SoC Design Flow
from DFT Engineers’ angle

Kang, Yong-Seok
DIS Gr.

LG Electronics / System IC Business Team
Agenda

- General SoC Design Flow and Guide
- DFT & Design Flow
- DFT Flow
- Conclusion
SoC design flow (Cont.)

A
Design Verification → Coverage Metrics → Verification Done?

B
Synthesis → Floorplan & Layout → Parasitic + routed netlist → Timing Analysis → Timing OK?

C
Area + Power → Pin reports → Package Selection → Ball map

Really code frozen

Synthesis SDC

Tape out

STA SDC

DRC/DFM Timing OK?
What is RTL?

- Register Transfer Level (RTL) description is a way of describing the operation of a **synchronous digital circuit**
- Synchronous circuit
  - Register: Synchronize the circuit’s operation
  - Combinational logic: Performs all the logical functions

![Synchronous Circuit Diagram]

- RTL coding
  - Declares the registers
  - Describes the combination logic by using constructs such as if-then-else and arithmetic operations
  - Focuses on describing the flow of signals between registers

**All outputs of module/entity should be registered!**
Clock

Clocking

- One of the great innovations which enabled the semiconductor industry to progress to where it is today
- Quantizes time, enabling Trs to be abstracted to sequential state machines and from state machines into a simple and intuitive programming paradigm for chip design, RTL
- Fundamental assumption: Sequential execution
  - All parts of a state machine stay “in-step” with respect to each other
  - Flip-flop always makes a forward step from state n to state n+1
    - Setup constraints
  - No flip-flop ever makes more than one forward step from state n to state n+2 on a single clock tick
    - Hold constraints

Setup Constraint: \( L + G_{\text{setup}} < T + C \)
Hold Constraint: \( L \geq G_{\text{hold}} > C \)
Clock Design

First rule: "Simple is Beautiful"
Clock Design (Cont.)

Second rule: “Simple is Beautiful”
> Third rule: "Simple is Beautiful"
Clock Design (Cont.)

» Do not make masked clocks
» Do not use phase aligned clocks
» Do not use negative edge of source clock to make generated-clocks
» Do not use negative active clocks
» Do not use deep ripple-dividers in case of high speed designs

» All clocks should be generated in the separated clock generation module on the top hierarchy level
» In case of multiple clock sources, clock domain crossing should be checked
» Do not use manually gated clocks
Clock domain crossing issues

➤ Meta-stability
  - Data from one clock domain may violate setup/hold time in the other clock domain

➤ Data loss
  - Data from one clock domain may not hold long enough to be captured by the other clock domain

➤ Data correlation
  - A group of conversing synchronized control signals are not in sync at a particular clock cycle

➤ Design intent
  - More complex synchronization scheme (FIFO, handshake, C-element and etc.) may not behave as your spec.
Metastability

- General scientific concept which describes states of delicate equilibrium
  - A system is in a metastable state when it is in equilibrium (not changing with time) but is susceptible to fall into lower-energy states with only slightly interaction

- Metastability in electronic circuits
  - Metastability cannot be avoided, but the occurrences can be predicted by using the mean time between failures (MTBF) formula
  - Designers can increase the MTBF using various synchronization schemes

\[
MTBF = \frac{e^{C_2 \cdot \tau_{MET}}}{C_1 \cdot f_{clk} \cdot f_{data}}
\]
Clock Gating

- Do not use manually gated clocks => Use enable signal

```verilog
always@(posedge clk or negedge rst_n)
begin
    if (!rst_n)
        q <= 1'b0;
    else
        if (enable)
            q <= d;
end
```

- CG cell for clock gating
  - Glitch free
  - No DFT problem
  - RTL designers can consider free running clock
Reset

- **Synchronous reset**

```verilog
testbench

module sync_resetFFstyle (
    output reg q,
    input d, clk, rst_n);

    always @(posedge clk)
        if (!rst_n) q <= 1'b0;
        else q <= d;
endmodule
```

- **Asynchronous reset**

```verilog
testbench

module async_resetFFstyle (
    output reg q,
    input d, clk, rst_n);

    // Verilog-2001: permits comma-separation
    // @(posedge clk, negedge rst_n)
    always @(posedge clk or negedge rst_n)
        if (!rst_n) q <= 1'b0;
        else q <= d;
endmodule
```

```verilog
process (clk)
begin
    if (clk'event and clk = '1') then
        if (rst_n = '0') then
            q <= '0';
        else
            q <= d;
        end if;
    end if;
end process;
```

```verilog
process (clk, rst_n)
begin
    if (rst_n = '0') then
        q <= '0';
    elseif (clk'event and clk = '1') then
        q <= d;
    end if;
end process;
```
Synchronous Reset Design

- Around 80+% of the gathered articles focused on synchronous reset issues in collection of ESNUG and SOLVE-IT articles

- **Advantages**
  - Smaller flip-flops, but the combinational logic gate count grows, so the overall gate count savings may not be that significant
  - The clock works as a filter for small reset glitches, however, if these glitches occur near the active clock edge, the flop could go metastable
  - In some designs, the reset must be generated by a set of internal conditions. A synchronous reset is recommended for these types of designs because it will filter the logic equation glitches between clocks

- **Disadvantage**
  - Need a pulse stretcher to guarantee a reset pulse width wide enough to ensure reset is present during an active edge of the clock
  - During simulation, depending on how the reset is generated or how the reset is applied to a functional block, the reset can be masked by X’s
  - Timing closure problem exists if the reset is generated by combinational logic in the ASIC or if the reset must traverse many levels of local combinational logic
  - If a gated clock is used to save power, the clock may be disabled coincident with the assertion of reset
Asynchronous Reset Design

➢ Advantages
  - Data path is guaranteed to be clean
  - Circuit can be reset with or without a clock present
  - Synthesis interface tends to be automatic

➢ Disadvantage
  - For DFT, asynchronous reset should be controllable
  - The reset tree must be timed for both synchronous and asynchronous resets to ensure that the release of the reset can occur within one clock period
  - Asynchronous reset can have, depending on its source, spurious resets due to noise or glitches on the board or system reset
  - Attention must be paid to the release of the reset so as to prevent the chip from going into a metastable unknown state when reset is released.
Reset Design Guide

➤ Guideline I
- Every ASIC using an asynchronous reset should include a reset synchronizer circuit!!

➤ Guideline II
- In general, change the testbench reset signal on the inactive clock edge using blocking assignments.
Reset Design Guide (Cont.)

- Reset-glitch filtering
  - Diagram showing components such as glitch, delayed, reset, masterrst_n, etc.

- Multi-clock reset issues
  - Diagram illustrating reset functionalities for different clock domains.
Function verification

Verification methodology
In order to avoid design problems at late stages of the design flow & to reduce TAT, LG template-based RTL design guide is absolutely required.
Logic Synthesis

- Do you remember “Boolean algebra”, “Karnaugh maps”, “Quine-McCluskey algorithm”, “Espresso heuristic”, and “BDD”?

- Logic synthesis
  - Optimizes the Boolean equations generated by RTL synthesizers
  - Maps them to technology specific gate-level implementations utilizing detailed functional and timing information from technology libraries.

- Operations in the logic synthesis process
  - Multilevel minimization, factorization, and equation flattening
  - Area, power, and timing metrics are optimized in some manner
  - Resource sharing, arithmetic function architecting, topological approaches, congestion consideration…
RTL Synthesizer

Synopsys DC flow

- Timing is everything !!!
  - The weight on timing constraints of the synthesis cost function is most critical
Design Constraints

- SDC (Synopsys Design Constraints)
  - A common language between design processes
  - Specify the design intent, including the timing, power, and area constraints for a design

- SDC Commands
  - **Operating conditions**
    - Wire load models
    - System interface
    - Design rule constraints
  - **Timing constraints**
  - **Timing exceptions**
  - Area constraints
  - Multi-voltage and power optimization constraints
  - Logic assignments
  - See “Using the Synopsys® Design Constraints Format, Application Note”
Gate Level Simulation

➢ There are 3 kinds of homo sapiens on the earth
  ▪ Gate level simulation is not required
  ▪ Gate level simulation is required only in a zero-delay and ideal clock mode
  ▪ Fully back annotated simulation is a mandatory design flow

➢ Usefulness of gate level simulation
  ▪ Since scan and other test structures are added during and after synthesis, they are not checked by the RTL simulations and therefore need to be verified by gate level simulation
  ▪ Static timing analysis tools do not check asynchronous interfaces, so gate level simulation is required to look at the timing of these interfaces
  ▪ Careless wildcards in the static timing constraints set false path or multicycle path constraints where they don’t belong
  ▪ Design changes, typos, or misunderstanding of the design can lead to incorrect false paths or multicycle paths in the static timing constraints
  ▪ Using “create_clock” instead of “create_generated_clock” leads to incorrect static timing between clock domains
  ▪ Gate level simulation can be used to collect switching factor data for power estimation
  ▪ X’s in RTL simulation can be optimistic or pessimistic. The best way to verify that the design does not have any unintended dependence on initial conditions is to run gate level simulation
  ▪ It’s a nice “warm fuzzy” that the design has been implemented correctly
Gate Level Simulation (Cont.)

Sources of trouble in gate level simulation

- Issues caused by Library Models
  - Pointing to Library Models
  - Specify Blocks
  - #1 Delays
  - Pessimistic X propagation

- Issues caused by Design
  - Registers that don’t get reset
  - Clock dividers

- Issues caused by Synthesis
  - Synchronous reset back further into the middle of the logic cloud
  - Incomplete Logic Optimization Interferes With Reset
  - Feedback problems

```verilog
class always @(posedge clk) begin
  if (rst) begin
    count[4:0] <= 5'h10;
  end else begin
    Logic cloud
  end
end
```
Gate Level Simulation (Cont.)

Sources of trouble in gate level simulation (Cont.)

- Issues when running SDF based gate level simulation
  - Effects of timing failures: error messages can be caused by a false timing violation
  - Expected Timing Violations
    - Synchronizers for clock domain crossings
    - Multi-cycle paths

Guideline

- While some of the problems can be avoided through proper influence on your library vendor’s models or proper design guidelines and planning, other problems will simply need to be addressed as they are encountered
- By knowing the areas where these problems can occur, we hope that you are able to more quickly find appropriate solutions
Floorplanning
- Not practical now
- Jupiter ✨ ICC
- Virtual prototyping

Flowchart:
1. Read the netlist (read_netlist)
2. Read the I/O constraints (read_io_constraints)
3. Read the SDC constraints (read_sdc)
4. Initialize the floorplan (initialize_floorplan)
5. Perform initial virtual list placement (create_vpl)
6. Legalize the placement (legalize_vpl)
7. Crack and shape plan groups (create_fp_plan_groups, shape_fp_blocks)
8. Perform power network synthesis (synthesize_fp_rail)
9. Perform power network analysis (analyze_fp_rail)
10. Uniquify to design (uniquify_to_design)
11. Perform plan group sorting using (sort_fp_groups)
12. Perform global routing (route_fp)
13. Perform in-place optimization (optimize_fp)
14. Perform dock planning (dock_to_clock_plan)
15. Perform pin assignment (place_fp)
16. Perform timing budgeting (allocate_fp_budget)
17. Commit the hierarchy (commit_fp)
18. Define the pin assignment (check_fp_pin_assignment)
Auto Placement & Route (Cont.)

- Mapped design
- Timing constraints
- Logical and physical libraries

Tedious manual efforts reduction

- Design planning and power planning
- Placement and optimization (place_opt)
- Clock tree synthesis and optimization (clock_opt)
- Routing and postroute optimization (route_opt)
- Chip finishing and design for manufacturing

Completed design

Timing correlation at each step (parasitics/xtalk/power/ocv)

Zroute: QoR, Speed, DFM
Importance of CTS

➤ Ideal clock model vs. propagated clock model
- An ideal clock model of timing simplifies the propagated clocks model of timing by assuming that the launch and capture clock paths have the same delay
- Clock based design is itself often referred to as “synchronous design” even though there is nothing fundamentally synchronous about clock based design itself!

➤ Role of CTS
- Chip design begins in a ideal clock world but ends in a propagated clock worlds
- Transition between these two worlds

Balanced Clocks Design Flow
Clocks Tree under 65nm

Clock Timing Gap = \( \sigma \left( \frac{L[i] - C[i]}{T} \right) \)

Setup Constraint: \( L + G_{\text{max}} \leq T + C \)

Hold Constraint: \( L + G_{\text{min}} > C \)
Low power design

Power dissipation

\[ E = \int_0^t (V_{DD}I_{\text{leak}} + CV_{DD}^2f_c)dt \]

Total Power Dissipation

\[ \int_0^t V_{DD}I_{\text{leak}}dt \]

Static Power Dissipation

\[ \int_0^t CV_{DD}^2f_c dt \]

Dynamic Power Dissipation

Minimize \( I_{\text{leak}} \) by:
- Reducing operating voltage
- Fewer leaking transistors

Minimize \( I_{\text{switch}} \) by:
- Reducing operating voltage
- Less switching cap
- Less switching activity
Low power design (Cont.)

Power management spectrum

- Process: Multi-Threshold, Multi-Voltage, SOI, Low-K, Body bias, Copper metal, SiGe substrates
- Design: Multi-Threshold, Multi-Voltage, Clock Gating, Power Gating, Low-power circuits, Power-aware memories
- Architecture: Hardware vs. Software Allocation, Algorithm/Implementation, Trade-offs
- Software: Compilers, Power-aware OS, Hibernation modes, Memory access
# Low Power Methodology Comparison

<table>
<thead>
<tr>
<th>Low power Design methods</th>
<th>Multi-Vt library &amp; IP</th>
<th>Multi-Vt library</th>
<th>Long channel library</th>
<th>Tapless library, tap cell</th>
<th>Cell with footer, retention FF</th>
<th>Header/footer switch, mother-daughter switch, switch for always_on rows, retention FF, isolation cell</th>
<th>Dynamic Volt/Freq Scaling (DVFS)</th>
<th>AVS</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSMC low power library &amp; IP</td>
<td>Active leakage</td>
<td>Active leakage</td>
<td>Active leakage</td>
<td>Standby leakage</td>
<td>Standby leakage</td>
<td>Standby Leakage</td>
<td>Dynamic &amp; active Leakage</td>
<td>Dynamic &amp; active Leakage</td>
</tr>
<tr>
<td>Power Reduction</td>
<td>Active leakage</td>
<td>Active leakage</td>
<td>Active leakage</td>
<td>Standby leakage</td>
<td>Standby leakage</td>
<td>Standby Leakage</td>
<td>Dynamic &amp; active Leakage</td>
<td>Dynamic &amp; active Leakage</td>
</tr>
<tr>
<td>Dyn pwr Reduction</td>
<td>(30%VDD) 50%</td>
<td>(7%VDD) 10~15%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Leakage Reduction</td>
<td>2~10x base</td>
<td>7%~15%</td>
<td>2x~10x</td>
<td>10x</td>
<td>30x ~ 60x</td>
<td>60x ~ 120x</td>
<td>3x</td>
<td>10~20%</td>
</tr>
<tr>
<td>Area Overhead</td>
<td>0%</td>
<td>~0%</td>
<td>2%</td>
<td>5%~15%</td>
<td>4%~6%</td>
<td>2%~3%+DNW</td>
<td>2%~5%</td>
<td>2%</td>
</tr>
<tr>
<td>Performance Penalty</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>2%~4%</td>
<td>2%~4%</td>
<td>2%~4%</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Data Retention</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes (HVT), retention FF</td>
<td>Retention FF Extra PG</td>
<td>Retention FF Extra PG</td>
<td>yes VCCmin</td>
<td>Retention Extra PG</td>
</tr>
<tr>
<td>Voltage Regulator needed</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

---

*Header/footer switch, mother-daughter switch, switch for always_on rows, retention FF, isolation cell.*

* Dynamic Volt/Freq Scaling (DVFS) and AVS IP, Level Shifter.*
Agenda

- General SoC Design Flow and Guide
- DFT & Design Flow
- DFT Flow
- Conclusion
DFT Strategy: Design specification step

- **Digital core**
  - Scan
    - Fault models
    - Hierarchical approaches
    - Compression
    - Multi-mode
    - Power/Test time
    - Test clock
  - MBIST
    - RTL vs netlist level insertion
    - # of controller
    - Power
    - TAP or TAPless
    - Test clock

- **Ad-hoc**
  - Isolation test
  - BIST
  - Multi-site test
  - Parallel test

- **ATE**
  - Test options
  - Memory capacity
  - Available signaling speed
  - Vector format
  - Board design

- **IO**
  - Digital IO boundary scan
  - Analog IO
  - High speed IO
  - SERDES Tx/Rx

- **System level test**
  - Go/no-go test
  - HW/SW preparation
DFT Rule Check: RTL design step

- **Clock rules**
  - Do not switch on both edges of a clock
  - Do not use data as clocks
  - No "ANDing" of generated clocks
  - No logic in common with clock and data

- **Asynchronous rules**
  - Active phase of all set and reset pins buffer connected to the same root level pin must be the same level
  - Do not use flip-flops with both asynchronous set & reset
  - Set and reset lines on the same flip-flop should not be simultaneously active

- **Latch rules**
  - Do not use latches unless transparent during power ground mode
  - No combinational loops from transparent latches
  - No synchronous latches

- **Tri-state rules**
  - Do not infer tristate components
  - Tristate buses should have a pull-up or pull-down connection
  - Tristate bus enables must be fully decoded so that exactly one driver is active at any time
Scan clocking

➤ Stuck-at

➤ At-speed
Multiple scan clock

- **Advantages**
  - Compact test patterns
  - Short ATPG run time
- **Disadvantages**
  - Careful clock analysis
  - CTS for test mode
  - No flexibility

- **Advantages**
  - Safest approach
  - Flexibility
- **Disadvantages**
  - More sharing pins
  - Large patterns
  - Huge ATPG run time to reduce pattern size
Clock Domain Crossing

➢ Scan shift mode
  - Lock-up latch/Skew clock

➢ Capture mode
  - Clock domain Analysis: False path, Multi-cycle path, CDC
  - Capture by clock/Capture by clock group/Fine CTS for test clocks
Design constraints: Synthesis/P&R/STA

➢ At least 3 design constraints are required
  - Scan shift/Scan capture
  - MBIST

➢ Clock definition
  - Test clock and function clock
  - False path, multi-cycle path
  - On chip clocking
  - Test modes

➢ Hold timing violation
  - Huge area overhead

➢ MCMM is mandatory
Agenda

- General SoC Design Flow and Guide
- DFT & Design Flow
- DFT Flow
- Conclusion
Scan Compression

Source from MentorGraphics
Physically Aware ATPG

- Slack-aware tests: Small delay defects
  - ATPG selects observation path with lowest slack

- Layout-aware tests
  - Bridge defects, single via defects, crosstalk defects

Source from Magma
Low Power Test

- **Problem: Scan**
  - Typical ATPG will target as many faults as possible
  - May cause a large number of flip-flops (FF) toggling
  - May exceed the design power intent or limits
  - May cause unintended ATE failures, or worse, product damage

- **Problem: MBIST**
  - Typical MBIST will target as many memories as possible

- **Solution**
  - Clock gating (ATPG and DFT)
  - Low toggling (ATPG or DFT)
  - Trade-off between test time and power consumption (MBIST)
  - Dynamic power analysis for test modes
“It is not the strongest of the species to survive, nor the most intelligent, but rather the one most responsive to change.”