ECEN 240 - Lab 9 – Register File

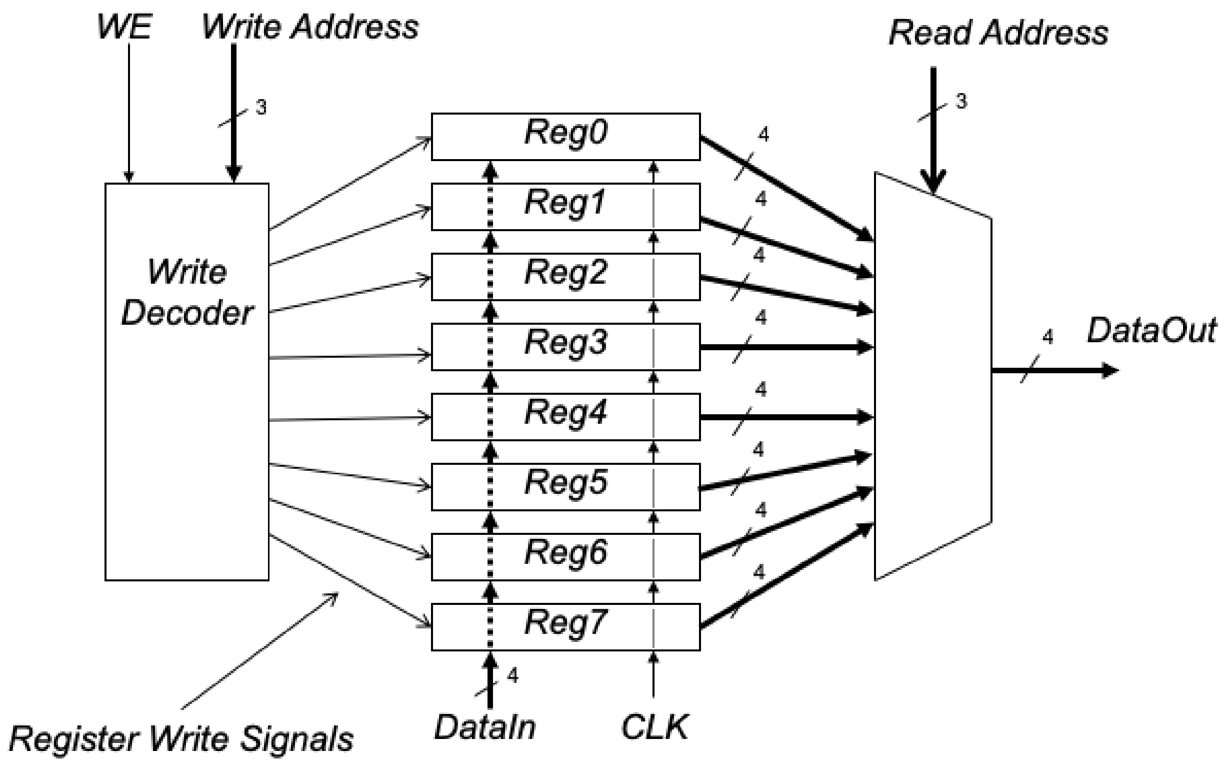
Name: Brodric Young

Purpose:

1. Become more familiar with:
    - Decoder Usage
    - Loadable D Flip Flop Configurations
    - Multi-bit MUXes
    - SystemVerilog Multi-dimensional Arrays
2. Learn about a type of memory structure (a register file) used in many processors and in many digital circuits

Background Information

In this lab project, you will implement a 4-bit by 8 word dual-ported register file (it is called "dual-ported" because it has an input port and an output port). Note that a register file is one form of a memory device. Multi-ported registers are used as the smallest, but fastest, memory component in CPUs. The following block diagram shows the target design:



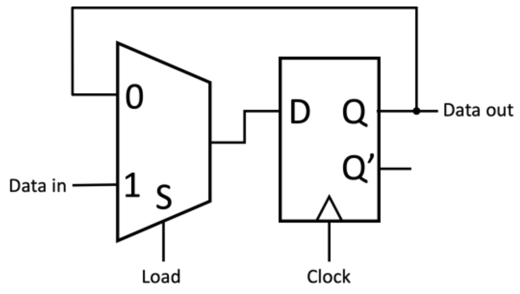
A register files can be addressed for writing or reading. A memory with *m* address lines will contain 2^m locations. For example, a memory with 4 address lines will contain 16 memory locations.

In this project you are asked to complete the design of the dual-ported 4X8 register file (four bits may be stored in each of the 8 address locations). It will have 3 write address bits, 3 read address bits, 4 data-in lines, and 4 data-out lines. The register file will also have a write enable signal that will allow a write signal (or load signal) to be given to a selected register. The register will receive (load) new data on the rising edge of the clock when its write enable signal is active.

A key advantage of a dual ported register file is that writing and reading may be performed independently. For example, on a given clock cycle you could write data at address location 2 but be reading from address location 1!

There will be 3 major components to this design:

1. A 3 to 8 decoder circuit which selects the destination address of the data to be written.
2. The memory portion of the register file. This will consist of four "loadable" D flip flop-based memory elements per memory location. Use rising edge triggered D flip flops for your design. For each flip flop, you need to be able to control when the flip flop loads a new value. Construct "loadable" registers by adding a 2:1 multiplexer to each flip flop as shown:



3. A four-bit multiplexer with 3 select one of the 8 memory locations to read.

Procedure

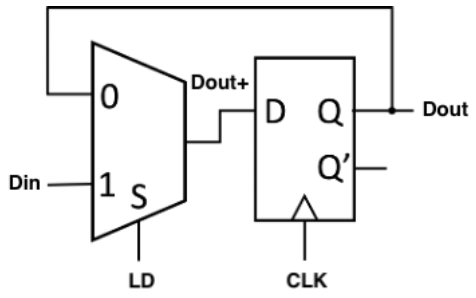
Part 1
Design the Register File Using Logisim Evolution

Download the *LogisimEvolution* circuit file as a starting point for the design.

Loadable Memory Cells

The One-bit Loadable Register

For the D Flip-flop and MUX circuit below (a one-bit loadable register), write the transition table and derive a Boolean equation which describes the Dout+ signal.



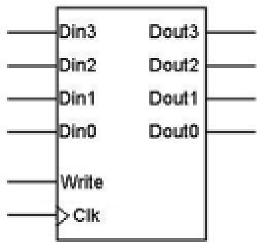
Din	LD	Dout	CLK	Dout+
0	0	0	↑	0
0	0	1	↑	1
0	1	0	↑	0
0	1	1	↑	0
1	0	0	↑	0
1	0	1	↑	1
1	1	0	↑	1
1	1	1	↑	1

Boolean Equation for Dout+ :

Dout+ = Dout LD’ + Din LD

The Four-bit Loadable Register Subcircuit

Open the “Four_bit_register” subcircuit and build a 4-bit loadable register subcircuit that performs the function of the block diagram, below:

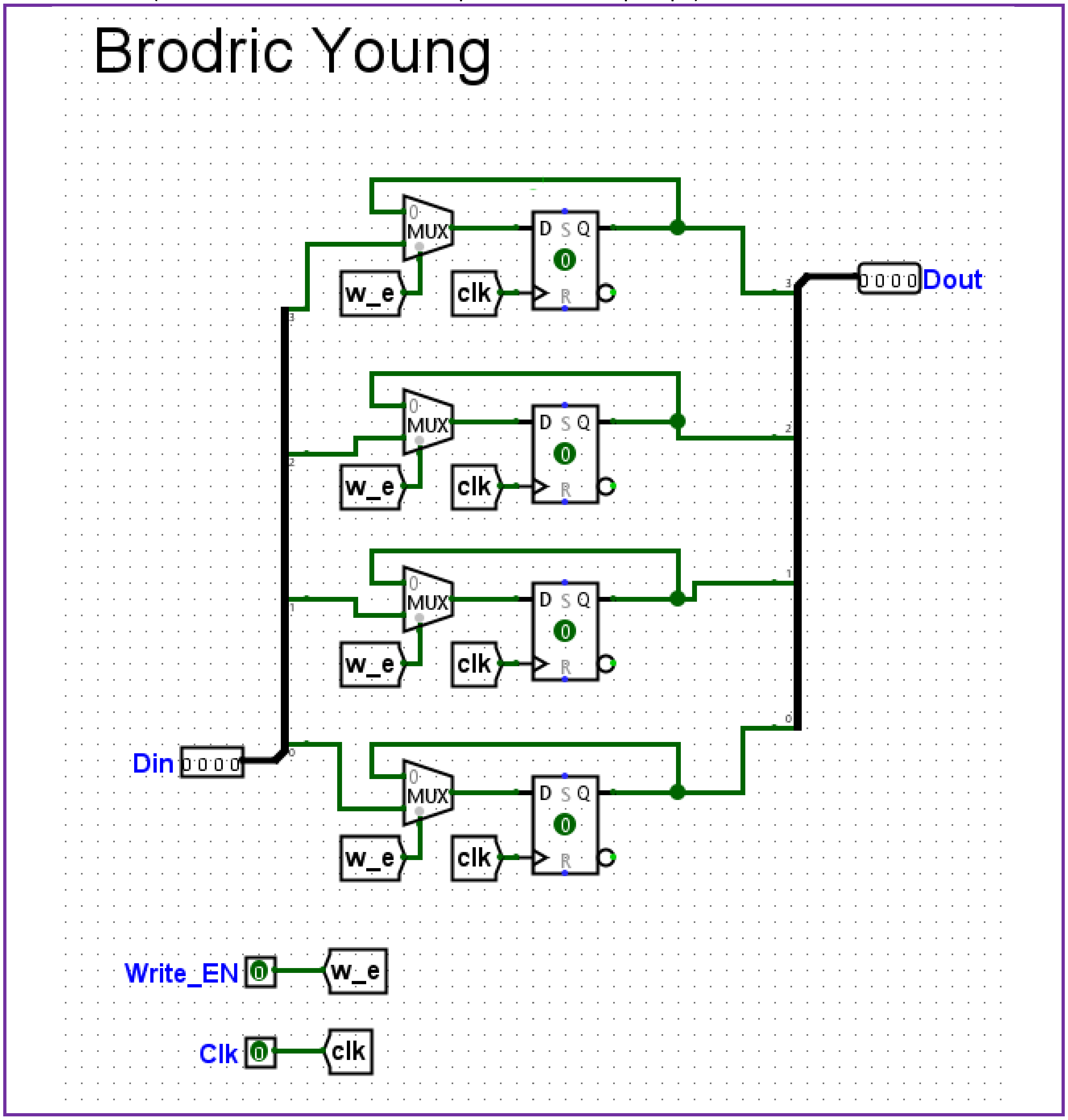


4 Bit Register

It will require four, one-bit wide 2:1 MUXes and four, D flip flops.

NOTE: Within each 4-bit register, all of the clocks are shared and all of the Write (Load) signals are shared.

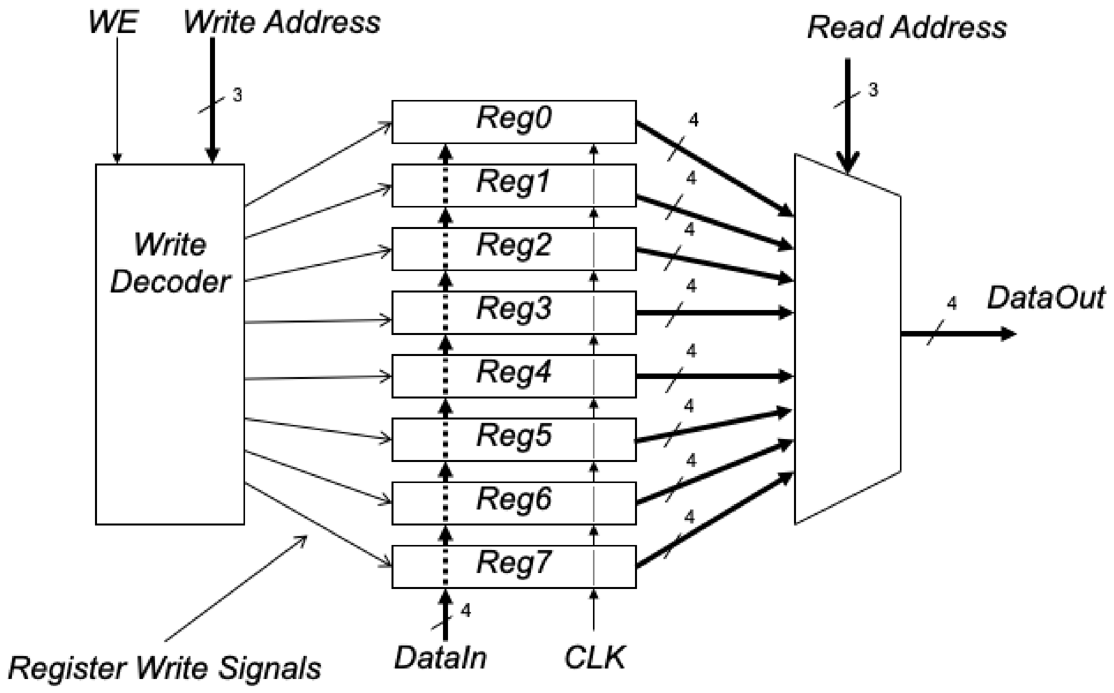
Paste your *Logisim* 4-bit loadable register circuit (including your name) in the box below (this should consist of 4 multiplexers and 4 flip flops):



4-bit Loadable Register Circuit (20 points)

Completed Register File Schematic

Connect all of the building blocks of the register file as shown:


WE
Write Address
3
Write Decoder
Register Write Signals
Reg0
Reg1
Reg2
Reg3
Reg4
Reg5
Reg6
Reg7
4
DataIn
CLK
Read Address
3
4
DataOut

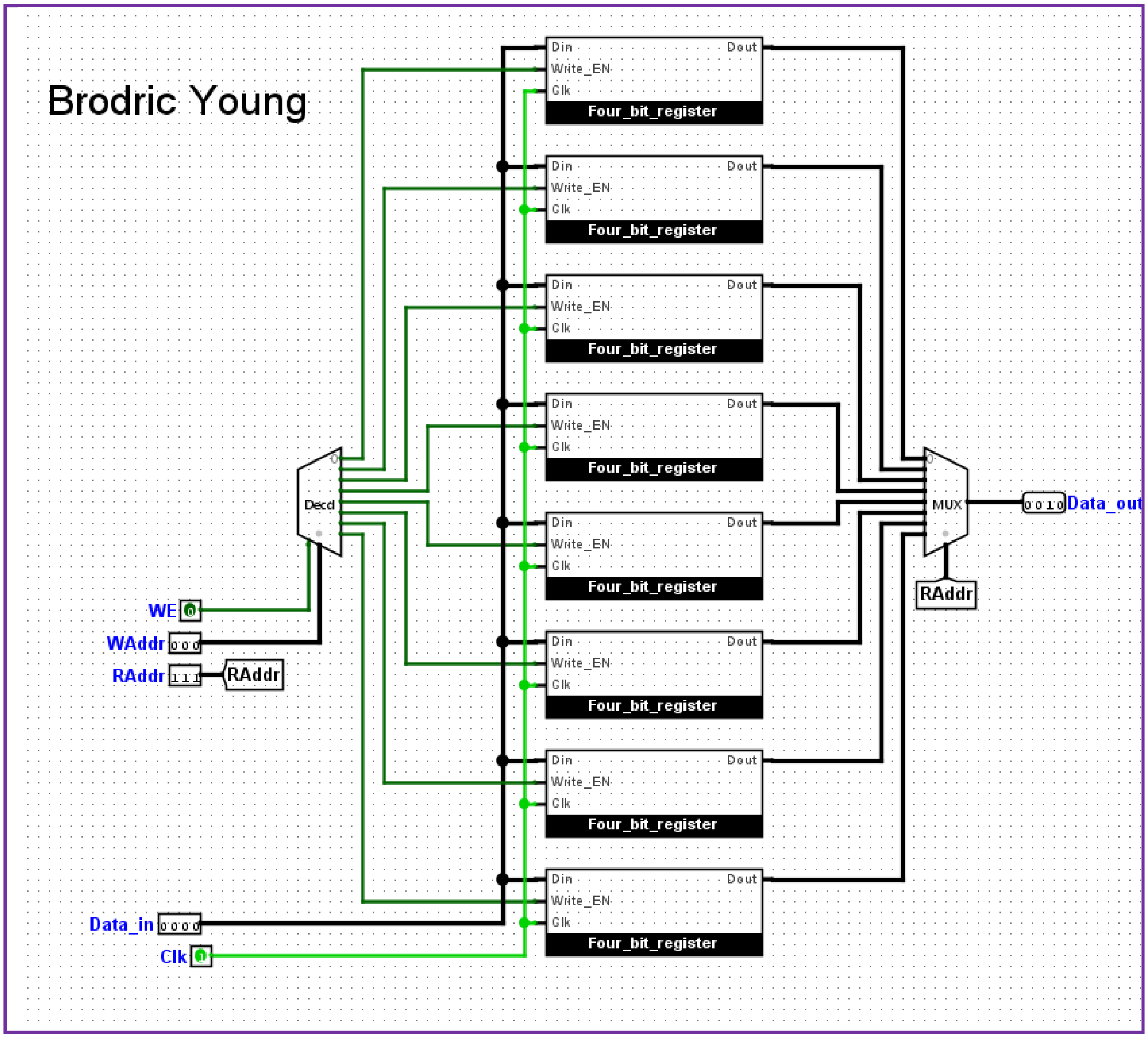
After construction, it is necessary to functionally test your circuit (there will be no test vector). Verify that your circuit performs properly by testing the following patterns in your register. Since memory is involved, you must record your results in the order shown in the transition table:

DataIn	Write Address (WA)	Write Enable (WE)	CLK	Read Address (RA)		DataOut
0011	000	1	↑	000		0011
0101	001	1	↑	000		0011
1100	010	1	↑	000		0011
1010	011	1	↑	000		0011
1101	100	1	↑	000		0011
1011	101	1	↑	000		0011
0100	110	1	↑	000		0011
0010	111	1	↑	000		0011
0000	000	0	↑	000		0011
0000	000	0	↑	001		0101
0000	000	0	↑	010		1100
0000	000	0	↑	011		1010
0000	000	0	↑	100		1101
0000	000	0	↑	101		1011
0000	000	0	↑	110		0100
0000	000	0	↑	111		0010

Take Lab 9 Quiz 2

(This Quiz is worth 20 points)

Paste your *Logisim* circuit of your complete register file (including your name) in the box below:



Completed Register File Circuit (20 points)

Part 2
Design the Register File Using SystemVerilog

Refer to the SystemVerilog instruction document to implement the Register File on a Basys3 board.

Paste your Register File module code in the box below:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/14/2024 09:30:00 AM
// Design Name:
// Module Name: Register_File
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module
Register_File(

   input [3:0] Data_In,
   input [2:0] WAddr,
   input [2:0] RAddr,
   input WE,
   input clk_100MHz,
   output [3:0] Data_Out,
   output [6:0] seg,
   output [3:0] an,
   output dp
   );

   clk_div(clk_100MHz, clk, seg, an, dp);

   logic [3:0] RegFile [8];
```

```
    assign Data_Out = RegFile[RAddr];

    always_ff @ (posedge clk)
       if (WE)
          RegFile [WAddr] <= Data_In;


endmodule
```

Register File Module Code (10 points)

Pass Off the Register File Implementation Using Lab 9 Quiz 3

(Worth 20 points)

Conclusions Statement

Write a brief conclusions statement that discusses the original purposes of the lab found at the beginning of this lab document:

- How does a Loadable D Flip Flop circuit differ from a simple D Flip Flop?
- What is the purpose of the MUX in the circuit?
- What is the purpose of the Decoder in the circuit?
- A Microprocessor uses memory to hold the input and output data of an ALU. How might a register file be used in a microprocessor?
- How does SystemVerilog simplify the design process of a register file?

Please use complete sentences and correct grammar to express your thoughts:
(The conclusions box will expand as you write)

In this lab we learned about registers using flipflops, decoders, and multiplexers. A loadable D flip flop is different than a simple D flip flop in that you can load it with a value in addition to just setting it using the D value. In this circuit, the MUX was what allowed us to read the information stored in a given register address by allowing that specific address to pass through. The decoder sent the write_enable value to the correct address to be written to. By passing in the address, it sent a 1 to that address allowing it to write. A register file like the one we made could be used in a microprocessor because it can store information. This is essential to microprocessors because it can hold the information and read it whenever it's needed as well as change it when it needs changing so it can preform complex things. Implementing this circuit in Logisim took a lot more time and effort than in SystemVerilog because SystemVerilog simplifies the process. Instead of building the MUX and Decoder and Registers manually, we can do this in just a few lines of code.

Conclusions Statement (10 points)

Congratulations, you have completed the lab!

You may now submit this document.