Outline.

1. *Design flow*
2. *Synthesis*
3. *Place & Route*
1. Design flow
Design flowgraph

- **Tools**
  - Synthesis: Synopsys Design Compiler
  - Place & Route: Cadence SOC Encounter

- **Process**
  - UMC L90 SP

- **Standard cell library**
  - Faraday:
    - fsd0a_a_generic_core
    - fod0a_b25_t25_generic_io
2. Synthesis
Synthesis flow

Asic Synthesis Flow

Register Transfer
Architectural HDL

Synthesis

Our focus will be here.

Gate-Level Netlist

Physical Device
Silicon

Design Compiler Flow

Write RTL code and simulate

Create start-up file

Create constraints file

Select appropriate compile flow

Load designs and libraries

Apply design constraints

Synthesize the design

Write out design data
Synthesis Transformations

Synthesis = Translation + Logic Optimization + Gate Mapping

```
residue = 16’h0000;  // RTL Source
if (high_bits == 2'b10)
    residue = state_table[index];
else
    state_table[index] = 16’h0000;
```

1. Translate
   - read_verilog
   - read_vhdl

2. Constrain (source)
   - set_max_area...
   - create_clock...
   - set_input_delay...

3. Optimize + Map (compile)

Constraints
- set_max_area...
- create_clock...
- set_input_delay...

Generic Boolean Gates
- GTECH or unmapped ddc format

Technology-specific Gates

The verb “to compile” is used synonymously with “to synthesize”
Load Design and Libraries

• **Analyze**
  - Reads source code files (Verilog or VHDL RTL)
  - Checks syntax and issues errors/warnings
  - Converts both Verilog and VHDL files into intermediate binary format files, placed in CWD
  - Can use define_design_lib to redirect the files/directories to a sub-directory

• **Elaborate**
  - Reads the intermediate .pvl files and builds the ‘GTECH’ design in DC memory (unmapped ddc format)
  - Sets the current design to the specified design
  - Links and auto-loads the specified design
  - Allows specification of parameter values: `elaborate MY_TOP -parameters “N=8, M=3”`. 
Area and Timing Constraints

- Static Timing Analysis (STA)
- Modeling clocks
- Constraining input paths
- Constraining output paths
- Environmental attributes
Static Timing Analysis

- Main steps of STA
  - Break the design into sets of timing paths
  - Calculate the delay of each path
  - Check all path delays to see if the given timing constraints are met

- Four types of paths

```
Start point

PI

Reg

Combinational logic

Reg

End point

PO
```
Static Timing Analysis

- Path-based STA
  - Calculate the Arrival Time (AT) by adding cell delay in timing paths
  - Check all path delays to see if the given Required Arrival Time (RAT) is met

Path-based:

- $2+2+3 = 7$ (OK)
- $2+3+1+3 = 9$ (OK)
- $2+3+3+2 = 10$ (OK)
- $5+1+1+3 = 10$ (OK)
- $5+1+3+2 = 11$ (Fail)
- $5+1+2 = 8$ (OK)
### Static Timing Analysis (Cell delay)

**Cell Delay**

\[ D_{\text{cell}(I_2)} = f(D_{\text{transition}(I_1)}, C_{eq}) \]

**Transition Delay**

\[ D_{\text{transition}(I_2)} = g(D_{\text{transition}(I_1)}, C_{eq}) \]

<table>
<thead>
<tr>
<th>Output Capacitance</th>
<th>Input Transition</th>
<th>0</th>
<th>0.5</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td></td>
<td>0.123</td>
<td>0.234</td>
<td>0.456</td>
</tr>
<tr>
<td>0.2</td>
<td></td>
<td>0.222</td>
<td>0.432</td>
<td>0.801</td>
</tr>
</tbody>
</table>

*Index1: input transition Index2: output capacitance*
Static Timing Analysis (Setup Time)

- To meet the setup time requirement:
  - $T_{\text{require}} \geq T_{\text{arrival}}$

- Reg to Reg
  - $T_{\text{arrival}} = T_{\text{clk1}} + T_{\text{DFF1(clk->Q)}} + T_{\text{path}}$
  - $T_{\text{require}} = T_{\text{clk2}} - T_{\text{DFF2(setup)}}$
  - $T_{\text{slack}} = T_{\text{require}} - T_{\text{arrival}}$
Static Timing Analysis (Setup Time)

- PI to Reg

  - $T_{arrival} = T_{PI(delay)} + T_{path}$
  - $T_{require} = T_{cycle} + T_{clk1} - T_{DFF1(setup)}$
  - $T_{slack} = T_{require} - T_{arrival}$
Static Timing Analysis (Setup Time)

- Reg to PO

- $T_{\text{arrival}} = T_{\text{clk1}} + T_{\text{DFF1(clk->Q)}} + T_{\text{path}}$
- $T_{\text{require}} = T_{\text{cycle}} + T_{\text{clk1}} - T_{\text{PO(output delay)}}$
- $T_{\text{slack}} = T_{\text{require}} - T_{\text{arrival}}$
Static Timing Analysis (Setup Time)

- PI to PO

\[
T_{\text{arrival}} = T_{\text{PI(delay)}} + T_{\text{path}} \\
T_{\text{require}} = T_{\text{cycle}} - T_{\text{PO(output delay)}} \\
T_{\text{slack}} = T_{\text{require}} - T_{\text{arrival}}
\]
Static Timing Analysis (Hold Time)

• To meet the hold time requirement:
  • \( T_{\text{require}} \leq T_{\text{arrival}} \)

• Reg to Reg
  • \( T_{\text{arrival}} = T_{\text{clk1}} + T_{\text{DFF1}(\text{clk} \rightarrow Q)} + T_{\text{path}} \)
  • \( T_{\text{require}} = T_{\text{clk2}} - T_{\text{DFF2}(\text{hold})} \)
  • \( T_{\text{slack}} = T_{\text{arrival}} - T_{\text{require}} \)
Static Timing Analysis (Hold Time)

- **PI to Reg**
  - $T_{arrival} = T_{PI(delay)} + T_{path}$
  - $T_{require} = T_{clk} - T_{DFF(hold)}$
  - $T_{slack} = T_{arrival} - T_{require}$

- **Reg to PO**
  - $T_{arrival} = T_{clk} + T_{DFF(clk->Q)} + T_{path}$
  - $T_{require} = - T_{PO(output delay)}$
  - $T_{slack} = T_{arrival} - T_{require}$

- **PI to PO**
  - $T_{arrival} = T_{PI(delay)} + T_{path}$
  - $T_{require} = - T_{PO(output delay)}$
  - $T_{slack} = T_{arrival} - T_{require}$
Default Clock Behavior

- **Defining the clock in a single-clock design constrains all timing paths between registers for single-cycle, setup time**

- **By default the clock rises at 0ns and has a 50% duty cycle**

- **By default DC will not “buffer up” the clock network, even when connected to many clock/enable pins of flip-flops/latches**
  - The clock network is treated as “ideal” - infinite drive capability
    - Zero rise/fall transition times
    - Zero skew
    - Zero insertion delay or latency
  - Estimated skew, latency and transition times can, and should be modeled for a more accurate representation of clock behavior
Defining a Clock

Default clock with 50% duty cycle

Clock with specified duty cycle

```
create_clock -period 2 [get_ports Clk]
```

TCL: Command option

```
clock get_ports Clk
```

Port object Clk

Same name by default

```
clock set_period 0 2
```

Unit of time is 1ns in this example. Defined in the technology library.

```
create_clock -period 2 -waveform {0 0.6} -name My_CLK
```

Clock object My_CLK

```
clock get_period
```

Period

```
clock get_period
```

0ns 1ns 2ns

```
clock get_period
```

My_CLK

```
clock get_period
```

0ns 0.6ns 2ns
Modeling Clock

Pre-Layout: clock skew + jitter + margin

set_clock_uncertainty –setup Tu [get_clocks CLK]

reset_design
create_clock –p 5 –n MCLK Clk
set_clock_uncertainty 0.5 MCLK
set_clock_transition 0.08 MCLK
set_clock_latency –source –max 4 MCLK
set_clock_latency –max 2 MCLK
Specifying Setup-Timing Constraints

- **Objective:** Define setup timing constraints for all paths within a sequential design
  - All input logic paths (starting at input ports)
  - The internal (register to register) paths
  - All output paths (ending at output ports)
Constraining Input Paths

The user must specify the latest arrival time of the data at input A.

What is $T_{\text{max}}$ for N?

The maximum delay for path N = 1.5 ns

```python
create_clock -period 2.5 [get_ports Clk]
set_input_delay -max 0.9 -clock Clk [get_ports A]
```

---

```python
create_clock -period 2 [get_ports Clk]
set_input_delay -max 0.6 -clock Clk [get_ports A]
```

The maximum delay for path N = 1.5 ns

```
create_clock -period 2.5 [get_ports Clk]
set_input_delay -max 0.9 -clock Clk [get_ports A]
```
Constraining Output Paths

The user must specify the latest arrival time of the data at output B.

What is $T_{\text{max}}$ through S?

create_clock –period 2 [get_ports Clk]
set_output_delay –max 0.8 –clock Clk [get_ports B]

The maximum delay to port B = 0.7 ns

create_clock –period 2 [get_ports Clk]
set_output_delay –max 1.3 –clock Clk [get_ports B]
Environmental attributes

- Input drivers and transition times
- Capacitive output loads
- Process/Voltage/Temperature (PVT) operating conditions
- Interconnect parasitic RCs
Input drivers and transition times

- Rise and fall transition times on an input port affect the cell delay of the input gate

```plaintext
set_input_transition 0.12 [get_ports A]
set_driving_cell –lib_cell OR3B [get_ports A]
set_driving_cell –lib_cell FD1 –pin Qn [get_ports A]
```
Capacitive output loads

Capacitive loading on an output port affects the transition time and thereby the cell delay of the output driver

set_load [expr 30.0/1000] [get_ports B]
Load budget

Assumptions:
1. The maximum fanout capacitance of any block’s input port is limited to the equivalent of 10 “and2a1” gates
2. Output ports can drive a maximum of 3 other blocks
3. The driving gate of every output is the cell inv1a1
Interconnect parasitic RCs

- A wire load model calculates one parasitic R and one C for each net, based on the net’s fanout:
  - Models for various design sizes are supplied by your vendor
  - R/C values are average estimates based on data extracted from similar designs which were fabricated using this process

<table>
<thead>
<tr>
<th>Fanout</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14.15</td>
</tr>
<tr>
<td>2</td>
<td>32.31</td>
</tr>
<tr>
<td>3</td>
<td>52.48</td>
</tr>
<tr>
<td>4</td>
<td>74.91</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>952.16</td>
</tr>
</tbody>
</table>

| Name    | 140000 |
| Location| 90nm_com |
| Resistance| 0.000331 |
| Capacitance| 8.6e-05 |
| Area    | 0.1    |
| Slope   | 93.7215 |

<table>
<thead>
<tr>
<th>Fanout</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>24.58</td>
</tr>
<tr>
<td>2</td>
<td>58.28</td>
</tr>
<tr>
<td>3</td>
<td>98.54</td>
</tr>
<tr>
<td>4</td>
<td>146.54</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>2946.37</td>
</tr>
</tbody>
</table>

| Name    | 8000000 |
| Location| 90nm_com |
| Resistance| 0.000331 |
| Capacitance| 8.6e-05 |
| Area    | 0.1    |
| Slope   | 334.957 |

Extrapolation slope

kΩ/unit length
pF/unit length
area/unit length
Synthesize Design

- RTL Description or unmapped ddc
- Netlist or mapped ddc

- Compile
  - Architectural level
  - Logic level
  - Gate level

- Optimized Netlist

Flow:
- Read RTL Description or unmapped ddc
- Read Netlist or mapped ddc
- Compile
  - Architectural level
  - Logic level
  - Gate level
- Write Optimized Netlist

Steps:
- High-Level Synthesis
- Structuring
- Mapping
Three interfaces to Design Compiler

1. **Design Vision** (interactive GUI)
   - `unix% design_vision`

   ![Design Vision interface](image)

   DC invoked in ‘TCL mode’ and ‘XG mode’

2. **DC Shell** (interactive shell)
   - `unix% dc_shell-t`
   - `dc_shell-xg-t>`

3. **Batch mode**
   - `unix% dc_shell-t -f RUN.tcl | tee -i my.log`
Filter Example

\[ Z_{n} = \sum_{i=0}^{127} Y_{n-i} \cdot K_{i} \]

Filter Parameters:

- **FIR filter**
- 128 taps
- Sample Frequency: 48 kHz
- Pass band: 9.6 kHz
- Stop band: 11.0 kHz
- Data width: 16 bits
- Coefficient bit-width: 16 bits
Techmap I/O Cells

Techmap library independent I/O cells

Generic spec:
- Tech: io technology (faraday)
- Limit: pad/core limited (core)
- Slew: fast/slow slew rate (slow)
- Width: vector width (only for vector IO)

Port spec:
- Single io
  - inpad (pad, o)
  - outpad (pad, i)
  - iopad (pad, i, e, o)
  - toutpad (pad, e, i)
- Vector io
  - inpadv (pad, o)
  - outpadv (pad, i)
  - iopadv (pad, i, e, o)
  - iopadvv (pad, i, e, o)
  - toutpadv (pad, e, i)
  - toutpadvv (pad, e, i)
Memory Macro Cells

- Macro cells generated by dedicated memory compilers

  - SHAA90_ : single-port static RAM (256-15k, 1-32)
  - SJAA90_ : dual-port static RAM (32-8k, 1-64)
  - SYAA90_ : single-port register-file (8-2k, 1-144)
  - SZAA90_ : two-port register-file (8-2k, 1-144)
  - SPAA90_ : via programmable ROM (512-128k, 1-128)

Examples:

SYAA90_ 128X16X1CM2
SPAA90_ 512X16X1BM1A
3.

Place and Route
SOC Encounter Place & Route Flow

- Verilog mapped design file
- io_spec file
- Timing library files
- Layout library files

Design Import and Load Libraries

- Floorplan
- Power Plan
- Placement
- Clock Tree Synthesis
- Route
- Verification

Write output files

Optimization

- Verilog routed design file
- GDS2 Layout file
Chip Planning

Requirements:

- Die size choices
  - 2mm x 2mm (approx. € 5000)
  - 4mm x 2mm (approx. € 10000)
- Number of IO cells
  - Max. 84 (2mm x 2mm)
  - Max. 140 (4mm x 2mm)
- Libraries Faraday 90 nm
  - Timing: fsd0a_a_generic_core*.lib, fod0a_b25_t25_generic_io*.lib
  - Layout: fsd0a_a_generic_core*.lef, fod0a_b25_t25_generic_io*.lef
Ratio Signal IO/ Power, Ground IO

- Considering 2 effects
  - Electro-Migration
  - Simultaneous Switching outputs

- As a rule of thumb the ratio lies between 4 and 16 depending on clock rate, slew rate and load capacitance
Ratio Signal IO/ Power, Ground IO

- Considering Electro-Migration
  - N buffers/PWR-GND pair:
  - f = 100 MHz
  - V = 2.5 V
  - C = 50 pF

\[ I = NC \frac{dv}{dt} = NCVf \]

\[ N = \frac{156mA}{50pF + 2.5V + 100MHz} = 12.5 \]

\[ \Rightarrow N = 13 \]
Concept of Ground Bounce

Current flow (red) during a high to low transition causing “bounce.” This can change the input thresholds to the device as well as result in output pulses being transmitted to a receiver.
Pad/Core limited IO

Depends on the size of the core
- Pad limited: small width, large height
- Core limited: large width, small height
Generate IO file

Script to generate iofile:

```plaintext
% genIoFile <#N> <#E> <#S> <#W> [<#W> <#W>] < io.spec > filter_top.io
```
Soc Encounter GUI
Design Import

Design ⇒ Design Import

- Verilog technology mapped design file
- Timing libraries
  - Max: containing worst-case conditions for setup-time analysis
  - Min: containing best-case conditions for hold-time analysis
- IO Assignment File
Floorplan (1)

Purpose:
- Develop early physical layout to ensure that design objectives can be archived
  - Minimum area for low cost
  - Minimum congestion for design routable
  - Estimate parasitic for delay calculation
  - Analysis of power for reliability
  - Gain early visibility into implementation issues

How:
- Specify size of die, core, IO
- Placement of macro blocks
- Fix the areas where standardcell will be placed
Floorplan (2)

Floorplan ⇒ Specify Floorplan

Floorplan ⇒ Automatic Floorplan ⇒ Plan Design...
Floorplan (3)

Alternative placement by hand leads to better performance
Floorplan (4)

Module constraint options:

- None
- Guide
- Fence
- Region
- Soft Guide
Floorplan(5): Edit Halo

A Halo:
- Keeps standard cells from being placed too close to macro cells
- Avoids problems when routing power lines of standard cells
Powerplan (1)

Important issues with power nets:

- IR Drop: voltage drop to resistance in power lines decrease VDD resulting in slower circuits or may lead to violation of noise margins
- Electromigration: Thermally agitated metal ions are washed away by flowing electrons

- Create connectivity for power/ground nets not specified in the Verilog netlist.
- Add core power/ground rings that distribute power/ground around the core
- Add block power/ground rings that distribute power/ground around the macro blocks (e.g. Memories)
- Add stripes for better power distribution especially with fast designs
- Add filler cells to fill the gaps between I/O cells
Powerplan(2): Add rings

*Power ⇒ Power Planning ⇒ Add Rings...*

- Power ring around the core
- Power ring around macrocells
Powerplan: Global Net Connections

*Floorplan → Connect Global Nets...*

- Specify global power and ground nets
Powerplan: Sroute and Add iofill cells

- Fill the gaps between I/O cells with filler cells to close the power rings by abutment.
- Use script PAR/BIN/fillperi.tcl

**Route ⇒ Special Route...**
- Sroute
- Padpins
- Blockpins
- StandardCellPins
Cell placement (1)

**Place ➔ Standard Cells**
- Prototyping: Runs quickly, but components may not be placed at legal location.

- Timing Driven:
  - Build timing graph before place.
  - Meeting setup timing constraints with routability.
  - Limited IPO by upsizing/downsizing instances.
- Reorder Scan Connection
  - nets connected to either the scan-in or scan-out are ignored.
- Check placement after placed

**Place ➔ Check Placement**
Process:
- Verify ➔ Geometry
Clock Tree Synthesis

Clock problem:

- Heavy clock net loading
- Long clock insertion delay
- Clock skew
- Skew across clocks
- Clock to signal coupling effect
- Clock is power hungry
- Electromigration on clock net
Clock Tree Synthesis (2): flow

- Create Clock Tree Spec
- Specify Clock Tree
- Synthesis Clock Tree
- Display Clock Tree

Modifying the clock spec:

- Netlist
- Synthesis report
- Clock nets
- Routing guide
Pre-CTS clock tree tracer

Clock ⇒ Tracer Pre-CTS Clock Tree ....

Things to look for include:

- Clock, reset, or scan-enable connecting to unexpected input pins.
- Unexpected latches on the clock tree
- Discrepancy between the number of endpoints of clock, reset and scan trees.

For our example numbers are as follows:

- clock tree: 443 with 442 flip-flop CK pins + 1 RAM CK pin
- reset tree: 441 flip-flop RB pins
- scan tree: 447 with 441 flip-flop SEL pins + 6 mux S pins, to choose between the functional and test (scan chain) output signal.
Mapping SDC to CTS spec

<table>
<thead>
<tr>
<th>Timing Constraints</th>
<th>Clock Tree Specs</th>
</tr>
</thead>
<tbody>
<tr>
<td>create_clock</td>
<td>AutoCTSRootPin / ClkGroup</td>
</tr>
<tr>
<td>create_generated_clock</td>
<td>ThroughPin</td>
</tr>
<tr>
<td>set_clock_latency</td>
<td>Maxdelay</td>
</tr>
<tr>
<td>set_clock_uncertainty</td>
<td>Maxskew</td>
</tr>
<tr>
<td>set_clock_transition</td>
<td>BufMaxTran / SinkMaxTran</td>
</tr>
</tbody>
</table>
Clock Tree Synthesis (2)

CT Synthesis report:
- Summary report and detail report
  - number of sub trees
  - rise/fall insertion delay
  - trigger edge skew
  - rise/fall skew
  - buffer and clock pin transition time
  - detailed delay ranges for all buffers add to clocks
- Clock nets
  - Saves the generated clock nets
  - used to guide clock net routing
- Clock routing guide
  - Saves the clock tree routing data
  - used as preroute guide while running Trial Route
Clock Tree Display

Clock $\Rightarrow$ Display Clock

Display By phase delay
Route

Route $\Rightarrow$ Nanoroute $\Rightarrow$ Route...

- Specify eventually routing features
Finishing

Verify ⇒ Connectivity....

Verify ⇒ Geometry....

Design ⇒ Save GDS ...
Using Scripts

- toplevel.tcl (The overall Place&Route script, calls all following scripts)
  - start.tcl (Imports design)
  - fplan.tcl (generates a floorplan)
  - pplan.tcl (generates the core, block rings and stripes and routes power)
  - pts.tcl (clock tree synthesis)
  - route.tcl (routes the design)
  - verify (generates verification logs, verilog, gds2 of routed design)