CPF 2.0 Tutorial

Anmol Mathur  Calypto Design Systems
Prasad Subbarao  LSI
Steve Urish  IBM
Qi Wang  Cadence Design Systems

March 2011
Motivations for CPF 2.0 Extension

- Improving Interoperability from IEEE 1801 to CPF
  - Constructs in **BLUE**

- New Features for Advanced Low Power Design
  - Constructs in **RED**
Innovation Through Collaboration

Agenda

- **Generic mode vs power mode**  
  Anmol Mathur
- Improved hierarchical flow
- Macro model enhancements
  - Voltage regulator, mixed-signal IP, IO ring
- Simulation related enhancements
  - simulation semantics for pure behavioral constructs
- Modeling new low power IPs
  - Multi-bit level shifter and combo cells, clamp cells, ground shifters, etc
- Rules related enhancements
- Misc enhancements
  - Links to .lib pg_type attribute
  - find_design_objects
Need for Generic Modes

- Generic modes provide a mechanism to specify functional states of a design that are relevant to power analysis and optimization.
- Generic modes are different from power modes that specify the power states of all the power domains in the design:
  - The same power mode may be associated with multiple generic modes.
  - One generic mode may span multiple power modes (although this is probably rare).
- Generic modes can be used as a mechanism to specify key functional modes of a design, along with information needed to use this information for power analysis and optimization:
  - Functional constraints associated with generic mode.
  - Probability of design being in a generic mode in a typical workload.

Typical Workload:

- Idle: 65%
- Audio = 15%
- Video = 20%
Examples of Generic Modes

- Designs are often in specific modes for long durations of times
  - “normal” mode vs reset/test mode – resets and test enables are inactive
  - “idle/standby” mode vs “active” mode – often some global “idle_mode” signal is an input to the design and can be used to do coarse-grained clock gating
  - Designs with multiple functional modes (typically programmed via configuration registers) are common
    - Multi-media IP could be in audio mode or video mode or just processing still pictures
Generic Mode Syntax

Command Syntax

```
create_mode -name string -condition expr [-probability float] [-illegal]
```

Options and Arguments

- `condition expr`: The condition when the design is in the specified mode, described by a Boolean or arithmetic expression of the following objects:
  - design pins and ports
  - domain conditions in the form of `power_domain@nominal_condition`
  - other mode or power mode definitions in the form of `@mode`

- `illegal`: Identifies the mode as illegal. By default, a mode is legal.

- `name string`: Specifies the name of the mode. The name of a mode must be unique among other modes and power modes within the same design scope.

- `probability float`: Specifies a floating point value between 0 and 1 to indicate the probability of the design being in this mode.

Examples

```
create_mode -name idle -condition {((mode_reg == IDLE) & PD_ON@ON & PD_audio@sleep & PD_video@sleep)} -probability 0.3
```

- This is an example of a generic mode related to a power domain being in a particular state

```
create_mode -name audio -condition {((mode_reg == AUDIO_ONLY) & PD_ON@ON & PD_audio@ON)} -probability 0.7
```

- Design is in a functional state where only an audio stream is being processed and this is indicated by the mode_reg being in a specific state
Power Modes and Generic Modes

- Power modes of design -- default domain is always on, PD_audio and PD_video has ON and sleep nominal conditions
  - PM_idle : {PD_ON@ON PD_audio@sleep PD_video@sleep}
  - PM_audio_only : {PD_ON@ON PD_audio@ON PD_video@sleep}
  - PM_audio_video : {PD_ON@ON PD_audio@ON PD_video@ON}

- Generic modes of design
  - Idle : {(mode_reg == IDLE) & PD_ON@ON & PD_audio@sleep & PD_video@sleep}
    - This generic mode is a refinement of the PM_idle power mode
  - Audio: {(mode_reg == AUDIO_ONLY) & PD_ON@ON & PD_audio@ON}
    - This generic mode can be active in both PM_audio_only and PM_audio_video power modes
  - Video : {(mode_reg == VIDEO) & PD_ON@ON & PD_AUDIO@ON & PD_VIDEO@ON}
CPF of Generic Mode Example

set_design multimedia_IP

create_nominal_condition –name sleep –voltage 0.6 –state standby
create_nominal_condition –name ON –voltage 1.0 –state on
create_power_domain –name PD_ON -default
create_power_domain –name PD_audio -instance “audio_unit”
create_power_domain –name PD_video –instance “video_unit”

## create power modes

create_power_mode –name PM_idle –domain_conditions {PD_ON@ON PD_audio@sleep PD_video@sleep}
create_power_mode –name PM_audio_only –domain_conditions {PD_ON@ON PD_audio@ON PD_video@sleep}
create_power_mode –name PM_audio_video –domain_conditions {PD_ON@ON PD_audio@ON PD_video@ON}

## create generic modes
create_mode -name idle -condition {((mode_reg == IDLE) & @PM_idle) –probability 0.3
create_mode -name audio -condition {((mode_reg == AUDIO_ONLY) & PD_ON@ON & PD_audio@ON) –probability 0.4
create_mode –name video –condition {((mode_reg == VIDEO) & @PM_audio_video} –probability 0.3

.....
end_design
Typical Generic Mode Usage

Specification of typical or maximum workloads for average or peak power analysis/power optimization of a design is a major problem in current power aware flows

- Design teams mostly have functional simulation traces aimed at exercising different functional states – not representative of typical workload

System designers do have information about the key functional modes in the design and how often the design is expected to be in these modes

- This information can be captured using generic modes

Generic modes provide a mechanism to specify functional constraints (via the condition associated with generic mode) and probability of the design being in that mode

- This can be used as a high-level specification of design activity by power analysis and optimization tools
- Functional constraints on key design registers/inputs are crucial for good power analysis/optimization of the design
What Generic Mode Will Not Do?

- **Generic mode shall not be used to drive implementation**
  - Generic mode cannot be associated to an analysis view
  - Implementation relies on analysis view to define multiple scenarios for implementation and optimization
Mode Transitions

Command Syntax

create_mode_transition -name string -from mode -to mode
[-assertions assertion_list]
{ -start_condition expression [-end_condition expression]
 [ -cycles [integer:]integer -clock_pin clock_pin
 | -latency [float:]float ]
 | -illegal }

Change in CPF 2.0

-illegal option has been added to explicitly state that a particular mode transition is not allowed. Simulators and other functional analysis tools can use this to error out if an illegal mode transition occurs.

Mode transitions can be specified for both generic and power modes.

Specification of mode transitions between generic modes allow optimization tools to extract functional and timing constraints associated with mode transitions (such as transition of an IP from sleep mode to normal mode and vice versa).
Change to Power Mode

Command Syntax

```
create_power_mode
   -name string [-default]
   {-domain_conditions domain_condition_list
    |-group_modes group_mode_list
    |-domain_conditions domain_condition_list -group_modes group_mode_list }
   [-condition expression]
```

Change in CPF 2.0
- condition option has been added

The expression specified in the –condition option is a Boolean function of pins and ports in the design. This associates a Boolean condition that is satisfied by the design when it is in the power mode.
Agenda

- Generic mode vs power mode
- Improved hierarchical flow
  - Steve Urish
- Macro model enhancements
  - Voltage regulator, mixed-signal IP, IO ring
- Simulation related enhancements
  - Simulation semantics for pure behavioral constructs
- Modeling new low power IPs
  - Multi-bit level shifter and combo cells, clamp cells, ground shifters, etc
- Rules related enhancements
- Misc enhancements
  - Links to .lib pg_type attribute
  - find_design_objects
Hierarchical Flow Enhancements

- **Power Design**
  - New CPF Object
  - A power model that can be bound to a logic modules
  - Provides better flexibility for reuse
  - A logic module may be compatible with one or more Power Designs
  - A Power Design may be compatible with one or logic modules

```bash
set_design power_design -modules list_of_modules
```
Multiple Power Designs for a Module

Verilog

```verilog
module uProcessor ();
...
endmodule
```

CPF

```cpf
set_design HIGHER_VDD
...
end_design

set_design ON_OFF
...
end_design

set_instance inst_uProcessor1 -design ON_OFF
set_instance inst_uProcessor2 -design HIGER_VDD
```

Two possible power designs for uProcessor

Bind a power design to an instance of a module using `set_instance`
One Power Design for Many Modules

Verilog

```verilog
module uProcessor_16b ();
...
endmodule
module uProcessor_32b ();
...
endmodule
```

CPF

```cpf
set_design HIGHER_VDD -modules "uProcessor_16b uProcessor_32b"
...
end_design
set_instance inst_uP16b -design HIGHER_VDD
set_instance inst_uP32b -design HIGHER_VDD
```

A single set_design can describe a power design for multiple module.

-modules option forces strict checking. This power design may only bind to instances of these modules.

set_instance binds the power design, but inst_uP16b and inst_uP32b must an instance of uProcessor_16b or uProcessor_32b.
Updating Power Design

- **update_design** *power_design_name*
  - A cleaner way to append to a power design
  - In CPF 1.1, a power design could be appended using another *set_design* but only if the power design was unbound.

```plaintext
set_design my_power_design
...
end_design

set_instance my_instance -design my_power_design

update_design my_power_design
...
end_design
```

*update_design will update the power model, even when bound to an instance.*
Virtual Ports

- `set_design -ports list_of_ports`
  - Provides a way to define new logic ports required for the power design – retention enable, isolation enable, etc...
  - In CPF 1.1, -ports only created input port

- **In CPF 2.0,**
  - New Command
    ```
    set_design power_design -input_ports port_list
    -output_ports port_list -inout_ports port_list
    ```
  - -ports is still supported, but is equivalent to -input_ports
Innovation Through Collaboration

Agenda

- Generic mode vs power mode
- Improved hierarchical flow
- Macro model enhancements
  - Voltage regulator, mixed-signal IP, IO ring
- Simulation related enhancements
  - simulation semantics for pure behavioral constructs
- Modeling new low power IPs
  - Multi-bit level shifter and combo cells, clamp cells, ground shifters, etc
- Rules related enhancements
- Misc enhancements
  - Links to .lib pg_type attribute
  - find_design_objects
Current ways to model IO
workable but tedious, non-intuitive, error prone

```
set_macro_model fooPad
create_power_domain –name COREIN –default -boundary_ports {din}
update_power_domain –name COREIN –primary_power_net CVDD \ 
              -primary_ground_net CVSS
create_power_domain –name IO –boundary_ports {pad}
update_power_domain –name IO –primary_power_net IOVDD \ 
              -primary_ground_net IOVSS
create_power_domain –name COREOUT
update_power_domain –name COREOUT –primary_power_net OVDD \ 
              -primary_ground_net OVSS
end_macro_model

set_design top
create_power_domain –name PD_PAD \ 
            –boundary_ports {padi_*/pad out*}
create_power_domain –name PD_CORE
set_instance pads { padi_0 padi_1 .... padi_127}
foreach i $pads {
    set_instance $i –domain_mapping \ 
        -mapping { {IO PD_PAD} {COREOUT PD_PAD} {COREIN PD_CORE}}
include fooPad.cpf
}
```
IO PAD Modeling in CPF 2.0

CPF 1.1 Approach

CPF 2.0 Approach

```
define_related_power_pins -cells {PDISDGZ PDDDGZ PDIDGZ} \ -data_pins C \ -power VDD -ground VSS
define_related_power_pins -cells {PDISDGZ PDDDGZ PDIDGZ} \ -data_pins {PAD POC} \ -power VDDPST -ground VSSPST
define_related_power_pins -cells {PDU08DGZ PDU16DGZ PDD08DGZ} \ -data_pins {PAD POC} \ -power VDDPST -ground VSSPST
define_related_power_pins -cells {PAD_CEC_A} \ -data_pins {CEC_A POC} \ -power VDDPST -ground VSSPST
define_related_power_pins -cells {PAD_tsmc_st} \ -data_pins {PAD} \ -power VDDPST -ground VSSPST
define_related_power_pins -cells {DDC_I2C_scan} \ -data_pins {PAD} \ -power VDDPST -ground VSSPST
define_related_power_pins -cells {PVDD1CDGx3_Switch_5mA_x2} \ -data_pins {POC} \ -power VDDPST -ground VSSPST
define_related_power_pins -cells {PVSS1DGZ PVDD2POC PVSS2DGZ PVDD2ANA PVDD2DGZ PVDD1DGZ} \ -data_pins {POC} \ -power VDDPST -ground VSSPST
```

```
## does not specify group for pins
## {PAD POCVDDPST VSSPST} which fall into the
## DEFAULT group

define_pad_cells -cells {PDISDGZ PDDDGZ PDIDGZ} \ -pad_pins {PAD} \ -pin_groups { CORE:{C VDD VSS} }
define_pad_cells -cells {PDU08DGZ PDU16DGZ PDD08DGZ} \ -pad_pins {PAD} \ -pin_groups { CORE:{C I OEN VDD VSS} }
define_pad_cells -cells {PAD_CEC_A} \ -pad_pins {PAD} \ -pin_groups { CORE:{C I OEN VDD VSS} }
define_pad_cells -cells {PAD_tsmc_st} \ -pad_pins {PAD} \ -pin_groups { CORE:{c sel i oen pe pu VDD VSS} }
define_pad_cells -cells {DDC_I2C_scan} \ -pad_pins {PAD} \ -pin_groups { CORE:{c scan_* wake_up sel i oen i VDD VSS} }
define_pad_cells -cells {PVDD1CDGx3_Switch_5mA_x2} \ -pad_pins {POC} \ -pin_groups { CORE:{ctrl_PS VDD VSS } }
define_pad_cells -cells {PVDD2POC} \ -pad_pins {POC}
```
IO PAD Modeling in CPF 2.0

create_power_domain -name PDaon_core
   -instances {u_tx_top/u_tx_io/u_always_on}
   -boundary_ports {u_tx_top/u_tx_io/u_TMODE_2/C}
   u_tx_top/u_tx_io/u_RESET_N_6/C
      u_tx_top/u_tx_io/u_HPD_3/c u_tx_top/u_tx_io/u_INT_5/C
      u_tx_top/u_tx_io/u_INT_5/I u_tx_top/u_tx_io/u_INT_5/OEN

#create_power_domain -name PDvddq
create_power_domain -name PDio
   -boundary_ports {TMODE INT HPD RESET_N
      SD2 SD1 SD0 WS DSDA DSCL CEC_D D2
      D1 D0 HSYNC VSYNC SPDIF MCLK SD3 D10 D9 D8 D7 D6 D5 D4 D3 D18 D17
      D16 D15 D14 D13 D12 D11 CEC_A CSCL CSA D23 D22 D21 D20 D19
      IO_STRAP CI2CA IDCK DE SCK POC POC1
      u_tx_top/u_tx_io/u_D19_61/PAD u_tx_top/u_tx_io/u_D20_62/PAD
      u_tx_top/u_tx_io/u_D21_63/PAD u_tx_top/u_tx_io/u_D22_64/PAD
      .... 57 more lines here
      u_tx_top/u_tx_io/u_gnd18v_15/POC
      u_tx_top/u_tx_io/u_VDDQ2_79/POC u_tx_top/u_tx_io/u_VDDQ1_78/POC
      u_tx_top/u_tx_io/u_vcc18v_67/POC u_tx_top/u_tx_io/u_vcc18v_66/POC
      u_tx_top/u_tx_io/u_vcc18v_45/POC u_tx_top/u_tx_io/u_vcc18v_18/POC
      u_tx_top/u_tx_io/u_vcc18v_17/POC
      u_tx_top/u_tx_io/u_vcc12v_switch_12/POC
      u_tx_top/u_tx_io/u_vcc12v_switch_21/POC
      u_tx_top/u_tx_io/u_vcc12v_switch_57/POC
      u_tx_top/u_tx_io/u_vcc12v_switch_73/POC
      u_tx_top/u_tx_io/u_vcc12v_bottom/POC
      u_tx_top/u_tx_io/u_vcc12v_4/POC
   }

create_pad_rules –name rule1
   –of_bond_ports *
   -mapping { {CORE PDaop_core}
      {DEFAULT PDio} }

CPF 1.1 Approach

CPF 2.0 Approach
New PAD Commands

create_pad_rule -name string
   {-of_bond_ports port_list / -instances instance_list}
   -mapping mapping_list

-of_bound_ports
   Selects pad cell instances that are connected to the specified list of top-level bonding ports. A selected instance must be an instantiation of a pad cell or a CPF macro model.

-instances
   Specifies a list of instances of pad cells.

-mapping
   Specifies the mapping of each pin group of the pad cell definition or a power domain in the macro model definition to a top-level domain.
   Use the following format to specify a mapping:
      {group_id_or_macro_model_domain power_domain}

Examples:
1. Use pad cell definition, see previous
2. Use macro model definition, see next
IO PAD Modeling Example 2

set_design top

create_power_domain –name PD_PAD \ –boundary_ports {padi_*/pad out*}

create_power_domain –name PD_CORE

create_pad_rule –name pad1 –of_bond_ports {out[*]} \ -mapping { {IO PD_PAD} {COREOUT PD_PAD} {COREIN PD_CORE}}
Can your pad cell be fully modeled by `define_pad_cells`?

**No**

The cell does not have:
1. internal power switch and/or state retention, and
2. complex internal isolation control, and
3. internal feed-through nets for data signals

- Use macro model for the cell modeling
- Use `create_pd_rule` for model instantiation
- Use `set_instance` for model instantiation

**Yes**

- Use `define_pad_cells` for the cell modeling
- Use `create_pad_rule` for cell instantiation

**New commands**

**Old CPF 1.1 approach**
● Current CPF 1.1 macro model syntax and semantics cannot accurately describe a voltage regulator and how it is used at chip level
  ◆ need to specify a range of voltages
  ◆ need to specify the reference supply
  ◆ need to propagate the supply attributes from the block level to the top level
  ◆ need to allow the output of a VR can be used as the source for multiple power domains at the top level
Voltage Regulator Example

2.5V input
HAVDD

1.1V input
AVDD
AVSS

C1
C2
VBB – variable output
1.1V, 1.2V, 1.3V

set_macro_mode regulator
create_power_domain –name PDVIN
update_power_domain –name PDVIN –primary_power_net HAVDD –primary_ground_net AVSS
create_power_domain –name PDVOUT -default –base_domains {PDVIN} –power_source
update_power_domain –name PDVOUT –pmos_bias_net VBB
create_power_domain –name PDREF –boundary_ports { C1 C2}
update_power_domain –name PDREF –primary_power_net AVDD –primary_ground_net AVSS
create_nominal_condition –name LDO_range –voltage 1.1 –pmos_bias_voltage 1.1:1.3
create_nominal_condition –name HVDD –voltage 2.45:2.55 –ground_voltage 0.0
create_nominal_condition –name REF –voltage 1.1 –ground_voltage 0.0
create_power_mode –name PM –default –domain_conditions \{ PDREF@REF PDVIN@HVDD PDVOUT@LDO_range \}
set_power_source_reference_pin AVDD –domain PDVOUT –voltage_range 1.1:1.1
end_macro_model regulator

simulation model may contain the output voltage behavior description but the CPF may not specify how the output voltage is controlled
• Special requirements for analog ports of a mixed signal IP
  ◆ Must be connected to the analog ports of other IPs
  ◆ Does not require (digital) isolation and level shifter inserted

• Extend to macro model by introducing the new command
  ◆ set_analog_ports

  • Semantics:
  ➢ The specified analog ports must be connected to ports that were declared as analog ports in either a macro or pad.
  ➢ Analog ports can be associated with a power domain of a macro.
  ➢ The implementation tools should not consider nets where the driver or receiver is an analog port for isolation or level shifter insertion.
  ➢ The verification tools should report an error if any isolation or level shifter logic is found on connections between ports specified with the set_analog_ports command.
Other macro model enhancements

- **Apply the same macro model to multiple cells**
  - `set_macro_model name [-cells list_of_cells]`

- **The -cells option has been added to the set_macro_model command to allow a single macro model to apply more than one cell.**
  - For example, different memory cells with the same architecture but different bits can use the same macro cell definition.

- **Use of this option also facilitates better error checking in cases where the user mistakenly applies the wrong macro model to an instance of a cell.**
Innovation Through Collaboration

Agenda

- Generic mode vs power mode
- Improved hierarchical flow
- Macro model enhancements
  - Voltage regulator, mixed-signal IP, IO ring
- **Simulation related enhancements**
  - Simulation semantics for pure behavioral constructs
- Modeling new low power IPs
  - Multi-bit level shifter and combo cells, clamp cells, ground shifters, etc
- Rules related enhancements
- Misc enhancements
  - Links to .lib pg_type attribute
  - find_design_objects
Motivation

- The corruption semantics for some pure behavioral constructs is not well defined
  - Initial statements: should it be re-evaluated at power up or not?
  - Real or integer type data: should it be corrupted or not?

- For netlist with low power logic inserted, simulation should not apply default simulation semantics
  - Needs to be done at the domain level and block level
  - Needs to be targeted for different low power semantics: isolation, state retention, power off corruption

- CPF 2.0 extension
  - A general command set_sim_control to provide the full controllability of above semantics.
set_sim_control [-targets target_list [-exclude target_list]]
   { -action power_up_replay [-controlling_domain domain ]
     |-action disable_corruption -type { real | wreal | integer | reg | module | instance}
     |-action {disable_isolation | disable_retention} }
   [-domains domain_list | -instances instance_list]
   [-modules module_list | -lib_cells lib_cell_list]

to control the initial statement replay at power up

to disable the default corruption semantics on selected RTL objects

to disable default inferencing of isolation and state retention in RTL and the corresponding simulation semantics
Innovation Through Collaboration

Agenda

- Generic mode vs power mode
- Improved hierarchical flow
- Macro model enhancements
  - Voltage regulator, mixed-signal IP, IO ring
- Simulation related enhancements
  - simulation semantics for pure behavioral constructs
- Modeling new low power IPs
  - Multi-bit level shifter and combo cells, clamp cells, ground shifters, etc
- Rules related enhancements
- Misc enhancements
  - Links to .lib pg_type attribute
  - find_design_objects

Qi Wang
Isolation Cell Enhancements

- Isolation cell can be placed in any domain
  - The cell’s main rail is not connected to any device of the cell

```
define_isolation_cell -cells cell_list [-library_set library_set] [-always_on_pins pin_list]
  [-power_switchable LEF_power_pin] [-ground_switchable LEF_ground_pin]
  [-power LEF_power_pin] [-ground LEF_ground_pin] [-valid_location { from | to | on | off | any}]  
  { -enable pin [-clamp {high|low}] | -no_enable {high|low|hold} } [-non_dedicated]
```

This type of cell has a dummy primary rail pin ‘VDD’ therefore it can be used in any location.
Clamp Cell Enhancements

- **Clamp Cell**
  - A simple NMOS/PMOS type of transistor to clamp a signal to 0/1
  -
  ```
  define_isolation_cell -cells cell_list [-library_set library_set] [-always_on_pins pin_list]
  [-power_switchable LEF_power_pin] [-ground_switchable LEF_ground_pin]
  [-power LEF_power_pin] [-ground LEF_ground_pin] [-valid_location { from | to | on | off | any}]
  { -enable pin [-clamp {high|low}] | -no_enable {high|low|hold} } [-non_dedicated]
  ```

  ```
  create_isolation_rule -name string [-isolation_condition expression | -no_condition]
  { -pins pin_list | -from power_domain_list | -to power_domain_list}...
  [-exclude pin_list] [-isolation_target {from|to}]
  [-isolation_output { low | high | hold | tristate | clamp_high | clamp_low}]
  [-secondary_domain power_domain]
  ```
Level Shifter Cell Enhancements

- Multi-bit isolation and level shifter cell
  - This is for macro isolation cells with multiple input/output paths
  
  ```
  define_isolation_cell  -cells cell_list [-library_set library_set] [-always_on_pins pin_list]
  [-power_switchable LEF_power_pin] [-ground_switchable LEF_ground_pin] [-non_dedicated]
  [-power LEF_power_pin] [-ground LEF_ground_pin] [-valid_location { from | to | on | off | any}]
  { -enable pin [-clamp {high|low}] | -no_enable {high|low|hold} | -pin_groups group_list }
  ```

Assume 'VDD' is the main rail pin and VDDC is the secondary power pin.
Level Shifter Cell Enhancements

- Multi-stage Level Shifters

  define_level_shifter_cell_cell .... [-multi_stage integer]

  update_level_shifter_rules -names rule_list { -location {from | to | parent}
  | -through power_domain_list | -within_hierarchy instance
  | -cells {cell_list | list_of_cell_lists} | -prefix string}...

![Diagram of level shifter cells](diagram)
Level Shifter Cell Enhancements

- By-pass Level Shifters

```plaintext
define_level_shifter_cell_cell .... [-bypass_enable expr]
create_level_shifter_rule –names string
    [ -bypass_condition expr] ...
```

![Diagram of level shifter cell](image)
Agenda

- Generic mode vs power mode
- Improved hierarchical flow
- Macro model enhancements
  - Voltage regulator, mixed-signal IP, IO ring
- Simulation related enhancements
  - simulation semantics for pure behavioral constructs
- Modeling new low power IPs
  - Multi-bit level shifter and combo cells, clamp cells, ground shifters, etc.
- Rules related enhancements
- Misc enhancements
  - Links to .lib pg_type attribute
  - find_design_objects

Prasad Subbarao
Isolation rule changes

- **Semantics of isolation cell location changed in update_isolation_rules**
  - CPF 1.1
    - -cell option took precedence to -location option. Cell location was determined by valid_location option in define_isolation cell
  - CPF 2.0
    - location option must match value specified in valid_location option in define_isolation_cell command

- **Semantics of -within_hierarchy option changed in update_isolation_rules**
  - CPF 1.1
    - Power domain of specified instance in -within_hierarch should match power domain of the default location or location specified by location option
  - CPF 2.0
    - Power domain of specified instance in -within_hierarch takes precedence.
    - Allows more flexibility
      - Isolation cell can be placed in separate domain other than ‘from’ or ‘to’ domain
Isolation rule enhancements

- **Support to place isolation cells in any domain**
  - Supports isolation cells with no power and ground pins that connect through abutment
  - `-valid_location` option in `define_isolation_cell` can be ‘any’
  - Secondary power and ground pins of isolation cells should connect to primary supplies of the secondary domain of the instance.

- **Support to place isolation cells in switchable domains**
  - `define_isolation_cell` should be specified with `–power_switchable` and `–ground_switchable` options.
    - CPF 1.1 only allows one of the two option can be specified for a cell

- **Support to place isolation cells in parent hierarchy**
  - `update_isolation_rules` command now accepts `–location` option with value ‘parent’
  - Isolation cell is placed in logic hierarchy of the parent instance of nets that were selected by the `–from`, `–to` and `–pins` options in `create_isolation_rule` command
Isolation rule enhancements

- **Support to force insertion of isolation cells**
  - `create_isolation_rule` command now accepts `-force` option
  - Specifies that isolation logic is inserted even in situations in which the rule would normally be ignored (such as on floating or undriven nets etc)
  - Exception: Isolation rule on a net connecting a top level port and a pad-pin of a pad instance will be ignored even with `-force` option.
  - Improves interoperability with UPF P1801-2009 standard

- **Support to specify name suffix for isolation cells**
  - Improves interoperability with UPF P1801-2009 standard

- **Support to enable multi-bit cells**
  - `-pin_groups` option in `define_isolation_cell` allows a list of input-output paths for multi-bit cells
  - Each group is a list of one cell input pin, one cell output pin and an optional enable pin
Level shifter rule changes

- Expanded support for input voltage tolerance
  - CPF 1.1
    - Input voltage tolerance can only be specified for input pins of a macro model
  - CPF 2.0
    - Input voltage tolerance can be specified for all inputs pins
    - Different tolerances can be specified for power and ground pins
    - `set_input_voltage_tolerance` command enhanced
    - Eg: `set_input_voltage_tolerance --power -0.2:0.4 --ground -0.3:0.1`
      - Specifies that driving power voltage can be 0.2V less or 0.4V more than the receiving voltage without requiring a power level shifter
      - Specifies that driving ground voltage can be 0.3V less or 0.1V more than the receiving voltage without requiring a ground level shifter
Level shifter rule changes

- **Support to specify lists for input and output voltages**
  - CPF 1.1
    - Either single voltage or voltage range for input and output supply voltages can be specified
    - Specifying ranges could imply unintended input and output combinations
  - CPF 2.0
    - Supports specification of ordered lists of voltages and voltage ranges
    - If ordered list of voltage or ranges is specified for input supply voltage, it must also be specified for output supply voltage and both lists should have same number of elements
    - Eg:
      - `define_level_shifter_cell -cells LSHL* -input_voltage_range { 0.8:0.9 1.0:1.1 } \ -output_voltage_range { 1.0:1.1 1.2:1.3 } --direction up`
      - This command indicates that the level shifter can shift from input range 0.8V to 0.9V to output range 1.0V to 1.1V or from input range 1.0V to 1.1V to output range 1.2V to 1.3V
Level shifter rule enhancements

- **Support to place level shifter cells in any domain**
  - Level shifter cells whose inputs and output pins are specified as non-followpins can be placed in any domain
  - `-valid_location` option in `define_level_shifter_cell` can be ‘any’

- **Support to place isolation cells in parent hierarchy**
  - `update_level_shifter_rules` command now accepts `-location` option with value ‘parent’
  - Isolation cell is placed in logic hierarchy of the parent instance of nets that were selected by the `-from`, `-to` and `-pins` options in `create_level_shifter_rule` command

- **Semantics of `--within_hierarchy` option changed in `update_level_shifter_rules`**
  - **CPF 1.1**
    - Power domain of specified instance in `--within_hierarchy` should match power domain of the default location or location specified by `-location` option
  - **CPF 2.0**
    - Power domain of specified instance in `--within_hierarchy` takes precedence.
    - Allows more flexibility
      - Level shifter cell can be placed in separate domain other than ‘from’ or ‘to’ domain
Level shifter rule enhancements

- **Support to force insertion of level shifter cells**
  - `create_level_shifter_rule` command now accepts `-force` option
  - Specifies that level shifter logic is inserted even in situations in which the rule would normally be ignored (such as on floating or undriven nets etc)
  - Improves interoperability with UPF P1801-2009 standard

- **Support to specify name suffix for level shifter cells**
  - Improves interoperability with UPF P1801-2009 standard
Enhancements to State Retention Rules

- **Enhancement for retention check and precondition**
  - `create_state_retention_rule` command now accepts `--retention_precondition` option
    - This allows specification of an additional condition that must be met for the retention operation to be successful
    - Condition can include clock, set and reset signals but cannot include signals that control the save and restore function of the cell
  - `define_state_retention_cell` command now accepts `--retention_check` option
    - This allows specification of an additional condition that must be met for the retention operation to be successful
    - Condition can be a boolean function of input pins
  - Improves interoperability with UPF P1801-2009 standard

- **Enhancement to preserve output while power is off**
  - `create_state_retention_rule` now accepts `--use_secondary_for_output` option
  - Output of retention logic will keep the previous value when primary domain is shutoff and secondary domain is on
  - Output pin is associated with secondary domain
  - Improves interoperability with UPF P1801-2009 standard

- **Enhancement to force the implementation of retention logic**
  - `create_state_retention_rule` now accepts `--required` option
  - Indicates that the registers selected by the rule must be implemented as retention flops or latches
Innovation Through Collaboration

Agenda

- Generic mode vs power mode
- Improved hierarchical flow
- Macro model enhancements
  - Voltage regulator, mixed-signal IP, IO ring
- Simulation related enhancements
  - simulation semantics for pure behavioral constructs
- Modeling new low power IPs
  - Multi-bit level shifter and combo cells, clamp cells, ground shifters, etc
- Rules related enhancements

Misc enhancements
- Links to .lib pg_type attribute
- find_design_objects

Steve Urish
Automatic connection of by pg_type
- create_global_connection -pg_type pg_type
- Specifies what net should connected to all pins defined with a pg_type in their Liberty model.

```
create_global_connection -net VDDB -power_domain PD1 -pg_type backup_power
```

find_design_objects
- Adds a powerful wildcard search to any CPF command option that accepts a design object

```
find_design_objects *SRAM* -pattern_type name -object inst
find_design_object {ADDER[16|32].*} -pattern_type module -hierarchical -regexp
```
**Boolean Expressions**

- CPF 1.1 supported only logical negation and bitwise and & or
- CPF supports bitwise and logical operations

| () | Parenthesis |
| ~ & | ^ | Bit-wise negation, AND, OR, XOR |
| ! && || | Logical negation, AND, OR |

```bash
create_isolation_rule \ 
-name ISO_1 \ 
-isolation_conditions (Iso && Mask)
```

Logical AND of two busses called Iso and Mask.

ex. 

(4'b1101 && 4'b0010) = 1

```bash
create_isolation_rule \ 
-name ISO_1 \ 
-isolation_conditions (Iso & Mask)
```

Bitwise AND of the same two busses yields a different answer.

ex. 

(4'b1101 & 4'b0010) = 0
- Allow inversion of equivalent control pins
  ```
  set_equivalent_control_pins -master pin
  -pins pin_expression_list { -domain domain / -rules rule_list }
  ```
- Useful if the low power control logic offer both polarity control signals to simplify the logic insertion
  - In the following example, the only available isolation cells are the regular AND gate and OR gate
  - I1/A and I1/B are opposite polarity
  - I1/A used for the OR gate and I1/B used for the AND gate to avoid always-on inverter insertion for I1/B

```latex
\begin{center}
\begin{tikzpicture}
  \node [draw, rectangle] (PCM) at (0,0) {PCM};
  \node [draw, rectangle] (PD1) at (0,-2) {PD1};
  \node [circle, fill=black] (A) at (1,0) {A};
  \node [circle, fill=black] (B) at (1,-2) {B};
  \node [draw, rectangle] at (1,-4) {I2};
  \node [draw, rectangle] at (2,-4) {X};
  \node [draw, rectangle] at (3,-4) {Y};

  \draw [->] (PCM) -- (A);
  \draw [->] (A) -- (B);
  \draw [->] (B) -- (PD1);
  \draw [->] (PD1) -- (X);
  \draw [->] (PD1) -- (Y);
\end{tikzpicture}
\end{center}
```

```latex
create_isolation_rule –name iso1
  -isolation_output high –from PD1 –pins I2/Y
  -isolation_conditions I1/A

create_isolation_rule –name iso2
  -isolation_output low –from PD1 –pins I2/X
  -isolation_conditions I1/A

set_equivalent_control_pins –master I1/A
  -pins {!I1/B} –rules iso*
```
Can assign a different library set to nominal_conditions and operating corners for power analysis

- create_operating_corner -power_library_set library_set_list
- update_nominal_condition -power_library_set library_set_list

Motivation:
- Typically, a library characterized as the worst corner for timing is not the worst corner for power
- The new feature allows the power analysis tool and timing analysis tool use different libraries for analysis

EX:

create_operating_corner -name opc1 ...
  -library_set timing.lib –power_library_set pow.lib

For all domains that are linked to this operating corner, use timing.lib for timing analysis and pow.lib for power analysis
Expanded Definition of Library Set for Operating Corners

- create_operating_corner -library_set library_set_list
- Each library_set represents an operating corner
- If I have 3 library set and each characterized for a different opc
  - In CPF 1.1, the user has to create 3 operating corner, one for each library set
  - In CPF 2.0, the user can create just one operating corner and list all library sets to the –library_set option of this corner