F8: VHDL for Synthesis

Outline

- VHDL Synthesis Flow and Synthesis Tasks
- One circuit many descriptions and many implementations
  - Synthesis of Combinational Circuits
  - Synthesis of Sequential Logic
- Synthesis of VHDL constructs
- Synthesizable VHDL subset
Advantages of Hardware Description Languages as Design Entry

- Description of the functionality is done at a higher level
- HDL description can be simulated and validated at an early stage in the design process
- The HDL description can be synthesised into a gate-level description of a chosen technology
- A design written in an HDL is more easily understood than a gate-level net-list or a schematic description.
- HDLs prevent errors, since they provide strong type checking.

VHDL: Modeling and Synthesis

- The hardware description languages VHDL and Verilog were designed for modeling hardware
  - The idea was to model hardware on a high abstraction level
  - Features for modeling: concurrency, timing, hierarchy, reuse of components, state behavior, synchronous behavior, asynchronous behavior, synchronization,…
VHDL: Modeling and Synthesis

- But then tools have been developed for synthesis!
- ... and could not use the high-level features of VHDL

VHDL Description Space

- The VHDL subset that is synthesizable is tool specific. Do not expect that your VHDL description is synthesizable with another tool!
- There is a IEEE standardization group that works on a standardized VHDL subset for synthesis.
A VHDL model is a concurrent system of sequential and combinational processes.
VHDL Based Design Flow

- Write a design description in VHDL
- Write a VHDL test bench
- Simulate the design
- Synthesize the VHDL description with your VHDL compiler
- Use Design Compiler/Leonardo/Quartus to output a gate-level netlist for your target technology
- Check the timing (i.e., perform a Static-Timing Analysis - STA)

AND/OR
- Use Design Compiler to output a gate-level VHDL-description for your target technology
- Simulate the gate-level netlist with your test bench (needed for race conditions)
- Compare the result with your original simulation.
One System – Many VHDL Descriptions

Idea...

Behavioral Description
Dataflow Description
Structural Description

Circuit 1
Circuit 2
Circuit 3

VHDL Styles

• Behavioral Style
  Expresses the functionality of the system as an algorithm(s).
  • Processes: Concurrent behavior
  • Sub-programs for hierarchical behavior

• Dataflow Style
  – A description which expresses the data dependencies and thus the flow of data.
  – It is expressed by concurrent statements
VHDL Styles

• Structural Style
  – Describe the system in terms of interconnection of components.
  – Components from a library / synthesized separately
  – Mechanism to build hierarchical systems using bottom-up approach
  – Component at the lowest level in hierarchy is described using behavioral description

Styles: Illustration

entity MUX4TO1 is
  port( A,B,C,D : in std_logic;
       S0,S1   : in std_logic;
       Y       : out std_logic);
end MUX4TO1;
Behavioral Style Architecture

Architecture BEHAVIORAL of MUX4TO1 is
Begin
    process(A,B,C,D,S1,S0)
    begin
        case S1&S0 is
            when "00" => Y <= A;
            when "01" => Y <= B;
            when "10" => Y <= C;
            when "11" => Y <= D;
            when others => null;
        end case;
    end process;
end BEHAVIORAL;

Dataflow Style Architecture

Architecture DATAFLOW of MUX4TO1 is
begin
    Y <= (A and not S1 and not S0) or
        (B and not S1 and S0) or
        (C and S1 and not S0) or
        (D and S1 and S0);
end DATAFLOW;
Architectural STRUCTURAL of MUX4TO1 is
Signal I0,I1,P,Q,R,S: std_logic;
-- Component Declarations
begin
INV1: INV port map (S0,I0);
INV2: INV port map (S1,I1);
A1:  AND3 port map (A,I1,I0,P);
A2:  AND3 port map (B,I1,S0,Q);
A3:  AND3 port map (C,S1,I0,R);
A4:  AND3 port map (D,S1,S0,S);
U1:  OR3 port map (P,Q,R,S,Y);
end STRUCTURAL;

Which style is best?

• Specification Complexity
  – Behavioral style is more natural and easy for very complex system
    Difficult for synthesis tools to handle
  – Structural style is similar to hardware and easy to synthesize.
    Difficult to describe complex systems
  – A mix is better!
Which style is best (ctd.)?

• Implementation performance: cost, delay, power..
  – Depends on synthesis tool
    • Maturity of the tool - Tools keep incorporating “expertise” of good designers
    • Optimization features
  – Depends on designer experience
    • Hardware designers prefer structural design since they can
      1) reuse their previous designs as components in larger designs
      2) trick the synthesis tool to produce exactly the hardware structure they want.
      3) easy to debug & test part by part bottom up.

Finite State Machines

• Most synthesis tools have optimisation tools for synthesis!
• In order to use these tools the synthesis tool has to be aware of the fact that an FSM exists.
• Synopsys has attributes that should be used for FSM modelling.
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY Counter03 IS
    PORT (  
        clk   : IN  std_logic;
        resetn  : IN  std_logic;
        three  : OUT std_logic);
END Counter03;

ARCHITECTURE FSM OF Counter03 IS
    SUBTYPE state_type IS integer RANGE 0 TO 3;
    SIGNAL pres_state, next_state : state_type := 0;
    ATTRIBUTE STATE_VECTOR : string;
    ATTRIBUTE STATE_VECTOR OF FSM : ARCHITECTURE IS "PRES_STATE";
Modeling of an FSM (ctd.)
(A modulo-4 counter)

BEGIN -- FSM
  U0:
  PROCESS (clk, resetn)
  BEGIN -- PROCESS
    IF resetn = '0' THEN
      pres_state <= 0;
    ELSIF clk'event AND clk = '1' THEN
      pres_state <= next_state;
    END IF;
  END PROCESS;
END FSM;

Modeling of an FSM (ctd.)
(A modulo-4 counter)

U1:
  PROCESS (pres_state)
  BEGIN -- PROCESS
    three <= '0';
    CASE pres_state IS
      WHEN 0 TO 2 => next_state <= pres_state + 1;
      WHEN 3 => next_state <= 0;
      three <= '1';
      WHEN OTHERS => next_state <= 0;
    END CASE;
  END PROCESS;
END FSM;
Extracting the FSM

Synopsys and Leonardo let you choose the FSM encoding.

Synopsys FSM encoding menus

Binary Encoding
Comparison
(Modulo-4-Counter)

<table>
<thead>
<tr>
<th></th>
<th>Setup Time [ns]</th>
<th>Delay [ns]</th>
<th>$f_{\text{max}}$ [MHz]</th>
<th>Comb. Area</th>
<th>Seq. Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary</td>
<td>0.35</td>
<td>0.87</td>
<td>816</td>
<td>9.75</td>
<td>4.5</td>
</tr>
<tr>
<td>Gray</td>
<td>0.35</td>
<td>0.66</td>
<td>990</td>
<td>3.00</td>
<td>4.5</td>
</tr>
<tr>
<td>OneHot</td>
<td>0.58</td>
<td>0.29</td>
<td>1149</td>
<td>0.00</td>
<td>10.75</td>
</tr>
</tbody>
</table>

- The design can be optimized a lot by choosing the best state encoding style.
- It depends on the design, which style is most efficient

Synthesis of an IF-statement

```vhdl
signal A, B, C, P1, P2, Z: BIT;
if (P1 = '1') then
  Z <= A;
elsif (P2 = '0') then
  Z <= B;
else
  Z <= C;
end if;
```
Synthesis of an IF-statement

The IF-statement implies a priority!

Synthesis of a case statement

type ENUM is (PICK_A, PICK_B, PICK_C, PICK_D);
signal VALUE: ENUM;
signal A, B, C, D, Z: BIT;
...
case VALUE is
  when PICK_A => Z <= A;
  when PICK_B => Z <= B;
  when PICK_C => Z <= C;
  when PICK_D => Z <= D;
end case;
Synthesis of a CASE statement

- The CASE statement implies no priority!

Exercise

- Write a VHDL description that generates this implementation.
- Can you improve the design?
**Code Transformations**

- **Associativity** - \((a+b)+c=a+(b+c)\)
- **Distributivity** - \(c(a+b)=ca+cb\)
- **Commutativity** - \(a+b=b+a\)

**Parentheses are important!**

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

![Diagram](image_url)
Parenthesis are important!

\[ Y \leq (A + B) + (C + D); \]

Give the tool a good starting position!

---

**VHDL -Hardware Correspondence**

<table>
<thead>
<tr>
<th>VHDL Construct</th>
<th>Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variables and Signals</td>
<td>Flip-flops, latches, wires</td>
</tr>
<tr>
<td>Arithmetic operators (+, -, *)</td>
<td>Adder, ALU, multiplier..</td>
</tr>
<tr>
<td>Logic operators</td>
<td>Gates</td>
</tr>
<tr>
<td>Relational operators</td>
<td>Comparator, ALU</td>
</tr>
<tr>
<td>Control Constructs (for, if-then, case, .....)</td>
<td>Decoders, Multiplexers, priority encoder, ...</td>
</tr>
<tr>
<td>Hierarchy Description</td>
<td>Hierarchical Hardware</td>
</tr>
<tr>
<td>Resolution Functions</td>
<td>Tri-state logic, wire-and, wire-or</td>
</tr>
</tbody>
</table>
Variable as Wire

```vhdl
... end process;
process(a,b)
variable x,y;
begin
...
x := a + b;
y := x + 1;
...
end process;
...
```

Variable as Latch

```vhdl
process(a,b)
variable x,y,z;
begin
...
if cond = '1' then
  x := a + b;
end if;
y := x + 1;
...
end process;
```
Edge Expressions and Clocked Bodies

Edge Expressions

- signal’event and signal = ’1’
- signal’event and signal = ’0’

Edge Expressions

process (x, y, z);
begin
  if x’event and x = ’0’ then
    ...
  end if;

There is no explicit concept of clock in VHDL. The synthesis tool has to infer the clock.

A WAIT or IF statement is normally used to make the sequential behavior synchronous.

Register/Latch Inference Rules

Type of Body

- Clocked Body
  - Signals
  - Variables
    - Flip-flops
      - RBW: Read before write
      - WBR: Write before read
  - RBW
  - WBR

- Non Clocked Body
  - Signals & Variables
    - Driven in all branches
    - Not driven in all branches
  - Wires
  - Latch
Exercise

- Create a synthesizable model in VHDL of a JK-Flip-Flop with active low asynchronous reset and preset.

Exercise

- What will be the synthesis result of the following code?

ENTITY SYNQ IS
  PORT (
    A : IN  bit;
    b : IN  integer RANGE 0 TO 3;
    c : IN  bit_vector(0 TO 3);
    d : OUT bit);
END SYNQ;

ARCHITECTURE Alg OF SYNQ IS
  SIGNAL x : bit;
BEGIN  -- Alg
    x <= c(b);
    PROCESS(x,a)
      BEGIN
        IF a = '1' THEN
          d <= x;
        END IF;
      END process;
END Alg;
Synthesis of Arithmetic Operators

• Only a subset of VHDL operators supported by synthesis tools
  – Supported operations: usually +, -, =, >, <, <=, abs, *
  – The designer must write VHDL description (behavioral or structural) of all un-supported operators: /, N**x

• Specifying a range of integer is important for efficient synthesis
  – Suppose integer x takes only two values 100, 101
    • It will be assigned 7 bits
    • It may be better to declare x as constrained integer

Synthesis of ”+”

ENTITY adder IS
  PORT(
    x,y: IN integer RANGE –128 to 127;
    z: OUT integer RANGE –128 to 127);
End ENTITY;

ARCHITECTURE adder_8bit OF adder IS
BEGIN
  z <= x+y;
END;
Sequential Statements

The statements within a process are executed sequentially.

```vhdl
process(a,b,c,y)
begin
  a <= b + c;
  p <= a + y;
end;
```

During simulation the statements are executed one after the other but it does not model the delay of addition.

Synchronous Behavior: Restrictions

```vhdl
process (clk)
begin
  if (clk'event and clk = '1') then
    p <= a+b;
  else
    q <= a-b;
  end if;
end process;
```

```vhdl
process (clk_a, clk_b)
begin
  if (clk_a'event and clk_a = '1') then
    p <= a+b;
  end if;
  if (clk_b'event and clk_b = '1') then
    q <= a+b;
  end if;
end process;
```

Illegal! Only one clock allowed!
Sensitivity List and Synthesis

• The sensitivity list is very important for making the simulation efficient but it is not used by the synthesizer.

```vhdl
process (x)
begin
    z <= x and y;  -- z is only evaluated when there is a change in value of x
end;
```

Be careful, when you use a sensitivity list that does not include all values!

Un-Synthesizable VHDL Subset

• Not supported: Ignored
  – The construct is ignored by the synthesis tool
  – Examples:
    • Initialization of variables or signals
    • Assert statements
    • Physical Type: Time
      – The synthesizer ignores the AFTER clause in expressions
      Example: x <= y AFTER 100ns

• Not Supported: Illegal
  – Data types
    • Files, real, generic that are not integers (Altera allows strings)
**Packages supported by Synopsys**

(SOLD HDL Compiler for VHDL, Reference Manual, App. B)

Package `std_logic_1164`

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

- Defines data type `std_logic`

Package `std_logic_arith`

```vhdl
library ieee;
use ieee.std_logic_arith.all;
```

- Defines data type `signed` and `unsigned` and arithmetic operations on these data types

Package `numeric_std`

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

- Defines data type `signed` and `unsigned`
- Be careful, `numeric_std` and `std_logic_arith` have OVERLAPPING definitions!

Package `ATTRIBUTES`

```vhdl
library SYNOPSYS;
use SYNOPSYS.ATTRIBUTES.ALL;
```

- Defines attributes which are later used by Design Compiler, e.g. `state_vector`, which is used for FSMs
Have in mind...

• Delay in Models:
  – *after* clauses are ignored.
  – Delay is technology dependent!

• Data Types
  – Use `std_logic`, `std_logic_vector`, `signed`, `unsigned` and constrained integers.
  – Unconstrained integers result in 32 bits!

• Initial Values are ignored!
  – All sequential circuits should have a reset!

Have in mind... (ctd.)

• Attributes
  – Use attributes to specify the state vector of an FSM.
  – You may use also an attribute for state encoding, to force the synthesis tool to choose an encoding style

• RAMs
  – Use a predefined RAM block, since a RAM is usually synthesized with Flip-Flops or Latches

• Latches
  – Avoid unnecessary latches and flip-flops
...and don’t forget!

THINK HARDWARE

Appendix
Synthesis of Logical Operators

... signal A, B, C: BIT_VECTOR(3 downto 0);
   signal D, E, F, G: BIT_VECTOR(1 downto 0);
   signal H, I, J, K: BIT;
   signal L, M, N, O, P: BOOLEAN;
   ...
   A <= B and C;
   D <= E or F or G;
   H <= (I nand J) nand K;
   L <= (M xor N) and (O xor P);

Synthesis of logical operators

[Diagram of logical operators]

[Diagram of logical operators]
Synthesis of relational operators

signal A, B: BIT_VECTOR(3 downto 0);
signal C, D: BIT_VECTOR(1 downto 0);
signal E, F, G, H, I, J: BOOLEAN;
G <= (A = B);
H <= (C < D);
I <= (C >= D);
J <= (E > F);
Synthesis of relational operators

signal J, K, L: INTEGER range 0 to 3;
J <= K + L; -- Simple addition

Synthesizing an adding operation
Synthesizing Multiplications with powers of 2

signal A, B, C, D, E, F, G, H:
    INTEGER range 0 to 15;
A <= B * 4;
C <= D / 4;
E <= F mod 4;
G <= H rem 4;
signal A, B: INTEGER range 0 to 15;
signal Y, Z: INTEGER range 0 to 31;
signal X: INTEGER range 0 to 1023;
...
A <= B * 3;
X <= Y * Z;
entity d_latch is
  port (GATE, DATA: in std_logic;
       Q : out std_logic);
end d_latch;
architecture rtl of d_latch is
begin
  infer: process (GATE, DATA) begin
    if (GATE = '1') then
      Q <= DATA;
    end if;
  end process infer;
end rtl;

Synthesis of a D-Latch

Synthesis of a Latch

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Type</th>
<th>Width</th>
<th>Bus</th>
<th>MB</th>
<th>AR</th>
<th>AS</th>
<th>SR</th>
<th>SS</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q_reg</td>
<td>Latch</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>N</td>
<td>N</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Synthesis of Latches and Flip-Flops

- Y in the SR and SS columns indicates that the flip-flop has a synchronous reset and a synchronous set.
- N in the AR, AS, and ST columns indicates that the flip-flop does not have an asynchronous reset, asynchronous set, or synchronous toggle.
- A dash (–) in the Bus and MB columns indicates that these columns are not relevant to this design.

Synthesis of a D-Flip-Flop

```vhdl
entity dff_pos is
  port (DATA, CLK : in std_logic;
        Q : out std_logic );
end dff_pos;

architecture rtl of dff_pos is
begin
  infer : process (CLK) begin
    if (CLK'event and CLK = '1') then
      Q <= DATA;
    end if;
  end process infer;
end rtl;
```
entity dff_async_reset is
  port (DATA, CLK, RESET : in std_logic;
        Q : out std_logic );
end dff_async_reset;

architecture rtl of dff_async_reset is begin
  infer : process ( CLK, RESET) begin
    if (RESET = '1') then
      Q <= '0';
    elsif (CLK'event and CLK = '1') then
      Q <= DATA;
    end if;
  end process infer;
end rtl;
Synthesis of a D-Flip-Flop with asynchronous reset

use synopsys.attributes.all;
entity dff_sync_reset is
port (DATA, CLK, SET : in std_logic;
    Q : out std_logic);
attribute sync_set_reset of SET : signal is "true";
end dff_sync_reset;

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Type</th>
<th>Width</th>
<th>Bus</th>
<th>MB</th>
<th>AR</th>
<th>AS</th>
<th>SR</th>
<th>SS</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q_reg</td>
<td>Flip-flop</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>
Synthesis of a D-Flip-Flop with synchronous reset

architecture rtl of dff_sync_reset is
begin
    infer : process (CLK) begin
        if (CLK’event and CLK = '1') then
            if (SET = '0') then
                Q <= '0';
            else
                Q <= DATA;
            end if;
        end if;
    end process infer;
end rtl;

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Type</th>
<th>Width</th>
<th>Bus</th>
<th>MB</th>
<th>AR</th>
<th>AS</th>
<th>SR</th>
<th>SS</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q_reg</td>
<td>Flip-flop</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>
entity three_state is
  port(IN1, ENABLE : in std_logic;
       OUT1 : out std_logic);
end;
architecture rtl of three_state is
begin
  process (IN1, ENABLE) begin
    if (ENABLE = '1') then
      OUT1 <= IN1;
    else
      OUT1 <= 'Z'; -- assigns high-impedance state
    end if;
  end process;
end rtl;

<table>
<thead>
<tr>
<th>Three-State Device Name</th>
<th>Type</th>
<th>MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUT1_tri</td>
<td>Three-State Buffer</td>
<td>N</td>
</tr>
</tbody>
</table>