1 LABORATION 4 – MICROPROCESSOR & MEMORIES

In this laboration, we will design a mini CPU. We will use what we developed in the previous labs to connect everything together. Our CPU will have dedicated program memory, where we will load the necessary instructions. We will also develop a testbench where we will test the functionality of the CPU.

The lab only requires simulations in Questasim or similar.

1.1 TASKS

1) Design a data SRAM model for your microprocessor.

2) Combine the ALU, FSM, and memories to complete your microprocessor design based on the following specifications.

3) Design a testbench that will test the correct functionality of the microprocessor

4) Simulate and verify the functionality of your microprocessor

1.2 ROM AND RAM MODELS

There are different types of memories. In this lab, we require two different types of memory: a read-only memory (ROM) and a dual port random access memory (RAM). Your design should implement a memory, as seen in Figure 4.1. The ROM design is given to you, but you have to design the SRAM based on the specification.

Keep your design parametric; the size of each row should be n-bit, and each memory should be m-rows. The memory models that you design here can only be used for simulation. The design of a memory macro is custom, and a specialist designs it. In a digital design, we use functional models of the memories to simulate our design, whereas in real design, we instantiate memory macros that are generated by a memory compiler.

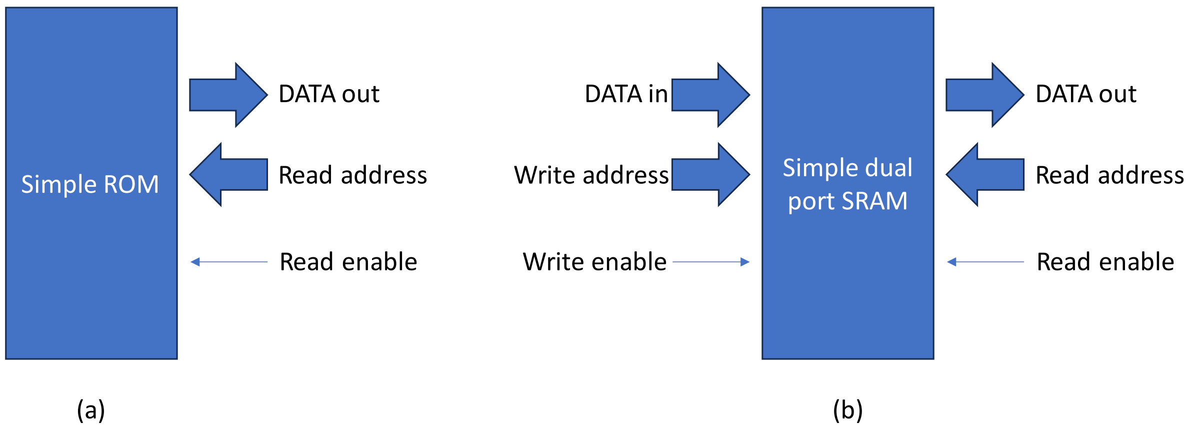


Figure 4.1 Simple models for a (a) a simple ROM and (b) a dual port RAM memories

In an Intel (previously altera) FPGA, you can use the Quartus Megawizard to generate the appropriate memory for your design. A similar tool is provided by the other FPGA vendors. When designing an ASIC, the foundry provides a technology library and also provides the designer with the appropriate memory compiler. The memory specifications depend on the technology and the design parameters. In this lab, we assume the same behaviour as in Figure 4.2.

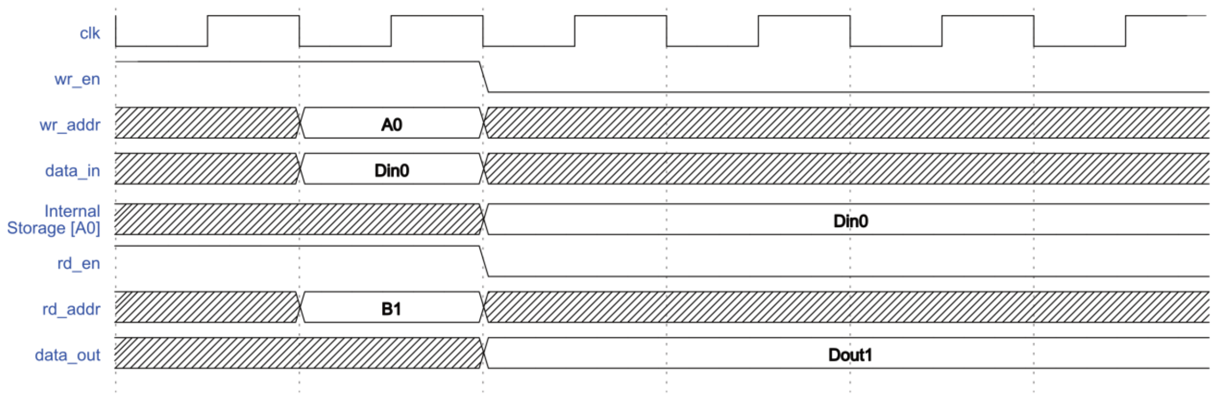


Figure 4.2. Timing diagram for a read and write operation in a simple memory

The ROM memory only has one read port and can be used as pre-programmed instruction memory. The ROM is given to you, and you do not have to design it. The SRAM memory in Figure 4.1 has two ports, one used as a read and one used as a write port. The two ports can be used simultaneously. When reading and writing the same row at the same time, the real value is the old value that is updated with the new value in the next clock cycle. The timing diagram of the SRAM behaviour can be seen in Figure 4.2. For simplicity, you can design your behavioural models of the SRAM and ROM memories as a register file.

1.3 MICROPROCESSOR SPECIFICATIONS

Combine the previously designed components to complete your microprocessor. The abstract diagram of the microprocessor design can be seen in Figure 4.3. Create two modules. The first module should be called a 'microprocessor' and combines the FSM, register file and ALU that you designed in the previous labs. The second module, called 'microprocessor_n_memory', combines the data memory you designed and the ROM given to you together with the 'microprocessor' to produce the final design. Your design should follow the following specifications:

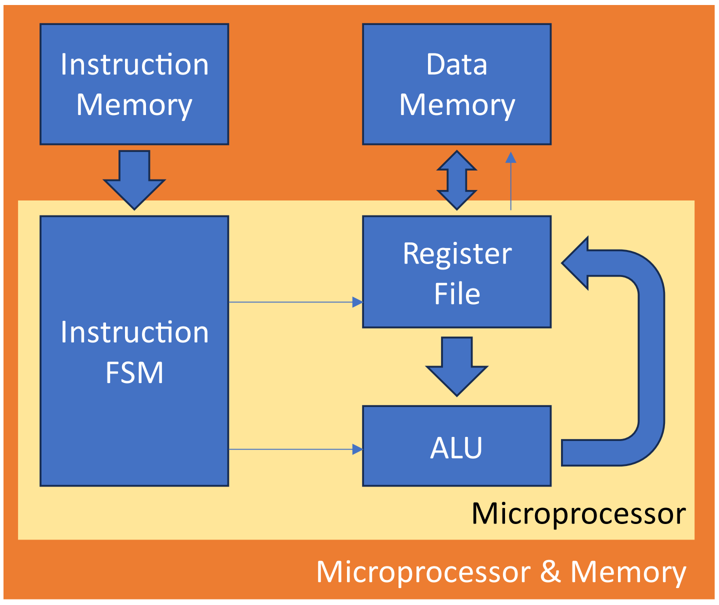


Figure 4.3. Block diagram of the microprocessor

1) Register 0 is always all zeros

2) Register 1 should always be all ones

3) The program counter (PC) is a special register and is not included in your RF. The program counter should be part of the FSM, NOT the datapath

4) The output of the ALU should be registered

5) All flags from the ALU should be registered in their own register

To connect the instruction and data memory to your microprocessor, make sure that you have the following correctly set up:

1) The instruction memory address size matches the one PC size. The PC should directly address the instruction memory and should be essentially treated as the address.
2) The data address memory should have the same size as a register file word. Since the register file will be used to address the data memory, make sure that the value of the register file, in that case, is treated as an unsigned number and not as an integer.
3) The register file should have 8 registers
4) The bit that addresses the register file in the instruction field should be the same as the register file address. In this case, use 3 bits for each OPA and OPB
5) The size of one word in the register file should be 8 bits
6) The PC should be 6 bits and treated as an unsigned number

1.3.1 INSTRUCTIONS

The microprocessor should implement the same instruction set as the one you develop in your instruction FSM.

Instr.	Inst. Code	OPA	OPB	Function	PC
ADD	0000	RA	RB	RA = RA+RB	PC+1
SUB	0001	RA	RB	RA = RA-RB	PC+1
AND	0010	RA	RB	RA = RA AND RB	PC+1
OR	0011	RA	RB	RA = RA OR RB	PC+1
XOR	0100	RA	RB	RA = RA XOR RB	PC+1
NOT	0101	RA	RB	RA = NOT RB	PC+1
MOV	0110	RA	RB	RA = RB	PC+1
NOP	0111	Null	Null	Null	PC+1
Load	1000	RA	RB	RA = mem(RB)	PC+1
Store	1001	RA	RB	Mem(RB) = RA	PC+1
Load Imed.	1010	RA	DATA	RA = Sign Ext. DATA	PC+1
BRN_Z	1011	Offset		Branch if zero	PC+offset or PC+1
BRN_N	1100	Offset		Branch if neg.	PC+offset or PC+1
BRN_O	1101	Offset		Branch if ovf.	PC+offset or PC+1
BRN	1110	Offset		Branch always	PC+offset

1.4 TESTBENCH SPECIFICATION

1.4.1 TEST YOUR MICROPROCESSOR

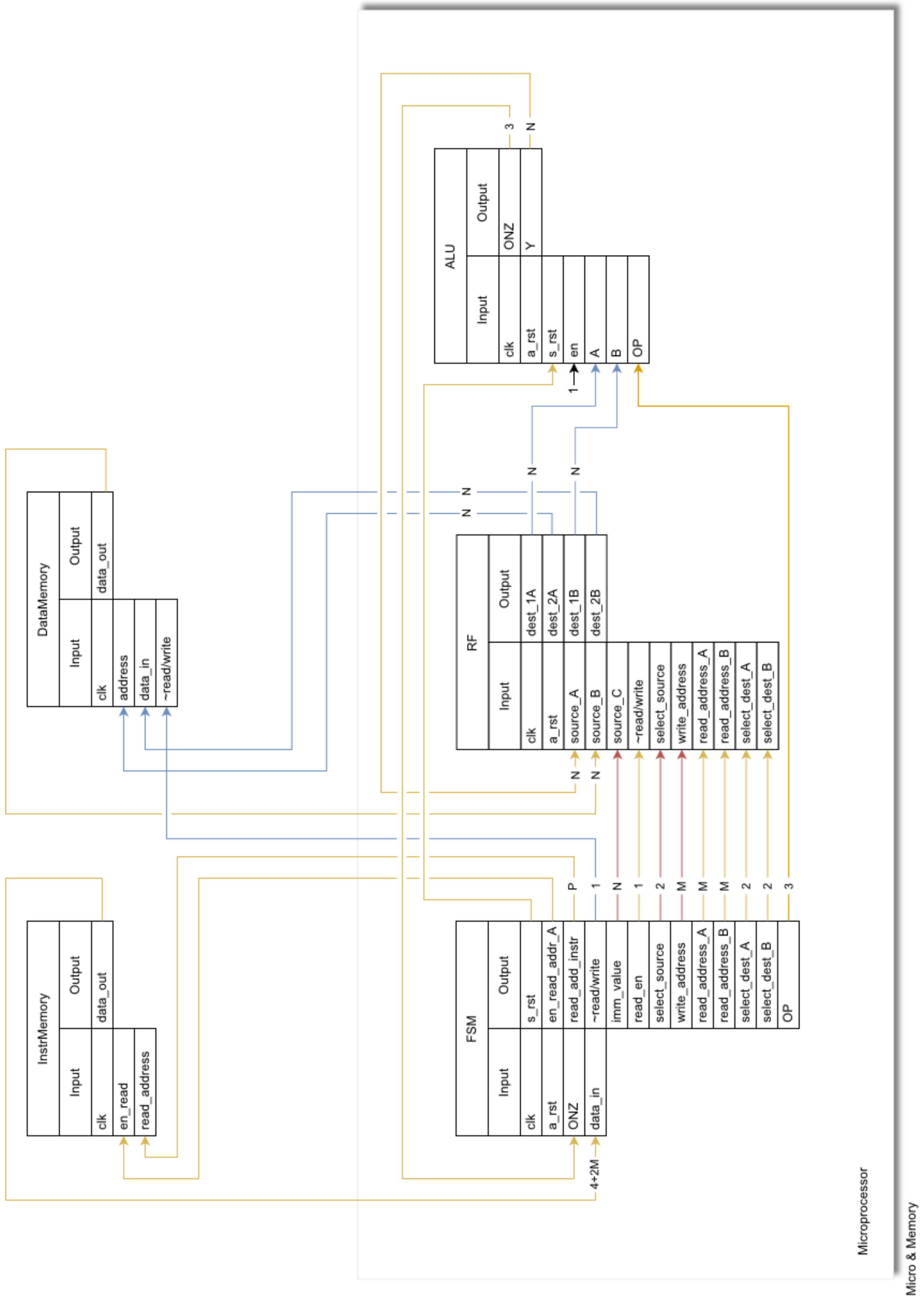
Your testbench should test the functionality of the microprocessor. Your testbench should generate instructions and data for your microprocessor. Essentially, it should act as the memory modules where random instructions and data are generated in response to the microprocessor requests. Specifically, your testbench should complete the following tasks:

1. Check that your design resets states are correct:
    a. The register file is reset to the specifications
    b. The ALU output and flag is reset according to the specifications
    c. The FSM state is reset to the correct specification (see lab 3)
2. Generate random instructions
    a. Constraint your instruction generation so that you first generate only load immediate instructions until your register file is filled with random data (except the positions 0 and 1)
    b. After the register file is filled, generate random instructions to test the functionality of the microprocessor
3. Generate random data to load the register file when requested
4. Create assertions that check the timing of the control signals
5. Create a scoreboard that checks the correctness of the operations
    a. The scoreboard should keep track of what the correct state of the register file should be (depending on the instructions generated)
    b. The scoreboard should check the output of the register file (when a store instruction is issued and the data from the register file is read out of the microprocessor) and keep track of the error ratio
6. Create cover groups to check the following:
    a. Type of instructions generated
    b. Type of ALU instructions generated
    c. A cover group for all the branch instructions, ignoring the other types of instructions
    d. A cover group for the randomly generated offset of the branch instruction
    e. A cross-cover group between the c and d

1.5 PROGRAM YOUR MICROPROCESSOR – NOT MANDATORY

The Instruction memory (ROM) is loaded directly with the binary data in the microcode.mem file. The instructions in the file follow the assembly code given in instruction_assembly.txt. You should be able to simulate your microprocessor with the microcode given to you. If your design is working correctly, you should be able to see the results 2,3,4,5,6,7 in the SRAM. **Make sure that the microcode.mem file is copied in the same location as your project in modelsim!**

1.6 DETAIL DIAGRAM


ALU
Input
Output
clk
a_rst
s_rst
en
A
B
OP
ONZ
Y
3
N
1
RF
dest_1A
dest_2A
dest_1B
dest_2B
source_A
source_B
source_C
~read/write
select_source
write_address
read_address_A
read_address_B
select_dest_A
select_dest_B
FSM
en_read_addr_A
read_add_instr
imm_value
read_en
data_in
P
M
2
4+2M
DataMemory
address
data_in
data_out
InstrMemory
en_read
read_address
Microprocessor
Micro & Memory