Concordia University
Department of Electrical and Computer Engineering
COEN 316 Computer Architecture
Lab 4 Part 2: Datapath/Control Unit Integration and system testing
Winter 2023

Introduction

The datapath designed in the previous part of this lab contains 10 control signals. The function of the control unit, which is the focus of this part, is to produce the correct values for all the control signals at the proper point in time. Tables 1 and 2 lists the 10 control signals and summarizes their operation.

Table 1: Single bit control signals.

Control signal	value = 0	value = 1
reg_write	do not write into register file	write into register file
reg_dst	rt is the destination register	rd is the destination register
reg_in_src	d_out of data_cache is the d_in to the register file	ALU output is the d_in to the register file
alu_src	out_b of register file (rt) is the y input of the ALU	sign extended immediate is the y input of the ALU
add_sub	ALU operation = addition	ALU operation = subtraction
data_write	do not write into data cache	write into data cache

Table 2: Two bit control signals.

Control signal	value = 00	value = 01	value =10	value = 11
logic_func	AND	OR	XOR	NOR
func	load upper immediate	set less	arithmetic	logic
branch_type	no branch	beq	bne	bltz

Table 2: Two bit control signals.

Control signal	value = 00	value = 01	value =10	value = 11
pc_sel	no jump (PC+1, or PC+target address if branch condition is true)	jump (PC = target address)	jump register (PC = rs)	not used

Table 3 lists the 20 instructions implemented by the CPU together with the values of the 6 bit opcode field and the 6 bit func field (contained within the instruction as per Figure 1 of Lab 4) together with the 10 control signals. Table 3 is **partially** completed, you are to **complete** the table by deriving the values of the 10 control signals based upon Tables 1 and 2 and knowledge of which control signals need to be activated during a particular instruction in order to achieve correct execution of the instruction. Refer to your datapath of Part 1 to assist in completing the table.

Table 3: 20 instructions with opcode and function fields and control signals. [1]

Inst.	op	func	reg_write	reg_dst	reg_in_src	alu_src	add_sub	data_write	logic_func	func	branch_type	pc_sel
`lui`	001111		1	0	1	1	0 (don't care)	0	00 (don't care)	00	00	00
`add`	000000	100000	1	1	1	0	0	0	00	10	00	00
`sub`	000000	100010										
`slt`	000000	101010										
`addi`	001000		1	0	1	1	0	0	00	10	00	00
`slti`	001010											
`and`	000000	100100	1	1	1	0	1	0	00	11	00	00
`or`	000000	100101										
`xor`	000000	100110										
`nor`	000000	100111										
`andi`	001100											
`ori`	001101											
`xori`	001110											
`lw`	100011		1	0	0	1	0	0	10 (don't care)	10	00	00

Table 3: 20 instructions with opcode and function fields and control signals. [1]

Inst.	op	func	reg_write	reg_dst	reg_in_src	alu_src	add_sub	data_write	logic_func	func	branch_type	pc_sel
sw	101011											
j	000010											
jr	000000	001000										
bltz	000001											
beq	000100		0	0	0	0	0	0	00	00	01	00
bne	000101											

Note that Table 3 has some entries which are “don’t care” values which have arbitrarily assigned with values.

Procedure

Complete Table 3 by deriving all the remaining values of the control signals. Design the control unit using VHDL. The control unit in this implementation is a combinational logic circuit whose inputs are the opcode and function fields of the instruction and the outputs are the 10 control signals. Test your control unit (through simulation) to ensure that it generates the correct values for the 10 control signal for each of the 20 instructions. You may either use a VHDL process for the control unit which will be added to your datapath designed in Part 1, or you may design the control unit as a separate entity and use it as an additional component to be added with a port map statement to your existing datapath. Alternatively, the datapath of Lab 4 may be added as a component together with an instance of your control unit to create a new top level entity (with name cpu).

Use the following VHDL entity specification for the final CPU design:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_signed.all;

entity  cpu  is
port(reset : in std_logic;
     clk   : in std_logic;
     rs_out, rt_out : out std_logic_vector(3 downto 0);

    -- output ports from register file

     pc_out : out std_logic_vector(3 downto 0); -- pc reg
     overflow, zero : out std_logic);
end cpu;
```

You will note that in the cpu entity, the top-level ports consist of the out_a (rs) and out_b (rt) ports of the register file and the PC register. As was done in previous labs, only the low-order 4 bits of these registers shall be constrained to LEDs on the Nexys A7 FPGA board. The overflow and zero output ports will be constrained to available LEDs. There is an asynchronous reset (which will be constrained to a switch input) and a clock input (constrained to a switch on the FPGA board). You may make use of the provided lab4.xdc file. **It is important that you use the above entity specification, as it is the one which will be used during the lab quiz**.

Test your complete CPU by writing different test programs into the I-cache and **verify correct operation of your CPU through simulation. Explain your test programs.**

Questions:

1. In order to demonstrate an awareness of the limitation of tools and hardware (i.e. an FPGA device), examine the FPGA resource utilization report found in the file:

```
project_name/project_name.runs/impl_1/toplevel_entity_name_utilization_placed.rpt
```

to determine the amount of FPGA resources used by your implementation of the entire CPU. In particular, determine the amount of **Slice Logic**, **Registers**, and **SLICEL** and **SLICEM** used. These values will be found in several tables within the report. For example:

```
1. Slice Logic
--------------

+-------------------------+------+-------+-----------+-------+
|        Site Type        | Used | Fixed | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| Slice LUTs              |  559 |     0 |     63400 |  0.88 |
|   LUT as Logic          |  559 |     0 |     63400 |  0.88 |
|   LUT as Memory         |    0 |     0 |     19000 |  0.00 |
| Slice Registers         | 1157 |     0 |    126800 |  0.91 |
|   Register as Flip Flop | 1157 |     0 |    126800 |  0.91 |
|   Register as Latch     |    0 |     0 |    126800 |  0.00 |

2. Slice Logic Distribution
---------------------------

+--------------------------------------------+------+-------+-----------+-------+
|                 Site Type                  | Used | Fixed | Available | Util% |
+--------------------------------------------+------+-------+-----------+-------+
| Slice                                      |  442 |     0 |     15850 |  2.79 |
|   SLICEL                                   |  281 |     0 |           |       |
|   SLICEM                                   |  161 |     0 |           |       |
| LUT as Logic                               |  559 |     0 |     63400 |  0.88 |
|   using O5 output only                     |    0 |       |           |       |
|   using O6 output only                     |  540 |       |           |       |
|   using O5 and O6                          |   19 |       |           |       |
```

Note: Your actual values may be different depending upon your design of the CPU.

Based upon these values, comment on whether it would be possible to implement on the FPGA device used in the labs, a design consisting of 5, 10, 20, 100 CPUs? For the purposes of this question, it is only necessary to consider the instance of the CPU, we will **not** consider any sort of interconnection network used to communicate between the CPUs. In order to obtain meaningful answers, you are to implement several designs (1 CPU, 5 CPUs, 10 CPUs and tabulate the data and extrapolate from these values. The VHDL `port map` statement is a convenient method to create a design consisting of multiple instances of your CPU. It will **also be necessary** to set a `DONT_TOUCH` VHDL **synthesis attribute** on the instantiated instances of the CPU components to prevent the logic synthesis tool from removing them during the optimization phase. To set a `DONT_TOUCH` attribute on a component instance, one first must declare and assign a value to the attribute. The VHDL syntax is:

```
attribute DONT_TOUCH : string;
attribute DONT_TOUCH of U0 : label is "TRUE";
```

The `DONT_TOUCH` attribute can either have value of "TRUE" or "FALSE". In the above, `U0` corresponds to the component instance name. One can next create a top-level entity which consists of a specific number of CPUs instantiated as component instances with several port map statements:

```
entity two_cpu is
port(reset : in std_logic;
     clk   : in std_logic); -- intentional that there are no
end  two_cpu;               -- output ports (for simplicity)

architecture rtl of  two_cpu  is

component cpu is
port(reset : in std_logic;
     clk   : in std_logic;
     rs_out, rt_out : out std_logic_vector(3 downto 0);
     pc_out : out std_logic_vector(3 downto 0);
     overflow, zero : out std_logic);
end component ;

-- declare internal signals used in the component
-- port map statements

signal rs_out0, rs_out1 : std_logic_vector(3 downto 0);
signal rt_out0, rt_out1 : std_logic_vector(3 downto 0) ;
signal pc_out0, pc_out1 : std_logic_vector(3 downto 0) ;
signal overflow0, overflow1 : std_logic;
signal zero0, zero1 : std_logic;

for MICK, KEITH :  cpu use entity WORK.cpu(rtl);
```

```
attribute  DONT_TOUCH : string;
attribute DONT_TOUCH of MICK : label is "TRUE";
attribute DONT_TOUCH of KEITH : label is "TRUE";

-- the DONT_TOUCH attribute is needed for all the  instances
-- of the 'cpu' component, otherwise during implementation the
-- optimizer does some "decloning" of all  into a single instance
-- and the implemented design consists of only 1 CPU with the
-- resource utilization  the same as that of just the
-- original single CPU design

begin

-- component instantiation

MICK: cpu port map(reset => reset, clk => clk, rs_out => rs_out0,
rt_out => rt_out0, pc_out => pc_out0, overflow => overflow0, zero
=> zero0);

KEITH: cpu port map(reset => reset, clk => clk, rs_out => rs_out1,
rt_out => rt_out1, pc_out => pc_out1, overflow => overflow1, zero
=> zero1);

end  rtl;
```

One may use a similar approach to design an entity with 5, 10, or more CPUs. It should be noted that within the top-level entity `two_cpu`, there are no output ports. This was intentionally done to simplify the design. Note how the internal signals are simply left "dangling". Alternatively, one may leave include the output ports within the `two_cpu` entity, and then simply logically OR the internal signals with the output of each OR gate driving the designated output port. For example,

```
rs_out <= rs_out0 or rs_out1;
rt_out <= rt_out0 or rt_out1 ;
pc_out <= pc_out0 or pc_out1;
overflow <= overflow0 or overflow1;
zero <= zero0 or zero1;
```

However, this will cause additional hardware to be created, thus it is easier to simply have no output ports (and the `DONT_TOUCH` attribute will preserve the instantiated instances).

2. Can you think of any other limitations when synthesizing a design with a software logic synthesis tool?

Hint: What resources does software require when executing on a computer?
Hint Hint: Einstein's Theory of Relativity provided new insight into the nature of space and ***time***.

Hint Hint Hint: *"**Memory** is a stranger,*
History's for fools,
The president is a fool
Why do I have to keep reading these technical manuals"
-Roger Waters, Perfect Sense Part 2.

Requirements

1. Modelsim simulation results for the complete CPU showing execution of one example of each class of instruction: arithmetic and logic with and without immediate operands, conditional branches, unconditional jumps, and memory access instructions.

2. The VHDL code for the complete CPU (including the control unit VHDL code).

3. Synthesis and implementation log files (runme.log) as generated by Xilinx Vivado.

4. The answers to the Questions.

Report Submission

The Lab 4 report and demo are due the week of the lab quiz for each respective lab section. The demo (if not done during the Lab 4 session) is to be performed at the BEGINNING of the lab quiz session. **It is in the best interest of each student to complete Lab 4 prior to performing the lab quiz as the lab quiz will involve a demo of the CPU for a provided machine code program.**

References

1. *Computer Architecture, From Microprocessors to Supercomputers*, Behrooz Parhami, Oxford University Press, ISBN 0-19-515455-x, 2006, p251.

Ted Obuchowicz
November 14, 2016.

Revised: Nov. 7, 2017.

Revised: Nov. 11, 2021. Updated for Xilinx Vivado and Nexys A7 FPGA board. T. Obuchowicz.

Revised: Feb. 21, 2023. Revised for merging of Lab 4 and 5 due to 12 week semester.

Revised: March 15, 2024. Added questions on awareness of limitation of engineering tools and DONT_TOUCH synthesis attribute.