united nations
educational, scientific
and cultural
organization

international atomic
energy agency

the
abdus salam
international centre for theoretical physics

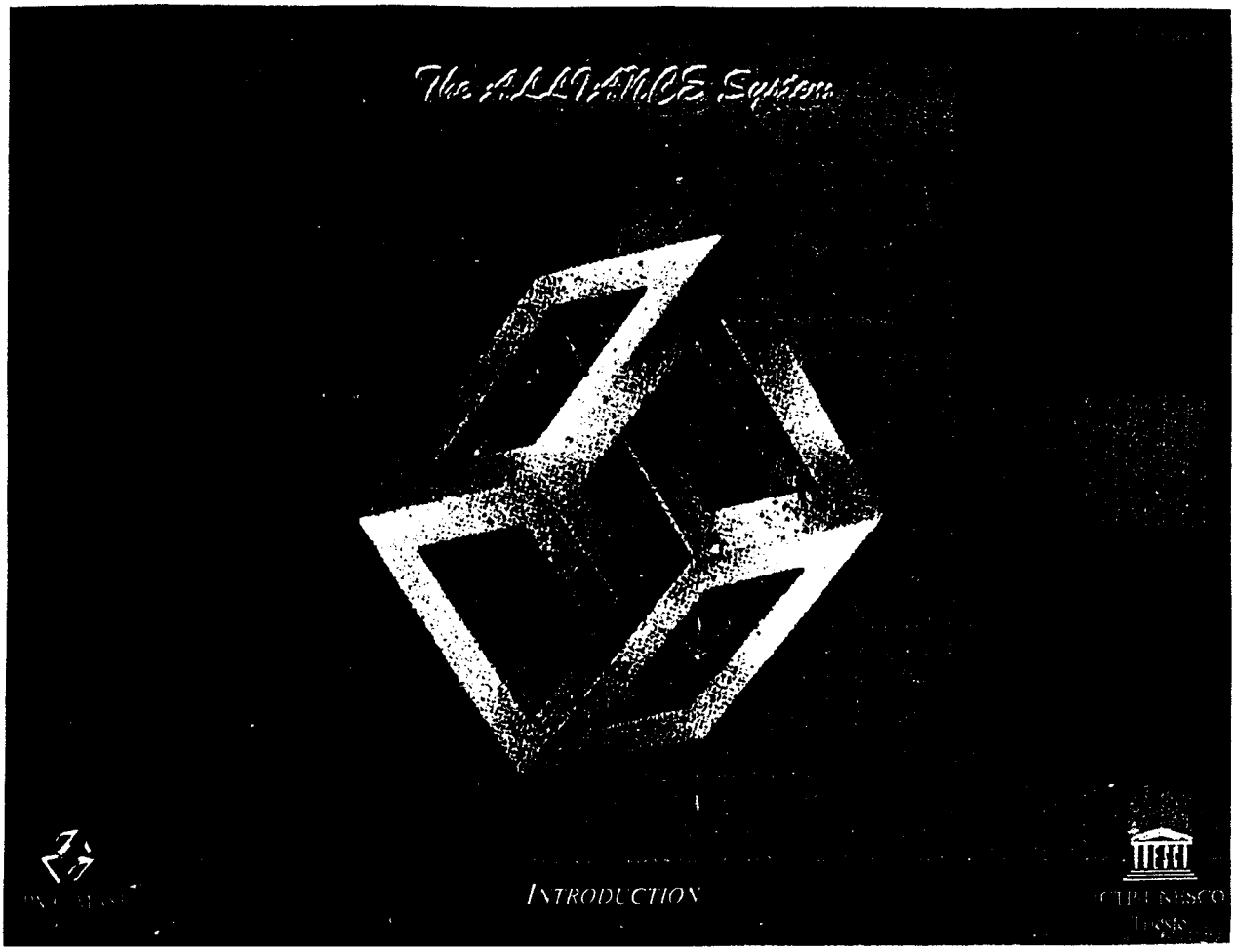
301/1352-4

MICROPROCESSOR LABORATORY SEVENTH COURSE ON BASIC VLSI DESIGN TECHNIQUES

29 October - 23 November 2001

VHDL, ALLIANCE

Nizar ABDALLAH
Actel Corporation
955 East Arues Avenue
Sunnyvale, CA 94-86-4533
U.S.A.



NIZAR ABDALLAH

LABORATOIRE MASI - EQUIPE CAO-VLSI

UNIVERSITE PIERRE ET MARIE CURIE (PARIS VI)

4, PLACE JUSSIEU, 75252 PARIS CEDEX 05
FRANCE

✉ : Nizar.Abdallah@masi.ibp.fr
☎ : 33 - 1 44 27 53 99

OUTLINE

I - INTRODUCTION

II - DESIGN METHODOLOGY: AN OVERVIEW

III - ABSTRACTION LEVELS IN ALLIANCE

IV - VHDL: A HARDWARE DESCRIPTION LANGUAGE

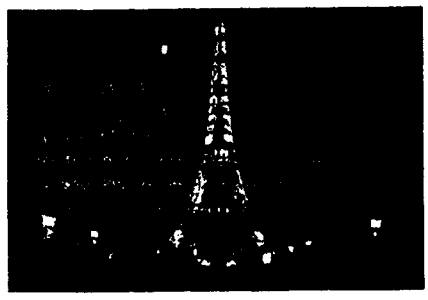
V - VHDL: THE ALLIANCE SUBSET

VI - ALLIANCE: A COMPLETE DESIGN SYSTEM

VII- TODAY'S CHALLENGES IN CAD TOOLS

THE MASI LABORATORY

UNIVERSITY PIERRE ET MARIE CURIE
NATIONAL CENTRE OF SCIENTIFIC RESEARCH



168 RESEARCHERS

ARCHITECTURE	59	NETWORKS & PERFORMANCES	30
DISTRIBUTED SYSTEMS	36	PARALLEL ALGORITHMS	17

THE MASI LABORATORY

UNIVERSITY PIERRE ET MARIE CURIE
NATIONAL CENTRE OF SCIENTIFIC RESEARCH

168 RESEARCHERS

• ARCHITECTURE	59	• NETWORKS & PERFORMANCES	30
• DISTRIBUTED SYSTEMS	36	• PARALLEL ALGORITHMS	17

THE ARCHITECTURE GROUP

CAD FOR VLSI		ARCHITECTURE	
PORTABLE LIBRARIES	9	SUPERSCALAR PROCESSOR	5
VERIFICATION	7	RCUBE ROUTER	8
LOGIC SYNTHESIS	5	RAPID COPROCESSOR	6
ARCHITECTURE SYNTHESIS	4		
TEST	5		

EDUCATION TARGET

- UNDERGRADUATE STUDENTS: (≈ 80 STUDENTS AND 72 HOURS)
  - ♦ ELECTRICAL ENGINEERING
  - ♦ COMPUTER SCIENCE
- POSTGRADUATE STUDENTS (≈ 60 STUDENTS AND 300 HOURS)
  - ♦ DEA MEMI
  - ♦ DESS CIMI

THE ALLIANCE SYSTEM

- A COMPLETE SET OF CAD TOOLS FOR DIGITAL CMOS VLSI DESIGN.
- PROPOSES A DESIGN METHODOLOGY.
- PORTABLE, COMPACT AND EASY TO LEARN.
- ALLIANCE IS TOTALLY FREE.

OUTLINE

I - INTRODUCTION.

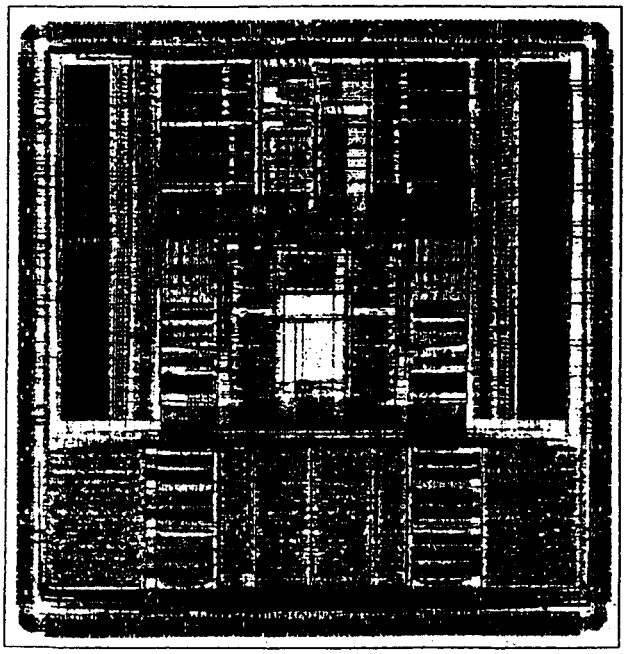
II - DESIGN METHODOLOGY: AN OVERVIEW.

III - ABSTRACTION LEVELS IN ALLIANCE.

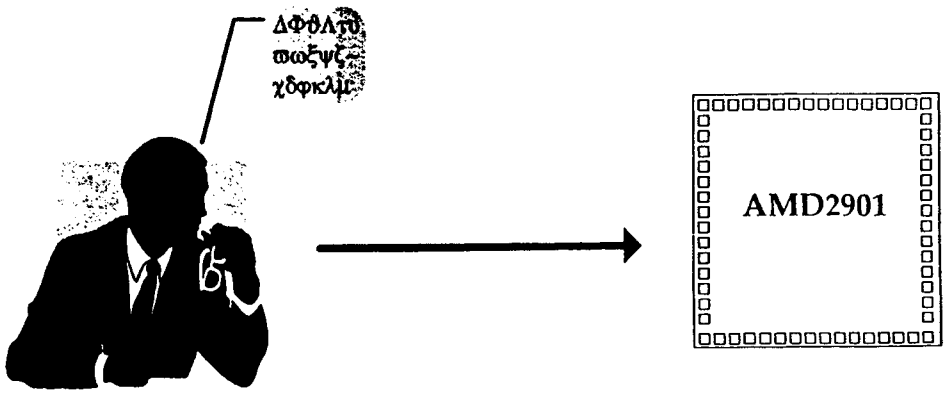
IV - VHDL: AN OVERVIEW.

V - VHDL: THE ALLIANCE SUBSET.

VI - ALLIANCE: A COMPLETE DESIGN SYSTEM.



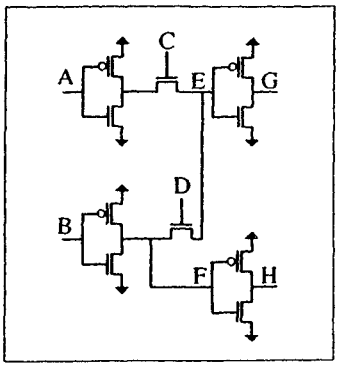
DESIGNER'S DREAM



MILLIONS OF SEGMENTS PUT TOGETHER.

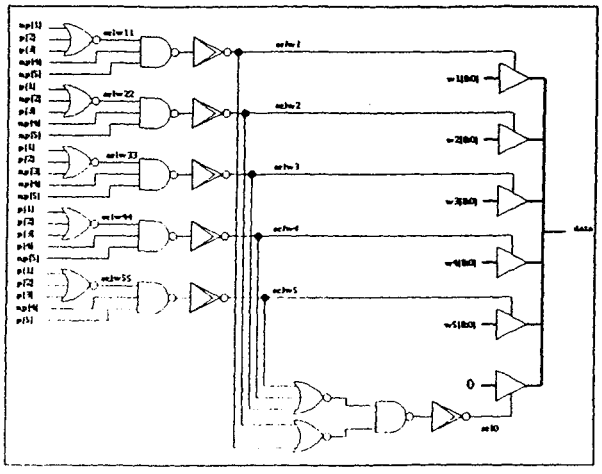
HOW TO DEAL WITH SUCH COMPLEXITY ?

ONE MILLION OF TRANSISTORS CONNECTED TOGETHER.



ß STILL TOO COMPLEX.....!!!

HUNDRED THOUSAND OF CELLS CONNECTED TOGETHER.



× STILL TOO COMPLEX.....!!!

DOZEN OF FUNCTIONAL BLOCKS THAT COMMUNICATE TOGETHER.

✓ I UNDERSTAND (OUF !!!)

A SET OF EQUATIONS THAT REFLECT THE WHOLE FUNCTIONALITY OF THE CIRCUIT.

```
entity  adder is
port (
        a, b  Bit;
        c, d : bit
      );

architecture  adder is

a <= b or c
d <= b and c,
end.
```

✓ I UNDERSTAND WHAT THIS CIRCUIT IS SUPPOSED TO DO.

SO,

HOW TO DEAL WITH SUCH COMPLEXITY ?

✓ **ABSTRACTION**

✓ **HIERARCHY**

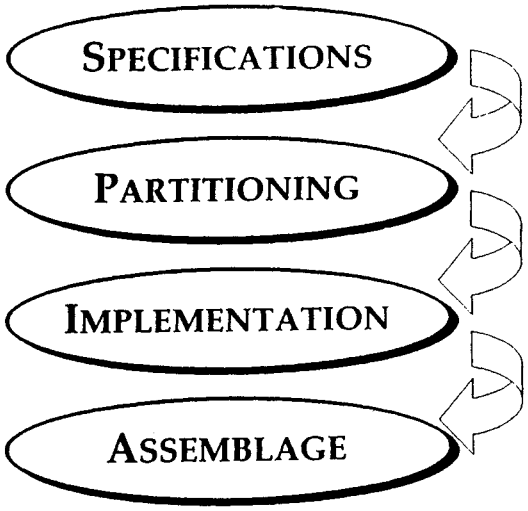
LEVELS OF ABSTRACTION

TO GO ACROSS THESE DIFFERENT LEVELS OF ABSTRACTION

I NEED

A DESIGN METHODOLOGY

DESIGN METHODOLOGY
TOP-DOWN METHODOLOGY



STEP 1: SPECIFICATIONS (1)

PUT DOWN THE CIRCUIT CONCEPT.

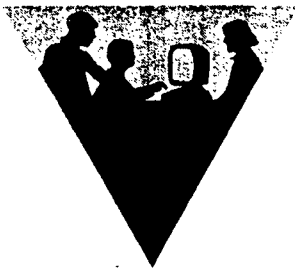
TWO REASONS:

- TO BE ABLE TO CHECK IT BEFORE MANUFACTURING.
- TO HAVE A REFERENCE MANUAL FOR COMMUNICATION.

STEP 1: SPECIFICATIONS (2)

COMMUNICATION LANGUAGE.

BETWEEN DIFFERENT PEOPLE ON THE PROJECT AND BETWEEN PEOPLE AND COMPUTERS.



✗ NO ORDINARY LANGUAGE.

✓ ACCURATE LANGUAGE.

✓ A LANGUAGE THAT CAN BE SIMULATED.

STEP 2: HOW TO ? (2)

THE CUTTING IS GUIDED BY:

1. REGULARITY OR NOT.

- IDENTIFY REGULAR BLOCKS.
- IDENTIFY RANDOM LOGIC BLOCKS.

STEP 2: HOW TO ? (3)

THE CUTTING IS GUIDED BY:

2. TIMING ASPECTS.

- COARSE ESTIMATION OF TIMING.
- LOOKING FOR A GOOD BALANCE.

STEP 2: HOW TO ? (4)

THE CUTTING IS GUIDED BY:

3. TOPOLOGY.

- ALREADY IN MIND THE CIRCUIT FORM.
- AN IDEA ABOUT THE SIZE OF EACH PART.
- AN IDEA ABOUT THE ROUTING.
- OPTIMIZING SILICON AREA USAGE.

STEP 2: HOW TO ? (5)

THE CUTTING IS GUIDED BY:

4. TECHNOLOGY.

- USING GAAS OR CMOS ?
- USING PALS OR STANDARD CELLS ?

STEP 2: HOW TO ? (6)

THE CUTTING IS GUIDED BY:

5. CAD TOOLS.

- WHAT TOOLS DO I HAVE TO MAKE MY CIRCUIT ?

EX: NO SYNTHESIS TOOLS SO I TRY TO REDUCE THE RANDOM LOGIC PART.

STEP 3: IMPLEMENTATION

EACH PART WILL BE IMPLEMENTED USING A PARTICULAR METHOD. WHEN I SPLIT MY CIRCUIT, I HAVE ALREADY DECIDED WHICH ONE.

STEP 4: ASSEMBLAGE

THE ASSEMBLAGE IS DONE IN A HIERARCHICAL WAY, STARTING FROM THE LOWEST LEVEL.

CONCLUSION (1)

AT EACH STEP, THE INFORMATION IS ENHANCED:

1. FROM THE IDEA DOWN TO THE SPECIFICATIONS.
2. WHEN STRUCTURING THE MODEL IN AN OTHER WAY.
3. .....

⇒ AT EACH STEP, A VERIFICATION IS TO BE DONE.

CONCLUSION (2)

ALL ALONG THE METHODOLOGY, WE HANDLED DIFFERENT VIEWS:

1. EQUATIONS.
2. NETLISTS.
3. LAYOUT.

CONCLUSION (3)

THERE IS A METHOD.

OUTLINE

I - INTRODUCTION.

II - DESIGN METHODOLOGY: AN OVERVIEW.

III - ABSTRACTION LEVELS IN ALLIANCE.

IV - VHDL: AN OVERVIEW.

V - VHDL: THE ALLIANCE SUBSET.

VI - ALLIANCE: A COMPLETE DESIGN SYSTEM.

3 DIFFERENT VIEWS

ALL ALONG THE METHODOLOGY, WE HANDLED DIFFERENT VIEWS:

1. BEHAVIORAL VIEW (EQUATIONS).
2. STRUCTURAL VIEW (NETLISTS).
3. LAYOUT VIEW.

BEHAVIORAL VIEW (1)

LOGICAL EQUATIONS

- DESCRIPTION FORMALISM.

A SET OF LOGICAL EQUATIONS (BOOLEAN) REPRESENTING BOOLEAN FUNCTIONS.

EXAMPLE: $U = A.(A+B)$ $V = C.D$ $T = D \oplus E$

$X = U.V$ $Y = V + T + X$ $Z = T.E$

BEHAVIORAL VIEW (2)

LOGICAL EQUATIONS

- REPRESENTATION.

  A DIRECTED ACYCLIC GRAPH INCLUDING THREE KINDS OF NODES: INPUT, INTERMEDIARY, OUTPUT.

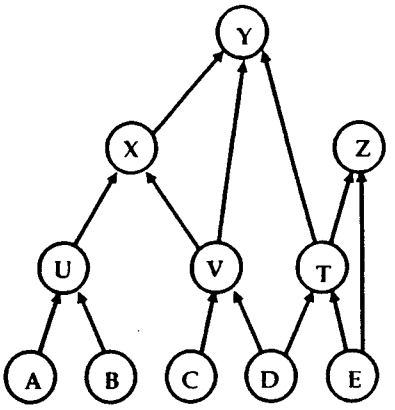
  EACH INTERMEDIARY OR OUTPUT NODE IS ASSOCIATED TO A LOGICAL EXPRESSION.

  EACH NODE IS ASSOCIATED TO A VARIABLE NAME.

BEHAVIORAL VIEW (3)

BOOLEAN NETWORK

- REPRESENTATION.



STRUCTURAL VIEW (1)

FOR ALL THESE VIEWS, WE ARE LOOKING FOR BASIC CONCEPTS: COMPLETELY INDEPENDENT FROM A GIVEN LANGUAGE.

IN THE STRUCTURAL VIEW:

- CONNECTORS: ID, DIRECTION, ETC....
- SIGNALS: ID, TYPE (EXTERNAL OR NOT), ETC....
- INSTANCE: ID, MODEL NAME, PORTS, ETC....

LAYOUT VIEW (1)

SYMBOLIC LAYOUT: PRINCIPLES

- PORTABILITY
- SIMPLICITY
- ROBUSTNESS

LAYOUT VIEW (2)

SYMBOLIC LAYOUT: OUR APPROACH

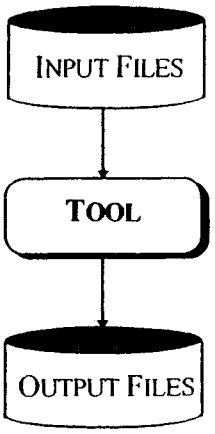
THIN FIXED GRID, SYMBOLIC LAYOUT.

DISTANCES FORM CENTER TO CENTER ⇒ GOOD DENSITIES.

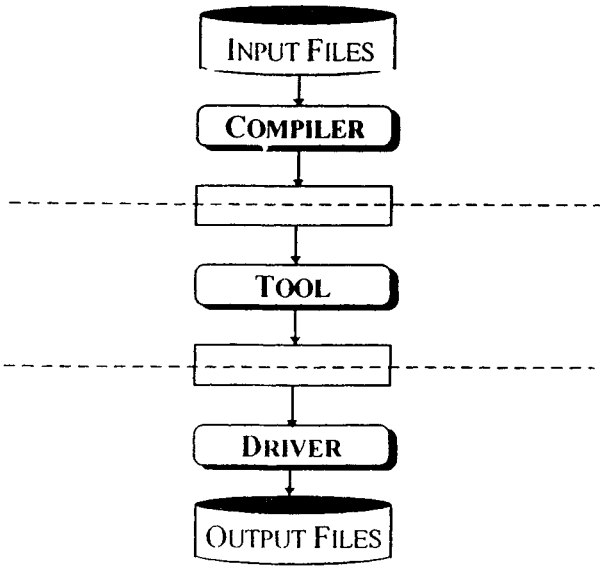
SPECIAL SYMBOLIC LAYOUT EDITOR.

AUTOMATIC TRANSLATION FROM SYMBOLIC TO PHYSICAL.

HOW TO DEAL WITH THESE VIEWS ? (1)



HOW TO DEAL WITH THESE VIEWS ? (2)



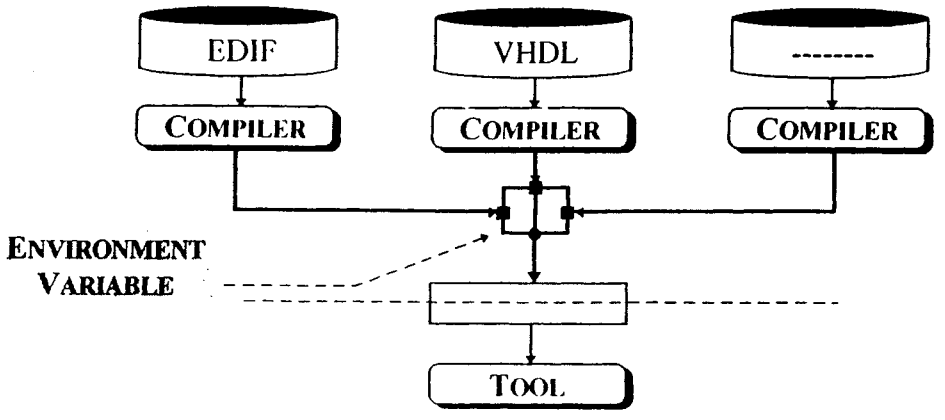
INDEPENDENCE (1)

A MAJOR IDEA IN ALLIANCE IS ITS INDEPENDENCE FROM ANY GIVEN LANGUAGE.

IDENTIFY THE CONCEPTS THAT:

- ✘ DO NOT DEPEND ON A LANGUAGE.
- ✓ DEPENDS ON THE ABSTRACTION LEVEL.

INDEPENDENCE (2)



OUTLINE

I - INTRODUCTION

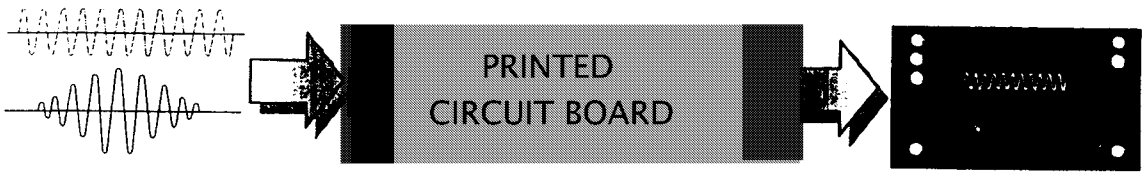
II - DESIGN METHODOLOGY: AN OVERVIEW

III - ABSTRACTION LEVELS IN ALLIANCE

IV - VHDL: A HARDWARE DESCRIPTION LANGUAGE

Why an HDL ? (1)

✘ Hardware Solutions Limits



INPUT WAVEFORM GENERATOR

DIGITAL ANALYZER

Why an HDL ? (2)

- ✘ Increasing Complexity
- ✘ Increasing Cost in Time & Investment
- ✘ Increasing Knowledge Requirement

A Software Solution is Needed

Why an HDL ? (3)

- ✘ Programming Language not Suited

A Special Purpose Language : HDL

Why VHDL ? (1)

Circuit Manufacturers Fully Satisfied with their Proprietary HDLs....

Why VHDL ? (2)

Problems for system manufacturers

✘ Different vendors ➡ different incompatible HDLs

✘ Impossible to verify a whole mixed-system

Why VHDL ? (3)

- ✘ Vendor dependency
- ✘ Design documentation exchange

A Standard HDL from the System Manufacturer's Point of View: V H D L

VHDL

Very High Speed Integrated Circuits (VHSIC)

Hardware

Description

Language

History

- 1981: an Extensive Public Review (DOD)
- 1983: a Request for Proposal

  (Intermetrics, IBM, and Texas Instruments)
- 1986: VHDL in the Public Domain
- 1987: a Standard Language VHDL'87 (IEEE-1076)
- 1992: a New Standard VHDL'92

Advantages & Drawbacks

Standard ⟷ Open language

✔ Vendor independent	✘ Complex tools
✔ User definable	✘ Slow tools
✔ Wide capabilities	

Abstraction Levels (1)

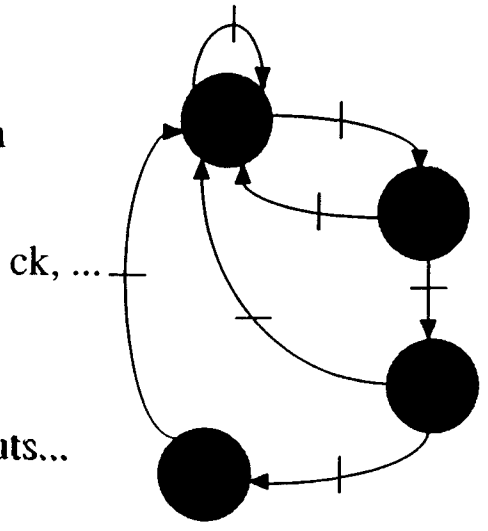
Algorithmic Level

- Very High Abstraction Level
- Functional Interpretation of a Discrete System
- No Implementation Details
- Sequential Program-Like Description
- Programmer's Point of View

Abstraction Levels (2)

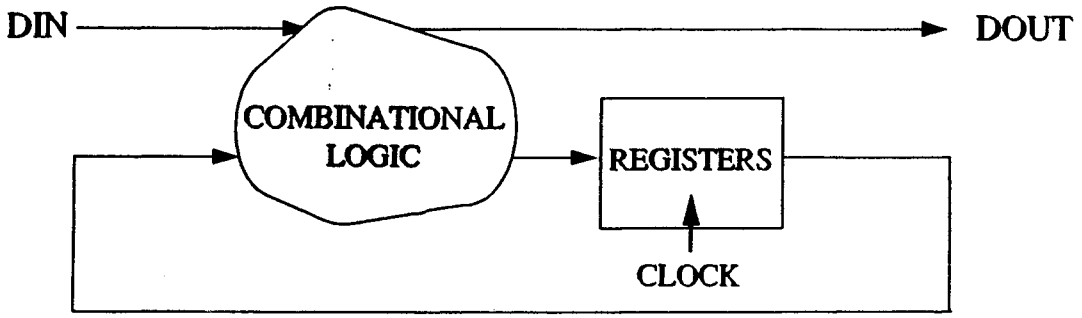
Finite State Machine Level

- Controller Part of a Digital Design
- Internal States
- State Changement Driven by:
  - Status Information
  - Clock and other External Inputs...



Abstraction Levels (3)

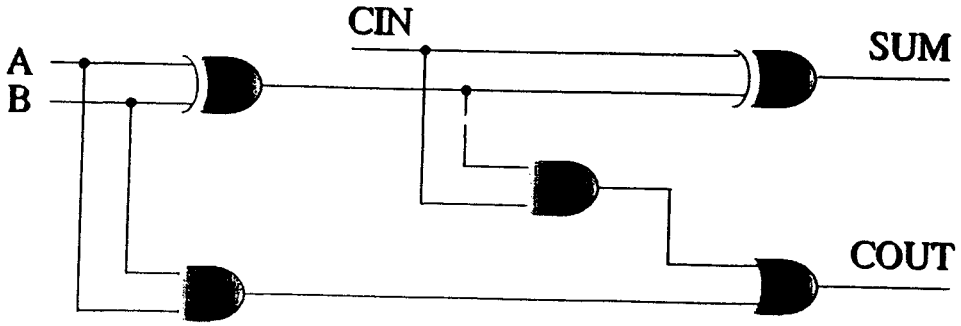
Register Transfer Level



- Registers Connected by Combinational Logic
- Very Close to the Hardware

Abstraction Levels (4)

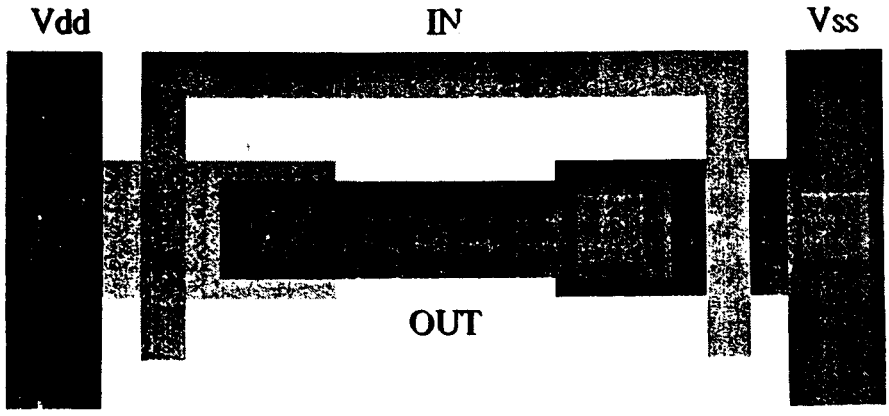
Gate Level



- A Gate Net-List Describing Instantiation of Models

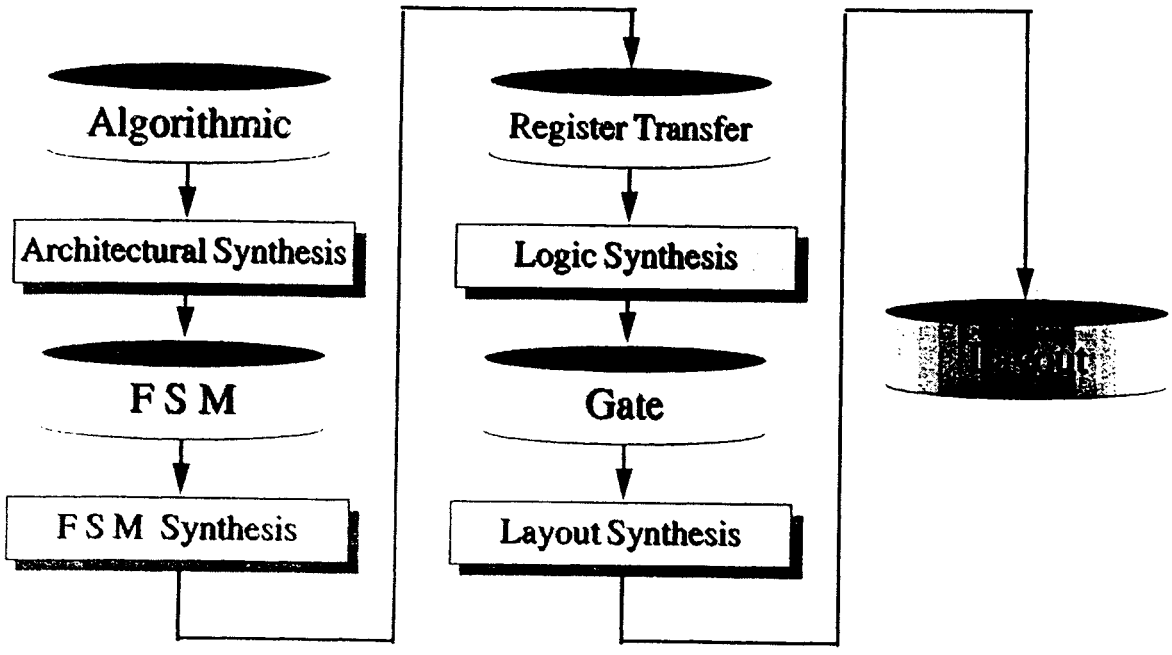
Abstraction Levels (5)

Layout Level

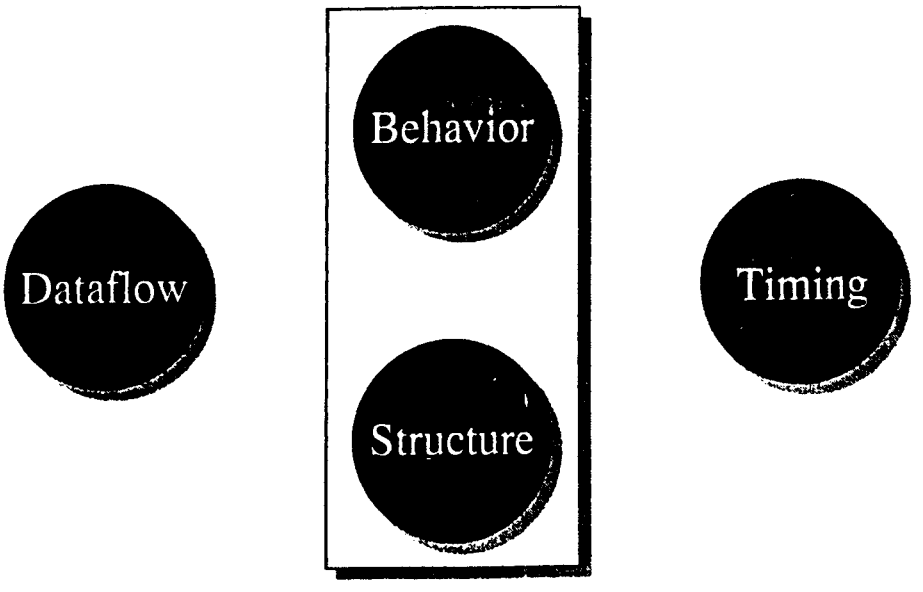


- A Set of Segments and Layers

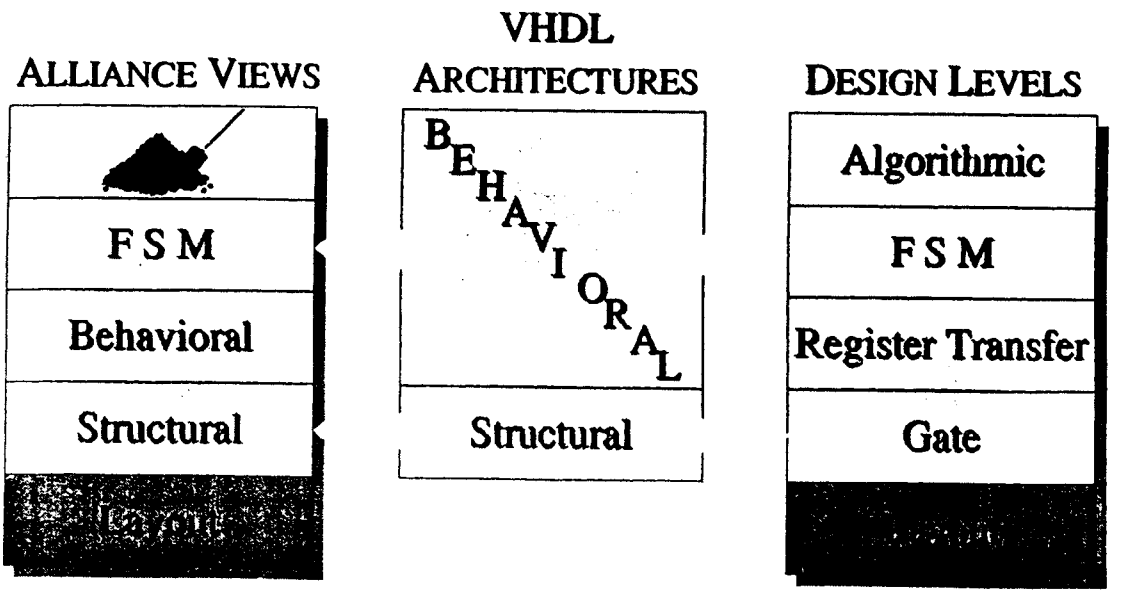
Synthesis Flow



VHDL Main Features



VHDL Architectures



A Dataflow Language (1)

CONTROLFLOW ≠ **DATAFLOW**

EX: C language assignment	EX: VHDL signal assignment
X = A & B;	**X <= A and B;**
X is computed out of A and B ONLY each time this assignment is executed	A PERMANENT link is created between A, B, and X X is computed out of A and B WHENEVER A or B change s

A Dataflow Language (2)

CONTROLFLOW ≠ **DATAFLOW**

EX: C language assignment	EX: VHDL assignment
X = A & B; **------** **X = C & D;**	**X <= A and B;** **------** **X <= C and D;**
✔ YES	✘ NO

Basic Structures

Basic Building Blocks

- **Entity**
- **Architecture**
- **Configuration**
- **Package**
- **Library**

Entity Declaration (1)

The External Aspect of a Design Unit

```
entity entity_name is
        [generic_declaration]
        [port_clause]
        {entity_declarative_item}
[begin
        entity_statement_part]
end [entity_name];
```

What is visible

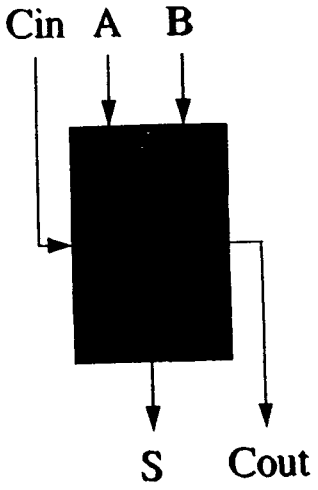
Entity Declaration (2)

Example

```
entity FULL_ADDER is
    port (A, B, Cin : in  BIT;
          S, Cout   : out BIT
    );
end FULL_ADDER;
```

MODE: in, out, inout...

DATA TYPE



Architectures (1)

The Internal Aspect of a Design Unit

```
architecture architecture_name of entity_name is
        {architecture_declarative_part}
begin
        {architecture_descriptive_part}
end [architecture_name];
```

How it works

- Collection of CONCURRENT Statements Executed in PARALLEL
- Concurrent Statements Communicate through SIGNALS

Architectures (2)

A Behavioral Style

```
entity FULL_ADDER is
    port (A, B, Cin : in  BIT;
          S, Cout   : out BIT);
end FULL_ADDER;
architecture DATAFLOW of FULL_ADDER is
    signal X : BIT;
begin
    X    <= A xor B;
    S    <= S xor Cin after 10 ns;
    Cout <= (A and B) or (X and Cin) after 5 ns;
end DATAFLOW;
```


Architectures (3)

A Structural Style

```
architecture STRUCTURE of FULL_ADDER is
    component HALF_ADDER
        port (I1, I2     : in  BIT;
              Carry, S   : out BIT);
    end component;

    component OR_GATE
        port (I1, I2     : in  BIT;
              O          : out BIT);
    end component;
    signal X1, X2, X3 : BIT;
```

DECLARATIVE PART

Architectures (4)

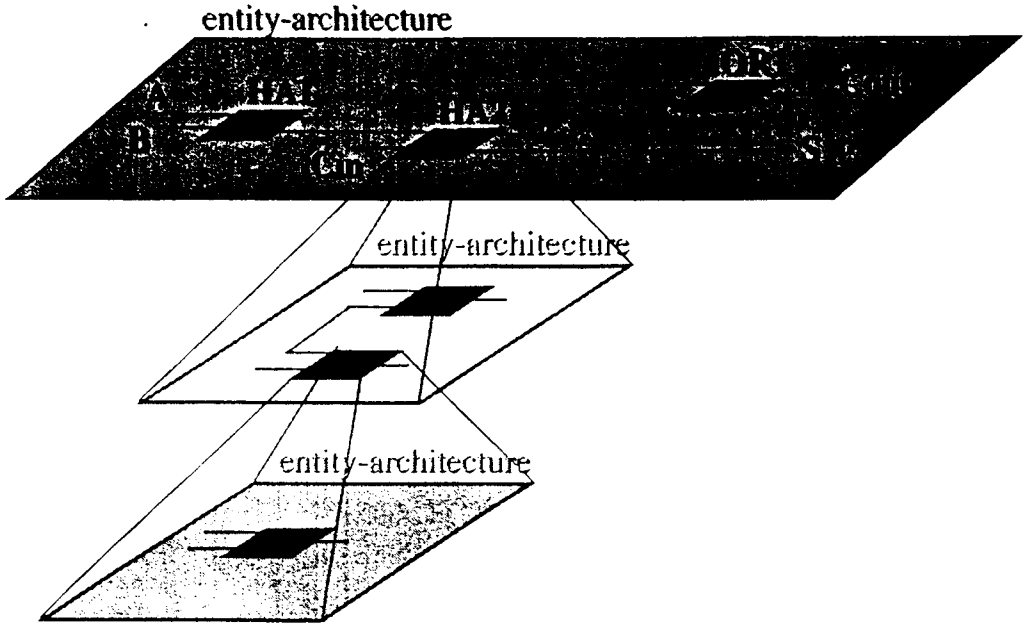
A Structural Style

```
begin
   HA1  : HALF_ADDER port map (
          I1 => A, I2 => B, Carry => X1, S => X2);
   HA2  : HALF_ADDER port map (
          I1 => X2, I2 => Cin, Carry => X3, S => S);
   OR1  : OR_GATE    port map (
          I1 => X1, I2 => X3, O => Cout);
end STRUCTURE;
```

DESCRIPTIVE PART

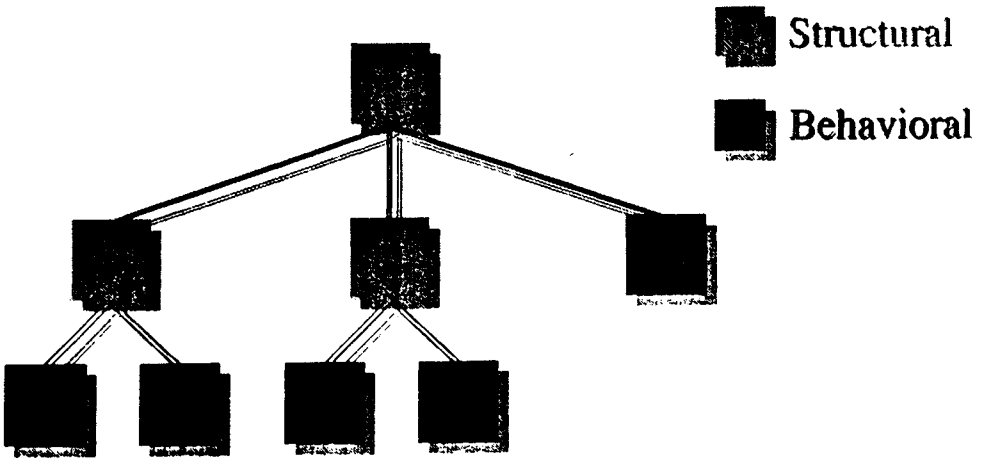
Architectures (5)

Structural Style to represent Hierarchy



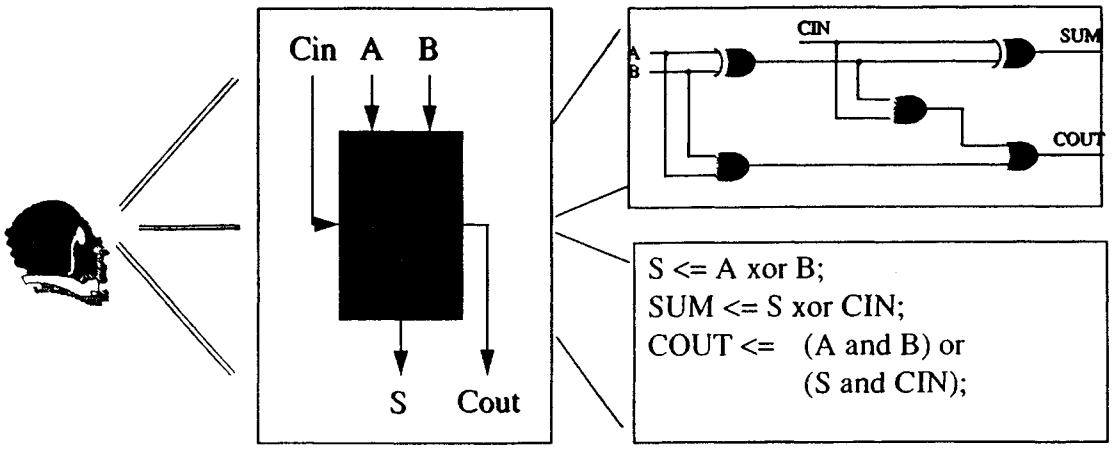
Architectures (6)

Structural & Behavioral in a Design Tree



Architectures (7)

entity/architecture: a One to Many Relationship



Configurations (1)

Specification Inside the Architecture Body

for instantiation_list: component_name **use** binding_indication;

use library_name.entity_name [(architecture_name)];

- **Binding a couple "entity/architecture" to each instance**

Configurations (2)

Declaration as a Separate Design Unit

configuration configuration_name **of** entity_name **is**
    for { **architecture** | **component** } binding_indication;
end [configuration_name];

- **Can be compiled separately and stored in a library**
- **It defines a configuration for a particular entity**

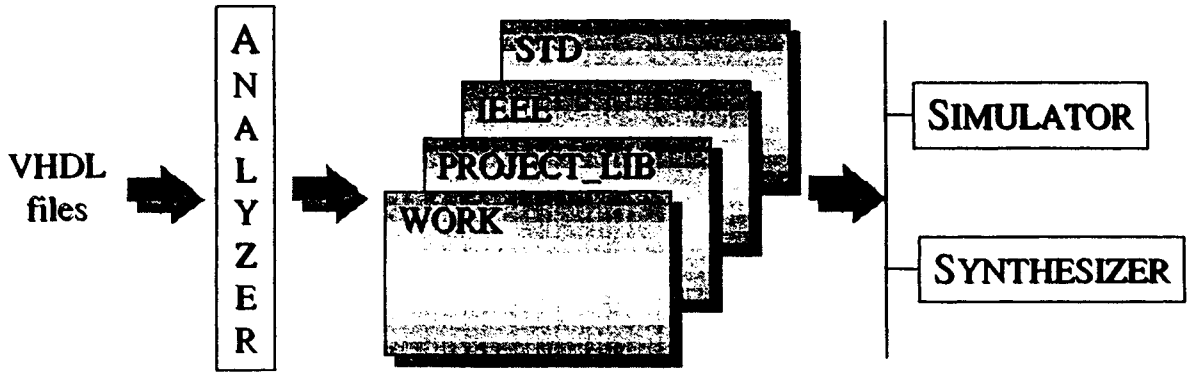
Packages

Global Design Unit

```
package package_name is
        {package_declarative_item}
end [package_name];
package body package_name is
        {package_body_declarative_item}
end [package_name];
```

- Same declarations visible by a number of design entities
- May contain subprograms, components, signals, ...

Design Libraries




```
library library_name ;
use library_name.package_name.all;
```

- May contain: packages, entities, architectures, configurations

Data Objects (1)

Three Classes

- **Constants**
  - **Initialized to a specific value and <u>never</u> modified**

    constant MSB : **INTEGER** := 5;

- **Variables**
  - **Used to hold temporary data**
  - **<u>Only</u> used within processes & subprograms**

    variable DELAY : INTEGER range 0 to 15 := 0;

Data Objects (2)

Three Classes

- **Signals**
  - **Used to communicate between processes**
  - **When declared in a package : <u>Global Signals</u>**
  - **Also declared within entities, blocks, architectures**
  - **Can be used but not defined in processes and subprograms**

    signal CLK : BIT;

Data Types (1)

Enumeration Types

- **The first identifier is the default value**

```
type COLOR is (RED, ORANGE, YELLOW);
type TERNARY is ( '1', '0', 'X' );
variable X : COLOR;
signal Y : TERNARY;
```


Data Types (2)

Integer Types

- **The range must be specified**
- **No logical operations on integer**

```
type MEMORY_SIZE is range 1 to 2048;
```

Data Types (3)

Predefined VHDL Data Types
IEEE 1076-1987 Standard Package

- **BOOLEAN : (false , true)**
- **BIT : ('0', '1')**
- **CHARACTER**
- **INTEGER : range -2 147 483 647 to +2 147 483 647**
- **NATURAL : Subtype of INTEGER (Non Negative)**
- **POSITIVE : Subtype of INTEGER (positive)**
- **BIT_VECTOR : array of BIT values**
- **STRING : array of CHARACTERS**
- **REAL : range -1.0E+38 to +1.0E+38**
- **TIME : Physical type used for simulation**

Data Types (4)

Array Types

- **Constrained Array**

```
type VEC_64 is array (0 to 63) of INTEGER;
variable S : VEC_64;
variable S1 : INTEGER;
S1 := S (1);
```

- **Unconstrained Array**

```
type BIT_VECTOR is array (POSITIVE range <>) of BIT;
signal S : BIT_VECTOR (4 downto 0);
```

- **Multiple Dimentional Arrays**

```
type TWO_D is array (0 to 7, 0 to 3) of INTEGER;
```

Data Types (5)

Record Types

```
type DATE is
        record
                YEAR : INTEGER range 1900 to 1999 ;
                MONTH : INTEGER range 1 to 12 ;
                DAY : INTEGER range 1 to 31 ;
        end record ;
signal S : DATE;
variable Y : INTEGER range 1900 to 1999 ;
Y := S.YEAR ;
```

Data Types (6)

STD_LOGIC Data Types
IEEE 1164-1993 Standard Logic Package

```
type STD_ULOGIC is (
        'U'     --      Uninitialized
        'X'     --      Forcing Unknown
        '0'     --      Forcing Low
        '1'     --      Forcing High
        'Z'     --      High Impedance
        'W'     --      Weak Unknown
        'L'     --      Weak Low
        'H'     --      Weak High
        '-'     --      Don't Care
);
```

Unresolved Data Type

Used in Synthesis

Data Types (7)

STD_LOGIC Data Types
IEEE 1164-1993 Standard Logic Package

- STD_LOGIC : Resolved (Resolution Function provided)
- STD_LOGIC_VECTOR
- STD_ULOGIC_VECTOR

Data Types (7)

Also,

- FILE : Useful for RAM Values or Stimuli Files
- ACCESS : Like "pointers" in High Level Languages
- TEXT : FILE of STRING (TEXTIO package)
- LINE : access STRING (TEXTIO package)

Subtypes

Subsets of Other Types

- **To Insure Valid Assignments**
- **Inherit All Operators and Subprograms from the Parent Type**

  subtype DIGIT is INTEGER range 0 to 9;

Operators

Six Classes

LOGIC OPERATOR	and , or , nand , nor , xor
RELATIONAL OPERATOR	= , /= , < , <= , > , >=
ADDING OPERATOR	+ , - , &
SIGN	+ , -
MULTIPLYING OPERATOR	* , / , mod , rem
MISCELLANEOUS OPERATOR	** , abs , not

PRECEDENCE ORDER

Operands (1)

- **Literals : 'x' , "1100" , 752 , B"11001" , O"277" , X"4C"**
  - **numeric, character, enumeration, or string**
- **Identifiers :**
  - **starts with (a-z) followed by letters, '_', or digits**
  - **Not case-sensitive**
  - **Some are reserved words**
- **Indexed Names : S (3) , DATA (ADDR)**

Operands (2)

- **Slice Names : variable ORG : BIT_VECTOR (7 downto 0)**
  - **Sequence of elements of an array object**
- **Aliases : alias MSB : BIT is ORG (7)**
  - **New name for a part of a range of an array**
- **Aggregates**
- **Qualified Expressions**
- **Function Calls**
- **Type Conversions**

Operands (3)

Attributes Names
A Data Attached to VHDL Objects

- **S'LEFT : Index of the leftmost element of the data type**
- **S'RIGHT : Index of the rightmost element of the data type**
- **S'HIGH : Index of the highest element of the data type**
- **S'LOW : Index of the lowest element of the data type**
- **S'RANGE : Index range of the data type**
- **S'REVERSE_RANGE : Reverse index range**
- **S'LENGTH : Number of elements of an array**


Operands (4)

Attributes Names
A Data Attached to VHDL Signals

- **S'EVENT : A change value at the current simulation time**
- **S'STABLE : No change value at the current simulation time**

  if (CK = 0 and not CK'STABLE)
- **.....**


Concurrent Statement

Natural Concept for Describing Hardware

- Concurrent Signal Assignment
- Conditional Signal Assignment
- Selected Signal Assignment
- Block Statement
- Concurrent Assertion Statement
- Process Statement

Concurrent Signal Assignment

Represent an Equivalent Process Statement

target <= expression [after time_expression] ;

- Signals are associated with Time
- With "after", the assignment is scheduled to a future simulation time
- Without "after", the assignment is scheduled at a Delta Time after the current simulation time

Conditional Signal Assignment

More than One Expression

target <= { expression [after time_exp] when condition else }
expression [after time_exp] ;

- Condition / expression except for the last expression
- One and only one of the expressions is used at a time

Selected Signal Assignment

Only One Target

with choice_expression select
target <= { expression [after time_exp] when choices , }
expression [after time_exp] when choices ;

- "when others" is used when all the cases were not treated

Block Statement (1)

A Set of Concurrent Statements

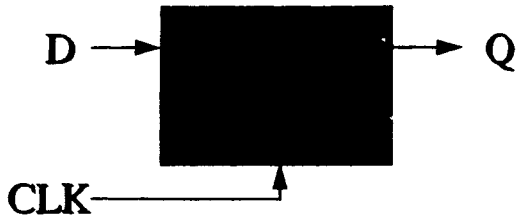
```
label : block
        { block_declarative_part }
begin
        { concurrent_statements }
end block [ label ] ;
```

- Used to organize a set of concurrent statements hierarchically

Block Statement (2)

In Synchronous Descriptions

```
latch : block ( CLK = '1' )
begin
    Q <= GUARDED D ;
end block latch ;
```



Assertion Statement

Only One Target

```
assert condition
        [ report error_message ]
        [ severity severity_level ] ;
```

- If the condition is false, it reports a diagnostic message
- Useful for detecting condition violation during simulation
- Not used in synthesis

Process Statement (1)

A Set of Sequential Statements

```
[ label : ] process [ ( sensitivity_list ) ]
        { process_declarative_part }
begin
        { sequential_statements }
end process [ label ] ;
```

- All processes in a design executes CONCURRENTLY
- At a given time, ONLY ONE sequential statement executed within each process
- Communicates with the rest of a design through signals

Process Statement (2)

A Pseudo Infinite Loop

```
process
begin
        sequential_statement_1 ;
        sequential_statement_2 ;
        ----------
        sequential_statement_n ;
end process;
```

- A Synchronization Mecanism is Needed

Process Statement (3)

Synchronization Mecanism

```
wait
        [ on signal_name { signal_name } ]
        [ until boolean_expression ]
        [ for time_expression ] ;
```

- Objects being waited upon should be **SIGNALS**

Process Statement (4)

The Sensitivity List

```
process [ ( sensitivity_list ) ]
begin
        { sequential_statements }
end process ;
```

- Equivalent to a "wait" statement as the last statement

  wait on sensitivity_list ;

Sequential Statement

Insight Into Statements within Processes

- Variable Assignment
- Signal Assignment
- If
- Case
- Null
- Assertion
- Loop
- Next
- Exit
- Wait
- Procedure Calls
- Return

Variable Assignment Statement

Immediate Assignment

target_variable := expression ;

- Always executed in **ZERO SIMULATION TIME**
- Used as temporary storages
- Can not be seen by other concurrent statements

Signal Assignment Statement (1)

Defines a DRIVER of the Signal

target_signal <= [transport] expression [after time_expression] ;

- Within a process, **ONLY ONE** driver for each signal
- When assigned in multiple processes, it has **MULTIPLE DRIVERS**. A **RESOLUTION FUNCTION** should be defined

CONCLUSION (1)

VHDL IS AN OPEN LANGUAGE WITH MANY FEATURES.

WITH VHDL, ANY DISCRETE SYSTEM CAN BE MODELED.

CONCLUSION (2)

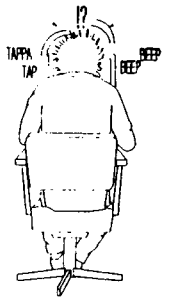
EACH USER HAS ITS OWN NEEDS DEPENDING ON:

- HIS BACKGROUND.
- HIS ENVIRONMENT.

WE DEFINED A SUBSET OF VHDL.

CONCLUSION (3)

WHY ?



COMPLEX LANGUAGE ⇒ DEVELOPING A COMPILER IS HARD AND TIME CONSUMING.

CONCLUSION (4)

WHY ?

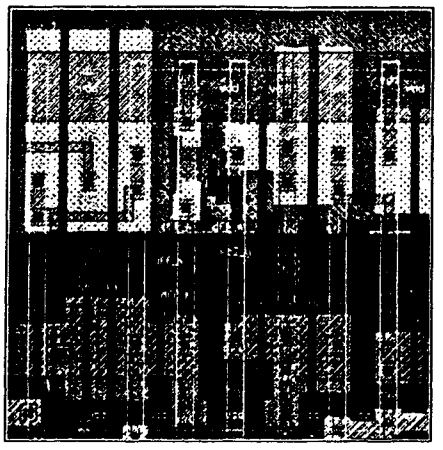
EDUCATIONAL NEEDS:

- UNDERSTANDING TIME.
- UNIVOCAL (ONE WAY FOR DESCRIBING A REGISTER).

CONCLUSION (5)

WHY ?

OUR ENVIRONMENT: VLSI.



OUTLINE

I - INTRODUCTION.

II - DESIGN METHODOLOGY: AN OVERVIEW.

III - ABSTRACTION LEVELS IN ALLIANCE.

IV - VHDL: AN OVERVIEW.

V - VHDL: THE ALLIANCE SUBSET.

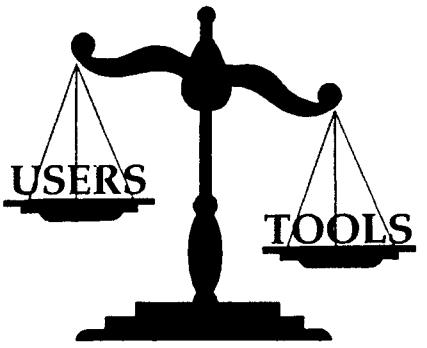
VI - ALLIANCE: A COMPLETE DESIGN SYSTEM.

WHY AND HOW ?

WHY ?

- DEVELOPMENT TIME.
- EDUCATION CONSTRAINTS.
- THE CURRENT ENVIRONMENT.

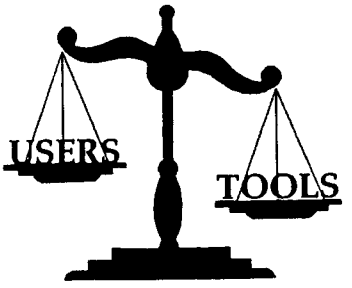
CRITERIONS FOR THE SUBSET DEFINITION.



TOOLS REQUIREMENTS (1)

WHICH TOOLS USE VHDL ?

- SYNTHESIS.
- FORMAL PROOVER.
- PLACER & ROUTER.
- SIMULATOR.
- FUNCTIONAL ABSTRACTOR.



TOOLS REQUIREMENTS (2)

SYNTHESIS TOOLS.

- A REGISTER MUST BE IDENTIFIED IN A SYNTACTICAL WAY.
- A BUS MUST BE IDENTIFIED IN A SYNTACTICAL WAY.
- SIGNALS MUST HAVE THE **BIT** TYPE ('0' , '1').
- NO TIMING.

FORMAL PROOVER.

- A REGISTER MUST BE IDENTIFIED IN A SYNTACTICAL WAY.
- A BUS MUST BE IDENTIFIED IN A SYNTACTICAL WAY.

TOOLS REQUIREMENTS (3)

FUNCTIONAL ABSTRACTOR.

- THE VHDL SUBSET MUST BE AS CLOSE AS POSSIBLE TO THE HARDWARE.

PLACER & ROUTER.

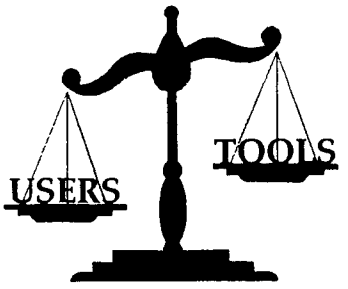
- NO MIXING BETWEEN STRUCTURAL AND BEHAVIORAL VIEWS.

SIMULATOR.

- NO ABSTRACT TYPES.
- NO TIMING.

USERS REQUIREMENTS

LOOKING FOR THE LARGEST SUBSET.



THE GOOD VHDL SUBSET:

- ✓ LETS THE USER DESCRIBE HIS CIRCUIT EASILY.
- ✓ DO NOT DETERIORATE THE TOOL WITH A COMPLEX LANGUAGE.

THE EXTERNAL ASPECT

IN VHDL A CIRCUIT (DESIGN UNIT) HAS TWO ASPECTS:

1. THE EXTERNAL ASPECT: (EXTERNAL VISIBILITY)

ALLIANCE

What is visible

- ✓ NAME
- ✓ INTERFACE (PORT)

- × COLOR
- × TEMPERATURE
- × ----------

THE INTERNAL ASPECT (1)

IN VHDL A CIRCUIT (DESIGN UNIT) HAS TWO ASPECTS:

2. THE INTERNAL ASPECT: (FUNCTIONALITY)

ALLIANCE	✓STRUCTURAL ✓DATA FLOW

how it works

THE INTERNAL ASPECT (2)

IN THE STRUCTURAL INTERNAL ASPECT, WE DESCRIBE THE CIRCUIT AS A NETWORK OF SMALLER CIRCUITS.

THE FOLLOWING OBJECTS ARE USED:

- SIGNAL.
- COMPONENT (MODEL).
- INSTANCE.

EXTERNAL ASPECT: EXAMPLE (1)

```
ENTITY PARITY IS
 PORT (
        A : IN BIT ;
        B : IN BIT;
        C : IN BIT ;
        D : IN BIT;
        P : OUT BIT
       )
END PARITY;
```

Circuit Name

Port Name

Input/output mode

Type

EXTERNAL ASPECT: EXAMPLE (2)

```
ENTITY ADDER_32 IS
 PORT (
        A : IN BIT_VECTOR (31 DOWNTO 0);
        B : IN BIT_VECTOR (31 DOWNTO 0);
        CIN : IN BIT;
        SUM : OUT BIT_VECTOR (31 DOWNTO 0);
        COUT : OUT BIT
       )
END;
```

INTERNAL STRUCTURAL EXAMPLE (1)

```
ARCHITECTURE PSTRUCT OF PARITY IS
    COMPONENT XOR_Y
        PORT (
                I0 : IN BIT ;
                I1 : IN BIT;
                T : OUT BIT
              );
    END COMPONENT;
SIGNAL PARITY_AB : BIT;
SIGNAL PARITY_CD : BIT;
```

DECLARATIVE PART

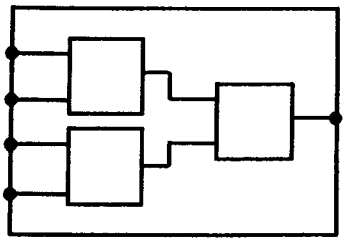
INTERNAL STRUCTURAL EXAMPLE (2)

```
BEGIN
    INSTANCE_AB : XOR_Y
        PORT MAP (
                    I0 => A,
                    I1 => B,
                    T => PARITY_AB
                  );

    INSTANCE_CD : XOR_Y
        PORT MAP (
                    I0 => C,
                    I1 => D,
                    T => PARITY_CD
                  );
```

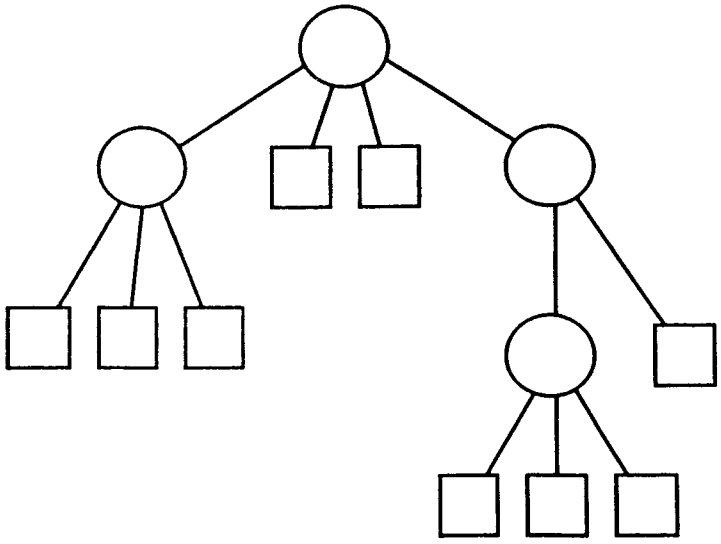
DESCRIPTION PART

INTERNAL STRUCTURAL EXAMPLE (3)

```
INSTANCE_ABCD : XOR_Y
    PORT MAP (
                I0 => PARITY_AB,
                I1 => PARITY_CD,
                T => P
               );
END;
```



STRUCTURAL & HIERARCHICAL



INTERNAL BEHAVIORAL ASPECT (1)

DESCRIBING EQUATIONS BETWEEN INPUTS AND OUTPUTS.

- BOOLEAN FUNCTIONS:
  - ♦ AND
  - ♦ OR
  - ♦ XOR
  - ♦ NAND
  - ♦ NOR
  - ♦ NOT

ALWAYS USE BRACKETS.

INTERNAL BEHAVIORAL ASPECT (2)

DESCRIBING EQUATIONS BETWEEN INPUTS AND OUTPUTS.

- ASSERT (CONDITION)
  REPORT "MESSAGE"
  SEVERITY LEVEL;

VERY USEFUL IN LARGE-SCALE DESIGN.

- ♦ ALLOWS ENCODING SPECIFIC CONSTRAINTS AND ERROR CONDITIONS
- ♦ PROVIDE USEFUL MESSAGES.
- ♦ STOP THE SIMULATION WHEN CONSTRAINTS ARE NOT MET.

INTERNAL BEHAVIORAL ASPECT (3)

- THREE KINDS OF ASSIGNMENTS:

  SIMPLE ASSIGNMENT:

  S <= A AND B;

  CONDITIONAL ASSIGNMENT:

  S <= A AND B WHEN (C = '0') ELSE
  D OR E;

Always

INTERNAL BEHAVIORAL ASPECT (4)

SELECTIVE ASSIGNMENT:

WITH ADDRESS(3 DOWNTO 0) SELECT
OUT <= "000100" WHEN "0000",
"000101" WHEN "0001",

"000000" WHEN OTHERS;

INTERNAL BEHAVIORAL ASPECT (5)

- REGISTERS:

SIGNAL MYREGISTER : REG_BIT REGISTER;

```
STORE : BLOCK ( CK = '0' AND NOT CK'STABLE )
BEGIN
MYREGISTER <= GUARDED I0;
END BLOCK STORE;
```

INTERNAL BEHAVIORAL ASPECT (6)

- BUS:

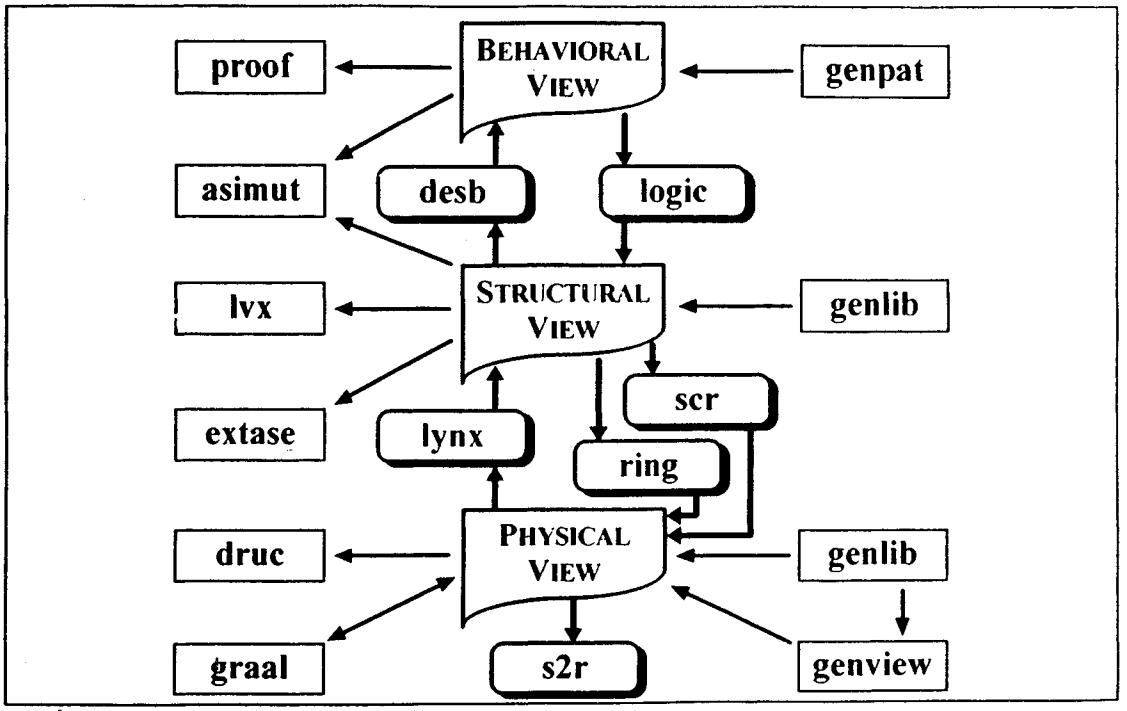
SIGNAL MY_BUS1 : MUX_BIT BUS;
ONLY ONE DRIVER ACTIVE AT THE SAME TIME.

SIGNAL MY_BUS2 : WOR_BIT BUS;
MANY DRIVERS DRIVE THE SAME VALUE.

INTERNAL BEHAVIORAL EXAMPLE

```
ARCHITECTURE DATA_FLOW OF PARITY IS
   SIGNAL PARITY_AB : BIT;
   SIGNAL PARITY_CD : BIT;
BEGIN
     PARITY_AB <= A XOR B;
     PARITY_CD <= C XOR D;
     P <= PARITY_AB XOR PARITY_CD;
End;
```

OUTLINE



SYNTHESIS LEVELS

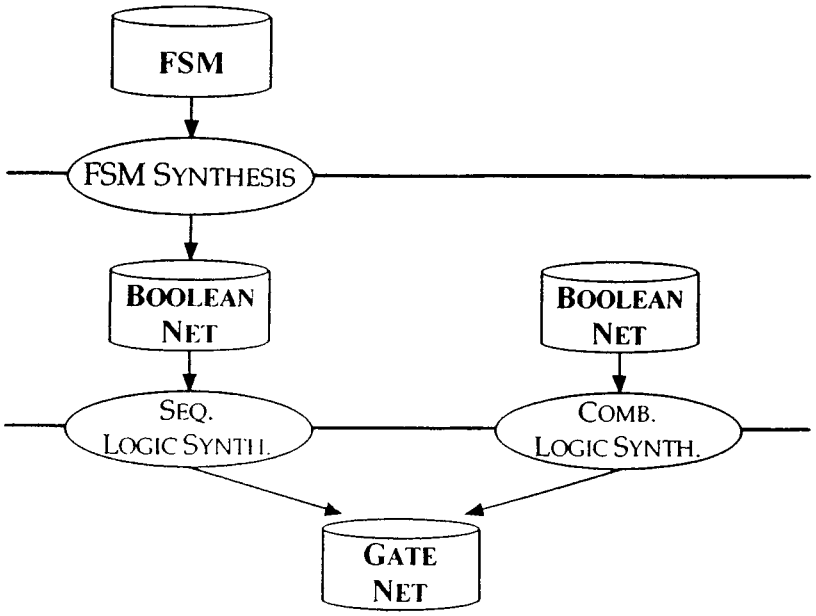
- ARCHITECTURAL
- FINITE STATE MACHINE
- LOGIC
- LAYOUT

SYNTHESIS AREA

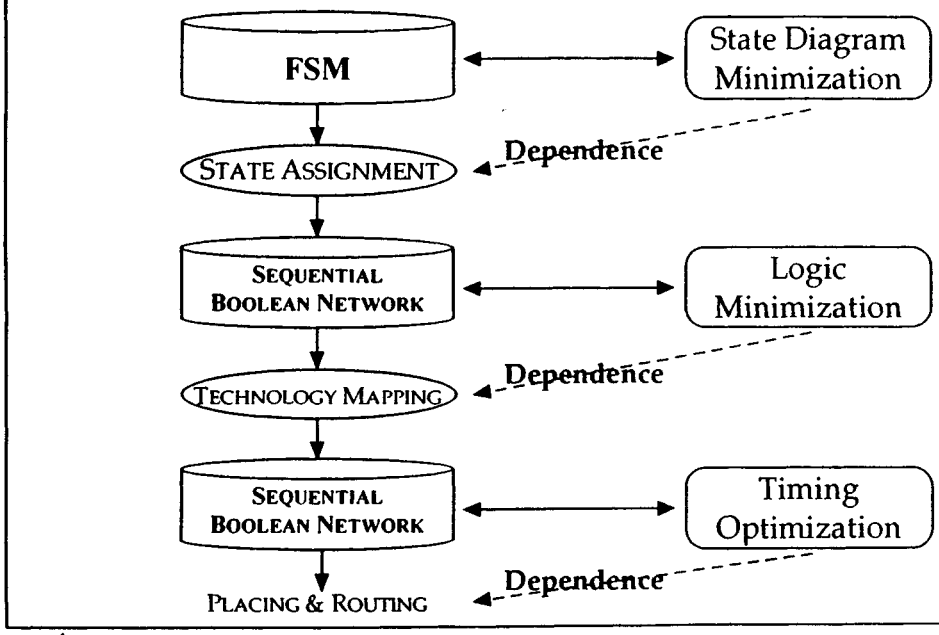
- CONTROL LOGIC
  EVERY CIRCUIT THAT MAY BE DESCRIBED AS FSM
  (NB STATES < 1000).

- RANDOM LOGIC
  EVERY CIRCUIT THAT MAY NOT BE DESCRIBED WITH REGULAR LOGIC.

SYNTHESIS ARCHITECTURE

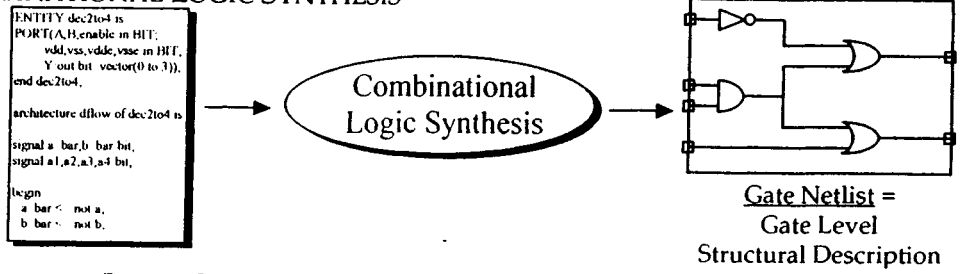


OPTIMIZATION AND SYNTHESIS FLOW



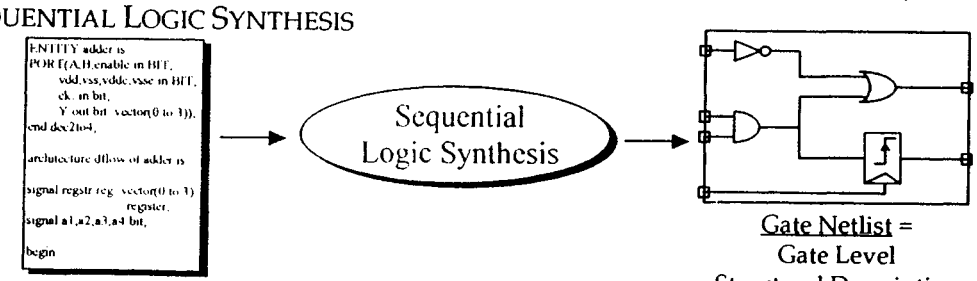
LOGIC SYNTHESIS

- COMBINATIONAL LOGIC SYNTHESIS



Gate Netlist =
Gate Level
Structural Description

- SEQUENTIAL LOGIC SYNTHESIS



Gate Netlist =
Gate Level
Structural Description

OPTIMIZATION

✓ IMPROVE DESCRIPTION AT EQUIVALENT LEVEL.

$$\begin{cases} X = A + \overline{A}.C.D \\ Y = C.D \end{cases} \Rightarrow \begin{cases} X = A + Y \\ Y = C.D \end{cases}$$

REPRESENTATION (1)

LOGICAL EQUATIONS

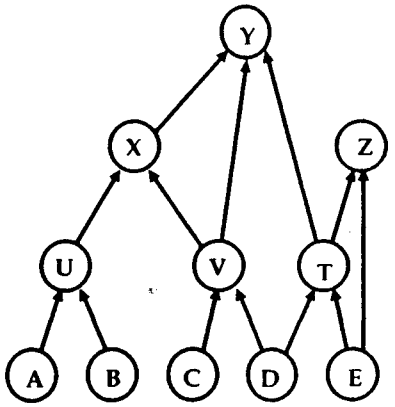
A DIRECTED ACYCLIC GRAPH INCLUDING THREE KINDS OF NODES: INPUT, INTERMEDIARY, OUTPUT.

EACH INTERMEDIARY OR OUTPUT NODE IS ASSOCIATED TO A LOGICAL EXPRESSION.

EACH NODE IS ASSOCIATED TO A VARIABLE NAME.

REPRESENTATION (2)

BOOLEAN NETWORK



BDD (BINARY DECISION DIAGRAM) (1)

BASED ON THE SHANNON THEOREM:

$$F(X_1, X_2, \ldots, X_n) =$$

$$\overline{X_1}.F(0, X_2, \ldots, X_n) + X_1.F(1, X_2, \ldots, X_n)$$

✓ CANONICAL REPRESENTATION OF A BOOLEAN EQUATION.

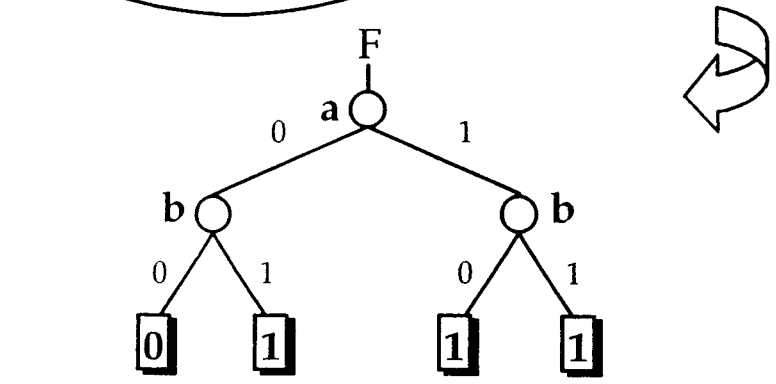
BDD (BINARY DECISION DIAGRAM) (2)

Ex: $F(a,b) = a + b$

$$F = \bar{a}.F(0,b) + a.F(1,b)$$
$$= \bar{a}.(0+b) + a.(1+b)$$
$$= \bar{a}.(b) + a.(1)$$
$$= \bar{a}.(\bar{b}.(0) + b.(1)) + a.(\bar{b}.(1) + b.(1))$$

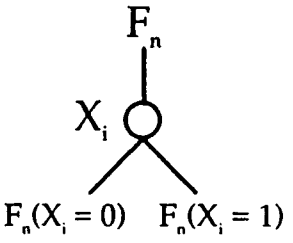
BDD (BINARY DECISION DIAGRAM) (3)

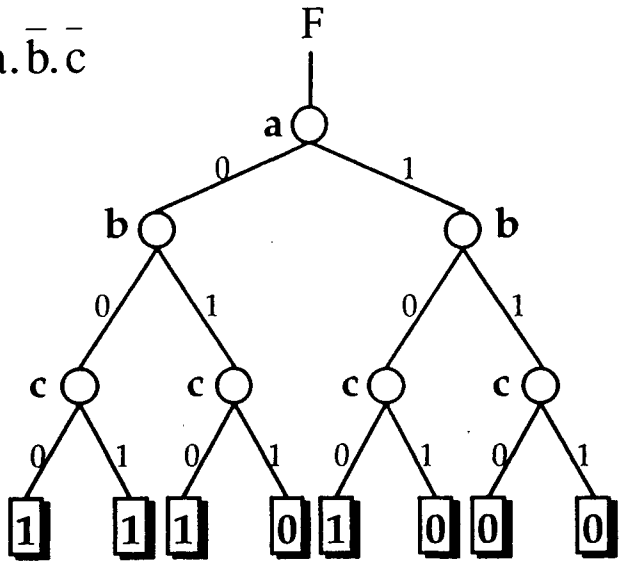
SO... $F = \bar{a}.(\bar{b}.(0) + b.(1)) + a.(\bar{b}.(1) + b.(1))$



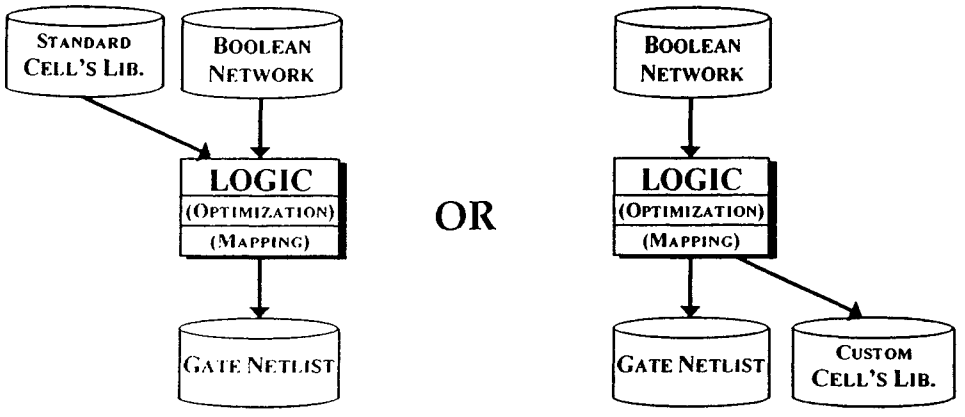
BDD (BINARY DECISION DIAGRAM) (4)

EX: $F = \bar{a}.\bar{b} + \bar{a}.b.\bar{c} + a.\bar{b}.\bar{c}$

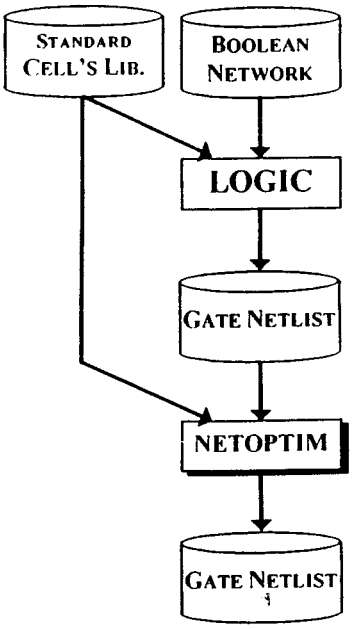




LOGIC: LOGIC SYNTHESIZER



NETOPTIM: TIMING OPTIMIZER



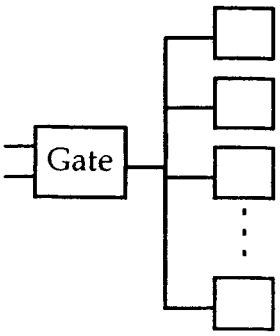
✓ TIMING OPTIMIZATION WITH LIMITED SURFACE LOSS.

TWO OPTIMIZATION OPTIONS:

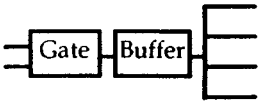
- FANOUT OPTIMIZATION (LOCAL VIEW).
- DELAY OPTIMIZATION WITH TIMING ANALYSIS (GLOBAL VIEW).

NETOPTIM: FANOUT OPTIMIZATION (LOCAL)

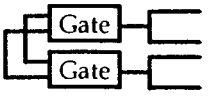
THREE METHODS:



- REPOWER.
- BUFFER INSERTION.



- GATE DUPLICATION.



✓ FAST TECHNIQUES.

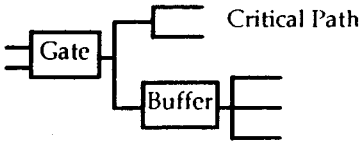
✗ NOT SHARP ENOUGH.

NETOPTIM: DELAY OPTIMIZATION (GLOBAL)

THE TIMING ANALYSIS COMPUTES THE CRITICAL PATH OF THE CIRCUIT.

TWO METHODS TO OPTIMIZE THE CRITICAL PATH:

- REPOWER.
- BUFFER INSERTION.



✓ GOOD RESULTS.

✗ FALSE PATH PROBLEM

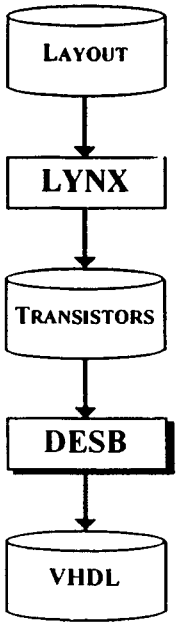
DESB: FUNCTIONAL ABSTRACTOR (1)

✓ GENERATES BEHAVIORAL DATA FLOW VHDL.

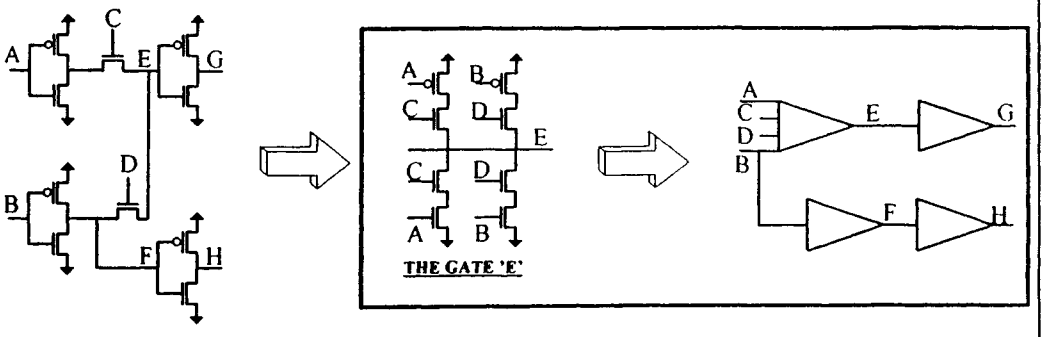
✓ PROVIDES FUNCTIONAL VERIFICATIONS.

✓ DOES NOT USE ANY CELL LIBRARY.

✓ ACCEPTS STANDARD TRANSISTOR NETLIST FORMAT (VTI, SPICE).



DESB: FUNCTIONAL ABSTRACTOR (2)




```
E <= (NOT A AND C) OR (NOT B AND D);
H <= NOT F;
G <= NOT E;
F <= NOT B;
```

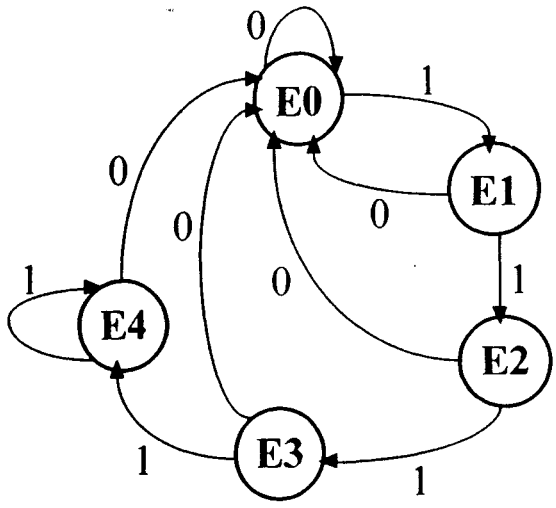

FSM (FINITE STATE MACHINE) (1)

- MODELS SEQUENTIAL CIRCUITS.
- TWO KINDS OF FSM.
- GRAPH REPRESENTATION.
- DEFINITION:

$$\text{STATE}(T+1) <= F(I_1,\ldots,I_n,\text{STATE}(T))$$
$$\text{OUTPUT}_i <= F(I_1,\ldots,I_n,\text{STATE}(T))$$

FSM (FINITE STATE MACHINE) (2)

EXAMPLE: FOUR CONSECUTIVE ONE'S COUNTER



FSM: THE DESCRIPTION LANGUAGE

- STANDARD.
- VHDL SUBSET.
- THE STATES ARE ENUMERATED TYPE.
- TWO SPECIAL SIGNALS.
- TWO PROCESSES.

```
--
Entity counter is port (ck, I, reset: in bit; O: out bit);
End counter;

Architecture automate of counter is
type STATE_TYPE is (E0, E1, E2, E3, E4);
signal CURRENT_STATE, NEXT_STATE: STATE_TYPE;
-- pragma CUR_STATE CURRENT_STATE;
-- pragma NEX_STATE NEXT_STATE;
-- pragma CLOCK ck;

begin
    Process(CURRENT_STATE, I, reset)
    begin
        if (reset = '1') then
            NEXT_STATE <= E0;
            O <= '0';
        else
```

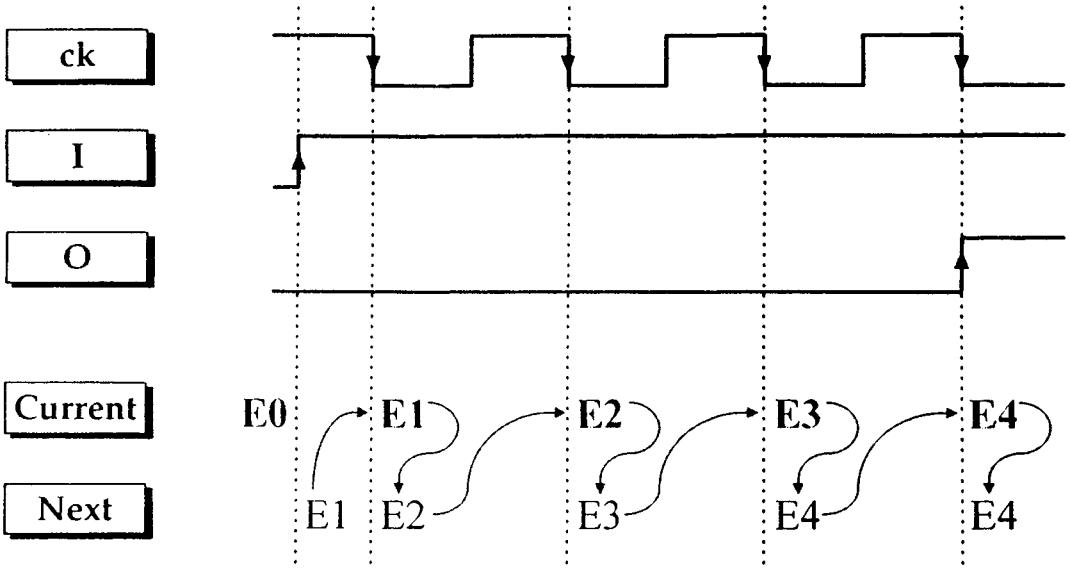

```
        case CURRENT_STATE is
            WHEN E0 =>
                if (I='1') then
                    NEXT_STATE <= E1;
                else
                    NEXT_STATE <= E0;
                end if;
                O<= '0';

            WHEN E1 =>
                if (I='1') then
                    NEXT_STATE <= E2;
                else
                    NEXT_STATE <= E0;
                end if;
                O<= '0';
```

```
            WHEN E2 =>
                if (I='1') then
                    NEXT_STATE <= E3;
                else
                    NEXT_STATE <= E0;
                end if;
                O<= '0';

            WHEN E3 =>
                if (I='1') then
                    NEXT_STATE <= E4;
                else
                    NEXT_STATE <= E0;
                end if;
                O<= '0';
```

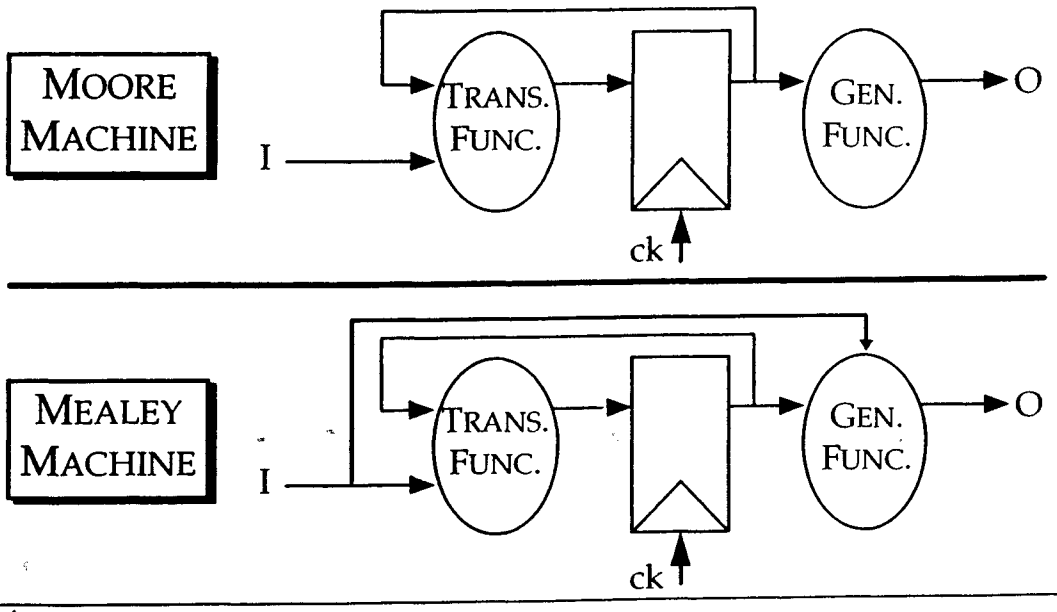

```
            WHEN E4 =>
                if (I='1') then
                    NEXT_STATE <= E4;
                else
                    NEXT_STATE <= E0;
                end if;
                O<= '1';

            WHEN others =>
                assert('1')
                report "Illegal State";

        end case;
    end if;
end process;
```

```
    Process(ck)
    begin
        if (ck = '0' and not ck'stable) then
            CURRENT_STATE <= NEXT_STATE;
        end if;
    end process;
end counter;
```


FSM: THE FOUR CONSECUTIVE ONE'S COUNTER

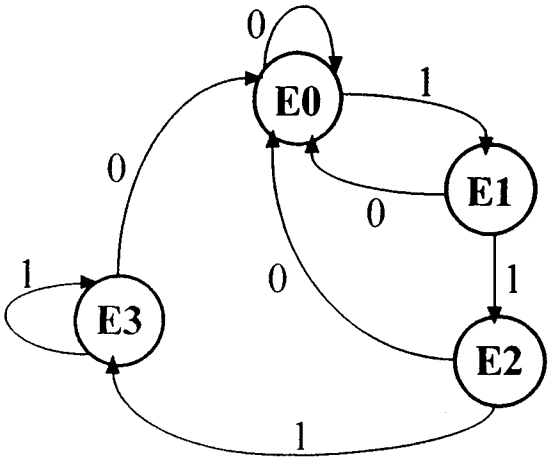


FSM: DIFFERENT MACHINE'S TYPE



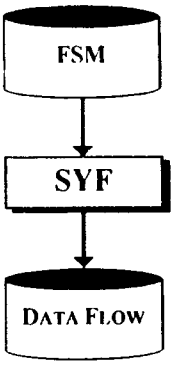
ACC

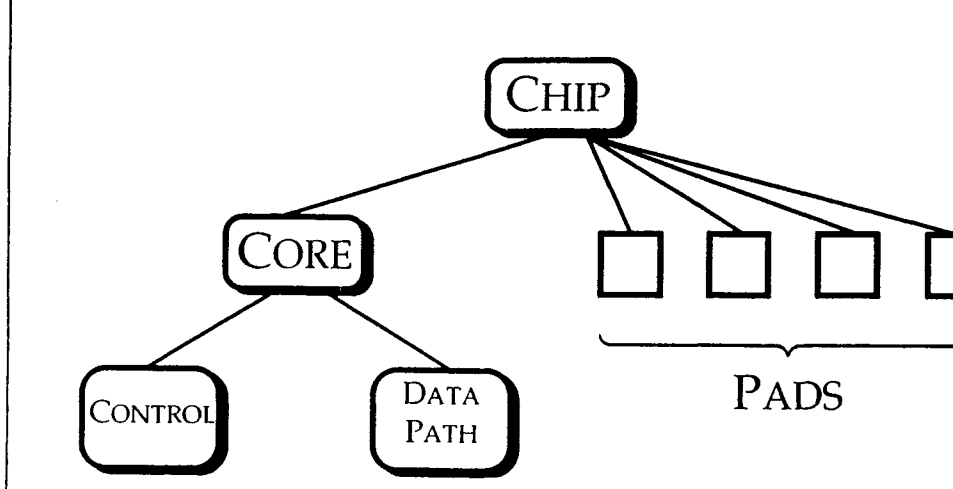
FSM: THE COUNTER WITH THE MEALEY MACHINE

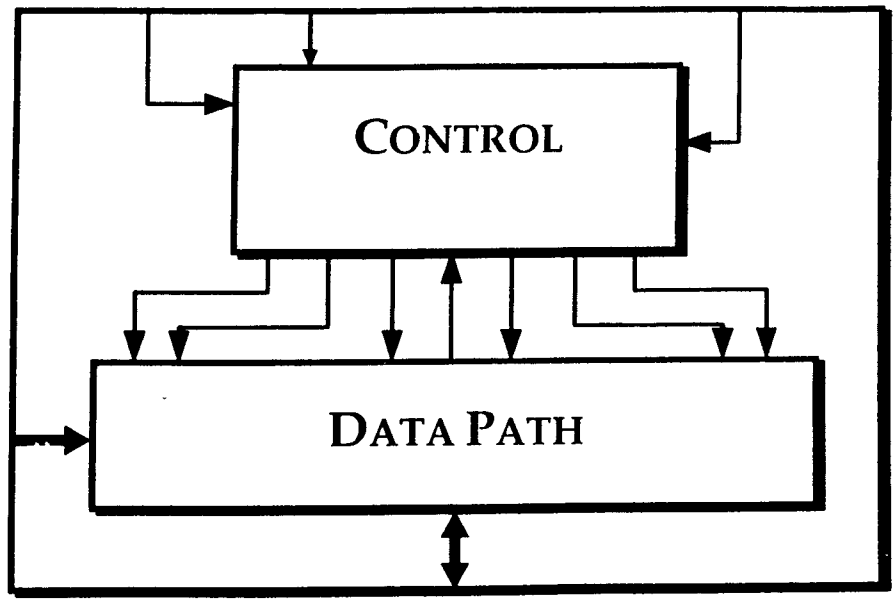


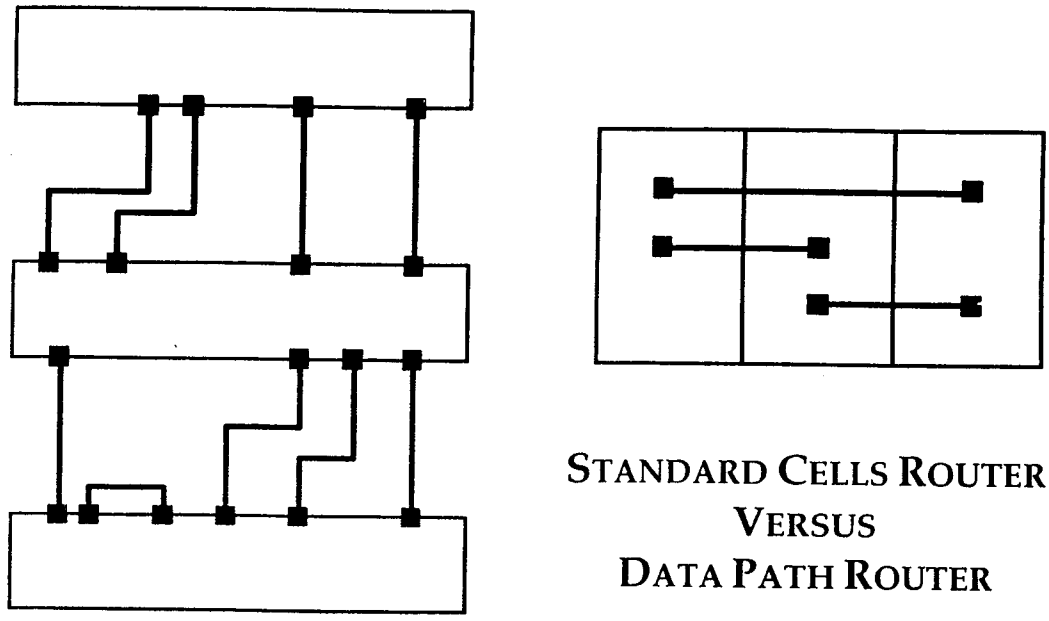
SYF: AN FSM SYNTHESIZER

- VERIFICATION.
- ENCODING.
- OPTIMIZATION.
- DRIVING DATA FLOW DESCRIPTION.



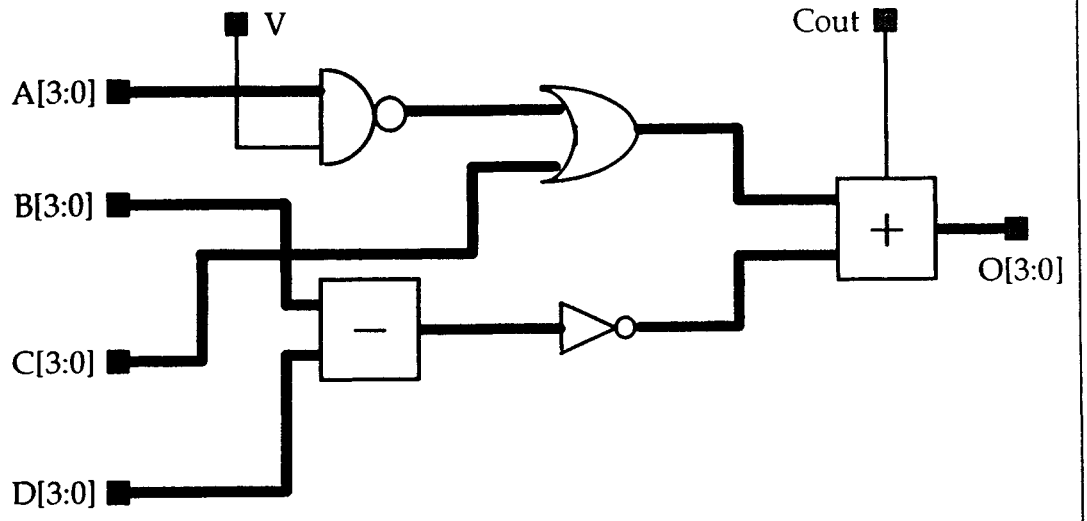




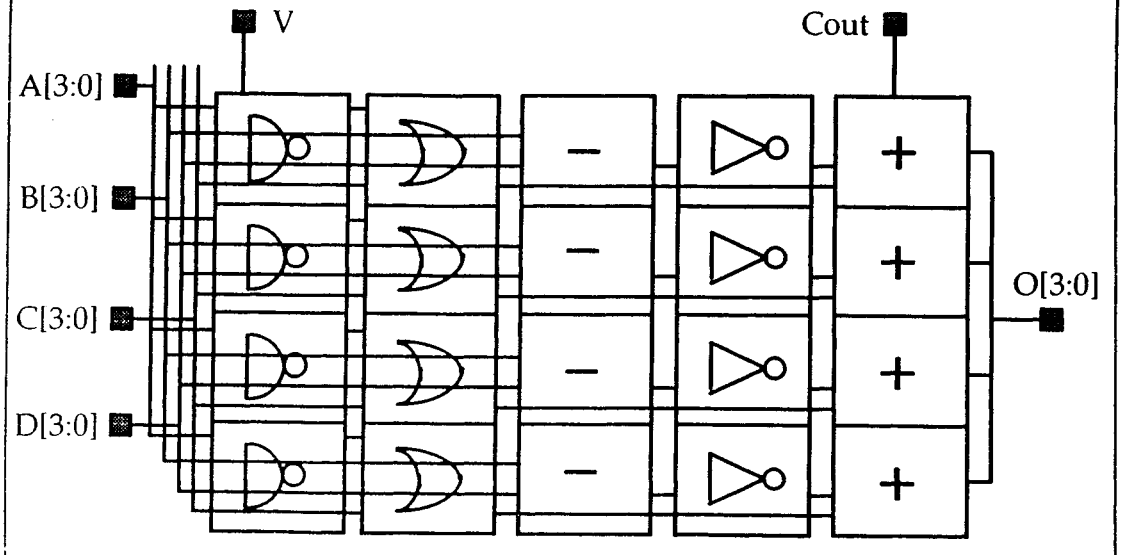


STANDARD CELLS ROUTER
VERSUS
DATA PATH ROUTER

DATA PATH: AN EXAMPLE (1)



DATA PATH: AN EXAMPLE (2)



TIMING VERIFICATION

- SIMULATORS
  - CIRCUIT-LEVEL.
  - TIMING.
  - SWITCH-LEVEL.
  - LOGIC-LEVEL.
- VERIFIERS (PATTERN INDEPENDENT)
  - TIMING.

It was a real pleasure working with you. I hope that our ALLIANCE *tools will help you in teaching* VLSI *once back home and I look forward to your feedback.*

Very truly yours...

Nizar

Actel

Low Power Design & Power Estimation

Actel

Outline

- Motivation for Power Tools
- Low-Power Design Methodology
- Principles for Power Reduction
- Principles for Power Estimation
- Conclusion

Motivation for Power Tools

Deep-Submicron Technologies

Higher Density and Performance Capabilities
(FPGAs: 100 000 Gates; 100 MHz Clock rates)

Power Dissipation Problem

Motivation for Power Tools

4 times / 3 Years Increase for the last 20 Years

- PowerPC / Motorola 8.5 Watts
- Pentium / Intel 16 Watts
- Alpha / Dec 30 Watts
- Alpha 300 Mhz / Dec 50 Watts

Motivation for Power Tools

Power = Cost

For Major Applications Today

- Battery Lifetime (Cellular, Medical, ...)
- Packaging Cost
- Reliability (Time to Failure)
- Green PC program (< 30 Watts)

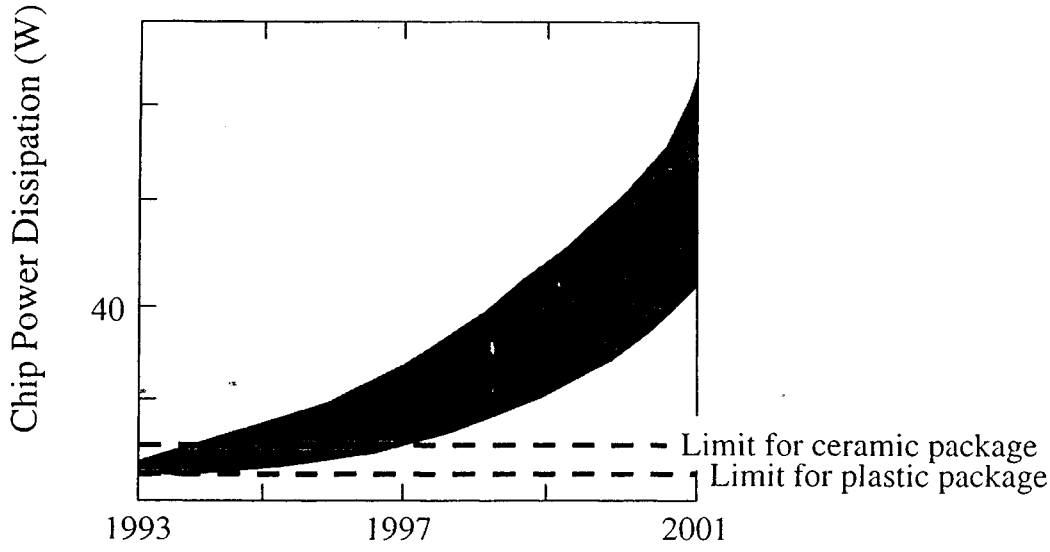
Motivation for Power Tools

Power = Less Performance

- Clock Frequency
- Temperature Increase
- Electromigration

Motivation for Power Tools

Packaging Cost is an Issue



Motivation for Power Tools

Today...

Design Win = f(performance, cost)

Tomorrow...

Performance = f(Power, ...)

cost = f(Power, ...)

Design Win will also be low-power dependent

Motivation for Power Tools

We Need...

✓ Low Power Design Methodology

✓ Power Estimation Tools

✓ Power Optimization Tools

Methodology

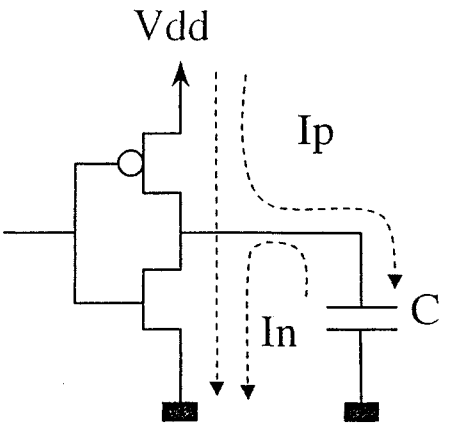
Analogous to Timing Methodologies

✓ All Levels of Abstraction

✓ Back-Annotation from physical Design

Sources of Power Consumption

- Dynamic Power (70-90%)
  (Switching activity)
- Short-Circuit Power (10-30%)
- Leakage Power (<5%)
  (Important for battery lifetime)



Expression for CMOS Power

- Gate Generating a Simple Clock Signal with Frequency f

$$P_{average} = C\ V_{dd}^2\ f$$

- In general, a signal with a transition density D

$$P_{average} = 1/2\ C\ V_{dd}^2\ D$$

Power Reduction

$$P_{average} = 1/2\ C\ V_{dd}^2\ D$$

- Reducing Switching Activity
  Prevent glitches (Architecture, Synthesis, ...)
  20% of power increase due to glitches
- Reducing Load Capacitance
  Gate sizing, Low-Power cell library
  Circuit techniques (Pass-Transistor, ...)
- Reducing Supply Voltage
  Drawback: Circuit delay increases

Power Estimation

Two Problems

- Design Dependent
  Tools Should be Available to the Customer
- Input Pattern Dependent (more Central Problem)
  Difficult when the application is not known
  A good vector set may be very long

High-Level Power Estimation

- FPGA: Block Macromodels Available
  Problem to estimate net consumption
- Models for Logical Level, RTL Level, Behavioral Level
  Need for a power cost function

What About Accuracy and Improvements?

Assuming we Have a represetative Vector Set,

- Low-Level Timing Simulation — 10% from Spice
- Low-Level Static Technique — 20-60% from Spice
- Low-Level Dynamic Technique — 10-20% from Spice

Improvement

- At the Logical Level is About — 5%
- At the RTL Level May Reach — 90%

Conclusion

- Power Consumption Issues Can no Longer be Ignored for High Density FPGA Design
- Timing / Power : The same challenge
  - Input pattern dependency
  - All abstraction levels
  - Power and timing constraints
  - Net consumption is becoming very significant
- DPCS IEEE 1481 is also for Power
- A Balance between Power, Area, and Delay
- Absolute Accuracy is not a Critical Issue

FPGA Solutions

Motivation

- Cost (Small Series, New Designs, ...)
- Rapid Prototyping
- Emulators
- Development Time
- Test Time

Motivation

- Relatively High Density (100 000 Gates)
- Relatively High Performance Capabilities (100MHz)

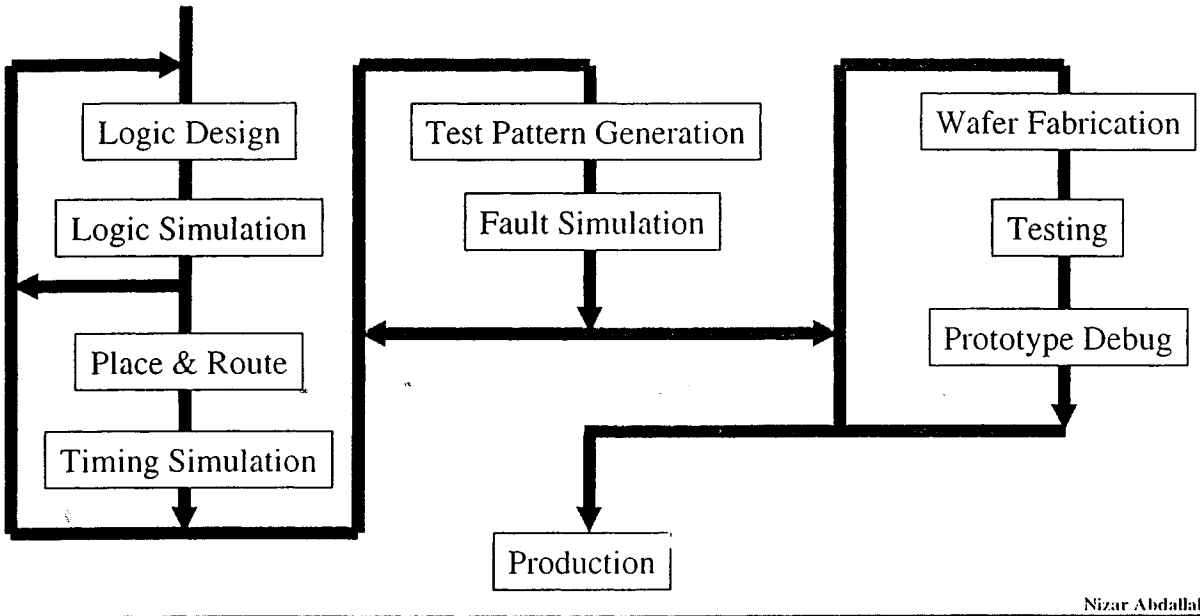
Motivation

- Market in 1993 : $539M
- Market in 1998 : $2124M
- Annual Growth Rate of 32%

Sales

Actel	$151.3M
Altera	$639.0M
Xilinx	$610.6M

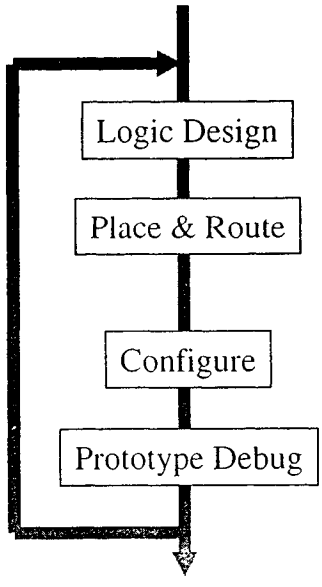
Typical ASIC



Nizar Abdallah

198

FPGA



Nizar Abdallah

190

Performance

- Relatively High Density (100 000 Gates)
- Relatively High Performance Capabilities (100MHz)

Memory Based Architecture

Can be ...

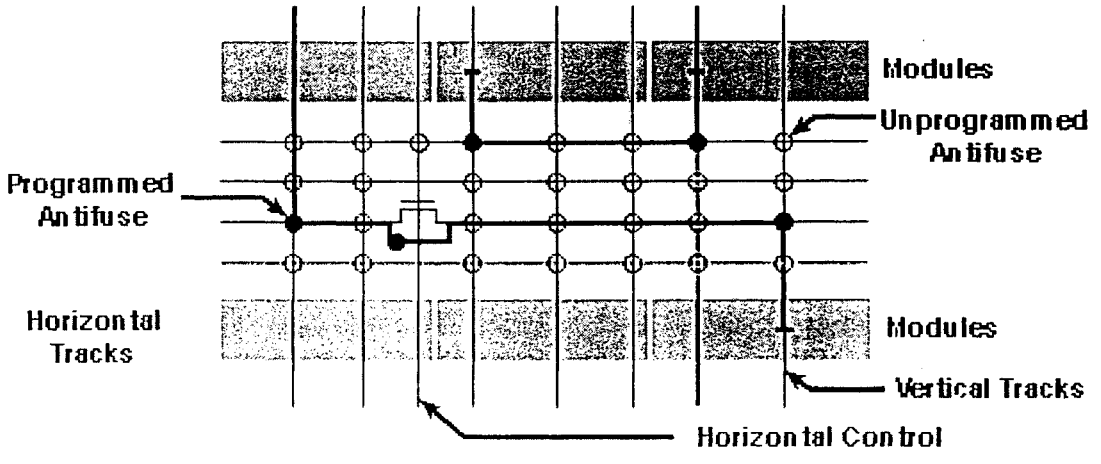
- Changed During the Development
- Updated after Delivery to the Customer
- Purchased in Larger Quantities
- Reused (No Inventory if not Sold)
- Fully Tested Prior to Delivery

One-Time Architecture (Antifuse)

Have ...

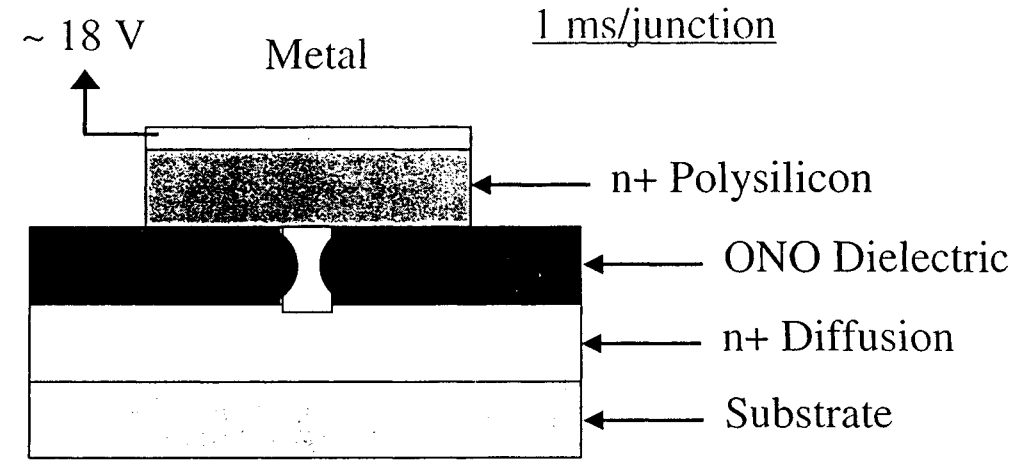
- Higher Speed (Less RC Delays on the Interconnections)
- High Reliability
- No Time-Delay to Reload the Interconnection Information (Available Immediately on Power-Up)

Actel Antifuse



Actel PLICE Antifuse

Programmable Low Impedance Circuit Element



Open Resistance = 10s of MOhms
Short Resistance = 500 Ohms

Nizar Abdallah

Actel SX Family

Features

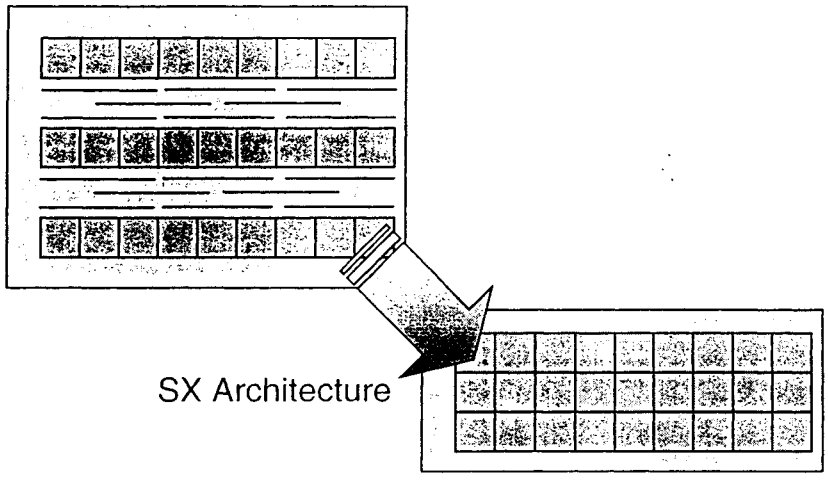
- 2 Global Clocks
- PCI Compliant I/O
- Mixed Voltage Operation
- Built-In JTAG

- More Routing Resources
- New Antifuse
- New Architecture

Nizar Abdallah

Routing Interconnects are Above Logic Modules

Pre-SX Architecture



A 0.6 Micron Technology

Direct Connects

- Connects Combinatorial Cell (C-Cell) to its Adjacent Sequential Cell (R-Cell)
- No Antifuses
- 0.1 ns Routing Delay

SX Routing Resources

Fast Connects

- Every Cell Output Connects to One
- Accessible by Cells in the Same Cluster or the One Below by One Antifuse
- 0.4 ns Routing Delay

SX Parts

Part#	SX08	SX16	SX16P	SX32	SX64
Gates	8,000	16,000	16000	32,000	64,000
MaxIO	129	177	177	246	340
Rcells	256	528	528	1080	2160
CCells	512	924	924	1800	3600
Availb.	98	98	98	98	99

http://www-asim.lip6.fr

Good Luck...