec_d))
                                    {
                                            showTime();
                                    }

                                    PORTD= 0x00;

                                    // desired time reached. so system is
running. // put the L293 code.

                                    // updating time values as to when to stop

                                    hr_d=hr_d + hr_dur;
                                    if (hr_d>24)
                                    {
                                            temp = hr_d - 24;
                                            hr_d = temp;
                                    }

                                    min_d= min_d + min_dur;
                                    if (min_d > 60)
```

```
{
        temp = min_d - 60;
        min_d = temp;
}

sec_d= sec_d + sec_dur;
if (sec_d > 60)
{
        temp = sec_d - 60;
        sec_d = temp;
}

//waiting to stop...

while(!(hr==hr_d && min==min_d &&
sec==sec_d))
{
        showTime();

}

PORTD= 0xFF;        //stopping the system


// updating for next operation

hr_d= hr_d + hr_int;
if (hr_d>24)
{
        temp = hr_d - 24;
        hr_d = temp;
}

min_d=min_d + min_int;
if (min_d > 60)
{
        temp = min_d - 60;
        min_d = temp;
}
```

```
                    sec_d= sec_d + sec_int;
                    if (sec_d > 60)
                    {
                            temp = sec_d - 60;
                            sec_d = temp;
                    }

                    turns--;

            }


        hr_d = arr[0]*10 + arr[1];
      min_d = arr[2]*10 + arr[3];
       sec_d = arr[4]*10 + arr[5];

                    }|

    }
```

Program memory- 3664 bytes
Data memory- 110 bytes