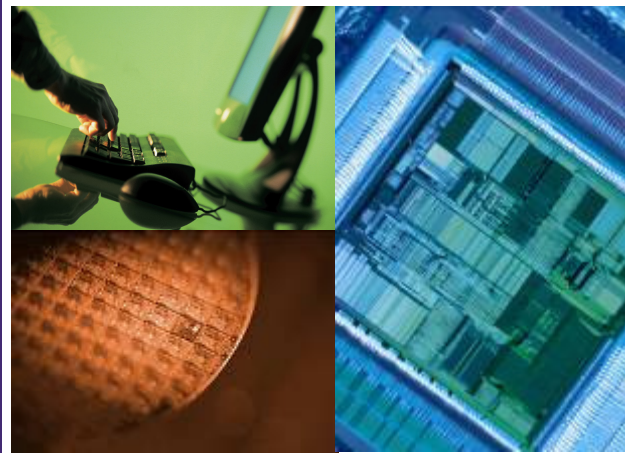


# Getting Started with UPF Tool Development

EDA Interoperability Developers' Forum  
October 25, 2007



Jim Sproch  
Sr. Director of R&D  
Synopsys, Inc.

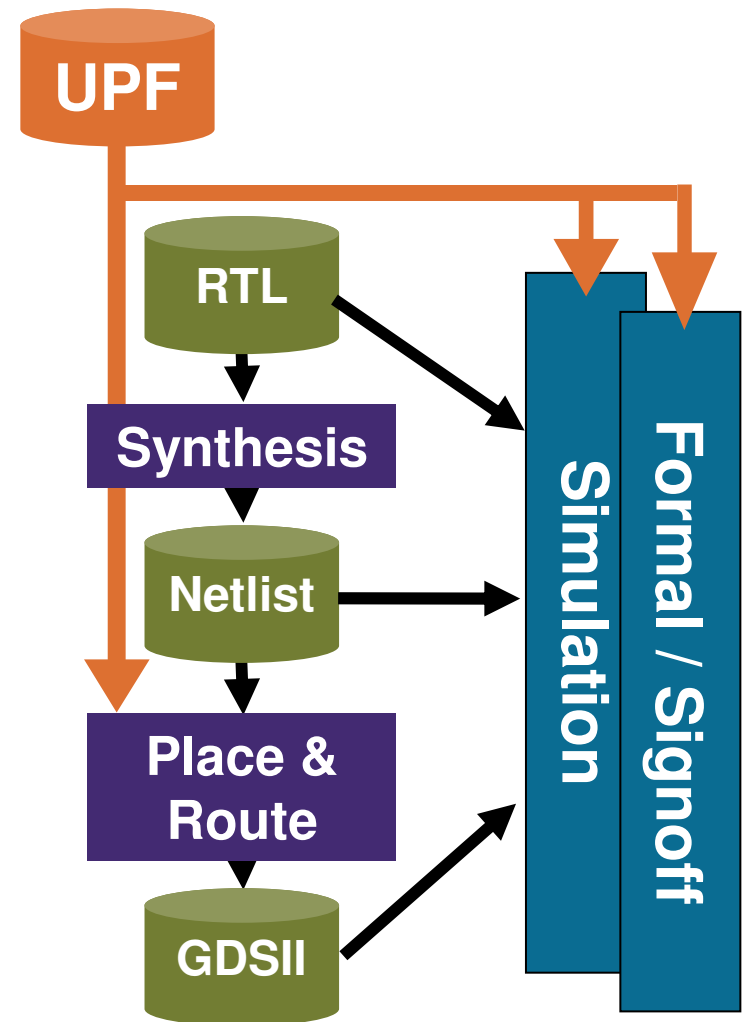
**SYNOPSYS**<sup>®</sup>  
Predictable Success

# Agenda

- UPF Low Power Design Flow
  - Power-aware Design Description
  - Side-file Syntax
- Data Model Updates
  - Power Domain Objects
  - Supply Nets
  - Power Component Instantiation
- Verification
  - Power-aware Rule Checking
  - Switching Activity
  - New Simulation Semantics
- References

# What Is UPF?

- **Unified Power Format**
- A single format serving entire low-power solution
- Extension of logic specification for low-power design intent
- Consistent semantics for implementation and verification

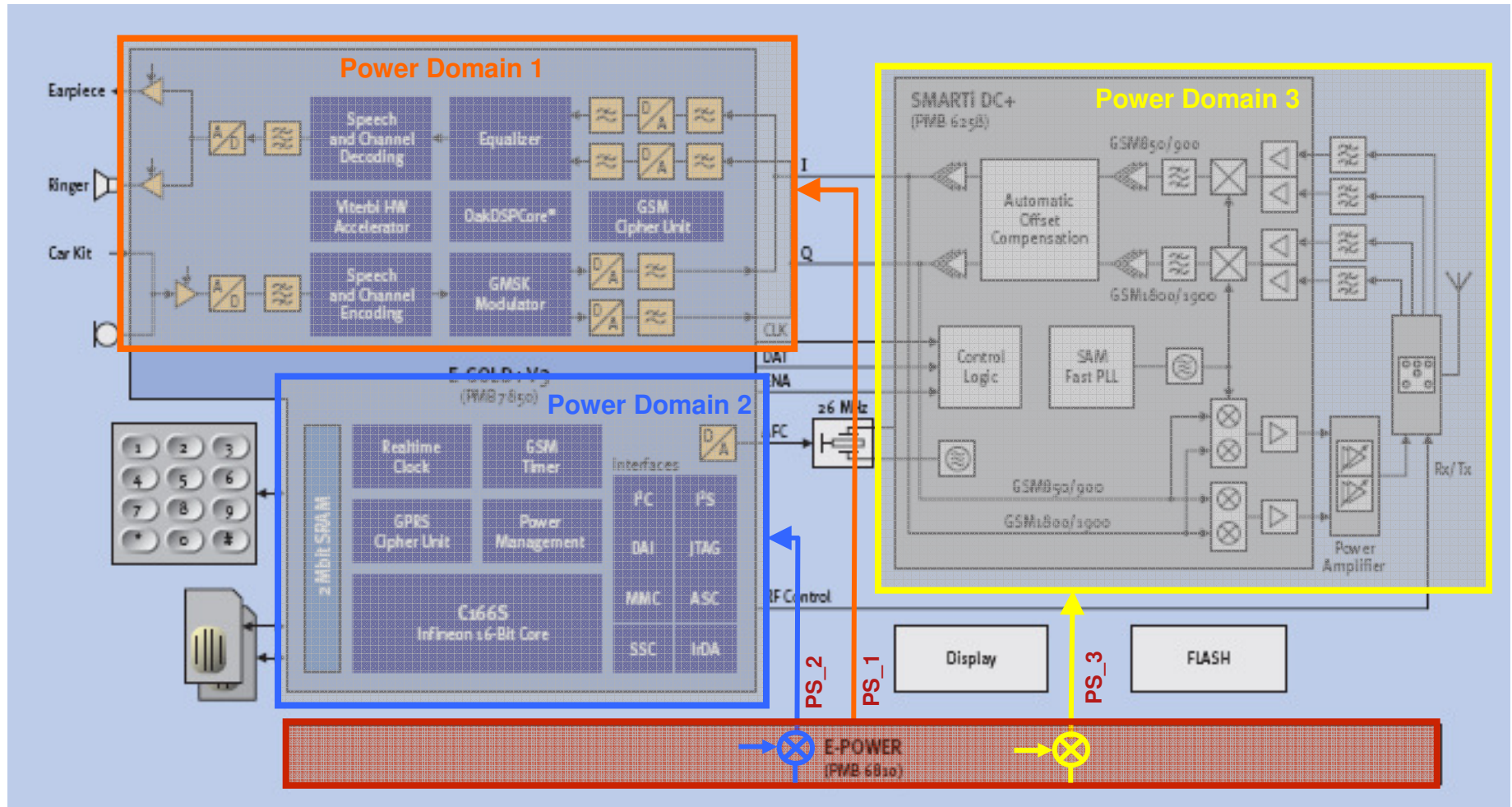


# UPF is a Hardware Description Language

- It describes hardware functionality
  - Like Verilog or VHDL
- It describes only a subset of hardware
  - Verilog does not allow description of power distribution
  - UPF only describes power distribution
- There is an enforced partition of role
  - UPF does not create functional logic signals
  - UPF creates power nets and power distribution objects

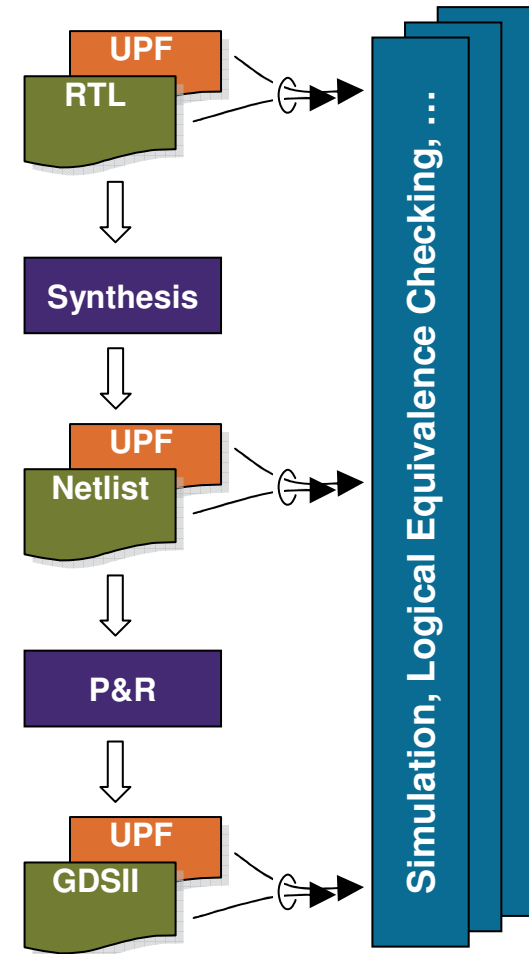
# UPF Conceptual Model

- Overlay power information on top of the design



# Complete Low Power Design Specification == HDL + UPF

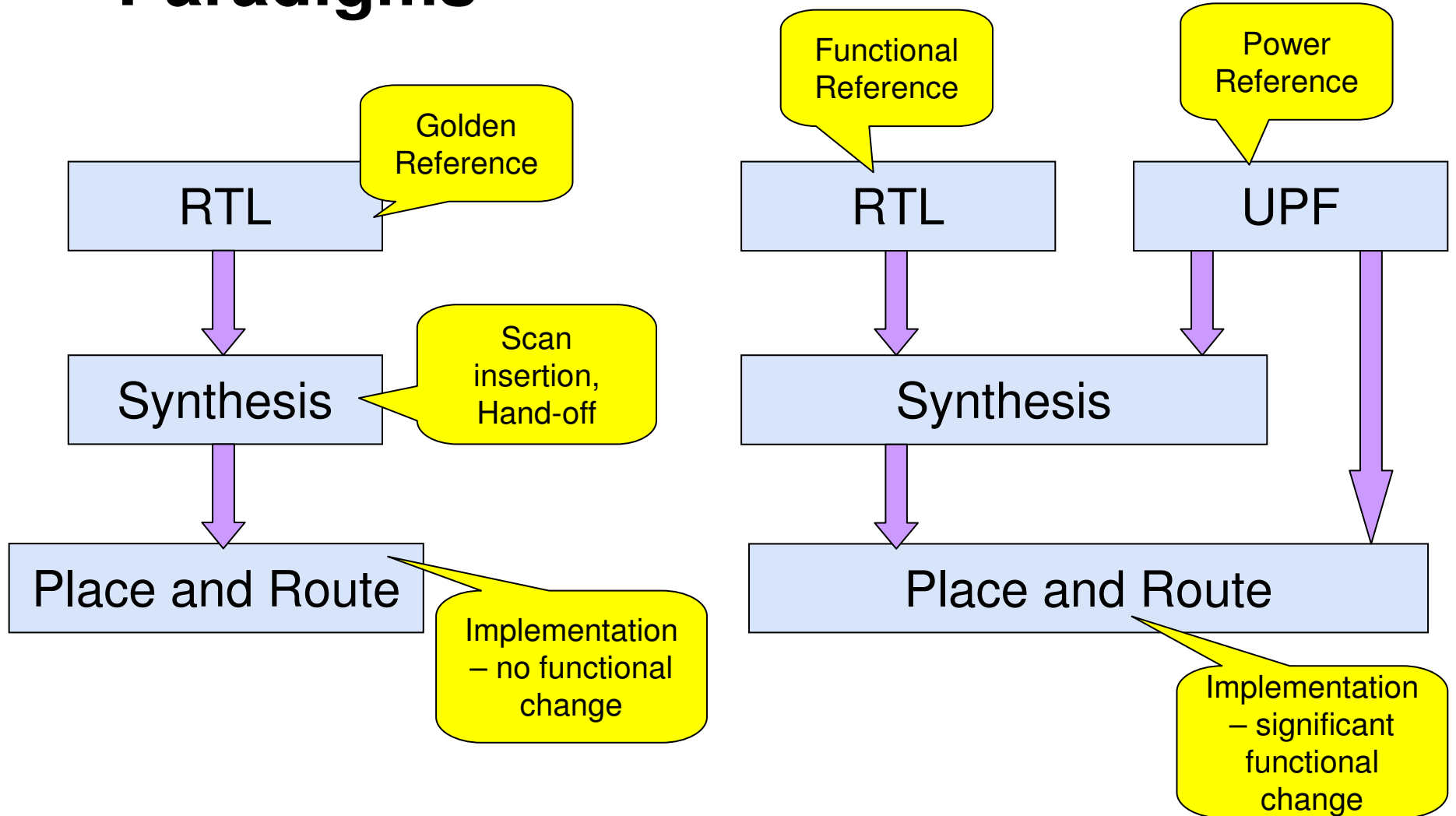
- Power Domains
- Power Distribution Network
  - Switches and Supply Nets
- Power State Table
- Level Shifting
- Isolation
- Retention
- Switching Activity



# Design Objects

- UPF creates
  - Power nets
  - Power ports
  - Power switches
  - Isolation cells
  - Level shifters
- UPF Connects Power Nets to
  - Power ports
  - Power switches
  - Isolation cells
  - Level shifters
  - Retention Flops
  - Standard cell logic
- UPF converts flops to retention flops
- UPF connects control signals to
  - Power switches
  - Isolation cells
  - Retention flops

# Paradigms



# Design Description Languages

- Functional Intent
  - RTL Hardware Description Languages
    - Verilog
    - SystemVerilog
    - VHDL
- Power Intent
  - UPF
    - Side file configuration
    - Industry-standard Tcl Syntax

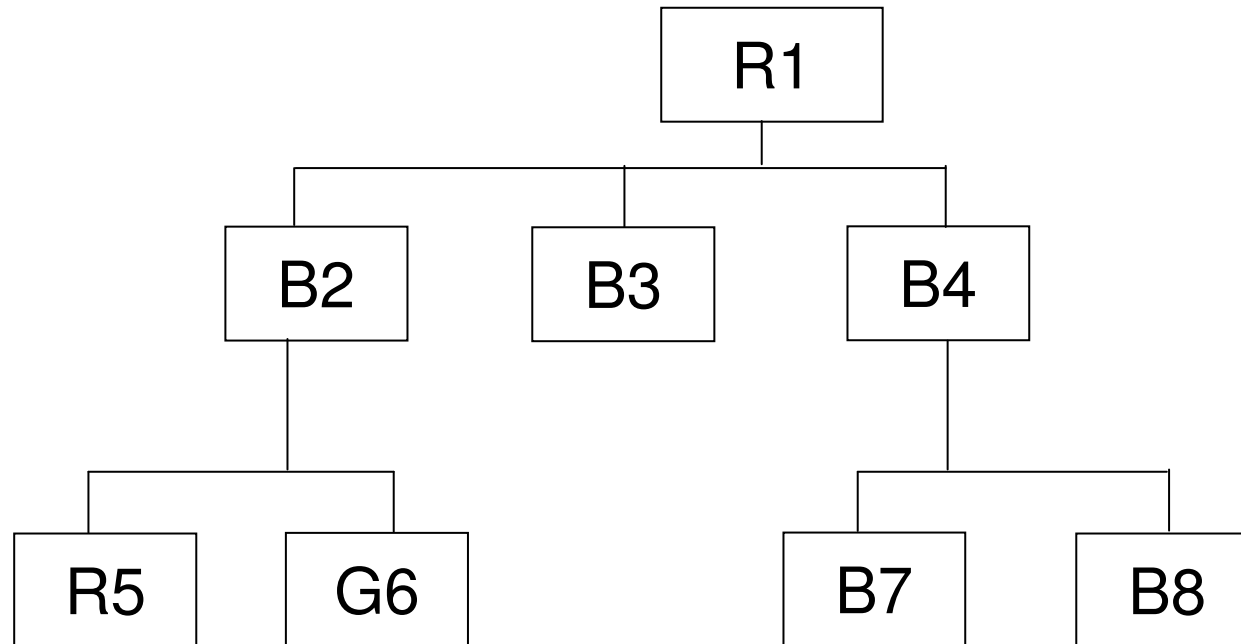
# UPF 1.0 Commands

## *Support All Aspects of Design and Verification*

<b>UPF Control</b>	load_upf save_upf upf_version
<b>Scope and Naming</b>	get_supply_net name_format set_design_top set_scope
<b>Power Distribution</b>	add_domain_elements connect_supply_net create_power_domain create_power_switch create_supply_net create_supply_port merge_power_domains set_domain_supply_net
<b>Isolation</b>	map_isolation_cell set_isolation set_isolation_control

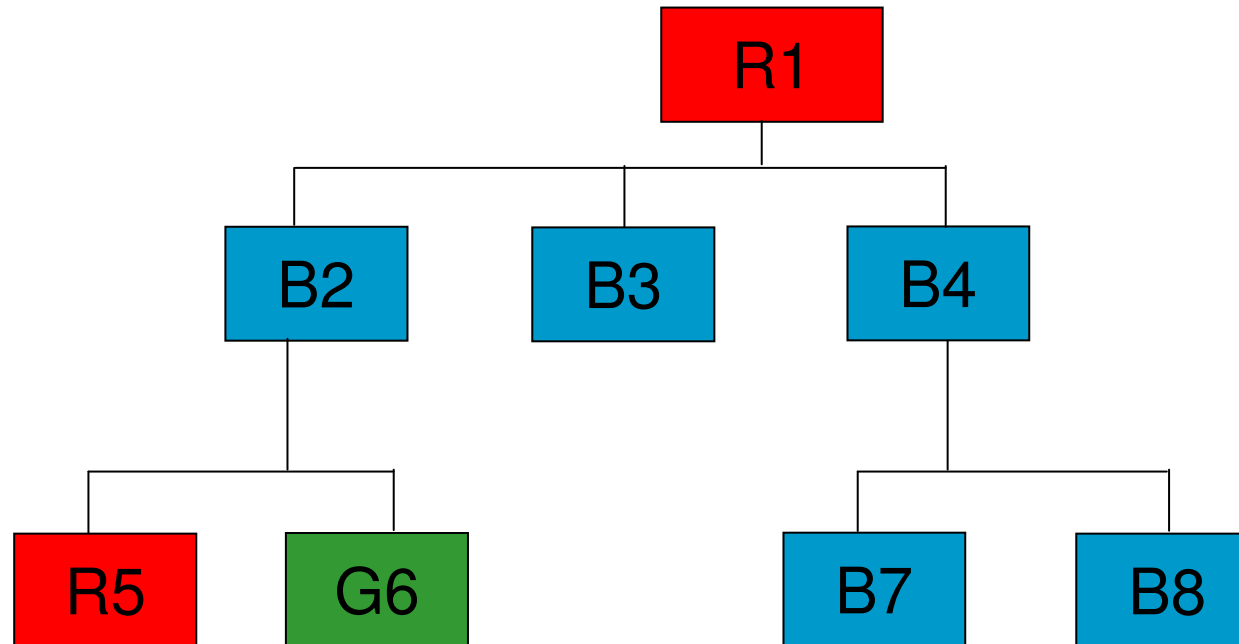
<b>Level Shifters</b>	map_level_shifter_cell set_level_shifter
<b>Power Strategy</b>	add_pst_state create_pst
<b>Power Switch</b>	map_power_switch set_power_switch
<b>Retention</b>	map_retention_cell set_retention set_retention_control
<b>Library</b>	set_pin_related_supply
<b>Verification Support</b>	add_port_state bind_checker create_hdl2upf_vct

# Logic Design Described by HDL



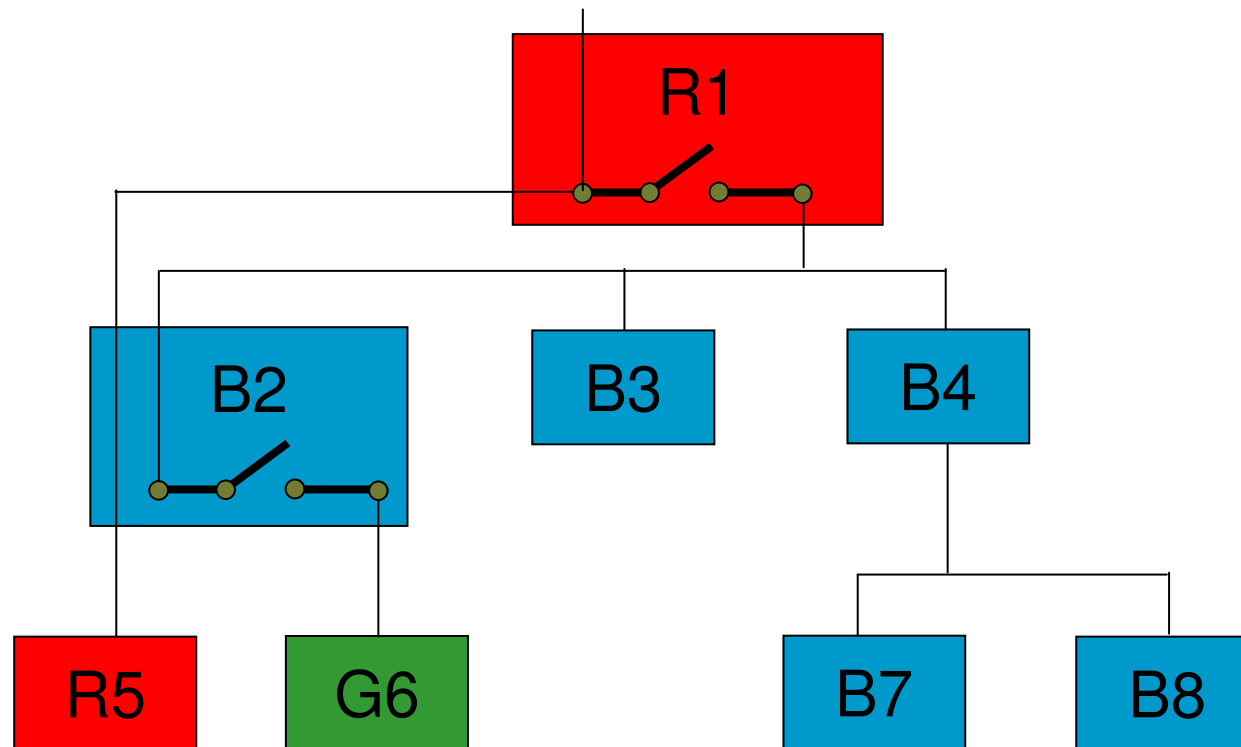
- Back in the days when there was only a single Vdd/Vss pair
- All power supply connections were implicit
- Everything was powered on all the time at a constant voltage

# Design Elements Colored by Power Domains



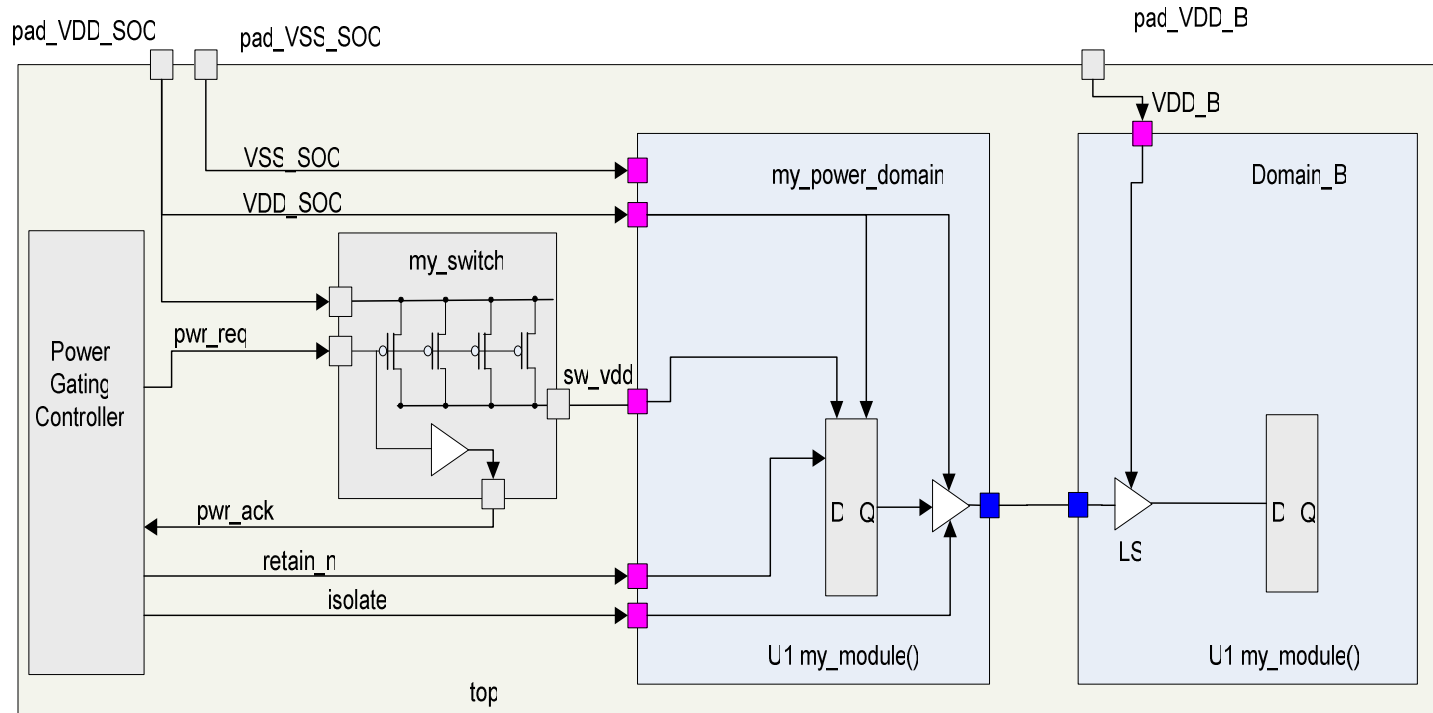
- Power-gated, multi-voltage designs break all of the assumptions
- Synthesis, simulation, implementation need consistent specifications
- There are many possible power implementations of the same logic design
- Abstract, concise specification of power aspects for high productivity
- Power view orthogonal to logical, physical, test, and other views

# Supply Distribution Network



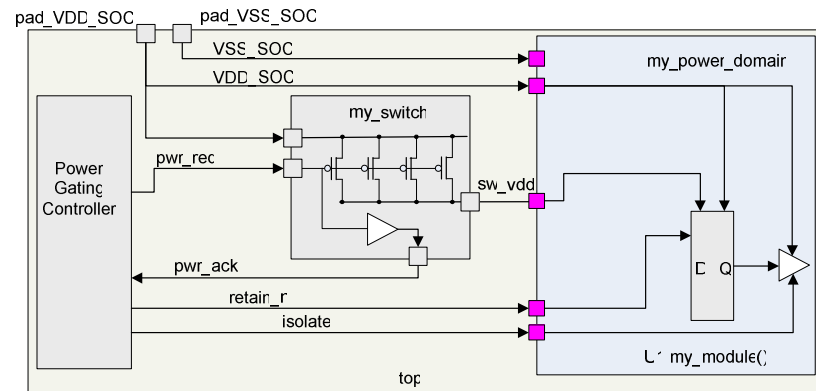
- Declare and connect switches to supply nets
- Switches and supply nets exist in the logical hierarchy
- Connectivity, port-punching, and renaming are automatic

# Goal – Support Low Power Design



- Created explicitly by UPF Commands
- Created implicitly by UPF Commands
- In the Original RTL

# Generic Power Gating Design



***set\_scope top***

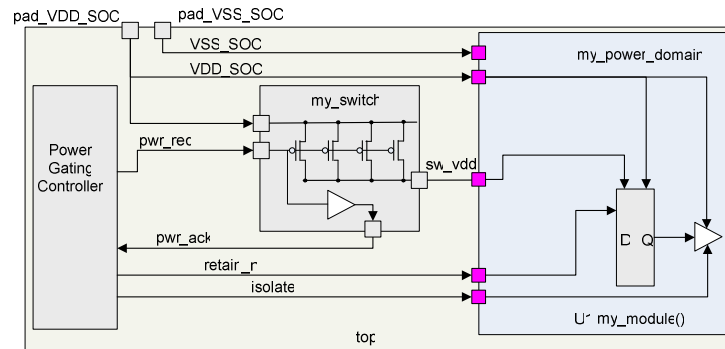
***create\_power\_domain top\_power\_domain -include\_scope***

***create\_supply\_net VDD\_SOC -domain top\_power\_domain***

***connect\_supply\_net VDD\_SOC -ports {pad\_VDD\_SOC}***

***create\_supply\_net VSS\_SOC -domain top\_power\_domain***

***connect\_supply\_net VSS\_SOC -ports {pad\_VSS\_SOC}***



```

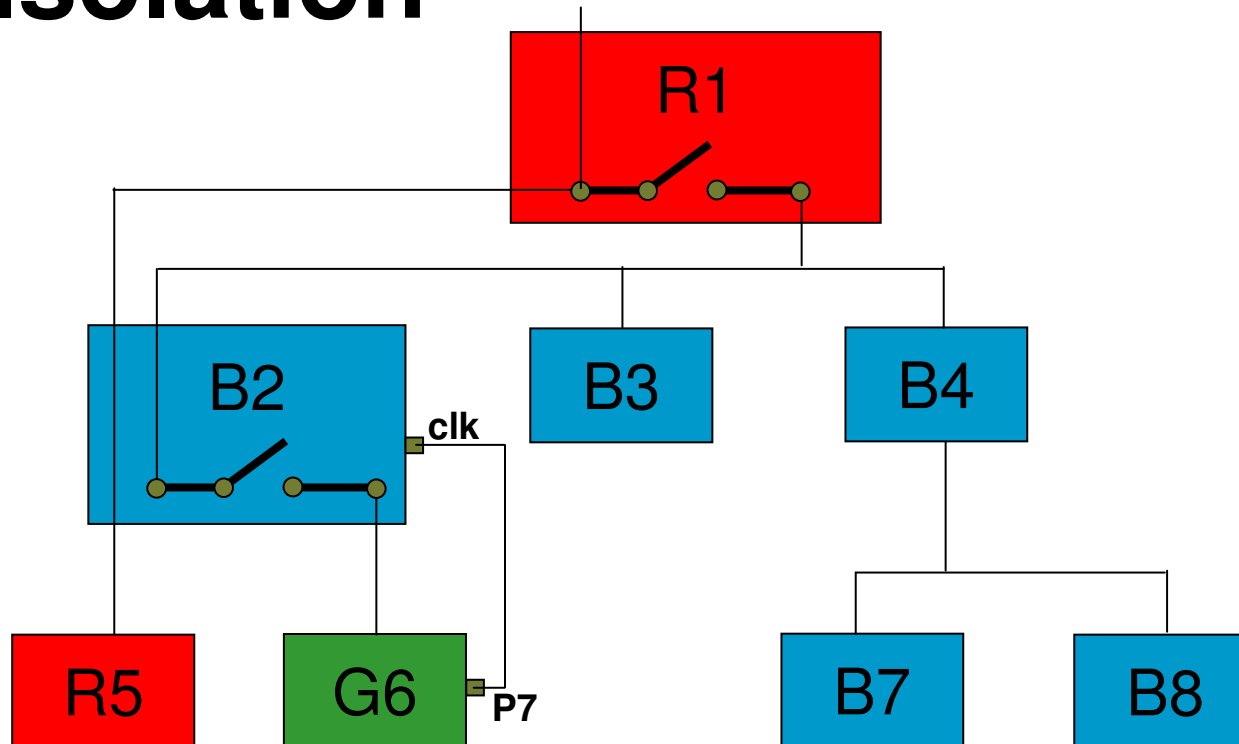
create_power_switch my_power_switch -domain my_power_domain
-input_supply_port {my_sw_input_port VDD_SOC}
-output_supply_port {my_sw_output_port sw_vdd}
-control_port {my_sw_control_port pwr_req}
-ack_port {my_ack_port pwr_ack}
-on_state {pwr_on_state my_input_port {my_sw_control_port == 1 }}
-off_state {pwr_off_state {my_sw_control_port == 0}}

```

# EDA Tool Data Models

- New Power Domain Object
  - Collection of power info for a set of design elements
- Hierarchical Supply Nets
  - Supply nets are no longer global !!!
  - Supply ports on power domain Interface
  - Supply net data type
  - Colored by power domain association
  - Multiple purposes
- Power Component Instantiation
  - Switches
  - Isolation
  - Level Shifters
  - Retention substitution

# Isolation



- Floating outputs of power-gated circuits  
Output port **G6/P7** connects to **B2/clk** input port
- Isolation control signals force known value
- Synthesis semantic requires an understanding of supply lifetime
- Simulation and verification semantic comprehends behavior change
- Things get more interesting when isolation and level shifting are merged

# set\_isolation Command Syntax

```
set_isolation isolation_name  
  -domain domain_name  
  <-isolation_power_net net_name  
    -isolation_ground_net net_name  
    | -no_isolation>  
  [-elements list]  
  [-clamp_value <0 | 1 | latch | Z>  
  [-applies_to <inputs | outputs | both>]
```

# set\_isolation Command Example

```
set_isolation iso3
```

```
-domain PDgreen
```

```
-isolation_power_net Vbu
```

```
-clamp_value 0
```

```
-applies_to outputs
```

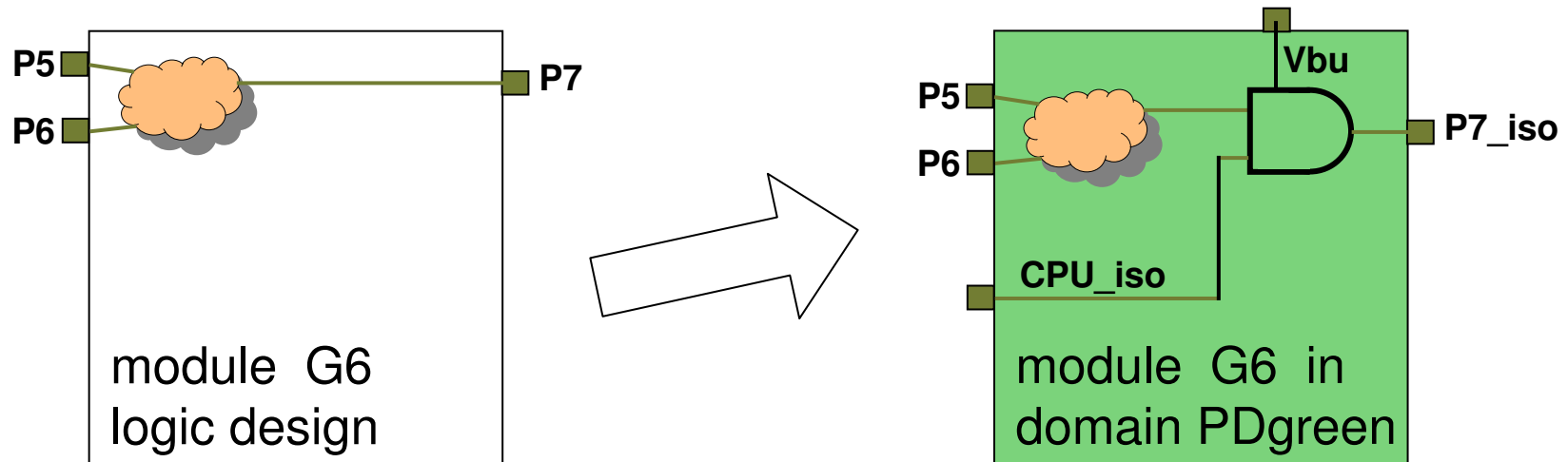
```
set_isolation_control iso3
```

```
-domain PDgreen
```

```
-isolation_signal CPU_iso
```

```
-isolation_sense low
```

```
-location self
```



# System power state specification

create_pst			pt		{	PS-red	PS-blue	R1/B2 /PS-green	}
add_pst_state	active	-pst	pt	-state	{	v13	v10	v09	}
add_pst_state	sleep	-pst	pt	-state	{	v10	v10	off	}
add_pst_state	standby	-pst	pt	-state	{	v13	v13	off	}

- Create a Power State Table
- Each row defines a valid combination of supply states
- Power states enable optimization and verification
  - **Power-aware Rule Checking**
  - Infer or verify level shifters and isolation gates

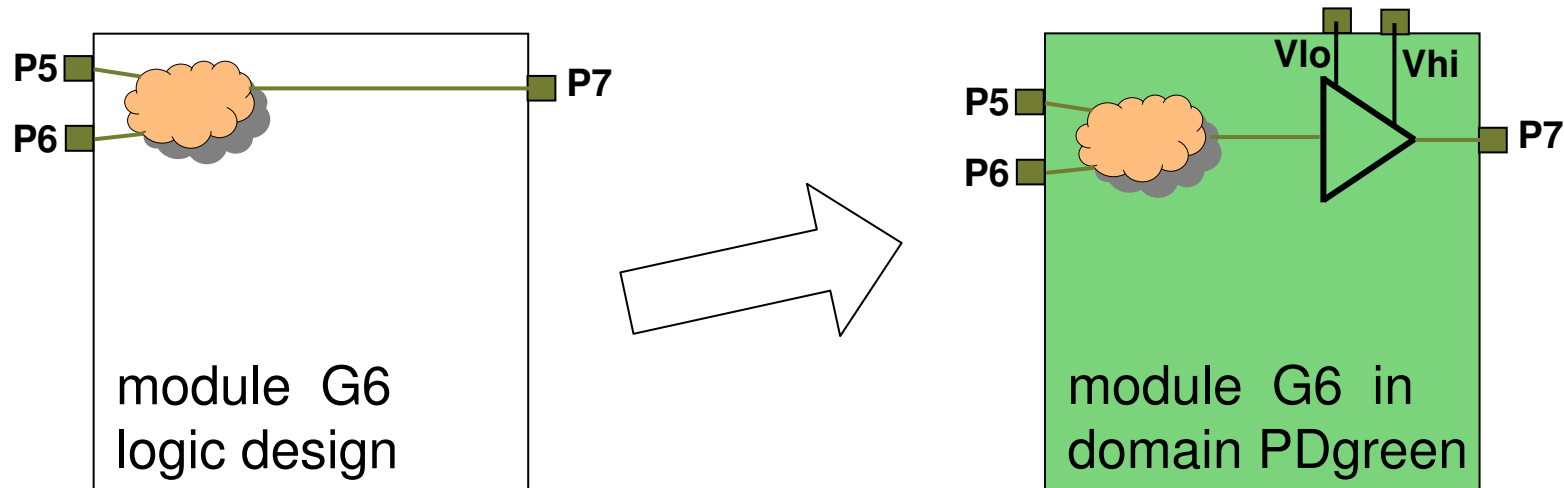
# Level shifting

- Level shifters translate from one voltage swing to another
- Manual insertion of level shifters is boring
- Specify an exciting strategy instead
  - Pick a power domain or a set of elements
  - Select input ports, output ports, or both
  - Tolerate a voltage difference threshold
  - UPshift or downSHIFT rule
  - Location (self, parent, sibling, fanout, auto)
  - Do or don't do it

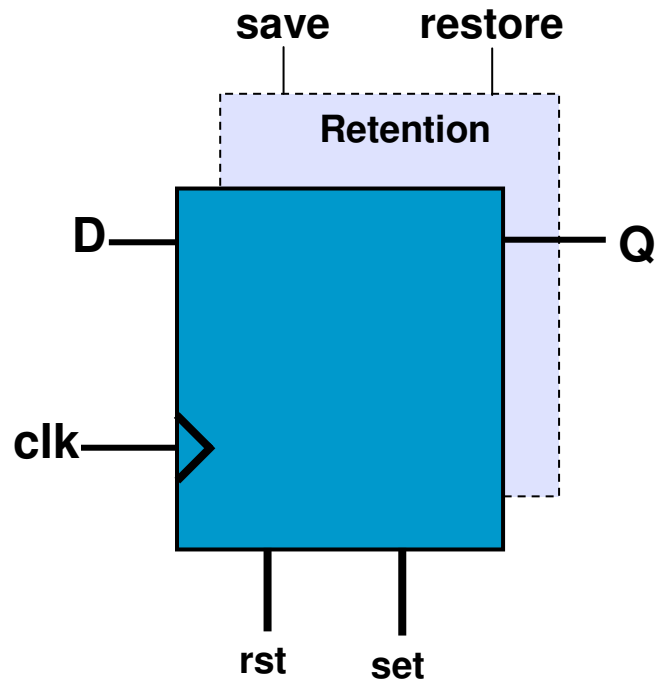
# set\_level\_shifter Command Example

```
set_level_shifter my_ls  
-domain PDgreen  
-rule low_to_high  
-location self  
-applies_to outputs
```

```
map_level_shifter_cell ls_L2H  
-domain PDgreen  
-lib_cells { /lib/ls_123 }
```



# State Retention for Latches and Flops

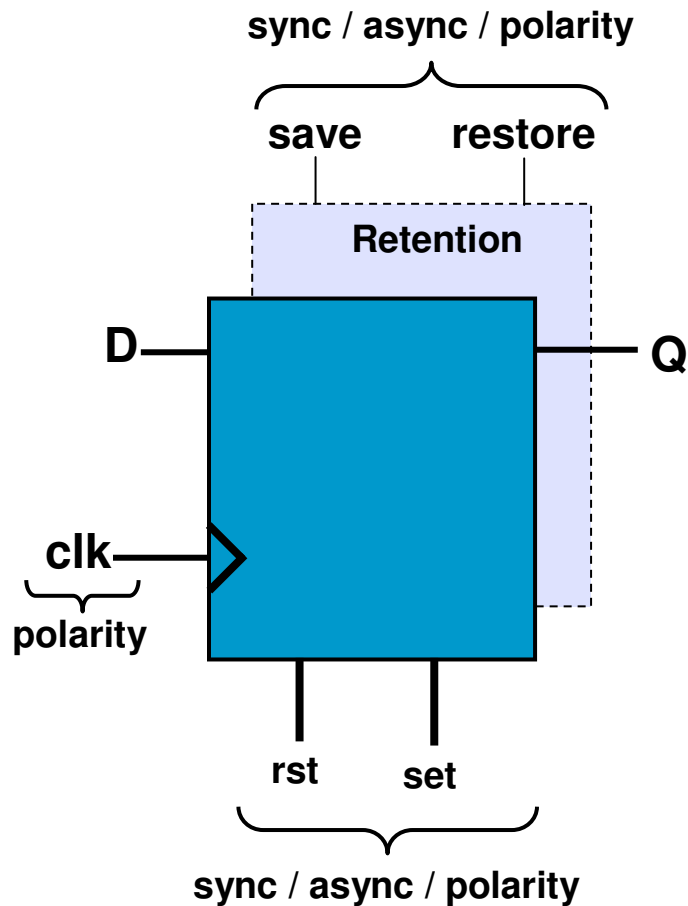


```
reg saveQ // shadow register
always @(posedge save) saveQ <= Q;

// Restore process
always @(negedge restore) Q <= saveQ;
```

```
// Normal operation
always @(posedge clk,
        rst, set,
        clear, preset)
    if (rst)
        Q <= '0';
    else
        if (set)
            Q <= '1';
        else
            Q <= D;
```

# Programmable State Retention



```
reg saveQ // shadow register
always @(posedge save) saveQ <= Q;

// Restore process
always @(negedge restore) Q <= saveQ;
```

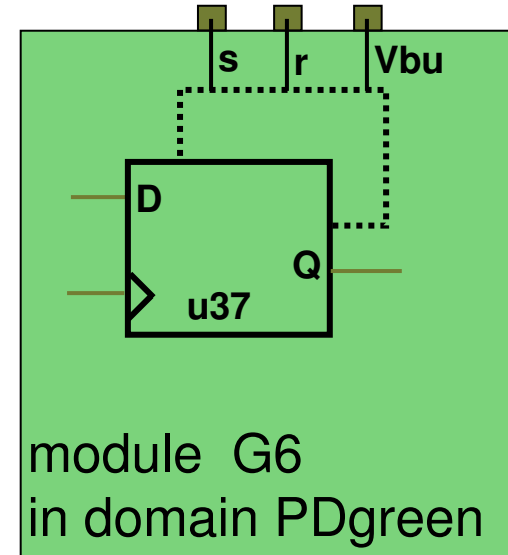
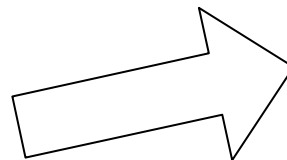
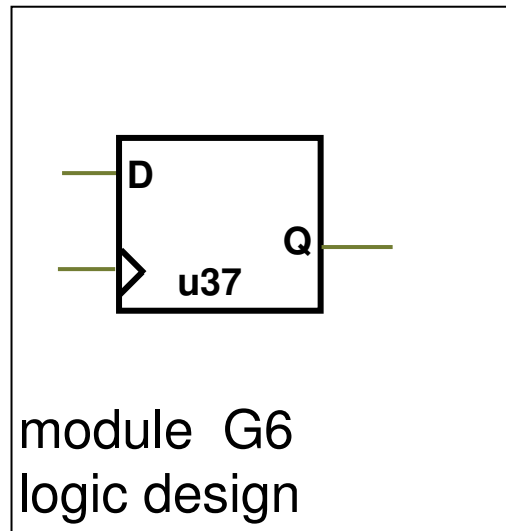
```
// Normal operation
always @(posedge clk,
        rst, set,
        clear, preset)
    if (rst)
        Q <= '0';
    else
        if (set)
            Q <= '1';
        else
            Q <= D;
```

```
// assert_rs_mutex
// assert_s_mutex
        (clk2 posedge)
```

# set\_retention Command Example

```
set_retention ret3
-domain PDgreen
-retention_power_net Vbu
-elements { u37 }
```

```
set_retention_control ret3
-domain PDgreen
-save_signal s
-restore_signal r
```



# SAIF Standard in UPF

- Switching Activity Interchange Format (SAIF)
- Allows the exchange of activity information among tools
  - Transition density
  - Static probability
  - Glitch effects
- Production-proven technology
  - Simulators
  - Analysis tools
  - Implementation tools
  - Reliability tools

# New Simulation Semantics

- New Supply Net Data Type
  - ON
  - OFF
  - Partial-On
  - Voltage level
    - Microvolts to Megavolts
- Signal value corruption
  - During module power-down
- Supply Net Resolution
  - Multiple switches connected to the same supply net
- Isolation Behavior
- Level Shifters
  - Voltage swing
  - No change in logic behavior
- Retention – Save and Restore Protocol
  - Waking up is hard to do

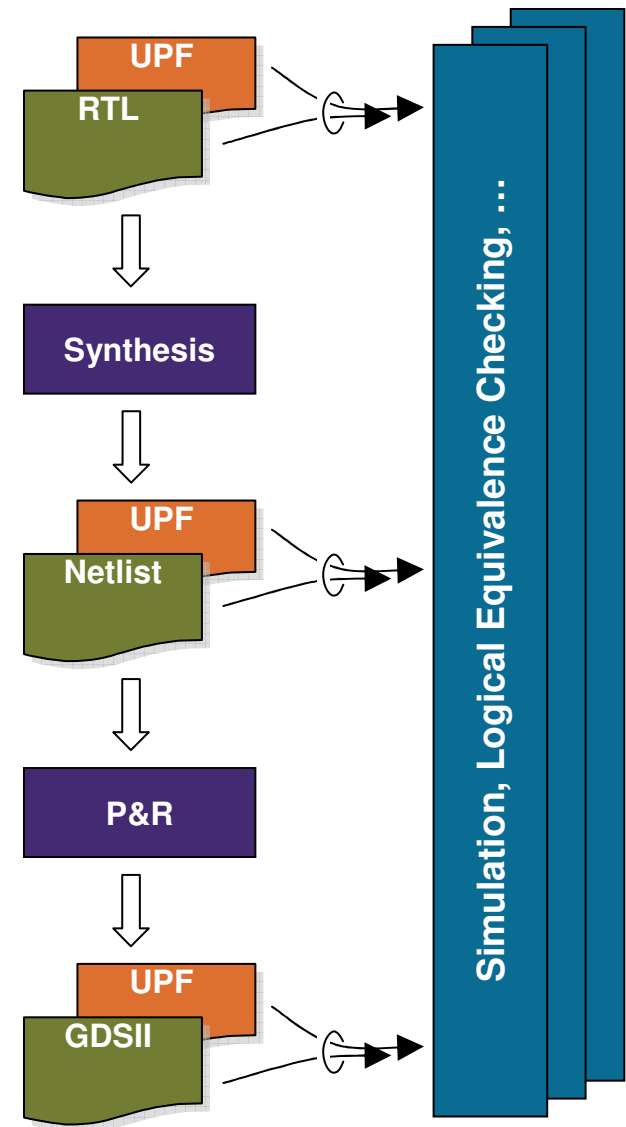
# How Can I Get UPF?

- Accellera UPF Version 1.0 Standard
- IEEE P1801 Low Power Design Working Group
- Hypermail archive, Mantis (bug tracking), etc.
- [http://www.accellera.org/activities/p1801\\_upf/](http://www.accellera.org/activities/p1801_upf/)

# Q & A

# UPF Review

- Unified Power Format
  - Abstract supply distribution and control network specification
  - Power-aware design intent
  - Used throughout design flow
- *Extends without changing the logic design specification*
  - Golden source is not touched
  - No re-verification of logic-only
  - UPF augments the HDL specification
- *Consistent simulation & implementation semantics*



# The Future of UPF

- UPF Accellera Standard as of February 2007
- IEEE P1801 Low Power Format working group
- Working group comprised of members from semiconductor companies, Accellera, etc.
- UPF is part of the technology foundation
- IEEE standard is expected to supersede UPF

**EDA Standards activity  
is alive *and competitive.***