Solutions to Statistic Module

**Outline**
- Block Diagrams
- Theory of Operation
- Major Entities
- Internal Signals
- Delayed signals
- Logic
- Multiplexers
- Memory Initialization
- Examples of Normal Operation
HW 2 / HW3 – Basic Block Diagram

Part 2: Statistics Counter Design

Select this data if attempt to write and read same address

Two instances of cntr_ram.vhd

Good Idea to flop inputs and outputs of Block RAM.

Counter to reset memory contents to zero.

cntr_ready gets set to ‘1’ when this reaches xFF

Select this data if attempt to write and read same address

CSE/CoE 535 : Lockwood

HW3 - Detailed Block Diagram

CSE/CoE 535 : Lockwood
HW 3 : Entity of Statistics Module (given)

-- This file ties a very simple increment/read interface for counters
-- stored in block RAM.

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity stat_mod is
  port (clk : in std_logic;
         reset_l : in std_logic;
         inc_event_1 : in std_logic;
         cntr_read : in std_logic;
         event_1_num : in std_logic_vector(7 downto 0);
         data_strobe : out std_logic;
         cntr_data : out std_logic_vector(31 downto 0);
         cntr_ready : out std_logic);
end stat_mod;

HW 3 : Entity of Block RAM (given)

component cntr_ram
  port (clka : in std_logic;
         clkb : in std_logic;
         addra : in std_logic_vector(7 downto 0);
         addrb : in std_logic_vector(7 downto 0);
         dinb : in std_logic_vector(15 downto 0);
         web : in std_logic;
         ena : in std_logic;
         enb : in std_logic;
         douta : out std_logic_vector(15 downto 0));
end component;
HW 3 : Black Box Declaration

- Inform synthesis tool to component has already been synthesized

```
-- Synplicity black box declaration
attribute syn_black_box : boolean;
attribute syn_black_box of cntr_ram: component is true;
```

HW 3 : Internal Signals

```
-- Increment Signals
signal event_1_num_sav        : std_logic_vector(7 downto 0) ;
signal inc_event_1_sav        : std_logic ;
signal cntr_read_sav          : std_logic ;

-- Address Signals
signal addr_delay, addr_delay2 : std_logic_vector(7 downto 0) ;
signal addr_mux_out, addr_write : std_logic_vector(7 downto 0) ;

-- Data Path Signals
signal data_from_mem, data_from_mem_delay : std_logic_vector(31 downto 0) ;
signal data_inc, data_mux_out, data_to_mem : std_logic_vector(31 downto 0) ;
signal data_to_mem_delay : std_logic_vector(31 downto 0) ;
signal data_from_mem_mux : std_logic_vector(31 downto 0) ;

-- Enable Signals
signal ena, ena_delay, ena_delay2 : std_logic ;
signal ena_delay3 : std_logic ;
signal we, data_strobe_int : std_logic ;
signal cntr_val : std_logic_vector(7 downto 0) ;
signal cntr_ready_int, cntr_ready_delay : std_logic ;
signal select_saved, addr_equal : std_logic ;
```
HW 3: Input Flops

-- Process Name: inoutflops
-- Sensitive to: clk
-- Description: This process is used to define all flip-flops on the boundary of the module.

inoutflops: process(clk)
begin
if (clk = '1' and clk'event) then
  if (reset_l = '0') then
    event_1_num_sav  <= x"00";
    inc_event_1_sav  <= '0';
    cntr_read_sav    <= '0';
    data_strobe      <= '0';
    cntr_data        <= x"00000000";
  else
    event_1_num_sav  <= event_1_num;
    inc_event_1_sav  <= inc_event_1;
    cntr_read_sav    <= cntr_read;
    data_strobe      <= data_strobe_int;
    cntr_data        <= data_from_mem;
  end if;
end if;
end process inoutflops;

HW 3: Delay Flip/Flops

-- Process Name: delayflops
-- Sensitive to: clk
-- Description: This process is used to define all flip-flops interior to the module.

delayflops: process(clk)
begin
if (clk = '1' and clk'event) then
  if (reset_l = '0') then
    ena_delay          <= '0';
    ena_delay2         <= '0';
    ena_delay3         <= '0';
    data_strobe_int    <= '0';
    data_from_mem_delay<= x"00000000";
    data_to_mem        <= x"00000000";
    data_to_mem_delay  <= x"00000000";
    addr_delay         <= x"00";
    addr_delay2        <= x"00";
    addr_delay3        <= x"00";
    cntr_ready_delay   <= '0';
    select_saved       <= '0';
  else
    ena_delay          <= inc_event_1_sav;
    ena_delay2         <= ena_delay;
    ena_delay3         <= ena_delay2;
    data_strobe_int    <= cntr_read_sav;
    data_from_mem_delay<= data_from_mem_mux;
    data_to_mem        <= data_mux_out;
    data_to_mem_delay  <= data_to_mem;
    addr_delay         <= event_1_num_sav;
    addr_delay2        <= addr_delay;
    addr_delay3        <= addr_delay2;
    addr_write         <= addr_mux_out;
    cntr_ready_delay   <= cntr_ready_int;
    select_saved       <= addr_equal;
  end if;
end if;
end process delayflops;
HW 3: Counter to Initialize Memory

- Note that Reset is active low [reset on 0]
- On reset, cycle through all memory (a MUX will clear data)

```vhdl
-- Process Name: acntr
-- Sensitive to: clk
-- Description: This process defines an 8-bit counter which
-- is used to reset the memory contents.
------------------------------------------------------------
acntr: process(clk)
begin
  if (clk = '1' and clk'event) then
    if (reset_l = '0') then
      cntr_val <= x"00" ;
    elsif (cntr_ready_int = '0') then
      cntr_val <= unsigned(cntr_val) + 1 ;
    end if ;
  end if ;
end process acntr ;
```

HW 3: Signal when counter is ready

- Indicate that the counter is ready when all
memory addresses have been accessed.

```vhdl
ready: process(clk)
begin
  if (clk = '1' and clk'event) then
    if (reset_l = '0') then
      cntr_ready_int <= '0' ;
    elsif (cntr_val = x"FF") then
      cntr_ready_int <= '1' ;
    else
      cntr_ready_int <= cntr_ready_int ;
    end if ;
  end if ;
end process ready ;
```
HW 3 : Logic

--- Logic

ena <= inc_event_1_sav or cntr_read_sav;
we <= ena_delay3 or (not cntr_ready_delay);
data_inc <= unsigned(data_from_mem_delay) + 1;

addr_equal <= '1' when (addr_write = event_1_num_sav) and
                     (ena = '1') and
                     (we = '1')
                     else '0';

HW 3 : Data Multiplexers

--- Multiplexors

addr_mux_out <= addr_delay2 when cntr_ready_int = '1'
                else cntr_val;

data_mux_out <= data_inc when cntr_ready_int = '1'
                 else x"00000000";

data_from_mem_mux <= data_to_mem_delay when select_saved = '1'
                      else data_from_mem;

cntr_ready <= cntr_ready_int;
HW 3 : Port map of Block RAMs

-- Port Maps

upper_16: cntr_ram -- Upper 16 bits of the value
port map (clka => clk, clkb => clk, addra => event_1_num_sav, addrb => addr_write, dinb => data_to_mem(31 downto 16), web => we, ena => ena, enb => we, -- enable & port to write douta => data_from_mem(31 downto 16));

lower_16: cntr_ram -- Lower 16 bits of the value
port map (clka => clk, clkb => clk, addra => event_1_num_sav, addrb => addr_write, dinb => data_to_mem(15 downto 0), web => we, ena => ena, enb => we, douta => data_from_mem(15 downto 0));
end structure;

Example of Normal Operation

But still returned the correct result

Read Invalid Data from Memory
Example of Memory Initialization

Done
Resetting, cntr_ready goes to '1'

Address xFF set to 0

Synthesis Flow – MP1
Back end Flow

Acceptable period = 46.8 MHz

Hints and Suggestions to make MP1 work

- Check sensitivity list
  - Be sure that all signals you use in a process are listed
- Check that initial signals
  - Set values in VHDL processes, not a signal declaration
  - Statements like:
    ```vhdl
    signal name := value
    ```
    do not synthesize as you might expect!
- Check Warnings in Synplicity List of Errors/Warnings
  - Some warnings MUST NOT be ignored!
- Check that values for signals and outputs are assigned for ALL cases
  - Provide logic for `else` and `when others`
- Check that flow control is not backpressured
  - `tca_out_appl_int` should be ‘1’ after `cntr_ready` becomes ‘1’.