1. Mini project: Drink Dispenser

Name(s): Grant Beatty, Peter Chau

Date: December 4, 2021

Lab Section number: 021

TA: Subed Lamichhane

Video:
https://www.youtube.com/watch?v=8-f-2Io5jfw&ab_channel=GrantBeatty

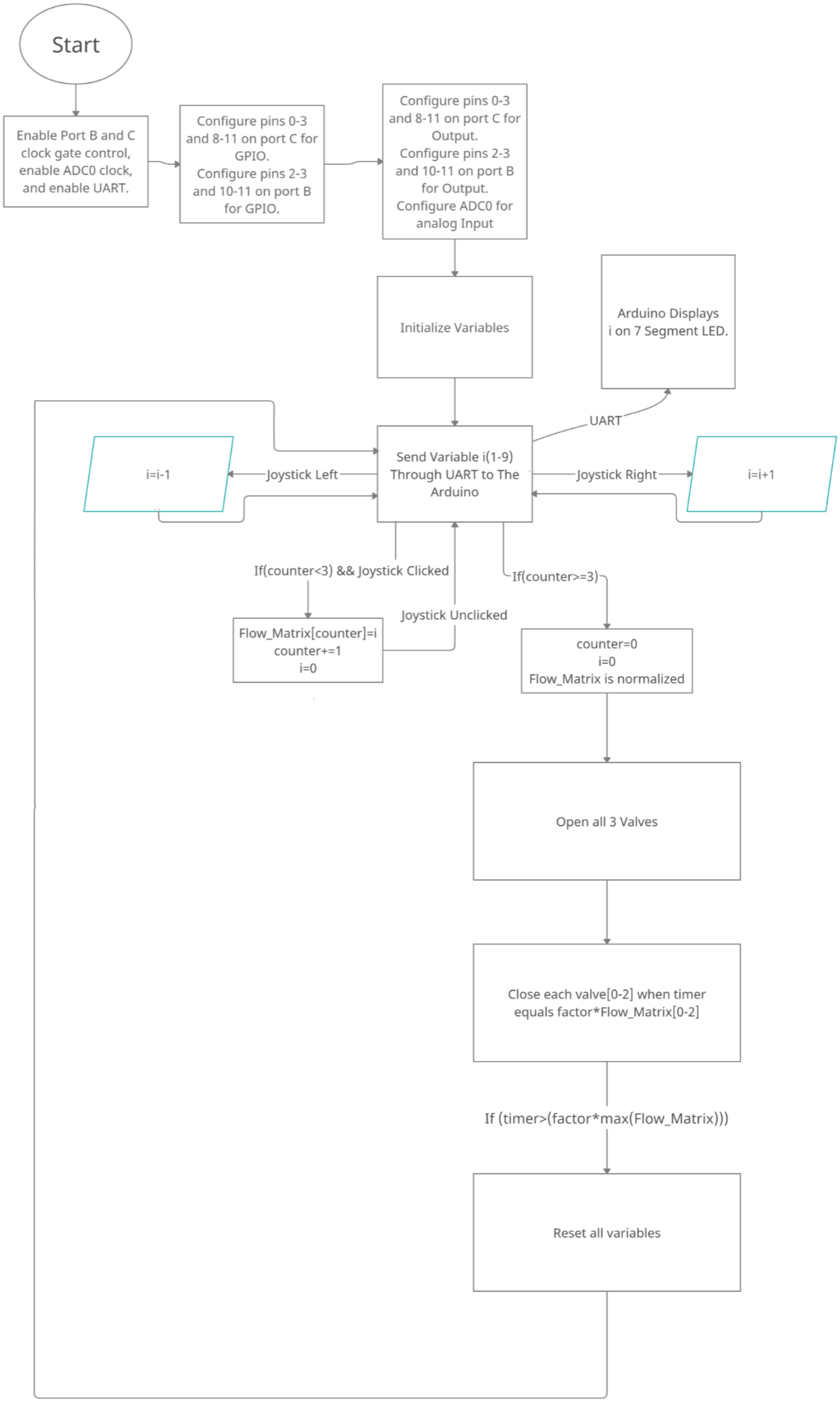
2. Project Description: summary, requirements/goals

The project is a custom selected beverage dispenser. The system has 3 “tanks” with tubes that connect them to a drink cup. Each tube has a valve with a motor that controls the flow of the liquid from the “tanks”. The system should be able dispense from three different sources and the user should be able to determine how long each motor will dispense liquid for.

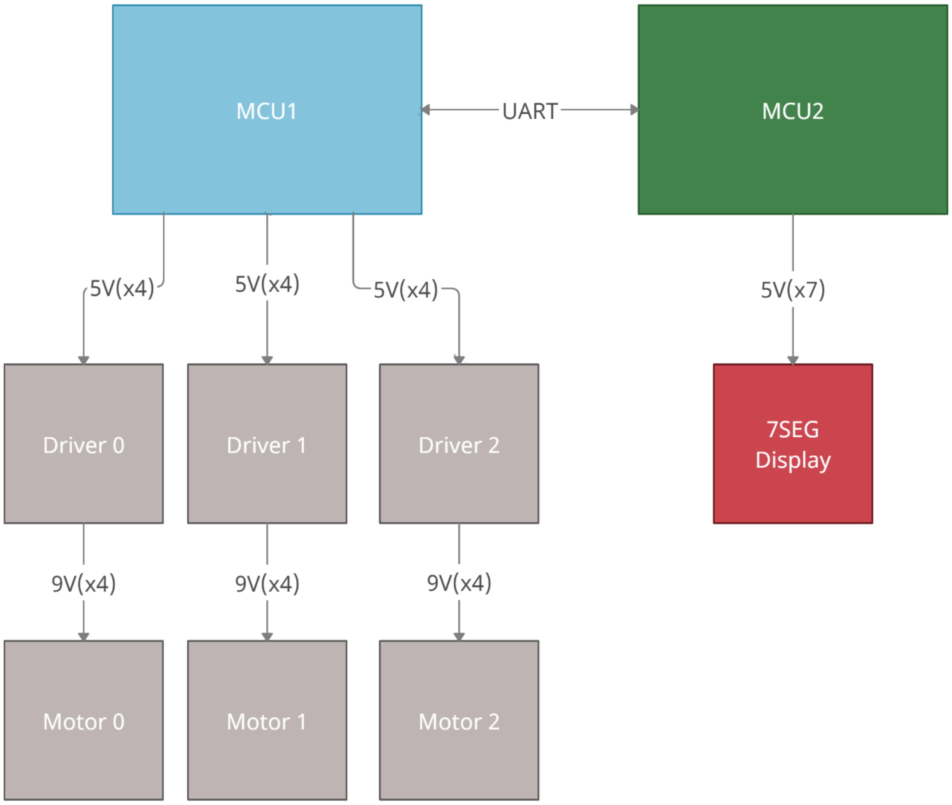
Here are the requirements and goals. We are to design a user interface that displays a number that the user can change and select. The interface should allow 3 numbers to be selected and should signal when each selection has been made. The 3 numbers represent the ratio of on time for each valve. After each number is selected, each valve twists and remains open for their corresponding time and releases a set amount of liquid into the cup.

3. System Design: block diagrams, flow charts

Flow Chart


Start
Enable Port B and C clock gate control, enable ADC0 clock, and enable UART.
Configure pins 0-3 and 8-11 on port C for GPIO. Configure pins 2-3 and 10-11 on port B for GPIO.
Configure pins 0-3 and 8-11 on port C for Output. Configure pins 2-3 and 10-11 on port B for Output. Configure ADC0 for analog Input
Initialize Variables
Arduino Displays i on 7 Segment LED.
UART
Send Variable i(1-9) Through UART to The Arduino
i=i-1
Joystick Left
Joystick Right
i=i+1
If(counter<3) && Joystick Clicked
If(counter>=3)
Joystick Unclicked
Flow_Matrix[counter]=i
counter+=1
i=0
counter=0
i=0
Flow_Matrix is normalized
Open all 3 Valves
Close each valve[0-2] when timer equals factor*Flow_Matrix[0-2]
If (timer>(factor*max(Flow_Matrix)))
Reset all variables

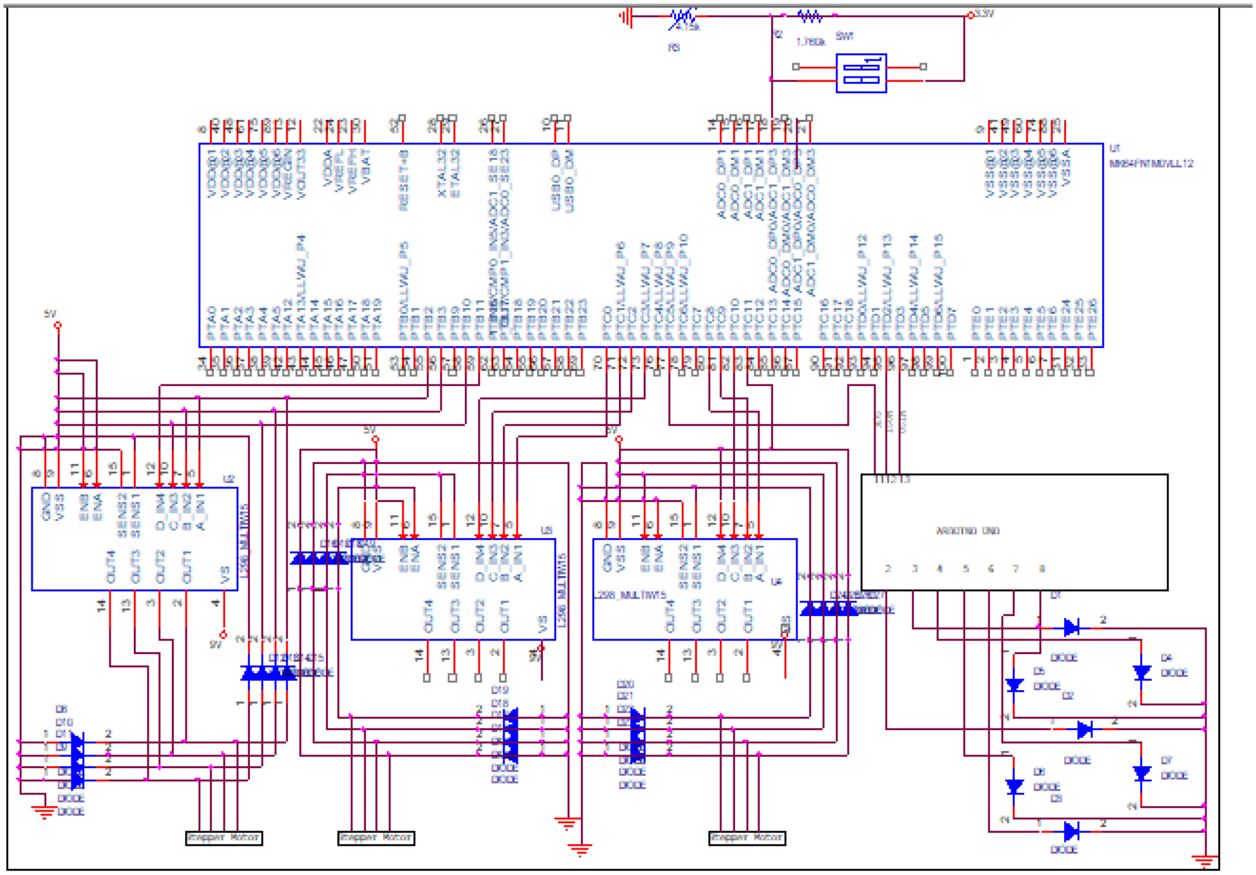
Block Diagram



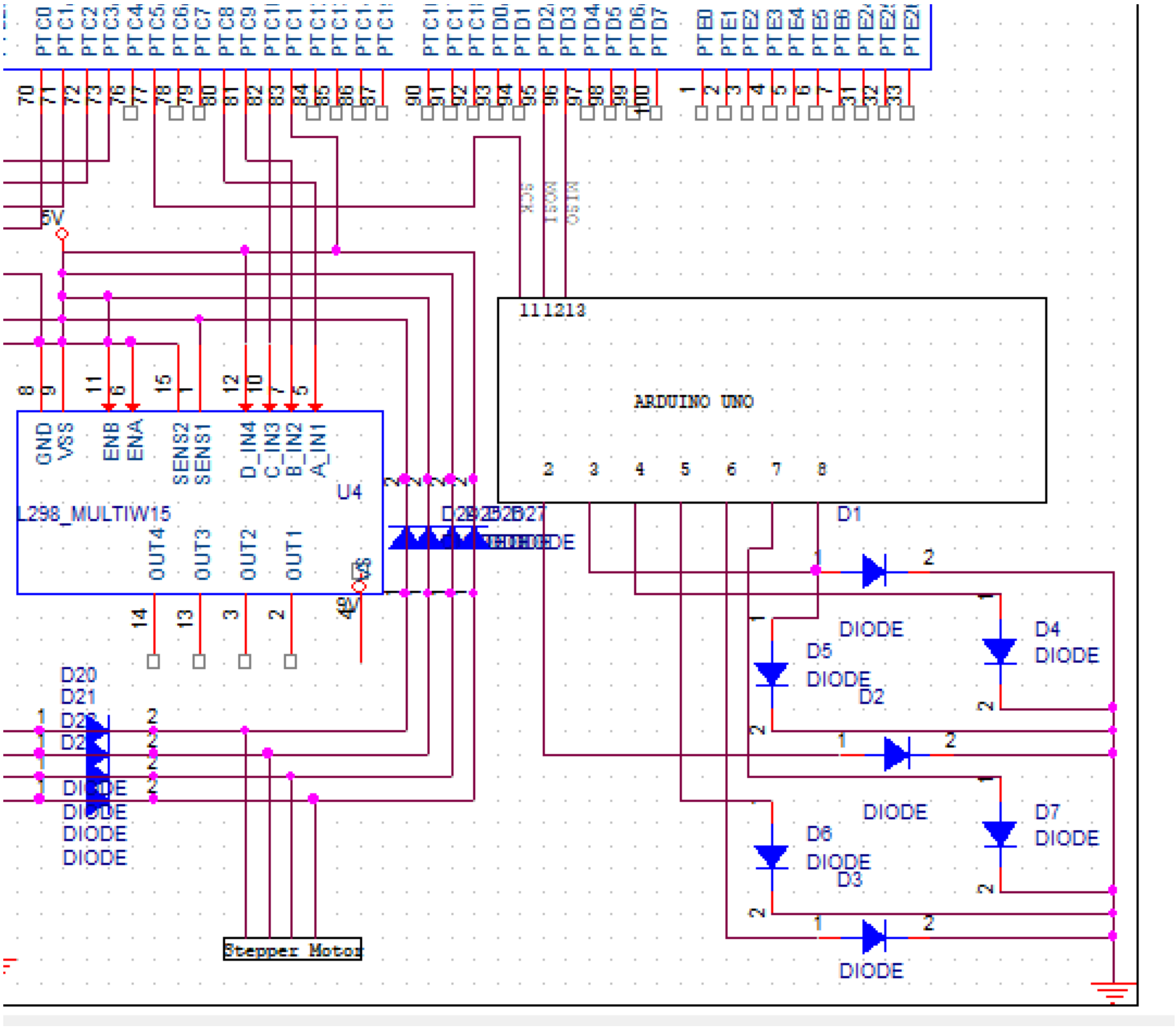
4. Implementation Details: schematics, some key portions

Schematics

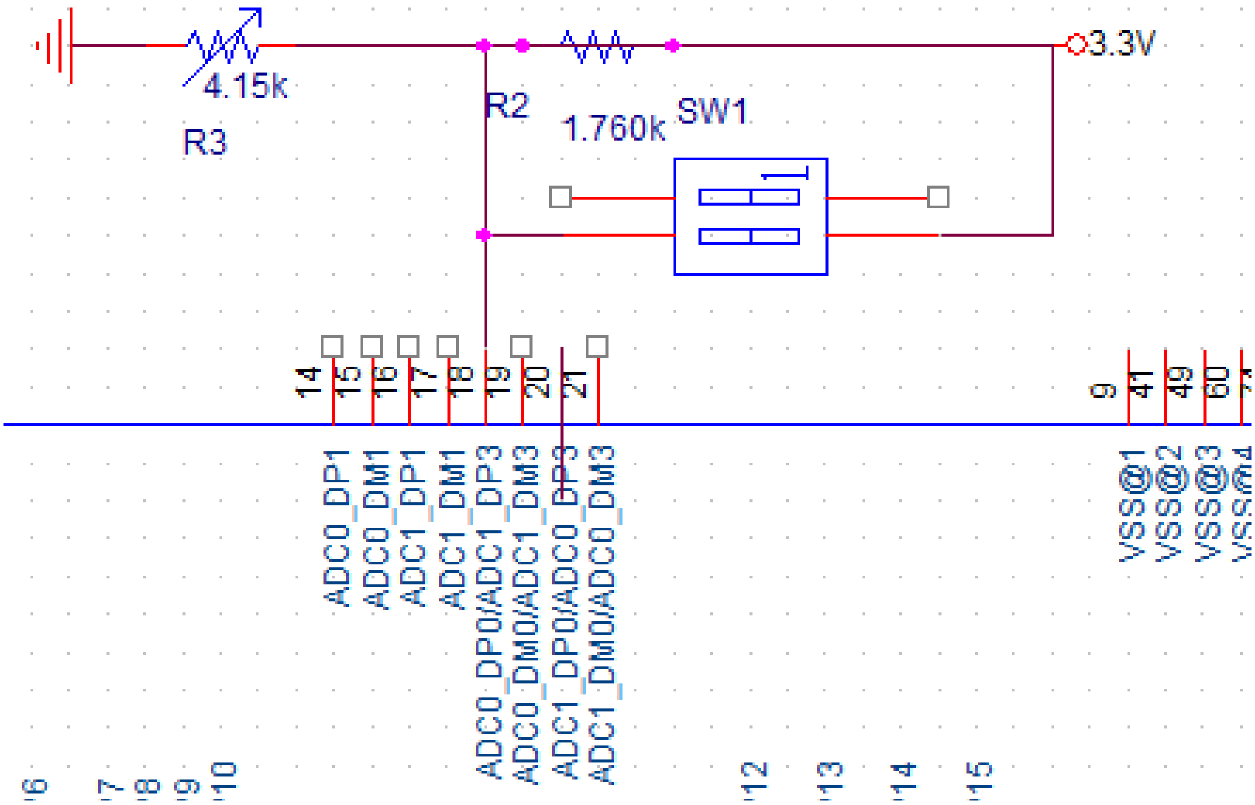
General overview



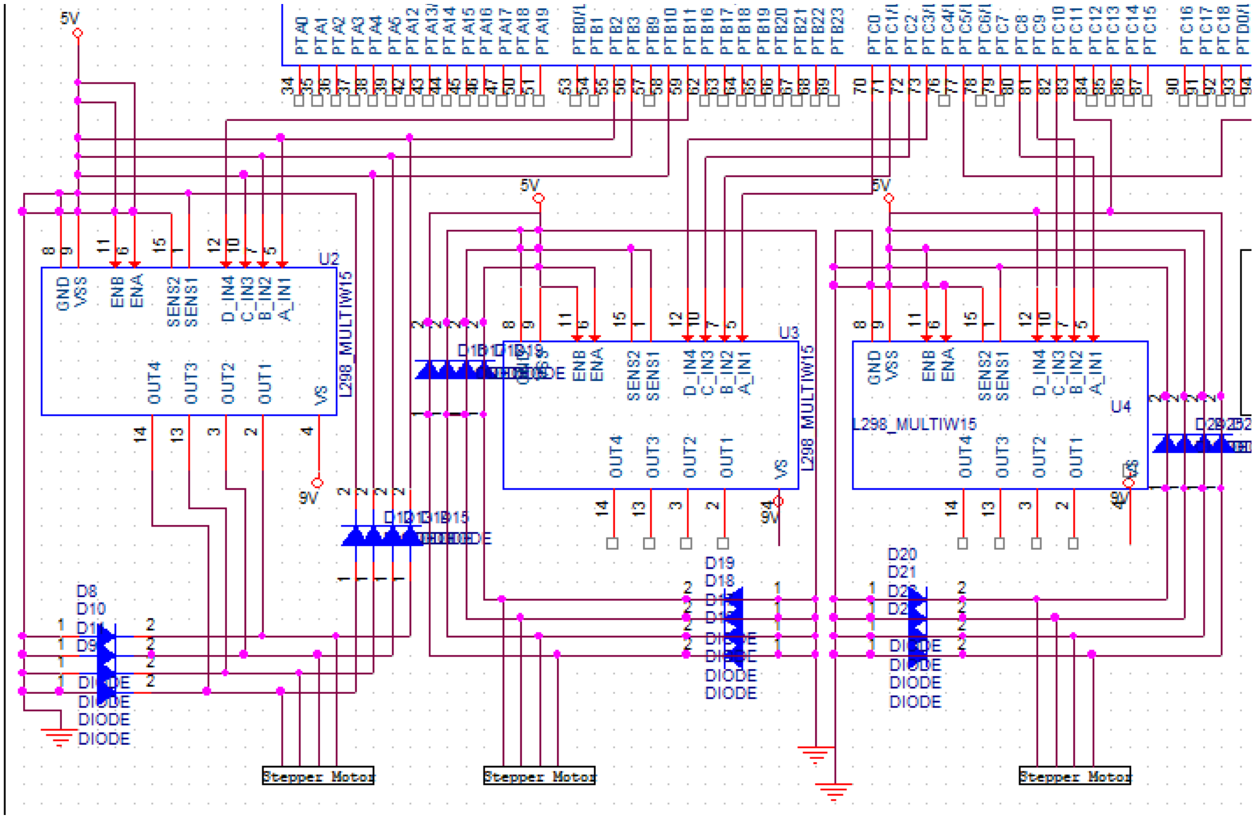
UART


PTC0
PTC1
PTC2
PTC3
PTC4
PTC5
PTC6
PTC7
PTC8
PTC9
PTD0
PTD1
PTD2
PTD3
PTD4
PTD5
PTD6
PTD7
PTE0
PTE1
PTE2
PTE3
PTE4
PTE5
PTE6
SCK
MOSI
MISO
5V
9V
GND
VSS
ENB
ENA
SENS2
SENS1
D_IN4
C_IN3
B_IN2
A_IN1
U4
L298_MULTIW15
OUT4
OUT3
OUT2
OUT1
111213
ARDUINO UNO
2
3
4
5
6
7
8
D1
D2
D3
D4
D5
D6
D7
D20
D21
D22
D23
D24
D25
D26
D27
DIODE
Stepper Motor

Joystick

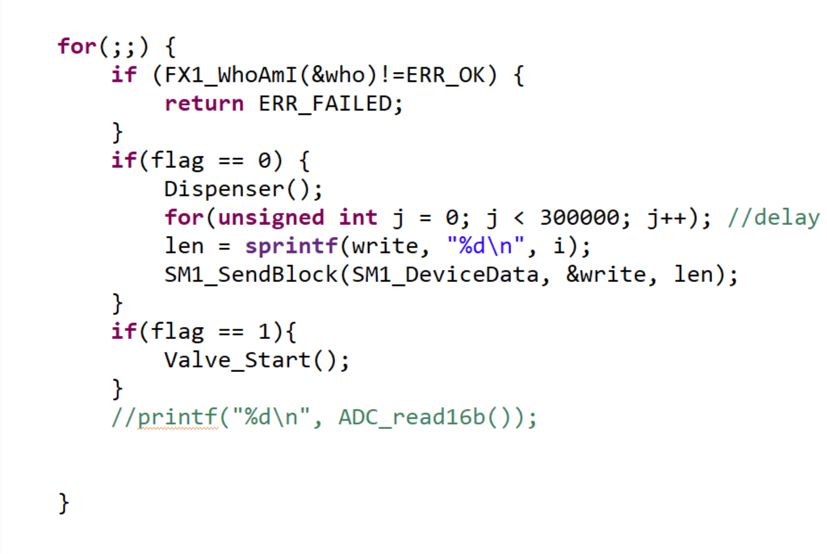

4.15k
R3
R2
1.760k
SW1
3.3V
14
15
16
17
18
19
20
21
9
41
49
60
ADC0_DP1
ADC0_DM1
ADC1_DP1
ADC1_DM1
ADC0_DP0/ADC1_DP3
ADC0_DM0/ADC1_DM3
ADC1_DP0/ADC0_DP3
ADC1_DM0/ADC0_DM3
VSS@1
VSS@2
VSS@3
'6
'7
'8
'9
'10
'12
'13
'14
'15

Stepper Motor Control



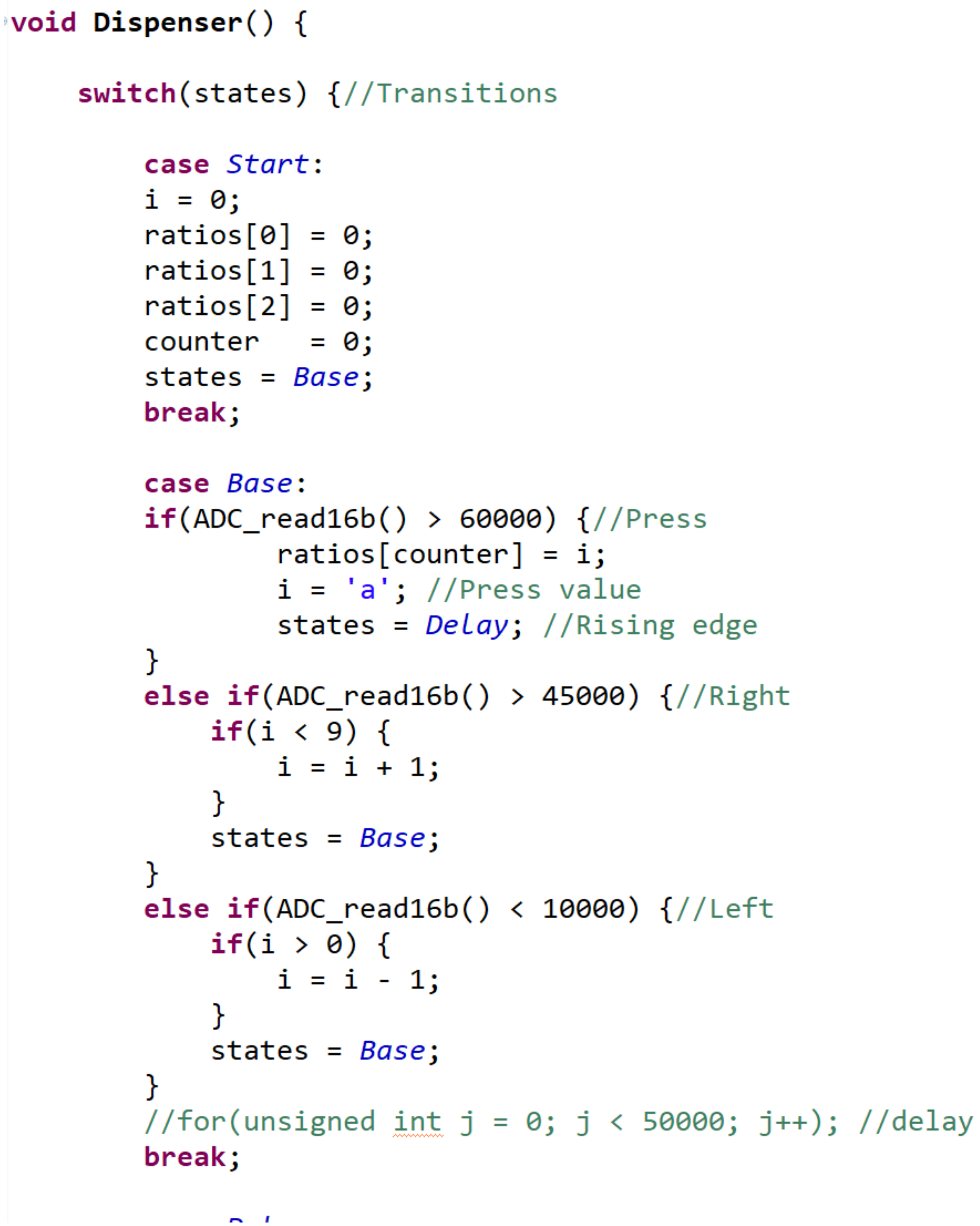
Key Portions
Source code:
K64F

This code makes sure that the Input FSM, Dispenser(), and the output FSM, Valve_Start(), run sequentially and individually. It also shows that the display data is sent to the Arduino.

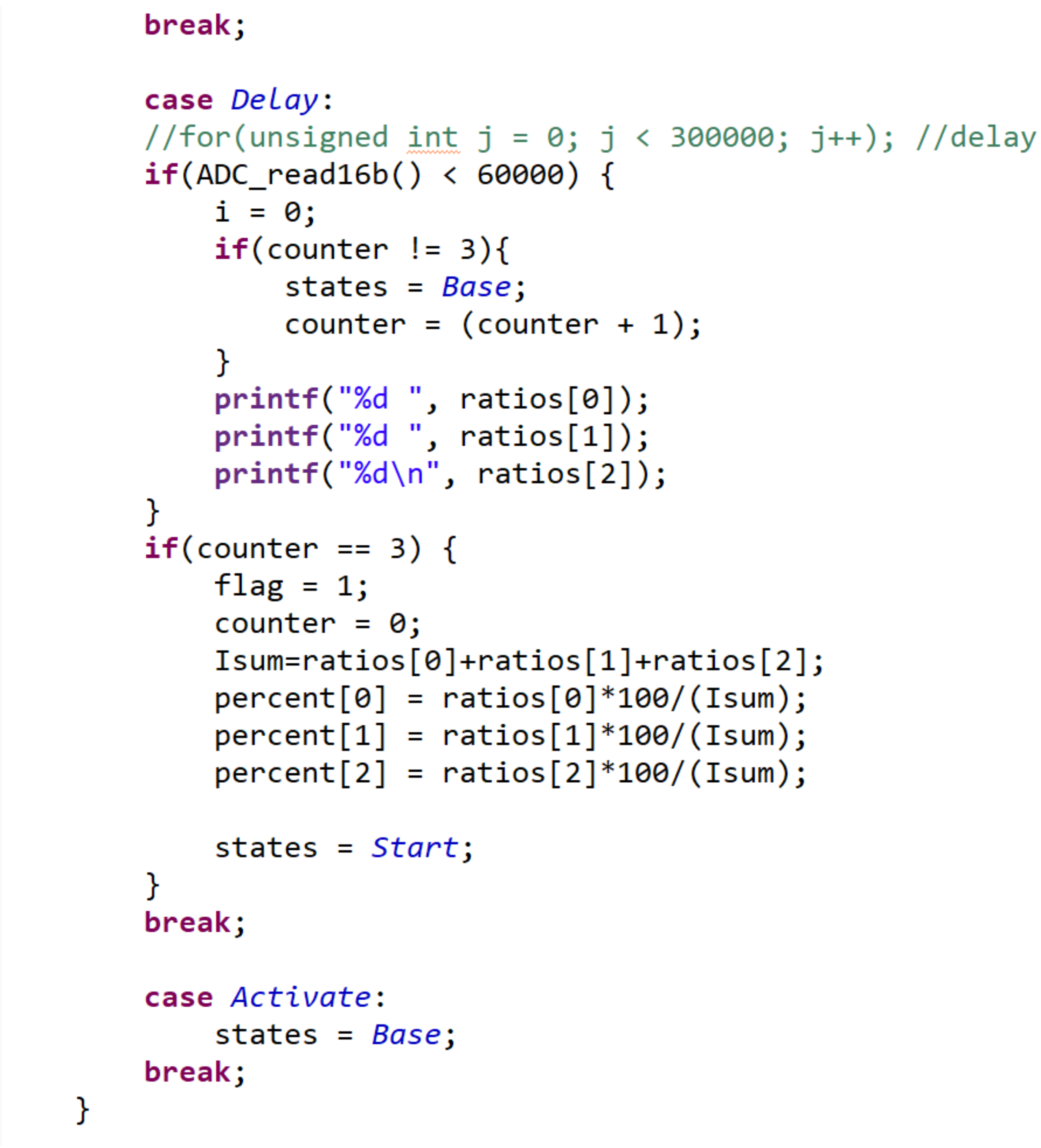


```
for(;;) {
    if (FX1_WhoAmI(&who)!=ERR_OK) {
        return ERR_FAILED;
    }
    if(flag == 0) {
        Dispenser();
        for(unsigned int j = 0; j < 300000; j++); //delay
        len = sprintf(write, "%d\n", i);
        SM1_SendBlock(SM1_DeviceData, &write, len);
    }
    if(flag == 1){
        Valve_Start();
    }
    //printf("%d\n", ADC_read16b());


}
```

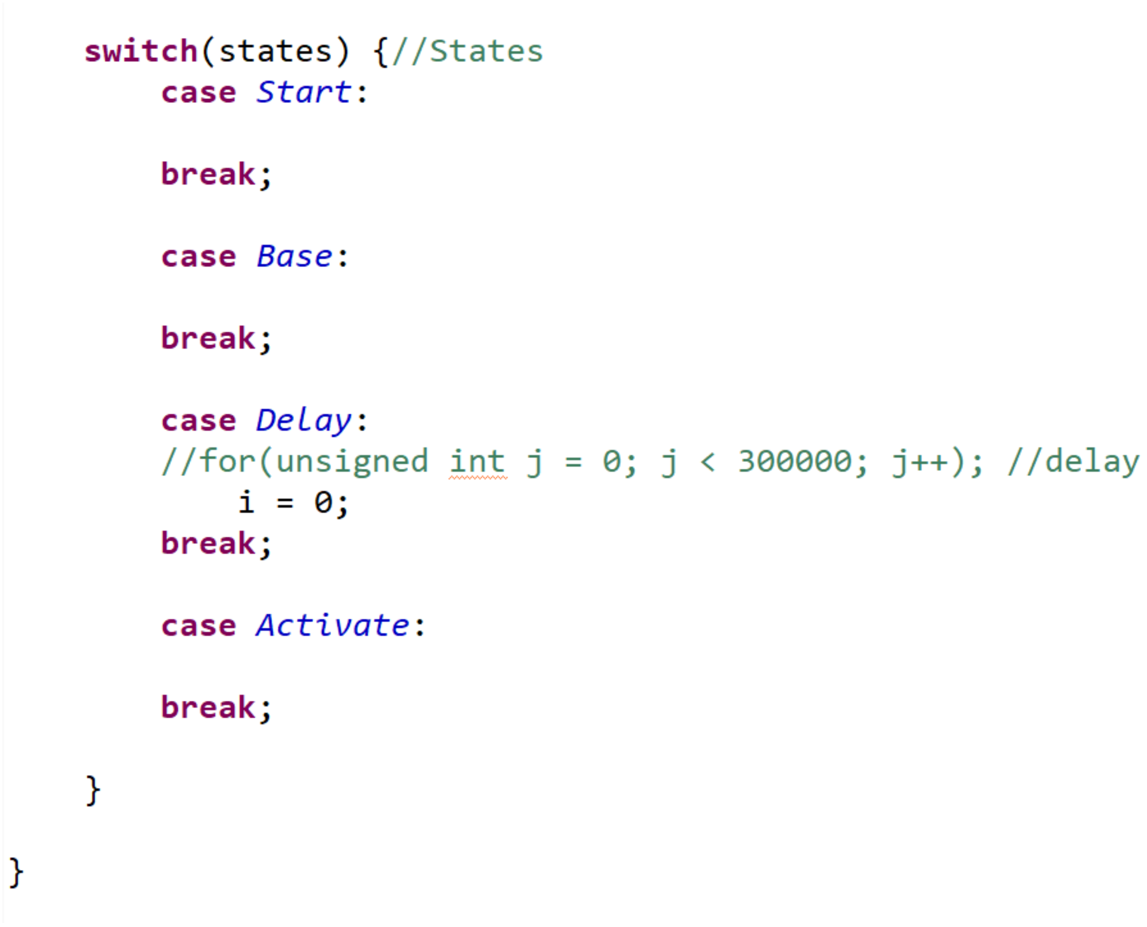
This is the input FSM. It takes input from the joystick in the form of ADC_read16b and generates an array of 3 for the output FSM. (Variable i is edited stored and displayed)



```
void Dispenser() {

    switch(states) {//Transitions

        case Start:
        i = 0;
        ratios[0] = 0;
        ratios[1] = 0;
        ratios[2] = 0;
        counter   = 0;
        states = Base;
        break;

        case Base:
        if(ADC_read16b() > 60000) {//Press
                ratios[counter] = i;
                i = 'a'; //Press value
                states = Delay; //Rising edge
        }
        else if(ADC_read16b() > 45000) {//Right
            if(i < 9) {
                i = i + 1;
            }
            states = Base;
        }
        else if(ADC_read16b() < 10000) {//Left
            if(i > 0) {
                i = i - 1;
            }
            states = Base;
        }
        //for(unsigned int j = 0; j < 50000; j++); //delay
        break;
```

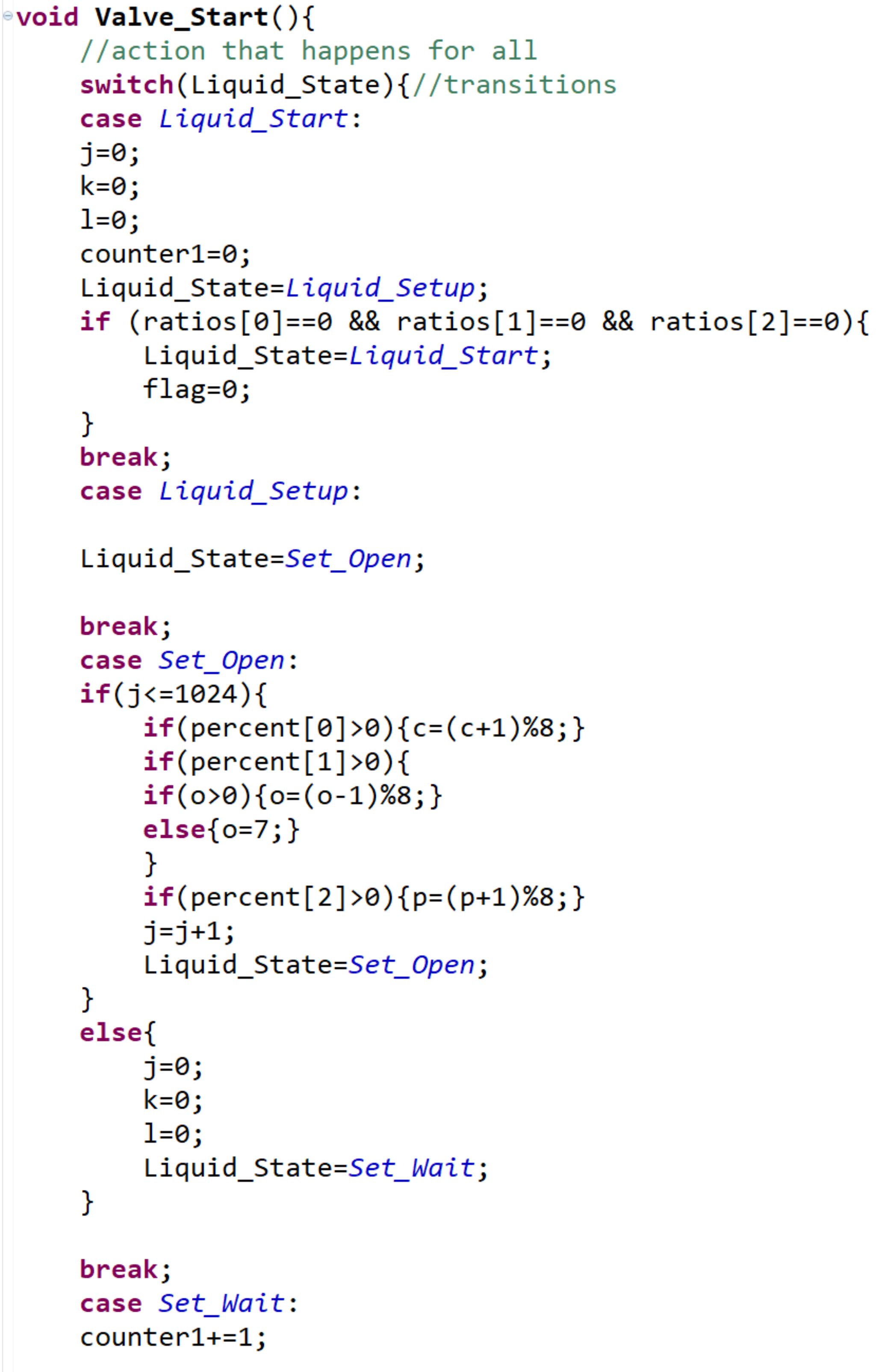


```
        break;

        case Delay:
        //for(unsigned int j = 0; j < 300000; j++); //delay
        if(ADC_read16b() < 60000) {
            i = 0;
            if(counter != 3){
                states = Base;
                counter = (counter + 1);
            }
            printf("%d ", ratios[0]);
            printf("%d ", ratios[1]);
            printf("%d\n", ratios[2]);
        }
        if(counter == 3) {
            flag = 1;
            counter = 0;
            Isum=ratios[0]+ratios[1]+ratios[2];
            percent[0] = ratios[0]*100/(Isum);
            percent[1] = ratios[1]*100/(Isum);
            percent[2] = ratios[2]*100/(Isum);

            states = Start;
        }
        break;

        case Activate:
            states = Base;
        break;
    }
```

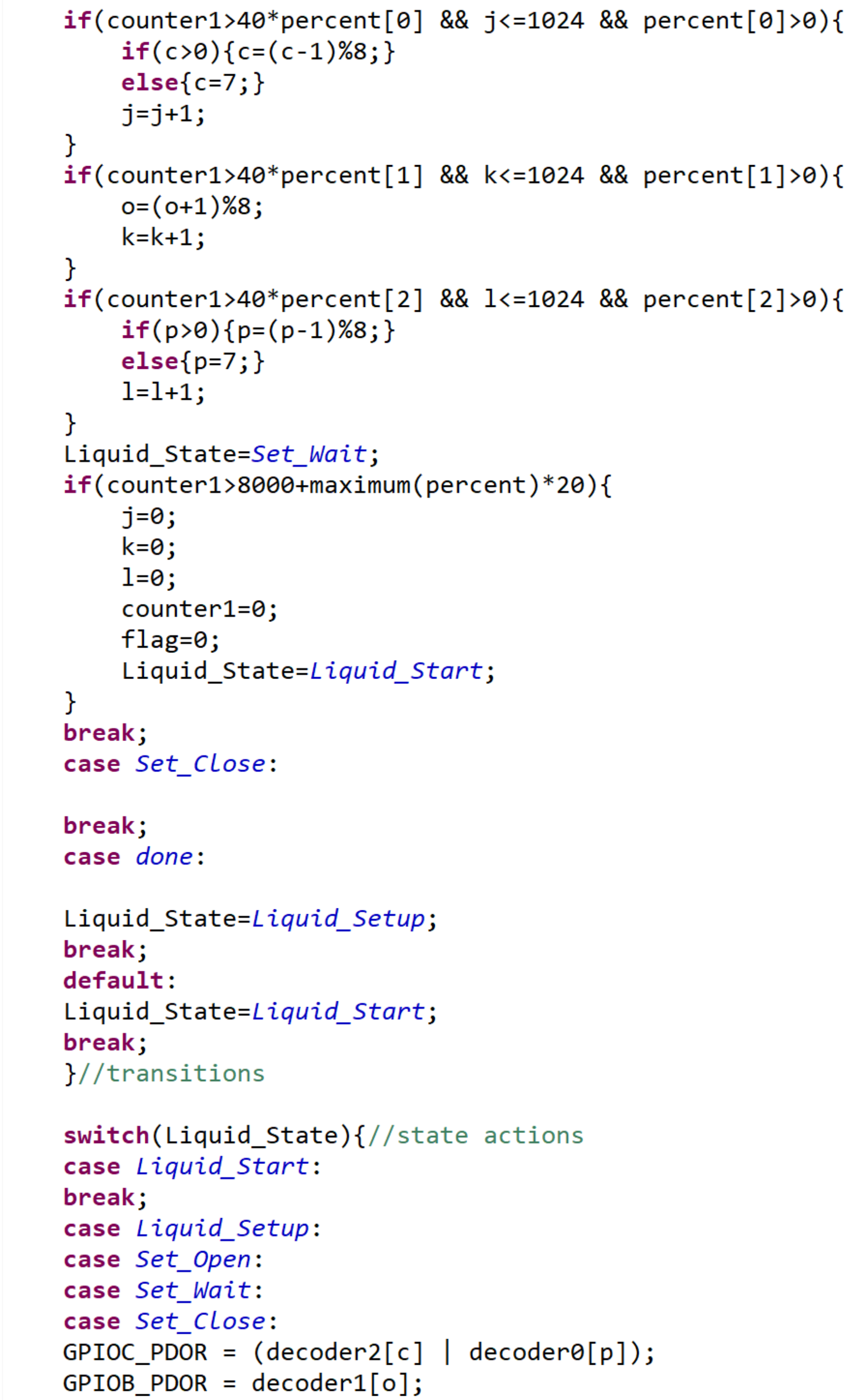


```
    switch(states) {//States
        case Start:

        break;

        case Base:

        break;

        case Delay:
        //for(unsigned int j = 0; j < 300000; j++); //delay
            i = 0;
        break;

        case Activate:

        break;

    }

}
```

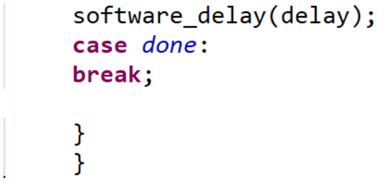
This is the output stepper motor fsm it takes in an array of 3 (ratios[]) to determine how long each valve is open.



```
void Valve_Start(){
    //action that happens for all
    switch(Liquid_State){//transitions
    case Liquid_Start:
    j=0;
    k=0;
    l=0;
    counter1=0;
    Liquid_State=Liquid_Setup;
    if (ratios[0]==0 && ratios[1]==0 && ratios[2]==0){
        Liquid_State=Liquid_Start;
        flag=0;
    }
    break;
    case Liquid_Setup:

    Liquid_State=Set_Open;

    break;
    case Set_Open:
    if(j<=1024){
        if(percent[0]>0){c=(c+1)%8;}
        if(percent[1]>0){
        if(o>0){o=(o-1)%8;}
        else{o=7;}
        }
        if(percent[2]>0){p=(p+1)%8;}
        j=j+1;
        Liquid_State=Set_Open;
    }
    else{
        j=0;
        k=0;
        l=0;
        Liquid_State=Set_Wait;
    }

    break;
    case Set_Wait:
    counter1+=1;
```



```
if(counter1>40*percent[0] && j<=1024 && percent[0]>0){
    if(c>0){c=(c-1)%8;}
    else{c=7;}
    j=j+1;
}
if(counter1>40*percent[1] && k<=1024 && percent[1]>0){
    o=(o+1)%8;
    k=k+1;
}
if(counter1>40*percent[2] && l<=1024 && percent[2]>0){
    if(p>0){p=(p-1)%8;}
    else{p=7;}
    l=l+1;
}
Liquid_State=Set_Wait;
if(counter1>8000+maximum(percent)*20){
    j=0;
    k=0;
    l=0;
    counter1=0;
    flag=0;
    Liquid_State=Liquid_Start;
}
break;
case Set_Close:

break;
case done:

Liquid_State=Liquid_Setup;
break;
default:
Liquid_State=Liquid_Start;
break;
}//transitions

switch(Liquid_State){//state actions
case Liquid_Start:
break;
case Liquid_Setup:
case Set_Open:
case Set_Wait:
case Set_Close:
GPIOC_PDOR = (decoder2[c] | decoder0[p]);
GPIOB_PDOR = decoder1[o];
```



```
software_delay(delay);
case done:
break;

}
}
```

5. Testing/Evaluation: test environment, required equipment, test scenarios.

With regards to UART, the testing involved using TeraTerm and ARDUINO IDE to see if the message was sent through. The only message that was sent through was the index i, which was used for a decoder for the seven segment display.

To test and calibrate the Joystick (potentiometer) input, a voltmeter was used to determine the range of voltage change, which was then thresholded on the software end to create the right, left and click inputs.

The Motors were tested individually for functionality, then they were tested together in their own FSM, and lastly they were tested in the final system. When They were tested in the final version the valves needed to be calibrated so that they would twist open enough to release the water and twist close to restrict the water.

6. Discussions: challenges, limitations, possible improvements

There were several challenges that were overcome in this project. The main challenge was getting UART to function consistently. This took a couple of tries and resets to get working. Another challenge was finding enough ports to use on the frdm board for the 3 stepper motors. Figuring out the best way to have the user select 3 drink ratio numbers was difficult as well.

In our original design, we had an LCD crystal display that would show drink options and would allow for a custom option, but the LCD code interfered with the UART on the arduino, so we got rid of the set drink options and only retained the custom option. We had the custom numbers displayed on a 7 Segment LED.

Here are some of the limitations. Only 3 different motors can be used because of the limited number of output pins on the frdm board and because of the limited amount of power that can be provided by the voltage generators in the lab. The UART interferes with the LCD so if one were to be used it would have to connect directly to the frdm board. The last limitation is that only a max of 2 seven segment displays can be used on the Arduino board because of limited pins.

Here are some of the improvements that could be made. The liquid flow was often inconsistent and overall not very good. This could be improved by having the liquid containers and tubes connect directly to each other allowing the system to utilize gravity to create liquid flow instead of relying on syphon energy. The delay after the system finished dispensing was inconsistent. Sometimes it was too long and required the user to wait. This could be fixed by making sure the system is ready for input after the final valve closes. Sometimes the button clicks would not register. This could be fixed by using a button press counter, a smaller delay for the button, and an on and off intermediate state.

7. Roles and Responsibilities of Group members

Peter designed the user interface and display. The display aspect included using UART to send display data to the Arduino and having the Arduino output to a seven segment LED. The user interface part used an analog to digital controller (joystick) for input and had an FSM for selecting the three stepper motor values for the dispensing phase of the project.

Grant performed the task of constructing the physical components of the drink dispenser. This includes the setup for the motors, cups, valves, and tubing. He also created the FSM for controlling the three stepper motors in the dispensing phase.

8. Conclusion

In short, we accomplished the task of designing a custom beverage dispenser, using analog to digital conversion, UART, a 7seg LED, and stepper motor control.

Code FRDM

```
/* ########################################################################
**     Filename    : main.c
**     Project     : Final Project
**     Processor   : MK64FN1M0VLL12
**     Version     : Driver 01.01
**     Compiler    : GNU C Compiler
**     Date/Time   : 2019-11-03, 17:50, # CodeGen: 0
**     Abstract    :
**         Main module.
**         This module contains user's application code.
**     Settings    :
**     Contents    :
**         No public methods
**
** ########################################################################*/
/*!
 ** @file main.c
 ** @version 01.01
 ** @brief
 **         Main module.
 **         This module contains user's application code.
 */
/*!
 **  @addtogroup main_module main module documentation
 **  @{
 */
/* MODULE main */


/* Including needed modules to compile this module/procedure */
#include "Cpu.h"
#include "Events.h"
#include "Pins1.h"
#include "FX1.h"
#include "GI2C1.h"
#include "WAIT1.h"
#include "CI2C1.h"
#include "CsIO1.h"
#include "IO1.h"
#include "MCUC1.h"
#include "SM1.h"
/* Including shared modules, which are used for whole project */
#include "PE_Types.h"
#include "PE_Error.h"
#include "PE_Const.h"
#include "IO_Map.h"
#include "PDD_Includes.h"
#include "Init_Config.h"
```

```
//#include "fsl_device_registers.h"
#include <stdint.h>

void Dispenser();
void Valve_Start();

void software_delay(unsigned long delay)
{
while (delay > 0) delay--;
}

int maximum(int arr[]){
int max=arr[1];
if(arr[0]>arr[1]){max=arr[0];}
if(arr[2]>max){max=arr[2];}
return max;
}

unsigned short ADC_read16b(void);

/* User includes (#include below this line is not maintained by Processor Expert) */

/*lint -save  -e970 Disable MISRA rule (6.3) checking. */

unsigned char write[512];
unsigned int i = 0;
unsigned int temp = 1;

enum Liquid_States {Liquid_Start, Liquid_Setup, Set_Open, Set_Wait, Set_Close, done}
Liquid_State;
enum Dispenser {Start, Base, Delay, Activate} states;

int percent[3]={50,25,25};
unsigned int ratios[3] = {0,0,0};
unsigned int counter = 0;
unsigned int flag = 0;
unsigned int lsum=0;
unsigned short c=0,o=0,p=0,j=0,k=0,l=0;
unsigned int decoder0[8] = { 0x04, 0x06, 0x02, 0x0A, 0x08, 0x09, 0x01, 0x05};
unsigned int decoder1[8] = { 0x400, 0x408, 0x008, 0x808, 0x800, 0x804, 0x004, 0x404};
unsigned int decoder2[8] = { 0x400, 0x600, 0x200, 0xA00, 0x800, 0x900, 0x100, 0x500};
uint32_t Input = 0x00, ROT_DIR1 = 0, ROT_DIR2 = 0, counter1 = 0;
unsigned long delay = 7000;


int main(void){
        PE_low_level_init();

        SIM_SCGC5 |= SIM_SCGC5_PORTC_MASK; /* Enable Port C Clock Gate Control*/
        SIM_SCGC5 |= SIM_SCGC5_PORTB_MASK; /* Enable Port B Clock Gate Control*/
```

```
    PORTC_GPCLR = 0x00F0F0100; /* Configures Pins 0-3 and 8-11 on Port C to be GPIO
*/
    PORTB_GPCLR = 0x00C0C0100; /* Configures Pins 0-3 and 8-11 on Port B to be GPIO
*/

    GPIOC_PDDR = 0x000000F0F; /* Configures Pins 0-3 and 8-11 on Port C to be Output
*/
    GPIOB_PDDR = 0x000000C0C; /* Configures Pins 0-3 and 8-11 on Port B to be Output
*/




    /* Write your local variable definition here */

    /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/

    /*** End of Processor Expert internal initialization.                  ***/
    /* Write your code here */
    uint8_t who;

    int len;
    LDD_TDeviceData *SM1_DeviceData;
    SM1_DeviceData = SM1_Init(NULL);


    FX1_Init();

    SIM_SCGC6 |= SIM_SCGC6_ADC0_MASK; // 0x8000000u; Enable ADC0 Clock
    ADC0_CFG1 = 0x0C; // 16bits ADC; Bus Clock
    ADC0_SC1A = 0x1F; // Disable the module, ADCH = 11111


    for(;;) {
        if (FX1_WhoAmI(&who)!=ERR_OK) {
            return ERR_FAILED;
        }
        if(flag == 0) {
            Dispenser();
            for(unsigned int j = 0; j < 300000; j++); //delay
            len = sprintf(write, "%d\n", i);
            SM1_SendBlock(SM1_DeviceData, &write, len);
        }
        if(flag == 1){
            Valve_Start();
        }
        //printf("%d\n", ADC_read16b());


    }
```

```
      /* For example: for(;;) { } */

      /*** Don't write any code pass this line, or it will be deleted during code generation. ***/
  /*** RTOS startup code. Macro PEX_RTOS_START is defined by the RTOS component.
DON'T MODIFY THIS CODE!!! ***/
  #ifdef PEX_RTOS_START
    PEX_RTOS_START();                  /* Startup of the selected RTOS. Macro is defined by the
RTOS component. */
  #endif
  /*** End of RTOS startup code.  ***/
  /*** Processor Expert end of main routine. DON'T MODIFY THIS CODE!!! ***/
  for(;;){}
  /*** Processor Expert end of main routine. DON'T WRITE CODE BELOW!!! ***/
} /*** End of main routine. DO NOT MODIFY THIS TEXT!!! ***/

unsigned short ADC_read16b(void) {
      ADC0_SC1A = 0x00; //Write to SC1A to start conversion from ADC_0
      while(ADC0_SC2 & ADC_SC2_ADACT_MASK); // Conversion in progress
      while(!(ADC0_SC1A & ADC_SC1_COCO_MASK)); // Until conversion complete
      return ADC0_RA;

}

void Dispenser() {

      switch(states) {//Transitions

            case Start:
            i = 0;
            ratios[0] = 0;
            ratios[1] = 0;
            ratios[2] = 0;
            counter   = 0;
            states = Base;
            break;

            case Base:
            if(ADC_read16b() > 60000) {//Press
                        ratios[counter] = i;
                        i = 'a'; //Press value
                        states = Delay; //Rising edge
            }
            else if(ADC_read16b() > 45000) {//Right
                  if(i < 9) {
                        i = i + 1;
                  }
                  states = Base;
            }
            else if(ADC_read16b() < 10000) {//Left
```

```
        if(i > 0) {
            i = i - 1;
        }
        states = Base;
    }
    //for(unsigned int j = 0; j < 50000; j++); //delay
    break;

    case Delay:
    //for(unsigned int j = 0; j < 300000; j++); //delay
    if(ADC_read16b() < 60000) {
        i = 0;
        if(counter != 3){
            states = Base;
            counter = (counter + 1);
        }
        printf("%d ", ratios[0]);
        printf("%d ", ratios[1]);
        printf("%d\n", ratios[2]);
    }
    if(counter == 3) {
        flag = 1;
        counter = 0;
        Isum=ratios[0]+ratios[1]+ratios[2];
        percent[0] = ratios[0]*100/(Isum);
        percent[1] = ratios[1]*100/(Isum);
        percent[2] = ratios[2]*100/(Isum);

        states = Start;
    }
    break;

    case Activate:
        states = Base;
    break;
}

switch(states) {//States
    case Start:

    break;

    case Base:

    break;

    case Delay:
    //for(unsigned int j = 0; j < 300000; j++); //delay
        i = 0;
    break;
```

```
        case Activate:

        break;

    }

}

void Valve_Start(){
    //action that happens for all
    switch(Liquid_State){//transitions
    case Liquid_Start:
    j=0;
    k=0;
    l=0;
    counter1=0;
    Liquid_State=Liquid_Setup;
    if (ratios[0]==0 && ratios[1]==0 && ratios[2]==0){
        Liquid_State=Liquid_Start;
        flag=0;
    }
    break;
    case Liquid_Setup:

    Liquid_State=Set_Open;

    break;
    case Set_Open:
    if(j<=1024){
        if(percent[0]>0){c=(c+1)%8;}
        if(percent[1]>0){
        if(o>0){o=(o-1)%8;}
        else{o=7;}
        }
        if(percent[2]>0){p=(p+1)%8;}
        j=j+1;
        Liquid_State=Set_Open;
    }
    else{
        j=0;
        k=0;
        l=0;
        Liquid_State=Set_Wait;
    }

    break;
    case Set_Wait:
    counter1+=1;

    if(counter1>40*percent[0] && j<=1024 && percent[0]>0){
        if(c>0){c=(c-1)%8;}
        else{c=7;}
        j=j+1;
}
if(counter1>40*percent[1] && k<=1024 && percent[1]>0){
        o=(o+1)%8;
        k=k+1;
}
if(counter1>40*percent[2] && l<=1024 && percent[2]>0){
        if(p>0){p=(p-1)%8;}
        else{p=7;}
        l=l+1;
}
Liquid_State=Set_Wait;
if(counter1>8000+maximum(percent)*20){
        j=0;
        k=0;
        l=0;
        counter1=0;
        flag=0;
        Liquid_State=Liquid_Start;
}
break;
case Set_Close:

break;
case done:

Liquid_State=Liquid_Setup;
break;
default:
Liquid_State=Liquid_Start;
break;
}//transitions

switch(Liquid_State){//state actions
case Liquid_Start:
break;
case Liquid_Setup:
case Set_Open:
case Set_Wait:
case Set_Close:
GPIOC_PDOR = (decoder2[c] | decoder0[p]);
GPIOB_PDOR = decoder1[o];
software_delay(delay);
case done:
break;


}
}
```

Code Arduino

```
#include <SPI.h>
char buff [255];
volatile byte indx;
volatile boolean process;
unsigned int i = 0;
unsigned int j = 0;
void setup (void) {
   pinMode(8, OUTPUT);
   pinMode(7, OUTPUT);
   pinMode(6, OUTPUT);
   pinMode(5, OUTPUT);
   pinMode(4, OUTPUT);
   pinMode(3, OUTPUT);
   pinMode(2, OUTPUT);
   pinMode(1, OUTPUT);
   Serial.begin (115200);
   pinMode(MISO, OUTPUT); // have to send on master in so it set as output
   SPCR |= _BV(SPE); // turn on SPI in slave mode
   indx = 0; // buffer empty
   process = false;
   SPI.attachInterrupt(); // turn on interrupt
}

ISR (SPI_STC_vect) // SPI interrupt routine
{
   byte c = SPDR; // read byte from SPI Data Register

   if (indx < sizeof(buff)) {
      buff[indx++] = c; // save data in the next index in the array buff
      if (c == '\n') {
        buff[indx - 1] = 0; // replace newline ('\n') with end of string
(0)
        process = true;
      }
   }
}

void loop (void) {
   if (process) {
      process = false; //reset the process
      Serial.println (buff); //print the array on serial monitor
      indx= 0; //reset button to zero
      display(buff[0] - 48);
   }
}
```

```
void display(int number) {
        if(number == 0) {
            digitalWrite(2, LOW);
            digitalWrite(3, HIGH);
            digitalWrite(4, HIGH);
            digitalWrite(5, HIGH);
            digitalWrite(6, HIGH);
            digitalWrite(7, HIGH);
            digitalWrite(8, HIGH);
       }
       else if(number == 1) {
            digitalWrite(2, LOW);
            digitalWrite(3, LOW);
            digitalWrite(4, HIGH);
            digitalWrite(5, LOW);
            digitalWrite(6, LOW);
            digitalWrite(7, HIGH);
            digitalWrite(8, LOW);
       }
       else if(number == 2) {
            digitalWrite(2, HIGH);
            digitalWrite(3, HIGH);
            digitalWrite(4, HIGH);
            digitalWrite(5, HIGH);
            digitalWrite(6, HIGH);
            digitalWrite(7, LOW);
            digitalWrite(8, LOW);
       }
       else if(number == 3) {
            digitalWrite(2, HIGH);
            digitalWrite(3, HIGH);
            digitalWrite(4, HIGH);
            digitalWrite(5, LOW);
            digitalWrite(6, HIGH);
            digitalWrite(7, HIGH);
            digitalWrite(8, LOW);
       }
       else if(number == 4) {
            digitalWrite(2, HIGH);
            digitalWrite(3, LOW);
            digitalWrite(4, HIGH);
            digitalWrite(5, LOW);
            digitalWrite(6, LOW);
            digitalWrite(7, HIGH);
            digitalWrite(8, HIGH);
       }
       else if(number == 5) {
            digitalWrite(2, HIGH);
            digitalWrite(3, HIGH);
            digitalWrite(4, LOW);
            digitalWrite(5, LOW);
            digitalWrite(6, HIGH);
            digitalWrite(7, HIGH);
            digitalWrite(8, HIGH);
       }
       else if(number == 6) {
            digitalWrite(2, HIGH);
            digitalWrite(3, HIGH);
            digitalWrite(4, LOW);
            digitalWrite(5, HIGH);
            digitalWrite(6, HIGH);
            digitalWrite(7, HIGH);
            digitalWrite(8, HIGH);
       }
       else if(number == 7) {
            digitalWrite(2, LOW);
            digitalWrite(3, HIGH);
            digitalWrite(4, HIGH);
            digitalWrite(5, LOW);
            digitalWrite(6, LOW);
            digitalWrite(7, HIGH);
            digitalWrite(8, LOW);
       }
       else if(number == 8) {
            digitalWrite(2, HIGH);
            digitalWrite(3, HIGH);
            digitalWrite(4, HIGH);
            digitalWrite(5, HIGH);
            digitalWrite(6, HIGH);
            digitalWrite(7, HIGH);
            digitalWrite(8, HIGH);
       }
       else if(number == 9) {
            digitalWrite(2, HIGH);
            digitalWrite(3, HIGH);
            digitalWrite(4, HIGH);
            digitalWrite(5, LOW);
            digitalWrite(6, HIGH);
            digitalWrite(7, HIGH);
            digitalWrite(8, HIGH);
       }

}
```