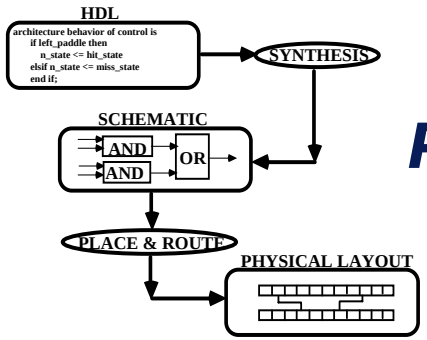
DESIGNING FPGAS & ASICS



Simulation and Testing

Prof. Don Bouldin, Ph.D.

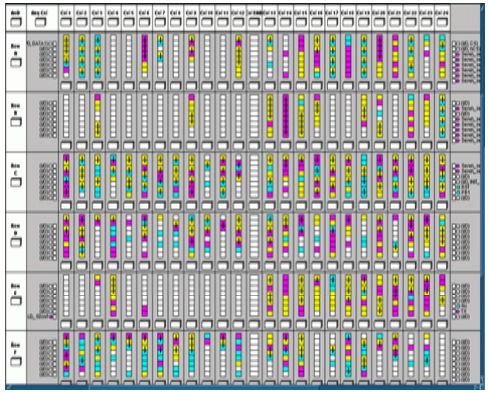
Electrical & Computer Engineering

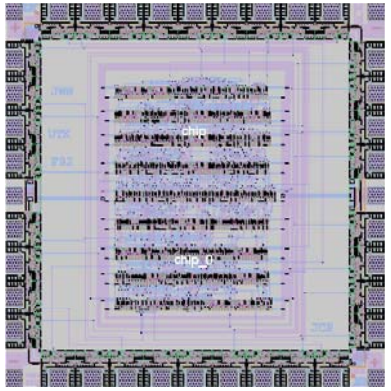
University of Tennessee

TEL: (865)-974-5444

FAX: (865)-974-5483

dbouldin@tennessee.edu





COURSE OUTLINE

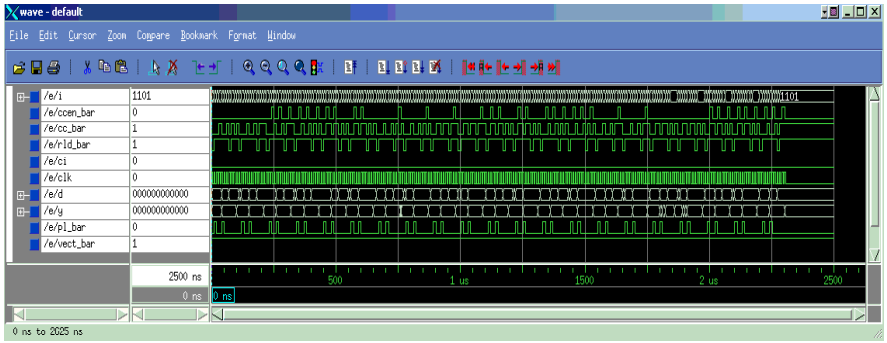
- Overview of FPGAs and ASICs
- Using Synthesis
- HDL Examples
- Simulation and Testing
- Physical Place and Route
- Testing ASICs
- Component Reuse

SEMI-CUSTOM DESIGN FLOW OF DIGITAL FPGAS/ASICS

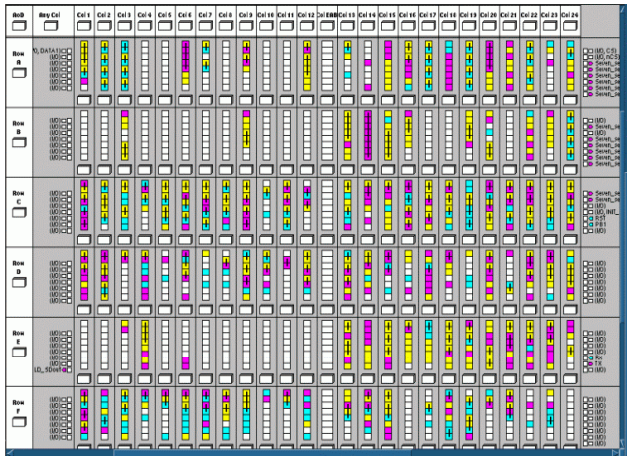
1—HDL

```
CASE w IS
   WHEN "00" => y <= "1000" ;
   WHEN "01" => y <= "0100" ;
   WHEN "10" => y <= "0010" ;
   WHEN OTHERS => y <= "0001";
END CASE ;
```

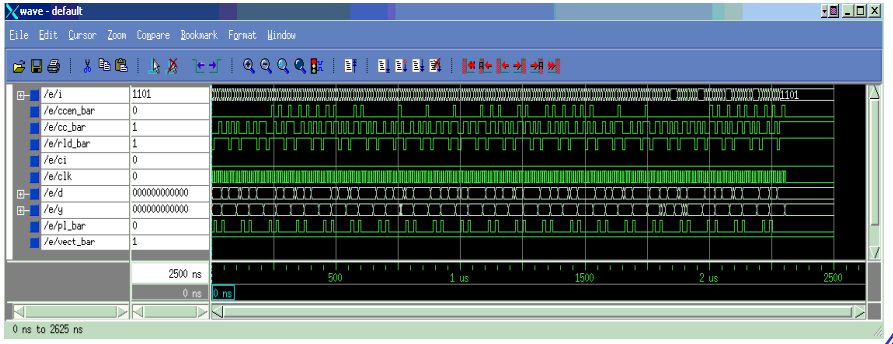
2--PRE-SYNTHESIS SIMULATION



3—SYNTHESIS/AUTO LAYOUT

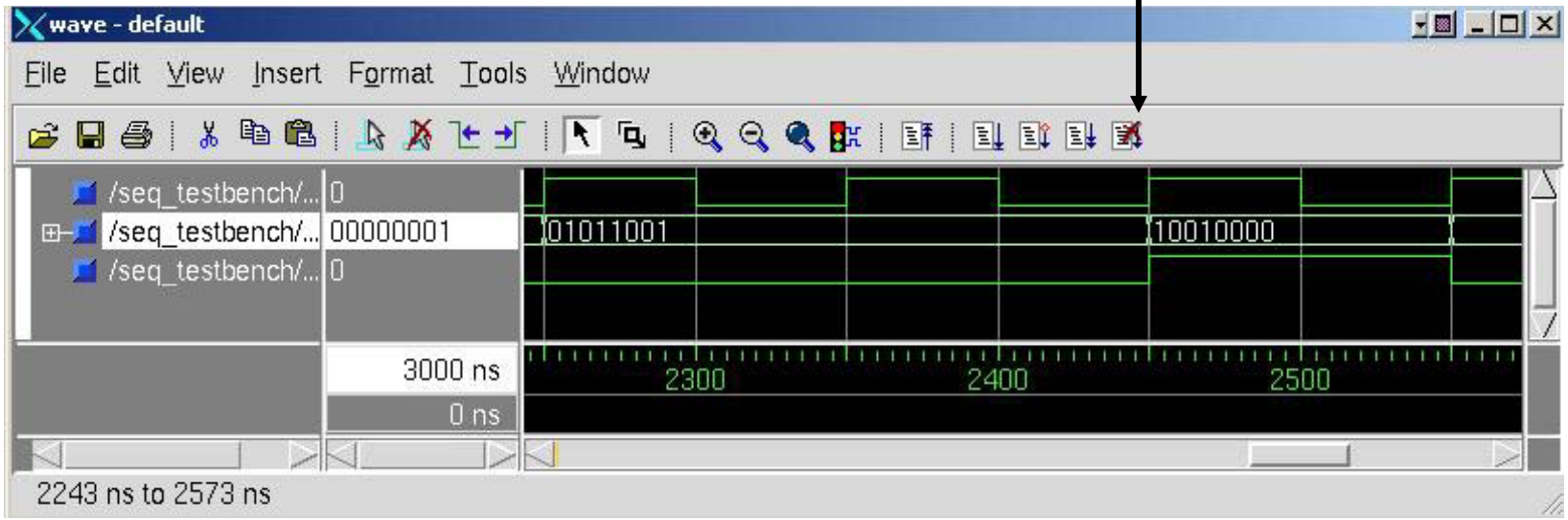


4--POST-LAYOUT SIMULATION



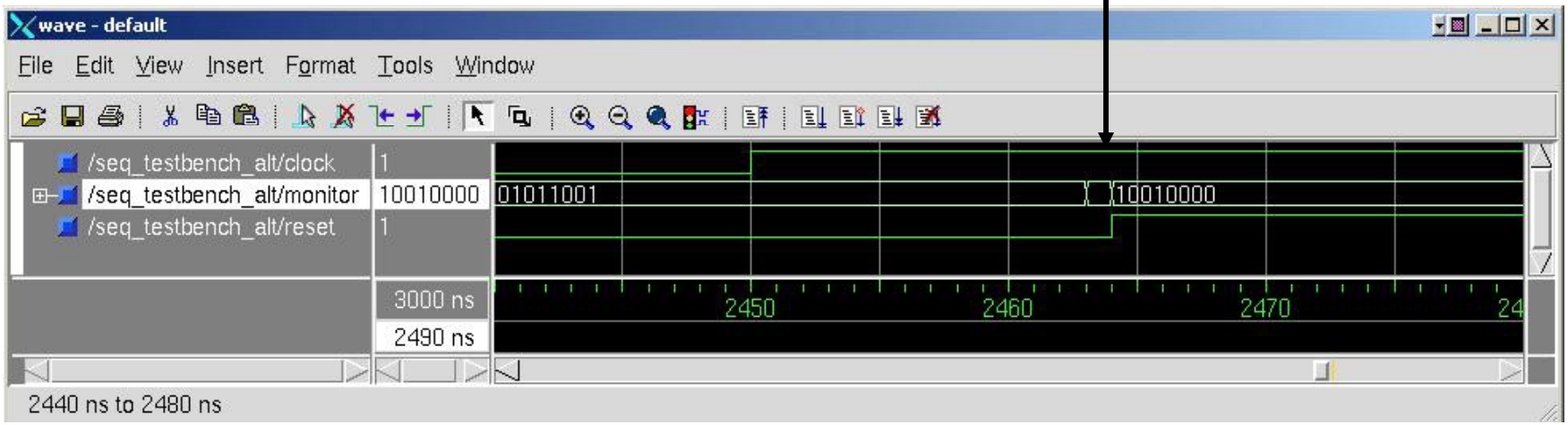
PRE-SYNTHESIS SIMULATION IS TECHNOLOGY-INDEPENDENT

Note the "zero" delay at 2450 ns:



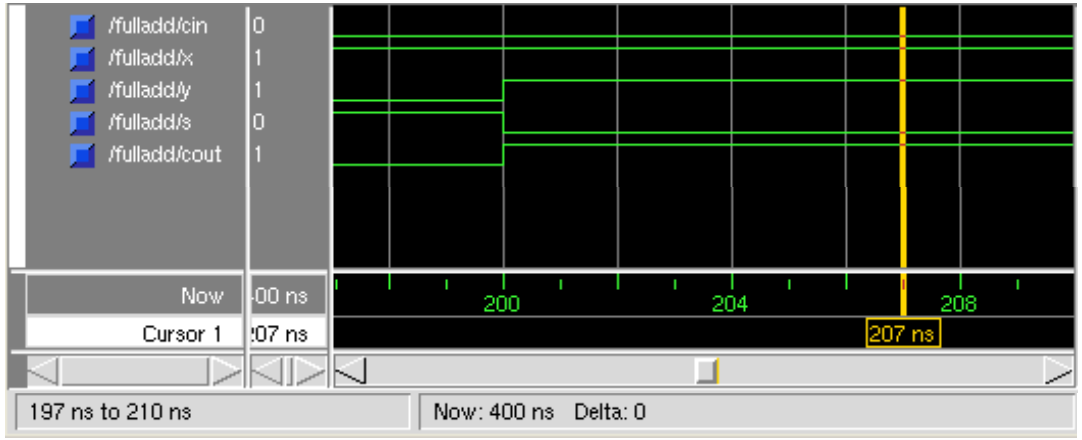
POST-LAYOUT SIMULATION INCLUDES COMPONENT AND WIRING DELAYS

Note the “14ns" delay at 2464 ns:

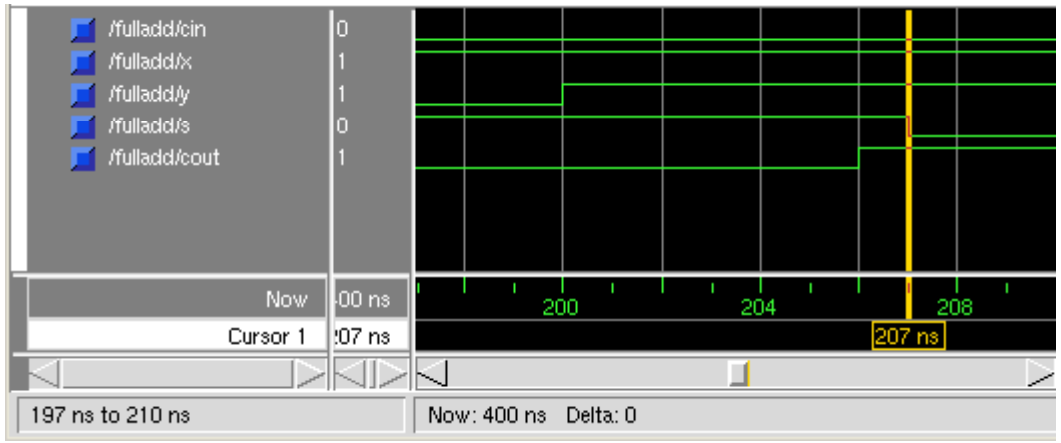


Simulation of fulladd.vhd

- **Pre-synthesis: outputs change instantly at 200 ns.**



- **Post-layout: outputs change at 206-207 ns.**



RECOMMENDED METHOD FOR LEARNING VHDL

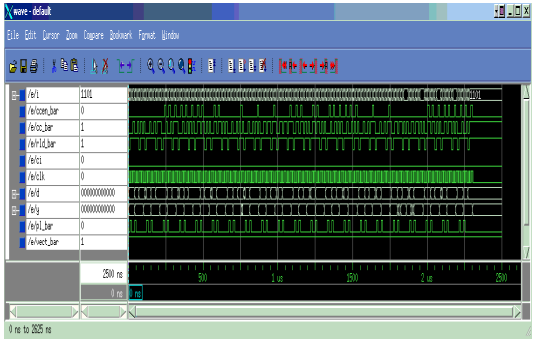
- Copy a known working file to use as a template with reserved words and syntax. Simulate it.
- Modify the VHDL source code to perform your intended operations and then simulate the revised code.
- Be wary of examples in textbooks and on the internet since not all VHDL code is synthesizable with a particular synthesis tool and software environment.
- Avoid use of the "FOR" statement which is confusing.
- Avoid using technology-dependent statements like "WAIT until 14 ns". Instead, use "WAIT_CLOCK(16)".

SIMULATION SHOWS THE RESPONSES OF THE VHDL TO STIMULI

STIMULI

RESPONSES

WAVEFORM

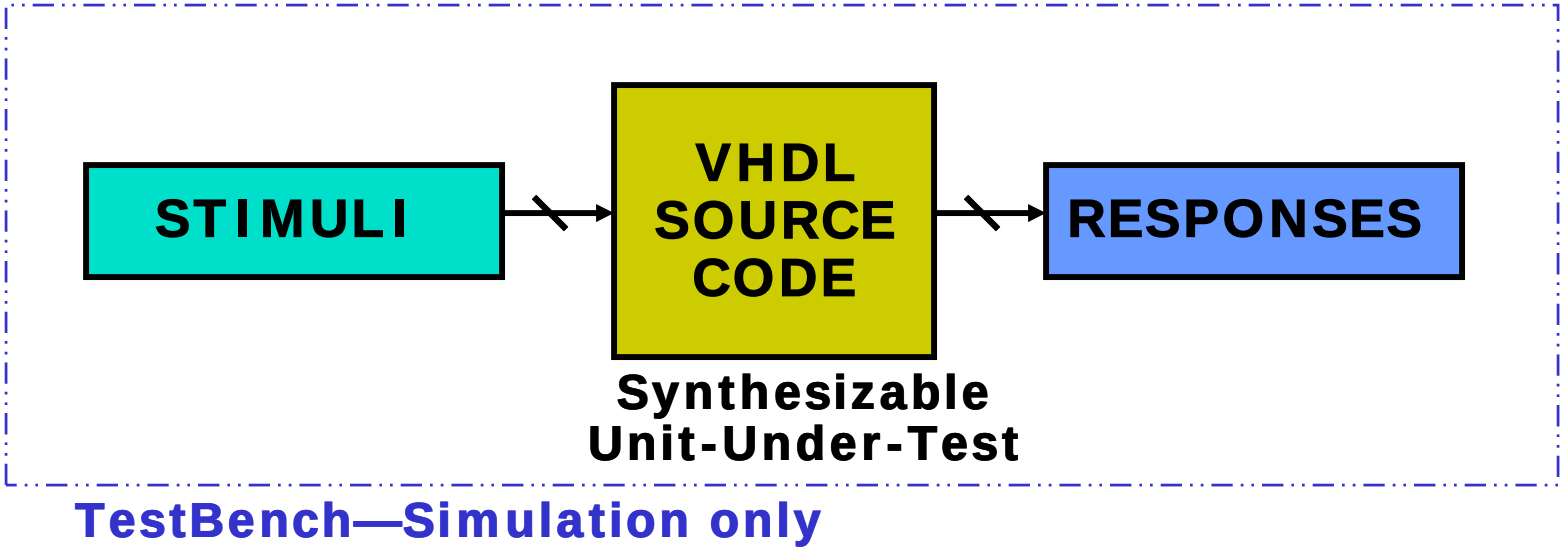


- **Functional stimuli are developed by the designer to mimic the system environment.**

VHDL SOURCE CODE

THE TESTBENCH CONTAINS THE STIMULI, RESPONSES AND UUT

- The testbench is written in VHDL but is **not** synthesized into the FPGA/ASIC.



EXPECTED RESPONSES ARE COMPARED TO ACTUAL ONES

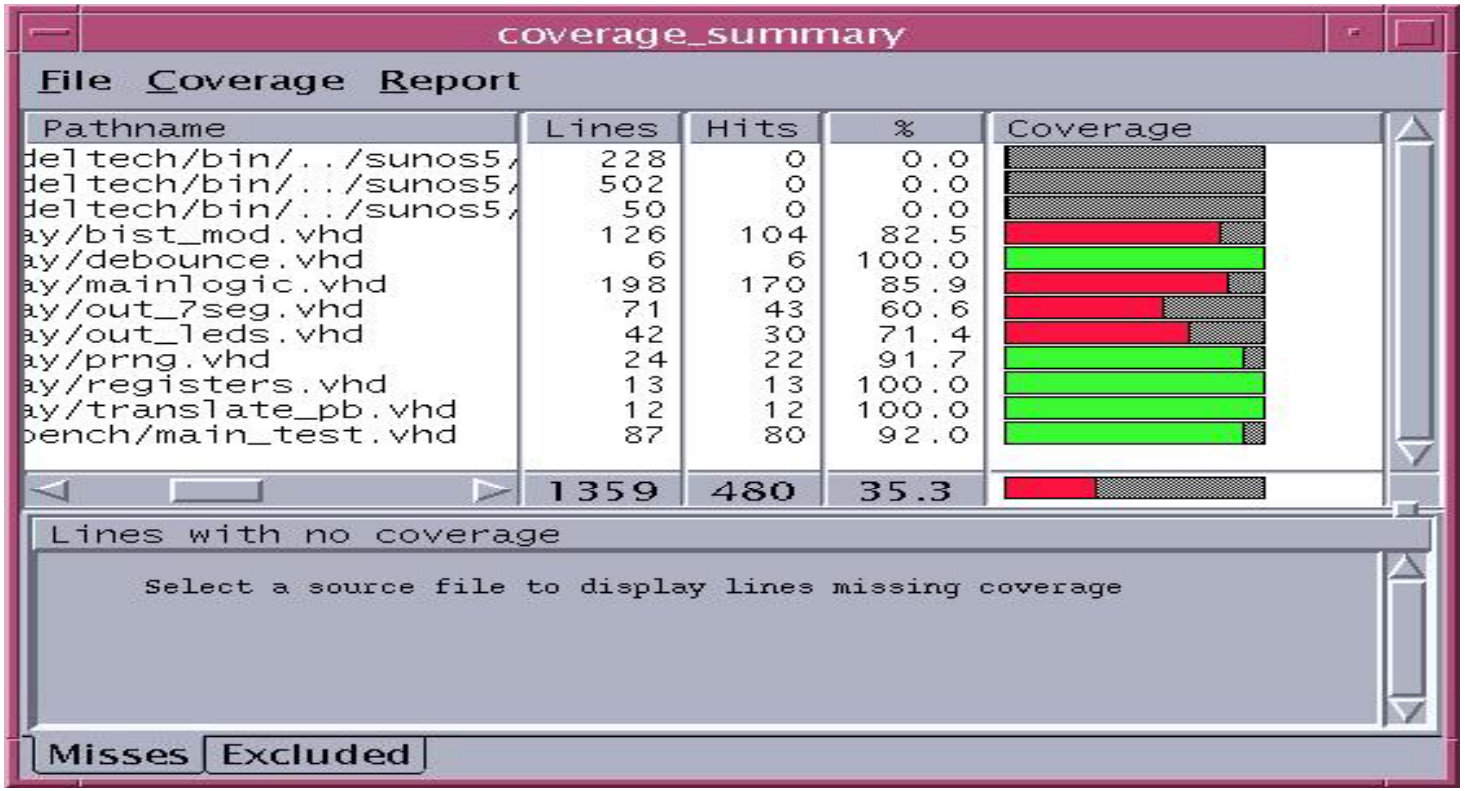
“Failure” stops simulation while “warning” does not.

“Note” is used to document a correct response.

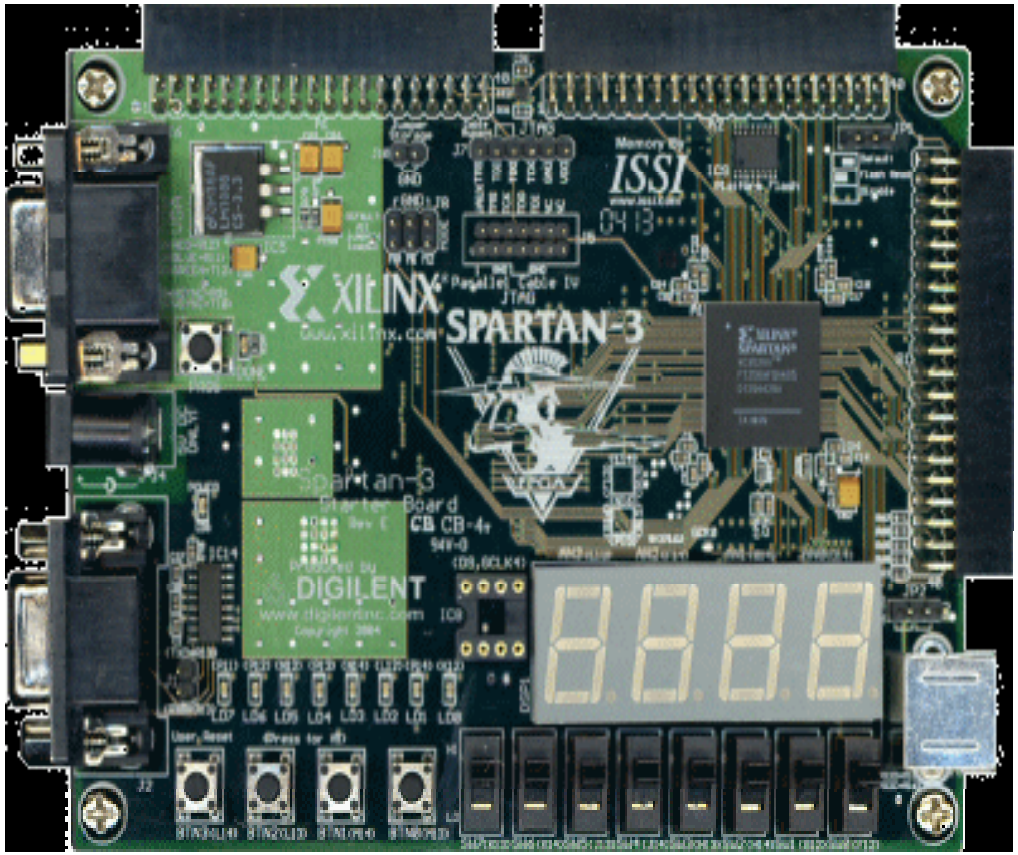
```
wait_clock(16);
IF (left_seg = X"6")
-- check second state of 7-segment display
THEN
ASSERT false
REPORT "Output signals set correctly (7-segment second state)"
SEVERITY note;
ELSE
ASSERT false
REPORT "Output not set correctly (7-segment second state)"
SEVERITY warning;
END IF;
```

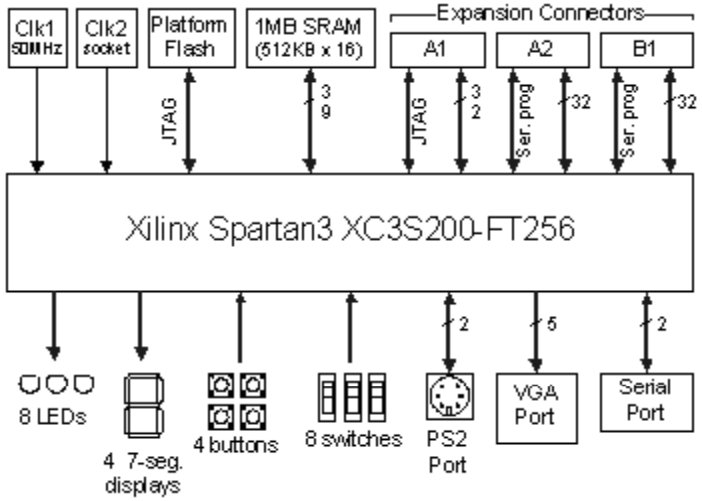
ALL VHDL SOURCE LINES SHOULD BE TESTED

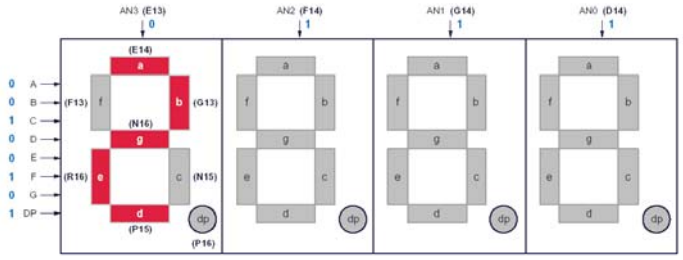
The simulator can produce coverage reports.



FINAL VERIFICATION IS PERFORMED USING A PROTYPING BOARD







Seven-Segment LED Digit Control

XILINX TOOLS ARE FREE BUT TARGET ONLY XILINX PARTS

- **Anyone can download Xilinx WebPACK tools for free from www.xilinx.com**

Free ISE *WebPACK* 7.1i

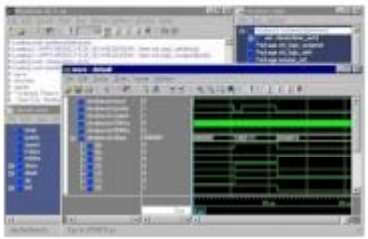
The free ISE *WebPACK*™ 7.1i is the most complete, easy-to-use software solution to complete a Xilinx CPLD or medium-density FPGA design

ISE *WebPACK* is the ideal downloadable desktop solution offering free software modules from ABEL and HDL synthesis to device fitting and JTAG programming. ISE *WebPACK* is a subset of our award winning ISE *Foundation*™ design tools providing instant access to the ISE tools at no cost. Xilinx has created a solution that allows convenient productivity by providing a design solution that is always up to date with error-free downloading and single file installation.

› Register

› Add to Cart

Model*Sim*™ Xilinx Edition-III



The Model*Sim*-III Family of Products

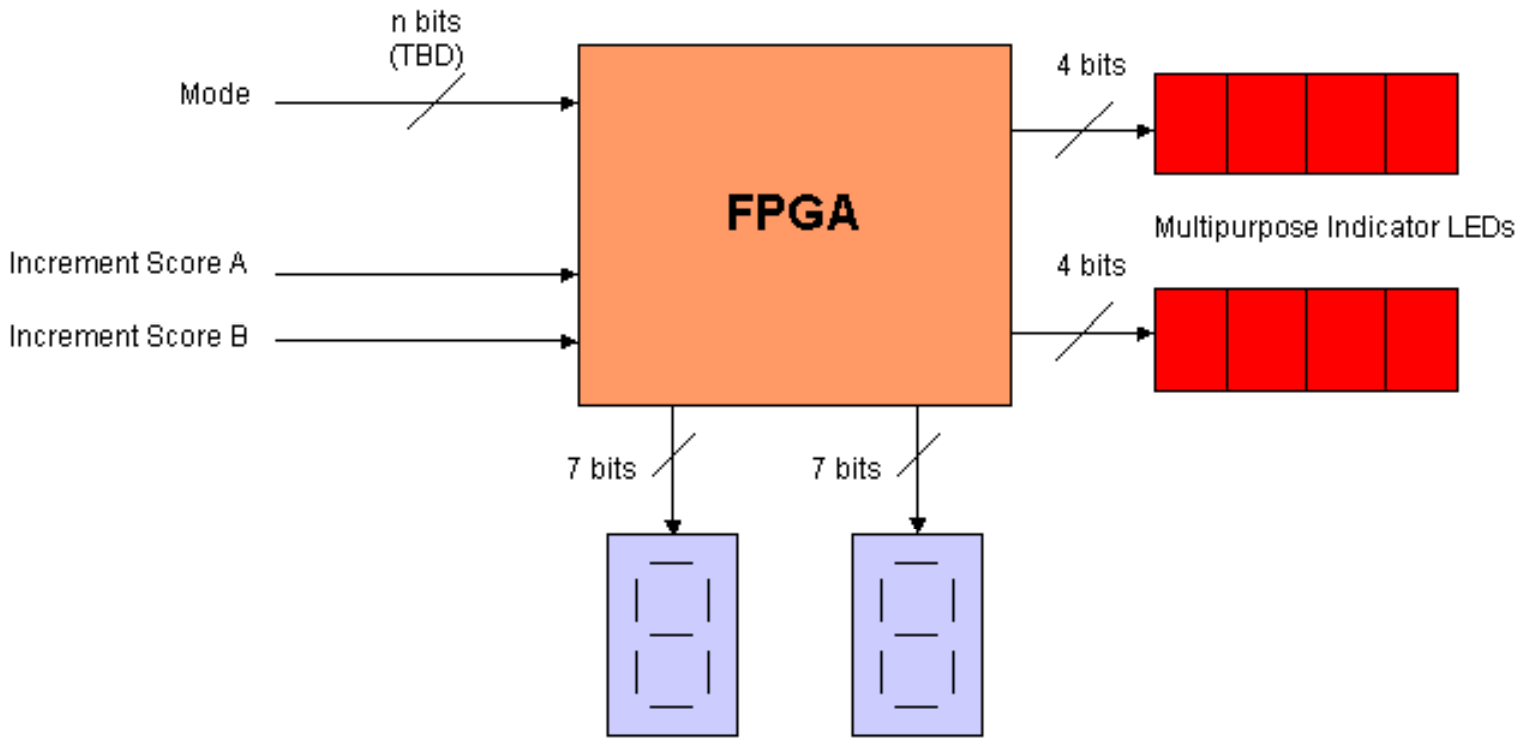
Product Name	Available From	
Model*Sim* XE-III Starter	Free from Xilinx	CPL sub-

- **WebPACK includes a free version of ModelSim that works only for Xilinx parts but you must register at www.mentor.com**

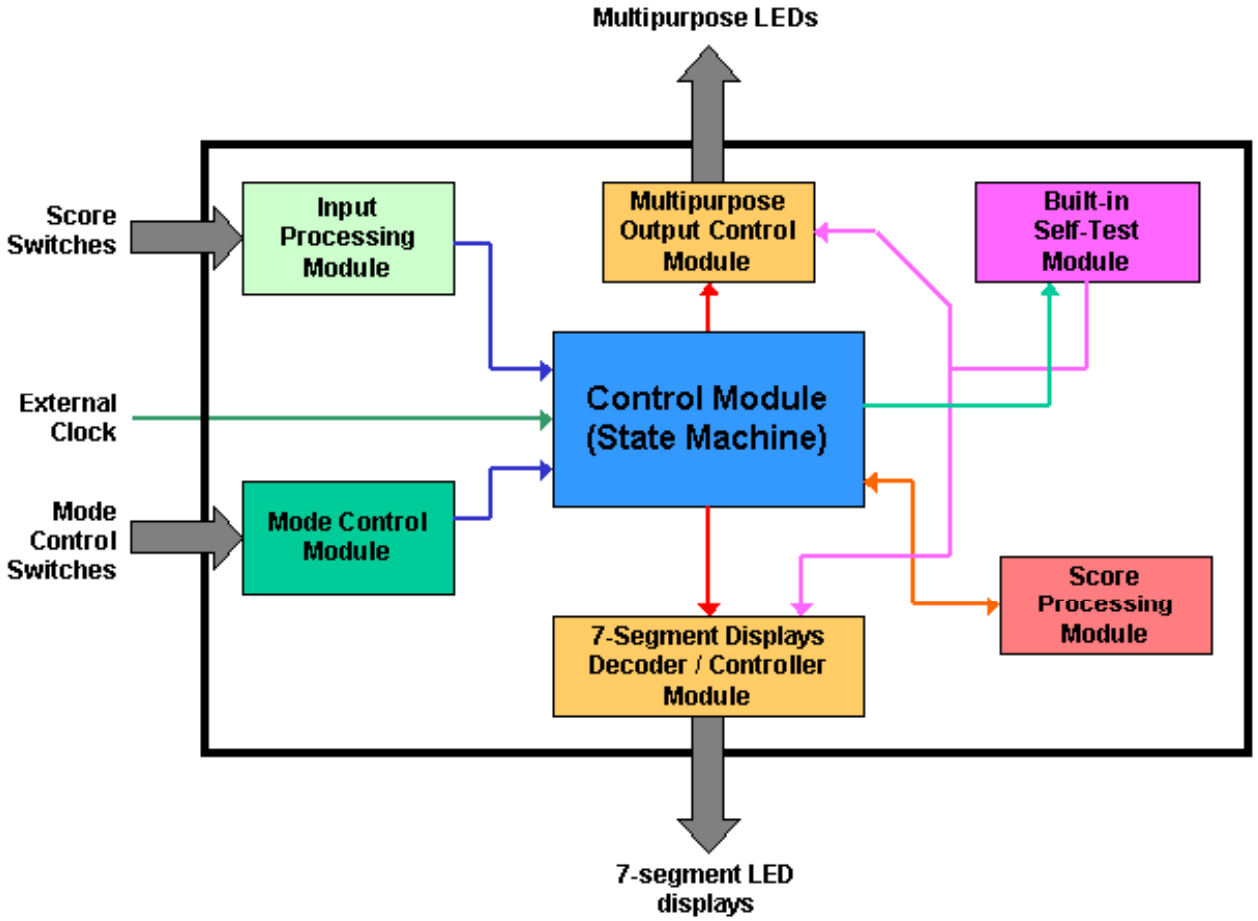
IMPLEMENTING A PROJECT

- Determine I/O Requirements
- Partition into 5-9 submodules and test individually before combining.
- Use/Enhance Built-In Self-Test
- Debounce Pushbutton Switches
- Filter an Input to Produce a Single Pulse
- Use hierarchy with Submodule Components
- Synchronize Externally Clocked Inputs

DETERMINE SYSTEM I/O REQUIREMENTS

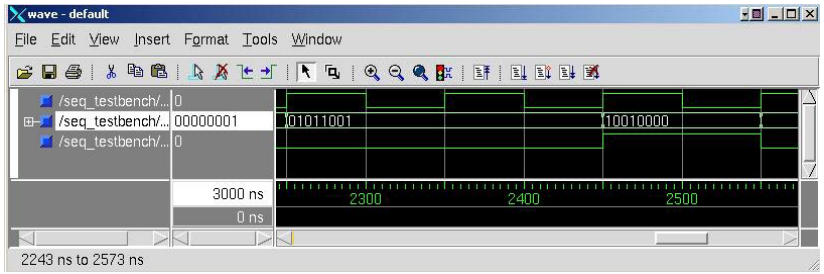


DECOMPOSE EACH LEVEL INTO 7 + / - 2 SUBMODULES

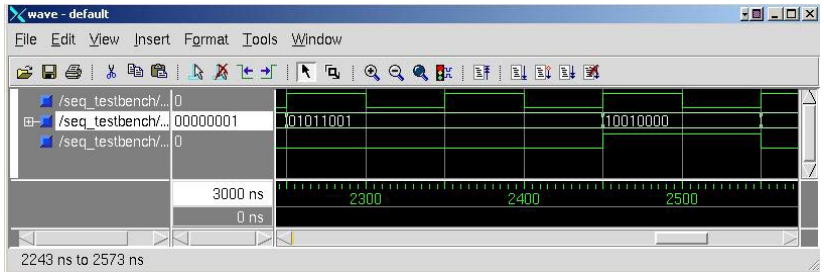


SIMULATE EACH SUBMODULE AND THEN INTEGRATE THEM

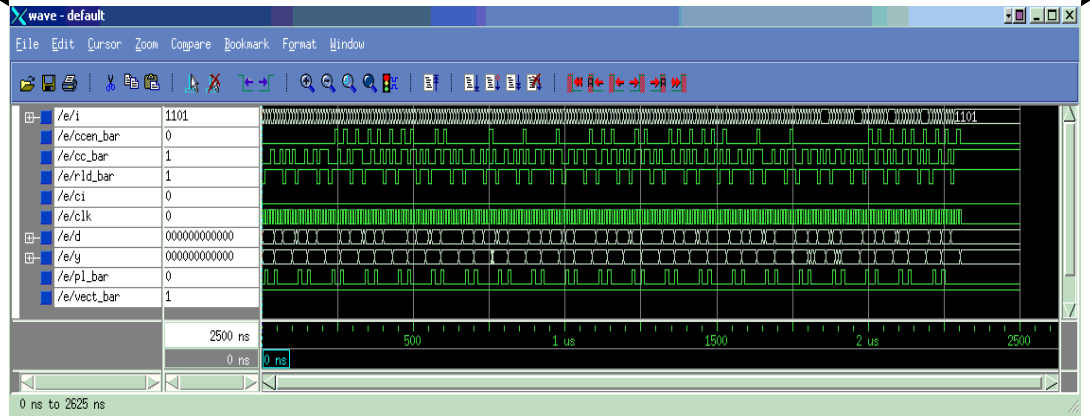
Submodule#1



Submodule#2

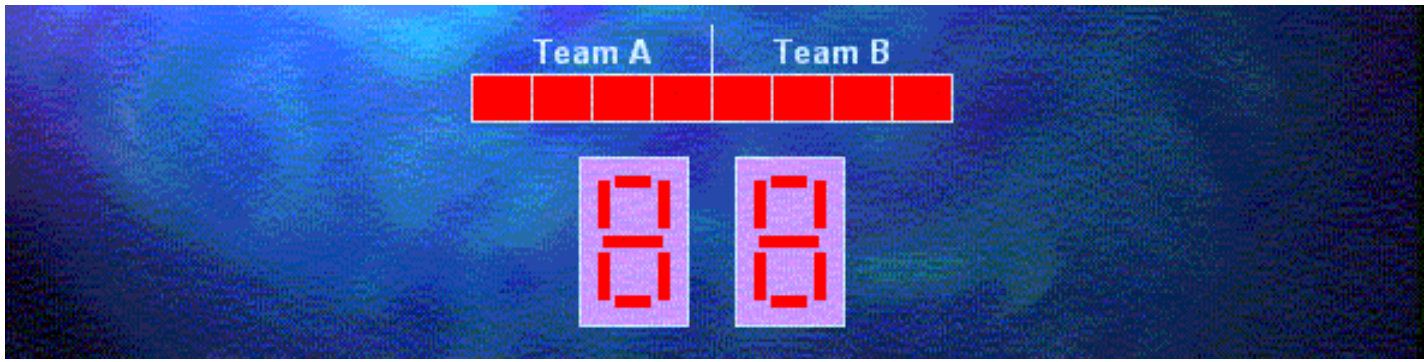


Composite of #1 and #2



BUILT-IN SELF-TEST

- Downloading hw3a (BIST) ensures the integrity of the connection between the CPU and the Spartan3 prototyping board AND then checks the input switches and the 7-segment displays.
- More thorough checks could be added to ensure the integrity of the board I/O for a specific project.



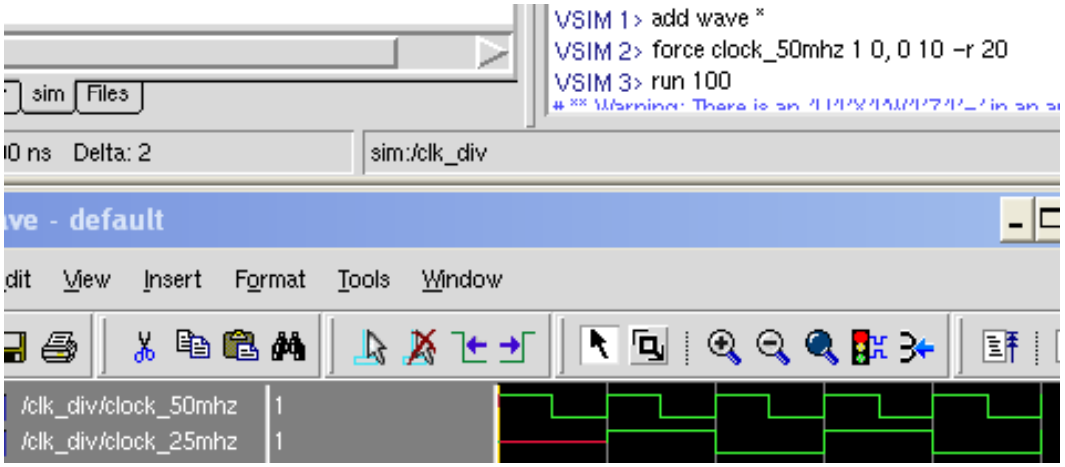
INTERNAL FREQUENCIES CAN BE DERIVED FROM THE EXTERNAL CLOCK

- A crystal oscillator on the Spartan3 prototyping board produces a 50 MHz clock.
- Counters can be used to divide down a frequency into a slower synchronized one:

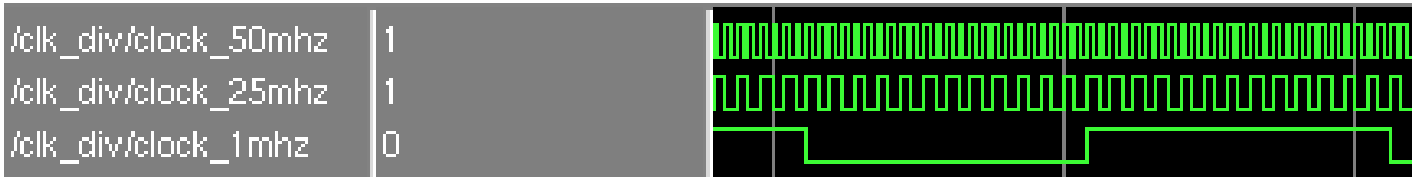
Divide By	Frequency	Duration
1	clock_50MHz	20 ns
2	clock_25 MHz	40 ns
25	clock_1MHz	1000 ns = 1us
10	clock_100KHz	10 us
10	clock_10KHz	100 us
10	clock_1KHz	1000 us = 1 ms
10	clock_100Hz	10 ms
10	clock_10Hz	100 ms
10	clock_1Hz	1000 ms = 1 s
10	clock_tenthHz	10 s

Simulation of clk_div.vhd (part 1)

- **50 MHz divided by 2 ➔ 25 MHz:**

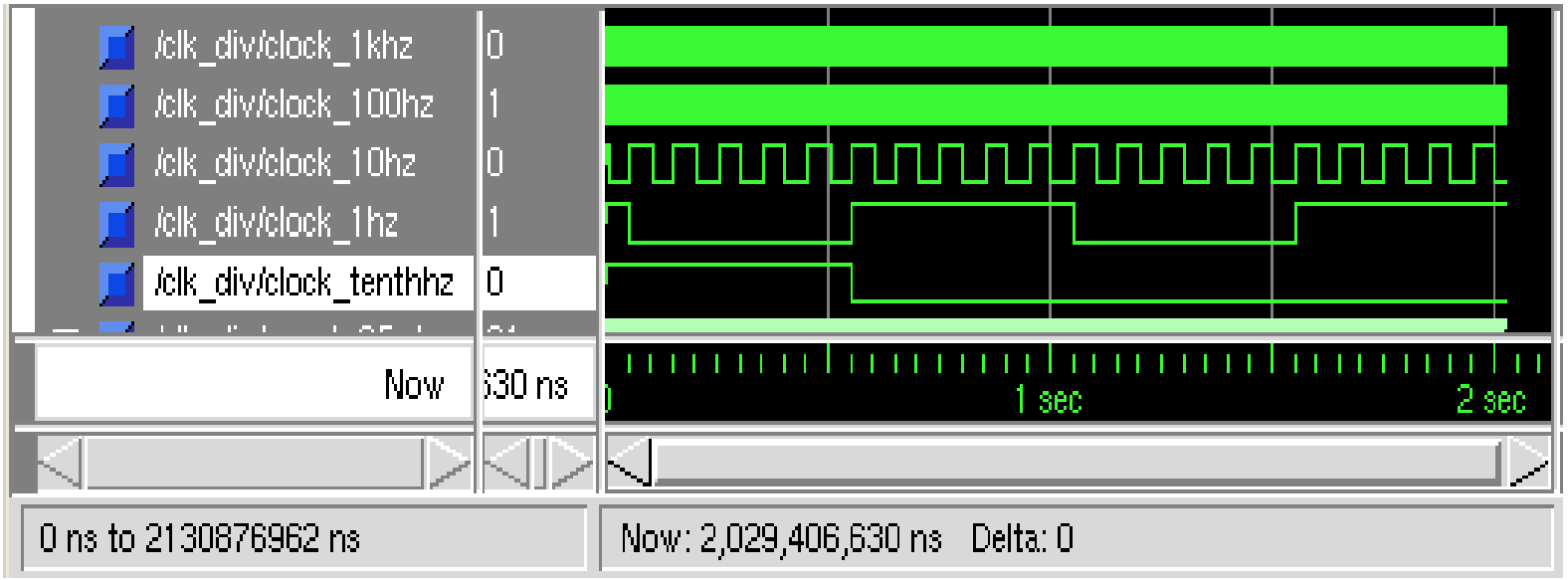


- **25 MHz divided by 25 ➔ 1 MHz:**



Simulation of clk_div.vhd (part 2)

- **10 KHz divided by 10 ➔ 1 KHz:**



clk_div.vhd (part 1)

ln # | sim:/clk_div : clk_div.vhd

```
 1 -- original from Jim Hamblen at Georgia Tech
 2 -- revised on 9/15/05 by Don Bouldin for 50Mhz and TenthHz
 3 LIBRARY IEEE;
 4 USE   IEEE.STD_LOGIC_1164.all;
 5 USE   IEEE.STD_LOGIC_ARITH.all;
 6 USE   IEEE.STD_LOGIC_UNSIGNED.all;
 7
 8 ENTITY clk_div IS
 9
10         PORT
11         (
12                 clock_50Mhz                     : IN    STD_LOGIC;
13                 clock_25Mhz                     : OUT   STD_LOGIC;
14                 clock_1Mhz                      : OUT   STD_LOGIC;
15                 clock_100Khz            : OUT   STD_LOGIC;
16                 clock_10Khz                     : OUT   STD_LOGIC;
17                 clock_1Khz                      : OUT   STD_LOGIC;
18                 clock_100Hz                     : OUT   STD_LOGIC;
19                 clock_10Hz                      : OUT   STD_LOGIC;
20                 clock_1Hz                       : OUT   STD_LOGIC;
21                 clock_tenthHz           : OUT   STD_LOGIC);
22
23 END clk_div;
24
25 ARCHITECTURE a OF clk_div IS
26
27         SIGNAL  count_25Mhz: STD_LOGIC_VECTOR(1 DOWNTO 0);
28         SIGNAL  count_1Mhz: STD_LOGIC_VECTOR(4 DOWNTO 0);
29         SIGNAL  count_100Khz, count_10Khz, count_1Khz: STD_LOGIC_VECTOR(3 DOWNTO 0);
30         SIGNAL  count_100hz, count_10hz, count_1hz, count_tenthhz : STD_LOGIC_VECTOR(3 DOWNTO 0);
31         SIGNAL  clock_25Mhz_int, clock_1Mhz_int, clock_100Khz_int, clock_10Khz_int, clock_1Khz_int: STD_LOGIC;
32         SIGNAL  clock_100hz_int, clock_10Hz_int, clock_1Hz_int, clock_tenthHz_int : STD_LOGIC;
33 BEGIN
```

clk_div.vhd (part 2)

ln # | sim:/clk_div : clk_div.vhd

```
33 BEGIN
34          PROCESS
35          BEGIN
36 -- Divide by 2
37                  WAIT UNTIL clock_50Mhz'EVENT and clock_50Mhz = '1';
38                          IF count_25Mhz < 1 THEN
39                                  count_25Mhz <= count_25Mhz + 1;
40                          ELSE
41                                  count_25Mhz <= "00";
42                          END IF;
43                          IF count_25Mhz < 1 THEN
44                                  clock_25Mhz_int <= '0';
45                          ELSE
46                                  clock_25Mhz_int <= '1';
47                          END IF;
48
49                  -- Ripple clocks are used in this code to save prescalar hardware
50                  -- Sync all clock prescalar outputs back to master clock signal
51                          clock_25Mhz <= clock_25Mhz_int;
52                          clock_1Mhz <= clock_1Mhz_int;
53                          clock_100Khz <= clock_100Khz_int;
54                          clock_10Khz <= clock_10Khz_int;
55                          clock_1Khz <= clock_1Khz_int;
56                          clock_100hz <= clock_100hz_int;
57                          clock_10hz <= clock_10hz_int;
58                          clock_1hz <= clock_1hz_int;
59                          clock_tenthhz <= clock_tenthhz_int;
60          END PROCESS;
```

clk_div.vhd (part 3)

ln # | sim:/clk_div : clk_div.vhd

```
61
62          PROCESS
63          BEGIN
64  -- Divide by 25
65                      WAIT UNTIL clock_25Mhz_int'EVENT and clock_25Mhz_int = '1';
66                              IF count_1Mhz < 24 THEN
67                                      count_1Mhz <= count_1Mhz + 1;
68                              ELSE
69                                      count_1Mhz <= "00000";
70                              END IF;
71                              IF count_1Mhz < 12 THEN
72                                      clock_1Mhz_int <= '0';
73                              ELSE
74                                      clock_1Mhz_int <= '1';
75                              END IF;
76          END PROCESS;
77
78          PROCESS
79          BEGIN
80  -- Divide by 10
81                      WAIT UNTIL clock_1Mhz_int'EVENT and clock_1Mhz_int = '1';
82                              IF count_100khz < 9 THEN
83                                      count_100khz <= count_100khz + 1;
84                              ELSE
85                                      count_100khz <= "0000";
86                              END IF;
87                              IF count_100khz < 5 THEN
88                                      clock_100khz_int <= '0';
89                              ELSE
90                                      clock_100khz_int <= '1';
91                              END IF;
92
93          END PROCESS;
```

clk_div.vhd (part 4)

ln # | sim:/clk_div : clk_div.vhd

```
 94
 95
 96          PROCESS
 97          BEGIN
 98 -- Divide by 10
 99                    WAIT UNTIL clock_100Khz_int'EVENT and clock_100Khz_int = '1';
100                           IF count_10khz < 9 THEN
101                                  count_10khz <= count_10khz + 1;
102                           ELSE
103                                  count_10khz <= "0000";
104                           END IF;
105                           IF count_10khz < 5 THEN
106                                  clock_10khz_int <= '0';
107                           ELSE
108                                  clock_10khz_int <= '1';
109                           END IF;
110
111          END PROCESS;                                                          ;
112
113
114          PROCESS
115          BEGIN
116 -- Divide by 10
117                    WAIT UNTIL clock_10Khz_int'EVENT and clock_10Khz_int = '1';
118                           IF count_1khz < 9 THEN
119                                  count_1khz <= count_1khz + 1;
120                           ELSE
121                                  count_1khz <= "0000";
122                           END IF;
123                           IF count_1khz < 5 THEN
124                                  clock_1khz_int <= '0';
125                           ELSE
126                                  clock_1khz_int <= '1';
127                           END IF;
128
129          END PROCESS;
130
```

clk_div.vhd (part 5)

ln # | sim:/clk_div : clk_div.vhd

```
130
131
132         PROCESS
133         BEGIN
134 -- Divide by 10
135                     WAIT UNTIL clock_1Khz_int'EVENT and clock_1Khz_int = '1';
136                             IF count_100hz < 9 THEN
137                                     count_100hz <= count_100hz + 1;
138                             ELSE
139                                     count_100hz <= "0000";
140                             END IF;
141                             IF count_100hz < 5 THEN
142                                     clock_100hz_int <= '0';
143                             ELSE
144                                     clock_100hz_int <= '1';
145                             END IF;
146
147         END PROCESS;
148
149
150         PROCESS
151         BEGIN
152 -- Divide by 10
153                     WAIT UNTIL clock_100hz_int'EVENT and clock_100hz_int = '1';
154                             IF count_10hz < 9 THEN
155                                     count_10hz <= count_10hz + 1;
156                             ELSE
157                                     count_10hz <= "0000";
158                             END IF;
159                             IF count_10hz < 5 THEN
160                                     clock_10hz_int <= '0';
161                             ELSE
162                                     clock_10hz_int <= '1';
163                             END IF;
164
165         END PROCESS;
166
```

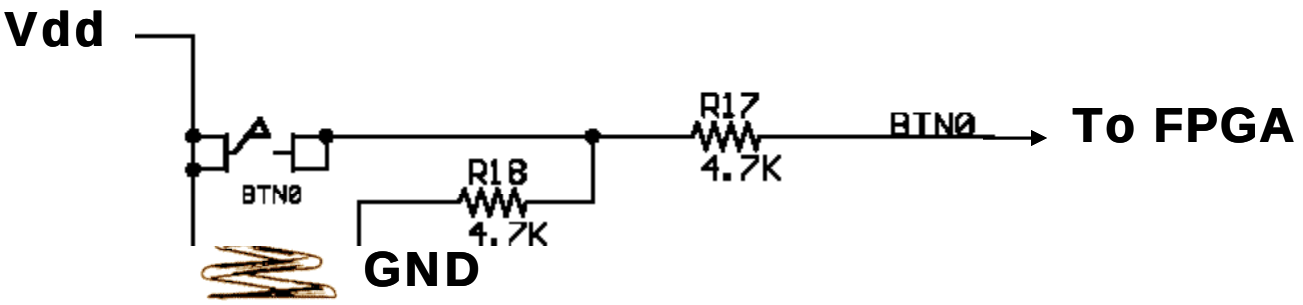
clk_div.vhd (part 6)

ln # | sim:/clk_div : clk_div.vhd

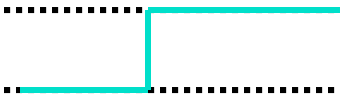
```
167
168         PROCESS
169         BEGIN
170 -- Divide by 10
171                 WAIT UNTIL clock_10hz_int'EVENT and clock_10hz_int = '1';
172                         IF count_1hz < 9 THEN
173                                 count_1hz <= count_1hz + 1;
174                         ELSE
175                                 count_1hz <= "0000";
176                         END IF;
177                         IF count_1hz < 5 THEN
178                                 clock_1hz_int <= '0';
179                         ELSE
180                                 clock_1hz_int <= '1';
181                         END IF;
182
183         END PROCESS;
184
185
186         PROCESS
187         BEGIN
188 -- Divide by 10
189                 WAIT UNTIL clock_1hz_int'EVENT and clock_1hz_int = '1';
190                         IF count_tenthhz < 9 THEN
191                                 count_tenthhz <= count_tenthhz + 1;
192                         ELSE
193                                 count_tenthhz <= "0000";
194                         END IF;
195                         IF count_tenthhz < 5 THEN
196                                 clock_tenthhz_int <= '0';
197                         ELSE
198                                 clock_tenthhz_int <= '1';
199                         END IF;
200
201         END PROCESS;
202
203
204 END a;
```

clk_div.vhd

AN INPUT PUSHBUTTON SWITCH MAY BOUNCE



Input With Bouncing



Desired Output

Fig. 6.11, Page 243
ASICs by M. Smith

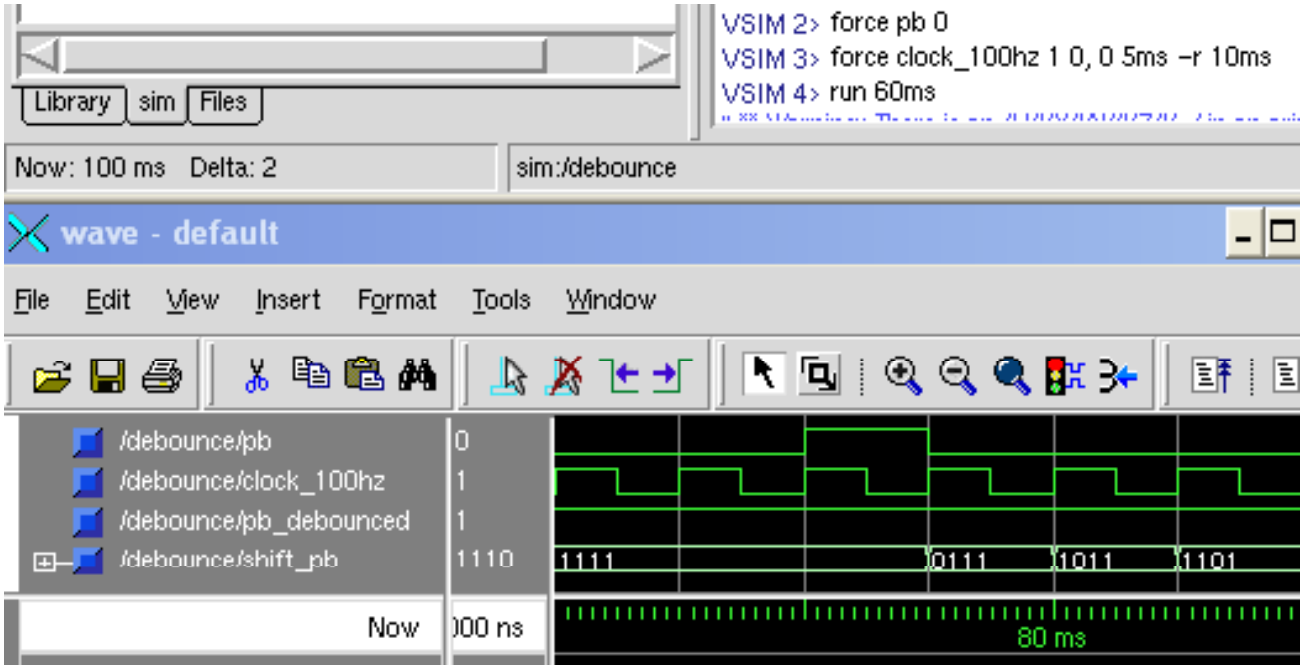
debounce.vhd

ln # | sim:/debounce : debounce.vhd

```
 1 LIBRARY IEEE;
 2 USE   IEEE.STD_LOGIC_1164.all;
 3 USE   IEEE.STD_LOGIC_ARITH.all;
 4 USE   IEEE.STD_LOGIC_UNSIGNED.all;
 5
 6         -- Debounce Pushbutton: Filters out mechanical switch bounce for around 40Ms.
 7 ENTITY debounce IS
 8         PORT(pb, clock_100Hz  : IN STD_LOGIC;
 9                  pb_debounced   : OUT   STD_LOGIC);
10 END debounce;
11
12 ARCHITECTURE a OF debounce IS
13         SIGNAL SHIFT_PB    : STD_LOGIC_VECTOR(3 DOWNTO 0);
14
15
16 BEGIN
17
18         -- Debounce clock should be approximately 10ms or 100Hz
19         PROCESS
20         BEGIN
21                  WAIT UNTIL (clock_100Hz'EVENT) AND (clock_100Hz = '1');
22                  -- Use a shift register to filter switch contact bounce
23                  SHIFT_PB(2 DOWNTO 0) <= SHIFT_PB(3 DOWNTO 1);
24                  SHIFT_PB(3) <= NOT PB;
25                  IF SHIFT_PB(3 DOWNTO 0)="0000" THEN
26                           PB_DEBOUNCED <= '0';
27                  ELSE
28                           PB_DEBOUNCED <= '1';
29                  END IF;
30         END PROCESS;
31 END a;
```

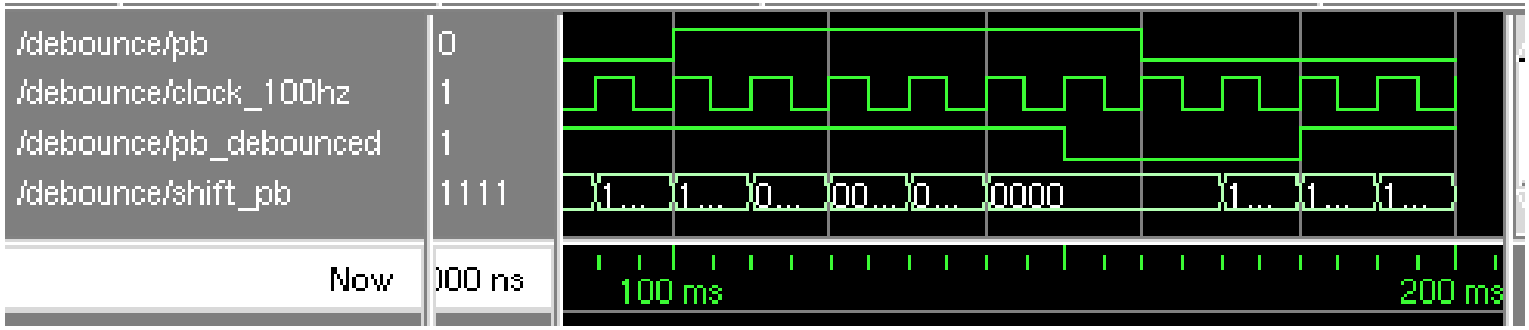
Simulation of debounce.vhd (part 1)

- **PB has been 0 (inactive), then is 1 (active) but only for 10 ms (too short to be valid) so PB_debounced stays 1 (inactive):**



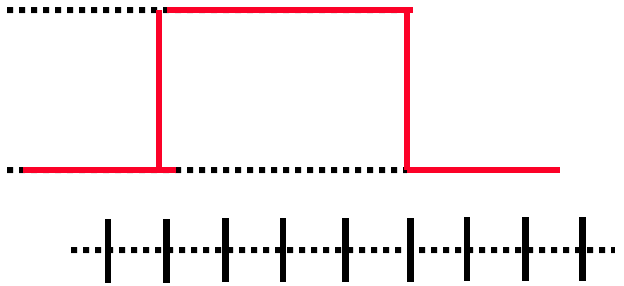
Simulation of debounce.vhd (part 2)

- **This time PB is 1 (active) for 60 ms (which is valid since it is at least 40ms) so PB_debounced becomes 0 (active):**

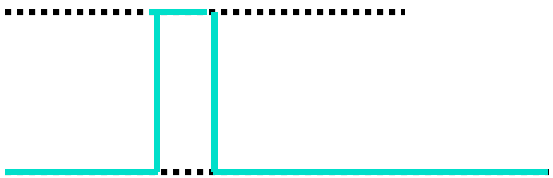


FILTER AN INPUT TO PRODUCE A SINGLE PULSE

An External Pushbutton Switch May Be Pressed For Multiple Clock Ticks

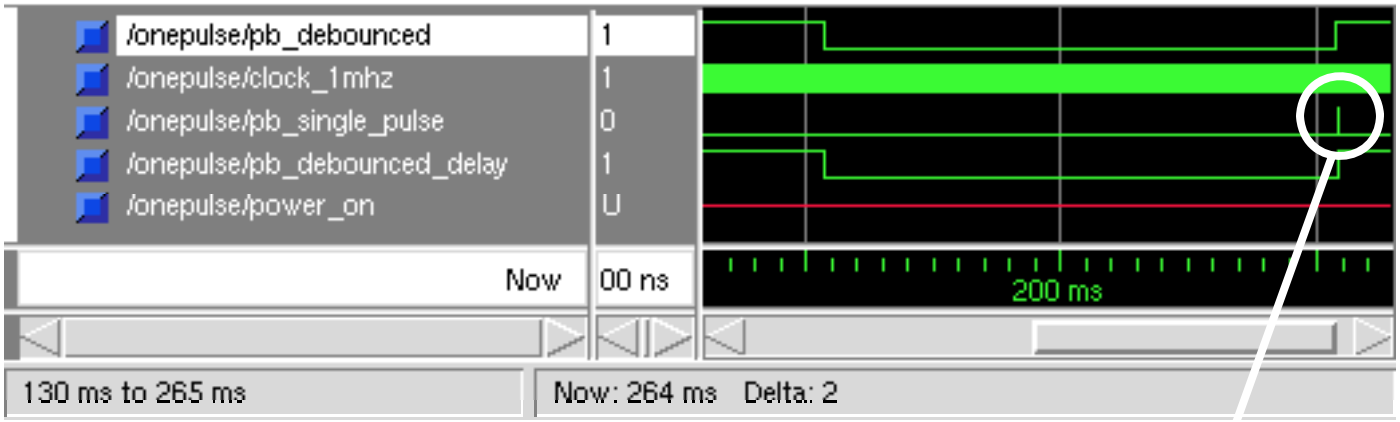


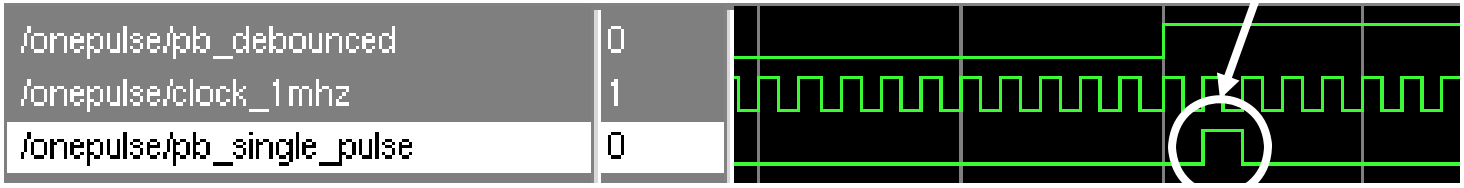
A Circuit Can Be Implemented to Produce Only a Single Pulse



Simulation of onepulse.vhd

- **PB_debounced has been 0 (active) for 100 ms but only a single 1 ms pulse is produced:**





onepulse.vhd (part 1)

ln # sim:/onepulse : onepulse.vhd

```
 1 LIBRARY IEEE;
 2 USE IEEE.STD_LOGIC_1164.all;
 3 USE IEEE.STD_LOGIC_ARITH.all;
 4 USE IEEE.STD_LOGIC_UNSIGNED.all;
 5
 6 -- Single Pulse circuit
 7 -- the output will go high for only one clock cycle
 8
 9 ENTITY onepulse IS
10
11          PORT(PB_debounced, clock_1MHz : IN STD_LOGIC;
12                    PB_single_pulse   : OUT STD_LOGIC);
13
14 END onepulse;
15
16
```

onepulse.vhd (part 2)

```
16
17 ARCHITECTURE a OF onepulse IS
18         SIGNAL PB_debounced_delay, Power_on : STD_LOGIC;
19
20 BEGIN
21  PROCESS
22  BEGIN
23     WAIT UNTIL (CLOCK_1MHz'event) AND (CLOCK_1MHz='1');
24                          -- Power_on will be initialized to '0' at power up
25     IF Power_on='0' THEN
26                          -- This code resets the critical signals once at power up
27           PB_single_pulse    <= '0';
28           PB_debounced_delay  <= '1';
29           Power_on                          <= '1';
30
31     ELSE
32                          -- A single clock cycle pulse is produced when the switch is hit
33                          -- No matter how long the switch is held down
34                          -- The switch input must already be debounced
35                  IF PB_debounced = '1' AND PB_debounced_delay = '0' THEN
36                          PB_single_pulse <= '1';
37                  ELSE
38                          PB_single_pulse <= '0';
39                  END IF;
40
41                  PB_debounced_delay <= PB_debounced;
42
43     END IF;
44
45  END PROCESS;
46
47 END a;
```

onepulse.vhd (part 1) (modified)

Original

ln # | sim:/onepulse : onepulse.vhd

```
 1 LIBRARY IEEE;
 2 USE IEEE.STD_LOGIC_1164.all;
 3 USE IEEE.STD_LOGIC_ARITH.all;
 4 USE IEEE.STD_LOGIC_UNSIGNED.all;
 5
 6 -- Single Pulse circuit
 7 -- the output will go high for only one clock cycle
 8
 9 ENTITY onepulse IS
10
11         PORT(PB_debounced, clock_1MHz : IN STD_LOGIC;
12                   PB_single_pulse   : OUT STD_LOGIC);
13
14 END onepulse;
15
16
```

Modified:

ln # | sim:/onepulse : onepulse.vhd

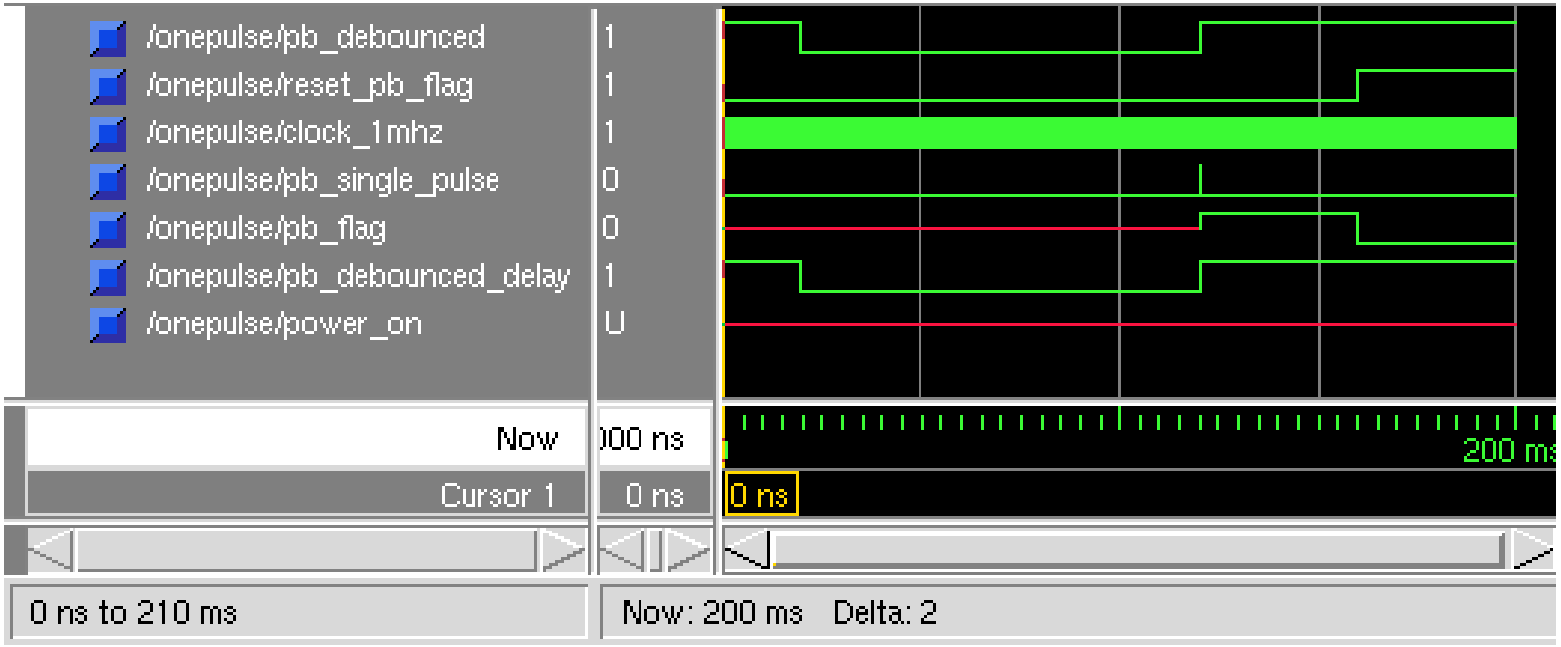
```
 1 LIBRARY IEEE;
 2 USE IEEE.STD_LOGIC_1164.all;
 3 USE IEEE.STD_LOGIC_ARITH.all;
 4 USE IEEE.STD_LOGIC_UNSIGNED.all;
 5
 6 -- Single Pulse circuit
 7 -- the output will go high for only one clock cycle
 8
 9 ENTITY onepulse IS
10
11         PORT(PB_debounced, reset_pb_flag, clock_1MHz : IN STD_LOGIC;
12                   PB_single_pulse, pb_flag : OUT STD_LOGIC);
13
14 END onepulse;
15
16
```

onepulse.vhd (part 2) (modified)

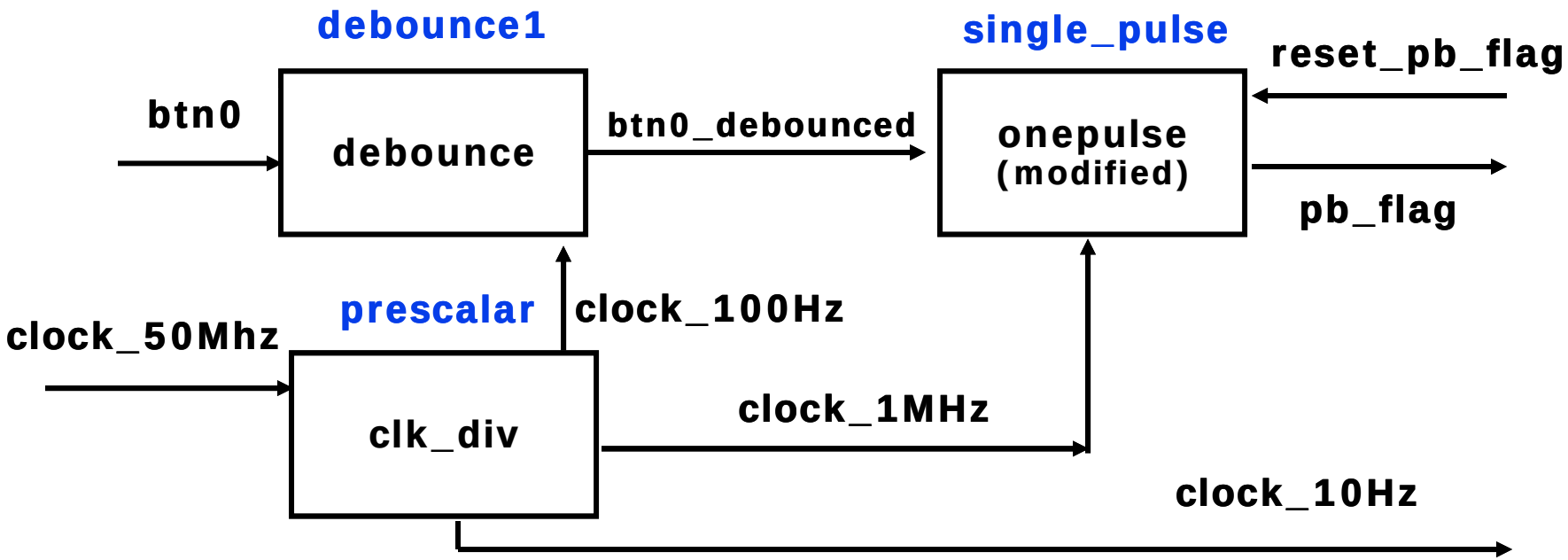
```
17 ARCHITECTURE a OF onepulse IS
18         SIGNAL PB_debounced_delay, Power_on : STD_LOGIC;
19
20 BEGIN
21  PROCESS
22  BEGIN
23     WAIT UNTIL (CLOCK_1MHz'event) AND (CLOCK_1MHz='1');
24                         -- Power_on will be initialized to '0' at power up
25     IF Power_on='0' THEN
26                         -- This code resets the critical signals once at power up
27           PB_single_pulse    <= '0';
28           PB_flag                 <= '0';
29           PB_debounced_delay  <= '1';
30           Power_on                        <= '1';
31
32     ELSE
33                         -- A single clock cycle pulse is produced when the switch is hit
34                         -- No matter how long the switch is held down
35                         -- The switch input must already be debounced
36                 IF PB_debounced = '1' AND PB_debounced_delay = '0' THEN
37                         PB_single_pulse <= '1';
38                         pb_flag <= '1';
39                 ELSE
40                         PB_single_pulse <= '0';
41                         IF reset_pb_flag = '1' THEN pb_flag <= '0';
42                         ELSE
43                         END IF;
44                 END IF;
45
46                 PB_debounced_delay <= PB_debounced;
47
48     END IF;
49
50  END PROCESS;
51
52 END a;
```

Simulation of onepulse.vhd (modified)

- **PB_debounced has been 0 (active) for 100 ms but only a single 1 ms pulse is produced and pb_flag is set. After being read, pb_flag is reset.**

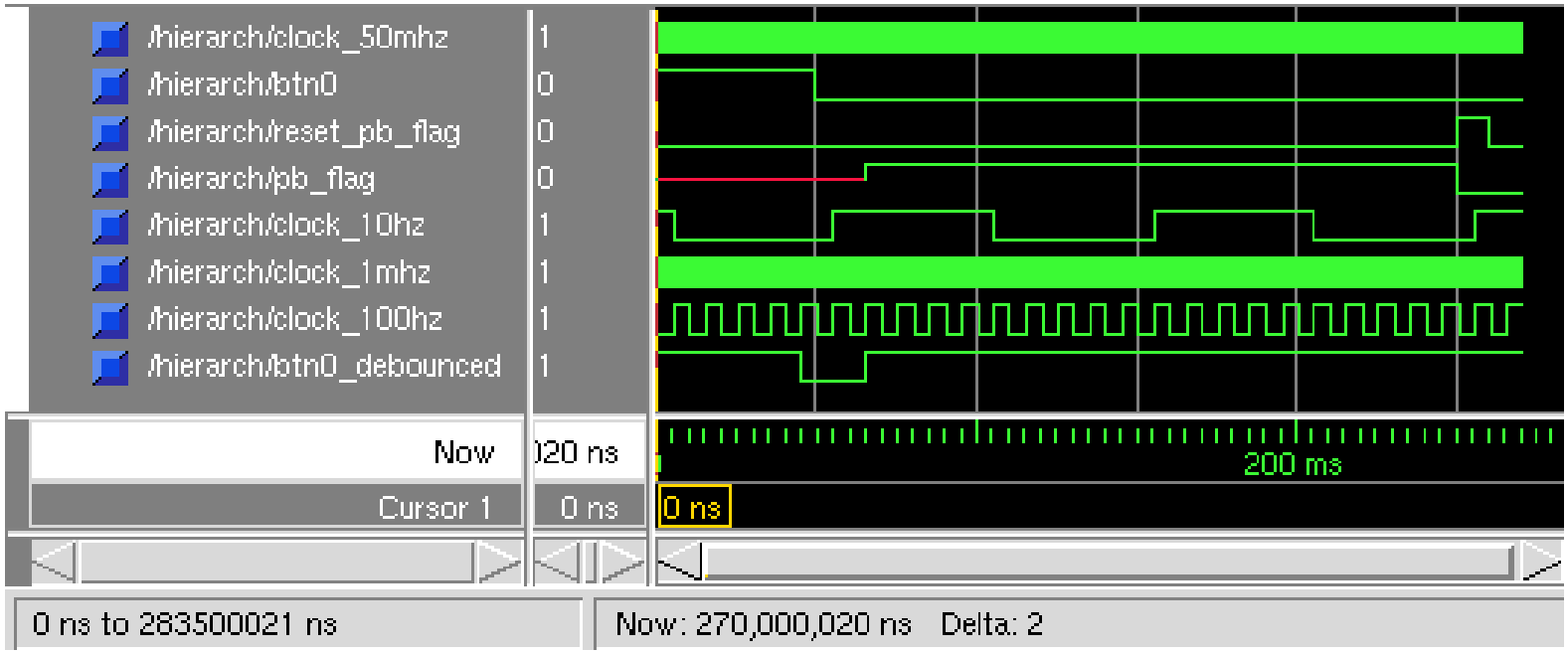


HIERARCHY WITH SUBMODULE COMPONENTS



Simulation of hierarch.vhd

- **A proper pressing of btn0 sets pb_flag to HIGH.**
- **Once pb_flag is read, it is reset to LOW.**



hierarch.vhd (part 1)

ln # | sim:/hierarch : hierarch.vhd

```
 1
 2 LIBRARY IEEE;
 3 USE   IEEE.STD_LOGIC_1164.all;
 4 USE   IEEE.STD_LOGIC_ARITH.all;
 5 USE   IEEE.STD_LOGIC_UNSIGNED.all;
 6
 7 ENTITY hierarch IS
 8         PORT (clock_50Mhz, btn0, reset_pb_flag : IN STD_LOGIC;
 9                   pb_flag, clock_10Hz : OUT STD_LOGIC);
10 END hierarch;
11 ARCHITECTURE a OF hierarch IS
12 -- Declare internal signals needed to connect submodules
13 SIGNAL clock_1MHz, clock_100Hz, btn0_debounced : STD_LOGIC;
14 -- Use Components to Define Submodules and Parameters
15         COMPONENT debounce
16                 PORT(pb, clock_100Hz : IN STD_LOGIC;
17                  pb_debounced : OUT STD_LOGIC);
18         END COMPONENT;
19
20         COMPONENT onepulse
21          PORT(PB_debounced, reset_pb_flag, clock_1MHz    : IN    STD_LOGIC;
22                           PB_single_pulse, pb_flag   : OUT   STD_LOGIC);
23         END COMPONENT;
24
25         COMPONENT clk_div
26                 PORT(clock_50Mhz                 : IN    STD_LOGIC;
27                 clock_25MHz                      : OUT   STD_LOGIC;
28                 clock_1MHz                       : OUT   STD_LOGIC;
29                 clock_100KHz            : OUT   STD_LOGIC;
30                 clock_10KHz                      : OUT   STD_LOGIC;
31                 clock_1KHz                       : OUT   STD_LOGIC;
32                 clock_100Hz                      : OUT   STD_LOGIC;
33                 clock_10Hz                       : OUT   STD_LOGIC;
34                 clock_1Hz                        : OUT   STD_LOGIC);
35         END COMPONENT;
```

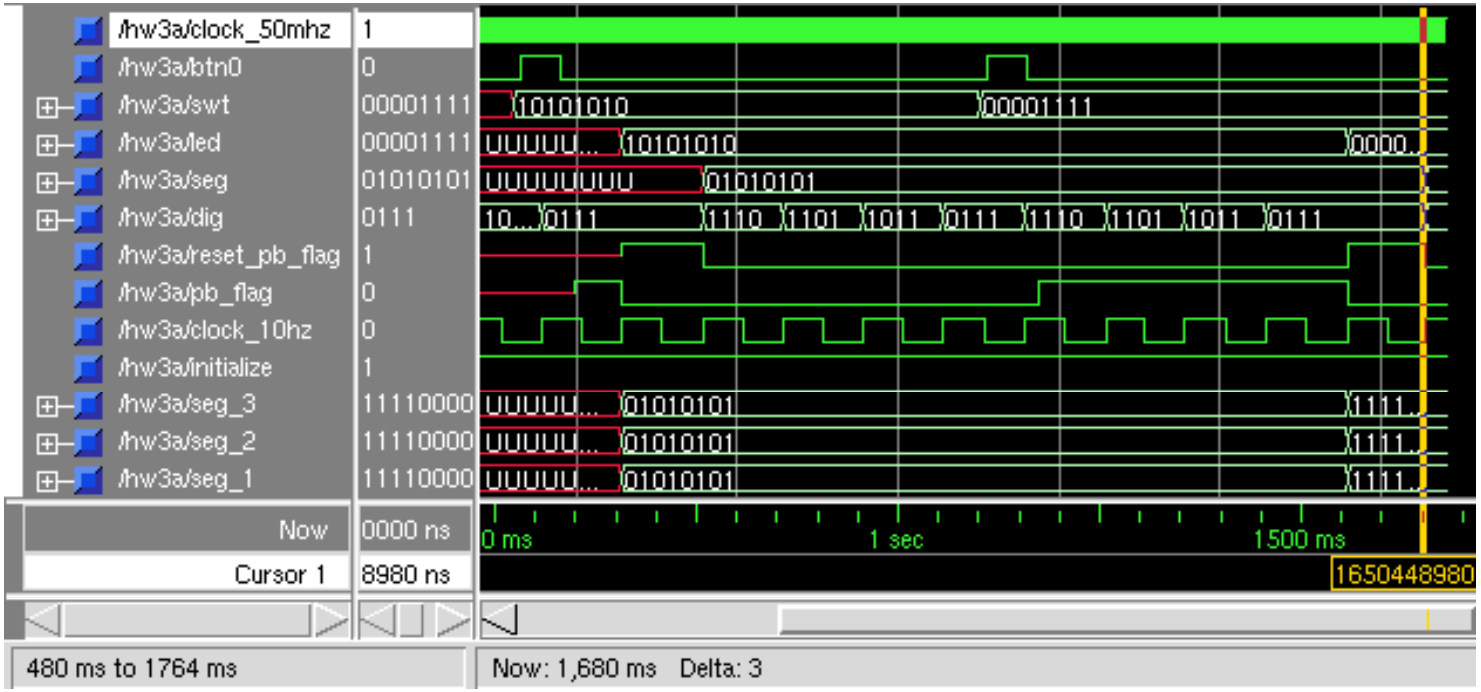
hierarch.vhd (part 2)

```
36 BEGIN
37
38 -- Use Port Map to connect signals between components in the hisarchy
39 debounce1 : debounce PORT MAP  (pb => btn0, clock_100Hz => clock_100Hz,
40                          pb_debounced => btn0_debounced);
41
42 prescalar : clk_div  PORT MAP (clock_50Mhz => clock_50Mhz, clock_1MHz => clock_1Mhz,
43                          clock_100hz => clock_100hz, clock_10hz => clock_10hz);
44
45 single_pulse : onepulse PORT MAP (pb_debounced => btn0_debounced,
46                             reset_pb_flag => reset_pb_flag,
47                             pb_flag => pb_flag,
48                             clock_1MHz => clock_1MHz);
49
50 END a;
51
```

hierarch.vhd

Simulation of hw3a.vhd

- **After btn0 is pressed (properly), the pb_flag is set. Then while dig “0111” is displayed , swt<7:0> are read and pb_flag is reset:**



hw3a.vhd (part 1)

ln # | sim:/hw3a : hw3a.vhd

```
 1 ----------------------------------------------------------------------
 2 -- hw3a.vhd -- Demonstrate basic functions
 3 ----------------------------------------------------------------------
 4 -- Author: Don Bouldin, Univ. of Tennessee, 9/19/05
 5 ----------------------------------------------------------------------
 6 -- This module tests basic functions and I/O on the Spartan3 board.
 7 ----------------------------------------------------------------------
 8
 9 library IEEE;
10 use IEEE.STD_LOGIC_1164.ALL;
11 use IEEE.STD_LOGIC_ARITH.ALL;
12 use IEEE.STD_LOGIC_UNSIGNED.ALL;
13
14 entity hw3a is
15 Port (
16 clock_50MHz : in std_logic;
17
18 btn0: in std_logic;
19
20
21 swt : in std_logic_vector(7 downto 0);
22 -- swt(7) = most  significant sliding switch; OFF = LOW; ON = HIGH
23 -- swt(0) = least significant sliding switch; OFF = LOW; ON = HIGH
24
25
26 led : out std_logic_vector(7 downto 0);
27 -- led(7) = most significant  led is ON when active HIGH
28 -- led(0) = least significant led is ON when active HIGH
29
30
31 seg : out std_logic_vector(7 downto 0);
32 -- segment lights when active LOW
33 -- seg(0) = seg-a
34 -- seg(1) = seg-b
35 -- seg(2) = seg-c
36 -- seg(3) = seg-d
37 -- seg(4) = seg-e
38 -- seg(5) = seg-f
39 -- seg(6) = seg-g
40 -- seg(7) = dp
41
```

hw3a.vhd (part 2)

ln # | sim:/hw3a : hw3a.vhd

```
41
42
43 dig : out std_logic_vector(3 downto 0)
44 -- dig(3) = most  significant digit is displayed when active LOW
45 -- dig(0) = least significant digit is displayed when active LOW
46
47 );
48 end hw3a;
49
50 architecture Behavioral of hw3a is
51
52 SIGNAL reset_pb_flag, pb_flag, clock_10hz: std_logic;
53
54 SIGNAL initialize : STD_LOGIC;
55
56 SIGNAL seg_3 : std_logic_vector(7 downto 0);
57 SIGNAL seg_2 : std_logic_vector(7 downto 0);
58 SIGNAL seg_1 : std_logic_vector(7 downto 0);
59 SIGNAL seg_0 : std_logic_vector(7 downto 0);
60
61 COMPONENT hierarch
62         PORT(
63       clock_50Mhz, btn0, reset_pb_flag: IN STD_LOGIC;
64       pb_flag, clock_10Hz : OUT STD_LOGIC
65                 );
66 END COMPONENT;
67
68 -- Use Port Map to connect signals between components in the hiearchy
69
70 BEGIN
71
72 hw3a : hierarch PORT MAP (clock_50Mhz => clock_50Mhz,
73                           btn0 =>  btn0,
74                   reset_pb_flag => reset_pb_flag,
75                                 pb_flag => pb_flag,
76                                 clock_10hz => clock_10hz
77                                           );
78
```

hw3a.vhd (part 3)

ln # | sim:/hw3a : hw3a.vhd

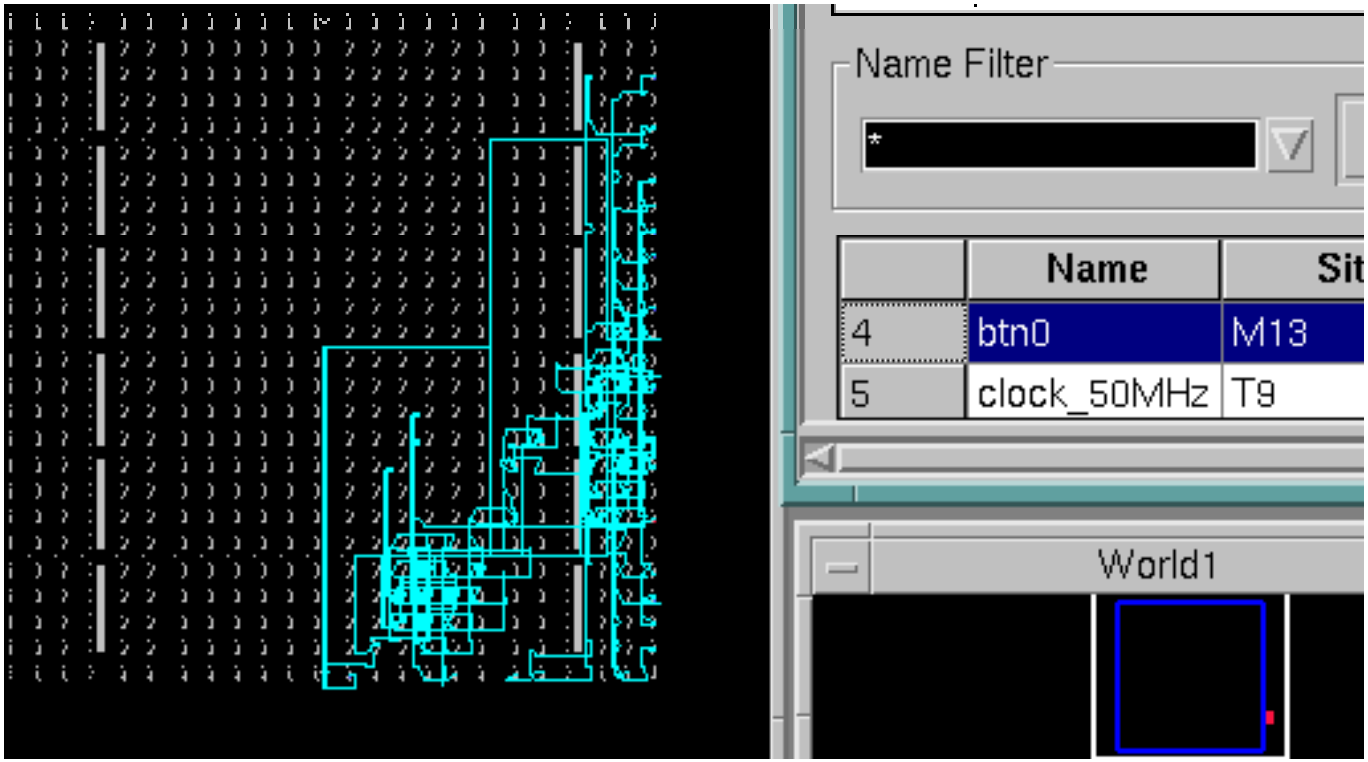
```
 79 -- begin loop
 80 PROCESS
 81 BEGIN
 82
 83                          -- initialize will be initialized to '0' at power up
 84     IF initialize = '0' THEN
 85                          -- This code resets the critical signals once at power
 86
 87 reset_pb_flag <= '0';
 88
 89 led(7 downto 0) <= "00000000";
 90
 91 seg_3(7 downto 0) <= "11111111";
 92 seg_2(7 downto 0) <= "11111111";
 93 seg_1(7 downto 0) <= "11111111";
 94 seg_0(7 downto 0) <= "11111111";
 95
 96 ELSE
 97         initialize <= '1';
 98
 99 -- display each of the four digits for 0.1 second each forever
100
101
102 seg(7 downto 0) <= seg_0(7 downto 0) ;
103 dig <= "1110"   ;                       --digit(0) is ON
104 --now wait for 0.1 second
105 WAIT UNTIL clock_10hz'EVENT and clock_10hz = '1';
106
107 seg(7 downto 0) <= seg_1(7 downto 0) ;
108 dig(3 downto 0) <= "1101"   ;                       --digit(1) is ON
109 --now wait for 0.1 second
110                 WAIT UNTIL clock_10hz'EVENT and clock_10hz = '1';
111
112
113 seg(7 downto 0) <= seg_2(7 downto 0);
114 dig(3 downto 0) <= "1011"   ;                       --digit(2) is ON
115 --now wait for 0.1 second
116                 WAIT UNTIL clock_10hz'EVENT and clock_10hz = '1';
117
118 seg(7 downto 0) <= seg_3(7 downto 0);
119 dig(3 downto 0) <= "0111"    ;                      --digit(3) is ON
120 --now wait for 0.1 second
121                 WAIT UNTIL clock_10hz'EVENT and clock_10hz = '1';
122
123
```

hw3a.vhd (part 4)

```
123
124 -- repeat display until USER sets sliding switches and then presses btn0
125
126 -- if btn0_pressed has NOT occurred then skip to repeat this loop,
127 -- else copy the switch settings and update the display
128
129   IF pb_flag = '1' THEN
130
131 -- copy the sliding switch settings to the leds
132 led(7 downto 0) <= swt(7 downto 0);
133
134 -- copy the sliding switch settings to the internal digit segments
135 seg_3(7 downto 0) <= not (swt(7 downto 0)) ;
136 seg_2(7 downto 0) <= not (swt(7 downto 0)) ;
137 seg_1(7 downto 0) <= not (swt(7 downto 0)) ;
138 seg_0(7 downto 0) <= not (swt(7 downto 0)) ;
139
140 reset_pb_flag <= '1';
141 --now wait for 0.1 second
142                  WAIT UNTIL clock_10hz'EVENT and clock_10hz = '1';
143 reset_pb_flag <= '0';
144 ELSE
145 END IF;
146
147 END IF;
148
149 -- repeat loop
150 END PROCESS;
151
152 END Behavioral;
153
```

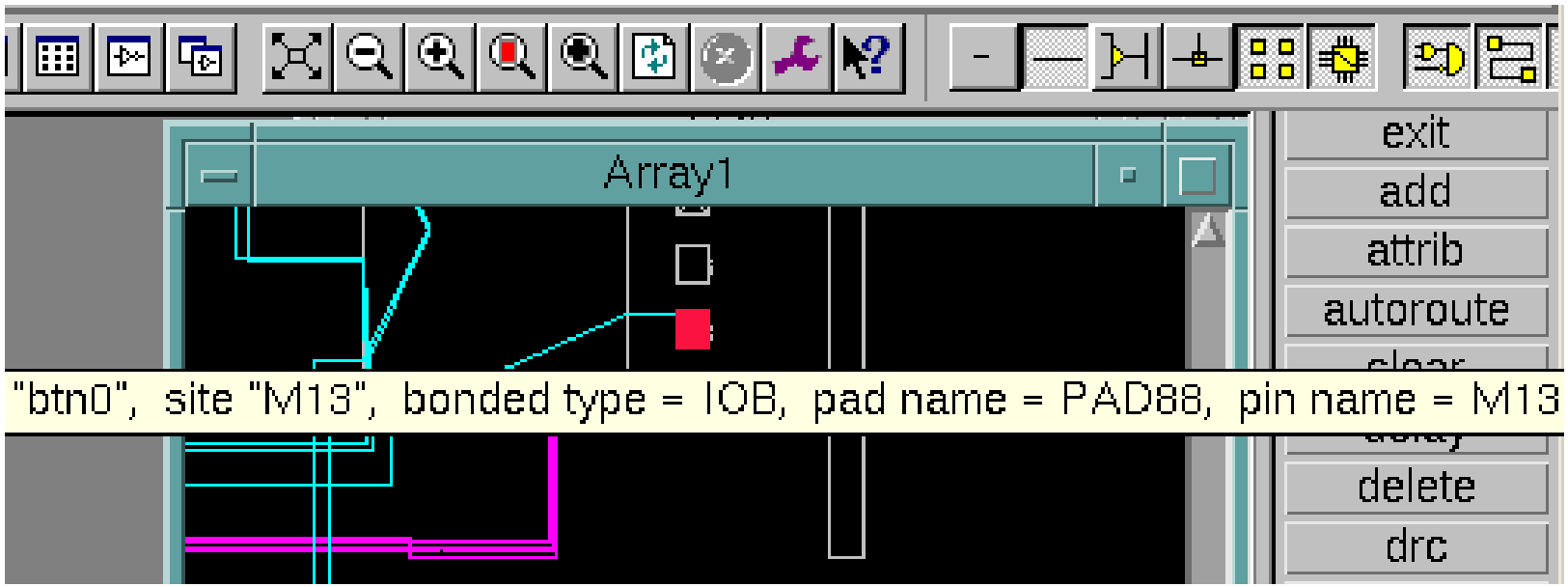
FPGA Resources for hw3a

- **Logic slices used: 58 out of 1920 = 3%**
- **The constraint file, hw3a.ucf, assigned the pins.**



Detailed Layout of hw3a

- **Pin M13 is connected to btn0:**



SYNCHRONIZER REDUCES RISK OF METASTABILITY PROBLEMS

- Signals from external circuits whose clock is independent must be synchronized with our internal clock.
- If not, the external input may occur during the decision window of our flip-flop and cause it to go into a metastable state.
- To reduce the likelihood of this occurring, the input can be double-buffered.

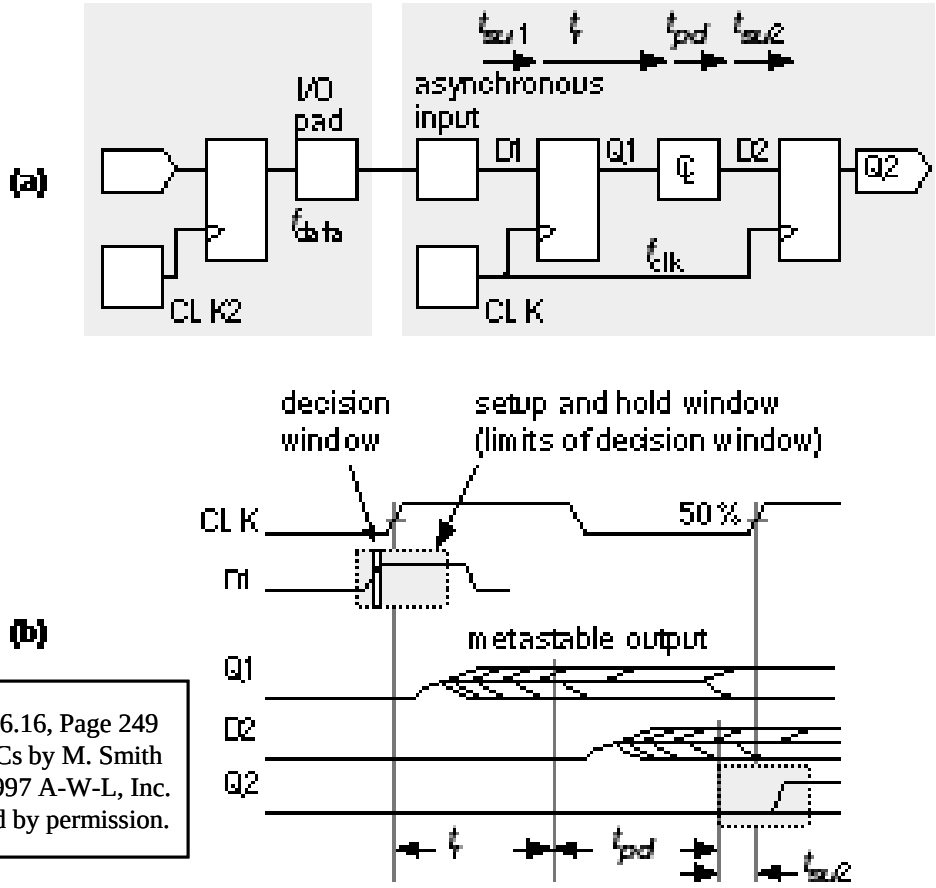


Fig. 6.16, Page 249
ASICs by M. Smith
© 1997 A-W-L, Inc.
Used by permission.