EECS 388

LABORATORY EXERCISE III

BUTTON INPUT AND SERIAL COMMUNICATION

October 9, 2018

Benjamin Streit

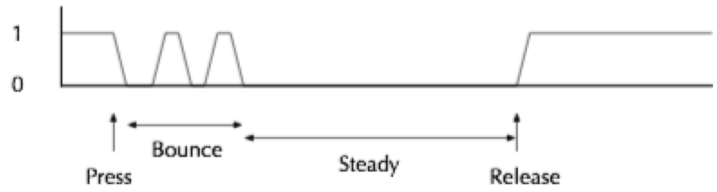
GTA: Ibikunle Oluwanisola

LABORATORY OVERVIEW

This laboratory involves the implementation of a new task to be run on the Tiva TM4C1294 evaluation board with the TI Audio BoosterPack add-on evaluation board. This new task will serve as a button monitor and will work in tandem with a task from laboratory exercise II, `Task_Speakerbuzz.c`, to emit a frequency for a given duration upon press and release of a switch on the evaluation board.

BACKGROUND

Before beginning an implementation for this exercise, the concept of switch bounce must be understood. When a switch is pressed, it does not generate a clean signal. Meaning, the signal could change between 1 and 0 several times before landing on a given value. This behavior is called switch bounce, and is a fundamental concept of switch mechanics. See below diagram for a visual representation of this concept.



In addition, the task created in laboratory exercise II, `Task_Speakerbuzz.c` has been modified slightly to work with the new button monitoring task. The general structure of this modified `Task_Speakerbuzz.c` is as follows:

```
// Include DAC drivers
// Define extern bool make_sound
// Define state variables
// Initialize the DAC
// Initialize state value
// Enter endless while loop
    // If state variable is high, toggle state variables
    // Else, toggle state variables
    // If make sound is true, write to DAC, delay for half period
    // Else, delay for 1 ms
    // Loop again
```

Finally, UART and PuTTY will be used to print a message upon release of a switch. PuTTY is prepared by setting the COM port to whichever COM port is being used on the workstation

by the evaluation board, and setting the serial connection speed to 115200. This will open a terminal accepting output from programs being run on the evaluation board. UART works similarly to a standard `printf` statement in C syntax, and example use is as follows:

```
// Import APIs
#include "Drivers/UARTStdio_Initialization.h"
#include "Drivers/uartstdio.h"
// Initialize UART
UARTStdio_Initialization();
// Print to PuTTY
UARTprintf( "FreeRTOS Starting!\n" );
```

PROCEDURE & RESULTS

Design of the task `Task_Monitor_Button.c` is in two parts: configuration of the GPIO pins on the evaluation board, and monitoring of the buttons on the evaluation board.
Configuration of the GPIO pins consist of the following steps:

(i) Enable PortJ with SysCtlPeripheralEnable.

(ii) Set PortJ<1..0> pins to input with GPIOPinTypeGPIOInput() subroutine from DriverLib.

(iii) Set PortJ<1..0> pins to weak pull-up with GPIOPadConfigSet.

The general structure of the monitoring of the buttons is as follows:

(i) Read the state of two buttons every 1 ms.

(ii) If a button is released, go to step (vi).

(iii) If no button is pressed, exit.

(iv) If a button is pressed, wait 10 ms.

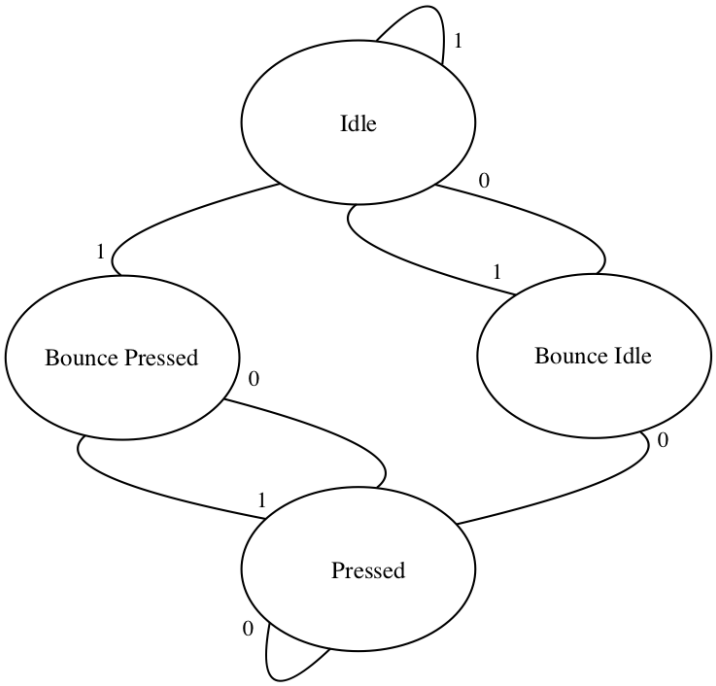
(v) If the same button is pressed after 10ms, declare a button press and emit a 0.2 S tone at 440 Hz.

(vi) On release of a button, emit a 0.5 s tone at 440 Hz and send a message using the UART.

This program makes use of a `bool make_sound` defined outside of the task, as well as an `enum` to model the various button states, this `enum` takes the following form:

```
enum ButtonState { Idle, BouncePress, BounceRelease, Pressed } button_state;
```

The transitions between states for this `enum` is graphically displayed in the state diagram below:



After configuring the GPIO pins to the above specification, the task enters an infinite `while` loop split up into two distinct segments: (`i`) Checking for button activity, setting `button_state` and `button_enabled` accordingly, and (`ii`) `switch` statement performing operations for given `button_state`, including transitioning between states and emitting a tone of the desired frequency.

The first segment of the `while` loop begins by reading in the button values using the following syntax:

```
button = GPIOPinRead( GPIO_PORTJ_BASE, GPIO_PIN_0 | GPIO_PIN_1 );
```

This value in in binary will either be 11, 10, or 01. These values in decimal are 3, 2, and 1. Given that 1 is idle, 3 corresponds to no switch activity, while 2 and 1 correspond to switch activity on switch n - 1. With this in mind, if the button value is less than 3, the program checks if the button state is currently `Idle`. If `Idle`, and if a `boolean` value `button_enabled` is `false`, it is known that the button may be transitioning to a pressed state, and thus the `button_state` becomes `BouncePress`, `button_enabled` becomes true, and the program delays for 10 ms. Else, it is known that the button is still pressed after delaying 10 ms during a previous iteration, and thus the `button_state` is set to `Pressed`. But, if `button_state` is not `Idle`, the program checks if it is `Pressed`. If `Pressed`, and `button_enabled` is `true`, it is

known that the button is being released, and thus `button_state` is set to `BounceRelease`, `button_enabled` is set to `false`, and the task delays for 10 ms. After performing above checks and operations, it is now time for the second half of the `while` loop.

As mentioned previously, the latter half of the `while` loop consists of a `switch` statement. This `switch` statement accepts the `button_state` as a parameter, and each case is each `ButtonState enum` value. The flow of the `switch` statement is as follows:

If `Idle`, `made_sound_pressed` is set to `false` so that upon next press a sound will emit via `Task_Speakerbuzz.c`, and then the program delays 1 ms.

If `BouncePress`, if `button_enabled` is `true`, `button_state` is set to `Pressed`, else it is set to `Idle`. This conditional is followed by a delay of 1 ms.

If `Pressed`, if `made_sound_pressed` is `false`, `make_sound` is set to `true`, followed by a delay of 200 ms, then `make_sound` is set to `false` and `made_sound_pressed` is set to `true`. Else, the program delays 10 ms. The toggling of `make_sound` in this case will trigger `Task_Speakerbuzz.c` to begin writing to the DAC, thus emitting a tone of the desired frequency. The delay of 200 ms before toggling `make_sound` again produces the requested duration of the tone.

If `BounceRelease`, print via UART, set `make_sound` to `true`, delay for 500 ms, set `make_sound` to `false` and `button_state` to `Idle`. Again, the toggling of `make_sound` allows the tone to begin emitting via `Task_Speakerbuzz.c`, and the delay produces the requested duration of the tone. Now the end of the `while` loop has been reached, and the next iteration begins.

ANALYSIS

The two tasks, `Task_Speakerbuzz.c` and `Task_Monitor_Button.c` are connected via the `extern bool make_sound`, in which a value of `true` triggers `Task_Speakerbuzz.c` to begin writing to the DAC and thus emitting a tone at the desired frequency, and a value of `false` triggers the tone to stop. This implementation allows for a level of abstraction that creates much more modular and readable code than many alternatives. Furthermore, by reading in the two switch values simultaneously and working with the decimal equivalent of the combined binary value, the code becomes even more extensible and appropriate as there not multiple read in statements, variables to hold these read values, and convoluted conditional

statements that correctly assign values to state variables.

CONCLUSIONS

The logic written for this laboratory exercise is sensible and effective, as all desired outcomes were achieved and in an appropriate manner. However, improvements could be made. Currently, the delays when emitting the tone for button press and button release are hard-coded, as well as the frequency at which these tones are emitted. This is workable, but not ideal as it inhibits code modularity, extensibility, and readability. Should this exercise be attempted a second time, these hinderances would be mediated by defining variables to house each value, rather than using the hard-coded values that currently exist in the program.

CODE

EECS_388_Program_Base_Fa18.c

```
/*
 * main.c
 */

#include    "inc/hw_ints.h"
#include    "inc/hw_memmap.h"
#include    "inc/hw_types.h"
#include    "inc/hw_uart.h"

#include    <stddef.h>
#include    <stdbool.h>
#include    <stdint.h>
#include    <stdarg.h>

#include    "driverlib/sysctl.h"
#include    "driverlib/pin_map.h"
#include    "driverlib/gpio.h"

#include    "Drivers/Processor_Initialization.h"
#include    "Drivers/UARTStdio_Initialization.h"
#include    "Drivers/uartstdio.h"

#include    "FreeRTOS.h"
#include    "task.h"
```

```
25
26 #include    <stdio.h>
27
28 extern void Task_Blink_LED_PortN_1( void *pvParameters );
29 extern void Task_Speakerbuzz( void *pvParameters );
30 extern void Task_Monitor_Button( void *pvParameters );
31 extern void Task_ReportTime( void *pvParameters );
32 extern void Task_ReportData( void *pvParameters );
33
34 int main( void ) {
35
36     Processor_Initialization();
37     UARTStdio_Initialization();
38
39     //
40     //  Create a task to blink LED, PortN_1
41     //
42     xTaskCreate( Task_Blink_LED_PortN_1, "Blinky", 32, NULL, 1, NULL );
43
44     //
45     // Create a task to change speaker frequency
46     //
47     xTaskCreate( Task_Speakerbuzz, "Speakerbuzz", 32, NULL, 1, NULL );
48
49     //
50     // Create a task to monitor buttons
51     //
52     xTaskCreate( Task_Monitor_Button, "MonitorButton", 32, NULL, 1, NULL );
53
54     //
55     //  Create a task to report data.
56     //
57     xTaskCreate( Task_ReportData, "ReportData", 512, NULL, 1, NULL );
58
59     //
60     //  Create a task to report SysTickCount
61     //
62     xTaskCreate( Task_ReportTime, "ReportTime", 512, NULL, 1, NULL );
63
64     UARTprintf( "FreeRTOS Starting!\n" );
65
66     //
```

```
67      //  Start FreeRTOS Task Scheduler
68      //
69      vTaskStartScheduler();
70
71      while ( 1 ) {
72
73      }
74
75  }
```

Task_Monitor_Button.c

```
1   /*--Task_Blinky.c
2    *
3    *  Author:      Gary J. Minden
4    *  Organization: KU/EECS/EECS 388
5    *  Date:      February 22, 2016
6    *
7    *  Description:  Blinks LED D1 on Tiva TMC41294 Evaluation board
8    *
9    */
10
11  #include  "inc/hw_ints.h"
12  #include  "inc/hw_memmap.h"
13  #include  "inc/hw_types.h"
14  #include  "inc/hw_uart.h"
15
16  #include  <stddef.h>
17  #include  <stdbool.h>
18  #include  <stdint.h>
19  #include  <stdarg.h>
20
21  #include  "driverlib/sysctl.h"
22  #include  "driverlib/pin_map.h"
23  #include  "driverlib/gpio.h"
24
25  #include    "Drivers/EECS388_DAC.h"
26
27  #include  "FreeRTOS.h"
28  #include  "task.h"
29
30  #include "Drivers/UARTStdio_Initialization.h"
31  #include "Drivers/uartstdio.h"
```

```
32
33 bool make_sound = false;
34
35 extern void Task_Monitor_Button( void *pvParameters ) {
36
37     //
38     // Variables
39     //
40     volatile uint32_t button = 0;
41     bool button_enabled = false;
42     uint32_t active_button = 0;
43     bool made_sound_pressed = false;
44     bool made_sound_released = false;
45
46     enum ButtonState { Idle, BouncePress, BounceRelease, Pressed } button_state;
47     button_state = Idle;
48
49     //
50     // Initialize UART
51     //
52     UARTStdio_Initialization();
53     UARTprintf( "FreeRTOS   Starting!\n"    );
54
55     //
56     // Initialize the EECS_388 DAC interface.
57     //
58     EECS388_DAC_Initialization();
59
60     //
61     // Enable the GPIO Port J.
62     //
63     SysCtlPeripheralEnable( SYSCTL_PERIPH_GPIOJ );
64
65     //
66     // Configure PortJ<1..0> for input
67     //
68     GPIOPinTypeGPIOInput( GPIO_PORTJ_BASE, GPIO_PIN_0 );
69     GPIOPinTypeGPIOInput( GPIO_PORTJ_BASE, GPIO_PIN_1 );
70
71     //
72     // Set weak pull up on PortJ<1..0>
73     //
```

```
74      GPIOPadConfigSet( GPIO_PORTJ_BASE,
75                                GPIO_PIN_0, GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU );
76      GPIOPadConfigSet( GPIO_PORTJ_BASE,
77                                    GPIO_PIN_1, GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU
    );
78
79      //
80      // Begin monitoring
81      //
82      while ( 1 ) {
83          button = GPIOPinRead( GPIO_PORTJ_BASE, GPIO_PIN_0 | GPIO_PIN_1 );
84
85          if ( button < 3 ) {
86              active_button = (button - 1);
87              if ( button_state == Idle ) {
88                  // going from idle to bounce press
89                  if ( button_enabled == false ) {
90                      button_state = BouncePress;
91                      button_enabled = true;
92                      vTaskDelay( pdMS_TO_TICKS(10) );
93                  } else {
94                      // still pressed after delay, go to pressed
95                      button_state = Pressed;
96                  }
97              }
98          } else {
99              if ( button_state == Pressed ) {
100                 if ( button_enabled == true ) {
101                     // button going from pressed to bounce release
102                     button_state = BounceRelease;
103                     button_enabled = false;
104                     vTaskDelay( pdMS_TO_TICKS(10) );
105                 }
106             }
107         }
108
109         switch ( button_state ) {
110             case Idle: {
111                 made_sound_pressed = false;
112                 vTaskDelay( pdMS_TO_TICKS(1) );
113                 break;
114             }
```

```
115             case BouncePress: {
116                 if ( button_enabled == true ) {
117                     button_state = Pressed;
118                 } else {
119                     button_state = Idle;
120                 }
121                 vTaskDelay( pdMS_TO_TICKS(1) );
122                 break;
123             }
124             case Pressed: {
125                 if ( made_sound_pressed == false ) {
126                     make_sound = true;
127                     vTaskDelay( pdMS_TO_TICKS(200) );
128                     make_sound = false;
129                     made_sound_pressed = true;
130                 } else {
131                     vTaskDelay( pdMS_TO_TICKS(10) );
132                 }
133                 break;
134             }
135             case BounceRelease: {
136                 UARTprintf("Button %d released.\n", active_button);
137                 make_sound = true;
138                 vTaskDelay( pdMS_TO_TICKS(500) );
139                 make_sound = false;
140                 made_sound_released = true;
141                 button_state = Idle;
142             }
143             default: {
144                 vTaskDelay( pdMS_TO_TICKS(1) );
145             }
146         }
147     };
148 }
```

Task_Speakerbuzz.c

```
1 /*--Task_Blinky.c
2  *
3  *  Author:      Gary J. Minden
4  *  Organization: KU/EECS/EECS 388
5  *  Date:      February 22, 2016
6  *
```

```
 *  Description:  Blinks LED D1 on Tiva TMC41294 Evaluation board
 *
 */

#include  "inc/hw_ints.h"
#include  "inc/hw_memmap.h"
#include  "inc/hw_types.h"
#include  "inc/hw_uart.h"

#include  <stddef.h>
#include  <stdbool.h>
#include  <stdint.h>
#include  <stdarg.h>

#include  "driverlib/sysctl.h"
#include  "driverlib/pin_map.h"
#include  "driverlib/gpio.h"

#include    "Drivers/EECS388_DAC.h"

#include  "FreeRTOS.h"
#include  "task.h"

#include "Drivers/UARTStdio_Initialization.h"
#include "Drivers/uartstdio.h"

extern bool make_sound;

extern void Task_Speakerbuzz( void *pvParameters ) {

  uint32_t  DAC_State;
  bool        high;

  //
  // Initialize UART
  //
  UARTStdio_Initialization();
  UARTprintf( "FreeRTOS   Starting!\n"    );

  //
  // Initialize the EECS_388 DAC interface.
  //
```

```
49    EECS388_DAC_Initialization();
50
51    //
52    // Set boolean value
53    //
54    high = false;
55
56    UARTprintf("called speakerbuzz\n");
57
58    while ( 1 ) {
59        if ( high ) {
60            //
61            // Set DAC value
62            //
63            DAC_State = 0x0000;
64            high = false;
65        } else {
66            //
67            // Set DAC value
68            //
69            DAC_State = 0x3FFF;
70            high = true;
71        }
72
73        if ( make_sound == true ) {
74            EECS388_WriteDAC( DAC_State );
75            vTaskDelay( pdMS_TO_TICKS( ( 1 / 440 ) / 2 ) );
76        } else {
77            vTaskDelay( pdMS_TO_TICKS(1) );
78        }
79    }
80 }
```