ECE 2049 A Term 2022

Sakshi Gauro
sgauro@wpi.edu

Table 9-2. Memory Organization (1)

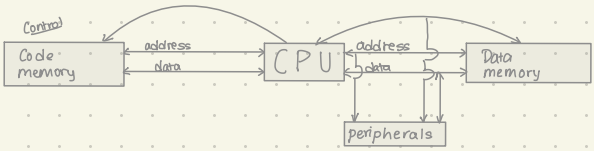
		MSP430F5522 MSP430F5521 MSP430F5513	MSP430F5525 MSP430F5524 MSP430F5515 MSP430F5514	MSP430F5527 MSP430F5526 MSP430F5517	MSP430F5529 MSP430F5528 MSP430F5519
Memory (flash) Main: interrupt vector	Total Size	32KB 00FFFFh to 00FF80h	64KB 00FFFFh to 00FF80h	96KB 00FFFFh to 00FF80h	128KB 00FFFFh to 00FF80h
Main: code memory	Bank D	N/A	N/A	N/A	32KB 0243FFh to 01C400h
	Bank C	N/A	N/A	32KB 01C3FFh to 014400h	32KB 01C3FFh to 014400h
	Bank B	15KB 00FFFFh to 00C400h	32KB 0143FFh to 00C400h	32KB 0143FFh to 00C400h	32KB 0143FFh to 00C400h
	Bank A	17KB 00C3FFh to 008000h	32KB 00C3FFh to 004400h	32KB 00C3FFh to 004400h	32KB 00C3FFh to 004400h
RAM	Sector 3	2KB(2) 0043FFh to 003C00h	N/A	N/A	2KB 0043FFh to 003C00h
	Sector 2	2KB(3) 003BFFh to 003400h	N/A	2KB 003BFFh to 003400h	2KB 003BFFh to 003400h
	Sector 1	2KB 0033FFh to 002C00h	2KB 0033FFh to 002C00h	2KB 0033FFh to 002C00h	2KB 0033FFh to 002C00h
	Sector 0	2KB 002BFFh to 002400h	2KB 002BFFh to 002400h	2KB 002BFFh to 002400h	2KB 002BFFh to 002400h
USB RAM(4)	Sector 7	2KB 0023FFh to 001C00h	2KB 0023FFh to 001C00h	2KB 0023FFh to 001C00h	2KB 0023FFh to 001C00h
Information memory (flash)	Info A	128 bytes 0019FFh to 001980h	128 bytes 0019FFh to 001980h	128 bytes 0019FFh to 001980h	128 bytes 0019FFh to 001980h
	Info B	128 bytes 00197Fh to 001900h	128 bytes 00197Fh to 001900h	128 bytes 00197Fh to 001900h	128 bytes 00197Fh to 001900h
	Info C	128 bytes 0018FFh to 001880h	128 bytes 0018FFh to 001880h	128 bytes 0018FFh to 001880h	128 bytes 0018FFh to 001880h
	Info D	128 bytes 00187Fh to 001800h	128 bytes 00187Fh to 001800h	128 bytes 00187Fh to 001800h	128 bytes 00187Fh to 001800h
Bootloader (BSL) memory (flash)	BSL 3	512 bytes 0017FFh to 001600h	512 bytes 0017FFh to 001600h	512 bytes 0017FFh to 001600h	512 bytes 0017FFh to 001600h
	BSL 2	512 bytes 0015FFh to 001400h	512 bytes 0015FFh to 001400h	512 bytes 0015FFh to 001400h	512 bytes 0015FFh to 001400h
	BSL 1	512 bytes 0013FFh to 001200h	512 bytes 0013FFh to 001200h	512 bytes 0013FFh to 001200h	512 bytes 0013FFh to 001200h
	BSL 0	512 bytes 0011FFh to 001000h	512 bytes 0011FFh to 001000h	512 bytes 0011FFh to 001000h	512 bytes 0011FFh to 001000h
Peripherals	Size	4KB 000FFFh to 0h	4KB 000FFFh to 0h	4KB 000FFFh to 0h	4KB 000FFFh to 0h

(1) N/A = Not available
(2) MSP430F5522 only
(3) MSP430F5522 and MSP430F5521 only
(4) USB RAM can be used as general purpose RAM when not used for USB operation.

1) In lab we are using the MSP430F5529. Using the MSP430x5xx Datasheet and MSP430F5529 User's Guide (under Useful Links on the class website) and class notes, *fully* answer the following questions about the architecture and memory map of the MSP430F5529. (25 pts)

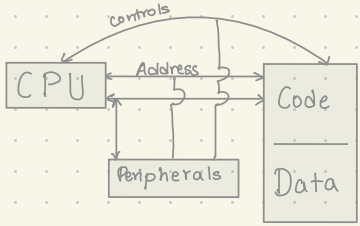
a) The MSP430 uses a Von Neumann architecture. What is the main difference between the Harvard architecture and the Von Neumann architecture? Sketch both architectures showing CPU, memory, peripherals and all buses. Name one strength and one weakness for each architecture.

a)



Harvard Architecture

pros: faster operation
cons: complicated



Von Neuman Architecture

pros: slower
cons: less complicated

Harvard Architecture separates memory address space for code & data making it faster. However, in Von Neumann Architechture, there's a single memory address space for code & data.

b) The MSP430F5529 memory space contains both RAM and FLASH. Why? What is the difference between RAM and FLASH? Why not use just RAM or just FLASH?

The MSP430F5529 memory space contains both RAM & FLASH because FLASH is non-volatile which means that the content is stored even when the power is removed or interrupted. It is much slower. However, RAM is volatile which means that the content is lost when power is interupted. Therefore, it is much faster. We need a combination of both volatile and non-volatile memory since, we want to store some data permanently & we need to write some data.

c) Flash serves a number of purposes in the MSP430. How much Flash memory does the MSP430F5529 have? Into how many banks is code memory divided? How many bytes per bank? Which bank of code memory contains the address 0x0CFA2?

Flash has 128 KB of memory in MSP430F5529. The code memory is divided into 4 memories (A,B,C,D). Each banks gets 32 KB. Bank B contains the address 0x0CFA2.

d) How much RAM does the MSP430F5529 have and how is it organized (number of banks, etc). What is the beginning address of RAM? What is RAM used for?

MSP430F5529 has 8 kB of RAM which is divided into 4 sectors (Sector 0, 1, 2, 3). There's also an additional 2kB of RAM which is a USB RAM in sector 7. The begining address of RAM is 002400h. It is used to retrieve or write data.

e) In lab, when CCS downloads a program to the MSP430F5529, where is the executable code stored? Beginning at what address?

The executable code is stored in Flash. The beginning address is 0400h.

f) When you declare a local variable inside your main() where in memory is thatvalue stored? What if you declared a global variable, where in memory are globals stored?

Local variables & global variables are going to be stored in RAM. If the variable is a constant it is going to be stored in Flash.

g) Why does the MSP430F5529 have a 20-bit address bus when its word size is 16 bits? Also why are registers R0 (i.e., Program Counter) and R1 (the Stack Pointer) 20 bit registers instead of 16 bit?

MSP430F5529 have a 20-bit address bus when its word size is 16 bits is because 20 bit memory address bus to allow access to up to "1 MB" of memory. R0 & R1 is 20 bit registers instead of 16 bit because they have dedicated functions.

h) What is the *address* for Port 6 direction register, the Port 3 input register, and the Port 7 select register. (See Datasheet Tables 9-26).

Port 6 direction register: 0245h [240+5=245h]
Port 3 input register: 0220h [0220h + 00h = 0220h]
Port 7 select register: 026Ah [0260h + 0Ah = 026Ah]

i) What MSP430 peripheral device uses addresses 00700h-0073Fh? What *package* pins numbers are used for Port 8 digital IO?

ADC12-A (analog to digital converter) uses addresses 00700h - 0073Fh. Package pins 15, 16 & 17 are used for Port 8 digital IO.

j) Most of the MSP430F5529 package pins have multiple functions that are selected using a select register. Why are all these pins multiplexed? What functionality is multiplexed with the Port 4.4-4.7 digital I/O? With Port 2.2 and 3.3? What package pin(s) is analog V_{cc} and which package pin(s) is GND (V_{ss})?

Package pin are multiplexed to incorporate the largest amount of protocol into the smallest package possible. PM_UCA1TXD & PM_UCA1SIMO are multiplexed with port 4.4. PM-UCA1RXD and PM_UCA1SOMI multiplexed with Port 4.5. PM_NONE are multiplexed with port 4.6 & Port 4.7. TA2CLK/SMCLK multiplexed with Port 2.2. UCA0TXD and UCA0SIMO are multiplexed with Port 3.3. Package pins 18 and 50 are analog Vcc, and package pins 19 and 49 are analog Vss or ground.

2) The MSP430 is a small, 16-bit microcontroller. The device's forte is that it is capable of remarkably low power operation. It is not a processor suited to computationally heavy applications. It is a fixed point (i.e., integer) processor. That means the CPU contains a digital circuit called the Arithmetic Logic Unit (ALU) which performs addition and subtraction and the logic operations AND, OR, NOT and XOR on integers but it does not contain a Floating Point Unit (FPU). That is, there is no circuit inside an MSP430 that performs arithmetic operations on floating point numbers. The Code Composer Studio compiler does provide full support for floating point operations, as all C compilers must, but all floating point math on an MSP430 is *emulated* in software which has significant implications for both code size and run time. You need to connect the lab board to the computer for this assignment. (10 pts)

i. Add a comment to each line of code below saying what it does.

Code a:

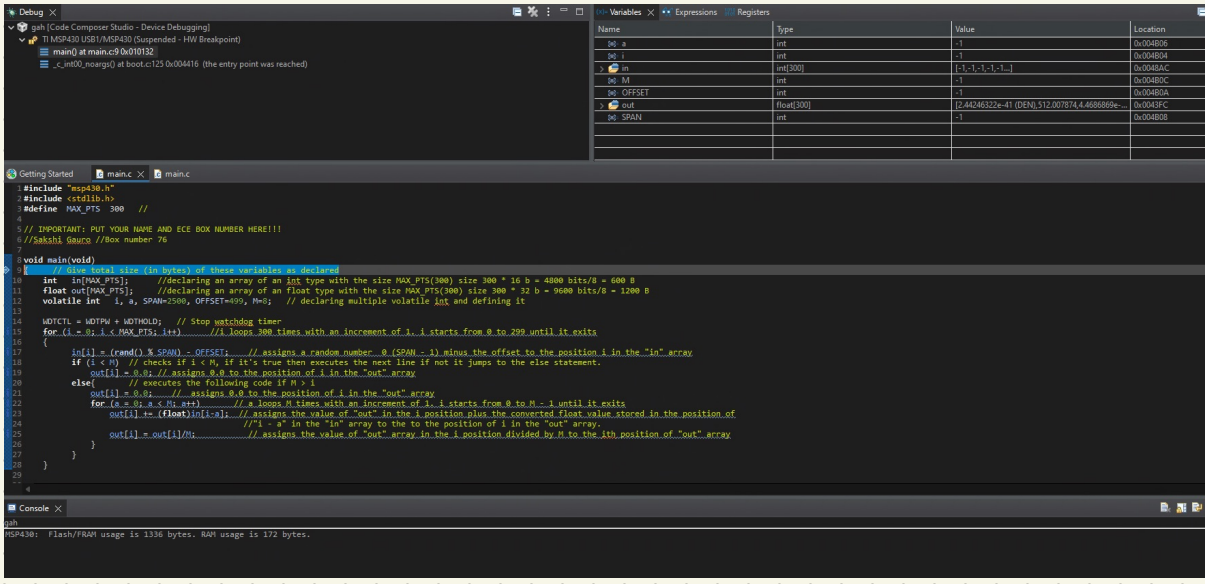
```
#include "msp430.h" #include <stdlib.h>
#define  MAX_PTS  300   //

// IMPORTANT: PUT YOUR NAME AND ECE BOX NUMBER HERE!!!
//Sakshi Gauro //Box number 76

void main(void)
{     // Give total size (in bytes) of these variables as declared
    int   in[MAX_PTS];      //declaring an array of an int type with the size MAX_PTS(300) size 300 * 16 b = 4800 bits/8 = 600 B
    float out[MAX_PTS];     //declaring an array of an float type with the size MAX_PTS(300) size 300 * 32 b = 9600 bits/8 = 1200 B
    volatile int   i, a, SPAN=2500, OFFSET=499, M=8;   // declaring multiple volatile int and defining it

    WDTCTL = WDTPW + WDTHOLD;   // Stop watchdog timer
    for (i = 0; i < MAX_PTS; i++)      //i loops 300 times with an increment of 1. i starts from 0 to 299 until it exits
    {
          in[i] = (rand() % SPAN) - OFFSET;     // assigns a random number  0 (SPAN - 1) minus the offset to the position i in the "in" array
          if (i < M)  // checks if i < M, if it's true then executes the next line if not it jumps to the else statement.
              out[i] = 0.0; // assigns 0.0 to the position of i in the "out" array
          else{       // executes the following code if M > i
              out[i] = 0.0;     //  assigns 0.0 to the position of i in the "out" array
              for (a = 0; a < M; a++){       // a loops M times with an increment of 1. i starts from 0 to M - 1 until it exits
                  out[i] += (float)in[i-a];  // assigns the value of "out" in the i position plus the converted float value stored in the position of
                                             //"i - a" in the "in" array to the to the position of i in the "out" array.
                  out[i] = out[i]/M;           // assigns the value of "out" array in the i position divided by M to the ith position of "out" array
              }
          }
    }
}
```

code b

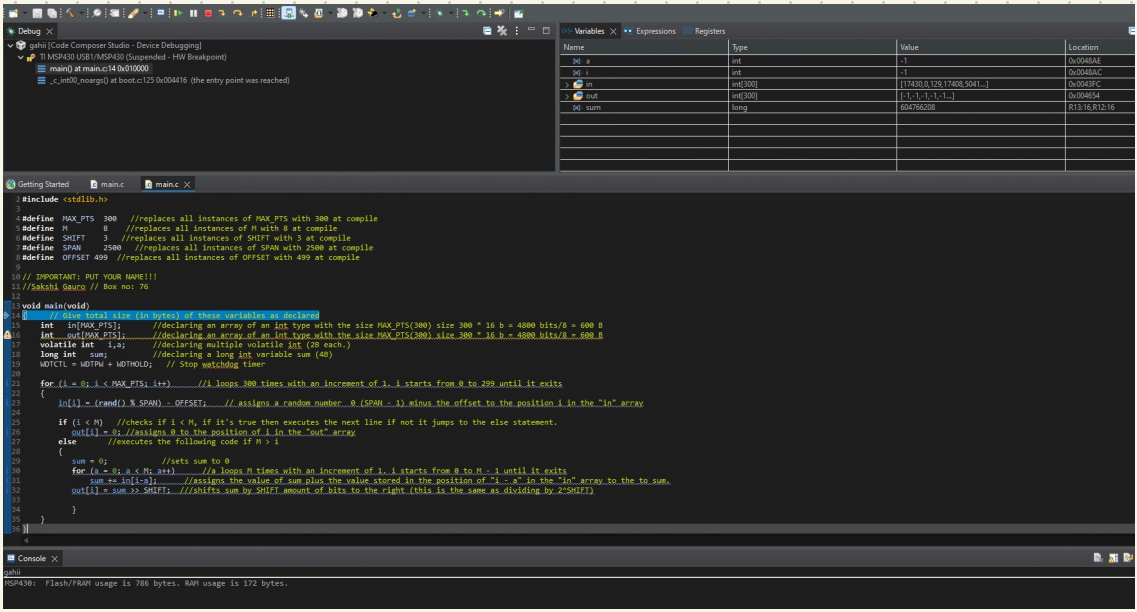
```
#include "msp430.h"
#include <stdlib.h>

#define  MAX_PTS  300   //replaces all instances of MAX_PTS with 300 at compile
#define  M        8    //replaces all instances of M with 8 at compile
#define  SHIFT    3   //replaces all instances of SHIFT with 3 at compile
#define  SPAN     2500   //replaces all instances of SPAN with 2500 at compile
#define  OFFSET 499  //replaces all instances of OFFSET with 499 at compile

// IMPORTANT: PUT YOUR NAME!!!
//Sakshi Gaurq // Box no: 76

void main(void)
{     // Give total size (in bytes) of these variables as declared
    int   in[MAX_PTS];       //declaring an array of an int type with the size MAX_PTS(300) size 300 * 16 b = 4800 bits/8 = 600 B
    int   out[MAX_PTS];      //declaring an array of an int type with the size MAX_PTS(300) size 300 * 16 b = 4800 bits/8 = 600 B
    volatile int   i,a;      //declaring multiple volatile int (2B each.)
    long int   sum;          //declaring a long int variable sum (4B)
    WDTCTL = WDTPW + WDTHOLD;   // Stop watchdog timer

    for (i = 0; i < MAX_PTS; i++)        //i loops 300 times with an increment of 1. i starts from 0 to 299 until it exits
    {
        in[i] = (rand() % SPAN) - OFFSET;      // assigns a random number  0 (SPAN - 1) minus the offset to the position i in the "in" array

        if (i < M)   //checks if i < M, if it's true then executes the next line if not it jumps to the else statement.
           out[i] = 0; //assigns 0 to the position of i in the "out" array
        else       //executes the following code if M > i
        {
           sum = 0;            //sets sum to 0
           for (a = 0; a < M; a++)        //a loops M times with an increment of 1. i starts from 0 to M - 1 until it exits
              sum += in[i-a];          //assigns the value of sum plus the value stored in the position of "i - a" in the "in" array to the to sum.
           out[i] = sum >> SHIFT;  ///shifts sum by SHIFT amount of bits to the right (this is the same as dividing by 2^SHIFT)

           }
    }
}
```

ii. Run these 2 simple programs in Code Composer Studio (CCS). Take a screen capture of each being run in the debugger.

Code a



Code b:



iii. Comment on the efficiency (both memory space and run-time) of the integer math program vs the floating point program.

Hint: The Console window in the debugger gives the code size at the bottom. You should also look at the size of the equivalent assembly code produced for each program in the Disassembly window. You may assume that it takes an average of 4-6 instruction cycles per assembly instruction to execute.

The floating point program was more memory space efficient and time efficient in comparison to the integer math. The integer math used 1336 bytes flash whereas, the floating point program used 786 bytes flash.

3) Write the following functions using CCS C to complete the code framework below. You do not have to compile this code, but your code must be typed to be graded. (20 pts)

(a) A function setupP2_P3() that selects P3.7-4 for digital IO with those bits set as outputs and selects P2.7-4 for digital IO with those bits set as inputs. The inputs P2.7-4 are connected to external switches and require internal pull-down resistors. When the switch is open, a logic 0 is input and when the switch is closed, a logic 1 is input. None of the settings for P2 or P3 should be altered. Default drive strength is fine.

next page

(b) A function in2_out3() that reads in from P2.7-4 to determine which switches are open and which are closed. The function then outputs a logic 1 on P3.7-4 if the switch on the correspond port 2 input is closed and a logic 0 if it is open (i.e. P2 pin 4 input corresponds to P3.4 output, P2.5 → P3.5, etc.).

→ next page

```c
#include <msp430.h>
void setupP2_P3(){
    P3SEL = P3SEL & (BIT1 | BIT2 | BIT3 | BIT0);
    P3DIR = P3DIR | (BIT7 | BIT6 | BIT5 | BIT4);
    P2SEL = P2SEL & (BIT1 | BIT2 | BIT3 | BIT0);
    P2DIR = P2DIR & (BIT1 | BIT2 | BIT3 | BIT0);
    P2OUT = P2OUT & (BIT1 | BIT2 | BIT3 | BIT0);
    P2REN = P2REN | (BIT7 | BIT6 | BIT5 | BIT4);
}

void in2_out3(){
    P3OUT = P3OUT | ~(BIT1 | BIT2 | BIT3 | BIT0);
    P3OUT = P3OUT & (P2IN | (BIT1 | BIT2 | BIT3 | BIT0));
}

int main(void){
    WDTCTL = WDTPW | WDTHOLD; // stop watchdog timer
    setupP2_P3();
    while(1){
        in2_out3();
    }
    return 0;
}
```

(c) Write a simple main() that sets up any variables and shows how your functions would be used. What value would result in P3OUT in your in2_out3() function if P2IN =0xB4? If P2IN =0x8F? Does it matter what the values of P3IN or P2OUT are? Explain.

```c
/* Compiler directives (includes and defines) */
#include "msp430.h"

/* Function prototypes */
void setupP2_P3();
void in2_out3();

/*** Implement your functions here ***/

  ...

/* Write your main() here */
void main()
{
    WDTCTL = WDTPW | WDTHOLD; // stop watchdog timer
    setupP2_P3();
    while(1){
        in2_out3();
    }
    return 0;
}
```

[1011]0100 ↑

P3OUT = 0xBx where x is the value it was before the functions were called

If P2IN = 0xB4

If P2IN = 0xBF

P3OUT = 0xBx where x is the value before the functions were called.

The values at P3IN matter because we are leaving the last nibble changes on the Pins.

The values at P2OUT matter because if we change the bits we are using the pull down resistor to pull up resistors, depending on how the external switches are configured. (if they connect and disconnect the pin from an external voltage) - The input will stay 1 and never change to 0 for the pull up pins.

4) Our MSP430F5529 Launchpad-based lab board has several digital IO devices on it including 2 LEDs, 2 push buttons, and a 3 x 4 keypad. (20 pts)

(a) Referring to resources like the MSP430F5529 Launchpad User's Guide and posted schematics for our lab board, fill out a table like the one below listing all the ports, pins, and MSP430F5529 package pins to which each of these devices is connected. This will be very useful to have in labs. (5 pts)

Device	I/O Port & Pins	MSP430F5529 Package Pins
2 Launchpad User Buttons	P7.0, P3.6, P2.2, P7.4	Pins 5, 43, 31, 57
2 Launchpad Used LEDs	P6.4 (Yellow), P6.2 (Red) P6.1 (Green), P6.3 (Blue)	Pins 1, 79, 78, 80
Keypad (shown below)	Columns: P1.5, P2.4, P2.5 Rows: P4.3, P1.2, P1.3, P1.4	Pins 42, 23, 25

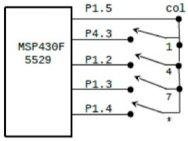
(b) Are the 2 user LEDs on the Launchpad inputs or outputs? How do you know? What logic level (1 or 0) will cause the user LEDs to light? Explain your answer. (2.5 pts)

The LEDs on the launchpads would be outputs because we want to be able to change their value without any external influences. Logic level 1 will cause the LEDs to light, because 1 sends voltage / current output to that pin.

(c) Here is the keypad configuration function and part of the getKey function which from the demo project and how column 1 of the keypad is connected to the MSP430F5529.

```
void configKeypad(void)
{
        // Configure digital IO for keypad
        // smj -- 27 Dec 2015

        // Col1 = P1.5 =
        // Col2 = P2.4 =
        // Col3 = P2.5 =
        // Row1 = P4.3 =
        // Row2 = P1.2 =
        // Row3 = P1.3 =
        // Row4 = P1.4 =
  // Select pins for digital IO
  P1SEL &= ~(BIT5|BIT4|BIT3|BIT2);
  P2SEL &= ~(BIT5|BIT4);
  P4SEL &= ~(BIT3);

  // Columns are ?
  P2DIR |= (BIT5|BIT4);
  P1DIR |= BIT5;
  P2OUT |= (BIT5|BIT4); //
  P1OUT |= BIT5;  //

  // Rows are ?
  P1DIR &= ~(BIT2|BIT3|BIT4);
  P4DIR &= ~(BIT3);
  P4REN |= (BIT3); //
  P1REN |= (BIT2|BIT3|BIT4);
  P4OUT |= (BIT3); //
  P1OUT |= (BIT2|BIT3|BIT4);
}

From getKey()
  P1OUT &= ~BIT5;
  P2OUT |= BIT4;
  P2OUT |= BIT5;
  if ((P4IN & BIT3)==0)
        ret_val = '3';
  if ((P1IN & BIT2)==0)
        ret_val = '6';
  if ((P1IN & BIT3)==0)
        ret_val = '9';
  if ((P1IN & BIT4)==0)
        ret_val = '#';
  P2OUT |= BIT5;
```

Fully describe the operation of the keypad. Indicate whether the various pins P1.5, P2.4,P2.5,P4.3,P1.2,P1.3 and P1.4 are inputs or outputs. On which ports and pins are the pull resistors being used. Are they pull up or pull down. Why are they necessary? Fully describe how the column read (from getKey) works. To what logic value (0 or 1) must P1.5 be set to be able to detect a key press in one of the rows in column 1. Explain. (10 pts)



The column pins (P1.5, P2.4, P2.5) are outputs, and the row pins (P4.3, P1.2, P1.3, P1.4) are input. Pull up resistors are being used on P1.4, P1.3, P1.2, P4.3. They are necessary because we set the column output to Ø, so when a key is pressed, the column shorts out the spot in that row and changes the signal in that input spot. The column read checks if each pin corresponding is on, and if it is, it changes the char value that will be returned to the char showing on that button. The code loops through each column and set one at the column pins to output Ø and then checks each row for an input "0" which could mean that the button in that row and column was pressed. P1.5 must be set to Ø to detect a key press. The rows are pull up, so connecting a row switch with col 1 will short it and change the input signal.

5) Assume that 4 slide switches like the one shown below are to be connected to the pins of Port 2 pins 6-3 such that when the switch is slid to the right it connects that pin to GND (0V) and when slid to the left it connects to V_{cc} (3.3V). (20 pts)

(a) Would these switches be digital inputs or outputs? How would they work (i.e. what switch operation would correspond to digital 1 or digital 0)?

3.3V P2.*x* GND

The switch would be digital inputs. Switching to the left would provide 3.3 V to the pin, so this corresponds to logic 1. Flipping the switch to the right would correspond to logic Ø.

(b) Describe an example application for these switches. How might they be used? How is their functionality different from the push buttons (i.e. Why would you choose use slide switches instead of buttons)?

You can use the switches to represent a 4 Bit Binary number, or to represent 2^4 different states in your code. You might chose slide switches instead of button switches because we can control the circuits current flow without manually cutting the wire. It is easier to choose which ones stay on and off so you don't have to hold down buttons the whole time.

(c) Write 2 functions in Code Composed Studio C to configure and use the switches and a simple main which uses your functions. The function switchConfig() should configure P2.6-3 for digital IO. You should leave Drive Strength in its default configuration and disable pull up/ pull down resistors. The settings for all other Port 2 pins should not be altered. The function switchIO() should return a value between 0-F hex which corresponds to the switch settings assuming P2.3 is the LSB and P2.6 is the MSB. You do not have to compile this code but it does need to be typed.

```
void switchConfig(){
    P2SEL &= ~(BIT6|BIT5|BIT4|BIT3);
    P2DIR &= ~(BIT6|BIT5|BIT4|BIT3);
    P2REN &= ~(BIT6|BIT5|BIT4|BIT3);
}
void switchIO(){
    char output;
    output = P2IN & 0x78;
    output = output >> 3;
    return output;
}
```

(d) Assume your program has properly configured the 4 slide switches for digital IO. How would your functions interpret the following settings of the Port 5 input and output registers P2IN = 0xB5 and P2OUT = 0x10, and P2IN = 0x6C and P2OUT = 0x08? What would the function switchIO() return in each case?

Since the pins are set to input the output only changes the pull up or pull down resistor. Since our switch is providing both a voltage in one stage and ground in another, the output settings make no difference as the inputs.

P2IN = 0xB5
↳ [0110] 1101

P2OUT = 0x10
000 10000 ← doesn't matter

↓
switch IO returns 0x08 [return type ⇒ char]

P2IN = 0x6D
↳ 0[110 1]101

P2OUT = 0x08
0000 1000 ← doesn't matter

↓
switch IO returns 0x0D

ECE2049 Homework #2

Submitted by: Sakshi Gaur

–

Date: 09/12/2022

Question	Grade
1-- 25 2-- 10 3-- 20	
4-- 20	
5-- 25	
Total: 100	