COMPUTATION BOOK

NAME

Grant Abella

COURSE ECE 322 02 Lab Book A

TOPS™

0 25932 35061 0

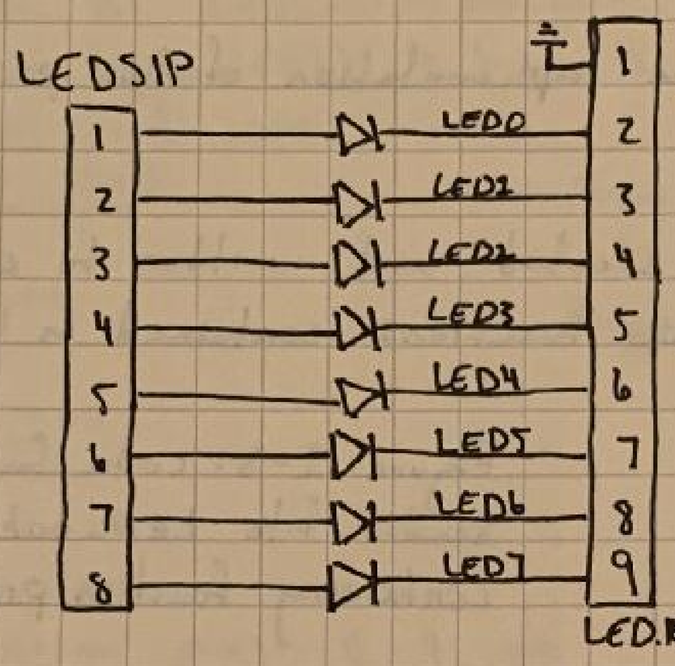
Laboratory #1: Switch and LED Interfacing ADVISOR

Grant Abella
8/23/18

Laboratory Objective: Review interface techniques for controlling the LEDs on the Atmega128 based STK128/64 starter kit. Partner

Equipment Used:

- Atmega 128A Kit # 26



this goes below the figure

Figure 1-1: LED arrangement schematic for the STK128/64 starter kit used in this lab.
Also, try to incorporate the page number i.e. Figure 3-1

1. a) In order to achieve a continuous cycle of the board's LEDs being ON for 1 second and OFF for 2 seconds, the C program will need to be split up into three files. By maintaining a modular program in this fashion, future adjustments can be made to the code quickly as the logic is organized into relevant files.

The flowchart for this program will be as follows:

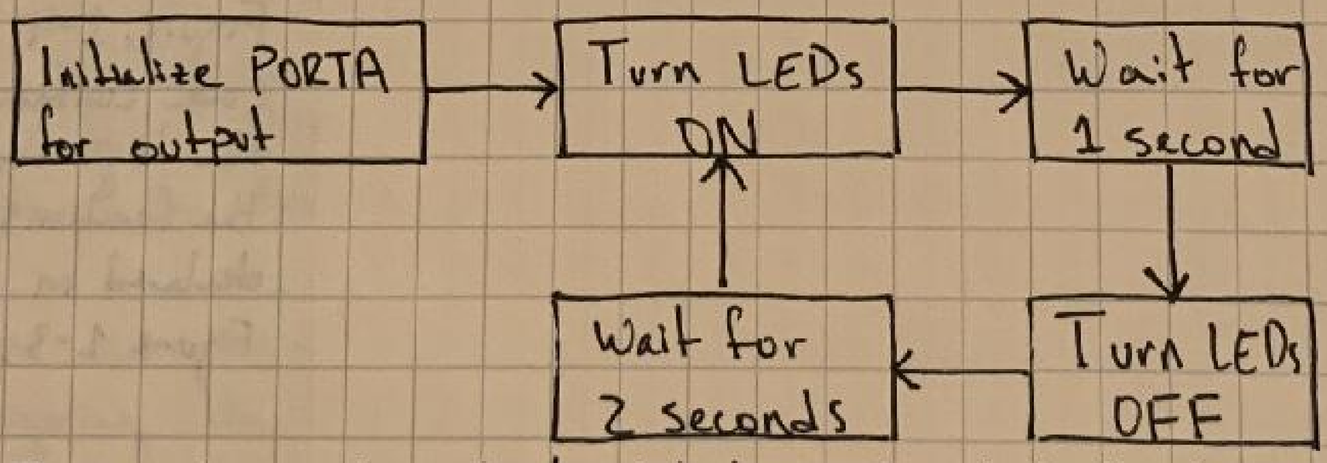


Figure 1-2: Flowchart detailing the high-level logic for cycling the LEDs on the Atmega 128A ON for 1 second and OFF for 2 seconds continuously.

Because the LEDs will be connected to PORTA on the board, the port must be configured for output. This will be accomplished by initializing the data direction of PORTA (DDRA) to 0xFF, meaning all 8 pins on the port will be set to output.

The toggling of LED states from ON to OFF ~~with~~ continuously will be accomplished by setting the value of the pins on PORTA to 1 and 0 (respectively) continuously.

8/23/18
2:26 AM

Laboratory #1: Switch and LED Interfacing

Grant Abella
8/23/18

Finally, the waiting in between the switching of LED states could be accomplished by either incorporating a timer interrupt or by simply using the delay function _delay_ms(). In the field, the use of such a function would be heavily discouraged as such a task would normally be handled using an onboard timer, however for the sake of time, the delay function will serve just fine.

As discussed on the previous page, the implementation of this program was split into 3 files.

The first file, LED_out.h, was created and written to contain the function prototypes for the two functions contained in LED_out.c.

```
/*
 * LED_out.h
 * Header file containing prototypes for functions to
 * initialize PORTA and handle the toggling of onboard LEDs.
 *
 */

#ifndef LED_OUT_H_
#define LED_OUT_H_

// Function to initialize PORTA for use with the LEDs.
void LED_init(void);

// Function to toggle LEDs
void toggle_LED(void);

#endif
```

Figure 1-3: code for the header file LED_out.h containing function prototypes.

The next file, LED_out.c, holds the actual logic for the functions declared above.

```
/*
 * LED_out.c
 * Program file containing definitions for functions to
 * initialize PORTA and handle the toggling of onboard LEDs.
 *
 */

#define F_CPU 16000000UL                    // Set 16 MHz oscillation
#include <avr/io.h>
#include <util/delay.h>
#include <LED_out.h>

// Function to initialize PORTA for use with the LEDs.
void LED_init(void)
{
        DDRA = 0xFF;        // Set PORTA to output
        PORTA = 0x00;       // Set initial PORTA value to 0 (off)
};

// Function to toggle LEDs
void toggle_LED(void)
{
        int toggle = 0;                                 // toggle value initially set to false
        while(1)
        {
                if (toggle)                                     // if toggle is true
                {
                        PORTA = 0x00;                   // set PORTA to off
                        _delay_ms(2000);    // wait 2s
                        toggle = 0;                             // set toggle to false
                }
                else                                            // if toggle is false
                {
                        PORTA = 0xFF;                   // set PORTA to on
                        _delay_ms(1000);    // wait 1s
                        toggle = 1;                             // set toggle to true
                }
        }
};
```

Figure 1-4: Code containing the logic for the functions declared in Figure 1-3.

GA
8/23/18
2:52 PM

Laboratory #1: Switch and LED Interfacing

Grant Abela
8/23/18

The code shown in Figure 1-4 implements the logic flow described by the flowchart in Figure 1-2, and the three paragraphs preceding it on pages 3 and 4.

Finally, everything is brought together in the third file, main.c, where the program will start. This file simply includes the first file, LED_out.h, and calls the LED_init() function to first initialize PORTA, and then calls toggle_LED() to begin the continuous process of turning the LEDs ON and OFF.

```
// Grant Abella    8/23/18
// ECE 322 Laboratory #1

#include <asf.h>
#include <LED_out.h>

int main (void)
{
        board_init();
        LED_init();
        toggle_LED();
}
```

Figure 1-5: code contained in main.c. Calls the two functions to initialize and toggle the LEDs on the board.

After wiring pins 0-7 on PORTA to pins 0-7 for the LEDs (Li), and programming the board, the implementation was successful. As intended, the LEDs cycled ON and OFF for the seemingly correct timings. However, further signal inspection is necessary to say that the timings are absolutely correct (see part 1b below).

1. b) In order to verify that the LED toggle timings are correct, one of the 8 signals on PORTA will have to be viewed on the oscilloscope. Doing this is a simple matter of connecting one wire from PORTA to an oscilloscope probe and configuring the display on the scope correctly to view the signal period.

After connecting pin 0 of PORTA to channel 1 of the scope, and configuring the window correctly, the trace below was found:

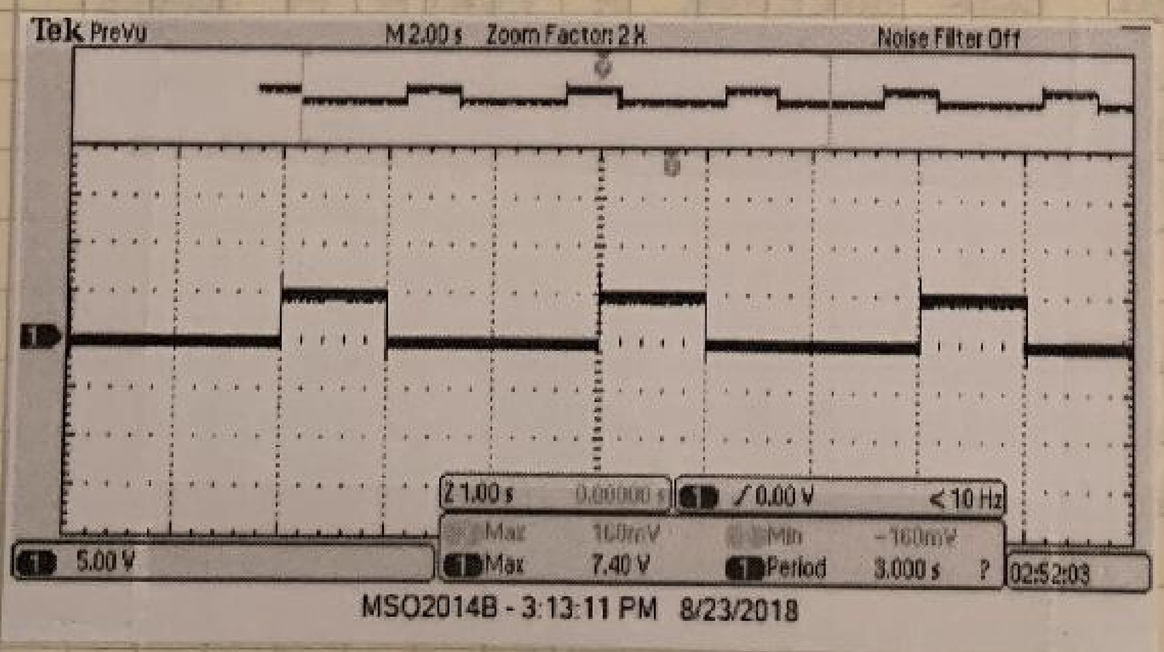


Figure 1-6: oscilloscope reading of pin 0 of PORTA showing LED switching timings. X-axis set to 1 sec/division, Y-axis set to 5 volts/division.

GA
8/23/18
3:14 PM

Laboratory #1: Switch and LED Interfacing

Grant Abella
8/23/18

With the X and Y axes set to 1 sec/division and 5 volts/division respectively, we can see that the signal holds a value of 5 volts for 1 second before holding at 0 volts for 2 seconds. Together, this makes for a period of 3 seconds as expected. This cycle repeats continuously.

Also, when connecting the oscilloscope probe to the system, the ground clip on the probe was connected to one of the ground pins (GND) on the board. It was important to use the board's ground since all of the signals coming off of it will be relative to that ground. This makes for a safe and accurate reading of the signal.

2. a) The general flowchart for a program that sequentially and individually lights up LEDs L0 through L7 with 1 second between switching is shown below:

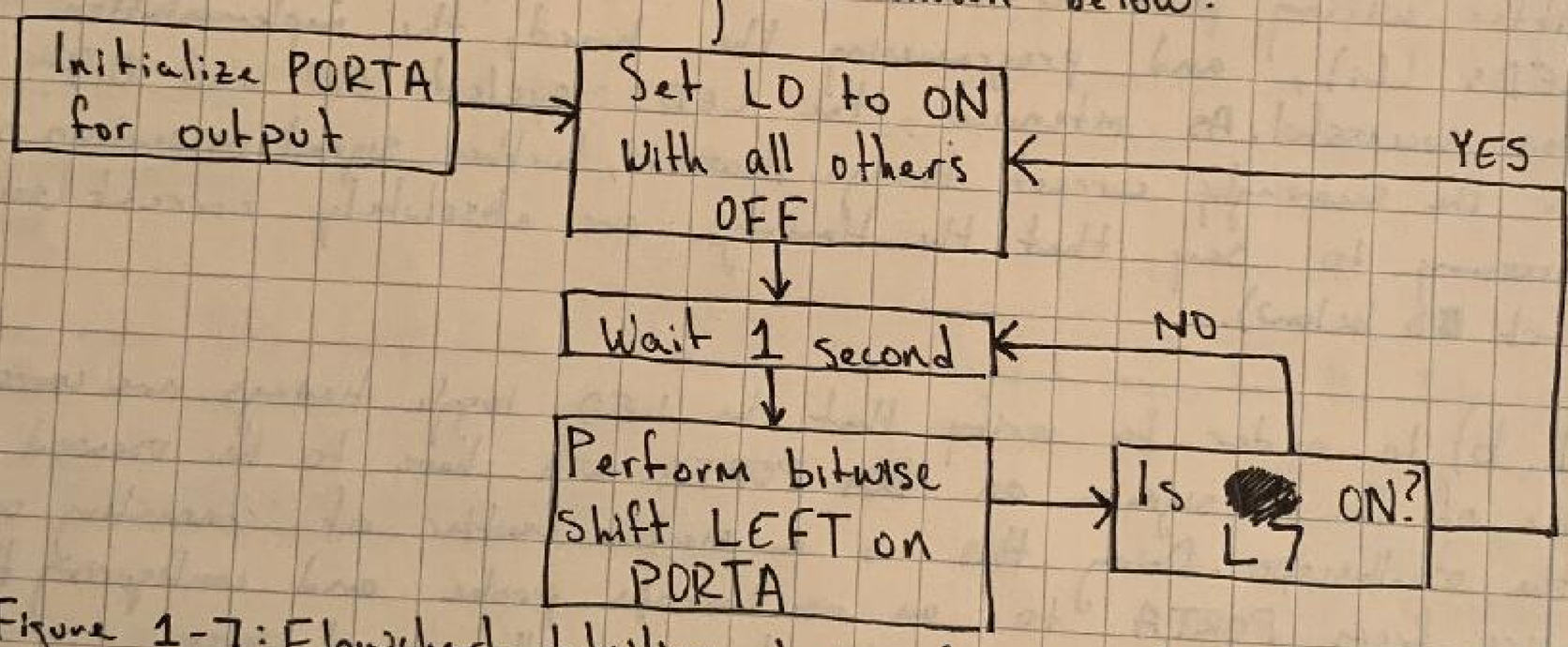


Figure 1-7: Flowchart detailing logic for sequentially turning ON LEDs L0 through L7 continuously.

Just like with the program in part 1 of this lab, this program will begin by setting the data direction of PORTA to output. This is obviously because the onboard LEDs will be wired up to the pins on PORTA, so they must be configured as output.

After port initialization, the program will then turn on only LED L0. This will be accomplished simply by setting the LSB on the port to 1 (or PORTA = 0x01).

Next, the delay_ms function will be utilized again to wait for 1 second before a bitwise shift

Dan
8/23/18
3:30PM

Laboratory #1: Switch and LED Interfacing

Grant Abella
8/23/18

left on the PORTA register. This shifting will change the illuminated LED L_i to L_{i+1}, which is the behavior we desire for this program. The only concern associated with this is when L7 is lit and a shift back to L0 must occur.

This problem is addressed by simply checking the value of PORTA. If L7 is lit, the register is reset back to 0x01 and the program repeats. If any LED besides L7 is ON, the wait is performed, and another shift occurs, followed by another check of the register, etc.

The project for this program was split into three files, with Main.c and LED_out.h containing the same code as in the previous lab section (seen in Figure 1-5 and Figure 1-3 respectively). With that said, the code for LED_out.c differed greatly from the previous part.

```
/*
 * LED_out.c
 * Program file containing definitions for functions to
 * initialize PORTA and handle the toggling of onboard LEDs.
 *
 */

#define F_CPU 16000000UL                // Set 16 MHz oscillation
#include <avr/io.h>
#include <util/delay.h>
#include <LED_out.h>

// Function to initialize PORTA for use with the LEDs.
void LED_init(void)
{
        DDRA = 0xFF;       // Set PORTA to output
        PORTA = 0x00;
};

// Function to toggle LEDs
void toggle_LED(void)
{
        PORTA = 0x01;                                   // Set initial value for the LEDS at 1
        while(1)
        {
                _delay_ms(1000);            // Wait 1s
                if(PORTA == 0x80)           // If current state of LED7 is 1, reset the PORTA value to 0x01
                        PORTA = 0x01;
                else
                        PORTA = PORTA << 1;         // Otherwise perform a bitwise shift left
        }
};
```

Figure 1-8: Code for performing the logic to sequentially turn on LEDs L0 through L7 individually and continuously.

When implementing the program, the LEDs performed as expected, with L0 being lit first, for 1 second, followed by L1, and so on. After 1 second of L7 being ON, the sequence is repeated continuously.

8/23/18

2.b) In order to verify LED switching times, one of 3:43 PM

Laboratory #1: Switch and LED Interfacing

Grant Abela
8/23/18

the 8 signals on PORTA will need to be viewed on the oscilloscope. To achieve this, a wire will be run from pin 0 of PORTA to the first probe on the oscilloscope, with the ground clip for the probe being similarly connected to one of the board's ground pins.

The following figure shows the reading collected from the scope:

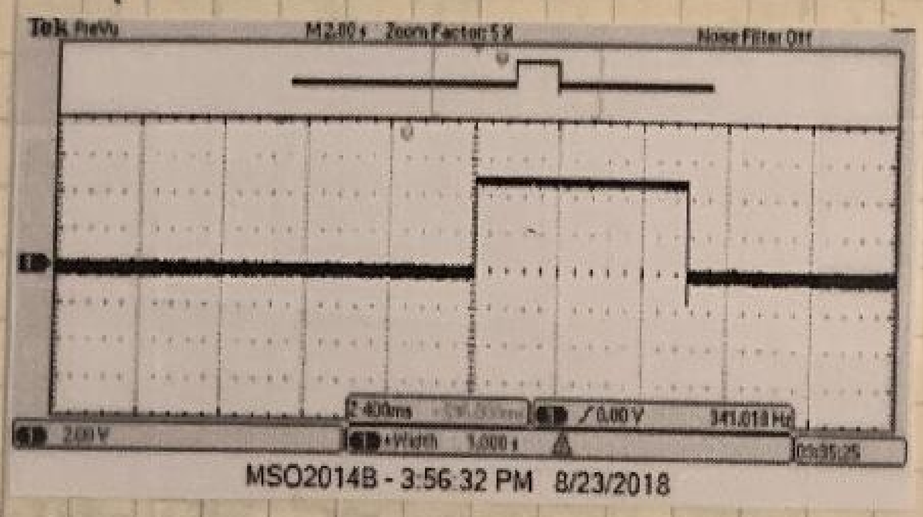


Figure 1-9: Oscilloscope reading of pin 0 of PORTA displaying a period of 8 seconds, 1 second ON, 7 seconds OFF.

With horizontal divisions of 400 ms and the voltage HIGH state remaining for 2.5 divisions, the 1 second switching period is verified.

3. a) The flowchart for a program which uses push buttons to light up LEDs for as long as the buttons are depressed is as follows:

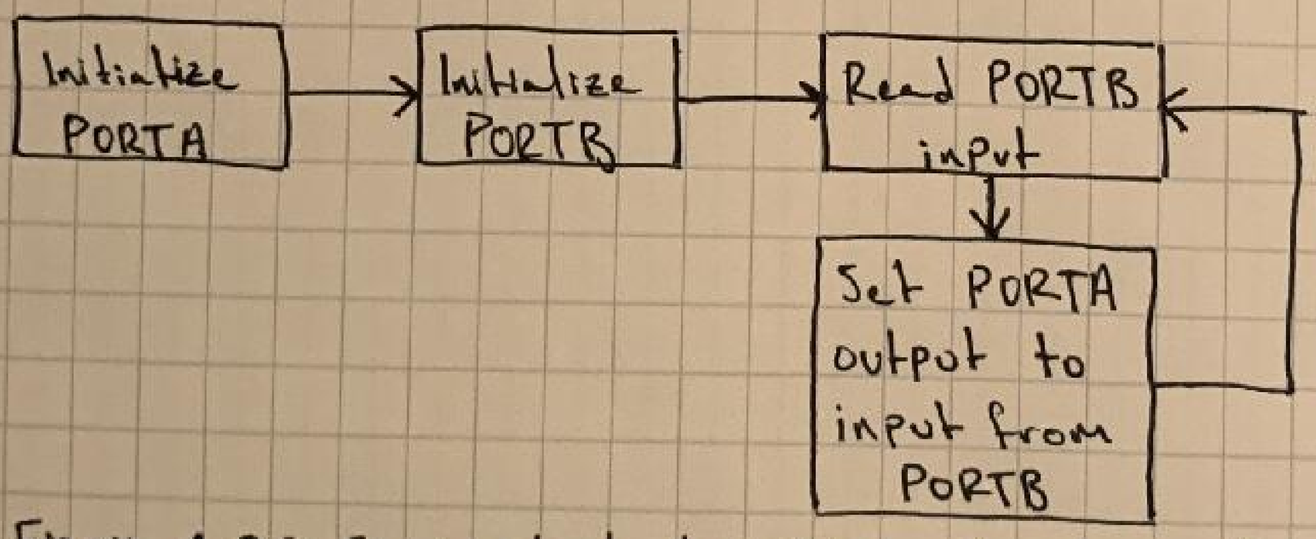


Figure 1-10: Flow chart describing the logic for a program that takes input from the board's key switches and outputs the input for as long as the key is depressed.

For this program, PORTA will be initialized in the same way as in the previous parts of this lab, and PORTB will be initialized as an input, since the key pins will be wired to PORTB.

A function will then read the value of the pins on the port and store that value in a variable. Finally, the LEDs will output

GA
8/23/18
4:06 PM

Grant Abelle
8/23/18

Laboratory #1: Switch and LED Interfacing

the value of this variable by setting PORTA to that value. The code for this logic is contained in the c file below:

```
/*
 * LED_out.c
 * Program file containing definitions for functions to
 * initialize PORTA and PORTD, handle the toggling of onboard LEDs,
 * and handling keypresses.
 *
 */

#define F_CPU 16000000UL          // Set 16 MHz oscillation
#include <avr/io.h>
#include <util/delay.h>
#include <LED_out.h>

// Function to initialize PORTA for use with the LEDs.
void LED_init(void)
{
        DDRA = 0xFF;     // Set PORTA to output
        PORTA = 0x00;
};

void KEY_init(void)
{
        DDRD = 0x00;     // Set PORTD to input
        PORTD = 0xFF;    // Set PORTD as pullup switch
        uint8_t key_pressed;
        while(1)
        {
                key_pressed = 0xFF ^ PIND;          // XOR operation performed with PIND (key) values
                PORTA = key_pressed;                // PORTA assigned the key_pressed value
                _delay_ms(50);                      // Delay of 50 ms to help debounce
        }
}
```

Good Comments!

Figure 1-11: Program file LED_out.c which contains two functions to initialize ports and handle logic for key switches and LEDs.

By performing an XOR operation on the key switches and 0xFF, the switches being pressed are extracted. From here, the resulting value is stored in the variable key_pressed. The LED port is then set to key_pressed's value, and the value is output to the LEDs.

When implementing this code on the board, the only issue that arose was some mild bouncing problems with the keys. This was fixed with the inclusion of the _delay_ms function and some trial and error with the value for time in milliseconds.

4.a) In order to create a program that functions the same as the program above but holds the input value on the LEDs until a different input is given, each time the value of the switches is checked, the value will need to be compared with the previous input. For this to happen, a temporary state variable will have to be implemented.

8/23/18
4:28 PM

Laboratory #1: Switch and LED Interfacing

Grant Abella
8/23/18

A high-level flowchart for this program is shown below:

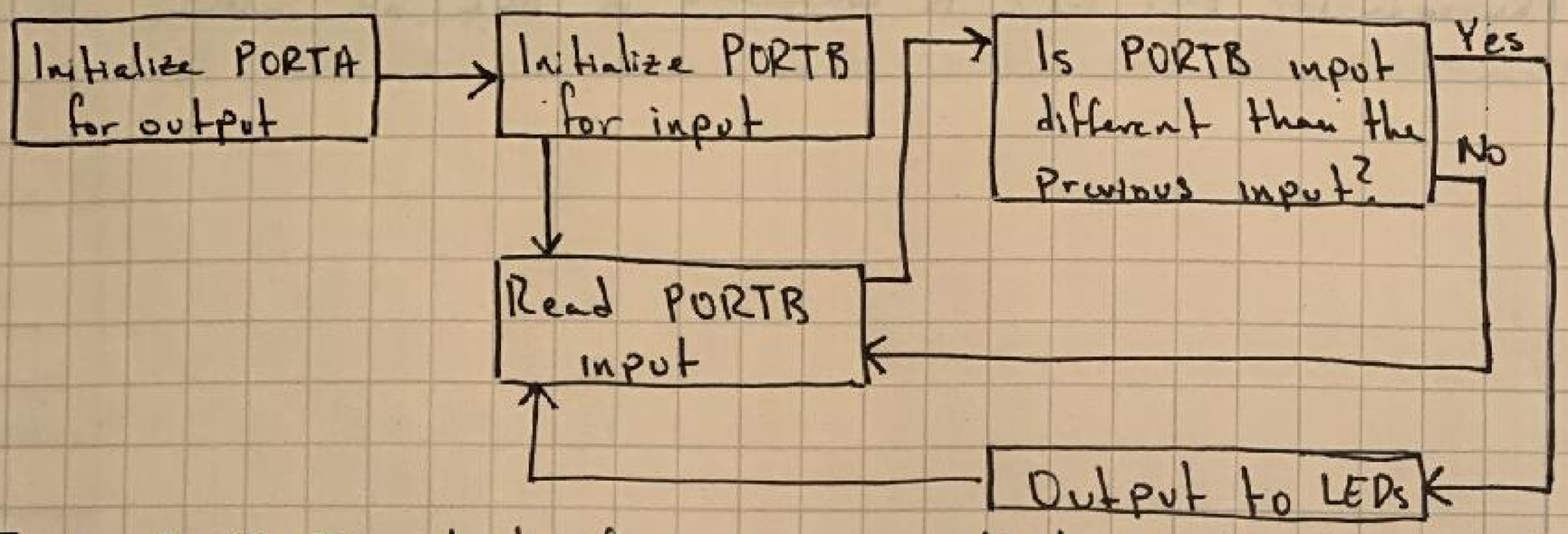


Figure 1-12: Flowchart for program that reads key switch input and outputs the value on the LEDs ~~with~~ until a different input is given.

The code for this logic is shown below:

```
/*
 * LED_out.c
 * Program file containing definitions for functions to
 * initialize PORTA and PORTB, handle the toggling of onboard LEDs,
 * and handling keypresses.
 *
 */

#define F_CPU 16000000UL        // Set 16 MHz oscillation
#include <avr/io.h>
#include <util/delay.h>
#include <LED_out.h>

// Function to initialize PORTA for use with the LEDs.
void LED_init(void)
{
    DDRA = 0xFF;    // Set PORTA to output
    PORTA = 0x00;
};

void KEY_init(void)
{
    DDRB = 0x00;    // Set PORTB to input
    PORTB = 0xFF;   // Set PORTB as pullup switch
    uint8_t key_pressed;
    uint8_t state = 0x00;
    uint8_t firstRun = 1;
    uint8_t loop = 0;
    while(1)
    {
        if (firstRun)
        {
            key_pressed = 0xFF ^ PINB;      // XOR operation performed with PINB (key) values
            state = key_pressed;            // State is simply the key(s) pressed
            firstRun = 0;                   // Not first run anymore
        }
        if (PINB != 0xFF && loop)
        {
            _delay_ms(35);                  // Delay of 50 ms to help debounce
            PORTA = state;                  // Set LEDs to state
            loop = 0;                       // Toggle loop
        }
        while(PINB == 0xFF)
        {
            _delay_ms(35);                  // Delay of 50 ms to help debounce
            PORTA = state;                  // Set LEDs to state
            loop = 1;                       // Toggle loop
        }

        state = state ^ (0xFF ^ PINB);      // state is made up of previous state XOR'd with the key being pressed.
    }
}
```

Figure 1-13: Implementation code for the flowchart shown above.

GA
8/23/18
4:46 PM

Grant Ahrtha
8/23/18

Laboratory #1: Switch and LED Interfacing

When initially implementing the code for this program on the board, there were major bouncing issues with the key switches. This was somewhat fixed by adjusting the delays in the program, however, in the future it will be better to incorporate interrupts for the keys since this issue has been present for the last two parts of this lab. The time spent troubleshooting these issues could have been utilized better if this trial-and-error adjusting of delays had been avoided. Also, the bounce factor will be different for each board, so implementing this program on a different kit might require even more troubleshooting and time.

Other than this problem, the program functioned as intended.

GA
8/23/18
4:56 PM

Postlab:

1. The approach I took when designing the software for this lab was to first design a high-level flowchart for the program, then write the code, implement it and test it, then adjust the code as needed. In the future, I believe adding lower-level blocks (i.e. "set PORTA to 0x0F") would be beneficial when referencing flowcharts for writing code. I found that sometimes when looking at the flowchart for a program it would have helped to break up some blocks into multiple blocks with lower-level code concepts inside them. This would have helped when writing code because in some ways, portions of program logic would have been written before even creating the project in Atmel Studio.

Also, it would have been better coding practice to separate functions relating to key switches and LEDs into separate files. This way if I need to revisit this lab to implement certain functions in this lab to use in future labs it will be easy to grab only specific sections of code or logic instead of having to delete unneeded lines of code.

2. The biggest hardware-related issues for this laboratory was key bouncing problems. I found that sometimes key presses would be registered as multiple presses, making it difficult to troubleshoot certain program functions. This bouncing is a result of the springs inside the key switches oscillating

GA
8/28/18
2:36 PM

Laboratory #1: Switch and LED Interfacing

Grant Abella
8/28/18

after presses. To make matters worse, this bounce factor is different for each board because it depends on the age and quality of the switches.

In the future, I will be implementing more interrupt key-press processing to avoid this issue. I didn't do this for this lab in an attempt to save time, but in hindsight I think I spent more time troubleshooting delays than it would have taken to set up an interrupt system. This will also make the program more compatible with other boards, since using delays like I was is not good because they will have to be adjusted if a different bounce factor is found when implementing on different boards.

3. Two techniques were used to protect the hardware when completing this lab. The first was using the onboard ground (GND) pin when measuring voltage signals on the oscilloscope. This was important because it eliminated any floating voltages by having a reference ground, and preventing any unintended occurrences from happening due to that.

The other ~~Segun~~ technique used in this lab was to pass the pin signals from the keys through a resistor pack. This prevents any large voltage signals from damaging the board. Although this outcome was unlikely, it was still good practice just in case.

Conclusion: By working through this lab, I was able to refresh my skills with the Atmega 128A, embedded C, and Atmel Studio. This is significant because these are all things that will be useful both in future labs and in the real world.

In looking back at this lab, I came across several changes I would make if I were to do the lab over again, and through this, I will be able to more effectively and efficiently complete labs in the future.

ga
8/28/18
3:02 PM

Great lab book! You used plenty of details and explained what you did in lab well!

Flow Chart : -0 24/24
Skeletal : -0 6/6
Post Lab : -0 10/10

40/40

Nonzin

Laboratory #4: External Interrupt Controlled Keypad and LEDs

Grant Abella
Brandon Lampert
9/25/18

Advisor: Dr. Gu[illegible]hlag

Laboratory Objective: Develop external interrupt control interface techniques for keypad control of the LEDs on the Atmega128A-based starter kit.

1	2	3	A	Row 1
4	5	6	B	Row 2
7	8	9	C	Row 3
*	0	#	D	Row 4
Col 1	Col 2	Col 3	Col 4	

Figure 14-1: Keypad configuration for the keypad used in this lab, rows and columns labeled.

Output Pin #	Symbol	PORTC
1	Col 1	bit 0
2	Col 2	bit 1
3	Col 3	bit 2
4	Col 4	bit 3
5	Row 1	bit 4
6	Row 2	bit 5
7	Row 3	bit 6
8	Row 4	bit 7

Table 14-1: Keypad output pin number and corresponding column or row number with PORTC bit included.

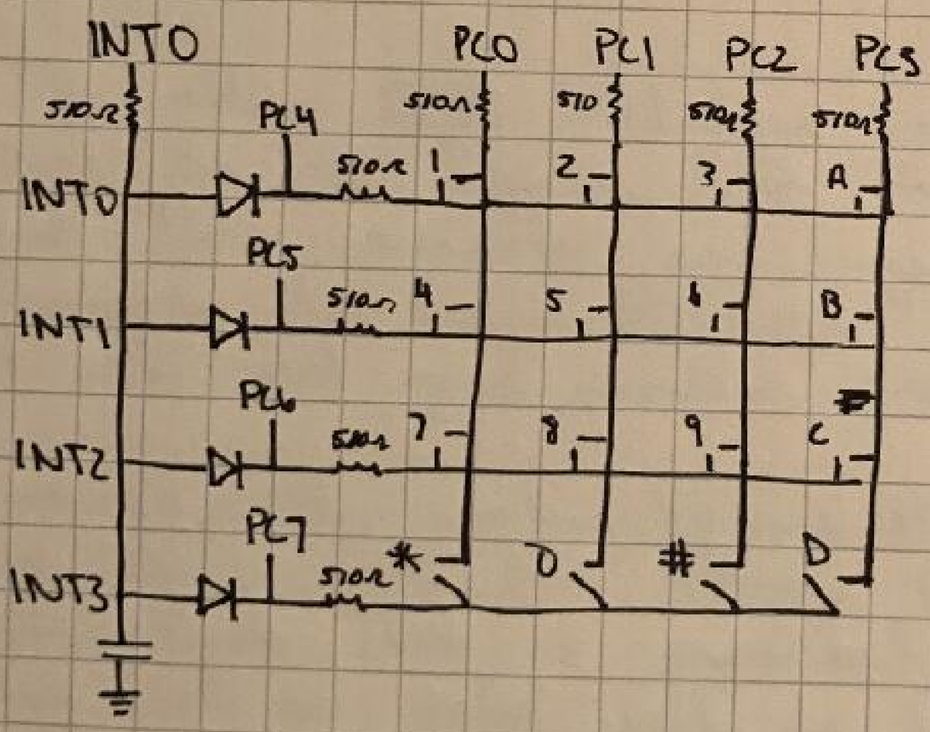


Figure 14-2: Low-level circuit diagram for keypad output with four interrupts used.

Equipment Used:

- Atmega 128A Kit #40
- Oscilloscope EQ-3017
- Breadboard, pin headers

9/25/18 2:50 PM

1. For this part, we have wired the 8 keypad outputs to a breadboard so that they pass through a 510 Ω resistor pack. This will ensure that the board pins will be protected from any current that may unexpectedly flow to them if we set up the ports incorrectly. From here we have split the purple wires

Grant Abella
Brandon Laupot
9/25/18

Laboratory #4: External Interrupt Controlled Keypad and LEDs

from the keypad (bits 4-7) so that they go to INT0-INT3. All 8 wires from the keypad are then wired to PORTC. Additionally we have LED pins L0-L7 hooked up to PORTA.

PORTB has been wired to the breadboard and will be used for debugging and wiggling.

A high-level flowchart for the program specified in this lab in which the keypad an four external interrupts are used to display hex values for any button pressed on the keypad onto the LEDs is shown below:

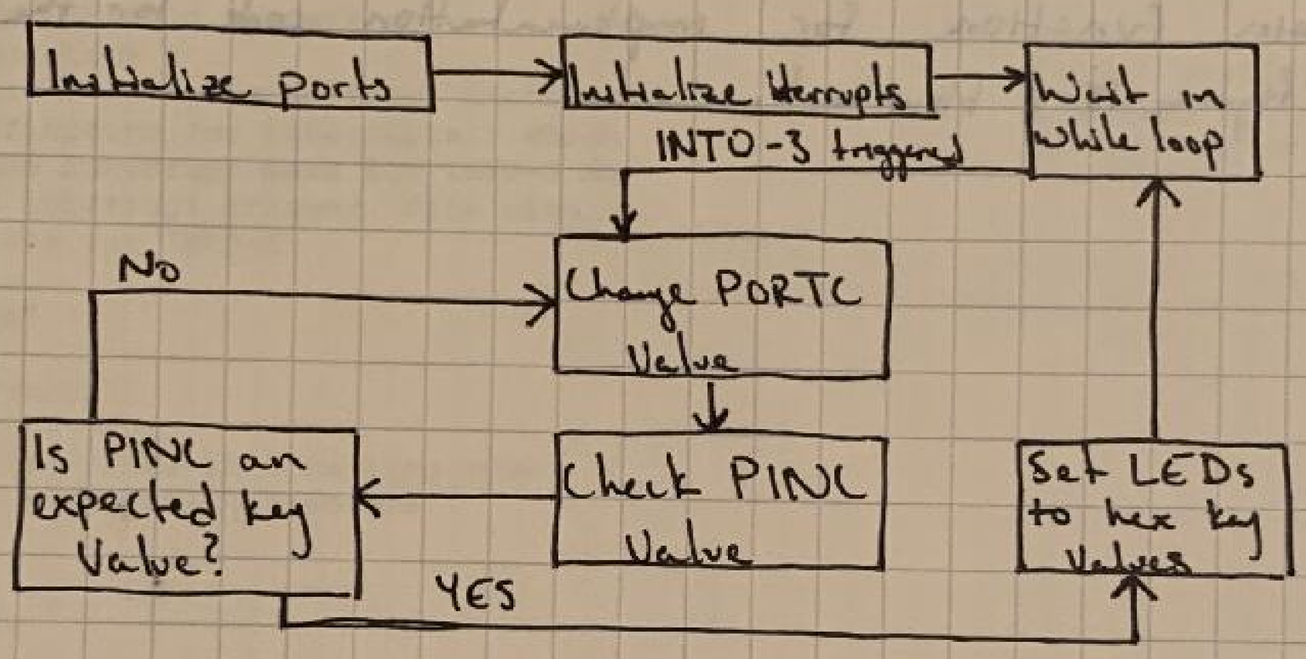


Figure 15-1: High-level flowchart for board logic to display hex value of keys pressed on an interrupt-controlled keypad.

After initializing both the ports on the Atmel board and the four interrupts to be used for this implementation, special consideration will need to be taken in order to successfully read user input via the keypad.

As shown in figure 14-2 on the previous page, interrupts 0-3 will be tied to the rows of the keypad. Because of this, we will know the row belonging to the key that is pressed, but not the column. To find out which column the pressed key is in, we will need to first change the value of the PORTC pullup resistors. By doing this, we can test the PINC input value against each of the four expected values for the full key pad box.

9/25/18
4:00 PM

The code used in this implementation is shown on the next page along with an explanation of the code and discussion of the difficulties. 4:47 pm

Laboratory #4: External Interrupt Controlled Keypad and LEDs

Grant Abella
Brandon Lampert
9/25/18

```
/*  Grant Abella, Brandon Lampert
 *  ECE 322 Laboratory 4
 *  main.c  :   main function calls
 *  functions to initialize ports &
 *  interrupts, sits in while loop.
 */

#include "lab4.h"

int main(void)
{
    ports();
    interupts();
     delay_ms(200); //stabalization
    sei();
    while (1)
    {
    }
}
```

Figure 16-1: Main function for implementation code for the logic shown in figure 15-1 flowchart.

```
/*  Grant Abella, Brandon Lampert
 *  ECE 322 Laboratory 4
 *  lab4.h  :   header file, contains function
 *  prototypes defined in ports.c and interrupts.c
 */

#ifndef LAB4_H_
#define LAB4_H_

/*************************************************
            Header Files
*************************************************/
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>

/*************************************************
            Definitions
*************************************************/
#define F_CPU 16000000UL //set board clock to 16MHz
#define LEDs PORTA
#define WiggleLED PORTB

/*************************************************
            Function ProtoTypes
*************************************************/
void ports(void);
void interupts(void);

#endif
```

Figure 16-2: Lab4.h header file used in implementation of the logic shown in the figure 15-1 flowchart.

Code figures continued on the next page.

9/25/18
4:09 PM

Laboratory #4: External Interrupt Controlled Keypad and LEDs

Grant Abella
Brandon Lampert
9/25/18

```
/*  Grant Abella, Brandon Lampert
 *  ECE 322 Laboratory 4
 *  ports.c :   program file, contains function
 *  definition for ports()  which initializes
 *  various board ports and sets their data
 *  directions.
 */
#include "lab4.h"

void ports(void)
{
    DDRA=0xFF; //set PORTA as an output
    DDRB=0xFF; //set PORTB as an output
    DDRC=0x0F; //set PORTC upper as an input
    DDRD=0x00; //set PORTD as an input
    PORTA =0x00; //PORTA off (LEDs)
    PORTB =0x00; //PORTB off
    PORTC =0xF0; // activate PORTC upper and pullup resistors
    PORTD =0xFF; // activate PORTD and pullup resistors
}
```

```
/*  Grant Abella, Brandon Lampert
 *  ECE 322 Laboratory 4
 *  interrupts.c    :   program file, contains
 *  function definition for interrupts() which
 *  sets sets the interrupt mask for INT0-3 and
 *  the type of interrupt trigger. File also has
 *  interrupt ISRs for INT0-3.
 */
#include "lab4.h"

void interupts(void)
{
    EICRA = 0xAA; //int0-int3 falling edge
    EIMSK = 0x0F; //int0-int3 enable
}

ISR (INT0_vect)
{
    uint8_t row1[4] = {0x01, 0x02, 0x03, 0x0A};
    WiggleLED = 0x01;
    uint8_t temp = 0x00;
    for (uint8_t i = 0; i <= 3; i++)
    {
        PORTC = ~(0x01 <<i);
        _delay_us(10);
        temp = PINC;
        if (temp == 0xEF-(0x01<<i))
        {
            LEDs = row1[i];
            i=4; //exits for loop
        }
    }
    PORTC = 0xF0;
    WiggleLED = 0x00;
}

ISR (INT1_vect)
{
    uint8_t row2[4] = {0x04, 0x05, 0x06, 0x0B};
    WiggleLED = 0x01;
    uint8_t temp = 0x00;
    for (uint8_t i = 0; i <= 3; i++)
    {
        PORTC = ~(0x01 <<i);
        _delay_us(10);
        temp = PINC;
        if (temp == 0xDF-(0x01<<i))
        {
            LEDs = row2[i];
            i=4; //exits for loop
        }
    }
    PORTC = 0xF0;
    WiggleLED = 0x00;
}

ISR (INT2_vect)
{
    uint8_t row3[4] = {0x07, 0x08, 0x09, 0x0C};
    WiggleLED = 0x01;
    uint8_t temp = 0x00;
    for (uint8_t i = 0; i <= 3; i++)
    {
        PORTC = ~(0x01 <<i);
        _delay_us(10);
        temp = PINC;
        if (temp == 0xBF-(0x01<<i))
        {
            LEDs = row3[i];
            i=4; //exits for loop
        }
    }
    PORTC = 0xF0;
    WiggleLED = 0x00;

}

ISR (INT3_vect)
{
    uint8_t row4[4] = {0x0F, 0x00, 0x0E, 0x0D};
    WiggleLED = 0x01;
    uint8_t temp = 0x00;
    for (uint8_t i = 0; i <= 3; i++)
    {
        PORTC = ~(0x01 <<i);
        _delay_us(10);
        temp = PINC;
        if (temp == 0x7F-(0x01<<i))
        {
            LEDs = row4[i];
            i=4; //exits for loop
        }
    }
    PORTC = 0xF0;
    WiggleLED = 0x00;
}
```

Figure 17-1: ports.c program file used in implementation of the logic shown in the figure 15-1 flowchart. Interrupts.c program file code also included.

9/25/18
4:16 PM

The first file (main.c) contains the main program function which calls the port initialization function defined in ports.c. After this call is made, the interrupt initialization function, interrupts(), is called which sets the interrupt mask and the way in

Grant Abella
Brandon Lampert
9/25/18

Laboratory #4: External Interrupt Controlled Keypad and LEDs

which each interrupt is triggered. Finally, after a 200 ms wait (for stabilization purposes), the sei function is called to enable global interrupts, and then the program sits in an infinite while loop and waits for interrupts to be triggered.

Figure 16-2 shows our code for the io64 header file. This file simply contains prototypes for the ports() and interrupts() functions defined in ports.c and interrupts.c respectively. This file also contains definitions for the LEDs port (PORTA), and the wiggle pin (PORTB) that we used for debugging purposes.

Bothe the ports.c and interrupts.c program files are shown in figure 17-1. Ports.c contains the function definition for the ports() function which initializes ~~as~~ Ports A, B, C, and D. The interrupts.c file contains the function definition for the interrupts() function which sets the mask and trigger types for INT0-3. In addition to this, the file contains four ISRs, one for each interrupt used in this implementation. The logic for these functions will be explained below.

For clarity on how each of the ISRs work, a detailed explanation is needed.

First we begin by declaring an array for the led values associated with each of the buttons in row one of the keypad. We then toggle ON the wiggle bit for debugging purposes (more on this later), and declare a temp variable that we will use to store the value of PINC.

To illustrate the logic of the for loop, the first to iterations will be mapped out and explained:

When i=0: ① PORTC = ~(0x01 << i) = 1111 1110 (Tests if [1] is pressed)
② delay of 10 us for stabilization
③ temp = PINC ⟶ this captures the input value after the change of PORTC has been made.
④ check if temp = 0xEF - (0x01 << i) = 1110 1110
↳ if it is, set the LEDs to row1[i] = 0x01 and set i=4 (exiting the loop)
⑤ if not, continue with the loop.

AL
9/25/18
5:02 PM

Grant Abella
Brandon Lungh
9/25/18

Laboratory #4: External interrupt controlled Keypad and LEDs

when i=1: ① PORTC = ~(0x01 << i) = 11111101 (Tests if [2] is pressed)
② delay of 10 ms for stabilization
③ temp = PINC → This captures the input value after the change of PORTC has been made.
④ check if temp = 0xEF - (0x01 << i) = 11101101
↳ if it is, set the LEDs to row[i] = 0x02 and set i = 4 (exiting the loop)
⑤ if not, continue with the loop.

This process continues for the other two possible button presses in row one. After the loop completes, PORTF is reset back to its value of 0xF0 so that interrupts can continue to be triggered, and the wiggle bit is toggled OFF.

A similar process is performed for the other interrupts to check button presses in the other rows. The only differences being the values in each row's array and the value that is checked for temp / PINC after the PORTC value is changed.

With that, the process of receiving input from the keypad using four interrupts and outputting the hex value to LEDs is pretty much fully explained.

Several difficulties were encountered on the way to this successful implementation:

① The first major issue we had when implementing this design was that our program was not entering any of the ISRs. We were able to determine this by checking our wiggle signal on the oscilloscope and discovering that it was not being driven high.

After a quick look at our breadboard we discovered that our diodes were positioned backwards. We switched them to the correct orientation and this fixed the issue.

② Another problem we encountered was that even though the interrupts were being triggered, the keypad was not working as intended with our code.

dA
9/25/18
5:39 PM

FL
4:46 pm

Upon inspection of the program code, we found that

Grant Abella
Brandon Lanfert
9/27/18

Laboratory #4: External Interrupt Controlled Keypad and LEDs

instead of changing the PORTC value in some places within the ISRs, we were changing the data direction. This was a simple programming error and once we remedied this problem, the program worked as intended.

Overall we were very pleased with how well this program worked. Aside from a few issues previously described, the implementation process for this portion of the lab was smooth and we believe our code will translate well to the next portion with a single interrupt.

2. For this ~~part~~ portion of the lab, we have modified the circuit shown in figure 19-2 to channel the row signals of the keypad into a single signal to go to INT0. This time around, we made sure our diodes were orientated correctly. After making this adjustment, we had our circuit checked by an instructor and we were okay to continue.

As for the code for this portion of the lab, we simply eliminated the ISRs for interrupts 1-3, and moved their code into the INT0 ISR.

We did this and then implemented the program onto the board to test it. The program worked exactly as intended, which is what we expected. The program functioned exactly like the one from the previous part, but it was now more efficient as only one interrupt was being used. [redacted]

Our code for this portion is shown below and continues onto page 21. Another important thing to note is that the interrupt mask and trigger type has been altered now that a single interrupt is being used.

```
#include "lab4.h"

void interupts(void)
{
    EICRA = 0xAA; //int0 falling edge
    EIMSK = 0x0F; //int0 enable
}

ISR (INT0_vect)
{
    uint8_t row1[4] = {0x01, 0x02, 0x03, 0x0A};
    uint8_t row2[4] = {0x04, 0x05, 0x06, 0x0B};
    uint8_t row3[4] = {0x07, 0x08, 0x09, 0x0C};
    uint8_t row4[4] = {0x0F, 0x00, 0x0E, 0x0D};
    WiggleLED = 0x01;
    uint8_t temp = 0x00;
```

9/27/18
2:40PM

Grant Ahrika
Brandon Lupt

9/27/18

Laboratory #4: External Interrupt Controlled Keypad and LEDs

```
    for (uint8_t i = 0x00; i <= 0x03; i++)
    {
        PORTC = ~(0x01 <<i);
        _delay_us(10);
        temp = PINC;
        if (temp == 0xEF-(0x01<<i))
        {
            LEDs = row1[i];
            i=0x04; //exits for loop
        }
    }
    for (uint8_t i = 0x00; i <= 0x03; i++)
    {
        PORTC = ~(0x01 <<i);
        _delay_us(10);
        temp = PINC;
        if (temp == 0xDF-(0x01<<i))
        {
            LEDs = row2[i];
            i=0x04; //exits for loop
        }
    }
    for (uint8_t i = 0x00; i <= 0x03; i++)
    {
        PORTC = ~(0x01 <<i);
        _delay_us(10);
        temp = PINC;
        if (temp == 0xBF-(0x01<<i))
        {
            LEDs = row3[i];
            i=0x04; //exits for loop
        }
    }
    for (uint8_t i = 0; i <= 3; i++)
    {
        PORTC = ~(0x01 <<i);
        _delay_us(10);
        temp = PINC;
        if (temp == 0x7F-(0x01<<i))
        {
            LEDs = row4[i];
            i=4; //exits for loop
        }
    }
    PORTC = 0xF0;
    WiggleLED = 0x00;
}
```

Figure 21-1: Interrupts.C program file containg logic for single interrupt driven keypad and output to LEDs.

The other header and program files for this implementation remained unchanged, so they have been omitted as they can be seen in the figures on pages 16 and 17.

We believe this implementation is very efficient, and will allow us to easily implement the logic for part 3 of this lab.

9/27/18 4:20PM

The only major change that we considered making to this code was having our key LED values for rows one through four being stored in a 2D array instead of four 1D arrays. We opted to not do this in order to keep our program code simple.

FL 9/27/18 4:16 PM

I have forgotten to include the flowchart that we followed while

Grant Abella
Brandon Lampert
9/27/18

Laboratory #4: External Interrupt Controlled Keypad and LEDs

designing this implementation, so I will include it below:

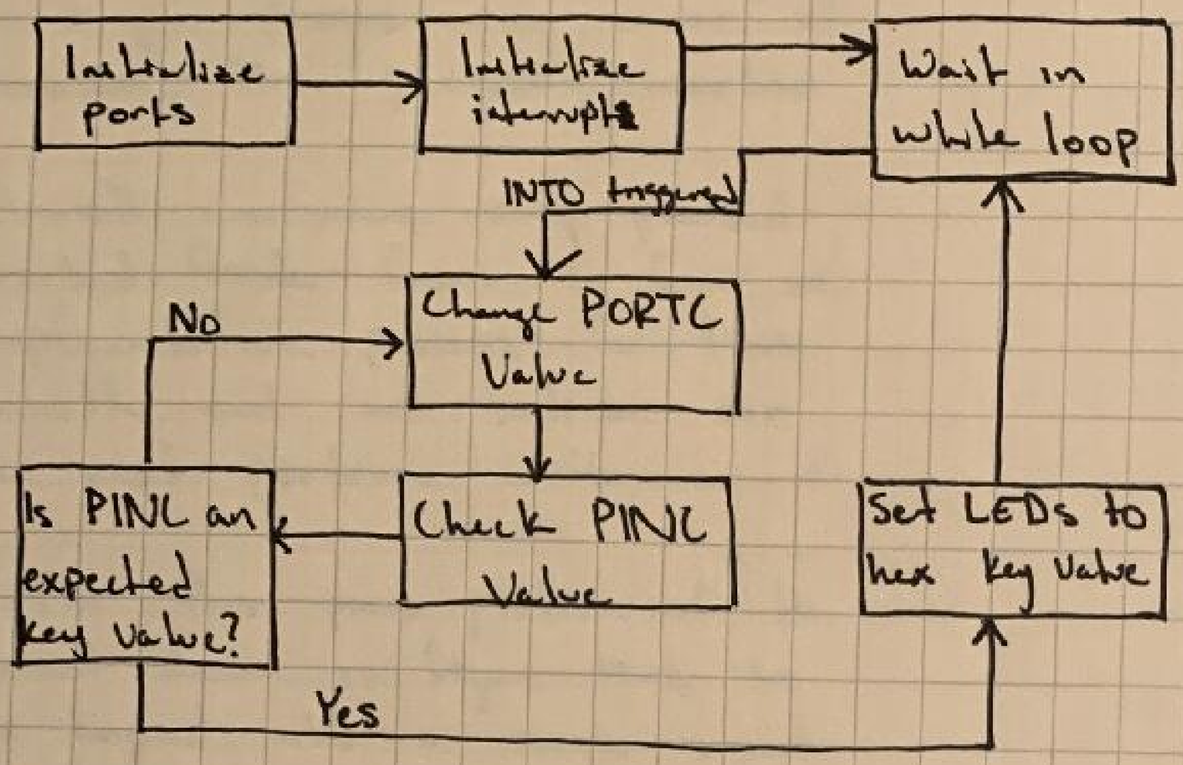


Figure 22-1: High level flowchart for logic of implementation and design of part 2 of this lab with one interrupt used.

This flowchart is almost identical to the chart for part one, only with one interrupt being used.

3. For the design of this part, we simply looked up an ASCII table of character values and replaced the values in our LED arrays with ASCII values.

The flowchart for this logic is shown below:

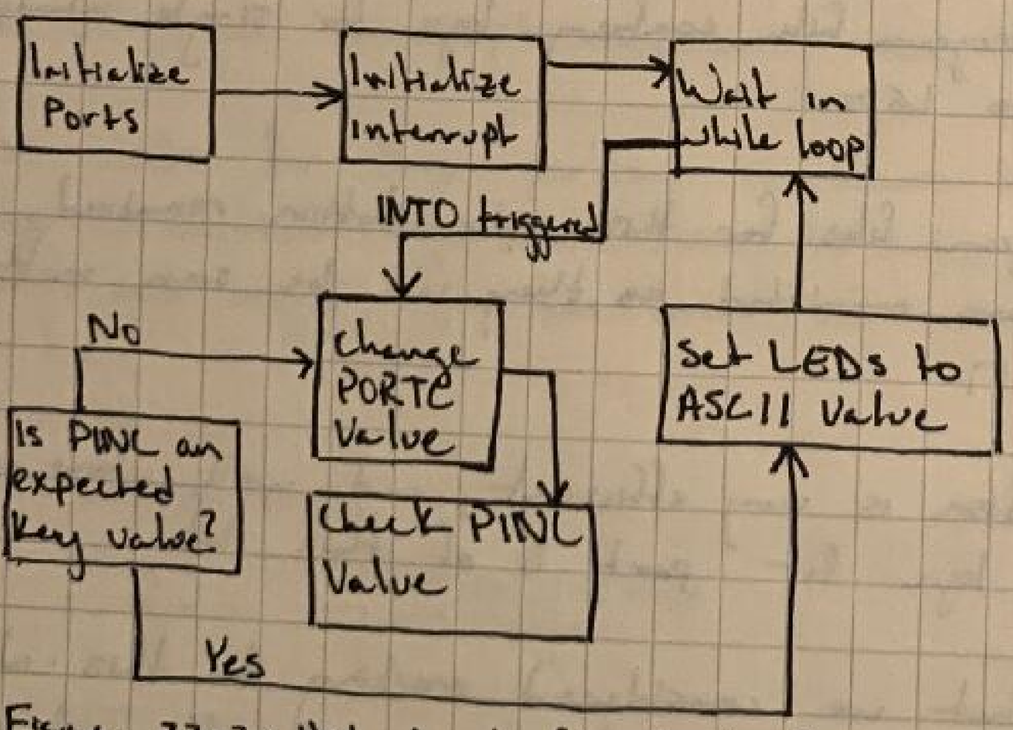


Figure 22-2: High level flowchart for implementation and design logic of part 3 of this lab.

SA
9/27/18
4:52PM

As you can see, the only modification of this flowchart from the previous is the change of hex values to ASCII values. The code

Grant Abelle
Brandon Laight
10/4/18

Laboratory #4: External Interrupt Controlled Keypad and LEDs

for this part containing the altered key values in the row arrays is shown below:

```
#include "lab4.h"

void interupts(void)
{
    EICRA = 0xAA; //int0 falling edge
    EIMSK = 0x0F; //int0 enable
}

ISR (INT0_vect)
{
    uint8_t row1[4] = {0x31, 0x32, 0x33, 0x41};
    uint8_t row2[4] = {0x34, 0x35, 0x36, 0x42};
    uint8_t row3[4] = {0x37, 0x38, 0x39, 0x43};
    uint8_t row4[4] = {0x2A, 0x30, 0x23, 0x44};
    WiggleLED = 0x01;
    uint8_t temp = 0x00;
    for (uint8_t i = 0x00; i <= 0x03; i++)
    {
        PORTC = ~(0x01 <<i);
        _delay_us(10);
        temp = PINC;
        if (temp == 0xEF-(0x01<<i))
        {
            LEDs = row1[i];
            i=0x04; //exits for loop
        }
    }
    for (uint8_t i = 0x00; i <= 0x03; i++)
    {
        PORTC = ~(0x01 <<i);
        _delay_us(10);
        temp = PINC;
        if (temp == 0xDF-(0x01<<i))
        {
            LEDs = row2[i];
            i=0x04; //exits for loop
        }
    }
    for (uint8_t i = 0x00; i <= 0x03; i++)
    {
        PORTC = ~(0x01 <<i);
        _delay_us(10);
        temp = PINC;
        if (temp == 0xBF-(0x01<<i))
        {
            LEDs = row3[i];
            i=0x04; //exits for loop
        }
    }
    for (uint8_t i = 0; i <= 3; i++)
    {
        PORTC = ~(0x01 <<i);
        _delay_us(10);
        temp = PINC;
        if (temp == 0x7F-(0x01<<i))
        {
            LEDs = row4[i];
            i=4; //exits for loop
        }
    }
    PORTC = 0xF0;
    WiggleLED = 0x00;
}
```

Figure 23-1: Interrupt.c file for implementation of logic shown in figure 22-2 flowchart.

Just as before, the other files for this project have been omitted as they are identical to the code shown on pages 16 and 17.

When we implemented this code, the program functioned as expected, the same as in part two, only with ASCII values being displayed instead of hex.

Postlab:

GA
10/4/18
2:18PM

1. Our approach to this lab was simple: we first made sure we had a full understanding of how the keypad would be interfaced with the board. This was critical because before we could write any code for this lab, we had to understand how we would successfully recieve input from the keypad. After this, we simply drafted up flowcharts and wrote

Grant Abelle
Brandon Lapert
10/4/18

Laboratory #4: External Interrupt Controlled Keypad and LEDs

our code based on those figures. Some debugging and testing was required after this, but we utilized our experiences in previous labs to do this.

In the future, we will definitely use the single-interrupt method for keypad decoding since it was easy and more efficient than using multiple interrupts.

2. The biggest difficulty we had was successfully decoding the keypad input. We had initial issues do doe our diode configuration and then with our ISRs, but our solution to these * was discussed on page 19.

The biggest help we had for debugging was the implementation of a wiggle bit. With this we were able to find out if our interrupts were even being triggered and which portions of code were being reached.

3. We used the standard technique of using a 510Ω resistor pack when wiring board inputs. We used 2 packs during this lab, one for inputs to the interrupts, and another for inputs into PORTC. These were both important because with the switching of the PORTC value, it is possible to create a circuit with a high voltage that could damage the board.

Conclusion:

+2 PTS

In this lab, we gained valuable knowledge on how to interface with a keypad. This was valuable because keypads are a unique device with regaurds to how input is read. Also, with keypads being such a common interfacing device, we will surely encounter them in the future.

10/4/18
4:25PM

In addition to this, we gained more experience with using a single interrupt to drive triggers for multiple inputs.

Diagram / Flow charts : -0 11/11
Discussion : -0 12/12
Code : -0 12/12
Post lab : -0 5
Format : -0

+2 Great Job!!

42/40

Mean Sir