Design Constraints User Guide
SmartFusion2, IGLOO2, RTG4, and PolarFire

NOTE: PDF files are intended to be viewed on the printed page; links and cross-references in this PDF file may point to external files and generate an error when clicked. **View the online help included with software to enable all linked content.**

Microsemi

Power Matters.™

About Microsemi

Microsemi Corporation (Nasdaq: MSCC) offers a comprehensive portfolio of semiconductor and system solutions for aerospace & defense, communications, data center and industrial markets. Products include high-performance and radiation-hardened analog mixed-signal integrated circuits, FPGAs, SoCs and ASICs; power management products; timing and synchronization devices and precise time solutions, setting the world's standard for time; voice processing devices; RF solutions; discrete components; enterprise storage and communication solutions; security technologies and scalable anti-tamper products; Ethernet solutions; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Microsemi is headquartered in Aliso Viejo, California, and has approximately 4,800 employees globally. Learn more at **www.microsemi.com.**

5-02-00046-26/01.17

Table of Contents

Design Constraints

Design constraints are usually either requirements or properties in your design. You use constraints to ensure that your design meets its performance goals and pin assignment requirements.

The Libero SoC software supports both SDC timing and PDC physical constraints. In addition, it supports netlist optimization constraints. You can set constraints by either using Microsemi's interactive tools (I/O Editor, Chip Planner, and Constraint Editor) or by importing constraint files directly into your design session.

With Enhanced Constraints Flow, use the Constraint Manager to manage all your design constraints.

SDC Timing Constraints

Timing constraints represent the performance goals for your designs. Microsemi software uses timing constraints to guide the timing-driven optimization tools in order to meet these goals.

You can set timing constraints either globally or to a specific set of paths in your design.

You can apply timing constraints to:

- Specify the required minimum speed of a clock domain
- Set the input and output port timing information
- Define the maximum delay for a specific path
- Identify paths that are considered false and excluded from the analysis
- Identify paths that require more than one clock cycle to propagate the data
- Provide the external load at a specific port

To get the most effective results from the Designer software, you need to set the timing constraints close to your design goals. Sometimes slightly tightening the timing constraint helps the optimization process to meet the original specifications.

PDC Physical Constraints

Designer software enables you to specify the physical constraints to define the size, shape, utilization, and pin/pad placement of a design. You can specify these constraints based on the utilization, aspect ratio, and dimensions of the die. The pin/pad placement depends on the external physical environment of the design, such as the placement of the device on the board.

There are three types of physical constraints:

- I/O assignments
  - Set location, attributes, and technologies for I/O ports
  - Specify special assignments, such as VREF pins and I/O banks
- Location and region assignments
  - Set the location of Core, RAM, and FIFO macros
  - Create Regions for I/O and Core macros as well as modify those regions
- Clock assignments
  - Assign nets to clocks
  - Assign global clock constraints to global, quadrant, and local clock resources

Netlist Optimization Constraints

The software enables you to set some advanced design-specific netlist optimizing constraints.

You can apply netlist optimization constraints to:

- Delete or restore a buffer tree
- Manage the fan-outs of the nets

- Manage macro combinations (for example, IO-REG combining)
- Optimize a netlist by removing buffers and/or inverters, propagating constants, and so on

See Also

Constraint Support by Family

Constraint Entry Table

Constraint File Format by Family

Naming Conventions

Families Supported

Constraint Support by Family

Use the Constraint Family Support table to see which constraints you can use for your device family. Click the name of a constraint in the table for more information.

When we specify a family name, we refer to the device family and all its derivatives, unless otherwise specified.

Table 1 · Constraint Support by Family

	IGLOO	SmartFusion2, IGLOO2, RTG4, PolarFire	SmartFusion and Fusion	ProASIC3
Timing				
Create a clock	X	X	X	X
Create a generated clock	X	X	X	X
Remove clock uncertainty	X	X	X	X
Set clock latency	X	X	X	X
Set clock uncertainty	X	X	X	X
Set disable timing	X	X	X	X
Set false path	X	X	X	X
Set input delay	X	X	X	X
Set load on output port	X	X	X	X
Set maximum delay	X	X	X	X
Set minimum delay	X	X	X	X
Set multicycle path	X	X	X	X
Set output delay	X	X	X	X
Physical Placement				
-Clocks				
Assign Net to Global Clock	X		X	X
Assign Net to Local Clock	X		X	X
Assign Net to Quadrant Clock	X		X	X
-Regions				

	IGLOO	SmartFusion2, IGLOO2, RTG4, PolarFire	SmartFusion and Fusion	ProASIC3
Timing				
Assign Macro to Region	X	X	X	X
Assign Net to Region	X	X	X	X
Create Region	X	X	X	X
Delete Regions	X	X	X	X
Move Region	X	X	X	X
Unassign macro(s) driven by net	X	X	X	X
Unassign Macro from Region	X	X	X	X
-I/Os				
Assign I/O to pin	X	X	X	X
Assign I/O Macro to Location	X	X	X	X
Configure I/O Bank	X	X	X	X
Reset attributes on I/O to default settings	X	X	X	X
Reset I/O bank to default settings	X	X	X	X
Reserve pins	X	X	X	X
Unreserve pins	X	X	X	X
-Block				
Move Block	X	X	X	X
Set Port Block	X	X	X	X
Set Block Options	X	X	X	X
-Nets				
Assign Net to Global Clock	X		X	X

	IGLOO	SmartFusion2, IGLOO2, RTG4, PolarFire	SmartFusion and Fusion	ProASIC3
Timing				
Assign Net to Local Clock	X		X	X
Assign Net to Quadrant Clock	X		X	X
Assign Net to Region	X	X	X	X
Reset net's criticality to default level				
Set Net's Criticality				
Unassign macro(s) driven by net	X	X	X	X
Netlist Optimization				
Delete buffer tree	X		X	X
Demote Global Net to Regular Net	X		X	X
Promote regular net to global net	X		X	X
Restore buffer tree	X		X	X
Set preserve	X	X	X	X

See Also

Constraint Entry Table

Constraint File Format by Family

Constraint Entry

Use the Constraint Entry table to see which tools and file formats you can use to enter constraints for your device family.

Click the name of a constraint, a constraint entry tool, file format type, editor, or checkmark in the table for more information about that item.

Table 2 · Constraint Entry by Tool and File Format

Constraint	SDC	PDC	PIN	ChipPlanner	I/O Attribute Editor	PinEditor	SmartTime	Compile Options
Timing								
Create a clock	X						X	
Create a generated clock	X						X	
Remove clock uncertainty	X						X	
Set clock latency	X						X	
Set clock uncertainty	X						X	
Set disable timing	X						X	
Set false path	X						X	
Set input delay	X						X	
Set load on output port	X				X	X	X	
Set maximum delay	X						X	
Set minimum delay	X						X	
Set multicycle path	X						X	
Set output delay	X						X	
Physical Placement								
-Clocks								
Assign Net to Global Clock		X						
Assign Net to Local Clock		X		X				
Assign Net to Quadrant Clock		X		X				

-Regions							
Assign Macro to Region		X		X			
Assign Net to Region		X		X			
Create Region		X		X			
Delete Regions		X		X			
Move region		X		X			
Unassign macro(s) driven by net		X		X			
Unassign macro from region		X		X			
-I/Os							
Assign I/O to pin		X	X	X	X	X	
Assign I/O Macro to Location		X		X			
Configure I/O Bank		X		X		X	
Reset attributes on I/O to default settings		X		X	X		
Reset I/O bank to default settings		X		X	X		
Reserve pins		X			X	X	
Unreserve pins		X			X	X	
-Blocks							
Move Block		X					
Set port block		X		X			
Set Block Options		X					X
-Nets							
Assign Net to Global Clock		X					
Assign Net to Local Clock		X		X			
Assign Net to Quadrant Clock		X		X			
Assign Net to Region		X		X			
Reset net's criticality to default level		X					
Set Net's Criticality		X					
Unassign macro(s) driven by net		X		X			
Netlist Optimization							

Constraint	SDC	PDC	PIN	ChipPlanner	I/O Attribute Editor	PinEditor	SmartTime	Compile Options
Timing								
Delete buffer tree		X						X
Demote Global Net to Regular Net		X						X
Promote regular net to global net		X						X
Restore buffer tree		X						
Set preserve		X						

See Also

Constraint Support by Family

Constraint File Format by Family

Constraint File Format by Family

Use the File Format by Family table to see which file formats apply to each type of constraint and each device family.

When we specify a family name, we refer to the device family and all its derivatives, unless otherwise specified.

Table 3 · Constraint File Format by Family

Family	Timing	Physical Placement	Netlist Optimiization
	SDC	PDC	PDC
IGLOO	X	X	
IGLOO2	X	X	X
SmartFusion2	X	X	X
SmartFusion and Fusion	X	X	X
ProASIC3	X	X	
PolarFire	X	X	X

SDC – Synopsys Design Constraints
PDC – Physical Design Constraints

See Also

Constraint Support by Family
Constraint Entry Table

Basic Concepts

Naming Conventions

The names of ports, instances, and nets in an imported netlist are sometimes referred to as their original names. Port names appear exactly as they are defined in a netlist. For example, a port named A/B appears as A/B in ChipPlanner, PinEditor, and I/O Attribute Editor in MultiView Navigator. Instances and nets display the original names plus an escape character (\) before each backslash (/) and each slash (\) that is not a hierarchy separator. For example, the instance named A/\B is displayed as A\/\\B.

The following components use the Tcl-compliant original names:

- PDC reader/writer
- SDC reader/writer
- Compile report
- SDF/Netlist writer for back annotation
- MultiView Navigator tools: NetlistViewer, PinEditor, ChipPlanner, and I/O Attribute Editor
- SmartTime
- SmartPower

See Also

[PDC Naming Conventions](#)

Clock

Specifying clock constraints is the most effective way of constraining and verifying the timing behavior of a sequential design. You must use clock constraints to meet your performance goals and to quickly reach timing closure.

Best practice is to specify and constrain all clocks used in the design.

To create a clock constraint, you must provide the following clock information:

Clock source: Specifies the pin or port where the clock signal is defined.

Clock period or frequency: Defines the smallest amount of time after which the signal repeats itself.

Duty cycle: Defines the percentage of time during which the clock period is high.

First edge: Indicates whether the first edge of the clock is rising or falling.

Offset: Indicates the shift of the first edge with respect to instant zero common to all clocks in the design.

Example 1:

create_clock -period 10 -waveform {2 7}

This example creates a clock with 10ns period, 2ns offset, and 50% duty cycle using the SDC command.
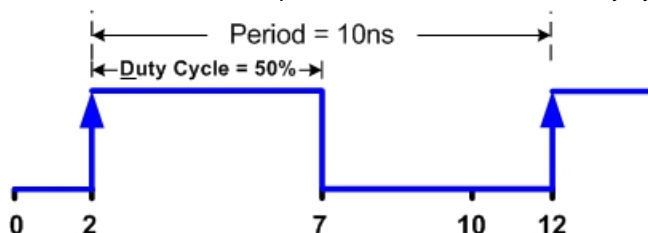
Example 2:

This example shows how to create a clock with 25MHz frequency, 4ns offset for its first rising edge, and 60% duty cycle using the SmartTime Constraints Editor. Using the Create New Clock Constraint dialog box is equivalent to using the SDC command: create_clock -period 40 -waveform {4, 28}.
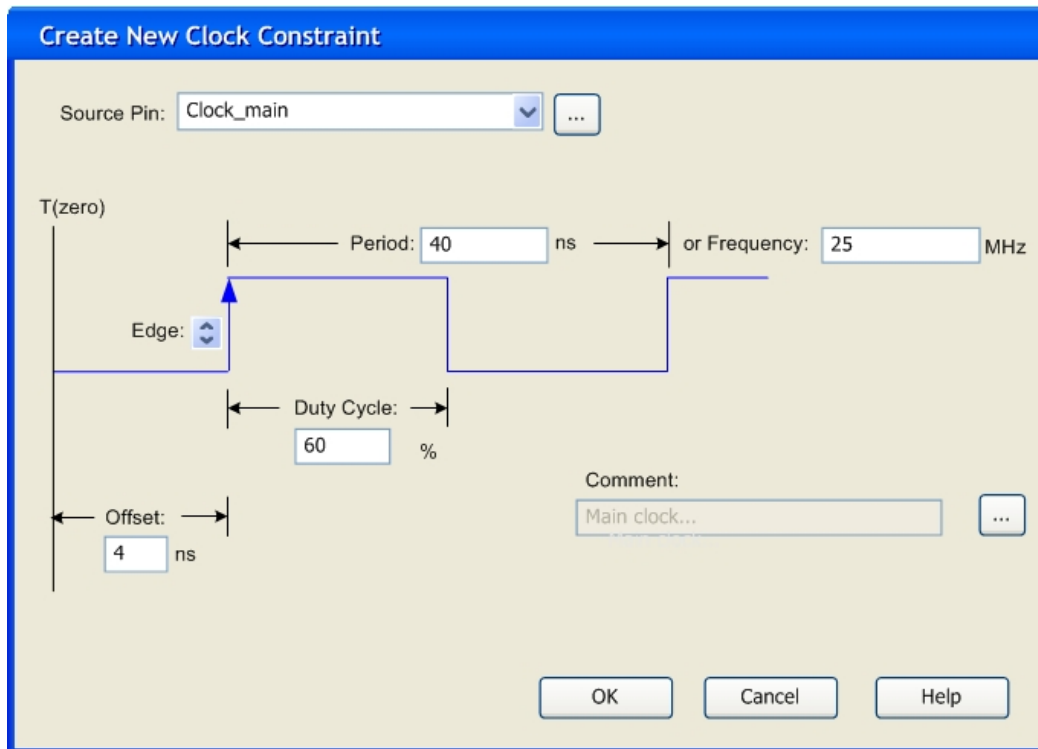
See Also

Constraint support by family

Constraint entry table

create_clock (SDC)

global_clocks (DCF)

Specifying Clock Constraints

Region

A region is a user-defined area on a chip into which you can constrain the physical placement of one or more macros. You can also constrain macros containing multiple tiles for cores, RAMs, and I/Os. The floorplanning process usually requires you to create several regions and assign logic to them. Logic can include core logic, memory, and I/O modules. When you run the place-and-route tool, it places the logic into their assigned regions.

Some regions are user-defined and others are automatically created by the tools to meet routing requirements (for example, Local clock regions).

You can use region constraints to:

- Create user-defined regions such as Inclusive, Exclusive, Empty, LocalClock, and QuadrantClock
- Assign and unassign macros to user-defined regions
- Constrain all the macros connected to a net by assigning them to a specific net region
- Move regions from one set of co-ordinates to another

See Also

Assign Macro to Region

Create Region

Delete Region

Move Region

Unassign macro from region

About Floorplanning, Creating Regions, Editing Regions

Location

Each core, RAM, and I/O macro in the design is associated with a location on the device. When you run the place-and-route tool, it places all of your logic into their assigned locations.

You can use location constraints to:

- Overwrite the existing placements of macros
- Tell the place-and-route tool where to initially place the macros
- Assign I/O macros to specific pins to meet your board's requirements

See Also

Assign I/O to pin

Assign macro to location

Assigning Logic to Locations, Moving Logic to Other Locations, Assigning Pins, Unassigning Pins

I/O Attributes

I/O attributes are the characteristics of logic macros or nets in your design. They indicate placement, implementation, naming, directionality, and other characteristics. This information is used by the design implementation software during the place-and-route of a design.

Input and output attributes are described in the documentation for the I/O Attribute Editor. Attributes applicable to a specific tool are described in the help for that tool.

See the topics in I/O Attributes Reference for more detailed information about each attribute. See also , for a table of attributes for each device family, and Welcome to I/O Attribute Editor.

See Also

I/O Attributes by Family

I/O Standards and I/O Attributes Applicability

I/O Standards Compatibility Matrix

I/O Attributes

I/O Attributes by Family and Device

For details about the I/O standards and attributes supported in this PolarFire release, see the following documents:

PolarFire FPGA User I/Os User Guide

PolarFire FPGA PDC Commands User Guide

I/O Standard

Purpose

Use the I/O standard attribute to assign an I/O standard to an I/O macro.

The following table shows a list of supported I/O standards.

Some I/O standards support only single I/O or differential I/Os while others support Single and Differential I/Os. The table below lists the different I/O standards and the type of I/O they support.

IO_STD Value	Single	Differential
LVTTL	YES	NO
LVSTL11I	YES	YES
LVSTL11II	YES	YES
LVCMOS33	YES	NO
LVCMOS25	YES	NO
LVCMOS18	YES	NO
LVCMOS15	YES	NO
LVCMOS12	YES	NO
PCI	YES	NO
POD12I	YES	YES
POD12II	YES	YES
PPDS33	NO	YES
PPDS25	NO	YES
PPDS18	NO	YES
SLVS33	NO	YES
SLVS25	NO	YES
SLVS18	NO	YES
HCSL33	NO	YES
HCSL25	NO	YES

HCSL18	NO	YES
SLVSE	NO	YES
SLVSE15	NO	YES
BUSLVDSE	NO	YES
BUSLVDSE25	NO	YES
MLVDSE	NO	YES
MLVDSE25	NO	YES
LVDS	NO	YES
LVDS33	NO	YES
LVDS25	NO	YES
LVDS18	NO	YES
BUSLVDS	NO	YES
MLVDS	NO	YES
MIPI12	NO	YES
MIPIE33	NO	YES
MINILVDS	NO	YES
MINILVDS33	NO	YES
MINILVDS25	NO	YES
MINILVDS18	NO	YES
RSDS	NO	YES
RSDS33	NO	YES
RSDS25	NO	YES
RSDS18	NO	YES
LVPECL (only for inputs)	NO	YES
LVPECL33	NO	YES

LVPECL25	NO	YES
LVPECLE33	NO	YES
HSTL15I	YES	YES
HSTL15II	YES	YES
HSTL135I	YES	YES
HSTL135II	YES	YES
HSTLI2I	YES	YES
HSTL12II	YES	YES
SSTL18I	YES	YES
SSTL18II	YES	NO
SSTL15I	YES	YES
SSTL15II	YES	NO
SSTL135I	YES	YES
SSTL135II	YES	YES
SSTL25I	YES	YES
SSTL25II	YES	YES
HSUL18I	YES	YES
HSUL18II	YES	YES
HSUL12I	YES	YES
HSUL12II	YES	YES
SUBLVDS33	NO	YES
SUBLVDS25	NO	YES
SUBLVDS18	NO	YES

Descriptions

Following are brief descriptions of the I/O standard attributes in the table above.

BUSLVDS

Enables multipoint configuration of LVDS; useful when point-to-point communication in LVDS is inadequate.

CMOS (Complementary Metal-Oxide-Semiconductor)

An advanced integrated circuit (IC) manufacturing process technology for logic and memory, characterized by high integration, low cost, low power, and high performance. CMOS logic uses a combination of p-type and n-type metal-oxide-semiconductor field effect transistors (MOSFETs) to implement logic gates and other digital circuits found in computers, telecommunications, and signal processing equipment.

CUSTOM

An option in the I/O Attribute Editor that enables you to customize individual I/O settings such as the I/O threshold, output slew rates, and capacitive loadings on an individual I/O basis. For example, PCI mode output can be set to low-slew rate. For more information, go to the Microsemi SoC web site and check the datasheet for your device.

GTL 2.5 V (Gunning Transceiver Logic 2.5 Volts)

A low-power standard (JESD 8.3) for electrical signals used in CMOS circuits that allows for low electromagnetic interference at high speeds of transfer. It has a voltage swing between 0.4 volts and 1.2 volts, and typically operates at speeds of between 20 and 40MHz. The VCCI must be connected to 2.5 volts.

GTL 3.3 V (Gunning Transceiver Logic 3.3 Volts)

Same as GTL 2.5 V above, except the VCCI must be connected to 3.3 volts.

GTL+ (Gunning Transceiver Logic Plus)

An enhanced version of GTL that has defined slew rates and higher voltage levels. It requires a differential amplifier input buffer and an open-drain output buffer. Even though output is open-drain, the VCCI must be connected to either 2.5 volts or 3.3 volts for SmartFusion, IGLOO, ProASIC3 and Fusion families.

HSTL Class I and II (High-Speed Transceiver Logic)

A general-purpose, high-speed 1.5 V bus standard (EIA/JESD 8-6) for signalling between integrated circuits. The signalling range is 0 V to 1.5 V, and signals can be either single-ended or differential. HSTL requires a differential amplifier input buffer and a push-pull output buffer. It has four classes, of which Microsemi SoC supports Class I and II. These classes are defined by standard EIA/JESD 8-6 from the Electronic Industries Alliance (EIA):

- Class I (unterminated or symmetrically parallel terminated)
- Class II (series terminated)
- Class III (asymmetrically parallel terminated)
- Class IV (asymmetrically double parallel terminated

LPDDR I and II

Low Power double data rate synchronous DRAM for mobile computers.

LVCMOS 3.3 V (Low-Voltage CMOS for 3.3 Volts)

An extension of the LVCMOS standard (JESD 8-5) used for general-purpose 3.3 V applications.

LVCMOS 2.5 V (Low-Voltage CMOS for 2.5 Volts)

An extension of the LVCMOS standard (JESD 8-5) used for general-purpose 2.5 V applications.

LVCMOS 2.5 V/5.5V (Low-Voltage CMOS for 2.5 and 5.0 Volts)

An extension of the LVCMOS standard (JESD 8-5) used for general-purpose 2.5 V and 5.0V applications.

LVCMOS 1.8 V (Low-Voltage CMOS for 1.8 Volts)

An extension of the LVCMOS standard (JESD 8-5) used for general-purpose 1.8 V applications. It uses a 3.3 V-tolerant CMOS input buffer and a push-pull output buffer.

LVCMOS 1.5 V (Low-Voltage CMOS for 1.5 volts)

An extension of the LVCMOS standard (JESD 8-5) used for general-purpose 1.5 V applications. It uses a 3.3 V-tolerant CMOS input buffer and a push-pull output buffer.

LVCMOS 1.2 V (Low-Voltage CMOS for 1.2 volts)

Note: An extension of the LVCMOS standard (JESD 8-5) used for general-purpose 1.2 V applications.

Note: 1.2 voltage is supported for ProASIC3 (A3PL), IGLOOe V2 only, IGLOO V2, and IGLOO PLUS.

LVDS (Low-Voltage Differential Signal)

A moderate-speed differential signalling system, in which the transmitter generates two different voltages which are compared at the receiver. It requires that one data bit be carried through two signal lines; therefore, you need two pins per input or output. It also requires an external resistor termination. The voltage swing between these two signal lines is approximately 350mV (millivolts).

LVPECL (Low-Voltage Positive Emitter Coupled Logic)

PECL is another differential I/O standard. It requires that one data bit is carried through two signal lines; therefore, two pins are needed per input or output. It also requires an external resistor termination. The voltage swing between these two signal lines is approximately 850mV. When the power supply is +3.3 V, it is commonly referred to as low-voltage PECL (LVPECL).

LVTTL/TTL (Low-Voltage Transitor-Transistor Level)

A general purpose standard (EIA/JESDSA) for 3.3 V applications. It uses an LVTTL input buffer and a push-pull output buffer.

MINILVDS

Signaling standard used for display applications with resolutions between video graphics arrays (VGAs) and ultra extended graphic arrays (UXGAs).

MLVDS

MLVDS has two types of receivers. Type-1 is compatible with LVDS and uses a +/- 50 mV threshold. Type-2 receivers allow Wired-Or signaling with M-LVDS devices. For MLVDS:

PCI (Peripheral Component Interface)

A computer bus for attaching peripheral devices to a computer motherboard in a local bus. This standard supports both 33 MHz and 66 MHz PCI bus applications. It uses an LVTTL input buffer and a push-pull output buffer. With the aid of an external resistor, this I/O standard can be 5V-compliant for most families, excluding ProASIC3 families.

PCI-X (Peripheral Component Interface Extended)

An enhanced version of the PCI specification that can support higher average bandwidth; it increases the speed that data can move within a computer from 66 MHz to 133 MHz. PCI-X is backward-compatible, which means that devices can operate at conventional PCI frequencies (33 MHz and 66 MHz). PCI-X is also more fault-tolerant than PCI.

RSDS

Reduced Swing Differential Signaling , a electronic signaling standard and protocol for a chip-to-chip interface. Signaling standard commonly used for display applications with resolutions between video graphics arrays (VGAs) and ultra extended graphic arrays (UXGAs).

SSTL15 Class I and II

I/O standard with a voltage-referenced signal, input (VREF) of 0.75, and an output (VCCIO) voltage of 1.5 V.

SSTL18 Class I and II

SSTL is an electrical interface commonly used with DDR [Double Data Rate] DRAM memory ICs and memory modules. SSTL_18 Series Stub Terminated, used with DDR II memory; requires Vddq = 1.8v, Vt = 0.5 x Vddq

SSTL2 Class I and II (Stub Series Terminated Logic 2.5 V)

A general-purpose 2.5 V memory bus standard (JESD 8-9) for driving transmission lines. This standard was designed specifically for driving the DDR (double-data-rate) SDRAM modules used in computer memory. It requires a differential amplifier input buffer and a push-pull output buffer. It has two classes; Microsemi SoC supports both.

SSTL3 Class I and II (Stub Series Terminated Logic for 3.3 V)

A general-purpose 3.3 V memory bus standard (JESD 8-8) for driving transmission lines.

I/O State in Flash*Freeze Mode

Purpose

Preserves the previous state of the I/O. By default, all the I/Os become tristated when the device goes into Flash*Freeze mode. (A tristatable I/O is an I/O with three output states: high, low, and high impedance.) You can override this default using the FF_IO_STATE attribute. When you set this attribute to True, the I/O remains in the same state in which it was functioning before the device went into Flash*Freeze mode.

Values

You can specify TRISTATE or LAST_VALUE for FF_IO_STATE.

Example

You can set your I/O to the last available value using FF_IO_STATE:

```
set_io A -FF_IO_STATE LAST_VALUE
```

Locked

Purpose

 Indicates whether you can change the current pin assignment during layout.

Values

Use this attribute to lock or unlock the pin assignment. Selecting the check box locks the pin assignment. Clearing the check box unlocks the pin assignment. If locked, you cannot change the pin assignment. PDC values are YES or NO.

Example

```
set_io -fixed YES
```

Output Drive

Purpose

 Every I/O standard has an output drive preset; however, for some I/O standards, you can choose which one to use. The higher the drive, the faster the I/O. The faster the I/O, the more power consumed by the I/O.

Values

Drive strength is programmable for some I/O technologies. See the device silicon user's guide for specific ranges.
 Some I/O technologies are not programmable. Using this attribute in conjunction with non-programmable I/O technology will generate an error no matter what value the attribute is set to.

Example

```
set_io -out_drive 4
```

Output Load

Purpose

Indicates the output-capacitance value based on the I/O standard selected in the I/O Standard cell. This option is not available in software.

Values

You can enter a capacitative load as an integral number of picofarads. The default value varies by device family. If necessary, you can change the output capacitance default setting to improve timing definition and analysis. Both the capacitive loading on the board and the Vil/Vih trip points of driven devices affect output-propagation delay.

SmartTime, Timing-Driven Layout and Back-Annotation automatically uses the modified delay model for delay calculations.

The default value is 5, and the range of possible values is 0-9999.

Example

```
-set_io Y -OUT_LOAD 5
```

Pin Number

Purpose

Use this attribute to change a pin assignment by choosing one of the legal values from the drop-down list. If the pin has been assigned, the pin number appears in this column. If it has not been assigned then Unassigned appears in this column.

Example

```
set_io -pinname AC30
```

Port Name

Purpose

Indicates the port name of the I/O macro. This value is read only.

Resistor Pull

Purpose

Allows inclusion of a weak resistor for either pull-up or pull-down of the input buffer.

Values

Use this attribute to set the resistor pull. Your choices are None, Up (pull-up), or Down (pull-down). The default value is None, except when an I/O exists in the netlist as a port, is not connected to the core, and is configured as an output buffer. In that case, the default setting is for a weak pull-down.

Example

```
-set_io A -RES_PULL NONE
```

Schmitt Trigger

Purpose

A schmitt trigger is a buffer used to convert a slow or noisy input signal into a clean one before passing it to the FPGA. This is a simple, low-cost solution for a user working with low slew-rate signals. Using schmitt-trigger buffers guarantees a fast, noise-free, input signal to the FPGA.

Schmitt-trigger buffers are categorized in three configurations:

- Fixed threshold voltages with non-inverted outputs
- Fixed threshold voltages and inverted outputs
- Variable threshold voltages with non-inverted outputs

With the aid of schmitt-trigger buffers, low slew-rate applications can also be handled with ease. Implementation of these buffers is simple, not expensive, and flexible in that different configurations are possible depending on the application. The characteristics of schmitt-trigger buffers (e.g. threshold voltage) can be fixed or user-adjustable if required.

Values

A schmitt trigger has two possible states: Off (default) or On. The trigger for this circuit to change states is the input voltage level. That is, the output state depends on the input level, and will change only as the input crosses a pre-defined threshold.

Not all I/O technologies support SCHMITT_TRIGGER. Including the attribute in a set_io statement that also specifies a non-supporting I/O technology will create an error.

For more information, see the Using Schmitt Triggers for Low Slew-Rate Input Application Note on the Microsemi SoC web site.

Example

```
set_io A -SCHMITT_TRIGGER On
```

Slew

The slew rate is the amount of rise or fall time an input signal takes to get from logic low to logic high or vice versa. It is commonly defined to be the propagation delay between 10% and 90% of the signal's voltage swing.

Purpose

Indicates the slew rate for output buffers. Generally, available slew rates are high and low.

Values

Values for slew for SF2 are: SLOW (default), MEDIUM, MEDIUM_FAST, and FAST. MSIO and MSIOD banks only accept SLOW for SLEW values on the I/O technologies that support SLEW. DDRIO supports all four values.

Not all I/O technologies support SLEW. Including the attribute in a set_io statement that also specifies a non-supporting I/O technology will create an error.

The SLOW slew rate is incompatible with 3.3 V PCI requirements.

For ProASIC3 families, you can edit the slew for designs using LVTTL, all LVCMOS, or PCIX I/O standards. The other I/O standards have a preset slew value. For those devices that support additional slew values, Microsemi SoC recommends that you use the SLOW and FAST values and let the software map to the

appropriate absolute slew value. The default slew displayed in the I/O Attribute Editor is based on the selected I/O standard. For example, PCI mode sets the default output slew rate to FAST.

Note: One way to eliminate problems with low slew rate is with external .

In some applications, you may require a very fast (i.e. high slew rate) signal, which approaches an ideal switching transition. You can accomplish this by either reducing the track resistance and/or capacitance on the board or increasing the drive capability of the input signal. Both of these options are generally time consuming and costly. Furthermore, the closer the input signal approaches an ideal one, the greater the likelihood of unwanted effects such as increased peak current, capacitive coupling, and ground bounce.

In many cases, you may want to incorporate a finite amount of slew rate into your signal to reduce these effects. On the other hand, if an input signal becomes too slow (i.e. low slew rate), then noise around the FPGA's input voltage threshold can cause multiple state changes. During the transition time, both input buffer transistors could potentially turn on at the same time, which could result in the output of the buffer to oscillate unpredictably. In this situation, the input buffer could still pass signals.

However, these short, unpredictable oscillations would likely cause the device to malfunction.

Example

```
-set_io slew MEDIUM
```

Use Register

Purpose

The input and output registers for each individual I/O can be activated by selecting the check box associated with an I/O. The I/O registers are NOT selected by default.

If this option is yes, the combiner combines the register into the I/O module if possible. This option overrides the default setting in the Compile options. I/O registers are off by default. The following table shows the acceptable values for the -register attribute:

Values

Possible values are yes or no.

Example

```
set_io -register no
```

See Also

[I/O Register Combining Rules](#)

User Reserved

Purpose

You can explicitly reserve a pin in one of the following ways:

- In the I/O Attribute Editor (Package Pins view), select the **User Reserved** checkbox associated with the pin to reserve.
- Select a pin in PinEditor, right-click it, and choose Reserve Pin from the right-click menu.
- Use the reserve command in a PDC file.

Values

The list of possible values for this attribute is the list of package pins.

Example

```
reserve -pinname "F2 B4 B3"
```

Add New Port Dialog Box

To access this dialog box, from the **I/O Attribute Editor** menu, choose **Add Port**. You can also right-click a row in the **Ports** tab of the **I/O Attribute Editor**, and choose **Add New Port** to display this dialog box.

Use this dialog box to add a new port to your design.

Name

Enter a name for the new port.

Direction

Select one of the following options:

Input

Select this option if the port is to receive a signal.

Output

Select this option if the port is to send a signal.

Bi-directional (Inout)

Select this option if the port will both send and receive a signal.

Modify Port Dialog Box

To access this dialog box, from the **I/O Attribute Editor** menu, choose **Modify Port**. You can also right-click a row in the **Ports** tab of the **I/O Attribute Editor**, and choose **Modify Port** to display this dialog box.

Use this dialog box to modify the name or direction of an existing port in your design.

Name

Enter a new name for the port.

Direction

Select one of the following options:

Input

Select this option if the port is to receive a signal.

Output

Select this option if the port is to send a signal.

Bi-directional (Inout)

Select this option if the port will both send and receive a signal.

Entering Constraints

You can enter design constraints in the following ways:

- **Importing constraint files**: You can import - PDC or SDC constraint files for PolarFire, SmartFusion2, IGLOO2, RTG4, SmartFusion, IGLOO, ProASIC3, Fusion families

- **Using constraint editor tools**: The constraint editor is a graphical user interface (GUI) tools for creating and modifying physical, logical, and timing constraints. Using these tools enables you to enter constraints without having to understand PDC or other file syntax. The constraints you enter in the interactive tools are saved in a PDC or SDC file inside the Libero SoC project.

For **SmartFusion, IGLOO, ProASIC3, Fusion**, use the tools within the MultiView Navigator:

- ChipPlanner - Sets location and region assignments
- PinEditor in MVN - Sets the pin location constraints
- I/O Attribute Editor - Sets I/O attributes
- SmartTime Constraints Editor - Enables you to view and edit timing constraints

For SmartFusion2, IGLOO2, RTG4 constraints in Classic Constraint Flow, see the SmartFusion2-specific content.

For SmartFusion2, IGLOO2, RTG4, PolarFire constraints in Enhanced Constraint Flow, see Constraint Manager.

See Also

Constraint Support by Family

Constraint Entry

Constraint File Format by Family

Designer Naming Conventions

Importing Constraint Files

For details on how to import Constraint Files into a Libero SoC PolarFire project, see Constraint Manager.

For details about how to import Constraint Files into a Libero SoC Enhanced Constraint Flow project, see Constraint Manager.

For all other Libero SoC projects, you can import a constraint file as either a source file or an auxiliary file. For details on how to import constraints files, refer to .

Source File

Import constraints file as source files if they were created with external tools that will be tracked (audited). This helps to coordinate the design changes better. For details on how to import source files, refer to .

The following table shows different constraints format files that can be imported as source files for specific families.

Table 4 · File Types You Can Imported as Source Files

Source Files	File Type Extension	Family
Physical Design Constraint File	*.pdc	PolarFire, SmartFusion2, IGLOO2, RTG4, SmartFusion, IGLOO, ProASIC3, Fusion
Synopsys Constraint File	*.sdc	PolarFire, SmartFusion2, IGLOO2, RTG4, SmartFusion, IGLOO, ProASIC3, Fusion

Auxiliary File

When you import a constraint file as an auxiliary file, it is not audited and is treated more as one-time data-entry or data-change events, similar to entering data using one of the interactive editors. For details on how to import auxiliary files, refer to .

The following table shows different constraints format files that can be imported as auxiliary files for specific families.

Table 5 · File Types You Can Import as Auxiliary Files

Auxiliary Files	File Type Extension	Family
SDC	*.sdc	SmartFusion, IGLOO, ProASIC3, Fusion
Physical Design Constraint[**]	*.pdc	IGLOO, Fusion, ProASIC3
Switching Activity Intermediate File/Format	*.saif	IGLOO, Fusion, ProASIC3
Value Change Dump file	*.vcd	IGLOO, Fusion, ProASIC3

(*) When you import SDC as an auxiliary file, you can specify only one file in the **File > Import Auxiliary Files** dialog box.

(**) Not all PDC commands are supported when a PDC file is imported as an auxiliary file; some must be imported as source files. When importing a PDC file as an auxiliary file, the new or modified PDC constraints are merged with the existing constraints. The software resolves any conflicts between new and existing physical constraints and displays the appropriate message. Most PDC commands can be imported as

auxiliary files. PDC commands that are not supported when the PDC file is imported as an auxiliary file are noted in their respective help topics.

You can either overwrite or retain your existing timing and physical constraints. For details on how to preserve the existing timing constraints, refer to . For details on how to preserve the existing physical constraints, refer to .

See Also

Importing source files

Importing auxiliary files

Keep Existing Timing Constraints

Keep Existing Physical Constraints

About SmartTime Constraints Editor

SmartTime Constraints Editor is an interface that enables you to view and edit timing constraints. Use this editor to view, edit, and create timing constraints used by the SmartTime timing analysis and timing-driven optimization tools. The editor includes powerful visual dialogs that guide you toward capturing your timing requirements and timing exceptions quickly and correctly. The editor is also closely connected to the analysis view of SmartTime (SmartTime Timing Analyzer) that enables you to quickly analyze the impact of constraint changes.

Exporting Constraint Files

The following table shows a complete list of constraint files that you can export along with the supported family.

File	File Extension	Families
SDC	*.sdc	SmartFusion2, IGLOO2, SmartFusion, RTG4, PolarFire, IGLOO, ProASIC3, Fusion
Physical Design Constraint	*.pdc	SmartFusion2, IGLOO2, RTG4, PolarFire, SmartFusion, IGLOO, ProASIC3, Fusion

Constraints by Name: Timing

Create Clock

Families Supported

The following table shows which families support this constraint and which file formats and tools you can use to enter or modify it:

Families	SDC	SmartTime
PolarFire	X	X
RTG4	X	X
IGLOO2	X	X
IGLOO	X	X
SmartFusion2	X	X
SmartFusion	X	X
Fusion	X	X
ProASIC3	X	X

Purpose

Use this constraint to create a clock constraint at a specific source and define its waveform. The static timing analysis tool uses this information to propagate the waveform across the clock network to the clock pins of all sequential elements driven by the defined clock source. The clock information is also used to compute the slacks in the specified clock domain, display setup and hold violations, and drive optimization tools such as place-and-route.

Tools /How to Enter

You can use one or more of the following methods to enter clock constraints:

- SDC - create_clock
- SmartTime - Specifying Clock Constraint

See Also

Constraint Entry

create_clock (SDC)

Clock Definition

Specifying Clock Constraint

Create Generated Clock

Families Supported

The following table shows which families support this constraint and which file formats and tools you can use to enter or modify it:

Families	SDC	SmartTime	Constraint Editor
PolarFire	X		X^2
IGLOO2	X	X [1]	X^2
SmartFusion2	X	X [1]	X^2
RTG4	X	X [1]	X^2
IGLOO	X	X	
SmartFusion	X	X	
Fusion	X	X	
ProASIC3	X	X	
1 For Libero SoC Design Flow (Classic Constraint Flow) 2 For Libero SoC Design Flow (Enhanced Constraint Flow) - SmartFusion2, IGLOO2, RTG4, and PolarFire			

Purpose

Use this constraint to create an internally generated clock constraint, such as clock dividers and PLL. The generated clock is defined in terms of multiplication and/or division factors with respect to a reference clock pin. When the reference clock pin changes, the generated clock is updated automatically.

Tools /How to Enter

You can use one or more of the following methods to enter clock constraints:

- SDC – create_generated_clock
- SmartTime - Specifying Generated Clock Constraint

See Also

Constraint Entry
create_generated_clock (SDC)
Specifying Generated Clock Constraint

Remove Clock Uncertainty

Families Supported

The following table shows which families support this constraint and which file formats and tools you can use to enter or modify it:

Families	SDC Constraints	Tcl command passed to SmartTime
PolarFire	No	No
RTG4	No	Yes[1]
IGLOO	No	Yes[1]
IGLOO2	No	Yes[1]
SmartFusion2	No	Yes[1]
SmartFusion	No	Yes[1]
Fusion	No	Yes[1]
ProASIC3	No	Yes[1]
Yes[1] = For Libero SoC Design Flow (Classic Constraint Flow)		

Purpose

Use this constraint to remove the timing uncertainty between two clock waveforms within SmartTime.

You can remove clock uncertainty constraints in an SDC file, which you can either create yourself or generate with Synthesis tools, at the same time you import the netlist. Alternatively, you can remove clock uncertainty using the GUI tools in the Designer software.

Tools /How to Enter

You can use one or more of the following commands or GUI tools to remove clock uncertainty:

- SDC – remove_clock_uncertainty
- SmartTime - Specifying Clock-to-Clock Uncertainty Constraint

See Also

Constraint Entry

set_clock_uncertainty(SDC)

SmartTime User's Guide: Specifying Clock-to-Clock Uncertainty Constraint

Set Clock Latency

Families Supported

The following table shows which families support this constraint and which file formats and tools you can use to enter or modify it:

Families	SDC	SmartTime	Constraint Editor
PolarFire	X		X^2
IGLOO2	X	X^1	X^2
SmartFusion2	X	X^1	X^2
RTG4	X	X^1	X^2
IGLOO	X	X	
SmartFusion	X	X	
Fusion	X	X	
ProASIC3	X	X	
1 For Libero SoC Design Flow (Classic Constraint Flow) 2 For Libero SoC Design Flow (Enhanced Constraint Flow) - SmartFusion2, IGLOO2, RTG4, and PolarFire			

Purpose

Use this constraint to define the delay between an external clock source and the definition pin of a clock within SmartTime.

You can set clock latency constraints in an SDC file, which you can either create yourself or generate with Synthesis tools, at the same time you import the netlist. Alternatively, you can set clock latency using the GUI tools in the Designer software when you implement your design.

Tools /How to Enter

You can use one or more of the following commands or GUI tools to set clock latency:

- SDC – set_clock_latency
- SmartTime - Specifying Clock Source Latency

See Also

Constraint Entry
set_clock_latency (SDC)
Specifying Clock Source Latency

Set Clock Uncertainty Constraint

Families Supported

The following table shows which families support this constraint and which file formats and tools you can use to enter or modify it:

Families	SDC	SmartTime	Constraints Editor
PolarFire	X		X^2
RTG4	X	X^1	X^2
IGLOO2	X	X^1	X^2
SmartFusion2	X	X^1	X^2
IGLOO	X	X	
SmartFusion	X	X	
Fusion	X	X	
ProASIC3	X	X	
1 For Libero SoC Design Flow (Classic Constraint Flow) 2 For Libero SoC Design Flow (Enhanced Constraint Flow) - SmartFusion2, IGLOO2, RTG4, and PolarFire			

Purpose

Use this constraint to define the timing uncertainty between two clock waveforms or maximum skew within SmartTime.

You can set clock uncertainty constraints in an SDC file, which you can either create yourself or generate with Synthesis tools, at the same time you import the netlist. Alternatively, you can set clock uncertainty using the GUI tools in the Designer software when you implement your design.

Tools /How to Enter

You can use one or more of the following commands or GUI tools to set clock uncertainty:

- SDC – set_clock_uncertainty
- SmartTime - Specifying Clock-to-Clock Uncertainty Constraint

See Also

Constraint Entry
set_clock_uncertainty(SDC)

Set Disable Timing Constraint

Families Supported

The following table shows which families support this constraint and which file formats and tools you can use to enter or modify it:

Families	SDC	SmartTime	Constraints Editor
PolarFire	X		X^2
RTG4	X	$X^{\,1}$	X^2
IGLOO2	X	$X^{\,1}$	X^2
SmartFusion2	X	$X^{\,1}$	X^2
IGLOO	X	X	
SmartFusion	X	X	
Fusion	X	X	
ProASIC3	X	X	
1 For Libero SoC Design Flow (Classic Constraint Flow) 2 For Libero SoC Design Flow (Enhanced Constraint Flow) - SmartFusion2, IGLOO2, RTG4, and PolarFire			

Purpose

Use this constraint disable the timing arc in the specified ports on a path.

You can disable the timing arc in an SDC file, which you can either create yourself or generate with Synthesis tools, at the same time you import the netlist. Alternatively, you can disable the timing arc using the GUI tools in the Designer software when you implement your design.

Tools /How to Enter

You can use one or more of the following commands or GUI tools to set maximum delay exception constraints:

- SDC – set_disable_timing

See Also

Constraint Entry

set_disable_timing(SDC)

Set False Path

Families Supported

The following table shows which families support this constraint and which file formats and tools you can use to enter or modify it:

Families	SDC	SmartTime	Constraints Editor
PolarFire	X		X^2
RTG4	X	X^1	X^2
IGLOO2	X	X^1	X^2
SmartFusion2	X	X^1	X^2
IGLOO	X	X	
SmartFusion	X	X	
Fusion	X	X	
ProASIC3	X	X	
1 For Libero SoC Design Flow (Classic Constraint Flow)			
2 For Libero SoC Design Flow (Enhanced Constraint Flow) - SmartFusion2, IGLOO2, RTG4, and PolarFire			

Purpose

Use this constraint to identify paths in the design that should be disregarded during timing analysis and timing optimization.

By definition, false paths are paths that cannot be sensitized under any input vector pair. Therefore, including false paths in timing calculation may lead to unrealistic results. For accurate static timing analysis, it is important to identify the false paths.

You can set false paths constraints in an SDC file, which you can either create yourself or generate with Synthesis tools, at the same time you import the netlist. Alternatively, you can set false paths using the GUI tools in the Designer software when you implement your design.

Tools /How to Enter

You can use one or more of the following commands or GUI tools to set false paths:

- SDC – set_false_path
- SmartTime - Specifying False Path Constraint

See Also

Constraint Entry
set_false_path (SDC)
Breaks Tab
Specifying False Path Constraint

Set Input Delay

Families Supported

The following table shows which families support this constraint and which file formats and tools you can use to enter or modify it:

Families	SDC	SmartTime	Constraints Editor
PolarFire	X		X^2
SmartFusion2	X	X [1]	X^2
IGLOO2	X	X [1]	X^2
RTG4	X	X [1]	X^2
SmartFusion	X	X	
Fusion	X	X	
ProASIC3 (except ProASIC3 nano and ProASIC3L)	X	X	
IGLOOe (not supported by other IGLOO devices)	X	X	
1 For Libero SoC Design Flow (Classic Constraint Flow)			
2 For Libero SoC Design Flow (Enhanced Constraint Flow) - SmartFusion2, IGLOO2, RTG4, and PolarFire			

Purpose

Use this constraint to define the arrival time relative to a clock.

Tools /How to Enter

You can use one or more of the following methods to set input delay constraint:

- SDC – set_input_delay
- SmartTime - Specifying Input Delay Constraint

See Also

Constraint Entry

set_input_delay (SDC)

Specifying Input Delay Constraint

Set Load on Output Port

Families Supported

The following table shows which families support this constraint and which file formats and tools you can use to enter or modify it:

Families	SDC	I/O Attribute Editor
RTG4	X	X
IGLOO	X	X
IGLOO2	X	X
SmartFusion2	X	X
SmartFusion	X	X
Fusion	X	X
ProASIC3	X	X

Purpose

Use this constraint to set the capacitance to a specified value on a specified port.

Delay on a given path depends on the load at the output pin of the device. For an accurate static timing analysis of a given design, it is important to set the load on the port which can be taken into account for delay calculations.

Tools /How to Enter

You can use one or more of the following commands or GUI tools to set the load on a port:

- SDC – set_load
- I/O Attribute Editor – Editing I/O Attributes
- SmartTime Constraints Editor GUI – Changing Output Port Capacitance

Note: You can also set the output load using the pin_assign command in a Tcl script.

See Also

Constraint Entry
set_load (SDC)
pin_assign
Editing I/O Attributes

Set Maximum Delay

Families Supported

The following table shows which families support this constraint and which file formats and tools you can use to enter or modify it:

Families	SDC	SmartTime	Constraints Editor
PolarFire	X		X^2
RTG4	X	X^1	X^2
IGLOO2	X	X^1	X^2
SmartFusion2	X	X^1	X^2
IGLOO	X	X	
SmartFusion	X	X	
Fusion	X	X	
ProASIC3	X	X	
1 For Libero SoC Design Flow (Classic Constraint Flow) 2 For Libero SoC Design Flow (Enhanced Constraint Flow) - SmartFusion2, IGLOO2, RTG4, and PolarFire			

Purpose

Use this constraint to set the maximum delay exception between the specified ports on a path.

You can set maximum delay exception in an SDC file, which you can either create yourself or generate with Synthesis tools, at the same time you import the netlist. Alternatively, you can set maximum delay exceptions using the GUI tools in the Designer software when you implement your design.

Tools /How to Enter

You can use one or more of the following commands or GUI tools to set maximum delay exception constraints:

- SDC – set_max_delay
- SmartTime – Specifying Maximum Delay Constraint

See Also

Constraint Entry

set_max_delay (SDC)

Set Minimum Delay

Families Supported

The following table shows which families support this constraint and which file formats and tools you can use to enter or modify it:

Families	SDC	SmartTime	Constraints Editor
PolarFire	X		X^2
RTG4	X	X [1]	X^2
IGLOO2	X	X [1]	X^2
SmartFusion2	X	X [1]	X^2
IGLOO	X	X	
SmartFusion	X	X	
Fusion	X	X	
ProASIC3	X	X	
1 For Libero SoC Design Flow (Classic Constraint Flow) 2 For Libero SoC Design Flow (Enhanced Constraint Flow) - SmartFusion2, IGLOO2, RTG4, and PolarFire			

Purpose

Use this constraint to set the minimum delay exception between the specified ports on a path.

You can set minimum delay exception in an SDC file, which you can either create yourself or generate with Synthesis tools, at the same time you import the netlist. Alternatively, you can set minimum delay exceptions using the GUI tools in the Designer software when you implement your design.

Tools /How to Enter

You can use one or more of the following commands or GUI tools to set maximum delay exception constraints:

- SDC – set_min_delay
- SmartTime – Specifying minimum delay constraint

See Also

Constraint Entry

set_min_delay (SDC)

Set Multicycle Path

Families Supported

The following table shows which families support this constraint and which file formats and tools you can use to enter or modify it:

Families	SDC	SmartTime	Constraints Editor
PolarFire	X		X[2]
RTG4	X	X [1]	X[2]
IGLOO2	X	X [1]	X[2]
SmartFusion2	X	X [1]	X[2]
IGLOO	X	X	
SmartFusion	X	X	
Fusion	X	X	
ProASIC3	X	X	
1 For Libero SoC Design Flow (Classic Constraint Flow) 2 For Libero SoC Design Flow (Enhanced Constraint Flow) - SmartFusion2, IGLOO2, RTG4, and PolarFire			

Purpose

Use this constraint to identify paths in the design that take multiple clock cycles.

You can set multicycle path constraints in an SDC file, which you can either create yourself or generate with Synthesis tools, at the same time you import the netlist. Alternatively, you can set multicycle paths using the GUI tools in the Designer software when you implement your design.

Tools /How to Enter

You can use one or more of the following commands or GUI tools to set multicycle paths constraints:

- SDC – set_multicycle_path
- SmartTime – Specifying Input Delay Constraint

See Also

Constraint Entry

`set_multicycle_paths` (SDC)

Specifying Input Delay Constraint

Set Output Delay

Families Supported

The following table shows which families support this constraint and which file formats and tools you can use to enter or modify it:

Families	SDC	SmartTime	Constraints Editor
PolarFire	X		X[2]
RTG4	X	X [1]	X[2]
IGLOO2	X	X [1]	X[2]
SmartFusion2	X	X [1]	X[2]
SmartFusion	X	X	
IGLOO	X	X	
Fusion	X	X	
ProASIC3	X	X	
	1 For Libero SoC Design Flow (Classic Constraint Flow) 2 For Libero SoC Design Flow (Enhanced Constraint Flow) - SmartFusion2, IGLOO2, RTG4, and PolarFire		

Purpose

Use this constraint to set the output delay of an output relative to a clock.

Tools /How to Enter

You can use one or more of the following methods to set output delay constraints:

- SDC – set_output_delay
- SmartTime – Specifying Output Delay Constraint

See Also

Constraint Entry

set_output_delay (SDC)

Assign I/O Macro to Location

Families Supported

The following table shows which families support this constraint and which file formats and tools you can use to enter or modify it:

Families	PDC	ChipPlanner
PolarFire	X	X
RTG4	X	X
IGLOO	X	X
IGLOO2	X	X
SmartFusion2	X	X
SmartFusion	X	X
Fusion	X	X
ProASIC3	X	X

Purpose

Use this constraint to assign one or more I/O macros to a specific location. You can define the location using array co-ordinates.

By confining macros to one area, you can keep the nets connected to that area, resulting in better timing and better floorplanning. Sometimes placing some macros at specific locations can also result in meeting timing closures.

Tools /How to Enter

You can use one or more of the following commands or GUI tools to assign a macro to a location:

- PDC - set_location
- ChipPlanner -

See Also

Constraint Entry

set_location (PDC)

MultiView Navigator User's Guide: Assigning Logic to Locations

ChipEditor User's Guide: Assigning Logic

Assign Macro to Region

Families Supported

The following table shows which families support this constraint and which file formats and tools you can use to enter or modify it:

Families	PDC	ChipPlanner
PolarFire	X	X
RTG4	X	X
IGLOO	X	X
IGLOO2	X	X
SmartFusion2	X	X
SmartFusion	X	X
Fusion	X	X
ProASIC3	X	X

Purpose

Use this constraint to assign one or more macros to a specific region.

By confining macros to one area, you can keep the nets connected to that area, resulting in better timing and better floorplanning.

You can use the define_region PDC command to create a region, and then use the assign_region PDC command to constrain a set of existing macros to that region.

You can also use the MultiView Navigator tool to create regions for any of the supported families.

Tools /How to Enter

You can use one or more of the following commands or GUI tools to assign a macro to a region:

- PDC - assign_region
- ChipPlanner - Assigning a macro to a region

See Also

Constraint Entry

assign_region (PDC)

Assigning a Macro to a Region

Assign Net to Region

Families Supported

The following table shows which families support this constraint and which file formats and tools you can use to enter or modify it:

Families	PDC	ChipPlanner
RTG4	X	X
IGLOO	X	X
IGLOO2	X	X
SmartFusion2	X	X
SmartFusion	X	X
Fusion	X	X
ProASIC3	X	X

Purpose

Use this constraint to place all the loads of a net into a given region. This constraint is useful for high fan-out or critical path nets or bus control logic.

Constraining nets to a region helps to control the connection delays from the net's driver to the logic instances it fans out to. You can adjust the size of the region to pack logic more closely together, hence, improving its net delays.

Suppose you have a global net with loads that span across the whole chip. When you constrain this net to a specific region, you force the loads of this global net into the given region. Forcing loads into a region frees up some areas that were previously used. You can then use these free areas for high-speed local clocks/spines.

Macros connected to a specific net can be assigned to a region in the device. The region can be defined using the define_region PDC command.

When assigning a net to a region, all of the logic driven by that net will be assigned to that region.

Using Regions for Critical Path and High Fan-out Nets

You should assign high fan-out or critical path nets to a region only after you have used up your global routing and clock spine networks. If you have determined, through timing analysis, that certain long delay nets are creating timing violations, assign them to regions to reduce their delays.

Before creating your region, determine if any logic connected to instances spanned by these nets have any timing requirements. Your region could alter the placement of all logic assigned to it. This may have an undesired side effect of altering the timing delays of some logic paths that cross through the region but are not assigned to it. These paths could fail your timing constraints depending on which net delays have been altered.

Tools /How to Enter

You can use one or more of the following commands or GUI tools to assign a net to a region:

- PDC
- ChipPlanner

See Also

[Constraint Entry](#)

`assign_net_macros` (PDC)

[Assigning a Net to a Region](#)

Configure I/O Bank

Families Supported

The following table shows which families support this constraint and which file formats and tools you can use to enter or modify it:

Families	PDC	ChipPlanner
PolarFire	X	X
RTG4	X	X
IGLOO	X	X
IGLOO2	X	X
SmartFusion2	X	X
SmartFusion	X	X
Fusion	X	X
ProASIC3	X	X

Purpose

Use this constraint to set the I/O supply voltage (VCCI) for I/O banks.

I/Os are organized into banks. The configuration of these banks determines the I/O standards supported. Since each I/O bank has its own user-assigned input reference voltage (VREF) and an input/output supply voltage, only I/Os with compatible standards can be assigned to the same bank.

For IGLOO, SmartFusion and Fusion devices you can use the `set_iobank` PDC command to set the input/output supply voltage and the input reference voltage for an I/O bank.

However, for ProASIC3 devices, you can use this command to set only the input/output supply voltage for an I/O bank.

Tools /How to Enter

You can use one or more of the following commands or GUI tools to configure I/O banks:

- PDC - set_iobank
- ChipPlanner - Manually Assigning Technologies to I/O Banks

See Also

[Constraint Entry](#)

[set_iobank](#)

MultiView Navigator User's Guide: [Manually Assigning Technologies to I/O Banks](#)

Create Region

Families Supported

The following table shows which families support this constraint and which file formats and tools you can use to enter or modify it:

Families	PDC	ChipPlanner
PolarFire	X	X
RTG4	X	X
IGLOO	X	X
IGLOO2	X	X
SmartFusion2	X	X
SmartFusion	X	X
Fusion	X	X
ProASIC3	X	X

Purpose

Use this constraint to create either a rectangular or rectilinear region on a device.

You can create a region within a device for setting specific physical constraints or for achieving better floorplanning. You can define regions with the array coordinates for that particular device.

You can use the `define_region` PDC command to create a rectangular or rectilinear region, and then use the `assign_region` PDC command to constrain a set of macros to that region.

You can also use the MultiView Navigator tool to create regions for any of the supported families.

Tools /How to Enter

You can use one or more of the following commands or GUI tools to create a region constraint:

- PDC -`define_region`define_region
- ChipPlanner - Creating RegionsCreating_regions

See Also

Constraint Entry
`define_region` (PDC)
Creating Regions

Move Region

Families Supported

The following table shows which families support this constraint and which file formats and tools you can use to enter or modify it:

Families	PDC	ChipPlanner
PolarFire	X	X
RTG4	X	X
IGLOO	X	X
IGLOO2	X	X
SmartFusion2	X	X
SmartFusion	X	X
Fusion	X	X
ProASIC3	X	X

Purpose

Use this constraint to move the location of a previously defined region.

Tools /How to Enter

You can use one or more of the following commands or GUI tools to move a region:

- PDC - move_region
- ChipPlanner - Editing Regions

See Also

Constraint Entry

move_region (PDC)

MultiView Navigator User's Guide: Editing Regions

Constraints by File Format - SDC Command Reference

About Synopsys Design Constraints (SDC) Files

Synopsys Design Constraints (SDC) is a Tcl-based format used by Synopsys tools to specify the design intent, including the timing and area constraints for a design. Microsemi tools use a subset of the SDC format to capture supported timing constraints. Any timing constraint that you can enter using Designer tools can also be specified in an SDC file.

Use the SDC-based flow to share timing constraint information between Microsemi tools and third-party EDA tools.

Command	Action
create_clock	Creates a clock and defines its characteristics
create_generated_clock	Creates an internally generated clock and defines its characteristics
remove_clock_uncertainty	Removes a clock-to-clock uncertainty from the current timing scenario.
set_clock_latency	Defines the delay between an external clock source and the definition pin of a clock within SmartTime
set_clock_uncertainty	Defines the timing uncertainty between two clock waveforms or maximum skew
set_false_path	Identifies paths that are to be considered false and excluded from the timing analysis
set_input_delay	Defines the arrival time of an input relative to a clock
set_load	Sets the load to a specified value on a specified port
set_max_delay	Specifies the maximum delay for the timing paths
set_min_delay	Specifies the minimum delay for the timing paths
set_multicycle_path	Defines a path that takes multiple clock cycles
set_output_delay	Defines the output delay of an output relative to a clock

See Also

Constraint Entry
SDC Syntax Conventions
Importing Constraint Files

SDC Syntax Conventions

The following table shows the typographical conventions that are used for the SDC command syntax.

Syntax Notation	Description
command - argument	Commands and arguments appear in `Courier New` typeface.
variable	Variables appear in blue, italic *Courier New* typeface. You must substitute an appropriate value for the variable.
[-argument *value*]	Optional arguments begin and end with a square bracket.

Note: SDC commands and arguments are case sensitive.

Example

The following example shows syntax for the create_clock command and a sample command:

```
create_clock -period period_value [-waveform edge_list] source
```

```
create_clock -period 7 -waveform {2 4}{CLK1}
```

Wildcard Characters

You can use the following wildcard characters in names used in the SDC commands:

Wildcard	What it does
\	Interprets the next character literally
*	Matches any string

Note: The matching function requires that you add a backslash (\) before each slash in the pin names in case the slash does not denote the hierarchy in your design.

Special Characters ([ ], { }, and \)

Square brackets ([ ]) are part of the command syntax to access ports, pins and clocks. In cases where these netlist objects names themselves contain square brackets (for example, buses), you must either enclose the names with curly brackets ({}) or precede the open and closed square brackets ([ ]) characters with a backslash (\). If you do not do this, the tool displays an error message.

For example:

```
create_clock -period 3 clk\[0\]
set_max_delay 1.5 -from [get_pins ff1\[5\]:CLK] -to [get_clocks {clk[0]}]
```

Although not necessary, Microsemi recommends the use of curly brackets around the names, as shown in the following example:

```
set_false_path -from {data1} -to [get_pins {reg1:D}]
```

In any case, the use of the curly bracket is mandatory when you have to provide more than one name.

For example:

```
set_false_path -from {data3 data4} -to [get_pins {reg2:D reg5:D}]
```

Entering Arguments on Separate Lines

If a command needs to be split on multiple lines, each line except the last must end with a backslash (\) character as shown in the following example:

```
set_multicycle_path 2 -from \
[get_pins {reg1*}] \
-to {reg2:D}
```

See Also

About SDC Files

Referenced Topics

create_clock

SDC command; creates a clock and defines its characteristics.

```
create_clock -name name -period period_value [-waveform edge_list]  source
```

Arguments

-name *name*

Specifies the name of the clock constraint. This parameter is required for virtual clocks when no clock source is provided.

-period *period_value*

Specifies the clock period in nanoseconds. The value you specify is the minimum time over which the clock waveform repeats. The period_value must be greater than zero.

-waveform *edge_list*

Specifies the rise and fall times of the clock waveform in ns over a complete clock period. There must be exactly two transitions in the list, a rising transition followed by a falling transition. You can define a clock starting with a falling edge by providing an edge list where fall time is less than rise time. If you do not specify -waveform option, the tool creates a default waveform, with a rising edge at instant 0.0 ns and ©a falling edge at instant (period_value/2)ns.

source

Specifies the source of the clock constraint. The source can be ports or pins in the design. If you specify a clock constraint on a pin that already has a clock, the new clock replaces the existing one. Only one source is accepted. Wildcards are accepted as long as the resolution shows one port or pin.

Supported Families

SmartFusion2, IGLOO2, RTG4, PolarFire, SmartFusion, IGLOO, ProASIC3, Fusion

Description

Creates a clock in the current design at the declared source and defines its period and waveform. The static timing analysis tool uses this information to propagate the waveform across the clock network to the clock pins of all sequential elements driven by this clock source.

The clock information is also used to compute the slacks in the specified clock domain that drive optimization tools such as place-and-route.

Exceptions

- None

Examples

The following example creates two clocks on ports CK1 and CK2 with a period of 6, a rising edge at 0, and a falling edge at 3:

```
create_clock -name {my_user_clock} -period 6 CK1
create_clock -name {my_other_user_clock} –period 6 –waveform {0 3} {CK2}
```

The following example creates a clock on port CK3 with a period of 7, a rising edge at 2, and a falling edge at 4:

```
create_clock –period 7 –waveform {2 4} [get_ports {CK3}]
```

Microsemi Implementation Specifics

- The -waveform in SDC accepts waveforms with multiple edges within a period. In Microsemi design implementation, only two waveforms are accepted.

- SDC accepts defining a clock on many sources using a single command. In Microsemi design implementation, only one source is accepted.
- The source argument in SDC create_clock command is optional. This is in conjunction with the -name argument in SDC to support the concept of virtual clocks. In Microsemi implementation, source is a mandatory argument as -name and virtual clocks concept is not supported.
- The -domain argument in the SDC create_clock command is not supported.

See Also

[Constraint Support by Family](#)

[Constraint Entry Table](#)

[SDC Syntax Conventions](#)

[Clock](#) Definition

[Create Clock](#)

[Create a New Clock Constraint](#)

create_generated_clock

SDC command; creates an internally generated clock and defines its characteristics.

```
create_generated_clock -name {name -source reference_pin [-divide_by divide_factor] [-
multiply_by multiply_factor] [-invert] source -pll_output pll_feedback_clock -pll_feedback
pll_feedback_input
```

Arguments

-name *name*

Specifies the name of the clock constraint. This parameter is required for virtual clocks when no clock source is provided.

-source *reference_pin*

Specifies the reference pin in the design from which the clock waveform is to be derived.

-divide_by*divide_factor*

Specifies the frequency division factor. For instance if the *divide_factor* is equal to 2, the generated clock period is twice the reference clock period.

-multiply_by *multiply_factor*

Specifies the frequency multiplication factor. For instance if the *multiply_factor* is equal to 2, the generated clock period is half the reference clock period.

-invert

Specifies that the generated clock waveform is inverted with respect to the reference clock.

source

Specifies the source of the clock constraint on internal pins of the design. If you specify a clock constraint on a pin that already has a clock, the new clock replaces the existing clock. Only one source is accepted. Wildcards are accepted as long as the resolution shows one pin.

-pll_output *pll_feedback_clock*

Specifies the output pin of the PLL which is used as the external feedback clock. This pin must drive the feedback input pin of the PLL specified using the –pll_feedback option. The PLL will align the rising edge of the reference input clock to the feedback clock. This is a mandatory argument if the PLL is operating in external feedback mode.

-pll_feedback *pll_feedback_input*

Specifies the feedback input pin of the PLL. This pin must be driven by the output pin of the PLL specified using the –pll_output option. The PLL will align the rising edge of the reference input clock to the external feedback clock. This is a mandatory argument if the PLL is operating in external feedback mode.

Supported Families

SmartFusion2, IGLOO2, RTG4, PolarFire, SmartFusion, IGLOO, ProASIC3, Fusion

Description

Creates a generated clock in the current design at a declared source by defining its frequency with respect to the frequency at the reference pin. The static timing analysis tool uses this information to compute and propagate its waveform across the clock network to the clock pins of all sequential elements driven by this source.

The generated clock information is also used to compute the slacks in the specified clock domain that drive optimization tools such as place-and-route.

Examples

The following example creates a generated clock on pin U1/reg1:Q with a period twice as long as the period at the reference port CLK.

```
create_generated_clock -name {my_user_clock} –divide_by 2 –source [get_ports {CLK}]
U1/reg1/Q
```

The following example creates a generated clock at the primary output of myPLL with a period ¾ of the period at the reference pin clk.

```
create_generated_clock –divide_by 3 –multiply_by 4 -source clk [get_pins {myPLL/CLK1}]
```

The following example creates a generated clock named system_clk on the GL2 output pin of FCCC_0 with a period equal to half the period of the source clock. The constraint also identifies GL2 output pin as the external feedback clock source and CLK2 as the feedback input pin for FCCC_0.

```
create_generated_clock -name { system_clk } \
-multiply_by 2 \
-source { FCCC_0/CCC_INST/CLK3_PAD } \
-pll_output { FCCC_0/CCC_INST/GL2 } \
-pll_feedback { FCCC_0/CCC_INST/CLK2 } \
{ FCCC_0/CCC_INST/GL2 }
```

Microsemi Implementation Specifics

- SDC accepts either –multiply_by or –divide_by option. In Microsemi design implementation, both are accepted to accurately model the PLL behavior.
- SDC accepts defining a generated clock on many sources using a single command. In Microsemi design implementation, only one source is accepted.
- The -duty_cycle ,-edges and –edge_shift options in the SDC create_generated_clock command are not supported in Microsemi design implementation.

See Also

Constraint Support by Family

Constraint Entry Table

SDC Syntax Conventions

Create Generated Clock Constraint (SDC)

set_clock_latency

SDC command; defines the delay between an external clock source and the definition pin of a clock within SmartTime.

```
set_clock_latency -source [-rise][-fall][-early][-late] delay clock
```

Arguments

-source

Specifies a clock source latency on a clock pin.

-rise

Specifies the edge for which this constraint will apply. If neither or both rise are passed, the same latency is applied to both edges.

-fall

Specifies the edge for which this constraint will apply. If neither or both rise are passed, the same latency is applied to both edges.

-invert

Specifies that the generated clock waveform is inverted with respect to the reference clock.

-late

Optional. Specifies that the latency is late bound on the latency. The appropriate bound is used to provide the most pessimistic timing scenario. However, if the value of "-late" is less than the value of "-early", optimistic timing takes place which could result in incorrect analysis. If neither or both "-early" and "-late" are provided, the same latency is used for both bounds, which results in the latency having no effect for single clock domain setup and hold checks.

-early

Optional. Specifies that the latency is early bound on the latency. The appropriate bound is used to provide the most pessimistic timing scenario. However, if the value of "-late" is less than the value of "-early", optimistic timing takes place which could result in incorrect analysis. If neither or both "-early" and "-late" are provided, the same latency is used for both bounds, which results in the latency having no effect for single clock domain setup and hold checks.

delay

Specifies the latency value for the constraint.

clock

Specifies the clock to which the constraint is applied. This clock must be constrained.

Supported Families

SmartFusion2, IGLOO2, RTG4, PolarFire, SmartFusion, IGLOO, ProASIC3, Fusion

Description

Clock source latency defines the delay between an external clock source and the definition pin of a clock within SmartTime. It behaves much like an input delay constraint. You can specify both an "early" delay and a"late" delay for this latency, providing an uncertainty which SmartTime propagates through its calculations. Rising and falling edges of the same clock can have different latencies. If only one value is provided for the clock source latency, it is taken as the exact latency value, for both rising and falling edges.

Exceptions

None

Examples

The following example sets an early clock source latency of 0.4 on the rising edge of main_clock. It also sets a clock source latency of 1.2, for both the early and late values of the falling edge of main_clock. The late value for the clock source latency for the falling edge of main_clock remains undefined.

```
set_clock_latency –source –rise –early 0.4 { main_clock  }
set_clock_latency –source –fall    1.2 { main_clock }
```

Microsemi Implementation Specifics

SDC accepts a list of clocks to -set_clock_latency. In Microsemi design implementation, only one clock pin can have its source latency specified per command.

See Also

Constraint Support by Family

Constraint Entry Table

SDC Syntax Conventions

set_clock_to_output

SDC command; defines the timing budget available inside the FPGA for an output relative to a clock.

```
set_clock_to_output delay_value -clock clock_ref [-max] [-min] [-clock_fall] output_list
```

Arguments

delay_value

Specifies the clock to output delay in nanoseconds. This time represents the amount of time available inside the FPGA between the active clock edge and the data change at the output port.

-clock *clock_ref*

Specifies the reference clock to which the specified clock to output is related. This is a mandatory argument.

-max

Specifies that delay_value refers to the maximum clock to output at the specified output. If you do not specify –max or –min options, the tool assumes maximum and minimum clock to output delays to be equal.

-min

Specifies that delay_value refers to the minimum clock to output at the specified output. If you do not specify –max or –min options, the tool assumes maximum and minimum clock to output delays to be equal.

-clock_fall

Specifies that the delay is relative to the falling edge of the reference clock. The default is the rising edge.

output_list

Provides a list of output ports in the current design to which delay_value is assigned. If you need to specify more than one object, enclose the objects in braces ({}).

Supported Families

SmartFusion2, IGLOO2, RTG4, PolarFire, SmartFusion, IGLOO, ProASIC3, Fusion

Description

The set_clock_to_output command specifies the clock to output maximum and minimum delays on output ports relative to a clock edge. This usually represents a combinational path delay from a register internal to the current design to the output port. For in/out (bidirectional) ports, you can specify the path delays for both input and output modes. The tool uses clock to output delays for paths ending at primary outputs.

A clock is a singleton that represents the name of a defined clock constraint. This can be an object accessor that will refer to one clock. For example:

```
[get_clocks {system_clk}]
[get_clocks {sys*_clk}]
```

Examples

The following example sets a maximum clock to output delay of 12 ns and a minimum clock to output delay of 6 ns for port data_out relative to the rising edge of CLK1:

```
set_clock_to_output 12 -clock [get_clocks CLK1] -max [get_ports data_out]
set_clock_to_output 6 -clock [get_clocks CLK1] -min [get_ports data_out]
```

See Also

Constraint Support by Family

Constraint Entry Table

SDC Syntax Conventions

set_clock_uncertainty

SDC command; defines the timing uncertainty between two clock waveforms or maximum skew.

```
set_clock_uncertainty uncertainty (-from | -rise_from | -fall_from) from_clock_list (-to | -
rise_to | -fall_to) to_clock_list [-setup | -hold]
```

Arguments

uncertainty

Specifies the time in nanoseconds that represents the amount of variation between two clock edges. The value must be a positive floating point number.

-from

Specifies that the clock-to-clock uncertainty applies to both rising and falling edges of the source clock list. You can specify only one of the -from, -rise_from, or -fall_from arguments for the constraint to be valid. This option is the default.

-rise_from

Specifies that the clock-to-clock uncertainty applies only to rising edges of the source clock list. You can specify only one of the -from, -rise_from, or -fall_from arguments for the constraint to be valid.

-fall_from

Specifies that the clock-to-clock uncertainty applies only to falling edges of the source clock list. You can specify only one of the -from, -rise_from, or -fall_from arguments for the constraint to be valid.

from_clock_list

Specifies the list of clock names as the uncertainty source.

-to

Specifies that the clock-to-clock uncertainty applies to both rising and falling edges of the destination clock list. You can specify only one of the -to, -rise_to , or -fall_to arguments for the constraint to be valid.

-rise_to

Specifies that the clock-to-clock uncertainty applies only to rising edges of the destination clock list. You can specify only one of the -to, -rise_to , or -fall_to arguments for the constraint to be valid.

-fall_to

Specifies that the clock-to-clock uncertainty applies only to falling edges of the destination clock list. You can specify only one of the -to, -rise_to , or -fall_to arguments for the constraint to be valid.

to_clock_list

Specifies the list of clock names as the uncertainty destination.

-setup

Specifies that the uncertainty applies only to setup checks. If you do not specify either option (-setup or -hold) or if you specify both options, the uncertainty applies to both setup and hold checks.

-hold

Specifies that the uncertainty applies only to hold checks. If you do not specify either option (-setup or -hold) or if you specify both options, the uncertainty applies to both setup and hold checks.

Supported Families

SmartFusion2, IGLOO2, RTG4, PolarFire, SmartFusion, IGLOO, ProASIC3, Fusion

Description

Clock uncertainty defines the timing between an two clock waveforms or maximum clock skew.

Both setup and hold checks must account for clock skew. However, for setup check, SmartTime looks for the smallest skew. This skew is computed by using the maximum insertion delay to the launching sequential component and the shortest insertion delay to the receiving component.

For hold check, SmartTime looks for the largest skew. This skew is computed by using the shortest insertion delay to the launching sequential component and the largest insertion delay to the receiving component. SmartTime makes this distinction automatically.

Exceptions

None

Examples

The following example defines two clocks and sets the uncertainty constraints, which analyzes the inter-clock domain between clk1 and clk2.

```
create_clock –period 10 clk1
create_generated_clock –name clk2 -source clk1 -multiply_by 2 sclk1
set_clock_uncertainty 0.4 -rise_from clk1 -rise_to clk2
```

Microsemi Implementation Specifics

- SDC accepts a list of clocks to -set_clock_uncertainty.

See Also

Constraint Support by Family

Constraint Entry Table

SDC Syntax Conventions

create_clock (SDC)

create_generated_clock (SDC)

remove_clock_uncertainty

set_disable_timing

SDC command; disables timing arcs within the specified cell and returns the ID of the created constraint if the command succeeded.

```
set_disable_timing [-from from_port] [-to to_port] cell_name
```

Arguments

-from *from_port*

Specifies the starting port.

-to *to_port*

Specifies the ending port.

cell_name

Specifies the name of the cell in which timing arcs will be disabled.

Supported Families

SmartFusion2, IGLOO2, RTG4, PolarFire, SmartFusion, IGLOO, ProASIC3, Fusion

Description

This command disables the timing arcs in the specified cell, and returns the ID of the created constraint if the command succeeded. The –from and –to arguments must either both be present or both omitted for the constraint to be valid.

Examples

The following example disables the arc between a2:A and a2:Y.

```
set_disable_timing -from port1 -to port2 cellname
```

This command ensures that the arc is disabled within a cell instead of between cells.

Microsemi Implementation Specifics

* None

See Also

Constraint Support by Family

Constraint Entry Table

SDC Syntax Conventions

set_external_check

SDC command; defines the external setup and hold delays for an input relative to a clock.

```
set_external_check delay_value -clock clock_ref [-setup] [-hold] [-clock_fall] input_list
```

Arguments

delay_value

Specifies the external setup or external hold delay in nanoseconds. This time represents the amount of time available inside the FPGA for the specified input after a clock edge.

-clock *clock_ref*

Specifies the reference clock to which the specified external check is related. This is a mandatory argument.

-setup

Specifies that delay_value refers to the setup check at the specified input. This is a mandatory argument if –hold is not used. You must specify either -setup or -hold option.

-clock_fall

Specifies that the delay is relative to the falling edge of the reference clock. The default is the rising edge.

input_list

Provides a list of input ports in the current design to which delay_value is assigned. If you need to specify more than one object, enclose the objects in braces ({}).

Supported Families

SmartFusion2, IGLOO2, RTG4, PolarFire, SmartFusion, IGLOO, ProASIC3, Fusion

Description

The set_external_check command specifies the external setup and hold times on input ports relative to a clock edge. This usually represents a combinational path delay from the input port to the clock pin of a register internal to the current design. For in/out (bidirectional) ports, you can specify the path delays for both input and output modes. The tool uses external setup and external hold times for paths starting at primary inputs.

A clock is a singleton that represents the name of a defined clock constraint. This can be an object accessor that will refer to one clock. For example:

```
[get_clocks {system_clk}]
[get_clocks {sys*_clk}]
```

Examples

The following example sets an external setup check of 12 ns and an external hold check of 6 ns for port data_in relative to the rising edge of CLK1:

```
set_external_check 12 -clock [get_clocks CLK1] -setup [get_ports data_in]
set_external_check 6 -clock [get_clocks CLK1] -hold [get_ports data_in]
```

See Also

Constraint Support by Family

Constraint Entry Table

SDC Syntax Conventions

set_false_path

SDC command; identifies paths that are considered false and excluded from the timing analysis.

```
set_false_path [-from from_list] [-through through_list] [-to to_list]
```

Arguments

-from *from_list*

Specifies a list of timing path starting points. A valid timing starting point is a clock, a primary input, an inout port, or a clock pin of a sequential cell.

-through *through_list*

Specifies a list of pins, ports, cells, or nets through which the disabled paths must pass.

-to to_list

Specifies a list of timing path ending points. A valid timing ending point is a clock, a primary output, an inout port, or a data pin of a sequential cell.

Supported Families

SmartFusion2, IGLOO2, RTG4, PolarFire, SmartFusion, IGLOO, ProASIC3, Fusion

Description

The set_false_path command identifies specific timing paths as being false. The false timing paths are paths that do not propagate logic level changes. This constraint removes timing requirements on these false paths so that they are not considered during the timing analysis. The path starting points are the input ports or register clock pins, and the path ending points are the register data pins or output ports. This constraint disables setup and hold checking for the specified paths.

The false path information always takes precedence over multiple cycle path information and overrides maximum delay constraints. If more than one object is specified within one -through option, the path can pass through any objects.

Examples

The following example specifies all paths from clock pins of the registers in clock domain clk1 to data pins of a specific register in clock domain clk2 as false paths:

```
set_false_path –from [get_clocks {clk1}] –to reg_2:D
```

The following example specifies all paths through the pin U0/U1:Y to be false:

```
set_false_path -through U0/U1:Y
```

Microsemi Implementation Specifics

SDC accepts multiple -through options in a single constraint to specify paths that traverse multiple points in the design. In Microsemi design implementation, only one –through option is accepted.

See Also

Constraint Support by Family

Constraint Entry Table

SDC Syntax Conventions

Set False Path Constraint

set_input_delay

SDC command; defines the arrival time of an input relative to a clock.

```
set_input_delay delay_value -clock clock_ref [-max] [-min] [-clock_fall] input_list
```

Arguments

`delay_value`

Specifies the arrival time in nanoseconds that represents the amount of time for which the signal is available at the specified input after a clock edge.

`-clock clock_ref`

Specifies the clock reference to which the specified input delay is related. This is a mandatory argument. If you do not specify -max or -min options, the tool assumes the maximum and minimum input delays to be equal.

`-max`

Specifies that delay_value refers to the longest path arriving at the specified input. If you do not specify -max or -min options, the tool assumes maximum and minimum input delays to be equal.

`-min`

Specifies that delay_value refers to the shortest path arriving at the specified input. If you do not specify -max or -min options, the tool assumes maximum and minimum input delays to be equal.

`-clock_fall`

Specifies that the delay is relative to the falling edge of the clock reference. The default is the rising edge.

`input_list`

Provides a list of input ports in the current design to which delay_value is assigned. If you need to specify more than one object, enclose the objects in braces ({}).

Supported Families

SmartFusion2, IGLOO2, RTG4, PolarFire, SmartFusion, IGLOO, ProASIC3, Fusion and IGLOOe, except ProASIC3 nano and ProASIC3L

Description

The set_input_delay command sets input path delays on input ports relative to a clock edge. This usually represents a combinational path delay from the clock pin of a register external to the current design. For in/out (bidirectional) ports, you can specify the path delays for both input and output modes. The tool adds input delay to path delay for paths starting at primary inputs.

A clock is a singleton that represents the name of a defined clock constraint. This can be:

- a single port name used as source for a clock constraint
- a single pin name used as source for a clock constraint; for instance reg1:CLK. This name can be hierarchical (for instance toplevel/block1/reg2:CLK)
- an object accessor that will refer to one clock: [get_clocks {clk}]

Examples

The following example sets an input delay of 1.2ns for port data1 relative to the rising edge of CLK1:

```
set_input_delay 1.2 -clock [get_clocks CLK1] [get_ports data1]
```

The following example sets a different maximum and minimum input delay for port IN1 relative to the falling edge of CLK2:

```
set_input_delay 1.0 -clock_fall -clock CLK2 -min {IN1}
set_input_delay 1.4 -clock_fall -clock CLK2 -max {IN1}
```

Microsemi Implementation Specifics

In SDC, the -clock is an optional argument that allows you to set input delay for combinational designs. Microsemi Implementation currently requires this argument.

See Also

Constraint Support by Family

Constraint Entry Table

SDC Syntax Conventions

Set Input Delay

set_max_delay (SDC)

SDC command; specifies the maximum delay for the timing paths.

```
set_max_delay delay_value [-from from_list] [-to to_list]
```

Arguments

delay_value

Specifies a floating point number in nanoseconds that represents the required maximum delay value for specified paths.

- If the path starting point is on a sequential device, the tool includes clock skew in the computed delay.
- If the path starting point has an input delay specified, the tool adds that delay value to the path delay.
- If the path ending point is on a sequential device, the tool includes clock skew and library setup time in the computed delay.
- If the ending point has an output delay specified, the tool adds that delay to the path delay.

-from *from_list*

Specifies a list of timing path starting points. A valid timing starting point is a clock, a primary input, an inout port, or a clock pin of a sequential cell.

-to *to_list*

Specifies a list of timing path ending points. A valid timing ending point is a clock, a primary output, an inout port, or a data pin of a sequential cell.

Supported Families

SmartFusion2, IGLOO2, RTG4, PolarFire, SmartFusion, IGLOO, ProASIC3, Fusion

Description

This command specifies the required maximum delay for timing paths in the current design. The path length for any startpoint in from_list to any endpoint in to_list must be less than delay_value.

The tool automatically derives the individual maximum delay targets from clock waveforms and port input or output delays. For more information, refer to the create_clock, set_input_delay, and set_output_delay commands.

The maximum delay constraint is a timing exception. This constraint overrides the default single cycle timing relationship for one or more timing paths. This constraint also overrides a multicycle path constraint.

Examples

The following example sets a maximum delay by constraining all paths from ff1a:CLK or ff1b:CLK to ff2e:D with a delay less than 5 ns:

```
set_max_delay 5 -from {ff1a:CLK ff1b:CLK} -to {ff2e:D}
```

The following example sets a maximum delay by constraining all paths to output ports whose names start by "out" with a delay less than 3.8 ns:

```
set_max_delay 3.8 -to [get_ports out*]
```

Microsemi Implementation Specifics

The –through option in the set_max_delay SDC command is not supported.

See Also

[Constraint Support by Family](#)

[Constraint Entry Table](#)

[SDC Syntax Conventions](#)

[Set Max Delay](#)

set_min_delay

SDC command; specifies the minimum delay for the timing paths.

```
set_min_delay delay_value [-from from_list] [-to to_list]
```

Arguments

delay_value

Specifies a floating point number in nanoseconds that represents the required minimum delay value for specified paths.

- If the path starting point is on a sequential device, the tool includes clock skew in the computed delay.
- If the path starting point has an input delay specified, the tool adds that delay value to the path delay.
- If the path ending point is on a sequential device, the tool includes clock skew and library setup time in the computed delay.
- If the ending point has an output delay specified, the tool adds that delay to the path delay.

-from *from_list*

Specifies a list of timing path starting points. A valid timing starting point is a clock, a primary input, an inout port, or a clock pin of a sequential cell.

-to *to_list*

Specifies a list of timing path ending points. A valid timing ending point is a clock, a primary output, an inout port, or a data pin of a sequential cell.

Supported Families

SmartFusion2, IGLOO2, RTG4, PolarFire, SmartFusion, IGLOO, ProASIC3, Fusion

Description

This command specifies the required minimum delay for timing paths in the current design. The path length for any startpoint in from_list to any endpoint in to_list must be less than delay_value.

The tool automatically derives the individual minimum delay targets from clock waveforms and port input or output delays. For more information, refer to the create_clock, set_input_delay, and set_output_delay commands.

The minimum delay constraint is a timing exception. This constraint overrides the default single cycle timing relationship for one or more timing paths. This constraint also overrides a multicycle path constraint.

Examples

The following example sets a minimum delay by constraining all paths from ff1a:CLK or ff1b:CLK to ff2e:D with a delay less than 5 ns:

```
set_min_delay 5 -from {ff1a:CLK ff1b:CLK} -to {ff2e:D}
```

The following example sets a minimum delay by constraining all paths to output ports whose names start by "out" with a delay less than 3.8 ns:

```
set_min_delay 3.8 -to [get_ports out*]
```

Microsemi Implementation Specifics

The –through option in the set_min_delay SDC command is not supported.

See Also

[Constraint Support by Family](#)

[Constraint Entry Table](#)

[SDC Syntax Conventions](#)

set_multicycle_path

SDC command; defines a path that takes multiple clock cycles.

```
set_multicycle_path ncycles [-setup] [-hold] [-from from_list] [-through through_list] [-to to_list]
```

Arguments

ncycles

Specifies an integer value that represents a number of cycles the data path must have for setup or hold check. The value is relative to the starting point or ending point clock, before data is required at the ending point.

-setup

Optional. Applies the cycle value for the setup check only. This option does not affect the hold check. The default hold check will be applied unless you have specified another set_multicycle_path command for the hold value.

-hold

Optional. Applies the cycle value for the hold check only. This option does not affect the setup check.

Note: If you do not specify "-setup" or "-hold", the cycle value is applied to the setup check and the default hold check is performed (*ncycles* -1).

-from *from_list*

Specifies a list of timing path starting points. A valid timing starting point is a clock, a primary input, an inout port, or a clock pin of a sequential cell.

-through *through_list*

Specifies a list of pins or ports through which the multiple cycle paths must pass.

-to *to_list*

Specifies a list of timing path ending points. A valid timing ending point is a clock, a primary output, an inout port, or a data pin of a sequential cell.

Supported Families

SmartFusion2, IGLOO2, RTG4, PolarFire, SmartFusion, IGLOO, ProASIC3, Fusion

Description

Setting multiple cycle paths constraint overrides the single cycle timing relationships between sequential elements by specifying the number of cycles that the data path must have for setup or hold checks. If you change the multiplier, it affects both the setup and hold checks.

False path information always takes precedence over multiple cycle path information. A specific maximum delay constraint overrides a general multiple cycle path constraint.

If you specify more than one object within one -through option, the path passes through any of the objects.

Examples

The following example sets all paths between reg1 and reg2 to 3 cycles for setup check. Hold check is measured at the previous edge of the clock at reg2.

```
set_multicycle_path 3 -from [get_pins {reg1}] -to [get_pins {reg2}]
```

The following example specifies that four cycles are needed for setup check on all paths starting at the registers in the clock domain ck1. Hold check is further specified with two cycles instead of the three cycles that would have been applied otherwise.

```
set_multicycle_path 4 -setup -from [get_clocks {ck1}]
set_multicycle_path 2 -hold -from [get_clocks {ck1}]
```

Microsemi Implementation Specifics

- SDC allows multiple priority management on the multiple cycle path constraint depending on the scope of the object accessors. In Microsemi design implementation, such priority management is not supported. All multiple cycle path constraints are handled with the same priority.

See Also

Constraint Support by Family

Constraint Entry Table

SDC Syntax Conventions

Set Multicycle Path

set_output_delay

SDC command; defines the output delay of an output relative to a clock.

```
set_output_delay delay_value -clock clock_ref [-max] [-min] [-clock_fall] output_list
```

Arguments

delay_value

Specifies the amount of time before a clock edge for which the signal is required. This represents a combinational path delay to a register outside the current design plus the library setup time (for maximum output delay) or hold time (for minimum output delay).

-clock clock_ref

Specifies the clock reference to which the specified output delay is related. This is a mandatory argument. If you do not specify -max or -min options, the tool assumes the maximum and minimum input delays to be equal.

-max

Specifies that delay_value refers to the longest path from the specified output. If you do not specify -max or -min options, the tool assumes the maximum and minimum output delays to be equal.

-min

Specifies that delay_value refers to the shortest path from the specified output. If you do not specify -max or -min options, the tool assumes the maximum and minimum output delays to be equal.

-clock_fall

Specifies that the delay is relative to the falling edge of the clock reference. The default is the rising edge.

output_list

Provides a list of output ports in the current design to which delay_value is assigned. If you need to specify more than one object, enclose the objects in braces ({}).

Supported Families

SmartFusion2, IGLOO2, RTG4, PolarFire, SmartFusion, IGLOO, ProASIC3, Fusion

Description

The set_output_delay command sets output path delays on output ports relative to a clock edge. Output ports have no output delay unless you specify it. For in/out (bidirectional) ports, you can specify the path delays for both input and output modes. The tool adds output delay to path delay for paths ending at primary outputs.

Examples

The following example sets an output delay of 1.2ns for port OUT1 relative to the rising edge of CLK1:

```
set_output_delay 1.2 -clock [get_clocks CLK1] [get_ports OUT1]
```

The following example sets a different maximum and minimum output delay for port OUT1 relative to the falling edge of CLK2:

```
set_output_delay 1.0 -clock_fall -clock CLK2 -min {OUT1}
set_output_delay 1.4 -clock_fall -clock CLK2 -max {OUT1}
```

Microsemi Implementation Specifics

- In SDC, the -clock is an optional argument that allows you to set the output delay for combinational designs. Microsemi Implementation currently requires this option.

See Also

[Constraint Support by Family](#)

[Constraint Entry Table](#)

[SDC Syntax Conventions](#)

[Set Output Delay](#)

Design Object Access Commands

Design object access commands are SDC commands. Most SDC constraint commands require one of these commands as command arguments.

Microsemi software supports the following SDC access commands:

Design Object	Access Command
Cell	get_cells
Clock	get_clocks
Net	get_nets
Port	get_ports
Pin	get_pins
Input ports	all_inputs
Output ports	all_outputs
Registers	all_registers

See Also

About SDC Files

all_inputs

Design object access command; returns all the input or inout ports of the design.

```
all_inputs
```

Arguments

- None

Supported Families

SmartFusion2, IGLOO2, RTG4, PolarFire, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

- None

Example

```
set_max_delay -from [all_inputs] -to [get_clocks ck1]
```

Microsemi Implementation Specifics

- None

See Also

Constraint Support by Family

Constraint Entry Table

SDC Syntax Conventions

all_outputs

Design object access command; returns all the output or inout ports of the design.

```
all_outputs
```

Arguments

- None

Supported Families

SmartFusion2, IGLOO2, RTG4, PolarFire, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

- None

Example

```
set_max_delay -from [all_inputs] -to [all_outputs]
```

Microsemi Implementation Specifics

None

See Also

Constraint Support by Family

Constraint Entry Table

SDC Syntax Conventions

all_registers

Design object access command; returns either a collection of register cells or register pins, whichever you specify.

```
all_registers [-clock clock_name] [-cells] [-data_pins ]
  [-clock_pins] [-async_pins] [-output_pins]
```

Arguments

-clock *clock_name*

Creates a collection of register cells or register pins in the specified clock domain.

-cells

Creates a collection of register cells. This is the default. This option cannot be used in combination with any other option.

-data_pins

Creates a collection of register data pins.

-clock_pins

Creates a collection of register clock pins.

-async_pins

Creates a collection of register asynchronous pins.

-output_pins

Creates a collection of register output pins.

Supported Families

SmartFusion2, IGLOO2, RTG4, PolarFire, SmartFusion, IGLOO, ProASIC3, Fusion

Description

This command creates either a collection of register cells (default) or register pins, whichever is specified. If you do not specify an option, this command creates a collection of register cells.

Exceptions

- None

Examples

```
set_max_delay 2 -from [all_registers] -to [get_ports {out}]
set_max_delay 3 -to [all_registers -async_pins]
set_false_path -from [all_registers -clock clk150]
set_multicycle_path -to [all_registers -clock c* -data_pins
 -clock_pins]
```

Microsemi Implementation Specifics

- None

See Also

Constraint Support by Family

Constraint Entry Table

SDC Syntax Conventions

get_cells

Design object access command; returns the cells (instances) specified by the pattern argument.

```
get_cells pattern
```

Arguments

pattern

Specifies the pattern to match the instances to return. For example, "get_cells U18*" returns all instances starting with the characters "U18", where "*" is a wildcard that represents any character string.

Supported Families

SmartFusion2, IGLOO2, RTG4, PolarFire, SmartFusion, IGLOO, ProASIC3, Fusion

Description

This command returns a collection of instances matching the pattern you specify. You can only use this command as part of a –from, -to, or –through argument for the following constraint exceptions: set_max delay, set_multicycle_path, and set_false_path design constraints.

Exceptions

None

Examples

```
set_max_delay 2 -from [get_cells {reg*}] -to [get_ports {out}]
set_false_path -through [get_cells {Rblock/muxA}]
```

Microsemi Implementation Specifics

- None

See Also

Constraint Support by Family

Constraint Entry Table

SDC Syntax Conventions

get_clocks

Design object access command; returns the specified clock.

```
get_clocks pattern
```

Arguments

pattern

Specifies the pattern to match to the SmartTime on which a clock constraint has been set.

Supported Families

SmartFusion2, IGLOO2, RTG4, PolarFire, SmartFusion, IGLOO, ProASIC3, Fusion

Description

- If this command is used as a –from argument in maximum delay (set_max_path_delay), false path (set_false_path), and multicycle constraints (set_multicycle_path), the clock pins of all the registers related to this clock are used as path start points.
- If this command is used as a –to argument in maximum delay (set_max_path_delay), false path (set_false_path), and multicycle constraints (set_multicycle_path), the synchronous pins of all the registers related to this clock are used as path endpoints.

Exceptions

- None

Example

```
set_max_delay -from [get_ports datal] -to \
[get_clocks ck1]
```

Microsemi Implementation Specifics

None

See Also

Constraint Support by Family

Constraint Entry Table

SDC Syntax Conventions

get_pins

Design object access command; returns the specified pins.

```
get_pins pattern
```

Arguments

pattern

Specifies the pattern to match the pins.

Supported Families

SmartFusion2, IGLOO2, RTG4, PolarFire, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

```
create_clock -period 10 [get_pins clock_gen/reg2:Q]
```

Microsemi Implementation Specifics

- None

See Also

Constraint Support by Family

Constraint Entry Table

SDC Syntax Conventions

get_nets

Design object access command; returns the named nets specified by the pattern argument.

```
get_nets pattern
```

Arguments

pattern

Specifies the pattern to match the names of the nets to return. For example, "get_nets N_255*" returns all nets starting with the characters "N_255", where "*" is a wildcard that represents any character string.

Supported Families

SmartFusion2, IGLOO2, RTG4, PolarFire, SmartFusion, IGLOO, ProASIC3, Fusion

Description

This command returns a collection of nets matching the pattern you specify. You can only use this command as source objects in create clock (create_clock) or create generated clock (create_generated_clock) constraints and as -through arguments in set false path (set_false_path), set minimum delay (set_min_delay), set maximum delay (set_max_delay), and set multicycle path (set_multicycle_path) constraints.

Exceptions

None

Examples

```
set_max_delay 2 -from [get_ports RDATA1] -through [get_nets {net_chkp1 net_chkqi}]
set_false_path -through [get_nets {Tblk/rm/n*}]
create_clcok -name mainCLK -per 2.5 [get_nets {cknet}]
```

Microsemi Implementation Specifics

None

See Also

Constraint Support by Family

Constraint Entry Table

SDC Syntax Conventions

get_ports

Design object access command; returns the specified ports.

```
get_ports pattern
```

Argument

pattern

Specifies the pattern to match the ports. This is equivalent to the macros $in()[<pattern>] when used as –from argument and $out()[<pattern>] when used as –to argument or $ports()[<pattern>] when used as a –through argument.

Supported Families

SmartFusion2, IGLOO2, RTG4, PolarFire, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

```
create_clock -period 10[get_ports CK1]
```

Microsemi Implementation Specifics

None

See Also

Constraint Support by Family

Constraint Entry Table

SDC Syntax Conventions

About Physical Design Constraint (PDC) Files

A PDC file is a Tcl script file specifying physical constraints. Any constraint that you can enter can also be used in a PDC file.

Command	Action
assign_net_macros	Assigns the macros connected to a net to a specified defined region
assign_region	Assigns macros to a pre-specified region
define_region	Defines either a rectangular or rectilinear region
move_region	Moves a region to new coordinates
reserve	Reserves the named pins in the current device package
set_io	Sets the attributes of an I/O
set_iobank	Specifies the I/O bank's technology and sets the VREF pins for the specified banks
set_location	Places a given logic instance at a particular location
set_preserve	Preserves instances before compile so that instances are not combined

Note: PDC commands are case sensitive. However, their arguments are not.

See Also

Constraint Entry

PDC Syntax Conventions

PDC Naming Conventions

Importing Constraint Files

PDC Syntax Conventions

The following table shows the typographical conventions that are used for the PDC command syntax.

Syntax Notation	Description
command -argument	Commands and arguments appear in Courier New typeface.
variable	Variables appear in blue, italic Courier New typeface. You must substitute an appropriate value for the variable.
[-argument *value*] [*variable*]+	Optional arguments begin and end with a square bracket with one exception: if the square bracket is followed by a plus sign (+), then users must specify at least one argument. The plus sign (+) indicates that items within the square brackets can be repeated. Do not enter the plus sign character.

Note: PDC commands are case sensitive. However, their arguments are not.

Examples

Syntax for the assign_local_clock command followed by a sample command:

```
assign_local_clock -type value -net netname [LocalClock_region ]+
```

```
        assign_local_clock -type hclk -net  reset_n tile1a tile2a
```

Syntax for the set_io command followed by a sample command:

```
set_io portname [-iostd value][-register value][-out_drive value][-slew value][-res_pull
value][-out_load value][-pinname value][-fixed value][-in_delay value]
```

```
            set_io ADDOUT2 \
            -iostd PCI \
            -register yes \
            -out_drive 16 \
            -slew high \
            -out_load 10 \
            -pinname T21 \
            -fixed yes
```

Wildcard Characters

You can use the following wildcard characters in names used in PDC commands:

Wildcard	What It Does
\	Interprets the next character literally
?	Matches any single character
*	Matches any string

Note: The matching function requires that you add a slash (\) before each slash in the port, instance, or net name when using wildcards in a PDC command and when using wildcards in the Find feature of the MultiView Navigator. For example, if you have an instance named "A/B12" in the netlist, and you enter that name as "A\\/B*" in a PDC command, you will not be able to find it. In this case, you must specify the name as A\\\/B*.

Special Characters ([ ], { }, and \)

Sometimes square brackets are part of the command syntax. In these cases, you must either enclose the open and closed square brackets characters with curly brackets or precede the open and closed square brackets characters with a backslash (\). If you do not, you will get an error message.

For example:

```
set_iobank {mem_data_in[57]} -fixed no 7 2
or
set_iobank mem_data_in\[57\] -fixed no 7 2
```

Entering Arguments on Separate Lines

To enter an argument on a separate line, you must enter a backslash (\) character at the end of the preceding line of the command as shown in the following example:

```
set_io ADDOUT2 \
-iostd PCI \
-register Yes \
-out_drive 16 \
-slew High \
-out_load 10 \
-pinname T21 \
-fixed yes
```

See Also

About PDC Files

PDC Naming Conventions

PDC Naming Conventions

Note: The names of ports, instances, and nets in an imported netlist are sometimes referred to as their original names.

Rules for Displaying Original Names

Port names appear exactly as they are defined in a netlist. For example, a port named A/B appears as A/B in ChipPlanner, PinEditor, and I/O Attribute Editor in MultiView Navigator.

Instances and nets display the original names plus an escape character (\) before each backslash (/) and each slash (\) that is not a hierarchy separator. For example, the instance named `A/\B` is displayed as `A\/\\B`.

Which Name Do I Use in PDC Commands?

The names of ports, instances, and nets in a netlist displayed in MultiView Navigator (MVN) for SmartFusion, IGLOO, ProASIC3 and Fusion devices are names taken directly from the imported netlist.

Using PDC Commands

When writing PDC commands, follow these rules:

- Always use the macro name as it appears in the netlist. (See "Merged elements" in this topic for exceptions.)
- Names from a netlist: For port names, use the names exactly as they appear in the netlist. For instance and net names, add an escape character (\) before each backslash (\) and each slash (/) that is not a hierarchy separator.
- Names from MVN and compile report: Use names as they appear in MultiView Navigator or the compile report.
- For wildcard names, always add an extra backslash (\) before each backslash.
- Always apply the PDC syntax conventions to any name in a PDC command.

The following table provides examples of names as they appear in an imported netlist and the names as they should appear in a PDC file:

Type of name and its location	Name in the imported netlist	Name to use in PDC file
Port name in netlist	A/:B1	A/:B1
Port name in MVN	A/:B1	A/:B1
Instance name in a netlist	A/:B1 A$(1)	A\\/:B1 A$(1)
Instance name in the netlist but using a wildcard character in a PDC file	A/:B1	A\\\\/:B*
Instance name in MVN or a compile report	A\/:B1	A\\/:B1
Net name in a netlist	Net1/:net1	Net1\\/:net1
Net name in MVN or a compile report	Net1\/:net1	Net1\\/:net1

When exporting PDC commands, the software always exports names using the PDC rules described in this topic.

Case Sensitivity When Importing PDC Files

The following table shows the case sensitivity in the PDC file based on the source netlist.

File Type	Case Sensitivity
Verilog	Names in the netlist are case sensitive.
Edif	Names in the netlist are always case sensitive because we use the Rename clause, which is case sensitive.
Vhdl	Names in the netlist are not case sensitive unless those names appear between slashes (\).

For example, in VHDL, capital "A" and lowercase "a" are the same name, but \A\ and \a\ are two different names. However, in a Verilog netlist, an instance named "A10" will fail if spelled as "a10" in the set_location command:

```
set_location A10 (This command will succeed.)
set_location a10 (This command will fail.)
```

See Also

[About PDC Files](#)
[PDC Syntax Conventions](#)

assign_net_macros

PDC command; assigns to a user-defined region all the macros that are connected to a net.

```
assign_net_macros -region_name region_name -net_name <net_name> -include_driver <true |
false>
```

Arguments

`-region_name`

Specifies the name of the region to which you are assigning macros. The region must exist before you use this command. See define_region (rectangular) or define_region (rectilinear). Because the define_region command returns a region object, you can write a simple command such as assign_net_macros [define_region]+ [net]+

`-net_name`

You must specify at least one net name. Net names are AFL-level (flattened netlist) names. These names match your netlist names most of the time. When they do not, you must export AFL and use the AFL names. Net names are case insensitive. Hierarchical net names from ADL are not allowed. You can use the following wildcard characters in net names:

Wilcard	What It Does
\	Interprets the next character as a non-special character
?	Matches any single character
*	Matches any string

`-include_driver`

Specifies whether to add the driver of the net(s) to the region. You can enter one of the following values:

Value	Description
true	Include the driver in the list of macros assigned to the region (default) .
false	Do not assign the driver to the region.

Notes:

- Placed macros (not connected to the net) that are inside the area occupied by the net region are automatically unplaced.
- Net region constraints are internally converted into constraints on macros. PDC export results as a series of assign_region <region_name> macro1 statements for all the connected macros.
- If the region does not have enough space for all of the macros, or if the region constraint is impossible, the constraint is rejected and a warning message appears in the Log window.
- For overlapping regions, the intersection must be at least as big as the overlapping macro count.
- If a macro on the net cannot legally be placed in the region, it is not placed and a warning message appears in the Log window.
- Net region constraints may result in a single macro being assigned to multiple regions. These net region constraints result in constraining the macro to the intersection of all the regions affected by the constraint.

Example

```
assign_net_macros -region_name UserRegion1 -net_name Y -include_driver false
```

assign_region

PDC command; constrains a set of macros to a specified region.

```
assign_region -region_name region_name -inst_name macro_name+
```

Arguments

region_name

Specifies the region to which the macros are assigned. The macros are constrained to this region. Because the define_region command returns a region object, you can write a simpler command such as assign_region [define_region]+ [macro_name]+.

inst_name

Specifies the macro(s) to assign to the region. You must specify at least one macro name. You can use the following wildcard characters in macro names:

Wildcard	What It Does
\	Interprets the next character as a non-special character
?	Matches any single character
*	Matches any string

The region must be created before you can assign macros to it. If the region creation PDC command and the macro assignment command are in different PDC files, the order of the PDC files is important.

You can assign only hard macros or their instances to a region. You cannot assign a group name. A hard macro is a logic cell consisting of one or more silicon modules with locked relative placement.

The macro name must be a name with full hierarchical path.

Notes:

- The region must be created before you can assign macros to it. If the region creation PDC command and the macro assignment command are in different PDC files, the order of the PDC files is important.
- You can assign only hard macros or their instances to a region. You cannot assign a group name. A hard macro is a logic cell consisting of one or more silicon modules with locked relative placement.
- The macro name must be a name with full hierarchical path.

Examples

In the following example, two macros are assigned to a region:

```
assign_region -region_name UserRegion1 -inst_name "test_0/AND2_0 test_0/AND2_1"
```

In the following example, all macros whose names have the prefix des01/Counter_1 (or all macros whose names match the expression des01/Counter_1/*) are assigned to a region:

```
assign_region -region_name User_region2 -inst_name des01/Counter_1/*
```

See Also

set_location

define_region

PDC command; defines either a rectangular region or a rectilinear region.

```
define_region -region_name <region_name> -type <inclusive|exclusive|empty> -x1 <integer> -y1
<integer> -x2 <integer> -y2 <integer> [-color <integer>] [-route <true|false>]
```

Note: The -color and -route parameters are optional.

Arguments

-region_name *region_name*

Specifies the region name. The name must be unique. Do not use reserved names such as "bank0" and "bank<N>" for region names. If the region cannot be created, the name is empty. A default name is generated if a name is not specified in this argument.

-type <*inclusive|exclusive|empty*>

Specifies the region type. The default is inclusive. The following table shows the acceptable values for this argument:
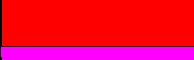
Region Type Value	Description
Empty	Empty regions cannot contain macros.
Exclusive	Only contains macros assigned to the region.
Inclusive	Can contain macros both assigned and unassigned to the region.

-x1 -y1 -x2 -y2

Specifies the series of coordinate pairs that constitute the region. These rectangles may or may not overlap. They are given as x1 y1 x2 y2 (where x1, y1 is the lower left and x2 y2 is the upper right corner in row/column coordinates). You must specify at least one set of coordinates.

-color *value*

Specifies the color of the region. The following table shows the recommended values for this argument:

Color	Decimal Value
	16776960
	65280
	16711680
	16760960
	255
	16711935
	65535
	33023
	8421631
	9568200

Color	Decimal Value
	8323199
	12632256

`-route value`

Specifies whether to direct the routing of all nets internal to a region to be constrained within that region. A net is internal to a region if its source and destination pins are assigned to the region. You can enter one of the following values:

Constrain Routing Value	Description
true	Constrain the routing of nets within the region as well as the placement.
false	Do not constrain the routing of nets within the region. Only constrain the placement. This is the default value.

Note: Local clocks and global clocks are excluded from the -route option. Also, interface nets are excluded from the –route option because they cross region boundaries.

An empty routing region is an empty placement region. If -route is "yes", no routing is allowed inside the empty region. However, local clocks and globals can cross empty regions.

An exclusive routing region is an exclusive placement region (rectilinear area with assigned macros) along with the following additional constraints:

- For all nets internal to the region (the source and all destinations belong to the region), routing must be inside the region (that is, such nets cannot be assigned any routing resource which is outside the region or crosses the region boundaries).

- Nets without pins inside the region cannot be assigned any routing resource which is inside the region or crosses any region boundaries.

An inclusive routing region is an inclusive placement region (rectilinear area with assigned macros) along with the following additional constraints:

- For all nets internal to the region (the source and all destinations belong to the region), routing must be inside the region (that is, such nets cannot be assigned any routing resource which is outside the region or crosses the region boundaries).

- Nets not internal to the region can be assigned routing resources within the region.

Description

Unlocked macros in empty or exclusive regions are unassigned from that region. You cannot create empty regions in areas that contain locked macros.

Use inclusive or exclusive region constraints if you intend to assign logic to a region. An inclusive region constraint with no macros assigned to it has no effect. An exclusive region constraint with no macros assigned to it is equivalent to an empty region.

Note: If macros assigned to a region exceed the area's capacity, the region's Properties Window displays the overbooked resources (over 100 percent resource utilization) in red.

Examples

The following example defines an empty rectangular region called UserRegion1 with lower-left co-ordinates (100,46) and upper-right co-ordinates (102,50).

```
define_region –region_name UserRegion1 -type empty -x1 100 –y1 46 –x2 102 –y2 50
```

The following example defines an inclusive rectilinear region with the name UserRegion2. This region contains two rectangular areas, one with lower-left co-ordinates (12,39) and upper-right co-ordinates (23,41) and another rectangle with lower-left co-ordinates (12,33) and upper-right co-ordinates (23,35).

```
define_region –region_name UserRegion2 -type exclusive -x1 12 –y1 39 –x2 23 –y2 41 –x1 12
–y1 33\
–x2 23 –y2 35
```

The following examples define three regions with three different colors:

```
define_region –region_name UserRegion0 -color 128 -x1 50 –y1 19 –x2 60 –y2 25
```

```
define_region –region_name UserRegion1 -color 16711935 -x1 11 –y1 2 –x2 55 –y2 29
```

```
define_region –region_name UserRegion2 -color 8388736 -x1 61 –y1 6 –x2 69 –y2 19
```

See Also

"assign_region" on page 107

move_region

PDC command; moves the named region to the coordinates specified.

```
move_region -region_name <region_name> -x1 <integer> -y1 <integer> -x2 <integer> -y2 <integer>
```

Arguments

-region_name <region_name>

Specifies the name of the region to move. This name must be unique.

-x1 -y1 -x2 -y2

Specifies the series of coordinate pairs representing the location in which to move the named region. These rectangles can overlap. They are given as x1 y1 x2 y2, where x1, y1 represents the lower-left corner of the rectangle and x2 y2 represents the upper-right corner. You must specify at least one set of coordinates.

Example

This example moves the region named UserRegion1 to a new region with lower-left co-ordinates (0,40) and upper-right co-ordinates (3,42):

```
move_region -region_name UserRegion1 -x1 0 -y1 40 -x2 3 -y2 42
```

See Also

"define_region" on page 108

reserve

PDC command; reserves the named pins in the current device package.

```
reserve -pin_name "list of package pins"
```

Arguments

-pin_name "list of package pins"

Specifies the package pin name(s) to reserve. You can reserve one or more pins.

Exceptions

None

Examples

```
reserve -pin_name "F2"
reserve -pin_name "F2 B4 B3"
reserve -pin_name "124 17"
```

set_io

PDC command; You can use the set_io command to assign an I/O technology, place, or lock the I/O at a given pin location. There are two I/O types available for PolarFire: GPIO and HSIO. Each I/O type supports different I/O standards.

I/O Type	Supported I/O Standards
HSIO	LVCMOS12, LVCMOS15, LVCMOS18, SSTL18I, HSUL18I, SSTL15I, HSTL15I, SSTL135I, HSTL135I, HSUL12I, HSTL12I, POD12I, SLVS25, HCSL25, SLVSE
GPIO	PCI, LVTTL, LVCMOS33, LVCMOS25, LVCMOS12, LVCMOS15, LVCMOS18, SSTL25I, SSTL18I, HSUL18I, SSTL15I HSTL15I, PPDS25, SLVS25, HCSL25, SLVSE, BUSLVDSE, MLVDSE, BUSLVDS, MLVDS, LVPECL, LVDS, RSDS, MINILVDS, LVDS33

```
set_io -port_name <port_name>\
[-pin_name <package_pin>]\
[-fixed <true|false>]\
[-io_std <io_std_values>]\
[-out_load <value>]\
[-res_pull <value>]\
[-lock_down <value>]\
[-ff_io_state <value>]\
[-clamp_diode <value>]\
[-schmitt_trigger <value>]\
[-slew <value>]\
[-vicm_range <value>]\
[-ODT <value>]\
[-ODT_VALUE]\
[-out_drive <value>]\
[-impedance <value>]
```

Arguments

-port_name

Specifies the portname of the I/O macro.

-pin_name <package_pin>

Specifies the package pin name(s) to place the I/O on it.

-io_std <value>

Sets the I/O standard for this macro. If the voltage standard used with the I/O is not compatible with other I/Os in the I/O bank, then assigning an I/O standard to a port will invalidate its location and automatically unassign the I/O.

The following table shows a list of supported I/O standards.

Some I/O standards support only either single I/O or differential I/Os while others support both Single and Differential I/Os. The table below lists the different I/O standards and whether they support single I/O, differential I/O, or both.

IO_STD Value	Single	Differential
LVTTL	YES	NO

IO_STD Value	Single	Differential
LVSTL11I	YES	YES
LVSTL11II	YES	YES
LVCMOS33	YES	NO
LVCMOS25	YES	NO
LVCMOS18	YES	NO
LVCMOS15	YES	NO
LVCMOS12	YES	NO
PCI	YES	NO
POD12I	YES	YES
POD12II	YES	YES
PPDS33	NO	YES
PPDS25	NO	YES
PPDS18	NO	YES
SLVS33	NO	YES
SLVS25	NO	YES
SLVS18	NO	YES
HCSL33	NO	YES
HCSL25	NO	YES
HCSL18	NO	YES
SLVSE	NO	YES
SLVSE15	NO	YES
BUSLVDSE	NO	YES
BUSLVDSE25	NO	YES
MLVDSE	NO	YES
MLVDSE25	NO	YES
LVDS	NO	YES
LVDS33	NO	YES
LVDS25	NO	YES

IO_STD Value	Single	Differential
LVDS18	NO	YES
BUSLVDS	NO	YES
MLVDS	NO	YES
MIPI33	NO	YES
MIPI12	NO	YES
MINILVDS	NO	YES
MINILVDS33	NO	YES
MINILVDS25	NO	YES
MINILVDS18	NO	YES
RSDS	NO	YES
RSDS33	NO	YES
RSDS25	NO	YES
RSDS18	NO	YES
LVPECL (only for inputs)	NO	YES
LVPECL33	NO	YES
LVPECL25	NO	YES
LVPECLE33	NO	YES
HSTL15I	YES	YES
HSTL15II	YES	YES
HSTL135I	YES	YES
HSTL135II	YES	YES
HSTLI2I	YES	YES
HSTL12II	YES	YES
SSTL18I	YES	YES
SSTL18II	YES	NO
SSTL15I	YES	YES
SSTL15II	YES	YES

IO_STD Value	Single	Differential
SSTL135I	YES	YES
SSTL135II	YES	YES
SSTL125I	YES	YES
SSTL125II	YES	YES
HSUL18I	YES	YES
HSUL18II	YES	YES
HSUL12I	YES	YES
HSUL12II	YES	YES
SUBLVDS33	NO	YES
SUBLVDS25	NO	YES
SUBLVDS18	NO	YES

```
-fixed <value>
```
Specifies if the location of this port is fixed (i.e., locked). Locked ports are not moved during layout. The default value is true. You can enter one of the following values:

Value	Description
true	The location of this port is locked.
false	The location of this port is unlocked.

Examples

```
set_io -port_name IO_in\[2\]
-io_std LVCMOS25 \
-fixed true\
```

I/O Directions Not Supported

The following table lists I/O directions that are **not** supported for the I/O standards shown in the table.

I/O Direction	IO_STD Value
Input	SLVSE15, MLVDSE25, BUSLVDSE25, MIPIE33, LVPECLE33
Output	SLVS33, HCSL33, LVPECL33, LVPECL25, MIPI12, LVDS18, RSDS18, MINILVDS18, SUBLVDS18, PPDS18, SLVS18, HCSL18, HSTL12I
Inout	LVDS33, LVDS18, LVDS25, RSDS18, RSDS33, RSDS25, MINILVDS18, MINILVDS33, MINILVDS25, SUBLVDS18, SUBLVDS33, SUBLVDS25, PPDS18, PPDS33, PPDS25, SLVS33, SLVS25, HCSL33, HCSL25, LVPECL33, LVPECL25, MIPI12, SLVS18, HCSL18, HSTL12I

```
-OUT_LOAD <value>
```

Sets the output load (in pF) of output signals.

The default is 5.

Direction: Output

```
-RES_PULL <value>
```

Allows you to include a weak resistor for either pull-up or pull-down of the input buffer. Not all I/O standards have a selectable resistor pull option.

The following table shows the acceptable values for the -res_pull attribute:

I/O Standard	Value	Description
LVCMOS25, LVCMOS33, LVTTL, PCI, LVCMOS18, LVCMOS15, LVCMOS12	Up	Includes a weak resistor for pull-up of the input buffer
	Down	Includes a weak resistor for pull-down of the input buffer
	Hold	Holds the last value
	None	Does not include a weak resistor

For all other I/O standards, the value is None.

The default is None.

Direction: Inout

```
-LOCK_DOWN <value>
```

Security feature that locks down the I/Os if tampering is detected.

Values are OFF, ON. The default is OFF.

Direction: Inout

```
-FF_IO_STATE <value>
```

Preserves the previous state of the I/O. You can override this default using the FF_IO_STATE attribute. When you set this attribute to LAST_VALUE, the I/O remains in the same state in which it was functioning before the device went into Flash*Freeze mode. Possible values are shown in the table below.

Value	Description
LAST_VALUE	Preserves the previous state of the I/O.
LAST_VALUE_WP	The last value with weak pullup.

The default is LAST_VALUE.

Direction: Inout

```
-CLAMP_DIODE <value>
```

Specifies whether to add a power clamp diode to the I/O buffer. This attribute option is available to all I/O buffers with I/O technology set to LVTTL. A clamp diode provides circuit protection from voltage spikes, surges, electrostatic discharge and other over-voltage conditions.

Values are OFF, ON.

The following table lists the values for GPIO standards.

I/O Standard	Values
LVCMOS12, LVCMOS15, LVCMOS18, SSTL18I, SSTL18II, SSTL15I, SSTL15II, HSTL15I, HSTL15II, LVTTL, LVCMOS33, LVCMOS25, SSTL25I, SSTL25II	OFF, ON. The default is ON.
HSUL18I, HSUL18II, SLVSE15, PCI, SLVS33, HCSL33, MIPIE33, LVPECL33, LVPECL25, LVPECLE33, LVDS25, LVDS33, RSDS25, RSDS33, MINILVDS25, MINILVDS33, SUBLVDS25, SUBLVDS33, PPDS25, PPDS33, MLVDSE25, BUSLVDSE25	ON
MIPI12	OFF

For HSIO standards, the value is always ON.

Direction: Inout

`-SCHMITT_TRIGGER <`*`value`*`>`

Specifies whether this I/O has an input schmitt trigger. The schmitt trigger introduces hysteresis on the I/O input. This allows very slow moving or noisy input signals to be used with the part without false or multiple I/O transitions taking place in the I/O.

For the following I/O standards, the values are OFF, ON. The default is OFF.

I/O Standard	Values
GPIO	
LVCMOS25, LVCMOS33, LVTTL, PCI	OFF, ON
HSIO	
LVCMOS18, LVCMOS15	OFF, ON

For all other I/O standards, the value is OFF.

Direction: Input

`-SLEW <`*`value`*`>`

Sets the output slew rate. Slew control affects only the falling edges for some families. Slew control affects both rising and falling edges. Not all I/O standards have a selectable slew. Whether you can use the slew attribute depends on which I/O standard you have specified for this command.

The following I/O standards have values OFF, ON. The default is OFF.

I/O Standard	Values
LVCMOS25, LVCMOS33, LVTTL, PCI	OFF, ON

For all other I/O standards, the value is OFF.

Direction: Output

`-VICM_RANGE`

Sets the VCM input range.

For SLVSE15, the value is MID.

Values for all other I/O standards are MID, LOW. The default is MID.

Direction: Input

`-ODT`

On-die termination (ODT) is the technology where the termination resistor for impedance matching in transmission lines is located inside a semiconductor chip instead of on a printed circuit board.

Values are ON, OFF. The default is OFF.

Direction: Input

`-ODT_VALUE`

Sets the ODT value (in Ohms) for On Die Termination.

Values vary depending on the I/O standard. The table below lists acceptable values.

The table below lists acceptable values.

I/O Standard	Values
LVCMOS12, LVCMOS15, LVCMOS18, HSUL12	60, 120, 240. The default is 60.
LVCMOS25	120, 240. The default is 120.
SSTL15I, SSTL15II, POD12I, POD12II, SSTL135I, SSTL135II	20, 30, 40, 60, 120. The default is 60.
SSTL18I, SSTL18II	50, 75, 150. The default is 50.
LVSTL11I, LVSTL11II	40, 48, 60, 80, 120. The default is 60.
BUSLVDSE25, LVDS33, LVDS25, LVPECL33, LVPECL25, MINILVDS33, MINILVDS25, MLVDSE25, RSDS33, RSDS25, SLVSE15, SUBLVDS33, SUBLVDS25	100
HSTL15I, HSTL15II, HSUL18I, HSUL18II	50

Direction: Input

`-OUT_DRIVE <value>`

Sets the strength of the output buffer to 1.5, 2, 3.5, 4, 6, 8, 10, 12, 16, or 20 in mA, weakest to strongest. The list of I/O standards for which you can change the output drive and the list of values you can assign for each I/O standard is family-specific. Not all I/O standards have a selectable output drive strength. Also, each I/O standard has a different range of legal output drive strength values. The values you can choose from depend on which I/O standard you have specified for this command. The table below lists acceptable values.

I/O Standard	Values
LVCMOS12, LVCMOS15	2, 4, 6, 8, 10. The default is 8.
LVCMOS18	2, 4, 6, 8, 10, 12. The default is 8.
LVCMOS25	2, 4, 6, 8, 12, 16. The default is 8.
LVCMOS33, LVTTL	2, 4, 8, 12, 16, 20. The default is 8.
LVDS25, LVDS33, MINILVDS25, MINILVDS33	3, 3.5, 4, 6. The default is 6

I/O Standard	Values
PPDS25, PPDS33, RSDS25, RSDS33	1.5, 2, 3. The default is 3.
SUBLVDS25, SUBLVDS33	1, 1.5, 2. The default is 2.
BUSLVDSE25, MLVDSE25	16
MIPIE33, SLVSE15	8
PCI	20

Direction: Output

```
-IMPEDANCE <value>
```

Sets the Impedance value (in Ohms).

Values vary depending on the I/O standard.

I/O Standard	Values
HSTL135I, HSTL15I	34, 40, 50, 60. The default is 60.
HSTL135II, HSTL15II, HSUL18II,	22, 25, 27, 30. The default is 25.
HSUL12I	34, 40, 48, 60, 80, 120. The default is 40.
HSUL18I	34, 40, 55, 60. The default is 55.
LVSTL11I, LVSTL11II	40, 48, 60. The default is 48.
POD12II, SSTL135II, SSTL15II	27, 30, 34. The default is 34.
SSTL135I, SSTL15I	40, 48. The default is 40.
SSTL18I	40, 48, 60, 80. The default is 60.
SSTL18II	30, 34, 40, 48. The default is 40.
SSTL25I	48, 60, 80, 120. The default is 80.
SSTL25II	34, 40, 48, 60. The default is 48.

Direction: Output

set_iobank

PDC command; sets the input/output supply voltage (vcci) and the input reference voltage (vref) for the specified I/O bank.

All banks have a dedicated vref pin and you do not need to set any pin on these banks.

There are two types of I/O banks: General-Purpose IO (GPIO) and High-Speed IO (HSIO).

Each bank type supports a different set of IO Standards as listed in the table below.

I/O Bank Type	Supported I/O Standards
High-Speed IO (HSIO)	LVCMOS12, LVCMOS15, LVCMOS18, SSTL18I, SSTL18II, HSUL18I, HSUL18II, SSTL15I, SSTL15II, HSTL15I, HSTL15II, SSTL135I, SSTL135II, HSTL135I, HSTL135II, HSTL12I, HSUL12I, SLVSE15, POD12I, POD12II, LVSTL11I, LVSTL11II, SLVS18, HCSL18, LVDS18, RSDS18, MINILVDS18, SUBLVDS18, PPDS18
General-Purpose IO (GPIO)	LVTTL, LVCMOS33, PCI, LVCMOS12, LVCMOS15, LVCMOS18, LVCMOS25, SSTL25I, SSTL25II, SSTL18I, SSTL18II, HSUL18I, HSUL18II, SSTL15I, SSTL15II, HSTL15I, HSTL15II, SLVS33, HCSL33, MIPI12, MIPIE33, LVPECL33, LVPECL25, LVPECLE33, LVDS25, LVDS33, RSDS25, RSDS33, MINILVDS25, MINILVDS33, SUBLVDS25, SUBLVDS33, PPDS25, PPDS33, SLVSE15, MLVDSE25, BUSLVDSE25

```
set_iobank -bank_name <bank_name>\
[-vcci vcci_voltage]\
[-vref vref_voltage]\
[-fixed value]\
[-update_iostd value]\
```

Arguments

```
-bank_name <bank_name>
```

Specifies the name of the bank. I/O banks are numbered 0 through N (bank0, bank1,...bankN). The number of I/O banks varies with the device. Refer to the datasheet for your device to determine how many banks it has.

```
-vcci vcci_voltage
```

Sets the input/output supply voltage. You can enter one of the following values:

vcci voltage	Compatible Standards
3.3 V	LVTTL, LVCMOS33, PCI, LVDS33, LVPECL33, LVPECLE33, SLVS33, HCSL33, RSDS33, MINILVDS33, SUBLVDS33
2.5 V	LVCMOS25, SSTL25I,SSTL25II, LVPECL25, PPDS25, SLVS25, HCSL25, MLVDSE25, MINILVDS25, RSDS25, SUBLVDS25, LVDS25, MLVDSE25, BUSLVDSE25
1.8 V	LVCMOS18, SSTL18I, SSTL18II, HSUL18I, HSUL18II, SLVS18, HCSL18, LVDS18, RSDS18, MINILVDS18, SUBLVDS18, PPDS18
1.5 V	LVCMOS15, SSTL15I, SSTL15II, HSTL15I, HSTL15II, SLVSE15
1.35 V	HSTL135I, HSTL135II, SSTL135I, SSTL135II
1.2 V	LVCMOS12, HSUL12I, HSTL12I, POD12I, MIPI12
1.1V	LVSTL11I, LVSTL11II

```
-vref vref_voltage
```
Sets the input reference voltage. You can enter one of the following values:

vref voltage	Compatible Standards
1.25 V	SSTL25I
1.0 V	SSTL18I,HSUL18I
0.75 V	POD12I, HSTL15I, SSTL15I, HSUL12I, HSTL12I
0.67 V	SSTL135I, HSTL135I

`-fixed_value`

Specifies if the I/O technologies (vcci and vccr voltage) assigned to the bank are locked. You can enter one of the following values:

Value	Description
true	The technologies are locked.
false	The technologies are not locked.

`-update_iostd value`

Specifies if the I/O technologies (vcci and vccr voltage) assigned to the bank are locked. You can enter one of the following values:

Value	Description
true	If there are I/O's placed on the bank, we keep the placement and change the host to one which is compatible with this bank setting. Check the I/O Attributes to see the one used by the tool.
false	If there are I/O's placed and locked on the bank, the command will fail. If they are placed I/Os they will be unplaced.

Exceptions

Any pins assigned to the specified I/O bank that are incompatible with the default technology are unassigned.

Examples

The following example assigns 3.3 V to the input/output supply voltage (vcci) for I/O bank 0.

`set_iobank -bank_name bank0 -vcci 3.3`

set_location

PDC command; assigns the specified macro to a particular location on the chip.

```
set_location -inst_name <macro_inst_name> -fixed <true|false> -x <integer> -y <integer>
```

Arguments

`-inst_name`

Specifies the instance name of the macro in the netlist to assign to a particular location on the chip.

`-fixed <true|false>`

Sets whether the location of this instance is fixed (that is, locked). Locked instances are not moved during layout. The default is yes. The following table shows the acceptable values for this argument:

Value	Description
true	The location of this instance is locked.
false	The location of this instance is unlocked.

`-x -y`

The x and y coordinates specify where to place the macro on the chip. Use the Chip Planner tool to determine the x and y coordinates of the location.

Exceptions

None

Example

This example assigns and locks the macro with the name "mem_data_in\[57\]" at the location x=7, y=2:

```
set_location -inst_name mem_data_in\[57\] -fixed true -x 7 -y 2
```

set_preserve

This command sets a preserve property on instances before compile, so compile will preserve these instances and not combine them.

```
set_preserve -inst_name <instance_name>
```

Arguments

`-inst_name`

Specifies the full hierarchical name of the macro in the netlist to preserve.

Exceptions

You must put this command in a PDC constraint file and associate it with Place and Route.

Example

```
set_preserve -inst_name "test1/AND2_0"
```

Placement Rules for PLLs and DLLs

This topic outlines the placement rules for PLL and DLL instances. You must place PLL and DLL instances using the set_location command.

The following error messages indicate non-compliance with placement rules for PLL and DLL:

PRPF_006: PLL/DLL <inst name> must be placed before running Place & Route.

All PLL and DLL instances must be placed before running Place and Route.

PRPF_010: There can be a maximum of 6 PLL/DLL reference and/or fabric clocks coming driven by the FPGA fabric in the <NW|SW|NE|SE> location.

There are four "corners" (NW, SW, NE, NW) that PLL and DLL instances can be placed in on each MPF300 or MPF200 FPGA device.

You can place multiple PLL/DLL instances in each corner. However, for each corner, the sum total of PLL/DLL reference clocks and fabric clocks that the fabric drives must be six or less.

PRPF_011: There can be a maximum of 2 PLL/DLL reference clocks coming driven by the FPGA fabric in the <NW|SW|NE|SE> location.

For each corner, only two PLL/DLL reference clocks can be driven by the fabric.

NOTE: For information about the set_location command, refer to the PDC Commands User Guide for PolarFire EAP3.

Placement Rules for RGMII, SGMII, and IOG CDR Interfaces

Placement rules must be adhered to for RGMII, SGMII, and IOG CDR interfaces. Non-compliance with these rules may result in the following errors:

PRPF_001: Port <port name> for Interface <inst name> must be placed before running Place & Route.

All PADs must be placed using the set_io command.

PRPF_002: Interface <inst name> has ports that must be assigned to the same physical lane. The current port assignment for this interface does not meet this requirement.

For the SGMII interface and IOG CDR, all RX_ and TX_ PADs must be placed in the same lane.

For the RGMII interface, all RX [] PADs and the RXCLK PAD must be placed in the same lane.

PRPF_003: The current Interface <inst name> port assignment requires that pin <pin name (functional pin name)> be reserved. You must not assign any port to that package pin.

For the SGMII interface and IOG CDR, the DQS_N pin of the lane is reserved for internal use. It must be left unused.

PRPF_004: You must not assign <inst name> to any location. Use the set_io command to assign any Interface port to package pins. This instance will automatically be placed.

IOD instances with TRAINING/OVERLAY should not be placed by users. These are internal instances, and will be handled by the tool.

PRPF_005: Port <port name> for Interface <inst name> must be assigned to <pin name (functional pin name)>.

For the RGMII interface, RX_CLK must be assigned to the DQS (P pad) of the lane.

NOTE: Refer to the "set_io" on page 113 help topic for more information about this command.

Placement Rules for Transceivers

For PolarFire designs with the transceiver (XCVR) interface, some placement rules apply. Non-compliance with these rules may result in the following errors:

PRPF_007: TxPLL <inst name> must be placed before running Place & Route.

Transceiver Tx PLLs must be placed by the user with the set_location command before running Place and Route.

PRPF_008: Dedicated XCVR ports <port name>* must be placed before running Place & Route.

The transceiver interface has dedicated ports. These must be placed using the set_io command. For information about rules, refer to UG0677: User Guide PolarFire FPGA Transceiver.

PRPF_009: Dedicated XCVR reference clock port <port name> must be placed before running Place & Route.

All transceiver reference clock PADs must be placed using the set_io command before running layout.

NOTES:

For information about the set_io command, refer to the "set_io" on page 113 help topic.

For information about the set_location command, refer to the PDC Commands User Guide for PolarFire EAP3.

For more information about rules for transceivers, refer to UG0677: User Guide PolarFire FPGA Transceiver.

I/O Standards

I/O Standards Table

Use the I/O Standards table to see which I/O standards can be applied to each family.

Table 6 · I/O Standards

I/O Standard	SmartFusion2, IGLOO2, and RTG4	PolarFire	IGLOO	SmartFusion and Fusion	ProASIC3
CMOS					
CUSTOM					
GTLP25			IGLOOe only	X	ProASIC3E and ProASIC3L only
GTLP33			IGLOOe only	X	ProASIC3E and ProASIC3L only
GTL33			IGLOOe only	X	ProASIC3E and ProASIC3L only
GTL25			IGLOOe only	X	ProASIC3E and ProASIC3L only
HSTL1	X	X	IGLOOe only	X	ProASIC3E and ProASIC3L only
HSTLII	X	X	IGLOOe only	X	ProASIC3E and ProASIC3L only
LVCMOS33	X	X	X	X	X
LVCMOS25	X	X	IGLOOe only	X	X
LVCMOS25_50			X	X	X
LVCMOS18	X	X	X	X	X
LVCMOS15		X	X	X	X
LVCMOS12	X	X	X		ProASIC3L only
LVTTL	X	X	X	X	X

I/O Standard	SmartFusion2, IGLOO2, and RTG4	PolarFire	IGLOO	SmartFusion and Fusion	ProASIC3
TTL			X	X	X
PCI	X	X	X	X	X
PCIX			X	X	X
SSTL2I and SSTL2II	X	X	IGLOOe only	X	ProASIC3E and ProASIC3L only
SSTL3I and SSTL3II			IGLOOe only	X	ProASIC3E and ProASIC3L only

Note: 1.2 voltage is supported for ProASIC3 (A3PL), IGLOOe V2 only, IGLOO V2, and IGLOO PLUS devices only.

See Also

I/O Standard