Laboratory No. 05:
BF533 Tools Laboratory

By: Derek Hildreth

Instructor: Brother Karl

BYU-Idaho CompE 460

April 12, 2018

I. INTRODUCTION

A. Purpose

The goal of todays lab is to re-design the LED/Switch system (delivered to students in the LEDTest1 programs) to include a hardware timer. This lab will entail updating the software to be able to configure the timer unit and respond to timer interrupts. Instead of using a delay loop for the timing of the LEDs, students need to configure a timer unit on the BF533 to provide accurate timing. When the timer times out, the timer should generate an interrupt and the software should respond to that interrupt by blinking the LEDs.

In this lab, students will use a round-robin with interrupt software architecture using the `LEDTest1.c` example program as the basis for the design. The software needs to do 4 main functions. First the SW needs to be able to initialize the peripheral units (timer and programmable flags). Second the SW needs to be able to respond to button. Third the SW needs to be able to respond to timer interrupts. And fourth, the SW needs to be able to control the LEDs per the buttons and the timer interrupt. From an architectural standpoint, the SW can be broken into 4 main sections:

- LED Control – functions to control the LED's

- Interrupt – ISR

- Timer Control – functions to control the timer units

- Main Loop

Students will need one timer interrupt. In this interrupt, they should set a flag indicating that the ISR was asserted. The main control SW needs to be able to do 2 things. First it needs to initialize the timer unit, the programmable flags and the interrupt. Next it needs to read the button press, the flags updated by the ISRs, and output the correct LEDs.

B. Equipment

There is a minimal amount of equipment to be used in this lab. The few requirements are listed below:

- Xilinx ISE Navigator Software (v10.0.1)

- Computer capable of running the software mentioned

- Spartan 3E FPGA Developer Board (For Hardware Simulation)

C. Procedure

1. Start the ISE Navigator. See the ISE 8.2i Quick Start Tutorial.

2. Create a new project.

3. Import copies of the Verilog modules AND_OR, MY_AND2, and MY_OR2 to the new project just created.

4. Create a Test Bench (called Test Fixture in Verilog).

5. Create the actual input stimulus.

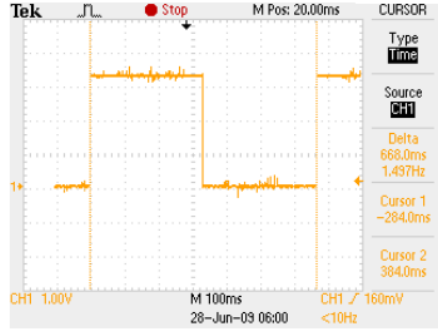6. Run the simulation, examine the waveforms, and verify functioniality.

II. SCHEMATIC DIAGRAMS

This section consists of screenshots taken during the laboratory procedure. One set of the screenshots is of the periods captured by the oscilliscope and the other set is captured by the logic analyzer.
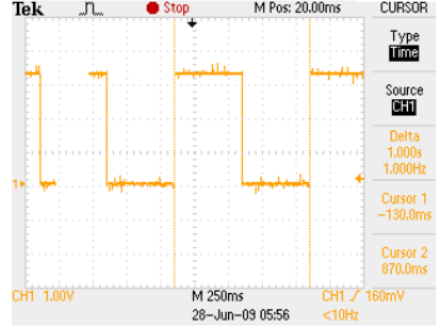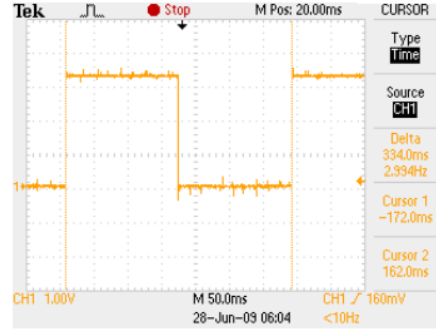

1S PERIOD FOR LED4

(a) LED4 Period

~666.7mS PERIOD FOR LED5

(b) LED5 Period

~333.3mS PERIOD FOR LED6

(c) LED6 Period

FIG. 1: Period of LED blink rate captured by osciliscope.

III. EXPERIMENT DATA

This section will consist of the important code blocks which were changed in order to meet the requirements of the lab.

```
/*********************************************************************
  Interrupt Service Routin for Timer 0.
 *********************************************************************/
// LED4
static ADI_INT_HANDLER(Timer0_ISR) {

    // See if this is a timer 0 event, by calling a function to
    // read corresponding bit (16) in the the SIC_ISR register

    if (adi_int_SICInterruptAsserted(ADI_INT_TIMER0) == ADI_INT_RESULT_NOT_ASSERTED)

    // This return value tells the interrupt manager to process the next
    // ISR in the chain for this IVG, because we haven't yet serviced the
    // peripheral that interrupted this time

        return (ADI_INT_RESULT_NOT_PROCESSED);

    // clear timer 0 interupt
    adi_tmr_GPControl(ADI_TMR_GP_TIMER_0, ADI_TMR_GP_CMD_CLEAR_INTERRUPT, NULL);

    // toggle the specified LED
    // ON|ON|ON
    if(button0 && button1 && button2){
        ezToggleLED(EZ_FIRST_LED);          // LED4,5,6 BLINK
    }

    //***CODE SNIPPED****//

    return (ADI_INT_RESULT_PROCESSED);

}

    //***CODE SNIPPED****//

/*********************************************************************
   Function: Init_Timers

   Set up timers for PWM mode and enale them.
 *********************************************************************/

void InitTimers(void)
{
    //Setting up command table for Timer 0
    ADI_TMR_GP_CMD_VALUE_PAIR Timer0ConfigurationTable [] = {
        { ADI_TMR_GP_CMD_SET_TIMER_MODE,         (void *)0x01             },
        { ADI_TMR_GP_CMD_SET_COUNT_METHOD,       (void *)TRUE             },
        { ADI_TMR_GP_CMD_SET_INTERRUPT_ENABLE,   (void *)TRUE             },
        { ADI_TMR_GP_CMD_SET_OUTPUT_PAD_DISABLE, (void *)TRUE             },
        //***CODE SNIPPED****//
        { ADI_TMR_GP_CMD_SET_WIDTH,              (void *)0x00400000       },
        { ADI_TMR_GP_CMD_END,                    NULL                     },
    };

        //***CODE SNIPPED****//

}

/*********************************************************************
 *  Function:   main
 *********************************************************************/
void main(void) {

    u32 ResponseCount;
    void *pExitCriticalArg;
    u32 i; //loop variable


    // initialize the EZ-Kit
    ezInit(1);

    // initialize the flag manager because the LEDs and buttons connect via flags
    // Since callbacks are not being used memory does not to be given to the service
    ezErrorCheck(adi_flag_Init(NULL, 0, &ResponseCount, NULL));

    //***CODE SNIPPED****//

    InitTimers();

    while (1) {
    //***CODE SNIPPED****//
        }

    } // END WHILE
} // END MAIN
```

A.   Formulas and Overall Descriptions Used

This part of the laboratory was done for Feedback Control. Most of this laboratory's calculations were completed and compiled by the folks at Quanser (the manufacturer of the inverted pendulum) and will give the lab a good starting place. Below are the state equation and gain values used initially in the lab:

$$
\begin{bmatrix} \dot{\alpha} \\ \ddot{\alpha} \\ \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 81.7 & 0 & 0 & -13.9 \\ 0 & 0 & 0 & 1 \\ 39.7 & 0 & 0 & -14.4 \end{bmatrix} \begin{bmatrix} \alpha \\ \dot{\alpha} \\ \theta \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ 24.5 \\ 0 \\ 25.4 \end{bmatrix} V
$$

$$
K = \begin{bmatrix} 21 & 2.8 & -2.2 & -2.0 \end{bmatrix}
$$

Other values, such as the $\frac{\text{Volts}}{\text{Degree}}$ and $\frac{\text{Degrees}}{\text{Volt}}$ were obtained by first determining the max angle of the pendulum on both extreme sides.

Using the max angles from above, these values were determined:

$$
\alpha = 0.062 \frac{\text{Volts}}{\text{Degree}}
$$

$$
\alpha = 15.105 \frac{\text{Degrees}}{\text{Volt}}
$$

I would also like to add that in order to calibrate α to get a perfect vertical $= 0$, a value of 0.09 needed to be added. The same applies to θ where 0.322 needs to be added.

B.   DC Motor Transfer Function and Parameters

Definitions:

$$
\theta(t) = Angular Position
$$
$$
\dot{\theta}(t) = Angular Velocity
$$
$$
\triangle t = t_{10\%} - t_{90\%}
$$
$$
90\% = e^{-t_{10\%}/\tau}
$$
$$
10\% = e^{-t_{90\%}/\tau}
$$

The Math:

$$
\frac{s\theta(s)}{V_a(s)} = \frac{K}{s+P}
$$
$$
\text{Let } V_a(s) = \frac{V_0}{s}
$$
$$
s\theta(s) = \frac{KV_0}{(S+P)S} = \frac{KV_0}{\frac{P}{S}} - \frac{\frac{KV_0}{P}}{s+P}
$$
$$
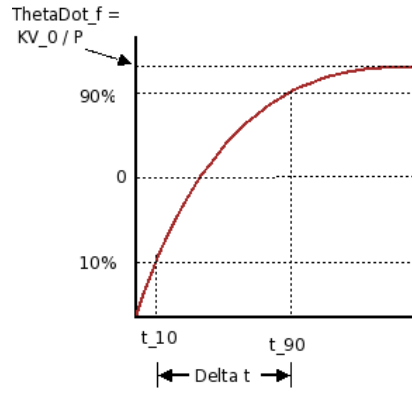L^{-1} \Rightarrow \dot{\theta}(t) = \frac{KV_0}{P}(1 - e^{-t/(1/P)})
$$
$$
\dot{\theta}(t) = (\dot{\theta}_i - \dot{\theta}_f)e^{-pt} + \dot{\theta}_f
$$

Final equations:

$$\dot{\theta}_f = \frac{KV_0}{P} \tag{1}$$

$$\frac{1}{P} = \tau = \frac{\triangle t}{ln(9)} \tag{2}$$

Graphically (Refer to Equation 1 and Equation 2) :

This section consists of tables and reductions which were used in this laboratory exercise.

PS	D	N	NS	P
$0.00	0	0	$0.00	0
	0	1	$0.05	0
	1	0	$0.10	0
	1	1	–	–
$0.05	0	0	$0.05	0
	0	1	$0.10	0
	1	0	$0.15	0
	1	1	–	–
$0.10	0	0	$0.10	0
	0	1	$0.15	0
	1	0	$0.15	0
	1	1	–	–
$0.15	–	–	$0.15	1

TABLE I: Symbolic Transition Table

(a) D1 = Q_1+D+Q_0N
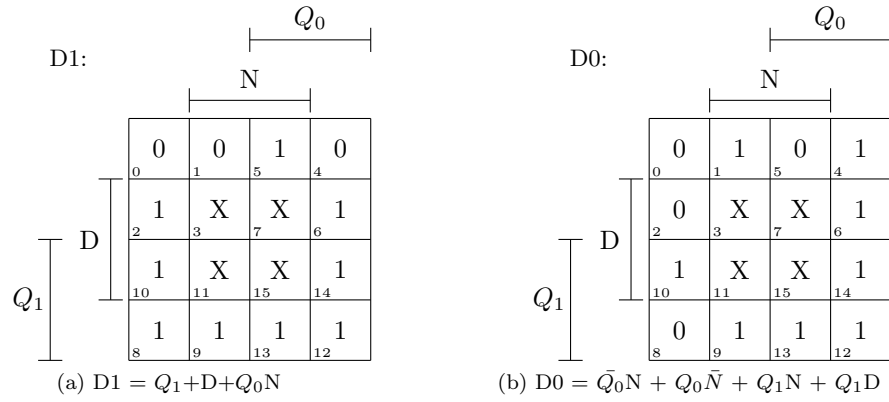
(b) D0 = $\bar{Q}_0$N + $Q_0\bar{N}$ + Q_1N + Q_1D

TABLE II: Karnaugh maps and the simplified results of the logic.

IV.   DISCUSSION & CONCLUSION

The goal of this lab was to re-design the LED/Switch system to include a hardware timer. By pressing eight different combinations of the three buttons, the LEDs on the board were to act in different ways using these timers. There was not a Q&A requirement for this lab.

I was able to accomplish the requirements of the lab by utilizing the `IntMgrTimerExample.c` project found within the analog devices example programs folder (and mentioned in the class lecture). There were some stumbling blocks to overcome. The most difficult for myself was actually getting the period of the LEDs just right. I was able to get it very close to the $333.3ms$, $666.7ms$, and $1s$ periods, but not exactly. My first method of getting these periods right was to take the clock speed in MHz, find the period by taking the inverse of the clock speed, and then solving for the value in hex that was needed to get the right period. This didn't yeild very accurate results at all, and so I then went through a trial and error session until I got a value of $1.1ms$. I used this value in hex to calculate the other periods. The results of this method can be seen in Figure 1 above in the schematics section.

Another observation I would like to point out is that I put all of my logic within the interrupts themselves. I feel that this was a hacked way of doing the lab to save time and that it's probably not the best programming method. After I was completed with my lab, I viewed other students solutions and they just seemed more elegant. Interestingly enough, the other students weren't incredibly happy with their solution either. If I were to go back and do this lab again, I would invest more time in both understanding how to utilze the interrupts as well as find a more elegant solution to blink the lights.

All in all, this laboratory gave me an insight on how interrupts work and I hope to be able to apply them to following labs. . .