

Lab1 Report: Modifying Blinky Example with Joystick Input

Introduction:

The uVision IDE features a preloaded sample project titled Blinky. This program demonstrates some of the features of the NXP LPC 1768 board such as the ADC (Analog to Digital) function and LED function to change the rate of a blinking LED output based on the position of an analog rheostatic dial. Lab1 asks students to modify the Blinky sample program to feature Joystick functionality of the NXP 1768 and to simulate the inputs of the Joystick direction through a print statement in the debug viewer window. Joystick implementation is accomplished by adding the Joystick API interface, “Joystick_MCB1700.c(Joystick)” board support to the run-time environment. The following C code functions within the “Joystick_MCB1700.c” file configures the NXP 17xx series board as follows. Joystick_Initialize() referencing the Appendix in the lab manual configures the pins by setting the Port 1 as the Joystick interface with Pins, 20,23,24,25 and 26 as the five possible directions of the physical joystick using the “PIN_Configure” command. This provides hardware definitions that in conjunction with the addition of the “Board_Joystick.h” header provides premade Joystick C functions. These premade C functions can be executed in the main program based on the movements of the joystick located on the board to accomplish some task, such as printing inputted direction being explored in this lab experiment. At the hardware level, the joystick grounds the pins 20,23-26 on port1/GPIO1 based on the direction of movement. Joystick movement can be simulated by accessing the GPIO1 pins manually on PORT1 and de-selecting the appropriate pins depending on the desired direction of joystick movement. On the software level, the joystick input is implemented by initializing the joystick using the Joystick_Initialize() function. The state of the joystick is determined by the Joystick_GetState() function. The states of the joystick are defined by the following variable names JOYSTICK_UP, JOYSTICK_DOWN, JOYSTICK_LEFT, JOYSTICK_RIGHT and JOYSTICK_CENTER. Both of these premade functions and variable definitions are provided by the “Board_Joystick.h” header and Joystick_MCB1700.c(Joystick) interface. Depending on the state of the joystick, a switch statement implemented in a loop can print the direction of the joystick input direction. This print statement can be observed in the debugger window to confirm functionality of the modified program.

Theory:

An examination of the “Joystick_MCB1700.c” to understand how the NXP 1700 series board implements the joystick. NOTE: The physical joystick is hardwired into Port 1 to pins 20,23,24,25 and 26 as outlined in the documentation for the development board being used for this course.

- 1) Multi-purpose pins on Port 1 must be configured for input:

The FIODIR register controls GPIO as input or output. As described in the lab manual FIODIR must configure pins 20,23,24,25 and 26 as input pins from the Joystick by setting each of these register values to zero using the code described in the lab manual below. (A zero-bit value configures the pin as an input and a one-bit value configures the pin as an output)

```
LPC_GPIO1->FIODIR &=~ ((1<<20) | (1<<23) | (1<<24) | (1<<25) | (1<<26))
```

Note: “&=” symbol sets a zero-bit value. “<<” symbol indicates left bit-shift operation. The code above assigns a zero-bit value on register FIODIR/GPIO1 to the location specified by the value of the shift. The following code is implemented in the underlying function GPIO_SetDIR() under the Joystick_Initialization() function found in the “Joystick_MCB1700.c” file.

- 2) Multi-purpose pins must be configured for GPIO operation:

There are eleven PINSEL registers and each of these registers controls a set of pins on a certain port. The pins associated with Port 1 can be configured for multiple different functions on the board, however in this case, the Port 1 needs to be configured to GPIO operation to accept input from the joystick. As described in NXP 1700 series manual, to set operation of the required pins 20,23,24,25, and 26 associated with the joystick to the GPIO operation mode, the value of zero needs to be set to the following bits of PINSEL3 registers in accordance with Table 82.

GPIO on pin20→ bits 9,8 on PINSEL3 need to be set to zero
GPIO on pin23→ bits 15,14 on PINSEL3 need to be set to zero
GPIO on pin24→ bits 17,16 on PINSEL3 need to be set to zero
GPIO on pin25→ bits 19,18 on PINSEL3 need to be set to zero
GPIO on pin26→ bits 21,20 on PINSEL3 need to be set to zero

This can be accomplished by following code described in the lab manual.

```
LPC_PINCON->PINSEL3 &= ~ ((3<< 8) | (3<<14) | (3<<16) | (3<<18) | (3<<20));  
#define MASK 0X79  
Uint32_t joystick_value;  
joystick_value = (LPC_GPIO->FIOPIN >> 20) & MASK
```

The following code is implemented in the underlying function `PIN_configure()` under the `Joystick_Initialization()` function found in the “`Joystick_MCB1700.c`” file.

- 3) The state of the joystick is determined by reading the register associated with the pins of interest:

The `FIOMASK` register provides a masking mechanism when used in conjunction with the `FIOPIN` register (which reads the current state of the GPIO pins) can be used to identify the joystick position. The following code below describes how the two registers would be implemented to indicate joystick direction.

```
#define MASK    0X79 // mask defines joystick pin location in binary

Uint32_t joystick_position;
joystick_position = (LPC_GPIO->FIOPIN >> 20) & MASK
```


Note: An active joystick button direction is the zero-bit value. Taking the AND operation between the mask and the input pin register `FIOPIN` reveals the active joystick direction.

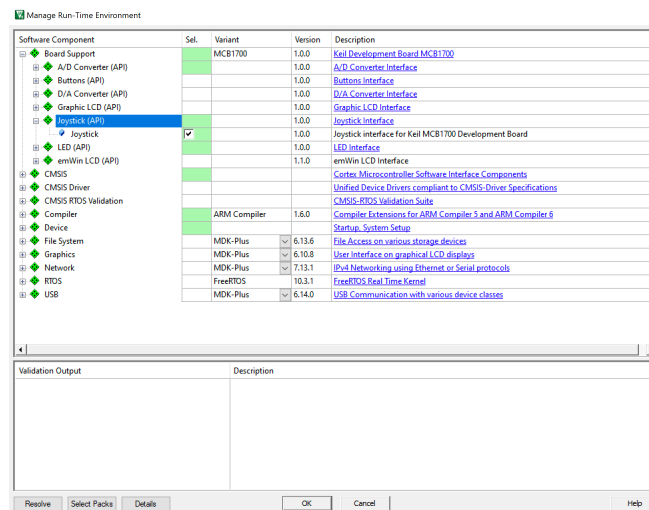
The following code or something similar is implemented in the underlying function `GPIO_PinRead()` under the `Joystick_GetState()` function found in the “`Joystick_MCB1700.c`” file.

To implement the joystick, you must add the required library “`Joystick_MCB1700.c`” to the run time environment. By running the `Joystick_Initialize()` and `Joystick_GetState()` command in the main of “`Blinky.c`”, the direction of the joystick can be determined and printed to the debug viewer window with the `printf()` command. (The required modification to “`Blinky.c`” is described in the procedure below.)

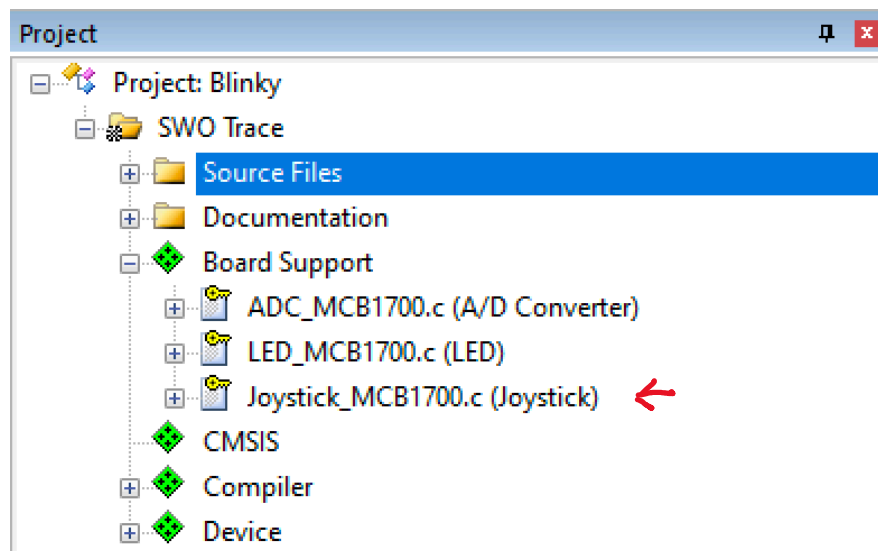
Procedure:

1) Load Blinky example project and complete Tutorial 1

2) Add Joystick Board Support by selecting the following icon in the uVision IDE  located in the top center of the toolbar. The following window will open. Select the Joystick API as follow below. Select OK button. (This adds software to hardware interface definitions required for Joystick functionality)



This step adds Joystick_MCB1700.c(Joystick) to the Board Support tab under the Project tree. (Located under: Project:Blinky--> SWO Trace-->Board Support) ---see Tutorial appendix for code.



4) Modify the C coded in the Blinky.c file located under the Project window on the left. Select SWO Trace--->Source Files-> Blinky.c

Add the following code according to the code below. **RED** is modified code which was added to the premade “Blinky.c” source file. See comments for design logic.

```
*-----
* Name:   Blinky.c
* Purpose: LED Flasher for MCB1700
*-----*/

#include <stdio.h>
#include "LPC17xx.h"          // Device header
#include "Board_LED.h"        // ::Board Support:LED
#include "Board_ADC.h"        // ::Board Support:A/D Converter
//This provides premade C functions to implement joystick interface ---see Tutorial 1 appendix for code
#include "Board_Joystick.h".   //:: Board Support: Joystick

char text[10];

/* Import external variables from IRQ.c file */
extern volatile unsigned char clock_1s;

// variable to trace in LogicAnalyzer (should not read to often)
volatile unsigned short AD_dbg;

uint16_t AD_last;           // Last converted value

/*-----
Main function
*-----*/
int main (void) {
    int32_t res;
    int32_t joy;             // Assign variable to hold state of joystick 32bit integer

    uint32_t AD_avg = 0;
    uint16_t AD_value = 0;
    uint16_t AD_print = 0;

    LED_Initialize();        // LED Initialization
    ADC_Initialize();        // ADC Initialization
    // Joystick Initialization ---premade function of “Board_Joystick.h” & “Joystick_MCB1700.c(Joystick)” interface
    Joystick_Initialize();

    SystemCoreClockUpdate();
    SysTick_Config(SystemCoreClock/100); // Generate interrupt each 10 ms

    while (1) {              // Loop forever

        // AD converter input
        res = ADC_GetValue();
```

```

/* Joystick input--->Get Joystick positional state and assign it a variable "joy" --- premade function of
"Board_Joystick.h" & "Joystick_MCB1700.c(Joystick)" interface */
joy = Joystick_GetState();

```

```

if (res != -1) {                                     // If conversion has finished
    AD_last = res;

    AD_avg += AD_last << 8;                          // Add AD value to averaging
    AD_avg++;
    if ((AD_avg & 0xFF) == 0x10) {                    // average over 16 values
        AD_value = (AD_avg >> 8) >> 4;              // average divided by 16
        AD_avg = 0;
    }
}

if (AD_value != AD_print) {
    AD_print = AD_value;                             // Get unscaled value for printout
    AD_dbg = AD_value;

    sprintf(text, "0x%04X", AD_value); // format text for print out
}

// Print message with AD value every second
if (clock_1s) {
    clock_1s = 0;

    printf("AD value: %s\r\n", text);
}

```

```


/* JOYSTICK_CENTER, JOYSTICK_DOWN, JOYSTICK_LEFT, JOYSTICK_RIGHT and JOYSTICK_UP
are defined in the "Board_Joystick.h" header. */



```

```

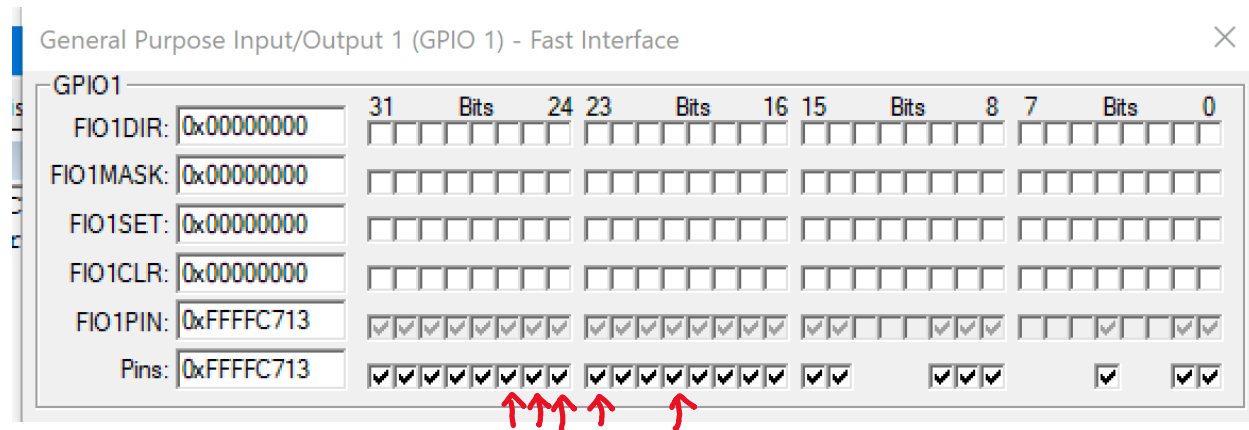
// Depending on output from Joystick prints the following direction
switch(joy){
    case JOYSTICK_CENTER:
        printf("Joystick center\n");
        break;
    case JOYSTICK_DOWN:
        printf("Joystick down\n");
        break;
    case JOYSTICK_LEFT:
        printf("Joystick left\n");
        break;
    case JOYSTICK_RIGHT:
        printf("Joystick right\n");
        break;
    case JOYSTICK_UP:
        printf("Joystick up\n");
        break;
}
}
}

```

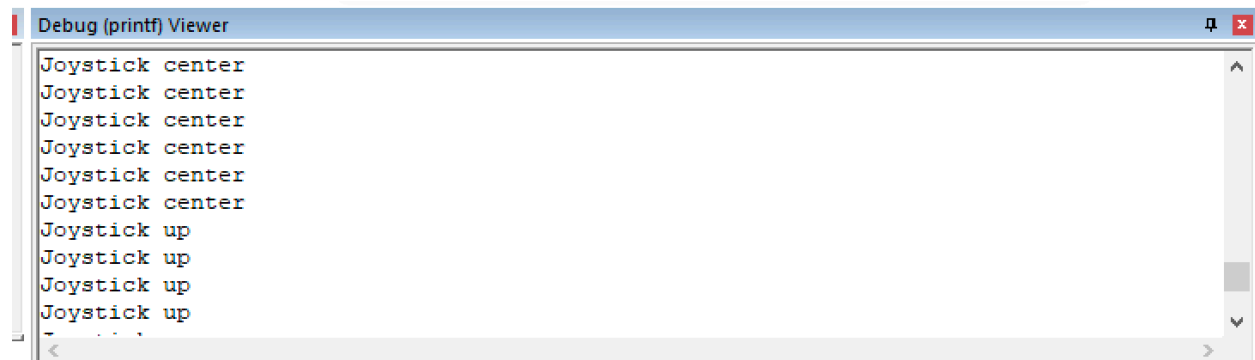
5) Compile project using the build  button.

- 6) Simulate software by selecting debug  button.
- 7) Select Peripherals-->GPIO Fast Interface--> Port 1 from toolbar
- 8) Select run  button from toolbar.
- 9) Under GPIO 1 window de-selecting the following check boxes under the “Pins” label to implement each direction of joystick movement.

Pin 20 → Center joystick
 Pin 26 → Left joystick
 Pin 24 → Right joystick
 Pin 23 → Left joystick
 Pin 25 → Down joystick



- 9) Observe Debug window for confirmation of each Joystick input.
 (Figure below shows output from Center then Up position being selected using joystick)



Summary:

During the execution of the Lab several key mistakes were made. The largest mistake was trying to implement joystick functionality without modifying the “Blinky.c” file directly but trying to copy only the bare functions required into a new Project file. This initial attempt would compile but became problematic as there were unknown files/definitions missing in the New Project file to execute simulation in the IDE. It became apparent that directly modifying “Blinky.c” for joystick implementation would be more time efficient. Similarly, there was a large mistake made when trying to implement the provided low-level C code provided in the Lab Manual. Initially the C code provided in the Lab Manual and reviewed in the Theory portion of this lab report was implemented directly into the “Blinky.c” file. However, it was discovered that the functions Joystick_GetState() and Joystick_Initialize() in the “Joystick_MCB1700.c” file implement these low-level register commands. It was determined that joystick hardware to software functionality was already implemented in the “Joystick_MCB1700.c” file. Consequently, all that was required to implement the print function specified in the Lab Manual was to first initialize the joystick using the Joystick_Initialize() function, call the joystick state using the Joystick_GetState() function and print the state using the Print() function. These functions should all be called from the “Blinky.c” file. After completing the lab, a better understanding of the GPIO was achieved, especially with the underlying registers FIOPIN, PINSEL and FIOMASK. In addition, a better understanding of how to implement any additional hardware with the NXP 1700 series chipset and uVision IDE was achieved.

Reference:

- 1) NXP User Manual, <https://www.nxp.com/docs/en/user-guide/UM10360.pdf>, 2020
- 2) ARM Keil User Guide, https://www.keil.com/support/man/docs/mcb1700/mcb1700_intro.htm, 2020