VHDL 입문

국 일 호

AnsLab Co.,Ltd.

E-Mail :

goodkook@nms.anslab.co.kr

URL:

http://www.anslab.co.kr

Previous Knowledge …

- Digital System
  - Number & Codes
  - Combinatorial Logic (Gates)
  - Sequential Logic (F/F, Registers, Counters, State-Machine)
  - CAD experience (Schematics?)
- **프로그래밍 언어**
  - C ? Pascal ? BASIC ? Fortran ? Cobol ?
  - HDL 도 Language 다!
  - **하지만, HDL 프로그래밍 언어처럼 하면 안 된다!!!**

ASIC/VHDL 설계 방법 개요

ASIC Overview
HDL vs. Programming Language
About PLD : Architecture & Prog.

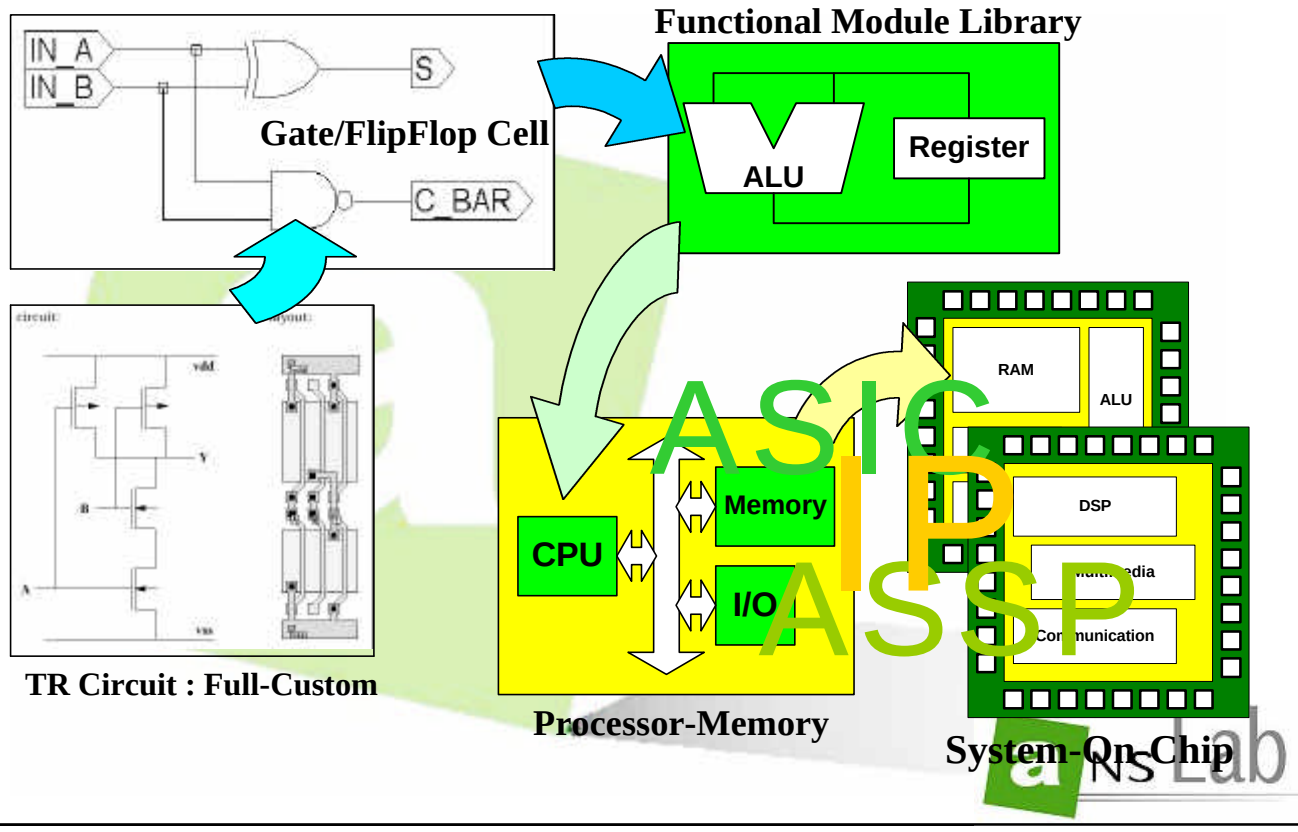
반도체 설계

- 반도체
  - Analog 반도체 : Amplifier
  - Digital 반도체 : Simple TR. Switch
- 반도체 설계 자동화
  - Analog 반도체 : 수동 설계, Circuit Simulation(SPICE)
  - Digital 반도체 : 완전자동화, Logic Simulation
- 반도체 설계규모
  - Analog 반도체 : RC+TR (<10)
  - Digital 반도체 : Switching TR. (>10M)

Digital 반도체 설계 수준

- Transistor Circuit 수준
  - TR Circuit : Full-Custom
- Gate/FlipFlop Cell 수준
  - Cell Library : Semi-Custom (SSI)
- Functional Module 수준
  - Adder, Multiplier, Register, Counter (LSI)
- Processor-Memory 수준
  - Processor, Memory (VLSI)
- System 수준
  - SOC (System-On-Chip)

Digital 반도체 설계 수준



반도체 설계 방법

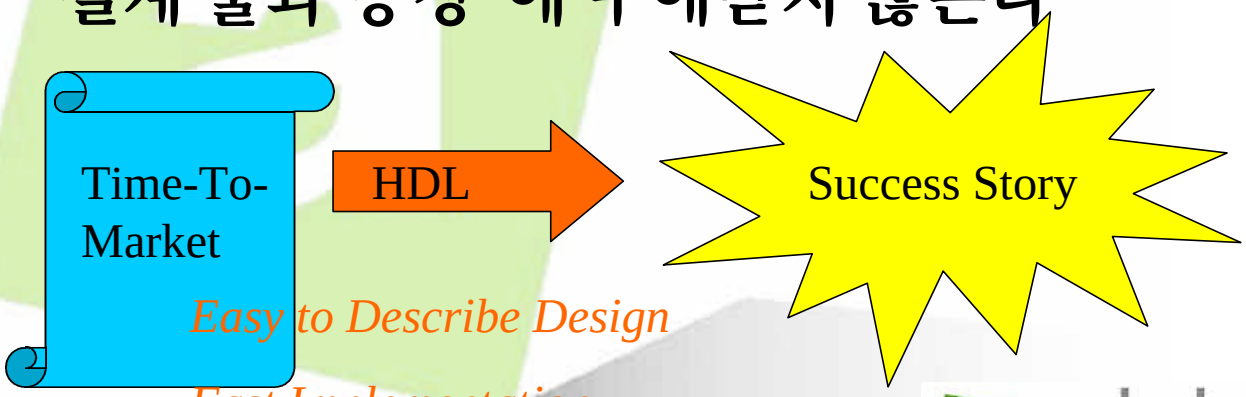
- Transistor Circuit
  - Layout Editor & Circuit/Switch Simulator
- Gate/FlipFlop
  - Schematic Editor & Logic Simulator
- ASIC
  - Hardware Description Language (HDL)
    - Verilog HDL/VHDL 표준화
  - HDL & Schematic Mixed

ASIC 설계

- ASIC
  - Semi-Custom (Standard Cell/Gate-Array)
  - PLD (SPLD/EPLD/CPLD/FPGA)
- PLD 의 발전
  - 간단한 Logic ➡ PAL/GAL ➡ TTL 소멸?
  - CPLD & FPGA 규모 극적인 확대 ➡ >100K Gate
  - 10,000~50,000 Gate 보편화
    ➡ 새로운 설계 방법론 필요 ⬅ Time-To-Market

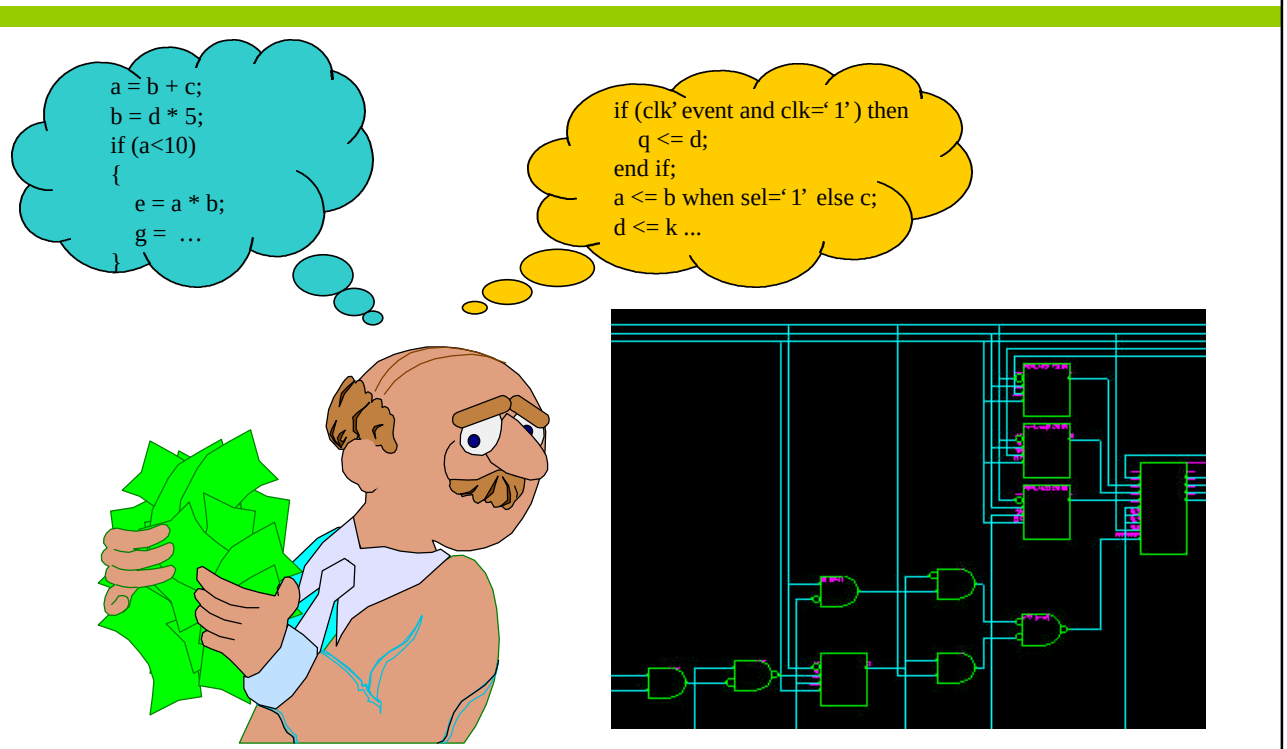
왜 HDL을 사용하는가?

- "아이디어"를 쉽게 표현 할수 있다
- "설계관리"가 용이하다
- "설계 툴과 공정"에 구애받지 않는다



ANS Lab

HDL vs. Programming Language



HDL : Language ?

- Hardware Description Language
  - Statements
    - Sequential
    - Concurrent
  - Syntax
  - Keywords
  - Objects
  - Operators

HDL : Hardware Description ?

- Hardware Description Language

Netlist/Dataflow	Behavior/Inference FF

-- *AND Gate*
a <= b AND c;
 	
-- *MUX*
d <= e when f=’1’ else g;
	-- *Flip-Flop*
	if rising_edge(clk) then
	  q <= d;
	end if;
	
	-- *MUX*
	if s=’1’ then
	  d <= e;
	else
	  d <= f;
	end if;

HDL and Programming Language

	Programming Language	Hardware Language
Purpose	Software (Executable Binaries)	Hardware (Hardwired Logics)
Entry	Text & Visual Tools	
Development	Compile & Link	Compile for Simulation & Synthesis for Hardwire
Debugging	Executes & View Result (Text or Graphics)	Simulation & View Waveform
Statement	Sequential Only	Basically Concurrent Sequential for Behavior Description

HDL：Concurrent & Sequential Statement

- Order of Statements

C Programming Language	Hardware Language
d = a * e; a = b + c;	a <= b OR c; d <= a AND c;
a = b + c; d = a * e;	

- Objects

C Programming Language	Hardware Language
All Registered	Signal Name & Registered

HDL : Recursive

- Recursive Statements

C Programming Language	Hardware Language (erroneous*)
d = a + c; a = b * d;	d <= a OR c; a <= b AND d;

** Use Latch or Flip-Flop to be recursive algorithm equiv.*

- Latch

Recursive (NO Good*)	Behavior

q <= a NAND q_bar;
q_bar <= b NAND q;
	if (l='1') then
	q <= d;
	end if;

*** No real-gates, just equivalent functions on PLD*

HDL : Multiple Drive

- Multiple Drive Error (cause ERC/DRC Violation)

C Programming Language	Hardware Language (Error)

a <= b + c;
…
a <= d * e;
	a <= b OR c;
	…
	a <= d AND e;

- Resolve Multiple Drived Signal
  - Multi-Valued Signal & Resolution Function
    - std_logic : 9-value {' U' , ' X' ' 0' ' 1' ' Z' ' W' ' L' ' H' ' -' }
    - bit : 2-value, {' 0' , ' 1' }, NOT RESOLVED

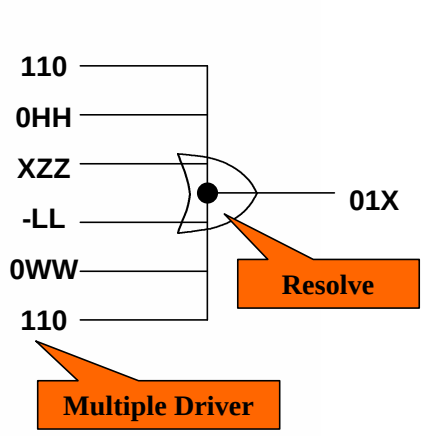
Multiple Drive

- On Concurrent Assignment, multiple source to one signal must be resolved
- Multiple Drive Problem is common when modeling the buses with three-state drivers
- std_logic_1164 provides resolved type and resolution function “resolved”.

subtype **std_logic** is **resolved std_ulogic;**

Resolution Function (IEEE 1164)

```
-------------------------------------------------------------
-- resolution function
-------------------------------------------------------------
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' )  -- | - |
);

FUNCTION resolved ( s : std_ulogic_vector ) RETURN std_ulogic IS
    VARIABLE result : std_ulogic := 'Z';
BEGIN
    IF (s'LENGTH = 1) THEN RETURN s(s'LOW);
    ELSE
        FOR i IN s'RANGE LOOP
            result := resolution_table(result, s(i));
        END LOOP;
    END IF;
    RETURN result;
END resolved;
```



Resolve Multiple Drive Error

```
entity test_resolver is
   port ( a, b : bit ;
          o : out bit ) ;
end test_resolver ;
architecture dataflow of test_resolver is
signal tmp : bit ;
begin
   tmp <= a ;
   tmp <= b ;
   o   <= tmp ;
end dataflow ;
```

write resolution function

```
entity test_resolver is
   port ( a, b : bit ;
          o : out bit ) ;
end test_resolver ;

architecture dataflow of test_resolver is

-- Write the resolution function that ANDs the elements:
function my_and_resolved (a : bit_vector) return bit is
   variable result : bit := '1' ;
begin
   for i in a'range loop
      result := result AND a(i) ;
   end loop ;
   return result ;
end my_and_resolved ;

-- Declare the signal and attach the resolution function to it:
signal tmp : my_and_resolved bit ;
begin
   tmp <= a ;
   tmp <= b ;
   o <= tmp ;
end dataflow ;
```

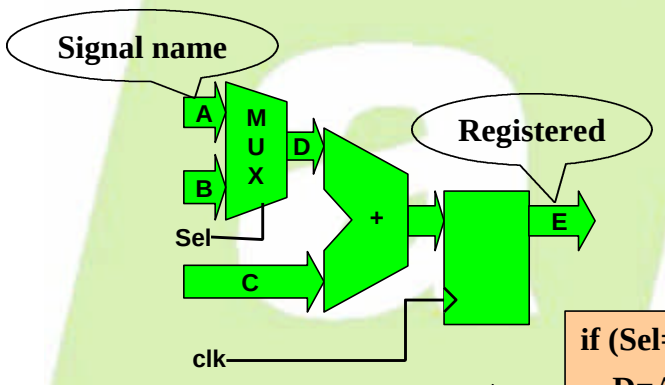
Use resolved type

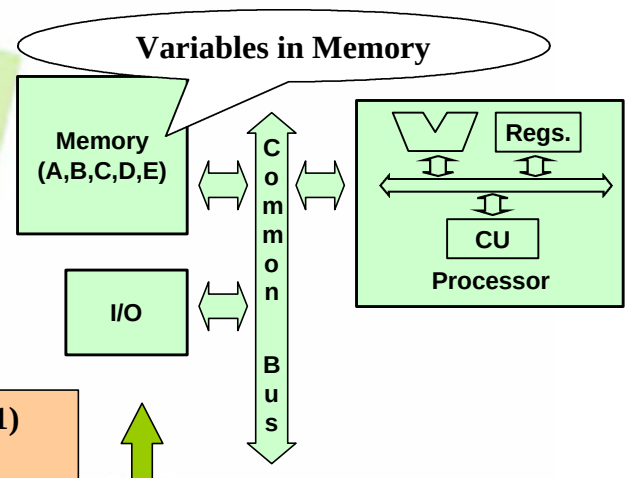
```
library ieee;
use ieee.std_logic_1164.all;
entity test_resolver is
   port ( a, b : std_logic ;
          o : out std_logic ) ;
end test_resolver ;
architecture dataflow of test_resolver is
signal tmp : std_logic ;
begin
   tmp <= a ;
   tmp <= b ;
   o   <= tmp ;
end dataflow ;
```

aNS Lab

HDL vs. Programming Language

- HDL
  - Instancited Unit
  - Register Inference
- Programming Lnguage
  - Processor-Memory






```
if (Sel==1)
   D=A;
else
   D=B;
E=D+C;
```

aNS Lab

It's VHDL

What is VHDL
Abstraction Level
HDL Design Flow

What is VHDL ?

- **V**HSIC **H**ardware **D**escription **L**anguage
  - VHSIC : Very High Speed Integrated Circuits
- Originally Designed for Documentation
- A Simulation Language for Digital System
- Highly Structured, Strongly Typed(ADA-Like)
- A Multi-Purpose Language

VHDL: Multi-Purpose Language

- Sequential Description
- Concurrent Description
- Netlist Language
- Timing Specification
- Self-Contained Test Language ("TestBenches")

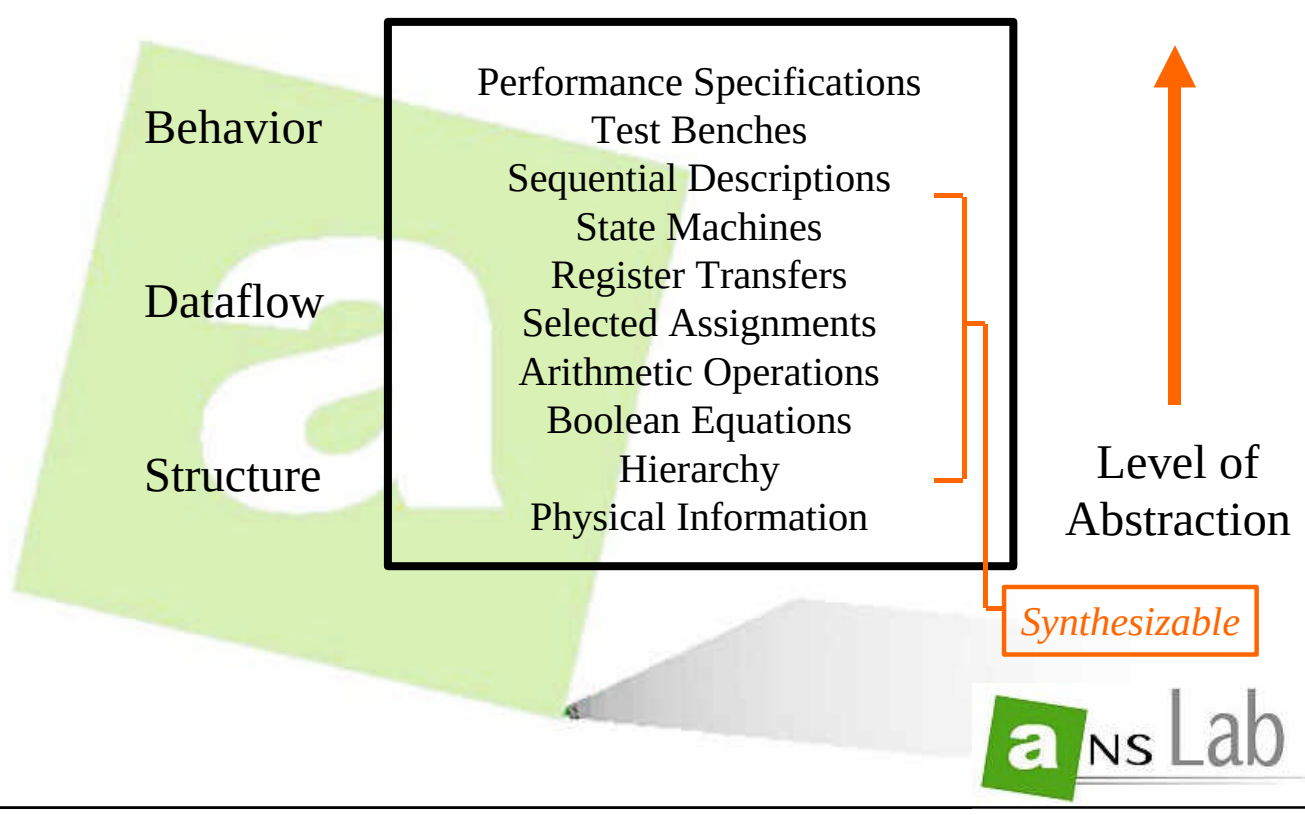
VHDL : History

- First Generated in 1981 under VHSIC program
- Development Started in 1983
- Maintained by IEEE since 1987(IEEE1076-87)
- MIL-STD-454 Requires VHDL for All ASICs Delivered to DoD (1988)
- IEEE1164: Multi Valued Logic System, 1992
- Revised Standard "VHDL-93"
- VITAL Initiative Defines Standard Format for Gate Libraries,1994

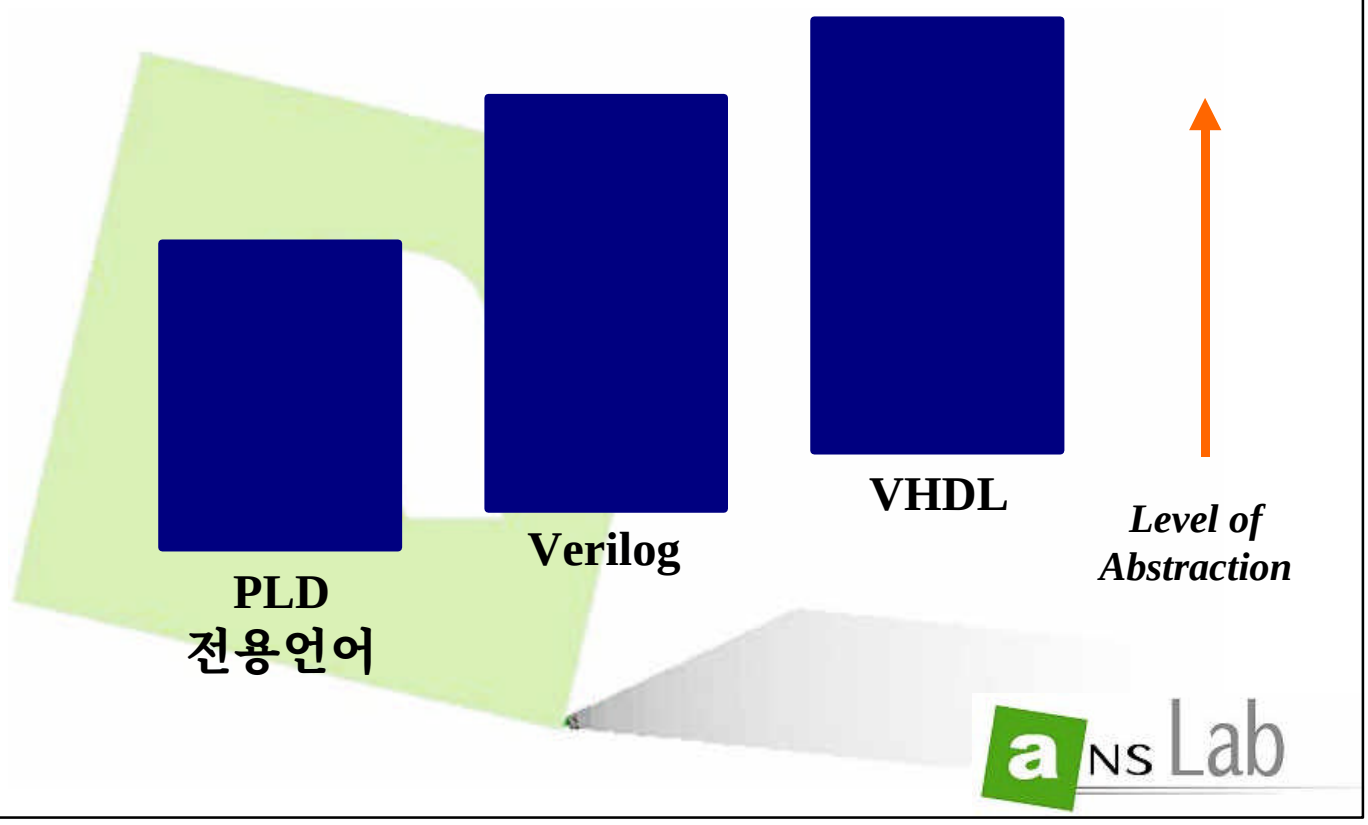
VHDL : Subgroups

- 8 Subgroups
- IEEE 1076 Subgroups
  - IEEE 1076.1 - Analog Standard
  - IEEE 1076.2 - Math Package
  - IEEE 1076.3 - Synthesis Package (std_logic, arithmetic)
  - IEEE 1076.4 - VITAL
- Other EDA Industry WG/SG (http://www.eda.org)
  - SDF : Standard Delay Format (PAR1497)
  - VHDL-MW : Microwave Extension
  - WAVES : Waveform and Vector Exchange(STD1029.1-91)
  - OOVHDL, OMF, VHDL-MW,CODESIGN, ….

Abstraction (추상화) Level



Abstraction Level Comparison



Abstraction Level : Behavioral

- Sequential statements
- Implied registers (like a software programming language)
- Few (expensive) Synthesis tools available at this level
- Simulation, Modeling, and Specification

```
/* Programming
   Language */
int a, b, c;
a = b;
b = c;
```

```
-- VHDL
-- testbench clock generator
clock: process
constant PERIOD := 100 ns;
variable c: std_ulogic := '0';
begin
   wait for PERIOD / 2;
   c := not c;
   Clk <= c;
end process;
```

```
-- VHDL
-- Register inference
clock: process(Clk)
begin
   if (Clk=' 1'  AND Clk' EVENT) then
      q <= d;
   end if;
end process;
```

Abstraction Level : Dataflow

- Concurrent assignments
- Explicit registers (like a PLD language)
  - User specify where registers are.
- Most Synthesis tools are at this level.

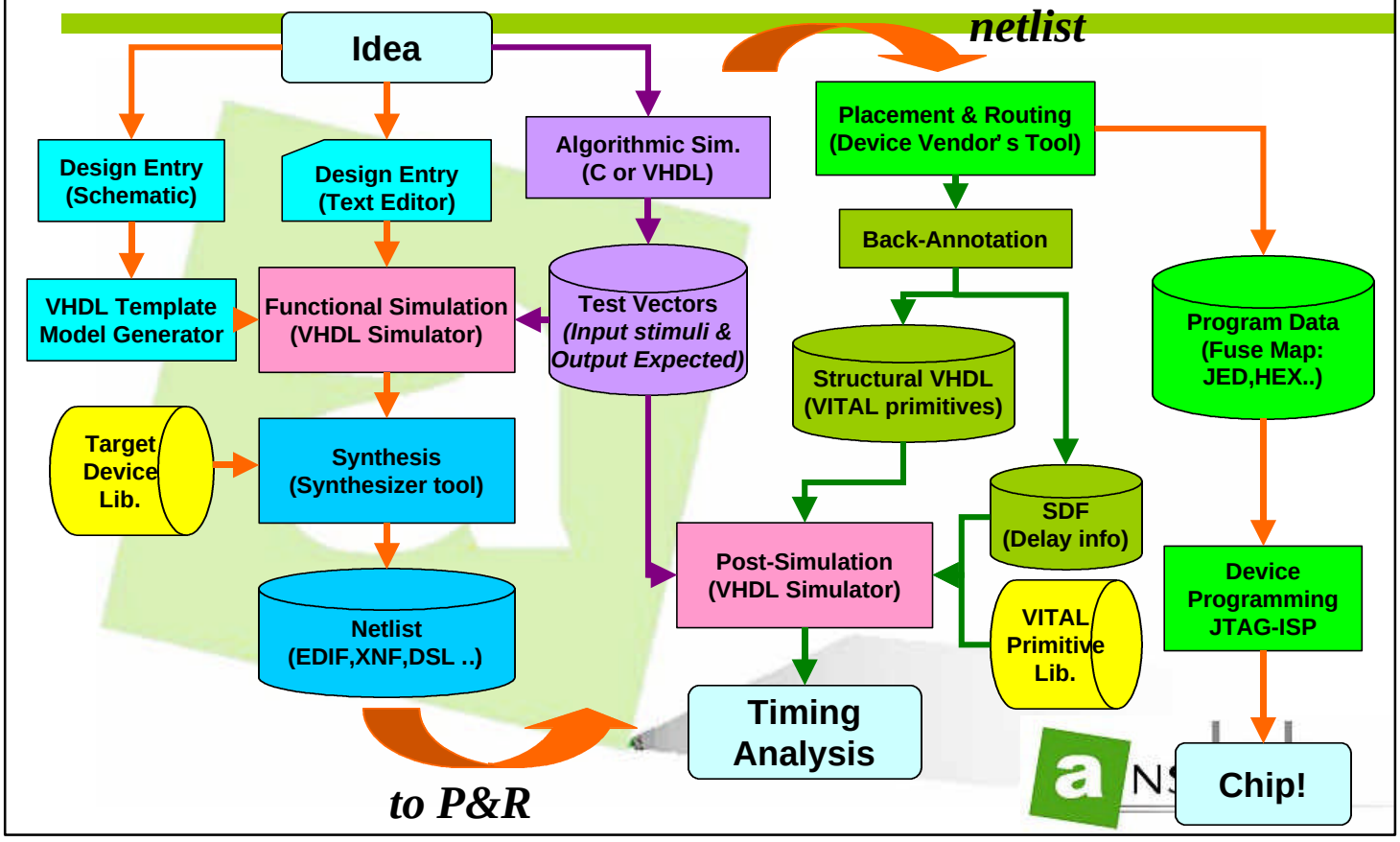
```
signal a,b,c,d,q;
begin
   c <= a and b;
   b <= c and d;
   q <= d when (clk=‘ 1’  and clk’ event);
end;
```


Abstraction Level : Structural

- Connected components (like a netlist)
- Extracted VITAL/Timing models after Place & Route

```
component nand2
   port ( i0, i1 : std_logic; o : out std_logic);
end component;
signal a, b, c, d;
begin
   u1 : nand2 port map (i0 => a, i1 => b, o => c);
   u2 : nand2 port map (i0 => c, i1 => d, o => a);
end;
```

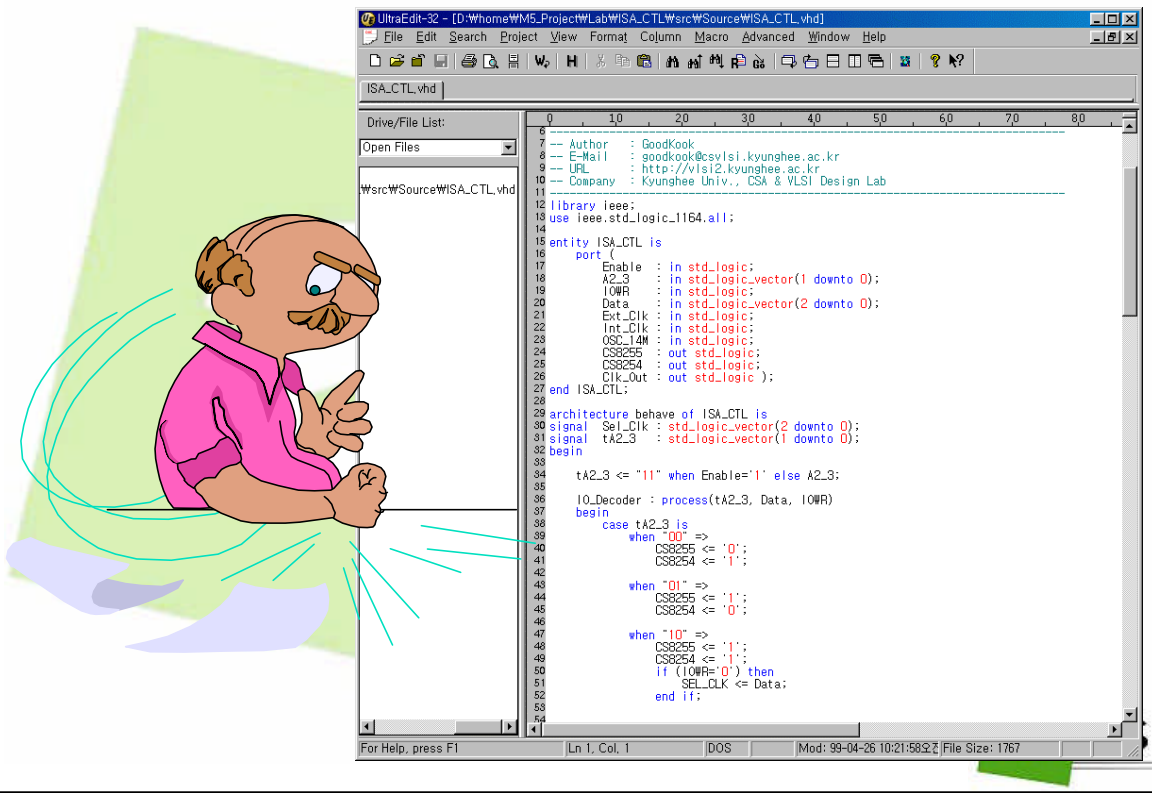
HDL Design Flow



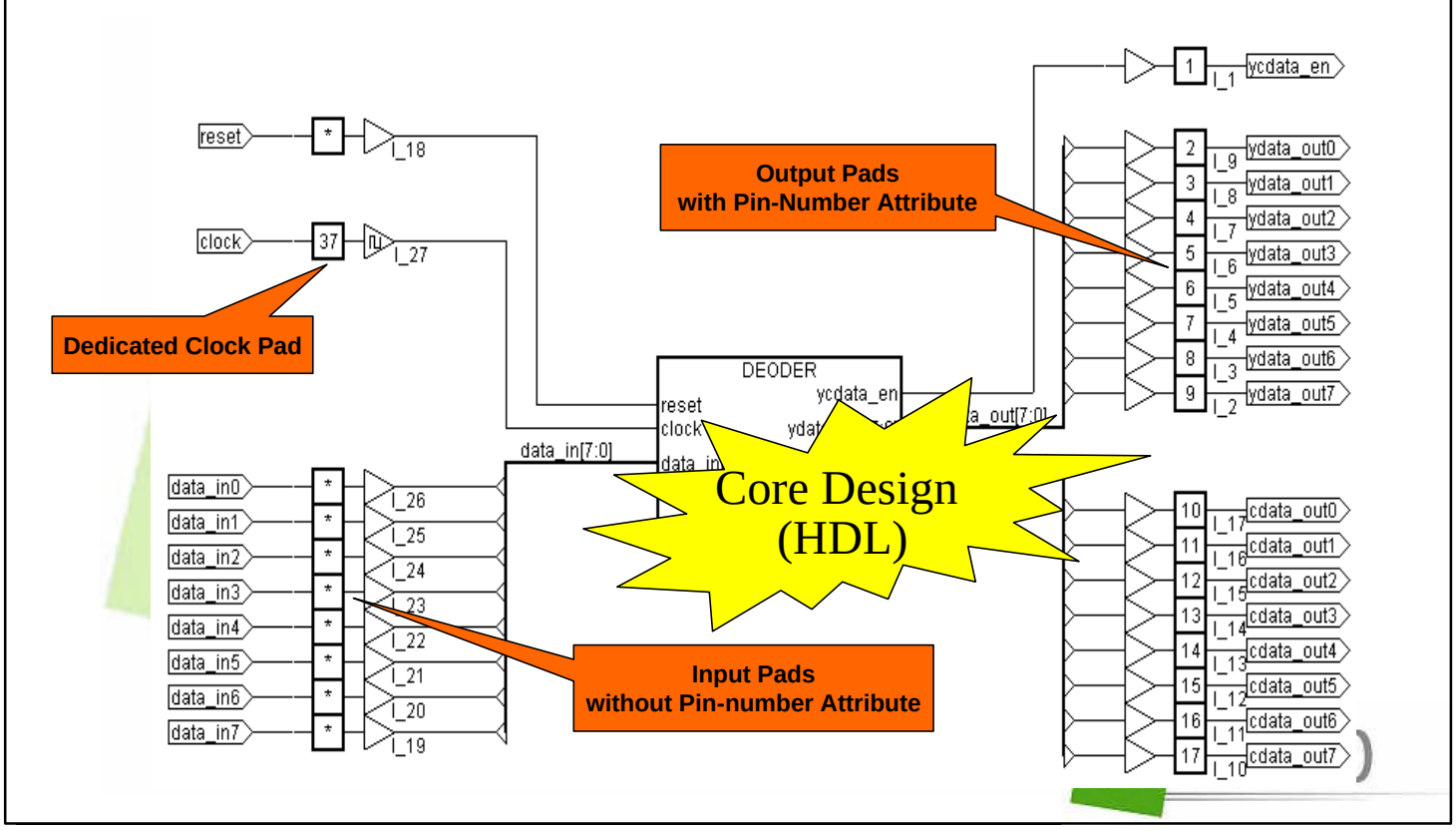
Design Entry (HDL)

- Text Editor
- Language Sensitive Text Editor
  - Keyword coloring
  - Statement Template
- Visual Entry Tools
  - Template & Translate (Structural)
- Testbench
  - Converter (Waveformer)

HDL Design Entry (Text Editor)

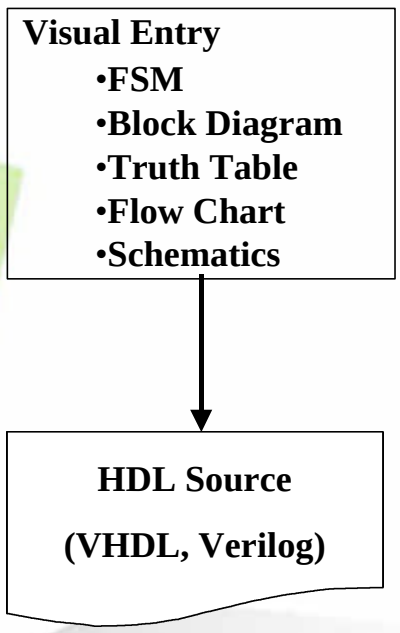


HDL & Schematic Mixed

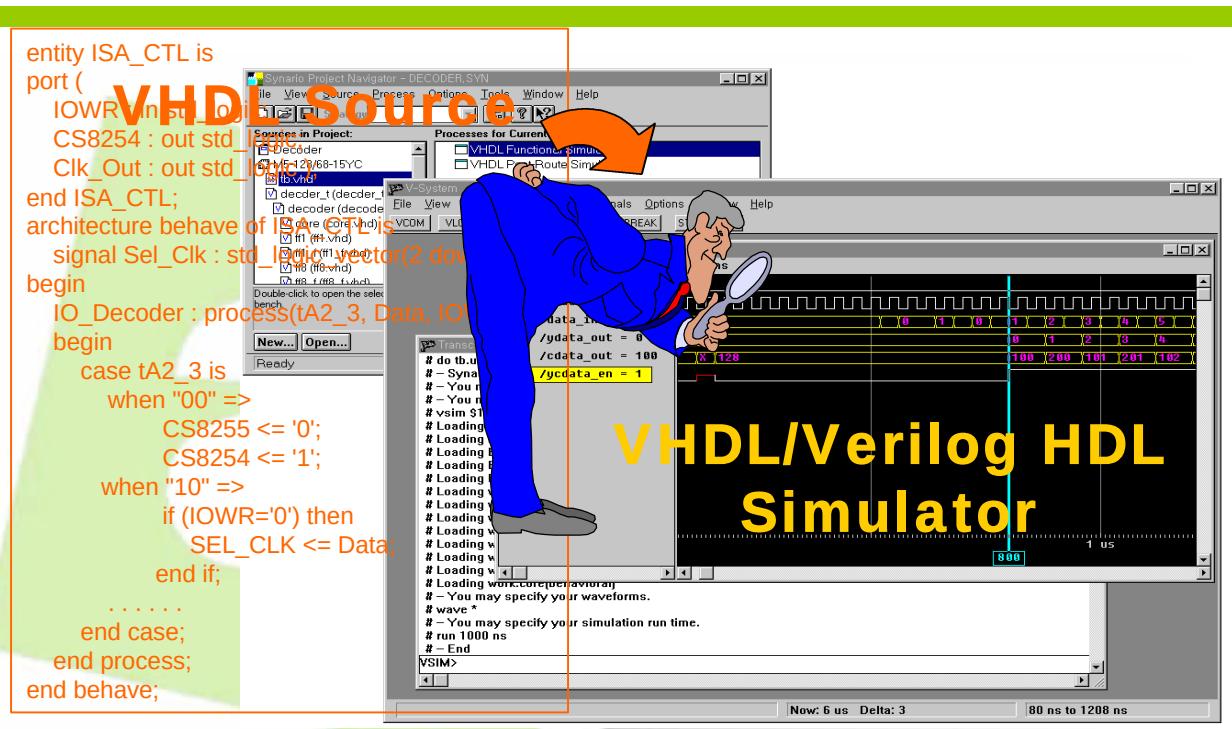


HDL Design Entry (Visual Tools)

- Mentor’ s Renoir
- Aldec’ s ActiveHDL
- StateCAD
- Synario
- Summit VisualHDL



HDL Simulation

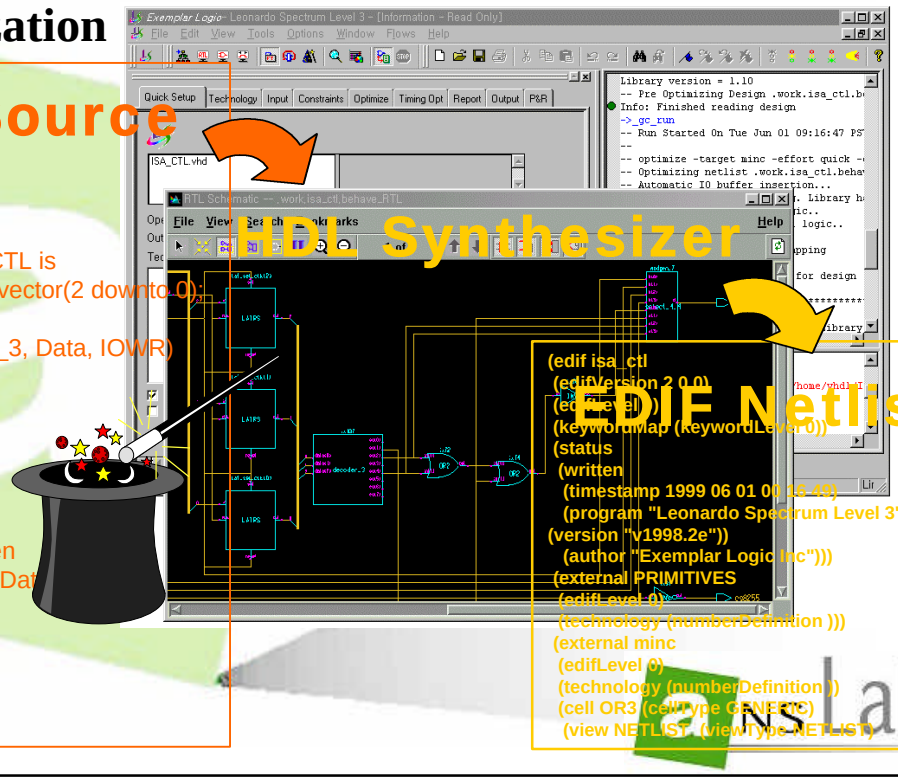


HDL Synthesis

- HDL 구문에서 Hardware 로 변환 (Netlist)
- Logic Optimization

```
entity ISA_CTL is
port (
   IOWR : in std_logic;
   CS8254 : out std_logic;
   Clk_Out : out std_logic );
end ISA_CTL;
architecture behave of ISA_CTL is
   signal Sel_Clk : std_logic_vector(2 downto 0);
begin
   IO_Decoder : process(tA2_3, Data, IOWR)
   begin
      case tA2_3 is
         when "00" =>
               CS8255 <= '0';
               CS8254 <= '1';
         when "10" =>
               if (IOWR='0') then
                  SEL_CLK <= Dat
               end if;
         . . . . . .
      end case;
   end process;
end behave;
```

VHDL Source



HDL Synthesizer

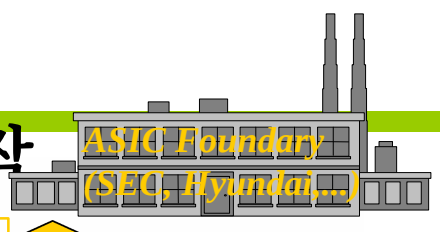
```
(edif isa_ctl
  (edifVersion 2 0 0)
(edifLevel 0)
(keywordMap (keywordLevel 0))
(status
  (written
   (timestamp 1999 06 01 00 16 48)
   (program "Leonardo Spectrum Level 3"
(version "v1998.2e"))
   (author "Exemplar Logic Inc")))
(external PRIMITIVES
  (edifLevel 0)
  (technology (numberDefinition )))
(external minc
  (edifLevel 0)
  (technology (numberDefinition ))
  (cell OR3 (cellType GENERIC)
   (view NETLIST (viewType NETLIST)
```

EDIF Netlist

CSA Lab

Device Implementation

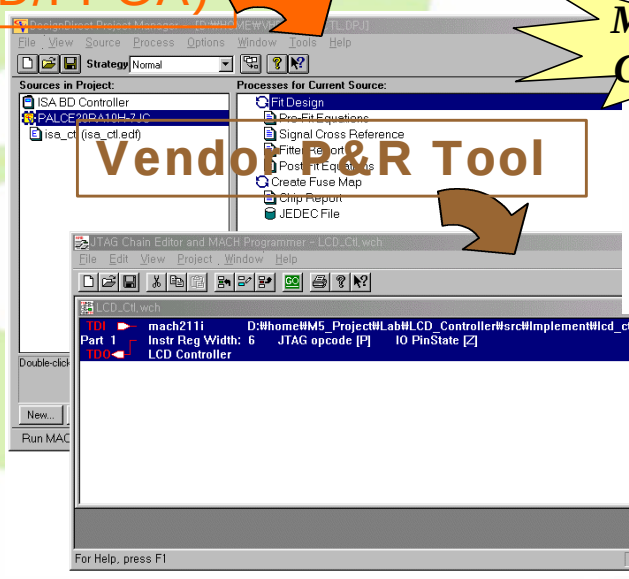
- Place & Route/ASIC Chip 제작



ASIC Foundary
(SEC, Hyundai,...)

Full/Semi-Custom Design(STD Cell/Gate Array)

PLD(CPLD/FPGA)



Vendor P&R Tool

My Custom Chip NOW

JTAG

PC LPT: Port

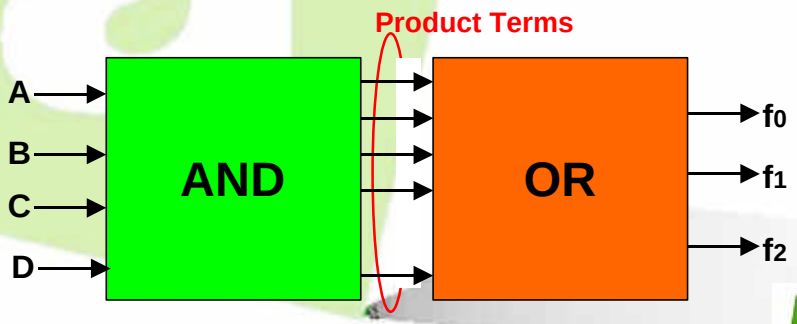
CSA Lab

About PLD

Programmable Logic Device

PLD : “Sum of Products”

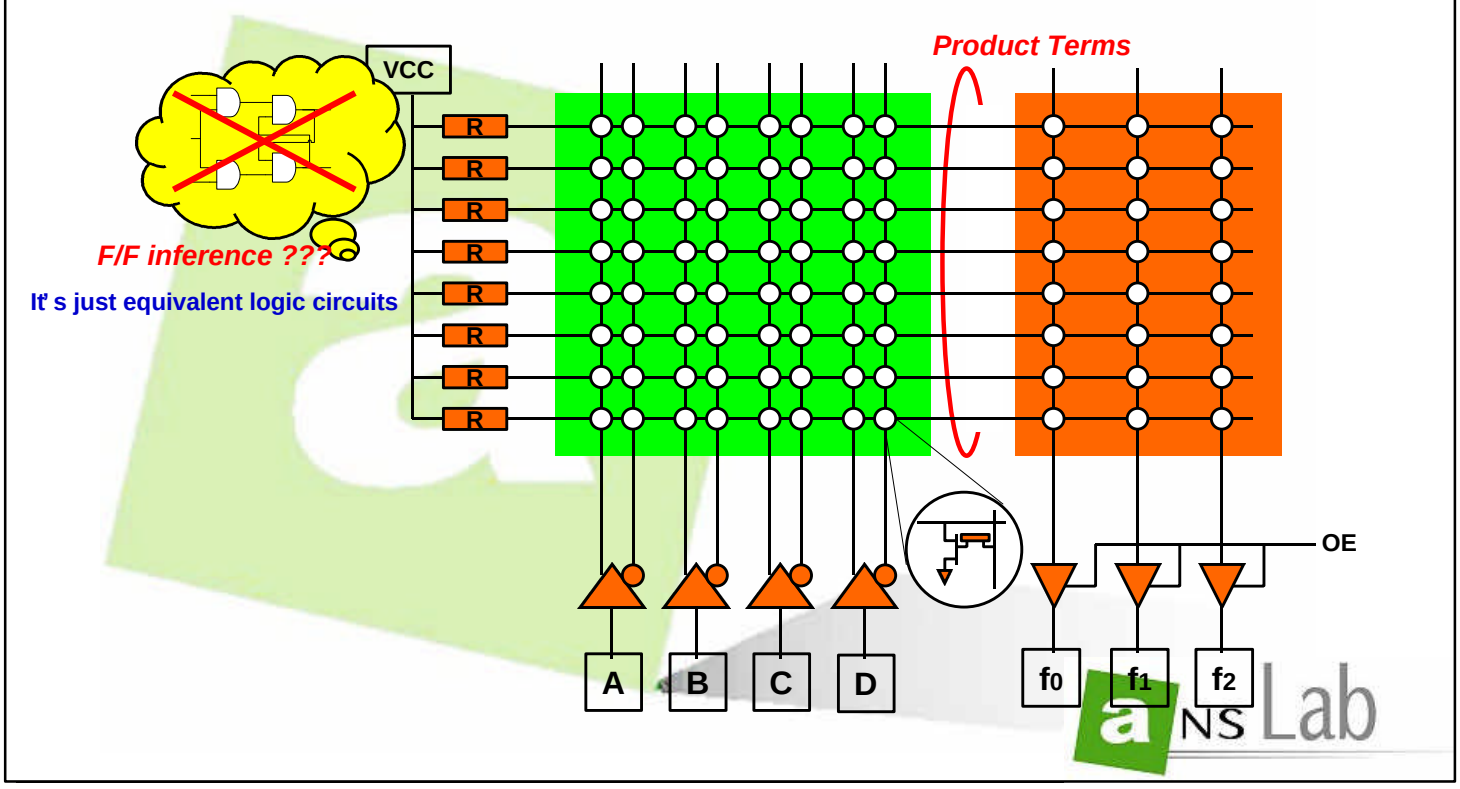
- Programmable Logic Device
- Combinational Logic Circuits
  - “Sum of Products”
    - AND Plane (product term)
    - OR Plane (sum)



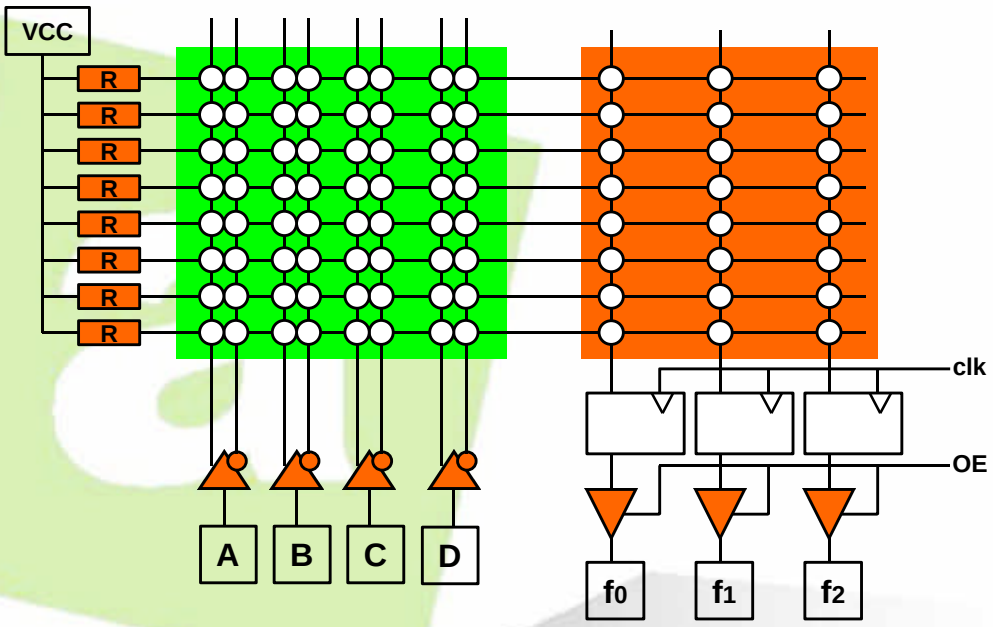
PLD : ROM and PLA

- Combinational Logic Circuit
  - ROM Device
    - AND Plane Fixed (Address Decoder)
    - OR Plane Programmable
  - PLA (Programable Logic Array)
    - AND Plane Programmable
    - OR Plane Programmable

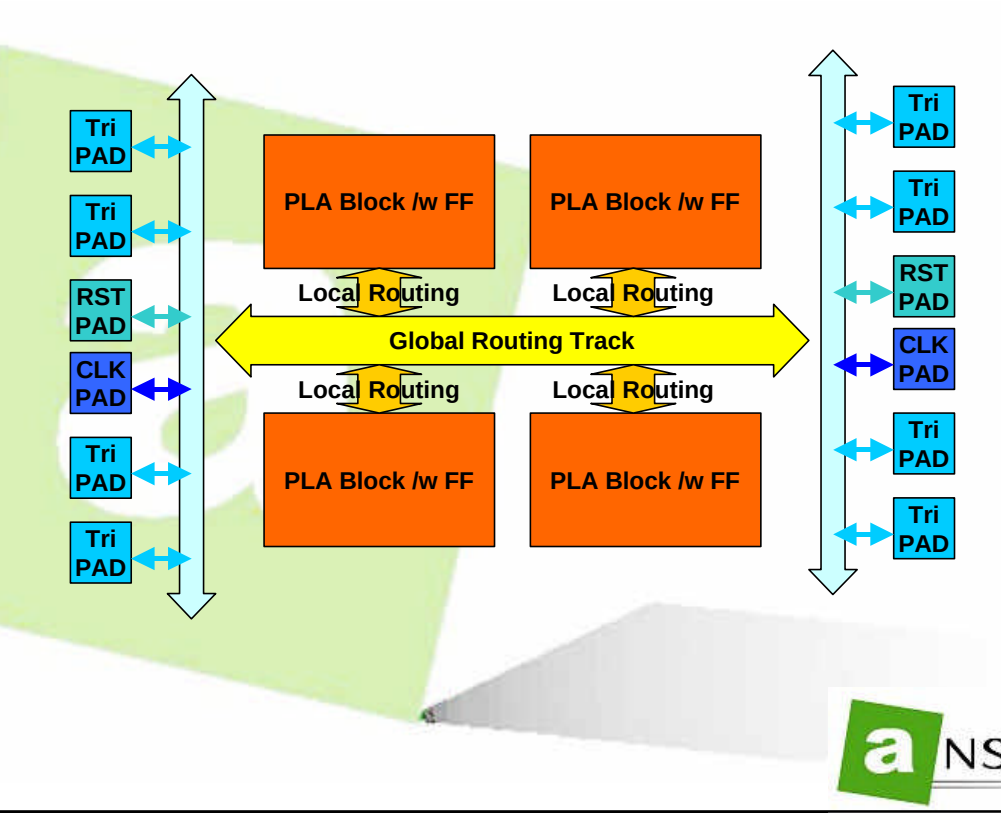
PLD : PLA Internal



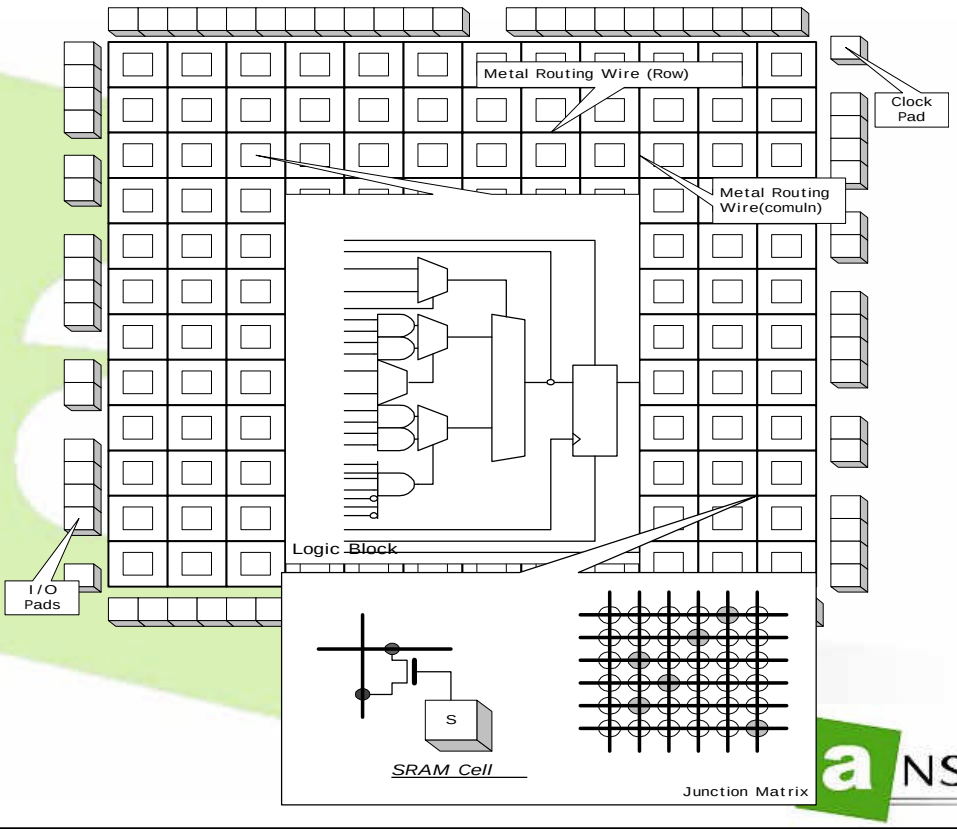
PLD : PLA with output F/F



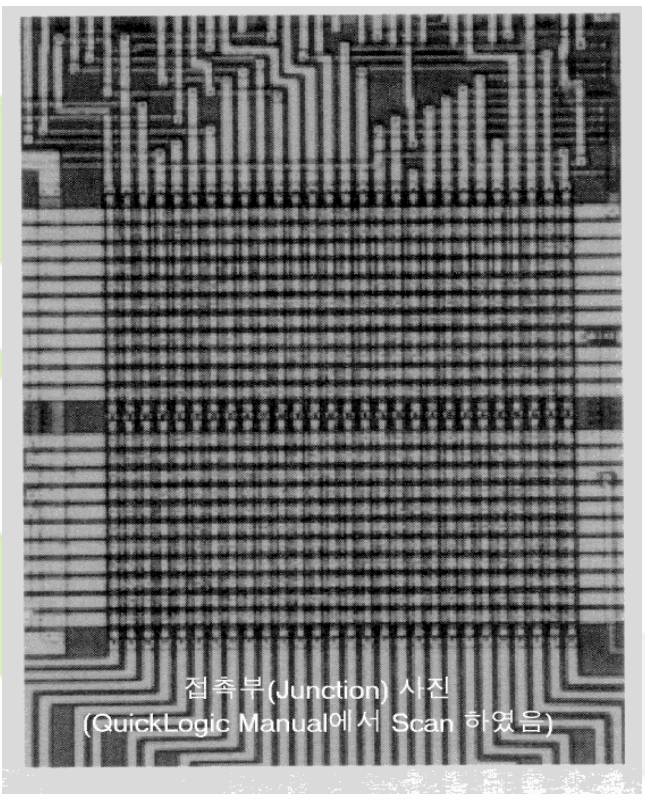
PLD : Complex PLD



PLD : FPGA Internal

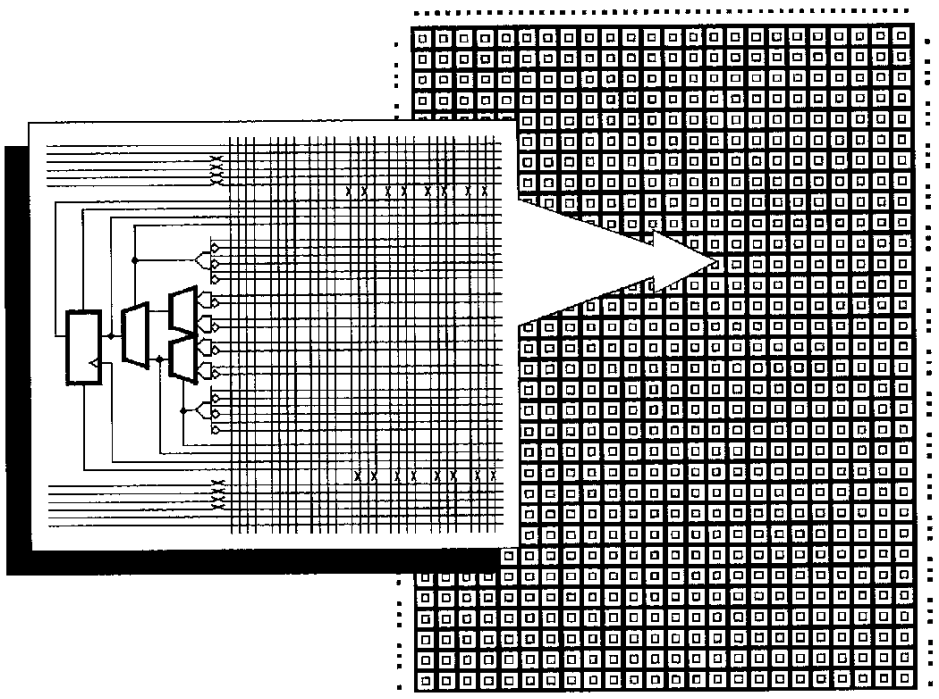


PLD : Interconnect (Junction Photo)

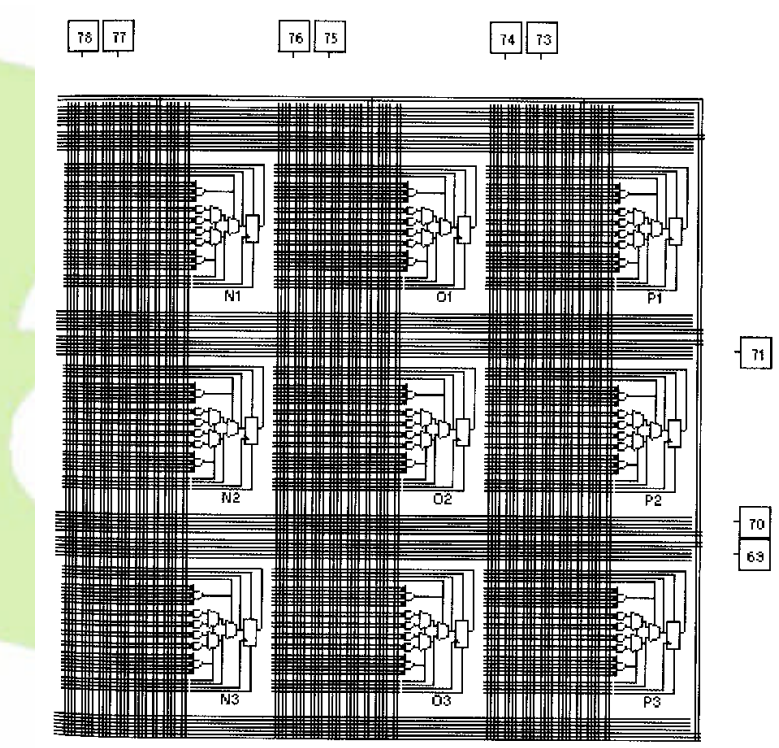


접촉부(Junction) 사진
(QuickLogic Manual에서 Scan 하였음)

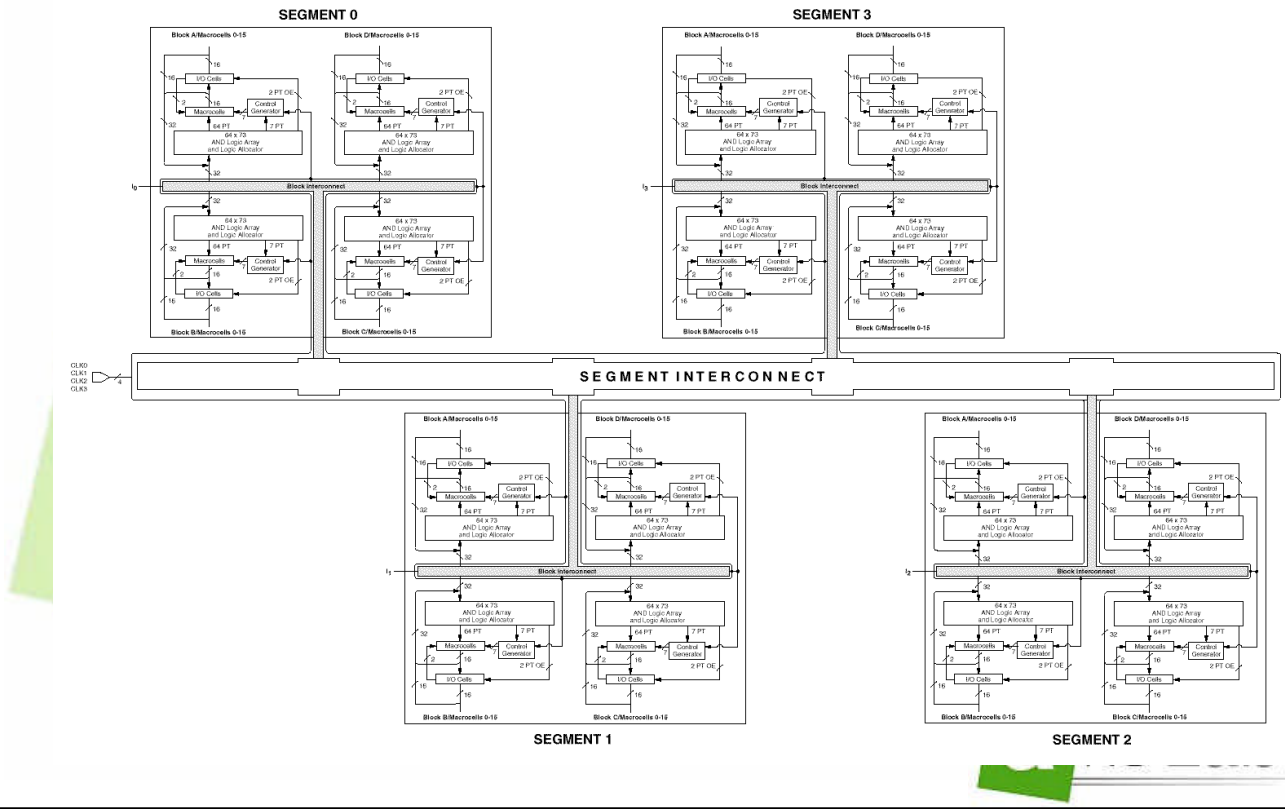
PLD : Logic Cell (QuickLogic)



PLD : Logic Cell (cont' ed)



PLD : CPLD (Mach5 Internal)



PLD : CPLD (Mach5 PLA Block)

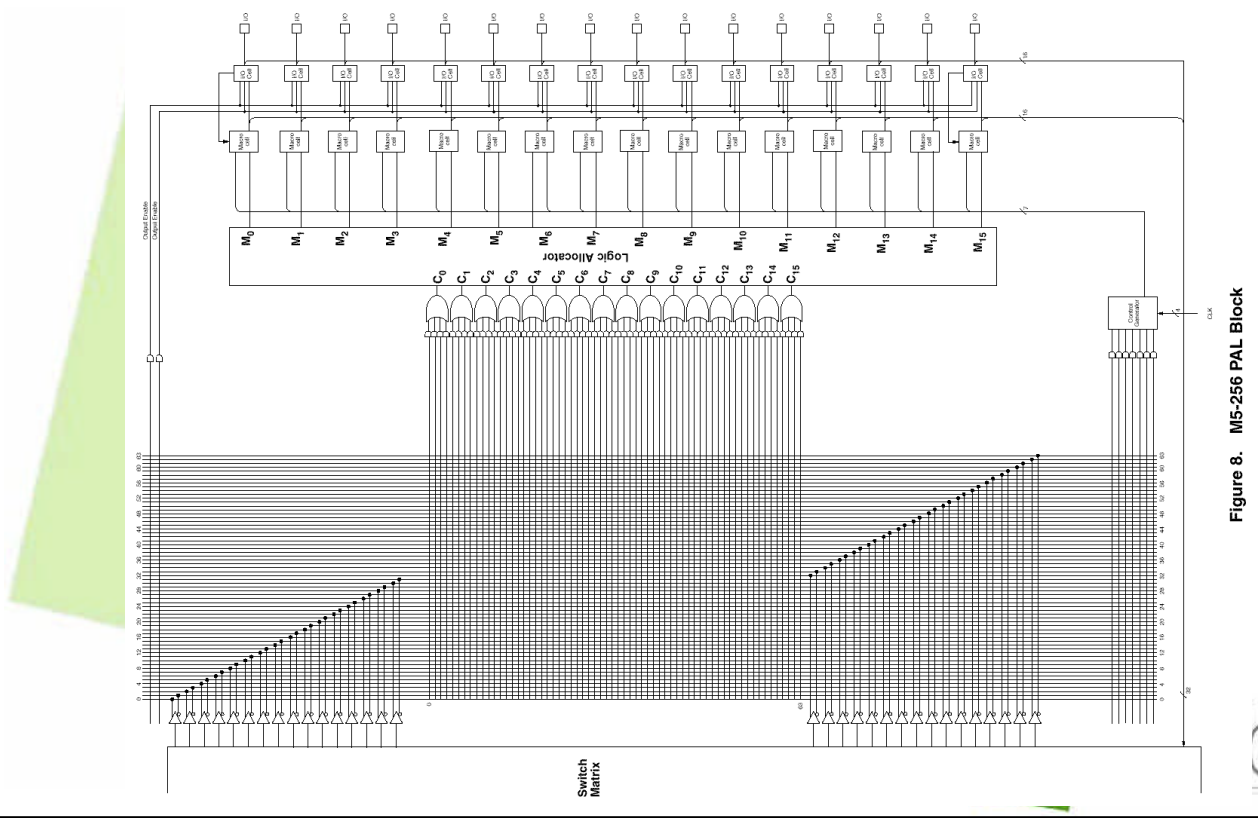
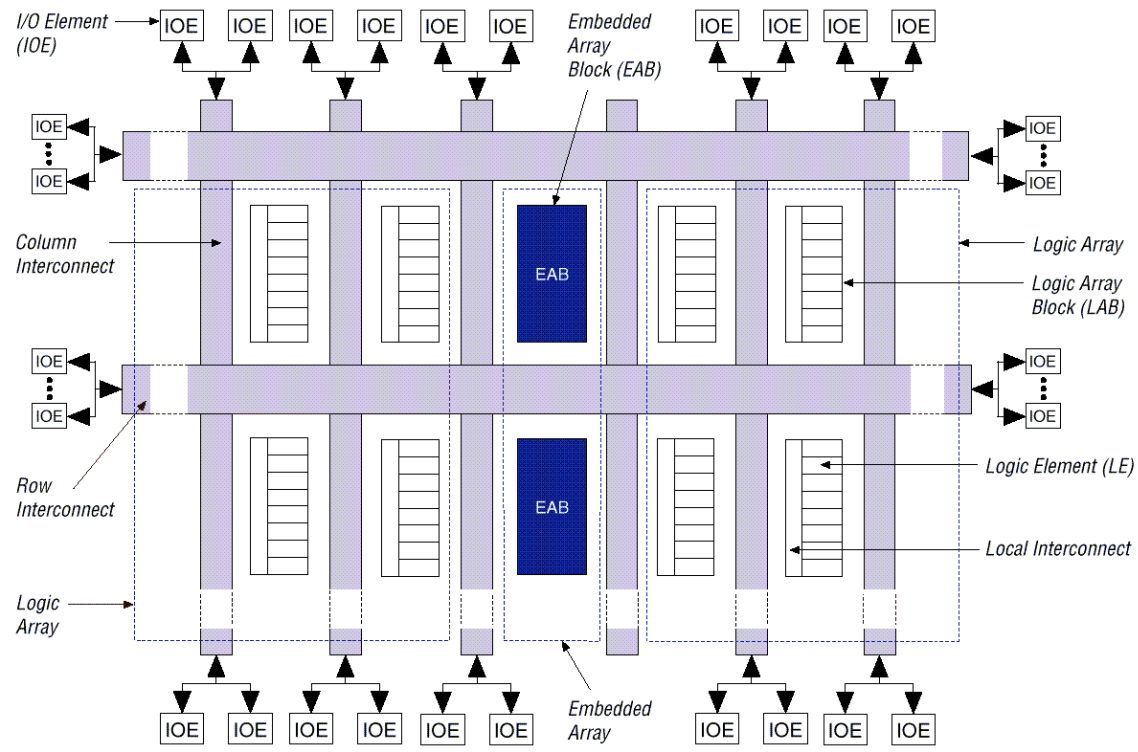


Figure 8. M5-256 PAL Block

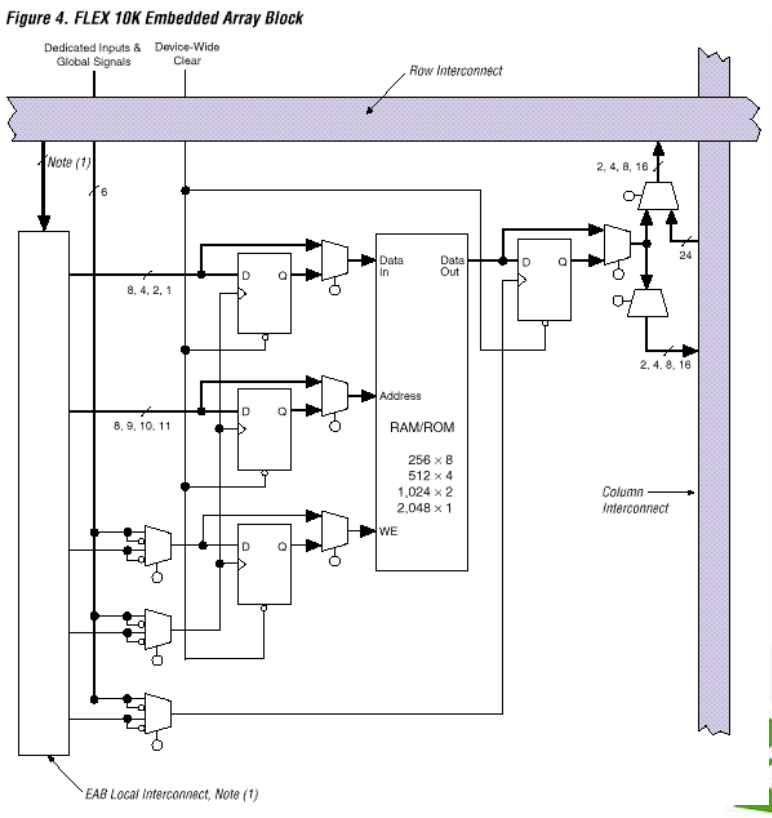
PLD : Flex10k Internal Block

Figure 1. FLEX 10K Device Block Diagram



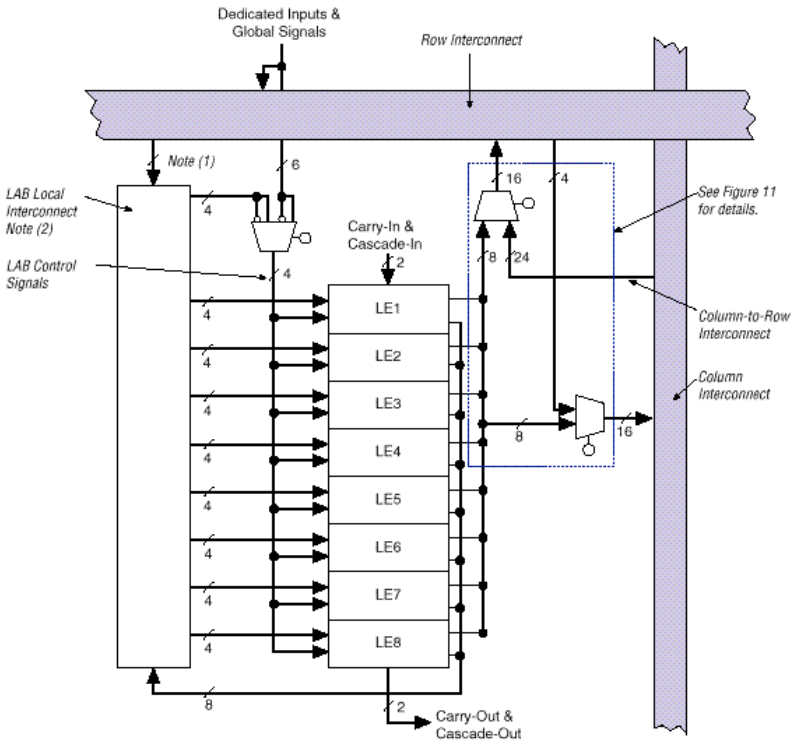
PLD : Flex 10k EAB

Figure 4. FLEX 10K Embedded Array Block



PLD : Flex 10k LAB

Figure 5. FLEX 10K LAB

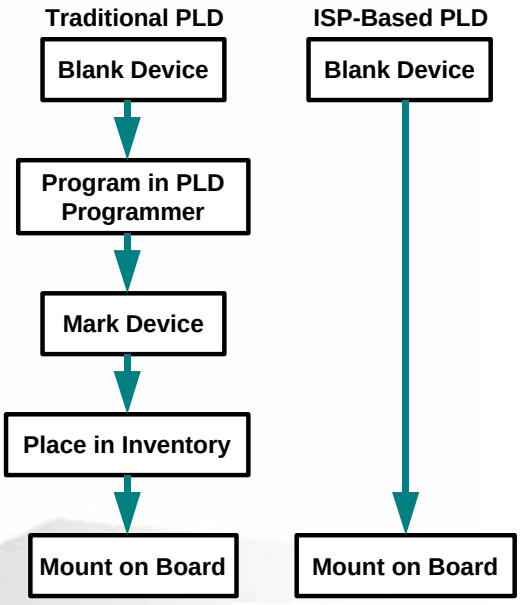


PLD : Programming

- ROM
  - Standard Programming Method
  - Generalized Writing Device
    - PROM
    - UV-EPROM
    - EEPROM
- CPLD/FPGA
  - No Standard internal structure
  - Vendor’ s Writing Device
    - UV-ROM Type
    - PROM Type (OTP, Anti-Fuse)
  - ISP & JTAG
    - EEPROM Type (ISP)
    - SRAM Type

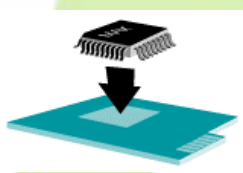
PLD : Programming - ISP

- In-System Programmability
  - Time-to-Market
  - Low price
  - PCB testing capabilities
- JTAG
  - Joint Test Action Group
  - Standard IEEE 1149.1-1990
  - PLD Program/Test
  - Verify PCB & PLD Internal
  - Simplifyed system test



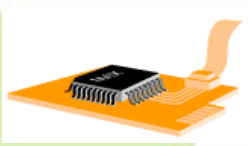
ISP Benefits

Mount Unprogrammed



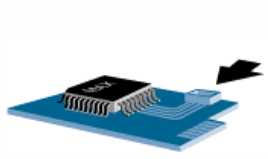
- Eliminates handling of devices
- Prevents bent leads

Program In-System



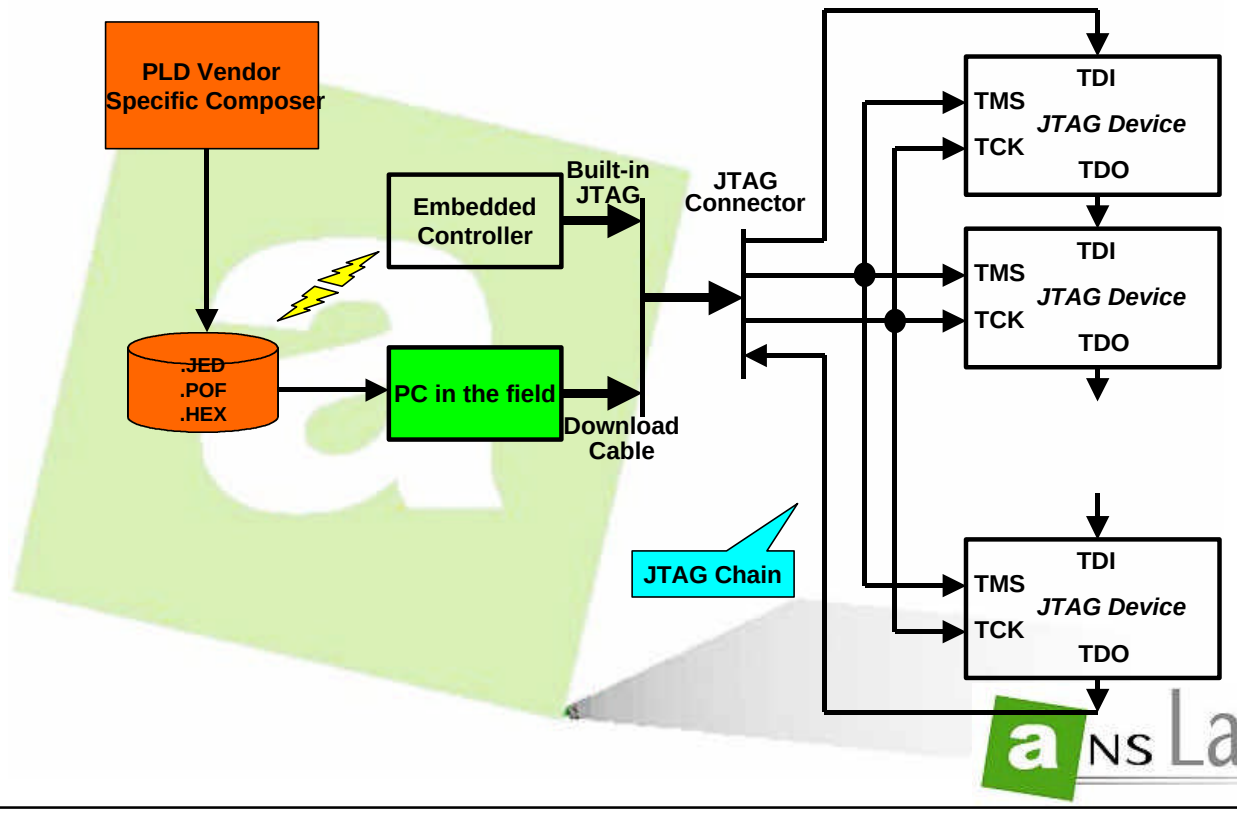
- Allows generic end-product inventory
- Specific test protocol or algorithm can be programmed during prototyping, manufacturing or test flow

Reprogram in the Field



- No need to return system for upgrades
- Add enhancements quickly & easily

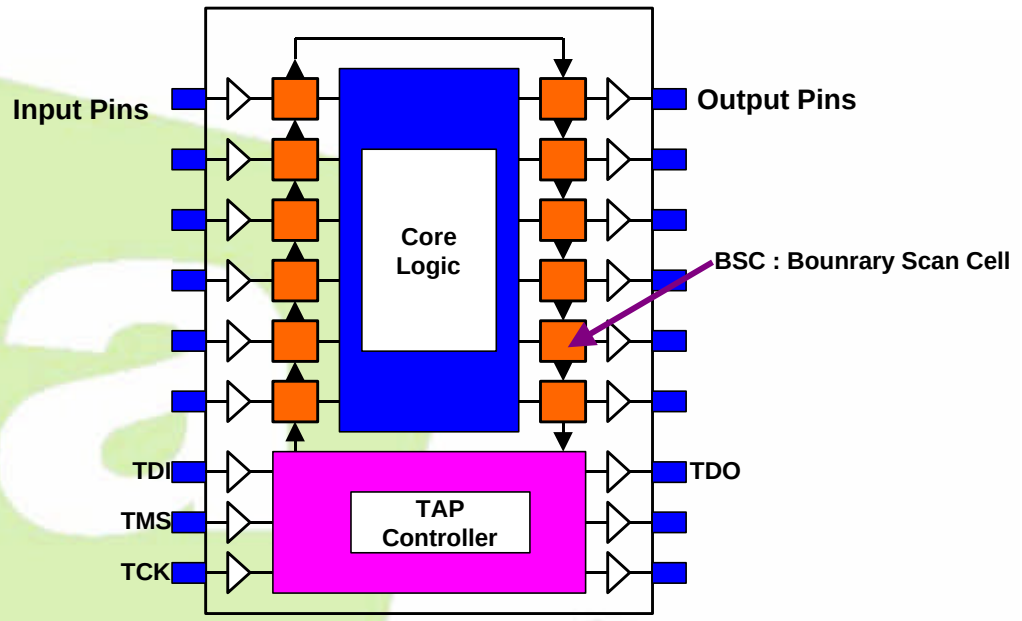
Basic ISP Flow & JTAG



JTAG - Test Pins

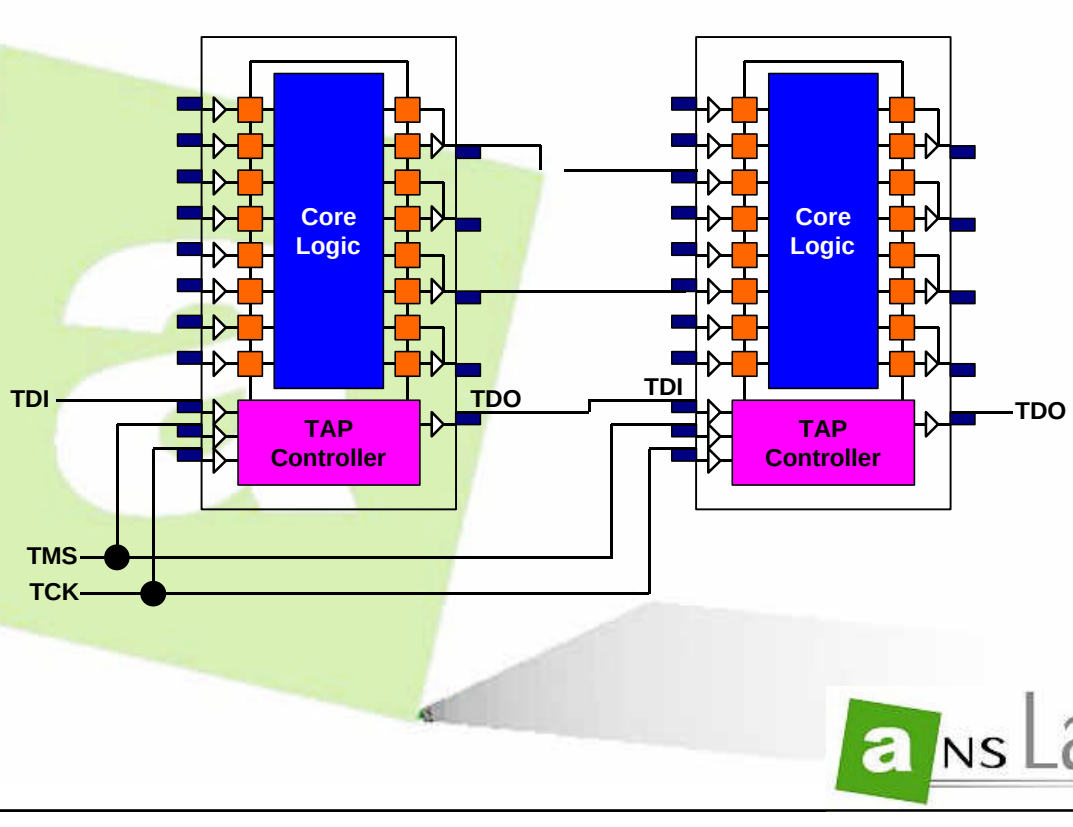
- TCK - Test Clock
  - Provides the clocks (Data Pattern Shifting)
- TMS - Test Mode Select
  - Controls test operation (instruction/data)
- TDI - Test Data Input
  - Receives serial test instruction and data
- TDO - Test Data Output
  - Serial output for test instruction and data
- /TRST - Test Reset (Optional)
  - Resets both the JTAG state machine and the instruction register

Boundary Scan Architecture



Test Access Port Controller controls Boundary Scan-Path

JTAG Connectivity Test Example



TAP Controller

- A State Machine to control TAP
  - TAP : TDI,TDO,TMS,TCK,/TRST
- Control Operations
  - Shifting Test Data
  - Sampling Data
  - Driving a value
- Can pause while shifting registers
- Controls input BSC and output BSC separately

VHDL 시작

VHDL을 시작하려면 필요한 것들...

- Programming Language Compiler
  - Algorithm Test
  - Test Vector Generation
- HDL Simulator
  - Functional Simulation
  - Timing Simulation
- HDL Synthesizer
  - HDL to Netlist
  - Optimization
- Device Vendor Tool (Design Kit)
  - Place and Route
  - Timing Extraction

VHDL Simulator (PC Version)

- ModelSim (Ex. V-System)
  - http://www.model.com
  - Support VHDL/Verilog, FLI/PLI
- Aldec ActiveVHDL
  - http://www.aldec.com
  - Visual tool included(FSM, Block editor)
- PeakVHDL
  - http://www.acc-eda.com
- VeriBest
  - http://www.veribest.com
- MyVHDL (서두로직)
  - http://www.seodu.co.kr

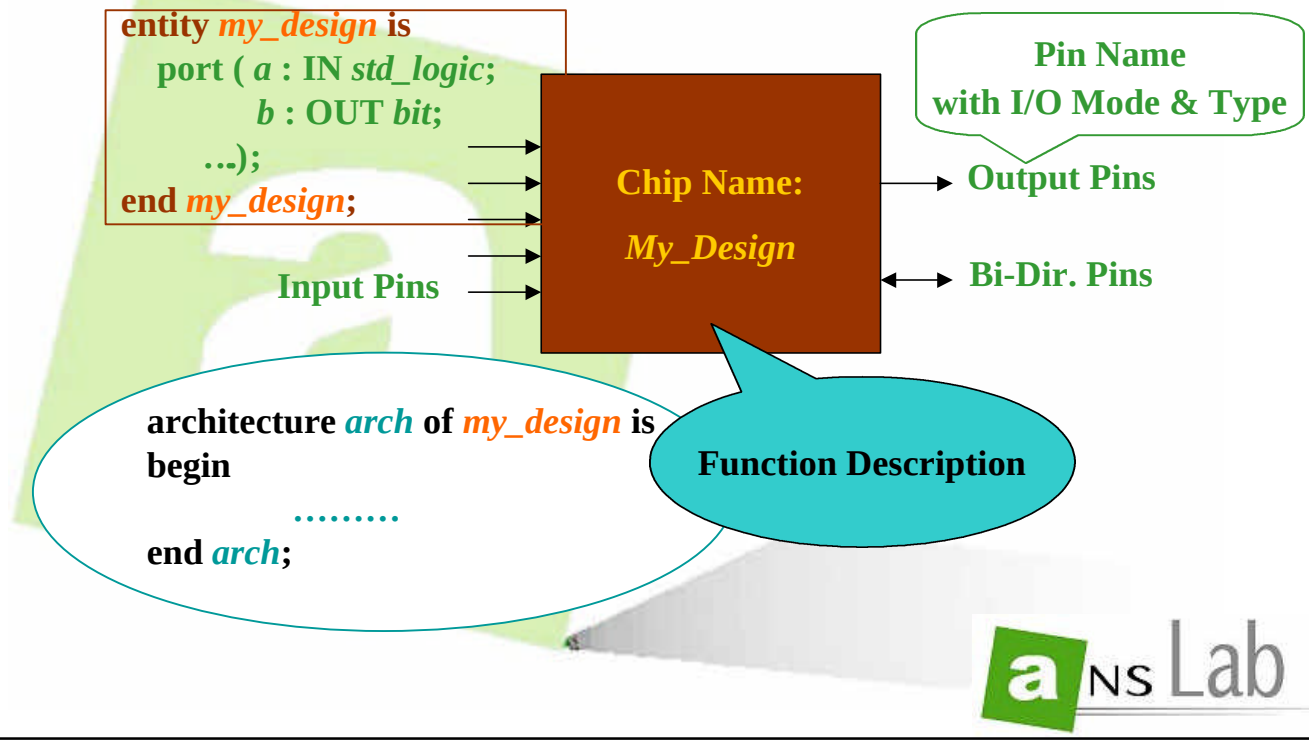
VHDL Synthesizer (General PLD)

- Metamor Logic Synthesizer
- MINC PLSynthesizer
- Synopsys’ FPGA Express/FPGA Compiler
- Synplicity’ s Synplify
- Exemplar Galileo/Leonardo/Spectrum
- 참고:
  - http://www.optimagic.com

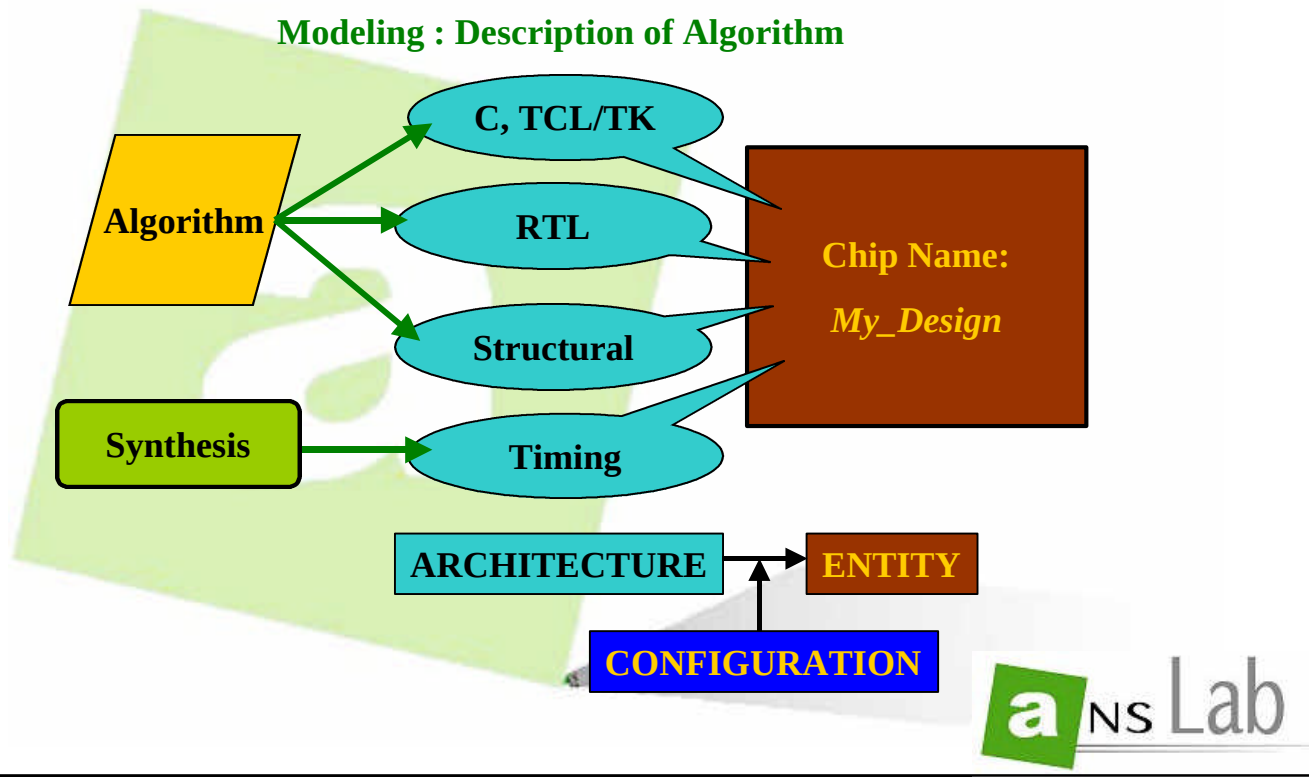
VHDL Designer

VHDL: Design Unit

- Entity and Architecture



VHDL : Configuration



VHDL Testbench

VHDL Testbench

- NO Synthesis VHDL
- Large/Complex Test Vectors and Verify Automated
  - *“Think DSP Application or MPU’s Complex IO”*
- Test Vectors
  - Input Vector
  - Output Vector
  - Expected Vector
- UUT : Unit Under Test
- Clock Generator
- Error Report Utility

Library : STD

- **Library for Testbench**

  use std.standard.all; ***-- Char. & String type define***

  use std.textio.all; ***-- Text IO Buffer & File IO***

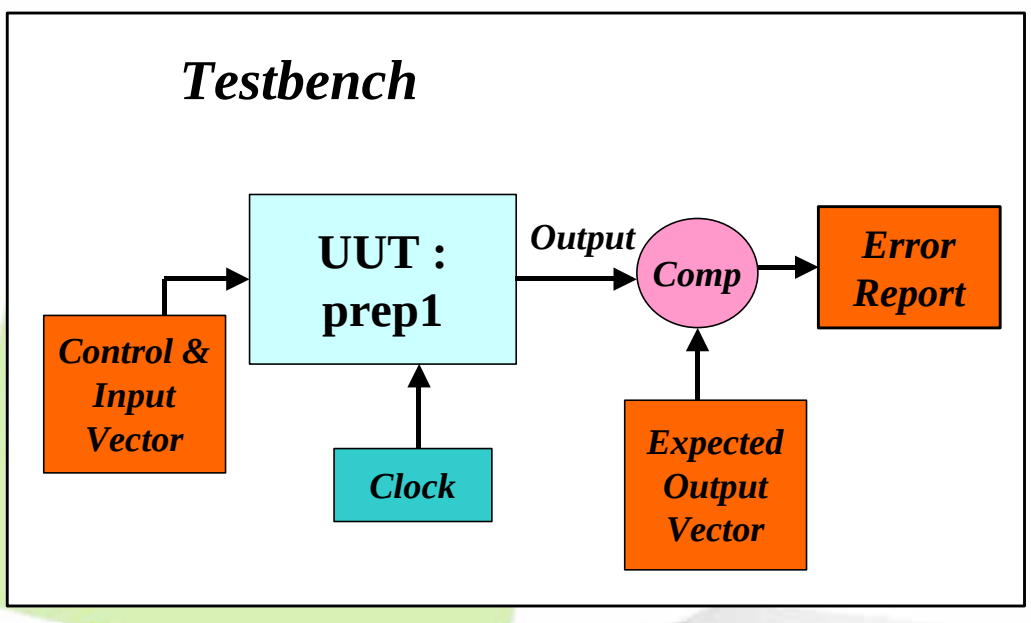
- **Testbench Entity**

  ENTITY *test_prep1* IS

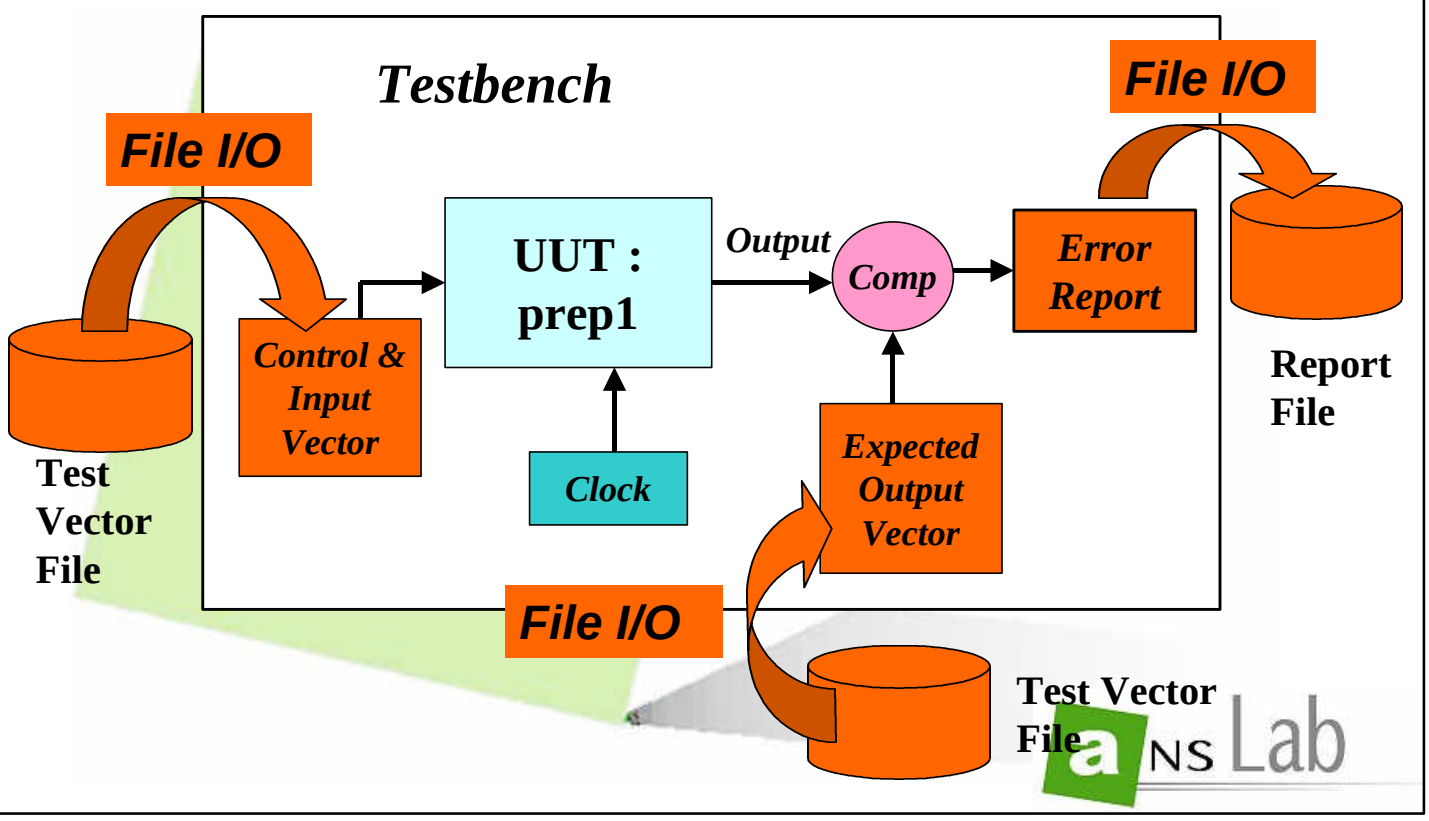
  -- Testbench has NOT need Port ….

  END *test_prep1*;

Testbench Component



Testbench (Test Vector : File I/O)



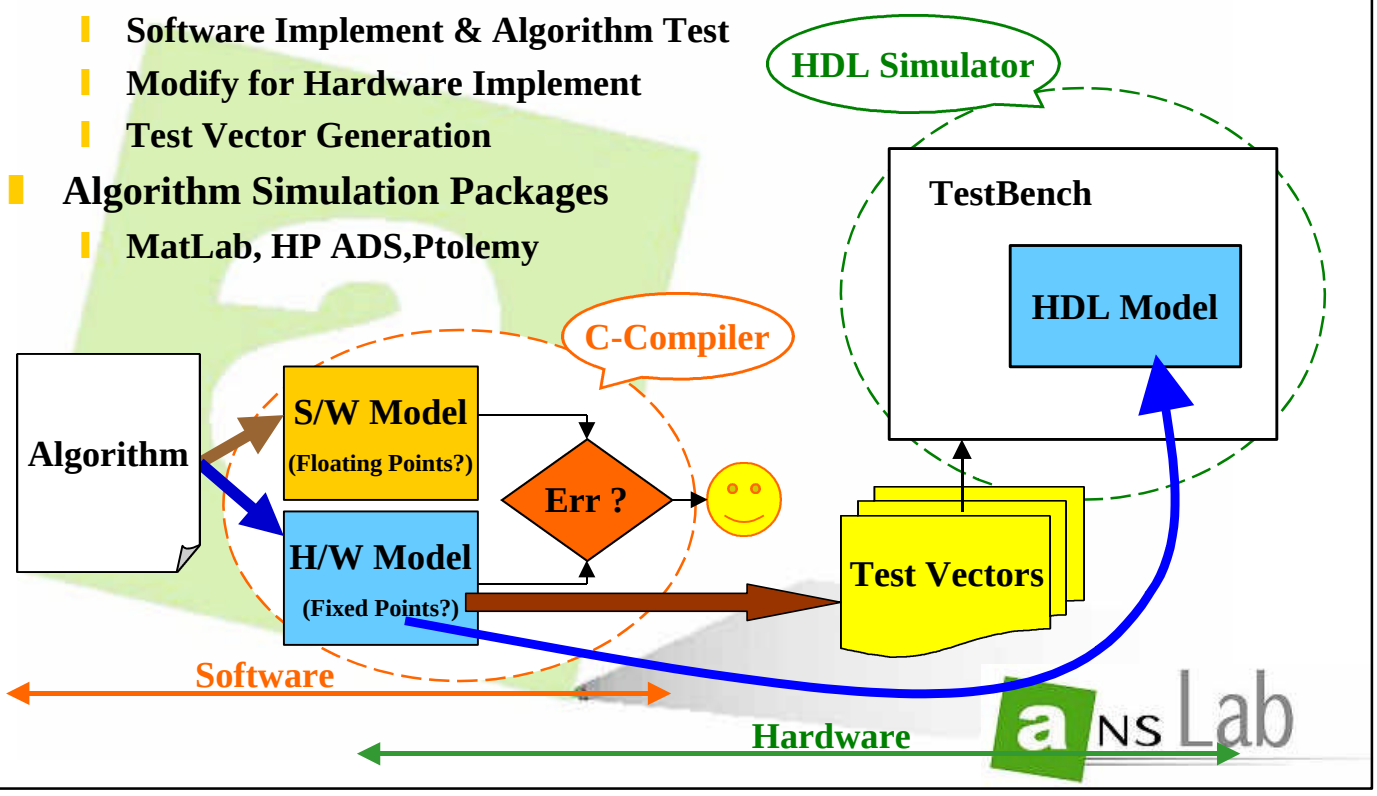
aNS Lab

Advanced VHDL Environment

H/W & S/W Co-Simulation
Scripting (Macros, Tcl/Tk)
Visual Tools

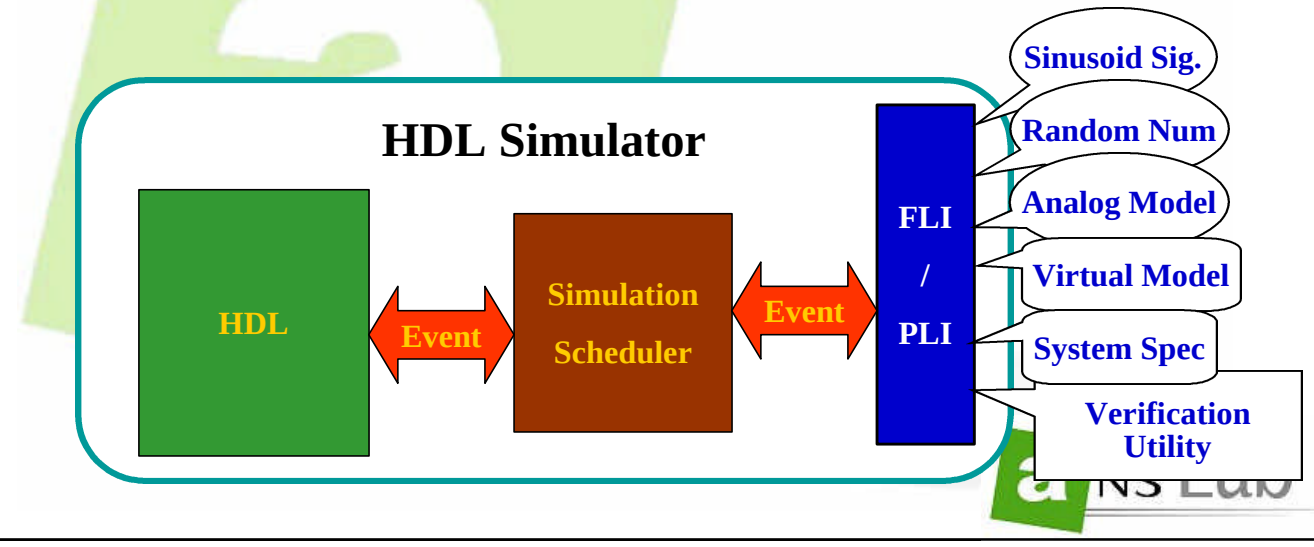
S/W & H/W Modeling/Simulation

- Programming Language Model
  - Software Implement & Algorithm Test
  - Modify for Hardware Implement
  - Test Vector Generation
- Algorithm Simulation Packages
  - MatLab, HP ADS,Ptolemy



H/W & S/W Co-Simulation

- Software Model for Test & Verification
  - Easy & Fast : Specification/Documentation/Test
  - Complicated test signal generation
  - Direct verification at modeling process
  - Expecting synthesis result
- Co-Simulation How ?
  - Use Shared Library Executables (.dll, .so)
  - Call-Back to signal event by HDL Simulator (Scheduling)
  - Interface to H/W Signal Types



Using Scripts

- Need for Reusability of IP
- Macro Files
  - Simulation Commands
- Tcl/Tk
  - GUI : Visualize Simulation Result
  - Integration of External EDA Tools/Utilities
  - Build Design Specific User Interface

Visual Tools

- **Visual Tool**
  - **Graphical Design Entry & HDL Generation**
  - **Easy & Familiar**
  - **View whole design : Interconnection between IP Blocks, Good for System integration**
- **What Tools avail ?**
  - **Keyword Colored Text Editor, Context Sensitive Help**
  - **Schematics**
  - **Block Diagram Editor**
  - **Truth Table Entry**
  - **Finite-State Machine**
  - **Flow Chart**
  - **TestBench Generation, Language Template, etc.,..**

VHDL Data Types

Scalar Types

Composite Types

Data Types (VHDL 자료형)

- All signals have a type
- Type checking is strict !
- Each type has a defined set of valid operations
- Each type has a defined set of values
- Types are user-extendable

VHDL Data Types (분류)

- Scalar Types
  - Enumeration Type
  - Integer Type
  - Floating-Point Types
  - Physical Types
- Composite Types
  - Array Types
  - Record Types
- Access Types
- File Types

Common Types

- Bit (or std_logic, defined by IEEE 1164)
- Bit_vector (or std_logic_vector)
- Boolean
- Integer
- Character
- String
- Enumerated type

More Types

- Real (floating point)
- Physical (time, current, etc.)
- Arrays (single- and multi-dimensional, constrained or unconstrained)
- Records
- Access types (pointers)
- File types

Enumerated Types

- Many common VHDL types are actually enumerated types:
  - Boolean (TRUE, FALSE);
  - Bit (1,0)
  - char
- User extendable:

```
type Boolean is ( FALSE, TRUE);
type bit is (‘ 0’ , ‘ 1’ );
type std_ulogic is (‘ U’ ,’ X’ ,’ 0’ ,’ 1’ ,’ Z’ ,’ W’ ,’ L’ ,’ H’ ,’ -’ );
type states is ( IDLE, RECEIVE, SEND);
```

Enumerated Type Encoding

- One Hot Encoding (State Machine)
- Binary Encoding

	bit2	bit1	bit0
IDLE	-	-	1
RECEIVE	-	1	-
SEND	1	-	-

OneHot

	bit1	bit0
IDLE	0	0
RECEIVE	0	1
SEND	1	0

Binary

Integer and Real Type

- Integer Type (Synthesizable)

```
type integer is range -2147483647 to 2147483647; -- VHDL Predefined
                                                 -- 32-bit signed integer
type my_integer is range 0 to 15;  -- 4-bits
signal count : my_integer;
signal count : integer range 0 to 15; -- count(0) : LSB, count(3) : MSB
```

- Floating-Point Type (NOT Synthesizable)

```
type real is range -1.0E38 to 1.0E38; -- VHDL Predefined
type my_real is range 0.0 to 1.0; -- VHDL Predefined
```

Physical types

- Represent relations between quantities
  - NOT Synthesizable
- Example : “time”

```
type time is range -2147483647 to 2147483647
units
        fs;
        ps = 1000 fs;
        ns = 1000 ps;
        us = 1000 ns;
        ms = 1000 us;
        sec = 1000 ms;
        min = 60 sec;
        hr = 60 min;
end units;
```

Array Types

- Synthesizable
- Array Example (Unconstraint)

```
type std_ulogic_vector is array ( NATURAL RANGE <> ) OF std_ulogic;
type UNSIGNED is array (NATURAL range <> ) of BIT;
type SIGNED is array (NATURAL range <> ) of BIT;
type matrix is array (natural range <>, natural range <>) of  bit;
```

- Array Example (Constraint)

```
type byte is array (7 downto 0) of  bit;
```

Record Type

- Example

```
type month_name is (jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dev);
type date is
record
   day : integer range 1 to 31;
   month : month_name;
   year : integer range 0 to 4000;
end record;
```

```
constant my_birthday : date := (21, nov, 1963);
```

```
SIGNAL my_birthday : date;

my_birthday.year <= 1963;
my_birthday.month <= nov;
my_birthday.day <= 21; -- 5-bit
                       -- my_birthday(0) : LSB
                       -- my_birthday(4) : MSB
```

Subtypes and Type Conversion

- Type checking is strict !
- Type with constraint

  subtype <subtype_name> is <base_type> [<constraint>];

- Type Conversions

  <type>(<expressions>)

VHDL type checking is strict !

```
type big_integer is range 0 to 1000;
type small_integer is range 0 to 7;

signal intermediate : small_integer;
signal final : big_integer;

final <= intermediate * 5; -- type mismatch error
```

or

```
type big_integer is range 0 to 1000;
subtype small_integer is big_integer range 0 to 7;

signal intermediate : small_integer;
signal final : big_integer;

final <= intermediate * 5; -- Same Base type
```

```
type big_integer is range 0 to 1000;
type small_integer is big_integer range 0 to 7;

signal intermediate : small_integer;
signal final : big_integer;

final <= big_integer(intermediate * 5); -- Type Conversion
```


IEEE 1076 Predefined Types

```
type bit is (‘0’, ‘1’);
type bit_vector is array (integer range <>) of bit;
type integer is range MININT to MAXINT;
subtype positive is integer range 1 to MAXINT;
subtype natural is integer range 0 to MAXINT;
type boolean is (TRUE, FALSE);
```

IEEE 1164 Predefined Types

- std_logic, std_logic_vector (resolved)
- std_ulogic, std_ulogic_vector (unresolved)
- Nine valued enumerated type
- Used as replacements for bit, bit_vector
- Used as the standard data types for system interfaces

IEEE 1164 Predefined Types

```
TYPE std_ulogic IS ( 'U',  -- Uninitialized
                     'X',  -- Forcing  Unknown
                     '0',  -- Forcing  0
                     '1',  -- Forcing  1
                     'Z',  -- High Impedance
                     'W',  -- Weak     Unknown
                     'L',  -- Weak     0
                     'H',  -- Weak     1
                     '-'   -- Don't care
                   );
type std_ulogic_vector is array (natural range <>) of std_ulogic;
subtype std_logic is resolution_func std_ulogic;
type std_logic_vector is (natural range <>) of std_logic;
subtype X01Z is resolution_func std_ulogic range 'X' to 'Z';
```

Type Conversion Example

```
package my_type is
      type big_integer is range 0 to 1023;
      subtype small_integer is big_integer range 0 to 7;
end package;

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

entity subtype_test is
      port (   a : in  small_integer;
               b : in  small_integer;
               c : out std_logic_vector(9 downto 0) );
end subtype_test;
architecture behave of subtype_test is
      signal t_int : big_integer;
begin
      t_int <= a + b;  -- for subtype, type casting not needed !

      c <= std_logic_vector(to_unsigned(natural(t_int), 10)); -- type casting/type conversion
end;
```


Type Conversion/Type Casting

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

type big_integer is range 0 to 1023; -- 10-bits
signal t_int : big_integer;
signal       c : out std_logic_vector(9 downto 0);

c <= std_logic_vector( to_unsigned( natural( t_int ) , 10) ) ;
```

Type casting "unsigned" to "std_logic_vector"

Type conversion "natural" to "unsigned"

Type casting "integer" to "natural"

big_integer

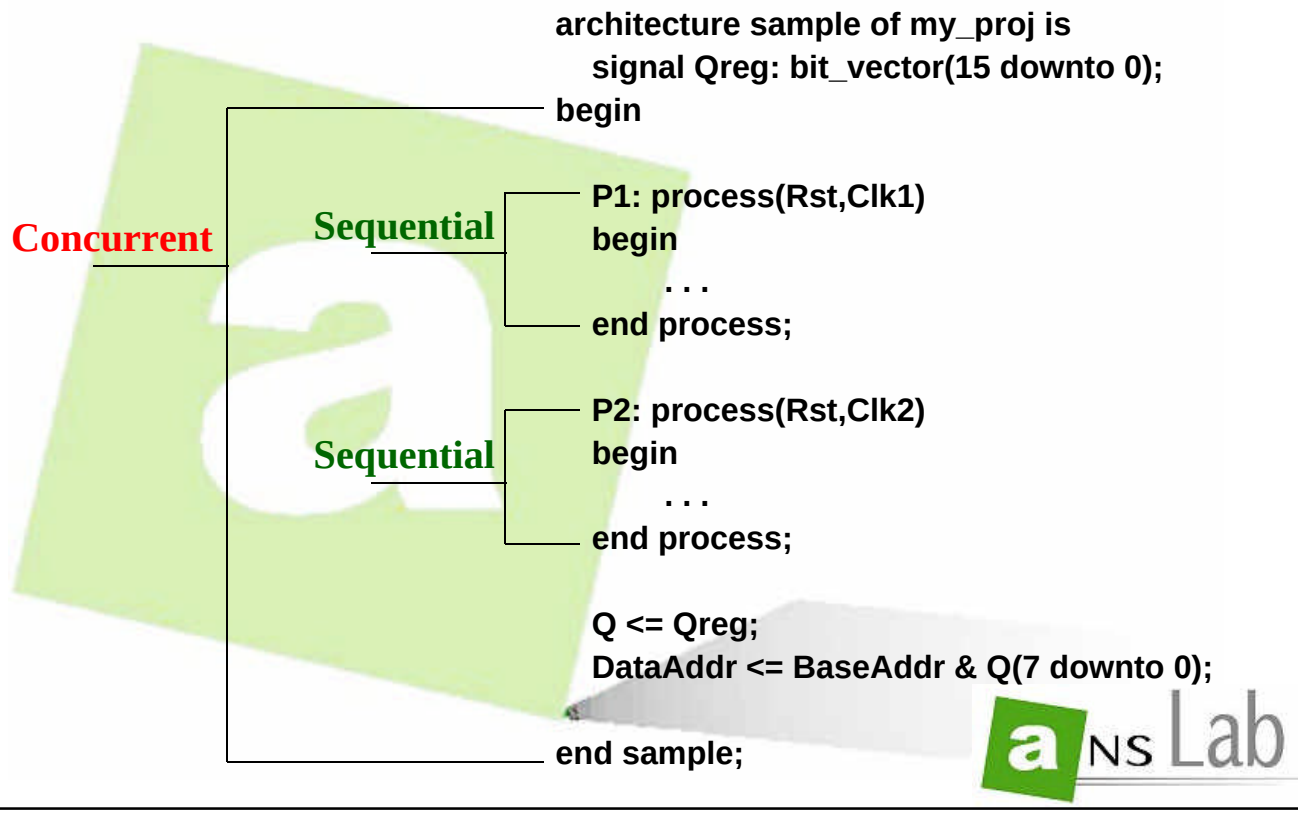
VHDL Design Partition

Processes
Blocks
Functions
Procedures

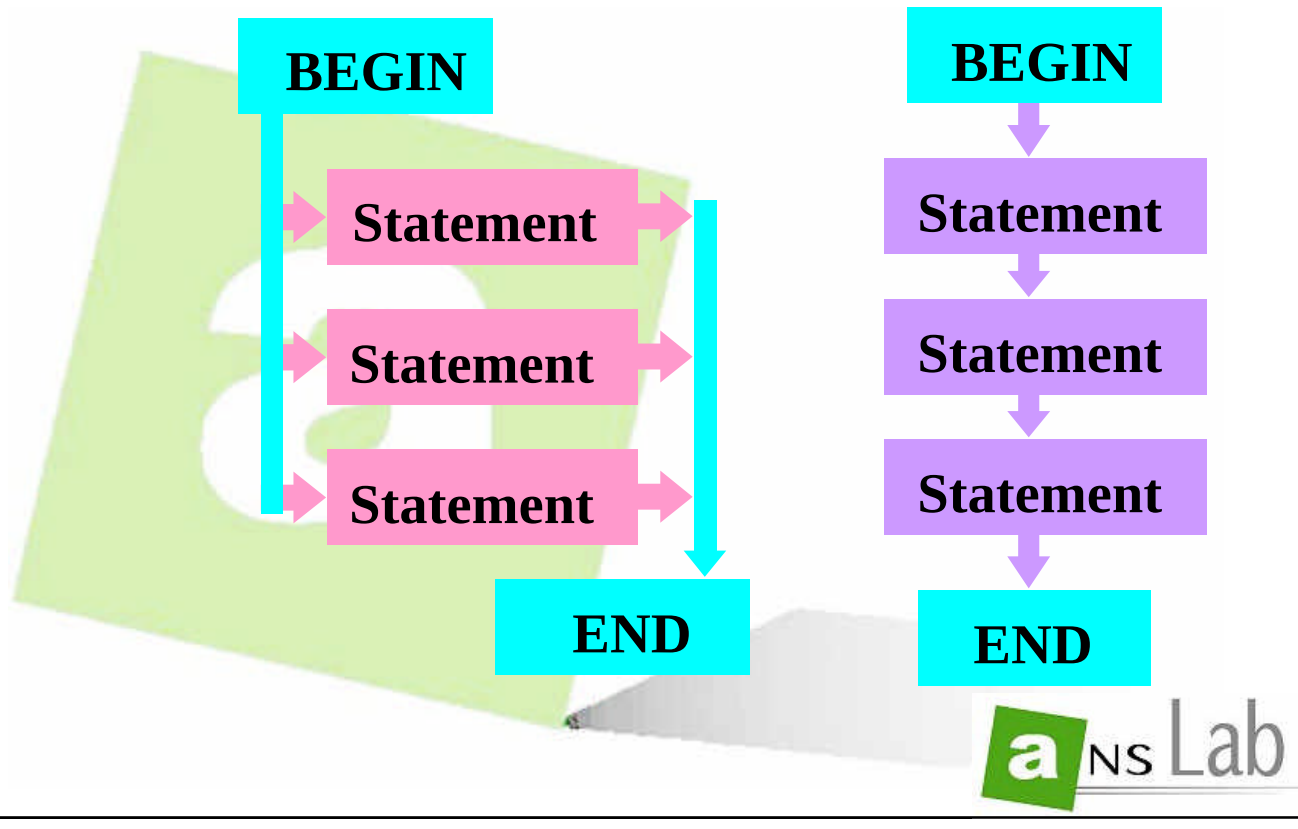
Processes and Blocks

- **Process**
  - **Sequential Env.**
  - **Local VARIABLE**
  - **NOT Nested Process**
  - **Sensitivity list or WAIT statement**
  - **communicate with SIGNAL**
- **Block**
  - **Concurrent Env.**
  - **Local SIGNAL**
  - **Nested Blocks**
  - **GUARDED expression**

Concurrent vs. Sequential



Concurrent vs. Sequential



Process Example

```
Entity experiment is
   port ( source : in bit_vector(0 to 3);
          ce : in bit;
          wrclk : in bit;
          selector : in bit_vector(0 to 1);
          result : out bit );
end experiment;

architecture behave od experiment is
   signal intreg : bit_vector(0 to 3);
begin -- dataflow(concurrent) environment

   writer : process -- process statement
   -- variable declarative region (empty here)
   begin
      -- sequential (clocked) statement
      wait until wrclk' event and wrclk=' 1' ;
      if (ce=' 1' ) then
         intreg <= source;
      end if;
   end process;

   reader : process (intreg, selector) -- process
                                       -- with sensitivity list
   -- variable declarative region (empty here)
   begin
      -- sequential (not-clocked) statements
      case selector is
         when "00" =>
            result <= intreg(0);
         when "01" =>
            result <= intreg(1);
         when "10" =>
            result <= intreg(2);
         when "11" =>
            result <= intreg(3);
      end case;
   end process;

end behave;
```

Block Example

```
Architecture dataflow of example is
   signal global_sig, g1, g2, c, bit;
begin

   B1 : block                    -- Block declarative region
      signal local_sig : bit;    -- Local SIGNAL
   begin                         -- Block concurrent statements

      local_sig <= global_sig;  -- Communicate with global signal directly

                                    -- Block in a Block
      B2 : blobk (c=' 1' )          -- Block has "GUARD" expression
         port ( o1, o2 : out bit)   -- Block port declarations
         port map (o1=>g1, o2=>g2); -- Block port mapping to global signal
      begin
         o1 <= guarded local_sig;   -- Latch inference
         o2 <= global_sig
      end block;

   end block;

end dataflow;
```

Subprograms : Functions & Procedures

- Execute Sequentially
- Local variable
- Can be called from the dataflow (concurrent) or any sequential environment
- Can be overloaded (Synthesizer resolve this)
- Functions
  - all input argument, return single value
- Procedure
  - in, out, inout argument
  - Side effect

Subprogram Example

- **Procedure**

```
procedure increment ( vect : inout bit_vector; ena : in bit := '1' )  is  -- in, inout argument
begin
   if (ena='1') then
      vect := vector_adder (x=>vect, "000001" ) ;  -- Cause side effect
   end if ;
end increment ;
```

- **Functions**

```
function vector_adder (x : bit_vector; y : bit_vector) -- input argument
return bit_vector is                                   -- return type
   variable carry : bit := '0' ;
   variable result : bit_vector(x'RANGE) ;
begin
   for i in x'RANGE loop
      result (i) := x(i) XOR y(i) XOR carry ;
      carry := carry AND (x(i) OR y(i)) OR x(i) AND y(i) ;
   end loop ;
   return result ;  -- one return value
end vector_adder ;
```

VHDL Objects

SIGNAL
CONSTANT
VARIABLE
PORT
Loop Variable
Generic

SIGNAL

- **Wire in a Logic Circuit**
- **CANNOT be declared in Sequential Blocks; “Function”, “Procedure” and “Process”**
- **Range restrintion**
- **Initial value (but ignored by synthesis)**
- **Signal has Attribute; ’EVENT**
- **Assignment (<=) is scheduled after delta delay**

VARIABLE

- CANNOT be declared in the dataflow(concurrent block) or Package
- Declared in Process, Function, and Procedure
- Initial value (but ignored by synthesis)
- Assignment (:=) is immediate

CONSTANT

- CANNOT be assigned after declaration
- Initialization is required

```
cosntant ZEE_8 : std_logic_vector (0 to 7) := “ZZZZZZZZ”;
```

PORT

- Interface terminal of Entity
- Normal signal with Direction Mode restriction:
  - IN, OUT : unidirectional

    s_out <= s_in1 + s_in0;
  - INOUT : bidirectional

    s_inout <= s_in1 + s_in0;

    s_out <= s_inout + s_in1;
  - BUFFER : bidiretional

    buff <= buff + s_in;
- CAN have initial value

GENERIC

- Property of Entity
- Definition size of interface and Delay,etc.

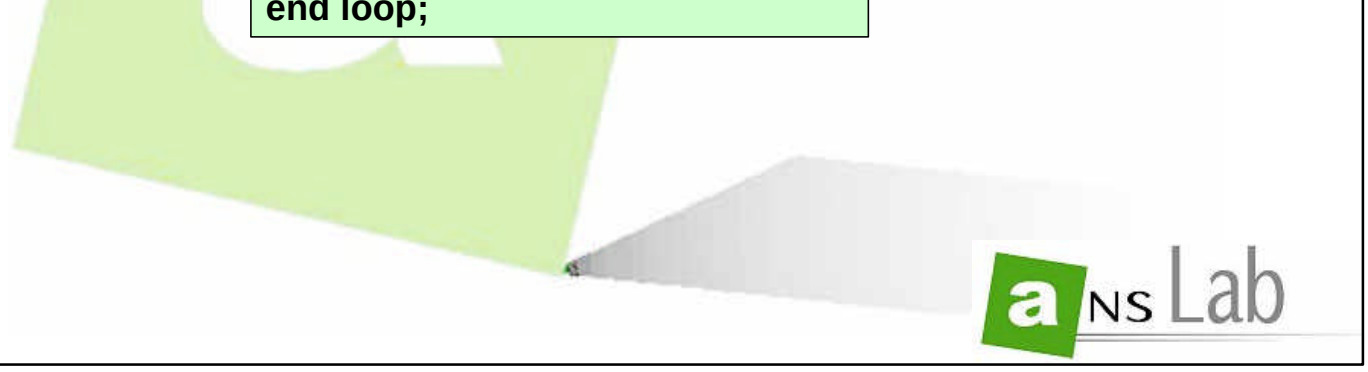
```
entity counter is
   generic ( size : integer := 8;
             delay : time := 10 ns; );
   port ( clk, reset : IN std_logic;
          count : OUT std_logic_vector( size-1 downto 0) );
end counter;
```

```
inst_1 : counter
         generic map (size => 16, delay => 5 ns)
         port map (clk=>clk, reset=>reset, count=>count);
```

Loop Variable

- **NOT declared**
- **Get its type and value from specified range in the iteration scheme**

```
for i in 0 to ( b’ LENGTH-1) loop
    a(i) <= b(i) and ena;
end loop;
```



VHDL Statements

Conditional Statements
Selection Statements
Loop and Generate Statements
Assignment Statements

Conditional Statements

Dataflow

```
output_signal <=
                    x when a=1 else
                    y when a=2 else
                    z when a=3 else
                     (others=>”0”);
```

Behavior

```
if a=1 then
   output_signal <= x;
elsif a=2 then
   output_signal <= y;
elsif a=s then
   output_signal <= z;
else
   output_signal <= (others=>”0”);
```

aNS Lab

Selection Statements

Dataflow

```
with sel select
   output_signal <=
                x when “0010”,
                y when “0100”,
                z when “1000”,
               (others=>”0”)
                        when others;
```

Behavior

```
case sel is
   when “0010” =>
      output_signal <= x;
   when “0100” =>
      output_signal <= y;
   when “1000” =>
      output_signal <= z;
   when others =>
      output_signal <=
                        (others=>”0”);
end case;
```

aNS Lab

for / while loop statements

- Inerative/Repeative operation
- Loop statement can be only used in sequential block (behavior)

```
-- for loop : Synthesizable
signal result, input_sig : bit_vector( 0 to 5);
signal ena;
. . . . .
for i in 0 to 5 loop
   result(i) <= input_sig(i) and ena;
end loop;
```

```
-- while loop : partially synthesizable
variable i : integer;
. . . . .
i := 0;
while ( i<6 ) loop
   result(i) <= input_sig(i) and ena;
   i := i + 1;
end loop;
```

aNS Lab

for / while loop statements

- Sequential environment allow EXIT/NEXT statement in the Loop
- while loop is NOT synthesizable, if loop condition is evaluate at run-time

How many Loop ? → Synthesizer → **How many bits of Hardware?**

```
i := -1;
while (TRUE) loop
      i := i + 1;
      exit if ( i > 5 );
      if ( input_sig(i) = '0' ) then
            result(i) <= '0';
            next;
      end if;
      result(i) <= ena;
end loop;
```

Synthesizable
But NOT RECOMENDED style

aNS Lab

for - generate loop

- Generate block execute concurrently
- NOT allow NEXT/EXIT statement
- Synthesizable

```
-- for-generate : Synthesizable
signal result, input_sig : bit_vector( 0 to 5);
signal ena;
    . . . . .
for i in 0 to 5 generate
        result(i) <= input_sig(i) and ena;
end generate;
```

VHDL Assignment

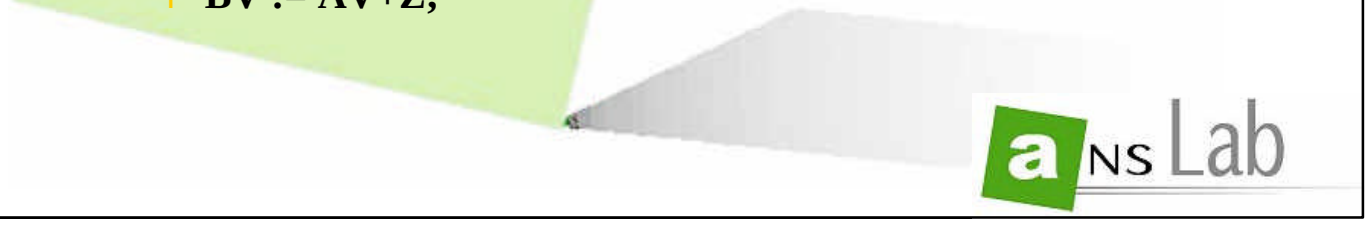
SIGNAL 과 VARIABLE

Concurrent/Sequential Assignment

- SIGNAL
  - 병렬구문 (concurrent statement)
- VARIABLE
  - 순차구문 (sequential statement) : PROCESS
- CONSTANT
  - C언어의 #define, Assembly언어에서의 EQU
  - 컴파일시 값이 결정 된다

VARIABLE

- VARIABLE :
  - “:=”
  - 순차적인 즉시 할당(instaneous assignment)
  - PROCESS BLOCK에서 sequential statement
  - Example:
    - AV := X*Y;
    - BV := AV+Z;



SIGNAL

- **SIGNAL :**
  - **“<=”**
  - **지연 할당(delayed assignment)**
  - **Concurrent statement**
  - **Attributes**
  - **Example:**
    - **AS <= X*Y after 2 ns;**
    - **BS <= AS + Z after 2 ns;**

Assignment Execution

```
AS <= X * Y after 2 ns;
BS <= AS + Z after 2 ns;
```

	Initial	T1	T1+2	T1+4	T1+6
X	1	4	5	5	3
Y	2	2	2	3	2
AS	2	2	8	10	15
Z	0	3	2	2	2
BS	2	2	5	10	12

Delayed Concurrent Assignment

```
AV :=  X * Y;
BV :=  AV + Z;
```

	Initial	T1	T1+2	T1+4	T1+6
X	1	4	5	5	3
Y	2	2	2	3	2
AV	2	8	10	15	6
Z	0	3	2	2	2
BV	2	11	12	17	8

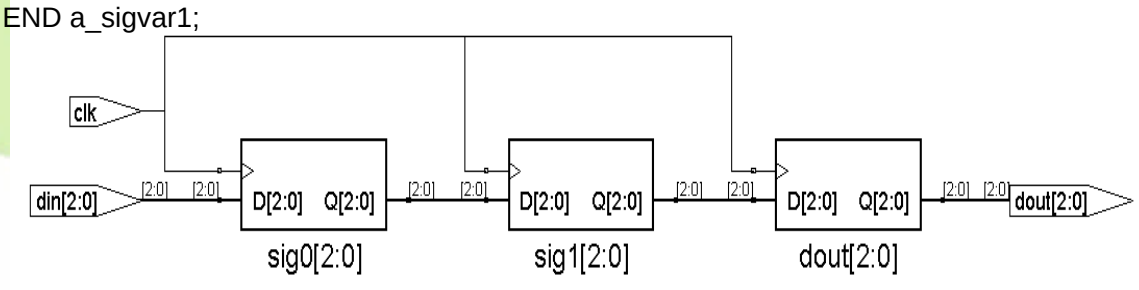
Instaneous Sequential Assignment

SIGNAL 과 VARIABLE 합성

```
ENTITY sigvar1 IS
  PORT(
      clk : IN std_logic;
      din : IN std_logic_vector(2 DOWNTO 0);
      dout : OUT std_logic_vector(2 DOWNTO 0) );
END sigvar1;
```

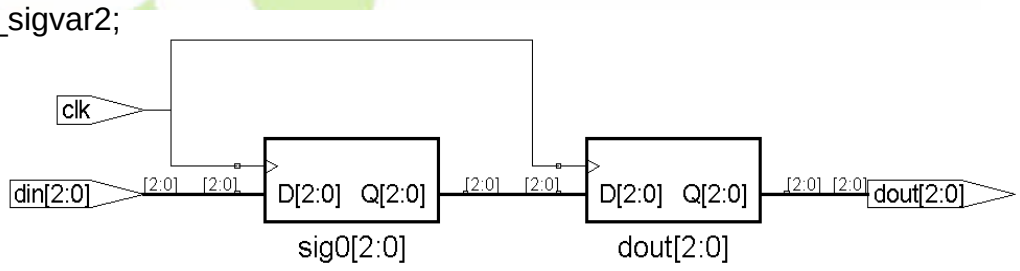
SIGNAL 과 VARIABLE 합성 (1)

```
ARCHITECTURE a_sigvar1 OF sigvar1 IS
SIGNAL sig0, sig1 : std_logic_vector(2 DOWNTO 0);
BEGIN
  PROCESS (clk)
  BEGIN
    IF clk='1' AND clk'EVENT THEN
      sig0 <= din;
      sig1 <= sig0;
      dout <= sig1;
    END IF;
  END PROCESS;
END a_sigvar1;
```



SIGNAL 과 VARIABLE 합성 (2)

```
ARCHITECTURE a_sigvar2 OF sigvar2 IS
SIGNAL sig0, sig1 : std_logic_vector(2 DOWNTO 0);
BEGIN
  PROCESS (clk)
  BEGIN
    IF clk='1' AND clk'EVENT THEN
      sig0 <= din;  -- This statement infer F/F
      sig1 <= sig0; -- This statement infer another F/F
    END IF;
  END PROCESS;
  dout <= sig1;  -- "sig1" DOES NOTHING. Just Connected to "dout"
END a_sigvar2;
```

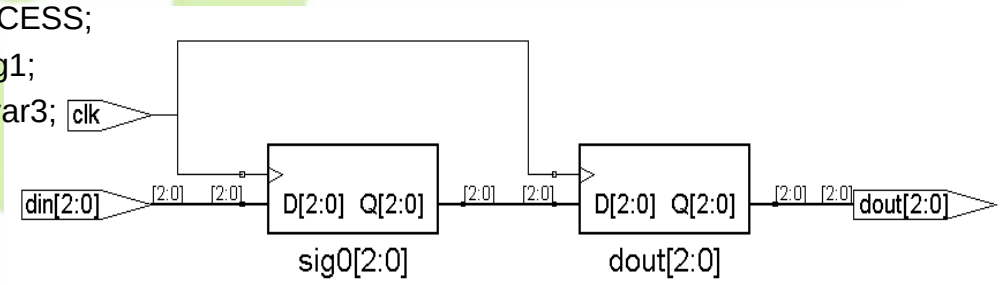


SIGNAL 과 VARIABLE 합성(2-1)

```
ARCHITECTURE a_sigvar2_1 OF sigvar2_1 IS
SIGNAL sig0, sig1 : std_logic_vector(2 DOWNTO 0);
BEGIN
  PROCESS (clk)
  BEGIN
    IF clk='1' AND clk'EVENT THEN
      sig0 <= din;  -- This Statement infers F/F
    END IF;
  END PROCESS;

  sig1 <= sig0; -- This is Concurrent assignment
  dout <= sig1; -- "sig0" and "sig1" is does NOTHING! Just Connected

END a_sigvar2_1;
```

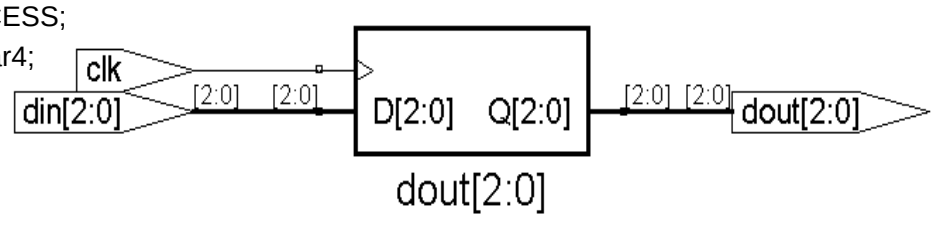
SIGNAL 과 VARIABLE 합성 (3)

```
ARCHITECTURE a_sigvar3 OF sigvar3 IS
SIGNAL sig0, sig1 : std_logic_vector(2 DOWNTO 0);
BEGIN
  PROCESS (clk)
  BEGIN
    IF clk='1' AND clk'EVENT THEN
      sig1 <= sig0; -- Concurrent assignment in the Process Block does
      sig0 <= din;  -- not cares it's sequence. Compare Example #2
    END IF;
  END PROCESS;
  dout <= sig1;
END a_sigvar3;
```



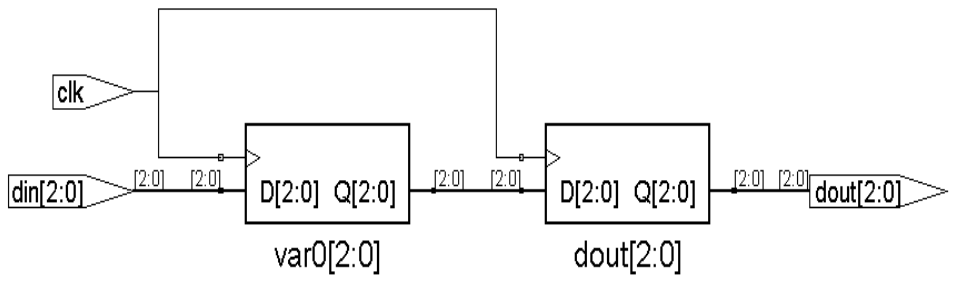
SIGNAL 과 VARIABLE 합성 (4)

```
ARCHITECTURE a_sigvar4 OF sigvar4 IS
BEGIN
  PROCESS (clk)
  VARIABLE var0, var1 : std_logic_vector(2 DOWNTO 0);
  BEGIN
    IF clk='1' AND clk'EVENT THEN
      var0 := din;  -- These two sequential assignment DOES NOTHING!
      var1 := var0; -- Compare with Concurrent assignment; Example #2 and #3
      dout <= var1; -- Just this statement infers F/F between "din" and "dout"
    END IF;
  END PROCESS;
END a_sigvar4;
```



SIGNAL 과 VARIABLE 합성 (5)

```
ARCHITECTURE a_sigvar5 OF sigvar5 IS
BEGIN
  PROCESS (clk)
  VARIABLE var0, var1 : std_logic_vector(2 DOWNTO 0);
  BEGIN
   IF clk='1' AND clk'EVENT THEN
     var1 := var0;  -- These Two sequrntial statement infer TWO stage F/F
     var0 := din;   -- The order of Sequential statements are very imfortant!
                    -- Compare with Example #4
     dout <= var1;  -- F/F infered between "var0" and "dout"
                    -- Another F/F is infered between "din" and "var0"
   END IF;
  END PROCESS;
END a_sigvar5;
```

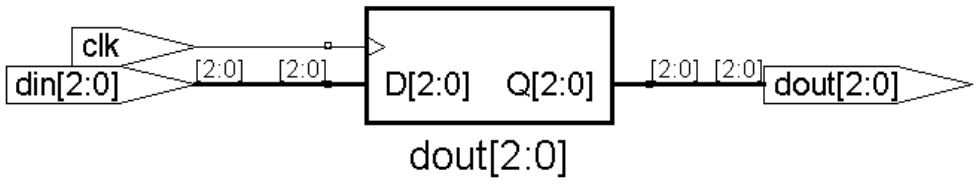


SIGNAL 과 VARIABLE 합성 (6)

```
ARCHITECTURE a_sigvar5 OF sigvar5 IS
BEGIN
  PROCESS (clk)
  VARIABLE var0, var1 : std_logic_vector(2 DOWNTO 0);
  BEGIN
   IF clk='1' AND clk'EVENT THEN
     var0 := din;   -- Synthesized as ONE stage F/F
     var1 := var0;  -- between "din" and "var1"
   END IF;
   dout <= var1;    -- But "var1" does nothing!, connected to "dout"
                    -- F/F infered between "din" and "dout"
  END PROCESS;
END a_sigvar5;
```

VARIABLEs are "local"

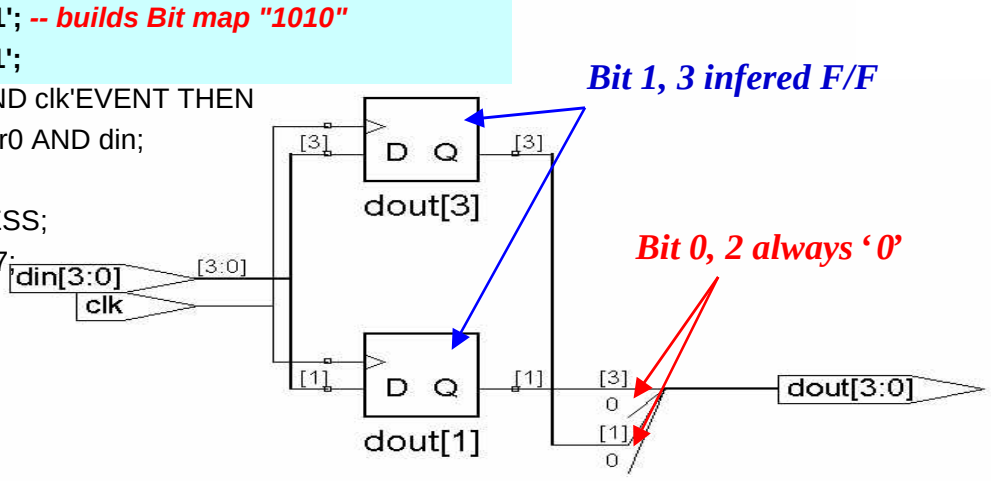
Pass local VARIABLE to global SIGNAL



SIGNAL 과 VARIABLE 합성 (7)

```
ARCHITECTURE a_sigvar7 OF sigvar7 IS
BEGIN

PROCESS (clk)
  VARIABLE var0 : std_logic_vector(3 DOWNTO 0);
  BEGIN
    var0 := "0000"; -- These Three sequential statement
    var0(3) := '1'; -- builds Bit map "1010"
    var0(1) := '1';
  IF clk='1' AND clk'EVENT THEN
    dout <= var0 AND din;
  END IF;
 END PROCESS;
END a_sigvar7;
```

Sequential assignment has NO multiple drive



SIGNAL 과 VARIABLE 합성 (8)

```
ARCHITECTURE a_sigvar7 OF sigvar7 IS
SIGNAL sig0 : std_logic_vector(3 DOWNTO 0);
BEGIN
   PROCESS (clk)
   BEGIN
     sig0 <= "0000"; -- The concurrent statement in the PROCESS block
     sig0(3) <= '1'; -- works sequentially...
     sig0(1) <= '1'; -- Compare with Example #7
     IF clk='1' AND clk'EVENT THEN
        dout <= sig0 AND din;
     END IF;
   END PROCESS;
-- Following concurrent Statements cause MULTIPLE Drive ERROR!
-- This region is CONCURRENT!!!!!!!!!!!!!!
-- sig0 <= "0000";
-- sig0(3) <= '1';
-- sig0(1) <= '1';
END a_sigvar7;
```

Sequential Block : NO multiple drive problem

Concurrent NOT allow multiple drive!

VHDL Operator

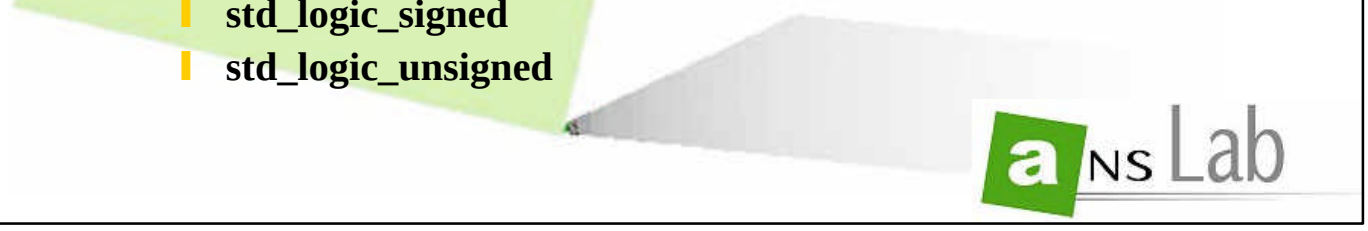
IEEE 1076 Predefined
IEEE 1164 Predefined
Synthesizable Arithmetics
Operator Overloading

VHDL Operators (IEEE 1076)

- **Binary operators:**
  - **AND, OR, NAND, NOR, XOR, NOT**
  - **predefined for BIT and BOOLEAN**
- **Relational operators:**
  - **=, /=, <, >, <=, >=**
  - **predefined for all types**
- **Arithmetic operators:**
  - **+, -, *, /, **, abs, mod, rem**
  - **predefined for integer types**
- **Concatenation operators:**
  - **&**

Synthesizable Arithmetic Operators

- IEEE 1076.3 Arithmetic Package
  - std_logic_1164 Package
    - New Logic types : std_logic, std_ulogic
    - Logical Operators
  - numeric_bit
    - +, -, *,abs between SIGNED, UNSIGNED of bit and bit_vector
  - numeric_std
    - +, -, *,abs between SIGNED, UNSIGNED of std_logic and std_logic_vector
- Synopsys Arithmetic Package
  - std_logic_arith
  - std_logic_signed
  - std_logic_unsigned



AND Operator (IEEE 1164)

```
-- truth table for "and" function
    CONSTANT and_table : stdlogic_table := (
--  ----------------------------------------------------
--  | U    X    0    1    Z    W    L    H    -       |   |
--  ----------------------------------------------------
    ( 'U', 'U', '0', 'U', 'U', 'U', '0', 'U', 'U' ),  -- | U |
    ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ),  -- | X |
    ( '0', '0', '0', '0', '0', '0', '0', '0', '0'
    ( 'U', 'X', '0', '1', 'X', 'X', '0', '1', 'X'
    ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X'
    ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X'
    ( '0', '0', '0', '0', '0', '0', '0', '0', '0'
    ( 'U', 'X', '0', '1', 'X', 'X', '0', '1', 'X'
    ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X'
);
```

```
-------------------------------------------------------------------
-- and
-------------------------------------------------------------------
FUNCTION "and"  ( l,r : std_logic_vector ) RETURN std_logic_vector IS
    ALIAS lv : std_logic_vector ( 1 TO l'LENGTH ) IS l;
    ALIAS rv : std_logic_vector ( 1 TO r'LENGTH ) IS r;
    VARIABLE result : std_logic_vector ( 1 TO l'LENGTH );
BEGIN
    IF ( l'LENGTH /= r'LENGTH ) THEN
        ASSERT FALSE
        REPORT "arguments of overloaded 'and' operator are not of the same length"
        SEVERITY FAILURE;
    ELSE
        FOR i IN result'RANGE LOOP
            result(i) := and_table (lv(i), rv(i));
        END LOOP;
    END IF;
    RETURN result;
END "and";
```

Synthesizable Arithmetic Operators

- IEEE 1076.3
  - Accepted types : “SIGNED”, “UNSIGNED”, NATURAL, INTEGER
  - Use Type Conversion for STD_LOGIC
  - Types defined (NUMERIC_STD)

```
type UNSIGNED is array (NATURAL range <>) of STD_LOGIC;
type SIGNED is array (NATURAL range <>) of STD_LOGIC;
```

  - Types defined (NUMERIC_BIT)

```
type UNSIGNED is array (NATURAL range <> ) of BIT;
type SIGNED is array (NATURAL range <> ) of BIT;
```

  - Validation Suit

Synthesizable Arithmetic Operators

- Arithmetic Operators
  - **Built-in Subprogram**
  - **In-Fix**
    - **+, -, /, *, =, /=, <, >, <=, >=, rem, mod**
    - **Synthesizer NOT support “/”,”rem”,”mod”**
  - **Unary**
    - **abs, -**
- Edge Detection (NUMERIC_BIT)
  - **RISING_EDGE**
  - **FALLING_EDGE**

Synthesizablr Arithmetic Operators

- SHIFT/ROTATE
  - SHIFT_RIGHT, SHIFT_LEFT
  - ROTATE_RIGHT, ROTATE_LEFT
- Type Conversion
  - RESIZE
  - TO_INTEGER,TO_UNSIGNED,TO_SIGNED
- Logical Operations
  - NOT,OR, NOR,AND, NAND,XNOR

Synthesis Example (Synopsys Library)

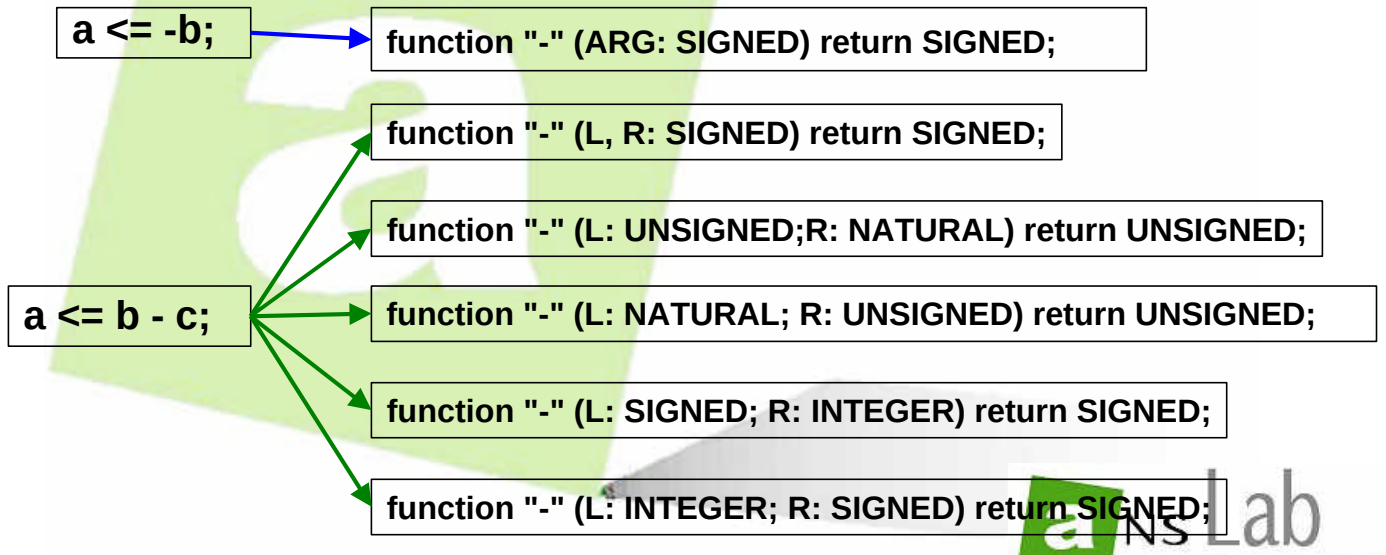
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;          -- Synopsys Arithmetic Package
use ieee.std_logic_unsigned.all; -- Synopsys Arithmetic Package

architecture behave of prep5 is
    signal q_i, adder_output, multiply_output: std_logic_vector (7 downto 0);
begin
    multiply_output <= a * b;  -- STD_LOGIC_VECTOR Type Multiply/Addition
    adder_output  <= (multiply_output + q_i) when mac = '1'  else
                              multiply_output;
     . . . . . .
```

Synthesis Example (IEEE 1076.3)

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;  -- IEEE 1076.3 Synthesis Library

architecture behave of prep5 is
   signal q_i, adder_output, multiply_output: unsigned (7 downto 0);
                                                    -- Unsigned SIGNAL
begin
   multiply_output <= signed(a) * signed(b); -- Convert to STD_LOGIC
                                             -- to UNSIGNED
   adder_output  <= (multiply_output + q_i) when mac = '1'  else
                    multiply_output;
   q <= std_logic_vector(adder_output); -- Interface to STD_LOGIC
end behave;
```


Operator Overloading

- VHDL is Object-Oriented
- Type check strictly, but Overloading



VHDL Attributes

Predefined Attributes

Vendor-Defined Attributes

Tool-Defined Attributes

Attributes

- **Predefined attributes**
  - **Return information about data types, signals, blocks, architectures, events, etc.**
  - **Examples: ’LENGTH, ’LEFT, ’EVENT**
- **Vendor defined attributes**
  - **Often used in synthesis**
  - **Example: enum_encoding, pinnum**
- **User defined attributes**
  - **Not often used**

VHDL Predefined Attributes

- Defined for SIGNAL
- Edge of SIGNAL
  - EVENT
- Value Returning
  - LEFT
  - RIGHT
  - HIGH
  - LOW
  - LENGTH
  - RANGE
  - REVERSE_RANGE

Predefined Attribute Example

```
signal vector_up : bit_vector( 4 to 9);
signal vector_dn : bit_vector( 25 to 0);

vector_up' LEFT -- returns integer 4
vector_dn' LEFT -- returns integer 25
vector_up' RIGHT -- returns integer 9
vector_dn' RIGHT -- returns integer 0
vector_up' HIGH -- returns integer 9
vector_dn' HIGH -- returns integer 25
vector_up' LOW -- returns integer 4
vector_dn' LOW -- returns integer 0
vector_up' LENGTH -- returns integer 6
vector_dn' LENGTH -- returns integer 26
vector_up' RANGE -- returns integer range 4 to 9
vector_dn' RANGE -- returns integer range 25 to 0
vector_up' REVERSE_RANGE -- returns integer range 9 to 4
vector_dn'  REVERSE_RANGE -- returns integer range 0 to 25
```

Vendor-Defined Attributes

- **Metamor Synthesizer**
  - **Pin Number Attribute**

```
ENTITY test IS
   PORT ( clk, start : IN bit;
                count0, count1 : out bit );
   attribute pinnum : string;    -- define pinnum attribute
   attribute pinnum of clk  : signal is "13";
   attribute pinnum of start : signal is "8";
   attribute pinnum of seg40 : signal is "19";
   attribute pinnum of seg41 : signal is "20";
END test;
```

User Defined Attributes

- **Why?**
- **Example**

```
signal my_vector : bit_vector (0 to 4);
attribute MIDDLE : integer;
attribute MIDDLE of my_vector : signal is my_vector' LENGTH/2;
. . . . .
   my_vector' MIDDLE -- returns integer 2
```

Usage of Attributes (Example)

```
Entity masked_parity is
   generic ( size: integer :=5);
   port ( source : in bit_vector(0 to size);
            mask : in bit_vector(source'RANGE); -- RANGE (0 to size)
            result : out bit );
end masked_parity;

architecture behave of masked_parity is
begin
   process (source, mask)
      variable tmp : bit;
      variable masked_source : bit_vector (source'RANGE); -- RANGE (0 to size)
   begin
      masked_source := source and mask;
      tmp := masked_source(source'LEFT); -- LEFT
      for I in source'LEFT+1 to source'RIGHT loop -- RIGHT
         tmp := tmp xor masked_source(I);
      end loop;
      result <= tmp;
   end process;
end behave;
```


Usage of Attribute (Example)

```
function TO_INTEGER (ARG: UNSIGNED) return NATURAL is
  constant ARG_LEFT: INTEGER := ARG'LENGTH-1; -- LENGTH
  alias XARG: UNSIGNED(ARG_LEFT downto 0) is ARG;
  variable RESULT: NATURAL := 0;
begin
  if (ARG'LENGTH < 1) then
    assert NO_WARNING
       report "NUMERIC_BIT.TO_INTEGER: null detected, returning 0"
       severity WARNING;
    return 0;
  end if;
  for I in XARG'RANGE loop  -- RANGE
    RESULT := RESULT+RESULT;
    if XARG(I) = '1' then
      RESULT := RESULT + 1;
    end if;
  end loop;
  return RESULT;
end TO_INTEGER;
```

BUS and REGISTER

BUS and REGISTER

- VHDL Entity Class
  - BUS
    - make GUARDED signal (three-state)
  - REGISTER
    - make Latch inference

BUS Example

```
-- include the IEEE 1164 package to use type std_logic.
library ieee ;
use ieee.std_logic_1164.all ;

entity test_bus is
   port ( c,d : std_logic ;
            o : out std_logic BUS) ; -- An entity with a BUS entity-class signal
end test_bus ;

architecture behave of test_bus is
begin
   process (c,d)
   begin
      if (c = '1') then
         o <= d ;
      else
         o <= NULL ;  -- Switch-Off (Three-State)
      end if ;
   end process ;
end behave ;
```


REGISTER Example

```
-- include the IEEE 1164 package to use type std_logic.
library ieee ;
use ieee.std_logic_1164.all ;

entity test_reg is
   port ( c,d : std_logic ;
            o : out std_logic);
end test_reg ;

architecture behave of test_reg is
signal s_o : std_logic REGISTER;  -- REGISTER entity-class signal
begin
   process (c,d)
   begin
      if (c = '1') then
         s_o <= d ;
      else
         s_o <= NULL ;  -- DO NOTHING, Latch inference
      end if ;
   end process ;
   o <= s_o;
end behave ;
```

VHDL Synthesis

Flip-Flops
Three-,Bi-Dir Buffers
State Machines
Arithmetic and Related Logics
Resource Sharing and Optimization
Mux and Selectors
ROM, PLA, Decoders

Goals of This Section

- **More closely examine synthesis conventions**
- **Find out what is and is not synthesizable in today's tools**
- **See how HDL statements relate to actual hardware**
- **Examine the most common synthesis pitfalls**

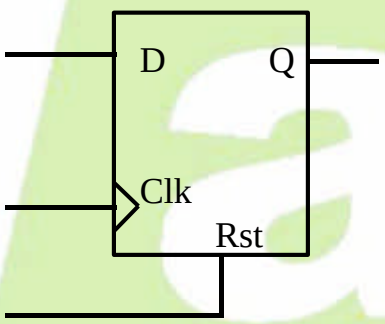
Flip-Flops

Sync/Async Rest or Reset

Latch

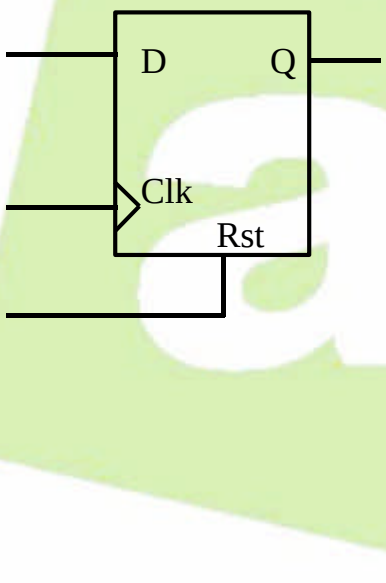
Chip Enable

Flip-Flop Convention



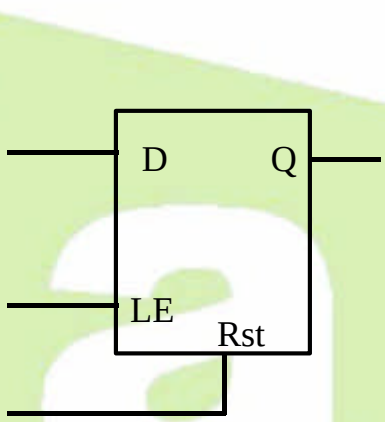

```
architecture behavior of dff is
begin
   process(Rst,Clk)
   begin
      if Rst = ‘1’ then   -- Async reset
         Q <= (others=>’0’);
      elsif (Clk = ‘1’ and Clk’event) then
         Q <= D;
      end if;
   end process;
end behavior;
```

Synchronous Reset Convention



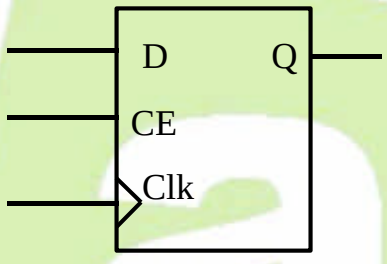

```
architecture behavior of dff is
begin
   process(Clk)
   begin
     if (Clk = ‘ 1’ and Clk’ event) then
      if Rst = ‘ 1’ then   -- Sync reset
         Q <= (others=>’ 0’ );
      else
         Q <= D;
      end if;
     end if;
   end process;
end behavior;
```

Latch Convention




```
architecture behavior of latch
is
begin
   process(LE, D)
   begin
     if LE = ‘ 1’ then
        Q <= D;
     end if;
   end process;
end behavior;
```

Clock Enable




```
architecture behavior of dff is
begin
   process(Clk)
   begin
      if Clk = ‘ 1’ and Clk’ event then
         if CE = ‘ 1’ then
            Q <= D;
         end if;
      end if;
   end process;
end behavior;
```

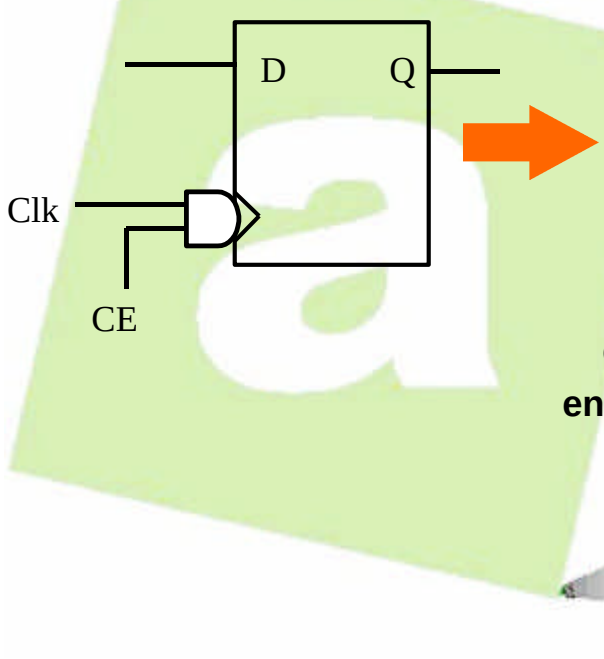
Flip-Flop Convention (block)

```
-- Level Sensitive
B1 : block (ena=‘ 1’ )
begin
      Q <=  guarded D;
end block;
```

```
-- Edge-Sensitive Sync reset
B2 : block (clk’ event and clk=‘ 1’ )
begin
      Q <=  guarded 0 when Rst=‘ 1’  else D;
end block;
```

```
-- Clock Enable
B2 : block (clk’ event and clk=‘ 1’ )
begin
      Q <=  guarded D when CE=‘ 1’  else Q;
end block;
```

Gated Clock




```
architecture behavior of dff is
begin
   process(Clk)
   begin
      if CE = ‘ 1’ and
            Clk = ‘ 1’ and Clk’ event then
         Q <= D;
      end if;
   end process;
end behavior;
```


Flip-Flop (Wait Statement)

```
architecture behavior of dff is
begin
   process                    NO Sensitivity List
   begin
      wait until clk’ event and clk=‘ 1’ ;
      Q <= D;
   end process;
end behavior;
```

Suggested Register Conventions

- Use a **process** statement (or procedure) to describe a flip-flop or latch
- Use **'event** attribute or **rising_edge** function to detect clock edge
- Use **if-then-else** style for reset and clock logic
- Avoid Gated Flip-Flop
- Avoid accidental Latch
- Verilog conventions are similar

Buffers

Three-State Buffers
Bi-Directional Buffers
Buses

Three-state Buffers

```
entity three-state is
port ( input_1, input_2 : in std_logic ;
       ena_1, ena_2 : in std_logic ;
       output : out std_logic ) ;
end three-state ;

architecture dataflow of three-state is
begin

  -- Dataflow
  output <= input_1 when ena_1 = '1' else 'Z' ;
  output <= input_2 when ena_2 = '1' else 'Z' ;

end dataflow ;
```

Multiple Drive!

Synthesizable, but "ena_1" and "ena_2" Never be '1' simultaneously!

```
-- Behavior
driver1 : process (ena_1, input_1)
begin
   if (ena_1='1') then
      output <= input_1 ;
   else
      output <= 'Z' ;
end if ;
end process ;

driver2 : process (ena_2, input_2)
begin
   if (ena_2='1') then
      output <= input_2 ;
   else
      output <= 'Z' ;
   end if ;
end process ;
```

Bi-Directional Buffers

```
entity bidir_function is
   port ( bidir_port : inout std_logic ;
          ena        : in std_logic ;
                ...
   ) ;
end bidir_function ;

architecture a_bidir of bidir_function is
signal internal_signal, internal_input : std_logic ;
begin

   bidir_port <= internal_signal when ena = '1' else 'Z' ;
   internal_input <= bidir_port ;
      . . . . .

   -- use internal_input
      . . . . . <= internal_input;

   -- generate internal_signal
   internal_signal <= . . . . . .

end a_bidir ;
```

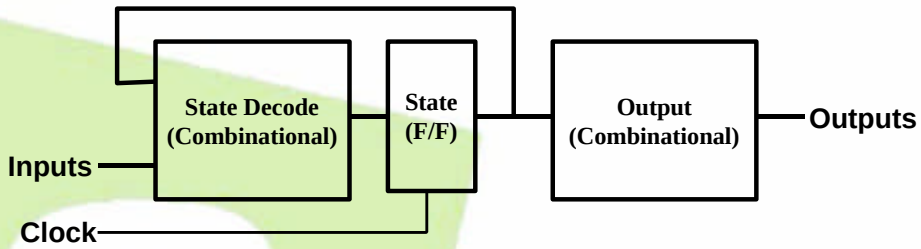
Buses

```
entity three-state is
    port ( input_signal_1, input_signal_2 : in   std_logic_vector (0 to 7) ;
           ena_1, ena_2 : in   std_logic ;
           output_signal : out std_logic_vector(0 to 7)
    ) ;
end three-state ;

architecture exemplar of three-state is
begin

    output_signal <= input_signal_1 when ena_1 = ' 1'  else ' ZZZZZZZZ' ;
    output_signal <= input_signal_2 when ena_2 = ' 1'  else ' ZZZZZZZZ' ;

end exemplar ;
```

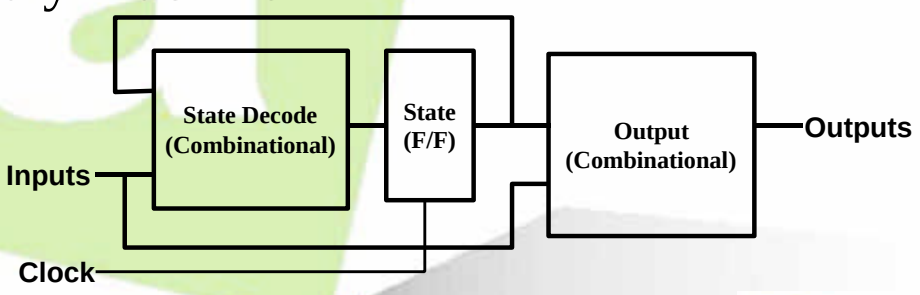
State Machine

Moore/Mealy Machine

- Moore Machine



- Mealy Machine



State Machine Coding Style

- **Use Enumeration Type to represent "State"**
- **Supply Power-Up and Reset Condition**
- **Use `case` statement for state transition**
- **DO NOT use `OTHER` entry in the `case`**
- **Use `if~else` for input condition**
- **Always assign state output under every condition**

State Machine Example

DRAM Refresh

```
entity ras_cas is
   port ( clk, cs, refresh, reset  : in    bit ;
          ras, cas, ready          : out bit ) ;
end ras_cas ;

architecture a_ras_cas of ras_cas is
   -- Define the possible states of the state machine
   type state_type is (s0, s1, s2, s3, s4) ;
   signal present_state, next_state : state_type ;
begin
```

Enumerate Type

```
registers : process (clk, reset)
begin
   -- process to update the present state
   if (reset=' 1' ) then
      present_state <= s0 ;
   elsif clk' event and clk = ' 1'  then
      present_state <= next_state;
   end if ;
end process ;
```

State F/F

State Machine Example (cont.)

```
transitions :
process (present_state, refresh, cs)
begin
   -- process to calculate the next state
   -- and the outputs
   case present_state is

   when s0 =>
      ras <= ' 1' ; cas <= ' 1' ; ready <= ' 1' ;
      if (refresh = ' 1' ) then
         next_state <= s3 ;
      elsif (cs = ' 1' ) then
         next_state <= s1 ;
      else
         next_state <= s0 ;
      end if ;

   when s1 =>
      ras <= ' 0' ; cas <= ' 1' ; ready <= ' 1' ;
      next_state <= s2 ;

   when s2 =>
      ras <= ' 0' ; cas <= ' 0' ; ready <= ' 0' ;
      if (cs = ' 0' ) then
         next_state <= s0 ;
      else
         next_state <= s2 ;
      end if ;

   when s3 =>
      ras <= ' 1' ; cas <= ' 0' ; ready <= ' 0' ;
      next_state <= s4 ;

   when s4 =>
      ras <= ' 0' ; cas <= ' 0' ; ready <= ' 0' ;
      next_state <= s0 ;

   end case ;
end process ;
end a_ras_cas ;
```

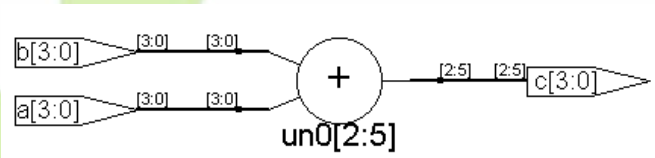
Arithmetic and Relational Logic

Overview

- VHDL Synthesizer Generate Arithmetic Logic
  - Technology Specific Macros
  - Use Optimized Library of Parametized Module
  - Synthesizable operators : “+”, “-”, “*”, “abs”
  - “-” apply same as “+”
  - “=”, “/=”, “<”, “>”, “<=”, “>=” generate comparator
- Avoid explosion of combinational logic
  - Constant operand
  - Division(“/”) by power of two (Shifter)
  - Synthesizable exponentiation(“**”), if both constant operands
- Integer Operation
  - Bit-width of generated arithmetic circuit is depend on operand’s range defined
  - 2’s complement implementation if integer range extends below 0
  - unsigned implementation, when positive range

Arithmetic Logic (RTL View)

```
entity add_int8 is
   port(
        a, b : in  integer range 0 to 15;
        c    : out integer range 0 to 15
   );
end add_int8;

architecture behave of add_int8 is
begin
   c <= a + b;
end behave;
```

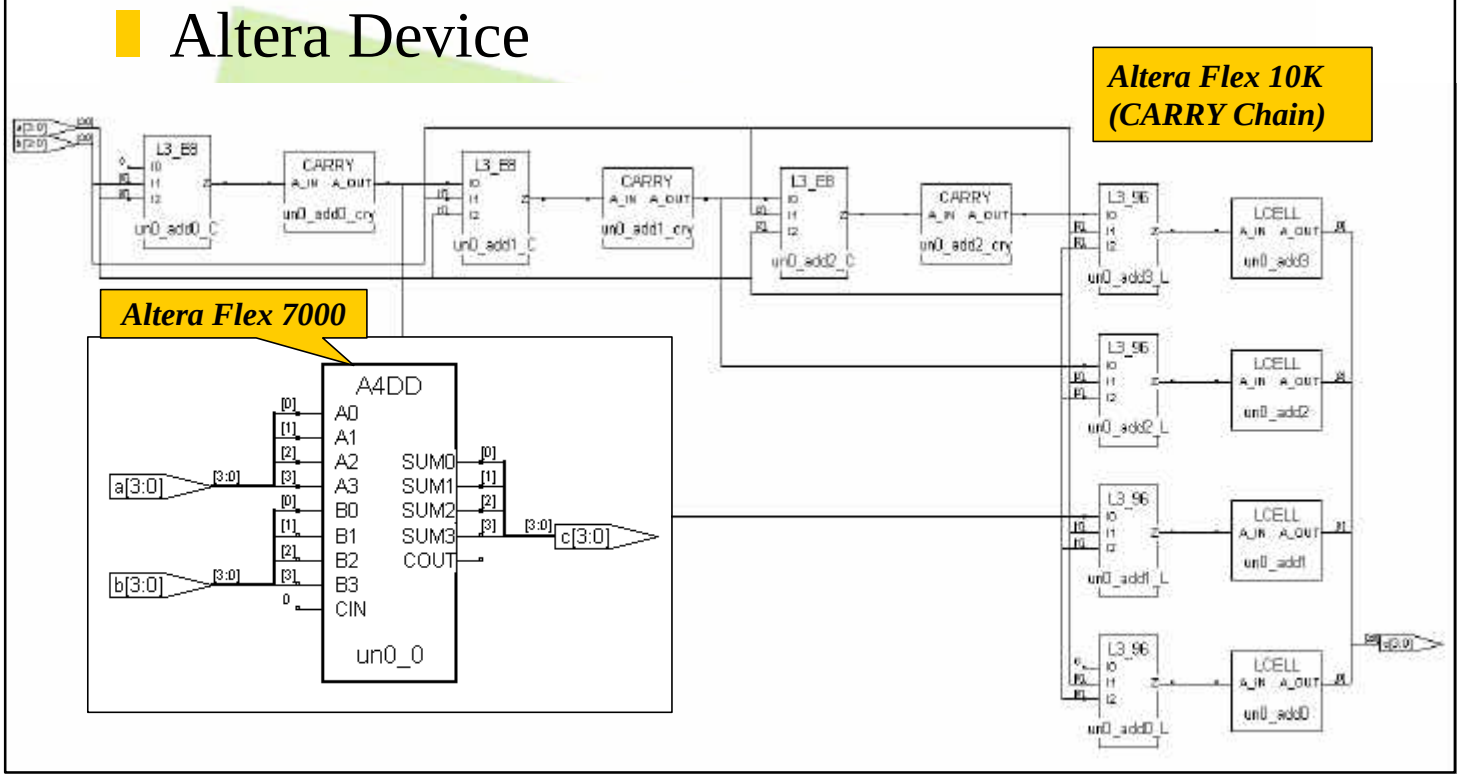


Synthesis RTL View

Arithmetic Logic (Tech. View)

- Altera Device

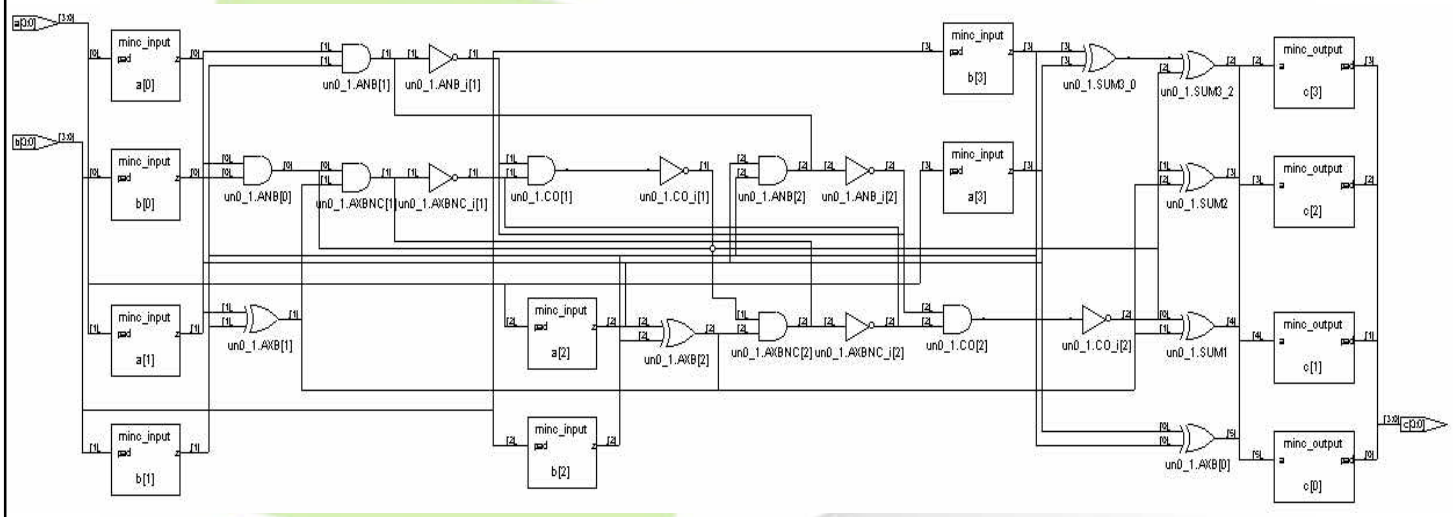
Altera Flex 10K (CARRY Chain)



Altera Flex 7000

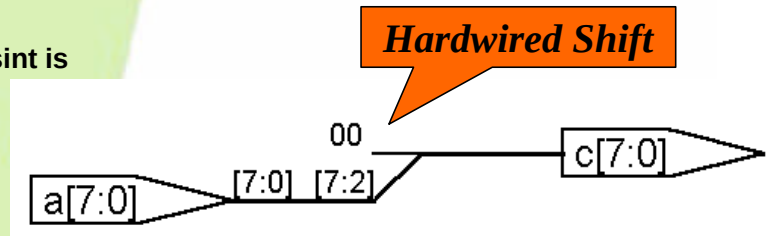
Arithmetic Logic (Tech. View)

- Vantis Mach Device



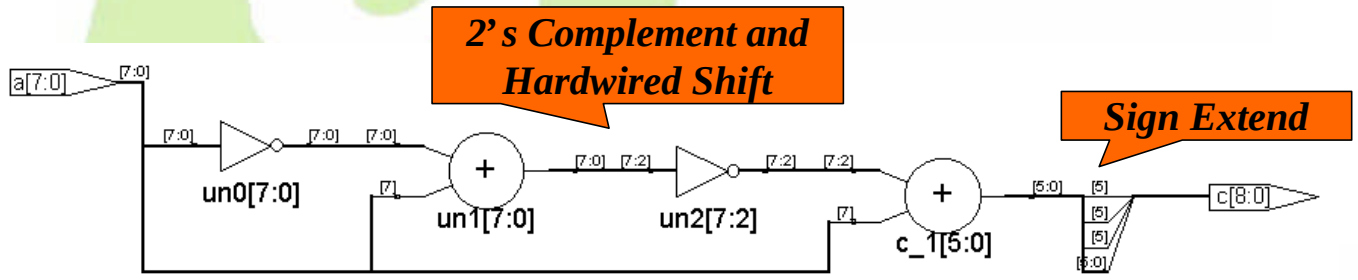
Integer Division Example (1)

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

entity div_posint is
   port( a : in  integer range 0 to 255;
         c : out integer range 0 to 255  );
end div_posint;

architecture behave of div_posint is
begin
   c <= a / 4;
end behave;
```



Integer Division Example (2)

```
entity div_cmpl is
   port( a : in  integer range -128 to 127;
         c : out integer range -256 to 255  );
end div_cmpl;

architecture behave of div_cmpl is
begin
   c <= a / 4;
end behave;
```



Improve Synthesis

Module Generation
Resource Sharing
Ranged Integer
Design Optimization

Module Generation

- Optimized generic modules (Parametized Library Modules)
  - Memory Module : ROM, RAM
  - Counters
  - Arithmetic and Relational logic
- Technology Specific Macros
- Mega/Macro Functions
- Most synthesizer tools support Technology specific macros and/or Module Generation options

LPM Package Example (Altera)

```
component LPM_ADD_SUB
    generic (
        LPM_WIDTH: positive;
        LPM_REPRESENTATION: string := SIGNED;
        LPM_DIRECTION: string := UNUSED;
        LPM_HINT : string := UNUSED;
        LPM_PIPELINE : integer := 0;
        LPM_TYPE: string := L_ADD_SUB );
    port (
        DATAA: in std_logic_vector(LPM_WIDTH-1 downto 0);
        DATAB: in std_logic_vector(LPM_WIDTH-1 downto 0);
        ACLR : in std_logic := '0';
        CLOCK : in std_logic := '1';
        CIN: in std_logic := '0';
        ADD_SUB: in std_logic := '1';
        RESULT: out std_logic_vector(LPM_WIDTH-1 downto 0);
        COUT: out std_logic;
        OVERFLOW: out std_logic );
end component;
```

Device Specific Synthesis

- Specifying place-and-route contraints, etc. can be tricky in an HDL.
- Examples: pin assignments, critical nets, clock buffers, etc.
- Hierarchy can be used to reference non-VHDL elements in some tools.
- No standards have been established; follow your synthesis documentation!

Pin Assignments

```
Entity CONTROL Is
    Port (Reset, Clk, Mode, VS, ENDFR: in std_ulogic;
          RAMWE, RAMOE: out std_ulogic;
          ADOE, INCAD: out std_ulogic);
    attribute pinnum: string;
    attribute pinnum of Clk: signal is "1"; -- Device Specific !
    attribute pinnum of Mode: signal is "2";
    attribute pinnum of Data: signal is "37,23,22,21,20,19,18,17";
    attribute pinnum of Addr: signal is "29,28,27,26,16,25,24,12” ;
    attribute pinnum of RAMWE: signal is "32";
    attribute pinnum of RAMOE: signal is "33";
    attribute pinnum of ADOE: signal is "34";
End CONTROL;
```

Device-specific Macros

```
architecture structure of lca_macro is
  -- Refer to vendor supplied Libraries
  component RAM16x1
      port (A0, A1, A2, A3, WE, D: in STD_LOGIC;
            O: out STD_LOGIC);
  end component;
begin
  . . .
  RAM1: RAM16x1 port map (A(0),A(1),A(2),A(3),WE,
                          DATAIN, DATAOUT) ;
  . . .
end structure;
```

Resource Share (use Temp signal)

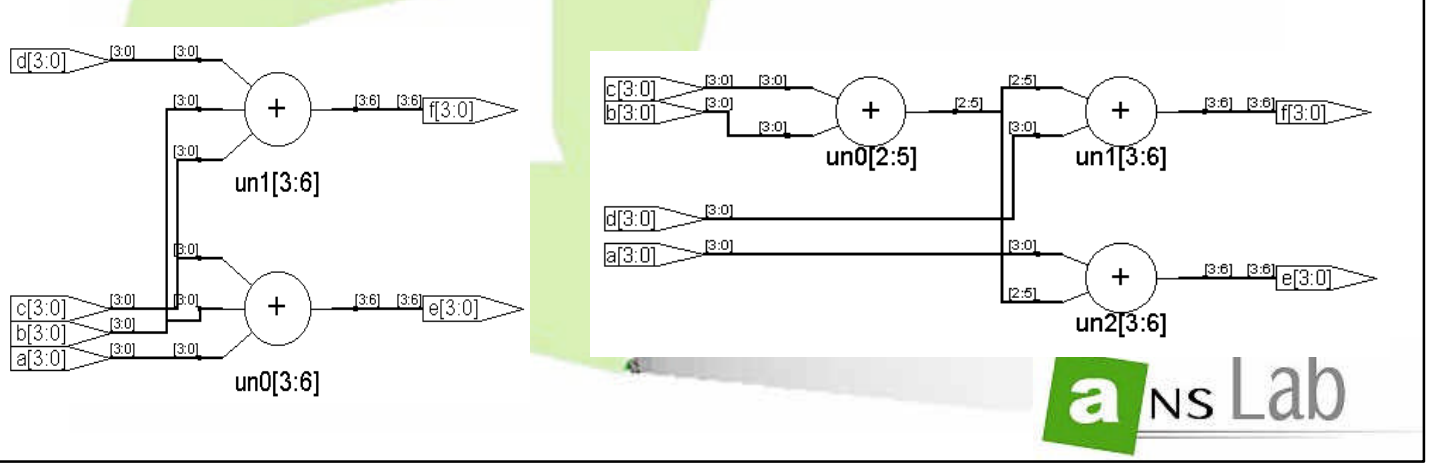
```
process(a,b,c,d)
begin
   if ( a+b=c ) then
      e <= a;
   elsif ( a+b=d ) then
      e <= b;
   else
      e <= c;
   end if;
end process;
```

```
process(a,b,c,d)
variable temp : integer range 0 to 15;
begin
   temp := a + b;
   if ( temp=c ) then
      e <= a;
   elsif ( temp=d ) then
      e <= b;
   else
      e <= c;
   end if;
end process;
```

Resource Share (use parenthesis)

```
process(a,b,c,d)
begin
   e <= a + b + c;
   f <= b + c + d;
end process;
```

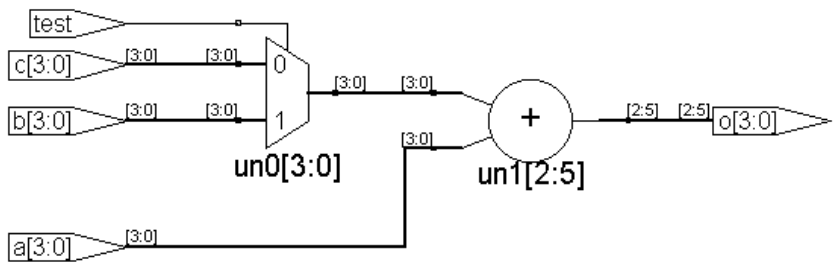
```
process(a,b,c,d)
begin
   e <= a + (b + c);
   f <= (b + c) + d;
end process;
```



Resource Share (mutual exclusive)

```
process(a,b,c,test)
begin
   if (test=TRUE) then
      o <= a + b;
   else
      o <= a + c;
   end if;
end process;
```

```
process(a,b,c,test)
variable temp : integer range 0 to
    15;
begin
   if (test=TRUE) then
      temp := b;
   else
      temp := c;
   end if;
   o <= a + temp;
end process;
```



Ranged Integer

- **Integer type : 32-bit default**

```
variable a, b, c : integer;
c := a + b; -- 32-bit adder
```

- **Ranged integer**

```
variable a, b, c : integer range 0 to 25[illegible]
c := a + b; -- 8-bit adder
```

Advanced Design Optimization

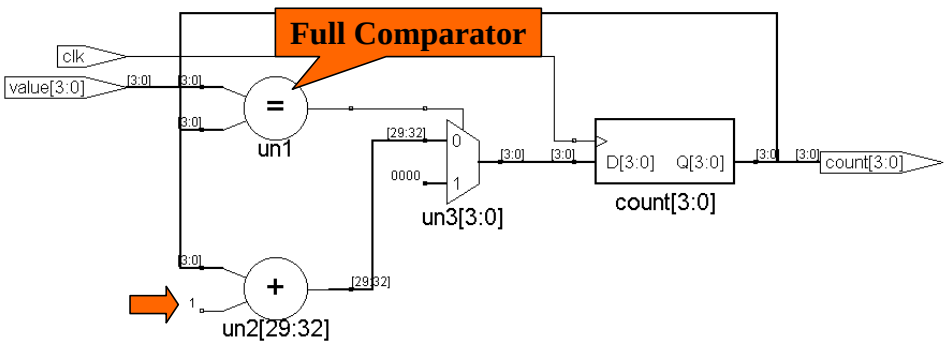
- **Change functionality without violating design spec.**
- **Understand VHDL and Circuit genetated by Synthesizer**
- **Example : Loadable counter**

```
L_count1 :
process
begin
   wait until clk'event and clk='1';
   if (count = value) then
      count <= 0;
      count_done <= TRUE;
   else
      count <= count + 1;
      count_done <= FALSE;
   end if;
end process;
```

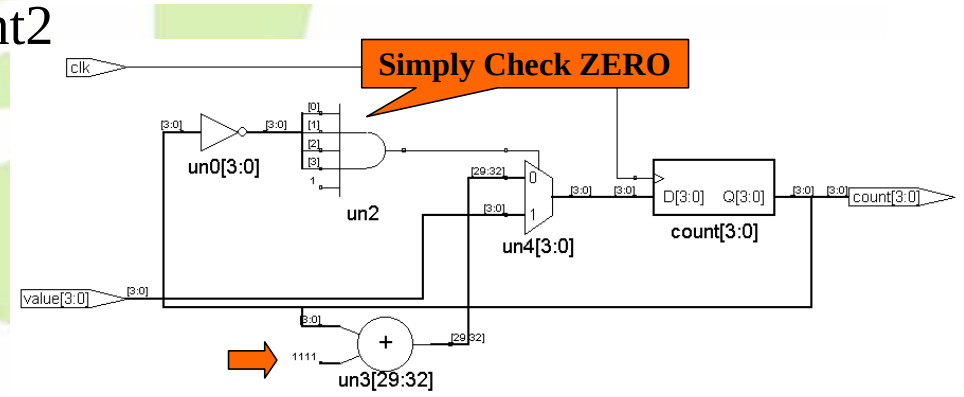
```
L_count2 :
process
   constant ZERO : integer range 0 to 15 := 0;
   begin
      wait until clk'event and clk='1';
      if (count = ZERO) then
         count <= value;
         count_done <= TRUE;
      else
         count <= count - 1;
         count_done <= FALSE;
      end if;
   end process;
```

Loadable counter

- L_count1



- L_count2



Multiplexer and selectors

- Use **case** statement

```
case test_vector is
   when "000" => o <= bus(0) ;
   when "001" | "010" | "100" => o <= bus(1) ;
   when "011" | "101" | "110" => o <= bus(2) ;
   when "111" => o <= bus(3) ;
end case;
```

- Use variable indexed array

```
signal vec : std_logic_vector (0 to 15) ;
signal o : std_logic ;
signal i : integer range 0 to 15 ;
...
o <= vec(i) ;
```

Equiv.

```
case i is
   when 0 => o <= vec(0) ;
   when 1 => o <= vec(1) ;
   when 2 => o <= vec(2) ;
   when 3 => o <= vec(3) ;
   ...
end case ;
```

ROMs, PLAs and Decoders

- **ROM**
  - **Address : case condition**
  - **Data : case statement**

```
case address is
   when “0000” => data <= “0111111”;
   when “0001” => data <= “0011000”;
   when “0010” => data <= “1101101”;
   when “0011” => data <= “1111100”;
   when “0100” => data <= “1011010”;
   when “0101” => data <= “1110110”;
   when “0110” => data <= “1110110”;
   when “0111” => data <= “0011100”;
   when “1000” => data <= “1111111”
   when “1001” => data <= “1111110”;
   when “1010” => data <= “1011111”;
   when “1011” => data <= “1110011”;
   when “1100” => data <= “0100111”;
   when “1101” => data <= “1111001”;
   when “1110” => data <= “1100111”;
   when “1111” => data <= “1000111”;
end case;
```


Seven-Segment Decoder

```
type seven_segment is array (6 downto 0) ;
type rom_type is array (natural range <>) of seven_segment ;
constant hex_to_7 : rom_type (0 to 15) :=
   ( "0111111", -- 0
     "0011000", -- 1
     "1101101", -- 2
     "1111100", -- 3
     "1011010", -- 4
     "1110110", -- 5
     "1110111", -- 6
     "0011100", -- 7
     "1111111", -- 8
     "1111110", -- 9
     "1011111", -- A
     "1110011", -- B
     "0100111", -- C
     "1111001", -- D
     "1100111", -- E
     "1000111” ) ; -- F
```

Display segment index numbers :

```
   2
1     3
   6
0     4
   5
```

-- Now, the ROM field can be accessed via a integer index

```
display_bus <= hex_to_7 (i) ;
```

PLA and ROM

- ROM
  - fixed input decoder Plane-as address
  - all cased `case` statement
- PLA
  - Input allows don’ t care condition “-”,”x”
  - Cannot use `case` statement
- Use Modeled ROM/PLA for large size design

2-Dimensional PLA

```
type std_logic_pla is array (natural range <>, natural range <>) of std_logic;
. . . . .
procedure pla_table (
   constant  invec   : in   std_logic_vector;
   signal      outvec : out std_logic_vector;
   constant  table    : in   std_logic_pla
) is
variable x : std_logic_vector (table'range(1)) ; -- product lines
variable y : std_logic_vector (outvec'range) ;   -- outputs
variable b : std_logic ;
begin
   assert (invec'length + outvec'length = table'length(2))
   report "Size of Inputs and Outputs do not match table size"
   severity ERROR ;
```

2-Dimensional PLA (cont' ed)

```
-- Calculate the AND plane
x := (others=>'1');
for i in table'range(1) loop -- PLA Table ROWs
   for j in invec'range loop
      b := table (i,table'left(2)-invec'left+j) ;
      if (b='1') then
         x(i) := x(i) AND invec (j) ;
      elsif (b='0') then
         x(i) := x(i) AND NOT invec(j) ;
      end if ;
      -- If b is not '0' or '1' (e.g. '-') product line is insensitive to invec(j)
   end loop ;
end loop ;
```

2-Dimensional PLA (cont' ed)

```
-- Calculate the OR plane
y := (others=>'0') ;
for i in table'range(1) loop
   for j in outvec'range loop
      b := table(i,table'right(2) - outvec'right+j) ;
      if (b='1') then
         y(j) := y(j) OR x(i);
      end if ;
   end loop ;
end loop ;
outvec <= y ;

end pla_table ;
```

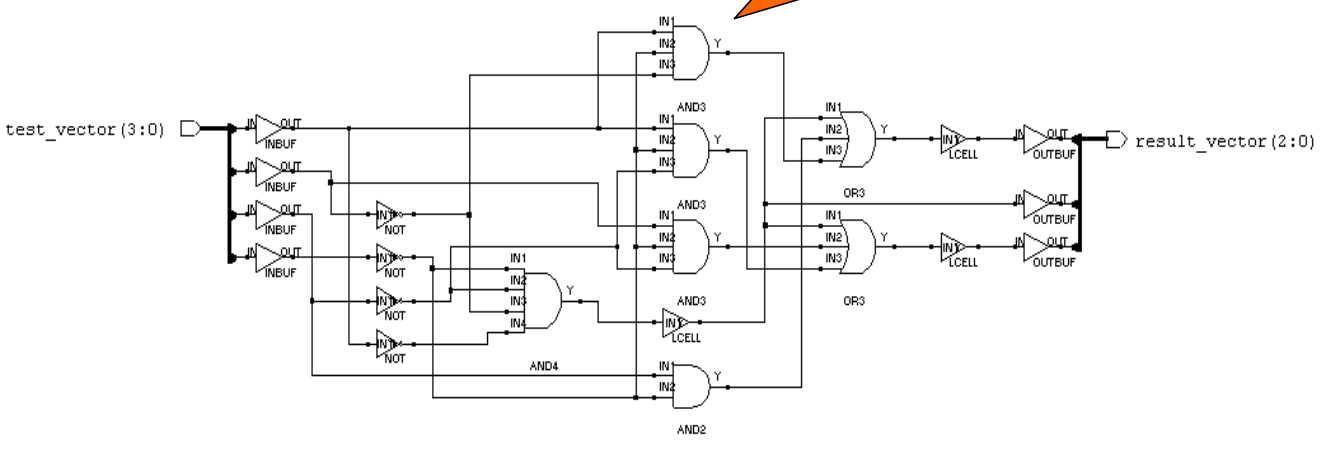
2-Dimensional PLA (cont' ed)

```
constant pos_of_fist_one : std_logic_pla (4 downto 0, 6 downto 0) :=
    ( "1- - - 000",    -- first ' 1' is at position 0
      "01- - 001",     -- first ' 1' is at position 1
      "001- 010",      -- first ' 1' is at position 2
      "0001011",       -- first ' 1' is at position 3
      "0000111" ) ; -- There is no ' 1' in the input
signal test_vector : std_logic_vector (3 downto 0) ;
signal result_vector : std_logic_vector (2 downto 0) ;
...
-- Now use the pla table procedure with PLA pos_of_first_one
-- test_vector is the input of the PLA, result_vector the output.
...
pla_table ( test_vector, result_vector, pos_of_first_one) ;
```

2-Dimensional PLA Synthesis (1)

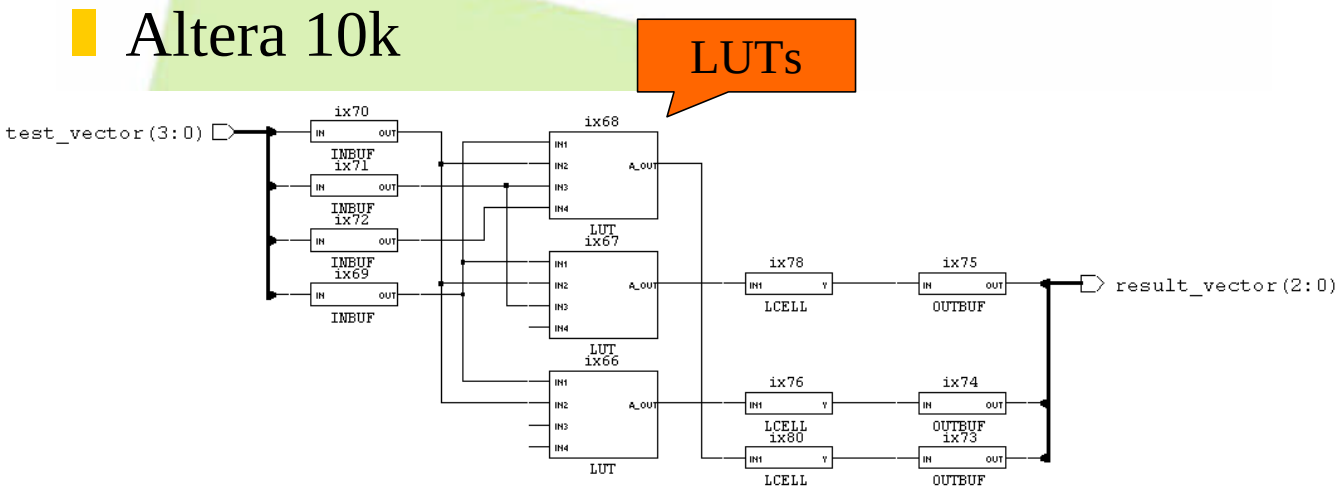
- Altera 7000

Synthesis as glue logic



2-Dimensional PLA Synthesis (2)

- Altera 10k

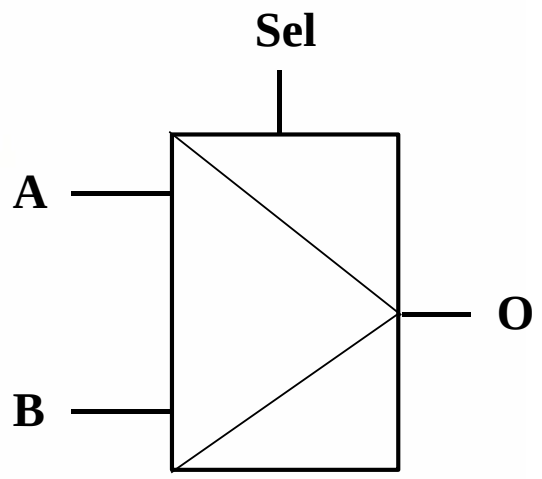


Synthesis pitfalls

Combinational Logic

- **IF-ELSE Statement:**

```
process (A,B,Sel)
begin
   if  (Sel = ‘1’) then
      O <= A;
   else
      O <= B;
   end if;
end process;
```



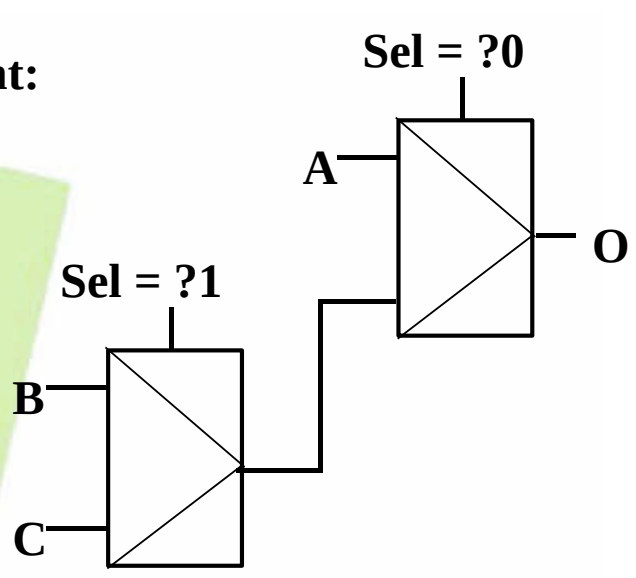
No problem here, but...

aNS Lab

Combinational Logic (cont)

- **IF-ELSIF-ELSE Statement:**

```
process (A,B,C,Sel)
begin
   if  (Sel = ‘00’) then
      O <= A;
   elsif (Sel = ‘01’) then
      O <= B;
   else
      O <= C;
   end if;
end process;
```



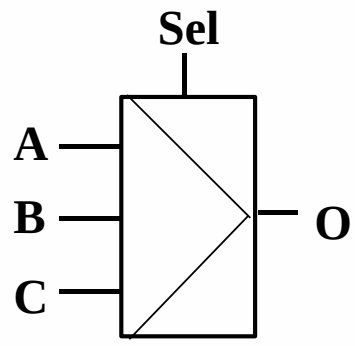
Be careful with priorities!

aNS Lab

Combinational Logic (cont)

- **CASE Statement:**

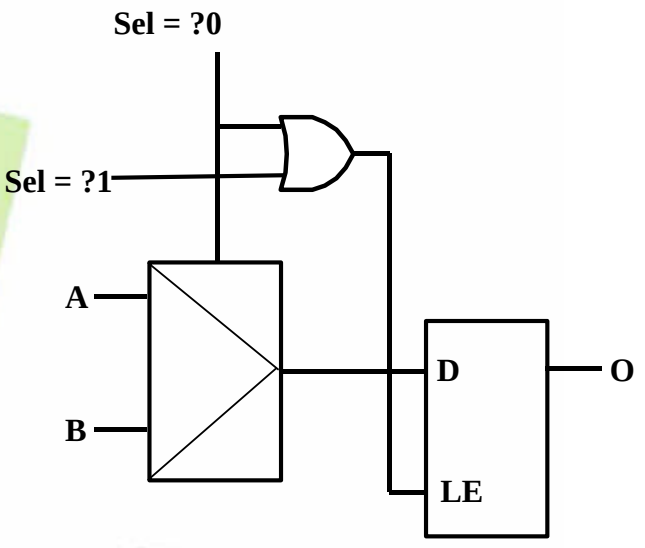
```
process (A,B,C,Sel)
begin
   case  Sel is
      when ‘00’ =>
         O <= A;
      when ‘01’ =>
         O <= B;
      when others =>
         O <= C;
   end case;
end process;
```



When in doubt, use a case statement!

Accidental Latches

```
process (A,B,Sel)
begin
   if  (Sel = ‘00’) then
      O <= A;
   elsif (Sel = ‘10’) then
      O <= B;
   end if;
end process;
```



Don’t leave out any conditions or you may get latches!

Accidental Latches (cont)

```
architecture behavior of dff is
begin
   process(Rst,Clk)
   begin
      if Rst = ‘ 1’  then
         Q1 <= (others=>’ 0’ );
      elsif rising_edge(Clk) then
         Q1 <= D1;
         Q2 <= D2;   -- Note: should generate error
      end if;
   end process;
end behavior;
```

Hint: what happens to Q2 when Rst is held high on a rising edge of Clk?

Accidental Latches (cont)

```
process (current_state)
begin
   case current_state is
      when S0 =>
         O1 <= ?; O2 <= ?;
      when S1 =>
         O1 <= ?; O2 <= ?;
      when S2 =>
         O1 <= ?;        -- O2 doesn’ t matter in state S2 or S3
      when S3 =>
         O1 <= ?;
   end case;
end process;
```

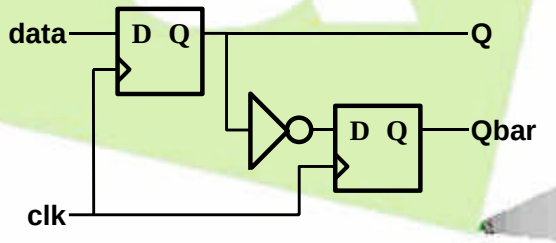
Accidental Latches (cont)

```
process (current_state)
begin
   O2 <= ?;                -- Be safe! Set default values here.
   case current_state is
      when S0 =>
         O1 <= ?; O2 <= ?;
      when S1 =>
         O1 <= ?; O2 <= ?;
      when S2 =>
         O1 <= ?;          -- O2 doesn' t matter in state S2 or S3
      when S3 =>
         O1 <= ?;
   end case;
end process;
```

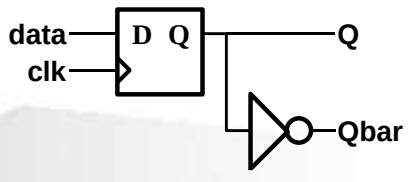

Q & Qbar

```
ENTITY qqbar IS
  PORT ( data, clk : IN     std_logic;
                q,qbar    : OUT std_logic );
END qqbar;
```

```
ARCHITECTURE a_qqbar OF qqbar IS
BEGIN
   PROCESS (data, clk)
   BEGIN
      IF clk='1' AND clk'EVENT THEN
         q <= data;
         qbar <= NOT q;
      END IF;
   END PROCESS;
END a_qqbar;
```




```
ARCHITECTURE a_qqbar_1 OF qqbar IS
BEGIN
   PROCESS (data, clk)
   BEGIN
      IF clk='1' AND clk'EVENT THEN
         q <= data;
      END IF;
   END PROCESS;
   qbar <= NOT q;
END a_qqbar_1;
```



Unsynthesizable Functions

```
function Vec_To_Int  (INVEC : bit_vector) return integer is
   variable result: integer := 0;
begin
   for i in INVEC'reverse_range loop
      result := result * 2;
      if (INVEC(i) = '1') then
         result <= result + 1;
      end if;
   end loop;
   return result;
end Vec_To_Int;
```

Theoretically synthesizable (check, it's just a wire) but you'll never get it through the synthesis tool!

UnSynthesizable VHDL

- **After clauses (delay specifications)**
- **Initialization values**
- **Signals that have multiple clocks**
- **Unconstrained loops**
- **Text I/O (File types)**
- **Access types**
- **Real types (floating point numbers)**

Initialization Values

```
type state_type is (S0,S1,S2,S3,S4) ;
signal current_state: state_type := S0 ;
```

- Simulation will start in state S0
- Synthesis will ignore initialization, so you get the device powerup for the initial state
- Summary: always provide a reset!

Multiple Clocks

```
Process(clk0, clk1)
begin
   if clk0'EVENT and clk0='1' then
      q <= d0;
   elsif clk1'EVENT and clk1='1' then
      q <= d1;
   end if;
end process;
```

```
Process(clk0)
begin
   if clk0'EVENT and clk0='1' then
      q <= d0;
   end if;
end process;

Process(clk1)
begin
   if clk1'EVENT and clk1='1' then
      q <= d1;
   end if;
end process;
```

- Not synthesizable
- Rewrite with multiple registers

Suggested Reading

- **IEEE, *IEEE Standard 1076-1993 VHDL Language Reference Manual***
- **IEEE, *IEEE Standard 1364 Verilog Language Reference Manual***
- **Pellerin, David and Douglas Taylor, *VHDL Made Easy*, Prentice Hall, 1996**
- **Armstrong and Gray, *Structured logic Design with VHDL*, Prentice-Hall,1996**
- **Internet: comp.lang.vhdl, comp.lang.verilog, http://www.vhdl.org**