Digital Alarm Clock

Matthew Screws, Binh Ton, Adam Schrader
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
ECE 470/570 Fall 2014 - Computer-Based System Design
Professor Daniel Llamocca
mdscrews@oakland.edu, bqton@oakland.edu, arschrad@oakland.edu

I. ABSTRACT

The final project consists of a digital time clock with sleep alarm capabilities. This device is made using the HCS12 Dragon12-Plus2 in communication with an Arduino Uno with Sparkfun's MP3 Shield connected. The utilization of the Arduino platform was to gain access to community created library functionality for connection with Sparkfun's MP3 Shield. The MP3 Shield allows an MP3 sound file to play as the alarm. The overall purpose and functionality of the project is within today's scope of a modern digital alarm clock device and is the driving decision for implementing this system on the HCS12 Dragon12-Plus2 microcontroller. The main controllers of the Dragon12-Plus2 and Arduino Uno boards have been programmed by using Freescale's CodeWarrior and Arduino's Sketch Pad Compilers, respectively.

II. INTRODUCTION

Our desire to utilize all the skills acquired in the class made us want to make something that we thought would be of interesting and applicable to our daily lives. As such we settled on making a fully functional and programmable alarm clock. This consists of a digital time clock with an MP3 music file as the output for alarm capabilities.

We are using the timer module of the Dragon12 Plus board to construct the real-time clock. The time is displayed utilizing the four 7-segment displays. Ante-meridian and post-meridian are shown using the Dragon12-Plus2 RGB LED. Use of the HEX keypad is the way the user interfaces with the clock functionalities. Keys on the HEX keypad have been defined and allow the user to interface with a menu displayed on the LCD module connected to the Dragon12-Plus2 board. Menu options range from core clock functionality like setting the current time as well as the alarm time. These options are reflected on the LCD module. For the actual alarm we are using an Arduino Uno with a MP3 shield. This is connected via USB and communicates serially using SCI.

III. METHODOLOGY

A. Clock

The clock has been designed around the use of an output compare timer module set to trigger an interrupt every 250ms. This is accomplished by using a prescaler of 128 in the timer module creating a timer clock that runs at 187.5KHz and triggers an interrupt every 46,875 clock cycles. Once in the interrupt, a couple of "if" statements are used to create 1 minute delays. Once a minute has passed, the variable named "min" is incremented by one and the delay counters "mcount" and "ms250_count" to zero to prepare them for the next "minute" count.

Using the RTI module, a real time interrupt was then used to take the "min" variable from the above interrupt and adjust the remaining digits of the clock to correspond to a time of day. This interrupt is triggered every 10.24ms and works by also using a series of "if" statements. After ten minutes have passed, this interrupt will increment the the variable "tmin" to keep track of each set of ten minutes and reset the "min" variable back to zero. Once sixty minutes have passed, the variable "hr" is incremented and while the previous two variable are again reset to zero. The rest of the "if" statements are used to control the hours making sure that the hours and tens of hours digits will adjust properly until twelve is reached and then reset to one. The RTI interrupt is also responsible for changing the status of AM and PM every time the 12th hour is reached. This interrupt prepares the each of the clock digits to be presented by the 7-segment display, and outputs the AM/PM indicator on the RGB LED.

B. Menu Interface through LCD Display

The LCD module is the main visual interface for the menu options the user will be allowed to configure for the core functionality of the digital alarm clock system. Working in conjunction with the HEX keypad for user input, the LCD display changes accordingly to user interaction. On initially startup of the system, the HELP menu will cycle through the different displays configured to show the user the key mapping for the HEX keypad, the functionality of each key, and its uses in our system. After the HELP menu has finished its sequence, the LCD display will initialize on the menu option to set the current time. The system is preconfigured to set the current time to 12:00 A.M. after each reset of the system.

The LCD interface was done through the use of a "case" structure with the current state of the LCD menus as the index to the cases. The "Set Time" menu has index 0, "Set Alarm" has index 1, etc. The index is incremented or decremented depending on the user input through the HEX keypad. Passing the current index value to the function *LCD_display(index)* displays the menu based upon the current menu index. When the user has selected to enter one of the menu options for setting the time/alarm and enters their desired time or alarm setting, the previous menu is displayed. This functionality is capable since the index is left unchanged and is prohibited from changing while the user is in the menu options to set the time/alarm. While the user has entered one of the menu options, they are allowed to cancel and go to the previous menu. This is done through a simple function call to *LCD_display(index)* since the index is left unchanged upon entering the menu options for configuring the time/alarm. A decision to cancel and return to the previous main menu disregards any user input while in the configuration menus. This makes sure that the user only commits appropriate changes

to the time/alarm settings upon pressing the “Enter” key on the HEX keypad. Please reference Section D for further information about user configuration through the use of the HEX keypad. Displayed below are pictures of the LCD module displaying various menu displays.

C. 7-segment Time Display

The 7-seg time display is initialized to 12:00 as a standard digital clock would be found. Programming the time using the HEX keypad incorporates a series of case statements. First each digit is defined as either the tens of hours, hours, tens of minutes and minutes. Once the user has loaded the time into the LCD display and pressed enter, the values are loaded into the programs variables. These variables are then passed into RTI interrupt 7 handler which decides how to increment the 7-seg using a series of IF statements. These IF statements ensure proper values are loaded into the 7-seg display for the end user. These rules include the tens of hours being one or not displaying at all. The hours go to zero, one, or two if the tens of hours is one. Otherwise the hours can go from one to nine. The tens of minutes can be equal to zero through five, and the minutes can go to nine. These IF statements describe a standard twelve hour clock. The digits are then pulled from the RTI 7 interrupt and loaded into the output compare interrupt OC6ICR where they are sent to their proper place holder. The OC6ICR is triggered every 1.01ms. This keeps each of the 7 segments illuminated for equal amounts of time, which gives the appearance to the user that they are all on simultaneously with each at the same brightness. Pictured below are the initialized 7-seg displays to 12:00 on startup/reset.

D. User Input through HEX Keypad

Access to prior libraries was used to get the HEX keypad to function. For the HEX keypad, the *getkey()*, *keyscan()*, and *wait_keyup()* functions were used. The *keyscan()* function is constantly reading each of the keycodes and puts this value in PORTA. If nothing is being pressed then it returns a value of 0x10 or 16. The *getkey()* function waits for a button to be pressed and goes through a debounce while the *wait_keyup()* function waits until the user has released the key.

A breakdown of the mapping for the HEX keypad and each digit’s functionality in the system is as follows: Keys 0-9 are used as input for the time and alarm settings. Keys ‘A’ and ‘B’ is for maneuvering up and down through the main menu displays, respectively. Key ‘C’ acts as a backspace key for users to delete the previous entry when setting the time or alarm. Key ‘D’ functions as a cancel button to disregard current user input and return to the previous main menu display based upon the menu index (reference Section B for further information about the LCD menu interface). Key ‘*’ acts as the enter button for confirmation of user selection and input. This key is used for selection into the different menu options as well as finalizing user input for the time and alarm. Key ‘#’ is not mapped and thus has no functionality in the system.

E. Sparkfun's MP3 Shield connected to an Arduino Uno

The purpose of utilizing a second microcontroller for the MP3 sound file as the alarm was based solely on the ease and availability of obtaining this functionality. The Arduino platform is a powerhouse in the field of microcontrollers and the Arduino Uno was a definitive choice when it came to which Arduino was to be used in this project. The option to expand the capabilities of the Arduino Uno through the addition of Arduino Shields allows for an even more robust and user-friendly platform. The shield that was utilized in this project was Sparkfun's MP3 Player Shield that connects on top of the Uno's PCB. The MP3 Shield pulls MP3 files from a microSD card. The audio decoder IC on the shield operates in slave mode and receives its bitstream through SPI. After the stream has been decoded, the audio is sent out to a 3.5mm headphone jack that is connected to stereo speakers to sound the music alarm.

We are using the powerful SFEMP3Shield Arduino Library which acts as a driver for the audio decoder IC on the MP3 Player Shield. The library allows for a straightforward implementation of playing an MP3 file. The Dragon12-Plus2 board simply has to establish serial communication between itself and the Arduino Uno. This is done through connecting the SCI1 pinouts for Tx and Rx to the Rx and Tx pinouts on the Arduino Uno and communicating at the same baud rate. The microSD card in the MP3 Player Shield has nine different MP3 files stored onto it. To play a track, the Uno simply has to receive a character of the track number. To stop the song from playing, a character of 's' is sent. For example, to play track 5 on the microSD card, the Dragon12 sends a character of "5" to the Uno and a character "s" is sent to stop the alarm. The system was programmed to play a random song out of the nine on the microSD as the alarm.

IV. TESTING AND RESULTS

During assembly, the program was tested several times along the way to check for any bugs or glitches in the operation. Along the way several issues were found in the 7 segment display, alarm trigger module, and the display menu functions. These issues were caught by thorough testing all conditions and potential loopholes that could be thought of by visually inspecting the outputs on the LCD and 7 segments, by listening for the output of the music from the MP3 shield, and by using a clock to time the 1 minute intervals of the output capture function of the clock timer.

Some major problems encountered during construction of the clock include finding a way to keep the clock displaying on the 7 segments at a stabilized brightness while allowing the user interact with the clock menus and having the alarm trigger only once allowing the program to return to previous main screen function to accept another command from the user. As for the first problem, it quickly became evident quickly that the user interface display and clock display must be separated for proper operation. These two sections of code conflict, only allowing one device to be displayed/used at a time. In order to display all 7 segments constantly, a loop must be run to keep each lit at a uniform period of time. Since the user interface is waiting for the user to input data, the 7 segments would not update in a proper fashion, leaving the last lit digit much brighter than all the rest. to eliminate this problem, both the RTI interrupt and output compare register OC6 were implemented to multitask the operations the HCS12 processor had to take on. Both interrupts stop the main program continuously for very short durations allowing for the 7 segments to be displayed at the desired uniform rate, while at the same time allowing the user to interact with the clock menu options.

The second problem was encountered while testing the alarm. The test and trigger for the alarm were initially placed inside RTI interrupt 7. This was so the alarm variables were continuously compared at a uniform fashion and at a rate fast enough to trigger with minimal delay. The problem with triggering the alarm there is that this stops the rest of the interrupts from performing their function and does not allow for the proper display of the clock or provide a means for the user to turn off the alarm. No other function can happen until the program returns from the interrupt. On the other hand placing the code for the alarm in the main program loop would not allow for the alarm to actually sound. This is due to the fact that the main program loop is always waiting for the user to press a key on the hex keypad. To correct this error, the function *alarm_trig( )* was created and implemented in RTI interrupt 7. This function uses a system of "if" statements to both trigger the alarm and to set flags letting the program know that the alarm was played once and should not be sounded again until alarm time is encountered again in another 24 hours. The main program was also modified so the program would not wait until the user entered a key but rather would respond to a command once the key was pressed. This was done by swapping out the *getkey( )* command provided by the EGR 280 library with the *keyscan( )* command encased in a conditional while loop which checked for the alarm triggered flag. This allowed the main program to break from its main function and quickly operate the alarm.

V. CONCLUSION

Overall, the project was a tremendous success, both in its operations and in the lessons obtained from taking on such an involved project. The main lesson learned was that

seemingly simple tasks quickly became surprisingly complicated as other simple tasks became intertwined. There were few issues individually in getting the 7-segment display to display all segments at the same intensity, the hex keypad to retrieve user input without holding up other parts of the program, the LCD interface to display the appropriate menus, the clock to function properly with the use of interrupts, or get the alarm to go off. However, when you combine all these facets of the project these simple tasks start to interfere with one another. A major portion of the project was finding and resolving bugs associated with improper logic or timing issues.

One feature we would add in the future is a snooze alarm. This could be done by generating a high or low edge over a certain amount of clock cycles and then using edge detection to trigger an interrupt for the alarm. The width of this pulse could be generated using a formula that would get increasingly shorter until it is perpetually on, and the user would have to turn the alarm off.

VI. References

[1] Budakoti, A. (2014, May 9). How to generate a random number in C? Retrieved from http://stackoverflow.com/questions/822323/how-to-generate-a-random-number-in-c

[2] Haskell, R., & Hanna, D. (2008). Hex Keypad. In *Learning by example using C: Programming DRAGON12-Plus using CodeWarrior* (Second ed., p. 28). Rochester: LBE Book

[3] Huang, H. (2006). *The HCS12/9S12: An introduction to software and hardware interfacing*(Second ed.). Clifton Park, NY: Delmar/Thomson Learning.

[4] Porter, B. (2012, January 28). Sparkfun MP3 Shield Arduino Library. Retrieved December 2, 2014, from http://www.billporter.info/2012/01/28/sparkfun-mp3-shield-arduino-library/