**Question 1:**

////////////////////

// Question 1 I A //

////////////////////

**#include** "msp430.h"

**#include** <stdlib.h>

**int** **main**(**void**)

{ // give size of these variable

**int** arr[100], i; // arr is 1600 bits, and i is 16 bits

**long** **unsigned** **int** sq\_sum; // sq\_sum is 32 bits

WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer

**for** (i=0; i<100; i++) // Loop 100 times

{

arr[i] = (**rand**() % 500)-150; // set arr[i] to a number between -150 and 350

sq\_sum += arr[i]\*arr[i]; // sq\_sum becomes the sum of the squares of all numbers in arr.

}

}

////////////////////

// Question 1 I B //

////////////////////

**#include** "msp430.h"

**#include** <math.h>

**#include** <stdlib.h>

**void** **main**(**void**)

{ // give size of these variable

**float** arr[100]; // arr is 3200 bits

**int** i; // i is 16 bits

**float** sq\_sum; // sq\_sum is 32 bits

WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer

i = 0;

**while** (i<100) {

arr[i] = (**rand**() % 500)-150.5; // Sets arr[i] to a number between 349.5 and -150.5 (always ending in 0.5)

sq\_sum += **pow**(arr[i],2.0); // sets sq\_sum to the sum of the squares of all numbers in arr.

i++; // Increment i. Causes loop to run 100 times

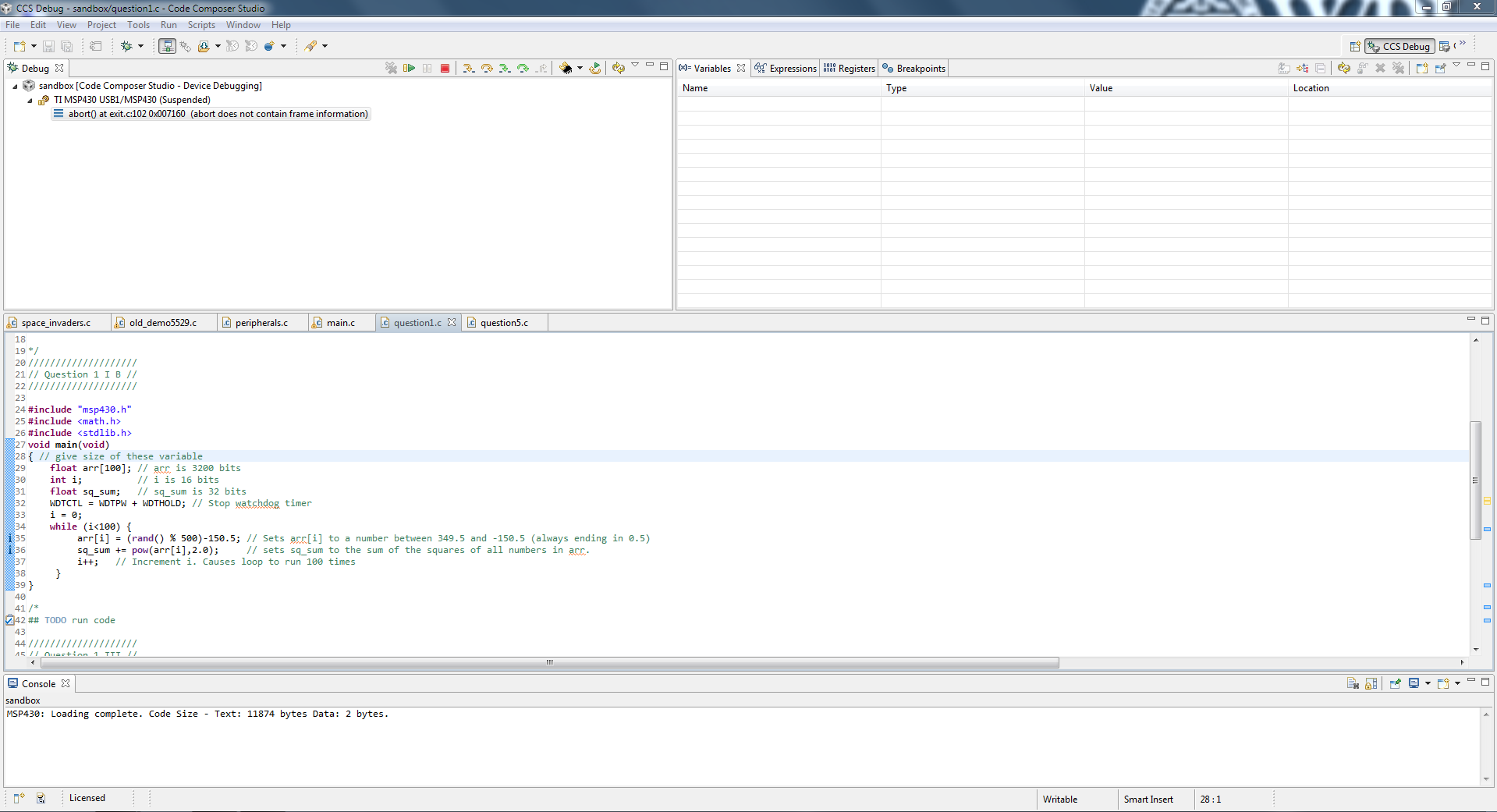
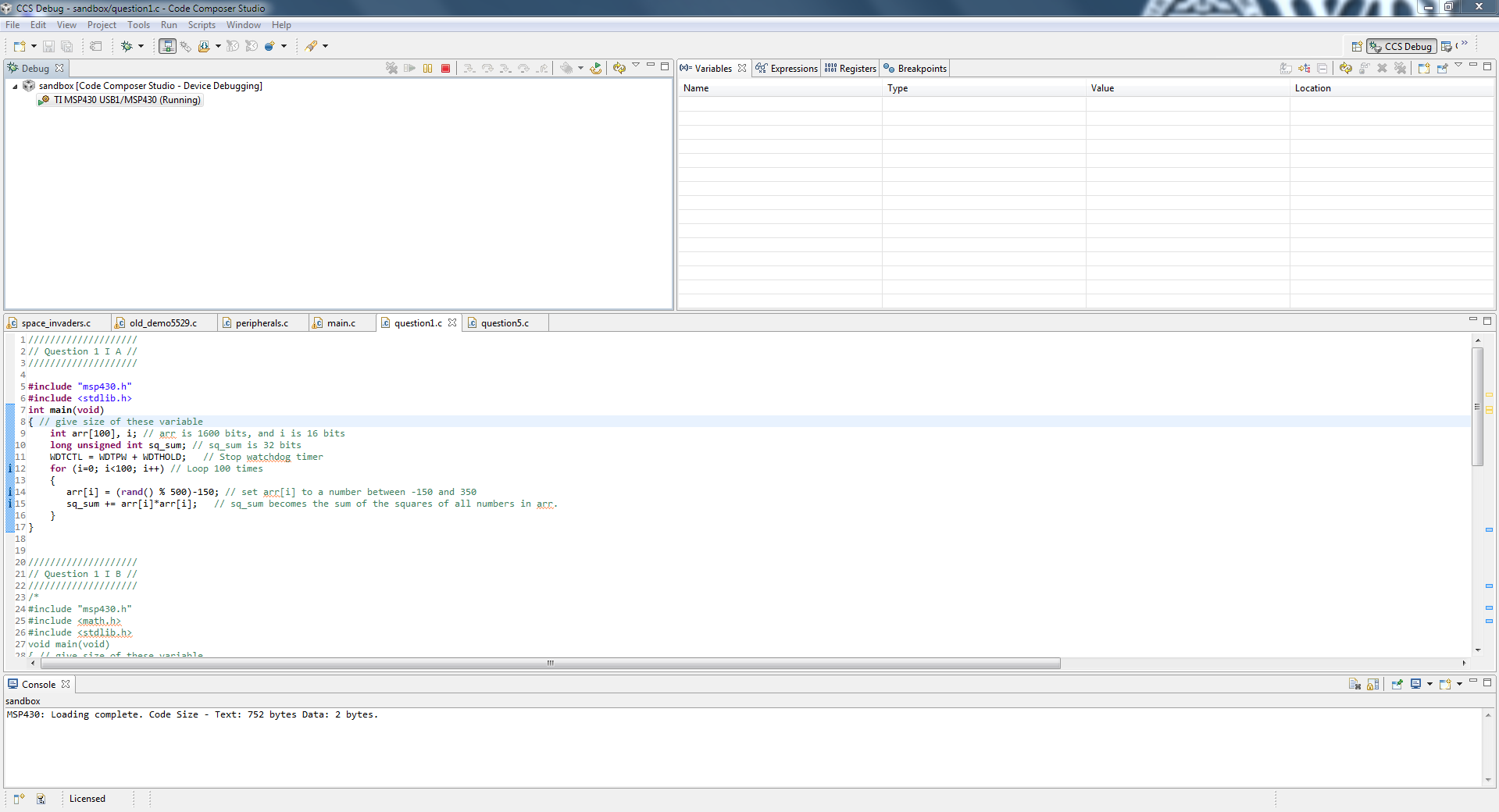
}

}

////////////////////

// Question 1 II //

////////////////////



////////////////////

// Question 1 III //

////////////////////

Integer math on the msp430 is much faster than **float** or **double** math because, not only are floats and doubles larger than the processor's word size, float and double operations have to be simulated in software. The declaration of the pow() function, and others in math.h also take up a lot of room because of their need to simulate complex floating point operations. Doubles are four times the size of integers, so they take up much more space. It is usually best to avoid using floats or doubles on a chip like the MSP430.

**Question 2:**

a)

The chip has a 20 bit address bus because if it had a smaller bus (e.g. the word size, 16 bits), it wouldn’t be able to address the amount of memory it needs to. 16 bits is enough to hold 65536 different addresses, but code memory (flash) has 131072 bytes alone!

The program counter (PC or R0) and the Stack Pointer (R1) registers both are 20 bits because they are required to hold an entire memory address to access memory through the 20-bit address bus.

b)

It uses both RAM and flash memory because they fundamentally serve different purposes - RAM is volatile memory (needs constant power) and is much faster than flash memory (which is non-volatile and can be used to save programs and data between resets).

c)

There is 128 KB of core memory, though there is 133632 bytes of flash memory total (128 KB of code memory, 512 bytes of information memory, and 2KB of memory for the bootstrap loader).

There are four banks of code memory, named A-D. They each have 32 KB of flash memory.

The address 0x0DF4A is located in bank B.

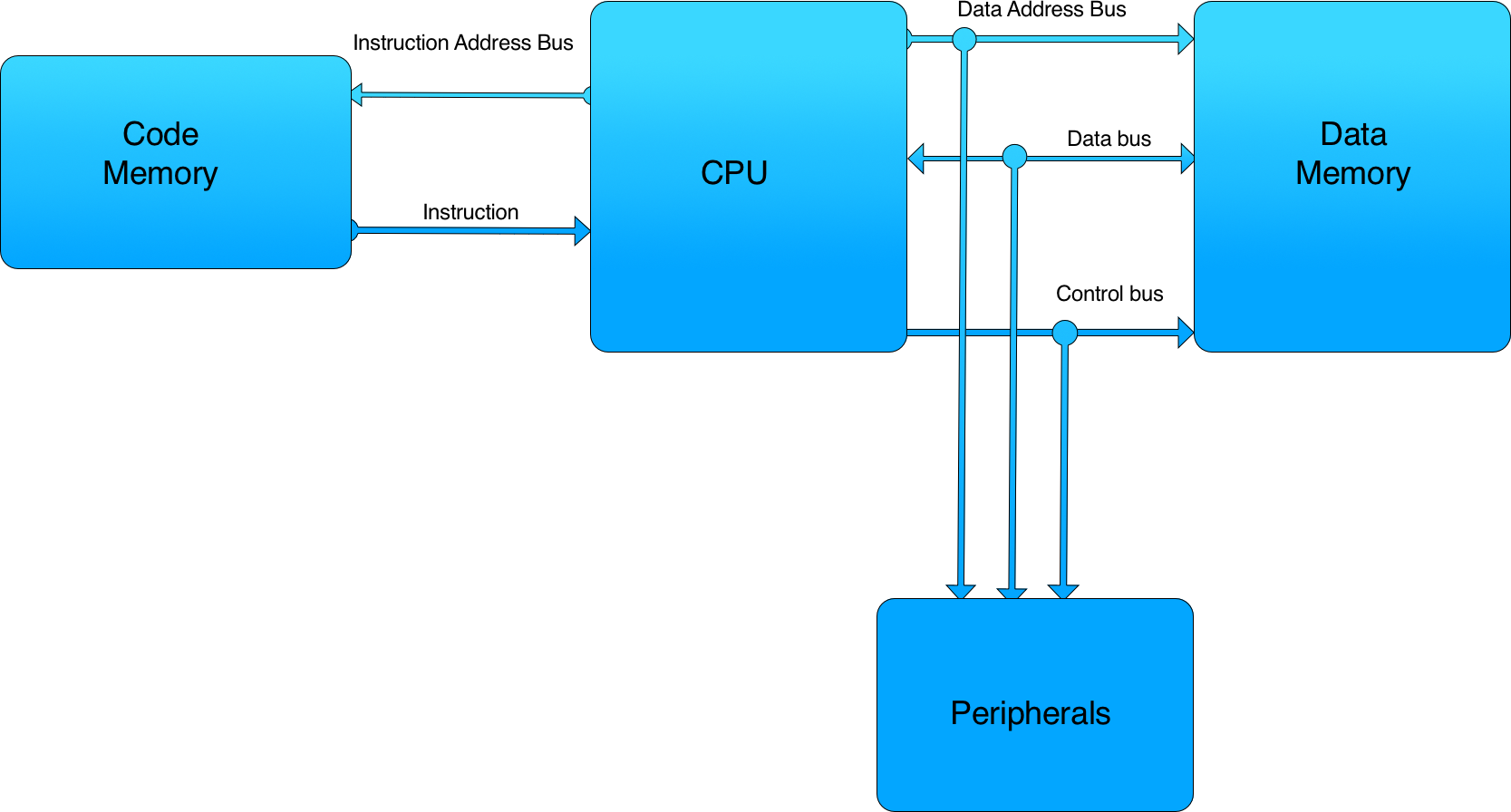
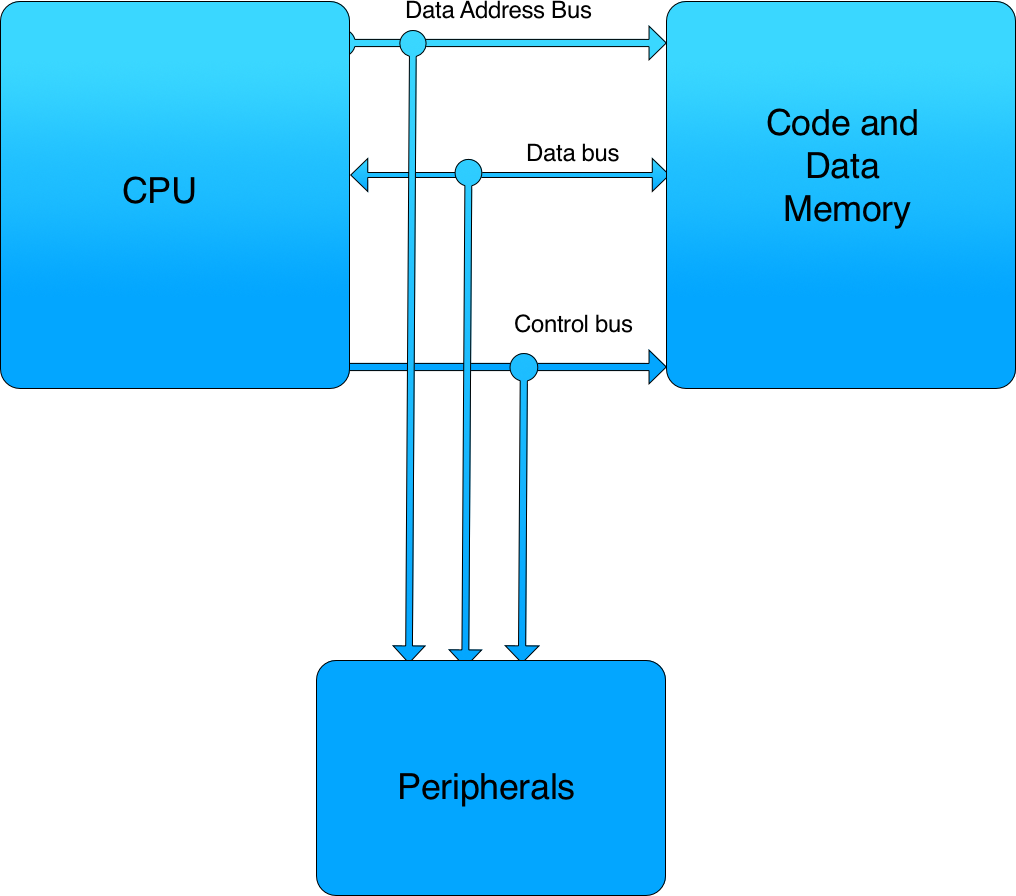
d)

CCS stores the executable code in code memory, the lowest address of which is 0x4400.

e)

If I declared a local variable inside of main(), that variable would be stored in the stack, which is in RAM. A global variable is also stored in RAM, but not in the stack.

**Question 3:**

a)

Von Neumann Architecture

Harvard Architecture

The Harvard architecture uses two different memory address spaces (one for instruction memory, and one for data) while the von Neumann architecture uses just one for both instructions and data. The von Neumann architecture is much simpler to implement and has advantages in memory allocation, but is less secure than the Harvard architecture (where code in data memory can’t be run) and isn’t as good at pipelining instructions and caching because the Harvard architecture has two memory busses.

b)

P6DIR is located at 0x0245

P3IN is located at 0x0220

P7SEL is located at 0x026A

c)

The 12 bit Analogue to Digital Converter uses addresses 0x700 to 0x73F.

And Port 8 uses package pins 15, 16, and 17 (pins 0, 1, and 2 respectively).

d)

The pins are multiplexed because adding a more pins to the MSP430 is much harder than giving those pins additional functionality. Texas Instruments does want to support all of the different operations, but in many projects, only a few of them are used, which leaves unused pins that could be used for digital I/O.

Port 4 pins 4-7 are used for a Universal Serial Communication Interface.

Port 1 pin 2 is used for Timer A 0 pin 1.

Port 2 pin 3 is used for Timer A 2 pin 0.

Package pin 18 is used for Digital Vcc, and 19 is used for GND (Digital VSS1). Analogue Vcc (AVCC1) is pin 11, and Analogue Vss is pin 14.

**Question 4:**

a)

|  |  |  |
| --- | --- | --- |
| **Device** | **I/O Ports and Pins** | **Package Pins** |
| Touch Pad LEDs | P1.1, P1.2, P1.3, P1.4, P1.5 | 22, 23, 24, 25, 26 (respectively) |
| LEDs 1-3 | P1.0, P8.1, P8.2 | 21, 16, 17 |
| Scroll Wheel | P8.0 | 15 |

b)

The push buttons and scroll wheel are all inputs, simply because the Experimenter Board doesn’t have the hardware to push the buttons or turn the scroll wheel itself.

c)

Port 3 pins 7, 5, 3, and 1 are being used as digital output and Port 1 pins 6, 4, 2, and 0 are being used as digital input.

The PxSEL register is set low for all of these pins, which means they are being used for digital I/O. P3DIR is set high for the Port 3’s odd pins, which indicates they are being used for output. P1DIR is set low for the bits corresponding to Port 1’s even pins which means they are configured for input.

If P1IN is 0x7E, then P3OUT is set to 0xFC ( 01111110b shifted to 11111100b).

**Question 5:**

/\* Compiler directives (includes and defines) \*/

**#include** "msp430x44x.h" // Should probably be #include "msp430.h".

// The way it currently is gives me errors when run

**#include** <stdio.h>

**#include** <stdlib.h>

**#include** <math.h>

/\* Function prototypes \*/

**void** **setupP6**();

**void** **P6inOut**();

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

// configures Port 6's pins as digital inputs or outputs

**void** **setupP6**() {

P6SEL = 0x00; // Configure all pins as digital IO

P6DIR = 0xF0; // Configure pins 3-0 as digital inputs, 7-4 as outputs

// Not changing P6OUT, P6REN, or P6DS

}

// Sets P6.7-4 to the complement of P6.3-0

**void** **P6inOut**() {

// Could have just set P6OUT to P6IN << 4, but I wanted to be tidy.

P6OUT = (~P6IN << 4) + (0x0F & P6OUT);

}

/\* Write your main() here \*/

**void** **main**() {

setupP6();

**while**(1)

P6inOut();

}

//////////////////

// Question 5 C //

//////////////////

If P6IN == 0xFA, P6OUT would be set to 0xAX where X is whatever P6OUT was originally set to.

If P6IN == 0x55, P6OUT would be set to 0x5X where X is whatever P6OUT was originally set to.

**Question 6:**

//////////////////

// Question 6 A //

//////////////////

Switches would be digital inputs. The switch in the right position would produce a digital 0, the left position would produce digital 1.

//////////////////

// Question 6 B //

//////////////////

Switches could be used to input a sequence of 0s and 1s (e.g. you have 4 toggleable states that change the type of program you’re going to run). Switches would be better than buttons, buttons aren’t helpful when you want to hold a state through restarts.

//////////////////

// Question 6 C //

//////////////////

**void** **switchConfig**() {

P6SEL &= ~(BIT5|BIT4|BIT3|BIT2); // Set P6.5-2 to digital IO

P6DIR &= ~(BIT5|BIT4|BIT3|BIT2); // Set P6.5-2 to digital In

P6DIR &= ~(BIT5|BIT4|BIT3|BIT2); // Explicitly disable pullup and

// pulldown resistors

}

// Outputs the inputs of P6.5-2 as a number 0 to 15

**char** **switchIO**() {

**return** (P6IN >> 2) & 0x0F; // Shifts P6.5-2 to bits 3-0, and sets other bits to 0.

}

//////////////////

// Question 6 D //

//////////////////

When P5IN == 0xB5 and P5OUT = 0x08

The switch connected to pin 3 is in the right position, all others are in the left position. switchIO() produces 0x0C (00001101b).

When P5IN == 0x6C and P5OUT == 0 (Because P4 doesn’t involve the switches at all, I’m assuming that is a typo. If it isn’t a typo, the switch behavior is undefined.)

The switch connected to pin 4 is in the right position, the rest are in the left position. switchIO() produces 0x0B (00001011b).