Design and Implementation of a Stopwatch on EFM8BB3

Camille Chow and Ostap Voynarovski

Abstract–A stopwatch with LCD display was designed and implemented using an EFM8 Busy Bee microcontroller, with all software coded in C using the SimplicityStudio^{TM} IDE. The stopwatch was designed to display total time in seconds (with tenth of a second precision), record and display lap times, and toggle on and off with a push button, with LEDs flashing at startup. Software to control the timers and display of the microcontroller were successfully implemented separately, however attempts to combine these two aspects were unsuccessful.

I. Introduction

This project utilized two main concepts in computer architecture: interrupts and assembly code. In system programming, an interrupt flag tells the processor that an event as occurred that needs to be handled immediately. The processor responds by saving its current state, pausing its current activities and jumping to an interrupt service routine that tells it how to deal with the interrupt. Three interrupt handlers were written for this project: one to increment the total time, one to record lap times, and one to start/stop the stopwatch.

Assembly code is a low level programming language in that the software instructions are very close to the architecture's machine code instructions. In this project we attempted to write an assembly code based on the 8051 instruction set that would flash the LEDs on the EFM8.

The following describes how our software incorporates the above concepts to work towards our design goal.

II. Design

The goal for this project was to have the EFM8 emulate a stopwatch by displaying time on its LCD display with precision to the tenth of a second. One push button would be used as a start/stop button, which would flash the EFM8's LEDs when the stopwatch was first started. The other push button would be used to record lap times, which would be displayed under the total time.

The design for the stopwatch involved three main software elements: timing, display, and assembly code. The timing aspect of the project utilized the built-in timers on the EFM8 as well as interrupt handling. The display aspect utilized the EFM8's LCD screen to interface with the user. The assembly portion of the project was intended to flash the LEDs (red, red, yellow, green) once the stopwatch was started up.

A. Timing and Interrupts

The timing portion of the project involved three interrupts: one timer interrupt and two external interrupts:

Timer0 Interrupt - The EFM8's Timer0 was set up such that an interrupt flag was thrown every millisecond, calling an interrupt handler that would increment a counter. When the counter reached 100 (representing 100 ms), the global variable totalTime was incremented, and the counter was reset.

External Interrupt 0 (INT0) - For the external interrupts, an interrupt flag was set every time one of the push buttons was pressed. The INT0 handler was accessed whenever push button 0 on the EFM8 was pressed. The handler would store the current time held by totalTime in an array of lap times.

External Interrupt 1 (INT1) - Every time push button 1 was pressed, the INT1 handler disabled the Timer0 interrupts, preventing totalTime from incrementing until push button 1 was pressed again.

B. Display to LCD

To get the screen to display things that we would want it to display we adapted example code from the “PowerModes” example found in the IDE this program essentially created a few menus and uses the BTN0 and BTN1 to toggle between them. For our purposes much of this functionality is not necessary. We were mostly interested in the code which they used to read button presses and display to the screen. Much of their code was removed to make it easier to debug and use.

The most important part of the code was the DrawScreenText() function. This function allows the user to input a variable or array of characters and the location vertically. Then it will take that data and display it to the screen.

This method allows for up to 16 lines to be printed on the display at once, and one can simply index the line height via a counter. The functionality of the displaying to the screen was not fully integrated with the stopwatch functionality initially because we got the LCD to work before the timer interrupts.

C. Assembly Code

In the Assembly portion of this project, we tried to get an LED to light up Red, Red, Yellow, Green. This can be accomplished by writing assembly code that will light on the various RGB lights on to produce the specified colors. To achieve this we tried to turn on pin numbers 1.4,1.5,1.6 on and off.

Initially, code was written to turn only one pin on and off. A delay was attempted to be created to allow for the assembly to keep it on and off. We believe that this code could have worked, however, the issue came when we tried to integrate the assembly into the C code and call them. It was believed initially that the assembly would compile if we named it main.asm, but after much hassle we realized that the file was being completely ignored.

We found out that there is a way where you can write C code with assembly routines inside it. This seemed to be the optimal option, however we were unable to make it work properly.

III. Conclusion

The code to control the timing and interrupts was able to flash an LED on the board with a period of 100 ms, and toggle this LED on and off using one of the pushbuttons. The other push button was able to toggle a different color LED separately. Global variables were then added for totalTime and lapTimes, and incorporated into the interrupt service routines in a way that would accomplish the goals of the project.

In a separate code, the LCD screen

was made to display an incrementing integer with each button click. The LCD aspect was achieved using the DrawScreenText() function. At one point, the screen printing was fully functional, however there were some issues that were encountered. Namely, the Init file and the Configurator were changed to get the functionality of the timer portion of the code to interface with the LCD display. These were two separate files and were supposed to be able to join together, however upon attempting this, the LCD display program was broken and it would no longer compile. Attempts were made to recover past versions, but were ultimately unsuccessful.

Separately, these two codes were able to accomplish their respective tasks, however when attempting to integrate the two codes, the IDE would no longer compile the code, either due to discrepancies in the initDevice files, unforeseen interference of the board's peripherals, or some other unknown reason. The assembly code did not compile either, and therefore the flashing LEDs were unable to be implemented. In these ways, we were unable to achieve the original goals of the project, however the results do demonstrate a working knowledge of timing, interrupts and interfacing with the user.