

LAB 5:  $\mu$ P1 Computer with Video Display  
(300 points)

**Objective:** In this lab you will design, simulate, synthesize and download a simple processor with a video display output. You will use the ALU that you designed in Lab 3 in your processor. The video output will display the contents of the processor's internal registers on a VGA monitor while the processor executes a program. This is a three-week lab.

**Pre-lab (60 points)**

This processor is based on the example design given in Ch. 8 of your Hamblen text (although the simulator and compiler used there are not the tools that we are using). Study that chapter thoroughly before attempting this lab. Complete the following assignment by the end your first lab session:

1. Give a complete drawing of your processor data resources. (You do not need to include the video display logic.) Identify all of the control signals that will be needed to control your data path. Draw a timing chart or waveform for *each* of the processor's instructions showing how many clock cycles are needed for each instruction and which control signals are asserted in each clock cycle. For example, for the first clock cycle of every instruction, show the control signals that must be asserted to load the IR and increment the PC, etc. Indicate clearly how the IR's setup time is met. (For a  $-4$  speed device, such as we have on our Spartan-3 board, the minimum setup time,  $T_{AS}$ , is  $0.53 \text{ ns}^1$ . In order to meet this time, you should assert the data at the IR register's D inputs the clock cycle before the data is clocked into IR. Since your clock period will be significantly greater than  $0.53 \text{ ns}$ , this will satisfy the setup time. See the Spartan-3 datasheet on the course website for more timing information.)
2. Based on your timing chart or waveforms, draw a detailed Algorithmic State Machine (ASM) chart for your design that shows how each of your control signals will be generated. Classify each control signal as either Moore or Mealy type.
3. Based on your individually assigned op codes, translate the Fibonacci sequence test program provided in this write-up into 16-bit hexadecimal machine code.

Show all these design details to your TA before the end of the first Lab 5 meeting and have your TA sign your verification sheet. NOTE: If you are working with a partner, only one pre-lab is required for the team. (More details later about partnerships.)

---

<sup>1</sup>  $T_{AS}$  = data setup time at F or G input before rising clock edge at input of CLB. See p. 28 of Spartan-3 FPGA Family: DC and Switching Characteristics.

## I. Processor Specifications

The  $\mu P1$  is a simple microprocessor with 16-bit data paths and a single  $256 \times 16$  address space for both instruction and data memory. The complete processor will be implemented in the Spartan-3 device. The block diagram of the  $\mu P1$  processor is given in Figure 1. The block diagram shows the general structure of the  $\mu P1$  processor, but it does not provide details such as the control signals needed for each register or device. It is up to you to specify the control signals needed to implement the design. That is your challenge for the first lab meeting.

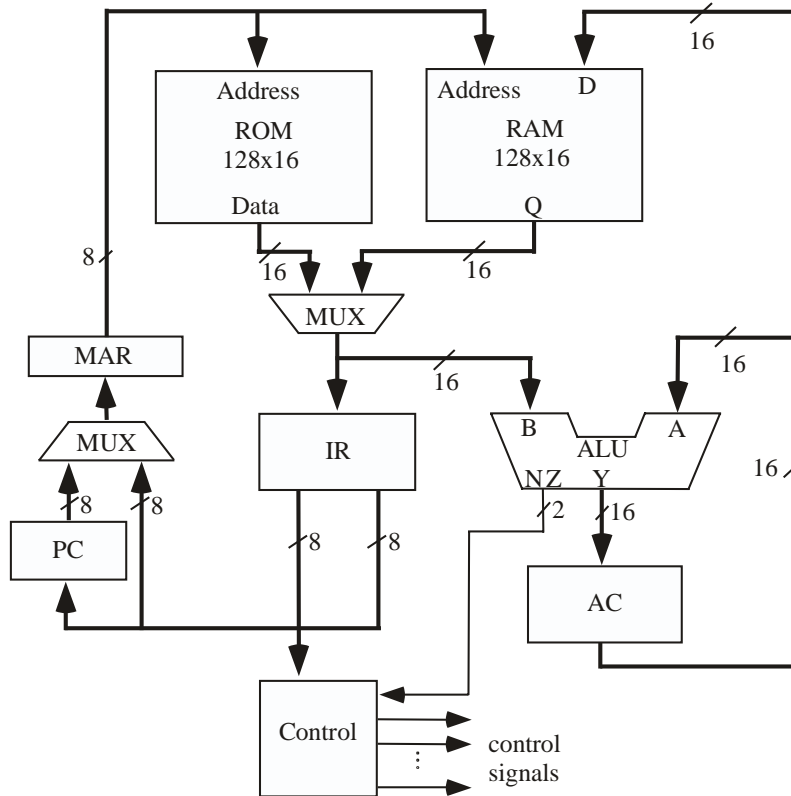


Figure 1.  $\mu P1$  Block Diagram

The instruction format for the  $\mu P1$  CPU is shown in Figure 2 below. The 8-bit immediate address field covers the full memory address space of 256 words. You will be assigned individual opcodes for the instructions. See posted codes (not on web).

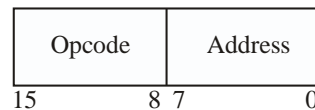


Figure 2.  $\mu P1$  Instruction Format

We have defined the basic instruction set of the  $\mu$ P1 in Table 1. This table gives the function of each instruction. Again, we are leaving the implementation details up to you.

Instruction	Operation	Description
Add address	$AC \leftarrow AC + M[\text{address}]$	Add
Adds address	$AC \leftarrow AC + M[\text{address}] (\text{sat})$	Add with saturation
Sub address	$AC \leftarrow AC - M[\text{address}]$	Subtract
Subs address	$AC \leftarrow AC - M[\text{address}] (\text{sat})$	Subtract with saturation
And address	$AC \leftarrow AC \text{ AND } M[\text{address}]$	Bitwise AND
Com	$AC \leftarrow \text{NOT } (AC)$	One's complement
Lw address	$AC \leftarrow M[\text{address}]$	Load word
Ldi	$AC \leftarrow M[\text{PC}+1]$	Load immediate ( <i>Note 1</i> )
Sw address	$M[\text{address}] \leftarrow AC$	Store word
Jmp address	$PC \leftarrow M[\text{address}]$	Jump to address
Jmpz address	$PC \leftarrow M[\text{address}] \text{ if } Z=1$	Jump to address if zero ( $Z \text{ flag} = 1$ )

Table 1:  $\mu$ P1 Instruction Set

**Note 1**  $M[\text{PC}+1]$  represents the next word in memory following the current instruction.

As you recall from your basic computer architecture class, such as EEC70, and your experience in EEC180A, the basic processor cycle for executing an instruction consists of three operations: Fetch, Decode and Execute. These operations are not necessarily a single clock cycle each. The text by Hamblen and Furman gives a detailed view of the Fetch, Decode and Execute operations for the  $\mu$ P1 in Ch. 8. You should study this material carefully since your processor is closely based on their example. However, there are different ways to implement the processor cycle so you need not rigidly follow their example. We provide the following information for a somewhat different perspective from the Hamblen text.

### Instruction Fetch

The first execution step of *any* instruction is the instruction fetch. All instructions must be fetched in identical fashion.

Fetch the instruction from the memory, latch it into the Instruction Register (IR) and increment the program counter (PC):

$$IR \leftarrow M[\text{MAR}]; PC \leftarrow PC + 1;$$

### Instruction Decode

In order to execute any instruction, the processor must first "decode" the instruction in order to determine which actions to perform.

Decode the instruction from the opcode field. Based on the opcode, your Control Unit should generate specific control signals in their proper sequence. Once the instruction has been decoded, the MAR can be loaded with the appropriate value to access the memory operand. For most instructions, the MAR will be loaded with the address field of the instruction,

although for an Ldi instruction the MAR will be loaded with the updated PC, which is an implied address field, in order to fetch the immediate data.

### **Computation, Jump Completion, or Memory access**

The third execution step varies depending on the type of instruction being executed. Thus, instead of a single execution step, there are actually three different steps depending on whether the instruction is an ALU instruction, a jump instruction, or a memory access instruction.

**Computation:** Perform the ALU operation on the operand or operands specified by the instruction. When any ALU instruction is executed, the status flags, N and Z, must be “registered” (clocked into a register) based on the result of the ALU operation. The status flags will only be updated by the ALU, Lw or Ldi instructions and will not be affected by other instructions such as Sw, Jmp and Jmpz. These status flags are used to determine the outcome of the conditional jump instructions such as Jmpz.

**Jump completion:** The PC and the MAR need to be loaded with the target address specified by the instruction. However, with this processor these actions could be done in the decode cycle, eliminating the need for this cycle. (This is not true in general for a conditional jump or branch.)

**Memory access:** Place the address on the address bus. For a store instruction, place the data on the data bus. You will need to satisfy setup and hold times for the Spartan-3 internal memory component. You must show calculations to support your design’s implementation. (Check the timing specifications on the Spartan-3 datasheets.) The execution stage of an Sw instruction may take more than a single clock cycle in order to satisfy the address and data setup and hold times. You should make this obvious in your ASM chart, after determining the timing requirements for writing data.

### **Memory and I/O Interface**

The  $\mu$ P1 processor will address memory using an 8-bit address bus and can therefore access up to 256 words of memory. The instruction memory (ROM) should be mapped to the lower 128 words, 0x00 – 0x7F, while the data memory (RAM) should be mapped to the upper 128 words, 0x80 – 0xFF. A memory-mapped 16-bit output port, PR, will be mapped to RAM address 0xFF. Writing to 0xFF will store data in two places, RAM location 0xFF and the PR port. Since PR is only an output port, reading from location 0xFF will just read from that location in RAM, avoiding any bus conflict. The output of the PR port will be displayed on both the 4-character seven-segment display and on the video display, to be described shortly.

### **System Clock**

In your final design, which will be downloaded to the Spartan-3 board, the system clock rate should be selectable based on SW7. With SW7='0', your system clock frequency should be on the order of 1 Hz so that you can easily read the video output display. You can compare the register values shown on the video display with your simulation results. On the other hand, with SW7='1', the system clock frequency should be faster so that the output of the seven-segment

display updates at a quick, but easily readable rate. You will be able to observe the Fibonacci number sequence by watching the PR's value on either the seven-segment displays or on the video display. To generate the system clock using the 50 MHz clock input, you will need to design a clock divider circuit. However, you should be able to bypass the clock divider during simulation. See the Lab 1 tutorial VHDL code for how it was done there with a generic.

### **Video Output Display**

Your system will also contain video control logic that will display the contents of internal registers of the  $\mu$ P1 processor. The display will look similar to the example given in Figure 9.8 of the Hamblen text, although the registers will be based on your  $\mu$ P1 processor rather than the MIPS processor as in that example. Your video output should show the following information: PC, IR, MAR, MDR (Data in), AC, PR, Z, CLK. You should display the corresponding hexadecimal or binary value next to each of the register or signal names, respectively. In addition, the top two lines of the display should show your name and your partner's name (if you have a partner).

The video display logic will not be controlled directly by the processor. Instead it will be dedicated logic that takes some of the processor signals as inputs and generates the appropriate video control signals. Therefore, you can test your processor independently from the video display. Once your processor works correctly, you can add the video output.

### **Lab Requirements**

**Partners:** You may choose a lab partner for this lab or you can go solo. To do so, you must complete a partnership declaration form and give copies to each Lab Section TA associated with the two individuals involved in the partnership. A form for submitting the declaration is posted on the course web page. This declaration must be done at or before the first lab meeting affiliated with Lab 5 for each person involved. Only one lab report from the partnership is required and a joint grade will be assigned to each person. You may use either of the unique op code assignments for the individuals. Choose your partner carefully because no "divorce" will be possible. A joint grade will be assigned no matter what happens during the execution and reporting of the lab.

After the pre-lab assignment, this lab project has 3 separate milestones (or millstones if you like).

#### **Part L1 – VHDL modeling and functional simulation of the processor (80 points)**

- a) Write a *synthesizable* VHDL description of your processor based on your pre-lab design. Do not include the video display logic until you have verified your processor design through simulation.
- b) Write a test bench program to functionally simulate your design. You will verify your design using the Fibonacci sequence test program that you translated in the pre-lab. In your simulation, you should trace all the signals in the entity of your processor as well as important registers and control signals. Save some of the traces for your final report.

- c) Demonstrate your functional simulation to your TA and have this TA sign a separate verification sheet. This must be completed by the end of the second Lab 5 meeting date.

**Part L2 – Synthesis to the Xilinx library and downloading to the Spartan-3 board (100 points)**

- a) Write a synthesizable VHDL description of your video display logic and add it to your processor model. You will need to modify the top-level entity to include the video control signals.
- b) Synthesize the design for the Spartan-3 board.
- c) Download your synthesized design to a Spartan-3 board and demonstrate your processor to your TA. The TA should sign a separate verification sheet if your design works. This must be done by the end of your third Lab 5 meeting date.

**Part L3 – Lab Report (60 points)**

Submit a consistent report of your design and its simulation, synthesis and operation. The following items are required.

- a) Machine language program using your unique op code assignment.
- b) List of data resources used in your design, registers, ALU blocks, buses, etc.
- c) VHDL code listings actually used for the synthesis and simulation.
- d) Test bench code and simulation results including sample traces of some control signals
- e) Schematics of typical pieces of the final synthesis results printed from Amplify.
- f) Separate signed verification sheets for the pre-lab and Parts L1 and L2 as described above
- g) Conclusions

**Hints**

In order to facilitate the debugging and synthesis of your VHDL model you should follow the guidelines given below:

1. Think before you write the VHDL code. For every VHDL construct that you use, make sure you know what it is likely to produce in hardware. Do not write redundant code.
2. Keep the VHDL model simple. You can accomplish a lot by just "if-then-else" or case statements. You don't need fancy loop, generate statements, and procedures for this exercise.
3. Remember that wait statements infer registers. Use them carefully.
4. Model the various signals using the appropriate data types. Do not use types unthinkingly. If a signal (bus) is expected to be of a certain size, make sure you model that using a subtype or integer range type.
5. The control unit is the crucial aspect of your design. After you draw the state diagram, spend some time thinking about it and discussing it with your partner. Make sure all the control signals that you need are there and are assigned correctly in each clock cycle.

6. Write behavioral code as far as possible. Let the tool do the synthesis for you. That means don't try to design individual registers, multiplexers, etc. from basic gates and flip-flops.
7. Be careful not to infer level-sensitive latches unintentionally. Check the report files to make sure that you have not inferred any latches.

## APPENDIX

### Fibonacci number generator test program

```

; Calculate the Fibonacci sequence between 0 and 0x7fff.
; First show the sequence in ascending order, then descending order, etc. in endless loop
; The program uses some RAM locations for specific variables:
; RAM addresses:   0x80           Low Fibonacci number
;                 0x81           High Fibonacci number
;                 0x82           New Fibonacci number or saturated value
;                 0xFF           Latest Fibonacci number stored in PR (display) register

```

<u>RAM Address</u>	<u>Hex Code</u>	<u>Label</u>	<u>Instruction</u>	<u>Comment</u>
0		ST:	Ldi	; load 0 into AC
1	0x0000			; immediate data
2			Sw 0x80	; store first Fib. number
3			Sw 0xFF	; display 0 on PR
4			Ldi	; load 1 into AC
5	0x0001			; immediate data
6			Sw 0x81	; store second Fib. number
7			Sw 0xFF	; display 1 on PR
8		UP:	Lw 0x80	; get low Fib. number
9			Adds 0x81	; AC gets Low + High (w/ sat)
A			Sw 0x82	; Store result
B			Ldi	; load 0x7FFF into AC
C	0x7FFF			; immediate data
D			Sub 0x82	; AC gets 0x7FFF – result
E			Jmpz 0x16	; if result = 0x7FFF, jump to DN
F			Lw 0x81	; shift Fib. numbers
10			Sw 0x80	; Low gets previous High
11			Lw 0x82	; get new Fib. number
12			Sw 0x81	; High gets new Fib. number
13			Sw 0xFF	; Display new Fib. number
14			Jmp 0x8	; jump back to UP
15				; memory location not used
16		DN:	Lw 0x81	; AC gets High
17			Sw 0xFF	; display next Fib. number
18			Subs 0x80	; AC gets High – Low
19			Sw 0x82	; store new Fib. number
1A			Jmpz 0x0	; if new Fib. = 0, jump to ST
1B			Lw 0x80	; shift Fib. numbers
1C			Sw 0x81	; High gets previous Low
1D			Lw 0x82	; get new Fib. number
1E			Sw 0x80	; Low gets new Fib. number
1F			Jmp 0x16	; jump back to DN