Lab 4 – A mini computer system design

Instructor: C.-K. Cheng
November 23, 2005
Agenda

- **Administrative**
  - Due on 12/02 before discussion session
  - 5% bonus on lab 4 if you submit on next Wednesday

- **Today**
  - A more detailed system diagram
  - How to design
    - instruction memory using read-only memory (ROM)
    - Interface signals using bus
    - Instruction decoder (control block) using VHDL module
Program Counter

- Program counter (PC) is a memory address pointer.
- Implementation-wise, PC is nothing more than a binary counter.
- The instruction memory is designed to hold 16 instructions, hence the PC output is a 4-bit address $\text{addr}[3:0]$, which cycles through 0 to 15.
Instruction Memory

- Instruction memory (IM) holds the binary encoding of the instructions.
- The size of IM is 16x7
  - 16 instructions in total
  - Each instruction is 7-bit wide
- To simply the implementation, we use read-only memory (ROM) to realize the IM.
Instruction Decoder

- Instruction Decoder (ID) generates the control signals for datapath and registers.
- ID is a bunch of combinational logic, and the most convenient way to design it is using VHDL module.

<table>
<thead>
<tr>
<th>Command</th>
<th>3-bit Inst.</th>
<th>4-bit Data</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Init</td>
<td>000</td>
<td>XXXX</td>
<td>Initialize all registers, i.e. R1=R2=R3=(0,0,0,0,0).</td>
</tr>
<tr>
<td>2 Move1</td>
<td>001</td>
<td>aaaa</td>
<td>Put aaaa value into register R1.</td>
</tr>
<tr>
<td>3 Move2</td>
<td>010</td>
<td>bbbb</td>
<td>Put bbbb value into register R2.</td>
</tr>
<tr>
<td>4 Store</td>
<td>011</td>
<td>XXXX</td>
<td>Take the previous result in R3 and store into register R1.</td>
</tr>
<tr>
<td>5 Add</td>
<td>100</td>
<td>XXXX</td>
<td>Add contents in R1 and R2. Present the result at R3 and output overflow flag.</td>
</tr>
<tr>
<td>6 Comp</td>
<td>101</td>
<td>XXXX</td>
<td>Compare the contents of R1 and R2. The flag bit is false iff R2 &gt; R1.</td>
</tr>
<tr>
<td>7 Shift</td>
<td>110</td>
<td>aaXX</td>
<td>Left rotate the content in R1 by aa bits, then store the result in R3.</td>
</tr>
<tr>
<td>8 Mask</td>
<td>111</td>
<td>XXXX</td>
<td>Mask (AND-ing) R1 and R2. Present the result in R3.</td>
</tr>
</tbody>
</table>
Datapath

- Datapath carries out the arithmetic operations.
- The operands are always from R1 and R2; the result is always stored in R3 except store command.
- Five arithmetic operations; use library components.
  - Load/store
  - Addition: ADD4
  - Shift: BRLSHFT4
  - Compare: COMPM4
  - Mask: bit-wise AND of the two operands
Registers

- Analysis of register usage
  - Move: load memory to R1, R2
  - Add, shift, mask: produce result in R3
  - Store: load R3 to R1
  - Each register is enabled only when it’s needed.
Suggestions

- Divide and conquer
  - Design the whole architect and interface signals first, and then work down to individual modules.

- Sketch your design on paper before going to Xilinx.
Instruction Memory Design

- Memory modules in Xilinx
  - RAM (Random Access Memory): Writable
  - ROM (Read-Only Memory)
- We will use ROM, hence the instructions are preloaded before the machine starts.
The data output (O) reflects the bit selected by the 4-bit address (A3 – A0).

The ROM is initialized to a known value during configuration with the INIT=value parameter.
IM Design III

- 16x7 memory block
16x7 instruction memory initialization

Init
Move1 0100
Move2 0110
Add

Stored in a ROM16x1 module

<table>
<thead>
<tr>
<th>Id of the ROM16x1 module</th>
<th>Content (hexadecimal)</th>
<th>Id of the ROM16x1 module</th>
<th>Content (hexadecimal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>4</td>
<td>0006</td>
</tr>
<tr>
<td>1</td>
<td>0004</td>
<td>5</td>
<td>0004</td>
</tr>
<tr>
<td>2</td>
<td>0002</td>
<td>6</td>
<td>0000</td>
</tr>
<tr>
<td>3</td>
<td>0000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
How to use bus in Xilinx

- Create a new schematic
- Before you place anything in the schematic, click *Tools -> create I/O markers*
You’ll see two I/O buses on the canvas.

Select the ROM16x1 module from the library and place multiple instances on the canvas.

Extend two I/O buses before and after the ROM modules by using “add wire” button (You will see thicker wires).
Add bus taps by using “Add Bus Tap” button.
You can change the direction of by selecting the orientation in the options window.
Connect the taps to module pins by wire.
Click on “Add Net Name” then type the net name in the options window. Now you will see the name appear after the cursor.

Click on the wire you want to name.
Name all the nets

Double click on a ROM module, the property window will pop up. Change the INIT value and make it visible.
Click OK. You will see the initial value appears.

Change the initial values for other ROM modules and save the diagram.
Create a symbol for the instruction memory block.
Instruction Decoder Design

- Create a new source, select **VHDL module**
In the next window, you can type in your input/output signals for the instruction decoder, or you can skip it.
Next you’ll see a VHDL template which does nothing.

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity instr_decoder is
end instr_decoder;

architecture Behavioral of instr_decoder is
begin

end Behavioral;
```
ID Design IV

- Define the input and output signals of your instruction decoder

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if instantiating any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.USComponents.all;

entity instr_decoder is
Port ( Instr : in std_logic_vector(4 downto 0);
clr : out std_logic;
ctrl1_sig1 : out std_logic;
ctrl1_sig2 : out std_logic;
ctrl1_sig3 : out std_logic;
 sel : out std_logic_vector(1 downto 0));
end instr_decoder;

architecture Behavioral of instr_decoder is
begin

end Behavioral;
```
ID Design V

Define the behavior the instruction decoder using process clause in VHDL

```vhdl
architecture Behavioral of instr_decoder is
begin
process (instr)
begin
  if (instr(4 downto 2)="000") then
    clr <= '1';
    ctrl_sig1 <= '0';
    ctrl_sig2 <= '0';
    ctrl_sig3 <= '0';
    sel <= "00";
  end if;
  if (instr(4 downto 2)="001") then
    clr <= '0';
    ctrl_sig1 <= '0';
    ctrl_sig2 <= '0';
    ctrl_sig3 <= '0';
    sel <= "01";
  end if;
  ...
end process;
end Behavioral;
```
ID Design VI

- Save and select the VHDL file

- Under the Design Utilities category in Process View, double click on Create Schematic Symbol

- Now the instruction decoder has been created and can be referenced in other designs
Demo of the addition operation

Show the simulation results for each of the 8 instructions.
The end.