Midterm Project
4-bit calculator Program

EGc332– Microcontrollers

March 28, 2021

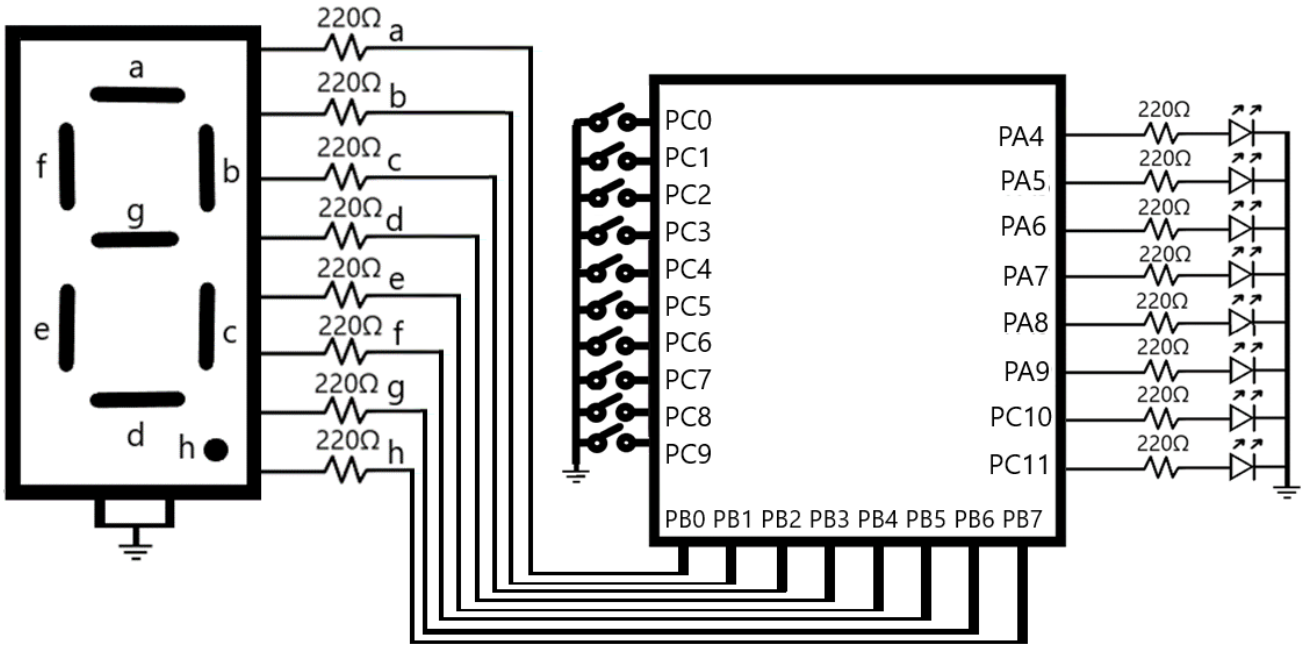
Christopher Lepore

Introduction

In this project we designed a 4-bit calculator that does the add, subtract, AND, and OR operation using a Nucleo-64 microcontroller. This introduced us to coding with the add and subtraction operations in C programing. This lab also introduced sorting and coding bit numbers to fit the desired input or output ports of the STM32F446RE circuit board. The 4-bit calculator cycles through determining the selector's value and doing the operation associated with that value. If the selector is 0 then the AND operation is done, for 1 it is the OR operation, for 2 it adds, and for 3 it does subtraction. This program has 2 selector bits, 4 bits for number A, and 4 bits for number B. The program outputs a 4-bit number to 4 LEDs and a 7-segment display and has a zero, carry, sign, and overflow flag. By designing this calculator, a deeper comprehension of bit sorting with the STM32F446RE circuit board was achieved.

Design

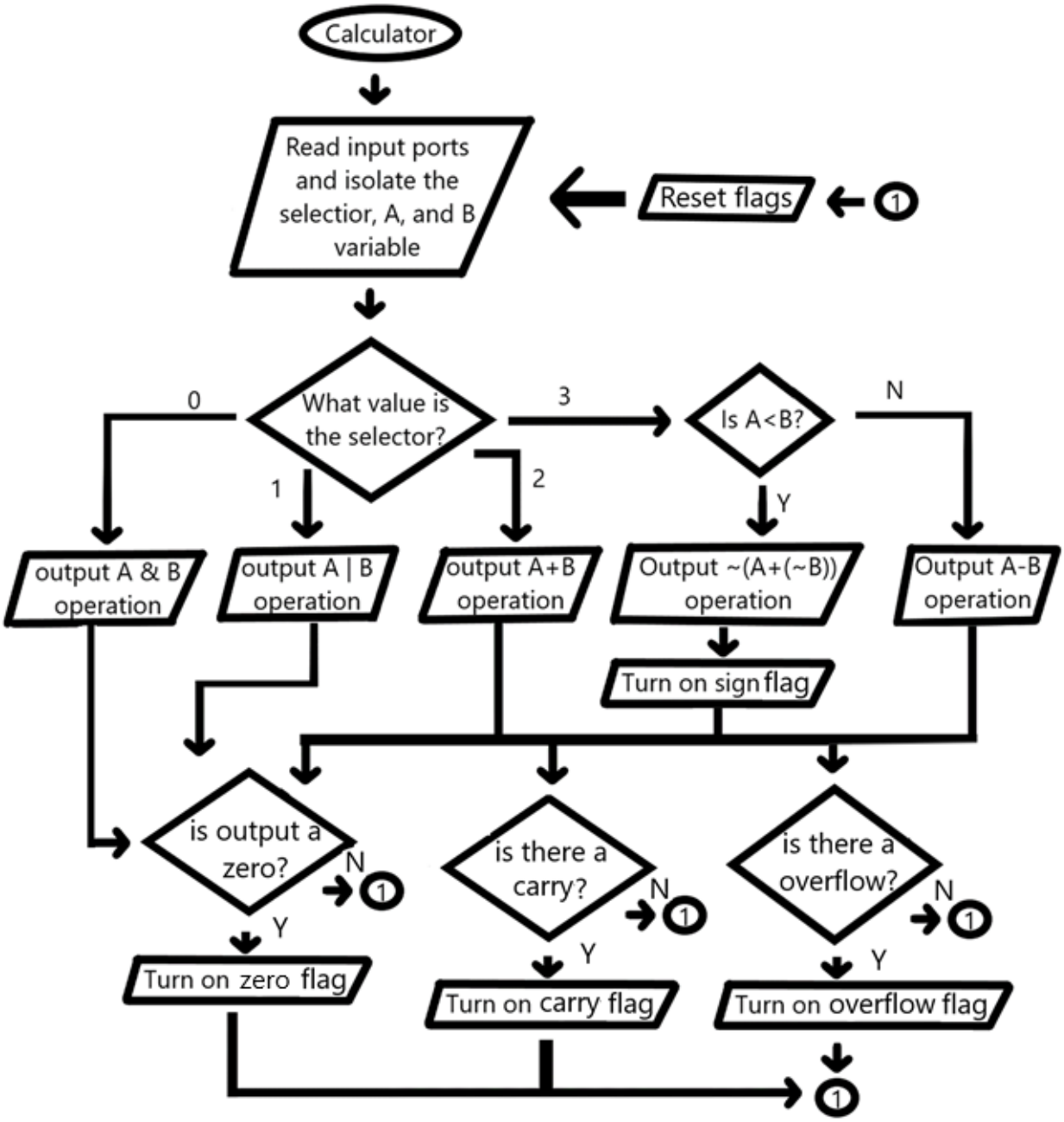
Hardware (Block Diagrams)



For the calculator circuit, port PA4 to PA9 and PC10 to PC11 were set to output mode and then each connected to a light emitting diode (LED). Port PA4 to PA7 represent the 4-bit output with PA7 as the most significant bit. For the flags, PA8 is the carry flag, PA9 is the sign flag, PC10 is the zero flag, and PC11 is the overflow flag. There are 220-ohm resistors in between them to prevent the LED

from burning out. Also, port PB0 to PB7 were set to output mode and then each connected to the seven-segment display from a to h accordingly. The common of the display are connected to ground. There are 220-ohm resistors in between them to prevent the LEDs of the display from burning out. Port PC0 to PC9 were set to input mode were PC1 and PC0 are selectors, PC5 – PC2 are number A, and PC9 – PC6 are number B.

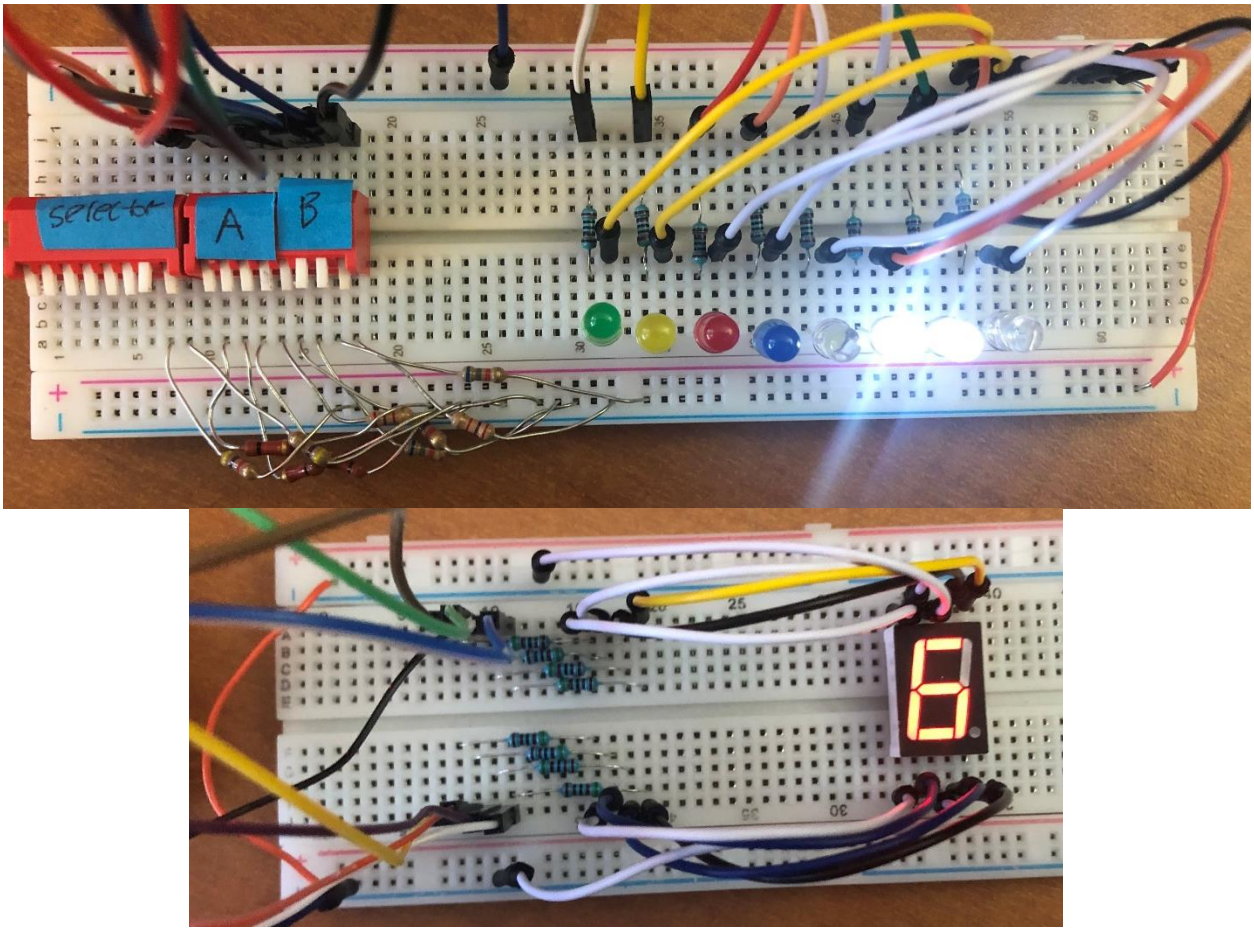
Software (Flow Charts)



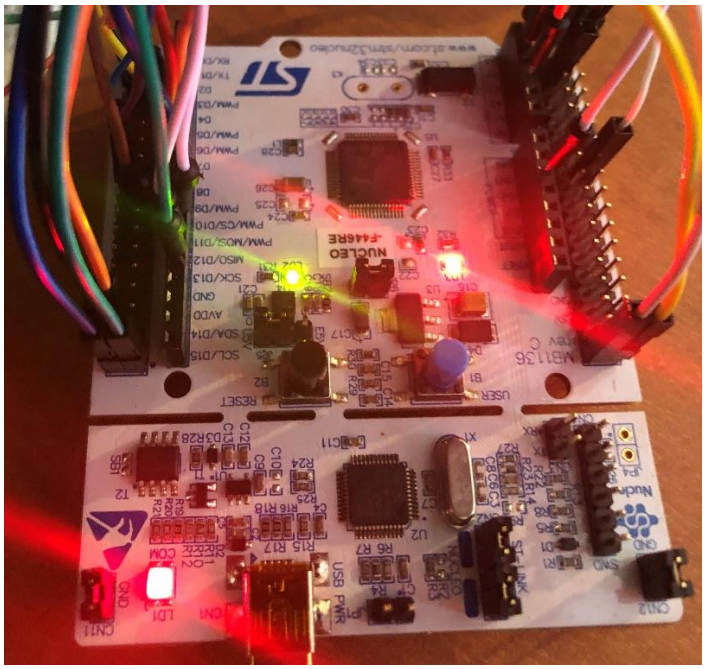
In the calculator program the code will cycle reading the input ports and isolating the selector, A number, and B number into their own variable. Then it will check the selector's value. If the selector is 0, then the AND operation will be outputted to the 7-segment display and LEDs, if it is 1 then the OR operation, if 2 then the ADD operation, and if 3 then the subtraction operation. For the subtraction operation it will check if A is less then B, if it is not then it will do simple subtraction but if it is then it will add A with the compliment of B and then complement the answer. After the operation, based on the output value the zero, carry, sign, and overflow flags will be set.

Verification

Nucleo and breadboard circuit



Breadboard wiring



NUCLEO circuit board

Summary and conclusion

In the 4-bit calculator circuit, port PC1 and PC0 were set as the operation selector where 0=AND, 1=OR, 2=ADD, and 3=Subtraction operation. Also, PC5 to PC2 are set to the A number and PC9 to PC6 are the B number. Port PA4 to PA7 represent the 4-bit output with PA7 as the most significant bit. Also, PA8 is the carry flag, PA9 is the sign flag, PC10 is the zero flag, and PC11 in the overflow flag. The code will cycle through checking the selector's value and doing the operation associated with it. For the subtraction operation the code will check if A is less than B, if it is not then the code will do simple subtraction but if it is then it will add A with the compliment of B and then complement the answer. The answer will then be outputted, and the sign flag will turn on. To make the output bits line up with its port the output was shifted to the left by 4 and then outputted to the LEDs. The non-shifted 4-bit output value was also used as an array pointer which outputted the array's value to the 7-segment display. The flags were set according to the operation and output value.

This project introduced sorting and coding bit numbers to fit the desired input or output ports of the STM32F446RE circuit board along with coding with the add and subtraction operations in C programing. This was done by analyzing the schematic to learn how to make the output bits line up with their ports. Due to the lessons above this project has prepared students to begin designing, coding, and wiring more complex projects with the operations used.

Appendix (Programs)

```
#include "stm32f4xx.h"

int main(void)
{
   RCC->AHB1ENR =  7;              /* enable GPIOA, GPIOB and GPIOC clock */

   GPIOC->PUPDR =  0x00055555;      /* set pin to input mode          PC 0 - 9 */
   GPIOC->MODER =  0x00500000;     /* set pin to output mode         PC 10 - 11 */
   GPIOA->MODER =  0x00055500;      /* set pin to output mode         PA 4 - 9 */
   GPIOB->MODER =  0x00005555;      /* set pin to output mode         PB 0 - 7 */

    while(1)
    {
        unsigned int selector = (GPIOC->IDR  & 0x03);
        int A = ((GPIOC->IDR >> 2) & 0x0F);
        int B = ((GPIOC->IDR >> 6) & 0x0F);

        int C0 = (((A & 0x1)+(B & 0x1)) & 0x2);
        int C1 = ((((A & 0x2)+(B & 0x2) + C0)) & 0x4);
        int C2 = ((((A & 0x4)+(B & 0x4) + C1)) & 0x8);
        int C3 = ((((A & 0x8)+(B & 0x8) + C2)) & 0x10);
        int Y = (C2 >> 3);
        int X = (C3 >> 4);
        int V = X ^ Y;

        int i;
        unsigned int lookup[16] =
{0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F,0x77,0x7C,0x39,0x5E,0x79,0x71};

        switch(selector)
        {
           case 0:                                                  // A & B
               GPIOA->ODR = (((A & B) << 4) & 0x03F0);
               if((A & B) == 0) GPIOC->ODR |= 0x400; else GPIOC->ODR &= 0xBFF;// z flag
               GPIOC->ODR &= 0x7FF;                                 //V flag shut off
               i = ((A & B) & 0x0F);
               GPIOB->ODR = lookup[i];
               break;
           case 1:                                                  // A | B
               GPIOA->ODR = (((A | B) << 4) & 0x03F0);
               if((A | B) == 0) GPIOC->ODR |= 0x400; else GPIOC->ODR &= 0xBFF; // z flag
               GPIOC->ODR &= 0x7FF;                                 //V flag shut off
               i = ((A | B) & 0x0F);
```

```
            GPIOB->ODR = lookup[i];
            break;
        case 2:                                                     // A + B
            GPIOA->ODR = (((A + B) << 4) & 0x03F0);
            if((A + B) == 0) GPIOC->ODR |= 0x400; else GPIOC->ODR &= 0xBFF; // z flag
            if(V == 1) GPIOC->ODR |= 0x800; else GPIOC->ODR &= 0x7FF;       // V flag
            i = ((A + B) & 0x0F);
           GPIOB->ODR = lookup[i];
           break;
        case 3:                                                     // A - B
           if(A < B)
           {
              GPIOA->ODR = ((((~(A + ~B)) << 4) & 0x03F0) | 0x0200);
              if( (~(A + ~B))== 0) GPIOC->ODR |= 0x400; else GPIOC->ODR &= 0xBFF;
                                                                                  //z flag
              if(V == 1) GPIOC->ODR |= 0x800; else GPIOC->ODR &= 0x7FF;  // V flag
              i = ((~(A + ~B)) & 0x0F);
              GPIOB->ODR = lookup[i];
           }else
           {
              GPIOA->ODR = (((A - B) << 4) & 0x03F0);
              if((A - B) == 0) GPIOC->ODR |= 0x400; else GPIOC->ODR &= 0xBFF; // z flag
              if(V == 1) GPIOC->ODR |= 0x800; else GPIOC->ODR &= 0x7FF;       //V flag
              i = ((A - B) & 0x0F);
              GPIOB->ODR = lookup[i];
           }
           break;
     }
   }
}
```