TO: Professors J. Hung and V. Nelson
FROM: Demetris Coleman
SECTION: 003 – Tuesday 3:30 p.m.
DATE: October 21, 2016
SUBJECT: Interfacing Devices Using Parallel IO (Directional Pad)

Abstract

This report describes the design and test of an interrupt-driven C program for the STML32L100 microcontroller to interface with a directional pad (D-pad) with auxiliary buttons, similar to one on a hand-held game. The pressed D-pad button is determined without interfering with the main programs function.

Problem Description

The design requires a C program for the STM32L1xx microcontroller to handle button presses from the Direction Pad in Figure 1. The device has to be physically connected to the microcontroller to interface through parallel I/O ports. The program is to display, on 4 LEDs, a continuously looping binary-coded decimal counter that ranges from 0 to 9 [1]. If a D-pad button is pressed, it should be detected via an interrupt, be identified, and the button's opcode (shown in Table 1) should be displayed on the LEDs, instead of the count [1]. The opcode should be shown for approximately 5 seconds without interrupting the count [1].

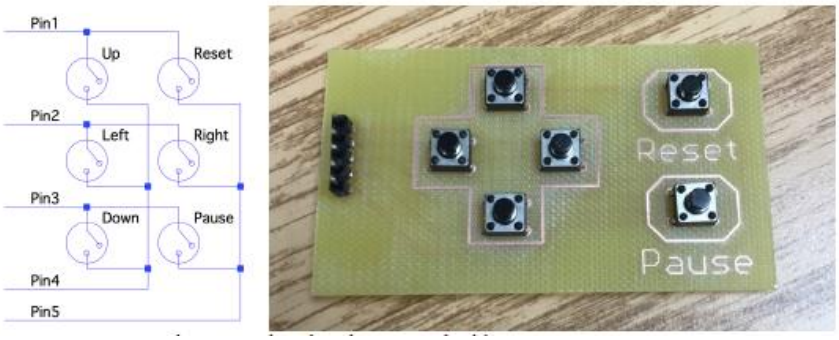

Figure 1: Directional Input Pad with two face buttons (right) and its circuit schematic (left). From Bolton, 2016.

Table 1: D-pad Button Opcode Values

D-pad Button	Opcode
Up	0x01
Left	0x02
Right	0x04
Down	0x08
Pause	0x10
Reset	0x20

Solution Description

The D-pad is a simple network of switches that shorted a row to a column when a button was pressed. For the setup, which can be seen in Figure 2, the GPIOC pins are set as outputs for the LED display. Three GPIOB pins are set as outputs and two were set as inputs with internal pull-up resistors. The STM32L100 input pins with pull-up resistors are connected to columns (pins 4 and 5 in Figure 1) which are also connected to AND gate inputs. The AND output is connected to GPIO pin PA1 to trigger an interrupt when the signal falls low. The D-pad rows (pins 1-3) are connected to output pins on the STML32L100.

The software is split in to two major parts; the main program and the interrupt service routine (ISR). The main program (Lab 5, line 145), seen in the first entrance of the appendix, calls the PinSetup function to configure the microcontroller and enters an infinite loop that contains a software delay (lab 5, line 62) that lasts approximately 1 second, a first_count (Lab 5, lin3 69) function that implements the rolling count, and an “IF” statement that chooses whether to display the counter or the button’s opcode using the global variable DisplayFor5.

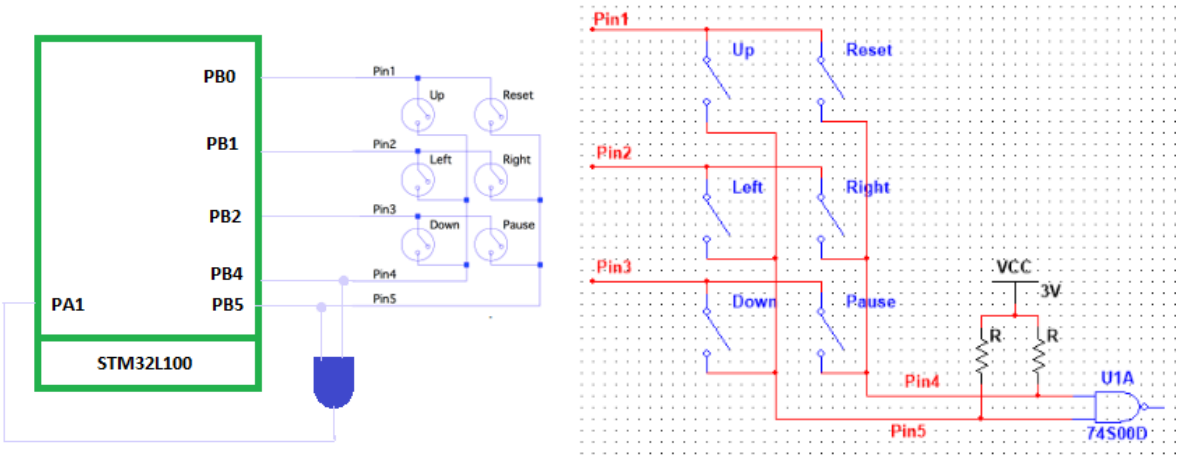

Figure 2: Physical connection of D-pad and STM32L100 (left) and schematic (right) showing pull-up resistor connection.

The ISR (lab 5, line 80) is called when the voltage falls from high to low on pin PA1. The ISR sets DisplayFor5 to 5, disables interrupts, and then implements a scanning algorithm. The algorithm checks if Pin5 is high to determine if the column for Pin4 or Pin5 has been pressed. After determining and saving the column, the rows are set high one by one until both columns are set high again. The event of both columns being set high occurs when the row connected to the column by the button press is set high. After determining the row and column, a nested "IF-ELSE" statement is used to assign the appropriate opcode. Then the external interrupt pending request and interrupt mask registers are reset, interrupts are re-enabled, and the rows are set back to logic LOW. This allows columns to fall from logic HIGH to LOW and trigger an interrupt when a button is pressed.

The choice to set the rows last and the implementation of the scanning algorithm caused errors in the program's performance. It seemed that the up button was always detected as the pressed button, no matter what was pressed. By placing breakpoints on the statements that set the opcode to the correct value, it was discovered that whenever a button was pressed the

correct opcode was actually selected. In the next lab period, it was found that this happened because the interrupt was triggered by the rows being set low after enabling the interrupts. This retriggered the interrupt, but the next time through, the algorithm saw that the pin5 column was high and assumed the pin4 column was pressed, even though they were both high. Then it selected the first row that was checked because the columns were both already high.

To fix this, the statement to set the rows low was moved before clearing the pending request register and some unnecessary statements were removed. The scanning algorithm was modified to check if the pin4 and pin5 columns were grounded individually and choose a default case that didn't affect the opcode if neither was grounded. This can be seen starting at line 240 in the *Modified ISR* section of the appendix compared to lines 87 – 92 in the *Lab 5 Program* section.

Experiments and Experimental Results

The program was tested to see if it entered the interrupt service routine by placing breakpoints inside of it. Initially, the program never entered the ISR. This happened because the IRQ Handler function, EXTI1_IRQHandler (lab 5, line 80), was misspelled and the ISR did not clear pending flags and setup the IMR and PR for external interrupts before exiting.

After making corrections, the period of first_count was tested by measuring the LSB of the counter. This was found to be 1.000168 s. Interrupt operation was verified by inspecting the LEDs and using the watch window to track the variable OPCODE for the debugger. The same tools were used with the addition of the variables row and column to verify that the correct opcode was being detected. Sometimes the buttons produced the correct opcode, but then the program immediately snapped to an incorrect code. Other times it produced the wrong code all

together. The problem seemed to be that the wrong row was sometimes detected. Adding short delays between setting a row high and checking to see if the column returned to high in our row scan algorithm fixed the problem. The logic analyzer (Figure 3) was used to show that the program worked to the specification explained in the design description section. However, the program returned to its former behavior before the end of lab. While it was working the display did not display anything for some opcodes because it had too few bits to display two of the opcodes (0x20, 0x10).

Results

The counter continuously counts from 0 to 9, and when a D-pad button is pressed, the interrupt routine displays an opcode on the counter's display for 5 seconds before displaying the counter again without a break in its count sequence. This operation can be seen in Figure 3. Showing the opcode on the display was not always optimal because they are 8 bit numbers with the highest (0x20) needing at least 6 bits to be sufficiently displayed. Instead of using the LED display, each button was verified to set the OPCODE variable to the correct opcode (as seen in Table 2) using the watch window debugging tool. As described in the previous section, the program did have a problem producing the value 0x01, despite what button was pressed. However, in following labs, each button was used to call a unique display on an LED and specific musical notes. The relationship between the LED matrix and the opcodes are described Table 2 and shown in Figure 4. This served as an alternative to showing the opcode. The visual, real time feedback allowed for a better understanding of the system and led to the modifications (lines 240-248) in the *Modified ISR* section of the appendix.

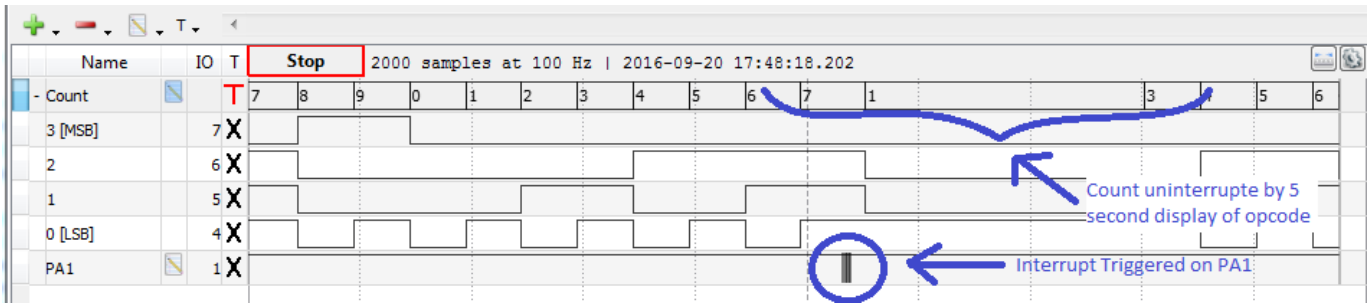

Figure 3: Counter display interrupted by Opcode value for UP button for 5 seconds

Table 2: Correlation of OPCODE and LED Matrix Display

D-pad Button	Opcode	Display
Up	0x01	Up Arrow
Left	0x02	Left arrow
Right	0x04	Right Arrow
Down	0x08	Down Arrow
Pause	0x10	Letter P
Reset	0x20	Invert Current Display

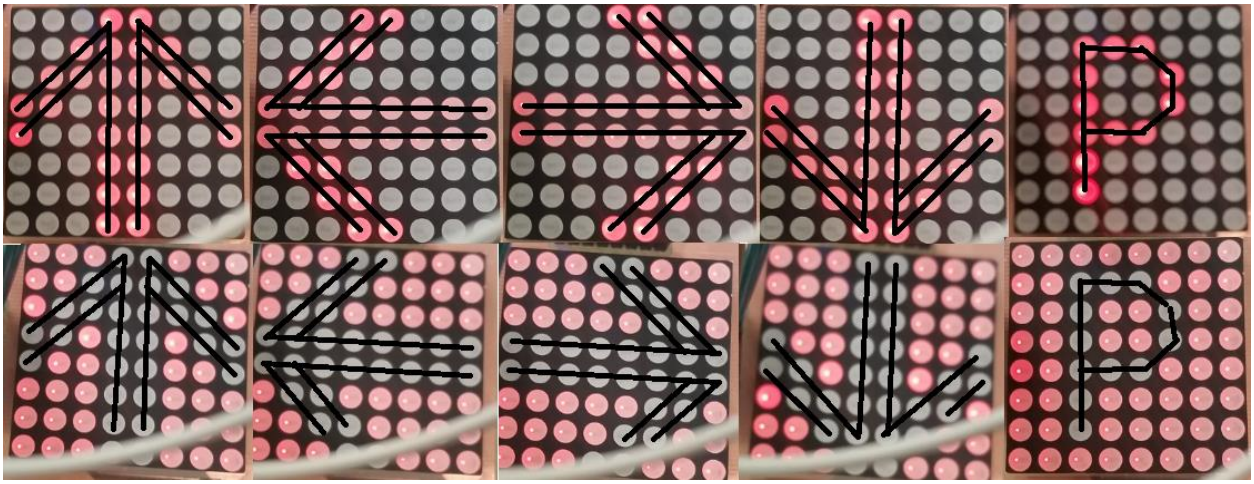

Figure 4: LED Matrix Display Confirming OPCODE Values

Conclusions

At the end of lab 5, all of the specifications for the program except the D-pad scanning algorithm completely met the design specifications. After modifying the algorithm in later labs, the D-pad interrupt meets all the desired design specifications. It successfully scans rows and columns to detect which button was pressed and output the desired opcode specified in Table

2. It also successfully sets the microcontroller conditions so the interrupt can be triggered again after leaving the ISR.

There are a few things to take away from interfacing devices using parallel IO. For hardware, it is important to make sure the microcontroller is configured correctly. On the software side, it is important to check the order of resetting the conditions for the interrupt at the end of the ISR. Having a command that triggers your interrupt after clearing flags and interrupt request will cause problems. Also, care should be taken to check each possible condition in the scanning algorithm, rather than assuming that it is another if the other conditions are not satisfied. A more efficient scanning algorithm could be implemented by giving the D-pad more carefully thought out Opcodes. For instance, the opcode could be calculated in the scanning algorithm if the opcode was based on the row column position of each button in the circuit.

References

[1] M. Bolton, *Lab 5 Alternate Assignment*. Auburn, AL: ELEC 3040/50 Lab Manual, 2016

Appendix

- Lab 5 program
- Modified ISR

Lab 5 Program

```
  1 /*
  2 *Lab 5 - Dpad Interface Using Parallel I/O
  3 *Demetris Coleman and Justin Whaler
  4 *ELEC 3040/ELEC 3050
  5 *9/11/2016
  6 */
  7
  8 /* old buggy code for row scan in ISR
  9     //may need to nest the 2nd and 3rd if statements if row always turns
out to be 3
 10     GPIOB->ODR |= 0x0001;//set row 1 high
 11     if((GPIOB->IDR & 0x0030) == 0x0030) {//if PB5=1 and PB4=1 row 1
presseed
 12        row = 1;
 13     }
 14     GPIOB->ODR |= 0x0002;//set row 2 high
 15     if ((GPIOB->IDR & 0x0030) == 0x0030){//if PB5=1 and PB4=1 row 2
presseed
 16        row = 2;
 17     }
 18     GPIOB->ODR |= 0x0004;//set row 3 high
 19     if ((GPIOB->IDR & 0x0030) == 0x0030){//if PB5=1 and PB4=1 row 3
presseed
 20        row = 3;
 21        */
 22
 23 //************************************************************
 24 //         Library includes and Variables
 25 #include "STM32L1xx.h"             /* Microcontroller information */
 26 int count = 0;                        //Global variables
```

```
 27 int DisplayFor5 = 0;
 28 int column, row;
 29 uint16_t OPCODE = 0;
 30
 31
 32 //**************  Pin Setup **********************
 33 void PinSetup () {
 34 /* Configure PA0 as input pin to read push button */
 35   RCC->AHBENR |= 0x01;           /* Enable GPIOA clock (bit 0) */
 36   GPIOA->MODER &= ~(0x000000FF); // set PA1 as input port
 37   RCC->AHBENR |= 0x02;           /* Enable GPIOB clock (bit 1) */
38   RCC->AHBENR |= 0x04;          /* Enable GPIOC clock (bit 2) */
 39 //GPIO pin setup
 40   //GPIOA->MODER &= ~(0x00000030); // set PA2 as input port
 41   GPIOC->MODER &= ~(0x000FFFFF); // Clears PC0-PC9
 42   GPIOC->MODER |=  (0x00055555); // and sets them as outputs
 43   GPIOB->MODER &= ~(0x0000FFFF); // Clears PB0-PB5
 44   //GPIOB->MODER |=  (0x0000057F); //sets PB0-PB2 (inputs) and PB3-
5(outputs)
 45   GPIOB->MODER |=  (0x00000015);//sets PB0-PB3 (outputs) and PB4-
5(inputs)
 46 //D-pad setup
 47    GPIOB->ODR &= 0xFFF8; //set rows 1-3 to ground
 48    GPIOB->PUPDR &= ~(0x00003FC0);// clear PB4 and PB5
 49    GPIOB->PUPDR |=  (0x00001540); //pull up resistors on PB4 and 5
 50 //Interupt Initialization
 51    EXTI->FTSR = 0x0000;
 52    EXTI->FTSR |= 0x0002;//falling edge trigger. RTSR for rising
 53    EXTI->IMR = 0x0003;
 54    EXTI->PR |= 0x0003;
 55    SYSCFG -> EXTICR[0] &= 0xFF00;
 56    NVIC_EnableIRQ(EXTI1_IRQn); //PA1
 57    NVIC_ClearPendingIRQ(EXTI1_IRQn);
 58    __enable_irq();
 59 }
 60 //************** Functions ******************
 61 /* Half Second Delay Function */
 62 void delay(void) {               // 1 second delay
 63   int i, j;
 64   for (i=0; i<100; i++) {        // outer loop
 65     for (j=0; j<20000; j++) {    // inner loop
 66       }
 67   }                              // Just letting time pass
68 }
 69 void first_count() {      // increments global variable count
 70     if (count >= 9) {            // loop around if counter reaches 9
 71       count = 0;
 72       }
 73       else {
 74          count++;                  // increment count
 75       }
 76 }
 77 //-------------------------------------------|
 78 //            Interupt Handlers              |
 79 //-------------------------------------------|
 80 void EXTI1_IRQHandler() {//inputs(0-2) and outputs(4-5) may need to be
switched
```

```
 81    int i=0;
 82    //if(DisplayFor5 != 5) {
 83    DisplayFor5=5;
 84    __disable_irq();
 85    //if ((GPIOB->IDR & 0x0030) == 0x0030) {
 86    for (i=0 ; i<90; i++){  } //debouncing delay
 87    if((GPIOB->IDR & 0x0030) == 0x0010) {//if PB5=0 and PB4=1 column
pressed is 5
 88           column = 5;
 89     }
 90     else {//else column is 4
 91        column = 4;
 92     }
 93 //scan rows
 94     GPIOB->ODR |= 0x0001;//set row 1 high
 95     for (i=0;i<10;i++);
 96     if((GPIOB->IDR & 0x0030) == 0x0030) {//if PB5=1 and PB4=1 row 1
presseed
 97        row = 1;
 98     }
 99     else {
100
101        GPIOB->ODR |= 0x0002;//set row 2 high
102          for (i=0;i<10;i++);
103        if ((GPIOB->IDR & 0x0030) == 0x0030){//if PB5=1 and PB4=1 row 2
presseed
104           row = 2;
105        }
106        else {
107           GPIOB->ODR |= 0x0004;//set row 3 high
108              for (i=0;i<10;i++);
109           if ((GPIOB->IDR & 0x0030) == 0x0030){//if PB5=1 and PB4=1 row 3
presseed
110             row = 3;
111           }
112        }
113     }
114
115 //check button by column and row combination
116     if (column == 4) {
117        if (row==1)
118           OPCODE=0x0001;//up
119        else if (row==2)
120           OPCODE=0x0002;//left
121        else if (row ==3)
122           OPCODE=0x0008;//down
123     }
124     else if (column==5) {
125        if (row==1)
126           OPCODE=0x0020;//reset
127        else if (row==2)
128           OPCODE=0x0004;//right
129        else if (row ==3)
130           OPCODE=0x0010; //pause
131     }
132  //}
133     EXTI->IMR = 0x0003;
```

```
134     EXTI->PR |= 0xFFFC;
135    for(i=0;i<40000;i++);
136     NVIC_ClearPendingIRQ(EXTI1_IRQn);
137     __enable_irq();
138     GPIOB->ODR &= 0xFFF8; //set rows 1-3 to ground
139     //set PB4 and 5 back high?
140 }
141
142
143 //-------------------------------------|
144 //              Main                   |
145 //-------------------------------------|
int main(void) {
147    // initialize port directions and variables
148    PinSetup();
149
150    // counter loop
151    while(1) {                     // endless loop
152       delay();               // Run Delay
153       first_count();   // Execute Counter by one step
154        if (DisplayFor5<=0){
155            GPIOC->ODR = (count); // write  count to PC0-PC7
156           }
157        else {
158           DisplayFor5--;
159           GPIOC->ODR = (OPCODE);
160        }
161    }
162 }
163
```

Modified ISR

```
233 void EXTI1_IRQHandler() {// Dpad - inputs(0-2) and outputs(4-5) may need
to be switched
234    int i=0;
235    //if(DisplayFor5 != 5) {
236    //DisplayFor5=5;
   //__disable_irq();
238    //if ((GPIOB->IDR & 0x0030) == 0x0030) {
239    for (i=0 ; i<50; i++){  } //debouncing delay
240    if((GPIOB->IDR & 0x0030) == 0x0010) {//if PB5=0 and PB4=1 column
pressed is 5
241           column = 5;
242     }
243     else if((GPIOB->IDR & 0x0030) == 0x0020) { //else column is 4
244        column = 4;
245     }
246     else {
247        column = 0;
248     }
249 //scan rows
250     GPIOB->ODR |= 0x0001;//set row 1 high
251     for (i=0;i<10;i++);
252     if((GPIOB->IDR & 0x0030) == 0x0030) {//if PB5=1 and PB4=1 row 1
```

```
presseed
253         row = 1;
254      }
255      else {
256
257         GPIOB->ODR |= 0x0002;//set row 2 high
258         for (i=0;i<10;i++){}
259         if ((GPIOB->IDR & 0x0030) == 0x0030){//if PB5=1 and PB4=1 row 2
presseed
260            row = 2;
       }
262         else {
263            GPIOB->ODR |= 0x0004;//set row 3 high
264               for (i=0;i<10;i++){}
265            if ((GPIOB->IDR & 0x0030) == 0x0030){//if PB5=1 and PB4=1 row 3
presseed
266              row = 3;
267            }
268         }
269      }

271 //check button by column and row combination
272       if (column == 0) {
273       }
274       else if (column == 4) {
275        if (row==1)
          OPCODE=0x0001;//up
277        else if (row==2)
278           OPCODE=0x0002;//left
279        else if (row ==3)
280           OPCODE=0x0008;//down
281      }
282      else if (column==5) {
283         if (row==1)
284            OPCODE=0x0020;//reset
285         else if (row==2)
286            OPCODE=0x0004;//right
287         else if (row ==3)
288            OPCODE=0x0010; //pause
289      }
290  //}
291
292      GPIOB->ODR &= 0xFFF8; //set rows 1-3 to ground
293     for(i=0;i<30000;i++);
294      EXTI->PR |= 0x0002;
295      //NVIC_ClearPendingIRQ(EXTI1_IRQn);
296      for(i=100; i>0; i--);
297      //__enable_irq();
298
299      //set PB4 and 5 back high?
300 }
```

Cover Sheet

Deleted colons from header.

Fixed grammar/formatting issues

Changed tense to make it consistent throughout the paper

Clarified Dr.Nelms' questions on page 3 in the solutions description section

Added more detail to Experiments and Experimental Results section

Added a bit more detail to conclusion

Modified Figure 4