Lab 10: Build a Computer

November 10, 2009

Contents

1 Prelab **3**

2 Lab **3**

3 Supplementary Material **4**

3.1 Quartus 4

3.1.1 Graphical Design 4

A conceptual block diagram of a simple computer is shown in Figure 1. In previous labs you have already designed the Addr_Mux, the ALU, the control unit and the required registers. In this lab you will put all the components together to build a computer. The only missing block is the memory block which you can find here. You will also need the mem_init.v in which you are going to insert your program code. You are going to use graphical design to implement the computer as shown in Figure 1. Instructions on how to do that is provided in the prelab section 1.

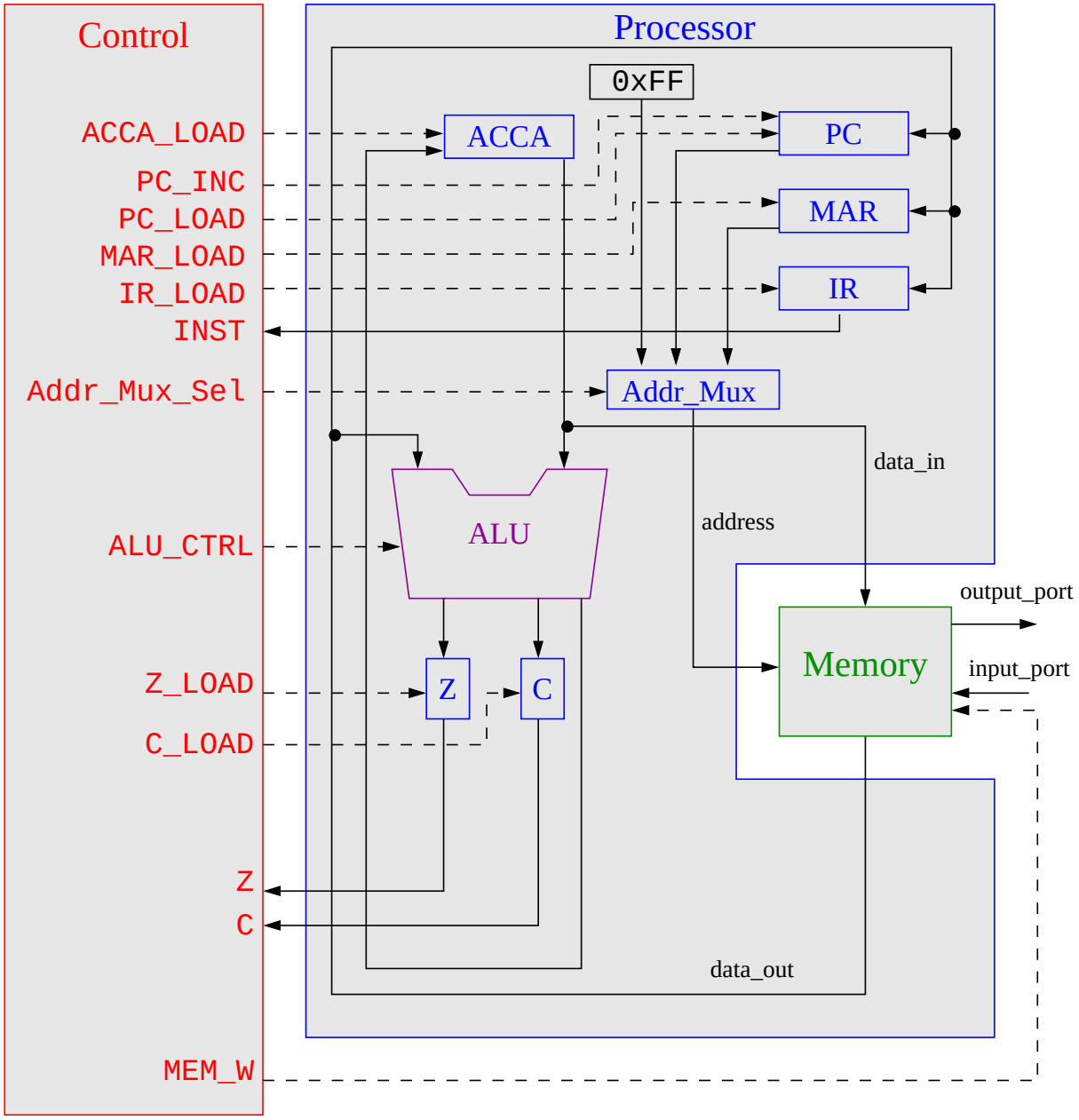

Figure 1: Simple Computer

1 Prelab

1. Using the instruction set provided in the Appendix 3.1.1, write a computer program to generate running lights. One way to accomplish this is to start with an 8-bit number 0000 0001 (where the one represents the LED that would be off). Then left shift that number once you have reached the end, jump back to the beginning of the program. Figure 2 shows the expected output at different time steps.

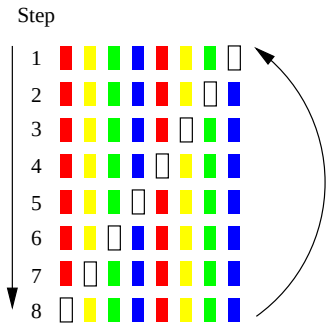

Figure 2: The sequence at which the LEDs turn on or off. The empty rectangles with black boarder represent and LED in the off state.

.

2. Using the graphical method as shown in Section 3, design and build the entire computer.

2 Lab

1. Enter your running light program in to the mem_init.v file.
2. Simulate the computer you have created in the prelab.
3. Run your code on the Altera board.

3 Supplementary Material

3.1 Quartus

3.1.1 Graphical Design

1. Start a new project and include the mem_block.v.
2. Open the mem_block.v file and select `File > Create/Update > Create Symbol Files for Current File`. This creates a memory block as shown below.

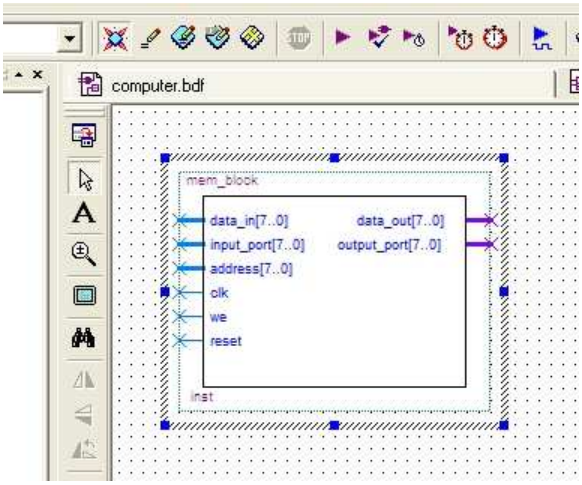

Figure 3: Memory block symbol

3. Open a block diagram file by
   `File > New > Design Files > Block Diagram/Schematic file` Input and output pins are located under `primitives/pin`. That creates a *.bdf file.
4. Right click in the main window and select `Insert > Symbol`.
5. Under you project directory select the mem_block you just created.
6. Keep adding the files that you have created in previous lab for remaining components of the computer. Each time create a Symbol file and added it to the your main *.bdf file.
7. For the address mux and the reset constant you can use already existing block in Quartus. You can do that by
   `Insert > Symbol`, select `../quartus/libraries/`, then select `lpm_mux` and/or `lpm_constant` which are located under `megafunctions/gates`. Input and output pins are located under `primitives/pin`.
8. Once you are done start connecting the blocks as shown in Figure 1.

Appendix

Table 1: Computer Instructions

	Mnemonic	Operation
0	`nop` (no operation)	Do nothing
1	`LDAA addr` (load `ACCA` from memory)	Loads `ACCA` with the value in memory at address `addr`. `C` stays the same, `Z` changes
2	`LDAA_IMM #num` (load `ACCA` with an immediate)	Loads `ACCA` with num, the value in memory at the address immediately following the `LDAA #num` command. `C` stays the same, `Z` changes
3	`STAA addr` (store `ACCA` in memory)	Stores the value in `ACCA` at memory address `addr`. `C` stays the same, `Z` changes
4	`ADDA addr` (add `ACCA` and value in memory)	Adds the value in memory location `addr` to the value in `ACCA` and saves the result in`ACCA`. `C` and `Z` change
5	`SUBA addr` (subtract value in memory from `ACCA`)	Subtracts the value in memory location `addr` from the value in `ACCA` and saves the result in `ACCA`. `C` and `Z` change
6	`ANDA addr` (logical `AND` of `ACCA` and value in memory)	Perform a logical `AND` of the value in memory location `addr` with the value in `ACCA`. Save the result in `ACCA`. `C` stays the same, `Z` changes
7	`ORAA addr` (logical `OR` of `ACCA` and value in memory)	Perform a logical `OR` of the value in memory location `addr` with the value in `ACCA`. Save the result in `ACCA`. `C` stays the same, `Z` changes
8	`CMPA addr` (Compares `ACCA` to the value in `addr`)	Compare `ACCA` to value in `addr`. This is done by subtracting the value in `addr` from `ACCA`. `ACCA` does not change. `C` and `Z` change
9	`COMA` (Complement `ACCA`)	Replace the value in `ACCA` with its one's complement. `C` is set to 1 and `Z` changes
A	`INCA` (INCA `ACCA`)	Increment value in `ACCA`. `C` stays the same and `Z` changes
B	`LSLA` (logical shift left `ACCA`)	Logical shift left of `ACCA`. `C` and `Z` change
C	`LSRA` (logical shift right `ACCA`)	Logical shift right of `ACCA`. `C` and `Z` change
D	`ASRA` (Arithmetic shift right `ACCA`)	Arithmetic shift right of `ACCA`. `C` and `Z` change
E	`JMP addr` (jump)	Jumps to the instruction stored in address `addr`. The `PC` is replaced with `addr`. `C` and `Z` stay the same
F	`JCS addr` (jump if carry set)	Jumps to the instruction stored in address `addr` if `C`=1. If `C` is not set, continue with next instruction. `C` and `Z` stay the same

10	`JCC addr` (jump if carry not set)	Jumps to the instruction stored in address `addr` if `C`=0. If `C` is set, continue with next instruction. `C` and `Z` stay the same
11	`JEQ addr` (jump if `Z` set)	Jumps to the instruction stored in address `addr` if `Z`=1. If `Z` is not set, continue with next instruction. `C` and `Z` stay the same