Basics of Synthesis

Simulation is great, but one of the foremost advantages of an HDL is its ability to create gate level designs thorough a different flavor compilation....synthesis.

We can take the previous example, and synthesize the VHDL code into a gate level design and represent it at a new structural VHDL netlist or a schematic.

We will not go into the details of how synthesis is done but lets see what happens anyway.

We usually synthesize VHDL designs using a script to direct the synthesis tool. Using a GUI to do this would be very time consuming.

Helpful Hint: Running a CAD tool is not like running a web browser. Learn to use scripts and command line interfaces.

What about this “Synthesis” thing? (cont.)

Here is a simple synthesis script for *elsyn* (a synthesis tool) that synthesizes our behavioral design for the aoi4 gate.

```
#simple synthesis script
set vhdl_write_component_package FALSE
set vhdl_write_use_packages {library ieee,adk; use
ieee.std_logic_1164.all; use adk.all;}
set edifout_power_ground_style_is_net TRUE
set sdf_write_flat_netlist TRUE
set force_user_load_values TRUE
set max_fanout_load 10


load_library ami05_typ


analyze src/aoi4.vhd     -format vhdl -work work
elaborate aoi4           -architecture data_flow -work work
optimize -ta ami05_typ -effort standard -macro -area


write ./edif/aoi4.edf -format edif
write ./vhdlout/aoi4.vhd -format vhdl


#to make a schematic do this in the edif directory
#edif2eddm aoi4.edf data_flow
```

What’s important to understand here?

```
load_library ami05_typ
```

The synthesis tool needs a known library of logic cells (gates) to build the synthesized design from.

```
analyze src/aoi4.vhd     -format vhdl -work work
```

Analyze (compile) the VHDL code and do initial processing.

```
elaborate aoi4           -architecture data_flow -work work
```

Create a generic gate description of the design.

```
optimize -ta ami05_typ -effort standard -macro -area
```

Map the generic gates to the “best” ones in the library ami05.

```
write ./edif/aoi4.edf -format edif
write ./vhdlout/aoi4.vhd -format vhdl
```

Write out the results in EDIF and VHDL formats.

How is the synthesis invoked?

The script is saved in a file called “script_simple”.

A work directory (if not already created) is created to put the compiled images by typing:

```
vlib work
```

Create the edif and vhdlout directories where the edif and VHDL netlist will be put.

```
mkdir edif
mldir vhdlout
```

Then, from the command line type:

```
elsyn
```

Eventually you get the prompt:

```
LEONARDO{1}:
```

Then type:

```
source script_simple
```

The tool *elsyn* reads the script file and executes the commands in the script.

What does the output look like?

The synthesis tool puts a synthesized version of the design in two directories, the vhdlout and edif directories. In the vhdlout directory:

```
--
-- Definition of  aoi4
--
--      Wed Jul 18 12:31:05 2001
--      Leonardo Spectrum Level 3, v20001a2.72
--

library ieee,adk; use ieee.std_logic_1164.all; use adk.all;

entity aoi4 is
   port (
      a : IN std_logic ;
      b : IN std_logic ;
      c : IN std_logic ;
      d : IN std_logic ;
      z : OUT std_logic) ;
end aoi4 ;

architecture data_flow of aoi4 is
   component aoi22
      port (
         Y : OUT std_logic ;
         A0 : IN std_logic ;
         A1 : IN std_logic ;
         B0 : IN std_logic ;
         B1 : IN std_logic) ;
   end component ;
begin
   ix13 : aoi22 port map ( Y=>z, A0=>a, A1=>b, B0=>c, B1=>d);
end data_flow ;
```

Examine the gate level VHDL

We see that the synthesized aoi4 looks much like what we initially wrote. The entity is exactly the same.

The architecture description is *different*. The design aoi4 is now described in a different way.

Under the architecture declarative section, a gate (aoi22) from the library was declared:

```
component aoi22
   port (
      Y : OUT std_logic ;
      A0 : IN std_logic ;
      A1 : IN std_logic ;
      B0 : IN std_logic ;
      B1 : IN std_logic) ;
end component ;
```

In the statement area, we see this gate is connected to the ports of the entity with a component instantiation statement.

```
ix13 : aoi22 port map ( Y=>z, A0=>a, A1=>b, B0=>c, B1=>d);
```

We will study component instantiation in more detail later.

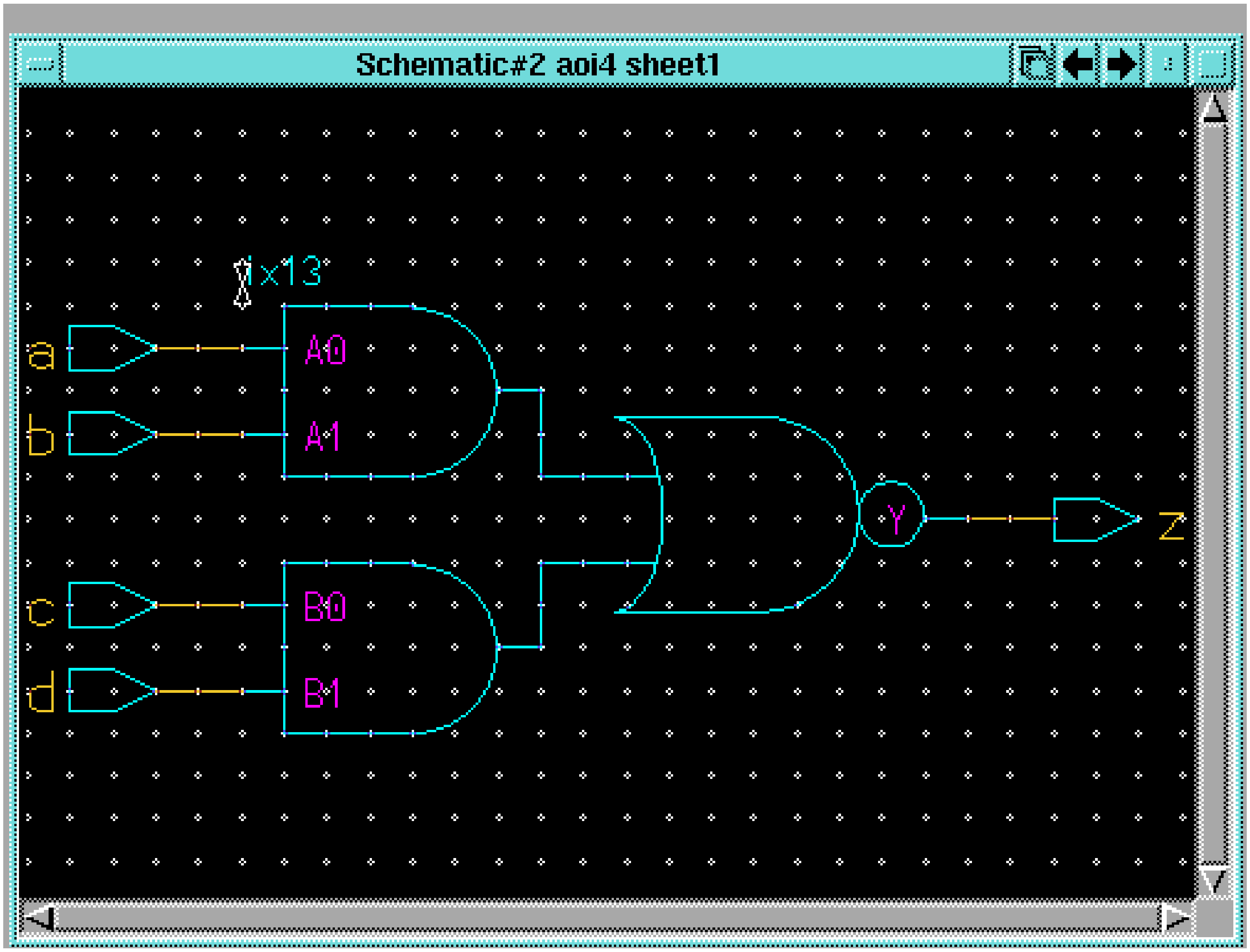
Note also, the intermediate signals temp1 and temp2 have optimized away.

Examine the schematic created by synthesis

The EDIF netlist is converted to a Mentor schematic by executing the command (in the edif directory):

```
edif2eddm  aoi4.edf  data_flow
```

When design architect is invoked upon the design we see the following:



Here we can see the direct correspondence between the gate pins and the entity pins in the statement:

ix13 : aoi22 port map (Y=>z, A0=>a, A1=>b, B0=>c, B1=>d);

The instance name (`ix13`) is also evident.

What you say is not what you get. (sometimes)

Looking at the VHDL code, one might expect something different.

```
BEGIN
  temp1 <= a AND b;
  temp2 <= c AND d;
  z     <= temp1 NOR temp2;
END data_flow;
```

This code seems to imply two AND gates feeding a NOR gate. However this is not the case. This description is a behavioral one. It does not in any way dictate what gates to use.

Two AND gates and a NOR gate would be a fine implementation, except for the fact that it is *slower, bigger,* and *consumes more power* than the single aoi22 gate.

The synthesis tool finds the “best” implementation by trying most possible implementations and choosing the optimum one.

What is a “best” implementation? Size, speed?