TOSHKENT SHAHRIDAGI INHA UNIVERSITETI
INHA UNIVERSITY IN TASHKENT

Embedded Software & Design (SOC3050)

Detailed SW Design Specification

Digital Alarm Clock by Team IMPERIUM

Team members (Name, ID, Group/Section):

1. Oybek Amonov	U1610176	CSE16-2/001
2. Rakhmatjon Khasanov	U1610183	CSE16-2/001
3. Bokhodir Urinboev(leader)	U1610249	CSE16-1/001

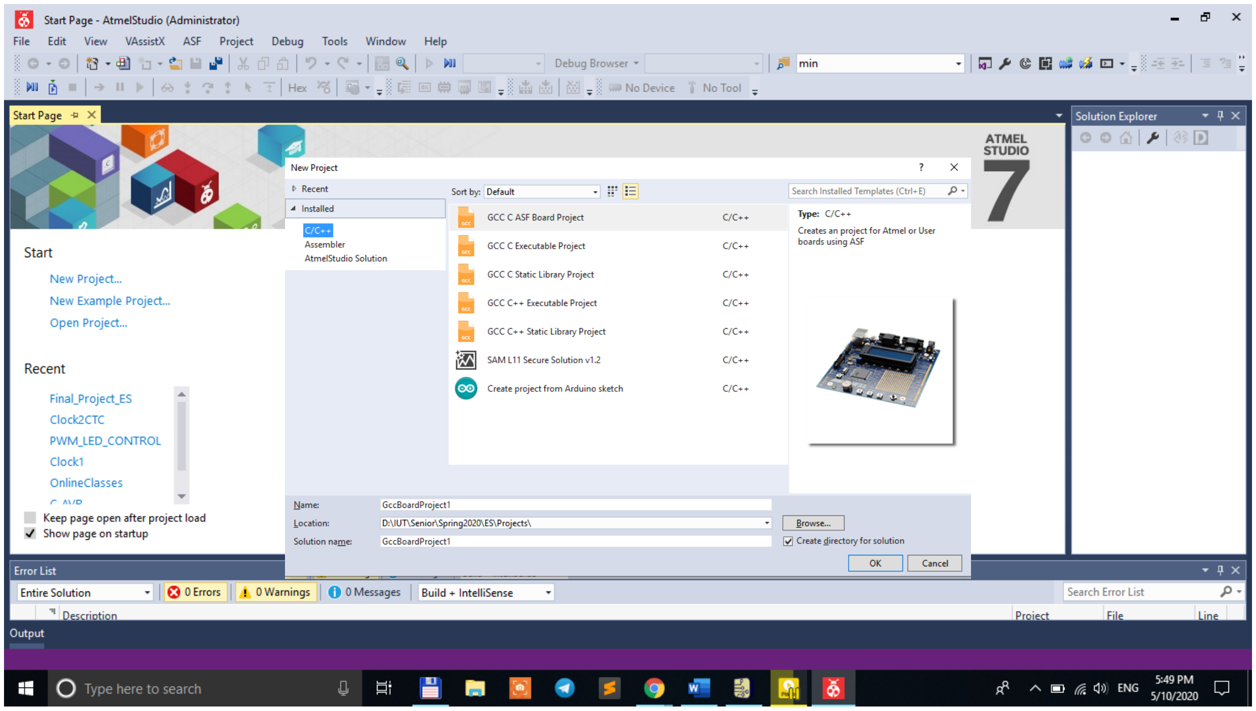
Table of Content

Introduction

Digital Alarm Clock using Atmega128 microcontroller project is aimed to achieve a digital clock with a few functional modes such as time display, alarm clock, and stopwatch. The clock is set by external interrupts, and it should operate under some provided requirements. The project must be coded on Atmel Studio and simulated with SimulIDE, considering simulator specifications. Since there is a difference between LCD used in actual Atmega128 Kit and LCD in simulator.

Purpose of the Project

The main objective of the project to use knowledge learned in microcontrollers, assembly and C languages. Students must learn to work with SimulIDE too, because the project needs real atmega128 kit. The working environment is Atmel Studio too where all necessary functions are coded.


Start Page - AtmelStudio (Administrator)
File Edit View VAssistX ASF Project Debug Tools Window Help
Debug Browser
No Device
No Tool
Start Page
Solution Explorer
ATMEL STUDIO 7
Start
New Project...
New Example Project...
Open Project...
Recent
Final_Project_ES
Clock2CTC
PWM_LED_CONTROL
Clock1
OnlineClasses
Keep page open after project load
Show page on startup
New Project
Recent
Installed
C/C++
Assembler
AtmelStudio Solution
Sort by: Default
Search Installed Templates (Ctrl+E)
GCC C ASF Board Project C/C++
GCC C Executable Project C/C++
GCC C Static Library Project C/C++
GCC C++ Executable Project C/C++
GCC C++ Static Library Project C/C++
SAM L11 Secure Solution v1.2 C/C++
Create project from Arduino sketch C/C++
Type: C/C++
Creates an project for Atmel or User boards using ASF
Browse...
Create directory for solution
OK
Cancel
Error List
Entire Solution
0 Errors
0 Warnings
0 Messages
Build + IntelliSense
Search Error List
Description
Project
File
Line
Output
Type here to search
ENG
5:49 PM
5/10/2020

Name: GccBoardProject1
Location: D:\IUT\Senior\Spring2020\ES\Projects\
Solution name: GccBoardProject1

Atmel Studio is an integrated development environment to develop assembly, and C/C++ projects. SimulIDE is a simulator tool for running the hex files generated after building Atmel projects.

System Overview

Functional Requirements

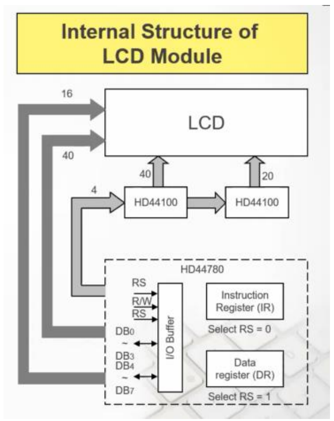
The digital clock must have following clock modes and Atmega (SimulIDE in our case) hardware handling functionalities :

- Display Mode (Normal clock with time information)
- Alarm Mode (Alarm time and LED blink notice when it is alarm time)
- Stopwatch Mode (Start/Pause/Reset operations)
- Set time/alarm Mode (Set hour, min, sec, year, day, month, weekday)

- External (Port D) and Internal (Timer) Interrupt Handling
- Clock Timer handling using 8-bit timer of Atmega128 (timer should be working either in following modes normal, CTC (Clear Timer on Compare Match), PWM (phase correct), or Fast PWM)

Hardware Requirements

To accomplish the project, Atmega128 kit is used, which includes atmega128 microcontroller, LCD monitor, and LEDs. LCD is used to display characters (English letters, numbers, special characters, and graphic display). There are different LCD available, with 20 characters / 4 lines, 16 characters / 2 lines, and 14 characters / 4 lines. The one we are using is Built-in HD44780 LCD. The LCD has following features:



- Interface with 4-bit, 8-bit microcontroller
- 5x8 dots, 5x10 dots display available
- Built-in 80x8-bit DDRAM (Display Data RAM: up to 80 characters)
- Built-in CGROM (Character Generator ROM) with 240 character font
- Built-in CGRAM (Character Generator RAM) for 64x8-bit characters
- Using a +5V single supply

Atmega128 microcontroller is from a family of Atmel group microprocessors with 128K flash memory. The clock frequency of the microcontroller is 14.7456 MHz And, following peripheral are available on the microcontroller:

- General I/O port, these ports can be programmed either to be input or output ports. If DDR register is set to 1s port is set to be output port, otherwise input port when DDR is 0.
- A/D Converter, some port can be programmed to work as A/D converter, sample rate, and reference signal can be set manually too.
- SPI (Serial Peripheral Interface) allows high-speed synchronous data transfer between Atmega128 and peripherals.

In our project, we used SimulIDE simulator to run our code and test the digital clock we created.

Software Requirements

As mentioned earlier, the project will be operating using Atmega128 microcontroller. The project is written fully in C language. Built-in libraries such as avr/io.h, util/delay.h, avr/interrupt.h, and stdio.h were used to complete the project. And, two manually written header files were used in the project as additional libraries. Interrupts.h and lcd_n.h were designed to help the main program look clean and constructive.

Design Considerations and Implementation

Let's discuss the design of whole architecture of the project step by step. As we know, we need to setup the different ports accordingly to accomplish the desired mission. Let's look at output module of the project, which is LCD display. All required functions related with LCD were developed in lcd_n.h header file. Main functions used in the project are LCD_Init(), LCD_Clear(), and LCD_pos(). LCD_Init() function is used to initialize the LCD. LCD_Clear() function is used to clear the data on LCD after writing some message on it. We need to use LCD_Clear() function before writing any message. LCD_pos() function is used to position the cursor on LCD display. Display has two lines, so we need to handle the display as 2D matrix. The function needs two arguments row and column to place the cursor to the correct place. The following is the LCD_pos() function implementation:

```
void LCD_pos(unsigned char row, unsigned char col){LCD_Comm(0x80 | (row + col*0x40));}
```

This implementation is based on HD44780 LCD display, however, we are using SimulIDE LCD display to show output results, that is why we had to change the implementation of above function a bit. Basically, we just need to exchange the attributes in the function. For example, if we wanted to place cursor to 2nd line and 1st character we passed (1,0) arguments to HD44780 display. Here, in simulator, we need to pass (0,1) arguments to achieve to put cursor to the 2nd line. Anyway, we fixed the problem by changing argument list of the function. We might need to use LCD_Shift() function which helps us to move the output of LCD to shift to right or left accordingly. LCD_STR() and LCD_CHAR() functions are used to output the variables to the display. We have LCD_STR() function to show string variables, LCD_CHAR() is used to implement the LCD_STR(), though. So, we actually show character by character on LCD. Next function is used to blink the LEDs, to accomplish this task we need to set Port B to all 1's to turn off and then to all 0's to turn on them, by that giving the sense of blinking. Port D is input port for the project; thus, we need to set it up. All interrupt handling tasks are going to be handled in interrupts.h header file. When the external interrupt is received, corresponding interrupt service routine will be invoked and handled accordingly. For different interrupts, different ISR will be implemented to achieve desired mission. There will be one change in implementation when stopwatch mode will be implemented. Because, as we know clock cycle for atmega128 is 17.8ms when 1024 prescaling is used. However, we need to count milliseconds in stopwatch mode. We have decided to use CTC mode with prescaling of 1024 prescaling. So, we have a clock cycle of around 9.9ms (~10ms) time. We need to show the result in 1/100 resolution. Therefore, we can handle each 10ms to increment millisecond count, and seconds, minutes, hours accordingly. The clock cycle is calculated from the following formulas and steps:

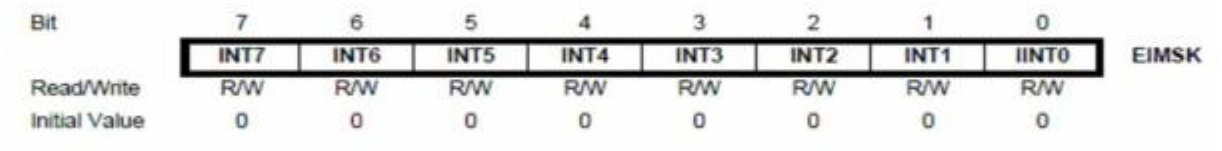
(1) $T = \frac{1}{14.7456*10^6} = 0.067817\, us$ - Period of one cycle of Atmega128 microcontroller

(2) 0.067817 us * 1024 = 69.44 us - Clock cycle after prescaling of 1024

(3) $x = 10\text{ms}/69.44\text{us} \approx 144$ - x is the OCR0 value, so we count each clock interrupt as 10 ms, and 100 clock cycles as 1 second.

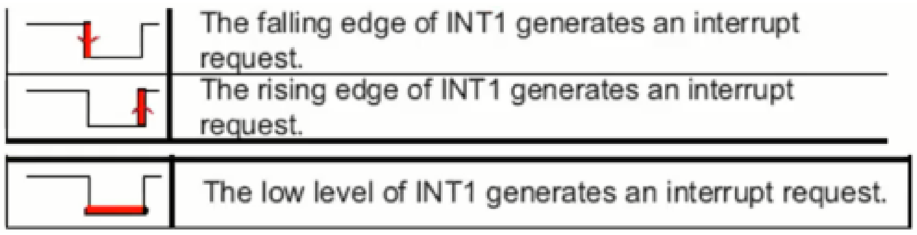
We decided to use 10 ms as 1 clock cycle for our project because it is easier to work with milliseconds for stopwatch mode, thus, for seconds, minutes, and hours too. Our 8-bit clock counter counts till 255 in Normal Mode, so we decided to CTC mode where we give OCR0 144 value. Therefore, it will give interrupts each time counter increments to 144 and reset to 0.

We need to use External Interrupts 0~7 to handle the functionality requirements, and properly implement interrupt service routines (ISRs). We need to work with the interrupt enabling and disabling registers such as EIMSK and EICRA. EIMSK is external interrupt mask register of 8 bits long. Each bit in the register is used to enable/disable only one interrupt. Thus, value stored in the register varies from 0x00 (all interrupts disabled) to 0xFF (all interrupts are enabled).



Bit	7	6	5	4	3	2	1	0	
	INT7	INT6	INT5	INT4	INT3	INT2	INT1	IINT0	EIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Also, the external interrupts are triggered by different edges like falling edge, rising edge, and low level triggers. The basic idea is as following:

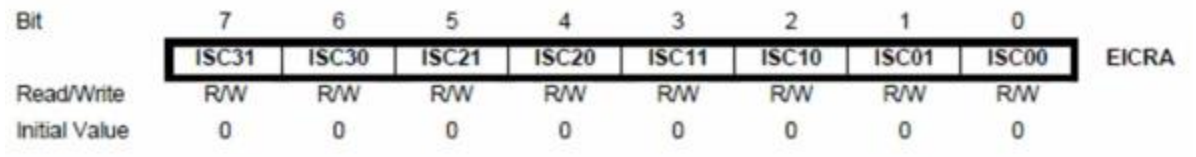


The falling edge of INT1 generates an interrupt request.

The rising edge of INT1 generates an interrupt request.

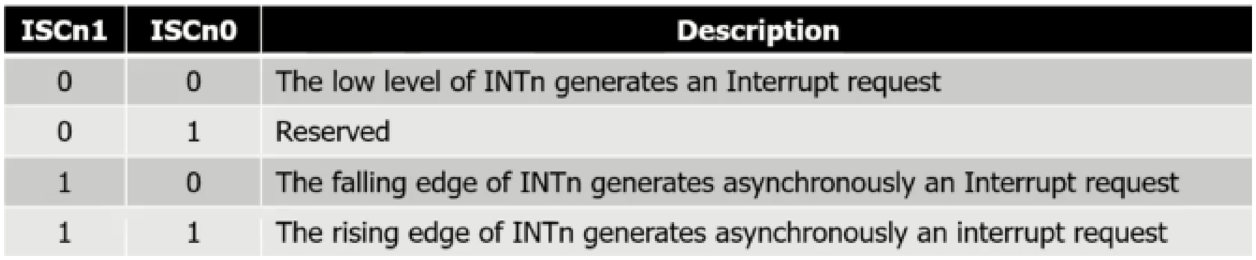
The low level of INT1 generates an interrupt request.

We need to work with EICRA (External Interrupt Control Register A) and EICRB (External Interrupt Control Register B) to choose which edge trigger to use to trigger the interrupts. Each EICR register controls 4 external interrupts each, so to control first 4 interrupts we need to use EICRA. It has 8 bits (2 bits for each interrups), since each interrupt can be triggered in 3 different edges.



Bit	7	6	5	4	3	2	1	0	
	ISC31	ISC30	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit combinations to use different edges are as follows:



ISCn1	ISCn0	Description
0	0	The low level of INTn generates an Interrupt request
0	1	Reserved
1	0	The falling edge of INTn generates asynchronously an Interrupt request
1	1	The rising edge of INTn generates asynchronously an interrupt request

So, if we want to use INT0, INT1 and INT2 interrupts with low-level, falling edge, rising edge requests respectively, we need to store 1111 1000 (0xF8) value in EICRA register.

Conclusion

By the end of the project, our clock will be able to switch between different required modes. And, it is expected to set time and alarm time. Also, it must have stopwatch mode where user can start, pause, and reset stopwatch. The most challenging part of the project is to handle port assignment and interrupt enabling during the runtime of the software. Second, it was difficult to clear the LCD properly with no garbage value and show a message again on the display.

Although, all team members had good knowledge in C programming, we had difficulty in construction whole architecture of the clock. Thus, implementing our idea into a real working project was a bit hard, but challenging at the same time.

REFERENCES

[1] Atmel 8-bit Microcontroller with 128 bytes in-System Programmable Flash