

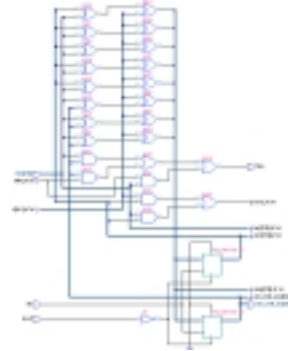
VHDL Application Workshop



```
library ieee, work;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.all;
library exemplar;
use exemplar.exemplar_1164.all;

entity counter is
port (clk, reset, enable: std_logic;
      numout: in std_logic_vector(3 downto 0);
      outnum: out std_logic_vector(3 downto 0));
end counter;

architecture rtl of counter is
  signal num: std_logic_vector(3 downto 0);
begin
  process(reset, clk)
  begin
    if reset = '1' then
      num <= "0000";
    elsif clk'event and clk = '1' then
      if enable = '1' then
        num <= num + 1;
      end if;
    end process;
    outnum <= num;
  end rtl;
end counter;
```



4-6 September 2000

Lecturer : Mr.Watcharakon Noothong
 Telephone: 739-2191..3 Ext 611
 E-Mail : noothong@notes.nectec.or.th
 WWW : <http://tmec.nectec.or.th/icdesign>

ฝ่ายออกแบบวงจรรวม ศูนย์วิจัยและพัฒนาเทคโนโลยีไมโครอิเล็กทรอนิกส์

Course Outline

Day 1

1. Introduction and Overview

- History, Limitation
- Electronic system design
- Levels of Abstraction

2. VHDL Objects

- Entity, Architecture
- Component, Package
- Process, Configuration

3. Signal and Datatypes

- Concept of a types
- Standard data types
- Scalar and array literals
- IEEE Standard logic types

4. Object Declarations

- Constant declaration
- Signal declaration
- Alias declaration
- Variable declaration

Day 2

5. VHDL Operators

- Logical Operators
- Relational Operators
- Concatenation Operator
- Arithmetic Operators

6. Sequential Statements

- Process Statement
- IF Statements
- CASE Statement
- LOOP Statements
- WAIT Statements

7. Concurrent Statements

- Condition signal assignments
- Selected signal assignments
- BLOCK/Guards Statement

8. Subprogram

- Functions
- Procedures

Day 3

9. Advanced Topics

- Advanced types
- Overloading
- RTL Styles

10. Testbench Coding Styles

- Testbench configuration
- Stimulus styles
- Assertion
- TextIO

11. VHDL & Logic Synthesis

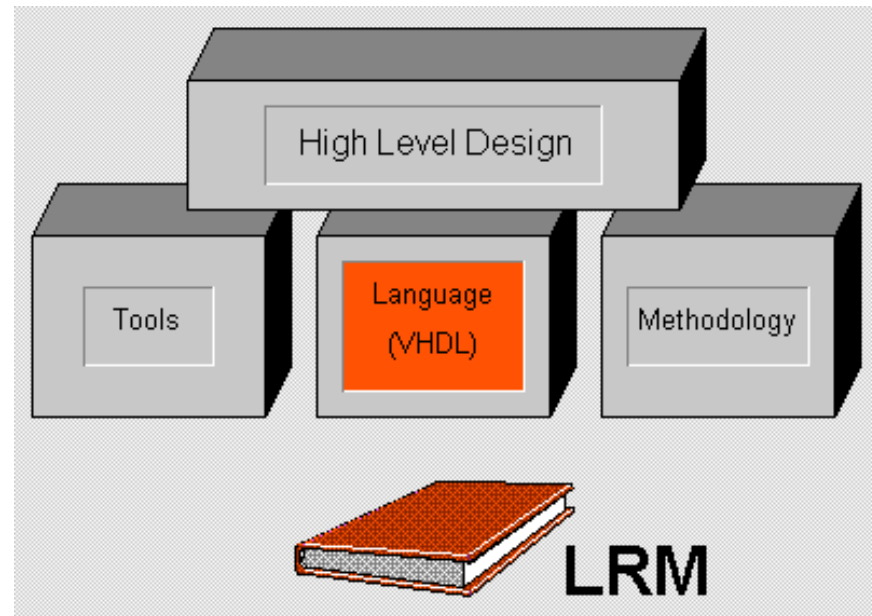
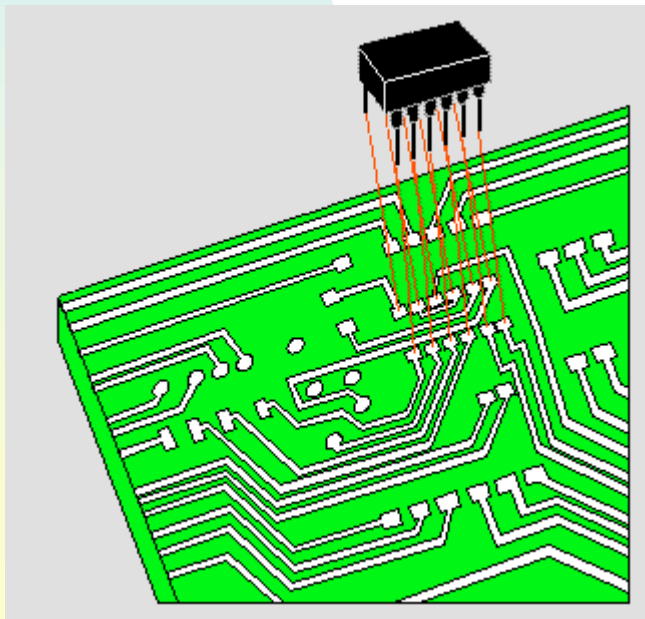
- Basic design methodology
- Multiplexer, Encoder, Decoder
- Tri-State, Bidirectional buffers
- ALU (Arithmetic Logic Unit)
- Finite State machine
- Latch, Asynchronous Flip-Flop & Synchronous Flip-Flop

Introduction and Overview

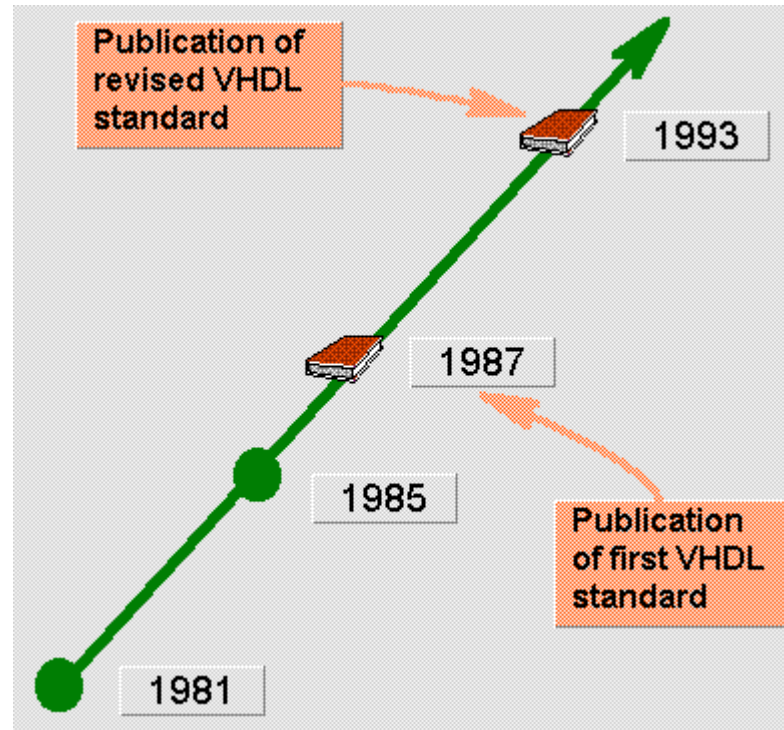
What is VHDL ?

VHSIC (V_{ery} H_{igh} S_{peed} I_{ntegrated} C_{ircuit}) H_{ardware} D_{escription} L_{anguage}

- Modelling DIGITAL Electronic Systems
- Both Concurrent and Sequential statements

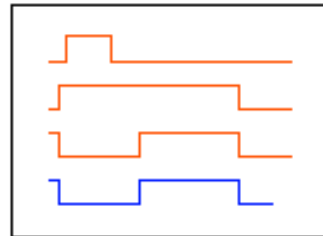


VHDL History

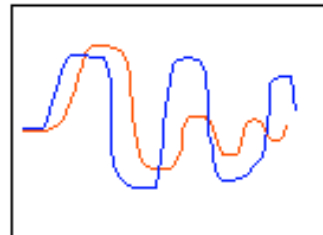


- Department of Defense (DoD) developed in 1981
- IEEE 1076-1987 Standard (VHDL '87)
- IEEE 1076-1993 Standard (VHDL '93)

VHDL Limitation



Digital



Analog

Now:



Soon:

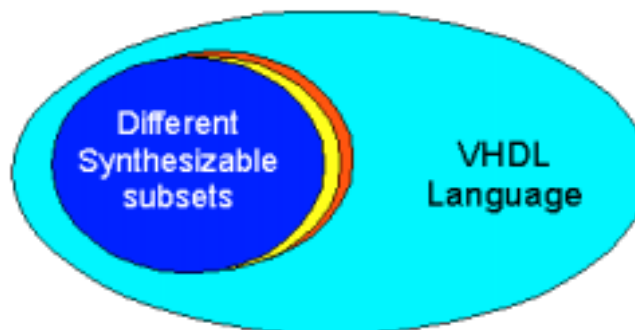


Hardware Description Language (HDL)

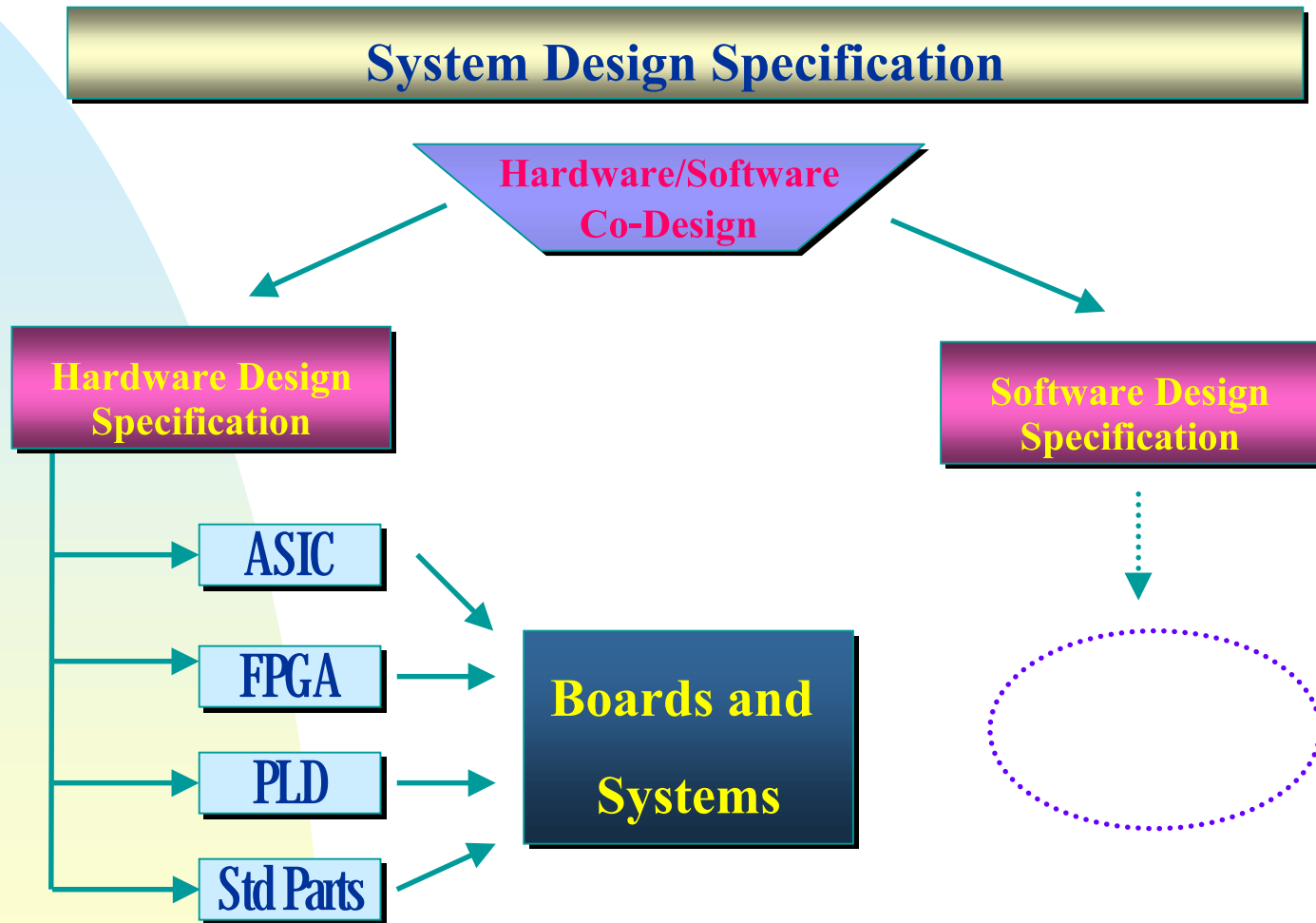
The VHDL is a software programming language used to model the intended operation of a piece of hardware, similar to Verilog-HDL.

Use of the VHDL Language

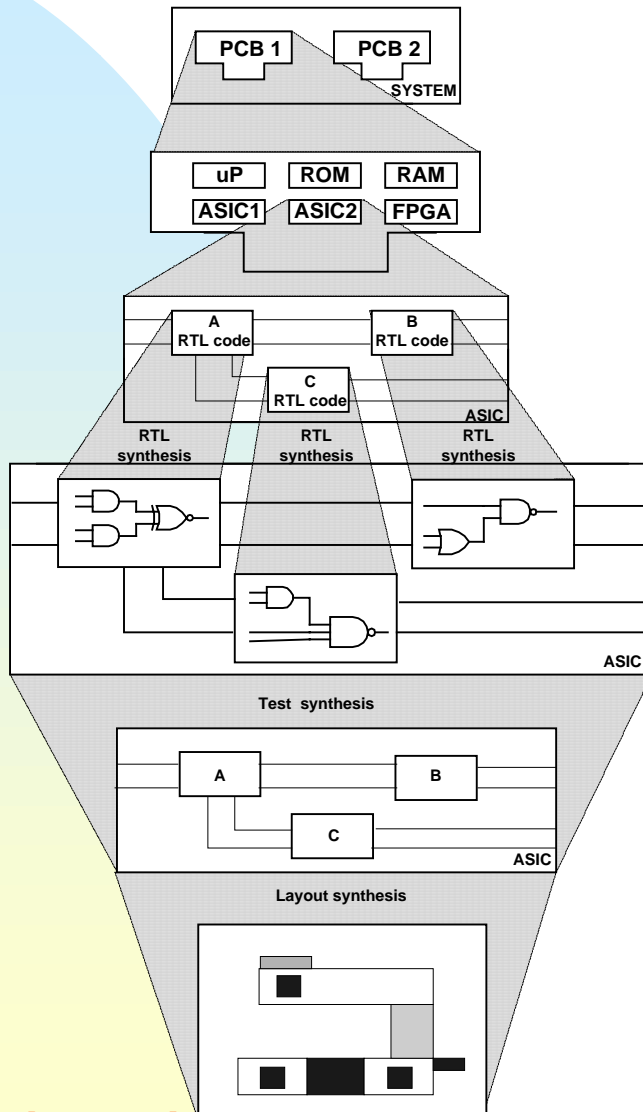
1. **Documentation Language:** To provide human and machine readable documentation
2. **Design Language:** To provide a structure reflecting hardware design and hierarchy and provide methods to handle complexity by partitioning the design.
3. **Verification Language:** To provide a concurrent method verifying hardware interaction and provide constructs for stimulating a design.
4. **Test Language:** To provide ability to generate test vectors, multiple testbench strategies and method to write self checking code.
5. **Synthesis Language:** To provide high-level constructs that can be translated to boolean equations and then translate them to gate as well as to provide constructs that can be optimized.



Electronic System Design



Top-Down Design



The top-level system is modeled for functionality and performance using a high-level behavioural description.

Each major component is modeled at the behavioural level and the design is simulated again for functionality and performance.

Each major component is modeled at the gate level and the design is simulated again for timing, functionality and performance.

Levels of Abstraction : Capture

V
H
D
L

Editing VHDL Code

Block Capture

System Level Tools

Schematic Capture

Layout Tools

Behavioural



RTL



Logic



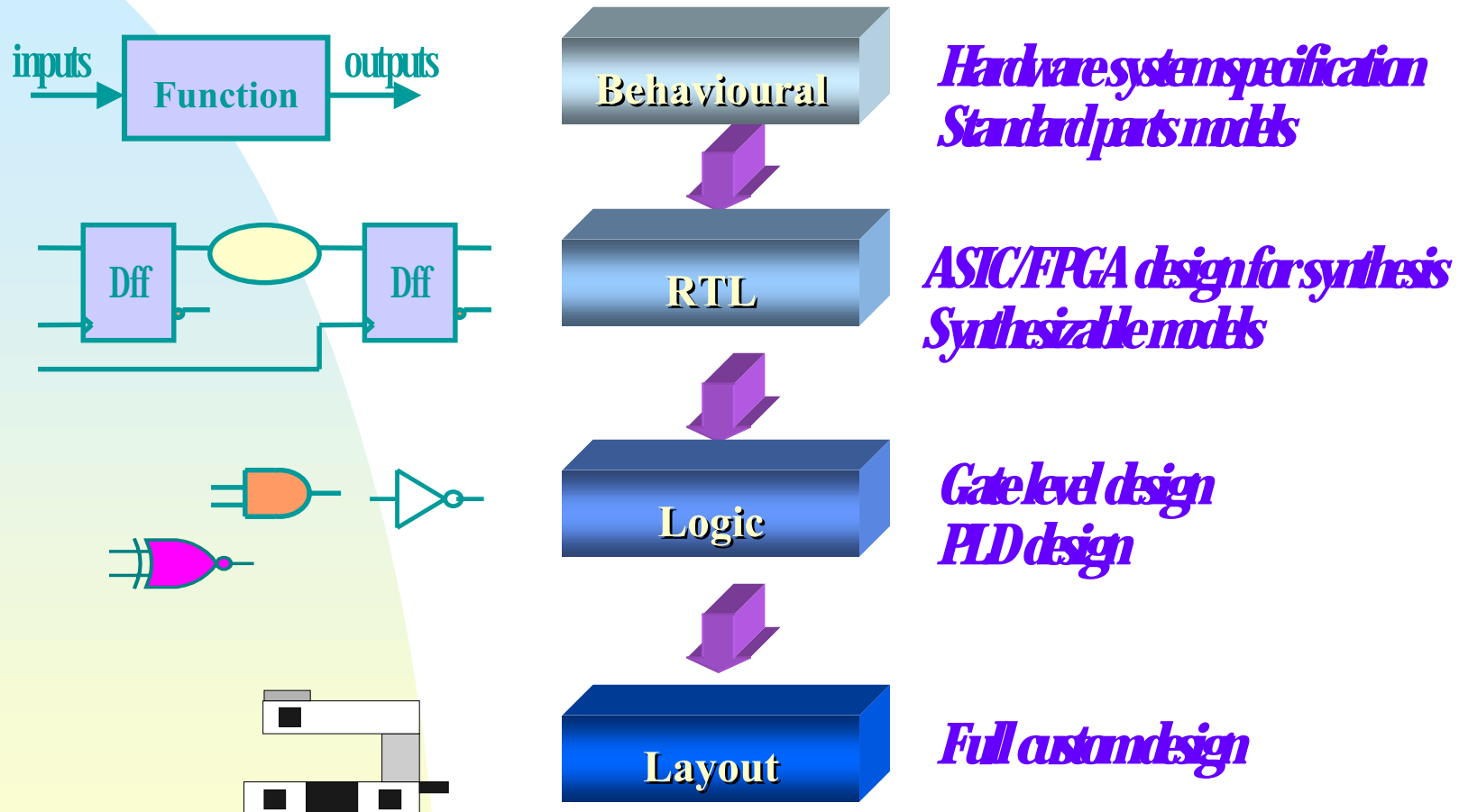
Layout

Behavioural Synthesis

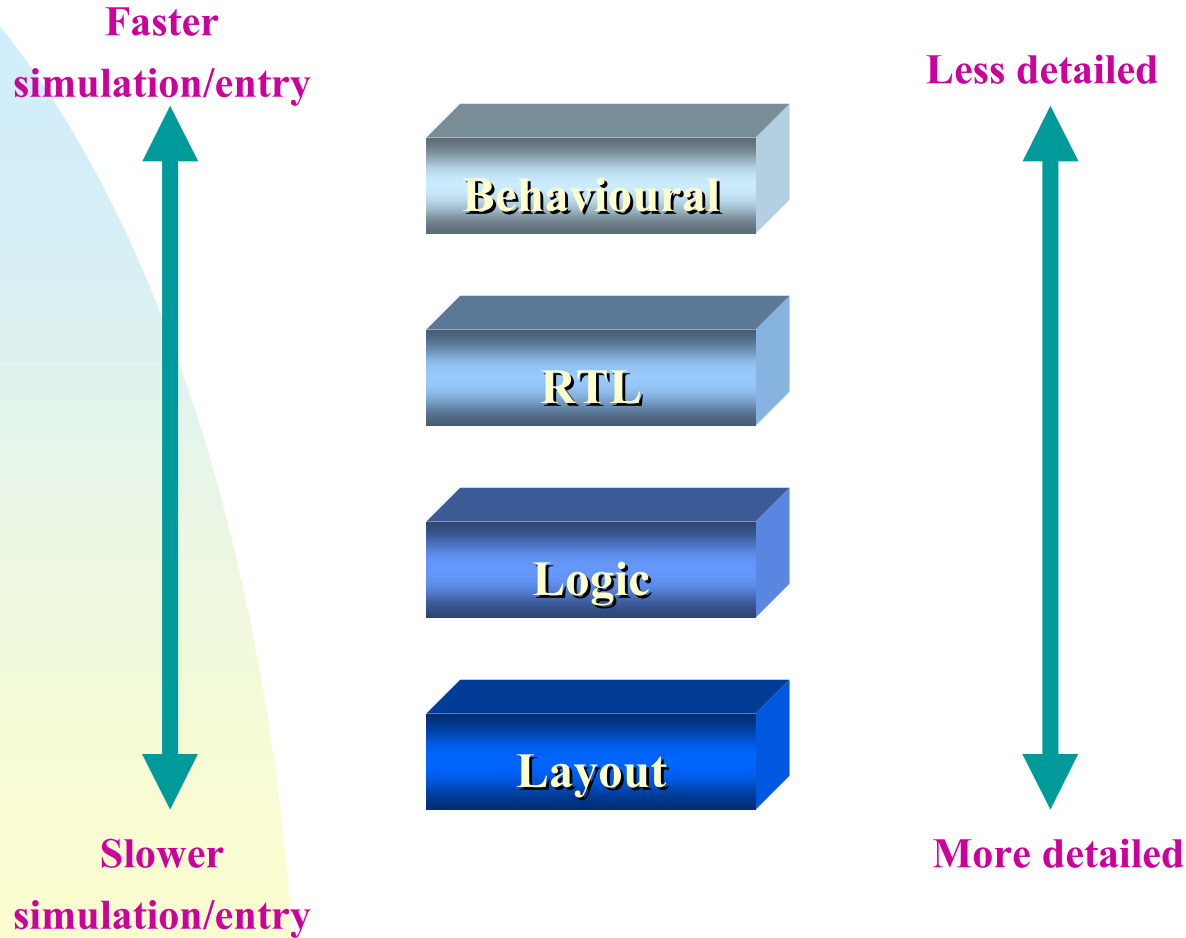
Logic Synthesis

Place and Route

Levels of Abstraction : Definition



Tradeoffs Between Abstraction Levels



The Benefits of Using VHDL

- Design at a higher level
 - ◆ Find problems earlier
- Implementation independent
 - ◆ Last minute changes
 - ◆ Delay implementation decisions
- Flexibility
 - ◆ Re-use
 - ◆ Choice of tools, vendors
- Language based
 - ◆ Faster design capture
 - ◆ Easier to manage

VHDL Organisations

- **VHDL International**
 - ◆ Promoting use of VHDL
 - ◆ Quarterly publication, “Resource Book”
 - ◆ “Simulator Certification Test Suite”
- **IEEE Committees**
 - ◆ VASG : defining the language : various others too
(VASG : VHDL Analysis and Standardization Group)
- **Europe**
 - ◆ European chapter of VASG
 - ◆ VHDL Forum for CAD in Europe
 - ◆ VHDL Newsletter, published by Esprit project
 - ◆ Regional user groups : VHDL-UK, etc . . .
- **Newsgroups**
 - ◆ `comp.lang.vhdl`

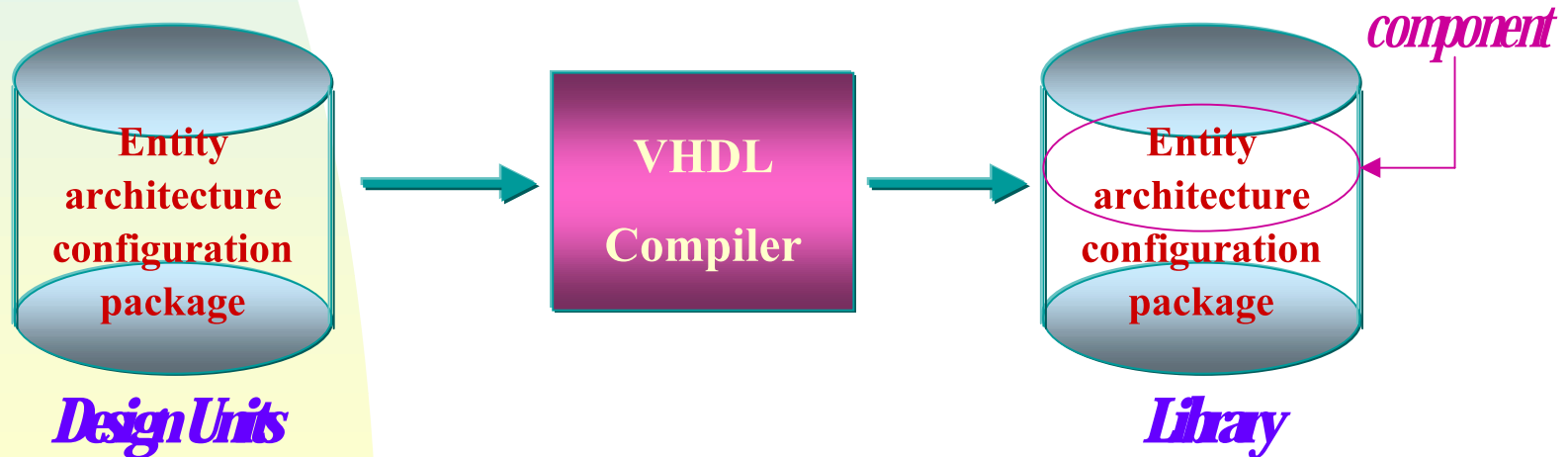
VHDL Objects

Aim and Topics

- Introduction to :
 - ◆ Entity
 - ◆ Architecture
 - ◆ Package
 - ◆ Configuration

Introduction

- VHDL design consists of several separate four design units :
 - ◆ Entity design unit
 - ◆ Architecture design unit
 - ◆ Configuration design unit
 - ◆ Package design unit



Component

VHDL Component

```
entity HALFADD is  
  port ( A, B : in bit;  
         SUM, CARRY : out bit );  
end HALFADD;
```

Entity declaration

```
architecture BEHAVE of HALFADD is  
  begin  
    SUM <= A xor B;  
    CARRY <= A and B;  
  end BEHAVE;
```

Architecture declaration

Entity declaration

Syntax:

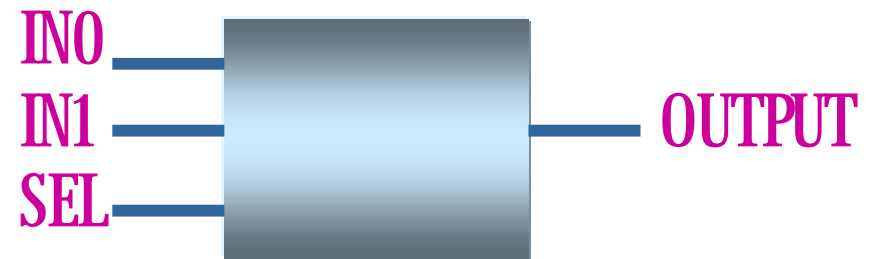
```
entity entity_name is  
    generic (generic_list);  
    port( port_name: <mode> port_type );  
end entity_name;
```

< mode > = in, out, inout, buffer, linkage
in = Component only read the signal
out = Component only write to the signal
inout = Component read or write to the signal (bidirection signals)
buffer = Component write and read back the signal (no bidirectional)
linkage = Used only in the documentation

VHDL 93! Can now optionally include the reserved word **entity** after the reserved word **end** in the entity declaration

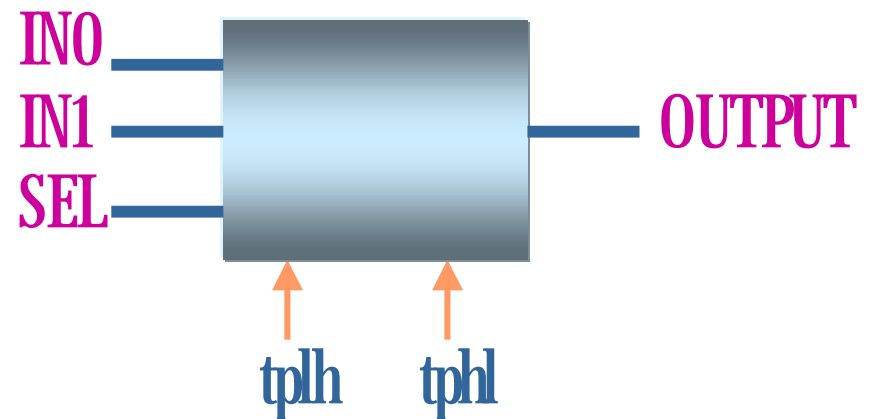
Entity declaration example

```
entity MUX is
  port ( IN0, IN1, SEL : in  bit;
        OUTPUT : out bit );
end MUX;
```



Propagation delay time

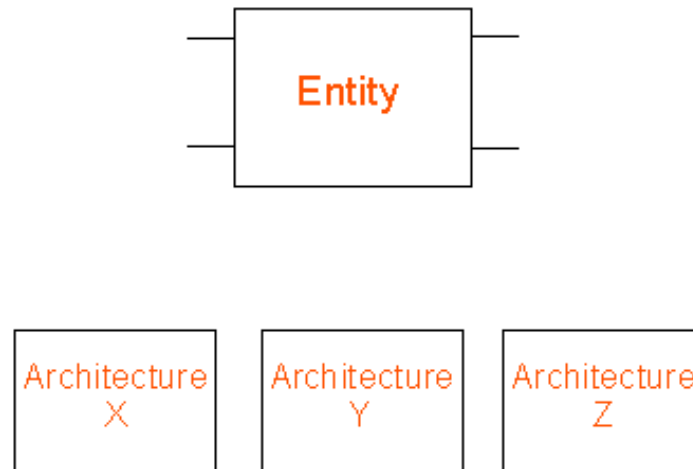
```
entity MUX is
  generic( TPLH : time:= 3 ns;
          TPHL : time:= 5 ns );
  port ( IN0, IN1, SEL : in  bit;
        OUTPUT : out bit );
end entity MUX;
```



VHDL 93

Architecture Design Unit

- Structural or behavioural description
- Describes an implementation of an entity
- Must be associated with a specific entity
- Single entity can have many architectures



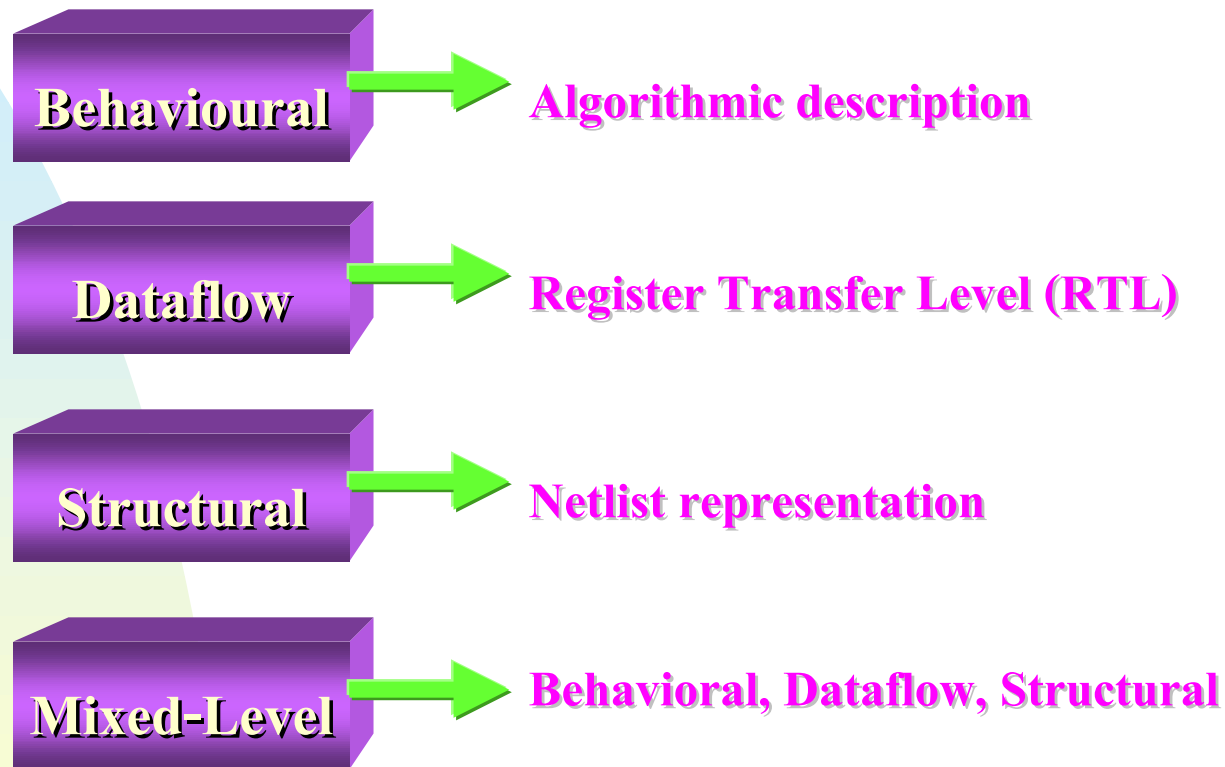
Architecture Declaration

Syntax:

```
architecture architecture_name of entity_name is
    declarations
begin
    concurrent_statements
end architecture_name;
```

VHDL 93! Can now optionally include the reserved word `architecture` after the reserved word `end` in the entity declaration

Architecture styles



Behavioural style architecture

```
architecture BEHAVIOURAL of MUX is
begin
    process (IN0, IN1, SEL)
    begin
        if (SEL = '0') then
            OUTPUT <= IN0;
        else
            OUTPUT <= IN1;
        end if;
    end process;
end BEHAVIOURAL;
```

Dataflow style architecture

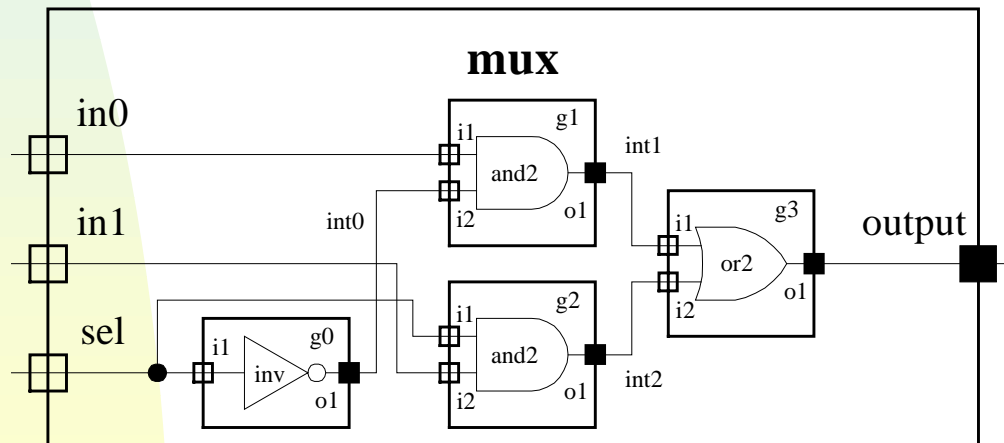
architecture DATAFLOW of MUX is

begin

$OUTPUT \leq ((\text{not SEL}) \text{ and } IN0) \text{ or } (\text{SEL} \text{ and } IN1);$

end DATAFLOW;

$$\text{output} = (\overline{\text{sel}} \cdot \text{in0}) + (\text{sel} \cdot \text{in1})$$



Structural style architecture

Component declarations

```
architecture STRUCTURAL of MUX is
    component INV
        port (I1 : in bit, O1 : out bit);
    end component;
    component AND2
        port (I1, I2: in bit; O1 : out bit);
    end component;
    component OR2
        port (I1, I2: in bit; O1 : out bit);
    end component;
    signal INT0, INT1, INT2: bit;
begin
    G0: INV : port map ( I1 => SEL, O1 => INT0);
    G1: AND2: port map ( I1 => IN0, I2=> INT0, O1 => INT1);
    G2: AND2: port map ( I1 => SEL, I2=> IN1, O1 => INT2);
    G3: OR2  port map ( I1 => INT1, I2=> INT2, O1 => OUTPUT);
end STRUCTURAL;
```

Component Declarations

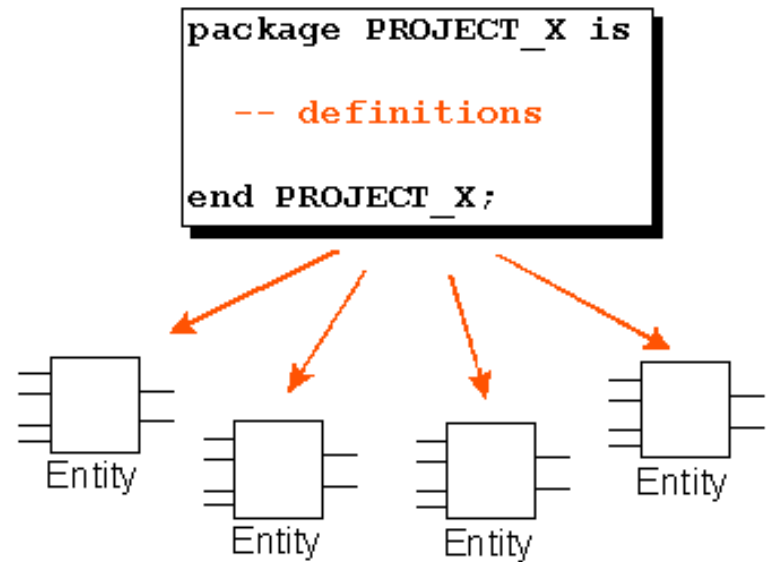
- Component instantiated must be declared within architecture
- Not instantiating the entity, but a socket...

```
architecture STRUCTURAL of MUX is
    component INV
        port (I1 : in bit; O1 : out bit);
    end component;
    component AND2
        port (I1, I2 : in bit; O1 : out bit);
    end component;
    component OR2
        port (I1, I2 : in bit; O1 : out bit);
    end component;
    signal INT0, INT1, INT2 : bit;
begin
    G0: INV : port map ( I1 => SEL, O1 => INT0);
    G1: AND2: port map ( I1 => IN0, I2 => INT0, O1 => INT1);
    G2: AND2: port map ( I1 => SEL, I2 => IN1, O1 => INT2);
    G3: OR2 : port map ( I1 => INT1, I2 => INT2, O1 => OUTPUT);
end STRUCTURAL;
```

Beware !

Component declaration before “begin” of architecture

- **Package declaration**
 - ◆ Subprogram declarations
 - ◆ Type declarations
 - ◆ Component declarations
 - ◆ Deferred constant declaration
- **Package Body**
 - ◆ Subprogram body
 - ◆ Deferred constant value



Packages declaration

Syntax:

```
package package_name is
    [exported_subprogram_declarations]
    [exported_constant_declarations]
    [exported_components]
    [exported_type_declarations]
    [attribut_declarations]
    [attribut_specifications]
end package_name;
```

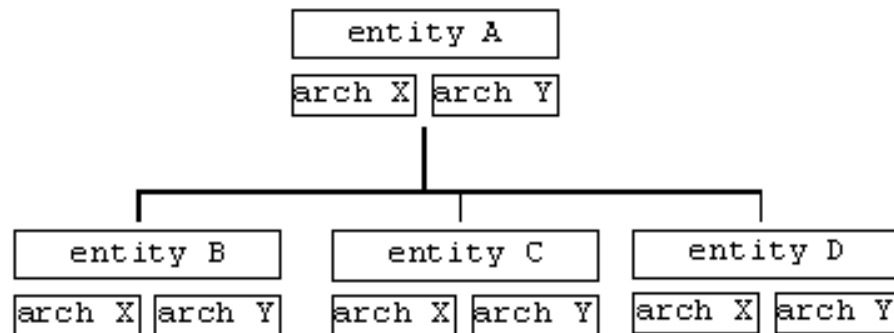
```
package body package_name is
    [exported_subprogram_declarations]
    [internal_subprogram_declarations]
    [internal_subprogram_bodies]
    [internal_type_declarations]
end package_name;
```

Packages declaration example

```
package MY_PACKAGE is
    type MY_TYPE1 is ...
    type MY_TYPE2 is ...
    function MY_MEAN (A, B, C: real) return real;
    procedure MY_PROC1 ( ...);
    ...
end MY_PACKAGE;
package body MY_PACKAGE is
    ...
    function MY_MEAN (A, B, C : real) return real is
    begin
        return(A + B + C)/ 3;
    end MY_MEAN;
    ...
end MY_PACKAGE;
```

Configuration Design Unit

- Select a particular architecture of an entity from library



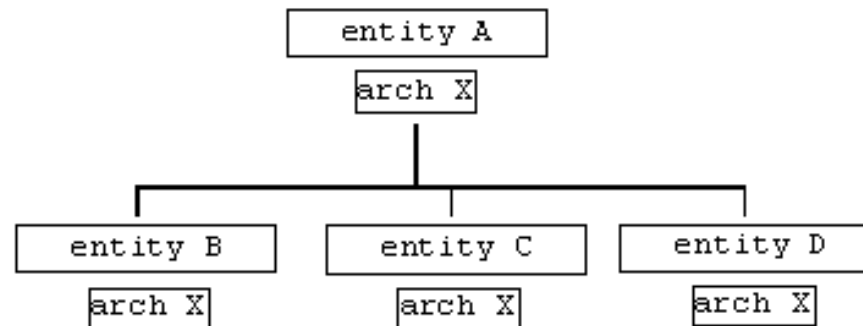
Configuration is like a parts list:

entity	arch
A	Y
B	X
C	X
D	Y

Configuration declaration

Syntax:

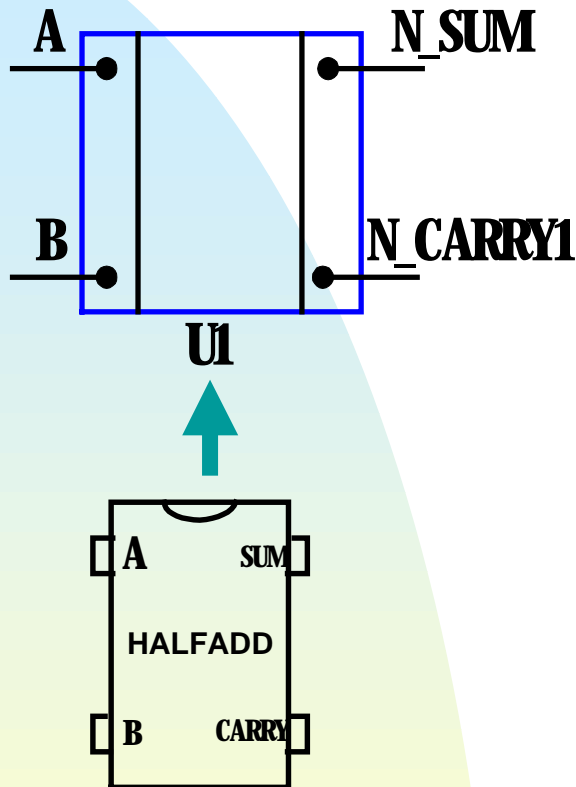
```
configuration configuration_name of entity_name is
    for architecture_name
end configuration_name;
```



Default configuration:

```
configuration CFG_A of A is
    for X
    end for;
end CFG_A;
```

Configuration Example



```
configuration CFG_HALFADD of HALFADD is
    for STRUCTURAL
        end for;
    end CFG_HALFADD;
```

- Selection of entities and architectures
 - ◆ Select architecture of top level entity
 - ◆ Select entity/architecture pairs for component instances
- Consider as “parts list” for hierarchy
- Default configuration
 - ◆ If names of entities & components match
 - ◆ Last analysed architecture of enties
- Non-default configuration
 - ◆ Map specifiec entities, architectures

Process

- Section containing sequential statements
- Exist inside an architecture
- Multiple processes interact concurrently

Concurrent statement

Sequential statements

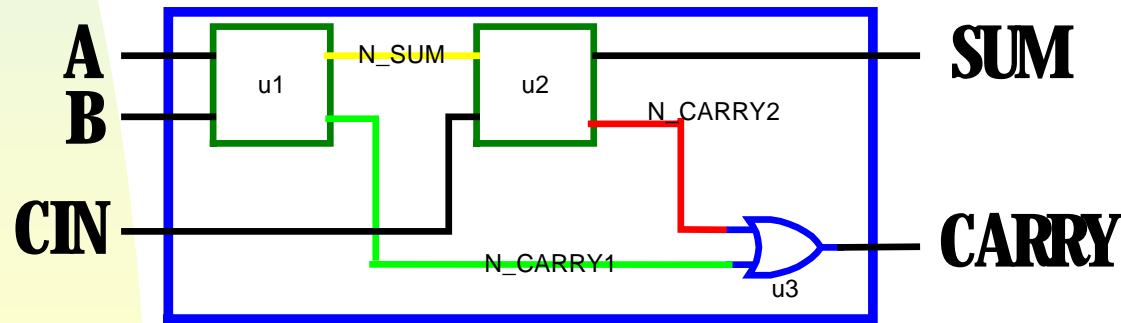
```
entity AND_OR is
    port(A, B : in bit;
          Z_OR, Z_AND : out bit);
end AND_OR;

architecture BEHAVE of AND_OR is
begin
    AND_OR_FUNC: process(A, B)
    begin
        if (A='1' or B='1') then
            Z_OR <= '1';
        else
            Z_OR <= '0';
        end if;

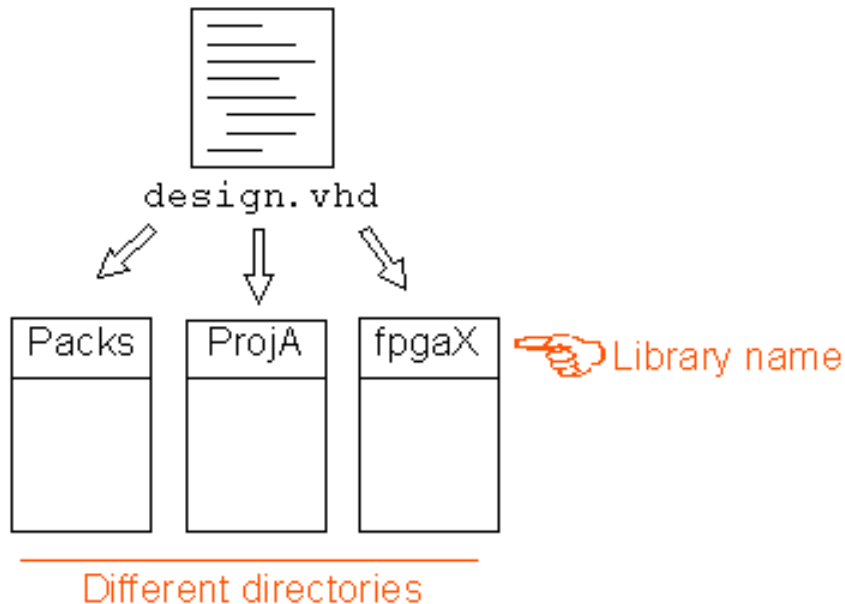
        if (A='1' and B='1') then
            Z_AND <= '1';
        else
            Z_AND <= '0';
        end if;
    end process AND_OR_FUNC;
end BEHAVE;
```

Design Hierarchy

```
entity FULLADD is
port (A, B, CIN : in bit;
      SUM, CARRY : out bit);
end FULLADD;
architecture STRUC of HALFADD is
signal N_SUM, N_CARRY1, N_CARRY2 : bit;
-- other declarations
begin
    U1 : HALFADD port map (A, B, N_SUM, N_CARRY1);
    U2 : HALFADD port map (N_SUM, CIN, SUM, N_CARRY2);
    U3 : ORGATE port map (N_CARRY2, N_CARRY1, CARRY);
end STRUC;
```



Library



Library name

```
Library IEEE;  
use IEEE.Std_Logic_1164.all;  
  
entity INCOMP_IF is  
port ( EN, D : in std_ulogic;  
       Q      : out std_ulogic );
```

- A collection of compiled design units
 - ◆ Entity, Architecture, Package, Package body, Configuration
- Physically exists as a directory
- Referred to by Library name
- Simulator startup file specifies mapping
 - ◆ Library name -> directory name
- Compile into “WORK”
- WORK mapped to specific library name

Comments

Comments

- This is a comment
- Each line must begin with --
- Comments end with a new line

```
Library IEEE;  
use IEEE.Std_Logic_1164.all;  
  
entity FULLADD is  
port (A, B, CIN : in std_logic;  
      SUM, CARRY : out std_logic);  
end FULLADD;
```

Signals and Datatypes

Aim and Topics

- Introduction to :
 - ◆ Type concept
 - ◆ Standard types
 - ◆ Scalar and Array Literal
 - ◆ IEEE Standard Logic

Concept of a Type

- Type must be defined when signal is declared
- Either in
 - ◆ Port section of an entity
 - ◆ Architecture, in a signal declaration
- Types must match up!

```
entity HALFADD is
port (A,B      : in bit;
      SUM, CARRY : out bit);
end HALFADD;
```

```
SUM <= A xor B;
```



Types must match...

```
entity FULLADD is
port (A,B,CIN : in bit;
      SUM, CARRY : out bit);
end FULLADD;
architecture STRUCT of FULLADD is
  signal N_SUM : bit;
  -- other declarations
begin
  -- Code
end STRUCT;
```



Standard Data Types

```
package STANDARD is
  type boolean is (FALSE, TRUE);
  type bit is ('0', '1');
  type character is (-- asc ii set);
  type integer is range
  type real is range
  -- bit_vector, string, time
end STANDARD;
```

```
FALSE
TRUE
```

Boolean

```
10 ns
200 fs
2.5 ps
```

Time

```
'0'
'1'
```

bit

```
"0000"
"111"
"010101010"
```

bit_vector

```
'a' '1'
'A'
'Z' '0'
```

character

```
"Hello"
"VHDL"
"Esperanto"
```

string

```
256 -45
0
1 137
```

integer

```
1.02
1.0
-37.4
```

real

Standard Data Types Provided within VHDL

Type	Range of values	Examples declaration
integer	-2,147,483,647 to +2,147,483,647	signal index : integer := 0;
real	- 1.0E +38 to +1.0E +38	variable val : real := 1.0;
boolean	(TURE, FALSE)	variable test : boolean := TRUE;
character	defined in package STANDARD	variable term : character := '@';
bit	0, 1	signal in1 : bit := '0';
bit_vector	array with each element of the type	variable pc : bit_vector(31 downto 0);
time	fs, ps, ns, us, ms, sec, min, hr	variable delay : time := 25 ns;
string	array with each element of type character	variable name : string(1 to 4) := "tmec";
natural	0 to the maximum integer value in the implementation	variable index : natural := 0;
positive	1 to the maximum integer value in the implementation	variable index : natural := 0;

Scalars and Array Literals

Scalar Type

character

bit

std_logic

boolean

real

integer

time

Array Type

string

bit_vector

std_logic_vector

The Problem With Type Bit

- Only has values '1' and '0'
 - ◆ Default initialisation to '0'
- Simulation and synthesis require other values
 - ◆ Unknown
 - ◆ High impedance
 - ◆ Don't care
 - ◆ Different signal strengths

MVL Systems

- Problem with type BIT : use of Multi-Valued Logic System (MVL)
 - ◆ Enumerated type defining unknown, high-impedance, etc
- Until early 1992, no standard
- Standard now set
 - ◆ Nine state system defined and agreed by IEEE
 - ◆ Standard IEEE 1164

IEEE Standard Logic types

```

type std_ulogic is (
    'U',      uninitialized
    'X',      unknown
    '0',      logic 0      | Strong drive
    '1',      logic 1
    'Z',      High impedance
    'W',      unknown
    'L',      logic 0      | Weak drive
    'H',      logic 1      | (not often used)
    '-') ;
    Don't care

```

- In package “Std_Logic_1164”
- “std_logic” has same values as “std_ulogic”

std_logic
std_logic_vector

std_ulogic
std_ulogic_vector

Rules for Using Std_Logic and Std_Ulogic

std_ulogic_vector

array of std_ulogic



'1'	'0'	'1'	'0'
-----	-----	-----	-----

makes library visible



library IEEE;

use IEEE.Std_Logic_1164.all;

makes all contents of package visible



entity MVLS is

port (A,B, : in std_logic;

Z : out **std_ulogic** ;

end MVLS ;

std_logic_vector

array of std_logic



'1'	'0'	'1'	'0'
-----	-----	-----	-----

signal RES_Z : std_logic;

Z <= A;



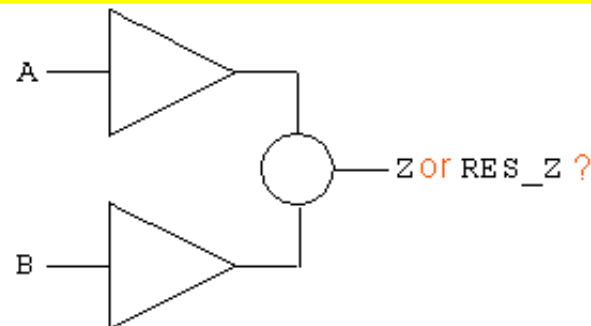
RES_Z <= A;



A <= RES_Z;



Cannot assign std_logic_vector to std_ulogic_vector



Z <= A;
Z <= B;



RES_Z <= A;
RES_Z <= B;



Object Declarations

Aim and Topics

- Introduction to :
 - ◆ Constant declaration
 - ◆ Signal declaration
 - ◆ Variable declaration

Constant Declaration

Syntax :

```
constant constant_name : type := value;
```

- Constant is name assigned to a fixed value
- You need only change the constant declaration in one place
- Constant makes a design more readable and makes updating code easy

Example

```
constant Vdd: real := - 45;  
constant CYCLE: time := 100ns ;  
constant PI: real := 3.14;  
constant FIVE: bit_vector := "0101" ;  
constant TEMP: std_logic_vector(8 to 11) := "0101" ;
```

Signal Declaration

Syntax :

```
signal signal_name : type ;
```

```
OR signal signal_name : type := initial_value;
```

- Signal connect design entities together and communicate changes in values between processes.
- Signal can be abstractions of physical wires, busses, or used to document wires in an actual circuit.
- Signal with a type must be declared before the signal is used.

Example

```
signal A_BUS, A_BUS, Z_BUS : bit_vector(3 downto 0);  
signal A_BIT, B_BIT, C_BIT, D_BIT : bit;  
signal BYTE : bit_vector(0 to 7);  
signal COUNT : integer range 1 to 50;  
signal GROUND: bit := '0';  
signal SYS_BUS : std_logic_vector(31 downto 0);
```


Signal Assignment

- Signal of same type
- Language defines
 - ◆ bit_vector (array of bit)
 - ◆ string (array of character)
- Define size when declaring signal/port

Legal declarations:

```
signal Z_BUS:bit_vector(3 downto 0);  
signal C_BUS:bit_vector(1 to 4);
```

```
signal Z_BUS:bit_vector(3 downto 0);  
signal C_BUS:bit_vector(1 to 4);
```

Legal:

```
Z_BUS(3 downto 2) <= "00";  
C_BUS(2 to 4) <= Z_BUS(3 downto 1);
```

Illegal declarations:

```
signal Z_BUS:bit_vector(0 downto 3);  
signal C_BUS:bit_vector(3 to 0);
```

Illegal:

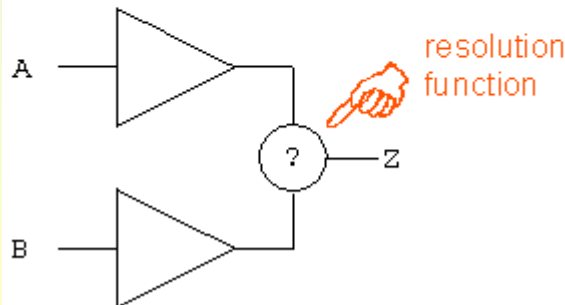
```
Z_BUS(0 to 1) <= "11";
```

Signals and Drivers

- Signal value changed with “Signal Assignment Statement”
- Signal driver
- Types must match
- Resolved signal

```
signal A,B,Z : bit;  
signal X_INT : integer;
```

```
Z <= A;  
Z <= B;
```



```
signal A,B,Z : bit;  
signal X_INT : integer;
```

signal assignment
statement

```
Z <= A;
```

Alias Declaration

Syntax :

```
alias alias_name : alias_type is object_name;
```

- An alias is an alternative name for an existing object (signal, variable or constant)
- It does not define a new object

Example

```
signal COUNT : bit_vector(7 downto 0);  
  alias SIGN_BIT : bit is COUNT(7);  
  alias LSB_BIT : bit is COUNT(0);  
  alias NIBBLE_L : bit_vector(3 downto 0) is COUNT(3 downto 0);  
  alias NIBBLE_H : bit_vector(3 downto 0) is COUNT(7 downto 4);
```

Variables

```
signal A, B, C, Y, Z : integer;  
begin  
  P1 : process (A, B, C)  
    variable M, N : integer;  
  begin  
    M := A;  
    N := B;  
    Z <= M + N;  
    M := C;  
    Y <= M + N;  
  end process P1;
```

- Declared within process
- Assignment immediately
 - ◆ Like “software”
- Only be used within process
 - ◆ Called its “scope”
- Retains value
- Assigns :
 - ◆ signal to variable
 - ◆ variable to signal

Variable Declaration

Syntax :

variable *variable_name* : *type*;

OR variable *variable_name* : *type* := *initial_value*;

- Variable is a name assigned to a changing value within a process
- Variable assignment occurs immediately in simulation
- Variable can be used as a temporary simulation value
- Variable must be declared before it is used

Example

```
variable INDEX : integer range 1 to 50;
```

```
variable MEMORY : bit_vector(0 to 7);
```

```
variable X, Y : integer;
```

```
variable CYCLE_TIME : TIME range 10 ns to 50 ns := 10 ns;
```

Variables Versus Signals

```

signal A, B, C, Y, Z : integer;
begin
  P1 : process (A, B, C)
    variable M, N : integer;
  begin
    M := A;
    N := B;
    Z <= M + N;
    M := C;
    Y <= M + N;
  end process P1;

```

```

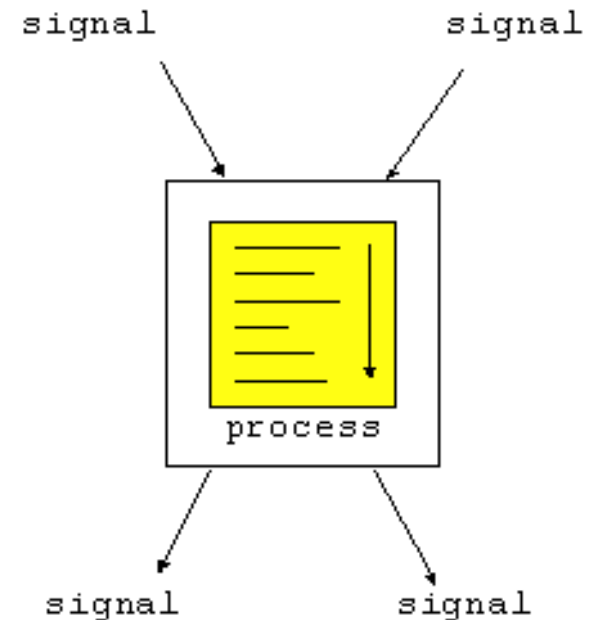
signal A, B, C, Y, Z : integer;
signal M, N : integer;
begin
  P1 : process (A, B, C, M, N)
  begin
    M <= A;
    N <= B;
    Z <= M + N;
    M <= C;
    Y <= M + N;
  end process P1;

```



Variables Usage Model

- Variables used for algorithms
 - ◆ Assign signals to variables
 - ◆ Perform algorithm
 - ◆ Assign variables to signals
- Variable cannot be accessed outside of its process



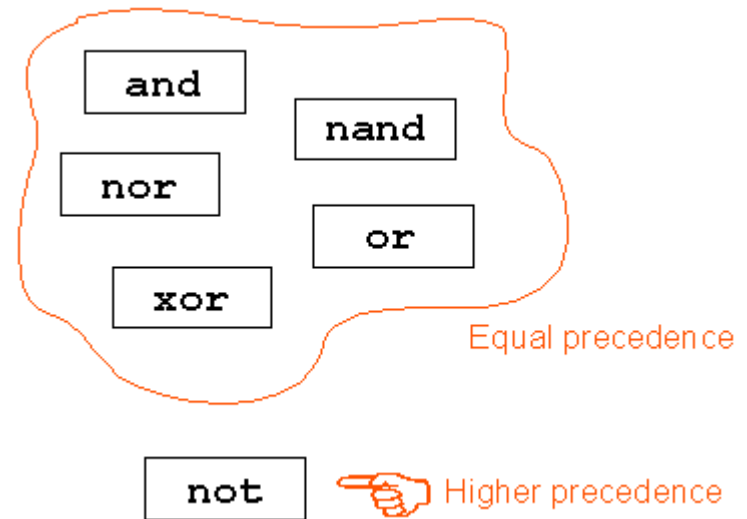
VHDL Operators

Aim and Topics

- Introduction to :
 - ◆ Logical Operators
 - ◆ Relational Operators
 - ◆ Concatenation Operator
 - ◆ Arithmetic Operators

Logical Operators

- and, or, nand, nor, xor (equal precedence)
- not (higher precedence)
- Pre-defined for
 - ◆ bit
 - ◆ bit_vector
 - ◆ std_ulogic, std_logic
 - ◆ std_ulogic_vector, std_logic_vector
- xnor



VHDL 93! Shift operators : sll (shift left logical), srl (shift right logical),
 sla (shift left arithmetic), sra (shift right arithmetic),
 rol (rotate left logical), ror (rotate right logical)

Logical Operators : Examples

```
entity AND2 is
port ( A_BIT, B_BIT : in bit;
      Z_BIT          : out bit);
end AND2;

architecture RTL of AND2 is
begin
  Z_BIT <= A_BIT and B_BIT;
end RTL;
```



Logical Operators on Arrays

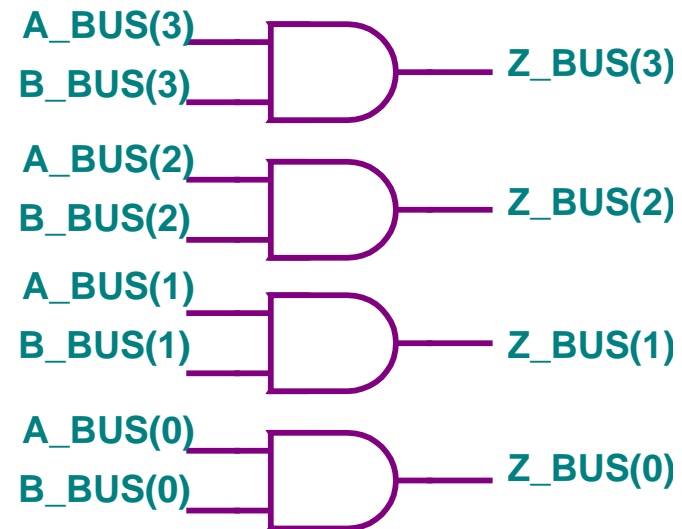
- Operands same length and type
- Operations on matching elements

```
signal A_BUS, B_BUS, Z_BUS : std_logic_vector(3 downto 0);
```

```
Z_BUS <= A_BUS and B_BUS;
```

↓ Equivalent to

```
Z_BUS(3) <= A_BUS(3) and B_BUS(3);
Z_BUS(2) <= A_BUS(2) and B_BUS(2);
Z_BUS(1) <= A_BUS(1) and B_BUS(1);
Z_BUS(0) <= A_BUS(0) and B_BUS(0);
```



Relational Operators

- Result is boolean (TRUE or FALSE)
- Often used with if - then - else

<

less than

<=

less than
or equal to

>

greater than

>=

greater than
or equal to

=

equal to

/=

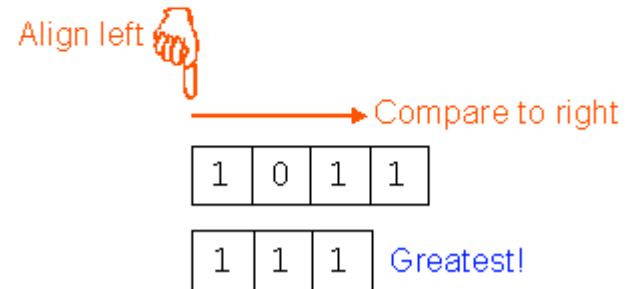
not equal
to

Relational Operations : Rules

- Operands must be of the same type
- Arrays
 - ◆ May be of different lengths
 - ◆ Aligned left and compared
 - ◆ Can lead to unusual result !
- No numerical meaning

Operands of same type

arrays of different lengths:



No numerical meaning!

1	1	1
---	---	---

Not 7... but a set of three '1's !

Concatenation Operator

- Ampersand (&) means concatenation in VHDL

```
signal Z_BUS : bit_vector( 3 downto 0 );  
signal A, B, C, D : bit;  
signal BYTE : bit_vector( 7 downto 0 );  
signal A_BUS : bit_vector( 3 downto 0 );  
signal B_BUS : bit_vector( 3 downto 0 );
```

```
Z_BUS <= A & B & C & D;
```

```
BYTE <= A_BUS & B_BUS;
```

concatenation



Arithmetic Operators

- Pre-defined for
 - ◆ integer
 - ◆ real (except mod and rem)
 - ◆ Physical types (e.g. time)
- Not defined for
 - ◆ bit_vector
 - ◆ std_ulogic_vector
 - ◆ std_logic_vector
- Generally, operands must be of the same type

+

addition

-

subtraction

*

multiplication

/

division

**

exponential

abs

absolute
value

mod

modulus

rem

remainder

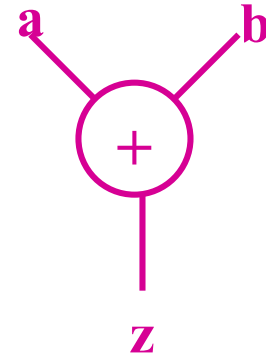


```
signal A, B, Z : integer;
...
  Z <= A + B;
```

Adder example

```
entity ADDER is
  port (A, B : in integer range 0 to 7;
        Z : out integer range 0 to 15);
end ADDER;

architecture ARITHMETIC of ADDER is
begin
  Z <= A + B;
end ARITHMETIC;
```



Arithmetic of Time

- Testbenches, cell delays
- Multiplication/ division
 - ◆ Multiply / divide by integer/real
 - ◆ returns type time

```
signal CLK : std_logic;  
constant PERIOD : time := 50 ns;
```

```
wait for 5 * PERIOD;
```

waits for 250 ns

```
wait for PERIOD * 1.5;
```

waits for 75 ns

```
CLK <= not CLK after PERIOD/2;
```

assignment after 25 ns

Sequential Statments

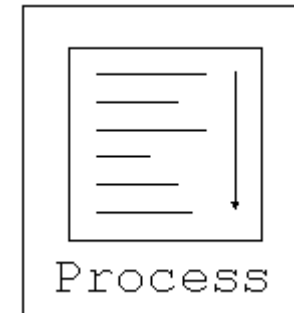
Aim and Topics

- To introduce some of the most commonly used concurrent and sequential statements
- The process statement
- Sequential statements
 - ◆ IF Statements
 - ◆ CASE Statement
 - ◆ LOOP Statement
 - ◆ WAIT Statement

Process Statement

Syntax :

```
optional_label: process (optional_sensitivity_list)
    subprogram_declarations
    type_declarations
    constant_declarations
    variable_declarations
    other_declarations
begin
    sequential_statements
end process optional_label;
```



Process Statement example

label

Sensitivity list

```
MUX : process (A, B, SEL)
```

```
begin
```

```
    if (SEL = '1') then
```

```
        Z <= A;
```

```
    else
```

```
        Z <= B;
```

```
    end if;
```

```
end process MUX;
```

Sequential statements

Processes within VHDL Code

architecture A of E is

begin

-- concurrent statements

P1 : process

begin

-- sequential statements

end process P1;

-- concurrent statements

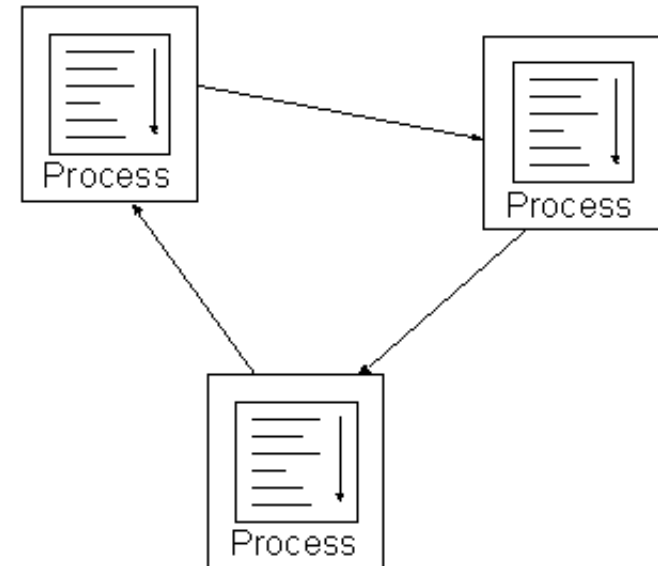
P2 : process

begin

-- sequential statements

end process P2;

end A;



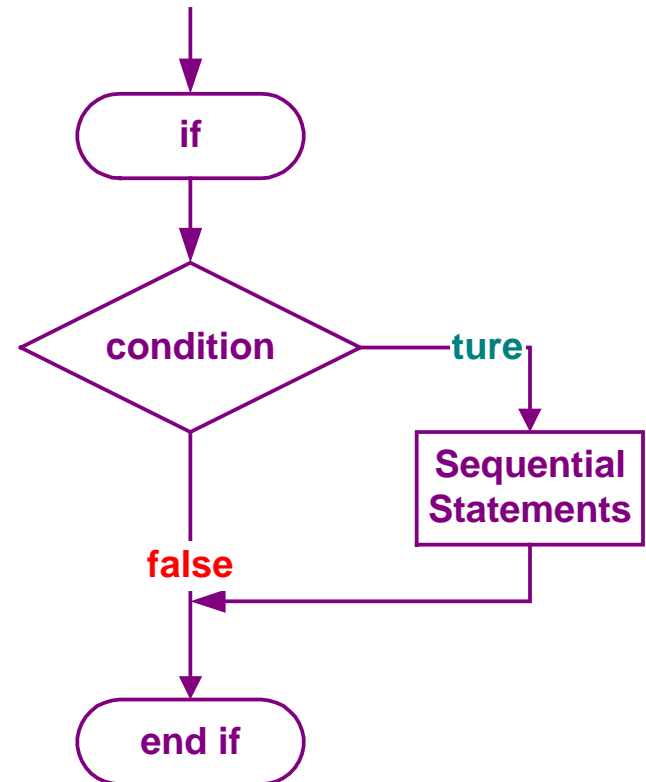
IF Statement

Syntax :

```
if condition then
    sequential_statements
    ...
end if;
```

Example

```
if (A = '1') then
    COUNT := COUNT + 1;
end if;
```



IF-ELSE Statement

Syntax :

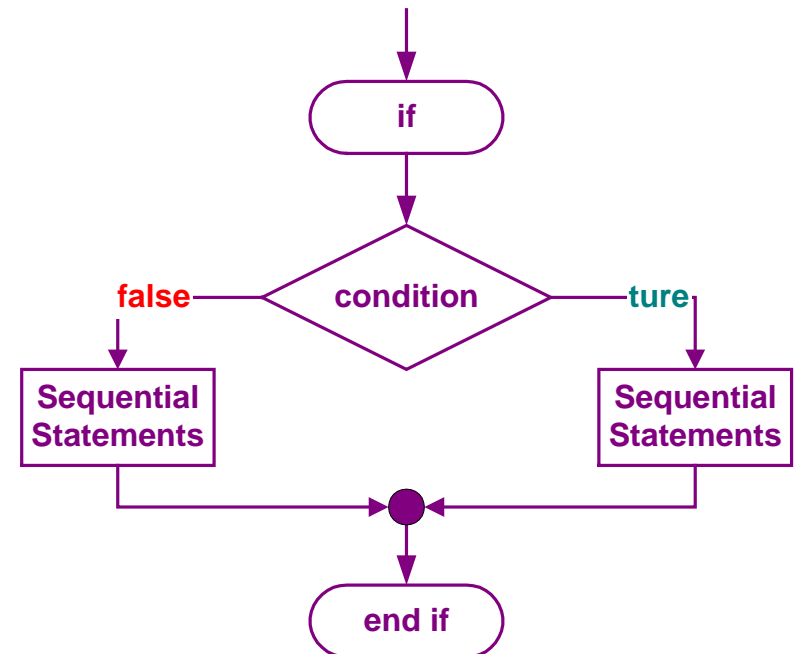
```

if condition then
    sequential_statements
    ...
else
    sequential_statements
end if;
    
```

Example

```

if (A = '1') then
    COUNT := COUNT + 1;
else
    COUNT := COUNT - 1;
end if;
    
```



IF-ELSIF Statement

Syntax :

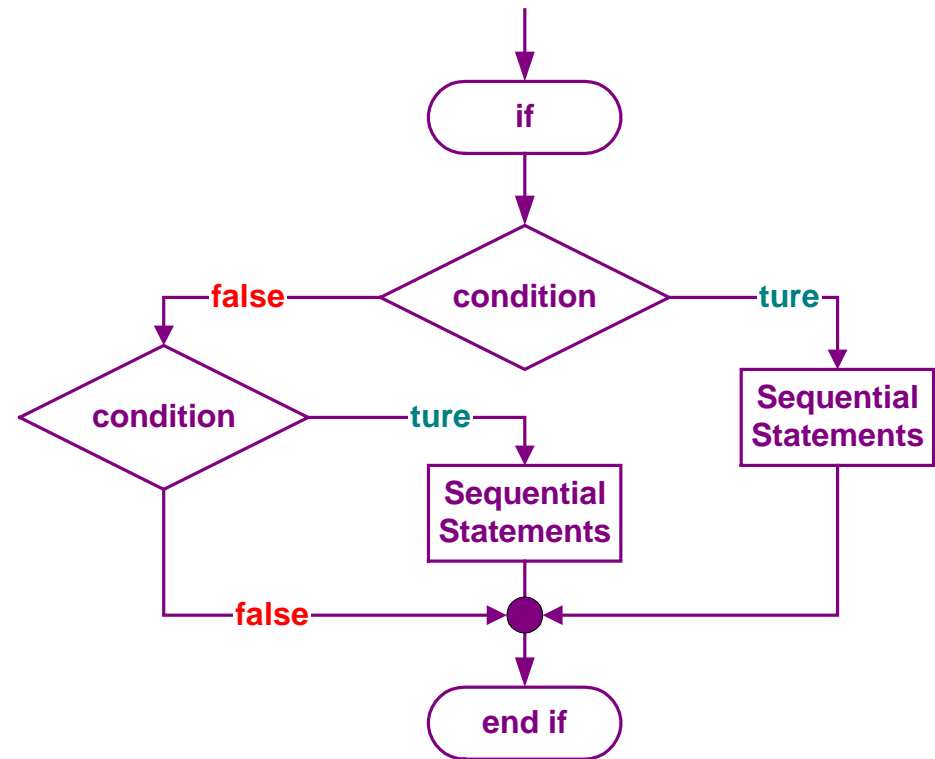
```

if condition 1 then
    sequential_statements
elsif condition 2 then
    sequential_statements
end if;
  
```

Example

```

if (A = '1') then
    COUNT := COUNT + 1;
elsif (A = '0') then
    COUNT := COUNT - 1;
end if;
  
```



IF Example

```

Library IEEE;
use IEEE.Std_Logic_1164.all;

entity IF_EXAMPLE is
port ( A, B, C, D : in  std_logic_vector( 3 downto 0);
      SEL      : in  std_logic_vector( 1 downto 0);
      Z        : out std_logic_vector( 3 downto 0) );
end IF_EXAMPLE;

architecture IF_STATE of IF_EXAMPLE is
begin
  process (A, B, C, D, SEL)
  begin
    if (SEL = "00") then
      Z <= A;
    elsif (SEL = "01") then
      Z <= B;
    elsif (SEL = "10") then
      Z <= C;
    else
      Z <= D;
    end if;
  end process;
end IF_STATE;

```

CASE Statement

Syntax :

```
case expression is
  when choice => sequential_statements ;
  when choice => sequential_statements ;
  ...
end case;
```

Example

```
case SEL is
  when "00"      => Z <= A ;
  when "01"      => Z <= B ;
  when "10"      => Z <= C ;
  when others    => Z <= D ;
end case ;
```

- No choices may overlap
- All choices must be specified
 - ◆ specifically or with "others"

Defining Ranges

```

Library IEEE;
use IEEE.Std_Logic_1164.all;

entity CASE_EXAMPLE is
port ( A, B, C, SEL : in integer range 0 to 15;
      Z : out integer range 0 to 15 );
end CASE_EXAMPLE;

architecture CASE_STATE of CASE_EXAMPLE is

begin
  process (A, B, C, SEL)
  begin
    case SEL is
      when 0 to 4 => Z <= B;
      when 5 to 10 => Z <= C;
      when others => Z <= A;
    end process;
  end CASE_STATE;

```

Beware !

- Array do not have a discrete sequence of values associated with them

case SEL is		
when "11"	=>	<input checked="" type="checkbox"/>
when "00" to "10"	=>	<input type="checkbox"/>
when others	=>	<input checked="" type="checkbox"/>

LOOP Statements

- Available in a process
 - ◆ For Loop Statement
 - ◆ While Loop Statement
- Label is optional
- Use range attributes
 - ◆ 'high
 - ◆ 'low
 - ◆ 'range
- For loop identifier
 - ◆ not declared
 - ◆ Can't be altered
 - ◆ not visible outside loop

For Loop Statement

Syntax :

```
optional_label:for parameter in range loop  
    sequential_statements  
end loop optional_loop;
```

Example

```
LOOP1 : for INDEX in 0 to 7 loop  
    DOUT(INDEX) <= DIN(INDEX);  
end loop LOOP1;
```

- Iterated around a loop
- Loop variable gets values in the range
- No need to declare loop variable
- It is illegal to make an assignment to the loop variable

For Loop Example

```

Library IEEE;
use IEEE.Std_Logic_1164.all;

entity EX is
port ( A, B, C : in  std_logic_vector( 4 downto 0);
      Q      : out std_logic_vector( 4 downto 0) );
end EX;

architecture FOR_LOOP of EX is

begin
  process (A, B, C)
  begin
    for I in 0 to 4 loop
      if A(I) = '1' then
        Q(I) <= B(I);
      else
        Q(I) <= C(I);
      end if;
    end loop;
  end process;
end;

```

While Loop Statement

Syntax :

```
optional_label: while condition loop
    sequential_statements
end loop optional_label;
```

Example

```
P1 : process (A)
    variable INDEX : integer := 0;
begin
LOOP1 : while INDEX < 8 loop
    DOUT(INDEX) <= DIN(INDEX);
    INDEX := INDEX + 1 ;
end loop LOOP1;
end process P1;
```

- Available in a process
- **Not** generally synthesizable
- Useful in testbenches
 - ◆ Reading from file . . .
 - ◆ Generation of clock for specific time

WAIT Statements

- Suspends execution of process
- Execution continues when statement satisfied
- Four formats . . .

wait for specific_time;

Wait for :
A specific time

wait on signal_list;

An event on signals (equivalent to a sensitivity list)

wait until condition;

A condition to occur (requires event)

wait;

Indefinitely (never to be reactivated)

Wait Examples

```
Library IEEE;
use IEEE.Std Logic_1164.all;
entity D_FLOP is
port ( D, CLK : in  std_ulogic;
      Q      : out std_ulogic );
end D_FLOP;
```

architecture A of D_FLOP is

begin

process

begin

wait until CLK'event and CLK = '1';
Q <= D;

end process;

end A;

No sensitivity list

STIMULUS : process

begin

SEL <= '0';

BUS_B <= "0000";

BUS_A <= "1111";

wait for 10ns;

SEL <= '1';

wait for 10ns;

-- ETC, ETC

end process STIMULUS;

Concurrent Statements

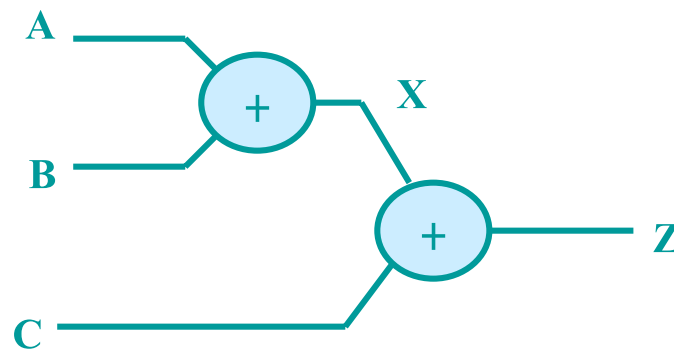
Aim and Topics

- To introduce some of the most commonly used concurrent
 - ◆ Concurrent signal assignment
 - ◆ Condition signal assignment
 - ◆ Selected signal assignment
 - ◆ BLOCK statement

Concurrent Assignment Statements

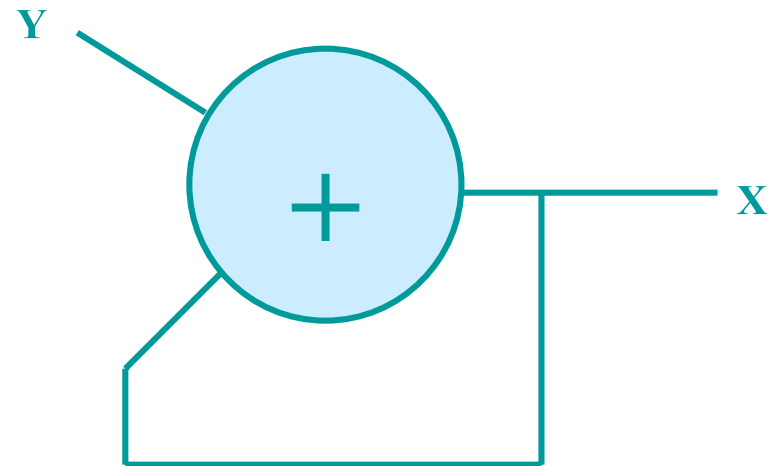
$$\begin{aligned} X &\leq A + B; \\ Z &\leq C + X; \end{aligned}$$
$$\begin{aligned} Z &\leq C + X; \\ X &\leq A + B; \end{aligned}$$

- Concurrent
- Order independent
- What you write is what you get...



Don't Create Feedback Loops !

$X \leq X + Y;$



Concurrent Versus Sequential

architecture A of E is

begin

-- concurrent statements

P1: process

begin

-- sequential statements

end process P1;

-- concurrent statements

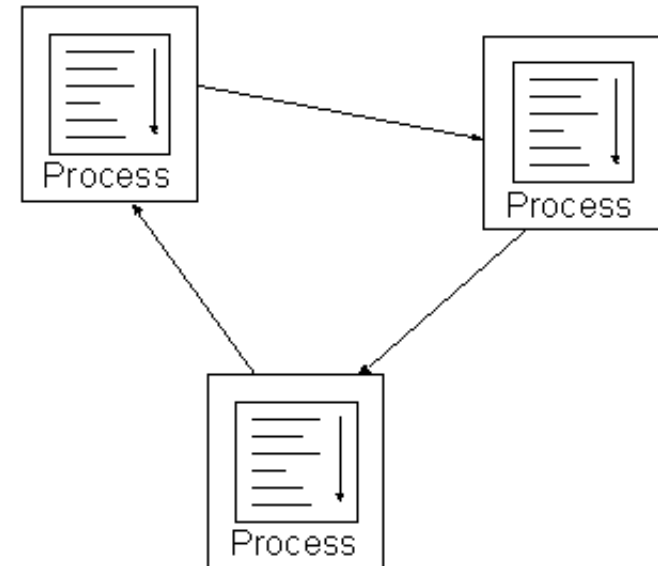
P2: process

begin

-- sequential statements

end process P2;

end A;



Concurrent Signal Assignments

Concurrent Form

```
architecture FORM_1 of V_VAR is
begin
    OUTPUT_Q <= DATA_IN;
end V_VAR;
```

=

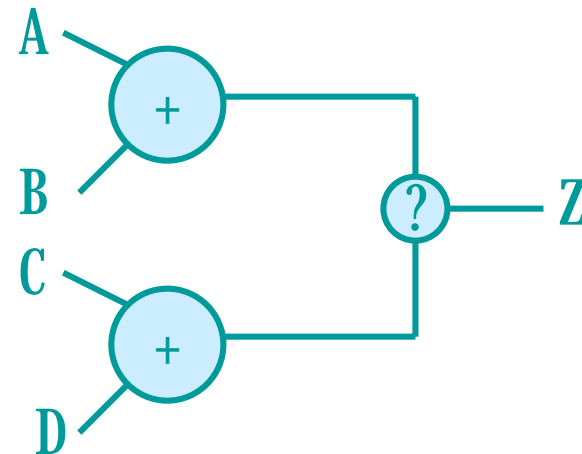
Sequential Form

```
architecture FORM_1 of V_VAR is
begin
    process (DATA_IN)
    begin
        OUTPUT_Q <= DATA_IN;
    end process;
end V_VAR;
```

Multiple Concurrent Assignments

- Signals are “wired together”
- VHDL requires a resolution function

```
Z <= A + B;  
Z <= C + D;
```

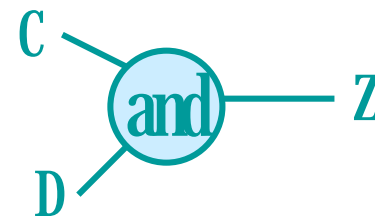
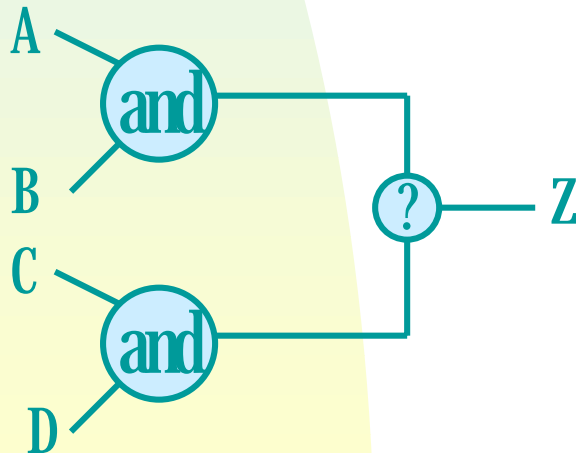


Multiple Assignments in a Process

```
architecture CONCURRENT of MULTIPLE is
  signal A, B, C, D : std_ulogic;
  signal Z : std_logic;
begin
  Z <= A and B;
  Z <= C and D;
end CONCURRENT;
```

≠

```
architecture SEQUENTIAL of MULTIPLE is
  signal Z, A, B, C, D : std_ulogic;
begin
  process (A, B, C, D)
  begin
    Z <= A and B;
    Z <= C and D;
  end process;
end SEQUENTIAL;
```



Multiple Assignments : Example

```
process (A, B, SEL)
begin
  if (SEL = '1') then
    Z <= A;
  else
    Z <= B;
  end if;
end process;
```

=

```
process (A, B, SEL)
begin
  Z <= B;
  if (SEL = '1') then
    Z <= A;
  end if;
end process;
```

- Last assignment takes effect
- After process suspends
- Code is equivalent

Conditional Signal Assignment

```
entity BRANCH is
port ( A, B, C, X : in integer range 0 to 7;
      Z           : out integer range 0 to 0 );
end BRANCH;
```

- Concurrent version of “If”
- Only one target
- Must have an unconditional else

```
architecture USE_IF of BRANCH is
begin
  process (A, B, C, X)
  begin
    if (X > 5) then
      Z <= A;
    elsif (X < 5) then
      Z <= B;
    else
      Z <= C;
    end if;
  end process;
end USE_IF;
```

```
architecture USE_WHEN of BRANCH is
begin
  Z <= A when X > 5 else
    B when X < 5 else
    C;
end USE_WHEN;
```

Selected Signal Assignment

```
entity BRANCH is
port ( A, B, C, X: in integer range 0 to 7;
      Z      : out integer range 0 to 0 );
end BRANCH;
```

```
architecture USE_IF of BRANCH is
begin
  process (A, B, C, X)
  begin
    case X is
      when 0 to 4 =>
        Z <= B;
      when 5 =>
        Z <= C;
      when OTHERS =>
        Z <= A;
    end case;
  end process;
end USE_IF;
```

```
architecture USE_WITH of BRANCH is
begin
  with X select
    Z <= B when 0 to 4,
        C when 5,
        A when OTHERS;
end USE_WITH;
```

- Concurrent version of “case”
- Only one target
- Rule as for case

Block/Guarded Statement

Syntax :

```

label: block (optional_guard_condition)
    declarations
    begin
        concurrent_statements
    end block label;
```

- Concurrent statement
- Without a **guard** condition a **block** is a grouping together of concurrent statements within an architecture

Example

```

B1: block (ENB = '1')
    begin
        DOUT <= guarded DIN;
    end block B1;
```

Level-Sensitive Latch

```

B2: block (CLK'event and CLK = '1')
    begin
        DOUT <= guarded DIN;
    end block B2;
```

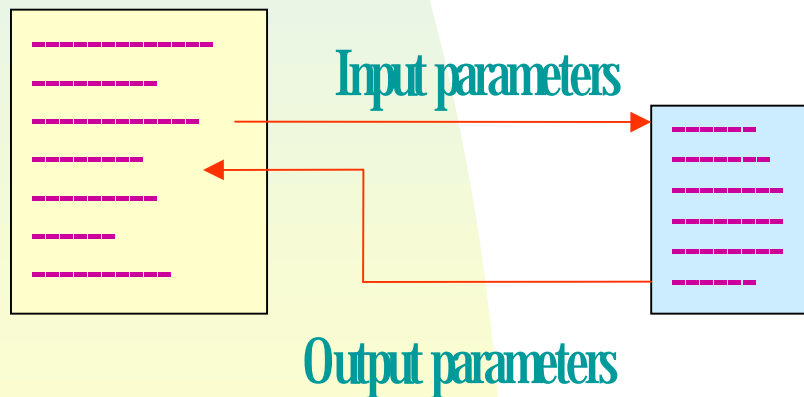
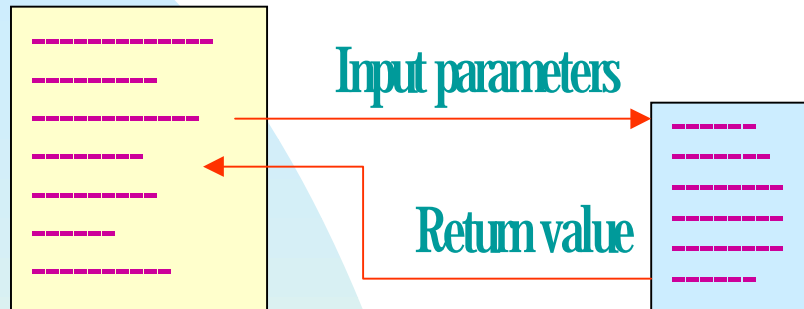
Edge-Sensitive Flip-Flop

Subprograms

Aim and Topics

- Explain these two types of subprogram
- Subprogram concepts
- Functions
- Procedure
- Resolution functions

Functions and Procedures



- “Subprograms”
 - ◆ Sequential statements
 - ✦ execute in sequence like “software”
- Function
 - ◆ One return value
- Procedure
 - ◆ Zero or more return value

Function declaration

Syntax :

```
function function_name (parameter_list) return type is
    types, constants, other functions, other declarations
begin
    sequential_statements
end function_name;
```

```
entity ... is
end ...
architecture ... of ... is
```

```
begin
```

```
process
```

```
begin
```

```
end process;
```

```
end ... ;
```

Function Declaration

Function Calls

Function Declaration

Function Calls

Function Examples

Function name

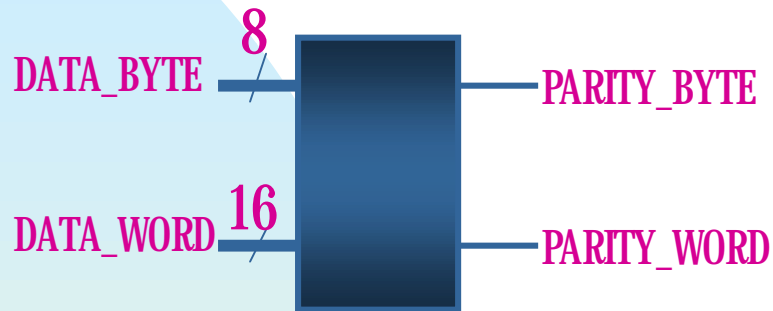
Parameter list

```
function PARITY_FUNC ( X : std_ulogic_vector)
    return std_ulogic is
    variable TMP : std_ulogic;
begin
    TMP := '0';
    for J in X'range loop
        TMP := TMP xor X(J);
    end loop;

    return TMP;
end PARITY_FUNC;
```

Returned value

Function Call



```
function PARITY_FUNC ( X : std_ulogic_vector)
return std_ulogic is
variable TMP : std_ulogic;
begin
    TMP := '0';
    for J in X'range loop
        TMP := TMP xor X(J);
    end loop;
    return TMP;
end PARITY_FUNC;
```

```
entity PARITY is
    port (
        DATA_BYTE : in std_ulogic_vector(7 downto 0);
        DATA_WORD : in std_ulogic_vector(15 downto 0);
        PARITY_BYTE : out std_ulogic;
        PARITY_WORD : out std_ulogic
    );
end PARITY;

architecture FUNC of PARITY is
    -- function declaration
begin
    PARITY_BYTE <= PARITY_FUNC(DATA_BYTE);
    PARITY_WORD <= PARITY_FUNC(DATA_WORD);
end FUNC;
```

Actual parameter

Procedure declaration

Syntax :

```
procedure procedure_name (parameter_list) is
    types, constants, other functions, other declarations
begin
    sequential_statements
end procedure_name;
```

```
entity ... is
end ...
architecture ... of ... is
```

```
begin
```

```
process
```

```
begin
```

```
end process;
```

```
end ... ;
```

← Procedure Declaration

← Procedure Calls

← Procedure Declaration

← Procedure Calls

Procedure Examples

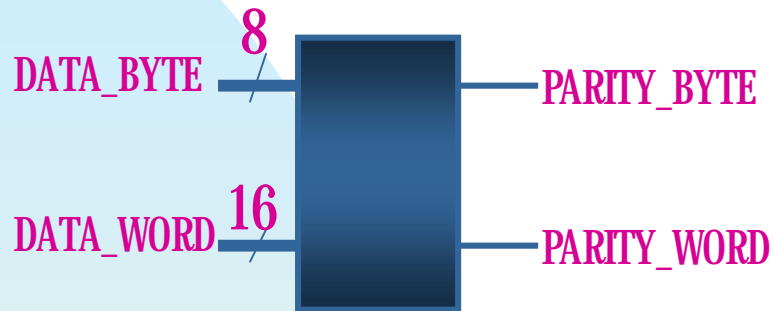
Procedure name

Parameter list

```
procedure PARITY_PROC ( X : std_ulogic_vector;  
                        signal PARITY_BIT : out std_ulogic) is  
    variable TMP : std_ulogic;  
begin  
    TMP := '0';  
    for J in X' range loop  
        TMP := TMP xor X(J);  
    end loop;  
  
    PARITY_BIT <= TMP;  
end PARITY_PROC;
```

Assignment to "out" parameters

Procedure Call



```

procedure PARITY_PROC ( X : std_ulogic_vector;
                        signal PARITY_BIT : out std_ulogic) is
    variable TMP : std_ulogic;
begin
    TMP := '0';
    for J in X'range loop
        TMP := TMP xor X(J);
    end loop;
    PARITY_BIT <= TMP;
end PARITY_PROC;

```

```

entity PARITY is
    port (
        DATA_BYTE : in std_ulogic_vector(7 downto 0);
        DATA_WORD : in std_ulogic_vector(15 downto 0);
        PARITY_BYTE : out std_ulogic;
        PARITY_WORD : out std_ulogic
    );
end PARITY;

architecture PROC of PARITY is
    -- procedure declaration
begin
    PARITY_PROC(DATA_BYTE, PARITY_BYTE);
    PARITY_PROC(DATA_WORD, PARITY_WORD);
end PROC;

```

Actual parameter

Defining Subprograms

```
package P_FUNCS is
architecture FUNCTIONS of PARITY is
    function PARITY_FUNC ( X : std_ulogic_vector)
        return std_ulogic is
        variable TMP : std_ulogic;
    begin
        TMP := '0';
        for J in X'range loop
            TMP := TMP xor X(J);
        end loop;
        return TMP;
    end PARITY_FUNC;
begin
    PARITY_BYTE <= PARITY_FUNC(DATA_BYTE);
    PARITY_WORD <= PARITY_FUNC(DATA_WORD);
end FUNCTIONS;
```

- Package
- Architecture
- Process

```
function PARITY_FUNC( x : std_ulogic_vector)
    return std_ulogic;
end P_FUNCS;

package body P_FUNCS is
    function PARITY_FUNC(x : std_ulogic_vector)
        return std_ulogic;
    begin
        variable TMP : std_ulogic;
        TMP := '0';
        for J in X'range loop
            TMP := TMP xor X(J);
        end loop;
        return TMP;
    end PARITY_FUNC;
end P_FUNC
```

Type Qualification

```

use work P_ARITHMETIC.all;
entity QUALIFY is
port (A, B : in UNSIGNED (3 downto 0);
      EQU : out boolean);
end QUALIFY;

architecture BEH of QUALIFY is
begin
  process (A, B)
  begin
    EQL <= (A <= B);
    -- This is legal
    EQL <= (A <= "1001");
    -- This is ambiguous
    EQL <= (A <= UNSIGNED("1001"));
    -- This is legal
  end process;
end BEH;

```

```

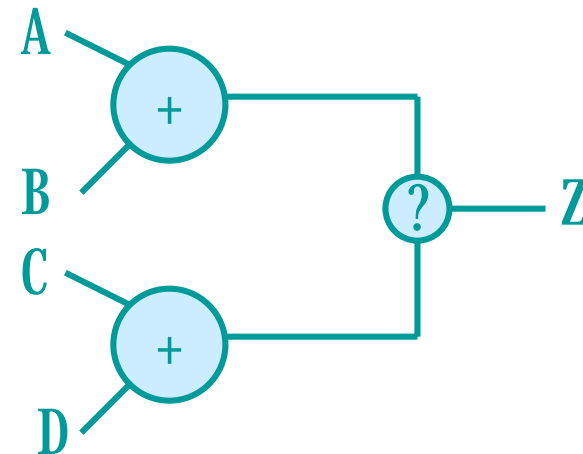
package P_ARITHMETIC is
...
  subtype UNSIGNED is std_logic_vector;
  type SIGNED is array (INTEGER range <>) of std_ulogic;

  function "<=" (L: UNSIGNED; R: UNSIGNED) return BOOLEAN;
  function "<=" (L: SIGNED; R: SIGNED) return BOOLEAN;
  function "<=" (L: UNSIGNED; R: SIGNED) return BOOLEAN;
  function "<=" (L: SIGNED; R: UNSIGNED) return BOOLEAN;
  function "<=" (L: UNSIGNED; R: INTEGER) return BOOLEAN;
  function "<=" (L: INTEGER; R: UNSIGNED) return BOOLEAN;
  function "<=" (L: SIGNED; R: INTEGER) return BOOLEAN;
  function "<=" (L: INTEGER; R: SIGNED) return BOOLEAN;
...

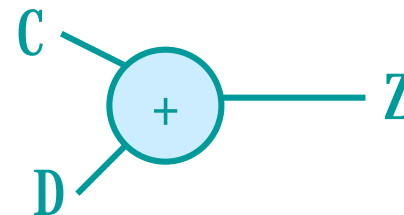
```

Resolution Functions

```
architecture CONCURRENT of MULTIPLE is
    signal A, B, C, D : integer;
    signal Z : RESOLVED_INTEGER;
begin
    Z <= A + B;
    Z <= C + D;
end CONCURRENT;
```



```
architecture SEQUENTIAL of MULTIPLE is
    signal Z, A, B, C, D : integer;
begin
    process (A, B, C, D)
    begin
        Z <= A + B;
        Z <= C + D;
    end process;
end SEQUENTIAL;
```



Resolution Function Definition

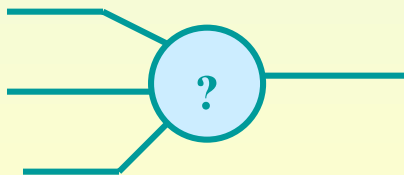
```
subtype STD_LOGIC is RESOLVED STD_ULOGIC;
```

```
function RESOLVED (S : STD_ULOGIC_VECTOR) return STD_ULOGIC is
  variable RESULT : STD_ULOGIC := '-'; --weakest state default
begin
  if (S'length = 1) then return S(S'low);
  else
    -- Iterate through all inputs
    for I in S'range loop
      RESULT := RESOLUTION_TABLE(RESULT, S(i));
    end loop;
    -- Return resultant value
    return RESULT;
  end if;
end RESOLVED;
```

```
CONSTANT resolution_table : stdlogic_table := (
```

	U	X	0	1	Z	W	L	H	-	
('U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U')	U									
('U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X')	X									
('U', 'X', '0', 'X', '0', '0', '0', '0', 'X')	0									
('U', 'X', 'X', '1', '1', '1', '1', '1', 'X')	1									
('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', 'X')	Z									
('U', 'X', '0', '1', 'W', 'W', 'W', 'W', 'X')	W									
('U', 'X', '0', '1', 'L', 'W', 'L', 'W', 'X')	L									
('U', 'X', '0', '1', 'H', 'W', 'W', 'H', 'X')	H									
('U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X')	-									

```
);
```



Advanced Topics

Aim and Topics

- Advanced types
- Overloading
- Definition RTL Code
- Synthesis Coding Styles

Advanced Types

- Enumerated types
- Subtypes
- Composite types
 - ◆ Arrays
 - ◆ Array of array
 - ◆ Record

Enumerated Type Definitions

- VHDL allows the user to define his own type
 - ◆ “Enumerated Type”
- Types cannot be intermixed !

```
type MY_STATE is
    (RESET, IDLE, RW_CYCLE, INT_CYCLE);
...
signal STATE : MY_STATE;
signal TWO_BIT : bit_vector (0 to 1);
...
STATE <= RESET;      ✓
STATE <= "00";       ✗
STATE <= TWO_BIT;    ✗
```

Beware !

Synthesis tools usually offer a way to map each enumeration to a bit pattern

Aggregates

```
signal Z_BUS : bit_vector(3 downto 0);  
signal A_BIT, B_BIT, C_BIT, D_BIT : bit;  
signal BYTE : bit_vector(7 downto 0);
```

```
Z_BUS <= ( A_BIT, B_BIT, C_BIT, D_BIT );
```

aggregates

```
BYTE <= ( 7 => '1', 5 downto 1 => '1', 6 => B_BIT, OTHERS => '0' );
```

Beware !

Some low cost synthesis tools may not support aggregates

Subtype declaration

Syntax :

```
subtype subtype_name is base_type range range_constraint;
```

Example

```
type WORD is array (31 downto 0) of bit;  
  subtype NIBBLE is WORD range 3 downto 0;  
  subtype BYTE is WORD range 7 downto 0;  
  subtype MY_BUS is WORD range 15 downto 0;
```

Array type declaration

Syntax :

type **type_name** is array (**array_range**) of **array_type**;

Example

```
type WORD8 is array (1 to 8) of bit;  
type WORD8 is array (integer range 1 to 8) of bit;  
type BYTE is array (7 downto 0) of bit;  
type WORD is array (31 downto 0) of bit;  
type MEMORY is array (0 to 4095) of word;  
type T_CLOCK_TIME is array (3 downto 0) of integer;
```

Equivalent statements

Arrays Assignments

```
signal Z_BUS : bit_vector(3 downto 0);  
signal C_BUS : bit_vector(0 to 3);
```

```
Z_BUS <= C_BUS;
```

Z_BUS (3)	<-----	C_BUS (0)
Z_BUS (2)	<-----	C_BUS (1)
Z_BUS (1)	<-----	C_BUS (2)
Z_BUS (0)	<-----	C_BUS (3)

Beware !

- Size of array on left and right must be equal
- Elements are assigned by position, not element number

Slice of an Arrays

```
signal Z_BUS, A_BUS : bit_vector(3 downto 0);
signal B_BIT : bit;
signal BYTE : bit_vector(7 downto 0);
```

```
BYTE(5 downto 2) <= A_BUS;
Z_BUS(1 downto 0) <= '0' & B_BIT;
```

```
B_BIT <= A_BUS(0);
Z_BUS <= A_BUS;
```

Beware !

The slice direction must be the same as the signal declaration

```
BYTE(5 downto 2) <= A_BUS;
Z_BUS(0 to 1) <= '0' & B_BIT; ✗
```


Multi-Dimensional Arrays

```

type MEMORY is array (0 to 7, 0 to 3) of bit;
constant ROM: MEMORY := (
    ('0', '0', '0', '0'),
    ('0', '0', '0', '1'),
    ('0', '0', '1', '0'),
    ('0', '0', '1', '1'),
    ('0', '1', '0', '0'),
    ('0', '1', '0', '1'),
    ('0', '1', '1', '0'),
    ('0', '1', '1', '1') );
  
```

Example Reference: `DATA_BIT := ROM(5, 3);`

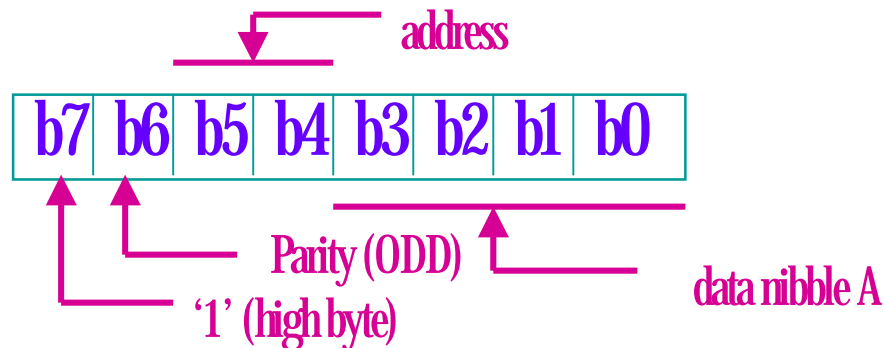
Array of Arrays

```
type WORD is array ( 0 to 3) of bit;
type MEMORY is array (0 to 4) of WORD;
variable ADDR, INDEX : integer;
variable DATA : WORD;
constant ROM: MEMORY := (      ('0', '0', '0', '0'),
                                ('0', '0', '0', '1'),
                                ('0', '0', '1', '0'),
                                ('0', '0', '1', '1'),
                                ('0', '1', '1', '1')      );
```

```
DATA := ROM(addr);
ROM(ADDR) (INDEX)
```

To access a single bit

Records declaration



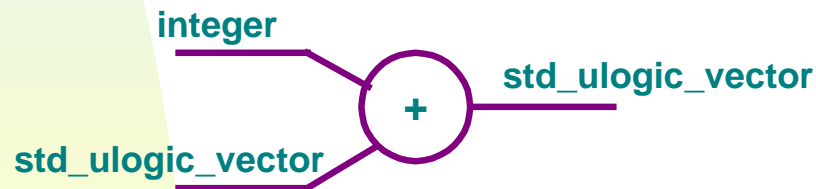
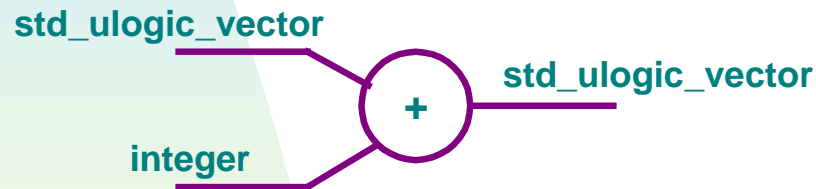
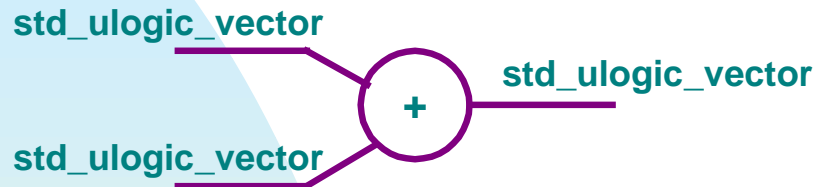
```

type T_PACKAGE is record
  BYTE ID : bit;
  PARITY  : bit;
  ADDRESS : integer range 0 to 3;
  DATA   : bit_vector(3 downto 0);
end record;
signal TX_DATA, RX_DATA : T_PACKAGE;

...
RX_DATA <= TX_DATA;
TX_DATA <= ('1', '0', 2, "0101");
TX_DATA.ADDRESS <= 3;
  
```

Reference name record

Overloading



- Re-define operators
- Different data types
- Called in context

Subprogram Overloading

```
package P_SUBP is
  function SINE (L : integer)          return real;      -- 1
  function SINE (L : real)             return real;      -- 2
  function SINE (L : std_logic_vector) return real;      -- 3
end package P_SUBP;
```

```
use work P_SUBP.all;
entity OVERLOADED is
  port ( A_BUS  : in  std_ulogic_vector(3 downto 0);
         B_INT  : in  integer range 0 to 15;
         C_REAL : in  real;
         A, B, C : out real );
end OVERLOADED;
architecture A of OVERLOADED is
begin
  A <= SINE(A_BUS);      ← Function 3
  B <= SINE(B_INT);      ← Function 1
  C <= SINE(C_REAL);     ← Function 2
end A;
```

- Any subprogram can be overloaded

Argument Overloading

```
package P_AVERAGE is
  function AVERAGE (A, B : integer) return integer;      -- 1
  function AVERAGE (A, B, C : integer) return integer;   -- 2
  function AVERAGE (A, B, C, D : integer) return integer; -- 3
end P_AVERAGE;
```

```
use work P_AVERAGE.all;

entity OVERLOADED is
  port ( A1, B1, C1, D1 : in integer,
         V1, V2, V3 : out integer );
end OVERLOADED;

architecture ARG_OVER of OVERLOADED is
begin
  V1 <= AVERAGE(A1, B1, C1);      ← Function 2
  V2 <= AVERAGE(A1, C1);         ← Function 1
  V3 <= AVERAGE(A1, B1, C1, D1); ← Function 3
end ARG_OVER;
```

arguments

Operator Overloading

```
package P_ARITHMETIC is
  function "+" (L: std_ulogic_vector, R: std_ulogic_vector) return integer;      -- 1
  function "+" (L: std_ulogic_vector, R: std_ulogic_vector) return std_ulogic_vector; -- 2
  function "+" (L: std_ulogic_vector, R: integer) return std_ulogic_vector;    -- 3
  function "+" (L: integer, R: std_ulogic_vector) return std_ulogic_vector;    -- 4
end P_ARITHMETIC;
```

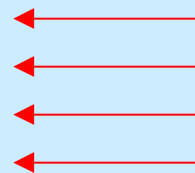
```
use work.P_ARITHMETIC.all;
entity OVERLOADED is
  port ( A_BUS, B_BUS : in std_ulogic_vector(3 downto 0);
         A_INT, B_INT : in integer range 0 to 15;
         Y_BUS, Z_BUS : out std_ulogic_vector(3 downto 0);
         Y_INT, Z_INT : out integer range 0 to 15 );
end OVERLOADED;
```

```
architecture A of OVERLOADED is
```

```
begin
```

```
Y_INT <= A_INT + B_INT;
Z_INT <= A_BUS + B_BUS;
Z_BUS <= A_BUS + B_BUS;
Y_BUS <= A_BUS + A_INT;
Z_BUS <= A_BUS + B_INT;
```

```
end A;
```

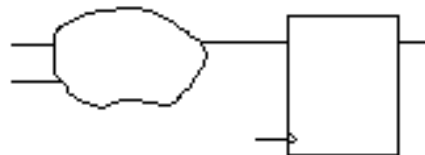


Function 1
Function 2
Function 3
Function 4

RTL (Register Transfer Level) Style

Clocked

```
process
begin
    wait ⏏
    --logic
end process;
```



Combinational

```
process( )
begin
    --logic
end process;
```



Complete Sensitivity Lists

```
process (A, B, SEL)
begin
  if (SEL = '1') then
    Z <= A;
  else
    Z <= B;
  end if;
end process;
```

- List all signals read
- Synthesis : complete for combinational logic

Incomplete Assignments

```
Library IEEE;
use IEEE.Std_Logic_1164.all;

entity INCOMP_IF is
port ( EN, D : in std_ulogic;
      Q      : out std_ulogic );

architecture A of INCOMP_IF is
begin
    process (EN, D)
    begin
        if (EN = '1') then
            Q <= D;
        end if;
    end process;
end A;
```

- What is the value of Q if EN = '0' ?
- What hardware should be built if this is synthesised?

* Answer Latch !

Rules for Synthesis of Combinational Logic

- Complete sensitivity list
- Default assignments to prevent latches

Sensitivity list

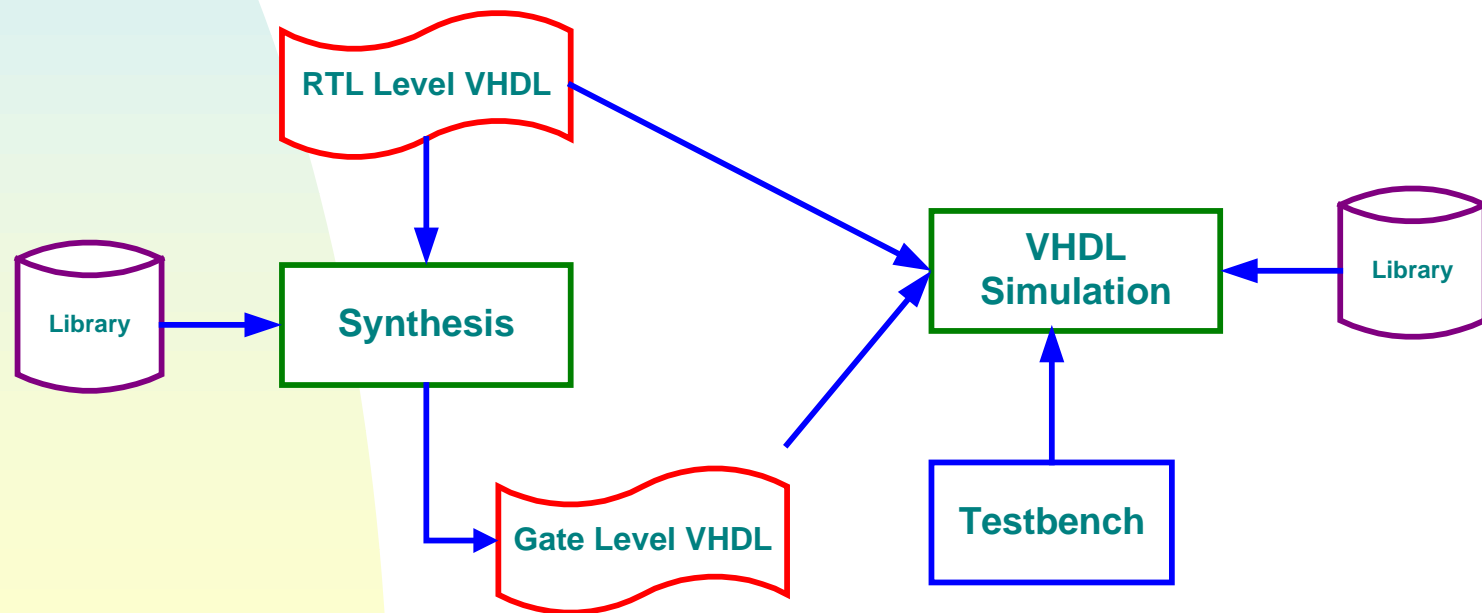
```
process (A, B, SEL)
begin
  if (SEL = '1') then
    Z <= A;
  else
    Z <= B;
  end if;
end process;
```

```
process (EN, D)
begin
  if (EN = '1') then
    Q <= D;
  end if;
end process;
```

What is the value of Q if EN = '0' ?


“Ideal” Design Flow

- Ideally do RTL and gate simulation in same simulator
- Historical issues
 - ◆ Library standard
 - ◆ Slow simulation




Describing a Rising Clock for Synthesis

```
process
begin
    wait until CLK'event and CLK = '1'
           and CLK'last_value = '0';
    Q <= D;
end process;
```




```
process
begin
    wait until CLK = '1';
    Q <= D;
end process;
```




```
process
begin
    wait until CLK'event and CLK = '1';
    Q <= D;
end process;
```



```
process (CLK)
begin
    if (CLK = '1') then
        Q <= D;
    end if;
end process;
```



```
process (CLK)
begin
    if (CLK'event and CLK = '1') then
        Q <= D;
    end if;
end process;
```



Register Inference in Synthesis

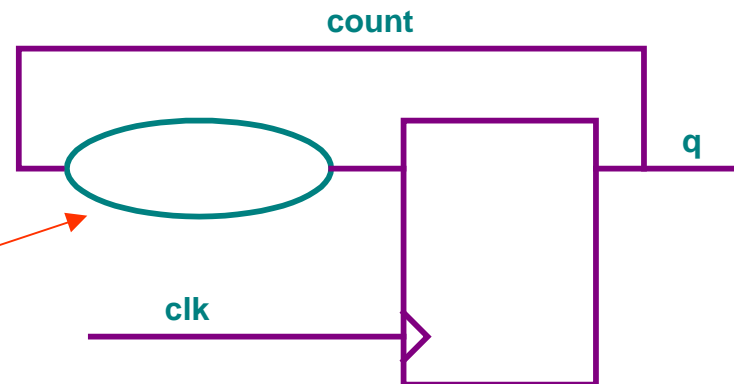
```

Library IEEE;
use IEEE.Std_logic_1164.all;

entity COUNTER is
port ( CLK : in std_logic;
      Q : out integer range 0 to 15);
end COUNTER;

architecture A of COUNTER is
  signal COUNT : integer range 0 to 15;
begin
  process (CLK)
  begin
    if CLK'event and CLK = '1' then
      if (COUNT >= 9) then
        COUNT <= 0;
      else
        COUNT <= COUNT + 1;
      end if;
    end if;
  end process;
  Q <= COUNT;
end A;
  
```

- Registers are inferred on all signal assignments in clocked processes



Asynchronous Reset Registers

```
Library IEEE;
use IEEE.Std_logic_1164.all;

entity ASYNC_FLOP is
port ( D, CLK, RST : in  std_ulogic;
      Q             : out std_ulogic);
end ASYNC_FLOP;

architecture B of ASYNC_FLOP is
begin
  process (CLK, RST)
  begin
    if (RST = '1') then
      Q <= '0';
    elsif (CLK'event and CLK = '1') then
      Q <= D;
    end if;
  end process;
end B;
```

- If / Elself structure
 - ◆ final elsif has edge
 - ◆ no else
- Has a sensitivity list !!

Clocked Process Rules

```
process (CLK, RST)
begin
  if (RST = '1') then
    -- reset all registers
  elsif (CLK'event and CLK = '1') then
    -- all combination logic
  end if;
end process;
```

```
process (CLK)
begin
  if (CLK'event and CLK = '1') then
    -- all combination logic
  end if;
end process;
```

- Wait form
 - ◆ No sensitivity list
- If form
 - ◆ clock and reset only in sensitivity list
- All signals assigned get a register

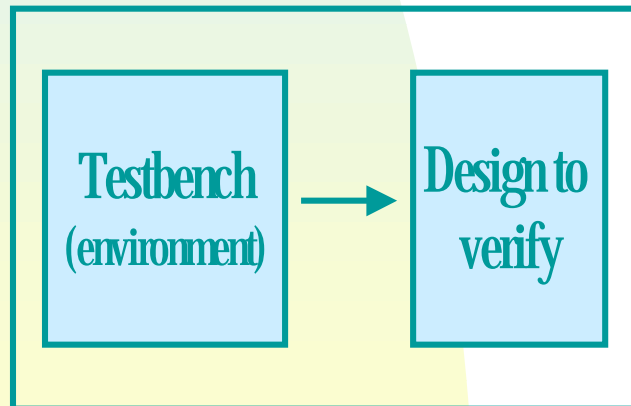
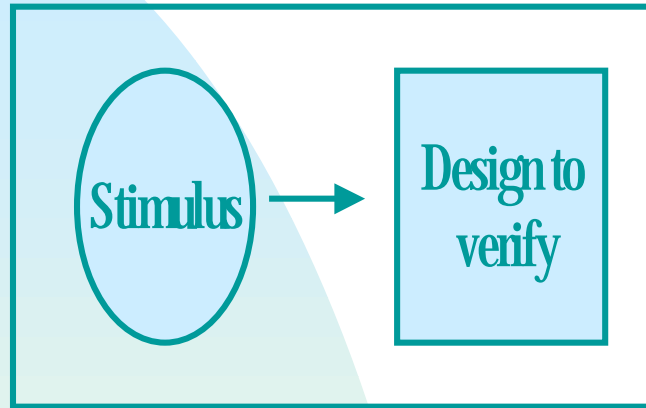
Testbench Coding Styles

Aim and Topics

- Testbench configurations
- Stimulus styles
- Assertions

Testbench Organisation

Testbench



■ Simple testbench

- ◆ Just send data to design
- ◆ No interaction
- ◆ Few processes

■ Sophisticated testbench

- ◆ Models environment around design
- ◆ Talks to design
- ◆ Evolves towards boards model

Stimulus from Loops

```
Library IEEE;
use IEEE.Std_logic_1164.all;

entity TESTBENCH is
end TESTBENCH;

architecture USE_LOOP of TESTBENCH is
    signal A_BUS : std_ulogic_vector(7 downto 0);
begin
    process
    begin
        for I in 0 to 4 loop
            A_BUS <= To_Std_ulogic( I, A_BUS'length);
            wait for 10 ns;
        end loop;
    end process;
end USE_LOOP;
```

Stimulus from Array Constant

```
architecture USE_ARRAY of TESTBENCH is
    signal A_BUS : std_ulogic_vector(7 downto 0);
    type T_DATA is array(0 to 4) of std_ulogic_vector(7 downto 0);
    constant DATA : T_DATA := ( "00000000",
                                   "00000001",
                                   "00000010",
                                   "00000011",
                                   "00000100" );

begin
    process
    begin
        for I in DATA'range loop
            A_BUS <= DATA(I);
            wait for 10ns;
        end loop;
    end process;
end USE_ARRAY;
```

“In line” Stimulus

```
architecture IN_LINE of TESTBENCH is
    signal A_BUS : std_ulogic_vector(7 downto 0);

begin
    process
    begin
        A_BUS <= "00000000";
        wait for 10ns;

        A_BUS <= "00000001";
        wait for 10ns;
        -- ETC, ETC
    end process;
end IN_LINE;
```

Assertions

Syntax :

```
assert condition
report string_expression
severity severity_level;
```

```
architecture BEHAVE of TESTBENCH is
    signal RESULT : bit;
    ...
begin
    ...
    process
        constant EXPECTED : bit := '1';
    begin
        assert RESULT = EXPECTED
            report "Unexpected result!"
            severity ERROR;
        wait;
    end process;
end BEHAVE;
```

Severity levels defined in package STANDARD

- ◆ NOTE
- ◆ WARNING
- ◆ ERROR (default)
- ◆ FAILURE
(usually causes simulation to halt)

Print if condition is false

Creating Clock and Resets

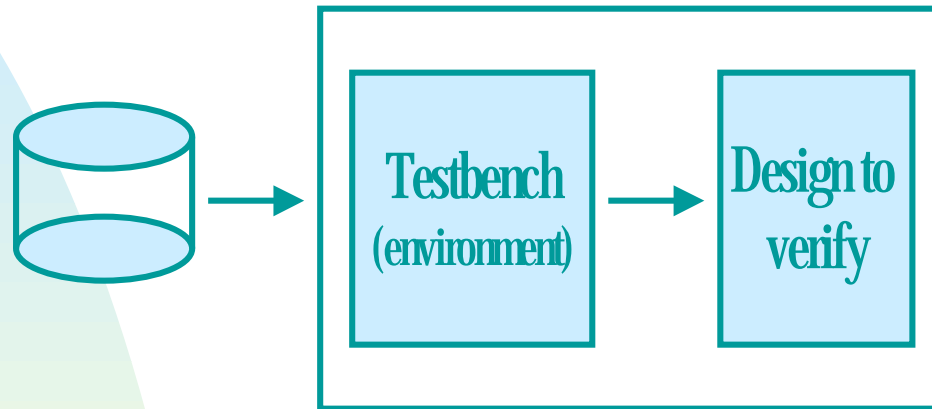
```
signal CLK : std_logic := '0';
constant PERIOD : time := 50 ns;
```

```
CLK <= not CLK after PERIOD/2;
```

```
RESET <= '1', '0' after 3* PERIOD;
```

```
constant INITIAL_CLOCK : std_logic := '1';
constant MAX_CYCLES : integer := 1000;
constant SIM_END_TIME : time := PERIOD * MAX_CYCLES;
process
begin
    while NOW <= SIM_END_TIME loop
        CLK <= INITIAL_CLOCK;
        wait for PERIOD/2;
        CLK <= not INITIAL_CLOCK;
        wait for PERIOD/2;
    end loop;
    assert FALSE
    report "Simulation is Over!";
    severity FAILURE;
end process;
```

Stimulus from a File



- E.g. image data
- Created by another program/tool

Reading from a File

```

use Std.TextIO.all;
architecture READFILE of TESTBENCH is
    signal A_INT, B_INT : integer;
begin
    process
        variable L : line;
        variable INT : integer;
        file TEXT_FILE : text is in "stimulus.vec";
        begin
            while not endfile(TEXT_FILE) loop
                readline(TEXT_FILE, L);
                read(L, INT);
                A_INT <= INT;
                read(L, INT);
                B_INT <= INT;
                wait for 10 ns;
            end loop;
        end process;
    end READFILE;

```

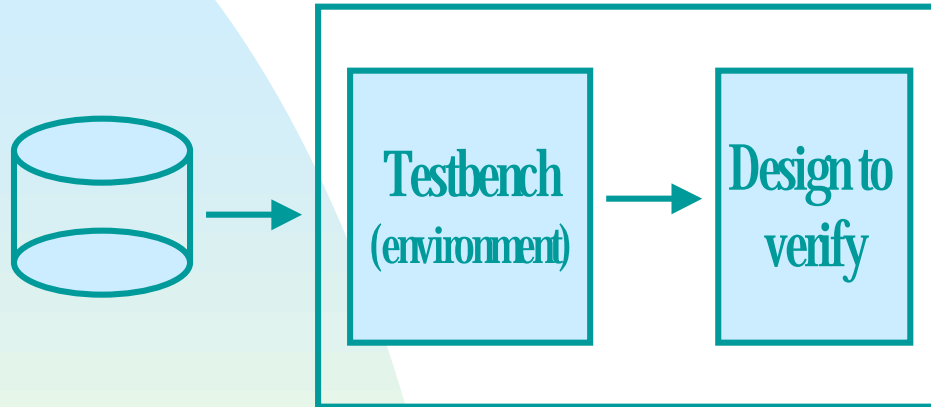
- - stores a complete line from file
- - reference to specific file

- - read a complete line into VECTOR
- - read the first integer into INT
- - read the second integer into INT

1	45
23	21
32	1
45	54
104	110

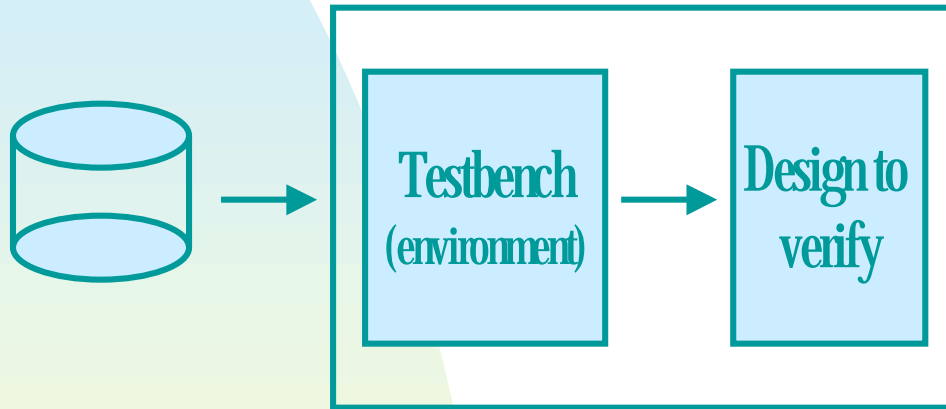
stimulus.vec

TextIO Capabilities



- **Able to read/write**
 - ◆ bit, bit_vector
 - ◆ boolean
 - ◆ character, string
 - ◆ integer, real
 - ◆ time
- **Procedures used for file IO:**
 - ◆ read(...)
 - ◆ readline(...)
 - ◆ write(...)
 - ◆ writeline(...)
- **Reference**
 - ◆ LRM section 14.3

TextIO Advantages



- Stimulus easy changing
- Less analysis/compile time

Use of Subprograms

```

procedure MEM_WRITE (DATA: in T_DATA; ADDR: in T_ADDR) is
begin
  -- Memory write procedure
end MEM_WRITE;
procedure MEM_READ (DATA: out T_DATA; ADDR: in T_ADDR) is
begin
  -- Memory read procedure
end MEM_READ;
  
```

```

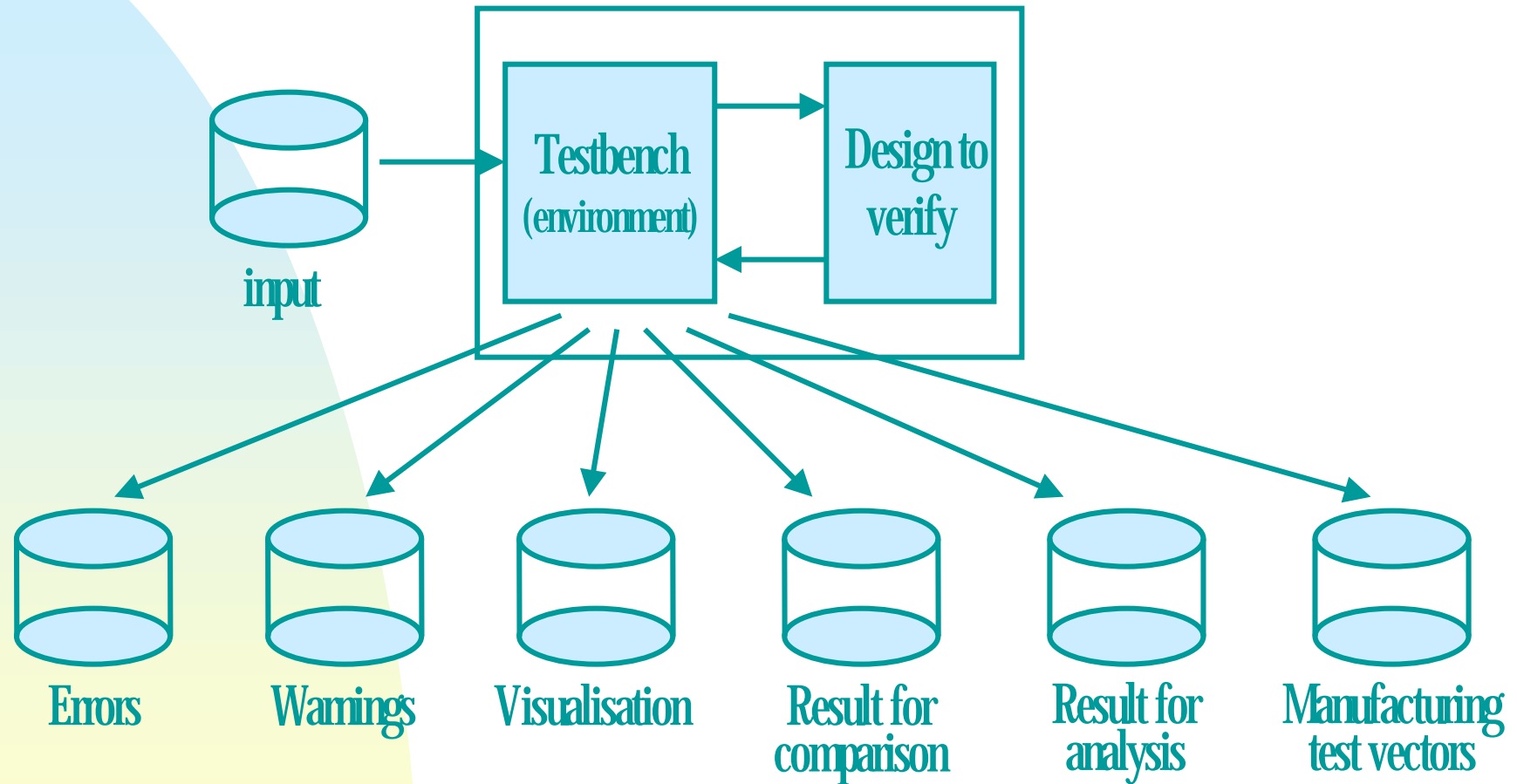
procedure MEM_INIT() is
begin
  MEM_WRITE("1000", "00");
  MEM_WRITE("0100", "01");
  MEM_WRITE("0010", "10");
  MEM_WRITE("0001", "11");
end MEM_INIT;
  
```

- Testbench is behavioural
- Gets large quickly
- Use software style structure
 - ◆ Function and procedures

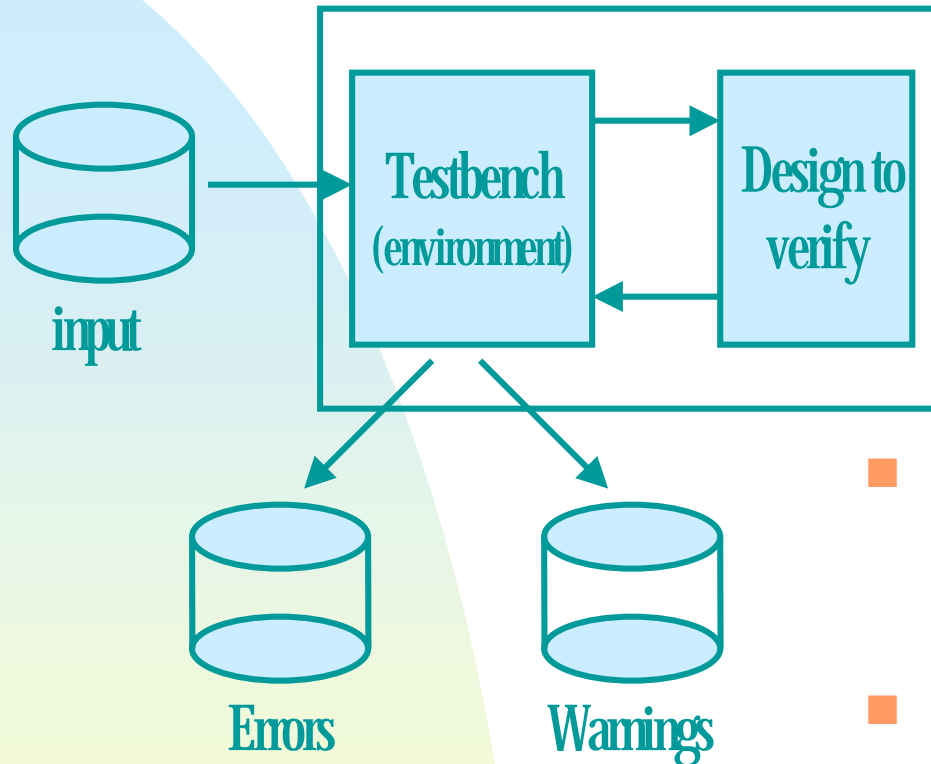
```

procedure MEM_TEST() is
  variable VALUE_READ: T_DATA;
begin
  MEM_INIT();
  MEM_READ(VALUE_READ, "01");
  if VALUE_READ /= "0100" then
    -- Error handling
  end if;
end MEM_TEST;
  
```

Output from Simulation

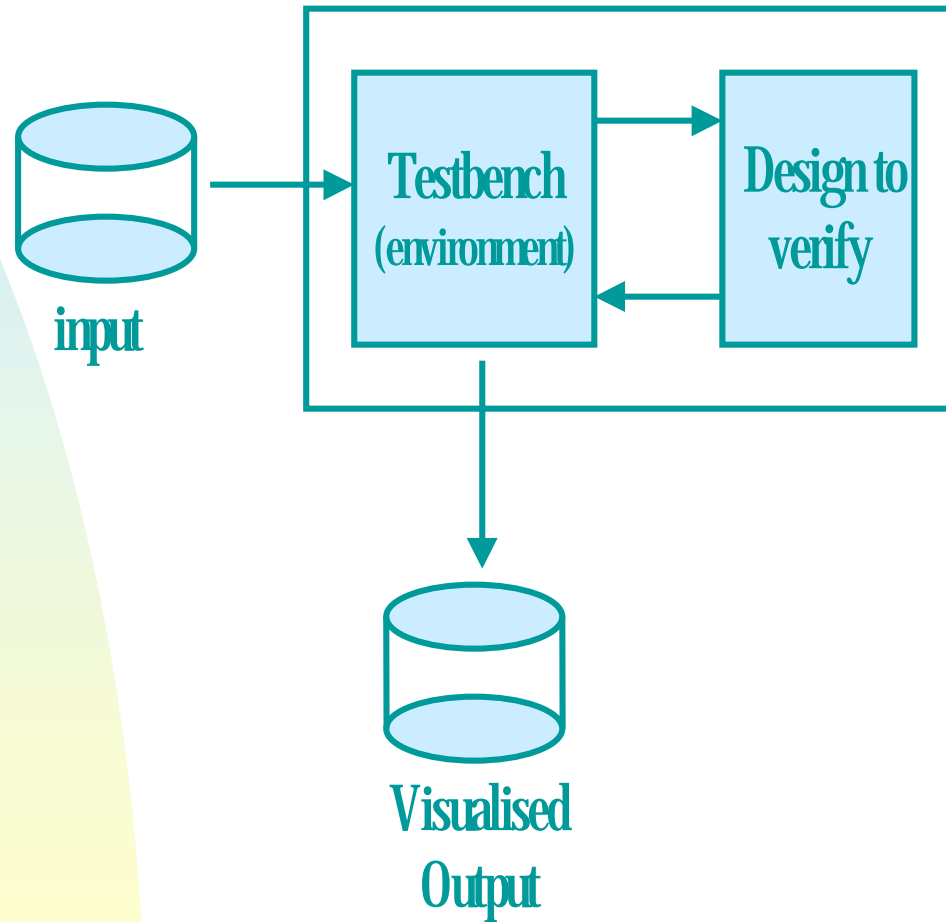


Output from Simulation

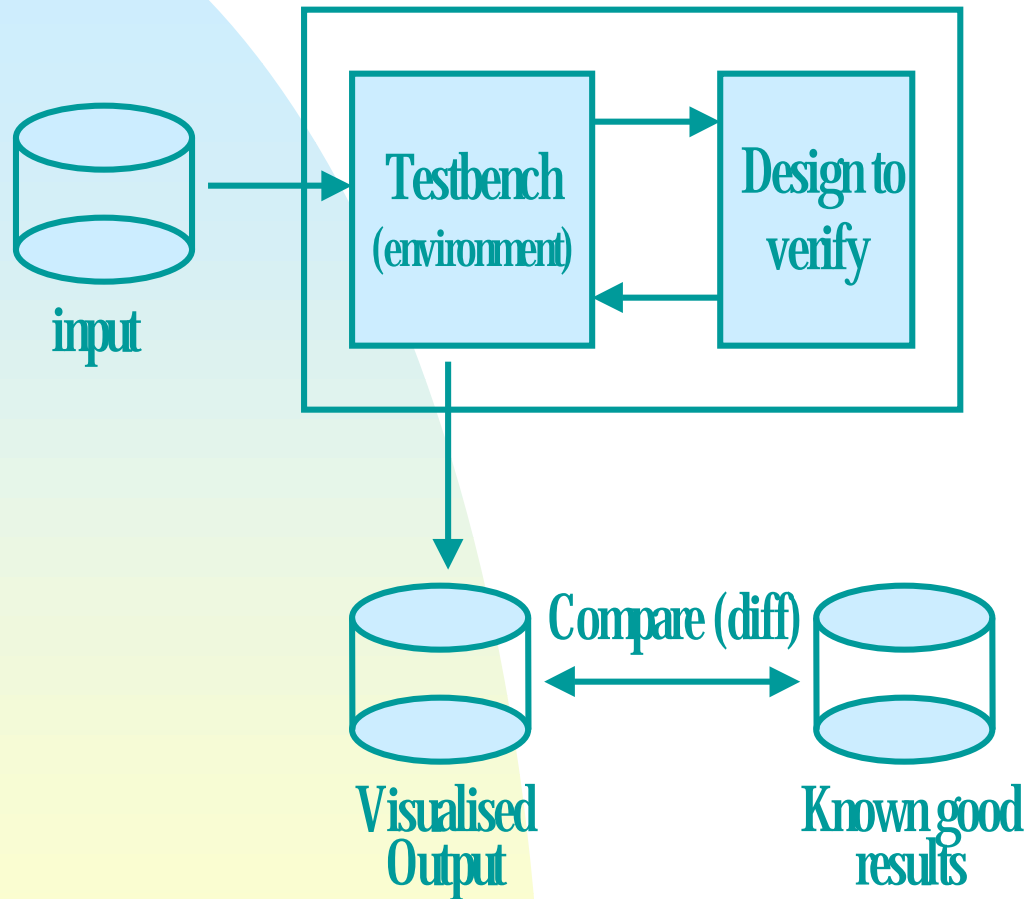


- Can output to
 - ◆ Simulator output (assertion)
 - ◆ File (textIO)
- Best to keep errors and warnings separate
 - ◆ Errors : have to fix these !
 - ◆ Warnings : help to debug issues

Visualised Output

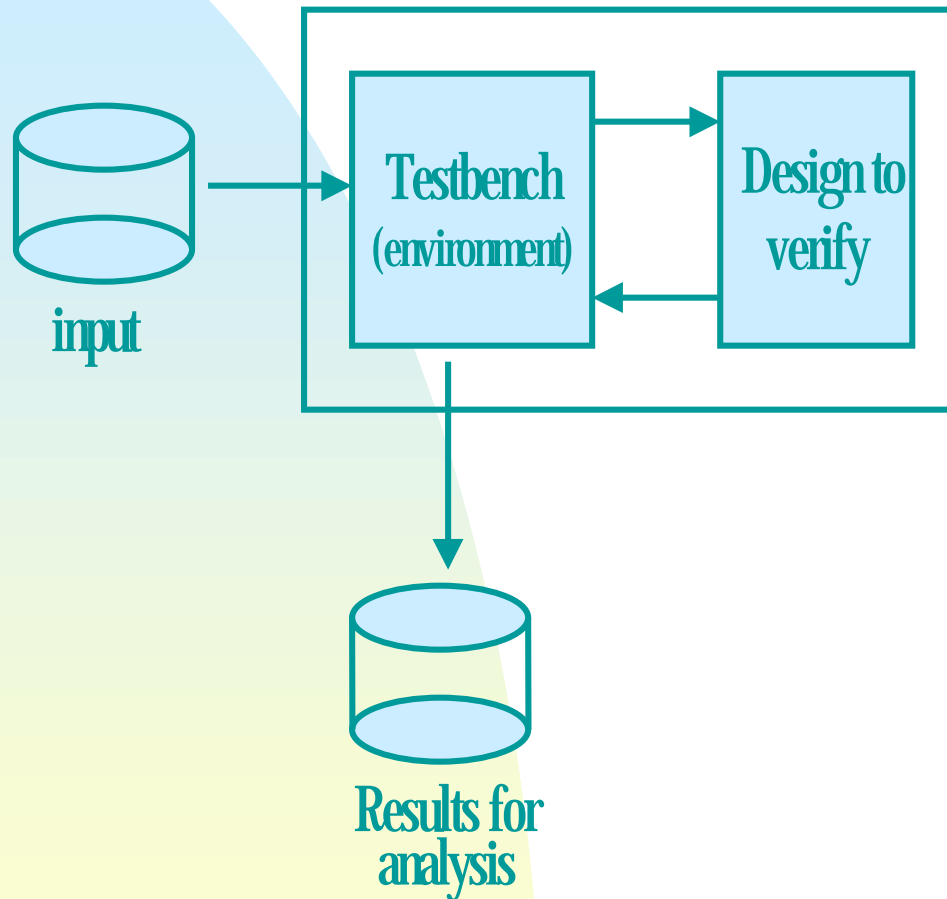


Results for Comparion



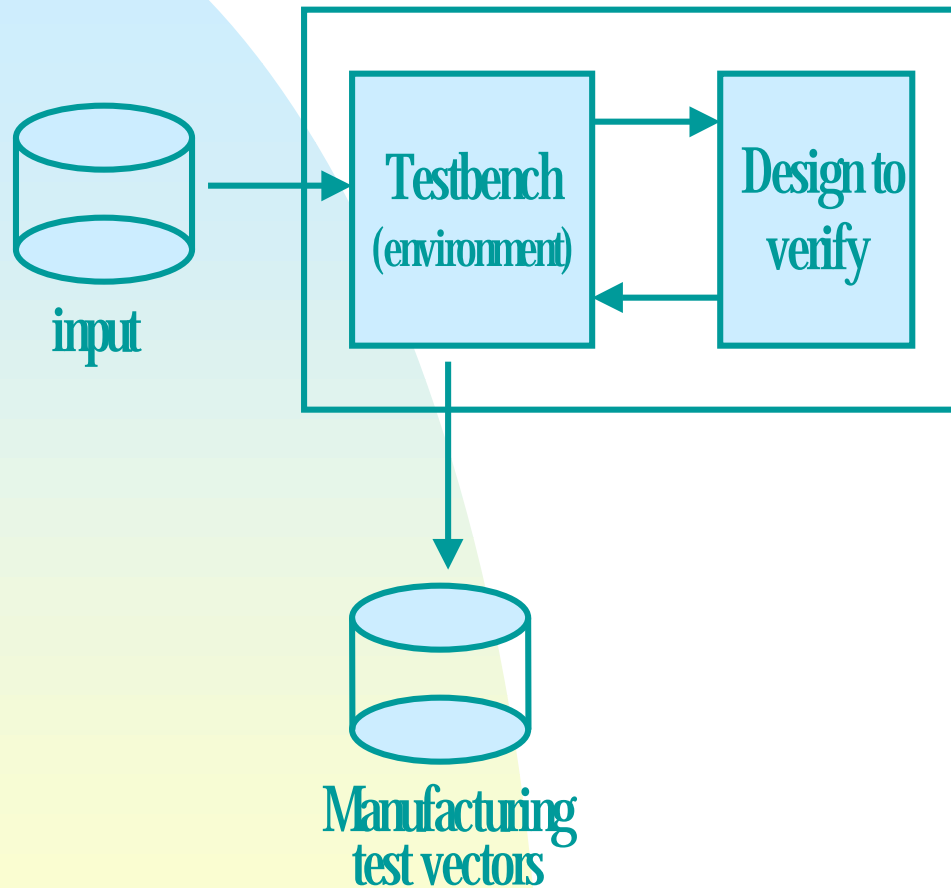
- Compare results for different simulations
- Essential for large designs
 - ◆ Studying waveforms very tedious!
- If comparing between abstraction levels
 - ◆ (Behavioural/RTL, RTL/Gates)
 - ◆ Output sequence of data values
 - ◆ Time independent

Results for Analysis



- When simulation goes wrong
- More detail than results for comparison
- Do not mix results for comparison with results for analysis

Manufacturing Test Vectors



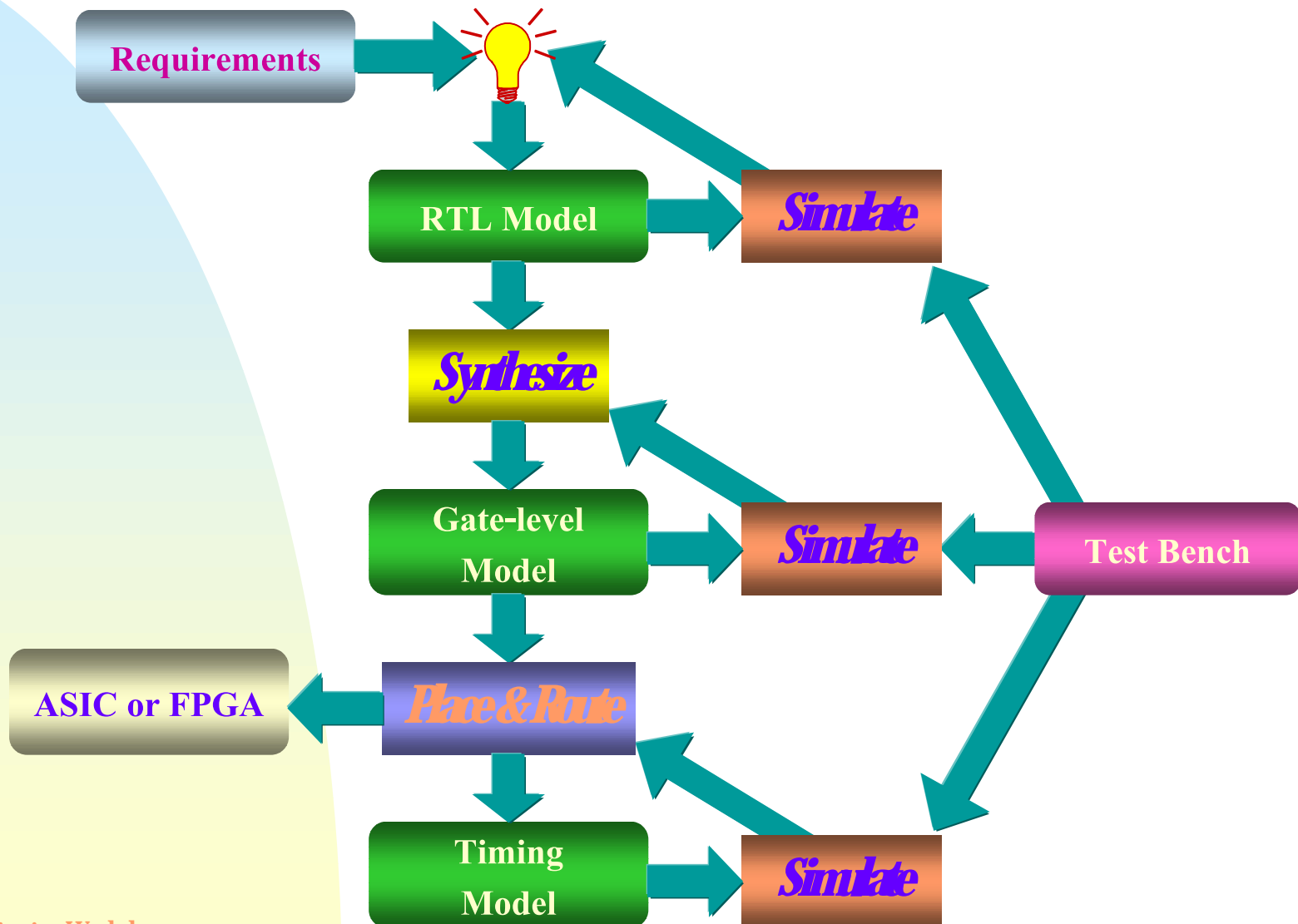
- May use simulation output for part/all of manufacturing test
- Strobe values when they will be stable in gate level design

VHDL & Logic Synthesis

Aim and Topics

- **Combinational circuit Synthesis**
 - ◆ Multiplexer
 - ◆ Encoder & Decoder
 - ◆ Tri-State buffer
 - ◆ Bidirection buffer
 - ◆ Arithmetic Logic Unit (ALU)
 - ◆ Finite State Machine
- **Sequential circuit Synthesis**
 - ◆ D-Latch
 - ◆ Asynchronous Reset Flip-Flop
 - ◆ Synchronous Reset Flip-Flop

Basic Design Methodology



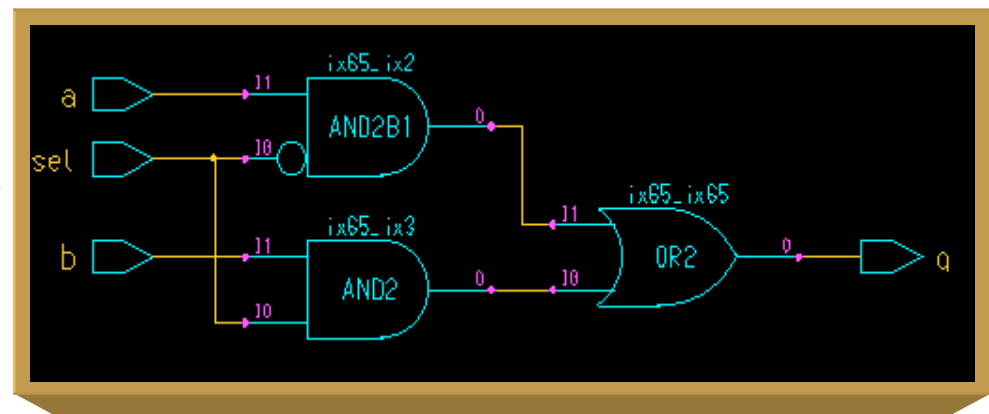
2 to 1 Multiplexer

```
Library IEEE;
use IEEE.Std_logic_1164.all;
```

```
entity MUX_2_1 is
port ( A, B : in std_logic;
      SEL : in std_logic;
      Q : out std_logic );
end MUX_2_1;
```

```
architecture B of MUX_2_1 is
begin
    Q <= A when (SEL = '0') else B;
end B;
```

Synthesized Circuit



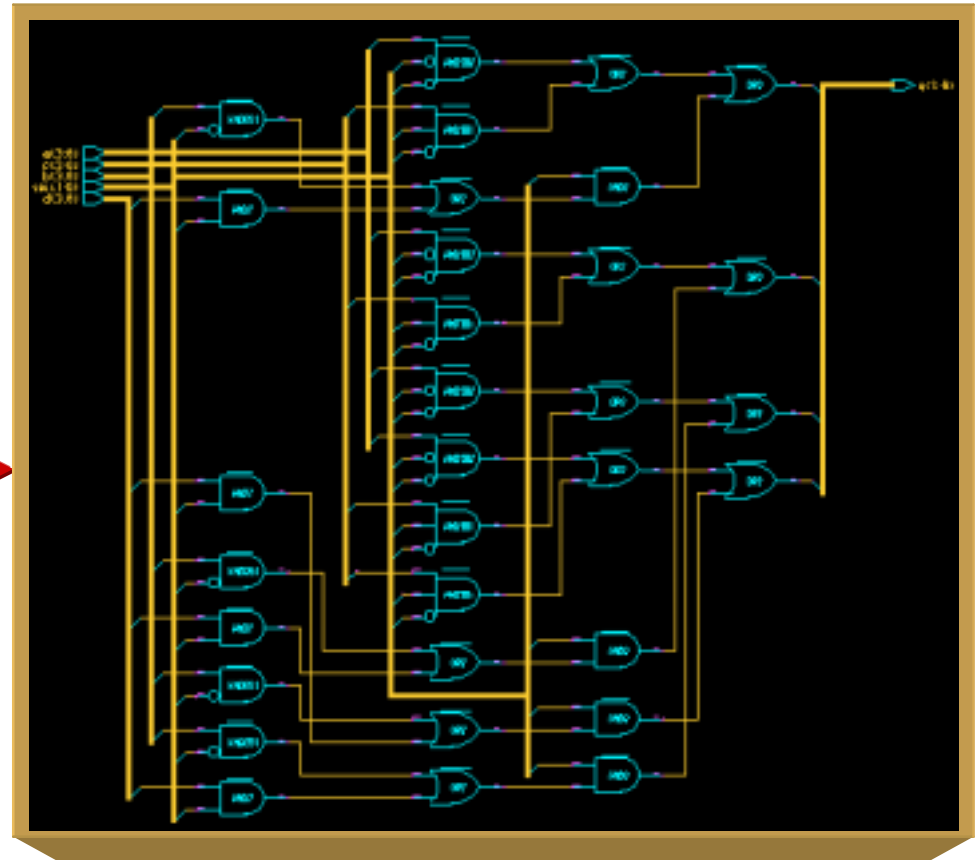
4 to 1 Multiplexer

```

Library IEEE;
use IEEE.Std_logic_1164.all;

entity MUX_4_1 is
    port(A, B, C, D: in std_ulogic_vector(3 downto 0);
         SEL: in std_ulogic_vector(1 downto 0);
         Q: out std_ulogic_vector(3 downto 0));
end MUX_4_1;

architecture B of MUX_4_1 is
begin
    process(A, B, C, D, SEL)
    begin
        if (SEL = "00") then
            Q <= A;
        elsif (SEL = "01") then
            Q <= B;
        elsif (SEL = "10") then
            Q <= C;
        else
            Q <= D;
        end if;
    end process;
end B;
    
```



Synthesized Circuit

8 to 3 Encoder

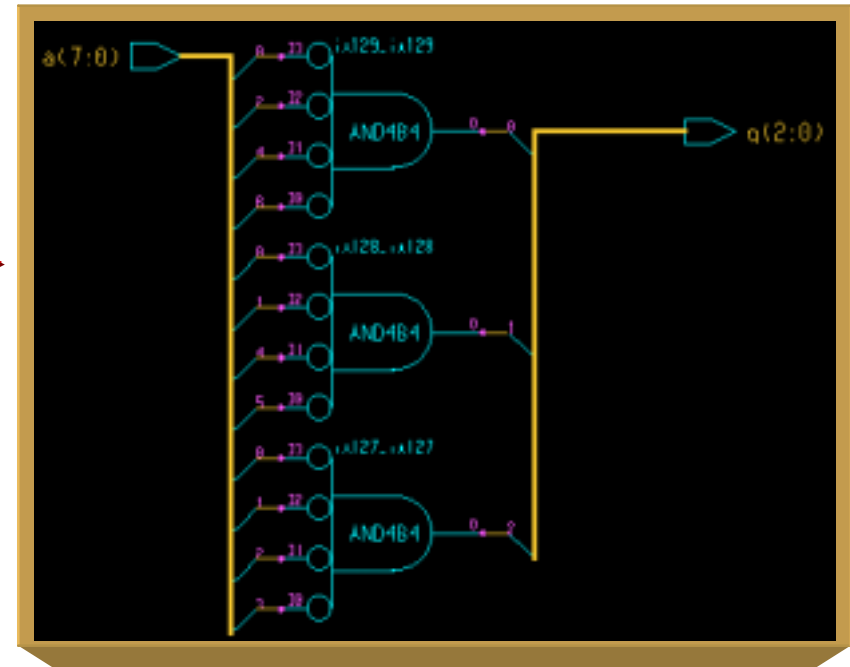
Synthesized Circuit

```

library IEEE;
use IEEE.Std_Logic_1164.all;

entity ENCODE_8_3 is
    port(A : in std_ulogic_vector(7 downto 0);
         Q : out std_ulogic_vector(2 downto 0));
end ENCODE_8_3;

architecture BEHAVE of ENCODE_8_3 is
begin
    P1: process(A)
    begin
        case A is
            when "00000001" => Q <= "000";
            when "00000010" => Q <= "001";
            when "00000100" => Q <= "010";
            when "00001000" => Q <= "011";
            when "00010000" => Q <= "100";
            when "00100000" => Q <= "101";
            when "01000000" => Q <= "110";
            when "10000000" => Q <= "111";
            when others => Q <= "XXX";
        end case;
    end process P1;
end BEHAVE;
    
```



```

Q <= "000" when A = "00000001" else
      "001" when A = "00000010" else
      "010" when A = "00000100" else
      "011" when A = "00001000" else
      "100" when A = "00010000" else
      "101" when A = "00100000" else
      "110" when A = "01000000" else
      "111" when A = "10000000" else
      "XXX";
    
```

3 to 8 Decoder

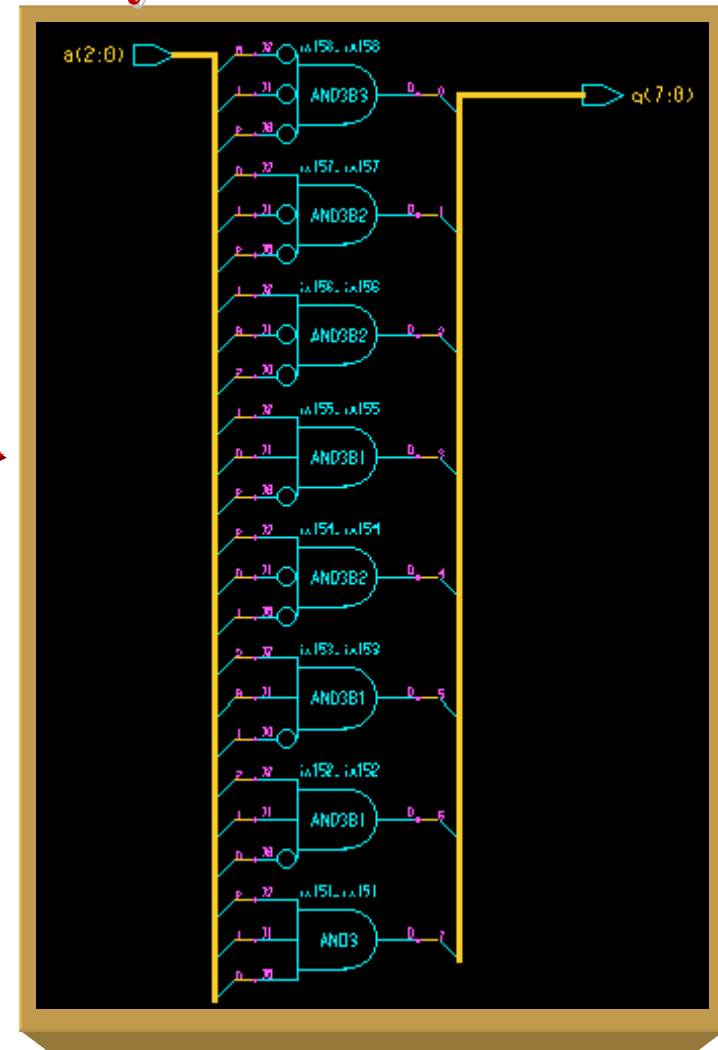
Synthesized Circuit

```

use IEEE.Std_Logic_1164.all;

entity DECODE_8_3 is
  port(A : in  std_ulogic_vector(2 downto 0);
       Q : out std_ulogic_vector(7 downto 0));
end DECODE_8_3;

architecture BEHAVE of DECODE_8_3 is
begin
  P1: process(A)
  begin
    case A is
      when "000" => Q <= "00000001";
      when "001" => Q <= "00000010";
      when "010" => Q <= "00000100";
      when "011" => Q <= "00001000";
      when "100" => Q <= "00010000";
      when "101" => Q <= "00100000";
      when "110" => Q <= "01000000";
      when others => Q <= "10000000";
    end case;
  end process P1;
end BEHAVE;
  
```



Tri-State buffers

```

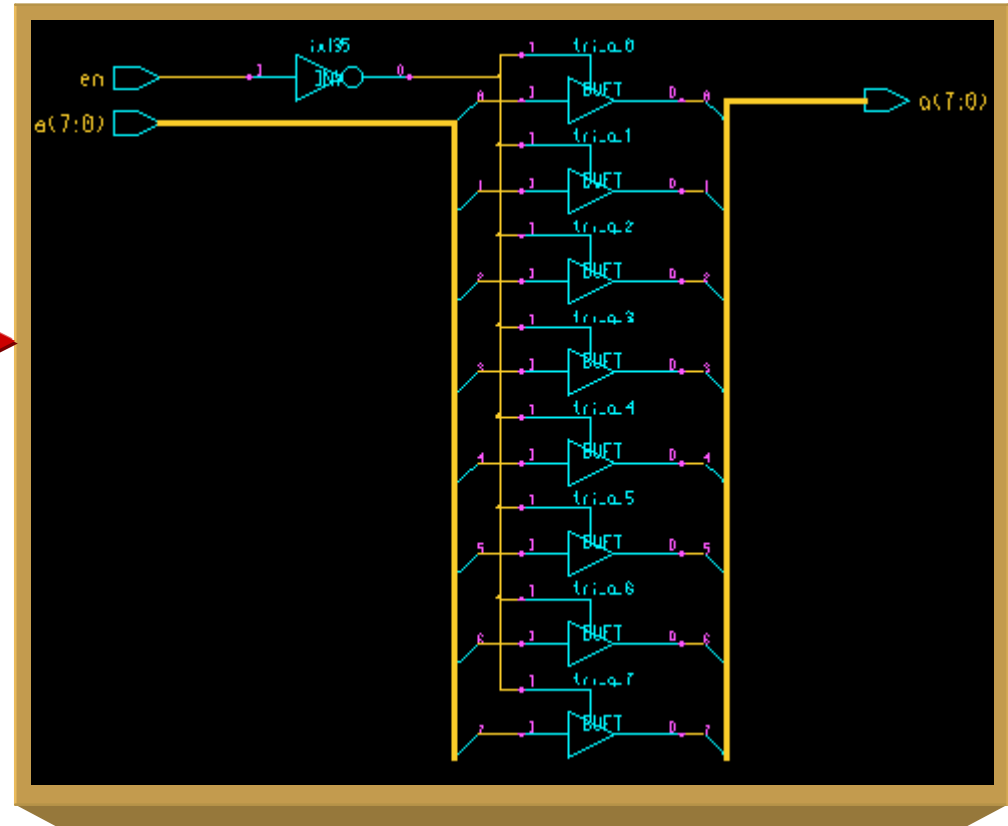
library IEEE;
use IEEE.Std_Logic_1164.all;

entity TRI_STATE is
  port(A : in std_ulogic_vector(7 downto 0);
        EN : in std_ulogic;
        Q : out std_ulogic_vector(7 downto 0));
end TRI_STATE;

architecture RTL of TRI_STATE is

begin
  process(A, EN)
  begin
    if (EN = '1') then
      Q <= A;
    else
      Q <= "ZZZZZZZZ";
    end if;
  end process;
end RTL;
  
```

Synthesized Circuit



Bidirectional Buffers

```

library IEEE;
use IEEE.Std_Logic_1164.all;

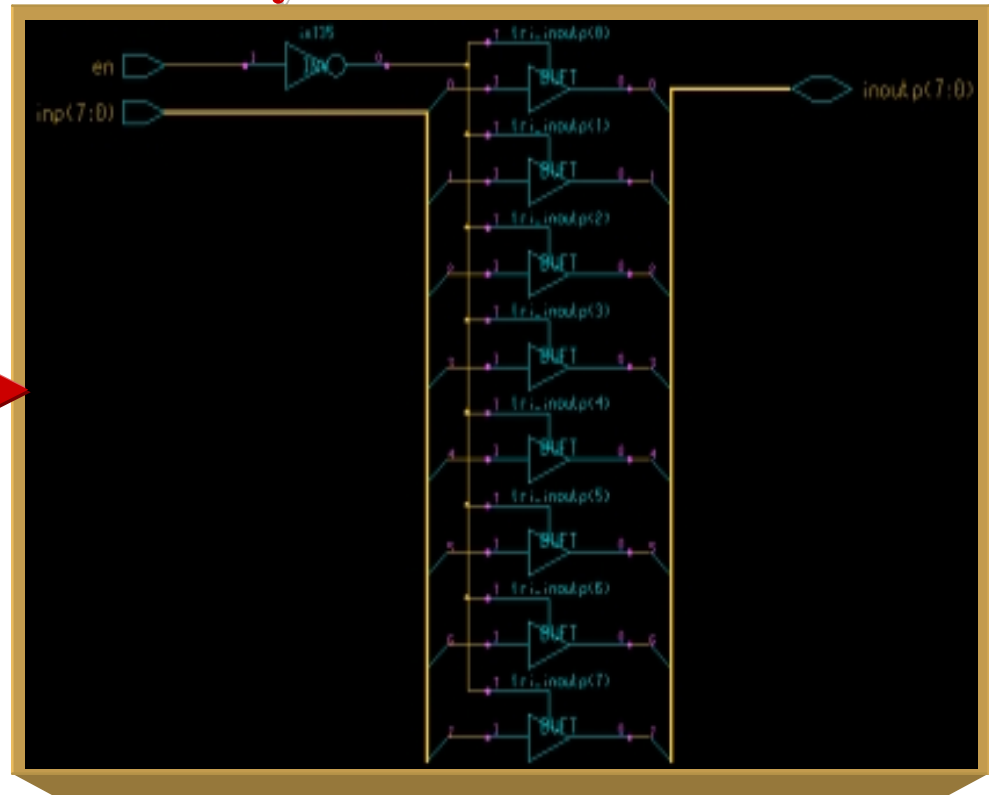
entity BIDIR is
port (INP: in std_logic_vector(7 downto 0);
      EN: in std_logic;
      INOUTP: inout std_logic_vector(7 downto 0)
);
end BIDIR;

architecture RTL of BIDIR is

begin
  INOUTP <= INP when EN = '1' else
    "ZZZZZZZZ";
end RTL;
  
```



Synthesized Circuit



4-Bit Arithmetic Logic Unit (ALU)



```
Library IEEE;
Use IEEE.Std_logic_1164.all;
```

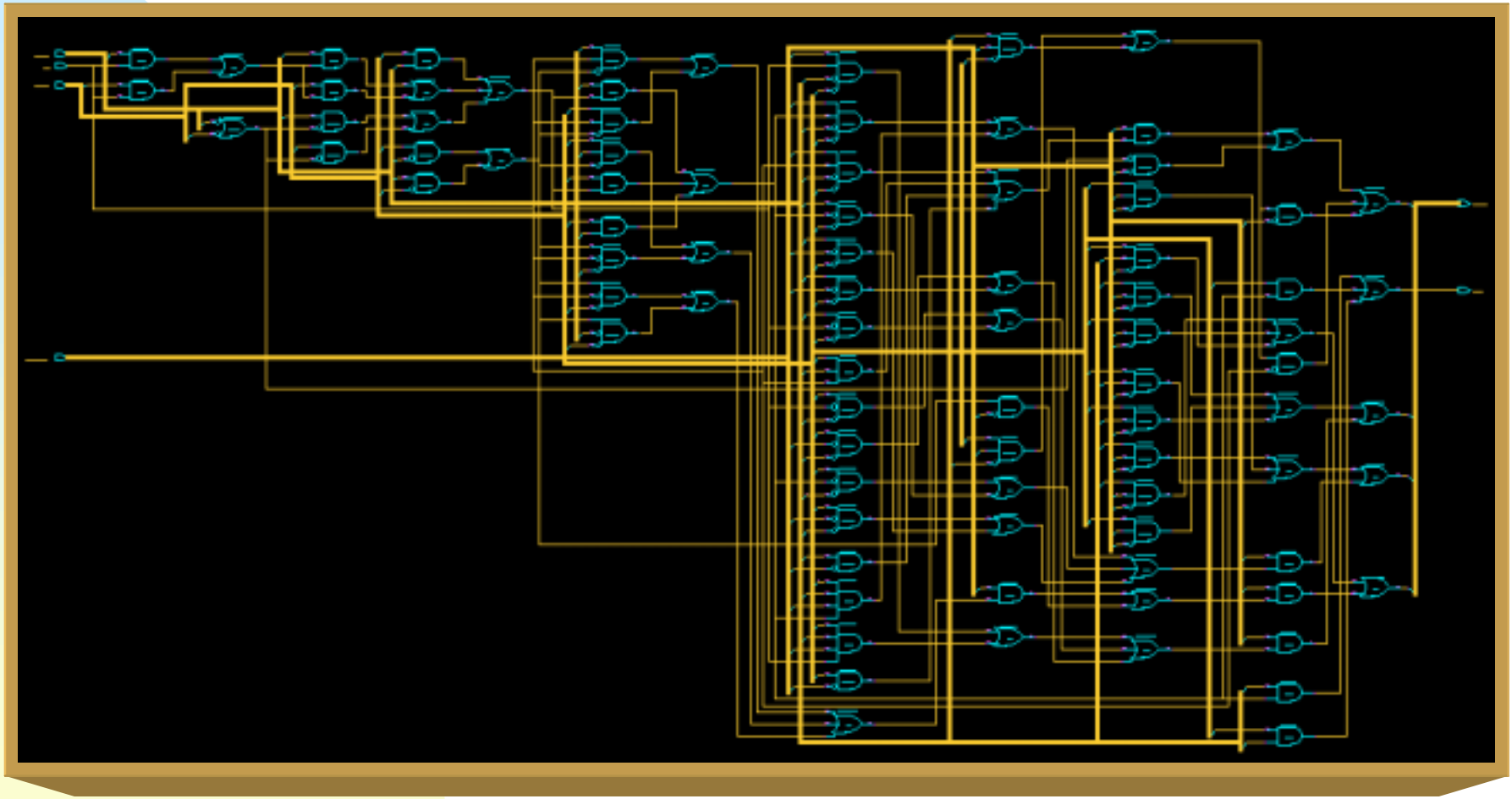
```
entity ALU4BIT is
  port (A, B: in std_logic_vector(3 downto 0);
        MODE: in std_logic_vector(1 downto 0);
        CIN: in std_logic;
        Q: out std_logic_vector(3 downto 0);
        COUT: out std_logic);
end ALU4BIT;
```

architecture BEHAVE of ALU4BIT is

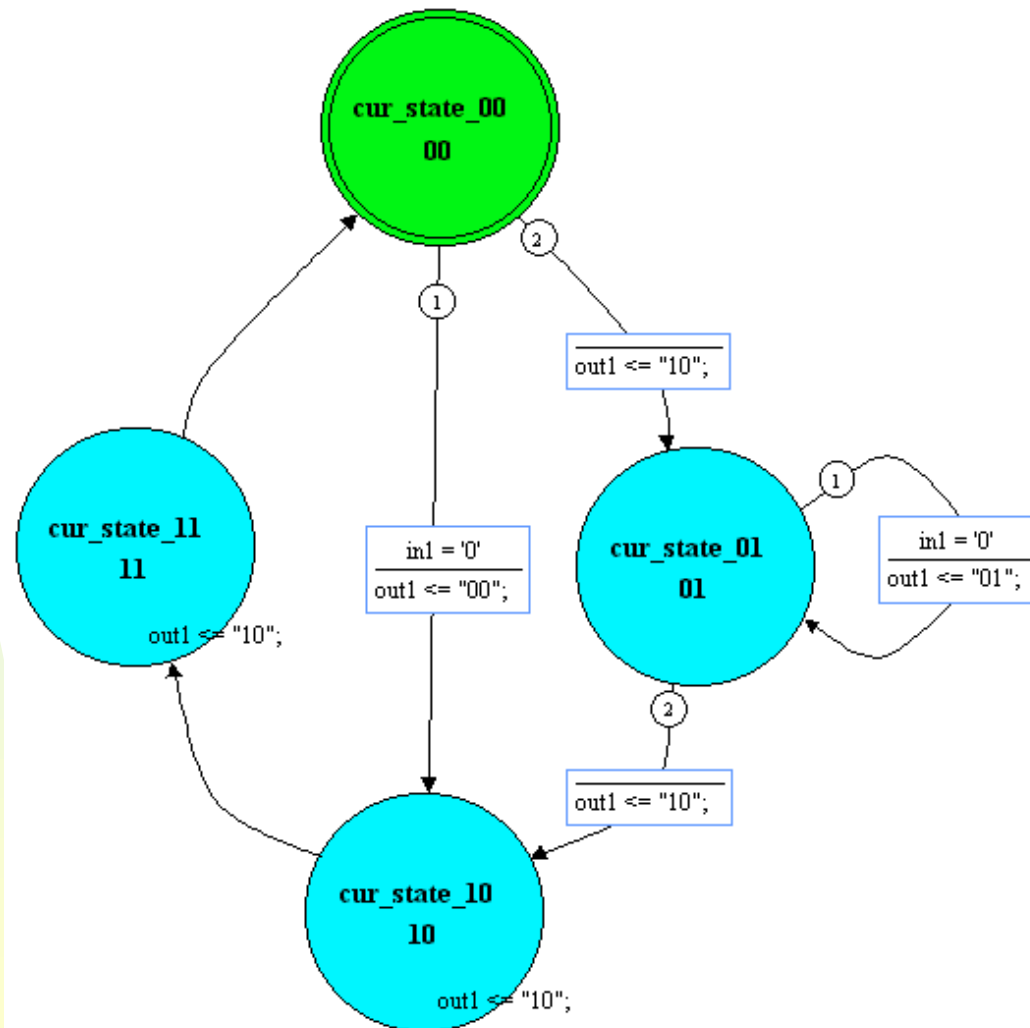
```
constant MODE_AND: std_logic_vector(1 downto 0) := "00";
constant MODE_OR: std_logic_vector(1 downto 0) := "01";
constant MODE_XOR: std_logic_vector(1 downto 0) := "10";
constant MODE_ADD: std_logic_vector(1 downto 0) := "11";
```

```
begin
  process(A, B, CIN, MODE)
    variable VSUM: std_logic_vector(3 downto 0);
    variable CY: std_logic;
  begin
    case MODE is
      when MODE_AND =>
        Q <= A and B;
        COUT <= '-';
      when MODE_OR =>
        Q <= A or B;
        COUT <= '-';
      when MODE_XOR =>
        Q <= A xor B;
        COUT <= '-';
      when others =>
        -- generate 4-bit adder
        CY := CIN;
        for I in 0 to 3 loop
          VSUM(I) := (A(I) xor B(I)) xor CY;
          CY := (A(I) and B(I)) or (CY and (A(I) or B(I)));
        end loop;
        Q <= VSUM;
        COUT <= CY;
      end case;
    end process;
  end BEHAVE;
```

Synthesized Circuit



Finite State Machine



Finite State Machine : Example

```

library IEEE;
use IEEE.Std_Logic_1164.all;

entity STATE_EX is
    port( IN1, CLOCK, RESET : in std_logic;
          OUT1 : out std_logic_vector(1 downto 0) );
end STATE_EX;

architecture STATE_EX_A of STATE_EX is

    signal CUR_STATE : std_logic_vector(1 downto 0);
    signal NEXT_STATE : std_logic_vector(1 downto 0);

begin
    process(CLOCK, RESET)
    begin
        if (CLOCK = '1' and CLOCK'event) then
            if (RESET = '0') then
                CUR_STATE <= "00";
            else
                CUR_STATE <= NEXT_STATE;
            end if;
        end if;
    end process;

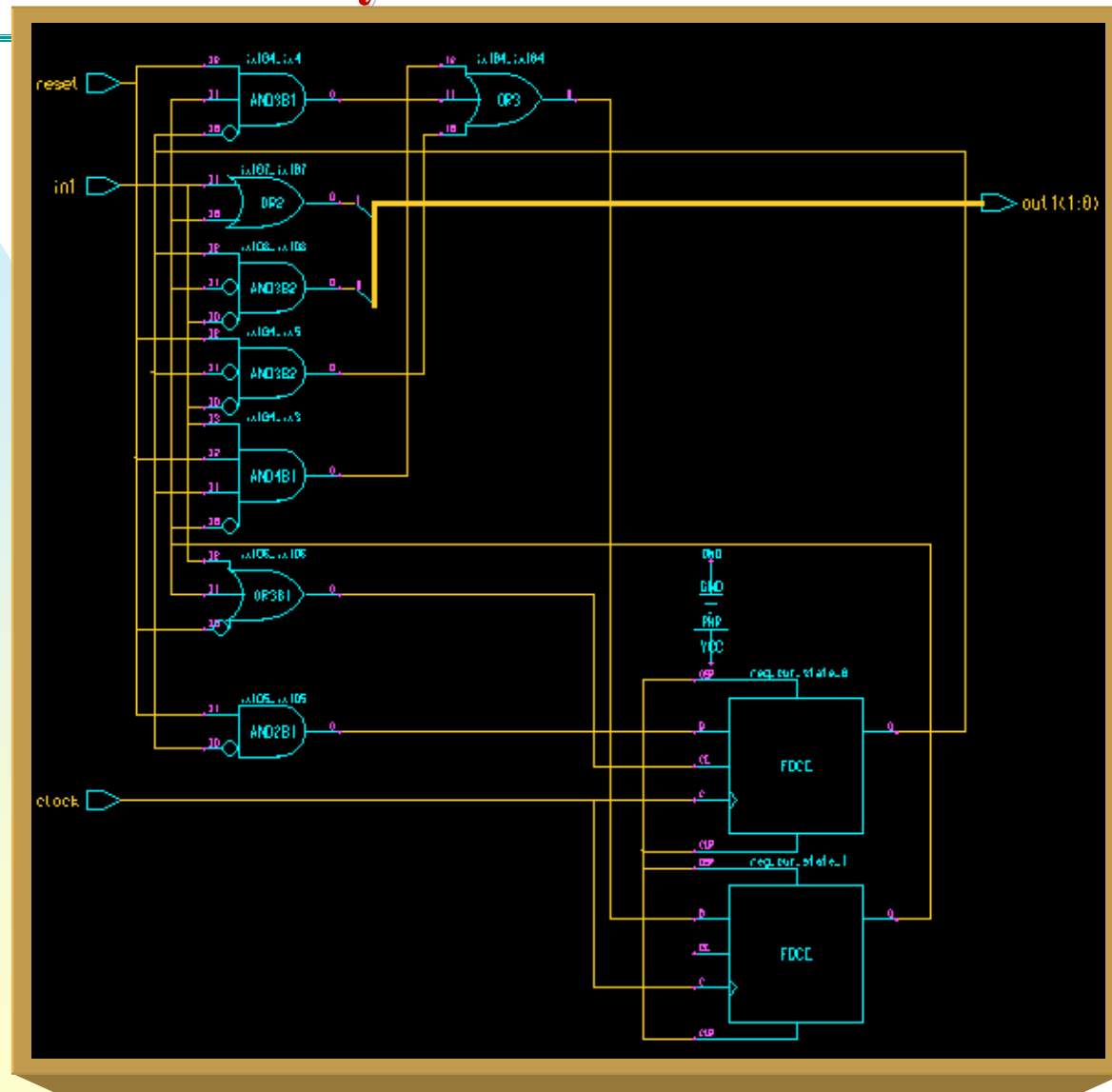
```

```

    process (IN1, CUR_STATE)
    begin
        case CUR_STATE is
            when "00" =>
                if (IN1 = '0') then
                    NEXT_STATE <= "10";
                    OUT1 <= "00";
                else
                    NEXT_STATE <= "01";
                    OUT1 <= "10";
                end if;
            when "01" =>
                if (IN1 = '0') then
                    NEXT_STATE <= CUR_STATE;
                    OUT1 <= "01";
                else
                    NEXT_STATE <= "10";
                    OUT1 <= "10";
                end if;
            when "10" =>
                NEXT_STATE <= "11";
                OUT1 <= "10";
            when "11" =>
                NEXT_STATE <= "00";
                OUT1 <= "10";
            when others => null;
        end case;
    end process;
end STATE_EX_A;

```

Synthesized Circuit



D-Latches

```

library IEEE;
use IEEE.Std_Logic_1164.all;

entity LATCH_4BIT is
  port(D : in std_logic_vector(3 downto 0);
       EN : in std_logic;
       Q : out std_logic_vector(3 downto 0));
end LATCH_4BIT;

architecture BEHAVE of LATCH_4BIT is

begin

  P1: process(D, EN)
  begin
    if (EN = '1') then
      Q <= D;
    end if;
  end process P1;

end BEHAVE;

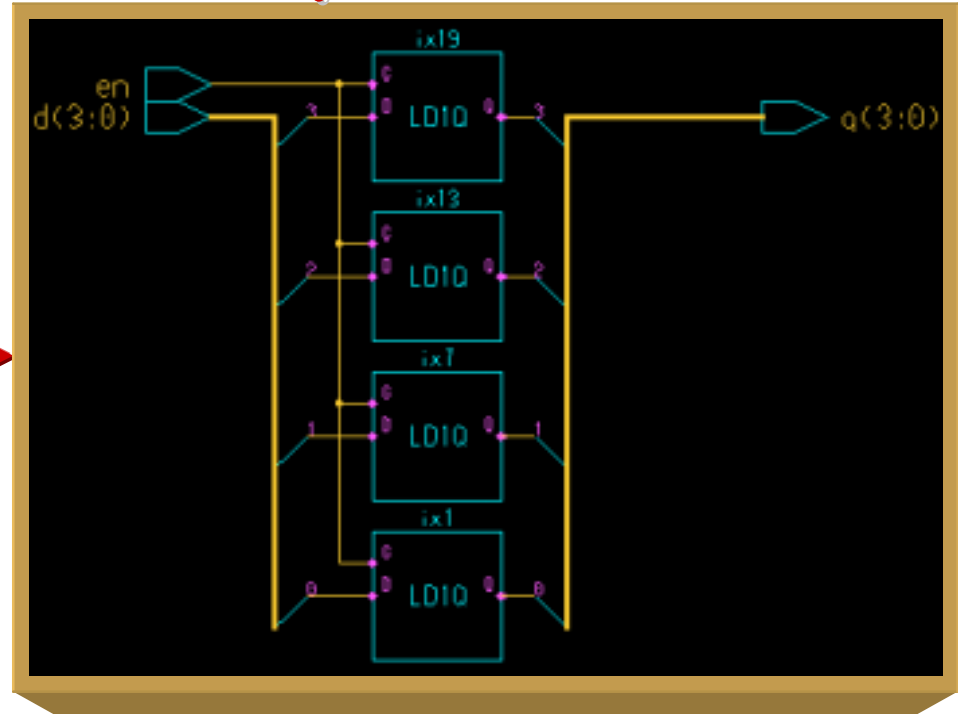
```

```

B1 : block (EN = '1')
begin
  Q <= guarded D;
end block ;

```

Synthesized Circuit



Asynchronous Reset Flip-Flop

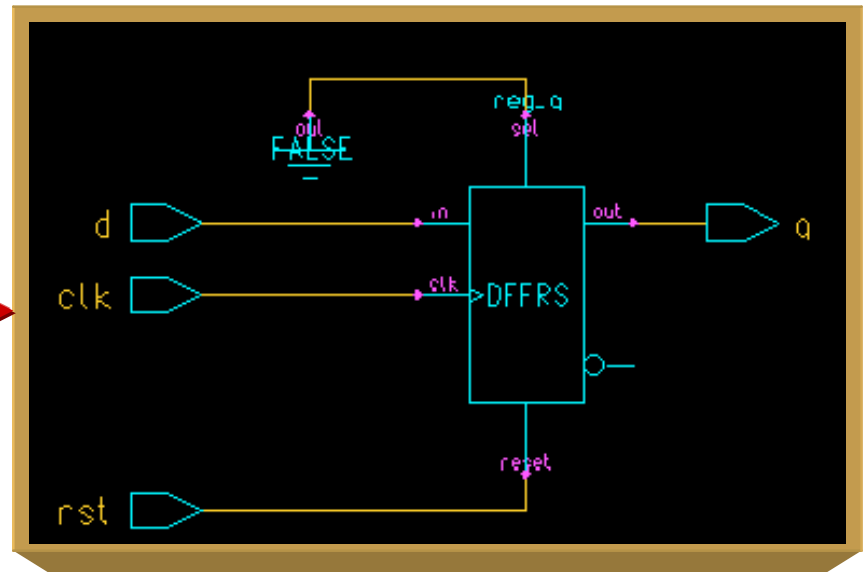
```

Library IEEE;
use IEEE.Std_logic_1164.all;

entity ASYNC_FLOP is
port ( D, CLK, RST : in std_ulogic;
      Q : out std_ulogic );
end ASYNC_FLOP;

architecture B of ASYNC_FLOP is
begin
  process (CLK, RST)
  begin
    if (RST = '1') then
      Q <= '0';
    elsif (CLK'event and CLK = '1') then
      Q <= D;
    end if;
  end process;
end B;
  
```

Synthesized Circuit



Synchronous Reset Flip-Flop

```
Library IEEE;
use IEEE.Std_logic_1164.all;
```

```
entity SYNC_FLOP is
port ( D, CLK, RST : in std_ulogic;
      Q             : out std_ulogic );
end SYNC_FLOP;
```

```
architecture B of SYNC_FLOP is
begin
```

```
    process
    begin
```

```
        wait until (CLK' event and CLK = '1');
        if (RST = '1') then
            Q <= '0';
```

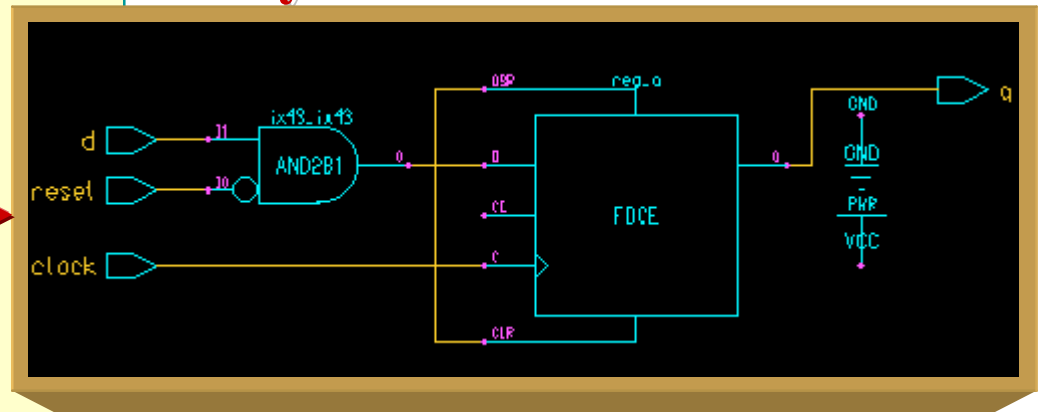
```
        else
            Q <= D;
```

```
        end if;
```

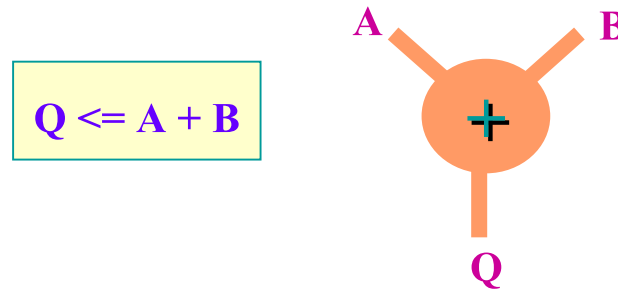
```
    end process;
```

```
end B;
```

Synthesized Circuit



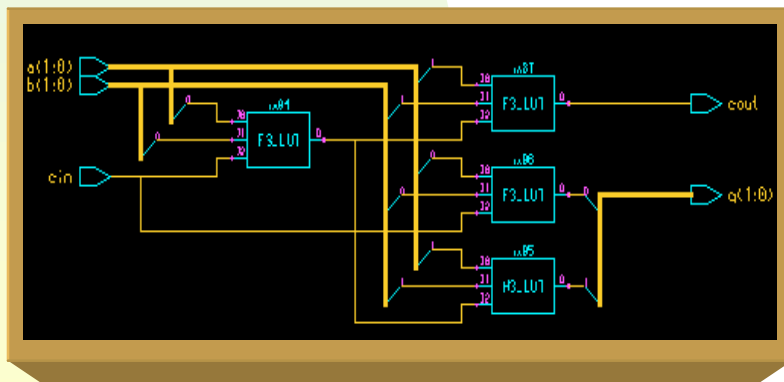
Synthesis Adder problem ?



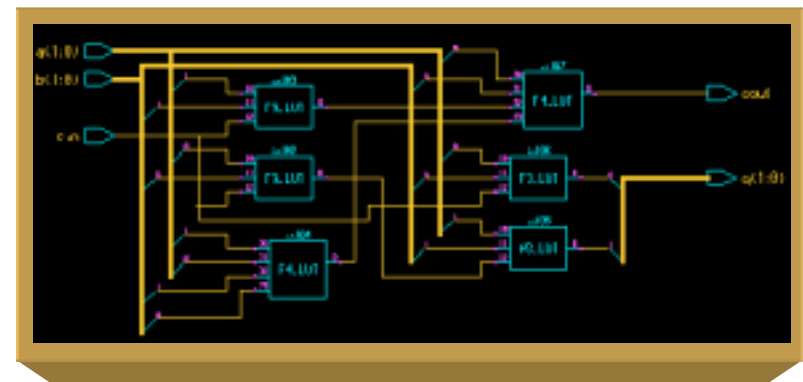
- Carry-look-ahead circuits are faster **but** larger than ripple-carry circuits

Synthesized Circuit

Ripple-carry Adder



Carry-look-ahead Adder



CASE style synthesis problem ?

```

library ieee;
use ieee.std_logic_1164.all;

entity case_latch is
    port (in1, in2: in std_logic;
          out1, out2: out std_logic);
end case_latch;

```

```

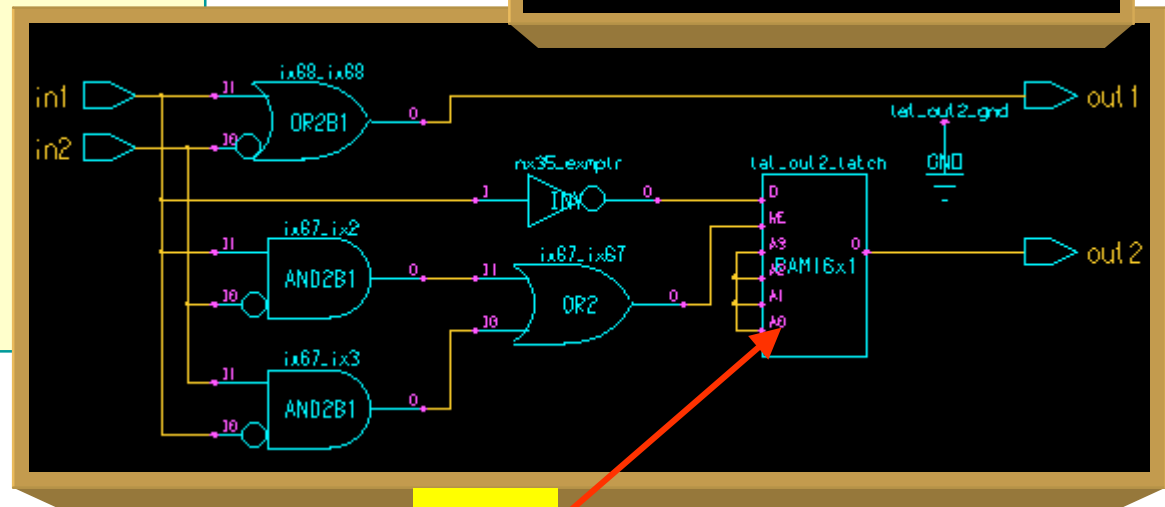
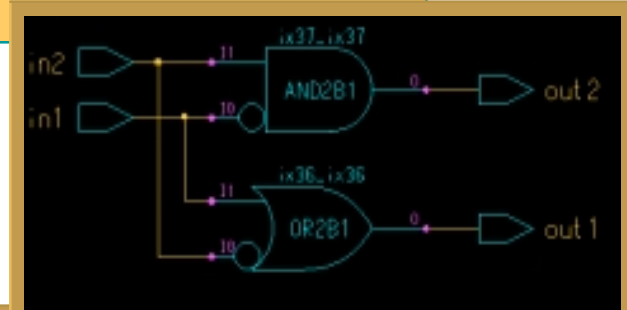
architecture case_latch_a of case_latch is
    signal b: std_logic_vector(1 downto 0);
begin
    process (b)
    begin
        case b is
            when "01" => out1 <= '0';
                       out2 <= '1';
            when "10" => out1 <= '1';
                       out2 <= '0';
            when others => out1 <= '1';
                       out2 <= '1';
        end case;
    end process;
    b <= in1 & in2;
end case_latch_a;

```

```

case b is
    when "01" => out1 <= '0';
                out2 <= '1';
    when "10" => out1 <= '1';
                out2 <= '0';
    when others => out1 <= '1';
                out2 <= '0';
end case;

```



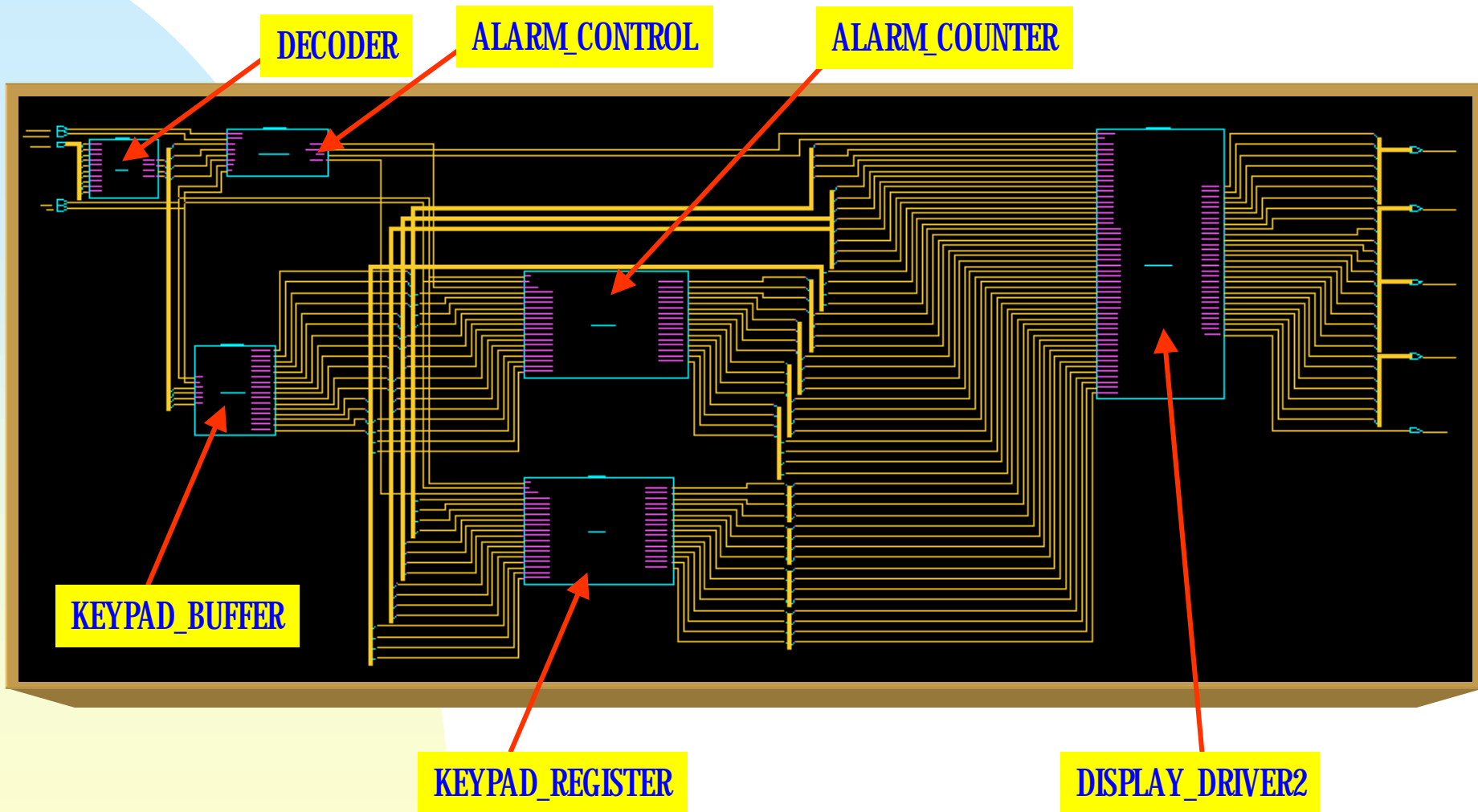
Latch

THE END



SYNTHESIS SOLUTIONS

ALARM CLOCK



Cell: ALARM_CLOCK View: STRUC Library: work

Total accumulated area:
 Number of CLB Flip Flops: 57
 Number of CY4: 12
 Number of FG Function Generators: 139
 Number of H Function Generators: 40
 Number of Packed CLBs: 70
 Number of STARTUP: 1

Number of ports: 43
 Number of nets: 362
 Number of instances: 262
 Number of references to this view: 0

Cell	Library	References	Total Area
CY4	xi4e	12x1	12 CY4
CY4_17	xi4e	1x1	1 CY4_17
CY4_18	xi4e	4x1	4 CY4_18
CY4_19	xi4e	3x1	3 CY4_19
CY4_21	xi4e	1x1	1 CY4_21
CY4_42	xi4e	3x1	3 CY4_42
F2_LUT	xi4e	9x1	9 FG Function Generators
F3_LUT	xi4e	18x1	18 FG Function Generators
F4_LUT	xi4e	112x1	112 FG Function Generators
EDCE	xi4e	56x1	56 CLB Flip Flops
EDPE	xi4e	1x1	1 CLB Flip Flops
GND	xi4e	1x1	1 GND
H2_LUT	xi4e	4x1	4 H Function Generators
H3_LUT	xi4e	36x1	36 H Function Generators
STARTUP	xi4e	1x1	1 STARTUP

Synthesis report

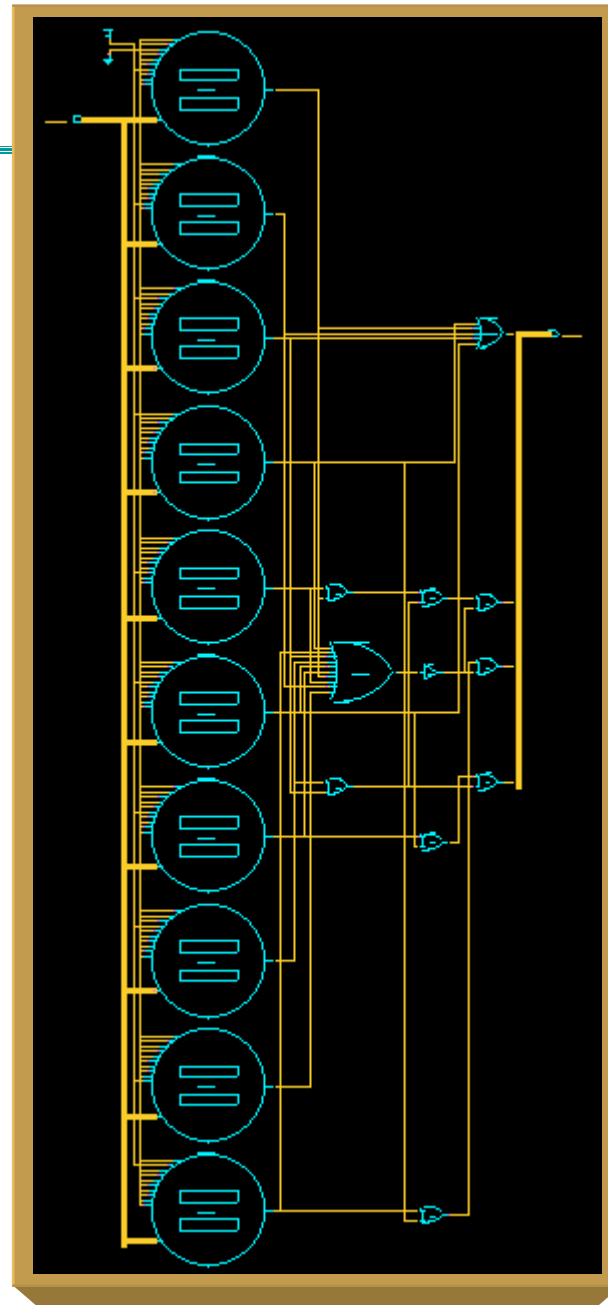
Device Utilization for 4013ePQ160

Resource	Used	Avail	Utilization
I/Os	43	129	33.33%
FG Function Generators	139	1152	12.07%
H Function Generators	40	576	6.94%
CLB Flip Flops	57	1152	4.95%

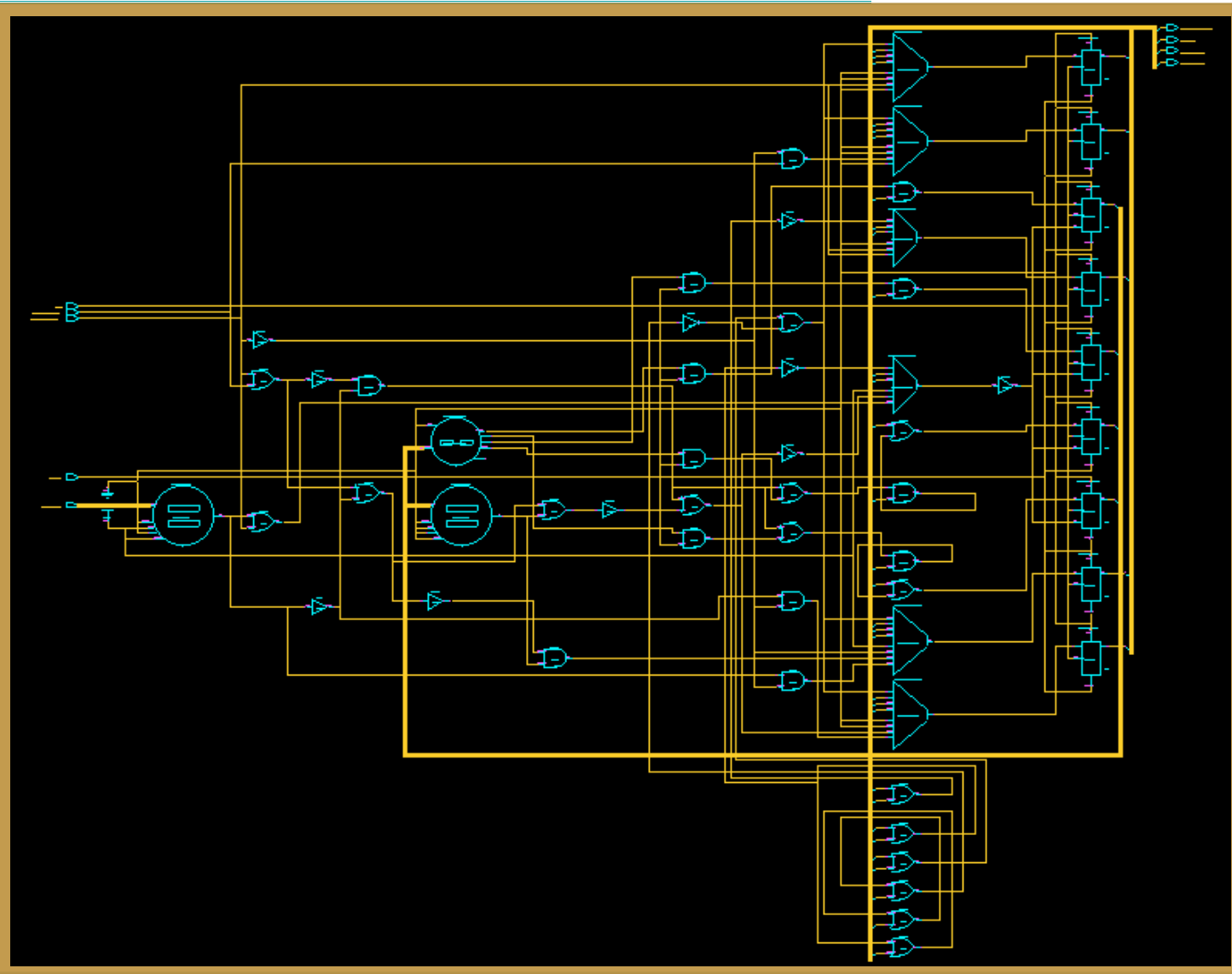
Clock Frequency Report

Clock	: Frequency
clk	: 18.2 MHz

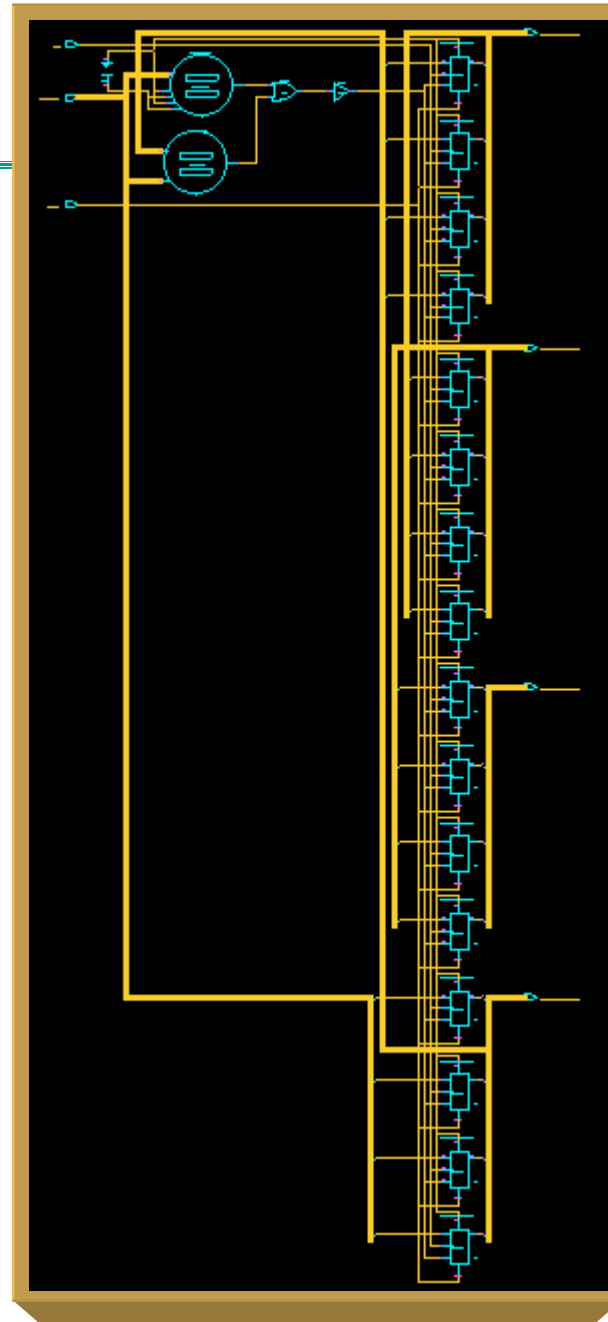
DECODER



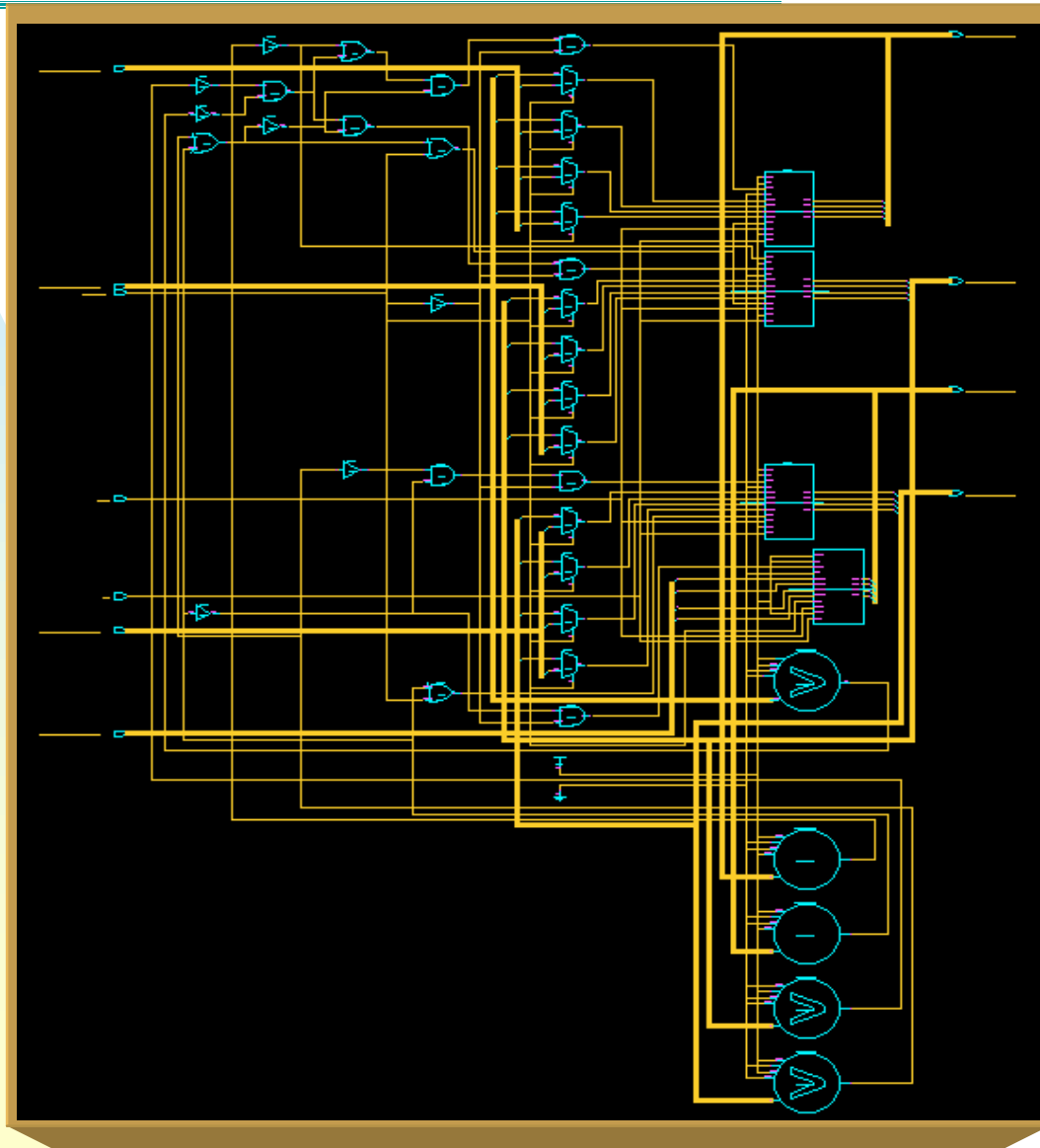
ALARM_CONTROL



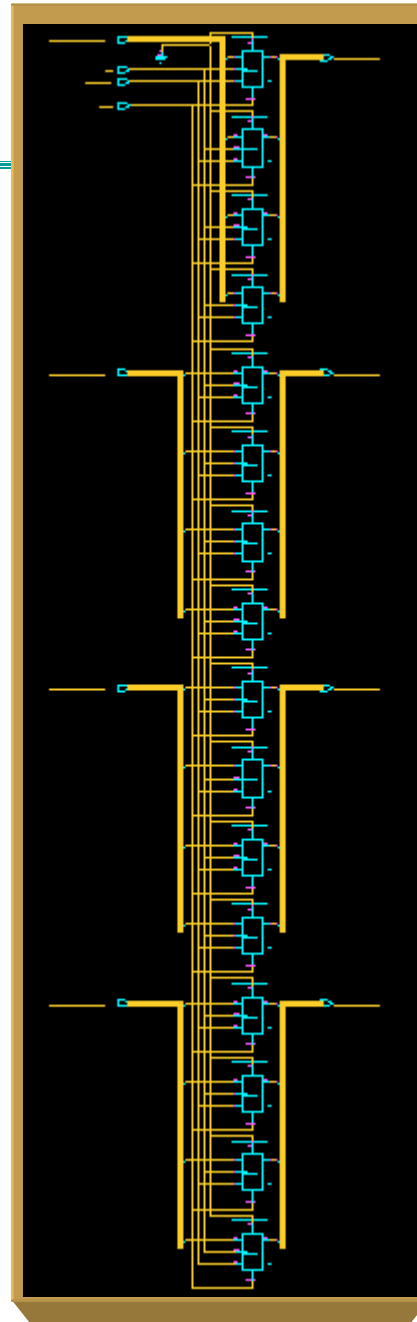
KEYPAD_BUFFER



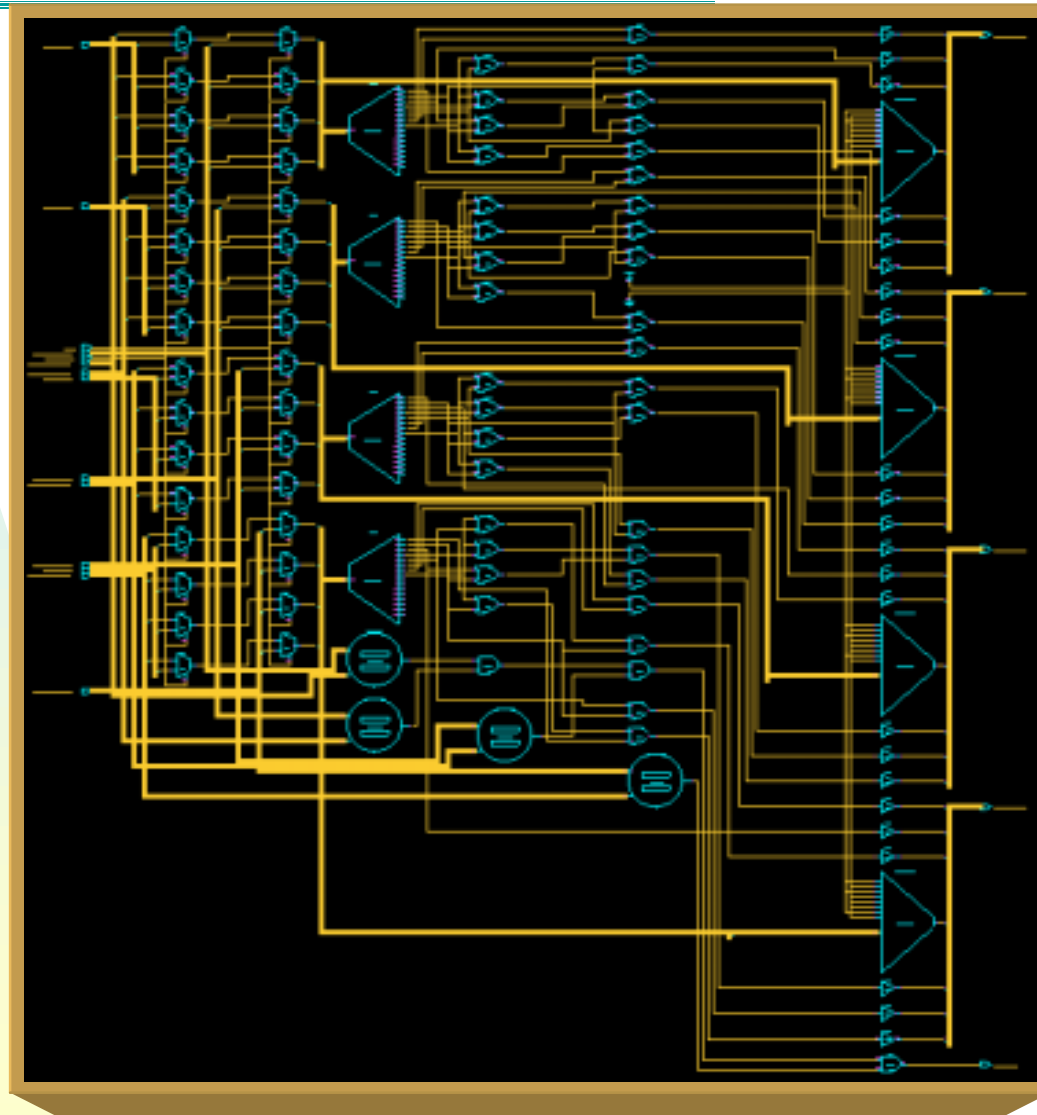
ALARM_COUNTER



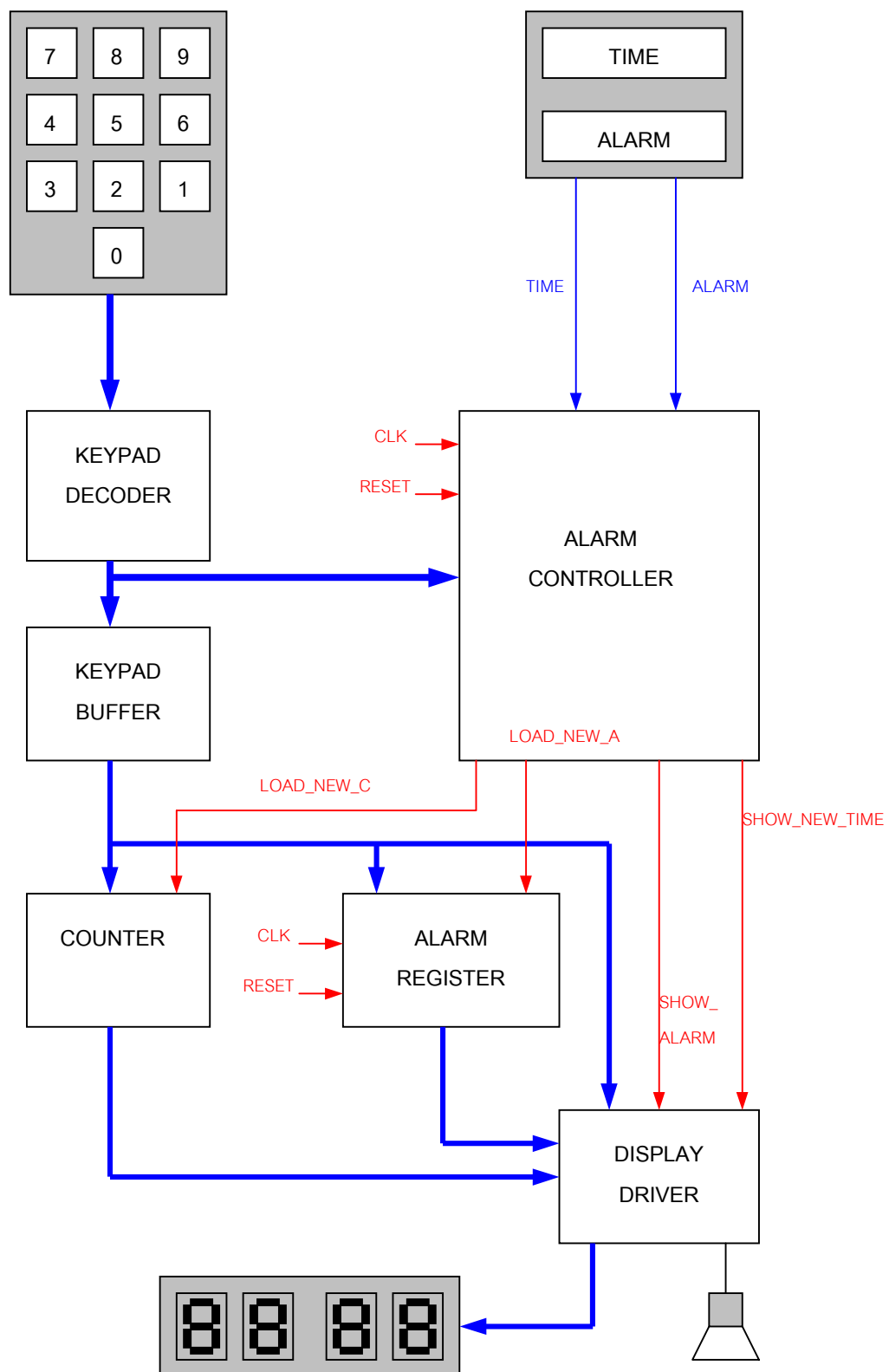
ALARM_REGISTER



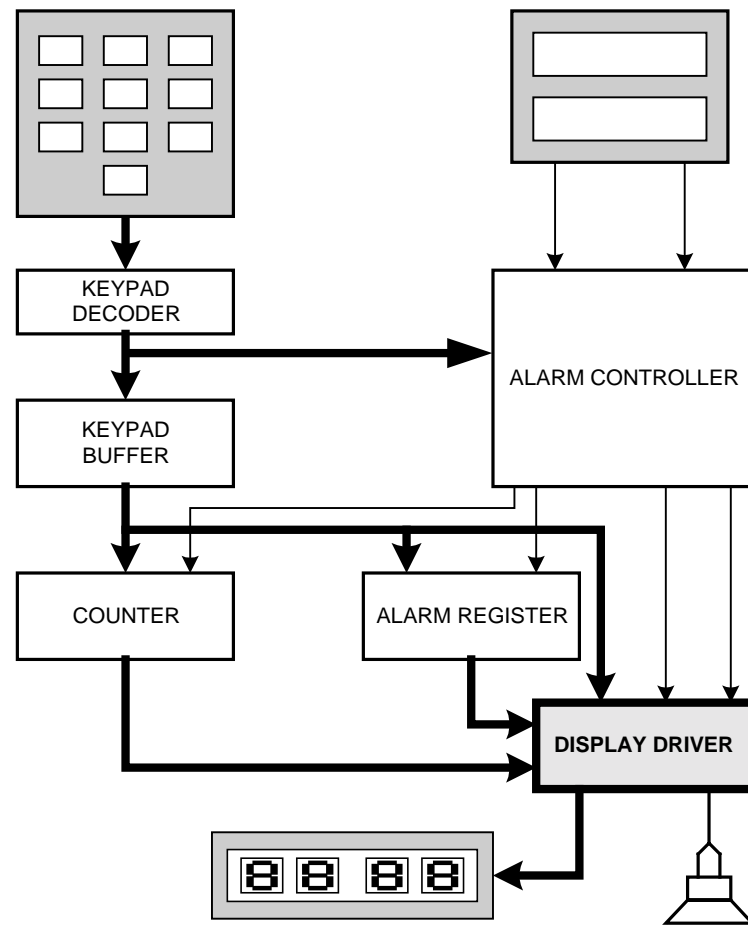
DISPLAY_DRIVER2



Workshop : Design DIGITAL CLOCK Module



Workshop 1 : Writing your first VHDL



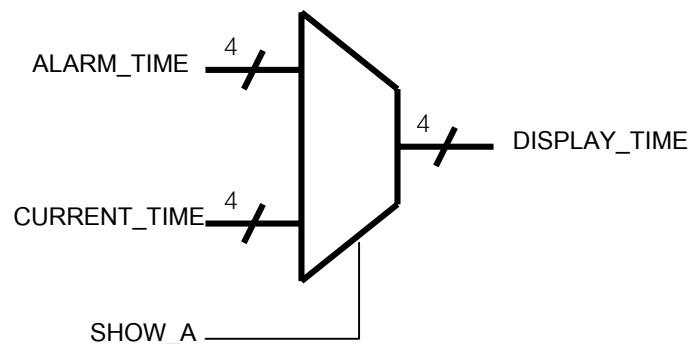
รูปที่ 1 แสดงโครงสร้างวงจรวางรนาพิกาดิจิตอล

จุดประสงค์

- ศึกษาและทดลองการเขียน VHDL อย่างง่ายและการใช้โปรแกรม ModelSim ในการซิมมูลเลชั่น
- ศึกษาการสร้างวงจร combination อย่างง่าย

ขั้นตอน

1. ออกแบบวงจร Multiplexer เพื่อใช้ในการเลือกข้อมูลที่จะแสดงผลที่ 7 segment ว่าจะเป็นเวลาปัจจุบัน (CURRENT_TIME) หรือ เวลาในการตั้งปลุก (ALARM_TIME) โดยเขียน VHDL code เพื่ออธิบายวงจрдังรูปที่ 2



รูปที่ 2 โครงสร้าง Multiplexer

2. โดยตั้งชื่อไฟล์ดังนี้ DISMUX.VHD โดยกำหนดให้ใช้ type ของ Input และ output เป็น **std_ulogic** และ **std_ulogic_vector** ซึ่งเป็น type ที่อยู่ใน Library IEEE.Std_Logic_1164 โดยการเรียกใช้จะต้องประกาศ Library ดังนี้

```
Library IEEE;
use IEEE.Std_Logic_1164.all ;
```

โดยกำหนดให้ตั้งชื่อ Design Unit ต่างๆ ดังนี้

Entity: DISPLAY_MUX

Architecture: RTL

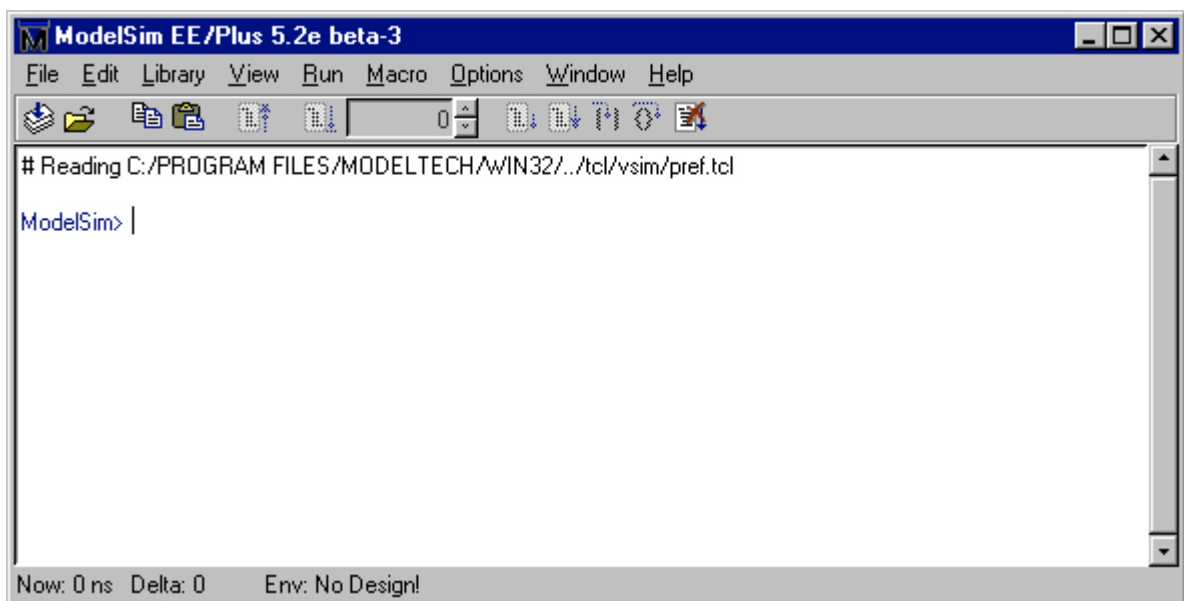
Coding Style: ใช้คำสั่ง Process และ If / then / else ในการสร้าง

Function:

- ALARM_TIME , DISPLAY_TIME และ CURRENT_TIME มีขนาด 4 bit
- SHOW_A มีขนาดข้อมูล 1 bit
- DISPLAY_TIME จะมีข้อมูลเท่ากับ ALARM_TIME เมื่อ SHOW_A = '1'
- DISPLAY_TIME จะมีข้อมูลเท่ากับ CURRENT_TIME เมื่อ SHOW_A = '0'

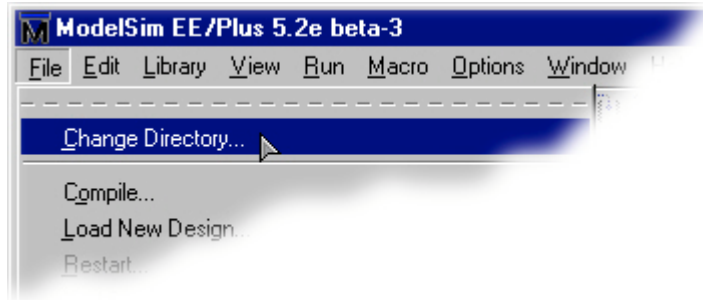
3. ทำความเข้าใจ Test bench file ชื่อ T_DISMUX.VHD
4. ทำการ Compile และ simulate Design ที่ออกแบบโดยใช้โปรแกรม ModelSim ดังมีขั้นตอนต่างๆ ดังนี้

4.1 เปิดโปรแกรม ModelSim

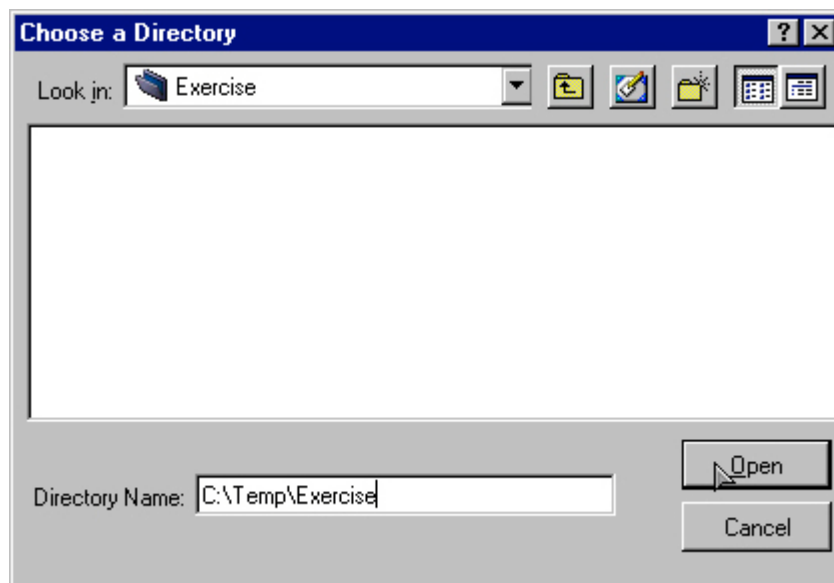


รูปที่ 3 หน้าต่างหลักของโปรแกรม ModelSim

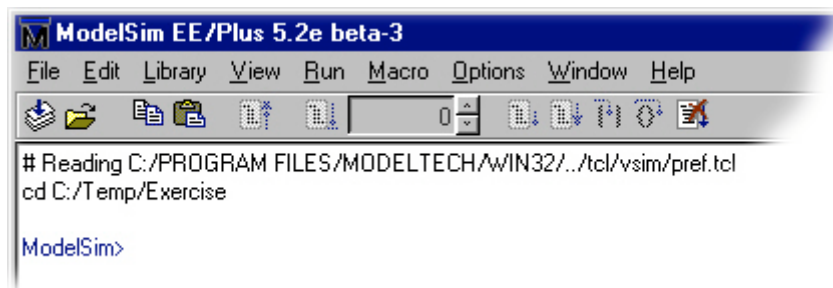
4.2 กำหนด Directory ที่จะใช้งาน โดยเลือก File → Change Directory แล้วเลือก Directory ที่ใช้งานโดยมักจะเป็น Directory ที่มี VHDL file ที่ต้องการทำการ Compile และ Simulate



รูปที่ 4 การกำหนด Directory ที่ต้องการใช้งาน

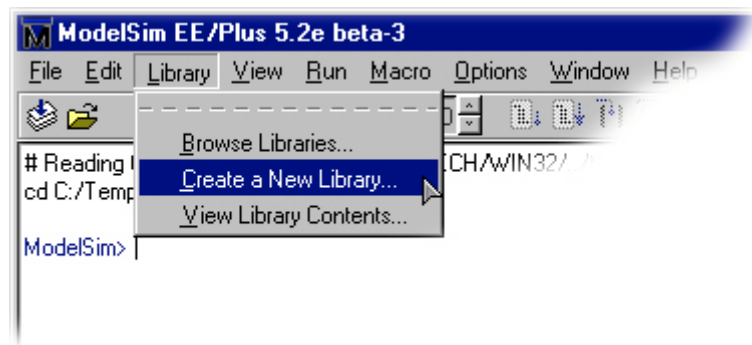


รูปที่ 5 หน้าต่างแสดง Directory ที่ต้องการทำงาน

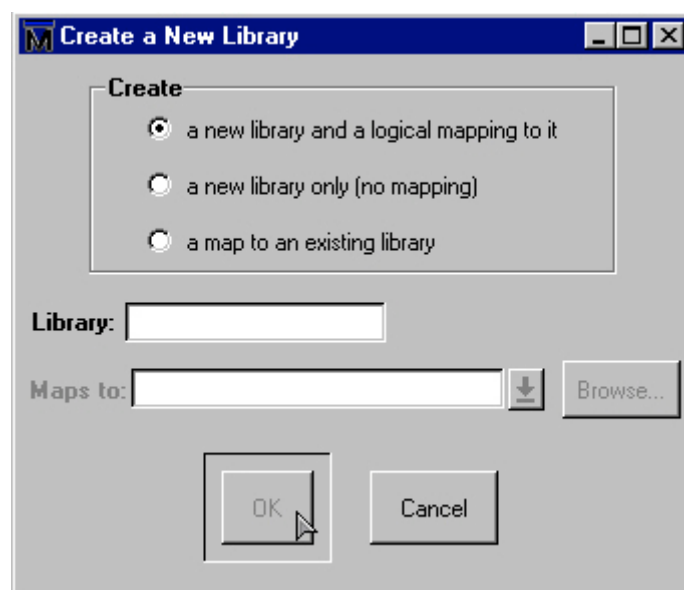


รูปที่ 6 เมื่อกำหนด Directory ที่ต้องการทำงานเรียบร้อยแล้ว

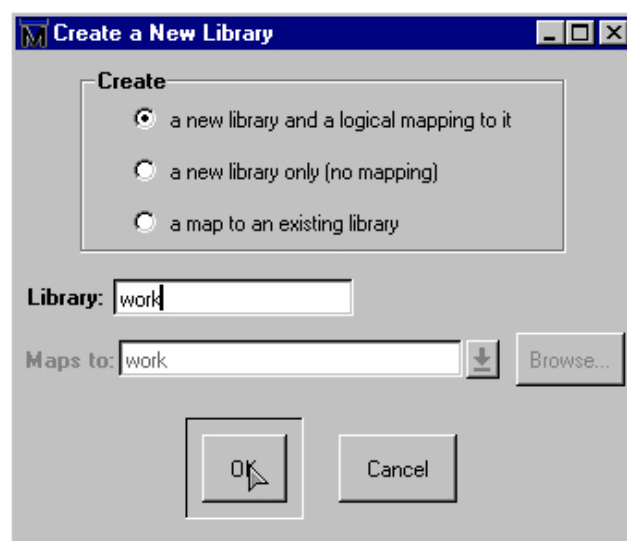
4.3 หากเป็นการเข้ามาใช้งาน Directory นี้ครั้งแรกต้องทำการสร้าง WORK library ก่อนโดยเลือก Library → Create a New Library จากนั้นให้ใส่คำว่า **WORK** ลงไป



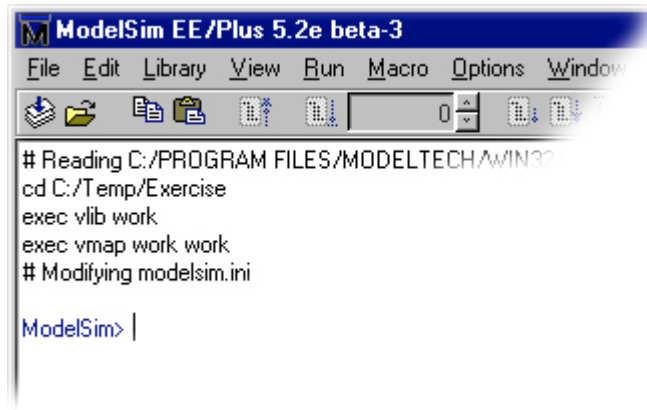
รูปที่ 7 การสร้าง Library ใหม่



รูปที่ 8 หน้าต่างกำหนดรูปแบบของ Library ที่ต้องการสร้าง

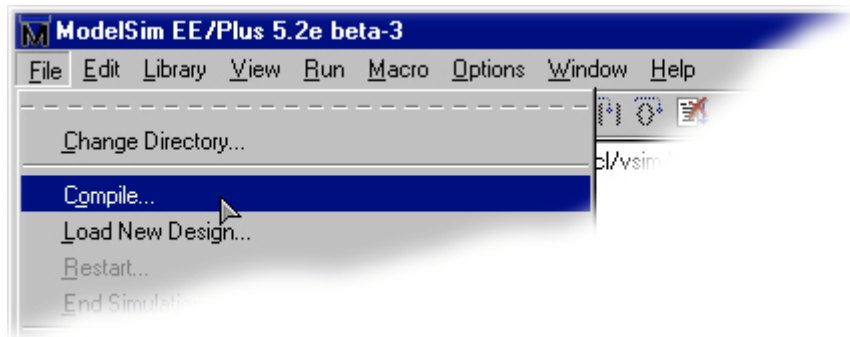


รูปที่ 9 การสร้าง Library ใหม่ที่ชื่อว่า work

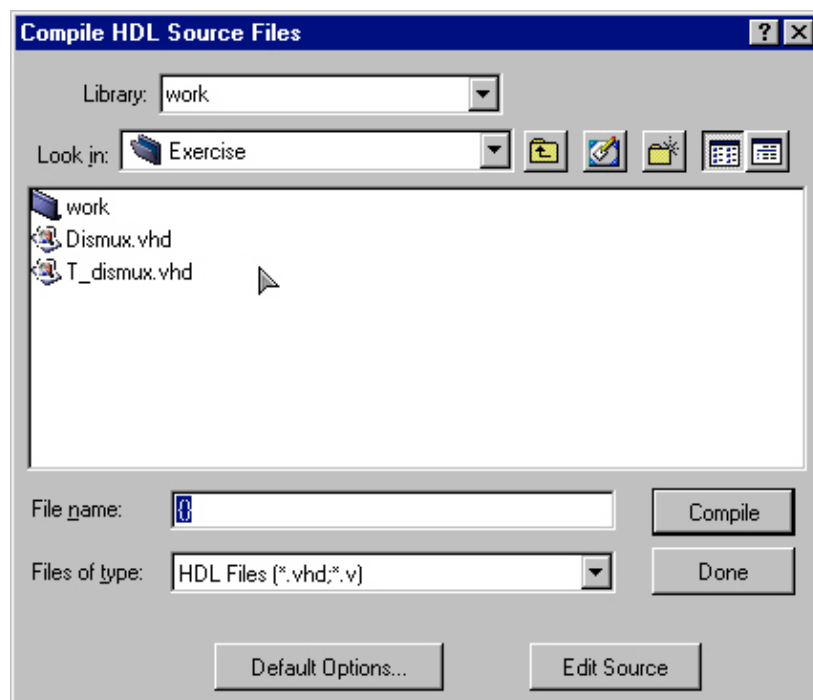


รูปที่ 10 หลังจากสร้าง Library ใหม่ที่ชื่อว่า work เรียบร้อยแล้ว

4.4 ทำการคอมไพล์ไฟล์โดยเลือกจาก File → Compile

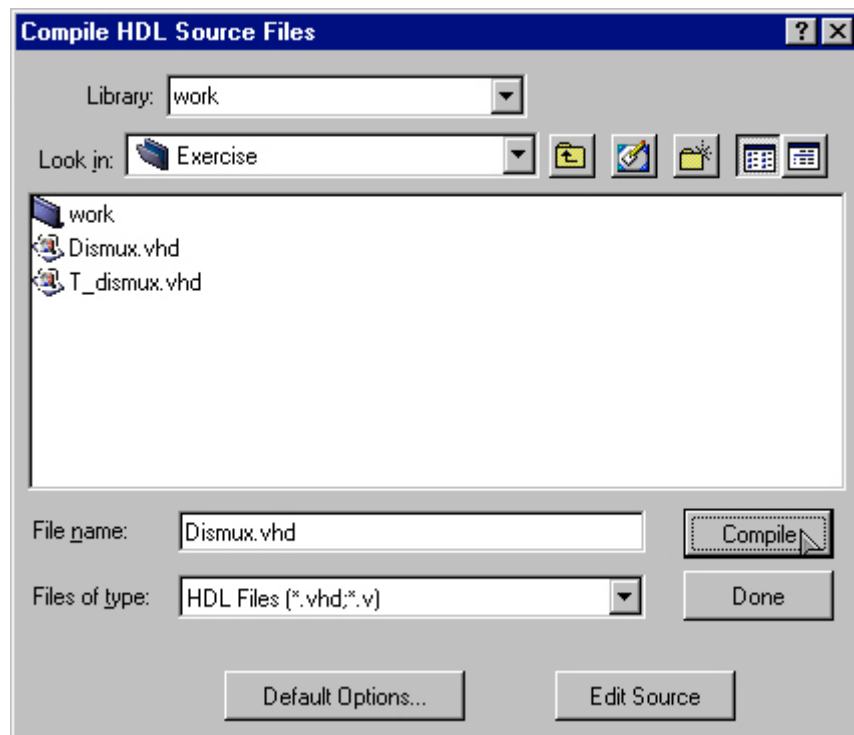


รูปที่ 11 การคอมไพล์



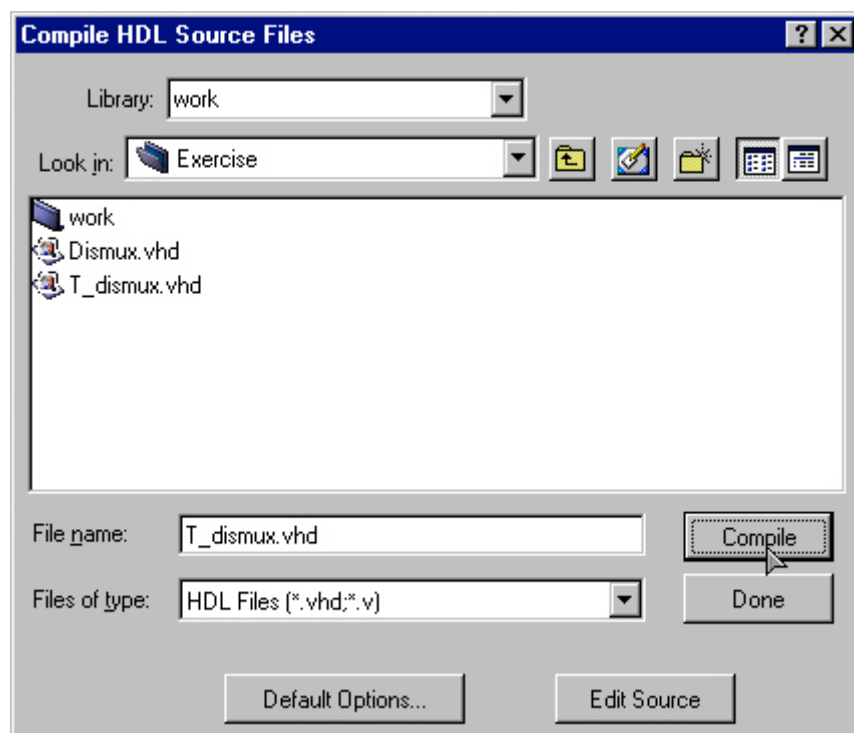
รูปที่ 12 หน้าต่างสำหรับเลือกไฟล์ที่ต้องการคอมไพล์

4.5 คอมไพล์ไฟล์ที่ชื่อว่า DISMUX.VHD ซึ่งเป็นไฟล์ข้อมูลที่ออกแบบไว้

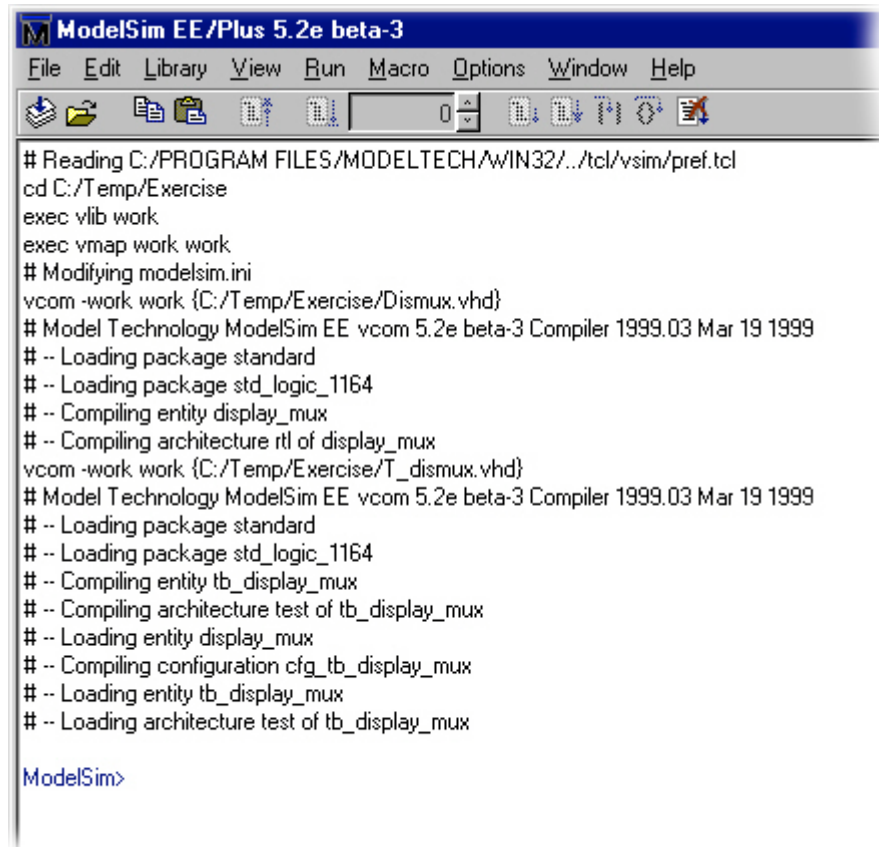


รูปที่ 13 การคอมไพล์ไฟล์ DISMUX.VHD

4.6 คอมไพล์ไฟล์ T_DISMUX.VHD เป็น Test bench ไฟล์ทำหน้าที่สร้างสัญญาณป้อนให้กับ DISMUX.VHD เพื่อทดสอบการทำงาน

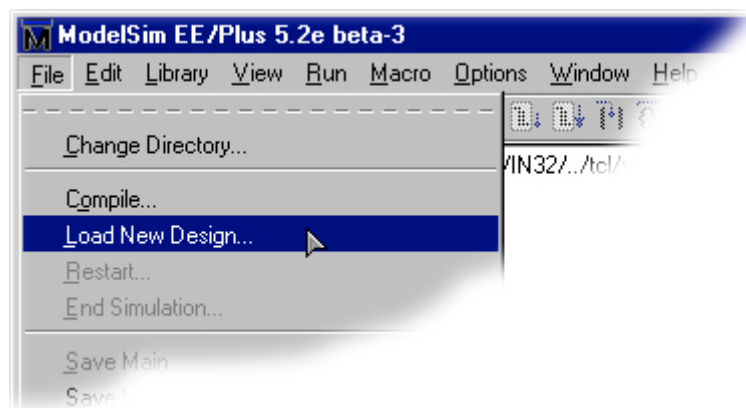


รูปที่ 14 การคอมไพล์ไฟล์ T_DISMUX.VHD



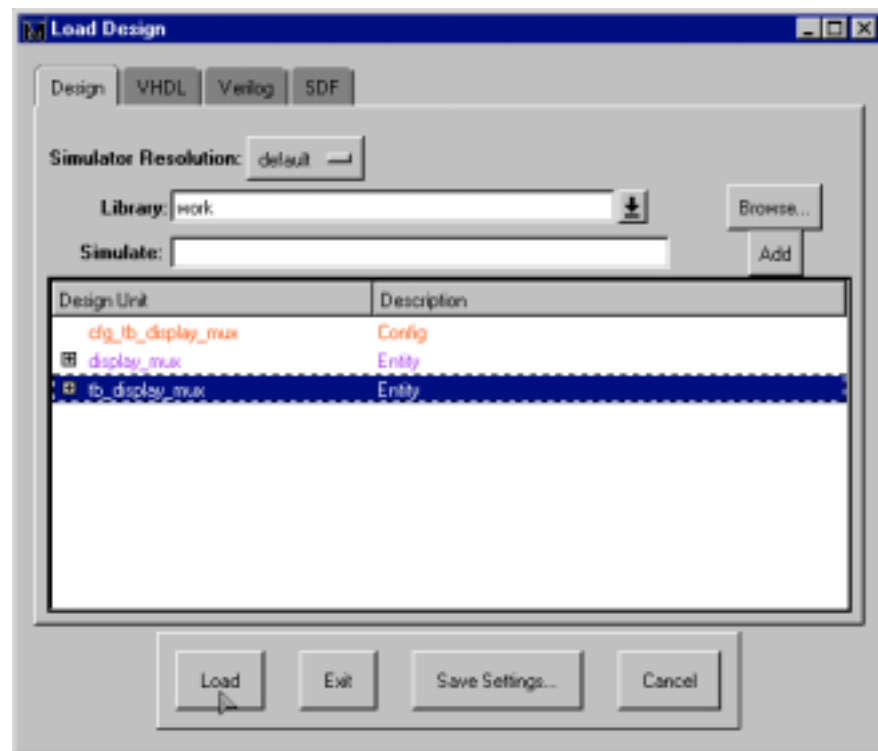
รูปที่ 15 ผลการคอมไพล์ไฟล์ทั้ง 2

4.7 ทำ Simulation โดยคลิกที่ File → Load New Design



รูปที่ 16 การ Load New Design เพื่อใช้ในการ Simulate

4.8 Load tb_display_mux เพื่อใช้ในการ Simulate

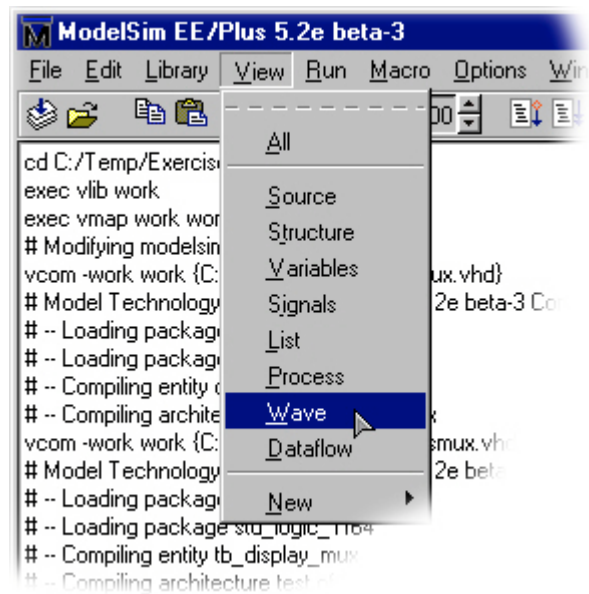


รูปที่ 17 การโหลด tb_display_mux

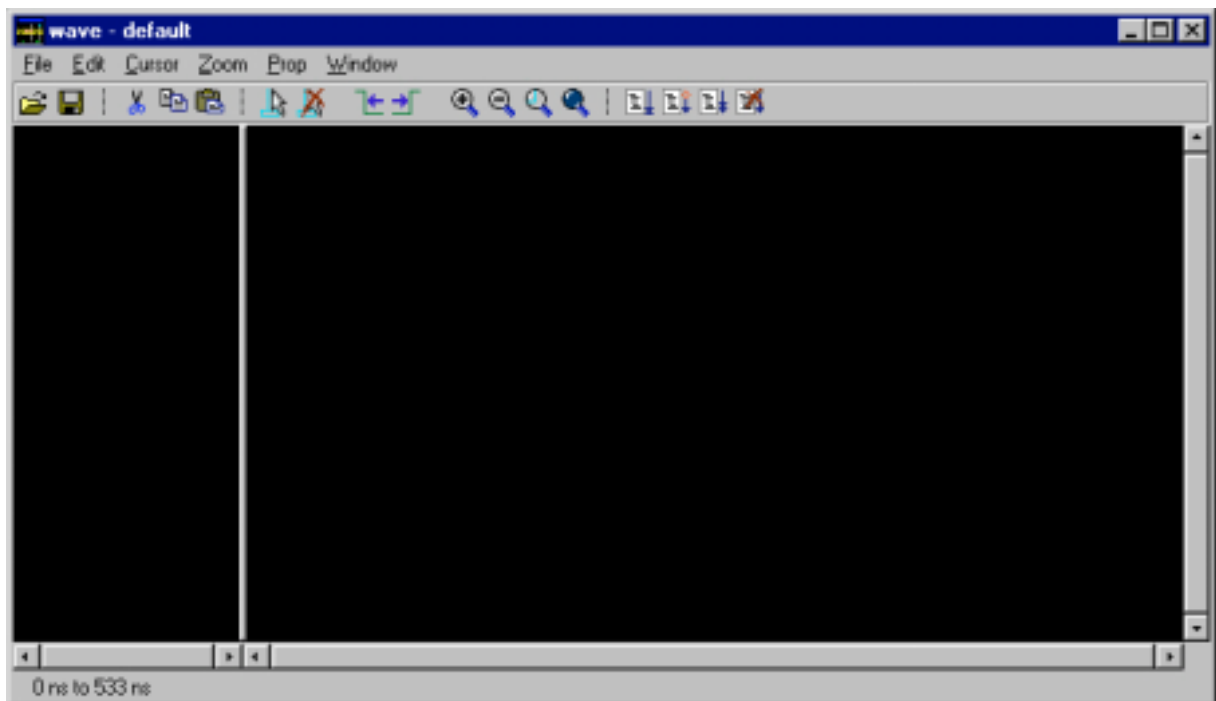


รูปที่ 18 ผลการ Load tb_display_mux

4.9 เปิดหน้าต่าง wave เพื่อดู Waveform ของสัญญาณภายในของวงจรที่ออกแบบ

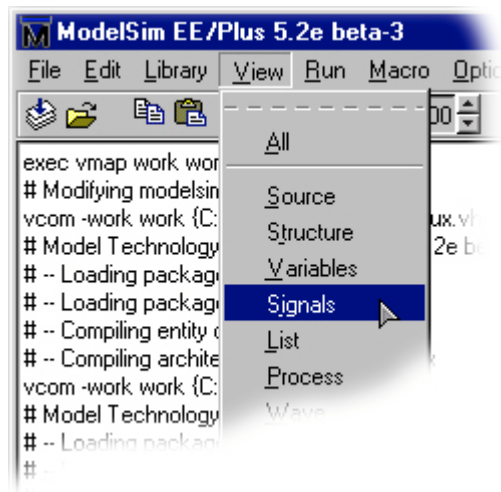


รูปที่ 19 การเปิดหน้าต่าง wave

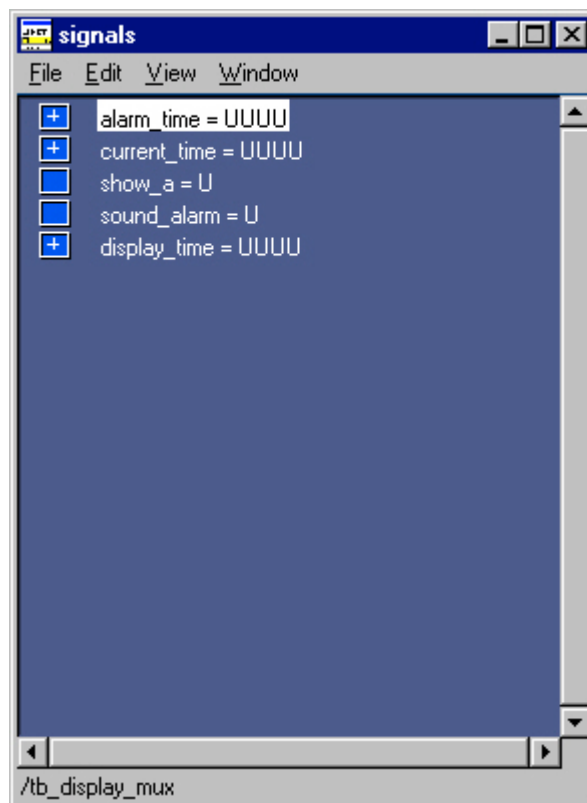


รูปที่ 20 หน้าต่าง wave

4.10 การเปิดหน้าต่าง Signal เพื่อทำการ Add signal ที่ต้องการดู Waveform ลงในหน้าต่าง Wave

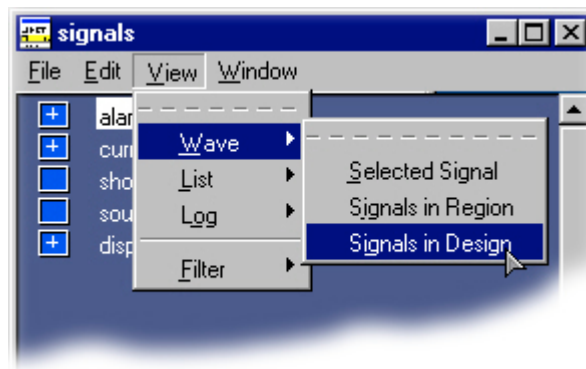


รูปที่ 21 การเปิดหน้าต่าง Signal

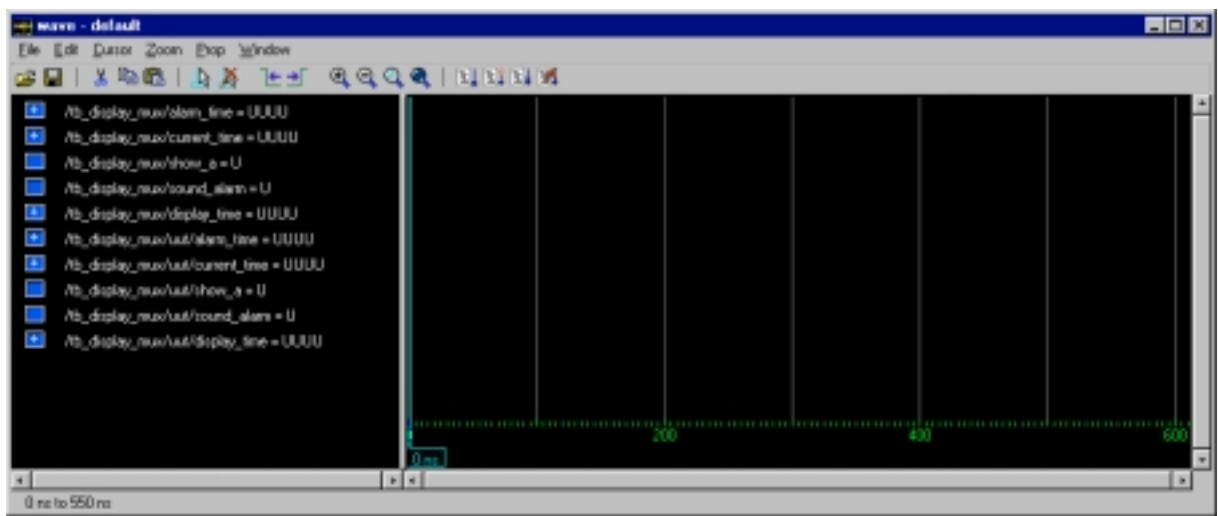


รูปที่ 22 หน้าต่าง Signal

4.11 เลือก View → Wave → Signals in Design เพื่อแสดงสัญญาณในการจำลองการทำงาน

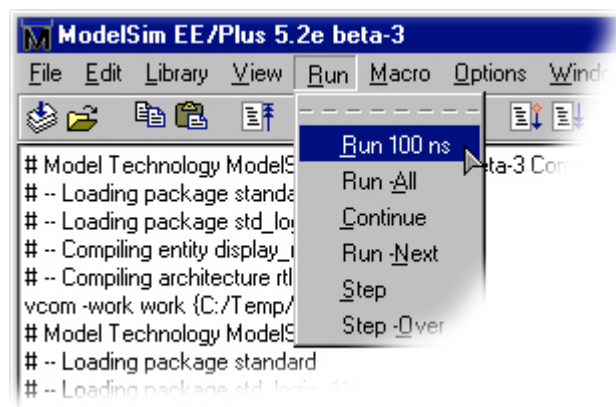


รูปที่ 23 การแสดงสัญญาณในการจำลองการทำงาน

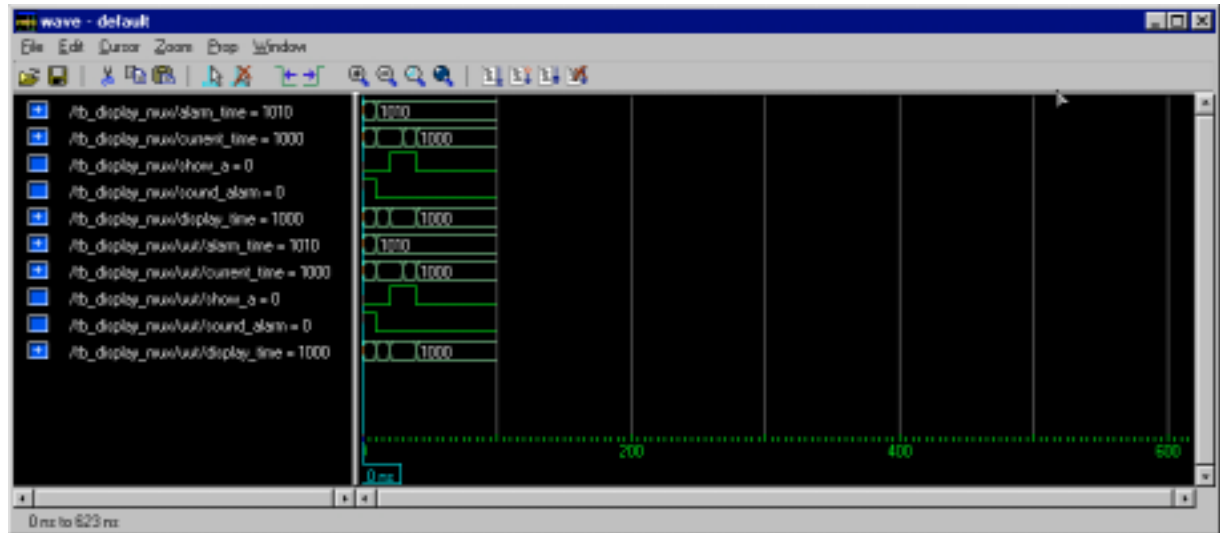


รูปที่ 24 หน้าต่าง Wave เมื่อกำหนดให้มีการแสดงสัญญาณในการจำลองการทำงาน

4.12 เลือก Run → Run 100 ns เพื่อสังเกตผลการทำงาน แล้ว แก้ไข VHDL code หากพบข้อผิดพลาด

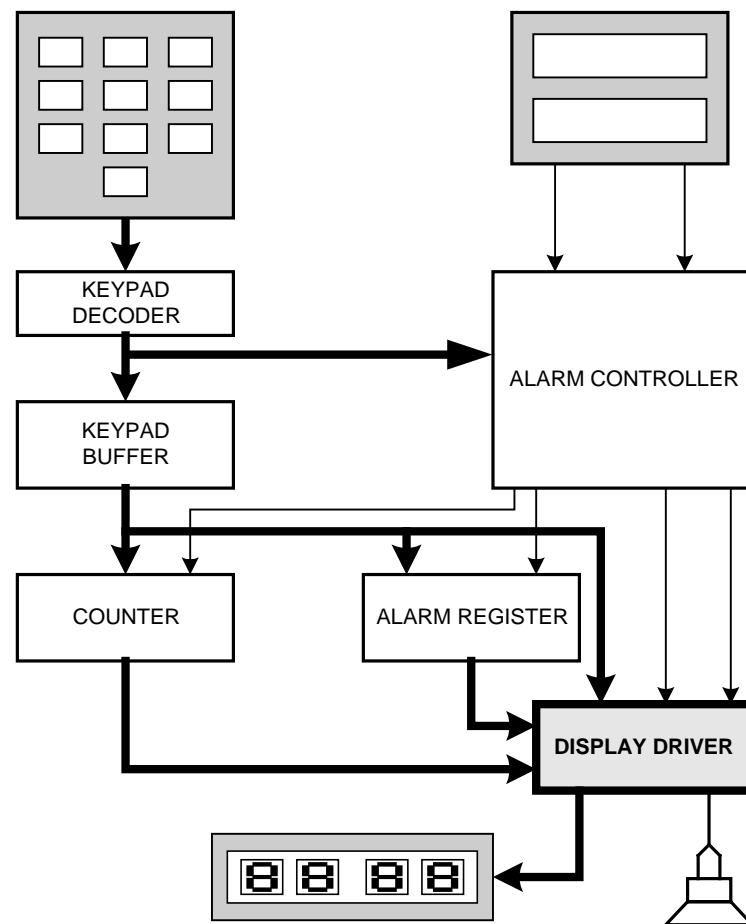


รูปที่ 25 การเลือก Run เพื่อตรวจสอบการทำงาน



รูปที่ 26 หน้าต่างแสดงผลการ Run

Workshop 2 : Adding the Alarm Signal

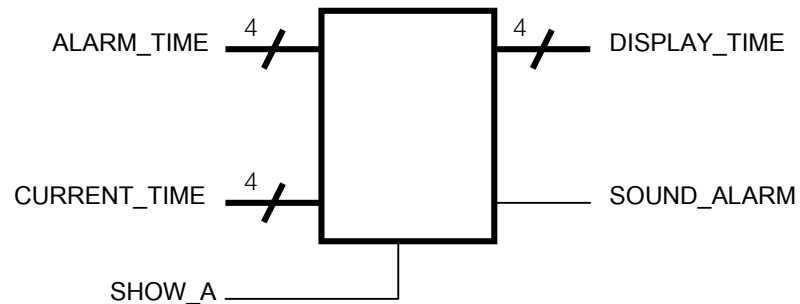


จุดประสงค์

- เพื่อสร้างสัญญาณในการนำไปกระตุ้นวงจรส่งเสียงปลุกเมื่อถึงเวลาที่กำหนดไว้

ขั้นตอน

1. นำไฟล์ **DISMUX.VHD** ที่สร้างใน Lab 1 มาแก้ไข เพิ่ม output port ชื่อ SOUND_ALARM ขนาด 1 bit โดยจะมีค่าเท่ากับ '1' เมื่อ ALARM_TIME มีค่าเท่ากับ CURRENT_TIME และมีค่าเท่ากับ '0' เมื่อข้อมูลของ ALARM_TIME และ CURRENT_TIME ทั้ง 2 สัญญาณไม่เท่ากัน

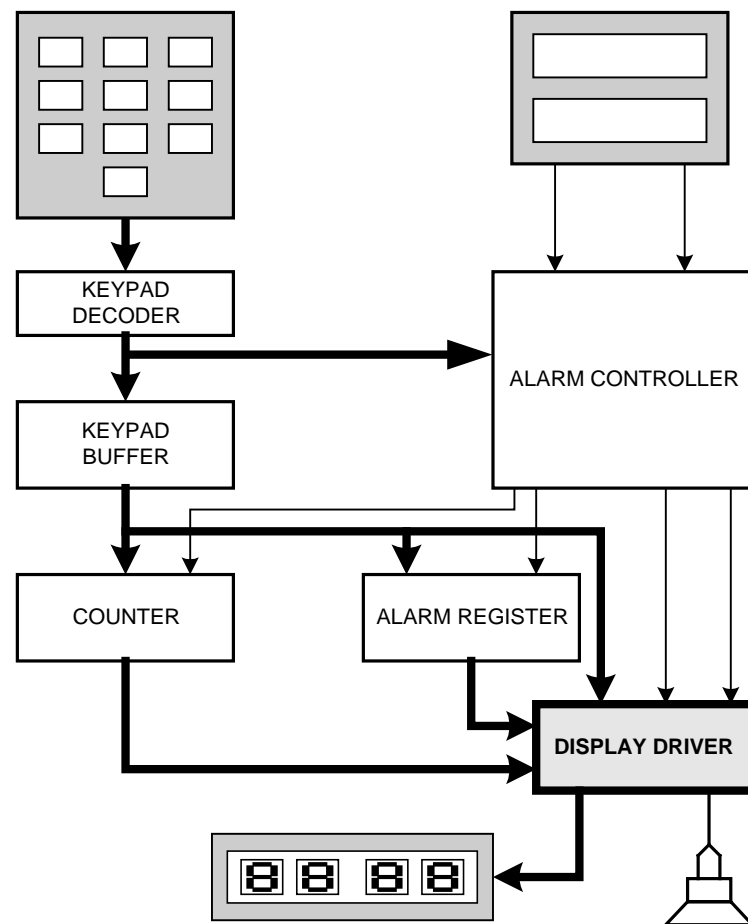


โดยกำหนดให้ตั้งชื่อ Design Units ต่างๆ ดังนี้

Entity: DISPLAY_MUX
 Architecture: RTL
 Data types: std_ulogic, std_ulogic_vector
 File names : DISMUX.VHD
 Testbench : T_DISMUX.VHD มาแก้ไขปรับปรุงใหม่

แล้วทำการ compile และ simulate ใหม่ โดยใช้โปรแกรม ModelSim

Workshop 3 : Adding a 7 Segment Display Driver

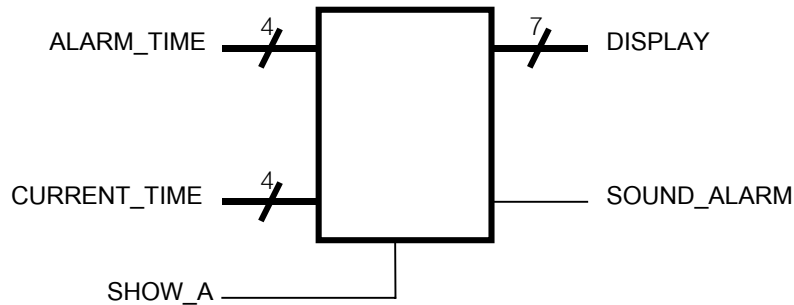


จุดประสงค์

- เพื่อสร้างวงจรที่ทำหน้าที่เป็น 7 segment driver
- เพื่อทดลองเรียกใช้งาน package design unit

ขั้นตอน

1. ให้ copy ไฟล์ DISMUX.VHD เป็น DDRVBIT.VHD แล้วแก้ไขให้สามารถขับ 7 segment ได้ และ copy ไฟล์ T_DISMUX.VHD เป็น T_DDRVBIT.VHD เพื่อใช้เป็น testbench file ของ Design ต่อไป
2. ทำแก้ไข design ซึ่งจะมีฟังก์ชันพื้นฐานคล้ายๆ กับ DISPLAY_MUX แต่มี output เป็น 7 บิต(std_ulogic_vector) สำหรับนำไป drive ตัว 7 segment ดังรูปที่



โดยกำหนดให้ตั้งชื่อ Design Unit ต่างๆ ดังนี้

Entity: DISPLAY_DRIVER

Architecture: RTL

Fuctionality:

- SOUND_ALARM = '1' เมื่อ ALARM_TIME = CURRENT_TIME
- DISPLAY เป็นสัญญาณ 7 bit ใช้ขับ 7 segment ให้แสดงผลเป็นตัวเลข integer ของค่า ALARM_TIME เมื่อ SHOW_A = '1' หรือ แสดง CURRENT_TIME เมื่อ SHOW_A='0'

Data type: std_ulogic, std_ulogic_vector

Coding Style: สร้าง process แยกออกมาอีกหนึ่ง process สำหรับใช้ assign ค่าของ 7 segment โดยใช้คำสั่ง CASE และใช้ค่า constant จากเพกเกจ P_ALARM ในการ Assign ค่า

- Testbench : entity ของ testbench ใช้ชื่อว่า TB_DISPLAY_DRIVER

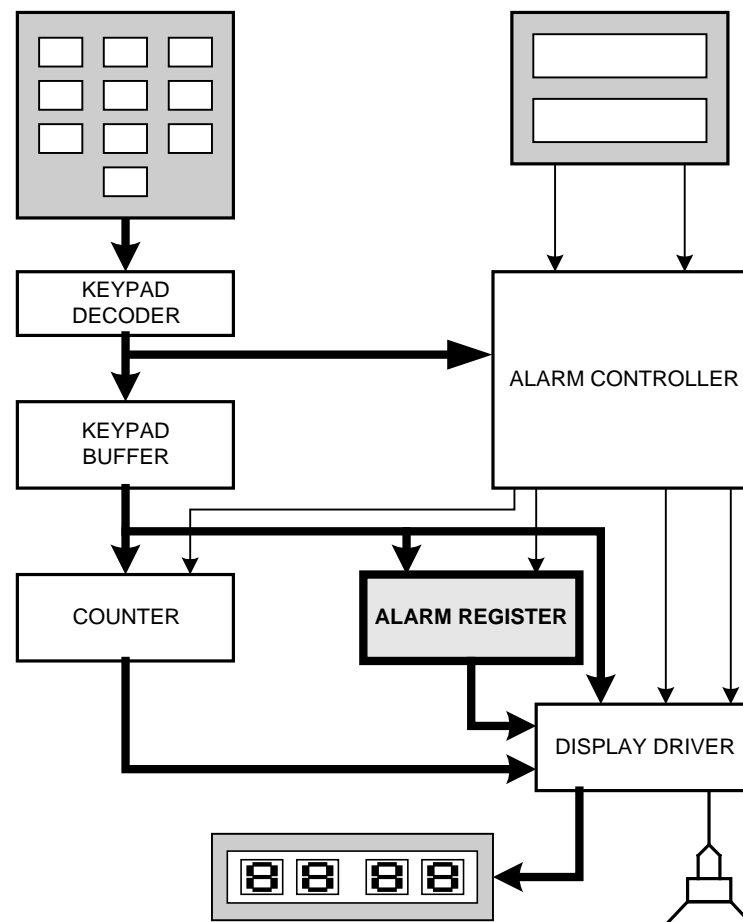
Package : ให้ใช้ package ที่ชื่อ P_ALARM ซึ่งเขียนอยู่ในไฟล์ที่ชื่อ P_ADIG.VHD โดยลองอ่านทำความเข้าใจก่อนว่า package อันนี้มีหน้าที่ทำอะไรบ้าง การเรียกใช้ package อย่าลืมกำหนดไว้ที่ต้นๆ ไฟล์ก่อนโดยใช้คำสั่ง use

```
use work.P_ALARM.all;
```

จากคำสั่งนี้บอกให้รู้ว่า Package นี้อยู่ใน Library ชื่อว่า work ดังนั้นก่อนที่จะ compile ไฟล์ DDRVBIT.VHD จะต้องทำการ compile ไฟล์ P_ADIG.VHD เสียก่อน

3. ทำการ compile (อย่าลืม compile ไฟล์ P_ADIG.VHD ก่อน) แล้ว simulate ทดลองปรับเปลี่ยน testbench เพื่อหาข้อผิดพลาดและแก้ไขจนได้ design ที่ถูกต้อง

Workshop 4 : The Alarm Register



จุดประสงค์

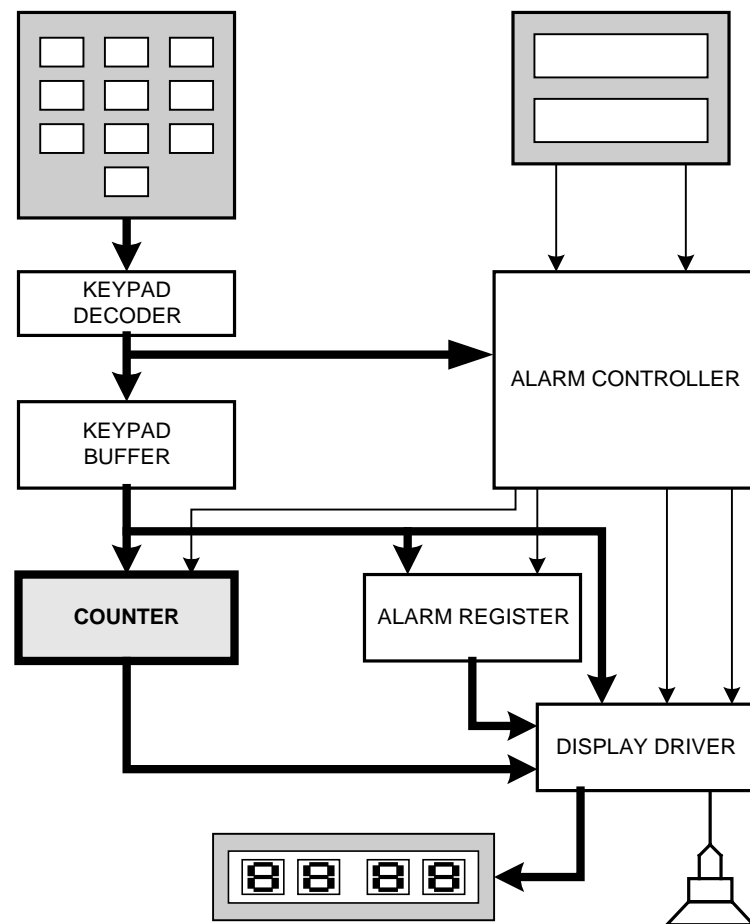
- เพื่อสร้าง Alarm Register สำหรับเก็บค่าเวลาที่ตั้งไว้สำหรับปลุก
- ศึกษารูปแบบการสร้าง Register หรือ Flip Flop

ข้อแนะนำ

การสร้างสัญญาณ clock ในการเขียน testbench

```
architecture ..... of ..... ;  
    signal clk      : std_ulogic := '0' ;  
begin  
    .....  
    .....  
    clk <= not clk after .... ns;  
    .....  
end ;
```

Workshop 5: A COUNTER

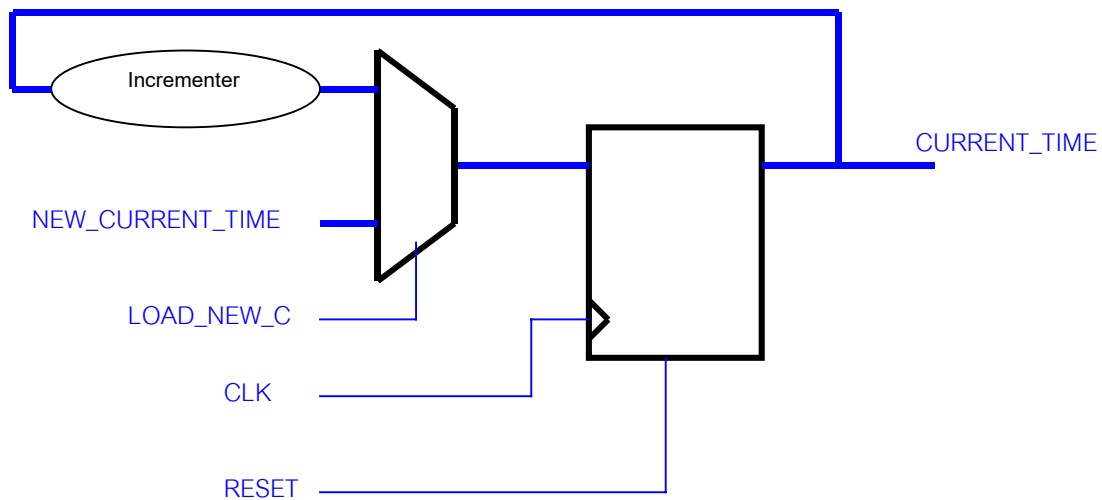


จุดประสงค์

- เพื่อสร้าง counter เป็นตัวเพิ่มค่าเวลาตามจังหวะสัญญาณ clock ที่เข้ามา
- ทดลองสร้าง combination ร่วมกับ register โดยใช้ process เพียง process เดียว

ขั้นตอน

- สร้าง design ใหม่ขึ้นมาโดยมีลักษณะวงจรดังรูปโดยใช้ process เพียง process เดียว



- Function:
 - ถ้า LOAD_NEW_C = '1' แล้ว register จะเก็บเอาค่า NEW_CURRENT_TIME
 - ถ้า LOAD_NEW_C = '0' แล้วค่าใน register จะเพิ่มทีละ 1 ตามจังหวะสัญญาณCLK
- Data_type:
 - CURRENT_TIME , NEW_CURRENT_TIME ใช้ integer range 0 to 9
 - สัญญาณที่เหลือใช้ std_ulogic
- File name: Entity (arch) :

- COUNT.VHD	- COUNTER (RTL)
- TB_COUNT.VHD	- TB_COUNTER (TEST)
- Coding Style: ใช้ process และ if / then / else ในการสร้าง register และใช้ statement และ variable ที่จำเป็นในการสร้าง incrementer กับ multiplexer ไว้ภายใน process เดียวกัน ดังรูปแบบต่อไปนี้

```

If rising clock then
    If need to load a new value load it
        Load it ;
    else
        .....
        Increment the existine value ;
    end if ;
end if;

```

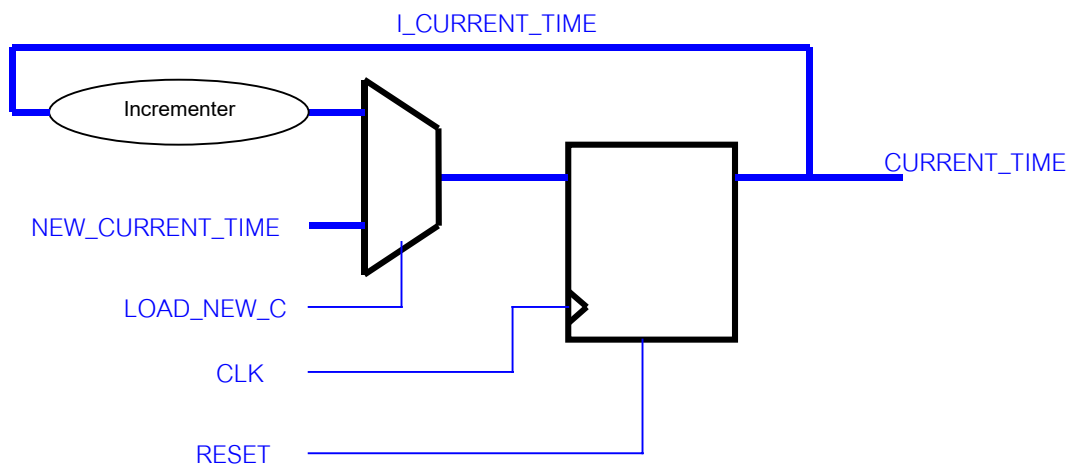
จาก block diagram จะเห็นว่าที่ output port (CURRENT_TIME) มีการป้อนกลับเข้ามาใช้อีก ในภาษา VHDL นั้น port ที่กำหนดเป็นประเภท OUT จะไม่สามารถอ่านเข้ามาใช้ได้ หากต้องการให้ Output Port สามารถอ่านกลับมาได้ต้องกำหนดเป็น BUFFER

เช่น CURRENT_TIME : buffer integer range 0 to 9 ;

แทนที่ CURRENT_TIME : out integer range 0 to 9 ;

แต่อย่างไรก็ตามการกำหนด port ประเภทนี้ จะมีปัญหากับ output port อื่นๆ ที่อยู่ใน Hierarchy ที่สูงขึ้น และต่อกับ port ตัวนี้จะต้องมีประเภท port เป็น buffer เหมือนกัน

***วิธีที่เหมาะสม** ในการออกแบบในลักษณะนี้ควรกำหนดสัญญาณที่ ป้อนกลับ เป็นสัญญาณภายในก่อนแล้วจึงต่อสัญญาณอันนี้เข้ากับ output port อีกทีดังนี้



architecture RTL of

signal I_CURRENT_TIME : integer range 0 to 9;

.....

begin

process.....;

begin

-- สร้างฟังก์ชันการทำงานโดยใช้สัญญาณ I_CURRENT_TIME

end process;

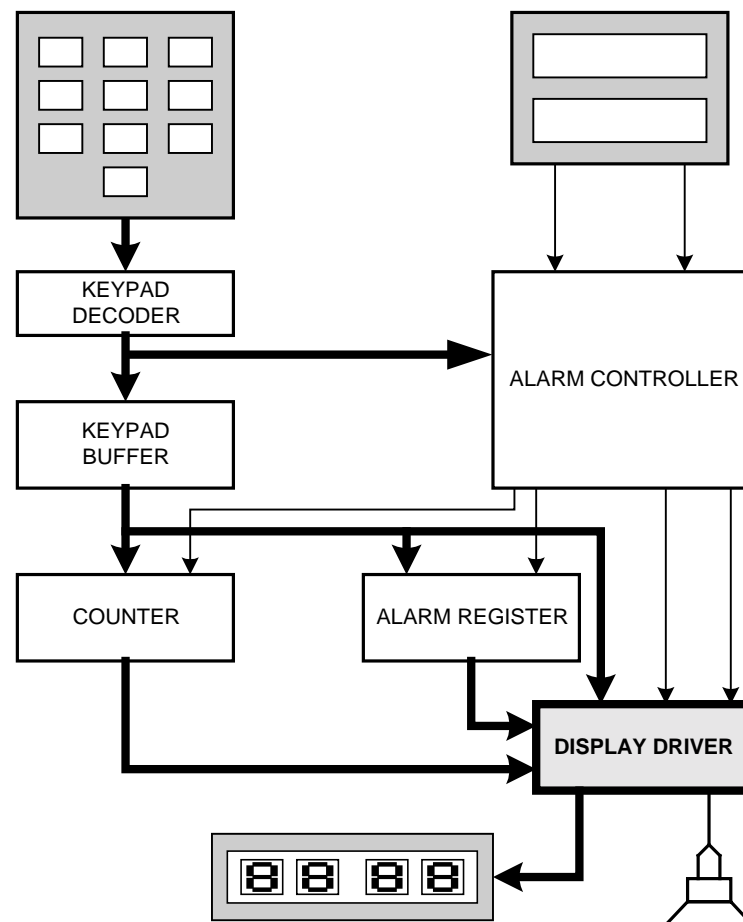
.....

-- สร้าง concurrent assignment (จะเห็นได้ว่าอยู่นอก process)

CURRENT_TIME <= I_CURRENT_TIME;

end RTL ;

Workshop 6 : Using Integers and Subtypes



การทดลองนี้จะกลับมาที่ Display Driver อีกครั้งเพื่อที่จะเปลี่ยนประเภทของ port คือ CURRENT_TIME, ALARM_TIME และ DISPLAY ให้เป็นแบบ integer

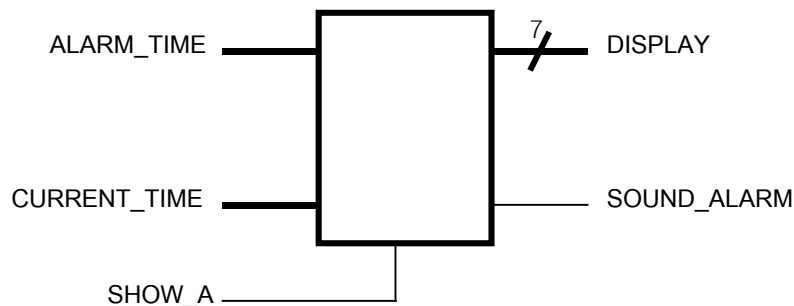
จุดประสงค์

- เพื่อทดลองใช้ subtype ในการกำหนดประเภทสัญญาณ โดยเปลี่ยนประเภทสัญญาณ CURRENT_TIME และ ALARM_TIME ซึ่งเป็นแบบ std_ulogic_vector มาเป็นแบบ integer

ขั้นตอน

นำเอา DISPLAY_DRIVER มาแก้ไขประเภทของ CURRENT_TIME, ALARM_TIME และ DISPLAY ให้เป็นประเภท integer โดยใช้ subtype ชื่อ T_CLOCK_TIME และ T_DISPLAY ซึ่งอยู่ในแพ็คเกจ P_ALARM (จากไฟล์ P_ADIG.VHD) โดย function การทำงานทั้งหมดยังคงเหมือนเดิมทุกประการ

- 1) เปิดไฟล์ P_ADIG.VHD สังเกตการใช้คำสั่ง subtype
- 2) นำเอา DISPLAY_DRIVER {DDRVBIT.VHD} มาแก้ไขประเภทของสัญญาณ
 - CURRENT_TIME, ALARM_TIME ใช้ประเภทสัญญาณเป็น T_CLOCK_TIME
 - DISPLAY ใช้ประเภทสัญญาณเป็น T_DISPLAY
- 3) แล้ว save เป็นไฟล์ใหม่ชื่อ DDRVINT.VHD
- 4) ทดลอง compile และ simulate



function: ยังเหมือนเดิมทุกประการ

File names: copy ไฟล์เดิมจาก Lab3 ทั้งตัว design และ testbench แล้วเปลี่ยนชื่อเป็น

DDRVBIT.VHD → DDRVINT.VHD

T_DDRVBIT.VHD → T_DDRVIT.VHD

Design Unit Names: ยังคงเหมือนเดิมทั้งส่วน design และ testbench

Package : ใช้แพ็คเกจ P_ALARM จากไฟล์ P_ADIG.VHD

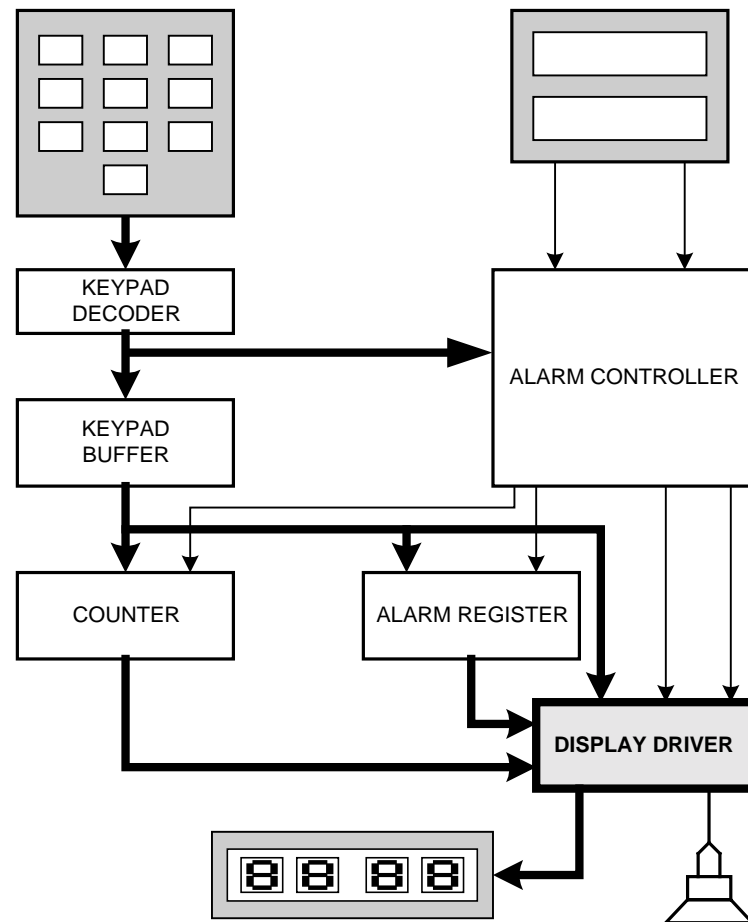
ใน Testbench สามารถเรียกดูการแสดงผลของ 7 Segment

ได้โดยใช้ Procedure ที่สร้างอยู่บนแพ็คเกจ P_DISPLAY (อยู่ในไฟล์ P_DISDIG.VHD) เป็นการจำลองรูปแบบการแสดงผลของ 7 segment โดยเขียนออกมาเป็น *text file* โดยใช้คำสั่งประเภท File Declaration ก่อนเรียกใช้งาน procedure นี้ต้องเรียกใช้ package ที่ชื่อ P_DISPLAY เสียก่อน (โดยคำสั่ง use work .P_DISPLAY.all;) แล้วเรียกใช้โดยใช้ concurrent statement ดังนี้ทั้งทำ testbench

```
DISPLAY_DIGIT(DISPLAY) ;
```

เมื่อทำการ simulation แล้วสามารถดูผลการแสดงผลได้ที่ *display_digit.txt*

Lab 7 : Using an Array of Integers



จุดประสงค์

- สร้าง DISPLAY_DRIVER สำหรับขับ 7 segment จำนวน 4 หลัก
- ทดลองออกแบบโดยใช้สัญญาณประเภท Array

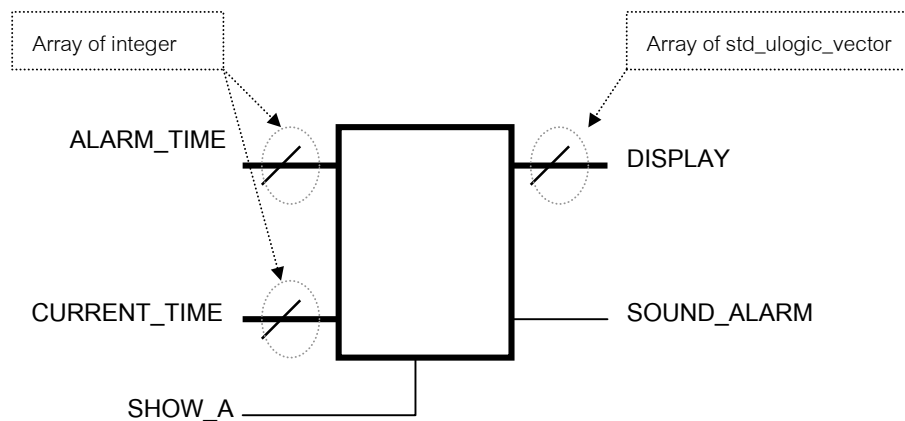
ขั้นตอน

จาก lab ที่ผ่านมาเรามี DISPLAY_DRIVER สำหรับ 7 segment เพียงหลักเดียว ต่อไปเราจะทำการออกแบบใหม่ให้สามารถแสดงผลได้ 4 หลัก (ชั่วโมง 2 หลัก, นาที 2 หลัก) โดยใช้สัญญาณประเภท Array โดยกำหนดให้มี 4 element แต่ละ element มีประเภทสัญญาณเป็น integer range 0 to 9 (สำหรับ ALARM_TIME, CURRENT_TIME) และ std_ulogic_vector(6 downto 0) (สำหรับ DISPLAY)

ประเภทของ signal ประกาศไว้ในแพ็คเกจ P_ALARM (อยู่ในไฟล์ P_ACLK.VHD) ดังนี้

```

type T_CLOCK_TIME is array (3 downto 0) of integer range 0 to 9;
type T_DISPLAY is array (3 downto 0) of std_ulogic_vector(6 downto 0);
  
```

- Functionality :

- SOUND_ALARM = '1' เมื่อ ALARM_TIME = CURRENT_TIME
- สัญญาณ DISPLAY เป็นกลุ่มข้อมูล 4 กลุ่ม กลุ่มละ 7 bit สำหรับ 7segment จำนวน 4ตัวในการแสดงผลเวลา โดยจะทำการแสดงค่า
CURRENT_TIME เมื่อ SHOW_A = '0'
ALARM_TIME เมื่อ SHOW_A = '1'
- ALARM_TIME และ CURRENT_TIME เป็นกลุ่มข้อมูลประเภท integer จำนวน 4 กลุ่ม แต่ละกลุ่มแทนค่าเวลาแต่ละหลัก

- File names :

ทำการ copy ไฟล์ DDRVINT.VHD → DDRVARRY.VHD

และ copy ไฟล์ T_DDRVIT.VHD → T_DDRVAY.VHD

- Data type :

- DISPLAY เป็น Array มี 4 element array แต่ละ array เป็นสัญญาณ 7 bit
- ALARM_TIME , CURRENT_TIME มี 4 element แต่ละ element สัญญาณเป็นประเภท integer

- Package :

P_ALARM {P_ACLK.VHD} สามารถเรียก package นี้มาใช้ได้เลย

- Design Unit Names :

- ใช้ชื่อ Entity และ Architecture เดิม คือ DISPLAY_DRIVER (RTL)

- Testbench :

ใช้ชื่อ Entity และ Architecture เดิม คือ TB_DISPLAY_DRIVER(TEST) อาจต้องมีการปรับปรุง ดัดแปลงการป้อนสัญญาณอินพุตเพื่อให้สอดคล้องกับประเภทสัญญาณของ DISPLAY_DRIVER ที่เปลี่ยนไป

- Seven Segment Display :

สามารถดูการแสดงผลของ 7 segment ได้โดยใช้ Procedure ที่สร้างอยู่บนไฟล์ P_DISCLK.VHD เป็นการจำลองรูปแบบการแสดงผลของ 7 segment โดยเขียนออกมาเป็น *text file* โดยใช้คำสั่งประเภท File Declaration (ดูที่บรรทัดที่ 48 ของ P_DISCLK.VHD)

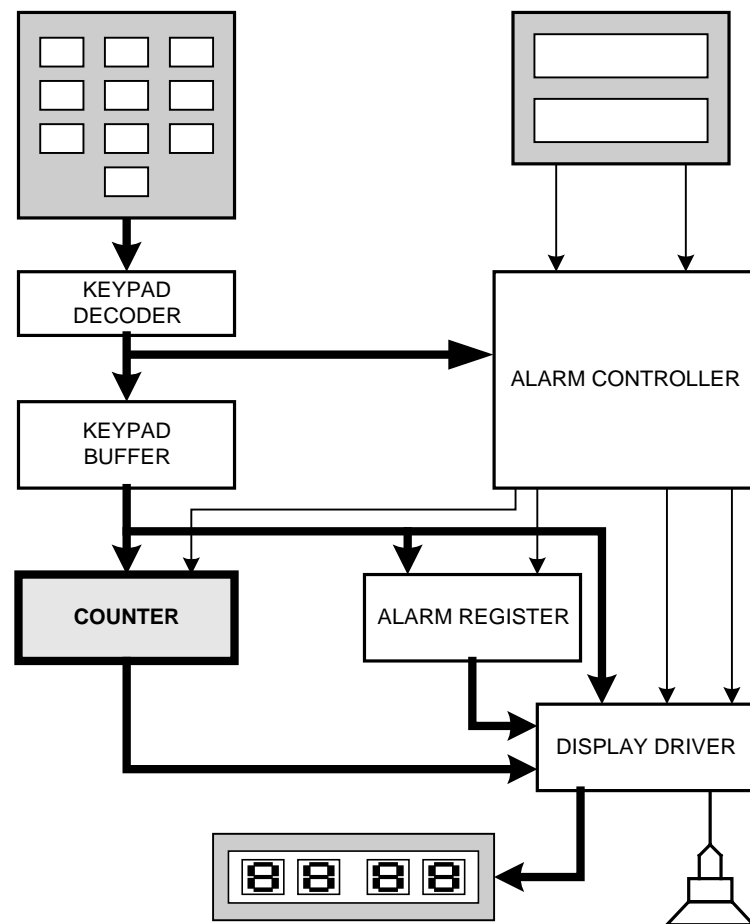
ก่อนเรียกใช้งาน procedure นี้ต้องเรียกใช้ package ที่ชื่อ P_DISPLAY เสียก่อน (โดยคำสั่ง `use work .P_DISPLAY.all;`) แล้วเรียกใช้โดยใช้ concurrent statement ดังนี้ลงท้าย testbench

```
DISPLAY_CLOCK (DISPLAY) ;
```

เมื่อทำการ simulation แล้วสามารถดูผลการแสดงผลได้ที่ *display_clock.txt*

ทำเช่นเดียวกันกับ ALARM_REGISTER โดยเปลี่ยนประเภทของพอร์ต NEW_ALARM_TIME และ ALARM_TIME ให้เป็น Array ของ integer โดยใช้ type จากแพ็คเกจเดียวกัน โดยยังคงชื่อ entity, architecture และชื่อไฟล์ไว้เช่นเดิม

Lab 8 : The full Counting Mechanism for the Alarm Clock



จุดประสงค์

- สร้าง counter ให้ครบทั้ง 4 หลัก (ชั่วโมง 2 หลัก, นาที 2 หลัก)


```

architecture RTL of ALARM_COUNTER is
    signal I_CURRENT_TIME : T_CLOCK_TIME;

begin
    .....

    process(CLK, RESET)
        variable MS_MIN, LS_MIN, MS_HOUR, LS_HOUR : integer range 0 to 9;
    begin
        -- { ใช้ variable ในการสร้างฟังก์ชันการทำงาน }
        .....
    end process;
    .....

end RTL ;

```

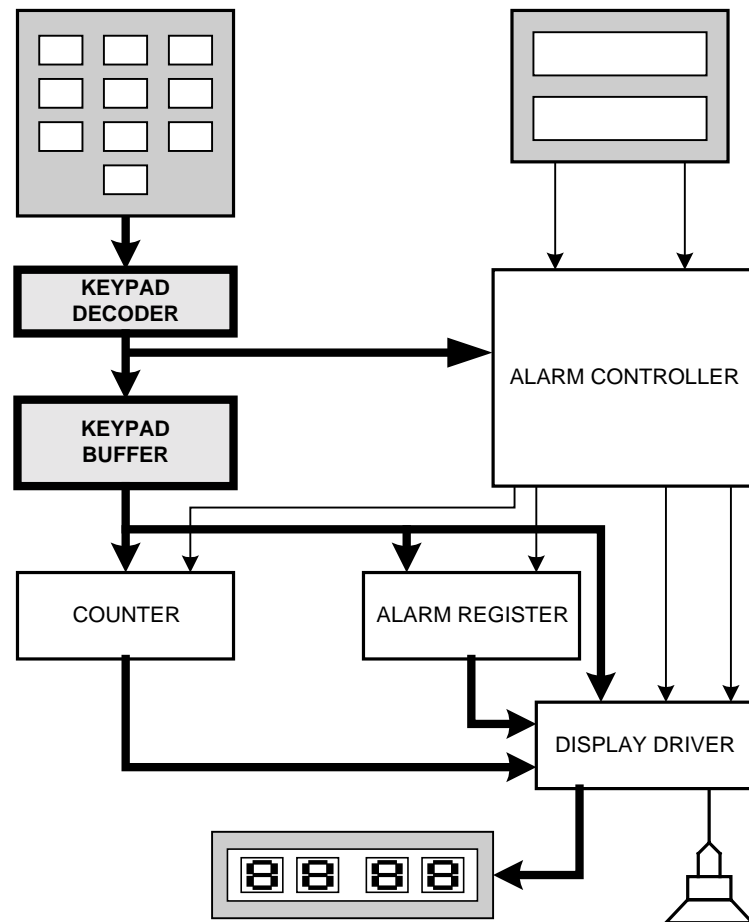
ใช้ constants ที่กำหนดไว้ใน P_ACLK.VHD ในการอ้างอิงข้อมูลแบบ Array ของ CURRENT_TIME แต่ละหลักเช่น

```

CURRENT_TIME(MS_HOUR_POSITION) <= ..... ;
CURRENT_TIME(MS_MIN_POSITION)   <= .....;

```

Lab 9 : Keypad Decoder & Keypad Buffer






จุดประสงค์

- ศึกษาการทำงานและหน้าที่ของ KEYPAD และ KEYBUFFER

{หมายเหตุ : ไม่ต้องทำการออกแบบ แต่ให้ศึกษารายละเอียดเพื่อนำไปใช้ใน lab ต่อไป}

KEYPAD DECODER

KEYPAD DECODER เป็นส่วนที่มี input ต่อกับสวิตช์แทนการกดเลข 0 – 9 มีหน้าที่เป็น decoder รับสัญญาณ std_ulogic_vector 10 bit เข้ามา แล้วแปลงให้เป็นสัญญาณ output ประเภท integer range 0 to 10 โดยจะมีค่าขึ้นกับการกดปุ่ม 0-9 ดังนี้

- 0 แทนการกดปุ่ม 
- 1 แทนการกดปุ่ม 
- 2 แทนการกดปุ่ม 
- :
- :

สามารถเขียนเป็น VHDL ได้ดังนี้

```
library IEEE;
use IEEE.Std_Logic_1164.all;
entity DECODER is
    port (KEYPAD : in std_ulogic_vector (0 to 9);
          VALUE  : out integer range 0 to 10 );
end DECODER;

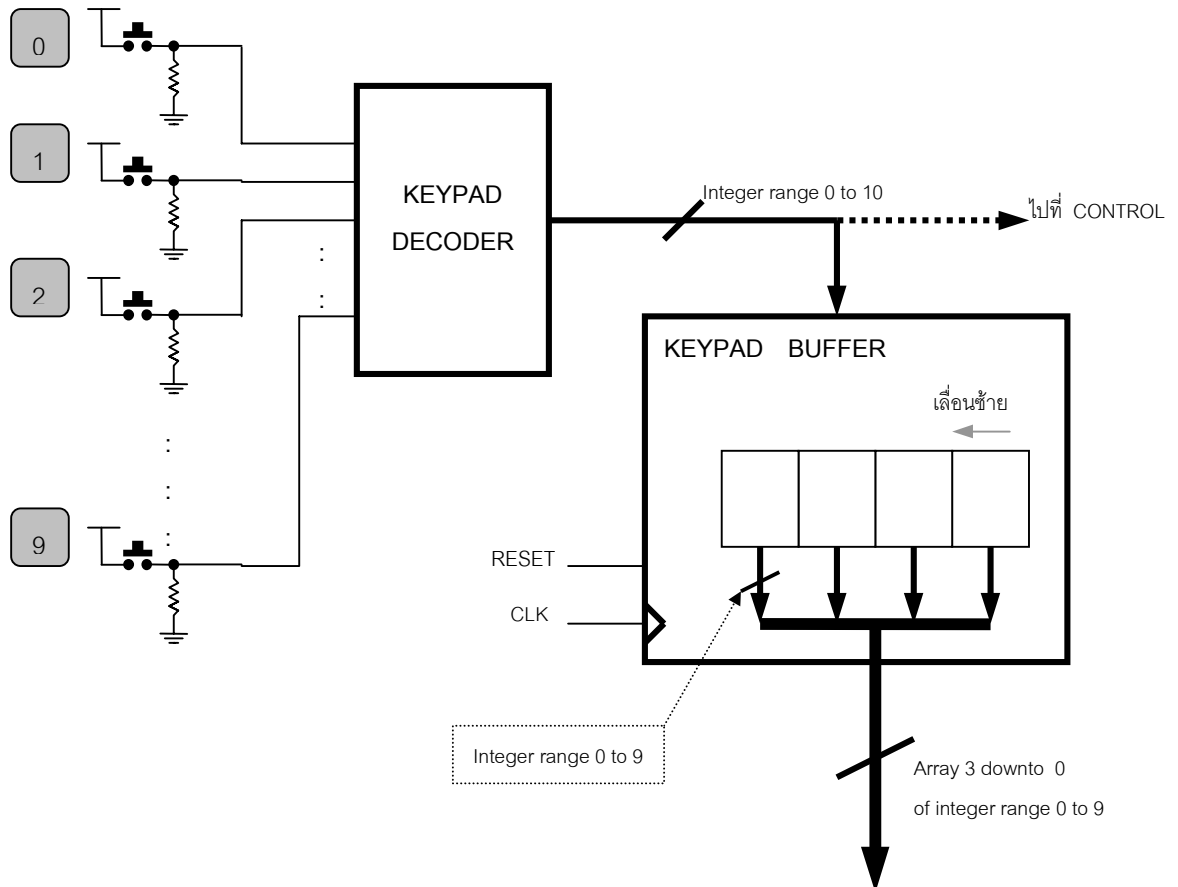
architecture RTL of DECODER is
begin
    TEN2FOUR: process(KEYPAD)
    begin
        case KEYPAD is
            when "1000000000" => VALUE <= 0;
            when "0100000000" => VALUE <= 1;
            when "0010000000" => VALUE <= 2;
            when "0001000000" => VALUE <= 3;
            when "0000100000" => VALUE <= 4;
            when "0000010000" => VALUE <= 5;
            when "0000001000" => VALUE <= 6;
            when "0000000100" => VALUE <= 7;
            when "0000000010" => VALUE <= 8;
            when "0000000001" => VALUE <= 9;
            when others         => VALUE <= 10;
        end case;
    end process TEN2FOUR;

end RTL;
```

DECODE.VHD

KEYPAD BUFFER

ทำหน้าที่รับสัญญาณ integer จาก KEYPAD DECODER เก็บไว้และทำการเลื่อนไปทางซ้ายเมื่อมีการกดตัวเลขตัวต่อไป KEYPAD BUFFER จึงทำหน้าที่เป็นเหมือน shift register ให้ out put เป็น 4 elements array แต่ละ element เป็น integer range 0 to 9 เพื่อส่งให้ส่วน ALARM_REGISTER , COUNTER และ DISPLAY_DRIVER ต่อไป




```
Library IEEE;
use IEEE.Std_Logic_1164.all;
use work.P_ALARM.all;

entity KEYPAD_BUFFER is
    port(KEY          : in integer range 0 to 10;
          CLK          : in std_ulogic;
          RESET        : in std_ulogic;
          NEW_TIME     : out T_CLOCK_TIME);
end KEYPAD_BUFFER;

architecture RTL of KEYPAD_BUFFER is
    signal SHIFT_REG : T_CLOCK_TIME;
begin
    SHIFTER: process(CLK, RESET)
    begin
        if RESET = '1' then
            SHIFT_REG <= (0,0,0,0);
        elsif CLK'event and CLK'last_value='0' and CLK='1' then
            if SHIFT_REG(LS_MIN_POSITION) /= KEY then
                -- We have a new key pressed, so
                -- if it is a valid key, shift the buffer

                if KEY /= 10 then
                    for i in (T_CLOCK_TIME'high - 1) downto T_CLOCK_TIME'low loop
                        SHIFT_REG(i+1) <= SHIFT_REG(i);
                    end loop;

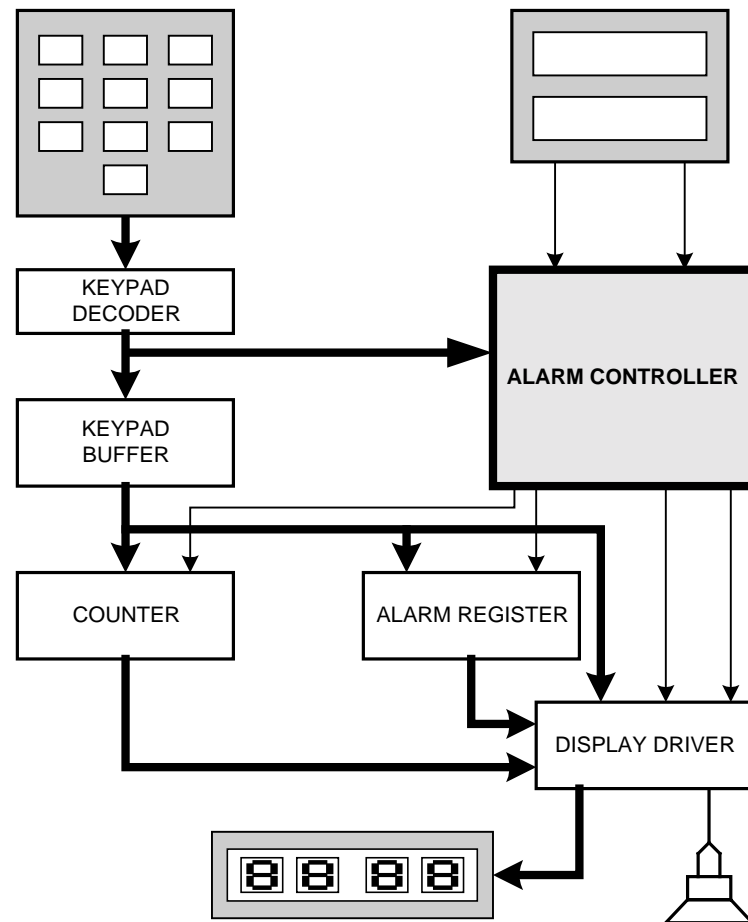
                    SHIFT_REG(LS_MIN_POSITION) <= KEY;
                end if;
            end if;
        end if;

        NEW_TIME <= SHIFT_REG;
    end process SHIFTER;

end RTL;
```

KEYBUF.VHD

Lab 10 : The Alarm Clock Controller

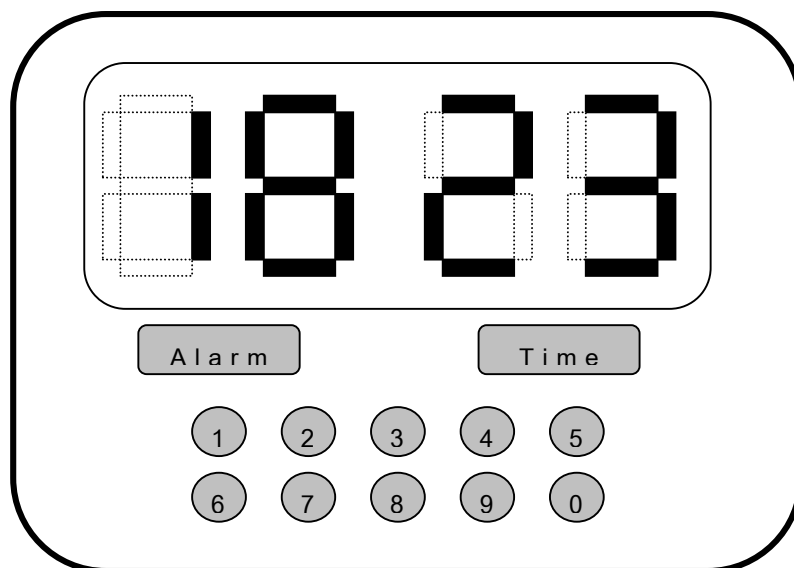


จุดประสงค์

- สร้างสัญญาณ control เพื่อควบคุมการทำงานของ ส่วนต่างๆของ Alarm Clock ที่ออกแบบไว้แล้ว
- ทดลองเขียนและออกแบบ VHDL ในรูปแบบ state diagram

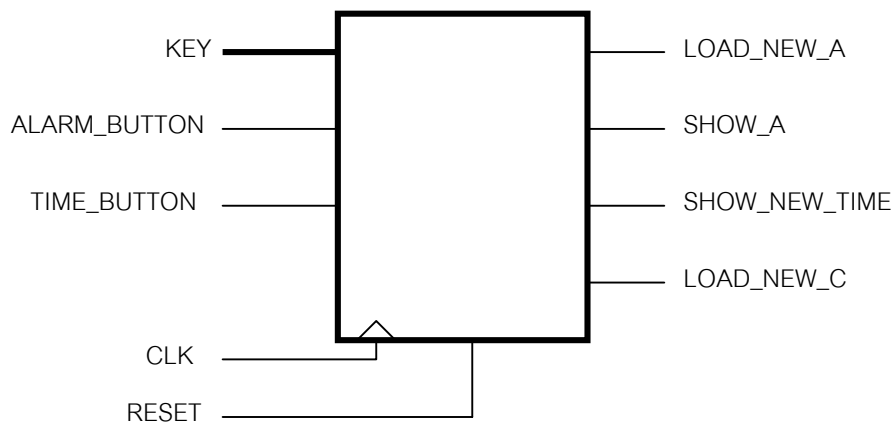
ขั้นตอน

ทำการออกแบบส่วน control เพื่อควบคุมการทำงานของ Alarm Clock ให้มีรูปแบบการทำงานดังนี้



- ปุ่ม 0-9
เมื่อกดปุ่มเหล่านี้ใช้ในการตั้งเวลาของนาฬิกา ทั้งเวลาปัจจุบันและเวลาที่ตั้งปลุก เมื่อกดปุ่มใดปุ่มหนึ่งโดนกด จะถูกแสดงค่าที่หลักทศของจอแสดงผลแล้วจะเลื่อนมาทางซ้าย
- ตั้งเวลาปลุก
กดปุ่ม 0-9 เพื่อตั้งเวลาแล้วตามด้วยปุ่ม Alarm จอdisplay จะแสดงเวลาปัจจุบัน ขณะเดียวกันเวลาที่ตั้งไว้จะถูกเก็บไว้ใน Alarm register
- ตั้งเวลาปัจจุบัน
กดปุ่ม 0-9 เพื่อตั้งเวลาแล้วตามด้วยปุ่ม Time จอแสดงผลจะแสดงเวลาใหม่ทันที
- ดูเวลาที่ตั้งปลุกไว้
โดยการกดปุ่ม Alarm ค้างไว้โดยก่อนหน้านี้ไม่ต้องกด 0-9 จอแสดงผลจะแสดงค่า Alarm time ที่ได้ตั้งไว้แล้วเมื่อปล่อยปุ่ม Alarm ก็แสดงเวลาปัจจุบันต่อไป

การออกแบบ Alarm Controller โดยใช้ State Machine



File name

ACTRL.VHD



TB_ACTRL.VHD

Entity (Arch)

ALARM_CONTROLLER (RTL)

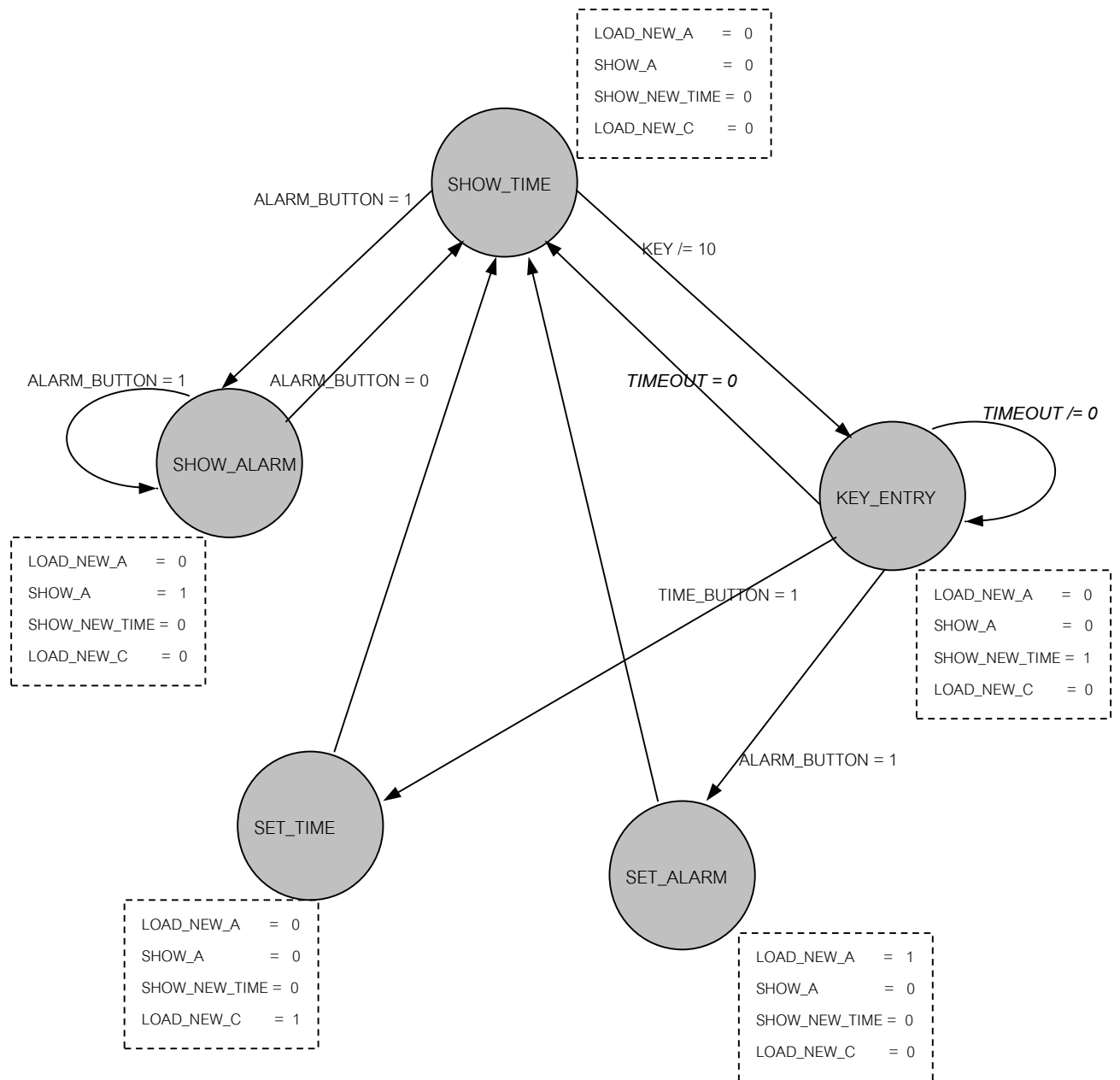
TB_ALARM_CONTROL (TEST)

- Functionality

- ALARM_BUTTON : เป็นport ที่ต่อกับ switch ภายนอก 
- TIME_BUTTON : เป็นport ที่ต่ออยู่กับ switch ภายนอก 
- KEY : เป็น port ที่รับสัญญาณที่มาจาก DECODER (ให้เปิดดูไฟล์ DECODE.VHD) ซึ่งทำหน้าที่รับสัญญาณจาก 0-9 switch แล้ว decode ให้เป็นข้อมูลแบบ integer โดยที่ขณะไม่กดปุ่มใดๆ จะให้ค่า 10 ออกมา ทำให้สามารถตรวจสอบการกด switch ได้
- LOAD_NEW_A : ใช้ควบคุม ALARM_REG
 - '1' : Alarm register ทำการเก็บค่าเวลาใหม่ในการตั้งปลุก
 - '0' : Alarm register ไม่เปลี่ยนแปลงค่า
- SHOW_A : ใช้ควบคุม DISPLAY_DRIVER
 - '1' : จอแสดงผลจะแสดงค่า ALARM_TIME ที่ตั้งไว้ใน Alarm register
 - '0' : จอแสดงผลจะแสดงค่า CURRENT_TIME
- SHOW_NEW_TIME : ใช้เป็นสัญญาณควบคุม DISPLAY_DRIVER ให้แสดงค่าเวลาตามที่กดปุ่ม 0-9 ซึ่ง DISPLAY_DRIVER ที่ออกแบบไว้ยังไม่มีport ที่จะรับสัญญาณนี้ ต้องเพิ่มเติมส่วนนี้เข้าไปอีก ดังนี้
 - '1' : จอแสดงผลจะแสดงเวลาตั้งไว้ขณะกด 0-9
 - '0' : จะแสดงเวลาปัจจุบันที่กำลังเดินอยู่ หรือเวลาที่ตั้งไว้ใน Alarm register
- LOAD_NEW_C ใช้ในการตั้งค่าเวลาปัจจุบันใหม่
 - '1' : ทำการตั้งเวลาปัจจุบันเป็นค่าใหม่หลังจากกด 0-9
 - '0' : เมื่อไม่ต้องตั้งเวลาใหม่และนาฬิกาเดินเป็นปกติ

- Finite state machine

จากเงื่อนไขทั้งหมดเราสามารถสร้างเป็น Finite state machine ได้ดังนี้



- SHOW_TIME

คือ state ปกติที่ไม่มีการกดปุ่มหรือ switch ใด ๆ เมื่อมีการกด Alarm (ALARM_BUTTON = '1') จะมีการเปลี่ยน state ไปที่ SHOW_ALARM

หรือเมื่อกดปุ่ม 0 – 9 ปุ่มใดปุ่มหนึ่ง จะเปลี่ยน state ไปที่ KEY_ENTRY โดยสามารถตรวจสอบได้จากค่า KEY ถ้าเป็น 10 แสดงว่าไม่มีการกดปุ่ม (ดูรายละเอียดเพิ่มเติมใน DECODE.VHD)

- KEY_ENTRY

เป็น state ขณะที่มีการกด 0–9 จอแสดงผลจะแสดงค่าตัวเลขที่กดไป และภายใน state นี้จะมีสัญญาณภายในชื่อ TIMEOUT ทำหน้าที่เป็น counter เพื่อหน่วงเวลาเมื่อไม่มีการกดปุ่มใดๆแล้วจะหน่วงเวลาไว้จนกระทั่งค่า TIMEOUT ลดค่าลงจนถึง 0 ก็จะมีการเปลี่ยน state กลับไปที่ SHOW_TIME อีกครั้งโดยไม่มีการตั้งเวลาใหม่

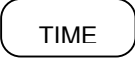
- SHOW_ALARM

State นี้เกิดขึ้นเมื่อมีการกดปุ่ม  โดยที่ไม่ได้กดปุ่ม 0 – 9 มาก่อน ใน state นี้จะทำการควบคุมให้ DISPLAY_DRIVER แสดงค่าเวลาดังปุ่ม (ซึ่งเป็นค่าที่อยู่ใน ALARM REGISTER) ออกมาทางจอแสดงผล จะแสดงจนกว่าจะปล่อยปุ่ม

- SET_ALARM

เกิดขึ้นหลังจากกดปุ่ม 0–9 แล้วตามด้วย  ส่วน control จะทำการส่งสัญญาณ LOAD_NEW_A ไปที่ ALARM REGISTER เพื่อเก็บค่าเวลาดังปุ่มค่าใหม่

- SET_TIME

State นี้เกิดขึ้นหลังจากกดปุ่ม 0 – 9 แล้วตามด้วย  ส่วน control จะทำการส่งสัญญาณ LOAD_NEW_C ไปที่ COUNTER เพื่อเก็บค่าเวลาปัจจุบันค่าใหม่

Coding Style :

สร้าง ALARM_CONTROLLER state-machine โดยใช้ clocked process แต่สำหรับ output signal ให้ใช้ concurrent statements นอก clocked process ยกตัวอย่าง

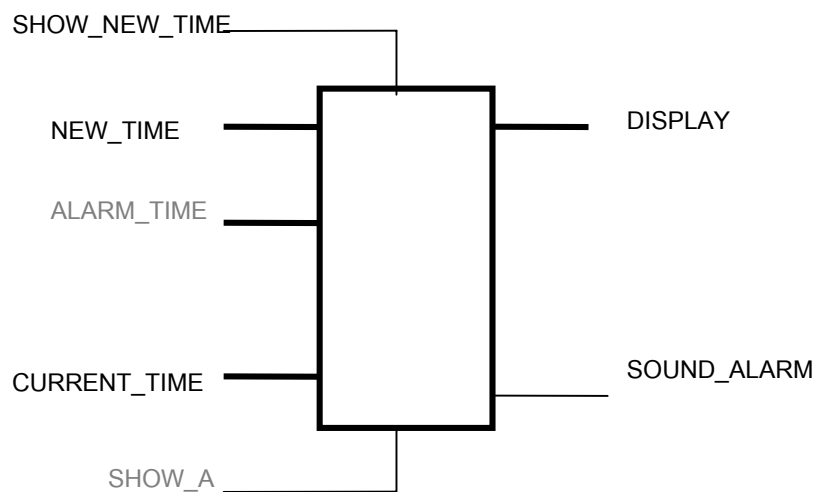
```

process .....
    < clock description >
    .....
    case STATE
        when < first state > =>
            if < input_condition > then
                STATE <= < second_state >;
            end if;
        when < second_state > => .....
        .....
    end case;
    .....
end process;
    .....
SHOW_NEW_TIME <= '1' when STATE = ..... else '0';
SHOW_A        <= '1' when STATE = ..... else '0';
LOAD_NEW_A    <= '1' when STATE = ..... else '0';
LOAD_NEW_C    <= '1' when STATE = ..... else '0';
    .....
end RTL ;

```

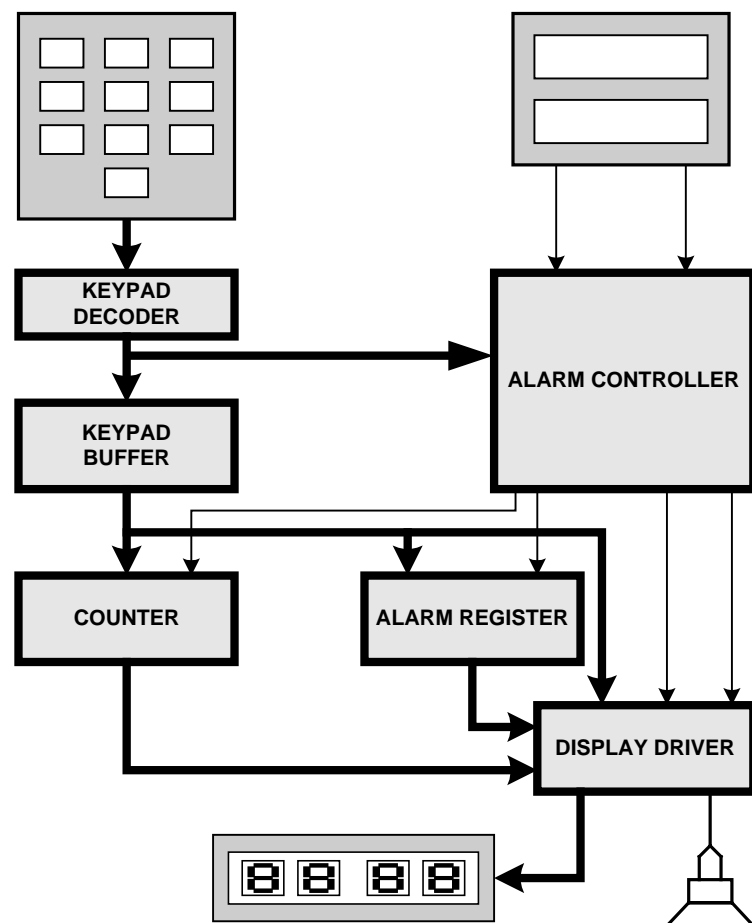
DISPLAY_DRIVER :

ที่ DISPLAY_DRIVER ให้ทำการเพิ่มวงจรตามรูปเพื่อแสดงตัวเลขขณะกดปุ่ม 0-9



- function : ถ้า SHOW_NEW_TIME = '1' ให้ DISPLAY มีค่าเท่ากับ KEYPAD
 ถ้า SHOW_NEW_TIME = '0' ให้ DISPLAY มีค่าเท่ากับ ALARM_TIME หรือ
 CURRENT_TIME ทั้งนี้ขึ้นกับ SHOW_A เหมือนกับที่ design ไว้แล้ว
- File name : Entity (Arch)
 DDRVARY2.VHD DISPLAY_DRIVER2 (RTL)

Lab 11 : Putting it All together



จุดประสงค์

- เพื่อรวมทุกๆ บล็อกโดยเชื่อมต่อสัญญาณเข้าด้วยกัน
- ศึกษาการเขียน VHDL ในรูปแบบ structure

ขั้นตอน

Lab นี้จะทำการรวมทุกบล็อกเข้าด้วยกัน โดยสร้างไฟล์ใหม่ขึ้นมา เขียนในรูปแบบ structure แสดงการเชื่อมต่อของแต่ละ component เข้าด้วยกัน โดยใช้คำสั่ง component , port map และการเขียน configuration

- เพื่อไม่ให้เป็นการเสียเวลาสามารถใช้ไฟล์ที่สร้างไว้แล้วชื่อ **ACLK.VHD** สังเกตการเขียน VHDL ในรูปแบบที่เรียกว่า *Structure* แล้วสร้าง *testbench* ขึ้นมาทดสอบการทำงาน

Exercise Solution

DISMUX.VHD

```

-----
-- Copyright Esperan 1992
-- This is the code for the display multiplexer used
-- in the labs of the Esperan Introduction to VHDL
-- Based Design Training Workshop
-----

Library IEEE;
use IEEE.Std_Logic_1164.all;

entity DISPLAY_MUX is
port (
    ALARM_TIME, CURRENT_TIME : in std_ulogic_vector (3 downto 0);
    SHOW_A                   : in std_ulogic;
    SOUND_ALARM              : out std_ulogic;
    DISPLAY_TIME              : out std_ulogic_vector (3 downto 0) );
end DISPLAY_MUX;

architecture RTL of DISPLAY_MUX is

begin

    process (ALARM_TIME,CURRENT_TIME,SHOW_A)
    begin
        if (SHOW_A='1') then
            DISPLAY_TIME <= ALARM_TIME;
        else
            DISPLAY_TIME <= CURRENT_TIME;
        end if;

        if (ALARM_TIME = CURRENT_TIME) then
            SOUND_ALARM <= '1';
        else
            SOUND_ALARM <= '0';
        end if;

    end process;

end RTL;

```

DDRVBIT.VHD

```

-----
-- Copyright Esperan 1992
-- Code for the bit based display driver used
-- in the labs of the Esperan Introduction to VHDL
-- Based Design Training Workshop
-----

Library IEEE;
use IEEE.Std_Logic_1164.all;
use work.P_ALARM.all;

entity DISPLAY_DRIVER is
port ( ALARM_TIME, CURRENT_TIME      : in std_ulogic_vector (3 downto 0);
    SHOW_A                          : in std_ulogic;
    SOUND_ALARM                      : out std_ulogic;
    DISPLAY                          : out std_ulogic_vector (6 downto 0) );
end DISPLAY_DRIVER;

architecture RTL of DISPLAY_DRIVER is
    signal DISPLAY_TIME : std_ulogic_vector (3 downto 0);
begin

```

```

-----
-- process to select what is displayed
-----
DISPLAY_MUX :
process (ALARM_TIME, CURRENT_TIME, SHOW_A)
begin

    if (SHOW_A = '1') then
        DISPLAY_TIME <= ALARM_TIME;
    else
        DISPLAY_TIME <= CURRENT_TIME;
    end if;
end process DISPLAY_MUX;

-----
-- process to sound the alarm
-----
GEN_ALARM:
process (ALARM_TIME, CURRENT_TIME)
begin

    if (ALARM_TIME = CURRENT_TIME) then
        SOUND_ALARM <= '1';
    else
        SOUND_ALARM <= '0';
    end if;
end process GEN_ALARM;

-----
-- process to drive the 7 segment display
-----
DISPLAY_PROCESS :
process (DISPLAY_TIME)
begin
    case DISPLAY_TIME is
        when "0000" => DISPLAY <= ZERO_SEG;
        when "0001" => DISPLAY <= ONE_SEG;
        when "0010" => DISPLAY <= TWO_SEG;
        when "0011" => DISPLAY <= THREE_SEG;
        when "0100" => DISPLAY <= FOUR_SEG;
        when "0101" => DISPLAY <= FIVE_SEG;
        when "0110" => DISPLAY <= SIX_SEG;
        when "0111" => DISPLAY <= SEVEN_SEG;
        when "1000" => DISPLAY <= EIGHT_SEG;
        when "1001" => DISPLAY <= NINE_SEG;
        when OTHERS => DISPLAY <= "0000000";
    end case;
end loop;
end process DISPLAY_PROCESS;

end RTL ;

```

AREGS.VHD

```

-----
-- Copyright Esperan 1992
-- Register bank to store the alarm time for the
-- Alarm Clock project in the Esperan Introduction to
-- VHDL Based Design training workshop
-- Revision Control
-- 0.1 First attempt
-----

Library IEEE;
use IEEE.Std_Logic_1164.all;
use work.P_ALARM.all;

entity ALARM_REG is
    port( NEW_ALARM_TIME      : in T_CLOCK_TIME;
          LOAD_NEW_A          : in std_ulogic;
          CLK                  : in std_ulogic;
          RESET                : in std_ulogic;
          ALARM_TIME           : out T_CLOCK_TIME );
end ALARM_REG;

architecture RTL of ALARM_REG is

```

```

begin
  process(CLK, RESET)
  begin
    if (RESET = '1') then
      ALARM_TIME <= (0,0,0,0);
    elsif (CLK'event and CLK='1') then
      if (LOAD_NEW_A = '1') then
        ALARM_TIME <= NEW_ALARM_TIME;
      end if;
    end if;
  end process;
end RTL;

```

COUNT.VHD

```

-----
-- Copyright Esperan 1992
-- This is the code for a counter in the alarm clock used
-- in the labs of the Esperan Introduction to VHDL
-- Based Design Training Workshop
-- Revision Control
-- 0.1 First attempt
-----

Library IEEE;
use IEEE.Std_Logic_1164.all;
use work.P_ALARM.all;

entity COUNTER is
  port( NEW_CURRENT_TIME   : in integer range 0 to 9;
        LOAD_NEW_C         : in std_ulogic;
        CLK, RESET         : in std_ulogic;
        CURRENT_TIME       : out integer range 0 to 9 );
end COUNTER ;

architecture RTL of COUNTER is
  signal I_CURRENT_TIME : integer range 0 to 9;
begin

  -----
  -- Simple Incrementer...
  -----
  TIME_COUNTER:
  process(CLK, RESET)
    variable LS_MIN: integer range 0 to 9;
  begin

    if (RESET = '1') then
      I_CURRENT_TIME <= 0;
    elsif (CLK'event and CLK='1') then
      if (LOAD_NEW_C = '1') then
        I_CURRENT_TIME <= NEW_CURRENT_TIME;
      else -- count!!
        LS_MIN := I_CURRENT_TIME;
        if (LS_MIN /= 9) then
          LS_MIN := LS_MIN + 1;
        else
          LS_MIN := 0;
        end if;
        I_CURRENT_TIME <= LS_MIN;
      end if;
    end if;

  end process TIME_COUNTER;

  CURRENT_TIME <= I_CURRENT_TIME;

end RTL ;

```

DDRVARRY.VHD

```

-----
-- Copyright Esperan 1992
-- This is the code for the integer based display_driver
-- used in the labs of the Esperan Introduction to VHDL
-- Based Design Training Workshop
-----

Library IEEE;
use IEEE.Std_Logic_1164.all;
use work.P_ALARM.all;

entity DISPLAY_DRIVER is
port ( ALARM_TIME, CURRENT_TIME      : in T_CLOCK_TIME;
      SHOW_A                          : in std_ulogic;
      SOUND_ALARM                     : out std_ulogic;
      DISPLAY                         : out T_DISPLAY );
end DISPLAY_DRIVER;

architecture RTL of DISPLAY_DRIVER is
  signal DISPLAY_TIME : T_CLOCK_TIME;
begin

  -----
  -- process to select what is displayed
  -----
  DISPLAY_MUX :
  process (ALARM_TIME, CURRENT_TIME, SHOW_A)
  begin

    if (SHOW_A = '1') then
      DISPLAY_TIME <= ALARM_TIME;
    else
      DISPLAY_TIME <= CURRENT_TIME;
    end if;
  end process DISPLAY_MUX;

  -----
  -- process to sound the alarm
  -----
  GEN_ALARM:
  process (ALARM_TIME, CURRENT_TIME)
  begin

    if (ALARM_TIME = CURRENT_TIME) then
      SOUND_ALARM <= '1';
    else
      SOUND_ALARM <= '0';
    end if;
  end process GEN_ALARM;

  -----
  -- process to drive the 7 segment display
  -----
  DISPLAY_PROCESS :
  process (DISPLAY_TIME)
  begin
    for i in T_DISPLAY'range loop
      case DISPLAY_TIME(i) is
        when 0 => DISPLAY(i) <= ZERO_SEG;
        when 1 => DISPLAY(i) <= ONE_SEG;
        when 2 => DISPLAY(i) <= TWO_SEG;
        when 3 => DISPLAY(i) <= THREE_SEG;
        when 4 => DISPLAY(i) <= FOUR_SEG;
        when 5 => DISPLAY(i) <= FIVE_SEG;
        when 6 => DISPLAY(i) <= SIX_SEG;
        when 7 => DISPLAY(i) <= SEVEN_SEG;
        when 8 => DISPLAY(i) <= EIGHT_SEG;
        when 9 => DISPLAY(i) <= NINE_SEG;
      end case;
    end loop;
  end process DISPLAY_PROCESS;

end RTL ;

```

ACOUNT.VHD

```

-----
-- Copyright Esperan 1992
-- This is the code for a counter of the alarm clock used
-- in the labs of the Esperan Introduction to VHDL
-- Based Design Training Workshop
-- Revision Control
-- 0.1 First attempt
-----

Library IEEE;
use IEEE.Std_Logic_1164.all;
use work.P_ALARM.all;

entity ALARM_COUNTER is
  port( NEW_CURRENT_TIME    : in T_CLOCK_TIME;
        LOAD_NEW_C          : in std_ulogic;
        CLK, RESET          : in std_ulogic;
        CURRENT_TIME        : out T_CLOCK_TIME    );
end ALARM_COUNTER ;

architecture RTL of ALARM_COUNTER is
  signal I_CURRENT_TIME : T_CLOCK_TIME;
begin

  -----
  -- Time counter ...
  -----
  TIME_COUNTER:
  process(CLK, RESET)
    variable MS_MIN, LS_MIN, MS_HOUR, LS_HOUR : integer range 0 to 9;
  begin

    if (RESET = '1') then
      I_CURRENT_TIME <= (others => 0);
    elsif (CLK'event and CLK='1') then
      -- default assignments to avoid latches
      MS_HOUR := 0;
      LS_HOUR := 0;
      MS_MIN  := 0;
      LS_MIN  := 0;
      if (LOAD_NEW_C = '1') then
        I_CURRENT_TIME <= NEW_CURRENT_TIME;
      else -- count!!
        -- the algorithm is start at the minutes, and ripple upwards
        -- first assign our signal to temporary variables
        MS_HOUR := I_CURRENT_TIME(MS_HOUR_POSITION);
        LS_HOUR := I_CURRENT_TIME(LS_HOUR_POSITION);
        MS_MIN  := I_CURRENT_TIME(MS_MIN_POSITION);
        LS_MIN  := I_CURRENT_TIME(LS_MIN_POSITION);

        if LS_MIN >= 9 then
          LS_MIN := 0;
          if MS_MIN >= 5 then
            MS_MIN := 0;
            if (MS_HOUR >= 2) and (LS_HOUR >= 3) then
              LS_HOUR := 0;
              MS_HOUR := 0;
            elsif LS_HOUR >= 9 then
              LS_HOUR := 0;
              MS_HOUR := MS_HOUR + 1;
            else LS_HOUR := LS_HOUR + 1;
            end if;
          else MS_MIN := MS_MIN + 1;
          end if;
        else LS_MIN := LS_MIN + 1;
        end if;

        -- load the newly calculated values back
        I_CURRENT_TIME(MS_HOUR_POSITION) <= MS_HOUR;
        I_CURRENT_TIME(LS_HOUR_POSITION) <= LS_HOUR;
        I_CURRENT_TIME(MS_MIN_POSITION)  <= MS_MIN;
        I_CURRENT_TIME(LS_MIN_POSITION)  <= LS_MIN;
      end if;
    end if;
  end process;

```

```

        end if; -- (LOAD_NEW_C = '1')

        end if; -- CLK'event
    end process TIME_COUNTER;

    CURRENT_TIME <= I_CURRENT_TIME;

end RTL ;

```

ACTRL.VHD

```

-----
-- Copyright Esperan 1996
-- This is the code for a controller of the alarm clock used
-- in the labs of the Esperan Introduction to VHDL
-- Based Design Training Workshop
-- Revision Control
-- 0.1 First attempt
-----

Library IEEE;
use IEEE.Std_Logic_1164.all;
use work.P_ALARM.all;

entity ALARM_CONTROLLER is
    port( CLK          : in std_ulogic;
          RESET        : in std_ulogic;
          KEY           : in integer range 0 to 10;
          ALARM_BUTTON : in std_ulogic;
          TIME_BUTTON   : in std_ulogic;
          LOAD_NEW_A     : out std_ulogic;
          SHOW_A         : out std_ulogic;
          SHOW_NEW_TIME  : out std_ulogic;
          LOAD_NEW_C     : out std_ulogic );
end ALARM_CONTROLLER;

architecture RTL of ALARM_CONTROLLER is

    type T_STATE is (SHOW_TIME,      -- clock just counts
                     KEY_ENTRY,      -- in key entry mode
                     KEY_STORED,     -- key entered and shifted into buffer
                     SHOW_ALARM,     -- shows time alarm is set
                     SET_ALARM_TIME, -- sets alarm time
                     SET_CURRENT_TIME); -- sets current time

    signal STATE : T_STATE; -- holds the internal state of block
    signal TIMEOUT : integer range 0 to 10; -- counts the keypad timeout

begin

    STATE_MACHINE: process (CLK, RESET)
    begin
        if RESET = '1' then
            STATE <= SHOW_TIME;
            TIMEOUT <= 0;
        elsif CLK'event AND CLK = '1' then
            case STATE is
                when SHOW_TIME =>
                    if ALARM_BUTTON = '1' then
                        STATE <= SHOW_ALARM;
                    elsif KEY /= 10 then
                        STATE <= KEY_ENTRY;
                        TIMEOUT <= 10;
                    end if;
                when KEY_ENTRY =>
                    if ALARM_BUTTON = '1' then
                        STATE <= SET_ALARM_TIME;
                    elsif TIME_BUTTON = '1' then
                        STATE <= SET_CURRENT_TIME;
                    elsif KEY /= 10 then
                        TIMEOUT <= 10;
                    else if TIMEOUT = 0 then
                        STATE <= SHOW_TIME;
                    else TIMEOUT <= TIMEOUT - 1;
                    end if;
                end if;
            end if;
        end if;
    end process STATE_MACHINE;
end architecture RTL of ALARM_CONTROLLER;

```

```

        when SHOW_ALARM =>
            if ALARM_BUTTON = '0' then
                STATE <= SHOW_TIME;
            end if;
        when SET_ALARM_TIME =>
            STATE <= SHOW_TIME;
        when SET_CURRENT_TIME =>
            STATE <= SHOW_TIME;
        when others => STATE <= SHOW_TIME;
        -- no meaning until mapped to logic!
    end case;
end if; -- closes asynch clock process construct

end process STATE_MACHINE;

-- Now assign outputs depending on internal state ...

SHOW_NEW_TIME <= '1' when STATE = KEY_ENTRY else '0';

SHOW_A <= '1' when STATE = SHOW_ALARM else '0';

LOAD_NEW_A <= '1' when STATE = SET_ALARM_TIME else '0';

LOAD_NEW_C <= '1' when STATE = SET_CURRENT_TIME else '0';

end RTL;

```

ACLK.VHD

```

-----
-- Copyright Esperan 1996
-- This is the code for a controller of the alarm clock used
-- in the labs of the Esperan Introduction to VHDL
-- Based Design Training Workshop
-- Revision Control
-- 0.1 First attempt
-----

Library IEEE;
use IEEE.Std_Logic_1164.all;
use work.P_ALARM.all;
use work.P_DISPLAY.all;

entity ALARM_CLOCK is
    port ( CLK, RESET      : in std_ulogic;
          KEYS             : in std_ulogic_vector(0 to 9);
          ALARM_BUTTON     : in std_ulogic;
          TIME_BUTTON      : in std_ulogic;
          SOUND_ALARM      : out std_ulogic;
          DISPLAY          : out T_DISPLAY );
end ALARM_CLOCK;

architecture STRUCT of ALARM_CLOCK is

    component ALARM_COUNTER
        port( NEW_CURRENT_TIME : in T_CLOCK_TIME;
              LOAD_NEW_C       : in std_ulogic;
              CLK, RESET       : in std_ulogic;
              CURRENT_TIME     : out T_CLOCK_TIME );
    end component;

    component ALARM_REG
        port( NEW_ALARM_TIME   : in T_CLOCK_TIME;
              LOAD_NEW_A       : in std_ulogic;
              CLK               : in std_ulogic;
              RESET             : in std_ulogic;
              ALARM_TIME       : out T_CLOCK_TIME );
    end component;

    component DISPLAY_DRIVER2
        port ( ALARM_TIME, CURRENT_TIME,
              KEYPAD           : in T_CLOCK_TIME;
              SHOW_A, SHOW_NEW_TIME : in std_ulogic;
              SOUND_ALARM      : out std_ulogic;
              DISPLAY          : out T_DISPLAY );
    end component;

```



```

end component;

component ALARM_CONTROLLER
port( CLK      : in std_ulogic;
      RESET    : in std_ulogic;
      KEY      : in integer range 0 to 10;
      ALARM_BUTTON : in std_ulogic;
      TIME_BUTTON : in std_ulogic;
      LOAD_NEW_A : out std_ulogic;
      SHOW_A     : out std_ulogic;
      SHOW_NEW_TIME : out std_ulogic;
      LOAD_NEW_C : out std_ulogic );
end component;

component KEYPAD_BUFFER
port(KEY      : in integer range 0 to 10;
      CLK     : in std_ulogic;
      RESET   : in std_ulogic;
      NEW_TIME : out T_CLOCK_TIME );
end component;

component DECODER
port (KEYPAD      : in std_ulogic_vector (0 to 9);
      VALUE       : out integer range 0 to 10 );
end component;

signal NEW_TIME      : T_CLOCK_TIME;
signal ALARM_TIME    : T_CLOCK_TIME;
signal CURRENT_TIME  : T_CLOCK_TIME;
signal KEY           : integer range 0 to 10;
signal LOAD_NEW_A    : std_ulogic;
signal LOAD_NEW_C    : std_ulogic;
signal SHOW_A        : std_ulogic;
signal SHOW_NEW_TIME : std_ulogic;

begin
-----
-- instantiate blocks
-----

U_ALARM_COUNTER : ALARM_COUNTER
port map ( NEW_CURRENT_TIME => NEW_TIME,
          LOAD_NEW_C       => LOAD_NEW_C,
          CLK              => CLK,
          RESET            => RESET,
          CURRENT_TIME     => CURRENT_TIME );

U_ALARM_REG : ALARM_REG
port map ( NEW_ALARM_TIME  => NEW_TIME,
          LOAD_NEW_A       => LOAD_NEW_A,
          CLK              => CLK,
          RESET            => RESET,
          ALARM_TIME       => ALARM_TIME );

U_DISPLAY_DRIVER : DISPLAY_DRIVER2
port map (ALARM_TIME      => ALARM_TIME,
          CURRENT_TIME    => CURRENT_TIME,
          KEYPAD          => NEW_TIME,
          SHOW_A          => SHOW_A,
          SHOW_NEW_TIME   => SHOW_NEW_TIME,
          SOUND_ALARM     => SOUND_ALARM,
          DISPLAY         => DISPLAY );

U_ALARM_CONTROL: ALARM_CONTROLLER
port map( CLK      => CLK,
          RESET    => RESET,
          KEY      => KEY,
          ALARM_BUTTON => ALARM_BUTTON,
          TIME_BUTTON => TIME_BUTTON,
          LOAD_NEW_A => LOAD_NEW_A,
          SHOW_A     => SHOW_A,
          SHOW_NEW_TIME => SHOW_NEW_TIME,
          LOAD_NEW_C => LOAD_NEW_C );

U_DECODER: DECODER
port map (KEYPAD      => KEYS,

```

```
                VALUE      => KEY   );  
  
U_KEYPAD_BUFFER :KEYPAD_BUFFER  
  port map (KEY      =>   KEY,  
            CLK       =>   CLK,  
            RESET     =>   RESET,  
            NEW_TIME  =>   NEW_TIME );  
end STRUCT;  
  
configuration CFG_ALARM_CLOCK of ALARM_CLOCK is  
  for STRUCT  
  end for;  
end CFG_ALARM_CLOCK;
```



จัดพิมพ์และเผยแพร่โดย

ฝ่ายออกแบบวงจรรวม

ศูนย์วิจัยและพัฒนาเทคโนโลยีไมโครอิเล็กทรอนิกส์

ศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ

73/1 ถนนพระราม 6, ราชเทวี, กทม. 10400

โทรศัพท์ (+662) 739-2185...95 ต่อ 600 โทรสาร (+622) 739-2198

<http://tmec.nectec.or.th>