EE 3320 7 Segment Display, Stopwatch Laboratory Report

Third Laboratory Report for
EE 3320: Microprocessors
Section Number 251

Submitted by

Kataylna Hinojosa
A05371527

Ingram School of Engineering
Texas State University – San Marcos

02/26/2026

Abstract

Laboratory 3 for EE 3320: Microprocessors used memory-mapped I/O in C to interface the Blackboard's 7-segment display and push buttons to create a stopwatch. This lab's goal was to build a counter that was shown in Binary Coded Decimal (BCD) format and use three pushbuttons—Start, Stop, and Reset—to operate it. To gain access to the seven-segment display registers as well as button registers via designated memory addresses, a C application was created in Vitis. The stopwatch effectively increases once every second, halts when instructed, and only resets to zero when paused. Memory-mapped register utilization, bit concealment, BCD digit retrieval, and real-time integrated system control were among the principles that were reinforced in this lab.

List of Symbols and Abbreviations

LED – Light Emitting Diode

FPGA – Field Programmable Gate Array

ARM – Advanced RISC Machine

GPIO – General Purpose Input/Output

I/O – Input/Output

BCD – Binary Coded Decimal

Table of Contents

Table of Figures

INTRODUCTION

In Laboratory 3, a stopwatch had to be implemented utilizing memory-mapped I/O in C and the Blackboard's seven-segment displays and push buttons. Creating a counter that shows values in four-digit Binary Coded Decimal (BCD) format and controlling the stopwatch with three push buttons—BT0 to begin the count, BT1 to stop it, and BT2 to reset it to zero only if the stopwatch is stopped—were the goals of this lab. During operation, the stopwatch constantly updates the display and increments once every second. Register-level hardware utilization, bit masking, digit manipulation, BCD conversion, and polling techniques for real-time embedded system control were among the ideas that were emphasized in this lab.mming, and pointer-based hardware access were all reinforced in this lab.

HARDWARE AND MEMORY MAPPING

Through memory-mapped I/O, in which every hardware device is given a unique memory address, the Blackboard facilitates interaction among the ARM processor and FPGA peripheral

```c
#include <stdint.h> // so I can use the integers I want
#include <stdio.h> // so I can use sprintf
#include "sleep.h" // Allows me to PAUSE the processor a specific period of tim
#define SEG_CTL *((uint32_t *) 0x43c10000)
//7-seg digit data register
#define SEG_DATA *((uint32_t *) 0x43C10004)
//7-seg decimal point data reg
#define SEG_DP *((uint32_t *) 0x43C10014)
#define Button_Data *((uint32_t *)0x41200000) // buttons are the lower 4 bits
```

Figure 1: Memory-Mapped Register Definitions for 7-Segment Display and Push Buttons

The 7-segment display, as well as push buttons in this lab, were directly accessible via the hardware addresses that were allocated to them. The control register, a digit information register, and a decimal point register are used by the CPU to connect with the display, while a button data register is used to access the push buttons. To enable the ARM processor to read and write values straight to the FPGA hardware, such memory addresses were specified in the C program as 32-bit pointers. Since just three buttons (BT0, BT1, and BT2) were needed, bit masking was utilized to separate only the lowermost four bits of the button register.

PROGRAM IMPLEMENTATION

Publishing a single number to one of the four locations on the seven-segment display is the responsibility of the Display_Digit function. The current screen value is first read from SEG_DATA after the control register has been activated.

```
//This function was made katayIna but with the the templet of welkers code
// this function pt 0ne number on one digit postion
void Display_Digit(uint8_t pos, uint8_t val) {
    uint32_t current_display = 0;

    SEG_CTL = 1;
    current_display = SEG_DATA;// read current display value
    // Clear the position we want to write to
    if (pos == 1) {
        current_display &= 0xfffffff0;
    }
    else if (pos == 2) {
        current_display &= 0xffff0ff;
    }
    else if (pos == 3) {
        current_display &= 0xfff0ffff;
    }
    else if (pos == 4) {
        current_display &= 0xf0ffffff;
    }
    // Put the new digit in the cleared pos
    current_display |= ((val & 0xF) << ((pos-1) * 8));
    current_display |= 0x80808080;

    SEG_DATA = current_display;// write back to the display
}
```

Figure 2: Single-Digit Display Function

The function then uses bit masking to clear only the particular digit location that has to be altered. Various masks are applied based on the position chosen because each digit takes up 8

bits in the 32-bit register. The new digit value is put in utilizing bit shifting ((pos-1) * 8) to position it correctly within the register once the desired location has been cleared. To make sure that just the lower four bits are used, the value is hidden with 0xF. Lastly, only the chosen digit is updated, and the others remain unchanged when the revised value is entered back into the display register.

To properly show a complete decimal number in Binary Coded Decimal (BCD) representation on a 7-segment display, the Disp_BCD function breaks it down into individual digits.

```
//this function takes a whole number and displays all the digits automatically
void Disp_BCD(uint16_t value) {
    uint32_t ones, tens, hundreds, thousands;

    ones = value % 10;// Extract the ones place digit (rightmost)
    tens = (value / 10) % 10; //Extract the tens place digit (divide by 10, then get remainder)
    hundreds = (value / 100) % 10; //// Extract the hundreds place digit
    thousands = (value / 1000) % 10; ///Extract the thousands place digit
    //Display each digit at its position (position 1 = rightmost)
    Display_Digit(1, ones); // display in pos 1
    Display_Digit(2, tens);// display in pos 2
    Display_Digit(3, hundreds); // display in pos 3
    Display_Digit(4, thousands);// display in pos 4
}
```

Figure 3: BCD Conversion and Multi-Digit Display Function

The modulus operator % 10 is used to extract the ones digit, and modulus operations are applied after the value has been divided to extract the tens, hundreds, and thousands digits. This guarantees the digital stopwatch counts in actual decimal instead of hexadecimal. The function places each digit in its proper display position by calling Display_Digit() four times after extracting each digit. The stopwatch value can be automatically structured and shown across all four numbers thanks to its modular architecture.

The stopwatch logic for control is implemented via the main function.

```
while(1){
    button_val = Button_Data & 0x0F;
    if(button_val & 0x01 ) // if button zero is pressed (start)
    {
       is_running = 1 ;

    }

    if(button_val & 0x02) // if button 1 is pressed (pause)
    {
       is_running = 0;
    }

    if((button_val & 0x04) && (is_running == 0))// if button 2 i
    {
        count = 0; // reset the inner counter ( the actual value
   Display_Digit(1, 0);  // Clear position 1
   Display_Digit(2, 0);  // Clear position 2
   Display_Digit(3, 0);  // Clear position 3
   Display_Digit(4, 0);  // Clear position 4
    }


    if(is_running == 1 ) { // if the timer is going
        count = count +1 ; // undate value of the count by 1
        usleep(1000000) ; // puase for 1 sec
    }
   Disp_BCD(count);
   usleep(5000);
}
```

Figure 4: Main Function – Stopwatch Control Logic

There are three variables declared: Count maintains the stopwatch value, is_running keeps track of the fact that the stopwatch is running, and button_val stores the button's current state. The program repeatedly checks the push button registers within the infinite loop, isolating the lower four bits by masking it with 0x0F. The stopwatch sets is_running to 1 to start counting if BT0 is pressed. The stopwatch pauses by setting is_running to 0 when BT1 is pressed. All display numbers are cleared, and the counter resets to zero if BT2 is pressed down while the stopwatch is halted. The counter uses usleep(1000000) to increase once every second while the stopwatch is operating.

CONCLUSION

Laboratory 3 effectively illustrated how to use memory-mapped I/O on the Blackboard platform to interface push buttons using a 7-segment display. Direct register access was used to

implement the stopwatch, enabling real-time communication between the ARM CPU and the FPGA peripherals. Starting the count with BT0, halting it with BT1, and resetting it to zero with BT2 only when the stopwatch stopped were all necessary operations that the program executed appropriately. The counter showed precise figures in Binary Coded Decimal (BCD) format spanning four digits and increased once every second.

Key ideas in embedded systems, including as bit masking, register modification, modular function design, pointer-based hardware access, and real-time polling approaches, were emphasized in this lab. The stopwatch's successful operation confirmed that the multi-digit display control and ARM–FPGA interface were implemented correctly. All things considered, this lab improved knowledge of embedded C programming and low-level hardware interface, two crucial abilities for designing microprocessor systems in the future.

REFERENCES

[1] Mr. Welker’s Laboratory 3 Example Code, EE 3420 Laboratory Manual, and the student's example report

[2] Grammarly AI on help with grammar and spell checking

APPENDICES

[1] Kartaylna’s Code.

///Katyalna H

```
 // lab3.c

#include <stdint.h> // so I can use the integers I want

#include <stdio.h> // so I can use sprintf

#include "sleep.h" // Allows me to PAUSE the processor a specific period of time.

#define SEG_CTL *((uint32_t *) 0x43c10000)

//7-seg digit data register

#define SEG_DATA *((uint32_t *) 0x43C10004)

//7-seg decimal point data reg

#define SEG_DP *((uint32_t *) 0x43C10014)

#define Button_Data *((uint32_t *)0x41200000) // buttons are the lower 4 bits


//This function was made kataylna but with the the templet of welkers code

// this function pt 0ne number on one digit postion

void Display_Digit(uint8_t pos, uint8_t val) {

    uint32_t current_display = 0;


    SEG_CTL = 1;

    current_display = SEG_DATA;// read current display value

    // Clear the position we want to write to

    if (pos == 1) {

        current_display &= 0xfffffff0;

    }

    else if (pos == 2) {

        current_display &= 0xffff0ff;

    }
```

```
    else if (pos == 3) {
        current_display &= 0xfff0ffff;
    }
    else if (pos == 4) {
        current_display &= 0xf0ffffff;
    }
    // Put the new digit in the cleared pos
    current_display |= ((val & 0xF) << ((pos-1) * 8));
    current_display |= 0x80808080;

    SEG_DATA = current_display;// write back to the display
}

//this function takes a whole number and displays all the digits automatically
void Disp_BCD(uint16_t value) {
    uint32_t ones, tens, hundreds, thousands;

    ones = value % 10;// Extract the ones place digit (rightmost)
    tens = (value / 10) % 10; //Extract the tens place digit (divide by 10, then get
remainder)
    hundreds = (value / 100) % 10; //// Extract the hundreds place digit
    thousands = (value / 1000) % 10; ///Extract the thousands place digit
    //Display each digit at its position (position 1 = rightmost)
    Display_Digit(1, ones); // display in pos 1
    Display_Digit(2, tens);// display in pos 2
    Display_Digit(3, hundreds); // display in pos 3
```

```
    Display_Digit(4, thousands);// display in pos 4
}

//This function was made by kataylna
//// Main program - implements stopwatch logic
int main(void)
{
    uint32_t button_val;
    uint32_t is_running = 0;// 0 is stopped , 1 is running
    uint32_t count = 0; // the stopwatches value

    while(1){
        button_val = Button_Data & 0x0F;
        if(button_val & 0x01 ) // if button zero is pressed (start)
        {
           is_running = 1 ;


        }

        if(button_val & 0x02) // if button 1 is pressed (pause)
        {
            is_running = 0;
        }

        if((button_val & 0x04) && (is_running == 0))// if button 2 is pressed while paused
(reset)
```

```
        {
            count = 0; // reset the inner counter ( the actual value used)
        Display_Digit(1, 0);  // Clear position 1
        Display_Digit(2, 0);  // Clear position 2
        Display_Digit(3, 0);  // Clear position 3
        Display_Digit(4, 0);  // Clear position 4
        }


        if(is_running == 1 ) { // if the timer is going
            count = count +1 ; // undate value of the count by 1
            usleep(1000000) ; // puase for 1 sec
        }
        Disp_BCD(count);
        usleep(5000);
    }

}
```