Objective

After finishing this training, you will be able to

- Understand the history of VHDL
- Understand the synthesis process
- Become familiar with structure and syntax
- Implement combinatorial and registered components with the language
- Complete a VHDL design in ISP Lever System

Introduction

VHSIC Hardware Description Language

- VHSIC - Very High Speed Integrated Circuit

Intended by DoD as a standard means to document complex circuit among contractors

Established as IEEE standard 1076 in 1987

Updated as IEEE standard 1164 in 1993

Nowadays VHDL is used to

- Document circuits
- Synthesize design descriptions
- Simulate circuits

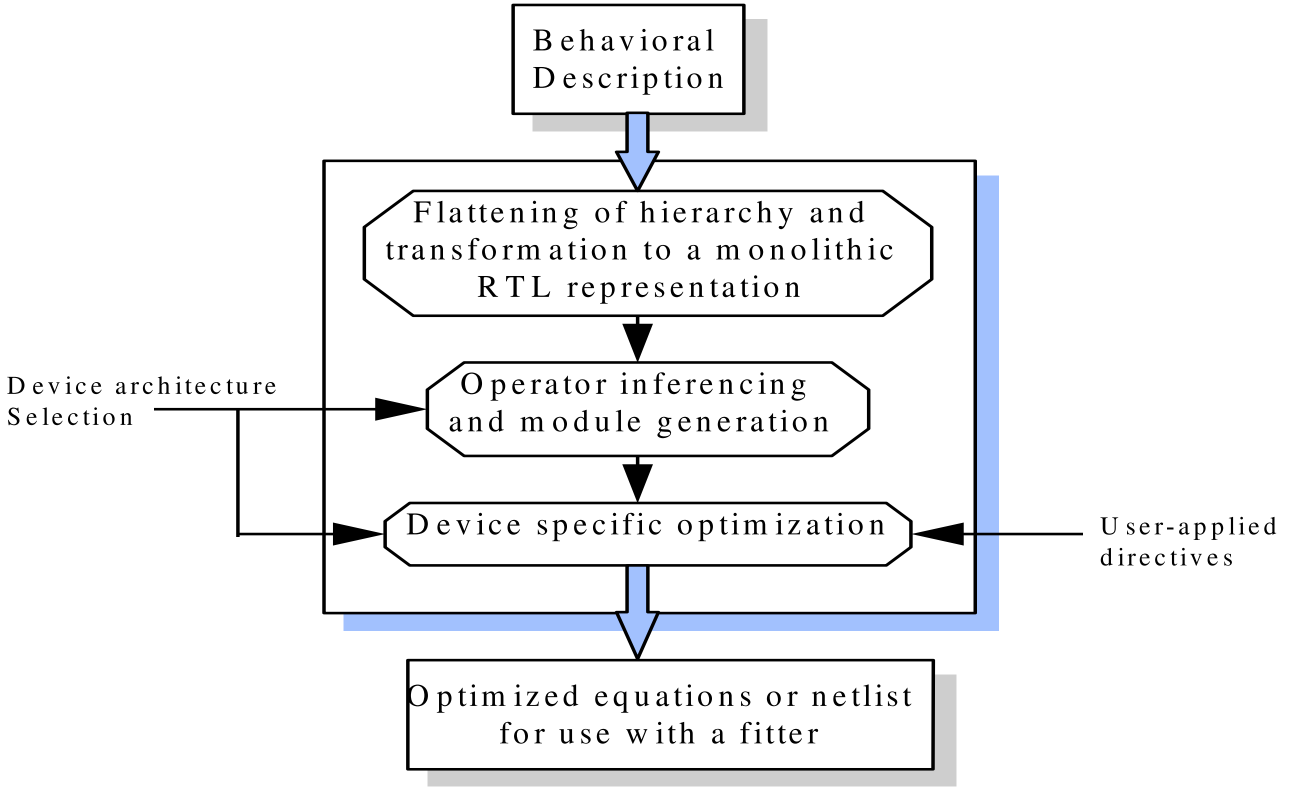
About VHDL

- Advantages
  - Device-independent design
    - do not have to be familiar with device architectures
  - Portability
    - same description for synthesis and simulation tools
    - same description for different platforms
  - Fast time-to-market and low cost
    - facilitates speedy design process and design iteration
- Possible Shortcomings
  - Loss control of defining gate-level circuit implementation
  - Inefficient logic implementations created by synthesis tools

About Synthesis

It is the realization of design descriptions into circuits

VHDL synthesis tools convert VHDL descriptions to technology-specific equations and netlists

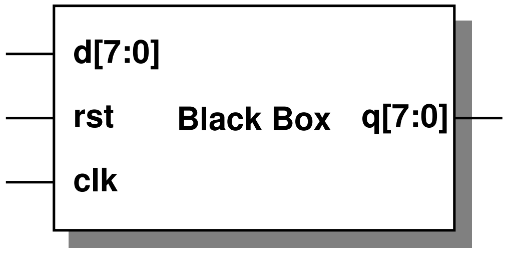


Behavioral Description
Flattening of hierarchy and transformation to a monolithic RTL representation
Device architecture Selection
Operator inferencing and module generation
Device specific optimization
User-applied directives
Optimized equations or netlist for use with a fitter

VHDL Design Descriptions

VHDL design descriptions consist of an ENTITY and ARCHITECTURE pair

- The ENTITY describes the design I/Os



- The ARCHITECTURE describes the content of the design
  - Structural Description:
    - Instantiation of building blocks referred to as components
    - Schematic type of placement and connections
  - Behavioral Description:
    - Abstract descriptions and Boolean equations
    - Increase productivity, portability and readability of the design

VHDL Overview

```
LIBRARY ...;
USE ...;
ENTITY black_box IS
        PORT    ( ... );
END black_box;
```

Libraries
Entity
Port

```
ARCHITECTURE black_box_arch OF black_box IS
        -- global signal declarations
        -- global constant declarations

BEGIN
        name:PROCESS(sensitivity list)
                -- local variable declarations
        BEGIN
                -- sequential statements
        END PROCESS name;
        ...

        -- concurrent statements
        ...
END black_box_arch;
```

Architecture
Declarations

Processes

Concurrent statements

The Entity - example

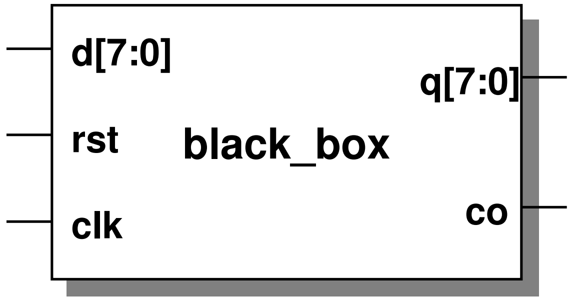
VHDL description of the black box:

```
ENTITY black_box IS
        PORT(rst, clk     : IN      std_logic;
             d            : IN      std_logic_vector (7 downto 0);
             q            : BUFFER  std_logic_vector (7 downto 0);
             co           : OUT     std_logic );
END black_box;
```

Name

Mode

Type



Names can be chosen by designer, BUT no reserved words, no äöüß..., no number as first character

Lattice Semiconductor Corporation

Bringing the Best Together

VHDL Reserved Words

Use of reserved words as names will cause errors

ABS
ACCESS
AFTER
ALIAS
ALL
AND
ARCHITECTURE
ARRAY
ATTRIBUTE

BEGIN
BLOCK
BODY
BUFFER
BUS

CASE
COMPONENT
CONSTANT

DOWNTO

ELSE
ELSIF
END
ENTITY
EXIT

FILE
FOR
FUNCTION

GENERIC

IF
IN
INOUT
IS

LABEL
LIBRARY
LINKAGE
LOOP

MAP

NAND
NEW
NEXT
NOR
NOT
NULL

OF
ON
OPEN
OR
OTHERS
OUT

PACKAGE
PORT
PROCECURE
PROCESS

RANGE
RECORD
REGISTER
REM
REPORT
RETURN

SELECT
SIGNAL
SUBTYPE

THEN
TO
TRANSPORT
TYPE

UNITS
UNTIL
USE

VARIABLE

WAIT
WHEN
WHILE
WITH

XOR
XNOR

The Entity

- Points of communication of ENTITY are PORTs
  - Always associate with I/Os of a components or device pins
  - Similar to “symbols” in a schematic
  - Does not describe the function of the block
- Each PORT must have
  - A name: must be unique within ENTITY
  - A list of properties
    - a direction - known as MODE
    - a value the PORT can take - known as TYPE

PORT Modes

IN	A signal that goes into the entity but not out
OUT	A signal that goes out of the entity but not in and is <u>not used internally</u>
INOUT	A signal that is bidirectional (goes into and out of the entity)
BUFFER	A signal that goes out of the entity and is also fed-back internally within the entity
BUFFER	is a subset of INOUT but it is not driven externally

PORT Types

VHDL is a strongly typed language. You cannot assign one signal type to another signal type

integer	Useful as index holders for loops and constants. Not usually used for I/O signals.	control structures
boolean	Can take values of 'TRUE' and 'FALSE'	control structures
std_logic	Standard industry logic type, has values of '0', '1', 'X', and 'Z' defined by IEEE std 1164.	hardware signals
std_logic_vector	A grouping of std_logic, standard industry logic type, definition for busses	hardware signals

The ENTITY - Example

`d` is a 16-bit input bus

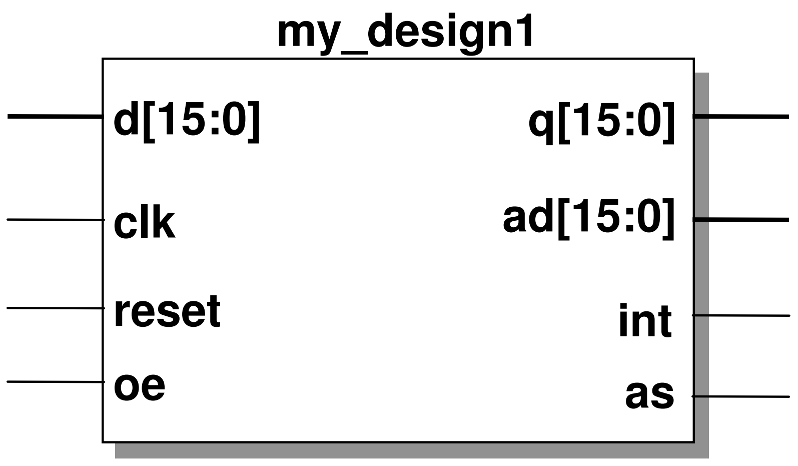
`clk`, `reset` and `oe` are input signals

`q` is a 16-bit tri-stateable output bus

`ad` is a 16-bit bi-directional bus

`int` is an output signal, that is also sensed internally

`as` is a tri-stateable output signal



The Entity - Example

```
ENTITY my_design1 IS
        PORT (d             : IN      std_logic_vector (15 downto 0);
            clk,reset,oe : IN      std_logic;
            q             : OUT     std_logic_vector (15 downto 0);
            ad            : INOUT   std_logic_vector (15 downto 0);
            int           : BUFFER  std_logic;
            as            : OUT     std_logic );
END my_design1;

--                q : OUT std_logic_vector (15 downto 0);
--                        MSB = q(15) ; LSB = q(0);
--                q : OUT std_logic_vector (0 to 15);
--                        MSB = q(0) ; LSB = q(15);

-- Two dashes indicate a comment line in VHDL
```

The Architecture

```
ARCHITECTURE black_box_arch OF black_box IS
        -- global signal declarations
        -- global constant declarations

BEGIN
        name:PROCESS(sensitivity list)
                -- local variable declarations
        BEGIN
                -- sequential statements
        END PROCESS name;
        ...

        -- concurrent statements
        ...
END black_box_arch;
```

Architecture

Declarations

Processes

Concurrent statements

Lattice Semiconductor Corporation
Bringing the Best Together

The Architecture

ARCHITECTURE describes what is in the ENTITY

It describes the behavior of the design

ARCHITECTURE contains the following statements:

Concurrent statements and PROCESS statements:

Statements outside of a PROCESS statement are "concurrent statements".

These statements and processes are evaluated concurrently and are evaluated independently of the order in which they appear.

Sequential statements :

Statements within a PROCESS statement are "sequential statements" and are evaluated sequentially in terms of simulation.

The Architecture - example

```
-- This example is simple logic

ENTITY logic IS
        PORT (a,b       : IN   std_logic;
              w, x, y   : OUT  std_logic;
              z         : OUT  std_logic_vector (3 downto 0) );
END logic;

ARCHITECTURE behavior OF logic IS

BEGIN

        y <= (a AND b);
        w <= (a OR b);
        x <= ‘1’;
        z <= “0101”;

END behavior;
```

Libraries

Library is a place to keep *precompiled packages* so that they can be used in other designs

```
LIBRARY ieee;  -- symbolic name for IEEE standard library
USE ieee.std_logic_1164.ALL;  -- name of the package
USE ieee.std_logic_unsigned.ALL;
```

The “ieee.std_logic_unsigned” library allows the use of certain operators on “std_logic” type signals.

eg. `(count <= count + 1;)`

Do not mix signed and unsigned!!! (scope of `USE` is file)

Entity / Architecture / Libraries - example

Every design has an ENTITY/ARCHITECTURE pair

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;

ENTITY logic IS
        PORT (   a,b       : IN   std_logic;
                 w, x, y   : OUT  std_logic;
                 z         : OUT  std_logic_vector (3 downto 0) );
END logic;

ARCHITECTURE behavior OF logic IS
 BEGIN

        y <= (a AND b);
        w <= (a OR b);
        x <= '1';
        z <= "0101";
 END behavior;
```

Concurrent Statements - examples

Concurrent statements include

boolean equations

```
x <= (a AND (NOT sel1)) OR (b AND sel1);
g(0) <= NOT (y AND sel2);
```

conditional assignments (i.e., when...else...)

```
y    <= d WHEN (sel1 = '1') ELSE c;
g(1) <= '0' WHEN (x = '1' AND sel2 = '0') ELSE '1';
z    <= "01010101";    -- Binary assignment
z    <= x"55";         -- Hexadecimal assignment
```

instantiations

```
inst1: fd11 PORT MAP (d=>din, clk=>clka, q=>qout);
```

WITH - SELECT - WHEN

Used for concurrent signal assignment (Example):

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY mux IS
   PORT (a, b, c, d  : IN   std_logic;
         s           : IN   std_logic_vector(1 downto 0;
         x           : OUT  std_logic );
END mux;

ARCHITECTURE mux_with OF mux IS
BEGIN
   WITH s SELECT
     x <= a WHEN “00”,     -- x is assigned based on s
          b WHEN “01”,
          c WHEN “10”,
          d WHEN “11”;
END mux_with;
```

WHEN - ELSE

Same example of 4-to-1 mux

```
ARCHITECTURE mux_when OF mux IS
BEGIN
        x <= a WHEN ( s = “00” ) ELSE
             b WHEN ( s = “01” ) ELSE
             c WHEN ( s = “10” ) ELSE
             d;
END mux_when;
```

WITH-SELECT-WHEN must specify mutually exclusive conditions

WHEN-ELSE does not have to

Standard VHDL Operators

Logical Operators

AND
OR
XOR
NOT

Relational Operators

=	Equal to
/=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

The ‘<=‘ operation is also used to signify ‘taking on the value of’

Sequential Statements

A PROCESS is used to describe <u>sequential</u> events and is included in the ARCHITECTURE of the design.

An ARCHITECTURE can contain several PROCESS statements.

PROCESS statements have 3 parts:

- Sensitivity list :
  - includes signals used in the PROCESS
  - process is activated when a signal in sensitivity list changes value
- PROCESS :
  - the description of behavior
- END statement:
  - describes the end of the PROCESS

Process - Sequential Statement

Simple example of PROCESS

```
mux: PROCESS (a, b, s) -- the sensitivity list
BEGIN
        IF ( s = '0' ) THEN
                x <= a;
        ELSE                                      -- define the process section
                x <= b;
        END IF;
END PROCESS mux;
```

Here the process 'mux' is sensitive to signals 'a','b' and 's'. Whenever signal 'a' or 'b' or 's' changes value, the statements inside the process will be evaluated

IF - THEN - ELSE

IF is a sequential statement and can only be used in PROCESS
To select a specific execution path based on Boolean evaluation of a condition or set of conditions

```
PROCESS (sel, a, b, c, d)
BEGIN
        IF (sel = “00”) THEN
                step <= a;
        ELSIF (sel = “01”) THEN
                step <= b;
        ELSIF (sel = “10”) THEN
                step <= c;
        ELSE
                step <= d;
        END IF;
END PROCESS;
```

Must have an “END IF” statement for every “IF” statement

Caution using “IF-THEN-ELSE”

The following PROCESS does not specify the value of “q” when “a1” is equal to “0”, thus creating a “Latch”.

```
PROCESS (a1, d)
BEGIN
        IF (a1 = ‘1’) THEN
                q <= d;
        END if;
END PROCESS;
```

The following PROCESS specifies the value of “q” when “a1” is equal to “0”, thus creating an AND gate.

```
PROCESS (a1, d)
BEGIN
        IF (a1 = ‘1’) THEN
                q <= d;
        ELSE null; -- don´t care
        END if;
END PROCESS;
```

CASE - WHEN

Lattice Semiconductor Corporation
Bringing the Best Together

CASE is a <u>sequential</u> statement and can only be used in PROCESS

```
ARCHITECTURE archdesign OF design IS
BEGIN
        decode: PROCESS (a, b, c, option)
        BEGIN
                CASE option IS
                        WHEN "00"    => output <= a;
                        WHEN "01"    => output <= b;
                        WHEN "10"    => output <= c;
                        WHEN OTHERS => output <= '0';
                END CASE;
        END PROCESS decode;
END archdesign;
```

OTHERS is all other possible value for signals of type std_logic

Caution using “CASE-WHEN”

This PROCESS generates an unwanted latch because not all states are defined.

```
PROCESS (set,a,b)
BEGIN
  CASE sel IS
    WHEN “00” => q<=a;
    WHEN “11” => q<=b;
  END CASE;
END PROCESS;
```

This PROCESS generates a multiplexer correctly because states “10” and “01” are defined.

```
PROCESS (set,a,b)
BEGIN
  CASE sel IS
    WHEN “00”   => q<=a;
    WHEN “11”   => q<=b;
    WHEN OTHERS => q<=‘0’;
  END CASE;
END PROCESS;
```

Data Objects

SIGNAL statement

- used to declare <u>internal</u> signals; external signals are declared in port statement of entity
- interconnects components or processes
- may be assigned to an external signal

```
ARCHITECTURE behavior OF example IS
    SIGNAL count:    std_logic_vector (3 downto 0);
    SIGNAL flag:     std_logic;
    SIGNAL mtag:     std_logic_vector (0 to 3);
    SIGNAL stag:     std_logic_vector (6 downto 0);
BEGIN
--  always declared in ARCHITECTURE section
```

- represents state elements in a state-machine

```
ARCHITECTURE behavior OF example IS
    TYPE states IS (state0, state1, state2, state3);
    SIGNAL memread:  states;
BEGIN
--  each state (state0, state1, etc.) represents a distinct state.
```

Data Objects

CONSTANT

holds a specific value of a type that cannot be changed within the design description

```
ARCHITECTURE behavior OF example IS
    CONSTANT width: integer := 8;
BEGIN
-- “width” is a constant with integer type and has a value of
“8”
```

VARIABLE

used to declare local values only within a given PROCESS.

```
PROCESS (s)
    VARIABLE count: std_logic;
BEGIN

-- value of “count” may be modified within this PROCESS
```

REGISTERS in Behavioral VHDL

3 ways to describe a register

```
PROCESS (clk)
     BEGIN
               IF (clk’event AND clk=‘1’) THEN    -- rising edge of clk
                        q <= d;
               END IF;
     END PROCESS;
-- falling edge of clk => (clk’event AND clk = ‘0’)

PROCESS (clk)
     BEGIN
               IF RISING_EDGE (clk) THEN
                        q <= d;
               END IF;
     END PROCESS;

PROCESS                                     -- no sensitivity list
     BEGIN
               WAIT UNTIL clk’event AND clk = '1';
                        q <= d;
     END PROCESS;
```

Synchronous Reset

```
upcount: PROCESS (clock)
         BEGIN
                  IF (clock’EVENT AND clock = '1') THEN
                           IF reset = '1' THEN
                                    count <= 0; -- synchronous
                           ELSE
                                    count <= count + 1;
                           END IF;
                  END IF;
         END PROCESS upcount;
```

Only CLOCK is in the sensitivity list because the process becomes activated only during clock transition

Asynchronous Reset

```
upcount: PROCESS (clock, reset)
        BEGIN
                IF (reset = '1') THEN    -- reset has higher priority
                      count <= 0;        -- asynchronous
                ELSIF (clock'EVENT AND clock = '1') THEN
                      count <= count + 1;
                END IF;
        END PROCESS upcount;
```

This process is sensitive to changes in both CLOCK and RESET, therefore both signals are included in the sensitivity list

LATCHES - in Behavioral VHDL

example of a D-Latch

```
ARCHITECTURE behavior OF dlatch IS
BEGIN
        PROCESS (ina, enable)
        BEGIN
                IF (enable = '1') THEN
                        outa <= ina;
                END IF;
        END PROCESS;
END behavior;
```

example of an SR-Latch

```
ARCHITECTURE behavior OF srlatch IS
BEGIN
        PROCESS (set, reset)
        BEGIN
                IF (set = '1' AND reset = '0') THEN
                        outa <= '1';
                ELSIF (set = '0' AND reset = '1') THEN
                        outa <= '0';
                END IF;
        END PROCESS;
END behavior;
```

Hierarchical Designs

Advantages:

- Allows duplication of common building blocks
- Components (VHDL models or schematic symbols) can be created, tested and held for reuse
- Smaller components can be more easily integrated with other blocks
- Design becomes more readable
- Design becomes easier to debug
- Design can be partitioned into groups and re-used by other design teams
- VHDL testbenches can be used to generate stimuli

COMPONENT

Lower level design

```
ENTITY add IS
PORT(op1,op2 : IN  std_logic_vector(2 downto 0);
     result  : OUT std_logic_vector(3 downto 0) );
END add;

ARCHITECTURE dataflow OF add IS
BEGIN
result <= op1 + op2;
END dataflow;
```

Lower level design can be instantiated in a higher level entity that may be in a separate file or in the same file
Device under test can be instantiated in the higher level testbench

COMPONENT (cont´d)

COMPONENT statement is a declaration of an VHDL entity which can be instantiated (placed) within other models

Example

```
ENTITY addmult IS
PORT (sig1,sig2 : IN  std_logic_vector (2 downto 0);
      res       : OUT std_logic_vector (4 downto 0) );
END addmult;

ARCHITECTURE structure OF addmult IS
        SIGNAL s_add: std_logic_vector (3 downto 0);
        COMPONENT add    -- component declaration (like entity
                                declaration)
        port(op1,op2 : IN  std_logic_vector (2 downto 0);
             result  : OUT std_logic_vector (3 downto 0));
        END component;
                                name of lower level entity
BEGIN                         key word connecting two levels
        add1: add PORT MAP(op1=>sig1, op2=>sig2, result=>s_add);
        res <= s_add * 2;
END structure;
```

VHDL Testbenches

A testbench in VHDL provides the design under test with user defined input signals described in VHDL
A VHDL testbench is tool and architecture independent
The testbench consists of

- an empty entity
- an architecture describing the behavior of the testbench
- the design under test included as a component
- signals for the port connections
- a port map connecting the design under test
- processes and concurrent statements defining the waveforms of the test signal

The testbench is the top level of a hierarchical design
For/Use statements connect entities and components in hierarchical designs
Configurations allow different combiations of entities and architectures

Testbench Example

```
LIBRARY ieee;

ENTITY device IS
PORT (clk, rst, din : IN std_logic;
      dout          : OUT std_logic);
END device;

ARCHITECTURE device_arch OF device IS
BEGIN
          PROCESS(rst,clk)
          BEGIN
          ...
          END PROCESS;
END device_arch;
```

```
LIBRARY ieee;
ENTITY test_device IS
END test_device;
ARCHITECTURE test_device_arch OF test_device IS
          COMPONENT device
          PORT (clk, rst, din : IN std_logic;
                dout          : OUT std_logic);
          END component;
SIGNAL clk_in,rst_in,din_in,dout_out : std_logic;
BEGIN
   dut:device PORT MAP(clk_in, rst_in, din, dout);
   clk_driver:process
          BEGIN
                    clk <= '1';
                    wait for 20 ns;
                    clk <= '1';
                    wait for 20 ns;

   END PROCESS;
   rst <= '0';
   din_in <= '1';
END device_arch;
```

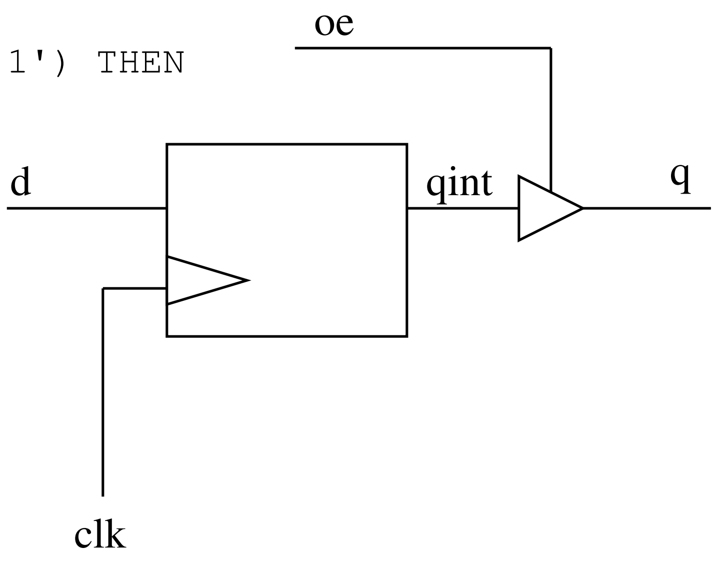
Device under test

Testbench

CPLD Optimization - Output Enables

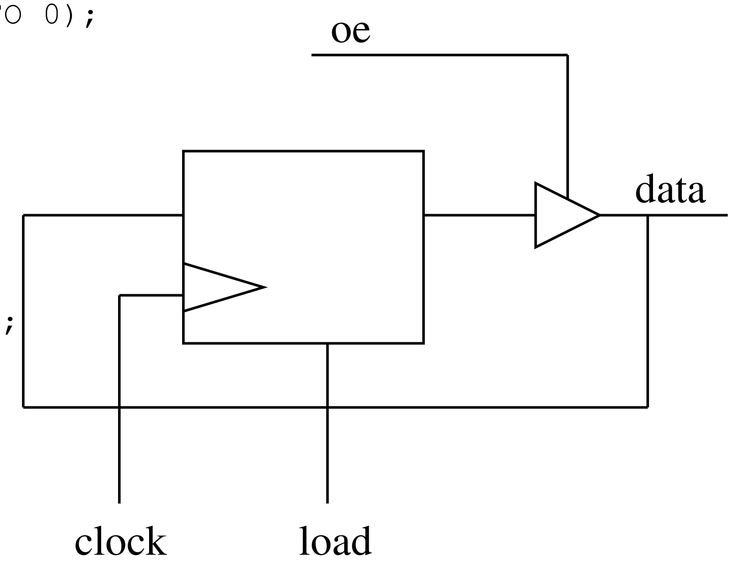
VHDL does not have an explicit OE, therefore need to describe the function of OE

```
ENTITY oe IS
        PORT (d      : IN  std_logic_vector (6 downto 0);
              q      : OUT std_logic_vector (6 downto 0);
              oe,clk : IN  std_logic );
END oe;

ARCHITECTURE behavioral OF oe IS
SIGNAL qint : std_logic_vector (6 downto 0);
BEGIN
        dff: PROCESS (clk)
        BEGIN
                IF (clk' event and clk='1') THEN
                   qint <= d;
                END IF;
        END PROCESS;
        q <= "ZZZZZZZ" WHEN (oe ='0')
             ELSE qint;
END behavioral;
```



CPLD Optimization - BiDirectional Signals

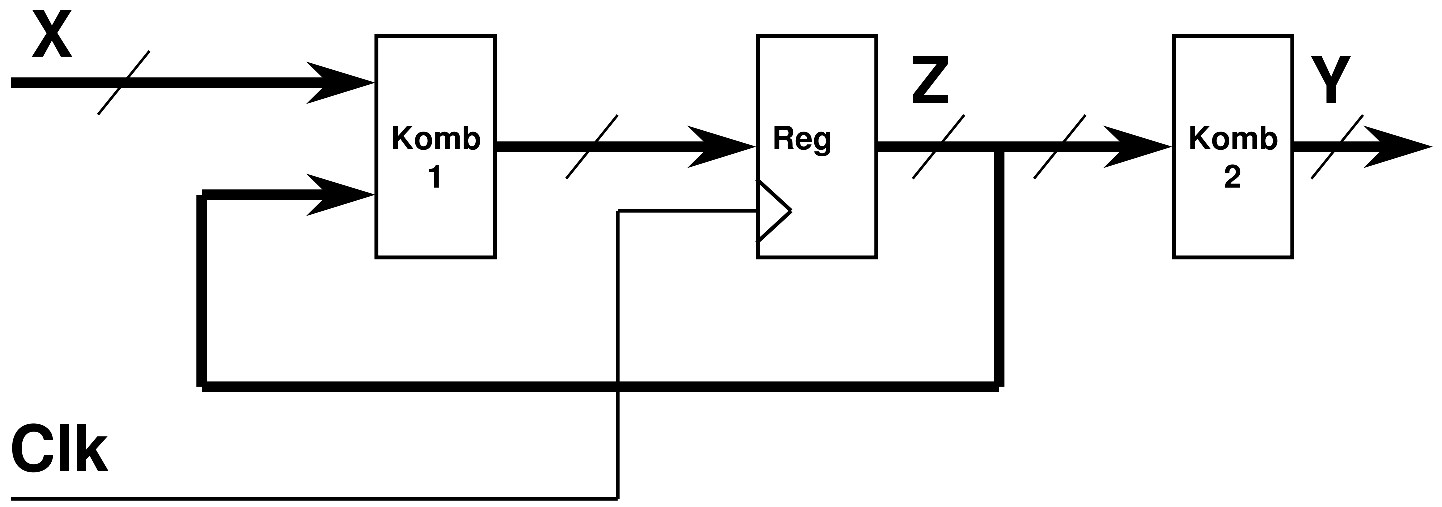
```
--        “data” is bi-directional EXTERNAL signal
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;

ENTITY loadcntr IS
         PORT (clock, load, oe   : IN    std_logic;
               data              : INOUT std_logic_vector(7 DOWNTO 0) );
END loadcntr;

ARCHITECTURE archloadcntr OF loadcntr IS
         SIGNAL count: std_logic_vector(7 DOWNTO 0);
BEGIN
    PROCESS (clock)
    BEGIN
         IF (clock'EVENT AND clock = '1') THEN
                  IF load = '1' THEN
                           count <= data;
                  ELSE
                           count <= count + 1;
                  END IF;
         END IF;
    END PROCESS;
    data <= “ZZZZZZZZ” WHEN oe = '0' ELSE count;
END archloadcntr;
```



Moore State Machine

Lattice Semiconductor Corporation
Bringing the Best Together

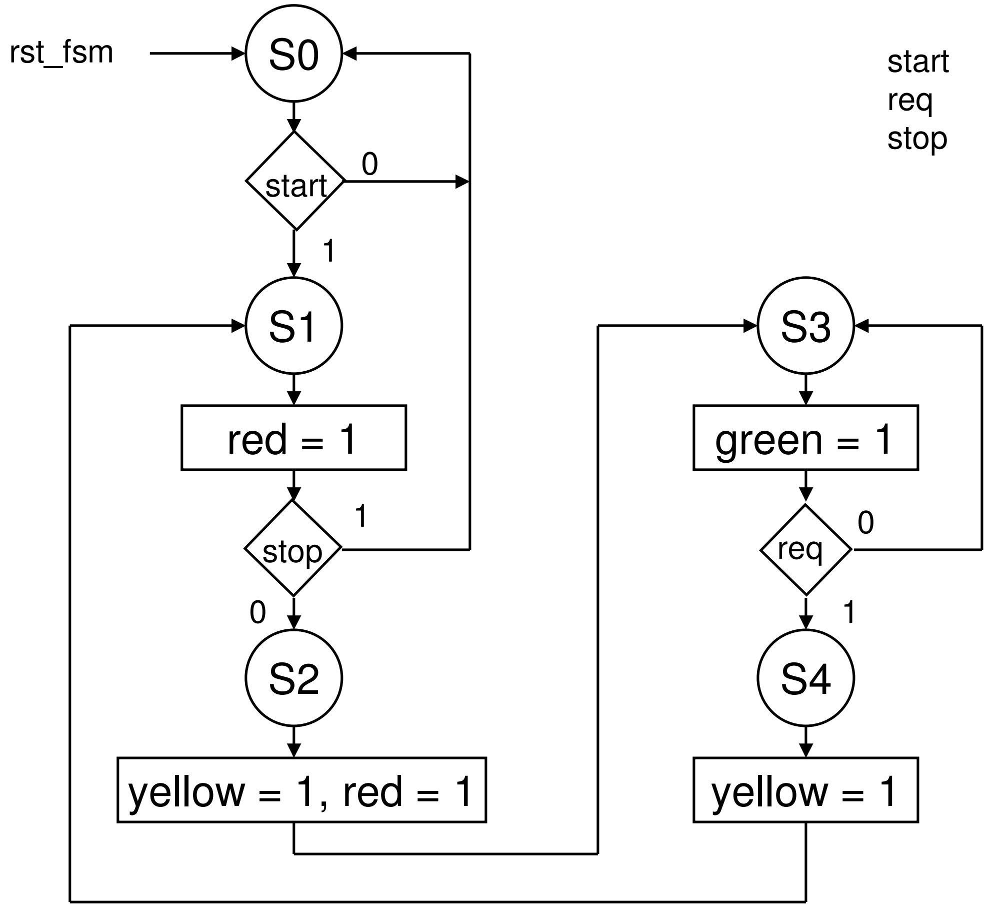
Moore State Machine

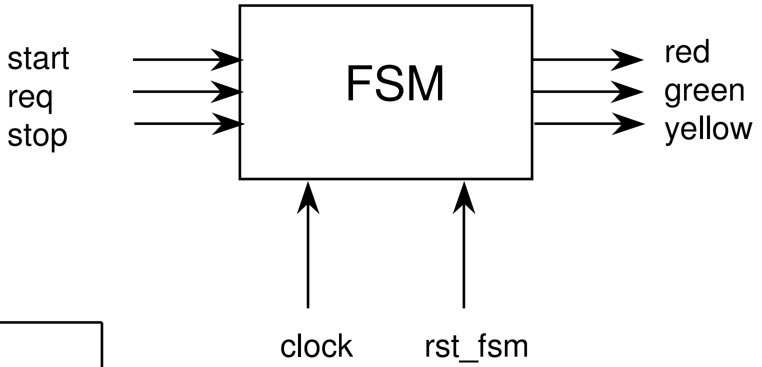


CPLD Optimization State Machine Moore

Lattice Semiconductor Corporation

Bringing the Best Together





CPLD Optimization State Machine Moore

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY traffic_light IS
        PORT(clock, rst_fsm       : IN  std_logic;
             start, req, stop     : IN  std_logic;
             red, green, yellow : OUT std_logic );
END;

ARCHITECTURE fsm_moore OF traffic_light IS

TYPE states IS (S0, S1, S2, S3, S4);
SIGNAL value, nextvalue : states;

BEGIN

        reg: PROCESS (clock, rst_fsm)
        BEGIN
                 IF (rst_fsm='1') THEN
                         value <= S0;
                 ELSIF (clock'EVENT AND clock='1') THEN
                         value <= nextvalue;
                 END IF;
        END PROCESS reg;
```

CPLD Optimization State Machine Moore

Lattice Semiconductor Corporation

Bringing the Best Together

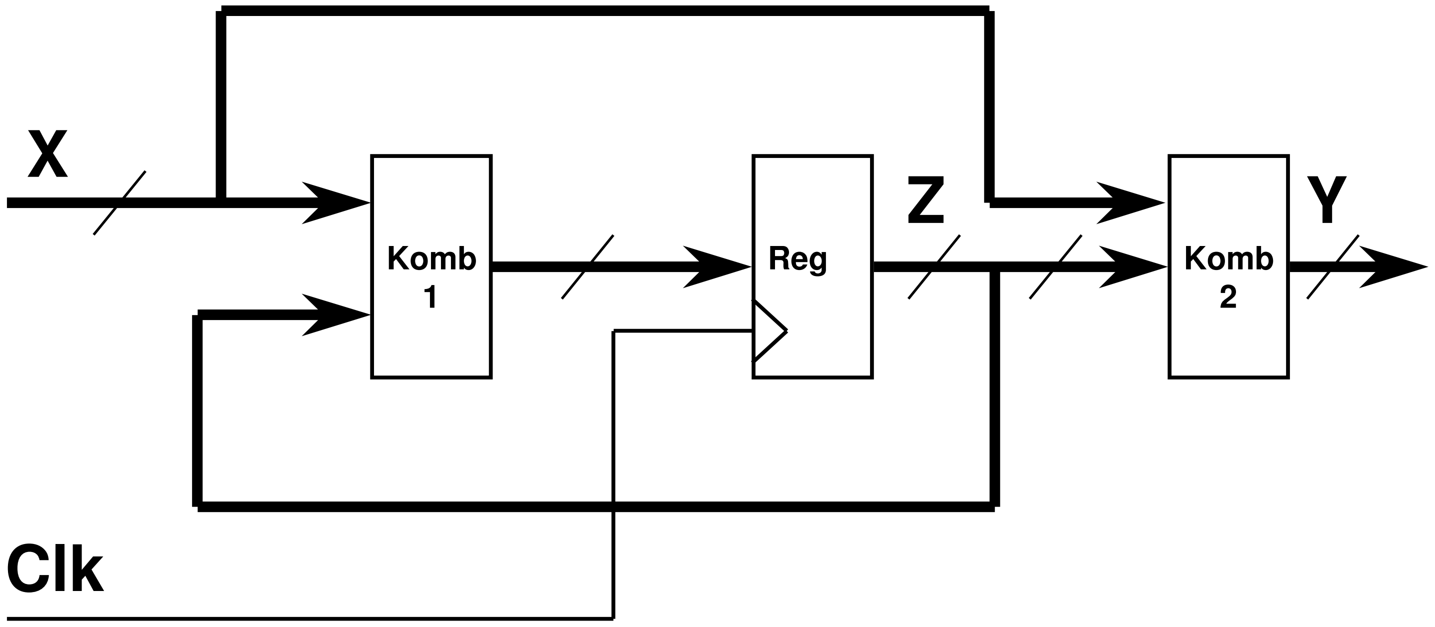
```
komb1: PROCESS (start, req, stop, value)
BEGIN
          CASE value IS
                    WHEN S0 =>
                              IF (start='1') THEN
                                        nextvalue <= S1;
                              ELSE
                                        nextvalue <= S0;
                              END IF;
                    WHEN S1 =>
                              IF (stop='1') THEN
                                        nextvalue <= S0;
                              ELSE
                                        nextvalue <= S2;
                              END IF;
                    WHEN S2 =>
                              nextvalue <= S3;
                    WHEN S3 =>
                              IF (req='1') THEN
                                        nextvalue <= S4;
                              ELSE
                                        nextvalue <= S3;
                              END IF;
                    WHEN S4 =>
                              nextvalue <= S1;
                    WHEN OTHERS =>
                              nextvalue <= S0;
          END CASE;
END PROCESS komb1;
```

CPLD Optimization   State Machine Moore

```
          komb2: PROCESS (value)
          BEGIN
                    red <= '0'; yellow <= '0'; green <= '0';
                    CASE value IS
                              WHEN S1 =>
                                        red <= '1';
                              WHEN S2 =>
                                        yellow <= '1';
                                        red <= '1';
                              WHEN S3 =>
                                        green <= '1';
                              WHEN S4 =>
                                        yellow <= '1';
                              WHEN OTHERS =>
                                        red <= '0';
                                        yellow <= '0';
                                        green <= '0';
                    END CASE;

          END PROCESS komb2;

END fsm_moore;
```

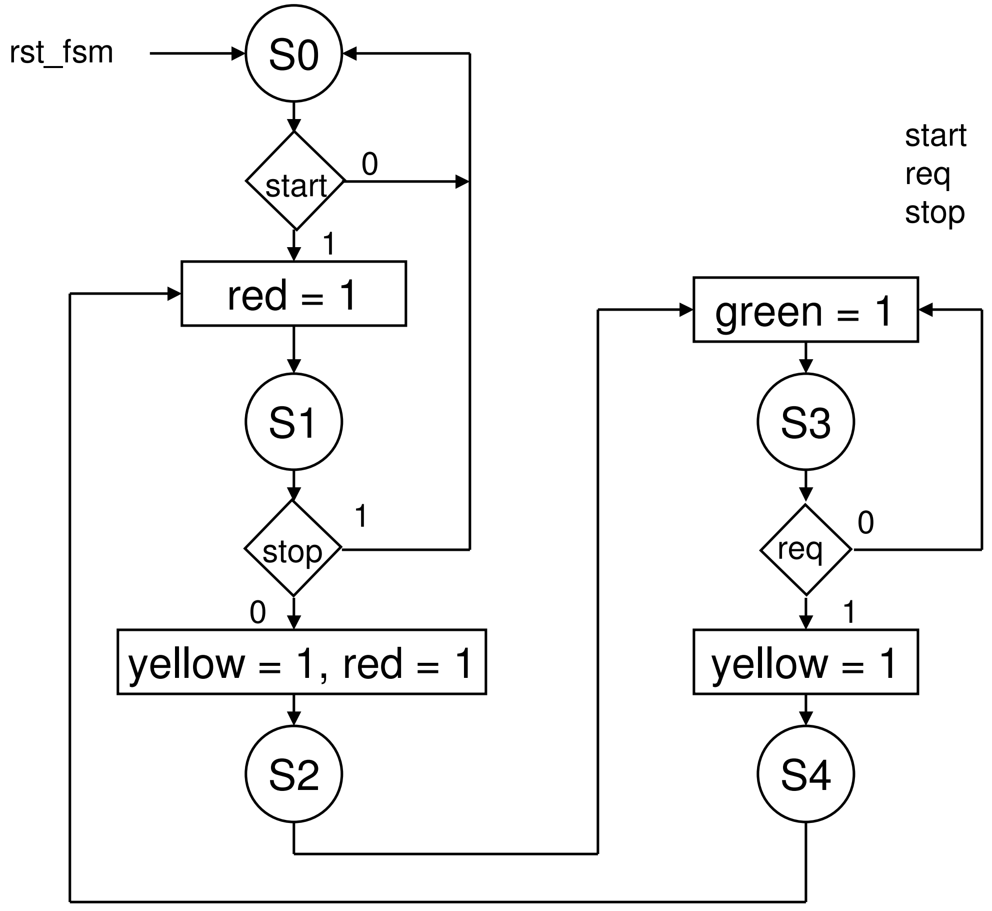
Mealy State Machine

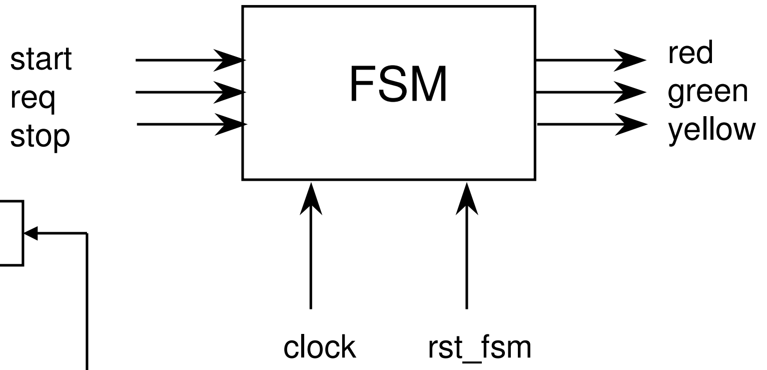
Mealy State Machine



CPLD Optimization State Machine Mealy

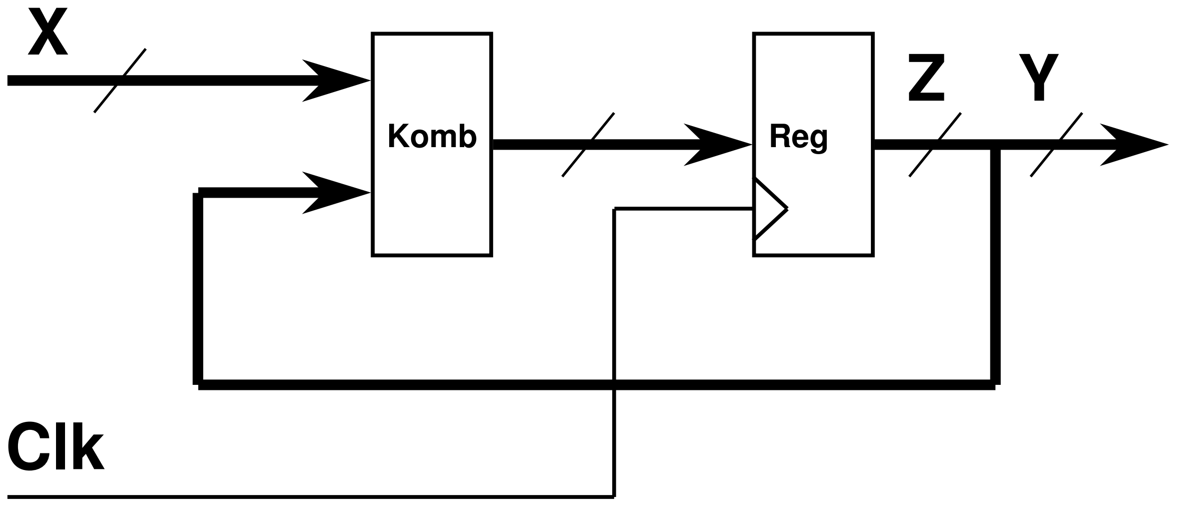
Lattice Semiconductor Corporation
Bringing the Best Together





CPLD Optimization   State Machine Medvedev

Medvedev State Machine



CPLD Optimization   State Machine Medvedev

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY traffic_light IS
          PORT(clock, rst_fsm     : IN  std_logic;
             start, req, stop   : IN  std_logic;
             red, green, yellow : OUT std_logic );
END;

ARCHITECTURE fsm_med OF traffic_light IS

SIGNAL states : std_logic_vector(2 downto 0);
-- Reihenfolge der Ausgangsregister: red, green,yellow
CONSTANT S0 : std_logic_vector(2 downto 0) := "000";
CONSTANT S1 : std_logic_vector(2 downto 0) := "100";
CONSTANT S2 : std_logic_vector(2 downto 0) := "110";
CONSTANT S3 : std_logic_vector(2 downto 0) := "001";
CONSTANT S4 : std_logic_vector(2 downto 0) := "010";

BEGIN
        komb_reg: PROCESS (clock, rst_fsm)
        BEGIN
                    IF (rst_fsm='1') THEN
                              states <= S0;
                    ELSIF (clock'EVENT AND clock='1') THEN
                              CASE states IS
                                        WHEN S0 =>
                                                  IF (start='1') THEN
                                                            states <= S1;
                                                  ELSE
                                                            states <= S0;
                                                  END IF;
```

CPLD Optimization   State Machine Medvedev

```
                                        WHEN S1 =>
                                                  IF (stop='1') THEN
                                                            states <= S0;
                                                  ELSE
                                                            states <= S2;
                                                  END IF;
                                        WHEN S2 =>
                                                  states <= S3;
                                        WHEN S3 =>
                                                  IF (req='1') THEN
                                                            states <= S4;
                                                  ELSE
                                                            states <= S3;
                                                  END IF;
                                        WHEN S4 =>
                                                  states <= S1;
                                        WHEN OTHERS =>
                                                  states <= S0;
                              END CASE;
                    END IF;

          END PROCESS komb_reg;

          red    <= states(2);
          yellow <= states(1);
          green  <= states(0);


END fsm_med;
```

CPLD Optimization - State Encoding

Many CPLD VHDL implementations use “one-hot” encoding. “One-hot” uses a register for each device state (state-per-bit), with only one register active (or “hot”) at a time.

Use “maximal” encoding for CPLDs it is typically faster and results in better device utilization

Don’t leave optimization solely to your tools; understand your device-architecture characteristics and tailor your design accordingly

Some tools automatically select an optimum state machine in response to your circuit implementation, guidance, or both (for example, prioritizing performance to the compiler), whereas other tools always default to one method unless you override them

Synplify has a checkbox “Symbolic FSM Compiler” which leads to an automatic recognition and special optimization of your FSMs if checked

CPLD Optimization - Undefined States

When all values are not explicitly defined, synthesis results may be unexpected

Whether using IF-THEN-ELSE, CASE-WHEN, WITH-SELECT-WHEN or WHEN-ELSE statements, not defining all possible values or states can result in unwanted “Latches”

Appendix I - Lattice Specific - Global OE

Lattice Semiconductor Corporation
Bringing the Best Together

To use the Global OE available in the CPLD, simply lock the OE signal to the Global OE pin in the Fitter.
If the OE controls several different outputs in the design, ensure that the polarity are the same and the polarity is the same as the Global OE in the CPLD.

```
q <= "ZZZZZZZ" when (oe='0') else qint;
```

Appendix I - Lattice Specific - Global Reset I

To use the Global Reset pin in the CPLD, ensure that ALL reset signals in the ENTITY have the same reset name

Keep in mind that ALL registers have implied resets that connect to the Global Reset pin. If you specify a reset signal in your design, then you need to specify a reset signal for EVERY register and latch used. Otherwise, the Fitter will assign it to a PT reset

In the Fitter, you must specify to “use Global Reset”

Appendix I - Lattice Specific - Global Reset II

Lattice Semiconductor Corporation
Bringing the Best Together

The following PROCESS does not specify a reset signal. There is however an implied reset signal (Global Reset) connected to the register.

```
PROCESS (clk)
BEGIN
        IF (clk'event AND clk = '1') THEN
                q <= d;
        END IF;
END PROCESS;
```

The following PROCESS specifies a reset signal. There is also an implied reset signal (Global Reset) connected to it. To make "rst" connect to Global Reset pin, must "use Global Reset" in the Fitter.

```
PROCESS (rst, clk)
BEGIN
        IF (rst = '1') THEN
                q <= '0';
        ELSIF (clk'event and clk = '1') THEN
                q <= d;
        END IF;
END PROCESS;
```

Appendix I - Lattice Specific - IO as a Register

VHDL file is similar to any Register description

```
ARCHITECTURE behavior OF ioreg IS

BEGIN

        PROCESS (rset, clka)
        BEGIN
                IF (rset = '1') THEN
                        qout <= '0';
                ELSIF (clka'event AND clka = '1') THEN
                        qout <= qin;
                END IF;
        END PROCESS;

END BEHAVIOR;
```

Appendix I - Lattice Specific - IO Cell as a Latch

VHDL file is similar to any D-type Latch description

```
ARCHITECTURE behavior OF iolatch IS

BEGIN

        PROCESS (le, qin)
        BEGIN
                IF (le = '1') THEN
                        qout <= qin;
                END IF;
        END PROCESS;

END BEHAVIOR;
```

Literature

Lattice Semiconductor Corporation
Bringing the Best Together

IEEE Standard VHDL Language Reference Manual
    IEEE-1076-1992/B
Structured Logic Design with VHDL
    J.R. Armstrong, F.G. Gray; Prentice Hall, Englewood Cliffs, 1993
Schaltungsdesign mit VHDL
    Lehmann, Wunder; Franzis, ISBN 3-7723-6163-3
The VHDL Cookbook
    P.J. Ashenden; via FTP *chook.adelaide.edu.au* or *du9ds4.fb9dv.uni-duisburg.de*
A VHDL Primer
    J. Bhasker, Prentice Hall 1992, ISBN 0-13-952987-X