Nikola Janjusevic
Joseph Bentivegna
April 3rd 2017

Documentation:
The 4 Bit Processor

Abstract
The 4 Bit Processor Project interfaces a 4 bit Arithmetic Logic Unit (ALU), built with digital logic components, with an EFM8BB3 microcontroller unit (MCU) programmed with C-code. The MCU provides the input and clocking mechanism, whilst the ALU provides the computation and output of the system. The current implementation allows for ALU operations to be dictated through a through a script run in the microcontroller. The current MCU code allows for basic commands to the ALU. Further implementation should strive for more abstraction and greater complexity in the operations of the ALU.

ALU

Design

The Arithmetic Logic Unit is designed to allow two inputs of 4 bit computation with bitwise AND, OR, and integer addition.

The two registers associated with the ALU are known as the Temporary Register and Accumulator Register. The Accumulator both stores the output of the ALU and provides one of the two inputs, thus giving the ALU a feedback system. The Temporary Register is the second input to the ALU, however its data must come from an external source. Both registers are parallel in parallel out (PIPO).

The ALU outputs 4 bits of data and 1 bit of overflow. Each data bit is the output of a universal gate: an AND gate, OR gate, and full adder multiplexed together. The adders from each bit communicate their carry ins and carry outs to function as a 4 bit adder.

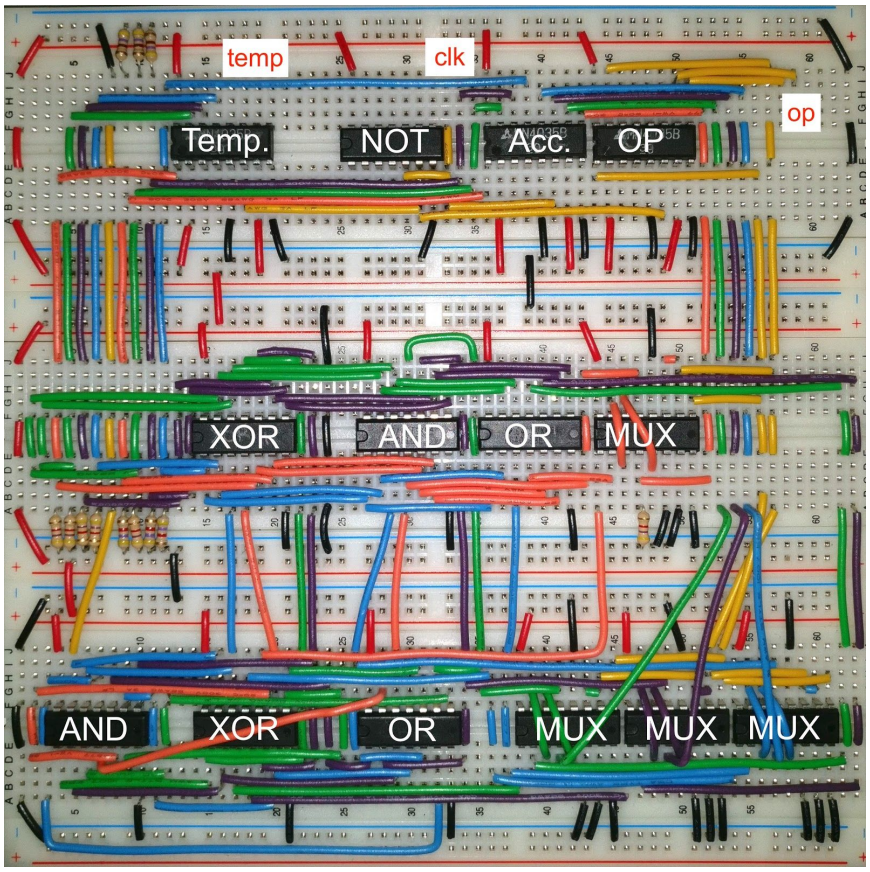
The address pins of each universal gate's multiplexer are tied together to keep each gate on the same state of operation. These two pins are driven by a register which stores the current operational code (Opcode) of the ALU. Inputs to the Op code register must come from an external source. In this design, the Op codes are as follows:

Op Codes:
00_2 Zero
01_2 AND (bitwise)
10_2 OR (bitwise)
11_2 Addition

The operation of a single computation within the ALU is controlled under 1 clock pulse (square pulse). Note each register latches on the positive edge of a clock pulse. To avoid a race

condition, we want the Temporary and Opcode registers to have their intended values stored before the Accumulator accepts its new value. To achieve this, Accumulator receives an inverted clock pulse while the the Temporary and Opcode registers receive a regular clock pulse. This allows a positive edge to reach the Temporary and Op code registers before a positive edge hits the Accumulator register. This clock pulse must come from a source external to the ALU.

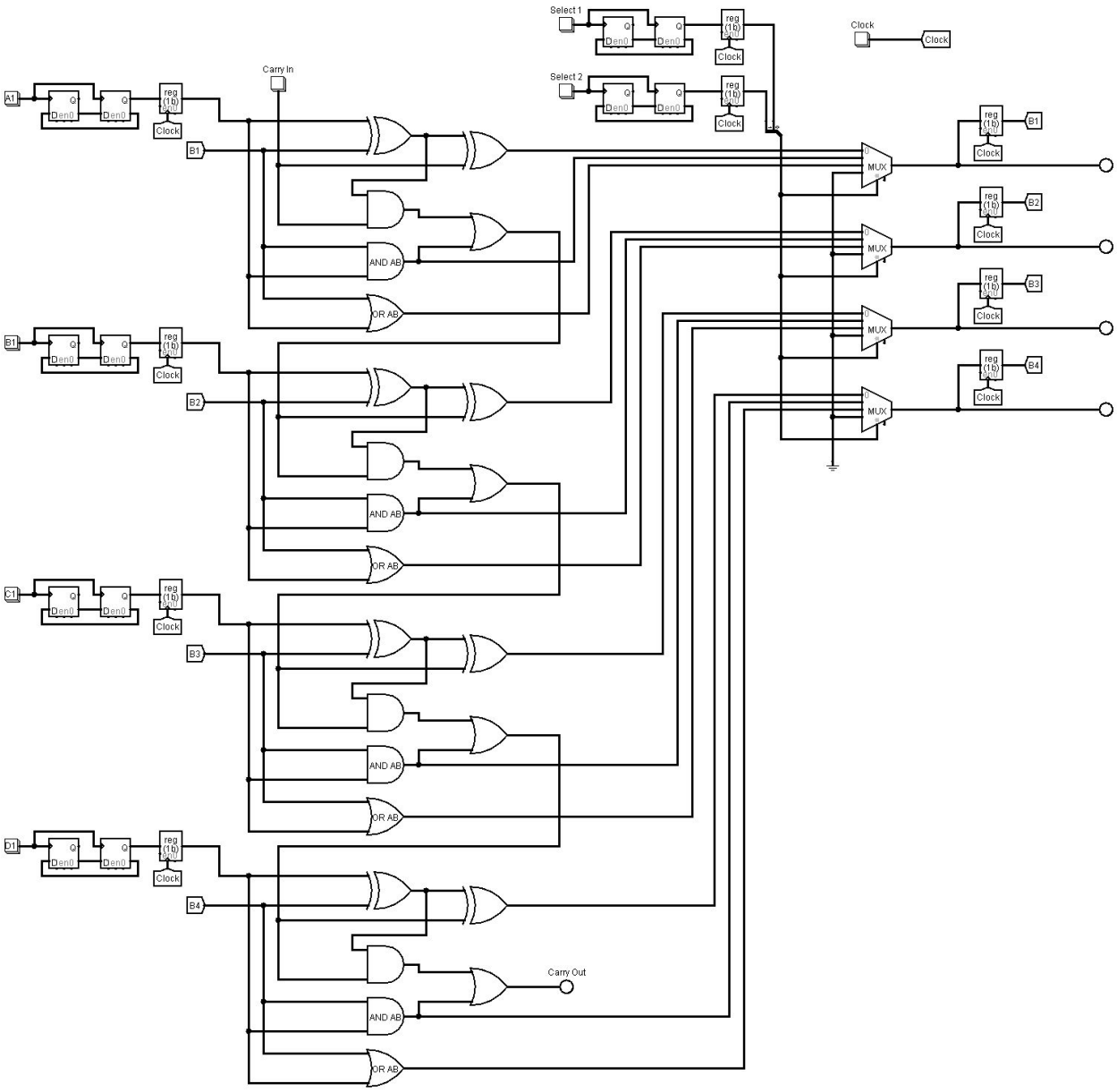
Breadboard Diagram



Key:

Temp., Acc., OP: 4035 register
NOT: 4069 hex not gate
XOR: 4030 quad dual in. xor gate
AND: 4081 quad dual in. and gate
MUX: 4053 triple 2:1 multiplexer
temp: temporary register 4 bit input (external)
clk: clock signal input (external)
op: op code 2 bit input (external)

Logic Diagram of ALU



Microcontroller C-Code

Pin Definitions

Pins on the MCU were assigned names for easier reference. Pins 1.4-1.6 were assigned for LED functionality and pins 0.2 and 0.3 for button outputs. There are 6 output pins in the project. 1.0 and 1.1 are for OPcodes, 1.3 is for the clock, and 3.1-3.4 are for the temporary register. Pins were connected to the ALU using male-male ribbon cables.

Functions

```
void delay(int n)
```

The delay function implements a timing delay by making the microcontroller's CPU perform thousands of addition, checking operations through a for-loop. The default for-loop length is 1000 loops, meaning 2000 computations. The input parameter, n, repeats this process n times.

```
void blink(int LED, int n)
```

The blink function utilizes the MCU's built in LEDs to blink them for visual aid when the function is called. Blink takes in integer parameters of LED to indicate which color is to be flashed (1, 2, or 3), and n which is the number of times the LED will blink. The blink function uses the previously defined delay function to turn LEDs high, let them stay high, then bring them low again. (note, the MCU LEDs light on logic 0).

```
void clk()
```

The clk function sends out a single square pulse through the defined GPIO pin, clk. It uses the delay function of input 1. When a single pulse is completed, the LED will blink red one time for visual aid.

```
void op(int opcode)
```

The op function translates a decimal input to a GPIO pin output. The function takes inputs 0-3 and, through a switch statement, assigns proper OPcode values to the OPcode pins. An input of 0 sets the accumulator to 0 whereas an input of 1, 2, or 3, corresponds to an operation of AND, OR or addition of the temporary register and the accumulator. When assignment of the opcodes is completed, the LED will blink blue one time for visual aid.

```
int* dec2bin(int c)
```

The dec2bin function converts a decimal input to a binary output and stores each bit of the binary output as an element in an array. The function first allocates 4 bits of memory for the array "arr". It then uses bitwise comparison to determine each bit of the binary number and put it in the array. Finally, the function returns the pointer to the array.

```
void uTemp(int num)
```

The uTemp function sets the values in the temporary register for input into the ALU. The function works by creating pointer, "binptr," that calls the dec2bin function with a user's decimal

input. It then sets the temporary register output pins to the values in the array from the dec2bin function. Finally, it frees the memory allocated by the dec2bin function and blinks once green for visual aid.

```
void slowDownCount(int num)
```

The slowDownCount function automatically decrements the value in the accumulator by 1 from a value input by the user. The function takes one parameter “num” and then uses a for loop to decrement the value of “num” by 1, each loop. Within the loop, the value in the accumulator is first set to 0000, then the temporary register is set to the current value of the loop. Then the temporary register is OR’d with the accumulator to send the values into the accumulator. There is then a delay before the process is repeated.

```
void slowUpCount(int num)
```

The slowUpCount function automatically increments the value in the accumulator by 1 from a value input by the user. The functionality of this function is nearly identical to that of slowDownCount but this time the value increments by 1 each loop.

```
int main(void)
```

The main function is used to run the ALU by allowing the user to set OPcodes and temporary register values that are run on a button press. The main first calls the watchdog timer function which disables the watchdog timer (see watchdog timer section). It then enables the crossbar which allows us to directly enable or disable pins we wish to use. Functions can be called within main to set the values of Opcodes and the temporary register for usage in the ALU.

Watchdog Timer

The watchdog timer is a hardware timer that periodically fires a system reset if the main program does not service it. The watchdog timer initially posed a problem where approximately every 10 seconds the system would completely reset and the value in the accumulator would be set to 0000. To fix this, the WDT_0_enter_DefaultMode_from_RESET(void) function was added which both enabled watchdog control and disabled the watchdog timer via a key sequence. Calling this function in the main properly disabled the watchdog timer and the periodic reset seised.

Summary of Code

To use the hardware in conjunction with the ALU, the user must alter the main to send OPcodes and temporary register values to the ALU. The functions clk(), op(int opcode) and uTemp(int num),

Interfacing

General Input Output (GPIO)

The EFM8BB3 Microcontroller interfaces with the digital logic ALU through a series of GPIO pins. These outputs serve as inputs to the ALU's temporary register, opcode register, and clocking mechanism.

Clocking

The clocking mechanism is driven by the clocking function and delay function described above. It currently relies on delays provided by wasting the MCU's processor with excessive operations in a for-loop. Improvements to the timing, such as using the timer peripheral on the MCU, will need to be made to improve the ALU's performance speed and obtain more consistent and precise clocking.