

|  |  |
| --- | --- |
| **Course Number** | **COE 718** |
| **Course Title** | **Embedded Systems Design** |
| **Semester/Year** | **F2020** |
| **Lab No** | **1** |
| **Instructor Name** | **Saber Amini** |
| **Section No** | **03** |

|  |  |
| --- | --- |
| **Submission Date** | **09/29/2020** |
| **Due Date** | **09/29/2020** |

|  |  |  |
| --- | --- | --- |
| **Name** | **Student ID** | **Signature\*** |
| **Vatsal Shreekant** | **500771363** |  |

*\*By signing above, you attest that you have contributed to this submission and confirm that all work you have contributed to this submission is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a “0” on the work, an “F” in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at:*

*www.ryerson.ca/senate/current/pol60.pdf*

**Introduction**

The uVision IDE features a preloaded 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.

The requirements of Lab1 are to modify the Blinky sample program to feature Joystick

functionality and to simulate the inputs of the Joystick direction through a print statement in the debug viewer window. The preinstalled C functions such as Joystick\_Initialize() can be called in the main Blinky file.

The 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. 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.

**Procedure**

1. Load Blinkly 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 ‘Manage Run-Time Environment’ window will open. Select the Joystick API. Select OK button.
3. Modify the C coded in the Blinkly.c file located under the Project window on the left. Select SWO Trace🡪Source Files🡪 Blinky.c. Add the following code:

\* 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

#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;

uint32\_t AD\_avg = 0;

uint16\_t AD\_value = 0;

uint16\_t AD\_print = 0;

int32\_t joy;

LED\_Initialize(); // LED Initialization

ADC\_Initialize(); // ADC Initialization

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 devided 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;

}

}

}

1. Compile project using the build button and start the simulation by selecting the debug button.
2. Select Peripherals🡪GPIO Fast Interface🡪Port 1 from toolbar and run.
3. Under GPIO 1 window, check boxes under the “Pins” label to implement each direction of joystick movement as follows:

Pin 20: Center joystick

Pin 26: Left joystick

Pin 24: Right joystick

Pin 23: Up joystick

Pin 25: Down joystick

**Conclusion**

While performing the Lab, some mistakes were encountered. One of them was 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.

Hence, after referring to the manual, it became clear that modifying “Blinky.c” for joystick implementation would be more efficient.

Moreover, during the execution of this lab, another hurdle was encountered when implementing the C code provided in the Lab Manual. The C code was implemented directly into the “Blinky.c” file. Thus, it was realized that the functions Joystick\_GetState() and Joystick\_Initialize() in the “Joystick\_MCB1700.c” already implement low-level register commands.

Lastly, the print function specified in the lab manual was a huge benefit as opposed to being physically present in the computer labs. By deselecting the pins in the GPIO Fast Interface, the ports could simulate the expected results. All the results were possible by initializing the joystick using the Joystick\_Initialize() function and then calling the joystick state using the Joystick\_GetState() function.

**References**

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