Design and Implementation of a Stopwatch Using the 8051 Architecture on a EFM8BB3 Microcontroller

Joseph Bentivegna and Nikola Janjušević

***Abstract*- A stopwatch was created from an EFM8BB3 microcontroller which uses the 8051 architecture. The project allowed the user to start and stop a timer, as well as log lap times of an event. Interrupts were used to track the overflow of the timer peripheral and the results were presented to the user through the LCD peripheral. The majority of the project was implemented in C code, however the ready-set-go functionality of the LED was programmed in assembly. Code design was optimized for efficiency and readability.**

I. INTRODUCTION

The stopwatch interface consists of two push buttons below an embedded LCD. The two push buttons allow the user to start/stop the stopwatch, and add lap values below the main time. The toggling of the stopwatch from stop to start triggers a sequence of 'start-up' LEDs. The bulk of the stopwatch is coded in the C programming language, however, the ready-set-go LED sequence is written in Assembly language, called from within the main C program. The timing of the stopwatch utilizes the timer peripherals from the microcontroller to ensure a precision to one tenth of a second.

II. DESIGN

A. Peripheral Implementation and External Interrupts

The functionality of the stopwatch requires the use of two main peripherals embedded in the microcontroller. The first is the timer peripheral which allows a higher level of precision by using a clock source derived from the crystal oscillator and separate from the operations of the central processing unit (CPU). The second is the Serial Peripheral Interface (SPI) bus which facilitates communication to the LCD.

The timer peripheral implemented uses an 8 bit timer (timer 0) in automatic reload mode. This means that every instance that the timer decrements from the start value to zero, an interrupt flag is set high and the start value is reloaded into the timer's register. The stopwatch implementation of timer 0 in reload mode uses the system clock prescaled by 1/48 as an input clock to cause an interrupt flag to fire every millisecond, derived by the following governing equation:

$$F_{\text{TIMER0}} = \frac{F_{\text{Input Clock}}}{2^8 - \text{TH0}} = \frac{F_{\text{Input Clock}}}{256 - \text{TH0}}$$

Fig. 1. frequency of timer 0 in auto-reload mode

Every one-hundred times the interrupt flag for timer 0 is fired the global time variable is incremented to give accuracy to one tenth of a second as per design choice. Future implementations could use the given operation of timer 0 to utilize the full precision of thousandths of a second.

The LCD implemented utilizes the framework of built in code from the Simplicity Studio IDE, in particular the Voltmeter demonstration code. This example lays the groundwork for writing to the display. The configuration for the SPI bus was minimal and only to ensure the simultaneous functionality of the LCD and timer peripherals. The implementation used in writing to the LCD is discussed in the next section of Design.

The external button interrupts play a key role in the fluidity of the stopwatch user interface. Each button is configured to pieces of hardware separate to the CPU such that an interrupt is fired upon the negative transition of the button, in this case as soon as the button is pressed the interrupt fires. This implementation free's the CPU from polling for a button press and ensures the responsiveness required of a stopwatch. A notable feature of the button interrupts is the dynamic disable and enable of the interrupts. Each time a button interrupt fires, the interrupt is disabled such that multiple "accidental" interrupts are not recognized. The button interrupts are then re-enabled at the end of the main stopwatch while-loop to resume their functionality. The result of this dynamic disable and reenable is an effective debouncing of the buttons.

B. Software Design

The software implementation of the stopwatch is shaped largely by three factors: initialization of hardware, communicating through interrupts, and writing to the display.

The initialization becomes especially important for the peripherals mentioned in the previous section. By choice of framework in the Voltmeter demonstration code the LCD came readily configured. This convenience came with drawbacks of conflicting interrupts enabled, such as the port-match interrupts and pin initializations unfavorable to the required use of the stopwatch. Through experimentation, the Voltmeter framework was able to be stripped to its bare essentials such that the LCD implementation was preserved and no code existed to conflict with the implementation of desired stopwatch functionality.

The initialization thus consists of the disabling of the default-on Watchdog timer (to allow un-interrupted operation of the microcontroller), set up of timer 0 to default state of off (non-recognition of interrupt flags), and an initialization of button interrupts.

The limitation of stack space allocated to the interrupt calls in conjunction with the high complexity function calls required to write to the LCD required a different approach to changing the state of the stopwatch through interrupts. No meaningful function could feasibly be called through any interrupt and therefore global binary valued variables exist to serve as pseudo function calls. These pseudo

function calls are created by changed the value of a global variable to a 1 or 0 to communicate that a function in the stopwatch main while loop is to be called on the next loop. This shapes the code to be heavily focused on global variable checking within if-else statements to see if a function should be called. As a result, a total of three functions are present in the stopwatch main loop: one to check the current time value and draw it to the LCD, one to check if a new lap has been created and draw it to the LCD, and one generate the ready-set-go LED sequence. Each function internally checks for a change of state in the global variables and responds accordingly.

The writing to the LCD is provided by the Voltmeter demonstration framework. This code allows rows of text to be written to the screen from arrays of characters. The limitation of only writing arrays of characters requires supplementary functions to convert the global time integer to a corresponding character array. The limitation of writing to rows of the display instead of specific coordinates on the display constrained the stopwatch to a left-justified interface.

C. LED Feedback Through Assembly code

The stopwatch was built to give the user visual feedback whenever they started the timer. Whenever the stop/start button was pressed when the timer was stopped, the LED would flash red-red-yellow-green before beginning the counting. This functionality was programmed in assembly code within an .A51 file. Delays were made within assembly by setting register values to 255 and then decrementing them my repeated calls to the DJNZ command. By nesting these delays, a significant pause can be utilized to keep the LED's on for a relevant amount of time. Functions were created to individually turn on each color light. These functions were called in in the primary function named start. Start was instantiated with the capability to be called within our C code. The function simulates the blinking aspect of the lights by periodically turning a color on, then off, with a delay. This assembly function was called within the stopwatch main function. It is located inside an if statement that checks if the timer is currently off.

III. CONCLUSION

The designed code allows the EMF8BB3 microcontroller to act as an accurate, functioning stopwatch. The project demonstrated the ability to access the hardware's timing capability and use it to count to a precision of 1/10th of a second. It also facilitated the creation and usage of interrupts so that lap times could be recorded from the current stopwatch time. The LED's ready-set-go feature at the start of a new timer required a function call to assembly code which directly accesses memory locations. Overall, the project strengthened skills in software design, interrupts, peripherals, and general microcontroller architecture.