Control Logic and Data paths

(disclaimer – I have not simulated/tested the examples presented in class today)

Data path / control path partitioning

It is not sufficient to describe a sequential systems with one or more FSMs. It is common practice to partition a system in to control logic and data paths.

Advantages

• A better structural and logic decomposition of a system
• Design reuse of common data path blocks
• Efficient for CAD tools. Simplifies synthesis, place and route.

Design Methodology

• We need to follow som kind of design methodology when designing digital systems
  » How do we specify the system?
  » How do we go from a system description to a RTL description?
  » How do we partition the system?
  » How do we separate the control logic from the data path?
  » How do we document the system?
• We need to use a structured design methodology and to ”think hardware”

”Think hardware” - Que?
Thinking hardware...

- VHDL can be used just like any programming language. Was initially intended for modeling hardware, so only a subset of the language can be used for synthesis. Using VHDL “just like any other programming language” is not “thinking hardware”
- Hardware is physically structured, this should be reflected in your VHDL code. Hierarchy (components) and processes allows structuring of your code. “A process is a piece of hardware”.
- Thinking hardware – think of:
  » Data path components (adders, muxes, registers, memory etc)
  » Control path components (FSMs)
- You must follow some sort of design methodology - a methodology that is different from software development

"ASM chart metod" – Lee’s design methodology

1. Create an algorithm, using pseudocode, to describe the desired operation of the device (component)
2. Convert the pseudo code into an ASM chart
3. Design the datapath based on the ASM chart
4. Create a detailed ASM chart based on the data path
5. Design the control logic based on the detailed ASM chart

My comments: The Lee method is good, but don’t treat the text book as a Bible. In this course we will not follow this design methodology strictly. But we will study the "ASM method"

For a complex system you will need several ASM charts

Example design

- Specification
  » A synchrononous digital device should have inputs C, X[2:0], Clk and Reset. There is one output named B[2:0]
  » In the initial state X[2:0] should be loaded into the device. Input C then determines if a multiply by 2 or a divide by 2 is performed in the next clock cycle. If a divide by two is performed the LSB is tested so that a round up is always performed. The process is then repeated

- Pseudo code in Register Transfer Notation (RTN)

1. \( B \leftarrow X \) // Load input
2. if \( C = 0 \) // Multiply by two
   B \leftarrow B[1:0] \& '0'
else if \( B[0] = '0' \) // Divide by two
   B \leftarrow '0' \& B[1:0]
else // Divide by two and round up
   B \leftarrow ('0' \& B[1:0]) + 1
end if
3. goto step 1

RTN
Derive ASM chart from pseudocode

1. \( B \leftarrow X \)
2. if \( C = 0 \)
   \( B \leftarrow B[1:0] \land '0' \)
   else if \( B[0] = '0' \)
   \( B \leftarrow '0' \land B[2:1] \)
   else
   \( B \leftarrow ('0' \land B[2:1]) + 1 \)
   end if
3. goto step 1

Derive "detailed ASM chart"

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity Ex is
port ( 
  Clk, Reset, C : in std_logic;
  X : in std_logic_vector(2 downto 0);
  B : out std_logic_vector(2 downto 0); 
end Ex;

architecture RTL of Ex is
  signal B_reg : std_logic_vector(2 downto 0);
  signal MX_out : std_logic_vector(2 downto 0);
  signal P_state, N_state : integer range 0 to 3;
begin
  Regs : process(Clk)
  begin 
  if rising_edge(Clk) then
  if Reset = '1' then
  P_state <= 0;
  else
  P_state <= N_state;
  end if;
  B_reg <= MX_out;
  end if;
  end process;
  B <= B_reg;

VHDL code

--- N-state decoding

Comb : process(P_state, C, X) begin
  case P_state is
  when 0 =>
  if C = '0' then
  N_state <= 1;
  elsif X(0) = '0' then
  N_state <= 3;
  else
  N_state <= 2;
  end if;
  when '1' =>
  N_state <= 0;
  end case;
end process;

MX_out : with P_state select
  MX_out <= X when 0,
  B_reg(1 downto 0) \& '0' when 1,
  std_logic_vector(unsigned('0' \& B_reg(2 downto 1)) + 1) when 2,
  '0' \& B_reg(2 downto 1) when 3;
end RTL;
Synthesis results

Design methodology where we skip some RTN stuff

1. Understand the specifications
2. Specify the interface (inout and output ports)
3. Identify what data path elements are needed
4. Draw a block diagram show the data path elements and interconnect
5. Identify necessary control signals for data path elements
6. Control signals specify the interface of the control path
7. Derive ASM chart(s) for the FSM(s) in the control path
8. Add the control path to block diagram
9. Now (not before) describe your design in VHDL. Use a structural approach
10. Simulation, synthesis, implementation...

Sample problem specification

- Create a synchronous RAM block that has a “zeroing” capability (writing “00000000” to memory). Memory do not have a reset ports.
- If Zero input asserted, assert Busy output and zero RAM block
- Can load a Low value, and High value that will set the range of the memory to zero
- RAM size will 64 x 8

This example origins from lecture notes for EE 4743/6743 Digital System Design at Mississippi State University. Original author Professor Bob Reese. I have slightly modified the example. Original documentation can be found here: http://www.erc.msstate.edu/~reese/EE4743/index.html

Interface

- Inputs
  - Clk, Reset
  - LowLoad - load LOW value; will be taken from address bus
  - HighLoad - load HIGH value; will be taken from address bus
  - WrEn – RAM write enable
  - Din[7..0] - Data bus to RAM
  - Addr[5..0] - Address bus to RAM.
  - Zero - start a zero cycle

- Outputs
  - Dout[7..0]
  - Busy - when assert, busy zeroing RAM
Identification of data path elements

• Two registers to hold Low and High value
• 6-bit loadable counter to cycle address lines of RAM
  » Counter needs to be loaded with LOW value when we start to zero the RAM
• A comparator to compare counter value and High value to determine when zeroing operation is finished
• RAM
• Muxes

Data path block diagram

What control lines do we need from control path? (look at each data path component)

• Registers
  » Load lines for LowReg and HighReg registers driven externally and not under FSM control
• Counter
  » Load (synchronous load)
  » CountEn (count enable)
• Mux Selects
  » When doing ‘zero’ operation, counter will be driving RAM address lines and RAM input data line will be zero. The same select line can drive both muxes.
• RAM
  » The WE (write enable) of the RAM needs to be an OR of the external WE and a WE that is provided by the control path

Control logic

• Inputs
  » Clk, Reset
  » Zero – starts zero operation
  » CountEq – AEQB from comparator
• Outputs
  » Busy – busy flag set during zeroing
  » Sel – Mux select line for data and address muxes
  » CountEn – synchronous counter enable
  » LoadCount – synchronous load counter
  » WrEn – Write Enable to RAM (should be OR’ed with external RAM WE)
Control logic – a single FSM will do the work

- Wait for Zero command (FSM simply waits for Zero input to be asserted).
- Load the counter with the Low value
- Write ‘0’ data value to RAM via address specified by counter, incrementing counter each clock cycle. Stop writing when High register value equals counter value.
- Three DISTINCT operations, need three STATES in FSM.

ASM Chart

- State S0 waits for Zero operation. In this state the external Addr, Data and WrEn lines are muxed to RAM. Busy flag is ‘0’
- State S1 loads counter with LowReg value. Busy flag is ‘1’
- State S2 does zero operation. Exit this state when counter value equals to HighReg value. Counter is enabled and muxes are set to zero RAM. Busy flag is ‘1’

Data and control path

VHDL coding

- Once the data path block diagram and ASM chart is done, the “design” work is done. What is left is VHDL coding.
- Decide what parts of the data path will be implemented in VHDL and what parts using Core Generator
  » Core Generator will provide you with an optimized core
  » But, your design will not be portable
- Do the data path first, then the control logic!
  » Sometime just hooking up the data path elements will expose a flaw in your reasoning
- If your design is “large” you should use hierarchy. Most likely you will need multiple control logic blocks (FSMs)
- Be systematic when you code. Simulate and debug smaller blocks of your design.
- If you are working with FPGAs test sub circuits in your design before you put everything together