Arithmetic Logic Unit with Simulated EEPROM

ECE 251A

Camille Chow and Ostap Voynarovski

**ABSTRACT**

The goal of this project was to breadboard an ALU that performed four operations: SUM, AND, OR, and RESET. These operations are performed by the ALU utilizing multiplexers, logic gates, and D type flip flops. A EFM8BB3 microcontroller was used to simulate the EEPROM of a computer by controlling the timing element and data input to the ALU.

**EFM8 CODE** (see appendix for reference)

Our code utilized three main functions in order to have the microcontroller simulate the EEPROM: setOpPins, setDataPins, and clockPulse.

setOpPins takes an int representing one of the ALU operations (1 = SUM, 2 = AND, 3 = OR, and 4 = RESET) as a parameter, and sets GPIO pins 0.0 and 0.1 on the EFM board to the corresponding address for the ALU’s multiplexer.

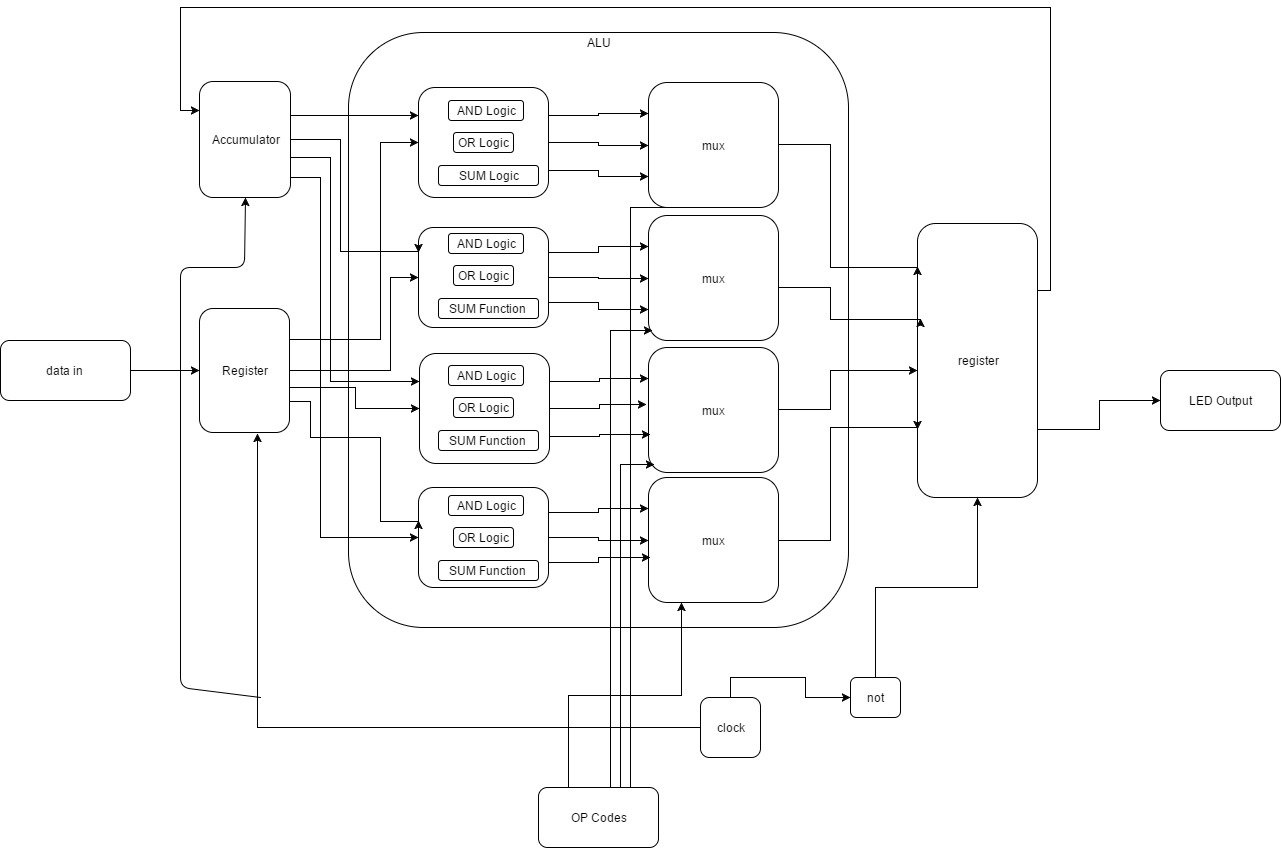
setDataPins takes in a decimal number from 0 to 15, converts it to binary and sets GPIO pins 0.4-7 on the board to the appropriate binary digits, pin 0.4 representing the least significant bit. These data pins will be fed into the D type flip flops of the ALU for data input.

clockPulse is a void function that simulates a square wave pulse utilizing an empty for loop as a constant time delay. This clock pulse is sent to pin 1.0 on the board, which is then used to clock the D type flip flops on the ALU.

In the main function, the board initializes by setting the operation to RESET and the input data to 1 (0001 in binary) and then pulsing the clock to send to the registers on the ALU. The op code is then set to SUM.

The rest of the main function is a while loop that waits for push button 0 on the EMF8 to be pressed, and once this button is pressed, clockPulse is called 15 times, summing the data input (0001) and the contents of the accumulator 15 times. This is effectively counting from 0 to 15 in binary. A delay is called after the clock pulses in order to prevent multiple calls to the contents of the if statement if the push button is held down for too long (because humans are slow :P).

**ALU**

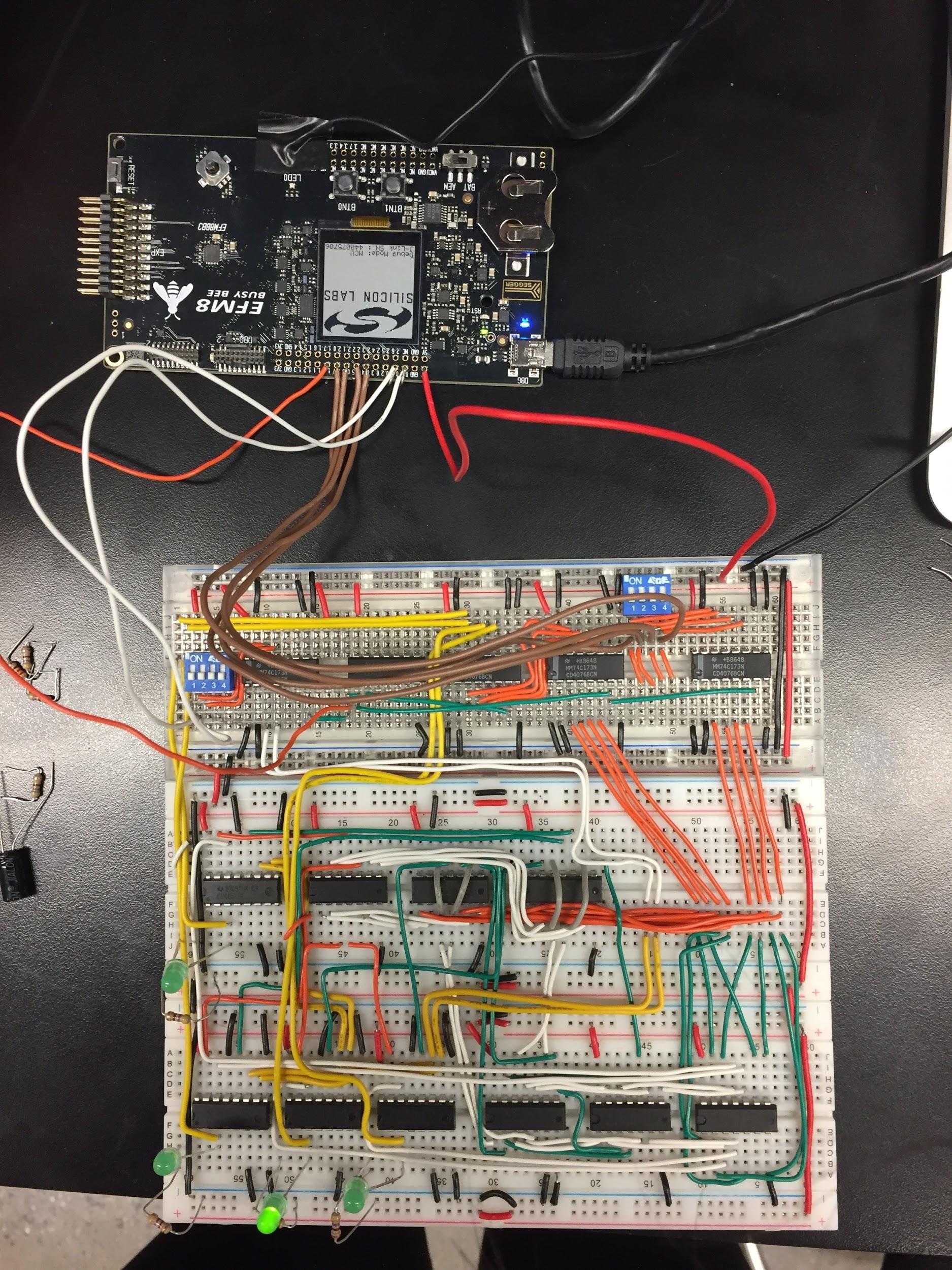


*Figure 1 - Functional block diagram of our ALU*

Our ALU, pictured above, takes in a 4 bit data input stored in a register and a 2 bit op code. The op code is used as the address for the 4 multiplexers (one for each bit). The data inputs are operated on bitwise by logic gates and fed into the input channels of the multiplexers, whose outputs are sent to an output register connected to output LEDs. The accumulator, input, and op code registers are clocked on the same cycle, while the output register is clocked on the opposite cycle. When implementing the EFM8, the input data, op codes, and clock pulse are provided by pins on the microcontroller.

**RESULTS**

Our code was able to be compiled and loaded onto our EMF8, and the proper pins were routed to the appropriate inputs on the breadboarded ALU, as pictured below:



*Figure 2 - ALU connected to EFM8, data pins are connected via the brown wires, op codes connected via the white wires, and clock pulse connected via the orange wire*

One issue we encountered and were unable to resolve was the output of GPIO pin 0.3, which was meant to represent the 2nd bit of the data input to the ALU. When the pin was supposed to be high, it would output the correct 3.3V, however when it was supposed to be low it would output 2.8V. We attempted to resolve this by simply outputting the 2nd bit to a different pin, however this only seemed to create further issues. This is most likely a hardware issue, and it is something that we are still currently trying to resolve.

**APPENDIX - EEPROM CODE**

//-----------------------------------------------------------------------------

// Includes

//-----------------------------------------------------------------------------

#include <SI\_EFM8BB3\_Register\_Enums.h> // SFR declarations

#include "InitDevice.h"

#include <string.h>

extern void WDT\_0\_enter\_DefaultMode\_from\_RESET(void) {

// $[WDTCN - Watchdog Timer Control]

SFRPAGE = 0x00;

//Disable Watchdog with key sequence

WDTCN = 0xDE; //First key

WDTCN = 0xAD; //Second key

// [WDTCN - Watchdog Timer Control]$

}

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

// Supporting Functions

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

const int DELAY = 25000;

void setOpPins(int i) {

if (i == 1) { //sum

P0\_B0 = 1;

P0\_B1 = 1;

}

else if (i == 2) { //or

P0\_B0 = 1;

P0\_B1 = 0;

}

else if (i == 3) { //and

P0\_B0 = 0;

P0\_B1 = 1;

}

else if (i == 4) { //reset

P0\_B0 = 0;

P0\_B1 = 0;

}

}

void setDataPins(int i) {

//assume i is a decimal integer less than 32

int r = 0;

int d[4]={0,0,0,0};

int index=0;

//converts decimal to binary

while(i != 0) {

r = i%2;

i = i/2;

d[index] = r;

index++;

}

//sets data pins

P0\_B4 = d[0]; //least significant bit

P0\_B5 = d[1];

P0\_B6 = d[2];

P0\_B7 = d[3];

}

void delay1() {

int x;

for (x = 0; x < DELAY; x++) {}

}

void clockPulse() {

P1\_B0 = 0;

delay1();

P1\_B0 = 1;

delay1();

P1\_B0 = 0;

}

//-----------------------------------------------------------------------------

// main() Routine

// ----------------------------------------------------------------------------

int main (void)

{

WDT\_0\_enter\_DefaultMode\_from\_RESET();

enter\_DefaultMode\_from\_RESET();

XBR2 |= 0x40; //Enable Crossbar so we can easily turn pins on and off.

setDataPins(1);

setOpPins(4); //Resets LEDs

clockPulse();

setOpPins(1); //sets operation to SUM

while (1) {

if (P0\_B2 == 0) {

int i;

for (i = 0; i < 15; i++) {

clockPulse();

}

delay1();

delay1();

delay1();

}

}

}