


CSCE 611

Introduction to Behavioral Logic Design



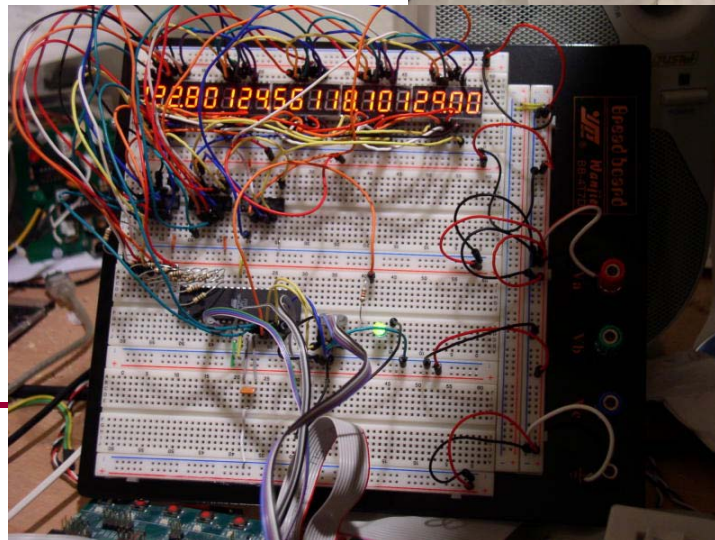
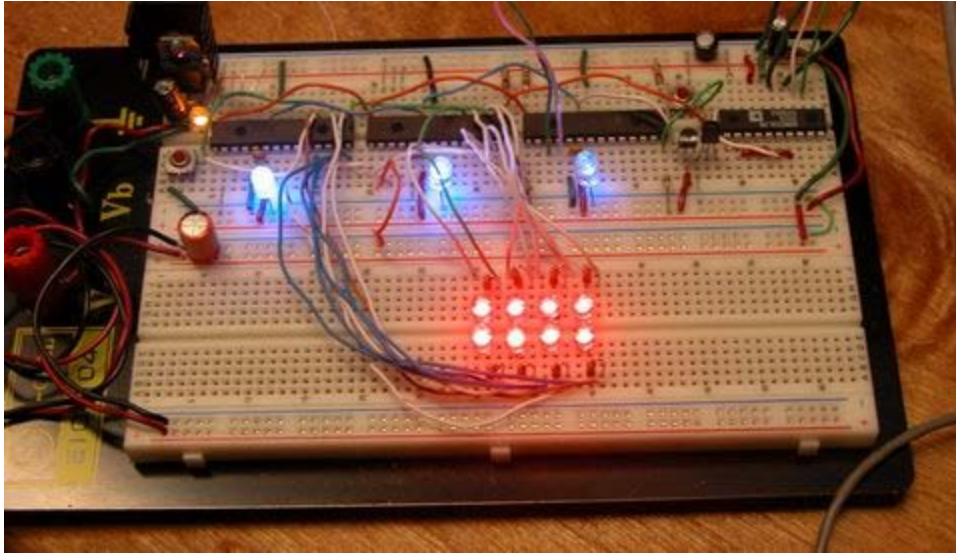
Instructor: Jason D. Bakos



Introduction to CSCE 611

- Teaching assistant:
 - Zheming Jin
 - Office: 3D15, back-left corner
 - email: jinz@email.sc.edu

Advanced Digital Design?



Introduction to CSCE 611

- Problems:
 1. How can we rapidly design large-scale digital systems?
 2. How can the designs be re-used and extended?
 3. How can we test/debug/verify designs?

Introduction to CSCE 611

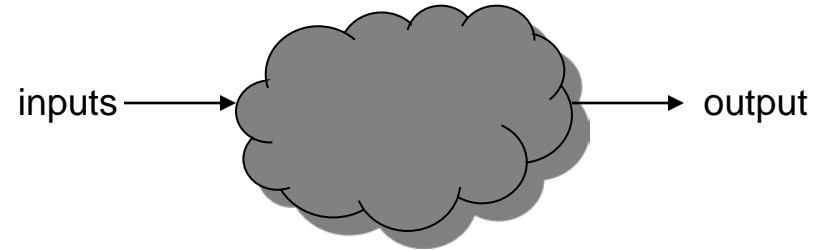
- Topics covered:
 1. Digital design using a modern industrial EDA/CAD flow
 2. Verification and test bench design
 3. Logic synthesis and implementation on FPGAs
 4. MIPS instruction set architecture
 5. Microarchitecture design
 6. Memory interfacing and I/O

Hardware Description Language

- Digital logic is either:

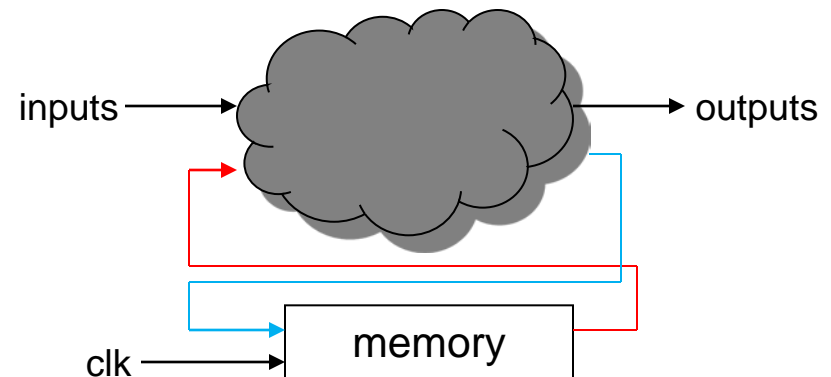
- Combinational

- $F(\text{inputs}) = \text{output}$



- Sequential

- $F(\text{inputs}, \text{input history}) = \text{output}$



Hardware Description Language

- Need a convenient way to specify logic *structure* and *behavior* using a high-level language
 - Boolean logic, simple arithmetic, registers
- Example 1:
 - Inputs: A, B, C
 - Outputs: Z
 - Description:
 - temp1 = A xor B
 - temp2 = temp1 + C
 - Z = temp2 * 12
- Example 2:
 - Inputs: A, B
 - Outputs: Z
 - Description:
 - on a rising clock edge...
 - temp1 = A * last_value_of_Z
 - Z = temp1 + B



VHDL

- HDL => VHDL / Verilog
- VHDL more verbose and self-documenting
- Not case-sensitive

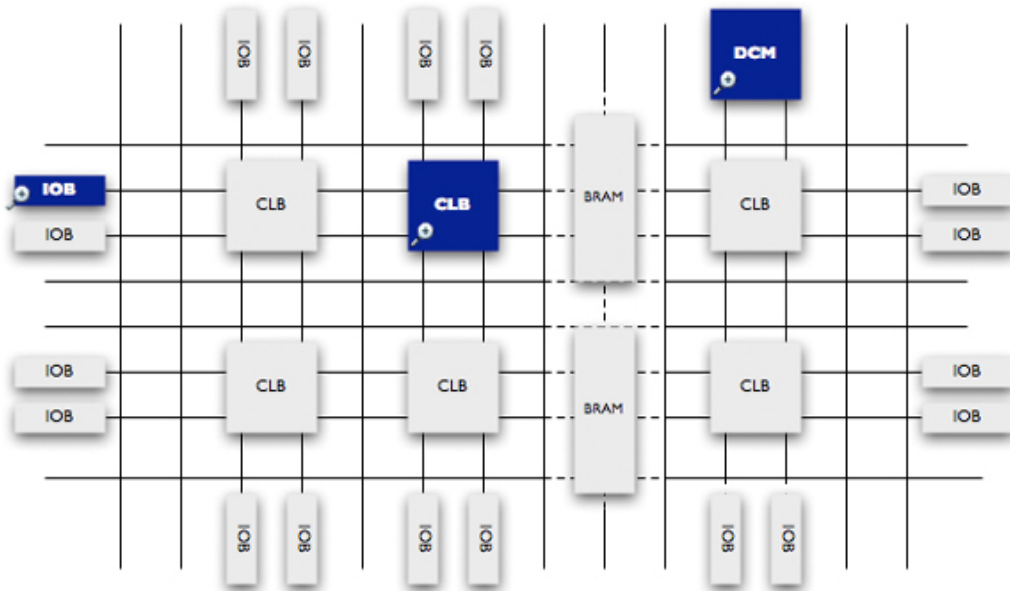
- VHDL => "VHSIC Hardware Description Language"
- VHSIC => "US DoD Very-High-Speed Integrated Circuit"
- DoD project
 - Document behavior of ASICs from suppliers
 - Alternative to manuals

- Used to describe behavior of digital logic
 - Extensions for analog
- High-level programming language, subset of Ada
 - Also looks like Pascal
- IEEE standards: 1987, 1993, 2000, 2002

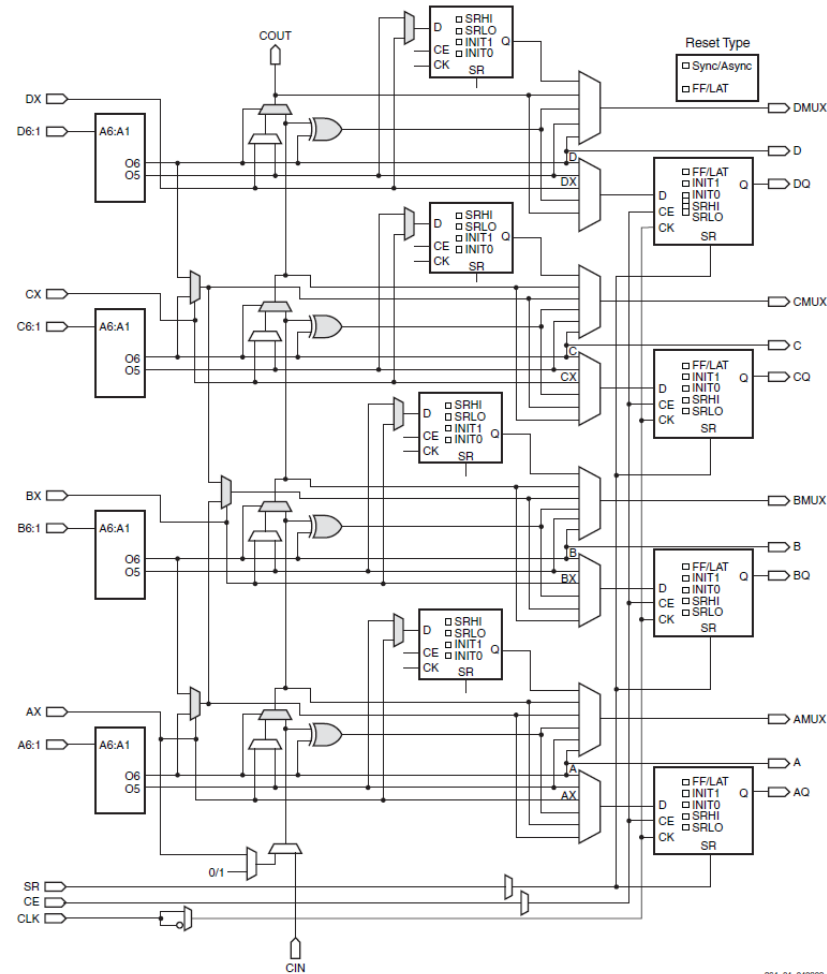
- First came the language...
- ...next came simulators...
- ...then came synthesizers (FPGA and ASIC)



Field Programmable Gate Arrays



- CLBs (2 slices)
- Block RAMs
- DCMs
- IOB
- DSP48s
- PCIe



vg364_04_042209

Sequential vs. Concurrent Semantics

- Problem:

- Programming languages with sequential semantics:

- Assume $B=0$, $C=5$

- ```
A = B + C
print A (output is 5)
B = C
print B (output is 5)
```

- Hardware is concurrent

- Each line of code executes concurrently (no ordering)

- ```
A = B + C
print A (output is 10)
B = C
print B (output is 5)
```

- Problem:

- ```
A = B + C
B = A
```

Creates an feedback loop; will cause simulation to fail (cannot converge)



---

# VHDL

---

- By its nature, VHDL is
  - Self-documenting
  - Allows for easy testbench design (simulators, instruments)
- Any VHDL code may be simulated
- Only some VHDL codes may be synthesized
  - Depends on packages, data types, and constructs
- VHDL descriptions (programs) have structure similar to C++
- Each design (component) is made up of
  - Entity section
    - Component interface (I/O)
    - Analogous to C++ header (public methods only)
  - Architecture section
    - Contains behavior (implementation)
    - Can have multiple architectures for any entity
    - Example: different types of adders with consistent interfaces



# Logic Synthesis

- Behavior:
  - $S = A + B$
  - Assume A is 2 bits, B is 2 bits, C is 3 bits

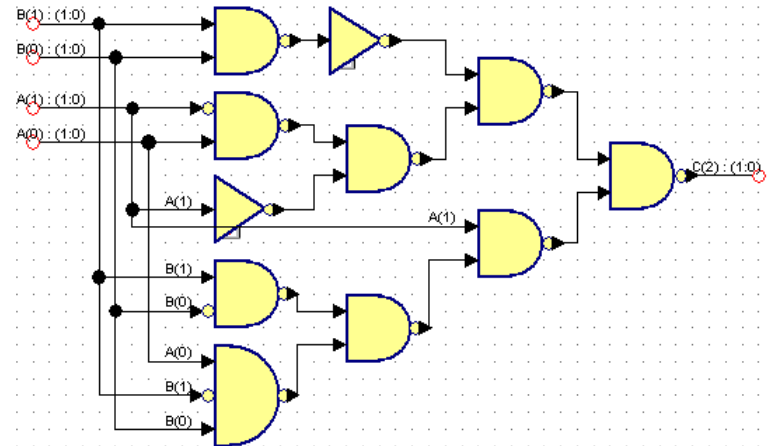
| A      | B      | C       |
|--------|--------|---------|
| 00 (0) | 00 (0) | 000 (0) |
| 00 (0) | 01 (1) | 001 (1) |
| 00 (0) | 10 (2) | 010 (2) |
| 00 (0) | 11 (3) | 011 (3) |
| 01 (1) | 00 (0) | 001 (1) |
| 01 (1) | 01 (1) | 010 (2) |
| 01 (1) | 10 (2) | 011 (3) |
| 01 (1) | 11 (3) | 100 (4) |
| 10 (2) | 00 (0) | 010 (2) |
| 10 (2) | 01 (1) | 011 (3) |
| 10 (2) | 10 (2) | 100 (4) |
| 10 (2) | 11 (3) | 101 (5) |
| 11 (3) | 00 (0) | 011 (3) |
| 11 (3) | 01 (1) | 100 (4) |
| 11 (3) | 10 (2) | 101 (5) |
| 11 (3) | 11 (3) | 110 (6) |

$$C_2 = \overline{A_1}A_0B_1B_0 + A_1\overline{A_0}B_1\overline{B_0} + A_1\overline{A_0}B_1B_0 + A_1A_0\overline{B_1}B_0 + A_1A_0B_1\overline{B_0} + A_1A_0B_1B_0$$

$$C_2 = B_1B_0(\overline{A_1}A_0 + A_1\overline{A_0} + A_1A_0) + A_1B_1\overline{B_0}(\overline{A_0} + A_0) + A_1A_0\overline{B_1}B_0$$

$$C_2 = B_1B_0(\overline{A_1}A_0 + A_1(\overline{A_0} + A_0)) + A_1B_1\overline{B_0} + A_1A_0\overline{B_1}B_0$$

$$C_2 = B_1B_0(\overline{A_1}A_0 + A_1) + A_1(B_1\overline{B_0} + A_0\overline{B_1}B_0)$$



---

# Entity / Architecture

---

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

entity buffer is
 port (
 a:in std_logic_vector(3 downto 0);
 y:out std_logic_vector(3 downto 0)
);
end;

architecture my_hypernifty_buffer of buffer is
 signal int_a : std_logic_vector(3 downto 0);
begin
 int_a <= not a;
 y <= not int_a;
end;
```



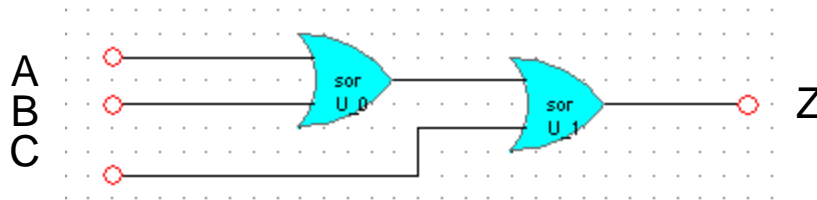
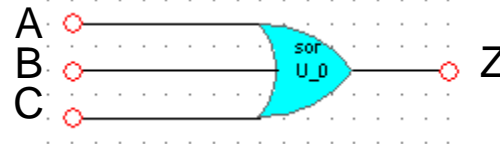
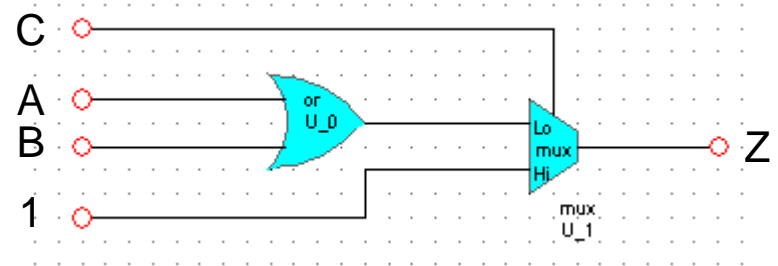
# VHDL Semantics

- Behavioral VHDL is fundamentally based on the *concurrent assignment statement*

- Example:**

```
Z <= A or B when C = '0' else
 '1';
```

| A | B | C | Z |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 |



---

# Constructs in Behavioral VHDL

---

- Concurrent assignment statement

```
[output] <= [function of inputs] after [delay] when [condition] else
 [function of inputs] after [delay] when [condition] else
 ...
 [function of inputs];
```

- Example:

```
out <= A and B when sel="00" else
 A or B when sel="01" else
 A nor B when sel="10" else
 A xor B;
sel <= "00" when (C or D)="0101" else
 "10";
```



---

# Data Types

---

- In this course, you will only use 2 data types
  - **std\_logic**
    - Represents a bit signal
    - Enumerated type: (1, 0, X, U, Z, -, H, L, W)
    - 1, 0 are logic values
    - X is “don’t know” – shorted (double-driven) signals
    - U is “unassigned” – un-initialized FF/register/memory
    - Z is “high-impedence” – for tristated/floating outputs
    - - is “don’t care” – for outputs, helps synthesizer minimize logic
    - H is weak 1, L is weak 0, W is weak unknown (not typically used)
  - Use ‘1’ to represent constant
  - **std\_logic\_vector**
    - Array of std\_logic
    - Represents a “bus” signal
  - Use “11” to represent constant



---

# Structural vs. Behavioral VHDL

---

- Structural VHDL
  - Resembles a netlist
    - Defines instantiated components
    - Interconnects
  - May contain library subroutine calls, operators, mux behavior
  - Can be directly (and easily) synthesized
- Behavioral VHDL
  - Defines how outputs are computed as function of inputs
  - Concurrent assignment statement
  - Process
    - Looks like a programming language
    - Internally has sequential semantics (but as a unit behaves like concurrent assignment statement)
    - Sensitivity list
    - Process block implements concurrent assignment
    - May contain variables
    - Constructs: if-then, for-loop, while-loop, inf-loop
    - Difficult to synthesize
    - Not synthesizable: timed waits, file I/O, some loop structures



---

# Priority

---

```
out <= A and B when sel="00" else
 A or B when sel="01" else
 A nor B when sel(1)='1' else
 A xor B;
```

- What's the problem with the above statement?

---

# Processes

---

- Complex concurrent assignment statement

```
out <= A when sel='0' else
 B;
```

```
process (A,B,sel)
begin
 if sel='0' then
 out <= A;
 else
 out <= B;
 end if;
end
```



---

# Processes

---

```
process (sel)
begin
 if sel='0' then
 out <= A;
 else
 out <= B;
 end if;
end
```

implies that A and B are registered when sel changes



---

# Processes

---

```
process (A,B,sel)
begin
 if sel='0' then
 out <= A;
 end if;
end
```

implies that out is  
registered



---

# Constructs in Process VHDL

---

- if-statement

```
if a="01" then
 y <= b;
elsif a="11" then
 y <= not(b)+1;
else
 y <= "0000";
end if;
```

- Loops

```
loop
 <statements>
end loop;
```

```
for i in 0 to 15 loop
 <statements>
end loop;
```

```
while <condition> loop
 <statements>
end loop;
```



# Example Process

```
-- right-shift arithmetic for 8-bit signed integer
rsa: process (a, shamt)
variable fill : std_logic_vector(3 downto 0);
variable temp : std_logic_vector(7 downto 0);
begin
 for i in 0 to 3 loop
 fill(i):='1' and a(7);
 end loop;
 if shamt(0)='1' then
 temp := fill(0) & a(7 downto 1);
 end if;
 if shamt(1)='1' then
 temp := fill(1 downto 0) & temp(7 downto 2);
 end if;
 if shamt(2)='1' then
 out <= fill(3 downto 0) & temp(7 downto 4);
 end if;
end process;
```

**signal assignments**  
(state changes) take effect only after entire process completes "execution"

**variable assignments**  
take effect immediately



---

# Memory

---

- Memory is inferred:

```
-- 8-bit rising-edge register with asynchronous reset
reg8 : process(clk,rst)
begin
 if rst='1' then
 q <= "00000000";
 elseif rising_edge(clk) then
 if en='1' then
 q <= d;
 end if;
 end if;
end process;
```



---

# Example Process

---

```
process (clk,rst)
begin
 if rst='1' then
 count <= (others=>'0');
 row_Id_In <= (others=>'1');
 elsif clk'event and clk='1' then
 if count = "0000000" then
 if random(3 downto 0) /= "0001" and random(3 downto 0) /= "0000" then
 count <= "000" & random(3 downto 0);
 else
 count <= "0000010";
 end if;
 row_Id_In <= row_Id_In + "1";
 report "set change";
 else
 count <= count - "1";
 end if;
 end if;
end process;
```



---

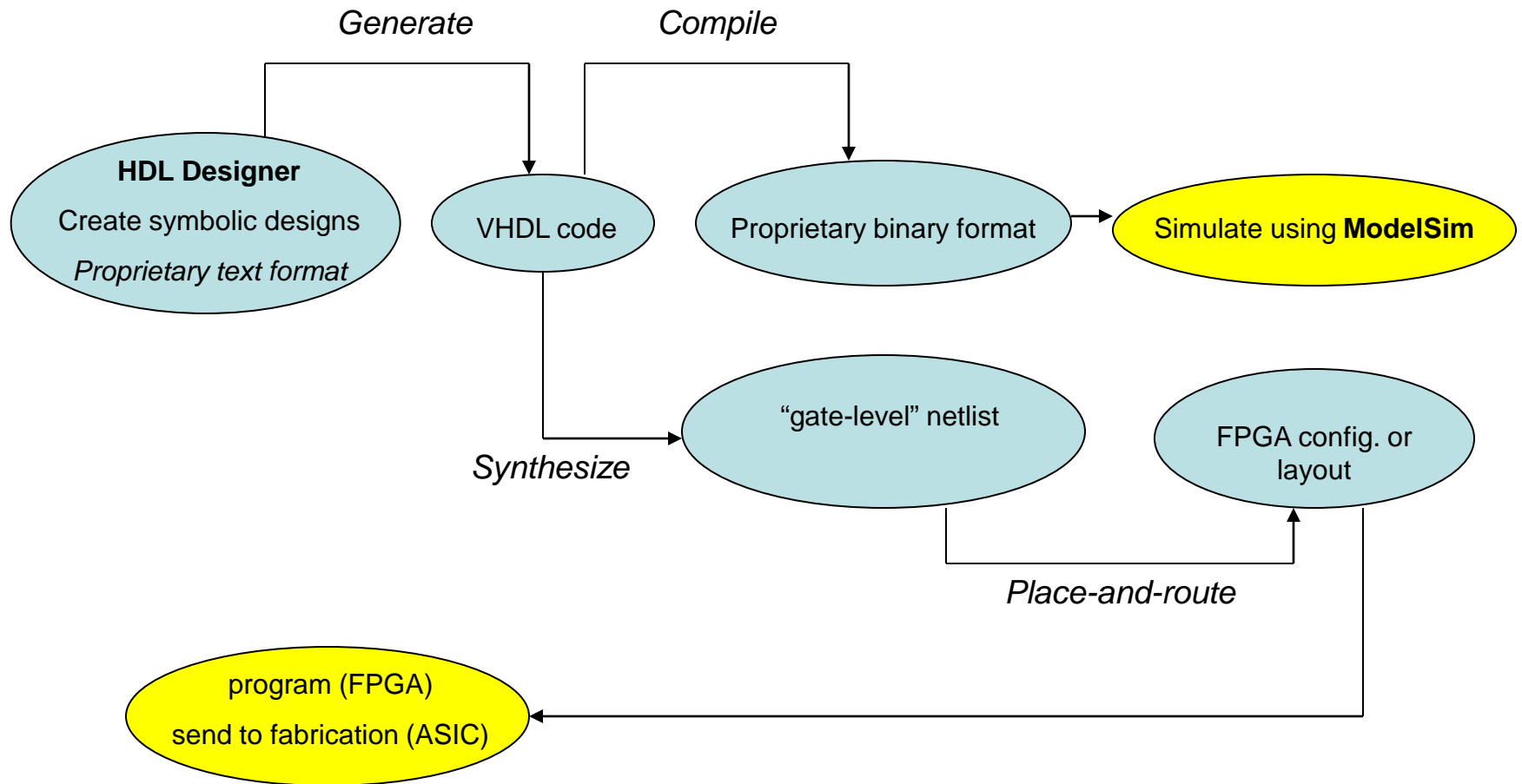
# HDL Designer

---

- Allows for rapid VHDL development
  - graphical design entry
  - generated VHDL
  - automated design flows
  
- Views
  - Block diagram
  - State machine
  - Truth table
  - Flow chart
  - VHDL view (combined or architecture-only)
  - Symbol



# HDL Designer Design Flow



---

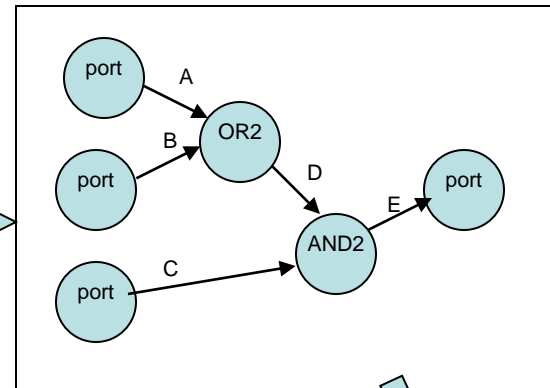
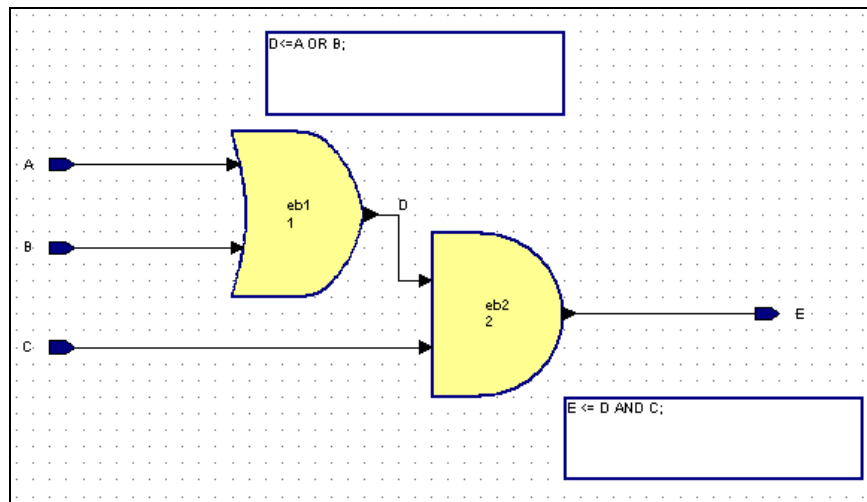
# Synthesis

---

- Idea:
  - “Compile” VHDL into a cell-level *netlist*
    - A netlist is a graph
      - Vertices represent cells (such as gates, latches, etc.)
      - Edges represent interconnection wires
    - To do this, we need
      - VHDL
      - Cell library
  - Place-and-route netlist onto FPGA/ASIC
    - To do this, we need
      - Netlist
      - CLB specification and routing matrix (FPGA)
      - Cell layouts and design rules (ASIC)
    - Output is:
      - Configuration bitmap (FPGA)
      - Layout (ASIC)



# Netlists



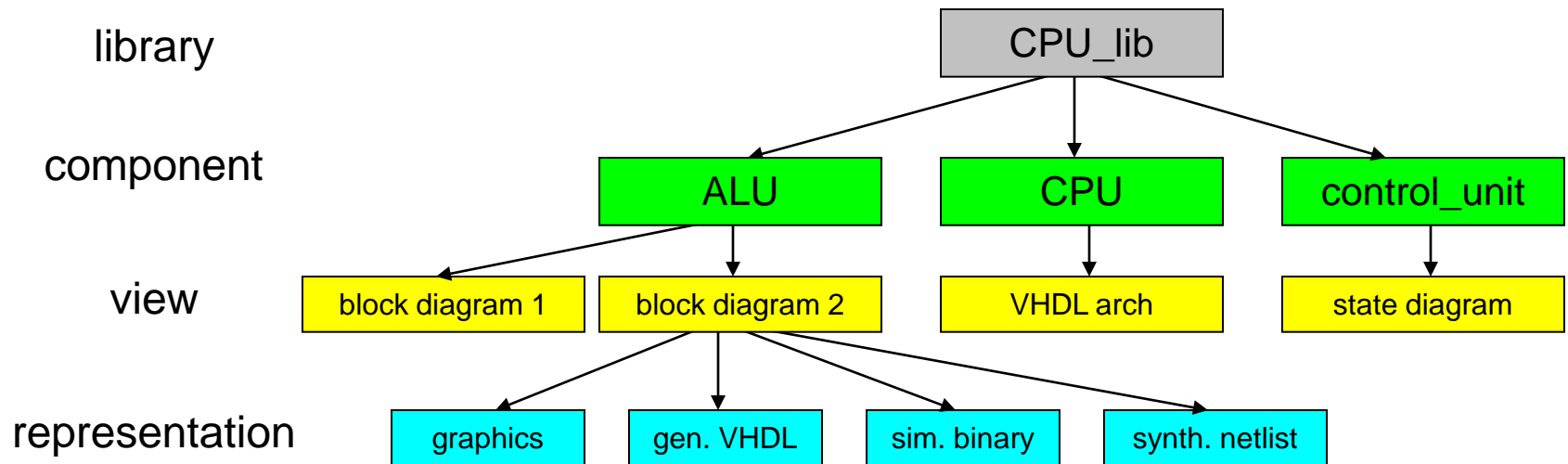
Note that AND2  
and OR2 are  
technology cells

```
instance OR2 as ix1
instance AND2 as ix3

ix1 A B D
ix3 D C E
```

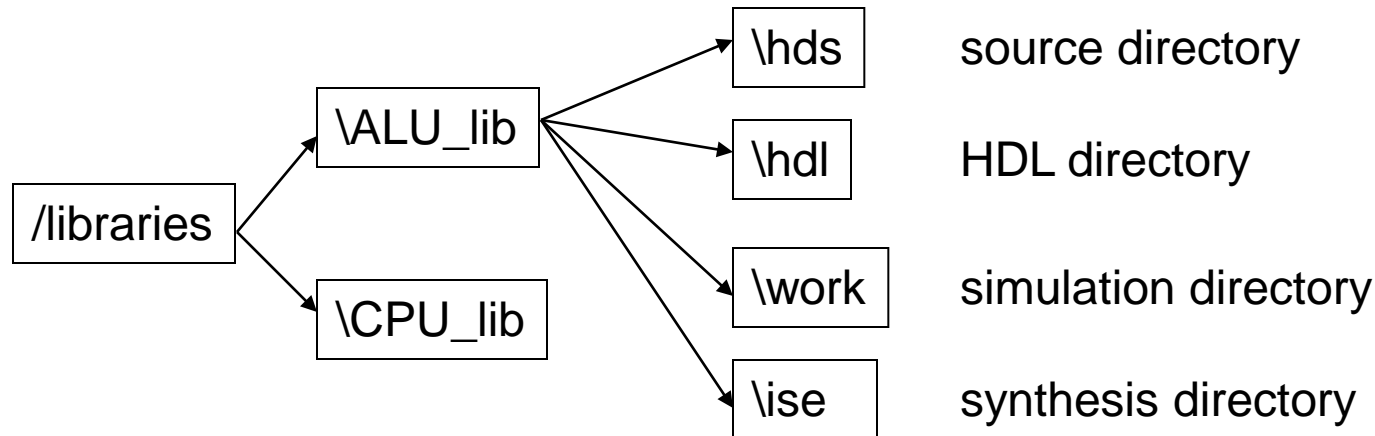
# Libraries in HDL Designer

- A library is a collection of components
  - Components have one or more *views* (implementations)
    - Block diagram, truth table, flow chart, state machine, VHDL architecture
  - Each view has representations:
    - Graphics, VHDL, simulator netlist, synthesis netlist



# Libraries in HDL Designer

- Libraries are stored in four subdirectories

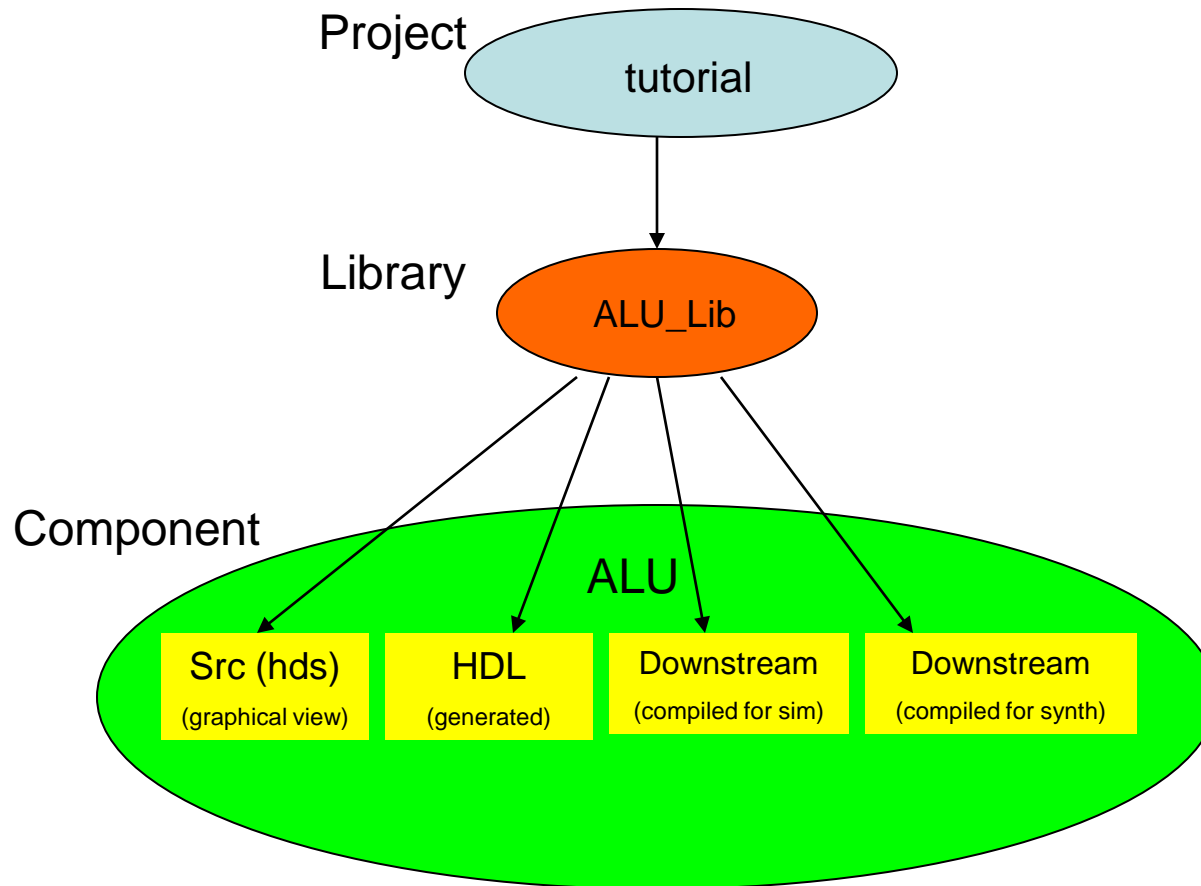


- For each library you use or create, *library mappings* to these directories must be specified
- The mappings for your set of libraries are stored in your project file

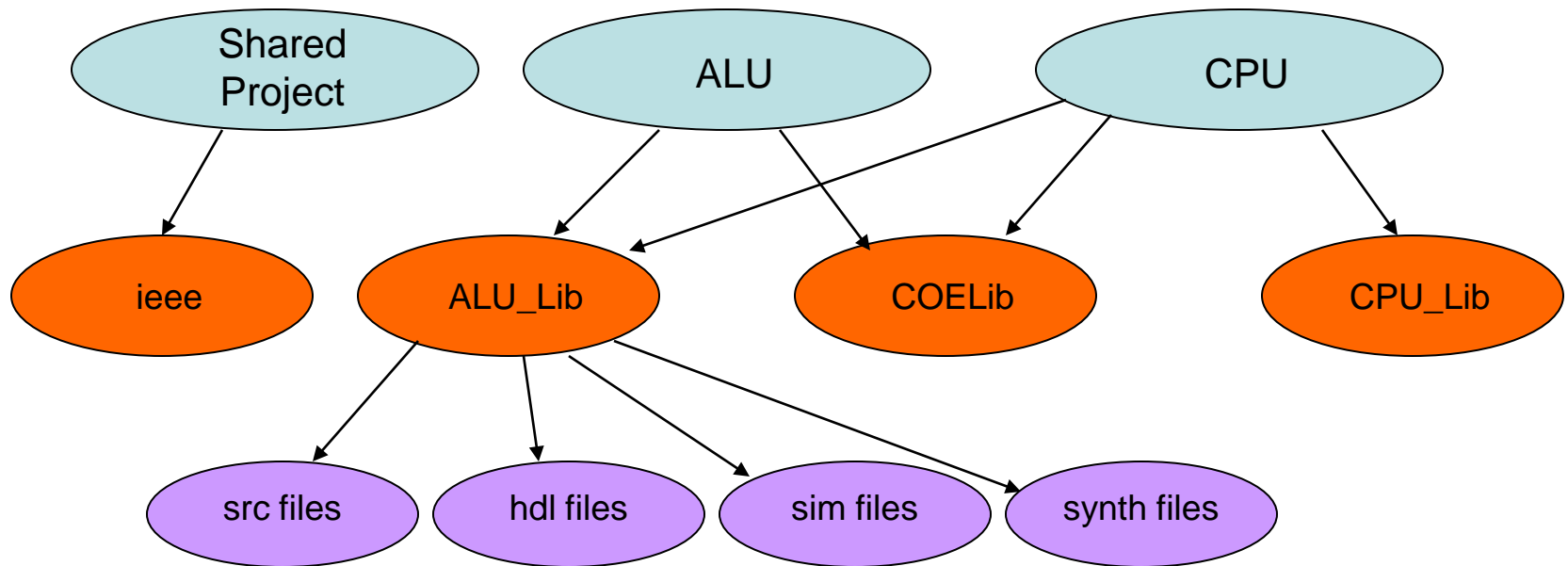


# Projects

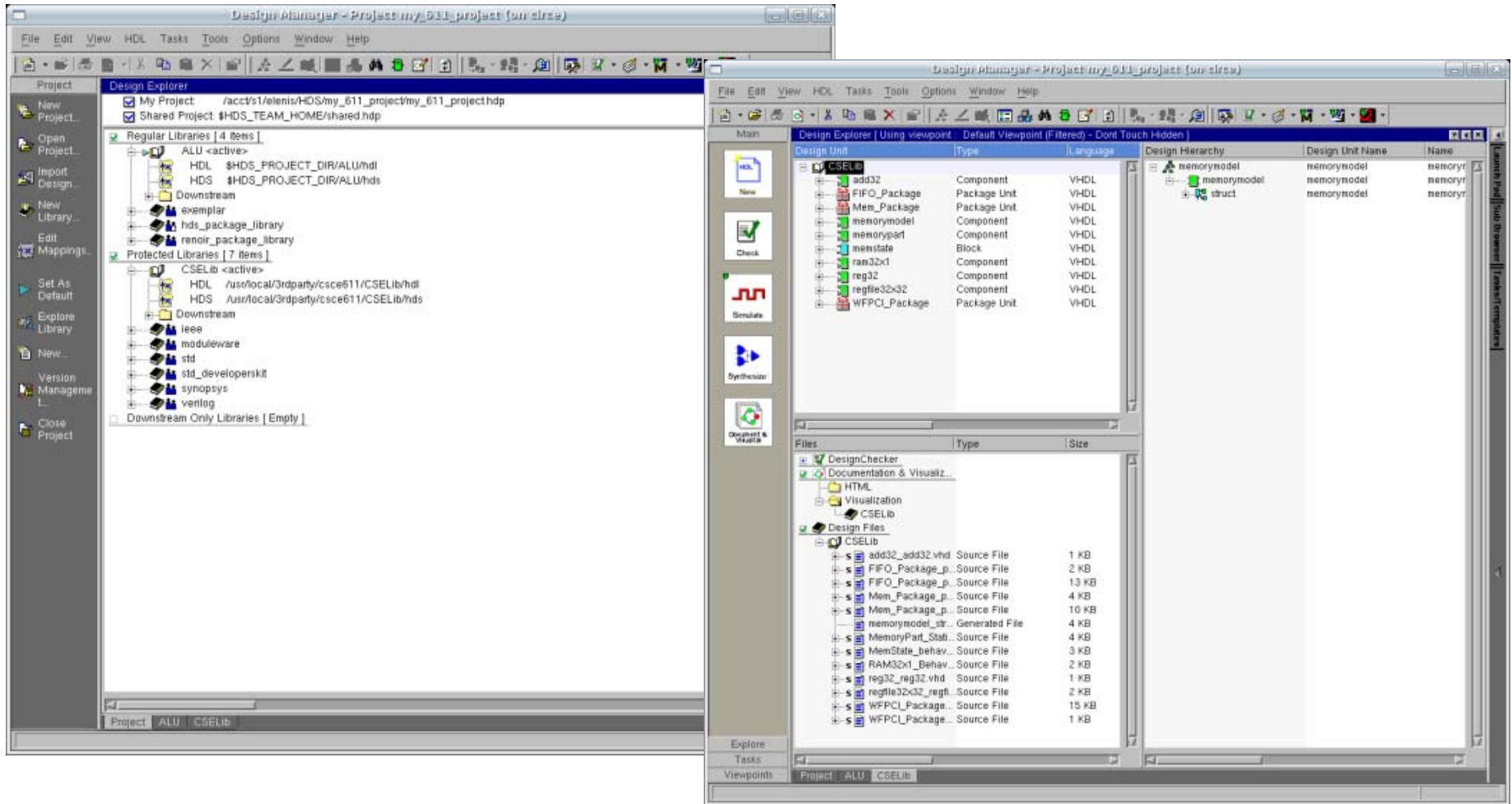
- Projects are a collection of library mappings



# Projects, Libraries, Files

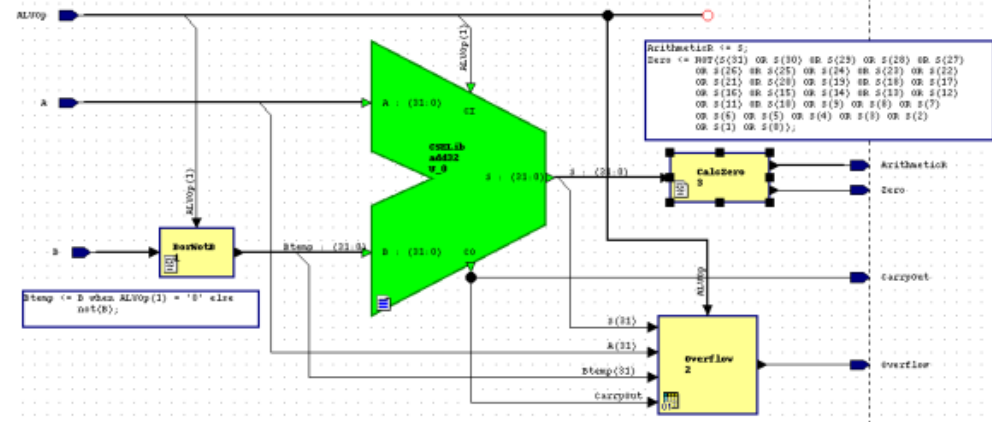
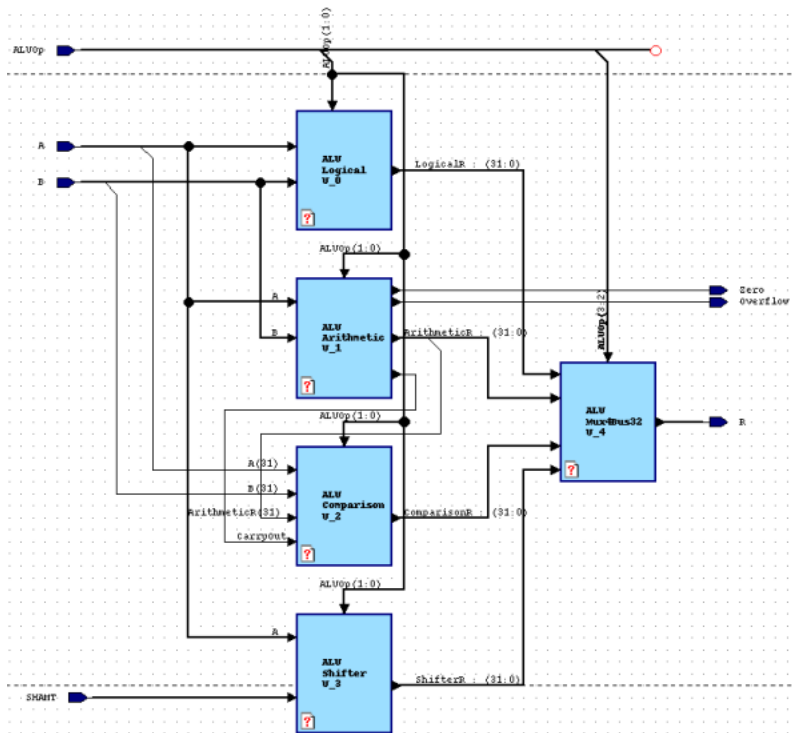


# HDL Designer GUI

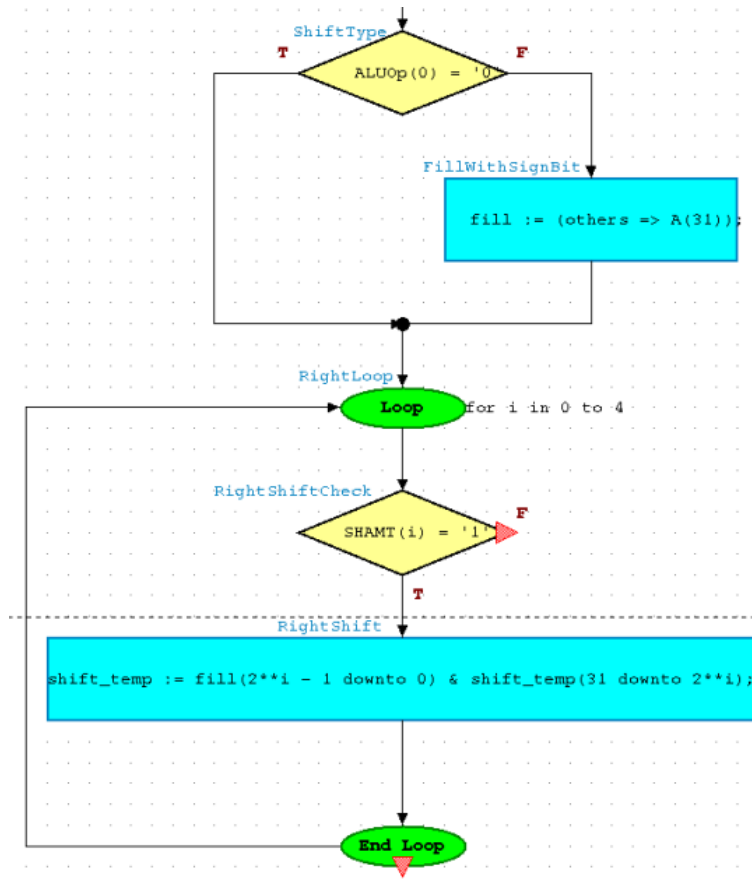




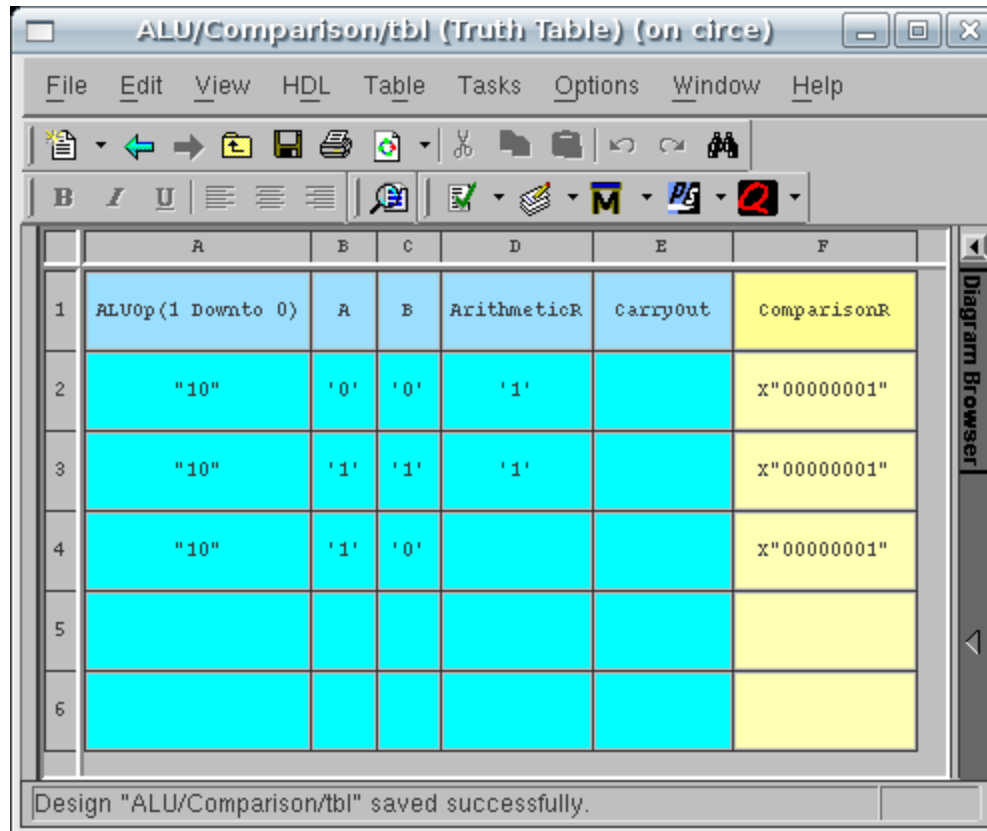
# Block Diagram Editor



# Flowchart Editor



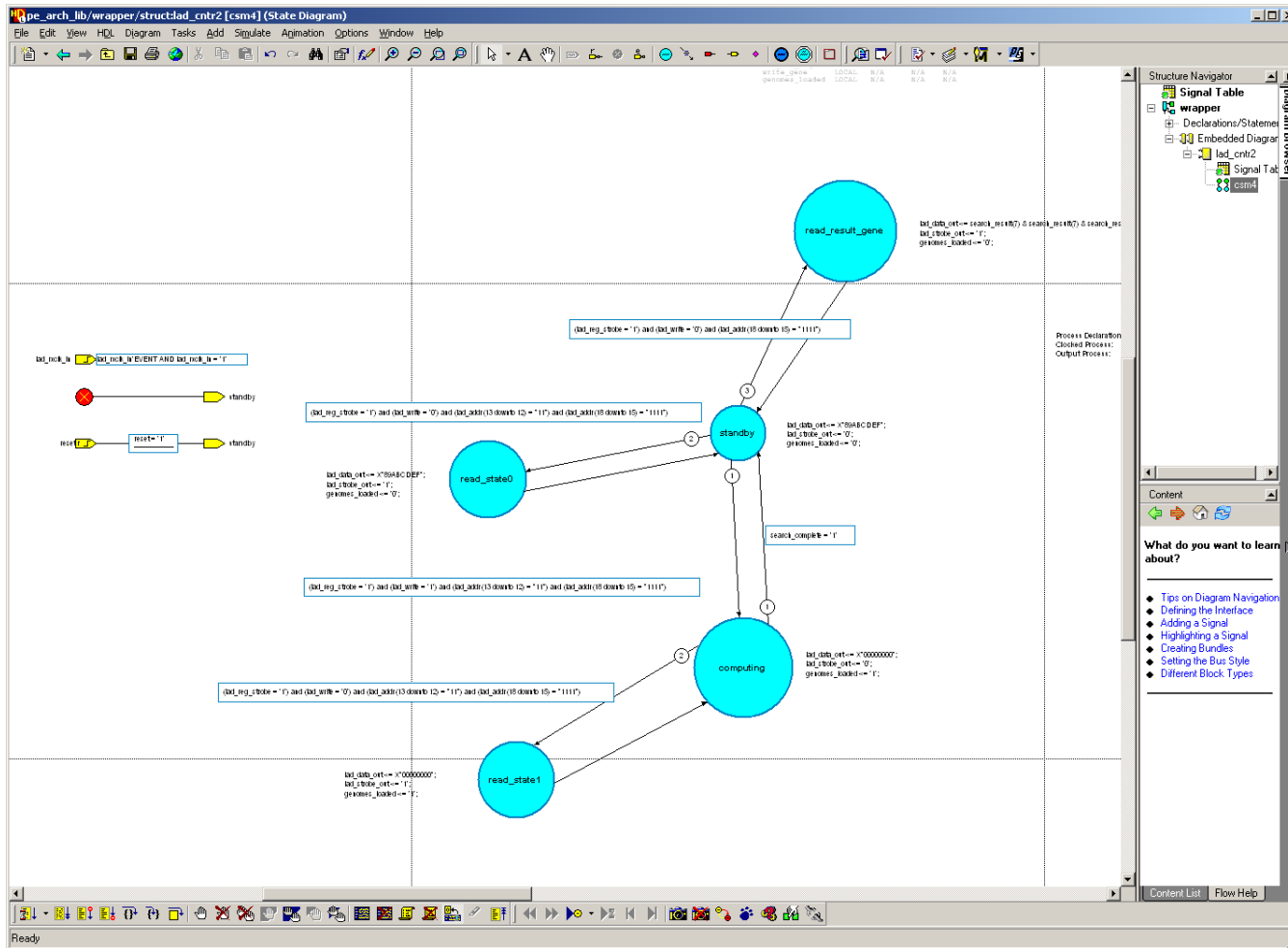
# Lookup Table Editor



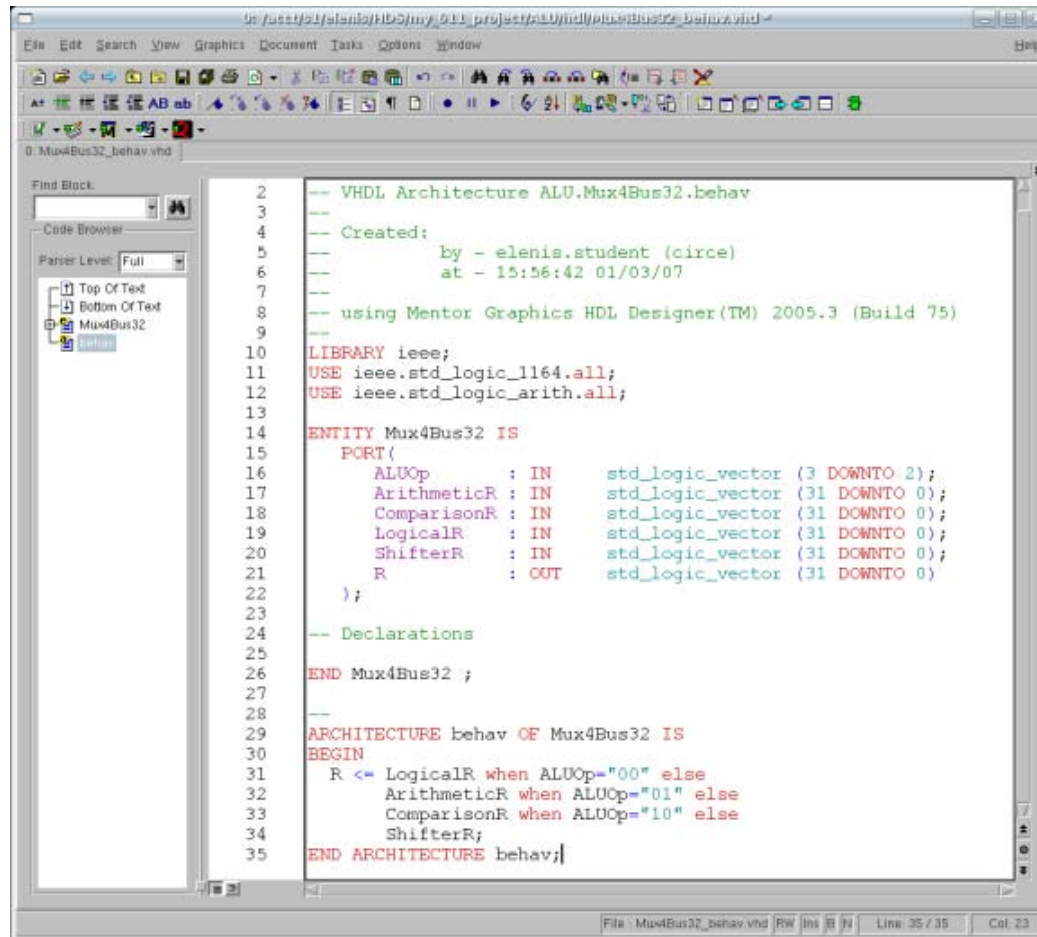
|   | A                 | B   | C   | D           | E        | F           |
|---|-------------------|-----|-----|-------------|----------|-------------|
| 1 | ALUOp(1 Downto 0) | A   | B   | ArithmeticR | CarryOut | ComparisonR |
| 2 | "10"              | '0' | '0' | '1'         |          | x"00000001" |
| 3 | "10"              | '1' | '1' | '1'         |          | x"00000001" |
| 4 | "10"              | '1' | '0' |             |          | x"00000001" |
| 5 |                   |     |     |             |          |             |
| 6 |                   |     |     |             |          |             |

Design "ALU/Comparison/tbl" saved successfully.

# State Machine Editor



# VHDL Editor



The screenshot shows a VHDL editor window titled "On justjo1@elena2/HDC/my\_011\_project/ALU/hdl/mux4Bus32\_behav.vhd". The window contains a menu bar (File, Edit, Search, View, Graphics, Document, Tasks, Options, Window) and a toolbar with various icons. The main text area displays the following VHDL code:

```
2 -- VHDL Architecture ALU.Mux4Bus32.behav
3 --
4 -- Created:
5 -- by - elena2.student (circe)
6 -- at - 15:56:42 01/03/07
7 --
8 -- using Mentor Graphics HDL Designer(TM) 2005.3 (Build 75)
9 --
10 LIBRARY ieee;
11 USE ieee.std_logic_1164.all;
12 USE ieee.std_logic_arith.all;
13
14 ENTITY Mux4Bus32 IS
15 PORT(
16 ALUOp : IN std_logic_vector (3 DOWNTO 2);
17 ArithmeticR : IN std_logic_vector (31 DOWNTO 0);
18 ComparisonR : IN std_logic_vector (31 DOWNTO 0);
19 LogicalR : IN std_logic_vector (31 DOWNTO 0);
20 ShifterR : IN std_logic_vector (31 DOWNTO 0);
21 R : OUT std_logic_vector (31 DOWNTO 0)
22);
23
24 -- Declarations
25
26 END Mux4Bus32 ;
27
28 --
29 ARCHITECTURE behav OF Mux4Bus32 IS
30 BEGIN
31 R <= LogicalR when ALUOp="00" else
32 ArithmeticR when ALUOp="01" else
33 ComparisonR when ALUOp="10" else
34 ShifterR;
35 END ARCHITECTURE behav;|
```

The status bar at the bottom indicates "File : Mux4Bus32\_behav.vhd RW Ins B N Line : 35 / 35 Col : 23".

# Components

- Library components can be instantiated in other designs
  - Shown as green blocks
    - For bottom-up design
  - Libraries also contain "blocks"
    - Attached to the design they were created in
    - Shown as blue blocks
    - For top-down design
  - Embedded blocks – embedded into block diagram
    - Shown as yellow blocks
    - Embeds behavior into structure
  - ModuleWare
    - Allows you to add integrated parameterized behaviors
    - registers, memories, gates, muxes, decoders, etc.

