Laboratory 4 – 7-Segment Applications

I. 7-Segment Display

Objectives

- To build an application to control the hardware 7-segment display on the Basys3 board
- To learn how to organize source and header files of a project

Activity Summary

1. Launch Xilinx SDK after opening and exporting the hardware platform
2. Create the Seg7Display application
3. Program the FPGA (if necessary), setup Run Configuration, and run the application
4. Repeat activity 3 for changes made to the application source file

ACTIVITIES

In this section you are required to complete an application that drives a 7-segment display in software. The circuit description of the 7-segment display on the Basys3 board and the expected illumination pattern are shown in Figure 1.

- There are four digits and each digit has 7 segments (A to G) and a decimal point (DP).
- All the segments have their anodes tied together. As a result a '1' on any segment's cathode turns that segment OFF while a '0' turns it ON.
- The 7 segments pins (CA to CG) and the DP pin are shared by all the digits.
- To select which of the digits uses the segments and DP pins, a multiplexing mechanism is used. Transistors AN0 to AN3 are used to multiplex the digits' access to the segment and DP pins.
- A '0' on AN0 for example will select the last digit (the first digit from the left).
- Note that only one digit has to be selected at a time.

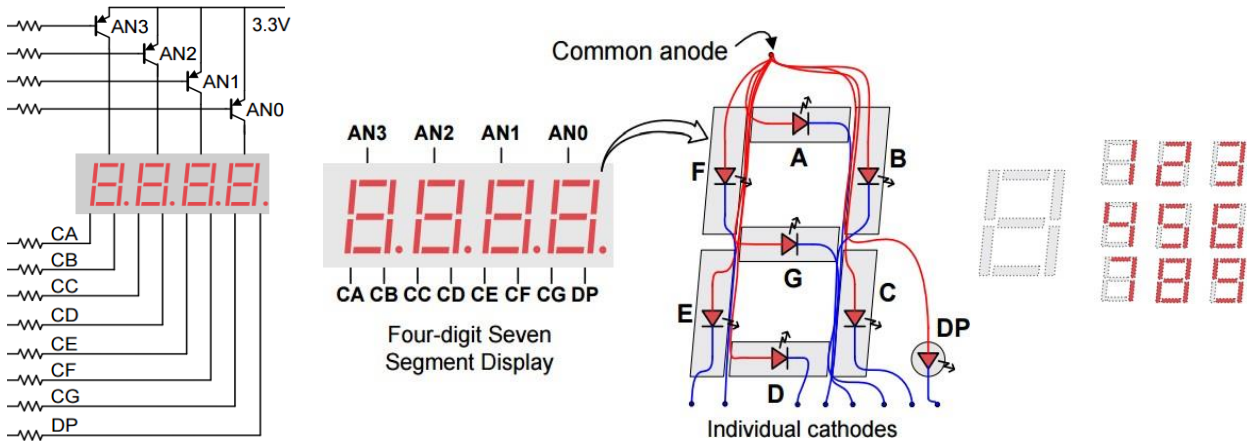

Figure 1: The 7-segment common-anode circuit and illumination pattern

7-Segment Display Pinout Description

The pin description for connecting to the 7-segment hardware is given in Tables 1 and 2. To write a binary-coded decimal (BCD) to any digit, the **`XGpio`** variable **`SEG7_HEX_OUT`** is used and the corresponding digit is selected by writing to the **`XGpio`** variable **`SEG7_SEL_OUT`**. These variables have been declared in the **gpio_init.h** file and initialized in the **gpio_init.c** file. Look inside the folder **Lab4 - Codes folder → 7-Segment Display** for these files.

Table 1: Segment selection pin out

Segment Selection	DP	CG	CF	CE	CD	CC	CB	CA
SEG7_HEX_OUT	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0

Table 2: Digit selection pin out

Digit Selection	AN3	AN2	AN1	AN0
SEG7_SEL_OUT	Bit3	Bit2	Bit1	Bit0

Refresh Frequency

Since all the anodes of the display are tied together, a multiplexer is used to select which digit is activated at a time. This requires that for all the digits to be lit, a time slice has to be given to each one at a frequency fast enough to avoid flickering. **A refresh period of between 1 ms and 16 ms is recommended**. Since all the four digits have to be lit every refresh period, each digit has only a quarter of the refresh period to be switched on. **We will use a refresh period of 16 ms (62.5 Hz refresh frequency) meaning that each digit has to be driven for 4 ms**.

Hardware Timer ISR

For this lab, the hardware timer has been redesigned for you to generate an interrupt every 4 ms (0.004 seconds). The interrupt service routine will be used to drive each digit. See pages 15 to 17 of the "Basys3™ FPGA Board Reference Manual" for more information on how to drive the 7-segment display.

Binary-Coded Decimal for Driving the Display

How do we write numbers to the 7-segment display? Let us assume for instance that we want to write the number '1' to the last digit of the display:

- The number in its original binary form is 0001.
- We have to write this to the 7 segments and DP of the digit.
- To do so, we need to encode the decimal number into an 8-bit binary and make a selection of AN0 as shown in Figure 2.
- Remember that a '1' turns OFF a segment while a '0' turns it ON.
- Therefore, this encoding gives us the 8-bit binary number 11111001 for the segments and 4-bit number 1110 for the digit selection. Note that the DP is turned off.
- As a result we would write 11111001 to **`SEG7_HEX_OUT`** and 1110 to **`SEG7_SEL_OUT`**.
- Table 3 shows the BCD codes for some of the other values that can be written to a digit. Note that in all these codes the decimal point is not used.

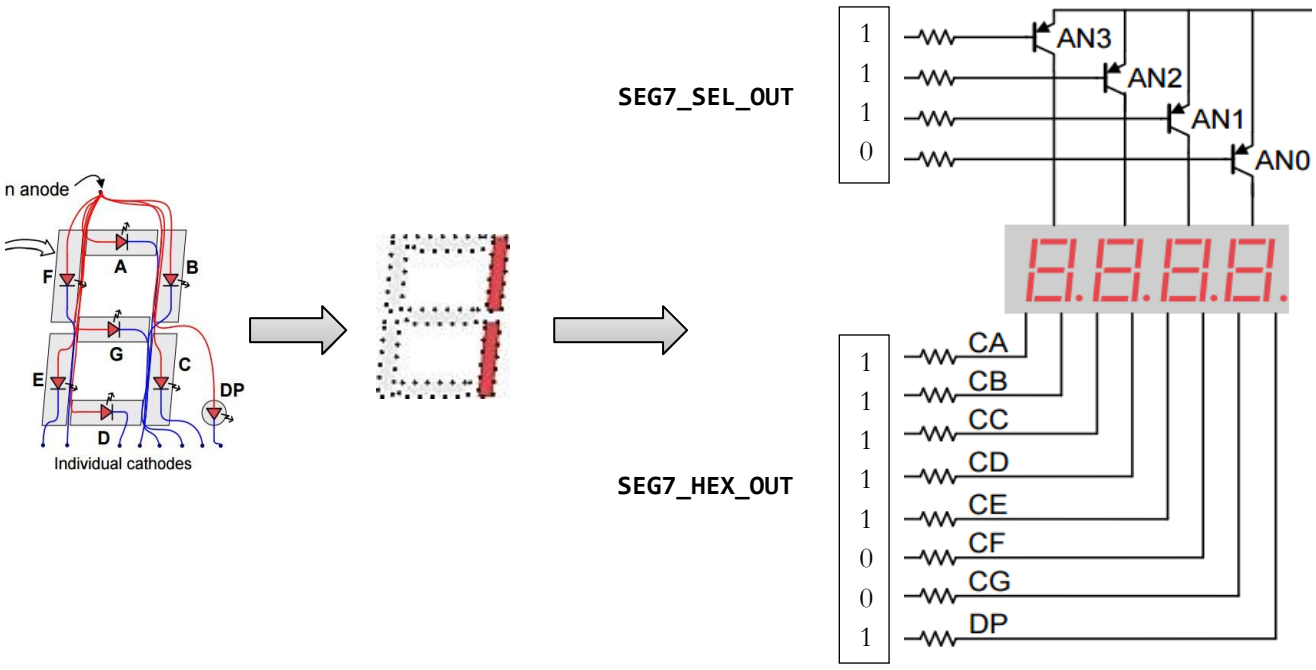

Figure 2: Segment and digit selection for writing a '1' to the 7-segment display

Table 3: Digit display BCD codes

Digit Value	SEG7_HEX_OUT (Hex)
Blank	1111 1111 (0xFF)
0	1100 0000 (0xC0)
1	1111 1001 (0xF9)
2	1010 0100 (0xA4)
3	1011 0000 (0xB0)
4	1001 1001 (0x99)
5	1001 0010 (0x92)
6	1000 0010 (0x82)
7	1111 1000 (0xF8)
8	1000 0000 (0x80)
9	1001 0000 (0x90)
Dash	1011 1111 (0xBF)

Project Files Organization

C applications are often organized in different files. All the macros, global variables and header file inclusions are usually placed in one or more header files (**.h files**), and functions are grouped in one or more source files (**.c files**). The main routine itself is placed in a file named **main.c**. The header files are then included in all the necessary source files. This structure has been used in the 7-segment applications and the subsequent ones.

Application Structure

1. There are four major methods involved in this application - `displayNumber(u16 number)`, `calculateDigits(u16 number)`, `hwTimerISR()`, and `displayDigit()`.
2. The `displayNumber(u16 number)` method is the topmost method which you should call to display any integer number. It is used to assign the digit number and the value to be displayed per digit when the interrupt occurs.
3. To `displayNumber(u16 number)` function calls the `calculateDigits(u16 number)` function which extracts the digits from the number to be displayed. Remember that the number Note that the maximum number of digits that can be displayed is four. That is, **you can only display numbers between 0 and 9999**, as there are only four digits in the display.
4. The `hwTimerISR()` method is used to call the `displayDigit()` method, which selects the segments and displays the digits on the 7-segment display.
5. All these functions are declared in the **seg7_display.h** header file and defined in the **seg7_display.c** source file.
6. Much of the codes have already been written for you and you will be guided to finish them up in the next section. The code has been fairly commented to aid your understanding of the flow of program execution.

Complete the Code and Run the Application

Now, take the following steps to complete the code:

1. Add all the files (see Figure 3) in the folder **Lab4 - Codes folder → 7-Segment Display** to the **src folder** of your project. All the codes have already been structured for you into appropriate header (**.h**) and source (**.c**) files, and **note that you are expected to follow a similar structure in your subsequent applications.**

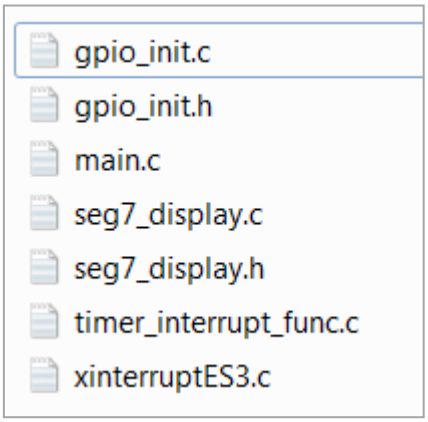

Figure 3

2. Open the **seg7_display.c** file to locate the functions `displayNumber(u16 number)`, `calculateDigits(u16 number)`, and `displayDigit()`.
3. In the `displayNumber(u16 number)` function type the codes in Figure 4 to display dashes ("----") to indicate that the number is **out of range** (**greater than 9999**). Think about how these codes work.

```
// Display "----" to indicate that the number is out of range
count = 1;
while (count < 5)
{
      digitToDisplay = NUMBER_DASH;
      digitNumber = count;
      count++;
      /* Wait for timer interrupt to occur and ISR to finish
       * executing digit display instructions
       */
      while (digitDisplayed == FALSE);
      digitDisplayed = FALSE;
}
```

Figure 4

4. In the `calculateDigits(u16 number)` function, we need to add codes to extract digits from two-digit and one-digit numbers. The codes in Figure 5 can be used for the two-digit extraction. **You should write a similar code for the one-digit extraction.**

```
fourthDigit  = number % 10;
thirdDigit = (number / 10) % 10;
secondDigit  = 0;
firstDigit = 0;
```

Figure 5

5. Now got to the `displayDigit()` function. You will notice that the codes that write blank (DIGIT_BLANK), 0 (DIGIT_ZERO), and 1 (DIGIT_ONE) digits to the segments of the display have been written.

   You may be wondering what DIGIT_BLANK, DIGIT_ZERO and DIGIT_ONE are. They are called **macros** and if you look inside the **seg7_display.h** file you will find the definitions of these macros (see Figure 6).

```
// Definitions for 7-segment BCD codes
#define DIGIT_BLANK   0xFF
#define DIGIT_ZERO    0xC0
#define DIGIT_ONE     0xF9
#define DIGIT_TWO     0xA4
#define DIGIT_THREE   0xB0
#define DIGIT_FOUR    0x99
#define DIGIT_FIVE    0x92
#define DIGIT_SIX     0x82
#define DIGIT_SEVEN   0xF8
#define DIGIT_EIGHT   0x80
#define DIGIT_NINE    0x90
#define DIGIT_DASH    0xBF
```

Figure 6

   The Xilinx SDK tool substitutes the hexadecimal numbers assigned to the macros wherever the macros are used. The hex numbers here are exactly the same BCD codes in Table 3.

   Now, complete the case statement for other digit values. All you need to do is write DIGIT_TWO through DIGIT_NINE for cases 2 to 9.

You should note that because DIGIT_BLANK and DIGIT_DASH are not digit numbers like 0 to 9, to represent their ***cases*** in the ***case statement***, we have used the numbers 10 and 11 respectively. We can do this because the numbers 10 and 11 are not single digits.

The case for DIGIT_BLANK has been written for you. You should add the one for DIGIT_DASH.

6. Now that you have completed the code,

   - **Program the FPGA**,
   - **Setup Run Configurations**, and
   - **Run the application**.

   The number 1234 should be seen on the display. This is because the `displayNumber(1234)` is called in the **main.c** file. Open this file to have a look.

   As a simple exercise, you can change the number to be displayed and rerun the application.

7. Now, modify the code so that instead of writing the value to be displayed in the code, you obtain it from the slide switches. You can follow the following steps:

   - Declare an `XGpio` object called **`SLIDE_SWITCHES`** in the **gpio_init.h** file and initialize it in the **gpio_init.c** file.
   - Declare in the **main.c** file, a global 16-bit unsigned integer (u16) called **`slideSwitchIn`** and give it an initial value of zero.
   - In the `while(1)` loop of the `main()` function, read into the variable **`slideSwitchIn`** from the slide switches using the `XGpio_DiscreteRead(XGpio *InstancePtr, unsigned Channel)` function.
   - Then call `displayNumber(slideSwitchIn)` to display the value read from the switch
   - Run the application and change the state of the slide switches. The value read from the slide switches should show on the 7-segment display.

[Hint: To locate the definition of a variable or method, press Ctrl and click on the variable or method]

II. Button Counter

Objectives

- To learn how to detect whether a button is pressed
- To build an application to display how many times a button is pressed

Activity Summary

1. Create an application named ButtonCounter
2. Write the code for the application
3. Program the FPGA (if necessary), setup Run Configuration, and run the application

Activities

The activities here involve counting the number of times a button on the Basys3 board has been pressed and displaying this on the 7-segment display. Create the ButtonCounter application. The file structure here is similar to that of the 7-segment display application and the files needed are already included in the "Button Counter folder" inside "Lab4 - Codes folder". The following steps have already been implemented for you. Just follow them to understand what has been done.

1. Declare an `XGpio` object for the left push button on the Basys3 board. Name it `P_BTN_LEFT` in the gpio_init.h file and initialize it in the gpio_init.c file. Remember to check the device ID of the left button, which is named `GPIO_P_BTN_L` in the xparameters.h file

2. In the main.c file, create two unsigned integer variables named `pushBtnLeftIn` and `counter`. In a forever loop in the `main()`, output the value of `counter` to be displayed on the 7-segment display

3. Read the input button (BTNL on the board) and check if it has been pressed by comparing it to 1. If BTNL is pressed wait until it is released and then increment `counter` by 1. If not, continue the loop.

   Note that because our 7-segment display is driven in software rather than hardware, while waiting for the button to be depressed, you may have to keep calling the method to display the counter value in order to avoid flickers on the display

4. Run the application and press the left button. The display should show how many presses you have made

III. Advanced Counter

Objectives

- To build a counter application for buttons and switches
- To learn how to detect the pressing of different buttons and switches

Activity Summary

1. Create a new application called AdvancedCounter
2. Write the application code
3. Program the FPGA (if necessary), setup Run Configuration, and run the application

Activities

Only two buttons (BTNL and BTNR) are to be used for button inputs. The middle button (BTNC) is reserved for reset signal. Pressing it will reset the MicroBlaze processor. In this application, button presses of BTNL need to be counted and displayed on the 7-segment display. Also, if the right button (BTNR) is pressed, the value read from the slide switches should be shown on the 7-segment display. The following steps are suggested:

1. Unfinished code template can be found in gpio_init.c and main.c
2. The `XGpio` objects `P_BTN_LEFT`, `P_BTN_RIGHT`, `LED_OUT`, and `SLIDE_SWITCHES` have all been declared in the gpio_init.h file. All you have to do is initialize them in the gpio_init.c file
3. In the main() function,
   - Output the counter values for initial display
   - Detect which button is pressed. If BTNL is pressed, count the number of times the button is pressed as was done previously. If the second button (BTNR) is pressed, extract the four LSB of switch input and add the extracted value to switch counter
4. Once you have completed the code test it on the board