



---

# ***A Tutorial on VHDL Synthesis, Place and Route for FPGA and ASIC Technologies***

Anup Gangwar

Embedded Systems Group,  
Department of Computer Science and Engineering,  
Indian Institute of Technology Delhi

*<http://embedded.cse.iitd.ernet.in>*

October 4, 2004



# Outline

---

- ▶ Overview
- ▶ VHDL Coding Styles
- ▶ FPGA Synthesis, Place and Route
- ▶ Demo of FPGA Synthesis, Place and Route
- ▶ ASIC Synthesis, Place and Route
- ▶ Demo of ASIC Synthesis, Place and Route
- ▶ Conclusions and Further Reading



# Overview

---

- ▶ VHDL is not a programming language
- ▶ What you can expect to learn:
  - Good VHDL coding styles
  - Overview of available tools
  - A typical design flow and scripting for automation
- ▶ Demos are planned but will only be shown if time permits
- ▶ All the scripts are in `~help/syntut`



# Outline

---

- ▶ Overview
- ▶ VHDL Coding Styles
- ▶ FPGA Synthesis, Place and Route
- ▶ Demo of FPGA Synthesis, Place and Route
- ▶ ASIC Synthesis, Place and Route
- ▶ Demo of ASIC Synthesis, Place and Route
- ▶ Conclusions and Further Reading



# *The Two Process Model*

---

```
architecture rtl of accu is
signal reg      : std_logic_vector( 7 downto 0 );
begin
    o <= reg;
    process( clk, resetn, i )
    begin
        if( resetn = '0' ) then
            reg <= (others => '0');
        elsif( clk'event and clk = '1' ) then
            reg <= reg + i;
        end if;
    end process;
end rtl;
```



# *The Two Process Model (contd...)*

.....

```
o <= reg;
process( i, reg )                --Combinational
begin
    tmp <= reg + i;
end process;
process( clk, resetn )           --Registers
begin
    if( resetn = '0' ) then
        reg <= (others => '0');
    elsif( clk'event and clk = '1' ) then
        reg <= tmp;
    end if;
end process;
```

.....



## ***The Two Process Model (contd...)***

---

- ▶ Dataflow model is difficult to understand
- ▶ Sequential parts of the code are easily identified
- ▶ Easy to understand and maintain
- ▶ Does Not compromise efficiency
- ▶ Simplifies debugging



# ***Making Use of Records***

---

- ▶ Records are analogous to structures in C
- ▶ Simplify port declarations
- ▶ Easier to add a new port to an entity





# ***Making Use of Records (contd...)***

---

## **Example:**

```
entity accu is
    port (
        clk      : in std_logic;
        resetn    : in std_logic;
        i         : in std_logic_vector( 7 downto 0 );
        o         : out std_logic_vector( 7 downto 0 ));
end accu;
```

To add a new port all modules using accu will need to be modified



# ***Making Use of Records (contd...)***

---

## **Package Declaration:**

...

```
type accu_in_type is record
```

```
    i      : std_logic_vector(7 downto 0);
```

```
end record;
```

```
type accu_out_type is record
```

```
    o      : std_logic_vector(7 downto 0);
```

```
end record;
```

...



# ***Making Use of Records (contd...)***

---

## **Entity Declaration:**

```
entity accu is
    port (
        clk      : in std_logic;
        resetn    : in std_logic;
        i         : in accu_in_type;
        o         : out accu_out_type);
end accu;
```

Addition of new port does not change entity declaration



# Writing Parameterized VHDL

---

```
entity accu is
generic(
    width    : integer := 8);
port (
    clk      : in std_logic;
    resetn   : in std_logic;
    i        : in std_logic_vector( width downto 0 );
    o        : out std_logic_vector( width downto 0 ));
end accu;
```



## Writing Parameterized VHDL (contd...)

...

```
ACCU: if (ACCUTYPE = ONE) generate
    ACCU_TYP_ONE: accu_one
        generic map(
            width      => DATAWIDTH,
        port map(
            clk        => clk,
            resetn     => resetn,
            i          => i,
            o          => o);
    end generate ACCU_TYP_ONE;
```

...

DATAWIDTH and ACCUTYPE are defined globally at one place making it easy to change configuration



# ***Under specified FSM and Inferred Latches***

---

- ▶ Under specified FSM means that values will be held on wires which would lead to latches being inferred by the synthesis tool
- ▶ Problems with inferred latched:
  - Verification tools will fail to match functionality
  - Unnecessary logic will get generated
  - The FSM might enter undefined state and get stuck i.e. incorrect functionality



# *Under specified FSM and Inferred Latches*

## **Example:**

```
...
architecture rtl of dummy is
    type states is ( free, working );
    signal curr_state, next_state : states;
begin
    process( curr_state )
    begin
        case curr_state is
            when free => if( inp = '1') then
                            next_state <= working;
                        end if;
        end case;
    end process;
    ...
end
```



## *Under specified FSM (contd...)*

---

...

```
process( clk, resetn )
```

```
begin
```

```
    if( resetn = '0' ) then
```

```
        curr_state <= free;
```

```
    elsif( clk'event and clk = '1' ) then
```

```
        curr_state <= next_state;
```

```
    end if;
```

```
end process;
```

...

Latch inferred for next\_state





## *Under specified FSM (contd...)*

### **Example:**

...

architecture rtl of dummy is

    type states is ( free, working );

    signal curr\_state, next\_state : states;

begin

process( curr\_state, inp )

begin

    case curr\_state is

        when free     => if( inp = '1') then

                    next\_state <= working;

                end if;

        when others => next\_state <= working;

    end case;

end process;



## *Under specified FSM (contd...)*

---

- ▶ FSM is now fully specified
- ▶ Using case statements for FSM is better than if-else as muxes are generated in first case and priority hardware in second
- ▶ Priority hardware makes the circuit slow



# ***Synchronous and Asynchronous Resets***

<b>Synchronous Reset</b>	<b>Asynchronous Reset</b>
<p>Large number of gates</p> <p>Reset pulse should be large enough for clock to become active</p> <p>No metastability problems</p> <p>Slow</p> <p>Consumes more power</p> <p>Small reset glitches are automatically filtered</p>	<p>Less number of gates</p> <p>Only to cover setup and hold times</p> <p>Metastability problems</p> <p>Fast</p> <p>Consumes less power</p> <p>May reset the circuit if glitch covers setup and hold time</p>

- ▶ No choice is in general better, depends on design
- ▶ Can convert asynchronous reset to synchronous and then use as asynchronous input to all flip-flops



# Attributes

- ▶ Attributes are similar to #PRAGMA directives for compilers
- ▶ These are dependent on the synthesis tool being used

## Example (for Synplify):

```
module STARTUP_VIRTEX (CLK, GSR, GTS) /*synthesis syn_black_box */;
    input          CLK          /* synthesis syn_isclock = 1 */;
    input          GSR, GTS;
endmodule

...

module OBUF (O, I) /* synthesis syn_black_box */;
    output O;
    input  I;
endmodule
```



# Utilizing Readily Available IP

---

- ▶ Xilinx Coregen
  - Adders, multipliers etc.
  - Build complex modules using hardware available in FPGA (multipliers)
- ▶ Synopsys DesignWare
  - Adders, multipliers, ALUs etc.
  - Simulation models are available
- ▶ Free IP from OpenCores.org
  - Complex hardware cores from floating-point unit to processors
  - Integration is complex with other larger cores
- ▶ Tradeoff between integration, validation and effort required to write the IP yourself



# ***VHDL Coding Styles: Summary***

---

- ▶ Try to use the two process model
- ▶ Group your port signals in records
- ▶ Try to write parameterized code, this doesn't sacrifice efficiency
- ▶ Fully specify your FSM
- ▶ Make intelligent choice about synchronous and asynchronous resets
- ▶ Make extensive use of parameterized and freely available IP



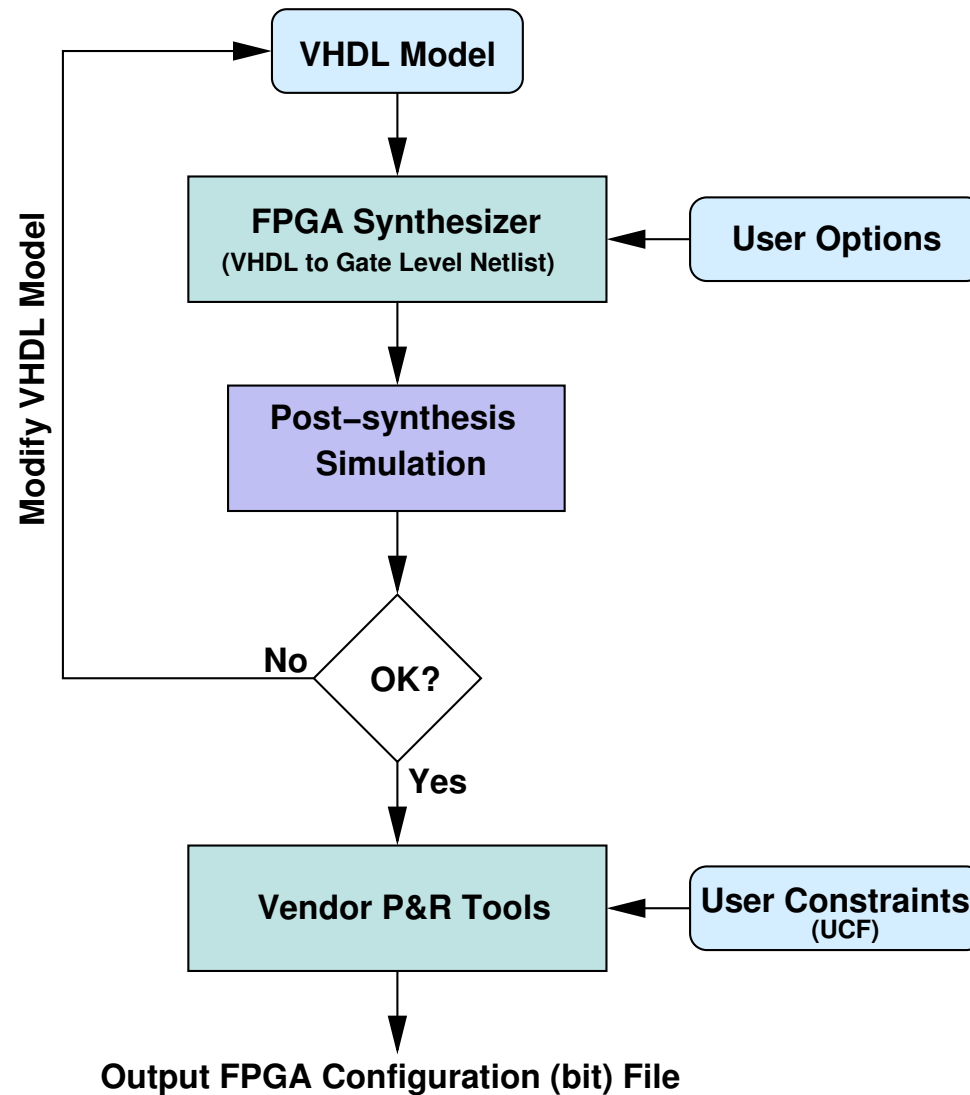
# Outline

---

- ▶ Overview
- ▶ VHDL Coding Styles
- ▶ **FPGA Synthesis, Place and Route**
- ▶ Demo of FPGA Synthesis, Place and Route
- ▶ ASIC Synthesis, Place and Route
- ▶ Demo of ASIC Synthesis, Place and Route
- ▶ Conclusions and Further Reading



# Typical Flow







# Available Synthesis Tools

---

## ► FPGA Express:

- Advantages: None, Our first LEON synthesized only with it -:)
- Disadvantaegs: Buggy, slow, runs only on Windows
- Vendor: Synopsys
- Status: Synopsys has discontinued it, no longer bundled with Xilinx tools

## ► Xilinx Synthesis Tool (XST):

- Advantages: Bundled with Xilinx ISE, Can automatically infer Xilinx FPGA components, runs on UNIX (Solaris and GNU/Linux)
- Disadvantaegs: Still buggy, Only supports Xilinx devices
- Vendor: Xilinx
- Status: Active support from Xilinx



# Available Synthesis Tools (contd...)

---

## ► Leonardo Spectrum:

- Advantages: Supports large family of FPGA devices, Reliable, Can automatically infer FPGA components
- Disadvantages: Runs on Solaris, Replaced by Precision-RTL
- Vendor: Mentor Graphics
- Status: To be replaced soon

## ► Synplify:

- Advantages: Most trusted in industry, Support large family of FPGA devices, Can automatically infer FPGA components
- Disadvantages: Some optimizers are still buggy (FPGA Compiler)
- Vendor: Synplicity
- Status: Active support from Synplicity



# ***Available Place and Route Tools***

---

## ► Xilinx ISE:

- Advantages: Vendor provided Place and Route tool, there is no other choice
- Disadvantages: No point of comparison
- Vendor: Xilinx
- Status: Active support from Xilinx



# ***Synthesis With Synplify***

---

```
add_file -vhdl -lib work accu.vhd
# Constraints and Options!
impl -add syn #implementation: "syn"
# Device Options
set_option -technology VIRTEX2;
set_option -part XC2V8000
set_option -package FF1517;
set_option -speed_grade -5
# Compilation/Mapping Options
set_option -top_module "accu";
set_option -default_enum_encoding onehot
set_option -resource_sharing 0;
set_option -symbolic_fsm_compiler 0
```



# Synthesis With Synplify

---

```
# Mapping Options
set_option -frequency 500.000;
set_option -disable_io_insertion 0
set_option -pipe 0;
set_option -modular 0;
set_option -retiming 0
# Output Options
set_option -write_verilog 0;
set_option -write_vhdl 1
set_option -write_apr_constraint 0 # Whether to output constraints
project -result_file "./accu.edf";
impl -active "syn" # Results
```

(Source: LEON2 synthesis scripts)



# ***Post-synthesis Simulation***

---

- ▶ Modelsim can be used
- ▶ UNISIM libraries are needed and available with Modelsim
- ▶ Vendor specific modules available from vendor tool or synthesis tool distributions



# User Constraints File (UCF)

```
# Define Logical-to-Physical pin mapping
NET "LCLKA"          LOC = "AG18";
# <> for FPGA Express and XST; () for Synplify
NET "RA<18>"         LOC = "AG6";
# Define attributes
NET "RA0<*>"         FAST;
# Define setup, hold and other timing constraints for
# peripheral devices
TIMEGRP  RAM_ADDR_PADS          = "pads (RA<*>) ";
TIMEGRP  "RAM_PADS_ADDR" OFFSET = IN  5.0ns BEFORE "LCLKA";
TIMEGRP  "RAM_PADS_ADDR" OFFSET = OUT 7.0ns AFTER  "LCLKA";
```

NOTE: TIMESPEC syntax is more intuitive

(Source: ADM-XRC2 UCF files)



## ***Place and Route with Xilinx ISE***

---

```
# Convert to Xilinx Database Format
ngdbuild -nt on -dd work -uc accu.ucf -p XC2V3000-5-FF1152\
    accu.ngd
# Map to target device
map -pr b -o accu_map.ncd accu.ngd accu.pcf
# Do Place and Route
par -w accu_map.ncd accu.ncd accu.pcf
# Timing Analysis
trce -e 3 -o accu.twr accu.ncd accu.pcf
# Final Bitfile Generation
bitgen -w -g drivedone:yes accu.ncd accu.bit
```





## Other Peculiarities

---

- ▶ Take care with array indices (<>, () or [])
- ▶ If you are using coregen generated components, you should select the correct synthesis tools
- ▶ Automatic inferring of devices and retiming optimization makes synthesis very time consuming
- ▶ Simulation with device modules such as *Virtex ChipStartup* will take very long
- ▶ RAM models are available from vendors which you can directly use



# Outline

---

- ▶ Overview
- ▶ VHDL Coding Styles
- ▶ FPGA Synthesis, Place and Route
- ▶ Demo of FPGA Synthesis, Place and Route
- ▶ ASIC Synthesis, Place and Route
- ▶ Demo of ASIC Synthesis, Place and Route
- ▶ Conclusions and Further Reading



# ***FPGA Synthesis, P&R Demo***

---

- ▶ Vanilla synthesis, place and route
- ▶ Using the on-chip BlockRAMs
- ▶ Using on-chip Digital Clock Managers (DCM)
- ▶ Using clock-buffers
- ▶ Interfacing to an external RAM
- ▶ Specifying voltage level for output pins

**UCF Ref:** <http://toolbox.xilinx.com/docsan/xilinx5/pdf/docs/cgd/cgd.pdf>



# Outline

---

- ▶ Overview
- ▶ VHDL Coding Styles
- ▶ FPGA Synthesis, Place and Route
- ▶ Demo of FPGA Synthesis, Place and Route
- ▶ ASIC Synthesis, Place and Route
- ▶ Demo of ASIC Synthesis, Place and Route
- ▶ Conclusions and Further Reading



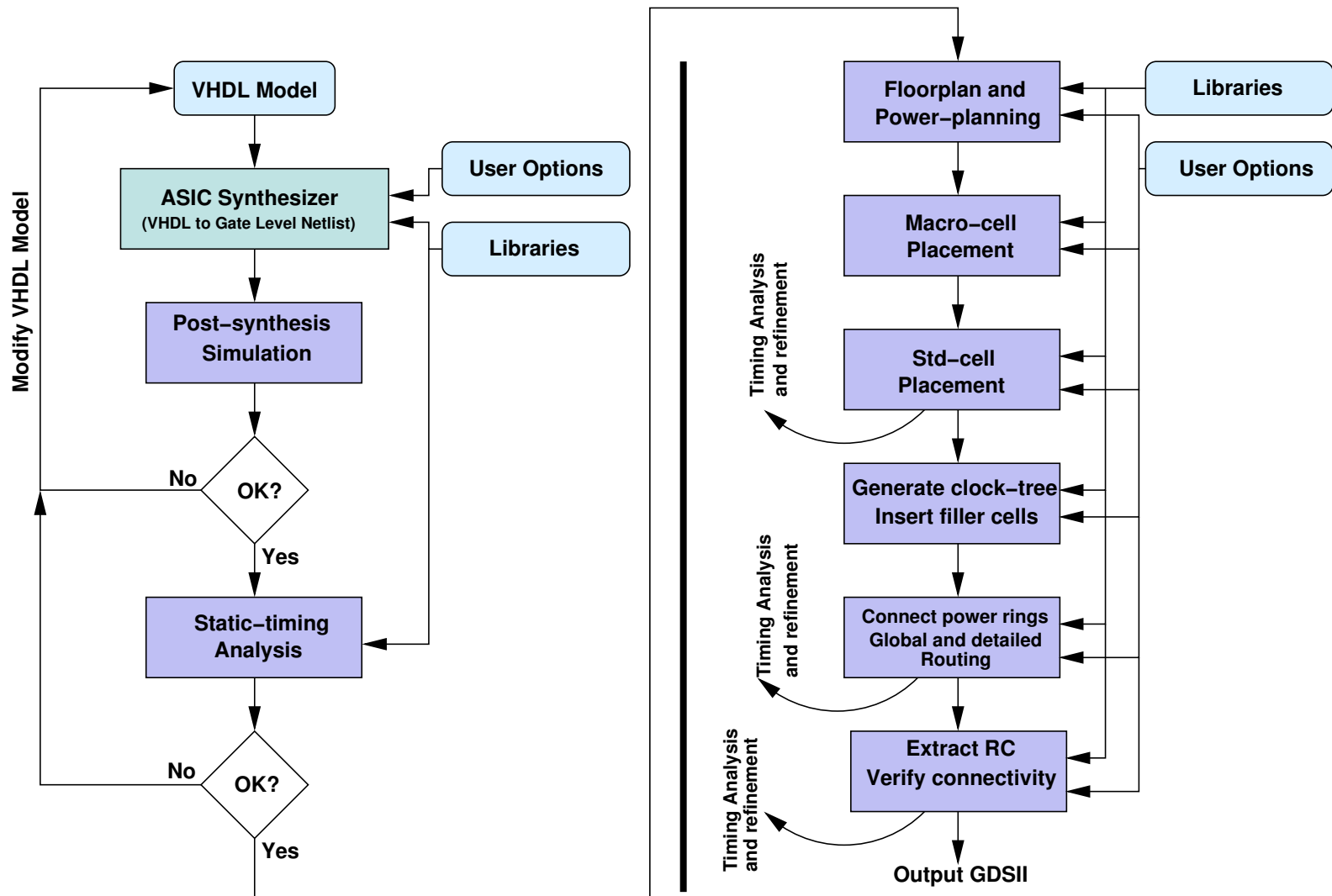
# ***Complications Over FPGA Synthesis, P&R***

---

- ▶ Massive amounts of hardware flexibility leads to complications and advantages
- ▶ Deciding parameters such as row utilization, chip aspect ratio etc. requires insight and experience
- ▶ Various stages of the flow are inter-dependent
- ▶ Difficult to estimate the effects of various stages of flow early on



# Typical Flow





# Available Synthesis Tools

---

## ► Synopsys Design Compiler:

- Advantages: Most heavily used (80% market share)
- Disadvantages: Grave problem with aggregates
- Vendor: Synopsys
- Status: Active support

## ► Cadence BuildGates:

- Advantages: Good scripting capabilities, integrates well with back-end Cadence tools, handles aggregates well, cheap
- Disadvantages: Low-end solution, not very established
- Vendor: Cadence
- Status: Being replace by Cadence RTL Compiler



# *Available Synthesis Tools (contd...)*

---

## ▶ Leonardo Spectrum:

- Advantages: Similar interface for FPGA and ASIC synthesis
- Disadvantages: Not established
- Vendor: Mentor Graphics
- Status: Being phased out

## ▶ Cadence RTL Compiler Ultra:

- Advantages: Well received in community, very fast, integrates well with back-end Cadence tools, cheap
- Disadvantages: New solution
- Vendor: Cadence
- Status: Active support





# ***Available Synthesis Tools (contd...)***

---

- ▶ Precision RTL:
  - Advantages: Similar interface for FPGA and ASIC synthesis
  - Disadvantages: ??
  - Vendor: Mentor Graphics
  - Status: Active support



# ***Available Place and Route Tools***

---

- ▶ Synopsys Apollo:
  - Advantages: Integrates well with front-end Synopsys tools
  - Disadvantages: Not established
  - Vendor: Synopsys
  - Status: Active support
  
- ▶ Cadence Silicon Ensemble Family (Qplace, Wroute and Groute):
  - Advantages: Very established set of tools, provide full control to the user
  - Disadvantages: Slow, based on very old technology
  - Vendor: Cadence
  - Status: Active support



# ***Available Place and Route Tools***

---

- ▶ Cadence SoC Encounter Family (Nanoroute, Amoeba):
  - Advantages: Next generation tools, run on GNU/Linux, very fast, allow hierarchical design flow without creation of explicit black-box models
  - Disadvantages: Difficult to control at finer level
  - Vendor: Cadence
  - Status: Active support
- ▶ Mento Graphics ICStation:
  - Advantages: ??
  - Disadvantages: ??
  - Vendor: Mentor Graphics
  - Status: Active support



## Other tools

Tool Name	Functionality
Qplace	Placer (Cadence)
Groute	Global router (Cadence)
Wroute	Local router (Cadence)
Nanoroute	Global and local router (Cadence)
CTGen	Clock-tree generator which may be used for reset (Cadence)
Fire & Ice	Parasitic extractor (Cadence)
HyperExtract	Parasitic extractor (Cadence)
Pearl	Timing analyzer (Cadence)
Primetime	Timing analyzer (Synopsys)
Primepower	Power analyzer (Synopsys)
CeltIC	IR drop analyzer (Cadence)



# *Flat Synthesis Using Design-Compiler*

---

- ▶ Specify library search path and libraries
- ▶ Set global constraints (clock period)
- ▶ Set operating conditions and wire load model
- ▶ Write output netlist, constraints and reports

(NOTE: Full synthesis script is in `~help/syntut` in philips lab.)



# ***Clock-tree Specification***

---

AutoCTSRootPin	clk
NoGating	NO
MaxDelay	0.4ns
MinDelay	0.1ns
MaxSkew	20ps
MaxDepth	20
Buffer	CLKBD2 CLKBD4 CLKBD8 CLKBD12 CLKBD16 CLKBD20 CLKB
End	



# Constraints File

---

```
set sdc_version 1.4
current_design accu
create_clock [get_ports {clk}] -name clk -period 1.0 -waveform {0 0}
set_false_path -from [get_ports {resetn}]
```



# ***I/O Position File***

---

Version: 2

Pin: clk N

Pin: resetn N

Pin: i[9] E

Pin: i[8] E

Pin: o[9] W

Pin: o[8] W





# Configuration File

---

- ▶ Specify timing and LEF libraries
- ▶ Specify timing constraints file
- ▶ Specify power and ground lines
- ▶ Specify core utilization, chip area and/or aspect ratio
- ▶ Specify chip orientation

(NOTE: Full configuration file is in `~help/syntut` in philips lab.)



# *Flat P&R Using SoC Encounter*

---

- ▶ Load configuration file, pin placements, Floorplan the chip
- ▶ Add power-rings and stripes
- ▶ Place cells using either the timing driven or power driven mode
- ▶ Perform timing optimization, timing violation fixing and congestion optimization
- ▶ Generate clock-tree
- ▶ Insert filler cells
- ▶ Connect power rings to power pins of cells
- ▶ Route clock-nets
- ▶ Perform global and detailed routing, extract RC
- ▶ Verify connectivity, geometry and output results

(NOTE: Full P&R script is in ~help/syntut in philips lab.)



# ***Hierarchical Flow***

---

The following steps are added to the existing flow:

- ▶ Specify the lower level modules as black-boxes in design-compiler
- ▶ Generate TLF and LEF models for macros (black-boxes)
- ▶ Place macros immediately after floorplanning, if multiple choices are generated manual selection must be performed later on
- ▶ Freeze the position of macros, so that amoeba doesn't displace them

(NOTE: Full P&R script is in `~help/syntut` in philips lab.)



# Outline

---

- ▶ Overview
- ▶ VHDL Coding Styles
- ▶ FPGA Synthesis, Place and Route
- ▶ Demo of FPGA Synthesis, Place and Route
- ▶ ASIC Synthesis, Place and Route
- ▶ Demo of ASIC Synthesis, Place and Route
- ▶ Conclusions and Further Reading



# Outline

---

- ▶ Overview
- ▶ VHDL Coding Styles
- ▶ FPGA Synthesis, Place and Route
- ▶ Demo of FPGA Synthesis, Place and Route
- ▶ ASIC Synthesis, Place and Route
- ▶ Demo of ASIC Synthesis, Place and Route
- ▶ Conclusions and Further Reading



# Conclusions and Further Reading

---

- ▶ When writing VHDL keep the target hardware in mind
- ▶ Optimization is good but not mandatory, keep the tradeoff between complexity of implementation and efficiency in mind
- ▶ Thorough simulation now will save you a lot of hassle later on
- ▶ FPGA implementation ends up taking more time than synthesis and simulation
- ▶ Tools typically change in three years, knowing idiosyncrasies of tools helps but is not permanent knowledge
- ▶ Further Reading:
  - VHDL Coding Styles at <http://www.gaisler.com/doc/vhdl2proc.pdf>
  - Product manuals: Cadence SoC Encounter User Guide, Synopsys Design Compiler User Guide etc.
  - Create Cadence Website <http://create.cadence.com>



***Thank You***

---

**Thank You**