Assignment

- Project 4: Using synthesis tools—Synplify Pro and Webpack
- Due 11/11 beginning of class

Last time

- Generics
  - Used to parameterize models
  - E.g., Delay, bit width
- Configurations
  - Configuration specification
  - Configuration declaration

Last time

- Default binding rules:
  - If the entity name is the same as the component name, then this entity is bound to the component
  - If there are multiple architectures for the same entity, the last compiled architecture for the entity is chosen

Back to the big picture

- So far, we've focused on learning VHDL for simulation
  - Functionality, timing correctness
- What you know how to do:
  - create models
  - check them with testbenches
  - share code with your partner
  - configure different versions

Back to the big picture

- What you don't know how to do:
  - Turn those models into working (non-smoking) hardware
- Now we'll start talking about synthesis
  - Generally: Transforming a less detailed representation into a more detailed representation
**A small example**

- **Draw a schematic for this:**
  ```vhdl
  Library ieee; use ieee.std_logic_1164.all;
  entity syn is
    port (a, b, c: in std_logic;
    z: out std_logic);
  end syn;
  architecture first of syn is
  begin
  process(a, b, c)
  if (a = '0') or (b = '0') or (c = '0') then
    z <= '1';
  else
    z <= '0';
  end if;
  end process;
  end first;
  ```

---

**Another small example**

- **Draw a schematic for this:**
  ```vhdl
  Library ieee; use ieee.std_logic_1164.all;
  entity syn2 is
    port (a, b, c: in std_logic;
    z: out std_logic);
  end syn2;
  architecture first of syn2 is
  begin
  process(a, b)
  if (a = '0') then
    z <= '0';
  elsif (b'event and b = '1') then
    z <= c;
  end if;
  end process;
  end first;
  ```

---

**High level design flow**

1. **Design specification**
2. **HDL capture**
3. **RTL simulation**
4. **Synplify Pro**
5. **RTL synthesis**
6. **Functional gate simulation**
7. **Webpack**
8. **Place and route**
9. **Post layout timing simulation**

---

**Synthesis introduction**

- **Synthesis:** Creating a new representation from a less detailed representation
- **Specifically for us,** creating an FPGA netlist from a behavioral VHDL model
- **Two parts to the problem:**
  - Convert a VHDL model into RTL schematic
  - **Map the RTL schematic onto the available hardware**

---

**Synthesis introduction**

- **Keep in mind:** Just because you can simulate a VHDL description doesn't mean you can synthesize it.
- **The synthesis tool:**
  - ignores delay expressions—after clauses, wait for statements.
  - is picky about certain constructs
    - Examples: if (a'event or b'event) then....
    - Synthesizer gives an error of “clock expression should contain only one signal”
    - cannot handle all types (e.g. multidimensional arrays, type TIME)
- **Questions:**
  - What type of hardware can implement
    - A signal?
    - A variable?
    - A process?
  - When does a VHDL description require only combinational logic to be implemented in hardware? Sequential logic?
  - How many bits are required to represent a signal?
### Synthesis introduction

- **Hardware inference:** How do synthesis tools infer a hardware implementation from VHDL code?

- Understanding how VHDL is converted into hardware helps us write descriptions that do what we expect when synthesized.

### Synthesis Semantics

- VHDL contains a rich variety of constructs
- Some constructs are intended for use in executable specifications and cannot be directly synthesized
- Most vendors are not willing to put in writing exactly which constructs their synthesis tool CANNOT handle.

### Common Synthesizable Data Types

- **Std_Logic**
- **Std_Logic_Vector**
- **Signed**
- **Unsigned**
- **Boolean**
- **Integer**

### Use Std_Logic Data Type for all Port Declarations

- Industry standard
  - Easy to interface to other models
  - Synthesis tools are motivated to support it
- Nine logic values are usually enough
- Automatically initialized to an unknown value
- Board level simulations do not need type conversions, which makes interfacing easier.
- User can integrate the synthesized netlist back into the design hierarchy without type conversions since **STD_LOGIC** is the output data type for synthesized designs

### Non-Synthesizable Semantics

- **Q <= 0 after XX ns;**
- **wait for XX ns;**

  Because: Synthesized circuit timing comes from the target library, not the source code.
Initialization

variable SUM: Integer := 0; -- Valid for simulation
-- Not valid for synthesis

Instead explicitly assign a value in your code:

process
variable SUM : Integer;
Begin
if reset = '0' then
  SUM := 0;
  . . .

Modeling Sequential Behavior

- In some languages specialized constructs are used to imply sequential behavior:
  - AHPL: <= (sequential) := (combinational)
  - Naming conventions can be employed, i.e. X CLK denotes a clock signal
- VHDL: By convention, specific language constructs are combined in specific ways to imply sequential behavior.

Concerns When Synthesizing Sequential Behavior

- Main concerns are:
  - Identification of the clocking control mechanism
  - Determination of what values have to be registered
  - Initializing the modeled circuit

Transition Functions

function Rising_Edge (signal S: Std_Ulogic) return Boolean is
begin
  return (S'event and (To_X01(S) = '1') and
    (To_X01(S'Last_Value) = '0'));
end Rising_Edge;

Synthesizable Clock Edge Detection Options

Rising Edge

- Rising_Edge(CLK)
- CLK'event and CLK='1'

Falling Edge

- Falling_Edge(CLK)
- CLK'event and CLK='0'

Synplicity Synthesis of Register with Synchronous Reset

Synthesis Rules

- Only 1 IF statement in process
- Only CLK in condition
- Only one clock edge in process
- CLK’event is redundant for simulation, but required for synthesis
Illegal Clock Edge Condition

if CLK'event and CLK='1' and ENABLE='1' then
...

Asynch_Reset: process (CLK, RESET) begin
if RESET='0' then
AREG <= '0';
elsif Rising_Edge(CLK) then
AREG <= A;
end if;
end process;

Synplicity Synthesis of Register with Asynchronous Reset

Synthesis Rules
Only 1 IF statement in process
Only CLK in condition
Only one clock edge in process
CLK'event is redundant for simulation, but required for synthesis

Gated Clocks

architecture BEHAVIORAL of GATE_CLOCK is
signal GATECLK: STD_LOGIC;
begin
GATECLK <= (IN1 and IN2 and CLK);
GATE_PR: process (GATECLK) begin
if (GATECLK'event and GATECLK='1') then
if (LOAD='1') then
OUT1 <= DATA;
end if;
end if;
end process;
end BEHAVIORAL;

Eliminating Gated Clocks

architecture BEHAV of CLOCK_ENABLE is
signal ENABLE: STD_LOGIC;
begin
ENABLE <= IN1 and IN2 and LOAD;
EN_PR: process (ENABLE,DATA,CLOCK) begin
if (CLOCK'event and CLOCK='1') then
if (ENABLE='1') then
DOUT <= DATA;
end if;
end if;
end process;
end BEHAV;
Synplicity Synthesis of Improved Model

Summary

- Synthesis semantics
  - Just because you can simulate your code doesn't mean you can synthesize it...
- Hardware inference
- Next time: More synthesis hints
- Reading: pp. 452-477