Part VII
VHDL Synthesis Techniques and Recommendations
Introduction

- Synthesis is the translation process from an abstract description of a hardware device into an optimized technology specific gate level implementation.

- May be done
  - Manually via schematic entry
  - Automatically via EDA tools that use a hardware description language (HDL) as an input medium to generate constraint driven gate configurations.
HDL Based Design Flow

- Pre-Synthesis Steps
  - Functional Specification of the design
  - Design Entry

- HDL Coding in VHDL/Verilog RTL

- Graphical Entry Tools
  - Summit Design (Visual XOR)
  - SpeedChart (Speed Electronics)
  - Simulation Graphical Environment (Synopsys SGE)
  - Design Source (Synopsys)
  - Design Manager (Mentor)
  - View Draw (Viewlogic)
  - Escalade (Design Book)
HDL Based Design Flow

- Pre-Synthesis Steps (cont.)
  - RTL/Behavioral or Functional Simulation of the HDL
    - Model Technologies (MTI)
    - Quicksim (Mentor)
    - Synopsys VHDL System Simulator (VSS)
    - Vantage (ViewLogic)
    - Leapfrog (Cadence)
    - Verilog XL (Cadence)
    - Chronologic (Viewlogic)

- SYNTHESIS:
  Logic synthesis is the process of translating and optimizing a high-level design description to gates from a technology library
Synthesis Definition

- **Translation**: Conversion of RTL into an unoptimized gate-level description.

- **Logic Optimization**: Transition from a suboptimal generic logic implementation to a closer-to-optimal implementation in terms of area and speed.

- **Technology Mapping**: Transition from an optimized generic netlist to cells from ASIC vendor’s library.
Synthesis
Post Synthesis

- **ASIC Synthesis**
  - Design Compiler (Synopsys)
  - Bool-Dozer (IBM)
  - Synergy (Cadence)
  - Autologic (Mentor)

- **FPGA Synthesis FPGA Compiler**
  - FPGA Compiler (Synopsys)
  - Logic Explorer (Galileo, Exempler)

- **Timing Analysis**
  - Static
  - Dynamic
  - Hybrit
Post Synthesis

- Test Insertion, Pad Synthesis & ATPG
  - Synopsys Test Compiler
  - Sunrise (ViewLogic)
  - Mentor
- Post Synthesis, gate-level Simulation
  - Model Technologies
  - Verilog XL, (Cadence)
- Floorplanning
- Place and Route
- Back Annotation
- In Place optimization (Design Compiler)
- Gate Level Simulation (MTI)
Post - Synthesis

- Transistor Level Simulation, Pad Synthesis & ATPG
  - TimeMill (EPIC)
  - PowerMill (EPIC))
- Engineering Change Order
- Tapeout
- Test Vector Generation
Disclaimer

- Though VHDL Is a Key Element of a Synthesis Activity It Is Only Part of the Solution
  - Target Library Design Rules
  - Hardware Timing Goals
  - Hardware Area Goals
  - Environmental Goals
- Digital Designer Must Also Master the Synopsys Proprietary GUI and Command Language Features in Order to Ensure That the Synthesized Hardware Will Best Satisfy All of Its Intended Real-World Design Constraints and Goals.
Forces Driving Synthesis Algorithm

HDL Coding Style

Design Constraints

Environmental Attributes

Target Technology
Hardware Description Languages and Synthesis

- Hardware Description Language (HDL) Used to Model Targeted Device Is Technology Independent
  - During Synthesis Appropriates Cells Are Selected From the User Specified Technology Library

- HDL Coding Style Is a Very Important Driving Force for the Automated Synthesis Algorithms and Will Play Role in the Final Hardware Configuration That Is Build.
During the synthesis NAND gates may be selected because of:

- Gate availability's in targeted Technology
- Area Minimization requirements
Synthesized Combinational Logic: Possible Alternatives (cont’d)

Additional Buffers may be included in order to satisfy an output drive specification.
Key Synthesis Fact

- In General, Synthesis Does Not Support the After Clause in a Signal Assignment

  \[ C <= A \text{ and } B \text{ after } 10 \text{ Ns} \quad -- \text{Not Supported by Baseline} \]
  \[ -- \text{Synthesis Tool Suite} \]

- In the Baseline Synthesis Domain Signal Updates Should Be Scheduled to Occur at the Next Delta Time Unit in the Simulation Domain

  \[ C <= A \text{ and } B \]
  \[ -- \text{Supported by Synthesis} \]

*Note:* Synopsys Behavioral Compiler Product Supports the After Clause
Static Sensitivity Rule

- If the synthesized process has a static sensitivity list, then every read signal must be a member of this list. Otherwise the synthesis tool will create a hardware configuration that concurs with this requirement even though the original process does not.

- Might result in possible VHDL simulation, Synthesis Mismatches.
Static Sensitivity Rule

Original VHDL Model

```vhdl
process (A, B)
begin
    D <= (A AND B) OR C;
end process;
```

Synthesis view and Re-interpretation of original VHDL code.

```vhdl
process (A, B, C)
begin
    D <= (A AND B) OR C;
end process;
```
Optimization of Redundant Test in ELSIF Branch

```
process (A, B, USE_B)
begin
    if USE_B = '1' then
        D_Out <= B;
    elsif USE_B = '0' then
        D_Out <= A;
    end if;
end process;
```

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

`else` clause is missing but Latch is not generated, because synthesis tool has reduced that the `if` and `elsif` conditions are mutually exclusive and exhaustive.
VHDL => LATCH

Latches are inferred during Synthesis. Whenever the Following Conditions Occur:

- Conditional Expression Are Not Completely Specified
  - Else Clause Is Omitted
- Objects Conditionally Assigned in an If Statement are not assigned a value before entering this if statement
- The VHDL attribute ‘Event is not present in the conditional expression
If Statement => LATCH

process (ENABLE, DATA) begin
    if ENABLE = '1' then
        Q <= DATA;
    end if;
end process;
A latch is inferred during synthesis so that the derived hardware will agree with the VHDL fact that signals and process declared variables maintain their respected value until they are assigned new ones.
Synthesis Rules Regarding ‘Z’

- ‘Z’ cannot occur in an expression
  - the following line will be flagged as an error during the synthesis elaboration phase
  - \( \text{DATAOUT} \leq \text{‘Z’ AND DATA\_IN}; \)

- Though accepted by synthesis comparison to ‘Z’ are always interpreted as being false
  - this assumption will lead to a simulation / synthesis
  - if DATA\_IN = ‘Z’ then
  - is synthesized the same as
  - if false then
Importance of Good Coding Style: Efficient Synthesized hardware

Because the WHEN conditions of a VHDL case statement are mutually exclusive and exhaustive synthesis rules can efficiently derive a single MUX equivalent circuit.

```
process( SEL, A, B, C, D)
begin
  case SEL is
    when "00"=>D_Out <=A;
    when "01"=>D_Out <=B;
    when "10"=>D_Out <=C;
    when "11"=>D_Out <=D;
    end case
end process
```
Observation Regarding: FOR Statement => Combinational Logic

- Variable Used Because Inertial Signal Updates Will Neither Simulate nor Synthesize Correctly
- In Essence for Loop Is Unraveled During Synthesis
  - Transport Delay Models Are Not Supported by Synthesis
- Assignment to Result Before Entering for Loop Is Required Because Synthesis Requires That Variables Be Written to Before They Are Read
- Otherwise a Warning Message Will Be Generated During Elaboration
  - Might Then Synthesize Into a Circuit With Feedback Loop
State Encoding

By default synthesis implements a state encoding that reflects the binary presentation of the `POS value of the machines state enumeration type.

- For example
  type STATE_TYPE is (SO, S1, S2, S3);
- Since STATE_TYPE’POS(S0) = 0, state SO is encoded into “00”
- Analogously S1, S2, S3 are encoded into “01” “10” and “11” respectively.