EE 3420: 7 Segment Display Stopwatch Laboratory Report 2

Second Laboratory Report for
EE 3420: Microprocessors
Section 001

Submitted by

Rabeea Fatima
A05420944

Ingram School of Engineering
Texas State University – San Marcos

10/02/2025

Abstract

This lab shows the implementation of a stopwatch using the Blackboard's 7-segment displays and push buttons. The stopwatch is programmed in C and works with three buttons: start, stop, and reset. The display increments in hex digits, with each button mapped to a function that controls the counter. By combining the 7-segment display hardware with simple logic and delays, the stopwatch works as expected.

List of Symbols and Abbreviations

LED – Light Emitting Diode
FPGA – Field Programmable Gate Array
ARM – Advanced RISC Machine
GPIO – General Purpose Input/Output
USB – Universal Serial Bus
BCD – Binary Coded Decimal

Table of Contents

INTRODUCTION..1
EXPLANATION OF CODE SECTIONS...1
CONCLUSION..3
REFERENCES...4
APPENDICES..4

List of Figures

Figures: Page

1. Register definitions 2

2. Display function 2

3. Stopwatch logic 3

INTRODUCTION

In this lab we used the Blackboard's GPIO hardware, which includes switches, push buttons, and 7-segment LED displays, to create a stopwatch. The board connects to the FPGA and ARM processor and allows memory-mapped access to the peripherals. Vitis was used to write and run the C program.

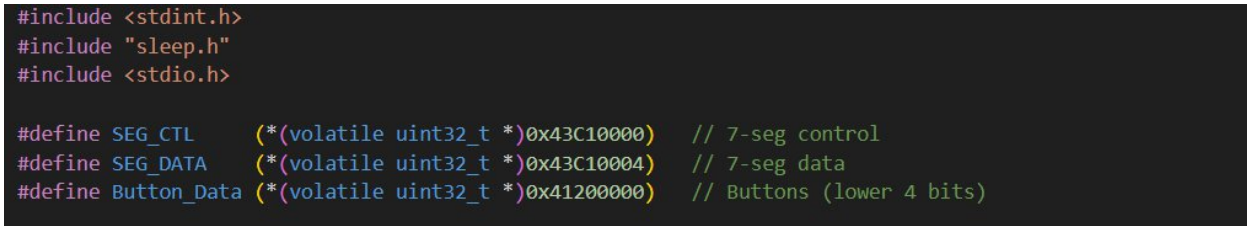
The objective was to program a stopwatch that increments once every second while showing the value on the 7-segment display. Three buttons were used:

- BTN0 → Start the stopwatch
- BTN1 → Stop the stopwatch
- BTN2 → Reset the stopwatch to zero

By addressing the hardware through memory registers, we controlled the display and read button states to manage stopwatch logic.

EXPLANATION OF CODE SECTIONS

The program starts by including the required libraries (stdint.h, stdio.h, sleep.h) for integer types, formatted printing, and timing delays. Then we define macros for the 7-segment control register, 7-segment data register, and button input register. These point to the hardware memory addresses.



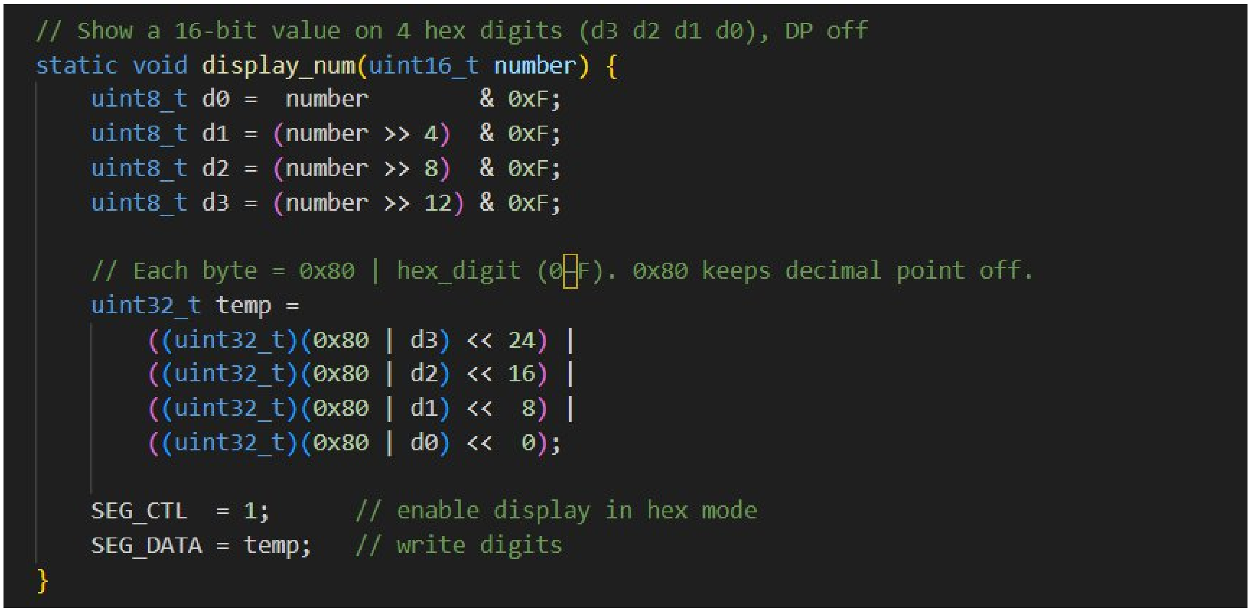
```
#include <stdint.h>
#include "sleep.h"
#include <stdio.h>

#define SEG_CTL     (*(volatile uint32_t *)0x43C10000)   // 7-seg control
#define SEG_DATA    (*(volatile uint32_t *)0x43C10004)   // 7-seg data
#define Button_Data (*(volatile uint32_t *)0x41200000)   // Buttons (lower 4 bits)
```

Figure 1. Register definitions.

Display Function:

The function display_num() is used to show a 16-bit number across the four digits of the 7-segment display. Each digit is masked and shifted into the correct byte. We also use 0x80 to keep the decimal point off.



```
// Show a 16-bit value on 4 hex digits (d3 d2 d1 d0), DP off
static void display_num(uint16_t number) {
    uint8_t d0 =  number        & 0xF;
    uint8_t d1 = (number >> 4)  & 0xF;
    uint8_t d2 = (number >> 8)  & 0xF;
    uint8_t d3 = (number >> 12) & 0xF;

    // Each byte = 0x80 | hex_digit (0-F). 0x80 keeps decimal point off.
    uint32_t temp =
        ((uint32_t)(0x80 | d3) << 24) |
        ((uint32_t)(0x80 | d2) << 16) |
        ((uint32_t)(0x80 | d1) <<  8) |
        ((uint32_t)(0x80 | d0) <<  0);

    SEG_CTL  = 1;      // enable display in hex mode
    SEG_DATA = temp;   // write digits
}
```

Figure 2. Display function

Main Stopwatch Logic:

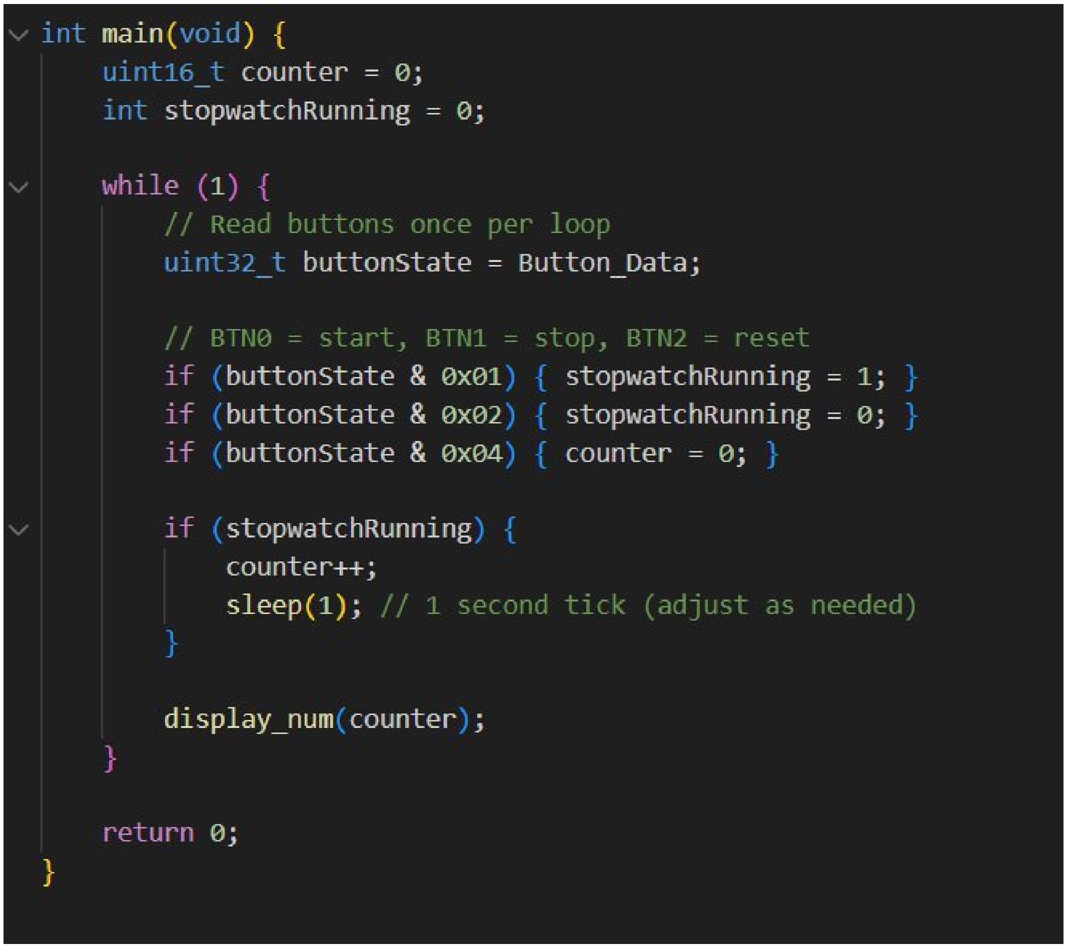
In main(), a counter variable holds the current stopwatch value, and stopwatchRunning is a flag that indicates if the stopwatch is active.

- If BTN0 is pressed, the stopwatch starts.
- If BTN1 is pressed, the stopwatch stops.

- If BTN2 is pressed, the counter resets to zero.

When the stopwatch is running, the counter increments once per loop iteration, with a sleep(1) call to make it increase every second. After each update, the value is displayed with display_num(counter).

The program stays in a while(1) loop, continuously checking for button presses and updating the stopwatch accordingly.



```c
int main(void) {
    uint16_t counter = 0;
    int stopwatchRunning = 0;

    while (1) {
        // Read buttons once per loop
        uint32_t buttonState = Button_Data;

        // BTN0 = start, BTN1 = stop, BTN2 = reset
        if (buttonState & 0x01) { stopwatchRunning = 1; }
        if (buttonState & 0x02) { stopwatchRunning = 0; }
        if (buttonState & 0x04) { counter = 0; }

        if (stopwatchRunning) {
            counter++;
            sleep(1); // 1 second tick (adjust as needed)
        }

        display_num(counter);
    }

    return 0;
}
```

Figure 3. Stopwatch logic

CONCLUSION

The code successfully created a functional stopwatch using the Blackboard’s 7-segment display and push buttons. The while(1) loop keeps the stopwatch responsive by always checking button states. Using sleep(1) gave a one-second increment delay. The stopwatch works as expected with start, stop, and reset controls. This lab demonstrated how to map hardware registers to software and use them in simple embedded C programs.

REFERENCES

[1] Dr. Welker’s example code for Lab 2.
[2] ChatGPT
[3] Class notes

APPENDICES

```
#include <stdint.h>
#include "sleep.h"
#include <stdio.h>

#define SEG_CTL     (*(volatile uint32_t *)0x43C10000)   // 7-seg control
#define SEG_DATA    (*(volatile uint32_t *)0x43C10004)   // 7-seg data
#define Button_Data (*(volatile uint32_t *)0x41200000)   // Buttons (lower 4 bits)

// Show a 16-bit value on 4 hex digits (d3 d2 d1 d0), DP off
static void display_num(uint16_t number) {
   uint8_t d0 =  number        & 0xF;
   uint8_t d1 = (number >> 4)  & 0xF;
   uint8_t d2 = (number >> 8)  & 0xF;
   uint8_t d3 = (number >> 12) & 0xF;

   // Each byte = 0x80 | hex_digit (0–F). 0x80 keeps decimal point off.
   uint32_t temp =
      ((uint32_t)(0x80 | d3) << 24) |
      ((uint32_t)(0x80 | d2) << 16) |
      ((uint32_t)(0x80 | d1) <<  8) |
        ((uint32_t)(0x80 | d0) <<  0);

    SEG_CTL  = 1;      // enable display in hex mode
    SEG_DATA = temp;   // write digits
}

int main(void) {
    uint16_t counter = 0;
    int stopwatchRunning = 0;

    while (1) {
        // Read buttons once per loop
        uint32_t buttonState = Button_Data;

        // BTN0 = start, BTN1 = stop, BTN2 = reset
        if (buttonState & 0x01) { stopwatchRunning = 1; }
        if (buttonState & 0x02) { stopwatchRunning = 0; }
        if (buttonState & 0x04) { counter = 0; }

        if (stopwatchRunning) {
            counter++;
            sleep(1); // 1 second tick (adjust as needed)
        }

        display_num(counter);
    }

    return 0;
}
```