EE 128

Lab 7:
Serial Communication (SPI and I2C)

Section 021
TA: Mohsen Karimi

Tanish Arora
Leya Zeng

1. Abstract

To get familiar with SPI and I2C interfacing and programming with the use of HCS12 board. IO manipulations and data transmit.

2. Procedure

Part1:

i) Assemble the circuit as shown in the schematic.

ii) Download the Lab7 template code and compile it and run it to check if the functionality is working properly or not.

iii) Modify the template code in such a way that the LED bar would display bits in reverse order and the also setup the hardware (change the output pins to input and input to output), you may have to modify code as well.

Part2:

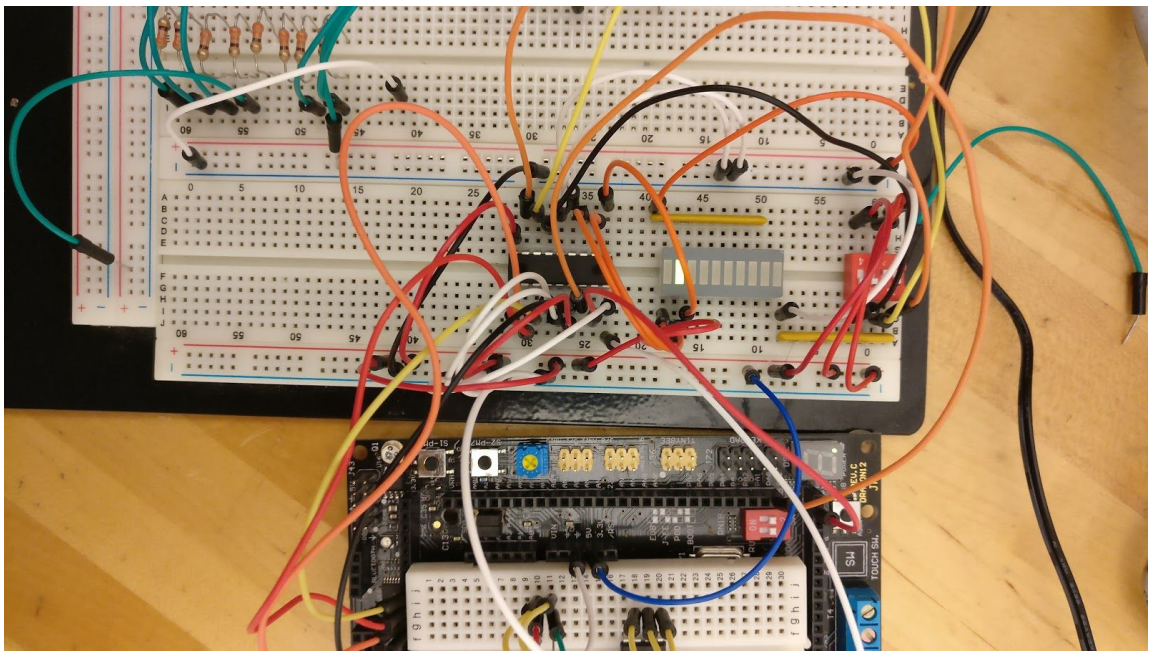
i) Assemble the circuit as shown in schematic.

ii) Download the template code from piazza, Compile and run to check if the connections are proper. If not, fix them.

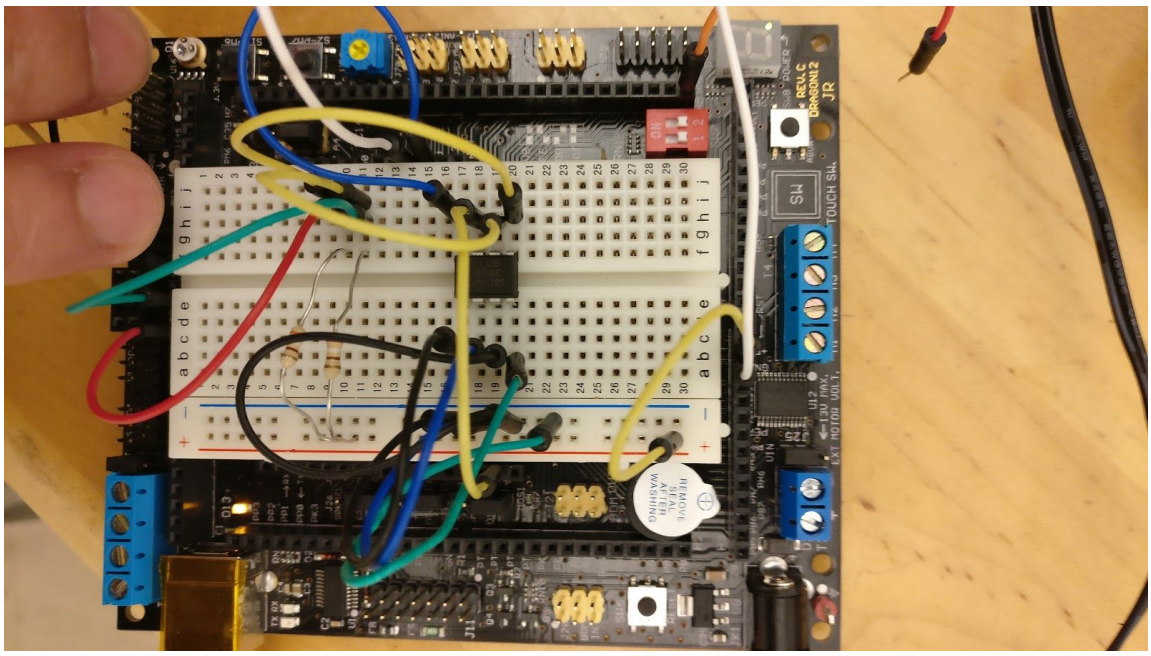
iii) Modify the code in such a way that you can write a page (8 bytes) of information in sequence with just a single command.

3. Experiment System Specification:

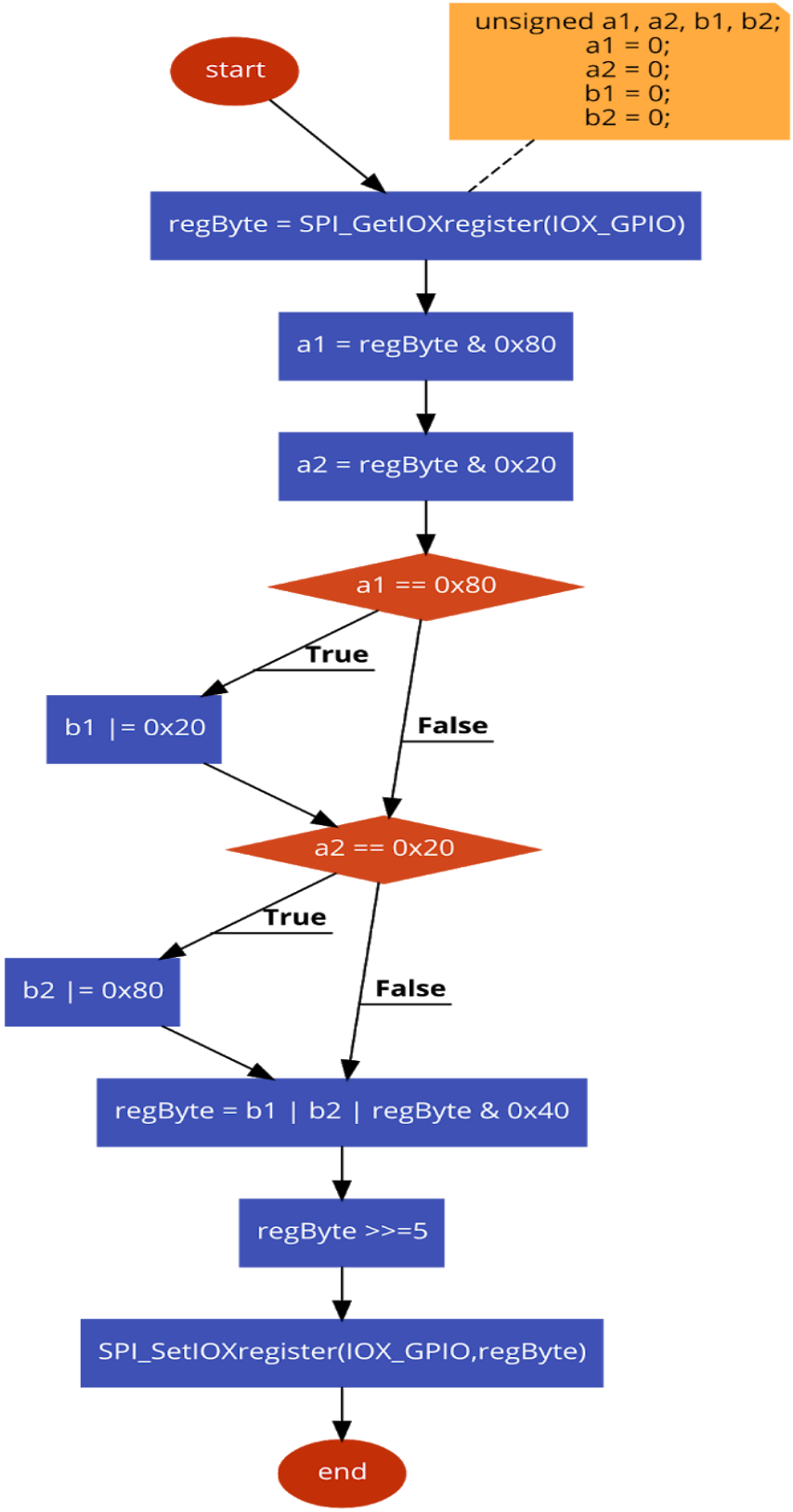
***Lab hardware photo part1*:**



***Lab hardware photo part2*:**

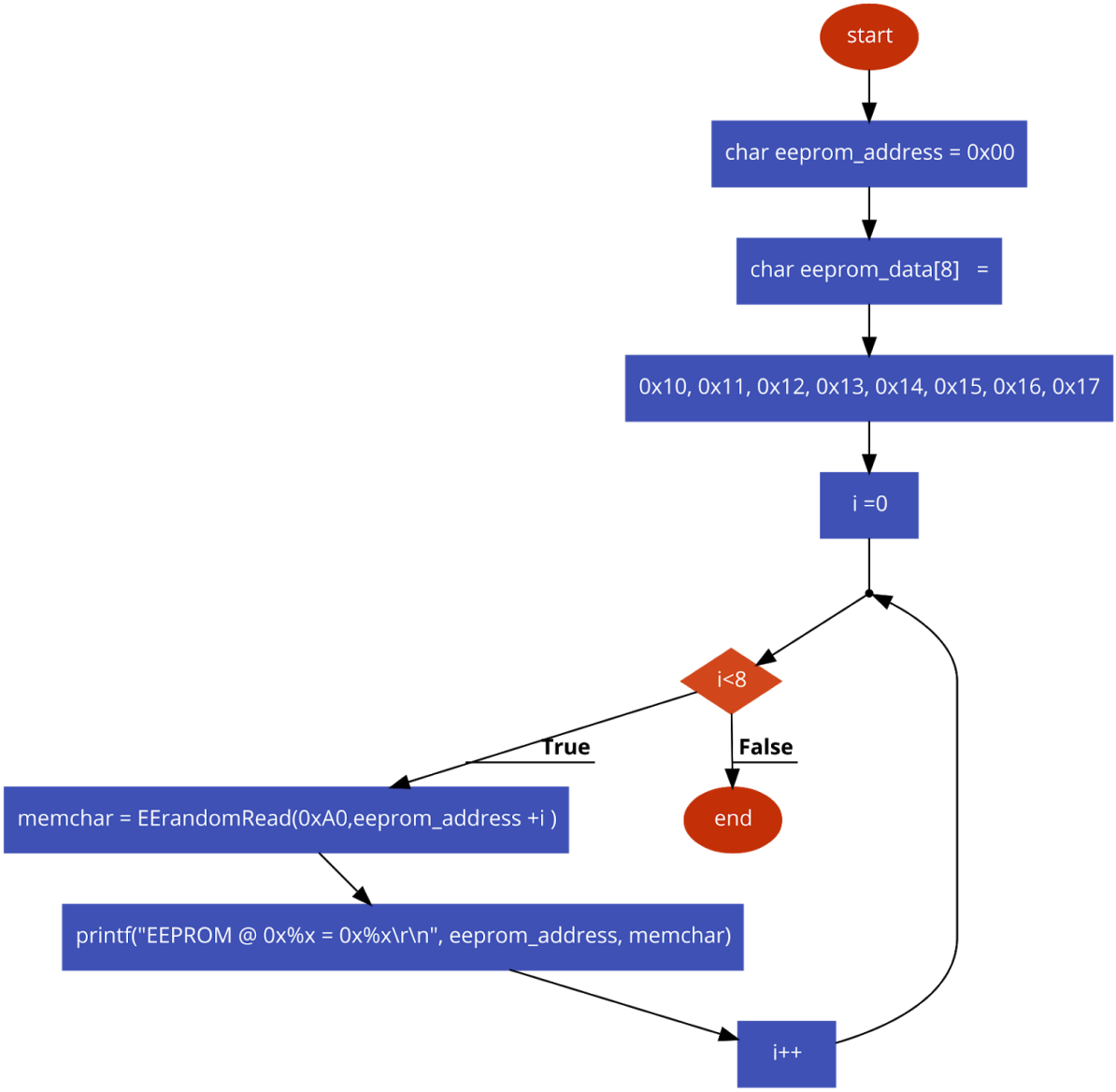


Flowchart; 9s12 Microcontroller part1:

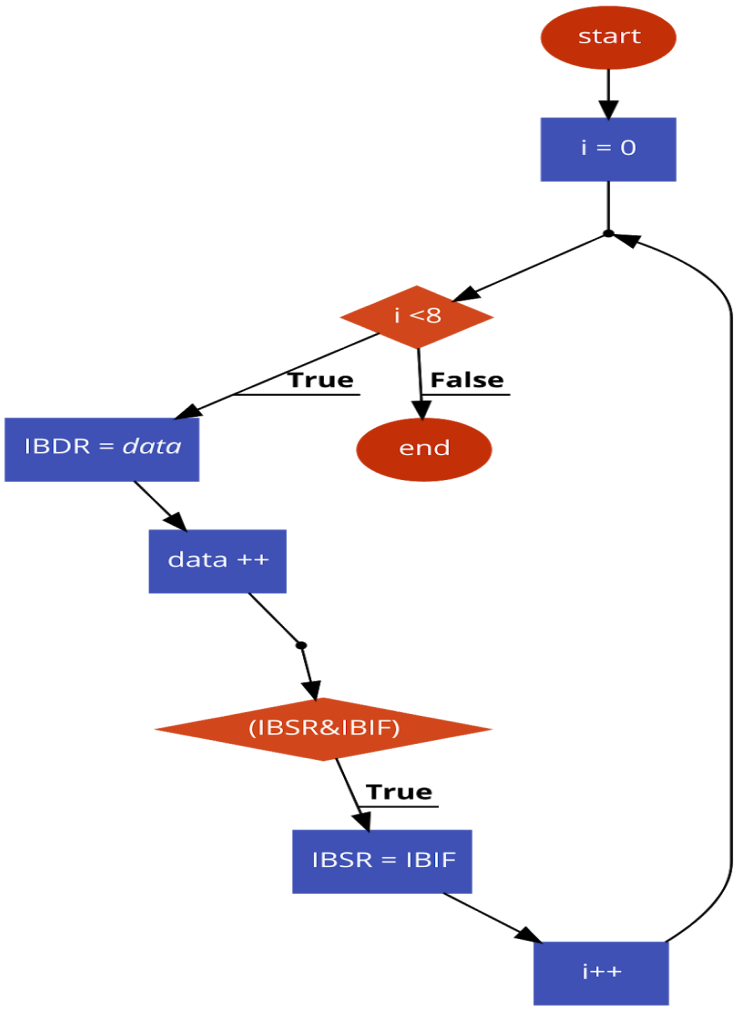


Flowchart; 9s12 Microcontroller part2:

Main :



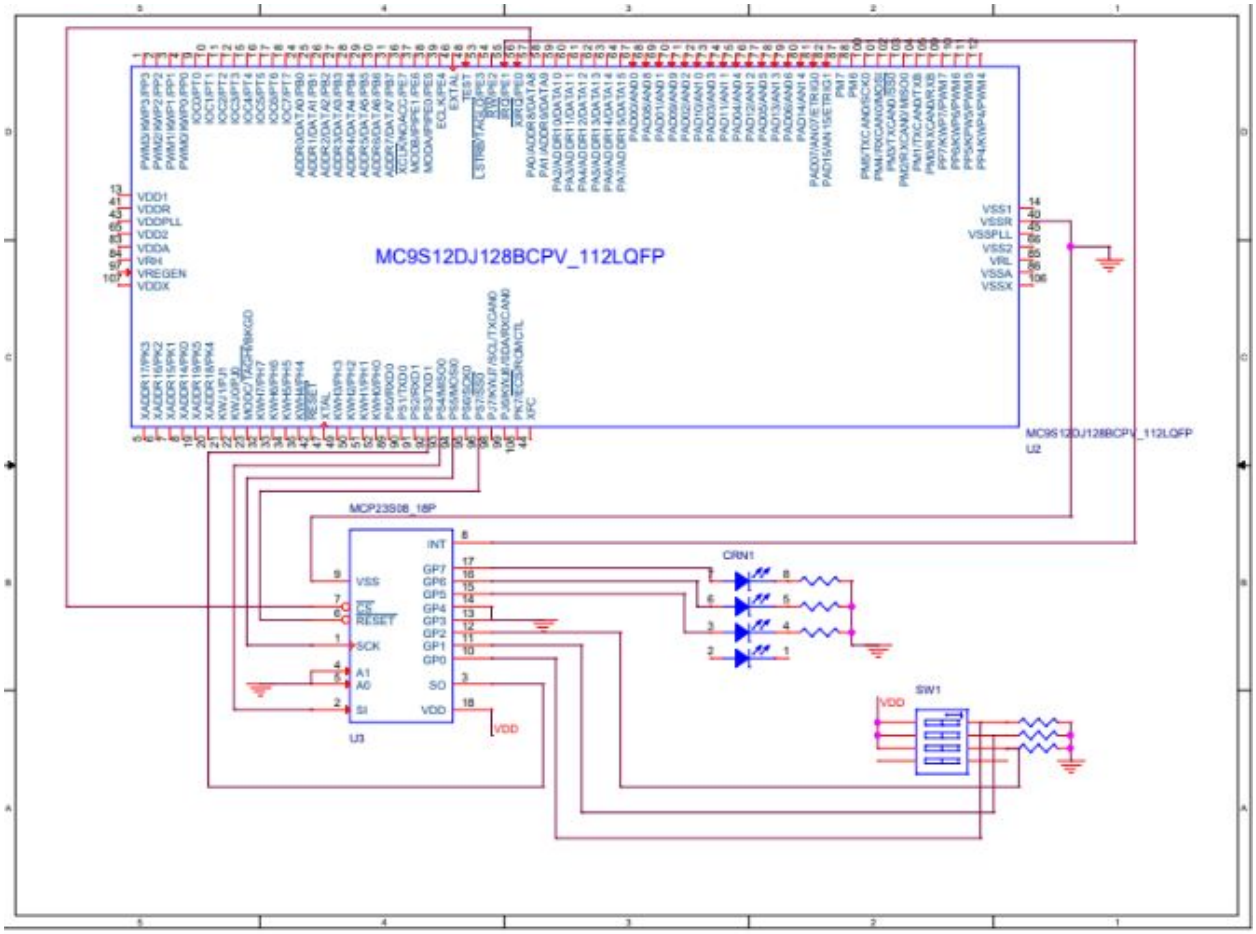
I2c:


start
i = 0
i <8
True
False
IBDR = data
end
data ++
(IBSR&IBIF)
True
IBSR = IBIF
i++

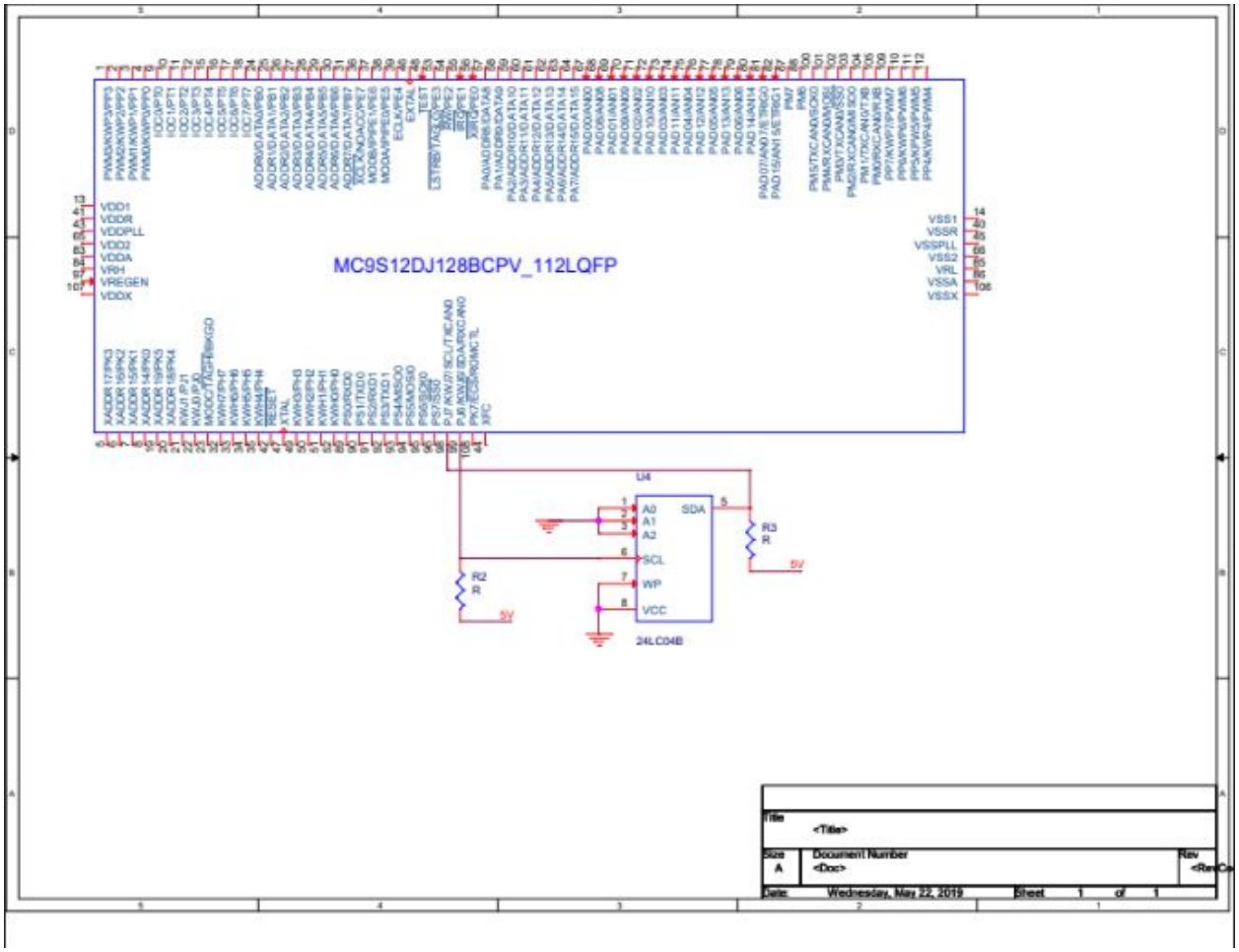
4. Hardware Design Schematics:

9S12 Controller;

Part1 :



Part 2:



5. Software Design:

Part1:

```
__interrupt void IRQISR(void)
/* interrupt routine to set new LED/Oscilloscope outputs */
{
  unsigned a1, a2, b1, b2;
  a1 = 0;
  a2 = 0;
  b1 = 0;
  b2 = 0;

  regByte = SPI_GetIOXregister(IOX_GPIO);

  a1 = regByte & 0x80;
  a2 = regByte & 0x20;
```

```
  if (a1 == 0x80) b1 |= 0x20;
  if (a2 == 0x20) b2 |= 0x80;
  regByte = b1 | b2 | regByte & 0x40;
  regByte >>=5;
  SPI_SetIOXregister(IOX_GPIO,regByte);
}

// Hardware change of IO
regByte = 0x1F;  /* GP7..GP5 - output (LED's); GP2..GP0 - input (switches) */
SPI_SetIOXregister(IOX_IODIR,regByte);
--------------
regByte = 0xE8;  /* GP2..GP0 - output (LED's); GP7..GP5 - input (switches) */
SPI_SetIOXregister(IOX_IODIR,regByte);
```

High level Description:

1) We first mask all the bits where we can get the input from.
2) Then we reverse it and output it out to a tempBit.
3) Then we use shifting to output it out to regByte.
4) Hardware change of IO, first was Pin GP7..GP5, then change it to GP2..GP0
5) After setting the IO both in hardware and software, we can use the code to change the behavior(order) of the LED.

Part2:

Main

```
char eeprom_address = 0x00;  /* address in EEPROM to write a byte to */
  char eeprom_data[8]   = {0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17};  /* data byte to be written to
EEPROM */

  for( i =0; i<8; i++) {
    memchar = EErandomRead(0xA0,eeprom_address +i );
    printf("EEPROM @ 0x%x = 0x%x\r\n", eeprom_address, memchar);
  }
```

High Level description :

1) First we initialize the eeprom_address to start from 0x00.
2) Then we initialize the eeprom _data array to store the data byte to be written to EEprom
3) Define a for loop to first read the data and to print as well.

i2c

```
  for (i = 0 ; i <8 ; i++){

    IBDR = *data;     /* send out the data byte */
    data ++;
    while(!(IBSR&IBIF));
    IBSR = IBIF;
    //data++;     /* clear the IBIF flag */
```

}

High Level Description:

1) We make changes to this file in order to store the page write pointer in the register to be able to print out
2) We basically copy paste the EEByte write and rename the copied one to EEPageWrite, which takes in pointer, ID and address as input.
3) And make changes to the code as given above. For i ->0 to 7, we send out the data byte to each pointer.

6. Problems Encountered:

We encountered a lot of problems while doing this lab, First our LED bar would not output out anything, with the help of the TA, we were able to figure out that our circuit was wrong. But since, we were facing so many problems , we decided to do part 2 first. After finishing part 2, we re tried part 1, unfortunately due to not a good planning , we were not able to finish it on time, but we were able to make changes to the code which theoretically seems to be right.

7. Questions:

1.

SCI1BD = 104 // 14400 baud

SCI1CR1 = 0; // SCI enabled in wait mode; enable parity and use odd parity

SCI1CR2 = 0xC; // Enable TDRE and RDRF interrupts; enable TX and RX

2.

SPI0BR = 0x52;

One possible way: 6x8 = 48 || (5+1)*2^3 -> 5 &2

SPI0CR1 = 0x5e;

SPI0CR2 = 0;

3.

a) What is the address of the slave?

00110100 -> 0x36

b) What is the data value being transferred? Is the master reading or writing the data?

00100101 -> 0x27, Master Write.

8. Conclusion:

This lab was overall really interesting , as we got to work with the 9S12 microcontroller and the spio and i2c integration. Although, we had to jump a lot of hoops in order to finish the lab, but we learnt a lot of things during this lab. Some more C programming techniques and hardware - software IO changes.