HW9 Report

107061112 王昊文

Design Concept:

One of the key point is that we have to come up with a method to detect whether the key pad is pressed or not. We can think of the system using the concept of finite state machine (FSM). There are totally nine states in the system, corresponding to nine different keypad number, it should jump to next state once the ScanKey() function detects a change. However, we have to implement the concept using C programming, so I use the variable “number” to detect the change, and variable “key_pressed” to “maintain” the state, since when no key pressed, ScanKey() is zero, and when no key pressed means that there is no need to change the state, only when an input change detected, we have to determine the state.

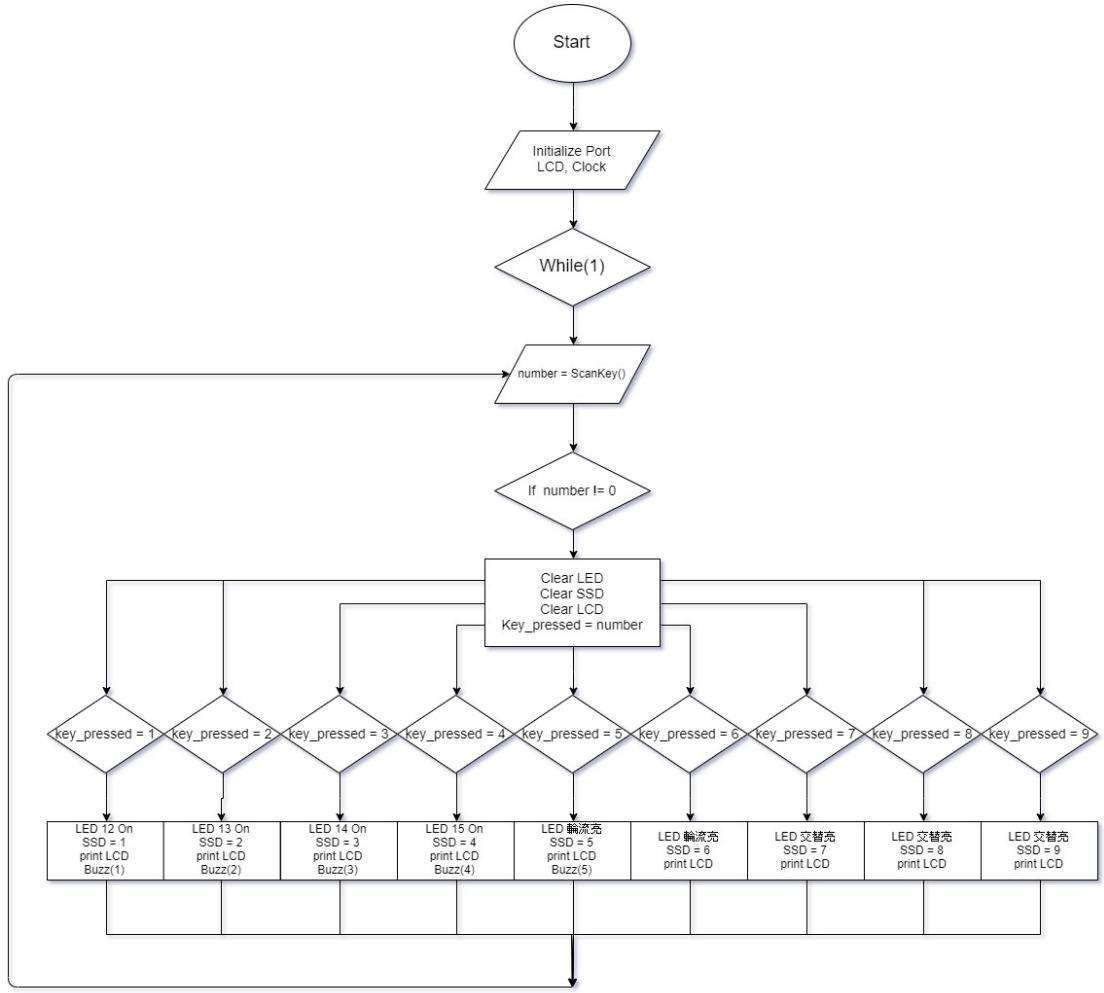
The other key point is that, since we are programming a hardware using a software, how are we able to detect while implement, for example, how are we able to switch state when the buzzer is buzzing, or the LED is still finishing its “lighting”?

Well, the answer is pretty simple. The board is using a pretty fast clock, so all the instructions will be finished pretty fast, so we don’t have to consider the problems with performing single instructions or single line in the C programming. The main problem comes with “Delay”. If we use the delay command “DrvSYS_Delay”, then during the delay we are not able to detect any change in the input. Hence, we have to change the way we delay. **Using a for loop to check for input change** after every 1us of delay. However, it has to take time for every detection and the for loop, the delay time for performing the “DrvSYS_Delay” will take much longer than 1us, so for example if we want to take an 1ms delay, we shouldn’t use the for loop for 1000 times, since it will delay more than 1ms. Also by using this method, we are not able to calculate the actual delay time.

Giving an overlook of my program:

If you look at my program, the first part is to initialize the hardware. Then, under the while loop, we begin to scan key. Note that, when “number != 0”, it means that a key is being pressed. As I mentioned before, only when input changes we will modify “key_pressed”. The remaining part of the program is to jump to the corresponding state according to the variable “key_pressed”.

Flow Chart:



My code:

```
//
// Smpl_GPIO_Keypad : scan keypad 3x3 (1~9) to control GPB0~8
//
// 4-port Relay
// VCC : to 3.3V
// IN1 : to GPB0 (press 1 will output 3.3V, else output 0V)
// IN2 : to GPB1 (press 2 will output 3.3V, else output 0V)
// IN3 : to GPB2 (press 3 will output 3.3V, else output 0V)
// IN4 : to GPB3 (press 4 will output 3.3V, else output 0V)
// GND : to GND

#include "stdio.h"

#include "NUC1xx.h"
#include "GPIO.h"
```

```
#include "SYS.h"
#include "Seven_Segment.h"
#include "Scankey.h"
#include "LCD.h"

void Init_LED();                        // Initialize the LED
void Buzz(int number);                  // buzz fuction
/*---------------------------------------------------------------------------
   MAIN function
   ---------------------------------------------------------------------------*/
int32_t main (void)
{
      int delay_cnt;                            // counting for delay time
      int16_t i = 12;                           // counting LED
      int8_t number, key_pressed = 0;           // for getting the data from keypad
      char TEXT[14];                                  // LCD displaying

      UNLOCKREG();
      DrvSYS_Open(50000000);              // set System Clock to run at 48MHz,
12MHz crystal input, PLL output 48MHz
      SYSCLK->PWRCON.XTL12M_EN = 1;     //Enable 12Mhz and set HCLK-
>12Mhz
      SYSCLK->CLKSEL0.HCLK_S = 0;
      LOCKREG();

      Init_LED();
      init_LCD();
      clear_LCD();
      OpenKeyPad();
      DrvGPIO_Open(E_GPB, 11, E_IO_OUTPUT); // initial GPIO pin GPB11 for
controlling Buzzer

      while(1) {
            number = ScanKey();
            // keep cheching if input changed
            if (number != 0) {
                  i = 12;                           // refresh LED count
                  DrvGPIO_SetBit(E_GPC, 12);        // output Hi to turn off LED
```

```
                DrvGPIO_SetBit(E_GPC, 13);          // GPC13 pin output Hi to turn
off LED
                DrvGPIO_SetBit(E_GPC, 14);          // GPC14 pin output Hi to turn
off LED
                DrvGPIO_SetBit(E_GPC, 15);          // GPC15 pin output Hi to turn
off LED
                close_seven_segment();              // turn off ssd
                clear_LCD();                        // clear LCD
                key_pressed = number;               // refresh number
            }
            // check what is the key_pressed
            if(key_pressed == 1) {
                DrvGPIO_ClrBit(E_GPC, 12);      // output Low to turn on LED
                show_seven_segment(0, key_pressed);             // show ssd
                sprintf(TEXT, "Key%d pressed", key_pressed);
                print_Line(0, TEXT);                // print LCD
                Buzz(1);
                for (delay_cnt = 500; (delay_cnt >= 0) && (ScanKey() == 0);
delay_cnt--) {
                    DrvSYS_Delay(1);
                }                                   // Buzz, then delay for next buzz
            }
            else if(key_pressed == 2) {
                DrvGPIO_ClrBit(E_GPC, 13);
                show_seven_segment(0, key_pressed);
                sprintf(TEXT, "Key%d pressed", key_pressed);
                print_Line(0, TEXT);
                Buzz(2);
                for (delay_cnt = 500; (delay_cnt >= 0) && (ScanKey() == 0);
delay_cnt--) {
                    DrvSYS_Delay(1);
                }                                   // Buzz, then delay for next buzz
            }
            else if(key_pressed == 3) {
                DrvGPIO_ClrBit(E_GPC, 14);
                show_seven_segment(0, key_pressed);
                sprintf(TEXT, "Key%d pressed", key_pressed);
                print_Line(0, TEXT);
```

```
                Buzz(3);
                for (delay_cnt = 500; (delay_cnt >= 0) && (ScanKey() == 0);
delay_cnt--) {
                        DrvSYS_Delay(1);
                }                                       // Buzz, then delay for next buzz
        }
        else if(key_pressed == 4) {
                DrvGPIO_ClrBit(E_GPC, 15);
                show_seven_segment(0, key_pressed);
                sprintf(TEXT, "Key%d pressed", key_pressed);
                print_Line(0, TEXT);
                Buzz(4);
                for (delay_cnt = 500; (delay_cnt >= 0) && (ScanKey() == 0);
delay_cnt--) {
                        DrvSYS_Delay(1);
                }                                       // Buzz, then delay for next buzz
        }
        else if(key_pressed == 5) {
                show_seven_segment(0, key_pressed);
                sprintf(TEXT, "Key%d pressed", key_pressed);
                print_Line(0, TEXT);
                DrvGPIO_ClrBit(E_GPC, i);               // output Low to turn on LED
                for (delay_cnt = 500; (delay_cnt >= 0) && (ScanKey() == 0);
delay_cnt--) {
                        DrvSYS_Delay(1);
                }
                (i == 15) ? (i = 12) : (i++);           // light up in one at all time
                Buzz(5);
                for (delay_cnt = 500; (delay_cnt >= 0) && (ScanKey() == 0);
delay_cnt--) {
                        DrvSYS_Delay(1);
                }                                       // Buzz, then delay for next buzz
        }
        else if(key_pressed == 6) {
                show_seven_segment(0, key_pressed);
                sprintf(TEXT, "Key%d pressed", key_pressed);
                print_Line(0, TEXT);
                print_Line(1, "107061112");
```

```
                DrvGPIO_ClrBit(E_GPC, 12); // output Low to turn on LED
                for (delay_cnt = 500; (delay_cnt >= 0) && (ScanKey() == 0);
delay_cnt--) {
                        DrvSYS_Delay(1);
                }
                DrvGPIO_ClrBit(E_GPC, 13); // output Low to turn on LED
                for (delay_cnt = 500; (delay_cnt >= 0) && (ScanKey() == 0);
delay_cnt--) {
                        DrvSYS_Delay(1);
                }
                DrvGPIO_ClrBit(E_GPC, 14); // output Low to turn on LED
                for (delay_cnt = 500; (delay_cnt >= 0) && (ScanKey() == 0);
delay_cnt--) {
                        DrvSYS_Delay(1);
                }
                DrvGPIO_ClrBit(E_GPC, 15); // output Low to turn on LED
                for (delay_cnt = 500; (delay_cnt >= 0) && (ScanKey() == 0);
delay_cnt--) {
                        DrvSYS_Delay(1);
                }
                // all lightened up, now turn off
                DrvGPIO_SetBit(E_GPC, 12);          // output Hi to turn off LED
                DrvGPIO_SetBit(E_GPC, 13);          // GPC13 pin output Hi to turn
off LED
                DrvGPIO_SetBit(E_GPC, 14);          // GPC14 pin output Hi to turn
off LED
                DrvGPIO_SetBit(E_GPC, 15);          // GPC15 pin output Hi to turn
off LED
                DrvGPIO_ClrBit(E_GPC, 15); // output Low to turn on LED
                for (delay_cnt = 500; (delay_cnt >= 0) && (ScanKey() == 0);
delay_cnt--) {
                        DrvSYS_Delay(1);
                }
                DrvGPIO_ClrBit(E_GPC, 14); // output Low to turn on LED
                for (delay_cnt = 500; (delay_cnt >= 0) && (ScanKey() == 0);
delay_cnt--) {
                        DrvSYS_Delay(1);
                }
```

```
                DrvGPIO_ClrBit(E_GPC, 13); // output Low to turn on LED
                for (delay_cnt = 500; (delay_cnt >= 0) && (ScanKey() == 0);
delay_cnt--) {
                    DrvSYS_Delay(1);
                }
                DrvGPIO_ClrBit(E_GPC, 12); // output Low to turn on LED
                for (delay_cnt = 500; (delay_cnt >= 0) && (ScanKey() == 0);
delay_cnt--) {
                    DrvSYS_Delay(1);
                }
                // all lightened up, now turn off
                DrvGPIO_SetBit(E_GPC, 12);          // output Hi to turn off LED
                DrvGPIO_SetBit(E_GPC, 13);          // GPC13 pin output Hi to turn
off LED
                DrvGPIO_SetBit(E_GPC, 14);          // GPC14 pin output Hi to turn
off LED
                DrvGPIO_SetBit(E_GPC, 15);          // GPC15 pin output Hi to turn
off LED
            }
            else if(key_pressed == 7) {
                show_seven_segment(0, key_pressed);
                sprintf(TEXT, "Key%d pressed", key_pressed);
                print_Line(0, TEXT);
                print_Line(1, "Howard");
                DrvGPIO_ClrBit(E_GPC, i);           // output Low to turn on LED
                DrvGPIO_ClrBit(E_GPC, i + 1);   // output Low to turn on LED,
alternating
                for (delay_cnt = 500; (delay_cnt >= 0) && (ScanKey() == 0);
delay_cnt--) {
                    DrvSYS_Delay(1);
                }
                DrvGPIO_SetBit(E_GPC, i);           // output Low to turn off LED
                DrvGPIO_SetBit(E_GPC, i + 1);   // output Low to turn off LED,
alternating
                for (delay_cnt = 500; (delay_cnt >= 0) && (ScanKey() == 0);
delay_cnt--) {
                    DrvSYS_Delay(1);
                }
```

```
            (i == 14) ? (i = 12) : (i += 2);         // control the LED count
        }
        else if(key_pressed == 8) {
            show_seven_segment(0, key_pressed);
            print_Line(0, "107061112");
            DrvGPIO_ClrBit(E_GPC, i);               // output Low to turn on LED
            DrvGPIO_ClrBit(E_GPC, i + 2);   // output Low to turn on LED,
alternating
            for (delay_cnt = 500; (delay_cnt >= 0) && (ScanKey() == 0);
delay_cnt--) {
                DrvSYS_Delay(1);
            }
            DrvGPIO_SetBit(E_GPC, i);               // output Low to turn off LED
            DrvGPIO_SetBit(E_GPC, i + 2);   // output Low to turn off LED,
alternating
            for (delay_cnt = 500; (delay_cnt >= 0) && (ScanKey() == 0);
delay_cnt--) {
                DrvSYS_Delay(1);
            }
            (i == 13) ? (i = 12) : (i++);            // control the LED count
        }
        else if(key_pressed == 9) {
            show_seven_segment(0, key_pressed);
            sprintf(TEXT, "Key%d pressed", key_pressed);
            print_Line(0, TEXT);
            print_Line(1, "Happy New Year");
            DrvGPIO_ClrBit(E_GPC, 12); // output Low to turn on LED
            DrvGPIO_ClrBit(E_GPC, 13); // output Low to turn on LED
            DrvGPIO_ClrBit(E_GPC, 14); // output Low to turn on LED
            DrvGPIO_ClrBit(E_GPC, 15); // output Low to turn on LED
            for (delay_cnt = 500; (delay_cnt >= 0) && (ScanKey() == 0);
delay_cnt--) {
                DrvSYS_Delay(1);
            }
            DrvGPIO_SetBit(E_GPC, 12); // output Hi to turn off LED
            DrvGPIO_SetBit(E_GPC, 13); // output Hi to turn off LED
            DrvGPIO_SetBit(E_GPC, 14); // output Hi to turn off LED
            DrvGPIO_SetBit(E_GPC, 15); // output Hi to turn off LED
```

```
                for (delay_cnt = 500; (delay_cnt >= 0) && (ScanKey() == 0);
delay_cnt--) {
                    DrvSYS_Delay(1);
                }
            }
        }
}
/*****************************************************************************
********
 * Initialize the LED
 *
*****************************************************************************
****/
void Init_LED() {
    // initialize GPIO pins
    DrvGPIO_Open(E_GPC, 12, E_IO_OUTPUT); // GPC12 pin set to output mode
    DrvGPIO_Open(E_GPC, 13, E_IO_OUTPUT); // GPC13 pin set to output mode
    DrvGPIO_Open(E_GPC, 14, E_IO_OUTPUT); // GPC14 pin set to output mode
    DrvGPIO_Open(E_GPC, 15, E_IO_OUTPUT); // GPC15 pin set to output mode
    // set GPIO pins to output Low
    DrvGPIO_SetBit(E_GPC, 12); // GPC12 pin output Hi to turn off LED
    DrvGPIO_SetBit(E_GPC, 13); // GPC13 pin output Hi to turn off LED
    DrvGPIO_SetBit(E_GPC, 14); // GPC14 pin output Hi to turn off LED
    DrvGPIO_SetBit(E_GPC, 15); // GPC15 pin output Hi to turn off LED
}
// Function: input the number of buzz
void Buzz(int number)
{
    int i, delay_cnt;                   // loop index and count for delay
    for (i=0; i<number; i++) {
        DrvGPIO_ClrBit(E_GPB,11); // GPB11 = 0 to turn on Buzzer
        for (delay_cnt = 100; (delay_cnt >= 0) && (ScanKey() == 0); delay_cnt--)
{
            DrvSYS_Delay(1);
        }
        DrvGPIO_SetBit(E_GPB,11); // GPB11 = 1 to turn off Buzzer
        for (delay_cnt = 100; (delay_cnt >= 0) && (ScanKey() == 0); delay_cnt--)
{
```

```
            DrvSYS_Delay(1);
        }
    }
}
```

Discussion:

這個作業整體而言算是簡單的，但剛開始上手的時候猶豫了很久，由於上學期學了硬體語言 Verilog，突然要使用 c 來完成這些任務突然之間還真的有點卡住。最棘手的，就是卡在一個 state 跳不出來，因為掃不到 input。一旦這個問題解決之後一切迎刃而解。要注意的地方除了掃 input 之外，其他小問題如 LCD 的字不能太長，或是忘記關閉七段顯示器導致功能異常，應該都不是什麼太大的問題。不過這次實驗之後，令我驚艷的地方在於 code compile 的速度，遠遠大於 Verilog 啊啊啊，不過當然 Verilog 也有其不可取代的地方，這讓我們在設計硬體的時候可以根據不同的需求去選擇。

Question:

I. From the example codes, some GPIO driver functions are provided in the GPIO.c and GPIO.h, e.g., DrvGPIO_Open(E_DRVGPIO_PORT port, int32_t i32Bit, E_DRVGPIO_IO mode),…, etc. These functions allow you to set or clear one GPIO port bit at a time. Implement the following functions so that they can set, get, or clear the whole 16 bits of a port at a time.

(1) int32_t DrvGPIO_Opens(port,E_DRVGPIO_IO mode);

(2) int32_t DrvGPIO_SetBits(port, int32_t i32Bit);

(3) int32_t DrvGPIO_GetBits(port);

(4) int32_t DrvGPIO_ClearBits(port, int32_t i32Bit);

I observe that some of the parameter send is redundant, since we are clearing the whole port, there is no need to send in the i32Bit parameter. I will modify the functions according to the example code.

(1)

```
int32_t DrvGPIO_Open(E_DRVGPIO_PORT port, E_DRVGPIO_IO mode)
{
    volatile uint32_t u32Reg;
    volatile uint32_t i32Bit;

    for (i32Bit = 0; i32Bit < 16; i32Bit++) {
        u32Reg = (uint32_t)&GPIOA->PMD + (port*PORT_OFFSET);
        if ((mode == E_IO_INPUT) || (mode == E_IO_OUTPUT) || (mode == E_IO_OPENDRAIN)) {
            outpw(u32Reg, inpw(u32Reg) & ~(0x3<<(i32Bit*2)));
```

```
            if (mode == E_IO_OUTPUT) {
            outpw(u32Reg, inpw(u32Reg) | (0x1<<(i32Bit*2)));
            }
            else if (mode == E_IO_OPENDRAIN) {
                outpw(u32Reg, inpw(u32Reg) | (0x2<<(i32Bit*2)));
            }
            else if (mode == E_IO_QUASI) {
                outpw(u32Reg, inpw(u32Reg) | (0x3<<(i32Bit*2)));
            }
            else
                return E_DRVGPIO_ARGUMENT;
    return E_SUCCESS;
}
```

(2)

```
int32_t DrvGPIO_SetBit(E_DRVGPIO_PORT port)
{
    volatile uint32_t i32Bit;
    GPIO_T * tGPIO;
    tGPIO = (GPIO_T *)((uint32_t)GPIOA + (port*PORT_OFFSET));
    for (i32Bit = 0; i32Bit < 16; i32Bit++) {
        tGPIO->DOUT |= (1 << i32Bit);
    }
    return E_SUCCESS;
}
```

(3)

```
#define PORT_OFFSET   0x40
int32_t DrvGPIO_GetBit(E_DRVGPIO_PORT port)
{
    volatile uint32_t u32Reg;
    volatile uint32_t i32Bit;

    u32Reg = (uint32_t)&GPIOA->PIN + (port*PORT_OFFSET);
    return inpw(u32Reg);
}
```

(4)

```c
#define PORT_OFFSET    0x40
int32_t DrvGPIO_ClrBit(E_DRVGPIO_PORT port, int32_t i32Bit)
{
    volatile uint32_t i32Bit;
    GPIO_T * tGPIO;
    tGPIO = (GPIO_T *)((uint32_t)GPIOA + (port*PORT_OFFSET));
    for (i32Bit = 0; i32Bit < 16; i32Bit++) {
        tGPIO->DOUT &= ~(1 << i32Bit);
    }
    return E_SUCCESS;
}
```

II. The LCD on the NUC140 experiment board is controlled by NUC140 via SPI interface. Please surf the web and write a short essay on SPI, including its theory, timing, functions, …, etc.

The Serial Peripheral Interface (SPI) is a synchronous serial communication interface used f0or short-distance communication, primarily in embedded systems.

As for the operating theory, it communicates in full duplex mode using a master-slave architecture, with only a single master and multiple slave devices. The word “duplex” indicates that even when only one-directional data transfer is intended, both the slave and the master will send and read. The master device originates the frame for reading and writing. Multiple slave-devices are supported through selection “slave select(SS)”, sometimes called “chip select(CS)”.

The SPI bus specifies four logic signals.

Pin names and its functionality

Pin	Full Name	functionality
MOSI	Master Out Slave in	Master Output Slave Input (data output from master)
MISO	Master in Slave Out	Master Input Slave Output(data output from slave)
SCLK	Serial Clock	Clock signal generated from Master

SS	Slave Select	Often active low, output from master

As for the timing of data transmission, the master first configures the clock. The master then selects the slave device with a logic 0 on the select line. If a waiting period is required, the master must wait for at least that period of time before issuing clock cycles.
Since the SPI is operating under full-duplex data transmission, the master and slave are both transmitting data, it normally involves two shift register of some given word-size, one in the slave and one in the master. They are connected in a virtual ring topology. On the clock edge, both master and slave shift out a bit and output it on the transmission line to the counterpart. On the next clock edge, at each receiver the bit is sampled from the transmission line and set as a new least-significant bit of the register. Transmission may continue for any number of clock cycles, depending on the data size. When complete, the master stops toggling the clock signal, and typically deselects the slave.

When we are discussing the functions while programming an embedded system using C, there are something we have to consider. Note that the function name may vary depending on the embedded system we are using.

1. MSB first or LSB first? this is controlled by the setBitOrder() function.
2. Send data on positive edge or negative edge? These options can be controlled by the function setDataMode().
3. Clock frequency generated by the master? Use the function setClockDivider().
4. Other functions including setting pins, deactivate SS……

Other detailed functional description are included in the block diagram

Figure 270. SPI block diagram