

Si2 Interoperability Guide for Power Format Standards™

IEEE Std 1801™-2009 and Si2 CPF V1.1™



V1.1

12 July 2010

Published by
Silicon Integration Initiative, Inc. (Si2™)
9111 Jollyville Road, Suite 250
Austin TX 78759

Copyright © 2010 by Si2, Inc.
All Rights Reserved.

ISBN: 1-882750-52-7

THE REQUESTED DOCUMENT DESCRIBING THE “Si2 Interoperability Guide for Power Format Standards” AND ALL MATERIALS AND INFORMATION THEREIN, ARE PROVIDED AS IS AND WITHOUT WARRANTY OF ANY KIND. NEITHER SI2, THE LOW POWER COALITION (LPC) MEMBER COMPANIES NOR ANY OTHER THIRD PARTY MAKES ANY REPRESENTATION OR WARRANTY WITH RESPECT TO THE SI2 COMMON POWER FORMAT, THE Si2 CPF SPECIFICATION OR ANY OF THE MATERIALS OR INFORMATION THEREIN. IN ADDITION, NO REPRESENTATIONS OR WARRANTIES OF ANY KIND, WHETHER WRITTEN, ORAL, IMPLIED OR STATUTORY, INCLUDING WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE

Si2 Interoperability Guide for Power Format Standards V1.1

Copyright © 2010 by Si2, Inc. All Rights Reserved

OR ARISING FROM COURSE OF DEALING OR USAGE IN TRADE ARE MADE OR SHALL APPLY.

NEITHER SI2 NOR ANY LPC MEMBER COMPANY SHALL BE LIABLE FOR ANY SPECIAL, INCIDENTAL, PUNITIVE, INDIRECT, OR CONSEQUENTIAL DAMAGES OF ANY NATURE ARISING FROM OR RELATING TO THE SI2 COMMON POWER FORMAT, THE SI2 CPF SPECIFICATION OR ANY MATERIALS OR INFORMATION THEREIN, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH LOSS OR DAMAGES.

Attention is called to the possibility that implementation of the constructs in this document may require use of subject matter covered by patent rights under which a license may be required. Si2 shall not be responsible for identifying patents or patent applications for which a license may be required to implement an Si2 specification or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention. By publication of this document, Si2 takes no position with respect to the existence or validity of any patent rights.

Si2 makes no representation as to the reasonableness or nondiscriminatory nature of the terms and conditions of the license agreements offered by any such holder(s). Further information may be obtained from Si2 upon request.

This document is subject to protection under Copyright Laws: Copyright © 2010 Si2. All Rights Reserved Worldwide. Requests for copyrighted material usage should be made to: Jo Anne Parks, Silicon Integration Initiative, 9111 Jollyville Rd., #250, Austin, TX 78759. Except as expressly set forth in the LPC Membership Agreement, no right or license is granted or implied under any copyrights in or to this document without written permission from Si2 .

No assurances are provided that this document will be compatible with subsequent versions or with any version that may be implemented in any products or technology.

Trademarks: Trademarks and service marks of Si2, Inc. contained in this document are attributed to Si2 with the appropriate symbol. All other trademarks are the property of their respective holders.

Disclaimer: Information in this publication is subject to change without notice and does not represent a commitment on the part of Si2 or any LPC member company. Except as may be explicitly set forth in such agreement, neither Si2 nor the LPC member companies make, and expressly disclaims, any representations or warranties as to the completeness, accuracy or usefulness of the information contained in this document. Neither Si2 nor the LPC member companies warrant that use of such information will not infringe any third party rights, neither does Si2 nor the LPC member companies assume any liability for damages or costs of any kind that may result from use of such information.

Restricted Rights: Use, duplication, or disclosure by the Government is subject to restrictions as set forth in FAR52.227-14 and DFAR252.227-7013 et seq. or its successor.

Contents

Si2 Interoperability Guide for Power Format Standards™	1
Contents	3
Document Status	4
Acknowledgments.....	5
Document Conventions.....	6
1 Introduction	7
2 Power Domain	9
3 Power State/Mode	18
4 State Retention	26
5 Isolation	33
6 Level Shifter	41
7 Power Switch	46
8 Miscellaneous	53
Appendix A Interoperability Tables	61
References	70

Document Status

Released version : si2_InteroperabilityGuideForPowerFormatStandardsV1.1.pdf

This section reflects the status of this document at the time of its publication. Other documents may supersede this document. Please contact Si2 for a complete list of current Si2 publications produced by the LPC.

Revision History

Version	Date	Comments
1.1	12-Jul-2010	Revised copyright section
1.0	01-Feb-2010	Initial version

Acknowledgments

This document has been produced by the LPC Format Working Group as part of the Low Power Coalition.

LPC Format WG

Gary Delp	CTO, Spirit Consortium
Nick English	Si2
Anmol Mathur	Calypto
Judith Richardson	AMD
Prasad Subbarao	LSI
Steve Urish	IBM
Qi Wang (Chair)	Cadence Design Systems

Editor

Susan Carver	Si2
--------------	-----

IEEE Std 1801™-2009

The IEEE 1801 command and argument names and descriptions in this document are used with permission from the IEEE, and are defined by the IEEE Std 1801™-2009, copyrighted by The Institute of Electrical and Electronics Engineers, Inc.

Document Conventions

To aid the reader's understanding, a consistent formatting style has been used throughout this document. The list below describes the syntax conventions used for both the CPF and 1801 constructs.

<code>Courier New font</code>	Text representing 1801 and CPF constructs and example code.
<code>literal</code>	Non-italic words indicate keywords that must be typed literally. These keywords represent command or argument names.
<i>argument_value</i>	Words in italics indicate user-defined arguments for which you must substitute a name or a value.
	Vertical bars (OR-bars) separate possible choices for a single argument.
[]	Brackets denote optional arguments.
{ }	Braces denote an argument list.
*	The preceding argument may be repeated.
<code>Blue text</code>	Command or argument is not recommended
<code>Orange text</code>	Command or argument is partially interoperable
<code>Red text</code>	Command or argument is non-interoperable

1 Introduction

This document describes the interoperable subset of commands between the two low power formats *IEEE Std 1801™-2009* (hereafter referred to as “1801”) and the [Si2 Common Power Format Specification™ V1.1](#) (hereafter referred to as “CPF”).

Most 1801 and CPF commands are at least partially interoperable, although some are not.

Interoperable commands and arguments do not always have a one-to-one correspondence in both formats. The mapping between 1801 and CPF may depend upon the arguments used or a particular combination of commands and/or arguments.

Most of the chapters in this document explore the interoperability of the listed 1801 commands in detail, organized by concept such as power domain, power state or mode, isolation, etc. There is one chapter per low power concept.

Appendix A contains quick-reference tables that present a general summary of the information presented in the chapters.

1.1 Objectives

The primary objective of this document is to provide guidance to facilitate interoperability between IEEE 1801-2009 and CPF 1.1 to support multi-vendor tool flows.

To achieve the goal, this document identifies a subset of commands and arguments between 1801 and CPF that can be used to describe a low power design with consistent semantics to drive verification and implementation.

The combination of this document and the [Si2 LPC Glossary 1.1](#) provides a complete and in-depth understanding of the key commonalities and differences between the two power formats.

1.2 Target Audience

This document is intended for system architects, RTL designers, verification engineers, logic and physical implementation engineers or anyone who is interested in creating low power design intent to drive a mixed format low power design flow.

1.3 Interoperability Considerations

To enhance interoperability, the following general rules should be considered when coding power intent in 1801:

- Use the supply set concept introduced in 1801, instead of individual power and ground nets, to specify power domains and power state tables.
- Do not use arguments that are considered non-interoperable in this document.
- Be careful when using arguments that are considered partially interoperable in this document. Check the document to understand the exact conditions or scenarios in which the argument is considered interoperable.
- Every 1801 supply set should be the primary supply set for at least one power domain to ensure an unambiguous mapping between an 1801 supply set and a CPF power domain.

1.4 Future Works

The current version of this document does not describe the subset of CPF commands that are interoperable with 1801. This will be covered in future releases of this document.

2 Power Domain

2.1 `create_supply_set set_name`

In 1801, a supply set is an abstract model of a set of supply nets. This is very close to the CPF power domain concept, which is also an abstract model of a set of physical supply nets.

In 1801, each power domain is associated with a unique primary supply. Similarly, each power domain in CPF must be associated with one unique set of supply nets. However, 1801 allows some supply sets to be defined without being associated with any power domains. To improve interoperability when coding with the 1801 specification, some virtual domains can be created to make sure each supply set has an associated power domain.

2.1.1 `-function {func_name [net_name]}`

The *net_name* can refer to either a virtual or real net in a design.

The *func_name* defines the functionality of the corresponding net. CPF does not have a similar way to define a supply net function. However, in CPF, the functions of different supply nets are explicitly referred to in the `update_power_domain` command. The mapping of *func_name* in 1801 with CPF supply net definitions in `update_power_domain` command is shown in Table 1:

Table 1 1801 - CPF 1.1 supply net functions

1801 <i>func_name</i>	CPF 1.1 argument
power	-primary_power_net
ground	-primary_ground_net
pwell	-nmos_bias_net
nwell	-pmos_bias_net
deep_pwell	n/a
deep_nwell	n/a

There are no corresponding `deep_pwell` and `deep_nwell` supply nets in CPF, and are considered non-interoperable.

2.1.2 `-reference_ground supply_net_name`

There are no corresponding constructs in CPF. This argument is used to adjust the voltages of each supply net defined in the supply set. It is considered a non-interoperable argument.

2.1.3 `-update`

This is not a power intent argument. The semantics can be fully interpreted when creating a corresponding CPF file for this command.

2.2 `create_power_domain domain_name`

This command corresponds to the CPF commands:

```
create_power_domain
update_power_domain
```

2.2.1 `-simulation_only`

There is no corresponding construct in CPF. The use of this argument may cause differences between models verified during simulation and the models used for implementation. It is considered a non-interoperable argument.

2.2.2 `-elements element_list/-exclude_elements exclude_list`

The `-elements` argument corresponds to the CPF `create_power_domain` command argument:

```
-instances
```

Even though CPF does not have a corresponding construct for `-exclude_elements`, the semantics of how to associate instances to power domains are the same in both 1801 and CPF. For example, the instances

excluded from one power domain must be assigned to another domain. By including those instances in another domain in CPF, the exclude semantics can be achieved automatically.

1801 cannot specify pins and ports in `-elements`. Instead, one of the commands `set_pin_related_supply` (8.14) or `set_port_attributes` (8.15) can be used to associate a pin or port to a power domain. When converting from 1801 to CPF, those constructs need to be described using the CPF `create_power_domain` command argument:

```
-boundary_ports
```

2.2.3 `-include_scope`

This argument corresponds to the CPF `create_power_domain` command argument:

```
-default
```

2.2.4 `-supply {supply_set_handle [supply_set_ref]}*`

This argument corresponds to the CPF `update_power_domain` command arguments:

```
-primary_power_net
-primary_ground_net
-pmos_bias_net
-nmos_bias_net
```

For non-default domain supply functions such as `default_retention` and `default_isolation`, there are no 1-1 corresponding constructs in the CPF power domain specification. However, the same information can be expressed in the 1801 retention or isolation strategy, which does have corresponding CPF constructs. For details, refer to the

```
-retention_supply_set of set_retention (4.1.6), and the
-isolation_supply_set of set_isolation (5.1.11).
```

For interoperability, it is recommended to specify a retention strategy or an isolation strategy to describe the corresponding supply set.

2.2.5 `-scope instance_name`

There is no corresponding argument in CPF.

There are no special semantics of this argument in 1801. It is considered a non-interoperable argument. However, even if it is used in an 1801 file, it shall be ignored when generating the corresponding CPF.

2.2.6 -define_func_type {*supply_function* *pg_type_list*}}*

This argument is an alternative way to describe the same intent as command `connect_supply_set`. See `connect_supply_set` (2.7) for interoperable details.

2.2.7 -update

This is not a power intent argument. The semantics for this command can be fully interpreted when creating a corresponding CPF file.

2.3 add_domain_elements *domain_name*

This is an alternative way to associate instances to a power domain. Even though there is no corresponding CPF command, the semantics can be fully described with the following CPF command and argument combination:

```
create_power_domain -instances
```

2.3.1 -elements *element_list*

This argument corresponds to the CPF argument

```
-instances
```

2.4 create_composite_domain *composite_domain_name*

There are two cases to be considered for this command.

In the first case, the elements of the composite domain are in the same design scope. In this case, the effect of this command is to merge all subdomains into a single composite domain. There is no corresponding CPF command for this

usage, but the semantics can be fully described in CPF by creating a single power domain that is a merge of all subdomains.

In the second case, the elements of the composite domain are in different design scopes. In this case, it corresponds to the CPF hierarchical construct

```
set_instance -domain_mapping
```

where one or more lower scope power domains are mapped into a higher scope power domain.

2.4.1 -subdomains *subdomain_list*

See the command description, above.

2.4.2 -supply {*supply_set_handle* [*supply_set_ref*]}*

This argument associates a supply set handle or supply set reference as the primary supply set to the composite domain. The same information shall be converted to the primary supply nets of the merged top-level power domain in CPF.

2.4.3 -update

This is not a power intent argument. The updated information is considered as part of the complete command.

2.5 *associate_supply_set supply_set_ref*

This is not a power intent command. This command simply provides a referencing between a handle and a supply set.

2.5.1 -handle *supply_set_handle*

In 1801, this links a supply set to a domain or a strategy. All the information can be fully converted into corresponding CPF constructs. For example, a retention strategy's supply set can be mapped into the secondary domain of a retention rule.

1801:

```
set_retention my_ret -domain PD ...
associate_supply_set some_supply_set -handle my_ret.supply
```

is equivalent to:

```
set_retention my_ret -domain PD \
    -retention_supply_set some_supply_set
```

CPF:

```
create_state_retention_rule my_ret -domain PD \
    -secondary_domain some_pd
```

where `some_pd` is the corresponding domain for the supply set `some_supply_set` in 1801.

2.6 `set_domain_supply_net domain_name`

The new and more flexible commands `associate_supply_set` (2.5) and `create_supply_set` (2.1) are recommended for interoperability.

Example:

1801

```
create_power_domain myPD ...
create_supply_set my_supply_set -function {power vdd} \
    -function {ground vss}
associate_supply_set my_supply_set -handle myPD.primary
```

CPF:

```
create_power_domain myPD ...
update_power_domain -name my PD -primary_power_net vdd \
    -primary_ground_net vss
```

2.7 `connect_supply_set supply_set_ref`

This command corresponds to the CPF command and arguments:

```
create_global_connection -net net_name
```

The CPF *net_name* must be derived from the supply nets associated with the *supply_set_ref*.

2.7.1 -connect {supply_function {pg_type_list}}*

This argument defines a connection of a specific supply net of the supply set to a list of pins with the specified *pg_type*, which is defined as a Liberty™ attribute in the technology library definition or an 1801 attribute using *set_port_attributes* (8.15).

There is no direct mapping in CPF for the *pg_type* specification. However, it is possible to analyze the library to select the pins that are marked with the corresponding *pg_type* attribute.

Example:

1801

```
create_power_domain myPD ...
create_supply_set my_supply_set -function { power vdd} \
                                -function {ground vss}
associate_supply_set my_supply_set -handle myPD.primary
connect_supply_set myPD.primary \
    -connect {power {primary_power}}
    -connect {ground {primary_ground}}
```

CPF

```
create_power_domain myPD ...
update_power_domain -name my PD -primary_power_net vdd \
                    -primary_ground_net vss
create_global_connection -domain myPD -net vdd \
                        -pins list_of_primary_power_pins
create_global_connection -domain myPD -net vss \
                        -pins list_of_primary_ground_pins
```

where *list_of_primary_power_pins*/*list_of_primary_ground_pins* is a list of all pins that has the *pg_type* attribute set as the *primary_power* or *primary_ground*.

Note: There is a potential issue when same-named cell pins in a library set are annotated with different *pg_type* attributes. In this case, the conversion may require expanding the pin names per instance to explicitly specify the connection.

2.7.2 **-elements *element_list***

This argument selects a list of instances or pins for the specified supply connection. The corresponding CPF command arguments are

```
-pins
-instances
```

2.7.3 **-exclude_elements *exclude_list***

There is no direct mapping of this argument in CPF. However, the intent can be fully translated by combining the information in `-elements` (2.7.2).

2.7.4 **-transitive <TRUE | FALSE>**

There is no direct mapping of this argument in CPF. However, the intent can be fully translated by combining the information in `-elements` (2.7.2) and `-exclude_elements` (2.7.3).

2.8 **connect_supply_net *net_name***

The new and more flexible command `connect_supply_set` (2.7) is recommended for interoperability.

This command corresponds to the CPF command:

```
create_global_connection -net net_name
```

2.8.1 **-ports *list***

This argument corresponds to the CPF argument

```
-pins
```

2.8.2 **[-pg_type {*pg_type_list element_list*}]***

See the description of the `-connect` argument in `connect_supply_set` (2.7.1).

2.8.3 **-vct** *vct_name*

There is no corresponding construct in CPF. It is considered a non-interoperable argument.

2.8.4 **-pins** *list*

This argument corresponds to the CPF argument

`-pins`

2.8.5 **-cells** *list*

The argument selects a list of cells whose corresponding instance pins are targeted for the specified connection.

Although there are no corresponding constructs in CPF, the semantics can be translated into the CPF arguments:

`-instances`
`-pins`

2.8.6 **-domain** *domain_name*

This argument corresponds to the CPF argument

`-domain`

2.8.7 **-rail_connection** *rail_type*

The argument is intended to link the Liberty `rail_connection` attribute with 1801. However, the attribute was phased out in Liberty, and as a result its usage is not recommended. It is considered a non-interoperable argument.

3 Power State/Mode

3.1 `add_port_state port_name`

This command associates a voltage or off state with a supply port. The new and more flexible command `add_power_state` (3.4) is recommended for interoperability.

This command corresponds to the CPF command

```
create_nominal_condition
```

However, multiple `add_port_state` commands are required to map into a single nominal condition, since nominal conditions in CPF apply to all defined supply nets in a power domain (power, ground, nwell, pwell).

3.1.1 `-state {name <nom | min max | min nom max | off>}`

This argument defines the voltage or off condition of this named state, and corresponds to these CPF `create_nominal_condition` arguments:

```
-voltage float
-ground_voltage float
-pmos_bias_voltage float
-nmos_bias_voltage float
-state {on | off | standby}
```

Note: the CPF `create_nominal_condition` command only supports a single voltage, whereas `add_port_state` supports a range.

3.2 `create_pst table_name`

This command creates a power state table and defines the header row. The new and more flexible command `add_power_state` (3.4) is recommended for interoperability.

This command is used to define the on/off combinations of the supply ports and corresponds to the CPF command

```
create_power_mode
```

3.2.1 **-supplies *supply_list***

This argument defines the order-dependent list of supply ports that will correspond to port states in the power supply table populated by subsequent calls to `add_pst_state`. In CPF, the `create_power_mode` command associates the nominal conditions with power domains directly using the '@' symbol in the `-domain_conditions` argument.

3.3 **add_pst_state *state_name***

This command creates a new row in the power state table. The new and more flexible command `add_power_state` (3.4) is recommended for interoperability.

3.3.1 **-pst *table_name***

This argument refers to a previously created PST template.

3.3.2 **-state *supply_states***

This argument corresponds to the CPF `create_power_mode` command argument

```
-domain_conditions
```

The following example illustrates the mapping of the commands in 1801 to CPF to describe power states or modes:

1801:

```
add_port_state VDD -state { ON 1.0 } -state { OFF off }
add_port_state VBAK -state { ON 1.0 } -state { OFF off }

create_pst MAIN_PST -supplies { VDD VBAK }
add_pst_state ALL_ON -pst MAIN_PST -states { ON ON }
add_pst_state SLEEP -pst MAIN_PST -states { OFF ON }
```

CPF:

```
create_nominal_condition -name ON -voltage 1.0 -state on
create_nominal_condition -name OFF -voltage 0.0 -state off

create_power_mode -name ALL_ON \
    -domain_conditions { PD_main@ON PD_keep_alive@ON } -default
create_power_mode -name SLEEP \
    -domain_conditions { PD_main@OFF PD_keep_alive@ON }
```

3.4 add_power_state *object_name*

This command corresponds to the CPF commands:

```
create_nominal_condition
create_power_mode
assert_illegal_domain_configurations
create_power_domain
```

and the *power mode control group*¹ concept for hierarchical power modes in CPF.

3.4.1 -state *state_name*

This argument corresponds to the CPF `create_power_mode` command argument:

```
-name
```

3.4.2 [-supply_expr {*boolean_function*}]

This argument can be used to define the voltage value of the supply nets, like the CPF `create_nominal_condition` command arguments:

```
-voltage
-ground_voltage
```

¹ *Si2 Common Power Format Specification Version 1.1*, p. 63

```
-pmos_bias_voltage
-nmos_bias_voltage
```

3.4.3 [-logic_expr {boolean_function}]

This argument corresponds to the CPF `create_power_domain` command arguments:

```
-active_state_conditions
-shutoff_condition
```

The `-active_state_conditions` argument describes the logical expression that cause the power domain to go into a particular on or standby nominal condition.

Likewise, the `-shutoff_condition` argument describes when the power domain turns off. However, CPF does not have an equivalent semantic for the interval expression. As a result, only logic expressions that do not use the `interval` function are considered interoperable.

3.4.4 [-simstate simstate]

Although CPF does not model simulation semantics directly like 1801, the CPF `create_nominal_condition` command argument

```
-state
```

can be used to infer the behavior of a power mode in simulation.

Table 2 shows the translation of nominal conditions in CPF to 1801 power domain simstates:

Table 2 CPF Nominal Conditions to 1801 Simstates

CPF Nominal Condition State	1801 Power Domain Simstate
off	CORRUPT
standby	CORRUPT_ON_ACTIVITY
on	NORMAL

There are no corresponding nominal condition states for **CORRUPT_STATE_ON_ACTIVITY** and **CORRUPT_STATE_ON_CHANGE**, which are considered non-interoperable.

3.4.5 [-legal | -illegal]

Legal power states in 1801 correspond to power modes in CPF, which by definition are the legal power supply configurations for the design. Illegal 1801 power states can be described by the CPF command

```
assert_illegal_domain_configurations
```

3.4.6 [-update]

This is not a power intent argument. The semantics for this command can be fully interpreted when creating a corresponding CPF file.

3.4.7 Example

CPF

```
create_power_domain -name PD_main -default -shutoff_condition shutoff_VDD \
    -external_controlled_shutoff
create_power_domain -name PD_keep_alive -instances { VI_keep_alive }

update_power_domain -name PD_main \
    -primary_power_net VDD -primary_ground_net GND
update_power_domain -name PD_keep_alive_main \
    -primary_power_net VBAK -primary_ground_net GND

create_nominal_condition -name ON_STATE -voltage 1.0 -state on
create_nominal_condition -name STANDBY_STATE -voltage 0.7 -state standby
create_nominal_condition -name OFF_STATE -voltage 0.0 -state off

create_power_mode -name ALL_ON \
    -conditions {PD_main@ON_STATE PD_keep_alive@ON_STATE} -default
create_power_mode -name KEEP_ALIVE \
    -conditions {PD_main@OFF_STATE PD_keep_alive@ON_STATE}
create_power_mode -name SLEEP \
    -conditions {PD_main@OFF_STATE PD_keep_alive@STANDBY_STATE}
```

1801 using power states

```
create_power_domain PD_main -include_scope
create_power_domain PD_keep_alive -elements { VI_keep_alive }
```

```

# two pairs of supply sets - VDD/GND and VBAK/GND
create_supply_set MAIN_SS      -function {power VDD}  -function {ground GND}
create_supply_set KEEP_ALIVE_SS -function {power VBAK} -function {ground GND}

# associate both supply sets with PD_main. This allows us to attach the top-level
# power states to PD_main.
create_power_domain PD_main      -update -supply {primary MAIN_SS} \
                                -supply {keep_alive KEEP_ALIVE_SS}
create_power_domain PD_keep_alive -update -supply {primary KEEP_ALIVE_SS}

add_power_state MAIN_SS -legal \
    -state {ON_MODE -simstate NORMAL
    -supply_expr {{VDD == `{FULL_ON, 1.0}} && {GND == `{FULL_ON, 0.0}}}}
    } \
    -state {OFF_MODE -simstate CORRUPT
    -supply_expr {{VDD == `{OFF}} || {GND == `{OFF}}}
    -logic_expr {shutoff_VDD}
    }

add_power_state KEEP_ALIVE_SS -legal \
    -state {ON_MODE -simstate NORMAL
    -supply_expr {{KEEP_ALIVE == `{FULL_ON, 1.0}} && {GND == `{FULL_ON, 0.0}}}}
    } \
    -state {STANDBY_MODE -simstate CORRUPT_ON_ACTIVITY
    -supply_expr {{KEEP_ALIVE == `{STANDBY, 0.7}} && {GND == `{FULL_ON, 0.0}}}}
    } \
    -state {OFF_MODE -simstate CORRUPT
    -supply_expr {{KEEP_ALIVE == `{OFF}} || {GND == `{OFF}}}
    }

add_power_state PD_main -legal \
    -state {ALL_ON -logic_expr {MAIN_SS == ON_MODE && KEEP_ALIVE_SS == ON_MODE}} \
    -state {KEEP_ALIVE -logic_expr {MAIN_SS == OFF_MODE && KEEP_ALIVE_SS == ON_MODE}} \
    -state {SLEEP -logic_expr {MAIN_SS == OFF_MODE && KEEP_ALIVE_SS == STANDBY_MODE}}

```

1801 using power states and a virtual top-level power domain

```

create_power_domain PD_top      -elements {} -simulation_only
create_power_domain PD_main     -include_scope
create_power_domain PD_keep_alive -elements {VI_keep_alive}

# two pairs of supply sets - VDD/GND and VBAK/GND
create_supply_set MAIN_SS      -function {power VDD}  -function {ground GND}
create_supply_set KEEP_ALIVE_SS -function {power VBAK} -function {ground GND}

# create a virtual top-level power domain that the top-level power states attach to
create_power_domain PD_top -update -supply {primary MAIN_SS} \
                                -supply {keep_alive KEEP_ALIVE_SS}
create_power_domain PD_main -update -supply {primary MAIN_SS}
create_power_domain PD_keep_alive -update -supply {primary KEEP_ALIVE_SS}

```

```

add_power_state MAIN_SS -legal \
  -state {ON_MODE -simstate NORMAL
    -supply_expr {{VDD == `{FULL_ON, 1.0}} && {GND == `{FULL_ON, 0.0}}}}
  } \
  -state {OFF_MODE -simstate CORRUPT
    -supply_expr { {VDD == `{OFF}} || {GND == `{OFF}}
    -logic_expr {shutoff_VDD}
  }

add_power_state VBAK_SS -legal \
  -state {ON_MODE -simstate NORMAL
    -supply_expr {{VBAK == `{FULL_ON, 1.0}} && {GND == `{FULL_ON, 0.0}}}}
  } \
  -state {STANDBY_MODE -simstate CORRUPT_ON_ACTIVITY
    -supply_expr {{VBAK == `{STANDBY, 0.7}} && {GND == `{FULL_ON, 0.0}}}}
  } \
  -state {OFF_MODE -simstate CORRUPT
    -supply_expr {{VBAK == `{OFF}} || {GND == `{OFF}}
  }

add_power_state PD_top -legal \
  -state {ALL_ON -logic_expr {MAIN_SS == ON_MODE && VBAK_SS == ON_MODE}} \
  -state {KEEP_ALIVE -logic_expr {MAIN_SS == OFF_MODE && VBAK_SS == ON_MODE}} \
  -state {SLEEP -logic_expr {MAIN_SS == OFF_MODE && VBAK_SS == STANDBY_MODE}}

```

1801 using a power state table ²

```

create_power_domain PD_main -include_scope
create_power_domain PD_keep_alive -elements {VI_keep_alive}

# two pairs of supply sets - VDD/GND and VBAK/GND
create_supply_set MAIN_SS -function {power VDD} -function {ground GND}
create_supply_set KEEP_ALIVE_SS -function {power VBAK} -function {ground GND}

create_power_domain PD_main -update -supply { primary MAIN_SS }
create_power_domain PD_keep_alive -update -supply { primary KEEP_ALIVE_SS }

### using psts ###
add_port_state VDD -state {ON_STATE 1.0} -state {OFF_STATE off}
add_port_state VBAK -state {ON_STATE 1.0} -state {OFF_STATE off} -state \
  {STANDBY_STATE 0.7}

create_pst MAIN_POWER_STATE_TABLE -supplies { VDD VBAK }
add_pst_state ALL_ON -pst MAIN_POWER_STATE_TABLE -states { ON_STATE ON_STATE }

```

² Using 1801 power states is recommended instead of power state tables for better interoperability. This example is provided in case compatibility with UPF 1.0 is required.


```
add_pst_state KEEP_ALIVE -pst MAIN_POWER_STATE_TABLE -states { OFF_STATE ON_STATE }
add_pst_state SLEEP      -pst MAIN_POWER_STATE_TABLE -states \
    { OFF_STATE  STANDBY_STATE }
```

3.5 **describe_state_transition** ***transition_name***

This command is similar to the `create_mode_transition` command in CPF. However, because CPF mode transition applies to power modes, and 1801 state transition applies to power domains, it is considered a non-interoperable argument.

4 State Retention

4.1 `set_retention retention_name`

This command corresponds to the CPF commands:

```
create_state_retention_rule  
update_state_retention_rules
```

4.1.1 `-domain domain_name`

This argument corresponds to the CPF `create_state_retention_rule` command argument:

```
-domain
```

4.1.2 `[-elements element_list]`

This argument corresponds to the CPF `create_state_retention_rule` command argument:

```
-instances
```

4.1.3 `[-exclude_elements exclude_list]`

This argument corresponds to the CPF `create_state_retention_rule` command argument:

```
-exclude
```

4.1.4 `[-retention_power_net net_name]`

The more accurate and robust argument `-retention_supply_set` (4.1.6) argument should be used for interoperability.

4.1.5 [-retention_ground_net net_name]

The more accurate and robust argument `-retention_supply_set` (4.1.6) should be used for interoperability.

4.1.6 [-retention_supply_set ret_supply_set]

This argument corresponds to the CPF `create_state_retention_rule` command argument:

`-secondary_domain`

where the set of supply nets associated with the secondary domain correspond to *ret_supply_set*.

4.1.7 [-no_retention]

This argument has no corresponding CPF constructs. When converting a retention strategy with this argument into CPF, the selected instances shall be excluded from any CPF `create_state_retention_rule` specification.

4.1.8 [-save_signal {{logic_net <high | low | posedge | negedge>}}]

This argument maps to the CPF `create_state_retention_rule` command arguments shown in Table 3.

Table 3 1801 -save_signal Mapping

1801 -save_signal setting	Corresponding CPF argument
posedge negedge	-save_edge
high low	-save_level

4.1.9 `[-restore_signal {{logic_net
<high | low | posedge | negedge>}}]`

This argument maps to the CPF `create_state_retention_rule` command arguments shown in Table 4.

Table 4 1801 -restore_signal Mapping

1801 -restore_signal setting	Corresponding CPF argument
posedge negedge	-restore_edge
high low	-restore_level

4.1.10 `[-save_condition { { boolean_function } }]`

This argument corresponds to the CPF `create_state_retention_rule` command argument:

`-save_precondition`

4.1.11 `[-restore_condition { { boolean_function } }]`

This argument corresponds to the CPF `create_state_retention_rule` command argument:

`-restore_precondition`

4.1.12 `[-retention_condition { { boolean_function } }]`

There is no corresponding construct in CPF. It is considered a non-interoperable argument.

4.1.13 `[-use_retention_as_primary]`

By default, the output driver retention logic is powered by the primary supply in both 1801 and CPF. This argument indicates the output driver is powered by the retention supply. There is no corresponding construct in CPF. It is considered a non-interoperable argument.

4.1.14 [**-parameters** {< <RET_SUP_COR | NO_RET_SUP_COR>
| <SAV_RES_COR | NO_SAV_RES_COR> > *}]

CPF simulation semantics match the default 1801 parameter settings of RET_SUP_COR and SAVE_RES_COR. The NO_RET_SUP_COR and NO_SAV_RES_COR settings are considered non-interoperable.

4.1.15 [**-instance** {{instance_name [signal_name]}*}]

This argument is used for power aware simulation of existing state retention logic in HDL. There are no corresponding CPF constructs. To facilitate interoperability, the cell must be defined with either the

- needed retention cell/pin attributes in Liberty, or
- define_state_retention_cell command in CPF.

4.1.16 [**-transitive** <TRUE | FALSE>]

There is no direct mapping of this argument in CPF. However, the intent can be fully translated by combining the information in `-elements` (4.1.2) and `-exclude_elements` (4.1.3).

4.1.17 [**-update**]

This is not a power intent argument. The semantics for this command can be fully interpreted when creating a corresponding CPF file.

4.2 **set_retention_control** *retention_name*

The more flexible command `set_retention` (4.1) should be used for interoperability.

4.3 **set_retention_elements** *retention_list_name*

This command corresponds to the CPF command:

```
create_state_retention_rule
```

4.3.1 {-applies_to <required | not_optional | not_required | optional>}]

This argument filters out any instances that do not have a UPF_retention attribute value (from the 1801 set_design_attributes (8.10) command) consistent with the selected filter choice. The selected instances are included in the specification of one of the CPF arguments

```
-instances  
-exclude
```

4.3.2 [-elements *element_list*]

This argument corresponds to the CPF argument:

```
-instances
```

4.3.3 [-exclude_elements *exclude_list*]

This argument corresponds to the CPF argument:

```
-exclude
```

4.3.4 [-retention_purpose <required | optional>]

By default, all instances specified in a CPF retention rule are optional. The required argument is considered non-interoperable.

4.3.5 [-transitive <TRUE | FALSE>]

There is no direct mapping of this argument in CPF. However, the intent can be fully translated by combining the information in the CPF arguments:

`-elements`
`-exclude_elements`

4.4 `map_retention_cell retention_name_list`

This command corresponds to CPF command:

`update_state_retention_rules`

4.4.1 `-domain domain_name`

This argument combined with the retention strategy name corresponds to the retention rule in CPF. The corresponding CPF argument is

`-names`

4.4.2 `[-elements element_list]`

There is no corresponding CPF construct. It is considered a non-interoperable argument since CPF can only give a single implementation instruction to all elements of a strategy.

4.4.3 `[-exclude_elements exclude_list]`

There is no corresponding CPF construct. It is considered a non-interoperable argument since CPF can only give a single implementation instruction to all elements of a strategy.

4.4.4 `[-lib_cells lib_cell_list]`

This argument corresponds CPF argument:

`-cells`

4.4.5 `[-lib_cell_type lib_cell_type]`

This argument corresponds to the CPF argument:

`-cell_type argument`

4.4.6 [-lib_model_name name {-port port_name
net_ref}*

There is no corresponding CPF construct. It is considered a non-interoperable argument.

5 Isolation

5.1 **set_isolation** *isolation_name*

This command corresponds to the CPF commands:

```
create_isolation_rule
update_isolation_rules
```

5.1.1 **-domain** *ref_domain_name*

There is no direct mapping of this argument in CPF. However, the purpose of this argument is to specify the targeted ports for isolation insertion. The identified targets can be described in CPF using one or a combination of the `create_isolation_rule` command arguments:

```
-from
-to
-pins
```

5.1.2 **[-elements** *element_list***]**

This argument corresponds to the CPF `create_isolation_rule` command argument:

```
-pins
```

5.1.3 **[-source** *source_supply_ref* **| -sink** *sink_supply_ref* **| -source** *source_supply_ref* **-sink** *sink_supply_ref***]**

Since 1801 associates each power domain with a unique supply set,

```
-source corresponds to -from
-sink corresponds to -to
```

of the CPF command `create_isolation_rule`.

5.1.4 `-applies_to <inputs | outputs | both>]`

There is no direct mapping of this argument in CPF. However, the purpose of this argument is to specify the targeted ports for isolation insertion. The identified targets can be described in CPF using one or a combination of the `create_isolation_rule` command arguments:

```
-from
-to
-pins
```

5.1.5 `[-applies_to_clamp <0 | 1 | any | z | latch | value>]`

There is no direct mapping of this argument in CPF. However, the purpose of this argument is to specify the targeted ports that have the same 1801 port attributes set for isolation insertion. The identified targets can be described in CPF using the `create_isolation_rule` argument:

```
-pins
```

5.1.6 `[-applies_to_sink_off_clamp <0 | 1 | any | z | latch | value>]`

There is no direct mapping of this argument in CPF. However, the purpose of this argument is to specify the targeted ports that have the same 1801 port attributes set for isolation insertion. The identified targets can be described in CPF using the following combination of `create_isolation_rule` arguments:

```
-pins pin_list -isolation_target to
```

5.1.7 `[-applies_to_source_off_clamp <0 | 1 | any | z | latch | value>]`

There is no direct mapping of this argument in CPF. However, the purpose of this argument is to specify the targeted ports that have the same 1801 port attributes set for isolation insertion. The identified targets can be described in CPF using the following combination of `create_isolation_rule` arguments:

```
-pins pin_list -isolation_target from
```

5.1.8 [-isolation_power_net net_name]

This argument is not considered part of the interoperable subset of 1801. The more accurate and robust argument `-isolation_supply_set` (5.1.11) should be used for interoperability.

5.1.9 [-isolation_ground_net net_name]

This argument is not considered part of the interoperable subset of 1801. The more accurate and robust argument `-isolation_supply_set` (5.1.11) should be used for interoperability.

5.1.10 [-no_isolation]

CPF semantics are that isolation logic is inferred if and only if there is some isolation rule applicable to a domain crossing. There is no direct mapping of this argument in CPF.

If a target port or crossing has only an isolation strategy specified with `-no_isolation`, the corresponding CPF shall not include this port or crossing in any isolation rule.

If a target port or crossing is covered by both an isolation strategy specified with `-no_isolation` and an isolation strategy specified without `-no_isolation`, the corresponding CPF shall have those targets described in `-exclude` for the corresponding isolation rule.

5.1.11 [-isolation_supply_set supply_set_list]

Since 1801 associates each power domain with a unique supply set, this argument corresponds to the CPF `create_isolation_rule` command argument:

```
-secondary_domain
```

However, it is considered non-interoperable if the `supply_set_list` has more than one supply set specified. For mapping purposes, only the first supply set will be used to match a power domain that can be used as the secondary domain specification for the corresponding CPF isolation rule.

5.1.12 [-isolation_signal *signal_list* [-isolation_sense {<high | low>*}]]

The combination of these two arguments corresponds to the following CPF `create_isolation_rule` command argument, which can take a complex Boolean expression:

```
-isolation_condition
```

Typically, each isolation strategy has only one isolation signal specified. If an isolation strategy has `-isolation_supply_set` specified with more than one argument, the isolation signal will take a list of signals corresponding to each supply set specified. However, it is considered non-interoperable if the `-isolation_supply_set` argument specifies more than one supply set. For mapping purposes, only the first supply set will be used to match a power domain that can be used as the secondary domain specification for the corresponding CPF isolation rule.

5.1.13 [-name_prefix *string*]

This argument corresponds to the CPF `update_isolation_rules` command argument:

```
-prefix
```

5.1.14 [-name_suffix *string*]

There is no corresponding argument in CPF. It is considered a non-interoperable argument.

5.1.15 [-clamp_value {< 0 | 1 | *any* | z | latch | *value*>*}]

This argument corresponds to the CPF `create_isolation_rule` command argument:

```
-isolation_output
```

The mapping of the values between the two arguments is shown in

Table 5:

Table 5 1801 - CPF -clamp_value mapping

1801	CPF 1.1
0	low
1	high
Z	tristate
Latch	hold
Value	low or high (see Note, below)

There is no corresponding CPF value for the 1801 value 'any'. It is considered a non-interoperable argument value. However, for implementation purposes for tools that convert 1801 into CPF, the corresponding CPF output isolation value can be considered 'low'.

Note: The 1801 value 'value' can be converted into a bitwise 'high' or 'low' in CPF.

5.1.16 [-sink_off_clamp <0 | 1 | any | Z | latch | value> [*simstate_list*]]

This argument specifies a clamp value for the isolation strategy. This corresponds to the CPF `create_isolation_rule` command argument:

`-isolation_output`

See

Table 5 for the mapping of argument values and interoperable semantics. When converting to CPF, this argument also implies that the corresponding CPF command shall specify

`-isolation_target to`

The `simstate_list` argument has no corresponding CPF construct and is considered non-interoperable.

5.1.17 [-source_off_clamp <0 | 1 | any | z | latch | value> [simstate_list]]

This argument also specifies a clamp value for the isolation strategy. This corresponds to the CPF `create_isolation_rule` command argument:

`-isolation_output`

See

Table 5 for the mapping of argument values and interoperable semantics. When converting to CPF, this argument also implies that the corresponding CPF command shall specify ‘

`-isolation_target from`

The `simstate_list` argument has no corresponding CPF construct and will be considered non-interoperable.

5.1.18 [-location <automatic | self | other | fanout | fanin | faninout | parent | sibling>]

The corresponding CPF argument is one of either

`-location`
`-within_hierarchy`

in the `update_isolation_rules` command. Table 6 shows a mapping of the 1801 `-location` value to CPF 1.1 arguments:

Table 6 1801 -location to CPF 1.1 mapping

1801	CPF
Self	from, if domain in <code>-domain</code> contains the driving logic for the applied crossing
	to, if domain in <code>-domain</code> contains the receiving logic for the applied crossing
other	to, if domain in <code>-domain</code> contains the driving logic for the applied crossing
	from, if domain in <code>-domain</code> contains the receiving logic for the applied crossing

fanout	to
fanin	from
faninout	to, if the applicable ports are output ports
	to, if the applicable ports are input ports
parent	Use <code>-within_hierarchy</code> to specify the logic hierarchy which matches the 'parent' specification in 1801

Because there are no corresponding CPF location values for 'automatic' and 'sibling', they are considered non-interoperable.

5.1.19 [-force_isolation]

An 1801 isolation strategy with this argument will simply be considered as an isolation rule in CPF, since each CPF isolation rule always infers isolation logic.

5.1.20 [-instance {{instance_name port_name}*}]

This argument is used for power aware simulation of existing isolation logic in HDL. There are no corresponding CPF constructs. To facilitate interoperability, the cell must be defined with either the

- needed isolation cell and pin attributes in Liberty, or
- `define_isolation_cell` command in CPF.

5.1.21 [-diff_supply_only <TRUE | FALSE>]

There is no corresponding CPF construct. The default value in 1801 is 'TRUE', which is consistent with default CPF isolation rule semantics, where isolation rules only apply to a crossing where the driver and receiver belong to different power domains. The value 'FALSE' is considered non-interoperable.

5.2 set_isolation_control isolation_name

The more flexible command `set_isolation` (5.1) should be used for interoperability.

5.3 **map_isolation_cell isolation_name**

This command describes implementation instructions, and corresponds to the CPF command:

```
update_isolation_rules
```

5.3.1 **-domain domain_name**

This argument combined with the isolation strategy name corresponds to the isolation rule in CPF. The corresponding CPF argument is

```
-names
```

5.3.2 **[-elements element_list]**

There is no corresponding CPF construct. It is considered a non-interoperable argument since CPF can only give a single implementation instruction to all elements of a strategy.

5.3.3 **[-lib_cells lib_cell_list]**

This argument corresponds to the CPF `update_isolation_rules` command argument.

```
-cells
```

5.3.4 **[-lib_cell_type lib_cell_type]**

This argument is not supported in 1801.

5.3.5 **[-lib_model_name name {-port port_name net_ref}]*]**

There is no corresponding CPF construct. It is considered a non- interoperable argument.

6 Level Shifter

6.1 `set_level_shifter level_shifter_name`

This command corresponds to the CPF commands:

```
create_level_shifter_rule
update_level_shifter_rules
```

6.1.1 `-domain domain_name`

There is no direct mapping of this argument in CPF. However, the purpose of this argument is to specify the targeted ports for level shifter insertion. The identified targets can be described in CPF using one or a combination of the following `create_level_shifter_rule` command arguments:

```
-from
-to
-pins
```

6.1.2 `[-elements element_list]`

This argument corresponds to the CPF `create_level_shifter_rule` command argument:

```
-pins
```

6.1.3 `[-no_shift]`

CPF semantics are that level shifter logic is inferred if and only if there is some level shifter rule applicable to a domain crossing. There is no direct mapping of this argument in CPF. If a target port or crossing has only a level shifter strategy specified with `-no_shift`, the corresponding CPF shall not include this port or crossing in any level shifter rule. If a target port or crossing is covered by both of a level shifter strategy specified with `-no_shift` and a level shifter strategy specified without `-no_shift`, the corresponding CPF shall have those targets described in `-exclude` for the corresponding level shifter rule

6.1.4 [-threshold value | list]

There is no direct or indirect mapping of this argument in CPF.

6.1.5 [-force_shift]

An 1801 level shifter strategy with this argument will simply be considered a level shifter rule in CPF, since each CPF level shifter rule always infers level shifter logic.

6.1.6 [-source domain_name] [-sink domain_name]

These arguments correspond to the CPF `create_level_shifter_rule` command arguments as follows:

```
-source corresponds to -from
-sink corresponds to -to
```

6.1.7 [-applies_to <inputs | outputs | both>]

There is no direct mapping of this argument in CPF. However, the purpose of this argument is to specify the targeted ports for level shifter insertion. The identified targets can be described in CPF using one or a combination of the following `create_level_shifter_rule` command arguments:

```
-from
-to
-pins
```

6.1.8 [-rule <low_to_high | high_to_low | both>]

There is no direct mapping of this argument in CPF. However, the purpose of this argument is to filter out the targeted ports for level shifter insertion. The identified targets that require level shifter insertion can be described in CPF using one or a combination of the following `create_level_shifter_rule` command arguments:

```
-from
-to
-pins
```

6.1.9 [-location <self | parent | sibling | fanout | automatic>]

This argument corresponds to one of either of the following CPF update_level_shifter_rules command arguments:

-location
-within_hierarchy

Table 7 shows a mapping of 1801 -location value to CPF 1.1 arguments:

Table 7 1801 -location value mapping

1801	CPF
self	from, if the domain in -domain contains the driving logic for the applied crossing
	to, if the domain in -domain contains the receiving logic for the applied crossing
fanout	to
parent	Use -within_hierarchy to specify the logic hierarchy that matches the 'parent' specification in 1801

Because there are no corresponding CPF location values for 'sibling' and 'automatic', they are considered non-interoperable.

6.1.10 [-name_prefix string]

This argument corresponds to the CPF update_level_shifter_rules command argument:

-prefix

6.1.11 [-name_suffix string]

There is no corresponding construct in CPF. It is considered a non-interoperable argument.

6.1.12 [-input_supply_set *supply_set_name*]

There is no corresponding argument in CPF. This argument defines the supply set used to power the input portion of the level shifter. This information

- is built-in with the level shifter rule semantics in CPF
- can also be converted into the CPF command

```
create_global_connection
```

to specify explicitly the power and ground nets needed to be connected to the input side of the power and ground pins of the level shifter cells.

6.1.13 [-output_supply_set *supply_set_name*]

There is no corresponding argument in CPF. This argument defines the supply set used to power the output portion of the level shifter. This information

- is built-in with the level shifter rule semantics in CPF
- can also be converted into the CPF command

```
create_global_connection
```

to specify explicitly the power and ground nets needed to be connected to the output side of the power and ground pins of the level shifter cells.

6.1.14 [-internal_supply_set *supply_set_name*]

There is no corresponding argument in CPF. It is considered a non-interoperable argument.

6.1.15 [-instance {{*instance_name port_name*}}*}]

This argument is used for power aware simulation of existing level shifter logic in HDL. There are no corresponding CPF constructs. To facilitate interoperability, the cell must be defined with either the

- needed level shifter cell and pin attributes in Liberty, or
- `define_level_shifter_cell` command in CPF.

6.2 **map_level_shifter_cell** *level_shifter_strategy*

This command describes implementation instructions, and corresponds to the CPF command:

```
update_level_shifter_rules
```

6.2.1 **-domain** *domain_name*

This argument combined with the level shifter strategy name corresponds to the level shifter rule in CPF. The corresponding CPF argument is

```
-names
```

6.2.2 **-lib_cells** *list*

This argument corresponds to the CPF argument:

```
-cells
```

6.2.3 **[-elements** *element_list***]**

There are no corresponding CPF constructs. This is considered non-interoperable since CPF can only give a single implementation instruction to all elements of a strategy.

7 Power Switch

7.1 `create_power_switch switch_name`

This command corresponds to the CPF commands:

```
create_power_switch_rule
update_power_switch_rule
```

7.1.1 `-output_supply_port {port_name [supply_net_name]}`

This argument specifies the output supply net to be connected to the port of the switch. The corresponding CPF `create_power_switch_rule` argument is:

```
-domain domain_name
```

In CPF, the primary power or ground net of the domain *domain_name* in `create_power_switch_rule` corresponds to the supply net *supply_net_name*. The *port_name* corresponds to the power switch output supply pin name in the CPF `define_power_switch_cell` command.

7.1.2 `{-input_supply_port {port_name [supply_net_name]}}*`

This argument specifies the input supply net to be connected to the port of the switch. The corresponding CPF `create_power_switch_rule` argument is one of:

```
-external_power_net
-external_ground_net
```

In CPF, the external power or external ground net specified in `create_power_switch_rule` corresponds to the supply net *supply_net_name*. The *port_name* corresponds to the power switch input supply pin name in the CPF `define_power_switch_cell` command.

For interoperability, each command should specify a single `-input_supply_port`. If multiple nets are needed, multiple `create_power_switch` commands can be used.

7.1.3 `{-control_port {port_name [net_name]}}*`

The corresponding CPF constructs are the control signals connected to a power switch. In CPF there are two approaches.

By default, the switch enable control signal is derived from the `-shutoff_condition` of the domain specified by `-domain`.

To override the default, either or both of these `update_power_switch_rule` command arguments can be used:

```
-enable_condition_1
-enable_condition_2
```

For interoperability, the command should specify at most two `-control_port` arguments. If two `-control_port` arguments are specified, the first one maps to `-enable_condition_1` and the second one maps to `-enable_condition_2`.

To define a set of power switches connected in parallel for a switchable domain, where each switch takes a different power gating enable signal, it is recommended to use multiple `create_power_switch` commands, where each command models one power switch instance.

7.1.4 `-on_state {state_name input_supply_port {boolean_function}}*`

The information specified here corresponds to the inversion of the `-shutoff_condition` of the power domain controlled by this power switch in CPF.

7.1.5 `[-off_state {state_name {boolean_function}}]*`

The information specified here corresponds to the `-shutoff_condition` of the power domain controlled by this power switch in CPF.

7.1.6 `[-supply_set supply_set_name]`

In 1801, the specified supply set powers the power switch and the acknowledge logic of the switch. In CPF, this corresponds to the supply nets specified for the base domain of the domain that is controlled by this switch strategy. If there is only one `-input_supply_port` (7.1.2) argument, the input supply net must be a supply net in the supply set.

However, it is not clear what the semantics are when there are multiple `-input_supply_port` arguments specified for this switch strategy. In this case, the argument is considered non-interoperable.

7.1.7 `[-on_partial_state {state_name input_supply_port {boolean_function}}]*`

There is no corresponding CPF construct. However, the semantics of partial-on in CPF is interpreted as power off, i.e. corruption semantics are implied.

7.1.8 `[-ack_port {port_name net_name [{boolean_function}]]]*`

This corresponds to the CPF `update_power_switch_rule` command arguments:

```
-acknowledge_receiver_1
-acknowledge_receiver_2
```

Since it is recommended to have at most two `-control_port` (7.1.3) arguments for interoperability, the `-ack_port` can be specified only twice at most. The first specification corresponds to the CPF argument `-acknowledge_receiver_1` and the second corresponds to `-acknowledge_receiver_2`.

7.1.9 `[-ack_delay {port_name delay}]*`

This argument specifies a power up time for the given switch. There is no direct CPF construct. However, by analyzing a power switch chain, the information can be described using the CPF `update_power_domain` command arguments:

```
-transition_delay
-transition_cycles
```


For interoperability, there can be at most two `-ack_delay` arguments specified. When two are specified, the corresponding CPF transition delay is the sum of the delay values specified in the two `-ack_delay` arguments.

7.1.10 [-error_state {state_name {boolean_function}}]*

There is no corresponding construct in CPF. However, the information can be translated into assertions in HDL during simulation. It is considered a non-interoperable argument.

7.1.11 [-domain domain_name]

This argument specifies the scope to which the power switch strategy applies. It can be used to reference a power switch strategy or a power switch instance that is instantiated in the specified domain. There is no direct mapping of this construct in CPF. However, ignoring this information has no impact on verification and implementation.

7.2 set_power_switch switch_name

This command is similar to the `create_power_switch` (7.1) command in terms of power intent specification. Unlike `create_power_switch`, this command extends an HDL instance by adding the input supply port(s), output supply port(s), and states to the switch. The supply ports are connected to the specified nets.

This command corresponds to the CPF commands:

```
create_power_switch_rule
update_power_switch_rule
```

7.2.1 -output_supply_port {port_name [supply_net_name]}

This argument specifies the output supply net to be connected to the port of the switch.

The corresponding CPF `create_power_switch_rule` argument is:

```
-domain domain_name
```

In CPF, the primary power or ground net of the domain *domain_name* in `create_power_switch_rule` corresponds to the supply net *supply_net_name*. The *port_name* corresponds to the power switch output supply pin name in the CPF `define_power_switch_cell` command.

7.2.2 `{-input_supply_port {port_name [supply_net_name]}}*`

This argument specifies the input supply net to be connected to the port of the switch.

The corresponding CPF `create_power_switch_rule` argument is:

```
-external_power_net-external_ground_net
```

In CPF, the external power or external ground net specified in `create_power_switch_rule` corresponds to the supply net *supply_net_name*. The *port_name* corresponds to the power switch input supply pin name in the CPF `define_power_switch_cell` command.

For interoperability, each command should specify a single `-input_supply_port`. If multiple nets are needed, multiple `set_power_switch` commands can be used.

7.2.3 `{-control_port {port_name}}*`

The corresponding CPF constructs are the control signals connected to a power switch. In CPF there are two approaches.

By default, the switch enable control signal is derived from the `-shutoff_condition` of the domain specified by `-domain`.

To override the default, either or both of these `update_power_switch_rule` command arguments can be used:

```
-enable_condition_1
-enable_condition_2
```

For interoperability, there should be at most two `-control_port` arguments specified by the command. If two `-control_port` arguments are specified, the first one maps to `-enable_condition_1` and the second one maps to `-enable_condition_1`.

7.2.4 `{-on_state {state_name input_supply_port {boolean_function}}}`*

The information specified here corresponds to the inversion of the `-shutoff_condition` of the power domain controlled by this power switch in CPF.

7.2.5 `[-supply_set supply_set_name]`

In 1801, the specified supply set powers the power switch and the acknowledge logic of the switch. In CPF, this corresponds to the supply nets specified for the base domain of the domain that is controlled by this switch strategy. If there is only one `-input_supply_port` (7.2.2) argument, the input supply net must be a supply net in the supply set.

However, it is not clear what the semantics are when there are multiple `-input_supply_port` arguments specified for this switch strategy. In this case, the argument is considered non-interoperable.

7.2.6 `[-on_partial_state {state_name input_supply_port {boolean_function}}]`*

There is no corresponding CPF construct. However, the semantics of partial on in CPF are interpreted as power off, i.e. corruption semantics are implied.

7.2.7 `[-off_state {state_name {boolean_function}}]`*

The information specified here corresponds to the `-shutoff_condition` of the power domain controlled by this power switch in CPF.

7.2.8 `[-error_state {state_name {boolean_function}}]`*

There is no corresponding construct in CPF. However, the information can be translated into assertions in HDL during simulation. It is considered a non-interoperable argument.

7.3 `map_power_switch {switch_name}`

This command corresponds to the CPF commands:

```
create_power_switch_rule  
update_power_switch_rule
```

7.3.1 **-domain *domain_name***

Per 1801, this argument has no semantics and is ignored

7.3.2 **-lib_cells {*list*}**

This argument corresponds to the CPF `update_power_switch_rule` command argument:

```
-cells
```

7.3.3 **[-port_map {{mapped_model_port switch_port_or_supply_net_ref}*}]**

This command creates a mapping between a power switch cell pin to a port in the abstract power switch model created by either the `create_power_switch` (7.1) or the `set_power_switch` (7.2) command.

The same mapping is achieved in CPF by extracting the semantics of each cell pin from the

- `define_power_switch_cell` CPF command, or
- Liberty definition of the power switch cell.

For interoperability with CPF, the power switch cell must be properly defined by either of these methods.

8 Miscellaneous

8.1 **bind_checker**

There is no corresponding CPF command. It is considered part of the non-interoperable subset of 1801. To facilitate interoperability between CPF and 1801, this command should be avoided. The same information can be included in the HDL model to be used by simulation tools to perform verification tasks.

8.2 **connect_logic_net** *net_name* **-ports** *port_list*

The equivalent CPF command and argument usage is

```
create_global_connection -net net_name -pins port_list.
```

8.3 **create_logic_port**

In 1801, this command is used to create a logic port of an IP block for the purpose of low power control signals such as power switch enable or isolation enable. In CPF, the concept is referred to as a virtual port, which can be declared using the CPF command and argument combination

```
set_design -ports
```

If the `-direction` argument specifies either 'out' or 'inout', the command is considered non-interoperable.

8.4 **create_hdl2upf_vct**

There is no corresponding CPF command. It is considered part of the non-interoperable set of 1801.

However, for simulation tools that support CPF, the default Value Conversion Tables defined in 1801 (Annex C)³ should be used for converting HDL logic values into the state values defined in `create_nominal_condition`.⁴

8.5 `create_upf2hdl_vct`

There is no corresponding CPF command. It is considered part of the non-interoperable set of 1801.

However, for simulation tools that support CPF, the default Value Conversion Tables defined in 1801 (Annex C)³ should be used for converting HDL logic values into the state values defined in `create_nominal_condition`.⁴

8.6 `load_simstate_behavior`

This is a command to load in a file containing one or more `set_sim_state_behavior` commands.

8.7 `load_upf upf_file_name`

If the `-scope` argument is omitted, this command is equivalent to the CPF command

```
include file_name
```

8.7.1 `-scope instance_name`

When this argument is used, the command is equivalent to the following sequence of CPF commands:

```
set_instance instance_name
```

³ IEEE Std 1801-2009, p. 207.

⁴ Si2 Common Power Format Specification Version 1.1, p. 109

`include` *file_name*

8.7.2 **-version string**

This is not a power intent argument. It specifies the version of the files to be loaded.

8.8 **load_upf_protected**

This is not a power intent command by itself. It should be handled the same as `load_upf` (8.7) except for the handling of global and local variables, which should be handled according to the semantics of the `-hide_globals` and `-params` arguments of this command. A general Tcl proc that implements this functionality can be found in Annex D⁵ of the 1801 document.

8.9 **name_format**

8.9.1 **-isolation_prefix**

To specify a name prefix for isolation logic, the new and more flexible command and argument combination of `set_isolation -name_prefix` (5.1.13) is recommended for interoperability.

8.9.2 **-isolation_suffix**

There is no corresponding CPF construct. It is considered part of the non-interoperable subset of 1801.

⁵IEEE Std 1801-2009, p. 211.

8.9.3 `-level_shift_prefix`

To specify a name prefix for level shifter logic, the new and more flexible command and argument combination of `set_level_shifter -name_prefix` (6.1.10) is recommended for interoperability.

8.9.4 `-level_shift_suffix`

There is no corresponding CPF construct. It is considered part of the non-interoperable set of 1801.

8.9.5 `-implicit_supply_suffix`

There is no corresponding CPF construct. It is considered part of the non-interoperable set of 1801.

8.9.6 `-implicit_logic_prefix`

There is no corresponding CPF construct. It is considered part of the non-interoperable set of 1801.

8.9.7 `-implicit_logic_suffix`

There is no corresponding CPF construct. It is considered part of the non-interoperable set of 1801.

8.10 `set_design_attributes`

Most power intent related design attributes can be converted into corresponding 1801 commands, as illustrated in “Table 1-Attribute and command correspondence”, section 4.6 of the 1801 specification⁶. See the corresponding 1801 command descriptions for interoperability analysis.

⁶ *IEEE Std 1801-2009*, p. 20.

The purpose of design attribute `UPF_is_leaf_cell` is to prevent the search of design objects going below the cell level. Even though there is no corresponding specification in CPF, the semantics of this attribute can be fully interpreted and used when creating a corresponding CPF file

8.11 `set_design_top root`

This command corresponds to the CPF command:

`set_design`

8.12 `set_scope instance`

This command corresponds to the CPF command:

`set_instance`

8.13 `set_partial_on_translation`

By default, CPF treats `PARTIAL_ON` the same as `OFF`, which is the same default semantics as 1801. It is non-interoperable if the command specifies that `PARTIAL_ON` is to be treated as `FULL_ON`.

8.14 `set_pin_related_supply`

This command corresponds to the CPF command

`define_related_power_pin`

The argument mapping is shown in Table 8.

Table 8 1801-CPF Argument Mapping

1801	CPF
<code>set_pin_related_supply</code>	<code>define_related_power_pin</code>
<code>library_cell</code>	<code>-cells</code>

-pins	-data_pins
-related_power_pin	-power
-related_ground_pin	-ground

8.15 **set_port_attributes**

There is no direct correspondence of this command to any particular CPF command. Most of the port attributes are used by other 1801 commands for specific operations, such as `set_isolation` and `connect_supply_set`, according to the semantics of each argument described in the 1801 document.

Arguments requiring special attention are detailed below.

8.15.1 **-related_power_port / -related_ground_port**

These arguments can be converted into the 1801 command `set_pin_related_supply`.

8.15.2 **-driver_supply / -receiver_supply**

These arguments specify the expected driving and receiving supply set of a top-level module's input and output ports respectively. The corresponding command and argument combination in CPF is

```
create_power_domain -boundary_ports
```

8.15.3 **-repeater_supply**

This argument has no corresponding 1801 command and is not used by any other 1801 command. It is considered non-interoperable.

8.16 **set_sim_state_behavior**

There is no corresponding CPF command. It is considered part of the non-interoperable set of 1801.

8.17 upf_version

This command corresponds to the CPF command

`set_cpf_version`

8.18 use_interface_cell *interface_implementation_name*

This command provides functionality similar to that of the 1801 commands `map_isolation_cell` (5.3) and `map_level_shifter_cell` (6.2).

8.18.1 -strategy *list_of_isolation_level_shifter_strategies*

See `map_isolation_cell` (5.3) or `map_level_shifter_cell` (5.3).

8.18.2 -domain domain_name

See the `-domain` argument of `map_isolation_cell` (5.3.1) or `map_level_shifter_cell` (6.2.1).

8.18.3 -lib_cells lib_cell_lists

See the `-lib_cells` argument of `map_isolation_cell` (5.3.3) or `map_level_shifter_cell` (6.2.2).

8.18.4 [-map {{port net_ref}*}]

This argument specifies the net to be connected to the specified port, and can be mapped into the CPF command

`create_global_connection`

8.18.5 [-elements element_list]

There are no corresponding CPF constructs. This is considered a non-interoperable option since CPF can only give a single implementation instruction to all elements of a strategy.

8.18.6 [-exclude_elements exclude_list]

There are no corresponding CPF constructs. This is considered a non-interoperable option since CPF can only give a single implementation instruction to all elements of a strategy.

8.18.7 [-applies_to_clamp <0 | 1 | any | z | latch | value>]

There are no corresponding CPF constructs. This is considered a non-interoperable option since CPF can only give a single implementation instruction to all elements of a strategy.

8.18.8 [-update_any <0 | 1 | known | z | latch | value>]

This argument is used to update the 'any' value specified in the strategy. The information for the specified strategy can be updated accordingly. See -clamp_value (5.1.15) and -applies_to_clamp (5.1.5) in the 1801 set_isolation command.

8.18.9 [-force_function]

There is no corresponding CPF construct. This argument is considered non-interoperable.

8.18.10 [-inverter_supply_set list]

By default, CPF requires that the same supply nets of the driving domain of the enable signal shall be used for the inverters or buffers on the isolation enable nets. It is interoperable if the argument specifies only one value and the specified supply set is the supply set for the driving logic of the isolation enable.

Appendix A

Interoperability Tables

Font color is used to visually denote commands and arguments of limited interoperability, as shown in Table 9.

Table 9 Color Coding

Color	Meaning
Blue	command is not recommended because there is a similar, more interoperable command
Orange	command or argument is partially interoperable
Red	command or argument is non-interoperable

Table 10 lists 1801 commands that are not recommended alongside their more interoperable counterparts.

Table 10 1801 Command Selection for Improved Interoperability

Not Recommended	Recommended
set_domain_supply_net	associate_supply_set create_supply_set
connect_supply_net	connect_supply_set
add_port_state	add_power_state
create_pst	add_power_state
add_pst_state	add_power_state
set_isolation_control	set_isolation

The remaining tables summarize the mapping between 1801 and CPF commands and arguments.

In some cases, a direct mapping is not possible, yet the power intent can be inferred from usage. Because the tables are a general summary only, these types of details are not included in them.

If an 1801 command maps to more than one CPF command, then each 1801 argument specifies which command is appropriate for that argument (e.g. `create_power_domain`). Otherwise, the corresponding CPF command is specified only in the table entry for the 1801 command (e.g. `connect_supply_set`).

Please refer to the command and argument descriptions in the appropriate chapters for complete information regarding each table entry.

Table 11 Power Domain Interoperability

1801 Command	Arguments	CPF Command Mapping	CPF Argument Mapping
add_domain_elements		create_power_domain	
	-elements		-instances
associate_supply_set		fully converted into CPF constructs depending upon the supply_set_ref	
	-handle		
connect_supply_net		create_global_connection	
	-cells		-instance -pins
	-domain		-domain
	-pg_type		See connect_supply_set -connect
	-pins		-pins
	-ports		-pins
	-rail_connection		
	-vct		
connect_supply_set		create_global_connection	
	-connect		no direct mapping, but library can be analyzed to select pg_type pins
	-elements		-pins -instances
	-exclude_elements		translated by combining the information in -elements
	-transitive		translated by combining the information in -elements and -exclude_elements
create_composite_domain (same scope)		create_power_domain	must encompass all subdomains
create_composite_domain (multi-scope)		set_instance	
	-subdomains		-domain_mapping
	-supply		converted to the primary supply nets of the merged top-level power domain
	-update		info is part of complete command
create_power_domain		create_power_domain update_power_domain	
	-define_func_type		See 1801 connect_supply_set
	-elements	create_power_domain	-instances
	-exclude_elements	create_power_domain	Must explicitly assign excluded elements to another domain
	-include_scope	create_power_domain	-default
	-scope		
	-simulation_only		
	-supply	update_power_domain	-primary_power_net -primary_ground_net -pmos_bias_net -nmos_bias_net
	-update		interpreted during CPF conversion
create_supply_set		encapsulated by CPF power domain concept	
	-function func_name	update_power_domain	-primary_power_net -primary_ground_net -nmos_bias_net -pmos_bias_net
	-reference_ground		
	-update		interpreted during CPF conversion
set_domain_supply_net			

Table 12 Power States / Modes Interoperability

1801 Command	Arguments	CPF Command Mapping	CPF Argument Mapping
add_port_state		create_nominal_condition	
	-state		-voltage -ground_voltage -pmos_bias_voltage -nmos_bias_voltage -state
add_power_state		create_nominal_condition create_power_mode assert_illegal_domain_configurations create_power_domain power mode control groups	
	-legal -illegal	assert_illegal_domain_configurations	
	-logic_expr	create_power_domain	-active_state_conditions -shutoff_condition
	-simstate	create_nominal_condition	-state
	-state	create_power_mode	-name
	-supply_expr	create_nominal_condition	-voltage -ground_voltage -pmos_bias_voltage -nmos_bias_voltage
	-update		interpreted during CPF conversion
add_pst_state			
	-pst	refers to a previously created PST template	
	-state	create_power_mode	-domain_conditions
create_pst		create_power_mode	
	-supplies		-domain_conditions
describe_state_transition			

Table 13 State Retention Interoperability

1801 Command	Arguments	CPF Command mapping	CPF Argument Mapping
map_retention_cell		update_state_retention_rules	
	-domain		-name
	-elements		
	-exclude_elements		
	-lib_cell_type		-cell_type
	-lib_cells		-cells
	-lib_model_name		
set_retention		create_state_retention_rule update_state_retention_rules	
	-domain	create_state_retention_rule	-domain
	-elements	create_state_retention_rule	-instances
	-exclude_elements	create_state_retention_rule	-exclude
	-instance	define_state_retention_cell	
	-no_retention	converted by excluding from any retention rule specification	
	-parameters		CPF matches default settings
	-restore_condition	create_state_retention_rule	-restore_precondition
	-restore_signal	create_state_retention_rule	-restore_edge -restore_level
	-retention_condition		
	-retention_ground_net		
	-retention_power_net		
	-retention_supply_set	create_state_retention_rule	-secondary_domain
	-save_condition	create_state_retention_rule	-save_precondition
	-save_signal	create_state_retention_rule	-save_edge -save_level
	-transitive	create_state_retention_rule	intent can be derived from combination of information defined with -elements and -exclude_elements
	-update	interpreted when creating corresponding CPF	
	-use_retention_as_primary		
set_retention_control			
set_retention_elements		create_state_retention_rule	
	-applies_to		-instances -exclude
	-elements		-instances
	-exclude_elements		-exclude
	-retention_purpose		all instances specified in CPF are optional
	-transitive		intent can be derived from combination of information defined with -elements and -exclude_elements

Table 14 Isolation Interoperability

1801 Command	Arguments	CPF Command mapping	CPF Argument Mapping
<code>map_isolation_cell</code>		<code>update_isolation_rules</code>	
	<code>-domain</code>		<code>-name</code>
	<code>-elements</code>		
	<code>-lib_cells</code>		<code>-cells</code>
	<code>-lib_cell_type</code>		
	<code>-lib_model_name</code>		
<code>set_isolation</code>		<code>create_isolation_rule</code> <code>update_isolation_rules</code>	
	<code>-applies_to</code>	<code>create_isolation_rule</code>	<code>-to</code> <code>-from</code> <code>-pins</code>
	<code>-applies_to_clamp</code>	<code>create_isolation_rule</code>	<code>-pins</code>
	<code>-applies_to_sink_off_clamp</code>	<code>create_isolation_rule</code>	<code>-pins</code> <code>-isolation_target to</code>
	<code>-applies_to_source_off_clamp</code>	<code>create_isolation_rule</code>	<code>-pins</code> <code>-isolation_target from</code>
	<code>-clamp_value</code>	<code>create_isolation_rule</code>	<code>-isolation_output</code>
	<code>-diff_supply_only</code>	interoperable if value is TRUE	
	<code>-domain</code>	<code>create_isolation_rule</code>	<code>-from</code> <code>-to</code> <code>-pins</code>
	<code>-elements</code>	<code>create_isolation_rule</code>	<code>-pins</code>
	<code>-force_isolation</code>	<code>create_isolation_rule</code>	
	<code>-instance</code>	<code>define_isolation_cell</code>	
	<code>-isolation_ground_net</code>		
	<code>-isolation_power_net</code>		
	<code>-isolation_signal &&</code> <code>-isolation_sense</code>	<code>create_isolation_rule</code>	<code>-isolation_condition</code>
	<code>-isolation_supply_set</code>	<code>create_isolation_rule</code>	<code>-secondary_domain</code>
	<code>-location</code>	<code>update_isolation_rules</code>	<code>-location</code> <code>-within_hierarchy</code>
	<code>-name_prefix</code>	<code>update_isolation_rules</code>	<code>-prefix</code>
	<code>-name_suffix</code>		
	<code>-no_isolation</code>	<code>create_isolation_rule</code>	<code>-exclude</code>
	<code>-sink</code>	<code>create_isolation_rule</code>	<code>-to</code>
	<code>-sink_off_clamp</code>	<code>create_isolation_rule</code>	<code>-isolation_output</code> <code>-isolation_target to</code>
	<code>-source_off_clamp</code>	<code>create_isolation_rule</code>	<code>-isolation_output</code> <code>-isolation_target from</code>
	<code>-source</code>	<code>create_isolation_rule</code>	<code>-from</code>
<code>set_isolation_control</code>			

Table 15 Level Shifter Interoperability

1801 Command	Arguments	CPF Command Mapping	CPF Argument Mapping
<code>map_level_shifter_cell</code>		<code>update_level_shifter_rules</code>	
	<code>-domain</code>		<code>-name</code>
	<code>-elements</code>		
	<code>-lib_cells</code>		<code>-cells</code>
<code>set_level_shifter</code>		<code>create_level_shifter_rule</code> <code>update_level_shifter_rules</code>	
	<code>-applies_to</code>	<code>create_level_shifter_rule</code>	<code>-from</code> <code>-to</code> <code>-pins</code>
	<code>-domain</code>	<code>create_level_shifter_rule</code>	<code>-from</code> <code>-to</code> <code>-pins</code>
	<code>-elements</code>	<code>create_level_shifter_rule</code>	<code>-pins</code>
	<code>-force_shift</code>	<code>create_level_shifter_rule</code>	An 1801 level shifter strategy with this argument will convert to a level shifter rule in CPF
	<code>-input_supply_set</code>	<code>create_level_shifter_rule</code> <code>create_global_connection</code>	
	<code>-instance</code>	<code>define_level_shifter_cell</code>	
	<code>-internal_supply_set</code>		
	<code>-location</code>	<code>update_level_shifter_rules</code>	<code>-location</code> <code>-within_hierarchy</code>
	<code>-name_prefix</code>	<code>update_level_shifter_rules</code>	<code>-prefix</code>
	<code>-name_suffix</code>		
	<code>-no_shift</code>	<code>create_level_shifter_rule</code>	<code>-exclude</code>
	<code>-output_supply_set</code>	<code>create_level_shifter_rule</code> <code>create_global_connection</code>	
	<code>-rule</code>	<code>create_level_shifter_rule</code>	<code>-from</code> <code>-to</code> <code>-pins</code>
	<code>-sink</code>	<code>create_level_shifter_rule</code>	<code>-to</code>
	<code>-source</code>	<code>create_level_shifter_rule</code>	<code>-from</code>
	<code>-threshold</code>		

Table 16 Power Switch Interoperability

1801 Command	Arguments	CPF Command Mapping	CPF Argument Mapping
map_power_switch		create_power_switch_rule update_power_switch_rule	
	-domain		argument is ignored
	-lib_cells	update_power_switch_rule	-cells
	-port_map	can be defined in Liberty or in define_power_switch	
create_power_switch		create_power_switch_rule update_power_switch_rule	
	-ack_delay	update_power_switch_rule	-transition_delay -transition_cycles
	-ack_port	update_power_switch_rule	-acknowledge_receiver_1 -acknowledge_receiver_2
	-control_port	update_power_switch_rule	-enable_condition_1 -enable_condition_2 or derived from the -shutoff_condition of the specified domain
	-domain		
	-error_state		can be coded as HDL assertions
	-input_supply_port	create_power_switch_rule	-external_power_net -external_ground_net
	-off_state		the -shutoff_condition of the domain controlled by this switch
	-on_partial_state		interpreted as power off
	-on_state		an inversion of the -shutoff_condition of the domain controlled by this switch
	-output_supply_port	create_power_switch_rule	-domain
	-supply_set		may be the same as the supply nets specified for the base domain of the domain controlled by this switch strategy
set_power_switch		create_power_switch_rule update_power_switch_rule	
	-control_port	update_power_switch_rule	-enable_condition_1 -enable_condition_2 or derived from the -shutoff_condition of the specified domain
	-error_state		can be coded as HDL assertions
	-input_supply_port	create_power_switch_rule	-external_power_net -external_ground_net
	-off_state		the -shutoff_condition of the domain controlled by this switch
	-on_partial_state		interpreted as power off
	-on_state		an inversion of the -shutoff_condition of the domain controlled by this switch
	-output_supply_port	create_power_switch_rule	-domain
	-supply_set		may be the same as the supply nets specified for the base domain of the domain controlled by this switch strategy

Table 17 Miscellaneous Commands Interoperability

1801 Command	Arguments	CPF Command Mapping	CPF Argument Mapping
<code>bind_checker</code>			
<code>connect_logic_net</code>		<code>create_global_connection</code>	
<code>create_logic_port</code>		<code>set_design -ports</code>	
<code>create_hdl2upf_vct</code>			
<code>create_upf2hdl_vct</code>			
<code>load_simstate_behavior</code>			
<code>load_upf</code>		<code>include</code> (if <code>-scope</code> is not specified)	
	<code>-scope</code>	<code>set_instance</code> <code>include</code>	
	<code>-version</code>		
<code>load_upf_protected</code>		See <code>load_upf</code> ; Globals and local variables handled according to <code>-hide_globals</code> and <code>-params</code> semantics	
<code>name_format</code>			
	<code>-isolation_prefix</code>		
	<code>-isolation_suffix</code>		
	<code>-level_shift_prefix</code>		
	<code>-level_shift_suffix</code>		
	<code>-implicit_supply_suffix</code>		
	<code>-implicit_logic_previx</code>		
	<code>-implicit_logic_suffix</code>		
<code>set_design_attributes</code>		these attributes are first converted into other 1801 commands	
<code>set_design_top</code>		<code>set_design</code>	
<code>set_scope</code>		<code>set_instance</code>	
<code>set_partial_on_translation</code>			
<code>set_pin_related_supply</code>		<code>define_related_power_pins</code>	
	<code>library_cell</code>		<code>-cells</code>
	<code>-pins</code>		<code>-data_pins</code>
	<code>-related_power_pin</code>		<code>-power</code>
	<code>-related_ground_pin</code>		<code>-ground</code>
<code>set_port_attributes</code>			
	<code>-related_power_port</code>	convert to 1801 <code>set_pin_related_supply</code> command	
	<code>-related_ground_port</code>		
	<code>-driver_supply</code>	<code>create_power_domain</code>	<code>-boundary_ports</code>
	<code>-receiver_supply</code>		
	<code>-repeater_supply</code>		
	all other attributes	used as attributes by other 1801 commands and converted to CPF accordingly	
<code>set_sim_state_behavior</code>			
<code>upf_version</code>		<code>set_cpf_version</code>	
<code>use_interface_cell</code>		See 1801 <code>map_isolation_cell</code> or <code>map_level_shifter_cell</code>	
	<code>-applies_to_clamp</code>		
	<code>-domain</code>	See 1801 <code>map_isolation_cell</code> or <code>map_level_shifter_cell -domain</code>	
	<code>-elements</code>		
	<code>-exclude_elements</code>		
	<code>-force_function</code>		
	<code>-inverter_supply_set</code>	a single supply set supplies the driving logic of isolation enable	
	<code>-lib_cells</code>	See 1801 <code>map_isolation_cell</code> or <code>map_level_shifter_cell -lib_cells</code>	
	<code>-map</code>	<code>create_global_connection</code>	
	<code>-strategy</code>	See 1801 <code>map_isolation_cell</code> or <code>map_level_shifter_cell</code>	
	<code>-update_any</code>	See 1801 <code>set_isolation -clamp_values</code> and <code>-applies_to_clamp</code>	

References

[1] *IEEE Std 1801™-2009*, “IEEE Standard for Design and Verification of Low Power Integrated Circuits”, 19 March 2009.

Available at <http://www.ieee.org/web/standards/home/index.html>

[2] *Si2 Common Power Format Specification™ V1.1*, 19 September 2008.

Available at

http://www.si2.org/openeda.si2.org/project/showfiles.php?group_id=51

[3] *Si2 LPC Glossary 1.1*, 20 July 2009.

Available at <http://www.si2.org/?page=1049>