SIEMENS EDA

Catapult® Synthesis Library Builder

Software Version v2021.1
August 2021

SIEMENS

Table of Contents

Chapter 1:   Creating Memory Libraries

Memory Generator within Catapult allows you to import VHDL, Verilog, or Liberty memory models and generate a memory library that can be used within Catapult. This tool will also generate a SystemC model of the memory and SystemC transactors that can be used with SCVerify. It also adds a dependency to the imported HDL and/or Liberty file for the netlisted RTL design. If you specify a datasheet (.ds) or PDF file, Catapult will add a link to display this file from the Architecture Constraints window.

During library compilation, Memory Generator writes all files to a temp directory within your specified Output directory. After it verifies that the compilation was successful, Memory Generator moves the files into your specified output directory and removes the temp directory. If there was an error, the files in the temp directory are intended to be used for debugging purposes.

The Memory Generator is accessible from the Task Bar, the Libraries Constraint Editor and the Tools menu. It provides a Wizard-like GUI that guides you through the process. Each of the screens in the wizard has online help. The Memory Generator GUI also performs dynamic error checks so you can isolate any memory issues prior to generating or using the memory within Catapult.

The Memory Generator can handle the following types of memory models:

- Single or Multiple Port memories (eg. Dual-port RAMs). Each *Port* must have it's own address, clock and data lines.
- Handle Read/Write conflict resolution
- Address and Data bus widths can be hard-coded or using generics (VHDL) or parameters(verilog)
- Reset can be defined as async/sync or active high/low.
- Chip-wide or Byte-wise write enables
- Register Files as memories
- Memories with latencies greater than 1.
- Pins on the memories can be automatically tied high/low.

The following list describes the open issues with using the Memory Generator:

- Dualport support. The control signals for all ports must have the same phase (eg. Active high/Active low).
- Async memory read. Memories with 1 readwrite port where the write port is synchronous and the read port is asynchronous are not supported. Async memory read ports can not have a Chip Select.
- Direct pin types are not connected correctly for SystemC designs. New ports are added to the top level interface. Expect ports to be connected by name for inferred SystemC memories.
- Verilog localparms are not supported.
- Include files in the memory must have absolute pathnames. If the paths are relative, then you must either concatenate the files together or use absolute paths.
- Latency / Delay values apply to all ports. Memories having ports with different timings are not supported.

1.1. Generating Memory Libs using GUI

The Memory Generator allows you to read existing HDL simulation memory models into Catapult and store them as libraries for re-use in other projects. You can access the Memory Generator from the Task bar, Tools menu or Libraries constraint editor. After you open the Memory Generator, follow the steps in the wizard. Whenever the Memory Generator requires a field or detects an incorrect value, it will turn the field to red.

The following sections describe each form within the Memory Generator:

1. Select Technology Files
2. Memory and HDL/Library Parameters
3. Specifying the Model Ports
4. Generating the Model

1.1.1. Select Technology Files

In the first form of the Memory Generator, you can specify one or more files associated with a single memory model. The resulting memory library will be saved in the Catapult working directory. When you have completed this form, you can click the Import button and Catapult will parse the specified files and attempt to pre-populate the Parameter and Port information.

Catapult should be able to parse the name, direction, and width of the pins in the model from VHDL and Verilog files.

Base Technology Library

The following three fields denote the default base library that will be used with the memory model. Memory Generator creates a listbox for each field based on the information that is currently available to Catapult.

- **RTL Synthesis Tool** specifies the flow that was used to characterize the base library. Each library is specific to the synthesis tool. If you use multiple synthesis tools, then you will need multiple libraries. Memory Generator pre-populates this list based on the Search Path defined in the Libraries Constraints Editor. The Catapult libraries have a pre-determined set of supported flows (synthesis tools).

- **Vendor** specifies the ASIC or FPGA vendor of the library. This listbox is pre-populated with the Vendor names of all libraries that are found in the library search path. You can select a specific Vendor or use the '*' character to have Catapult make the memory available for all vendors.

- **Technology** specifies the process point of the library (eg. 65 nm). This listbox is pre-populated with the Technology names for the Vendors specified in the Vendor Listbox. You can select a specific Technology or use the '*' character to have Catapult make the memory available for all technologies.

Import Memory Files

The main part of this form allows you to add one or more files associated with a single ram into the Memory Generator. During the file import, Catapult will parse the files and attempt to pre-populate the remaining forms in the Memory Generator. This section has the following options:

Using the **Add Files** button, you can add multiple files and use the **Parse** column checkbox to determine whether Memory Generator should attempt to extract values from the model file.

- Catapult determines the language of the memory model for the **File Type** column based on the file extension. Even if the model is not parsed, it will still be added to the memory library as a file

dependency. Verilog models must use *.v and VHDL models must use *.vhdl or *.vhd. Liberty files use a .lib extension and the Datasheet can use either .ds or .pdf.

- The **Path Type** column controls the final pathname and location of any memory files (eg. Verilog).

  ◦ *Copy Local* stores the dependant files in the *Output Directory* specified at the bottom of this form.

  ◦ *Relative* stores the dependant file pathnames relative to the *Output Directory* path. No dependant files are copied.

  ◦ *Absolute* stores the dependant file pathnames as an absolute path from it's current location. No dependant files are copied.

- The **Model Type** column specifies which downstream should use the dependant file. The generic setting will include the file in the dependecy list for all simulation and synthesis flows.

- The **Static Flag** determines whether Catapult should *copy* or *point* to the referenced file when a solution contains this memory. If set to *yes*, the file path is included in the file dependencies. If set to *no*, the file is copied into the solution when the netlister [PackageOutput](#) option is true.

- **VHDL Library Mapping:** specifies logical and physical library mapping that may be required to use the VHDL model of the Memory. If your VHDL memory model references another library outside of 'work', then you must supply the library mapping in this field.

Import Library Data from memory model allows you to override an existing settings in the Memory Generator with the currently specified HDL file. If this checkbox is selected, Memory Generator will parse the names of the pins and parameters/generics in the memory model HDL file and pre-populate the corresponding forms. Unchecking this option allows you to make changes to other options on this form without overriding any previously entered data on other forms within the Memory Generator.

Top Module denotes the name of the top level memory model. Some models may contain multiple modules/entities. This value should be the name of the top level Verilog module or VHDL entity.

Output Directory specifies the location to store the generated memory. This directory will contain the generated Catapult library for the memory (<module>.lib), SystemC models for simulation and a (optionally) simple test design to verify the memory in SCVerify. This field must contain a pathname. If an absolute pathname is not used, the memory library is stored relative to the current Working Directory specified in Catapult.

1.1.2. Memory and HDL/Library Parameters

In this form, you can configure the parameters on the imported memory model. The Memory Generator attempts to read the parameters from the imported VHDL and Verilog memory files. If any invalid data is specified in the form, Memory Generator will change the background of the field to light red.

This pre-determined list of parameters allow Catapult to map the functionality of the HDL model to the internal functions of Catapult's memory operators. Each of these fields will accept either numbers or HDL/library parameter names. You can not use expressions.

- **Module Name** contains the module/entity name from the HDL memory file.

- **Library** contains the name of the library.

- **Data Width** specifies the number of bits on the data bus. This value includes the entire width of the data bus regardless of whether the data bus uses write/read masks. This value may be a hard-coded number or a module parameter (eg. data_width).

> **NOTE***: It is recommended that you specify the module parameter in this entry and use the hard-coded value in the Parameters section.*

- **Number of Words** specifies the number of bits in the word. If this value is greater than '1', Catapult will be able to read/write multiple words within the same read/write cycle using the read/write mask pins. Memory Generator assumes the Data Width divided by this value results in an equal number of words. The upper bit of the write and read mask corresponds to the word containing the MSB of the data bus. The LSB of the mask corresponds to the word containing the LSB of the data bus.

  > **NOTE***: It is recommended that you specify the module parameter in this entry and use the hard-coded value in the Parameters section.*

- **Area** specifies the size of the memory in library units. Each base technology library may use a different unitless measure to specify the relative size of all cells in the library. Catapult adds this value to all area summary and BOM reports to give a more accurate measure of the overall area usage of the design.

- **Read / Write Behavior** specifies the Read/Write resolution—whether the Read or Write function should have precedence when the design attempts to read and write to the same memory address. This information should be designed into the HDL memory model logic. For example, if you specify Read before Write for Read/Write Behavior, then when the design attempts to read and write at the same memory address, the memory will act as follows:

  1. read the current value of the memory address to the data output

  2. write the data input value to the memory address

  If you select "Unknown", then the HDL memory model was designed to produce random values on the data output.

- **Read Latency** specifies the number of clock cycles that Catapult should schedule for each read operation.

- **Write Latency** specifies the number of clock cycles that Catapult should schedule for each write operation.

- **Read Delay (ns)** specifies the propagation delay from the active edge of the read clock until the data is available on the output pin of the memory model.

- **Write Delay (ns)** specifies the time for asynchronous writes. If your memory does not have this functionality, leave it as the default value.

- **Setup Time (ns)** specifies the time that the input data must remain stable prior to the active write clock edge.

- **Init Delay** specifies the number of clock cycles before starting the next accesses to this port. The read or write latency can be greater than this value. For example, if this value is set to '2', then the memory can be pipelined down to II=2, but not II=1.

- **VHDL Array Path** specifies the signal name of the internal 2-D array that stores the memory values within the HDL model. This value is only used in the SLEC flow.

- **Verilog Array Path** specifies the signal name of the internal 2-D array that stores the memory values within the HDL model. This value is only used in the SLEC flow.

HDL & Library Parameters

This section defines the parameters that can be used in other sections of the memory library in place of explicit values. There are two types of parameters: 1) *HDL parameters*, those imported from the HDL memory model itself, and 2) *user-defined library parameters*. You can specify default, minimum, and maximum values for each parameter. Where allowed, leaving the values blank will automatically use predetermined default values.

All HDL parameters must be utilized somewhere within the memory generator configuration or explicitly set to *Ignore* in the parameter configuration. All user-defined library parameters must have *Min* and *Max* defined to bound the parameter.

It is recommended that you mark all parameters as Ignore except the data width, address width and number of words. For these three parameters, you should set the default values to match the parameter values in the simulation model. This setting will ensure that the parameters have the correct value when simulating the RAM.

After you complete this step, click the **Next>** button to proceed to specifying the model ports.

1.1.3. Specifying the Model Ports

The *Port and Pin Configuration form* in the Memory Generator allows you to specify single-port, dual-port or multi-port memory models. These memory models may contain memories or arrays of registers.

Within the Memory Generator, there are difference between a *memory port* and the *port defined in the VHDL/Verilog model* (which will be referred to as the Physical Pin (Pin)). The Memory generator uses the following definitions for these terms:

- A **Port** defines the number of address buses that can access the Catapult memory model. Each port configuration defines how the set of physical pins on the model will be connected. During optimization, Catapult will search the memories in the library for available memory operations. A memory may have one or more ports for each mode (eg. single-port read, dual-port write, single-port readwrite).

- A **Pin** is the pin defined in the VHDL entity or Verilog module. A Pin can be defined on multiple ports in the Catapult memory model.

For each Memory Port, you will need to define the following information:

- **Port** displays a color-coded, generated id that is used within the Memory Generator. It is a read-only field. Each of this values are added to the Pin mapping table at the bottom of the form.

- **Port Identifier** allows you to specify the port name that will be used within the transactor of the SystemC model.

- **Port Mode** allows you to specify the type of port.

- **Del** allows you to delete a port from the tables.

For each pin, you must map functionality of the pin to a pre-defined list of functions that are available for the port. You will also need to assign each pin to one (or more) Memory Ports. Memory Generator checks for invalid entries in each row. The following list describes the columns in the PINS table:

- **Pin** infers the list of pins from the imported VHDL/Verilog source model.

- **Direction** infers the direction of the pin from the imported VHDL/Verilog source model.

- **Width** infers the number of bits on the Physical Pin defined in the previous Parameters form.

- **Pin Function** defines the functionality of the Physical Pin. You can view descriptions of the available pin functions by hovering your mouse over the Pin Function listbox of the respective pin. These pin functions use common terms for memory pins with the following exceptions that are specific to Catapult (neither of these options apply to SystemC designs):

  - **Direct** connects the Pin from the memory to the top level interface. Catapult will add additional ports as needed to route the pin through hierarchy. Each direct pin will have a unique port on the top level interface.Catapult uses the Pin name on the memory model as the top-level port name.

  - **Global** connects all Pins with the same name to a single port on the top level interface. Catapult uses the Pin name on the memory model as the top-level port name.

- **Phase** specifies the relevant phase options for the pin.

- **Default** specifies the initial value on the pin. This value should only be used to set a pin on the memory model to a constant.

- *Port Assignment* is a series of checkboxes that allow you to group a Pin with a Port. Memory Generator will automatically stipple-out any ports that are not relevant to a specific pin. For example, a Read Enable pin would not be used with a Write Port.

NOTE: The Direct and Global functions do not apply to SystemC designs.

NOTE: If you need to drive a pin with a constant value, set the Pin Function to Unconnected with a Default value of '1' or '0'.

*NOTE: You should mark all non-memory-access related pins (eg. Bist pins) as **Direct** and set their default value to configure the RAM for normal operaton.*

NOTE: All "Unconnected" input pins must be set to a default value to ensure quality design practices. "Unconnected" output pins may be left dangling. If a pin is not to be used by Catapult, but sill has some design function, you may want to declare the input pin as either "direct" or "global". This declaration would allow you to connect it at the top level of the design.

After you complete this step, click the **Next>** *button to proceed to generating the model ports.*

1.1.4. Generating the Model

In this dialog of the Memory Generator, you will generate the library model, SystemC simulation model, SystemC transactor, and a TCL script for the memory. Catapult execultes the TCL script in Catapult to generate the memory files. You can re-purpose this TCL file to generate other similar memories. Any errors will be reported in the transcript of the Memory Generator.

In the Output Folder specified in the initial dialog, Catapult will write a compiled library model (*.lib), SystemC simulation and transaction models for verification (*.h). The basename for all of the generated models is the Module Name.

After you have successfully generated the memory library, you can make the memory accessible to Catapult by adding the directory containing the Catapult memory library to one of the following locations:

- *Component Library Search Path Option*. In Catapult, select the Component Libraries pane in the Options form (Tools > Set Options) and add the directory path to the list.

- *Library Constraint Editor:* In Catapult, select the Libraries task and click on the Search Path button to add the directory containing the Catapult memory library.

After Catapult loads the memory library, it will be an available *Resource Type* to use when specifying Architecture constraints (eg. Mapping arrays in the core design to the memory).

1.2. Creating Memory Libraries from a file

In addition to using the Memory Generator GUI, you can also specify the functionality and parameters for memory libraries using the TCL file. This flow can be very useful in the situations where you need to generate libraries of several similar memories.

The Memory Generator creates a TCL file containing the necessary parameters for generating a Catapult memory library. This section includes the reference information for the fields within the TCL file.

To generate a memory from the file, you must load the Memory Generator flow and then run the MemoryGenerator_BuildLib option. Because there are a significant number of fields in the memory generator specification, it is highly recommended that you generate the initial memory using the Memory Generator GUI. The GUI will produce a template in the library output directory that you can copy and modify for subsequent memories. The basic format of the Memory Generator file is :

```
flow package require MemGen
flow run /MemGen/MemoryGenerator_BuildLib {
    <memory file fields>
}
```

NOTE: All values are required and must exist in the TCL file otherwise the library will be generated with empty values and may not function as expected. This file adheres to TCL syntax rules. All values in the file are case sensitive.

Since the file is read into Catapult via the command line, all fields within the memory specification file must adhere to TCL syntax rules. The following table describes the main fields of Memory TCL file. The remaining tables describe the syntax of the FILES, PARAMETERS, PORTS and PINMAPS sections of the Memory TCL file.

Table 1: MemoryGenerator_BuildLib file format

Keyword	Values	Default	Description
AREA	integer	0	Size of the memory in default area units of the base technology library.
DEPTH	Integer	0	Number of addressable locations within the memory. The default value of "0" implies that the full address width is available to access the memory locations.
FILES	file_struct		List of file dependencies for the memory. Catapult includes this path name in the dependency file when the RTL netlist is written. This list can include one or more files. Refer to Table 2, for information on the file_struct syntax.
INITDELAY	Integer	1	Initialization cycles. This value defines how may clock cycles the memory needs to write a value into memory.
INPUTDELAY	Float	0.01	Propagation Delay on the input pins.

LIBRARY	String	Module name from HDL file	Base Library name corresponding to the memory library. Catapult evaluates the Library Search Path to locate memory libraries that are associated with a Base technology library. When you select a base library in the GUI, Catapult will also display all supplemental library that are associated with the base library. If you specify a library name of "*", this memory library will be associated with all base libraries.
MODULE	String	Module name from HDL file	Memory name. This value should correspond to the module name in the verilog file or entity name in the VHDL file. Catapult uses this name when it instantiates the memory in the netlist.
OUTPUT_DIR	Pathname		Pathname to write the resulting library files. On successful generation of the memory, this directory will contain the .lib memory library for catapult and the SystemC models for simulation. This path can contain an absolute or relative pathname.
PARAMETERS	param_list	Empty list	List of memory parameters. In the GUI flow, this list is pre-populated when the HDL simulation model is imported. *NOTE: The values in this structure are not used to generate the library. This structure may be used in a future release.*
PINMAPS	pinmap_list	none	Maps pins on the simulation model to pins on Catapult's internal memory port components. Refer to for more information on the items in the PINMAP Section.
PORTS	port_list	none	List of Port Definitions. This list defines the names of the available port modes for the memory (eg. Read, Write). Refer to for more information on the items in the PORTS section.
RDWRRESOLUTION	RBW| WBR| UNKNOWN	UNKNOWN	Defines how the memory handles reading and writing to the same address on the same clock cycle. • RBW (0): Reads the values from memory prior to writing a new value • WBR (2): Writes the new value to memory and propagates the new value to the output • UNKNOWN (1): Write X(s) to the memory and output

			MemGen maps the corresponding numeric value (denoted in parenthesis) to the *RdWrResolution* property on the *all* binding of the module.
READDELAY	Float	0.1	Clock-to-Q delay of the memory during a read operation
READLATENCY	integer	1	Number of clock cycles to complete the read operation
RTLTOOL	flow name	DesignCompiler	Name of the flow that will be used for RTL synthesis of the output netlist containing the memory. This tool will be used with the 'go synthesize' command. The value can only contain one tool name. Wildcards are not allowed. This value must match the RTL tool name in the base library. The templates within Library Builder use a fixed number of flow names so these values should be consistent across multiple libraries. Catapult uses this value as the initial criteria for displaying libraries in the Libraries Constraint Editor.
TECHNOLOGY	String | *	none	Name of the Technology used with the Base Library. This value must match the base library name or the wildcard character. You can not use partial wildcards (eg. 'tsmc*'). The wildcard character implies that it is valid for all technologies in the specified flow.
VENDOR	String | *	none	Name of the Vendor used with the Base Library. This value must match the base library name or the wildcard character. You can not use partial wildcards (eg. '1*'). This wildcard character implies that the memory library is valid for all vendors in the specified flow.
VERILOGARRAYPATH	array_name	{}	Specifies the name of the array within the simulation model that contains the 2-D memory values. This value is only used by the SLEC verification tool.
VHDLARRAYPATH	array_name	{}	Specifies the name of the array within the simulation model that contains the 2-D memory values. This value is only used by the SLEC verification tool.
VHDL_LIB_MAPS	<mapping>	{work ./work}	One or more items that map the logical library name that are referenced in the VHDL model to the physical directory. By default, all simulation models are mapped to the work library.

WIDTH	Integer or parameter	{}	Width of the data port. This value can be a parameter or integer value.
WRITEDELAY	float	0.1	Propagation delay prior to writing a value to a memory with an asynchronous write. This value must be a positive, non-zero value. If you do not have an asynch write, use the default value.
WRITELATENCY		1	Number of clock cycles to complete a MEMORYWRITE operation.

FILES Section

Table 2 describes the format for the list item included within the FILES section. This section contains one list item for each HDL file that is required by the simulation memory model. These files will be included in the memory library as file dependencies. Whenever this memory is included in an output netlist, these simulation files will be referenced in the dependency file for the netlist. The following text shows the format of the FILES section:

```
FILES {
    {FILENAME <file_path> FILETYPE <file_type> MODELTYPE <modeltype>}
    . . . .
    {FILENAME <file_path> FILETYPE <file_type> MODELTYPE <modeltype>}
}
```

Here is an example of the FILES section for a memory model with a reference to a single HDL simulation model:

```
FILES {
    {FILENAME /u/catuser/mem/ram.v FILETYPE Verilog MODELTYPE GENERIC}
}
```

Table 2: Syntax of FILES Section

Object	Description
FILENAME *file_path*	Pathname to the simulation model. This value can be absolute or relative. Relative pathnames are relative to the Catapult working directory when you run the Memory Generator
FILETYPE *file_type*	HDL language of the reference simulation model. This value can be either VHDL or VERILOG.
MODELTYPE *model_type*	This value should be set to "GENERIC". At this time, ithe MODELTYPE is not used during library compilation. This field will be used in a future enhancement to the Memory Generator. It is included in this release for future compatibility.

PARAMETERS Section

Table 3 describes the format for the list item included within the PARAMETERS section. This section contains one list item for each parameter from the simulation memory model that Catapult would need to

modify to implement the variations in the memory. When using the Memory Generator GUI, each parameter in the HDL simulation model is added during import.

The following text shows the format of the PARAMETERS section:

```
PARAMETERS {
     {PARAMETER <name> MIN <min_value> MAX <max_value> DEFAULT <default_value> }
     . . . .
     {PARAMETER <name> MIN <min_value> MAX <max_value> DEFAULT <default_value> }
}
```

Table 3: Syntax of the PARAMETERS Section

Object	Description
PARAMETER <name>	Name of the parameter used in the HDL simulation model.
MIN <min_value>	Minimum value that Catapult can use for utilizing this memory. This value can be an integer or float value. If there is no minimum value, then specify an empty list ("{}") for the <min_value>.
MAX <max_value>	Maximum value that Catapult can use for utilizing this memory. This value can be an integer of float value. If there is no minimum value, then specify an empty list ("{}") for the <max_value>.
DEFAULT <default_value>	Default value for this parameter. This value can be a float or integer value. If there is no minimum value, then specify an empty list ("{}") for the <default_value>.

PINMAP Section

Table 4 describes the format for the list item included within the PINMAPS section. This section contains one list item for each pin on the simulation memory model that Catapult needs to connect when it instantiates the memory in a netlist. The following text shows the format of the PINMAPS section:

```
PINMAPS {
     {PHYPIN <phy_name> LOGPIN <log_name> <FUNCTION> DIRECTION <dir> WIDTH <width>
PHASE<phase> DEFAULT <default_value> PORTS <port_list> }
     . . . .
     {PHYPIN <phy_name> LOGPIN <log_name> <FUNCTION> DIRECTION <dir> WIDTH <width>
PHASE<phase> DEFAULT <default_value> PORTS <port_list> }
}
```

Here is an example of the PINMAPS section for a memory model with a reference to a single HDL simulation model:

```
PINMAPS {
  {PHYPIN clk LOGPIN CLOCK DIRECTION in WIDTH 1.0 PHASE 1 \
          DEFAULT {} PORTS {r w}}
  {PHYPIN reb LOGPIN READ_ENABLE DIRECTION in WIDTH 1.0 PHASE 1 \
          DEFAULT {} PORTS r}
  {PHYPIN dout LOGPIN DATA_OUT DIRECTION out WIDTH dwidth PHASE {} \
          DEFAULT {} PORTS r}
  {PHYPIN addr LOGPIN ADDRESS DIRECTION in WIDTH awidth PHASE {} \
          DEFAULT {} PORTS [r w]}
  {PHYPIN web LOGPIN WRITE_ENABLE DIRECTION in WIDTH 1.0 PHASE 1 \
          DEFAULT {} PORTS w}
```

```
    {PHYPIN din LOGPIN DATA_IN DIRECTION in WIDTH data_width PHASE {} \
        DEFAULT {} PORTS w}
}
```

Table 4: Syntax of PINMAPS Section

Object	Description
PHYPIN <phy_name>	Name of the pin in the HDL simulation model.
LOGPIN <pin_function>	Name of the pin on the internal Catapult memory operator. Valid values are: • **ADDRESS**: address for any type of port • **DATA_IN**: Data to be written into the memory • **DATA_OUT**: Data read out of the memory • **WRITE_ENABLE**: Single bit write control • **READ_ENABLE**: single bit read control • **PORT_ENABLE**: single bit enable for the port which is asserted during either read or write operations • **WRITE_MASK**: Write controlwith more than 1 bit. Supports partial word operations • **READ_MASK**: Controls reads of less than a full word • **CHIP_SELECT**: Enable/disable ALL memory operations (but pipelines continue to flush) • **DIRECT**: Pin that is routed to the top level interface. All direct pins will have unique names on the top level. • **GLOBAL**: Pin that is routed to top level interface. All global pins with the same name will be connected. • **UNCONNECTED**: Output pins will not be connected. Input pins will be tied to the value of the DEFAULT field. • **CLOCK**: Clock for the memory • **ENABLE**: Clock enable. This pin 'freezes' the pipelines in the memory if de-asserted. • **A_RST**: Asynchronous reset (of the memory pipeline) • **S_RST**: Synchronous reset (of the memory pipeline)
DIRECTION <dir>	Direction of the data flow for the pin. Valid values are 'in', 'out', 'inout'.
WIDTH <width>	Number of bits of the PHYPIN. Valid values are either an integer or a string with the name of a parameter that is defined in the PARAMETERS section.
PHASE <phase>	Active level or edge of the phypin. Valid values are '1' for active high or

	rising edge, '0' for active low or falling edge, or {} if the phypin is not sensitive to the level or edge of the signal (eg. Address and data buses).
DEFAULT <default_value>	Initial value of phypin. Valid values are '1' to initialize high, '0' to initialize low, '{}' to not initialize, or a string with the name of a parameter. At this time, this field is only used to define constants on UNCONNECTED phypins.
PORTS <port_list>	One or more PORT names that are defined in the PORTS section

PORTS Section

Table 5 describes the format for the list item included within the PORTS section. This section contains one list item for each port of the memory model. Each pin in the PINMAP section can use the PORTS object in the PINMAP section to map to a port defined in this PORTS section. The following text shows the format of the PINMAPS section:

```
PORTS {
    {PORT <name> MODE <type> }
    . . . .
    {PORT <name> MODE <type> }
}
```

Here is an example of the PINMAPS section for a memory model with a reference to a single HDL simulation model:

```
PORTS {
    {PORT r MODE Read }
    {PORT w MODE Write }
}
```

Table 5: Syntax of the PORTS Section

Object	Description
PORT <name>	Defines the name of the PORT. This value will be referenced in the PORT list in the PINMAP section.
MODE <mode>	Name of the memory mode on the memory operator. The values of mode are case-sensitive. Valid values are: • Read • Write • ReadWrite • AsyncRead

Chapter 2:   Creating Custom Components

Catapult provides you with built-in interface libraries that support simple protocols such as wire interfaces, two-way handshaking, and memory interfaces. The base libraries for both ASIC and FPGA provide all of the operators required to schedule an algorithm. This combination of interfaces and base operators is usually sufficient, but there are times when a designer may wish to leverage custom IP (MAC, Mult-Add, Wallace-tree multiplier, etc.) or interfaces (AMBA, APB, Avalon, PCIx, etc.).
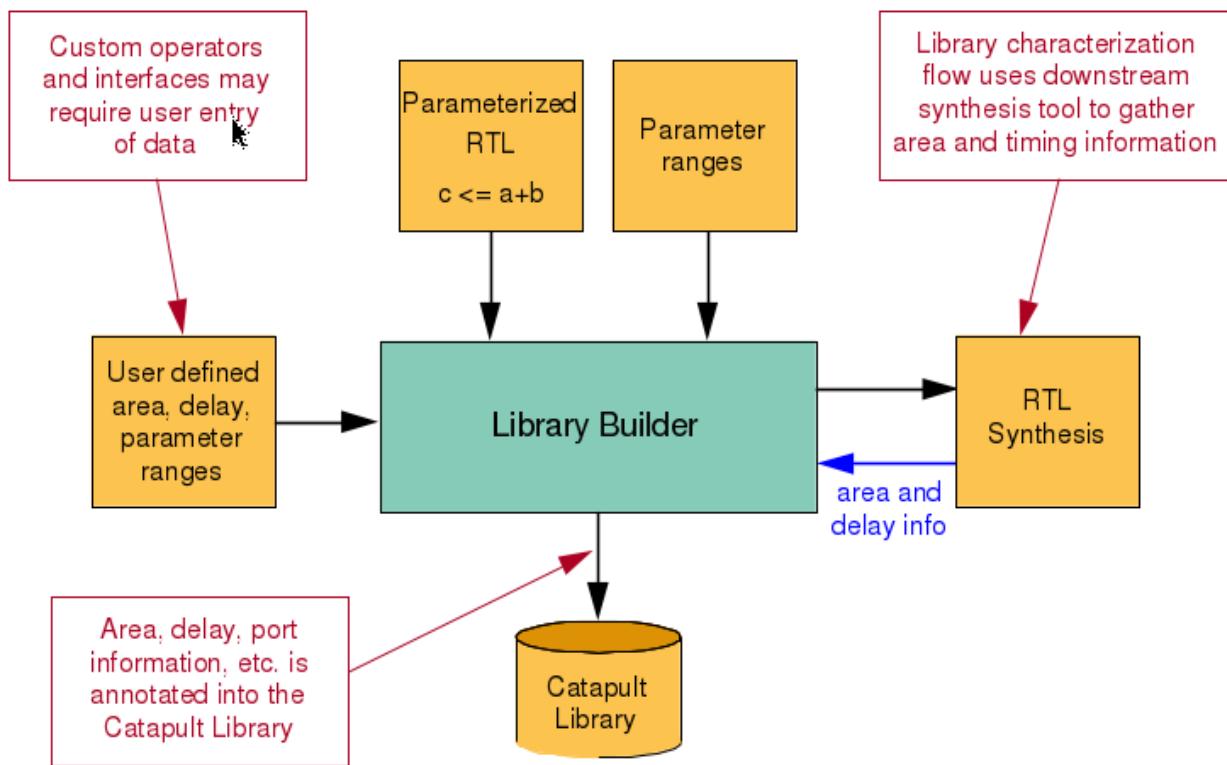
Catapult facilitates custom IP by enabling you to map an entire C++ function to a block of RTL to create a custom operator. This process is similar to the use of CCOREs, and can lead to huge performance gains in both ASIC and FPGA technologies when mapping to design ware or DSP blocks.

Creating a custom operator requires two main steps.

5. [Writing a Custom Operator C++ Function](#)

6. [Building Libraries for Custom Operators](#)

7. [Incorporating Custom Operators in Designs](#)

The Catapult Library Builder allows you to customize new libraries to match your custom C++ functions, thereby integrating existing RTL IP into the Catapult design flow.

A number of constraints limit this approach: HDL modules suitable for this process must have wire interfaces, all inputs must be be read in the first register stage, and all outputs must be written in the last register stage.

2.1. Writing a Custom Operator C++ Function

Although Catapult will not synthesize your custom operator C++ functions, you must still call these functions to inform Catapult where in the design custom IP should be inserted. A special pragma, map_to_operator, is used to indicate that the C++ function should be directly replaced with custom operator RTL implementation rather than synthesized. For example:

```
#pragma map_to_operator "multadd18x18"
void multadd18x18 (int18 a,
                   int18 b,
                   int18 c,
                   int36 &z) {
  z = (a * b) + c;
}
```

Your C++ functions must have the same functionality as the operator RTL. Any discrepancy will cause a simulation failure when verifying the C++ design against the RTL design. The function arguments must also match operator ports (in terms of number, name, and bit-widths). Additionally, the operator name must be an exact match for the operator name defined in the library.

Once the C++ function is complete, you can create a library in which to incorporate the function.

2.2. Building Libraries for Custom Operators

The Library builder provides a command, library import, that reads in custom operators from C++ source code, and their corresponding modules from RTL netlists. The command parses C++ code to extract the interface of the operator (port names, port directions and bitwidths) and automatically annotates that data in the "Operator" section of the library. It parses the RTL netlist to extract the module port names, port directions, and generics or parameters, and then annotates that information in the "Mods" section of the library.

The library creation process consists of five basic steps:

1. Creating the Blank Library
2. Importing the Operator from a C++ function
3. Importing the HDL module
4. Connecting the Library Module to the Operator
5. Saving the Library

Debugging with the Library builder GUI describes how you can use Library Builder to view and edit the properties of your Custom Component.

2.2.1. Creating the Blank Library

You can create custom libraries with Catapult either in batch mode or with the GUI. However, batch mode is typically easier to use, as you can create TCL files to execute the necessary steps. Verifying results is easiest with the GUI.

To launch Catapult Library builder in batch mode, enter **catapult -lib -shell** at the command line**.**

Next, create a blank library by using a command of the following form:

```
flow run /<package>/library add blank <options>
```

The <package> parameter can take different values depending on whether the design is ASIC or FPGA.

Supported ASIC packages	Supported FPGA packages
Design Compiler, RTLCompiler, OasysRTL	Precision, Quartus, Vivado, Synplify Pro

There are a number of <options> arguments:

- -libname <name for *.lib file>

- -libtitle <name displayed in Catapult>

- ASIC specific options:

  ◦ -Vendor

  ◦ -Technology

- FPGA specific options:

  ◦ -manufacturer <Xilinx, Altera>

  ◦ -family <device name>

  ◦ -part <device package>

  ◦ -speed <speed grade>

Pass in the synthesis tool of your choice, name the library, and set the appropriate vendor specific settings. Note that names are case sensitive. Catapult determines the available libraries based on the RTL Synthesis tool, Vendor (Manufacturer), and Family. If any of the names differ (including case) between your Base Technology Library and your Custom Component library, then Catapult will not load the Custom Component library nor show it in the GUI. You can use wildcards for *manufacturer*, *family*, and *part*.

For example, the following command creates a library named *DSPblock* for the Precision RTL synthesis flow with the *manufacturer* set to *Xilinx*. Since *family*, *part*, and *speed grade* are set to wildcards, this library applies to all Xilinx families, parts, and speed grades.

```
flow run /Precision/library add blank -libname DSPblock \
                                      -libtitle DSPblock \
                                      -manufacturer Xilinx \
                                      -family * -part * -speed *
```

2.2.2. Importing the Operator from a C++ function

Once a blank library exists, you can import a custom C++ function to the library to create the library operator with a command of the following form:

```
library import -libname <library name> \
               -operator <operator name> \
               <filename>.cpp
```

For example, to import the *multadd18x18* function defined earlier in this chapter use the following command:

```
library import -libname DSPblock -operator multadd18x18 multadd18x18.cpp
```

2.2.3. Importing the HDL module

To import a Verilog or VHDL module to the library, use a second *library import* command of the following

form:

```
library import -libname <library name> \
               -module <RTL module name> \
               -mod_type userop \
               -property_map <property name> <property value> \
               -port_default <port name> <default value> \
               -input_register <port name> \
               -verilog \
               -verilog_option <string> \
               <filename.v or filename.vhdl>
```

The *-module* and *-mod_type* arguments specify the module name, and that the operation associated with the module is user defined (*userop*). This command creates a module in the library with the same ports as the RTL.

To illustrate an actual case, the following sections describe how to populate the fields of each library import argument in relation to an example Verilog module multadd18x18.v:

```
module multadd18x18 (a, b, c, z, clk, srst);
   input  [17:0] a;
   input  [17:0] b;
   input  [17:0] c;
   output [35:0] z;
   input         clk;
   input         srst;

   reg    [35:0] z;

   always @(posedge clk) begin
     if ( srst ) begin
       z <= 36'b0;
     end
     else begin
       z <= ($signed(a) * $signed(b)) + $signed(c);
     end
   end

endmodule
```

A command to import this module begins by specifying the library name, module name, and module type:

```
library import -libname DSPblock -module multadd18x18 -mod_type userop ...
```

Next, you must specify required properties with the -property_map argument. Some of these properties only apply to sequential components.

Property Name	Behavior	Sequential Only
InputDelay	Input to register delay	Yes
Delay	Clock2Q (Sequential0 or combinational delay	No
Area	Area in library units used for the base technology.	No
SeqDelay	(# of register delays) - (input_register==true)	Yes
InitDelay	Ability to pipeline (e.g. InitDelay = 1 implies the design/module can be pipelined with II = 1).	Yes

MinClkPrd	Minimum clock period supported for scheduling.	Yes

The example *library import* command continues with the specification of multiple property mappings.

```
-property_map InputDelay 3.4 \
-property_map Delay 0.4 \
-property_map Area 0 \
-property_map SeqDelay 0 \
-property_map InitDelay 1 \
-property_map MinClkPrd 3.4 \
...
```

Lastly, to specify the Verilog module itself, the *-verilog* argument, with appropriate options must be passed to the library import command. For example, the following line, which concludes the example *library import* command, specifies the *multadd18x18.v* file, along with the *-verilog_option -sv* options.

```
-verilog -verilog_option -sv multadd18x18.v
```

The VHDL argument options are similar to those shown here. For a more detailed treatment of library import, and other Library builder commands, refer to the Catapult Library Builder manual.

Input Registers

Catapult enables the control of input registers with the input_registers argument provided to the library import command. This argument is not required, but you can set it to specify whether an RTL module has registers on its inputs. Doing so allows Catapult to remove output registers from the core process, and combinationally drive the RTL module. This allows for the chaining of other operations before the point that the RTL module is scheduled, resulting in a more efficient schedule and reduced latency.

The input register setting can also be used to force register removal on the Catapult process driving the module, even if the module does not have input registers. For this to work, the module must have at least one output register stage, the input delay must be set to account for the input to register delay of the RTL component, and the sequential delay must account for the input register setting on the ports.

Default Port Settings

You can set a default value for the RTL module ports with the *-port_default* argument to *library import* as shown below:

```
-port_default <port name> <default value>
```

Setting a default value for a port allows you to drive a value onto the port while the operator/module pair is inactive while the design is running.

2.2.4. Connecting the Library Module to the Operator

In the last major step of creating a custom operator, you add constraints to the module bindings to specify connections between the ports on the module and the ports on the operator, and also to specify which ports pertains to clocks, resets, and other signals. Use the library add command to pair signal types with the paths to each pin.

Connecting the library module to the operator requires that you assign a PINASSOC_TYPE to each module port with each library add command.

There are two types of PINASSOC_TYPES:

 • **SIGNAL** specifies clock, reset, and enable ports. The clock and reset phase in the RTL must match the

phase in the library. The PINASSOC_TYPE assignments have the following form for signals:

```
 library add <path to ports on binding>/<port name> – -PINASSOC_TYPE -SIGNAL
{[signal type]} -PHASE <0 | 1>
```

- *OPERATOR_PIN* specifies an operator port.

```
 library add <path to ports on binding>/<port name> – -PINASSOC_TYPE OPERATOR_PIN
<operator port>
```

Paths to module ports on the bindings have the form:

```
/LIBS/<library name>/<component name>/BINDING:1<component name>/PIN_MAPPING/<pin name>
```

For example:

```
/LIBS/DSPblock/multadd18x18/BINDINGS/1:multadd18x18/PIN_MAPPING/clk
```

The following code snippet shows an example of multiple port bindings and PINASSOC_TYPEs set within a TCL file.

```
set PIN_MAPPING "/LIBS/$libname/MODS/$compname/BINDINGS/1:$compname/PIN_MAPPING"

library add $PIN_MAPPING/srst -- -PINASSOC_TYPE SIGNAL  -SIGNAL {[S_RST]} -PHASE 1
library add $PIN_MAPPING/clk  -- -PINASSOC_TYPE SIGNAL  -SIGNAL {[CLOCK]} -PHASE 1

library add $PIN_MAPPING/a     -- -PINASSOC_TYPE OPERATOR_PIN -OPERATOR_PIN a
library add $PIN_MAPPING/b     -- -PINASSOC_TYPE OPERATOR_PIN -OPERATOR_PIN b
library add $PIN_MAPPING/c     -- -PINASSOC_TYPE OPERATOR_PIN -OPERATOR_PIN c
library add $PIN_MAPPING/z     -- -PINASSOC_TYPE OPERATOR_PIN -OPERATOR_PIN z
```

2.2.5. Saving the Library

Write the completed custom component library to a file with a command of the following form:

```
library save /LIBS/$libname -filename $libname.lib
```

For example, to save the example library created over the course of this flow:

```
library save /LIBS/DSPblock -filename DSPblock.lib
```

2.2.6. Debugging with the Library builder GUI

To confirm or adjust the properties of a custom library, you can load the library in Catapult Library builder by launching Catapult with the -lib argument.

```
catapult -lib <library name>.lib
```

The library explore menu allows you to view and edit variable, module port, binding, and operator parameters set during library construction.
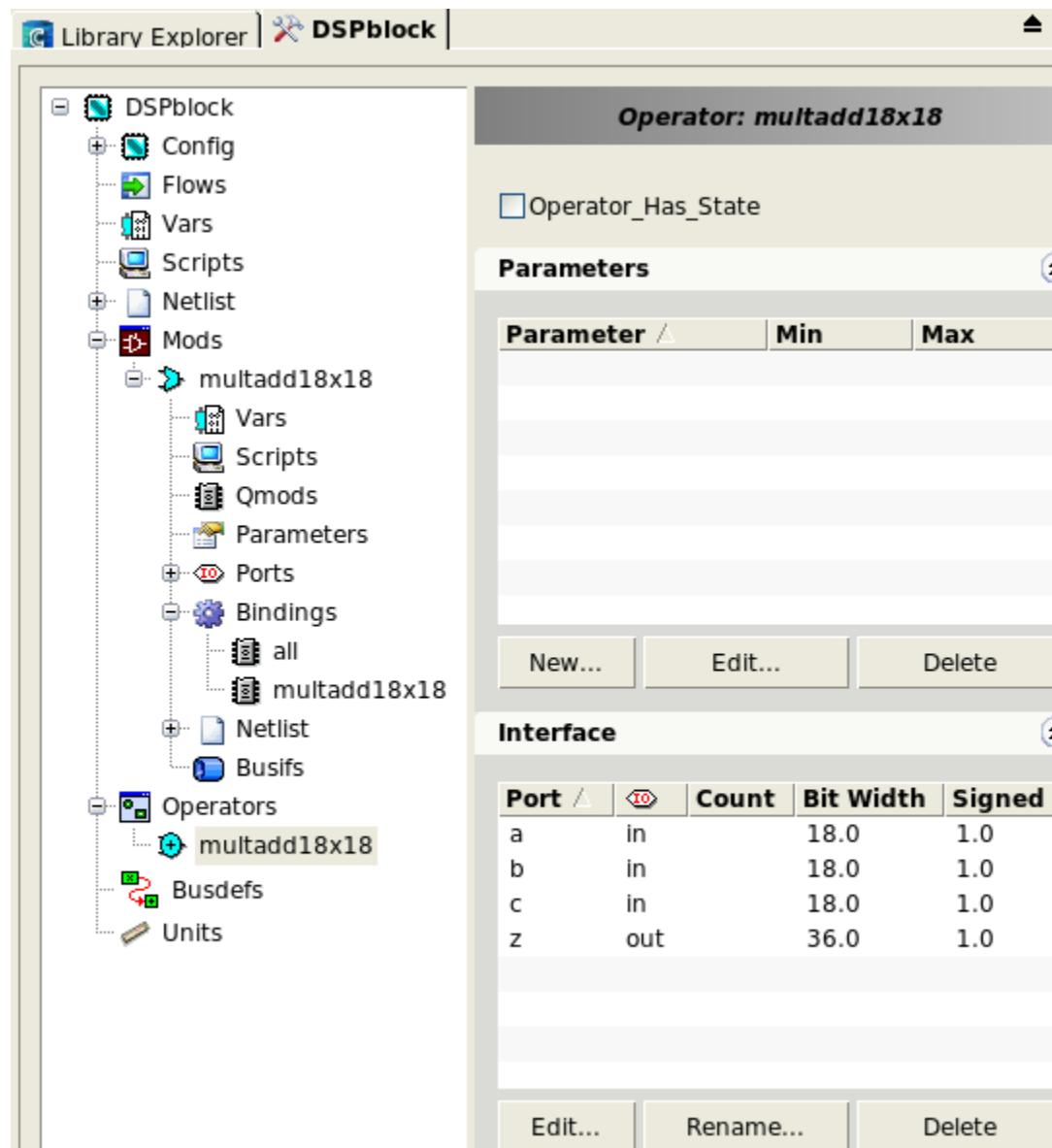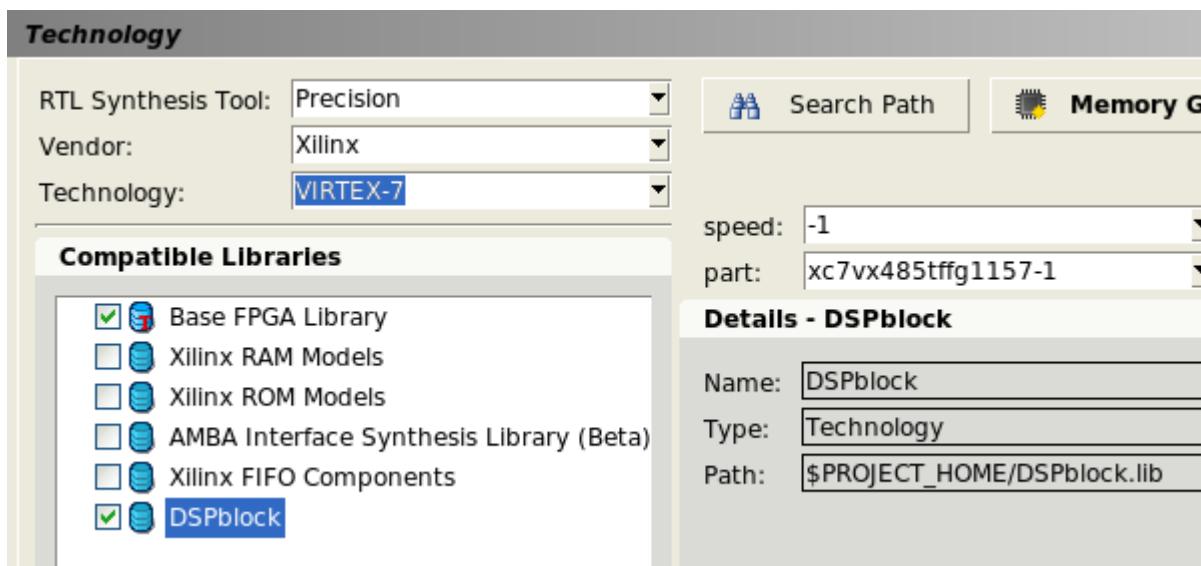
Illustration 1: Catapult Library Explorer
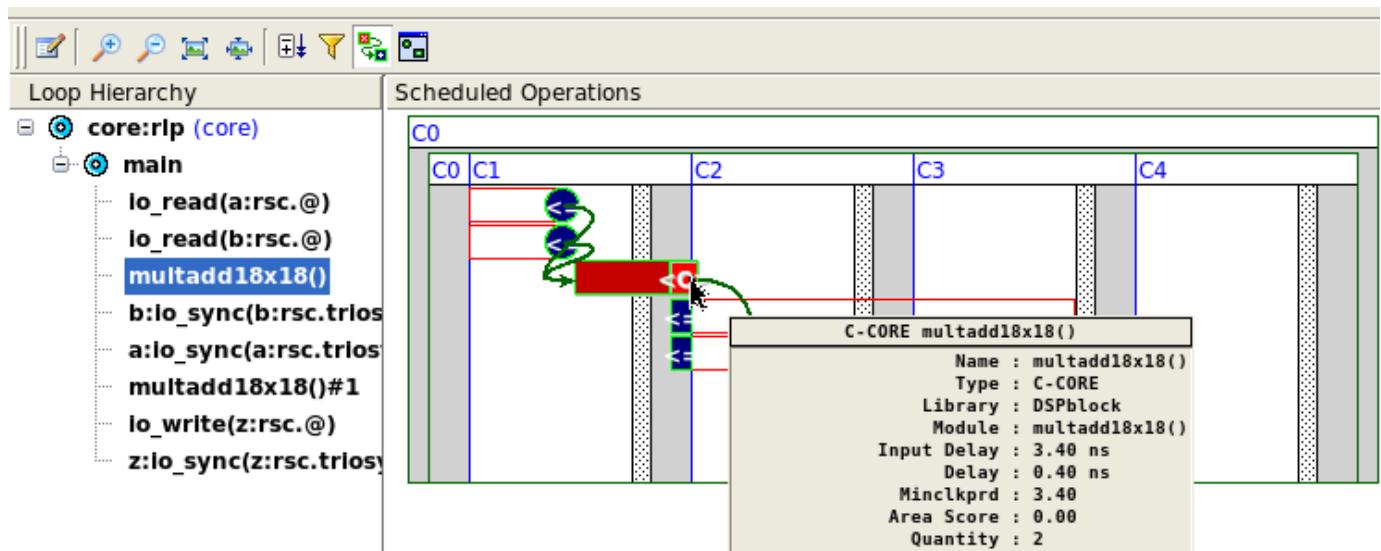
2.3. Incorporating Custom Operators in Designs

Once a custom operator is completed, incorporating it into a design is easy. Simply add the custom library to the Catapult library search path. As long as the design calls the custom C++ function created for the library, instances of custom IP will appear in the final netlist.

Illustration 2: Library Constraint Editor

The custom component will be scheduled as a single operation.

Illustration 3: Custom Component in Scheduled Operations

Chapter 3:   Characterizing ASIC Base Libraries

Catapult requires libraries created and characterized with Library Builder. Libraries that contain all the supported FPGA devices are pre-characterized and shipped with Catapult. However, for custom ASIC design, you must create and characterize a custom ASIC library (unless a Catapult library for your technology is provided by the foundry).

This chapter describes the procedure and best practices for creating and characterizing a new ASIC library using the Base ASIC Library template provided with Library Builder.

1. Preparing to Create a Library
2. Creating the Library
3. Characterizing Libraries or Components
4. Evaluating Library Characterization Results
5. Saving Libraries

NOTE: If this library will only be used within Catapult Physical using the on-the-fly methodology, then you do not need to perform the above-mentioned characterization or evaluation steps because Catapult will characterize any necessary operators as needed by the optimizer or scheduler.

The following sections describe each of these steps in detail:

- Troubleshooting Library Failures
- Improving the Accuracy of Estimated Characterization Values
- Using the Library Farm

3.1. Preparing to Create a Library

A custom ASIC library is created from the *Base ASIC Library* template and the technology library used with your RTL synthesis tool of choice. Library Builder generates the RTL components, runs them through RTL synthesis to gather timing, area, etc data, and saves that data to an ASIC library for Catapult.

Currently Library Builder supports characterization with Synopsys Design Compiler and Mentor Graphics' Oasys-RTL.

To ensure that the library characterization is adequate, you should prepare for library as described in the following topics:

- Basic RTL Technology Settings
- Specifying Wire Loads
- Determining Technology Libraries for RTL Synthesis
- Determining Licensing Approach

3.1.1. Basic RTL Technology Settings

The settings used for characterizing the ASIC library must match those used in your *in-house* RTL flow. Catapult generates RTL code that will likely be combined with other RTL code and synthesized into ASIC gates. If the synthesis flow used to characterize the new ASIC library differs from your standard flow, the area and timing estimates in Catapult are likely to be inaccurate.

You may need to contact your silicon vendor or your RTL synthesis team to determine what settings to use for the following RTL synthesis options:

- **Operating conditions** — define the process, temperature, and voltage settings.

- **Driving cell** — the technology cell driving your design inputs. We recommend this cell to be a relatively strong buffer (approximately 2/3 by area from those available in the library). For example if your library has 4 buffers with areas 100, 150, 200, and 250 we suggest picking the buffer with area of 200. This option is interchangeable with the *Input transition* setting but specifying the driving cell is recommended.

- **External load** — typical capacitive load attached to the outputs. It is recommended to be set to the capacitive load of a small buffer in your library. Increasing this load would make library more pessimistic.

- **Input transition** — input transition value. If the *Driving cell* option is used, you should leave this variable blank.

- **Other advanced settings** — extra settings such as boundary optimization, extra optimization settings, and so on. These options are based on individual preference, license availability, foundry/company guidelines etc.

3.1.2. Specifying Wire Loads

It is common for technology libraries to contain a default wire load. However oftentimes a wire load is responsible for poor correlation between Catapult and RTL synthesis estimations therefore we recommend against using wire loads.

To ensure that the wire loads are not used set the *interconnect_mode* to none.

3.1.3. Determining Technology Libraries for RTL Synthesis

Use the technology libraries associated with the RTL synthesis tool used in your *in-house* synthesis flow for Catapult library characterization. The format of the technology files vary based on the RTL synthesis tool as described in the following table:

Synthesis Tool	File Format
Design Compiler	.db
Oasys-RTL	.lib

Table 6: RTL Synthesis Tool Technology File Formats

At this time Catapult can only read technology files in Liberty format (.lib). Availability of the technology library in Liberty format allows for wizard-based automated library creation as well as for built-in

---

prototyping and other features unavailable otherwise.

For best results when creating an ASIC library, you should use the following guidelines for technology files:

- Most Low-Power technology libraries (65nm and below) use separate library files for High-Vt and Low-Vt cells along with a common library used by both. Specifying all of the libraries when you create your ASIC library is likely to make the RTL synthesis tool use the Low-Vt cells as they are faster with the same area. We recommend using the High-Vt library for characterization and adding the other libraries later so that they can be used during design synthesis.

- Do not use wire loads unless you understand their effect.

- Once the Catapult library is created, you may like to prune it to remove items that may cause unnecessarily long characterization run times. For example: Remove large multipliers (mgc_mul with an operand width of 64 bit and greater unless your designs are likely to require them). Use this approach for any other operator as long as you are sure the data points will never be needed. Catapult will warn you during design synthesis if it has to use the points beyond the Catapult library characterization range.

3.1.4. Determining Licensing Approach

To ensure a successful library characterization run, you should determine your tool licensing needs and verify their availability as follows:

- **Library Builder** — Full library characterization may take days using a single Library Builder license or overnight using a Library Builder Farm license, so start testing small.

  If possible, use the Library Builder Farm option with multiple machines or to connect with a Load Sharing Facility (LSF). Make sure paths and licenses are set up correctly in your shell startup files (eg. .cshrc), so they are found when running on remote machines. For more information, refer to Using the Library Farm.

- **RTL Synthesis Tools** — RTL synthesis tools often require more than a base license to enable the retiming needed to characterize pipelined multipliers, so you should characterize one pipelined multiplier to verify the proper license is available.

Licensed DesignWare for Design Compiler requires a separate license, so you need to verify that it is available for characterization. The DesinWare libraries usually have .sldb extension and should be specified using the synthetic_library variable.

3.2. Creating the Library

The Library Builder provides ASIC and FPGA library templates to help you create and characterize libraries for use in Catapult.

General procedure for characterization:

1. Create a working directory where the new library will be built (a working area).

2. Invoke the Library Builder and set the working directory within the tool. See Setting the Working Directory.

3. Create a new library from one of the supplied templates. Library Builder provides two methods:
   - Creating a New Library with the Library Characterization Wizard (requires technology library in Liberty format)

- ○ Creating a New Library manually (not recommended)
- ○ Command line library creation

4. Specify the path to technology library using the **Tools > Set Options** > Component Libraries in the Catapult.

5. Characterize the library as described in .Characterizing Libraries or Components

6. Save the library. See Saving Libraries.

3.2.1. Setting the Working Directory

The Library Builder works with the files, in a working directory, which you create before you invoke the tool. This is also the place where all generated output files are placed.

1. Click on the **Set Working Directory** task in the Task Bar window or use the **File > Set Working Directory...** menu item to open file system browser.

2. Browse to the directory that you want to set as the working directory and click **OK** to make it your working directory.
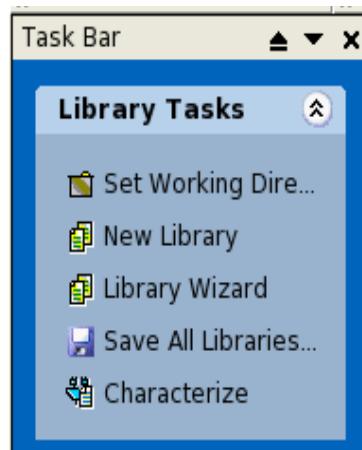
Alternatively, you can use the set_working_dir command.

3.2.2. Creating a New Library with the Library Characterization Wizard

The easiest way to get started making a new Base ASIC Catapult library for your technology is to use the Library Characterization Wizard. This wizard allows you to select one (or more) Liberty files representing the operating conditions and characteristics of your desired technology library. After selecting the target operating conditions and the downstream RTL Synthesis tool, the wizard automatically reads the Liberty file and extracts key pieces of data necessary to setup the Catapult Base Library.

Procedure for creating a new library using this wizard:
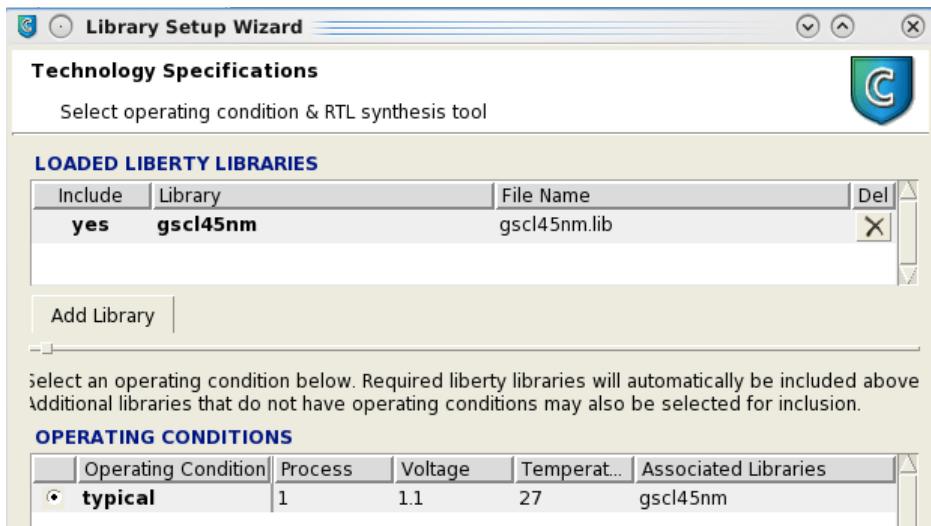
1. Click the **Library Wizard** task in the *Task Bar* window.

Illustration 4: Library Tasks

2. In the Library Setup Wizard, click on the **Add** button and navigate to your Liberty file(s). Make sure you add all Liberty files in your technology library. In the screen shot below, the file *gscl45nm.lib* was

added:
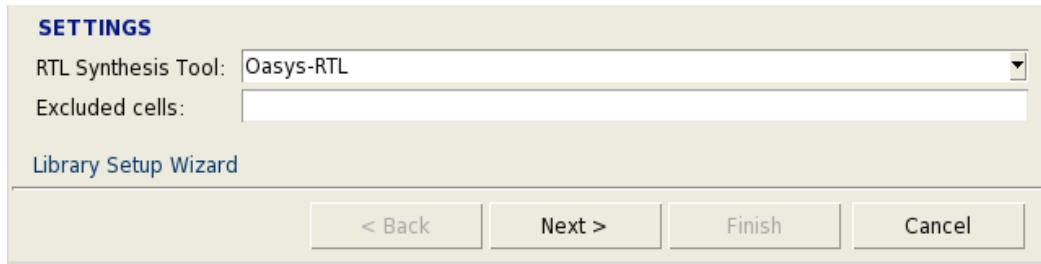
Illustration 5: Library Wizard: Loaded Liberty Library

3.   Select the Operating Conditions for the new Catapult Library by checking the radio button next to the operating condition name. For reference, the Process, Voltage and Temperature for that operating condition are extracted from the Liberty file and displayed in the table.

Note that the set of loaded libraries containing the selected operating conditions are selected (marked yes in the Include column). At this point you can also include the libraries that do not have any operating conditions specified. NOTE: The wizard does not allow you to select libraries that have operating conditions specified, but lack the selected operating conditions.

4.   Select the downstream RTL Synthesis tool:

Illustration 6: Library Wizard: Selecting Operating Conditions and RTL Tool

5.   Specify the cells to be excluded – dont_use attribute will be set on those cells. Cells specified here will not be considered as driving cells or external load. On this page the excluded cells have to be specified in the <library name>/<cell name> format. The wizard will check the specified cells against the technology library and only the cells found in the selected libraries will appear in the Excluded cells box on the next page.

6.   Click **Next**
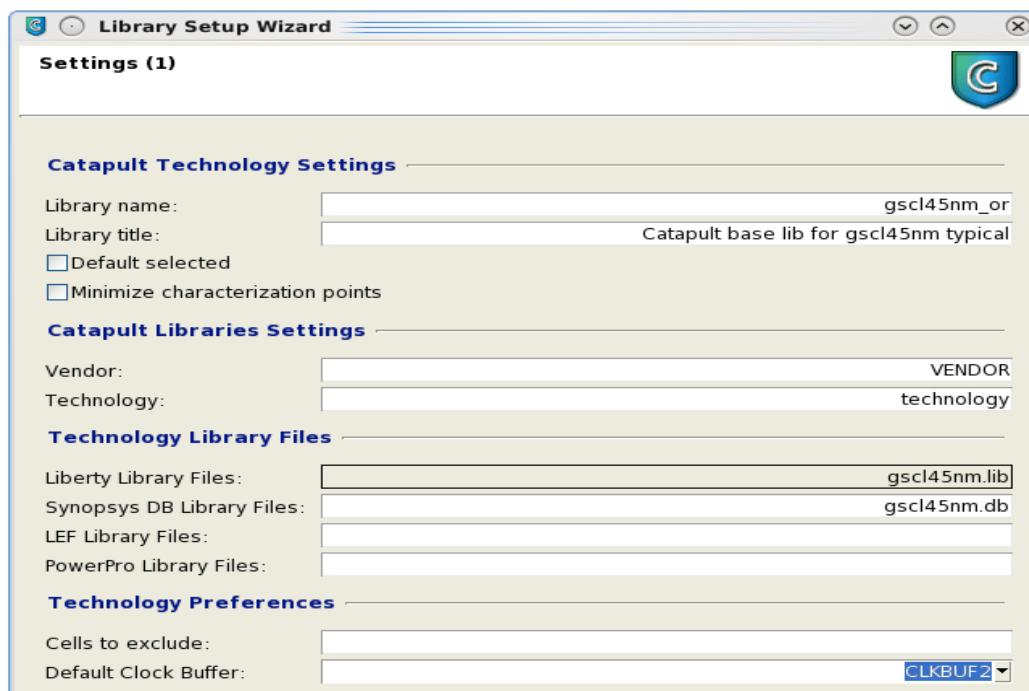
Now the wizard selected the appropriate defaults for setting up a Catapult Library for characterization from the libraries you selected and taking into account the selection of the RTL synthesis tool selection and excluded cells. These defaults are then shown on this and the following wizard pages to enable you to review them and make changes if necessary.

The first **Settings (1)** page allows you to name the library and control how it is displayed in Catapult. The *Technology*, *Library name*, *Library title* and *Vendor* fields on this page are arbitrary names that are only used by Catapult. By default, Library Builder derives the name based on the Liberty file and the selected synthesis tool. In this example, the library name is *gscl45nm_or* (eg. characterizing the gscl45nm using Oasys RTL).

This page also displays the Liberty files you selected on the previous page and allows you to specify the rest of files associated with this technology (.db, .lef etc). Note that for example .db files are required if you selected DesignCompiler as your RTL synthesis tool.

The Minimize characterization points box, if selected will create the library with the minimal set of characterization points. This approach usually reduces the characterization time but may affect the correlation. Refer to [Reducing Characterization Time,](#) for more information.

In the *Technology Preferences* section, you can provide additional information to assist with characterization and downstream flows. If you RTL synthesis flow excludes any cells from the technology library, then you should specify the cell name in a space-seperated list in the **Cells to exclude** field. Catapult Ultra allows you to optimize your RTL netlist using the PowerPro engine. In this flow, Catapult builds a clock tree for timing estimation, you can specify the **default Clock Buffer** for this flow.
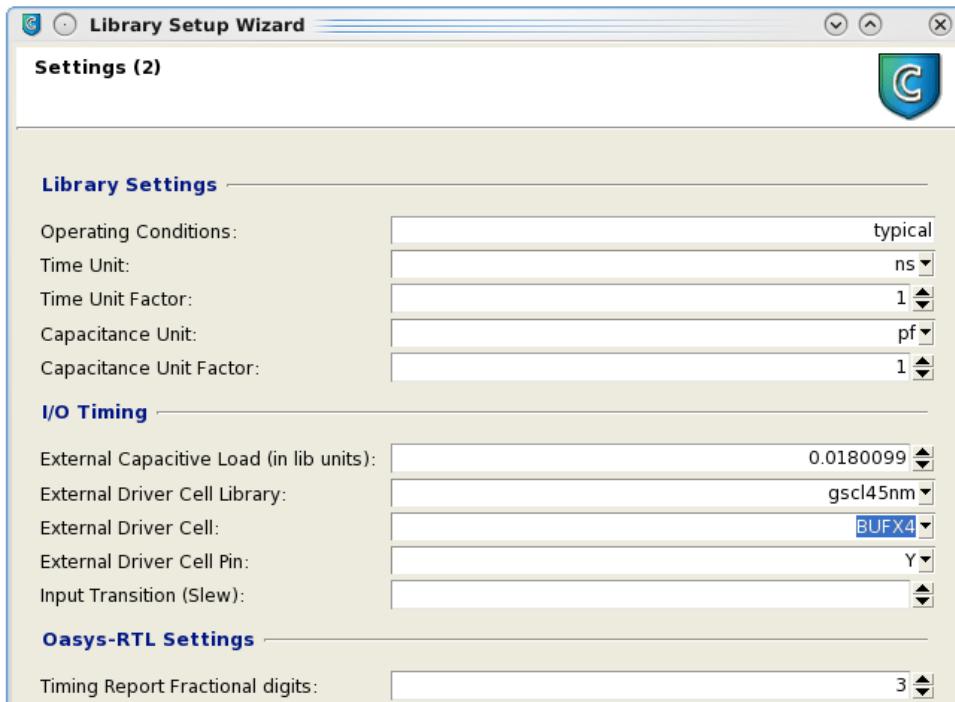

Illustration 7: Library Wizard: General Settings

7. The second **Settings (2)** page allows reviewing Basic technology settings including the operating conditions, physical units, timing and driving cell information, etc. The External Driver Cell field will pre-populate the a list of registers and buffers from the liberty file. You should select a cell that represents the typical loading of an operator in your designs.

    NOTE*: It is important that these values match the values used in your RTL Synthesis scripts. For more information on these fields, refer to the [Basic RTL Technology Settings](#) section.*
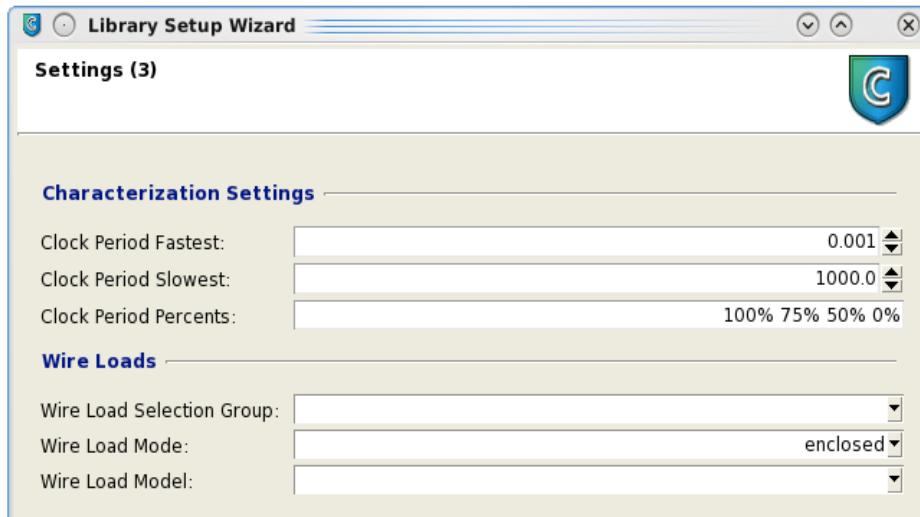
    The <RTL Synthesis Tool> Settings section allow you to specify library specific options that are supported in the corresponding automated RTL Synthesis flow in Catapult. The number of available

setting will vary depending on the RTL synthesis flow.

Illustration 8: Library Wizard: Library Settings

8. The third **Settings (3)** page has default values for driving the characterization process (eg. range of clock periods) and the wire load mode.

Illustration 9: Library Wizard: Characterization Settings

If you are satisfied with the settings, click Finish to complete the process and create the Catapult Library. You can now proceed to the library characterization process described in Characterizing Libraries or Components.

3.2.3. Creating a New Library manually

NOTE: It is highly recommended that you use the [Library Wizard](#) to setup the library characterization process in Library Builder.

Catapult Library Builder provides templates for the various types of libraries you can create. The Base ASIC Library template contains all of the basic components needed for synthesis. There are various templates for creating ROM, Custom RAM and RegisterFile libraries. Finally, the Blank Library template is an empty library that you populate with custom component modules and operators.

Base ASIC libraries should be created using the Library wizard. When manually creating a base library you would have to enter all the data set up by the wizard manually. This method is not recommended since no checking/suggesting is done by Catapult thus making this process more error prone. This method should only be used if the technology file in Liberty format is not available. Note that some of the library variables created by the library creation wizard cannot be entered through dialog boxes. Such variables must be created after the library creation as described in *[Editing Parameters](#)*.

The procedure for creating a library by using the GUI is as follows:

1. Click the **New Library** task in the Task Bar window, or select the **File > New > New Library** menu item to open the Library Creation dialog box.

2. In the Library Creation dialog box, select the synthesis tool to be used for characterization. For ASIC libraries, choose either **Synopsys Design Compiler** or **Mentor Graphics Oasis-RTL**.

3. Select the library template under the selected synthesis tool.

    Fill in the fields on the right side of the Library Creation dialog box. Items in red are required. If you have any questions about any of the entries, refer to the online Help for each form by clicking the **Help** button.

4. Click OK. The new library is created in the Library Builder database but not saved to disk until you explicitly save it.

5. Edit the library if necessary. In the Library Explorer window, double-click on the newly created library to open the Library Editor. (or right-click and select **Edit** from on the pop-up menu).

3.2.4. Command line library creation

NOTE: It is highly recommended that you use the [Library Wizard](#) to setup the library characterization process in Library Builder.

Library wizard and the dialog boxes are two ways of specifying the data to set up the library. Both of them eventually compose and call the library creation command with the library setup passed using command switches.

The library creation command [flow run](#) launches the *library add flow* for the specified RTL synthesis tool. Library Builder provides flow packages for each of the supported synthesis tools.(eg *DesignCompiler*). Run the flow for the target synthesis tool and specify a library template and its required settings. The flow will add the library to the Library Builder database.

The command format is:

```
flow run /<package>/library add <lib_template> <options>
```

For example, the following command creates a Base ASIC Library (base) for the Design Compiler tool.

```
flow run /DesignCompiler/library add base \
```

```
        -libname my_lib \
        -libtitle my_lib_title \
        -vendor Sample \
        -technology 180nm \
        -libs_liberty sample_180nm.lib \
        -libs_db sample_180nm.db
```

The set of valid options varies depending on the type of library template. To get a list of the valid template names for a flow package, use the "-help" switch as follows:

```
 flow run /<package>/library add -help
```

Similarly, to see the set of valid options for a particular library template, use the following command:

```
 flow run /<package>/library add <lib_template> -help
```

Some customers preferred the script based approach as opposed to the manual GUI based library creation. For base library creation we discourage this practice because new library settings get added from one release to another thereby invalidating the scrips or making them incomplete. The biggest problem with this approach is that no checking against the technology library is done making the process error prone.

3.3. Characterizing Libraries or Components

Catapult characterization is a process of synthesizing the components in a library using the RTL synthesis tool of choice in order to obtain their area and timing characteristics. The synthesis is performed by the RTL synthesis tool and the area/timing characteristics are then saved with the components in the library. Components in the Catapult *Base ASIC Library* template are characterized at multiple performance points, producing a robust library with which Catapult can deliver highly accurate area estimates.

NOTE: The library characterization process uses one of the supported RTL synthesis tools to synthesize a sufficient number of the qualified modules (QMODs) in a library in order to obtain a representative sample of the area and timing characteristics of the QMODs. Based on the actual measured values in the sample data set, the Library Builder uses curve fitting algorithms to estimate the characterization values for all other QMODs. Each QMOD is measured at various clock speeds based on the multi-point characterization settings for the library. Refer to [Determining Characterization Points](#) *for more information.*

A *module* (MOD) is a library component implementing one of the operators available to the scheduler. For example the MODs mgc_mul and mgc_mul_pipe (multiplier and pipelined multiplier) implement the functionality of the multiplication operator MUL.

A QMOD is module with all characterization parameters explicitly specified. For example mgc_and (1,2) is a qualified configuration of mgc_and with one output and two inputs (a single AND2 gate).

Library characterization consists of the following steps:

1. In the Library Explorer window, select the items to characterize. If the top level is selected, the entire library is characterized. Expand the tree to select individual MODs, QMODs or QMOD datasets.

2. Spot Checking Library Settings

3. Launch the characterization process on the selected items. See Running Characterization.

4. After characterization is complete, use the Plot window, reports and transcript files to view and analyze the characterization data and resolve any failures.

5. If necessary, modify the interpolation equations of individual MODs to refine the correlation between the estimated characterization values and the measured values obtained from the RTL synthesis tool. Then recharacterize MODs. Refer to Evaluating Library Characterization Results for more

information.

6. If errors occurred during characterization, use the characterization transcript to diagnose the problem. Refer to <u>Troubleshooting Library Failures</u>.

3.3.1. Spot Checking Library Settings

After the library is created we suggest spot checking the library before running full characterization:
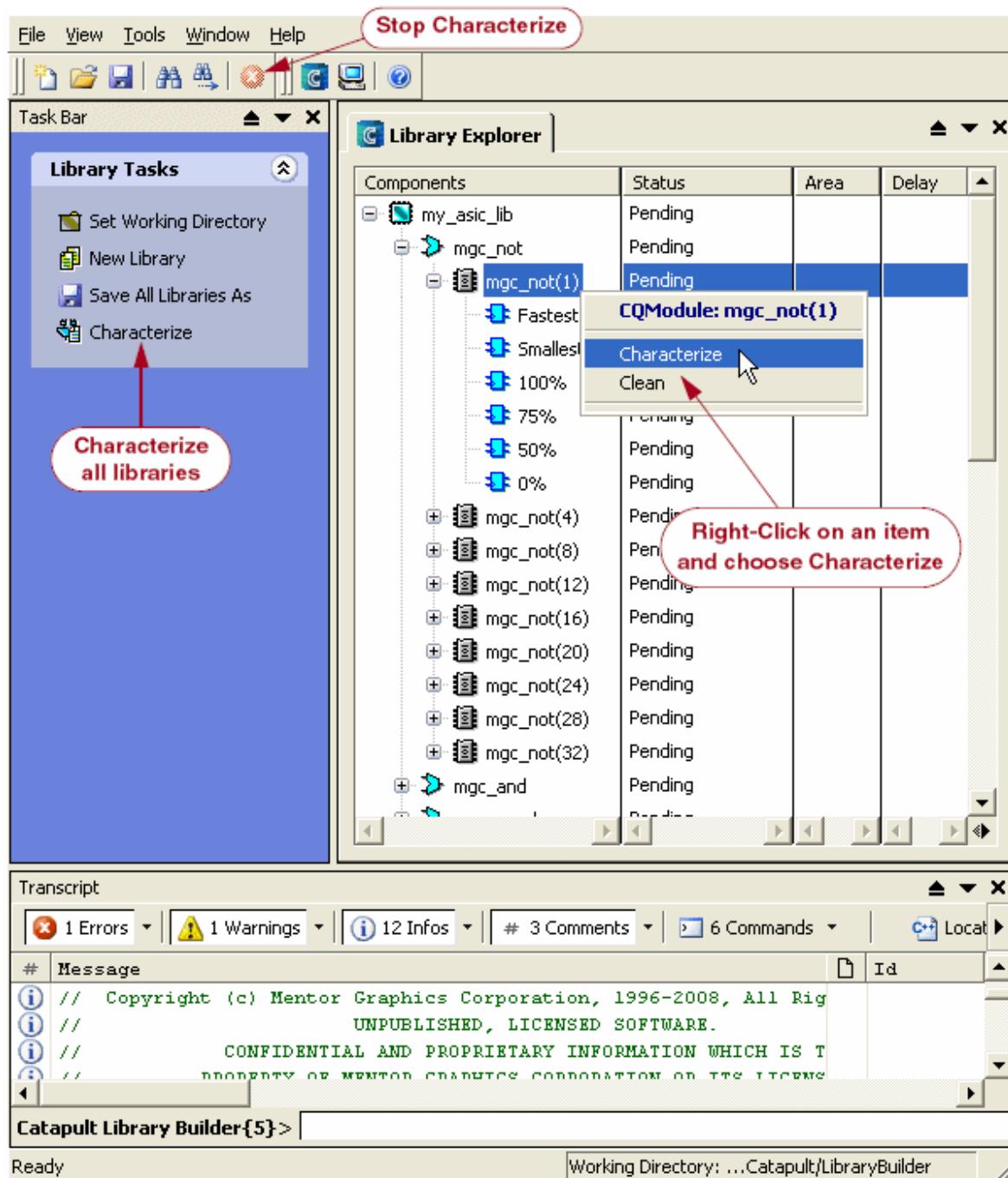
- Characterize a single inverter first. Make sure that, for the Fastest component, the reported Slack is negative. If it is not reduce the value of the characterize_clock_period_fastest variable.

- Characterize the largest multiplier and verify that the Smallest component has positive Slack reported. Otherwise increase the value of the characterize_clock_period_smallest variable.

- Characterize a pipelined multiplier to verify that retiming works properly.

3.3.2. Running Characterization

During the characterization process, the Catapult Library Builder commands are displayed in the Transcript window, giving you a record of the entire session. You can scroll through the transcript and print or save it.

Once the library is loaded into the Library Explorer window, right-click on the object you want to characterize and choose the *Characterize* command. The characterize command operates on all objects hierarchically contained under the selected object. For example, if a module is selected, each qualified module it contains will be characterized. Similarly, if the library object is selected, the entire library will be characterized.

Select *Characterize* on the Task Bar to characterize all libraries in the Library Explorer window. Illustration 10 shows how to initiate the characterization command.

Illustration 10: Characterizing the Library or Component

As the characterization process runs, it displays information about its progress in the Status column of the Library Explorer window, as well as in the Transcript window. A full transcript of the characterization task is automatically saved in the project directory.

To stop a characterization job in progress, click on the **Stop** button on the tool bar. It usually take a minute or so for the job to terminate. After it terminates, the component(s) that were being characterized will have a *Failed Generate* status.

When characterizing a large number of components, the Library Builder queues a series of characterization tasks, and each task will contain a multiple components. The maximum number of components that can be

grouped into a task is configurable. To adjust the default setting, edit the ***Maximum Components Per Task*** option (Refer to Set General Options. **Tools > Set Options >** General).

Illustration 11 shows an example of the how the Library Explorer window might appear while a characterization job is in progress. For this example, all of the *mgc_equal* components are being characterized, and the *Maximum Components Per Task* option has been set to four. The figure shows the first group of four components is in progress, indicated by the *Generating* status. The remaining *mgc_equal* components have a *Queued* status, and will be processed in turn.

Illustration 11: Characterization Status Information
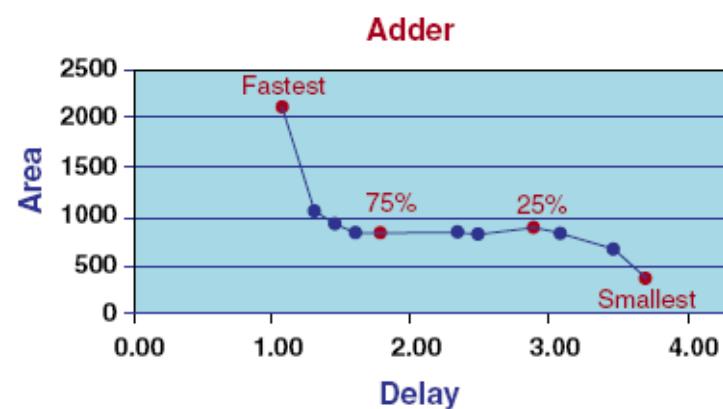
Illustration 11 also shows some of the other status values for characterization, such as *Passed*, *Pending* and *Failed Generate*. The following table defines the meaning of the status values that may appear in the **Status** column:

Status	Description
Pending	Component currently has no characterization data and is not currently queued for characterization.
Queued	Component is in the queue for automatic characterization.
Configuring	Initial state of the automatic characterization process which configures the design constraints and generates the characterization task script.
Generating	Characterization task is running. The component is being synthesized and the resulting area and timing characteristics are being collected.
Passed	Characterization task completed successfully and area/timing data was obtained.
Failed Analyze	An error was encountered when parsing the transcript to collect timing and area estimates for the design.
Failed Generate	Characterization task returned a non-zero exit status or did not obtain timing and area information.
Failed Invocation	The downstream synthesis tool failed to invoke.

Table 7: Component Characterization Status

3.3.3. Determining Characterization Points
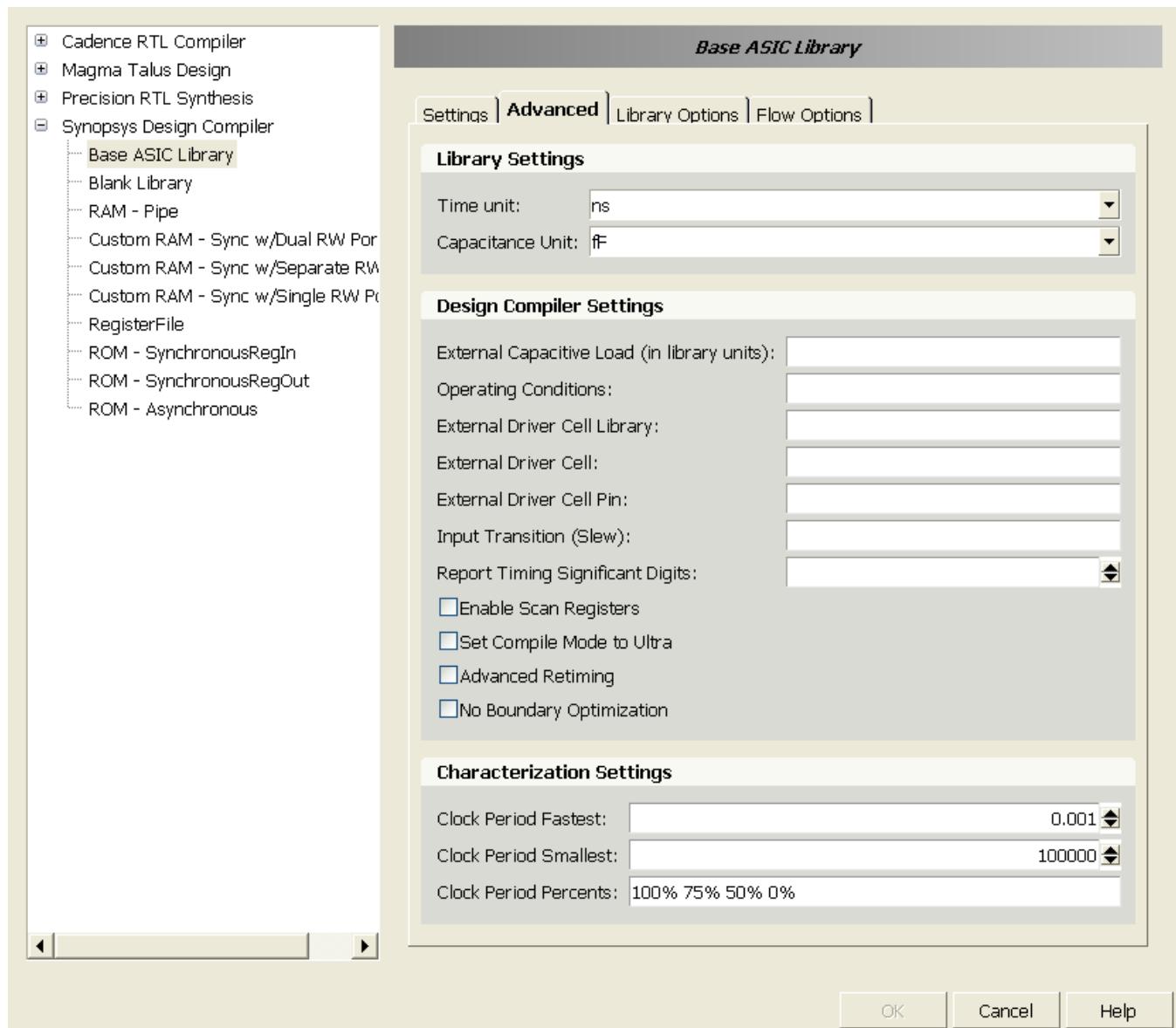
During the characterization process, Catapult Library Builder collects multiple sets characterization data (area and timing) for each component. You can think of each data set as a point along an area versus delay graph of the component. For example, Illustration 12 shows how the graph of a 16-bit Adder might look.


Illustration 12: Adder Characteristics Graph

Typically the characteristics of the two end points (Fastest and Smallest) are rarely used in designs. So the Library Builder allows you to specify multiple points between the end points at which area and timing data will be collected. The target points are specified as percentages of the range bounded by the fastest and smallest points.

By default, four data sets are collected and their distribution is: 100%, 75%, 50% and 0% (*Fastest* = 100% and *Smallest* = 0%). You can specify a different number of data sets and/or different percentages when creating a new library. Library Builder limits the number of data sets that you can specify to 15 values. You can edit the data set values in the *Clock Period Percents* field of the *Advanced* tab in the Library Creation dialog box as shown in Illustration 13. Refer to Creating a New Library manually for detailed information.

Illustration 13: Multi-Point Characterization Settings on Library Creation Dialog

The characterization process works as follows for each component:

- Obtain fastest data set:
  Run the synthesis tool with a target delay constraint specified by the Clock Period Fastest setting. (default is 0.01 ns.)

  NOTE: The clock periods must be able to generate the fastest design, i.e. the period must be small enough to constrain a simple logic gate such that it does not meet timing. If the tool is able to easily meet timing the curve described on the previous page will not be accurate.

- Obtain *smallest* data set:

  Run the synthesis tool with a target delay constraint specified by the *Clock Period Smallest* setting. (default is 100,000 ns.)

- Obtain each intermediate data set:

  Use the actual delay values obtained from the fastest and smallest characterizations to calculate the target delay constraints to be used for the intermediate characterization runs. Then run the synthesis tool for each intermediate point. Intermediate points are specified by the *Clock Period Percentages* setting.

For ASIC libraries, the Library Explorer window displays *Status*, *Area*, *Delay*, *Clockperiod* and *Minclkprd* columns. The *ClockPeriod* column is the clock period to which the design was constrained for characterization. The *MinClkPrd* column is for sequential components, this includes registers. The minimum clock period is the fastest time that the component can be scheduled. The delay is the maximum, input to register, or register to register delay. The Delay column for these components refers to the final register to output delay. The *Minclkprd* column will not appear until a pipelined component is selected in the *Components* column.

When you begin using your library in Catapult, if an operator in a design contains a parameter with a value that lies outside the range within which the corresponding component was characterized, Catapult will list the operator in the *adjust_lib_char.tcl* file in the Catapult solution directory.

3.3.4. Queuing Multiple Libraries for Multi-day Run

You can also set up a queue of libraries to characterize as follows:

1. Create all of the new libraries using the steps previously described in this section.
2. Click the **Characterize** task on the *Task Bar* to run the characterization process on all open libraries.

3.4. Evaluating Library Characterization Results

If the library characterization passes, the status is set to *Passed* and the area and timing results display in the Library Explorer windows shown in Illustration 14.

This component was characterized using four-point characterization (default setting). So the display shows separate line items for each target point, plus the fastest and smallest points. Refer to Determining Characterization Points for information about configuring multi-point settings.

Illustration 14: Characterization Passes

3.4.1. Delay Characterization Properties

After a successful characterization of a library/component, the Library Explorer window displays delay property values for each data set. These delay properties reflect the maximum values, regardless of the path. Depending on the component type, the following values display:

- **Delay** — If sequential, this value is the register to output value. If combinational, this value is the total input to output delay.
- **ClockPeriod** — Clock period constraint.
- **MinClkPrd** — Minimum clock period is the fastest time a pipelined component can be scheduled. It is either InputDelay + InputSetup, R2RDelay + R2RSetup, or Delay, whichever is largest.
- **Slack** — If sequential, this value is ClockPeriod minus MinClkPrd. If combinational, this value is ClockPeriod minus Delay.
- **InputDelay** — Input to register delay.
- **InputSetup** — Input register setup time.
- **R2RDelay** — Register to register delay.
- **R2RSetup** — Register to register setup time.

Illustration 15 shows how these timing delay values are determined.
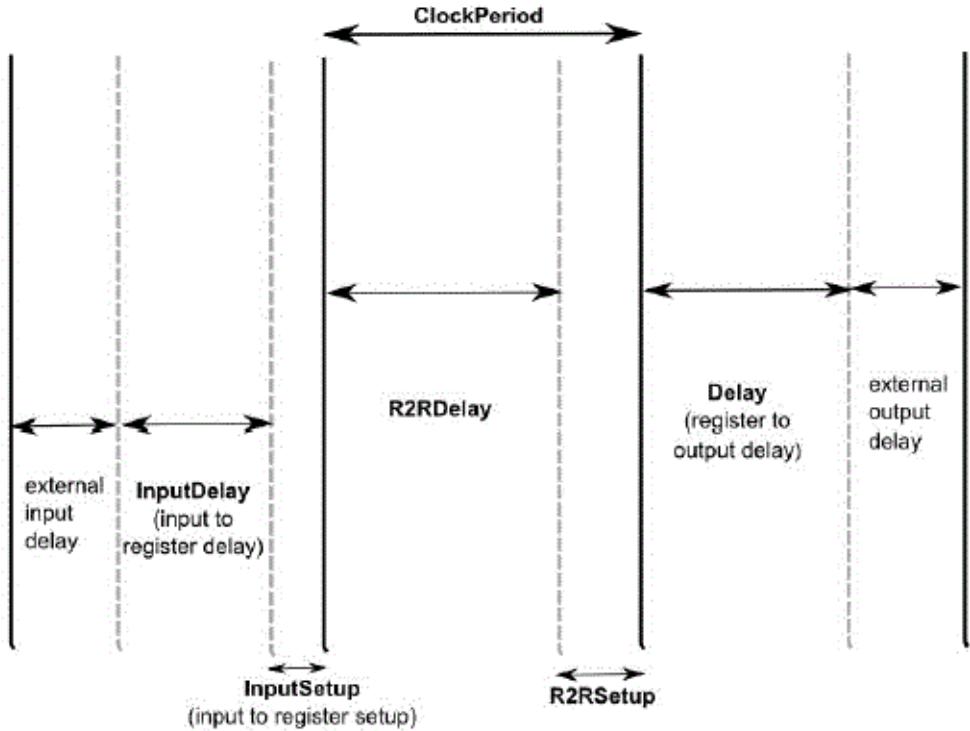
Illustration 15: Sequential Timing Delay Diagram

3.4.2. Resetting the Data Before Another Characterization

Before you can rerun characterization on a library that has passed characterization, you must clear the values. Right-click on the target object and select **Clean** from the popup menu. However you can recharacterize a single component without having to clean it beforehand.

Once the values are removed from the Area and Timing fields, right-click on the target object and select **Characterize** from the popup menu.

3.4.3. Plotting the Characterization Data

The Plot window provides a graphical view of the interpolated and measured characterization values for a user-specified range of qualified modules (QMODs) in a library module (MOD). The user interface of the Plot window allows you to dynamically plot different ranges of QMODs by selecting different combinations of properties and parameter values. It also allows you to add, remove and characterize QMODs.

As shown in Illustration 16, the Plot window is accessed from the Library Explorer window by right-clicking on a MOD and selecting Plot from the popup menu. The main area of the Plot window contains a graph of the characterization data. The red and gold circles on the graph represent characterized QMODs, their *measured* values obtained from the downstream RTL synthesis tool. The blue circles are estimated characterization values for valid QMOD configurations that do not yet exist in the MOD. The gold color circle indicates the QMOD whose measured value deviates the most from the Library Builder estimated value for it. In other words, the gold QMOD has the largest margin of error (Max Error) relative to all other measured QMODs on the graph.
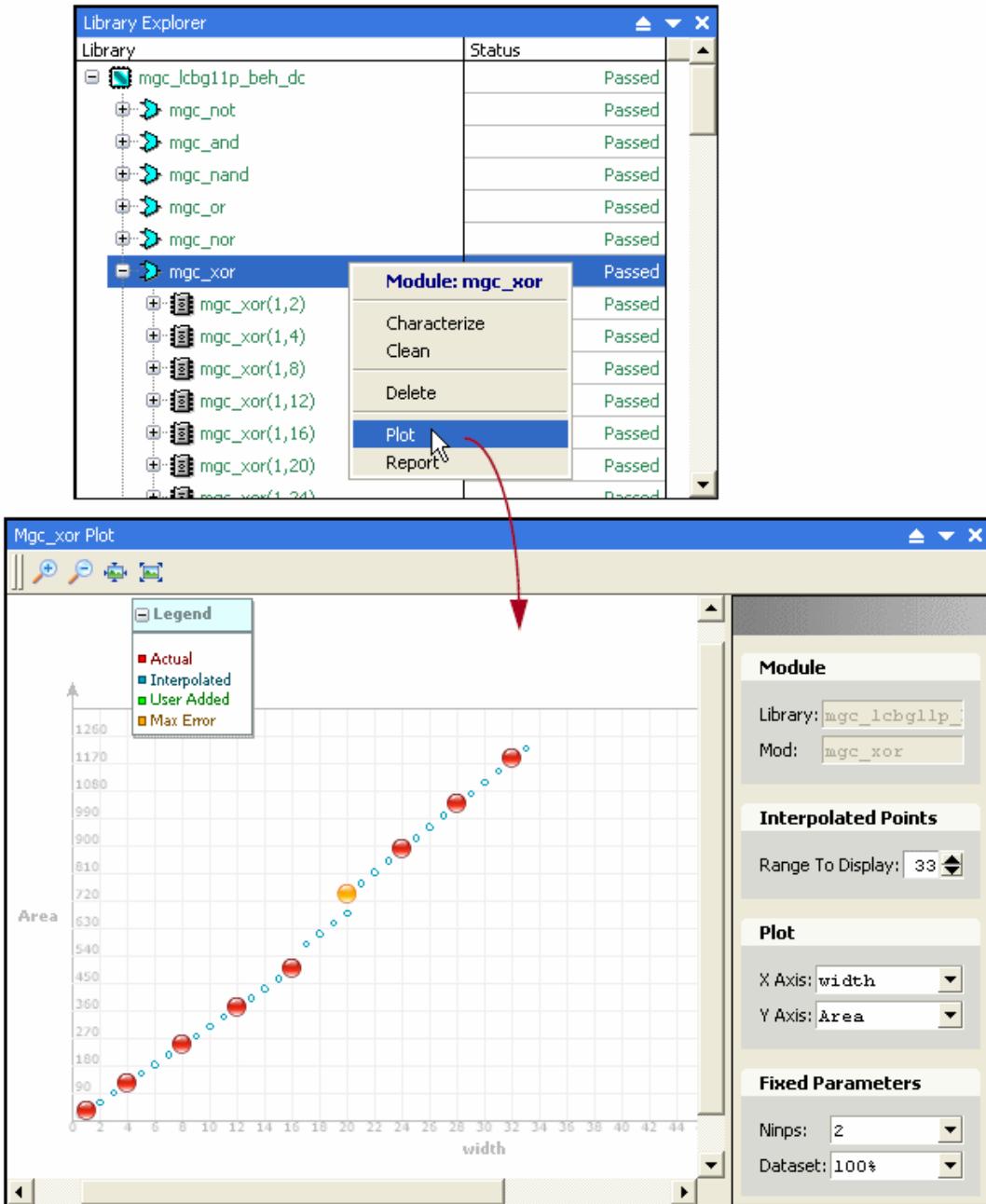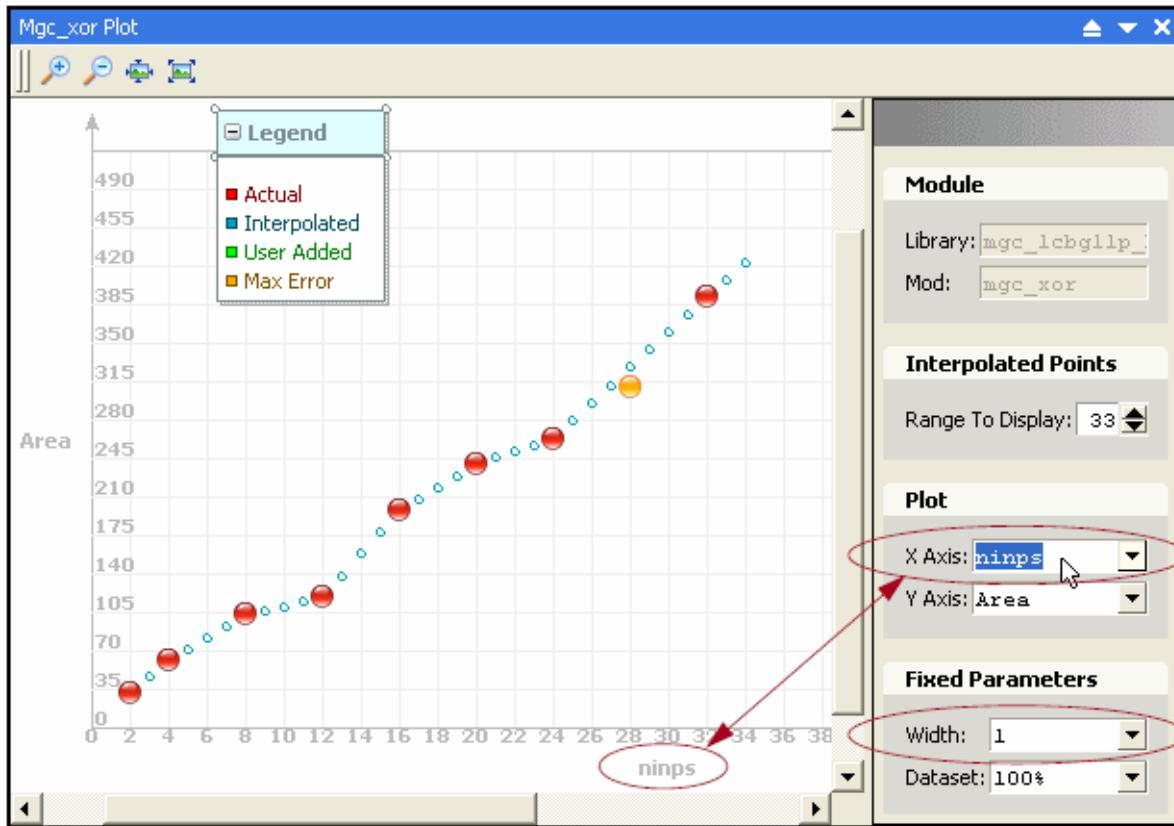
Illustration 16: Opening a Plot Window for the mgc_xor Module

The Y-xis is the range of values for one of the MODs characterization properties. The X-axis is the range of valid values for one of its parameters. All other parameters are set to fixed values within their respective valid ranges. By default, the first property listed in the MOD's All bindings is the initial Y-axis setting, and the first parameter listed is the initial X-axis setting. Similarly for each fixed parameter, the first valid value in its range is the default setting.

Use the interface panel on right side of the window to plot different configurations. For example, using the drop-down menu of X-Axis field, select the *ninps* parameter. The plot automatically updates to graph *ninps*

on the X-axis and *width* as a fixed value, shown in Illustration 17.

Illustration 17: Updated Plot with Ninps along X-Axis

In Illustration 17, of all the QMODs in *mgc_xor* MOD, only the following nine measured QMOD configurations are plotted because they are the only ones that match the *Plot* and *Fixed Parameters* settings. The first parameter, width, is fixed at the value 1, and all values of the *ninps* parameter are plotted along the X-Axis:

```
mgc_xor(1,2)              mgc_xor(1,20)
mgc_xor(1,4)              mgc_xor(1,24)
mgc_xor(1,8)              mgc_xor(1,28)
mgc_xor(1,12)             mgc_xor(1,32)
mgc_xor(1,16)
```

The *Range To Display* field allows you to change the number of estimated data points (blue circles) that are plotted. Generally, the *Range To Display* number should be close to the X-axis value of the highest measured QMOD. For example, in Illustration 17, the Range To Display is 33, and the highest *ninps* value in the range of QMODs is 32 (mgc_xor(1,32)).

3.4.4. Adding/Removing QMODs

You can use the Plot window to add new QMODs to the MOD by right-clicking the blue circles and selecting **Add QMOD** (or *Add and Characterize QMOD*) from the popup menu. The new QMOD will be immediately added to the library and have the same parameter values as the selected blue circle. If it is added but not characterized, it is represented on the graph as a green vertical line (unknown Y-Axis value). After it is characterized, it will become either a red or gold circle.

Illustration 18 illustrates the procedure by adding the new QMOD mgc_xor(12, 28). The Library Builder automatically updates the QMOD list in the Library Explorer window and in the module's *Qmods* section in the Library Editor window.

To remove a QMOD, right-click on it and choose **Remove QMOD** from the popup menu.
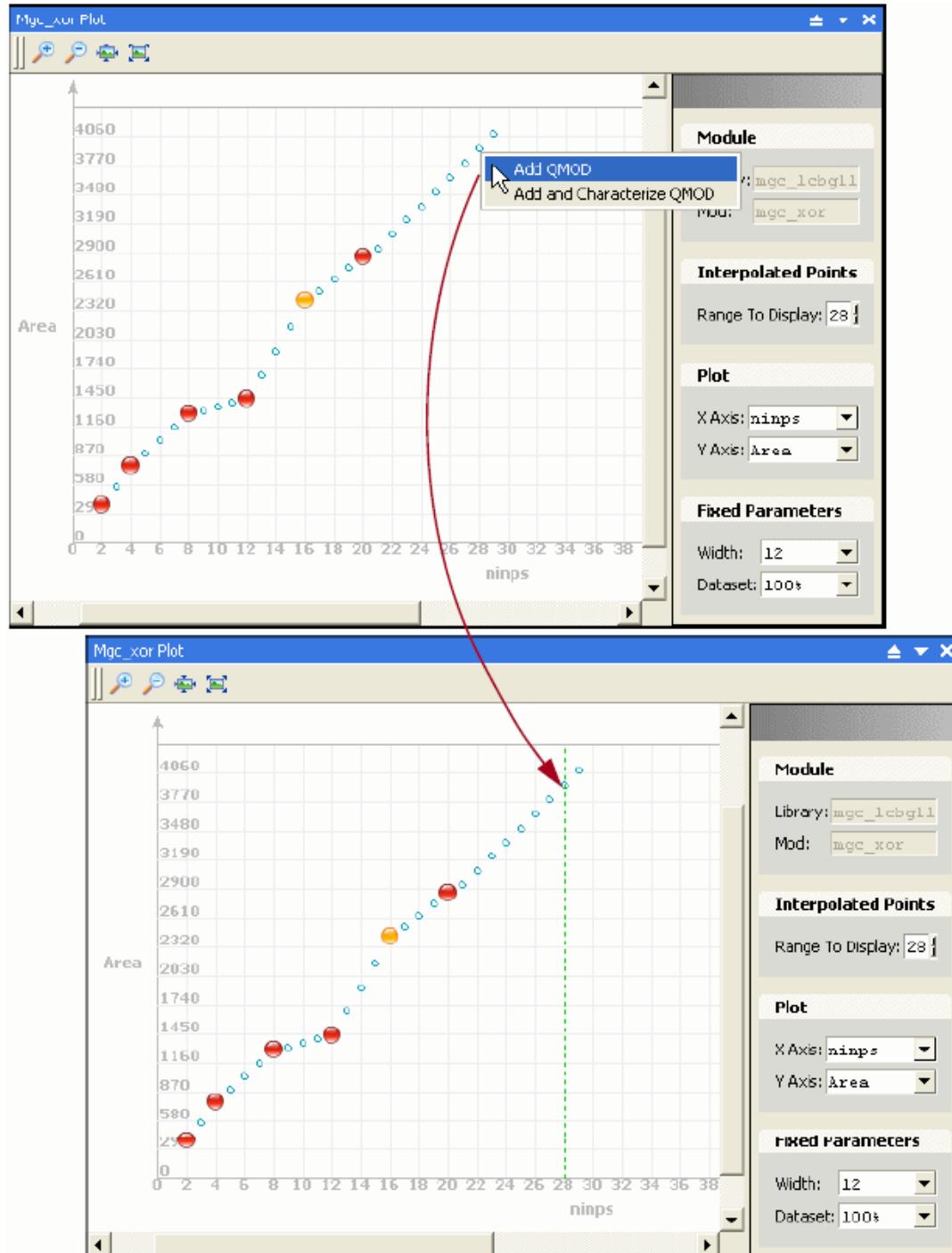
Illustration 18: Adding QMOD from the Plot Window

3.4.5. Viewing the Characterization Transcript

Once the library characterization starts, progress and commands being run are displayed in the Transcript window. Double-click on a component (or right-click and select View Transcript as shown in Illustration 19) to see a transcript of the commands and tasks performed by Catapult Library Builder.

If you run a *grouped* job, the transcript will collapse certain sections so that only the most relevant part are visible. It may be desirable to see others by expanding the +/- hierarchy points, or use the expand all.
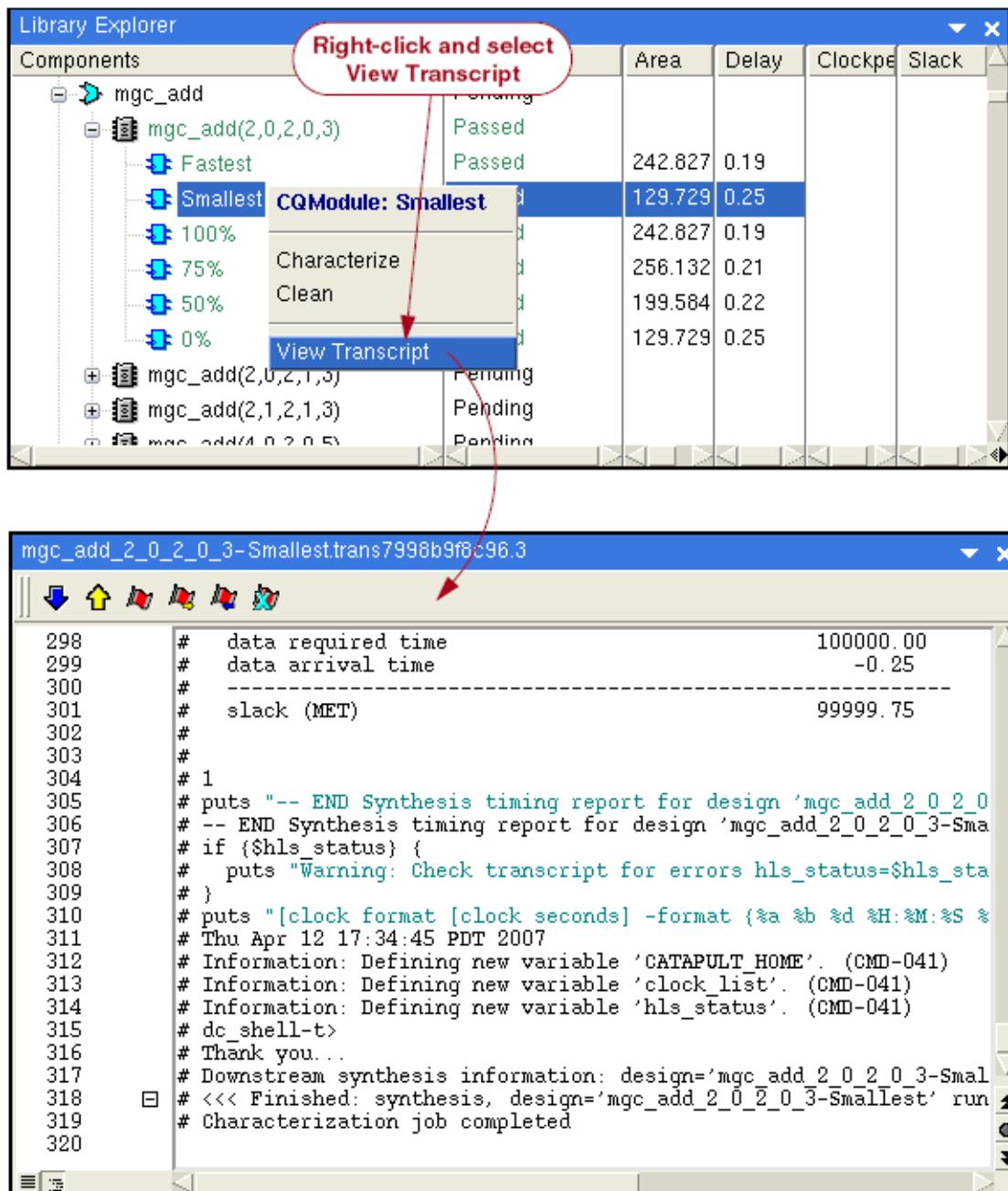
Illustration 19: Viewing the Characterization Transcript

3.4.6. Reducing Characterization Time

The characterization process can be very time consuming when a large number of QMODs are synthesized for each MOD. In general, the more data points (QMODs) that are measured, the greater the accuracy of the estimated (interpolated) area and timing data for the non-measured QMODs. The Library Builder's rapid characterization option allows you trade-off some amount of accuracy for faster characterization run times.

The rapid characterization method only measures the minimum number of QMODs necessary to achieve a user-specified error threshold (MaxError). The rapid process is iterative. It starts with the absolute minimum number of QMODs needed to construct an interpolation equation that defines the curve shape for the MOD. If after characterizing the initial set of QMODs the interpolation error exceeds the MaxError threshold, another QMOD is added and characterized in order to improve the interpolation accuracy. Additional QMODs are added until the MaxError threshold is achieved or the maximum number of QMODs have been characterized.
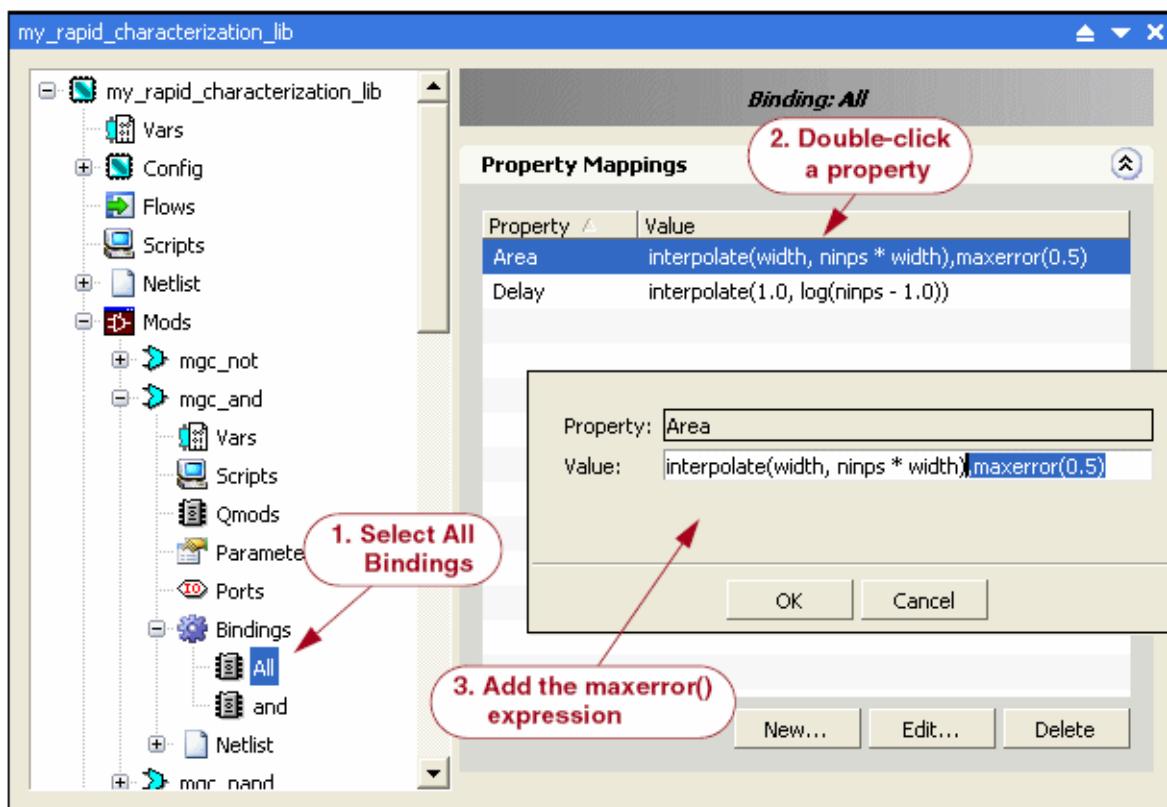
Creating a Rapid Characterization Library

The *rapid* characterization option must be set when a library is created. The newly created library contains fewer QMODs than a standard library. From the GUI, enable the **Minimize characterization points** option on the Settings tab of the Library Creation dialog box. From the command line, use -rapid switch. For example:

```
flow run /<rtl_synth_tool>/library add <lib_template> ...
                                  -rapid <true_of_false> ...
```

Setting the MaxError Threshold

MaxError threshold values are optional and are not defined in the initial rapid library. If after characterizing a MOD you determine that it needs a threshold defined, edit the interpolation expression(s) for the MOD's characterization properties and add the *maxerror*() expression.

Illustration 20: Defining MaxError for Module Property Mappings

The command syntax for editing an *All Bindings* property is as follows:

```
library set /LIBS/<lib_name>/MODS/<module_name>/BINDINGS/all/PROPERTY_MAPPING --
        -Area {<interpolation_expression>}
```

For example:

```
library set
/LIBS/my_rapid_characterization_lib/MODS/mgc_and/BINDINGS/all/PROPERTY_MAPPING -- -Area
{interpolate(width, ninps * width),maxerror(0.5)}
```

3.5. Saving Libraries

Library edits are not saved until you explicitly save the library to disk with one of the following options:

1. Select **Save All Libraries As** from the *Library Tasks* window or the *File* pulldown menu. This command saves all open libraries together as a single file. The default file name is *<synthesis_lib_prefix>_<RTL_synth_tool>.lib*.

2. From within the *Library Explorer* window, right-click on a library and select the **Save As...** menu item. The default file name is the name of the selected library.

In both cases a file system browser is provided for you to navigate to the location where you want the file saved. Place the library in the Catapult search path to make available to Catapult sessions. If you save it in an alternate location, you must add that path to the Catapult library search path (Use **Tools > Set Options > Component Libraries** pulldown menu within Catapult).

3.6. Troubleshooting Library Failures

If the library characterization fails, Library Builder displays a message about the failure. If this occurs, you can launch Catapult to further investigate the failure. Use Catapult to modify the component and synthesize it. The area and delay data obtained in Catapult can be copied into your library in Library Builder.

NOTE: This option is not available if you are using Library Farm. If you are experiencing farm failures but can't identify the issue, check that the necessary environment variables are getting passed to the farm machine.

Procedure:

1. If the *Remove Project Directories for Local Tasks* option is enabled, disable it and run the *Characterize* command on the object again.

   Library Builder always creates a Catapult project directory for each characterization task. By default, the project directories are deleted when the task is finished. Disabling this option will preserve the directory.

   To disable the option in the General Options pane (Tools > Set Options...), or use the following command:

   ```
   options set General RemoveProjectDirectories false
   # false
   ```

2. Launch Catapult by right-clicking on the failed library object in the Library Explorer window and select **Open Catapult Project** from the popup menu as shown in Illustration 22. This will open a Catapult session, load the failed module into the project and generate RTL.

3. Modify the component as needed to fix the problem and generate RTL.

4. Double-click on the *Synthesize rtl.vhdl* makefile to launch Design Compiler. Alternatively, right-click on *Synthesize rtl.vhdl* and select the *Launch DesignCompiler* command from the popup menu.

5. After obtaining the area and delay data in Catapult, use the Library Explorer window to manually enter the data for the failed object. Double-click on a data field to edit, such as **Area**, **Delay** or **Clockperiod** and enter the value. If a valid value is entered in the field, the status for the component will be changed to Passed.
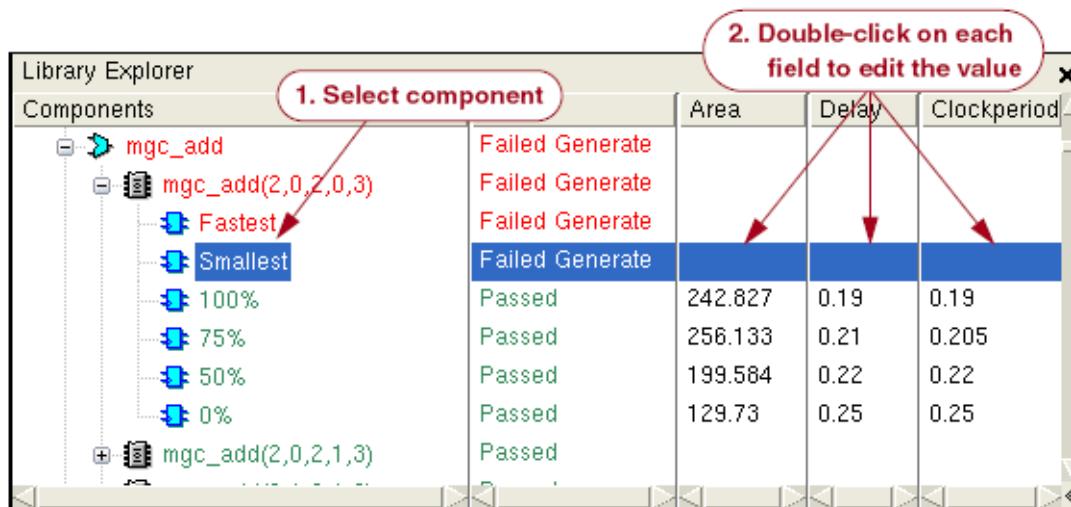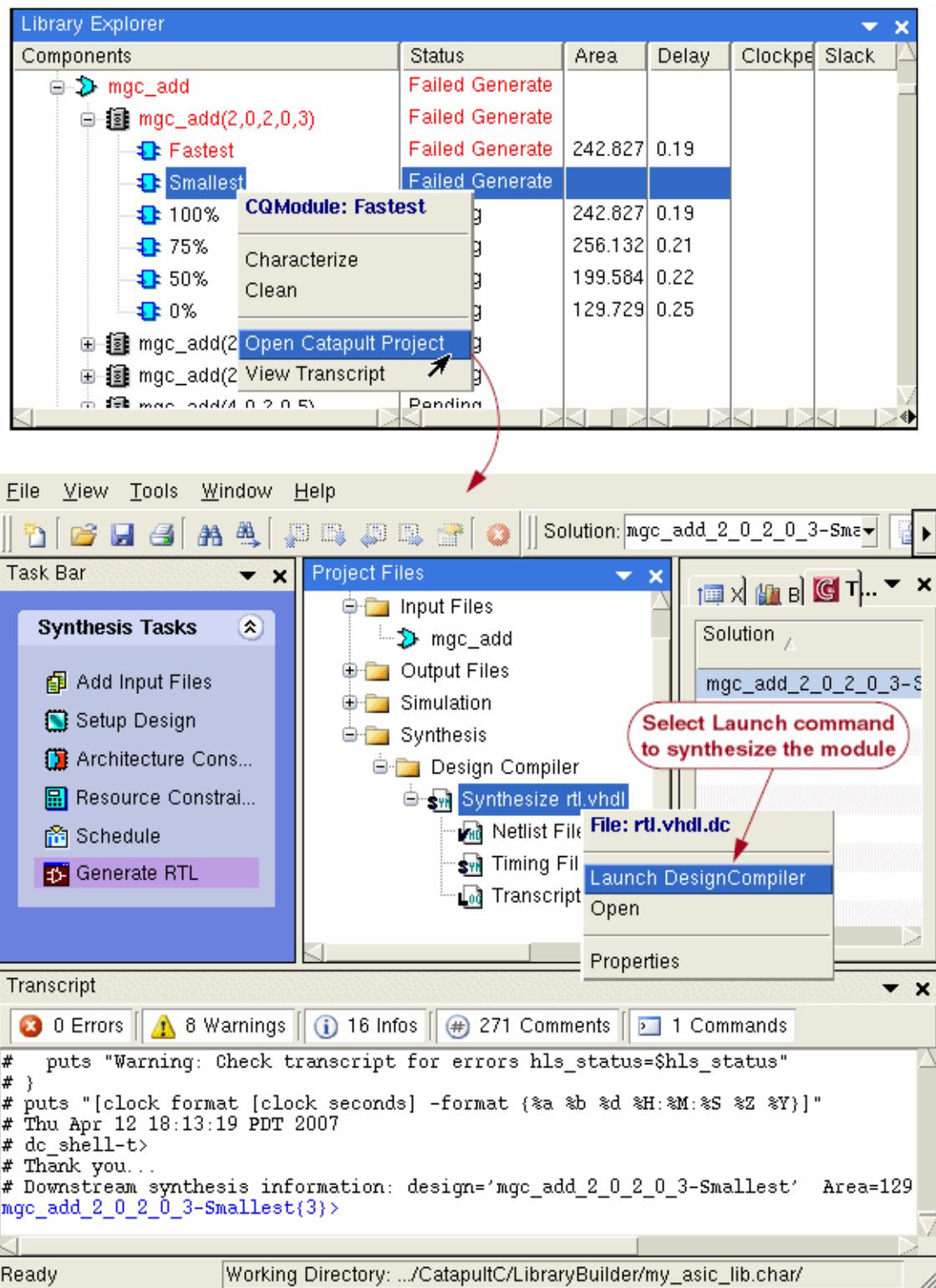
Illustration 21: Entering New Area and Timing Values

Illustration 22: Opening Failed Object in Catapult

3.7. Improving the Accuracy of Estimated Characterization Values

Using the measured characterization data obtained from RTL synthesis tools, the Library Builder uses a curve fitting algorithm to estimate intermediate area and delay values. The Library Builder generates a curve fitting equation for each characterization property (e.g. Area, Delay and MinClkPrd) in a MOD based on the *curve shape* expression for that property. The library templates define default characterization curves for each MOD property. It may be possible to achieve more accurate estimates by modifying default curve shapes. That is done by editing the property values in the library.

Other library settings also affect the number of estimated data points. The set of valid combinations of parameters is limited by parameter value ranges and the *Always* property, if set. The set of characterized (measured) combinations of parameters is limited by the specified characterization range (*dataset* parameter) and the *Always* property, if set.

3.7.1. Specifying Characterization Curve Shape

The characterization curve is specified through the multi-argument interpolate() and expand() expressions. The shape is defined as an expression of terms with the characterization parameters as variables. The interpolate() expression specifies the shape of a single curve to fit the full set of *measured* data points (obtained during characterization). The expand() expression can be used to refine the interpolate() curve by segmenting it and fitting curves to each segment.

Illustration 23 shows an example of how the interpolate() and expand() expressions are used together for the Area and Delay properties of the mgc_xor MOD.
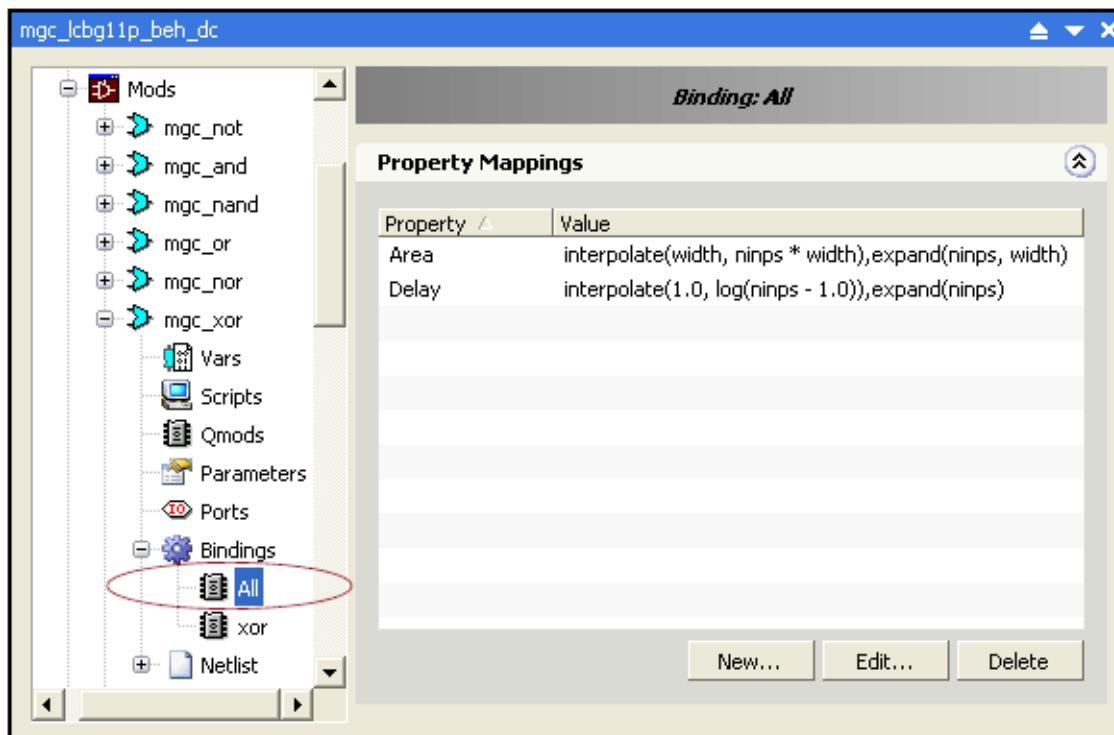
Illustration 23: Characterization Curve Shape Specifications

Syntax for the interpolate() expression:

```
interpolate(<term1>, ...)
```

where <term> is a characterization parameter for the given MOD or an expression.

Syntax for the expand() expression:

```
expand(<param>, ...)
```
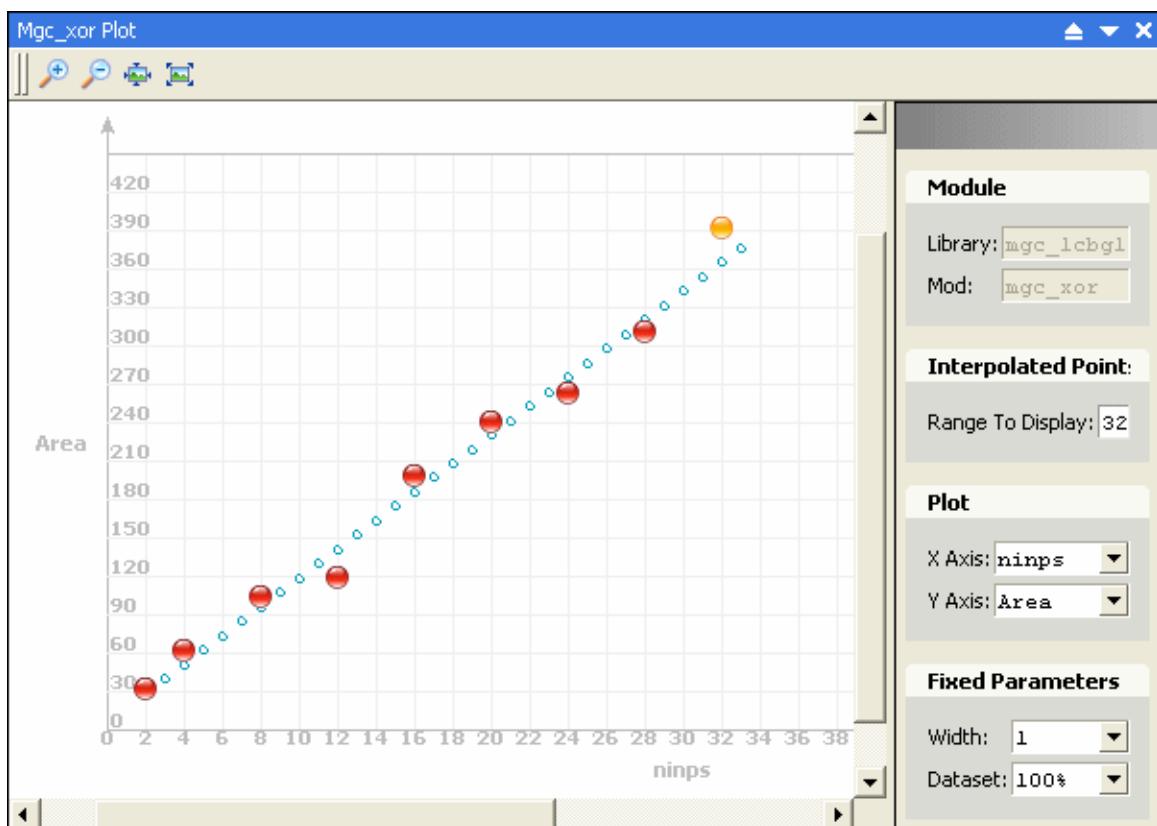
where <param> is a characterization parameter for the given MOD. The expand() expression will have different functionality depending on the kind of a parameter. Expanding a binary parameter will yield two curves, one for each value of the binary parameter.

For non-binary parameters, Library Builder will segment the interpolate() curve at its measured characterization points and fit curves at each segment, using at least *n* number of points to uniquely define the particular type of a curve (n=2 for straight line, n=3 for a parabola, and so on). The following example demonstrates how expand() can improve the accuracy of the interpolate() curve.

Illustration 24 shows a plot of the interpolate() curve for the Area property of the mgc_xor MOD. The interpolate() expression is specified by two terms:

```
interpolate(width, width*ninputs)
```

where *ninputs* is the number of inputs of a single XOR gate in the module and *width* is the number of independent outputs (XOR gates in parallel). In the plot, the red and yellow circles are the measured characterization points, and the small blue circles are the estimated points.
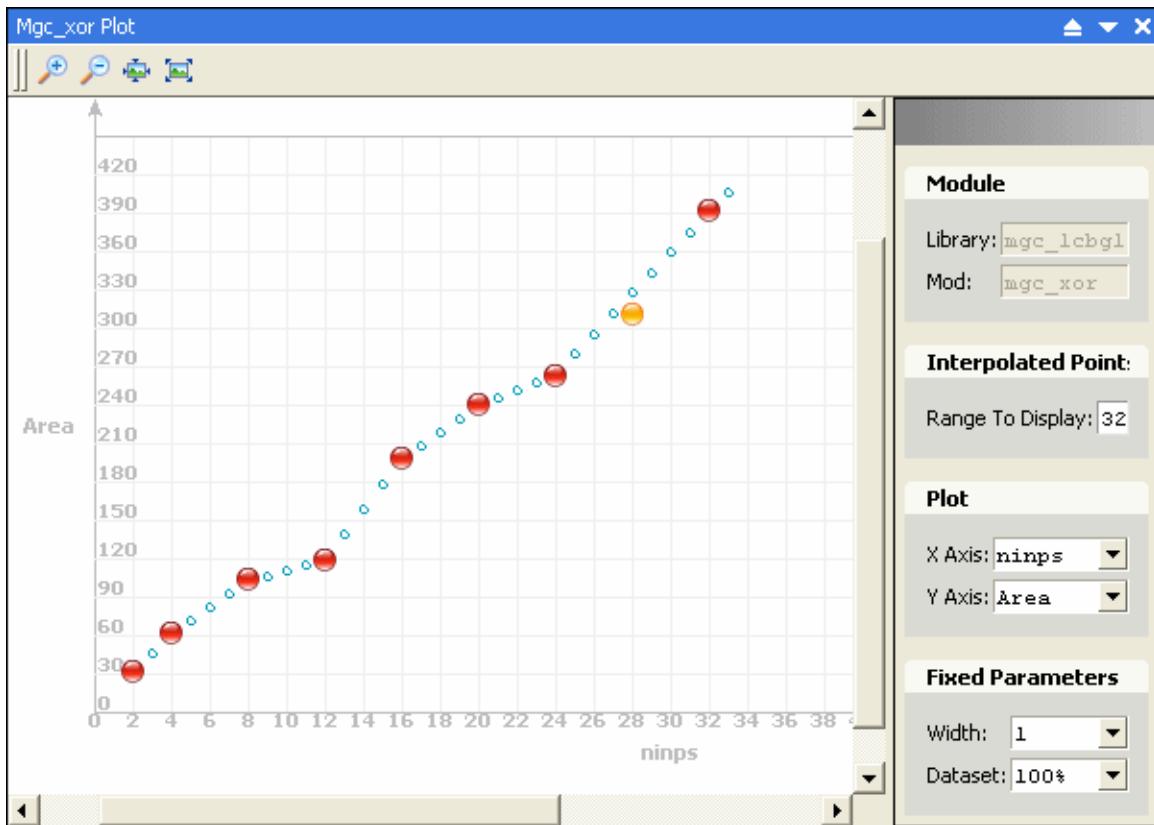
Illustration 24: Interpolate() Characterization Curve

The plot in Illustration 25 shows that better correlation between measured and estimated data values is achieved by including the expand() expression. The interpolate() expression is unchanged. The curve

specification for this plot is:

```
interpolate(width, ninps * width),expand(ninps, width)
```


Illustration 25: Interpolate() and Expand() Characterization Curve

3.7.2. Editing Characterization Properties

As shown Illustration 26, to edit a module property with the Library Editor GUI. select the *All* binding of a module to access its properties. To add a new characterization property, click the New button. Otherwise select an existing property and click the Edit button. Fill in the dialog box and click the OK button.

From the command line, use either the library add or library set commands to create or modify, respectively, library properties. The command syntax is the same for both commands:

```
library add|set /LIBS/<lname>/MODS/<mname>/BINDINGS/all/PROPERTY_MAPPING --\
    -<property_name> {<expression>}
```

Example:

```
library set /LIBS/mgc_lcbg11p_beh_dc/MODS/mgc_xor/BINDINGS/all/PROPERTY_MAPPING --\
    -Area {interpolate(width, ninps * width),expand(ninps, width)}
```
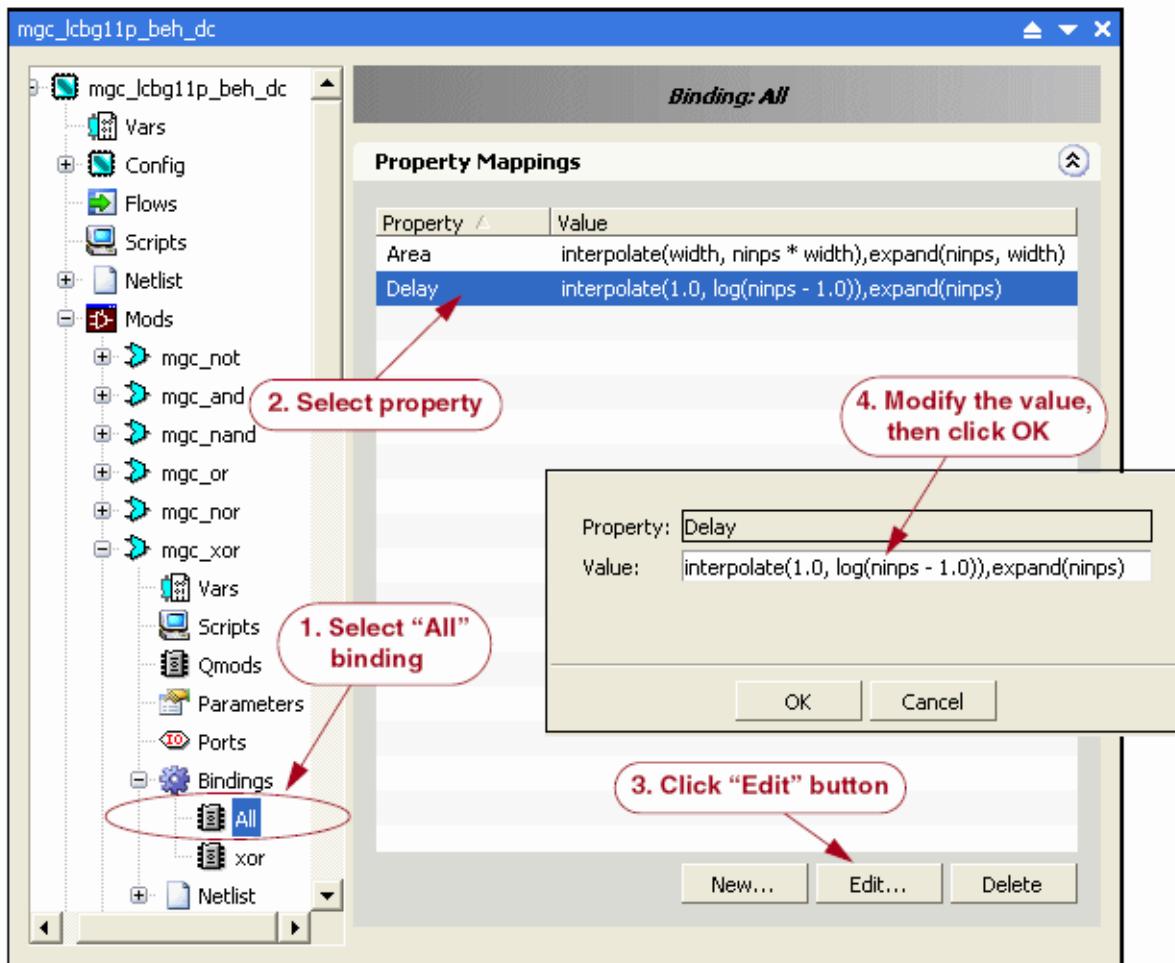
Illustration 26: Editing Characterization Properties

3.8. Using the Library Farm

Library Builder contains a Library Farm tool that can be used distribute library characterization tasks to host computers on your network, thereby speeding up library characterization processing by allowing the tasks to run in parallel.

NOTE: You should verify that the .cshrc sources all scripts required to enable the RTL synthesis tool and Catapult, so rsh is able to just start running the tools. Make sure to check the number of RTL synthesis tool licenses available or the Farm may consume more than are available.

Characterization reports do not include or reflect failures due to license issues. If you are unsure why a Library Farm job is failing, try checking to ensure that the necessary license environment variables are being passed to the farm machine.

To set up the Library Farm tool:

1. **Enable the Farm:**

   The factory default settings have the Farm disabled and the Farm window hidden. To make the window visible, select the **View** > **Farm** menu item. To enable the Farm, right-click in the Farm window and choose Enable Farm from the popup menu. If you want the Farm to be enabled by default, modify the

settings in the Farm Options (**Tools** > **Set Options....** > **Farm**)

2. **Configure Library Builder** for your farm. Most farms use of one of the following two load balancing software types. Examples on how to set up jobs on your farm (e.g. OS, RAM requirements, and licensing) can be found in the following options. These settings should be the same as your RTL Synthesis flow.

   ○ Load Sharing Facility (LSF) software

   ○ Sun Grid Engine (SGE)

   After you have configured the farm, you can add hosts.

3. Setting Up Library Farm Hosts will explain how to set up a host to run jobs on your farm.

4. **Test the farm configuration.**

   a. Pick one QMOD (e.g. mgc_not) and run that on your farm to make sure the RTL synthesis tool runs, a transcript is produced, and the Library Explorer table contains numbers from characterization. If the Library Explorer table is empty, right-click on the QMOD and select View Transcript from the menu.

   b. After one QMOD runs successfully, run one MOD on the farm. If the MOD runs successfully, you can run your library.

3.8.1. Setting Up Library Farm Hosts

The Library Farm window displays the list of hosts and the status of each task, such as idle, passed or failed. The initial Farm window is empty.
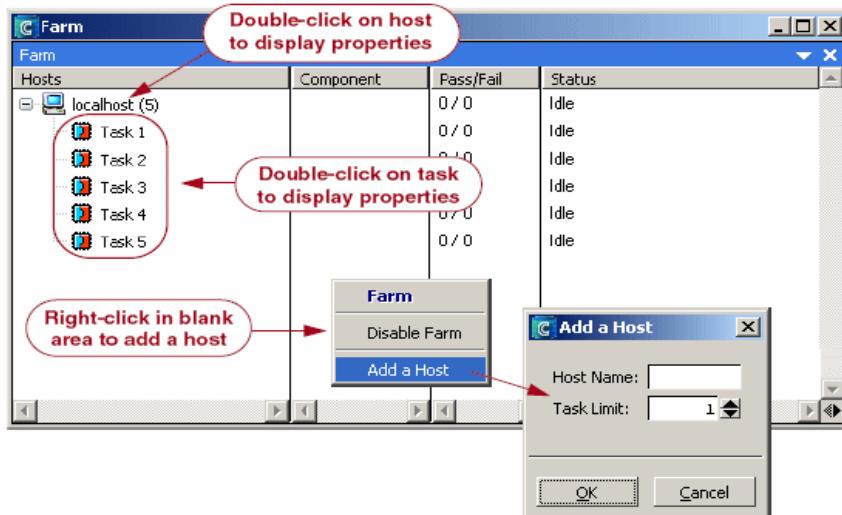
1. To set up the first host, right-click in the blank area on the Farm window and choose **Add a Host** from the pop-up menu to open the Add a Host dialog box, as shown in the illustration below.

   The host name localhost is a reserved name for the local machine. Jobs sent to localhost are run on the local machine without any network requirements. This can be useful for multiprocessor machines where you would like to run additional characterization tasks.

2. Enter the name of the host computer and specify the number of tasks for that host.

   The number of tasks you can set for each host is limited by the number of RTL synthesis tool licenses available.

3. Apply the settings by clicking **OK.**

Illustration 27: Library Farm Window

Right-click on a host name to open the Host menu. This menu allows you to add another task, remove the host, or see properties associated with the host.

- If you select **Add another Task**, a new task is added to the list in the Farm window. The pass/fail values are zero and the status is set to idle.

- If you select **Remove**, the host is removed from the Farm list.

- If you select **Properties**, the Farm Properties dialog box opens, with which you can change the number of tasks for that host. Use the scroll-arrows, or type the desired number of tasks and click **OK** or **Cancel**.

Task Menu: Right-click on a task to open its pop-up menu. The menu allows you to remove the selected task or view its properties.

For more information on setting up Library Farm hosts, see the Farm Window section.

Configuring Library Farm to Use the Load Sharing Facility (LSF) software

To configure Library Farm to work with LSF, modify the **Remote Shell Command** setting on the Farm options page and set to an appropriate LSF command line. The format of the command line will include LSF command switches and Catapult internal variables as arguments. For information about these internal variables, refer to the Farm Options help. You can also set the Remote Shell Command option by using the options set command as shown in the examples below.

NOTE: You must add placeholder "hosts" to the Library Farm in order to specify the number of jobs to submit to LSF. Any name may be used, except "localhost," which is a reserved name. For example, you might add a host named "LSF" and a task limit of 5. That would imply that five LSF jobs may be submitted concurrently. You may adjust the number of tasks while Library Builder is characterizing.

Example 1: Specify LSF *bsub* command line to run on a local machine

Set the following option in Catapult:

```
options set Farm RshCommand {bsub -o %OUTPUTFILE% -q long
-R "select[rh30_32b==1]" -L /bin/csh -K %COMMAND% -file %COMMANDFILE%}
```

- The **-o %OUTPUTFILE%** option specifies that upon completion of the job the standard output should be placed in the file named by **%OUTPUTFILE%** on the local host.

- The **-q long** option specifies the queue name for the job. The queue is named **long** in this example.

- The **-R "select[rh30_32b==1]"** option specifies the resource requirements of the job.

- The **-env** argument specifies a comma-separated list of job submission environment variables to propagate, this can be useful for ensuring the farm machine has access to licenses. Some common argument options include:

  - "<var_name>=<var_value>" – Specifies a variable name and value to pass to the farm machine. For example:

```
-env "VAR1=val1, VAR2=val2"
```

  - "none" – If you specify none with no other variables, no environment variables are passed along while submitting the job.

  - "all" – Specify this option at the beginning of the argument list to pass along all environment variables, you can also specify specific values after the all option

  If -env and -L conflict, -L takes priority. For full details about this and other arguments, consult the official LSF documentation.

- The **-L /bin/csh** option starts a login shell and configures the environment with the user's login scripts. This only matters if you are launching between different OS platforms. The **-L <arg>** option may be omitted if you are using the same OS for the submitted job.

- The **-K** option specifies that the **bsub** command should not return until the job is completed.

- The **%COMMAND%** option specifies the Catapult command to be invoked

  NOTE: When the Library Builder is configured to use dedicated licenses for characterization tasks instead of Catapult licenses, you must use the %COMMAND% variable in the Remote Shell Command field. Do not enter a literal invocation command line. The variable automatically supplies special command line flags that are required for the new license.

- The **-file %COMMANDFILE%** option is the Catapult command line option to supply a script file.

Example 2: Specify the LSF command to run from remote LSF *gate* machine:

Set the following option in Catapult:

```
options set Farm RshCommand {rsh %HOSTNAME% "cd %CWD%; bsub -o %OUTPUTFILE%
-q long -R \"select[rh30_32b==1]\" -L /bin/csh -K %COMMAND% -file %COMMANDFILE%"}
```

In this example the entire **bsub** command is passed as an argument to an **rsh** command. The **rsh** command takes two arguments, the name of the host machine (**%HOSTNAME%**) and the command string to be executed by **rsh**. Notice that the second argument is enclosed in double quotes and any nested quotes within it (such as `-R \"select[rh30_32b==1]\"`) must be escaped with a backslash.

Example 3: Running Library Builder from an xterm on a LSF host

If xterm is already running on a LSF host with some cores already allocated, you can run Library Builder from this xterm so that it appears to be '*local*' to Library Builder. To accomplish this task, set the following options in Catapult:

```
options set Farm/RshCommand %COMMAND%
options set Farm/EnableFarm true
utility farm add -name host -count 8
```

Using these options, Library Builder will run the characterization scripts on the xterm which is already running on a LSF host. The host name specified in the *utility farm add* command can be any arbitrary name because the *%HOSTNAME%* is not used (*Farm/RshCommand* only passes the *%COMMAND%*).

Configuring Library Farm to Use the Sun Grid Engine (SGE) software

To configure Library Farm to work with LSF, modify the **Remote Shell Command** setting on the Farm options page and set to an appropriate LSF command line. The format of the command line will include LSF command switches and Catapult internal variables as arguments. For information about these internal variables, refer to the [Farm Options](#) help. You can also set the Remote Shell Command option by using the [options set](#) command as shown in the examples below.

Example 1: Specify SGE command line to run on a local machine

Set the following option in Catapult:

```
options set Farm/RshCommand {grid qrsh -o %OUTPUTFILE% -q long
-R "select[rh30_32b==1]" -L /bin/csh %COMMAND% -f %COMMANDFILE%}
```

- The **-o %OUTPUTFILE%** option specifies that upon completion of the job the standard output should be placed in the file named by **%OUTPUTFILE%** on the local host.

- The **-q long** option specifies the queue name for the job. The queue is named **long** in this example.

- The **-R "select[rh30_32b==1]"** option specifies the resource requirements of the job.

- The **-L /bin/csh** option starts a login shell and configures the environment with the user's login scripts. This only matters if you are launching between different OS platforms. The **-L <arg>** option may be omitted if you are using the same OS for the submitted job.

- The **%COMMAND%** option specifies the Catapult command to be invoked

  NOTE: When the Library Builder is configured to use dedicated licenses for characterization tasks instead of Catapult licenses, you must use the %COMMAND% variable in the Remote Shell Command field. Do not enter a literal invocation command line. The variable automatically supplies special command line flags that are required for the new license.

- The **-file %COMMANDFILE%** option is the Catapult command line option to supply a script file.

Example 2: Specify the SGE command to run from remote *gate* machine:

Set the following option in Catapult:

```
options set Farm RshCommand {grid qrsh %HOSTNAME% "cd %CWD%; bsub -o %OUTPUTFILE%
```

```
-q long -R \"select[rh30_32b==1]\" -L /bin/csh -K %COMMAND% -file %COMMANDFILE%"}
```

In this example the entire **bsub** command is passed as an argument to an **rsh** command. The **rsh** command takes two arguments, the name of the host machine (**%HOSTNAME%**) and the command string to be executed by **rsh**. Notice that the second argument is enclosed in double quotes and any nested quotes within it (such as `-R \"select[rh30_32b==1]\"`) must be escaped with a backslash.

Example 3: Running Library Builder from an xterm on a SGE host

If xterm is already running on a LSF host with some cores already allocated, you can run Library Builder from this xterm so that it appears to be '*local*' to Library Builder. To accomplish this task, set the following options in Catapult:

```
options set Farm/RshCommand %COMMAND%
options set Farm/EnableFarm true
utility farm add -name host -count 8
```

Using these options, Library Builder will run the characterization scripts on the xterm which is already running on a LSF host. The host name specified in the *utility farm add* command can be any arbitrary name because the *%HOSTNAME%* is not used (*Farm/RshCommand* only passes the *%COMMAND%*).

Chapter 4:   Editing Libraries

After Library Builder has created the library, you can edit the library to provide additional information in order to build a valid library. This information consists of library data such as latency, throughput behavior, and parameter ranges. To edit the library, right-click on the library in the **Library Explorer tab** and select **edit.**
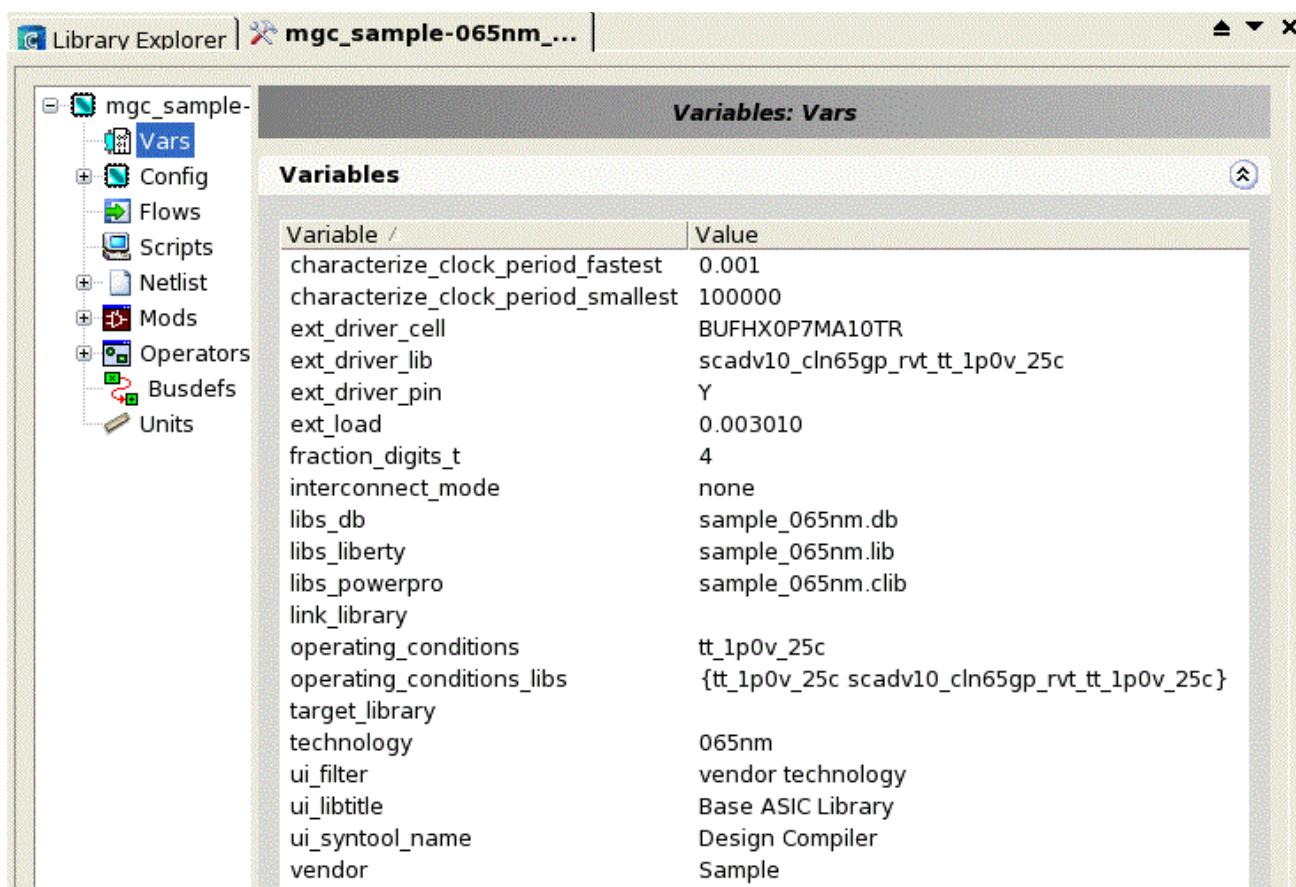
Each library consists of a set of Variables that affect the entire library and a set of Modules (MODs). Catapult determines the area and delay for operators using the modules in the library. Catapult also uses information in each module to determine how to instantiate the operator within the RTL netlist. The following sections describe how to view and edit each section of the library:

- Editing Library Variables describes how to edit variables in the GUI and provides a list of all variables in a library.

- Editing Module Parameters define the list of parameters on the Operator or interface.

- Editing Module Ports define the inputs and outputs to the operator or interface.

- Editing Module Bindings map the parameters on the operator to the parameters on the RTL netlist module.

- Editing Module Pin Associations map the pins on the module to the ports on the operator.

- Editing Module Property Mappings map the properties (attributes) on the module to the operator.

4.1. Editing Library Variables

To change library variables after a library is created, you must use the Library Builder as described in the following procedure.

1. Load the library into Library Builder (*catapult -product lb*)

2. Click on the Library Explorer tab. The Library Explorer window displays.

3. Double-click on the library to edit. Library edit tab opens to display the settings for the library as shown in the following illustration:

Illustration 28: Exposing Variables in Library Explorer Window

4. Click **Vars** as shown in Illustration 28. The defined variables are now displayed.

5. Depending on the modifications, use the New, Edit, and Delete buttons to modify the library variables as needed.

6. Save the changes to the library: click the Library Explorer tab, right-click the library that you would like to save and select Save from the menu.

The following table describes each of the library variables. For obsolete library variables, the table describes the variables to be used instead. Unless noted in the table, you can infer the following information from each variable:

• Library Builder stores variables under */LIBS/<lib_name>/VARS*.

• The corresponding switch for the flow run command is always *-set_<var_name>*.

• To ensure good correlation, most variable values should match the corresponding command in your RTL synthesis script.

VAR Name	Description
advanced_retiming	**Type***: boolean.* **Default***: true.* **RTL Tool:** *DC* **Wizard Name***: Advanced Retiming* If true the optimize_registers is used for retiming, the balance_registers is used otherwise.
cap_factor	**Type***: float.* **Default***: 1.* **RTL Tool:** *All* **Wizard Name***: none.* Must be equal to capacitance unit factor in technology library. Specifies the capacitance unit factor.
cap_unit	**Type**: *pick list.* **Default**:*ff* **RTL Tool:** *All* **Wizard Name**: *Capacitance Unit.* Allowed values are nf, pf, and ff. Must be equal to capacitance unit in technology library. In conjunction with cap_factor specifies the capacitance unit.
characterize_clock_period_fastest	**Type**: *double.* **Default:** "*0.01*" (nS). **RTL Tool:** *All* **Wizard Name**: *Clock Period Fastest* This value specifies the clock speed setting for RTL Synthesis when characterizing the fastest clock period (fastest implementation with largest area).
characterize_clock_period_percents	**Type**: *string.* **Default:** *{100% 75% 50% 0%}* **RTL Tool:** *All* **Wizard Name**: *Clock Period Percents* This field specifies a set of points within the range of clock speeds bounded by the fastest and smallest settings. The points are specified as percentages of the range. The default set consists of four points: 100%, 75%, 50% and 0%, where 100 = fastest and 0 = smallest.
characterize_clock_period_smallest	**Type***: double.* **Default:** *1000.0 (ns).* **RTL Tool:** *All* **Wizard Name**: *Clock Period Smallest* This value specifies the clock speed setting for RTL Synthesis when characterizing the slowest clock period (smallest area).
compile_to_placed	**Type**: *boolean.* **Default**:*false.* **RTL Tool:** *RC* **Wizard Name**: *Set Compile Mode to Placed* If set adds the -to_placed switch to the compile command in RC.
compile_ultra	**Type**: *boolean.* **Default**:*false.* **RTL Tool:** *DC* **Wizard Name**: *Set Compile Mode to Ultra* Specifies whether to run Design Compiler in *ultra* mode. Since the characterized operators are generally small, using DC Ultra tends to increase the run time of characterization without significant effect on the area or timing estimates. During characterization, you should set this variable to false. For your Design Compiler runs, you should continue to use this option during RTL synthesis.
dont_use_cells	**Type**: *string.* **RTL Tool:** *All* **Wizard Name***: Excluded Cells* Specifies a list of technology cells that will not be used during RTL synthesis. This optional list should contain <library name> <cell name> pairs. This value is passed to the dont_use command.
dont_get_license	**Type**: *string.* **RTL Tool:** *DC*

VAR Name	Description
	Wizard Name: *Design License not to use* Specifies a list of DesignWare licenses that the DC tool is not allowed to use. The list items are space separated. The list is assigned to the DC variable *synlib_dont_get_license* in the generated DC script. This value will be the default setting for users of the Catapult library.
default_clock_buffer	**Type**: *string*. Default: "". **RTL Tool:** *All* **Wizard Name**: *Default Clock Buffer* This variable defines the default value for the clock_tree_buffer option in the Low-Power flow. In this flow, you can override the buffer name with a clock buffer in the [Low-Power](#) Option form.
ext_driver_lib	**Type**: *string*. Default: "". **RTL Tool:** *All* **Wizard Name**: *External Driver Cell Library* Driver cell is used as a driver for all input pins. This field specifies the library in which the driver cell is found. If left blank no driver cell is used.
ext_driver_cell	**Type**: *string*. **Default**: "". **RTL Tool:** *All* **Wizard Name**: *External Driver Cell* Driver cell is used as a driver for all input pins. This field specifies the cell name. If left blank no driver cell is used. Library Builder attempts to select the buffer with an area in the 66th percentile of all buffers in the library. If you are specifying a driver cell, then you must specify the library, cell and pin or else Library Builder will not use a driver cell.
ext_driver_pin	**Type**: *string*. **Default**: "". **RTL Tool:** *All* **Wizard Name**: *External Driver Cell Pin* Driver cell is used as a driver for all input pins. This field specifies the name of the output pin of the driver cell. If left blank no driver cell is used. If you are specifying a driver cell, then you must specify the library, cell and pin or else Library Builder will not use a driver cell.
ext_load	**Type**: *float*. **Default:** *0*. **RTL Tool:** *All* **Wizard Name**: *External Capacitive Load* Specifies the capacitive load that is added to the each top-level output pin. This value is typically set to match the loading effect of a small buffer from the technology library. The specified value is a number of $cap_factor $cap_unit units. Higher value would result in more pessimistic and lower – more optimistic estimations.
fraction_digits	This variable is ***obsolete***. Use fraction_digits_t (for timing values) and fraction_digits_c (for capacitance values) instead.
fraction_digits_t	**Type**: *integer*. **Default**: *4*. **RTL Tool:** *All* **Wizard Name**: *Timing Report Fractional Digits* Range: 0-13. Specifies the number of digits after the decimal point to store for timing values in the Catapult library (controls the maximum resolution of the timing values).
fraction_digits_c	**Type**: *integer*. **Default**: *4*. **RTL Tool:** *All*

VAR Name	Description
	Wizard Name*: none* Range: 0-13. Specifies the number of digits after the decimal point to store for capacitance values in the Catapult library (controls the resolution of the capacitance values).
get_license	**Type**: *string*. **Default**: *none*. **RTL Tool:** *DC* **Wizard Name***: Design License(s) to check out* Specifies a space-separated list of Synopsys DesignWare license features to be obtained. If the value is set, it is passed directly to the DC command *get_license* in the generated DC script so the specified license features are checked out at the beginning of synthesis. (They are held until the remove_license command is used or until the program is exited or until the DC shell is closed). Refer to the license key file at your site to determine which licensed features are available. This value will be the default setting for users of the Catapult library.
Input_transition	**Type**: *float*. **Default**: *unset*. **RTL Tool:** *All* **Wizard Name***: Input Transition (Slew)* Specifies the input transition time on the input pins. We recommend using a driver cell (ext_driver_lib, ext_driver_cell, ext_driver_pin) instead. If set the setting will be used together with the driving cell. This setting usually overrides the driver cell setting – see the documentation for your RTL synthesis tools for further details.
interconnect_mode	**Type**: *pick list*. **RTL Tool:** *All* **Wizard Name***: Interconnect Mode* Allowed values: *none* and *wireload*. Specifies whether to use wire load models. By default, DC uses 'none'. If no wire load models are specified in Catapult library the RTL synthesis tool may still select a wireload automatically. *wireload*: Enables the use of wireloads, if the wireload model, mode and/or selection group are specified those will be communicated to the RTL synthesis script through the generated synthesis script. *none:* In DC, this value will disable automatic wireload selection. No wireload settings will be written into the script even if specified.
lef_library	This variable is ***obsolete***. Use libs_lef instead.
libs_db	**Type**: *string*. **Default**: "". **RTL Tool:** *All* **Wizard Name***: Synopsys DB Library Files* A space separated list of technology libraries in Synopsys db format. Not used by the flows providing integration with non-Synopsys tools.
libs_lef	**Type**: *string*. **Default**: "". **RTL Tool:** *Oasys* **Wizard Name***: LEF Library files* A space separated list of technology libraries in LEF format.
libs_liberty	**Type**: *string*. **Default**: "". **RTL Tool:** *All* **Wizard Name***: Liberty Library files* A space separated list of leaf filenames of technology libraries in Liberty format. The path to these files must be included in the [TechLibSearchPath] option.
libs_map	**Type**: *string*. **Default**: "". **RTL Tool:** *Oasys*

VAR Name	Description
	Wizard Name: *Oasys Layer Mapping file* .The filename that contains the mappings between the layer names in the PTF and technology LEF files.
libs_powerpro	**Type**: *string*. **Default**: "". **RTL Tool:** *All* **Wizard Name**: *PowerPro Library files* A space separated list of leaf filenames of technology libraries in PowerPro (encrypted Liberty) or plain Liberty format. In Catapult's Low-Power option, this list overrides the value of the libs_liberty. This variable allows the Low Power option in Catapult to use different libraries for power analysis. The path to these files must be included in the TechLibSearchPath option.
libs_ptf	**Type**: *string*. **Default**: "". **RTL Tool:** *Oasys* **Wizard Name**: *Oasys PTF files* A space separated list of technology libraries in PTF format. This file contains the resistance and capacitance values per unit length for each layer.
link_library	This variable is ***obsolete.*** Use libs_db and libs_liberty variables instead.
libname	**Type**: *string* **Default**: "". **RTL Tool:** *All* **Wizard Name**: *Library Name* The Library Wizard will append "_ccs" to the technology name in the liberty file. The Catapult library will be saved using this value as the root name with a ".lib" suffix.
libtitle	**Type**: *string* **Default**: "". **RTL Tool:** *All* **Wizard Name**: *Library Title* In Catapult, this name will appear in the Compatible Libraries list in the Libraries Constraint Editor.
no_boundary_optimization	**Type**: *boolean*. **Default**: *true*. **RTL Tool:** *All* **Wizard Name**: *No Boundary Optimization* Disables boundary optimization during RTL synthesis if set to true.
operating_conditions	**Type**: *string*. **Default**: "". **RTL Tool:** *All* **Wizard Name**: *Operating Conditions* Specifies the name of the operating conditions specified in the technology library.
operating_conditions_libs	**Type**: *char* **Default**: "" **RTL Tool:** *All* A list of <library name> <operating conditions name> pairs. Allows Catapult to be easily re-targeted to various operating conditions in the Low-Power flow. The path to these files must be included in the TechLibSearchPath option.
set_driving_cell	This variable is ***obsolete***. Use ext_driver_lib, ext_driver_cell, and ext_driver_pin instead.
set_load	This variable used with DC is ***obsolete***. Use ext_load instead.
set_input_transition	This variable is ***obsolete***. Use input_transition instead.
scan_registers	**Type**: *boolean*. **Default**: *false*. **RTL Tool:** *DC* **Wizard Name**: *Enable Scan Registers* Specifies whether to use scan registers.

VAR Name	Description
significant_digits	This variable used with DC is ***obsolete***. Use fraction_digits_t (for timing values) and fraction_digits_c (for capacitance values) instead.
synthetic_library	**Type**: *string*. Default: "". **RTL Tool:** *DC* **Wizard Name**: *Synthetic Libraries* Specifies a space separated list of DesignWare library files. dw_foundation.sldb standard.sldb is frequently used.
target_library	This variable is ***obsolete.*** Use libs_db and libs_liberty variables instead.
technology	**Type**: *string*. **Default**: *TECHNOLOGY* **RTL Tool:** *All* **Wizard Name**: *Technology* Catapult libraries are associated with a vendor and technology names used for filtering. In Catapult, when these names are selected in the Libraries Constraint Editor, the library will appear in the Compatible Libraries list of the dialog box. If you are creating a library that must work with an existing Base Library, then the technology and vendor names must match. For example, vendor name "Sample" and technology name "065nm".
time_factor	**Type**: float. **Default**: '*1*'. **RTL Tool:** *All* **Wizard Name**: *Time Factor* Must be equal to timing unit factor in technology library. Specifies the timing unit factor.
time_unit	**Type**: pick list (*ns*, *ps*, or *fs*). **RTL Tool:** *All* **Wizard Name**: *Time Unit* Must be equal to time unit in technology library. In conjunction with time_factor specifies the time unit.
vendor	**Type**: *string*. **Default**: *VENDOR* **RTL Tool:** *All* **Wizard Name**: *vendor* Catapult libraries are associated with a vendor and technology names used for filtering. In Catapult, when these names are selected in the Libraries Constraint Editor, the library will appear in the Compatible Libraries list of the dialog box. If you are creating a library that must work with an existing Base Library, then the technology and vendor names must match. For example, vendor name "Sample" and technology name "065nm".
wait_for_design_license	**Type**: *string*. **Default**: *""* **RTL Tool:** *DC* **Wizard Name**: *Design License to wait for* Specifies a list of DesignWare licenses that the DC tool should wait for if they are temporarily unavailable. The list items are space separated. The value of this variable is assigned to the DC variable *synlib_wait_for_design_license* in the generated DC script. This value will be the default setting for Catapult library users.
wire_load_libs	**Type**: *string*. **Default**: *""* **RTL Tool:** *All* **Wizard Name**: *Wire Load Libs* A list of <library name> <wireload name> pairs. Allows Catapult to easily select proper wireload based on the library name.
wire_load_mode	**Type**: *string (top, enclosed, or segmented)*. **Default**: *"enclosed"*. **RTL Tool:** *All*

| VAR Name	Description
	Wizard Name*: Wire Load Mode* Corresponds to *set_wire_load_mode* in SDC. This library variable can be overridden by the *wire_load_mode* option in the <u>Low-Power flow.</u> Options.
wire_load_model	**Type**: *string*. **Default**: "". **RTL Tool:** *All* **Wizard Name***: Wire Load Model* Corresponds to set_wire_load_model in SDC. Set the *wireload model name* only (no library name etc). This library variable can be overridden by the *wire_load_model* option in the <u>Low-Power flow</u> Options.
wire_load_selection_gr oup	**Type**: *string*. **Default**: *""*. **RTL Tool:** *DC* **Wizard Name***: Wire Load Selection Group* Corresponds to set_wire_load_selection_group *in SDC*.

4.2. Editing Module Parameters

Module parameters typically correspond to the generics/parameters on the RTL.
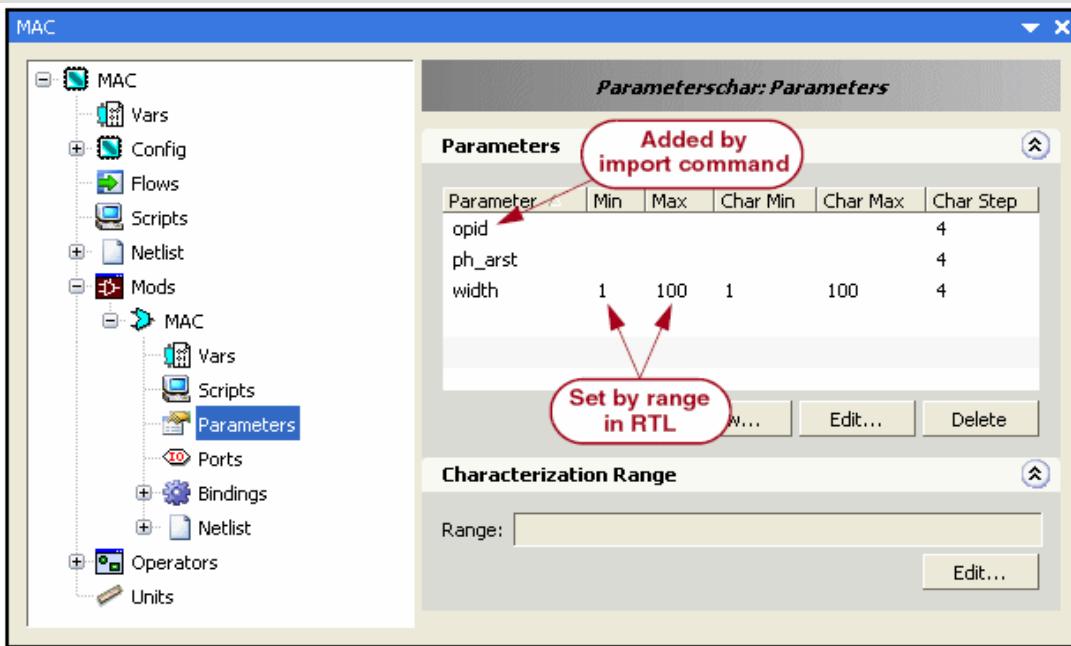
Note that for a base library the parameters of each module are set in the template and should not be edited.

The import command will add an additional parameter for interfaces, memories, and operators with state. For memories, a parameter called *ram_id* is added. For interfaces, a parameter called *rscid* is added. For custom operators with state, a parameter called *opid* is added. The user does not need to modify these parameters. For all other parameters the min/max range and min/max characterization range must be set. The min/max range tells Catapult the allowable range of the parameter. This controls things like how wide can a port be set on an interface, or how many words can a memory have. If the RTL generic/parameter has a range set on it, the import command will annotate this information for the min/max and characterization ranges. If the range is not specified in the RTL the user must set it manually. For example:

```
ENTITY MAC IS
GENERIC(width: natural range 1 to 100:= 2;
        ph_arst : natural := 1);
PORT (
      clk       : IN STD_LOGIC ;
      rst       : IN STD_LOGIC ;
      a : IN STD_LOGIC_VECTOR(width-1 DOWNTO 0) ;
      b : IN STD_LOGIC_VECTOR(width-1 DOWNTO 0) ;
      c : OUT STD_LOGIC_VECTOR(width*2-1 DOWNTO 0)
      ) ;
END MAC ;
```

Illustration 29: Parameters Imported from Netlist

Parameters can be set manually by double-clicking on the parameter or selecting edit.
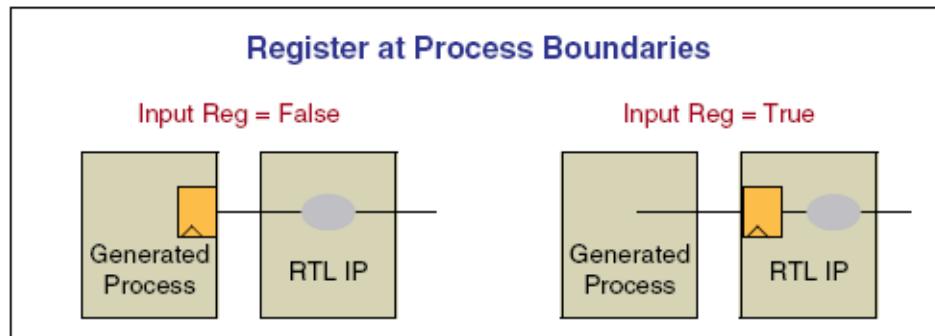
4.3. Editing Module Ports

Note that for a base library the ports for each module are set in the template and should not be edited.

Ports correspond to the ports on the RTL. Most of the port information is added during the RTL import, but there are some fields that must be set by the user.
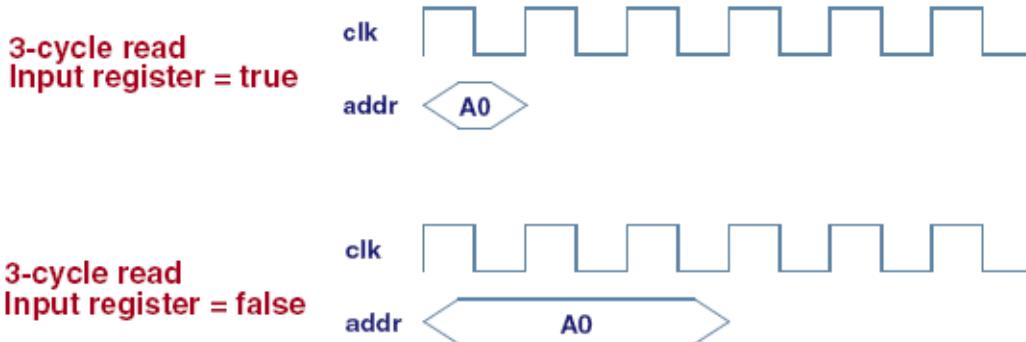
- **Input Register Setting**

  Catapult requires that either the outputs of the core process (RTL) that it generates or the inputs of the interface/operator RTL IP are registered. If input register is set to true on the module port, then Catapult does not put a register on the corresponding output of the core process. This is illustrated in the following illustration.


Illustration 30: Input Register Setting

There are cases where you may wish to set input register to false even though the interface/operator RTL has input registers. If you require that the data and control inputs to the interface/operator RTL

should be driven for multiple clock cycles, set the input register to false. This is only done for multi-cycle interface/operators.
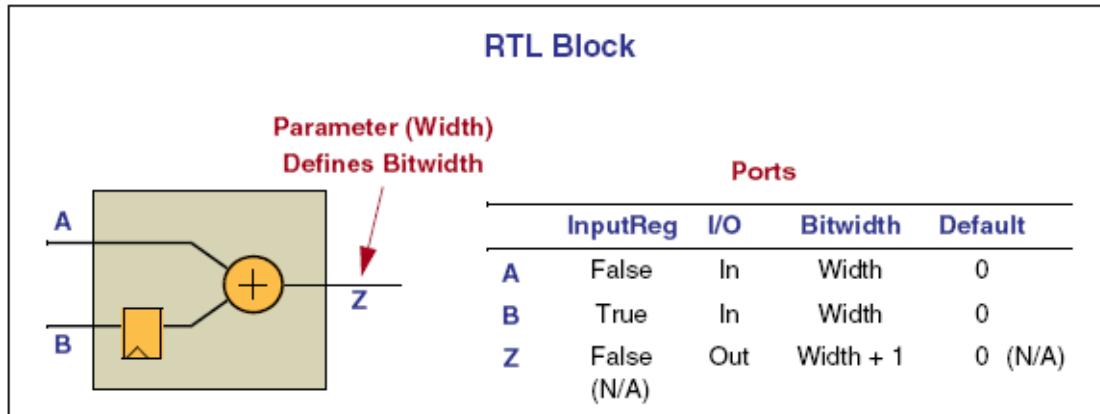

Illustration 31: Effects of Input Register Setting

- **Sign Bit**

  The port sign bit indicates that the port is signed or unsigned (1 == signed, 0 == unsigned). Catapult will sign extend inputs and outputs of signed ports. Single bit ports should leave the sign bit unassigned. Setting the sign bit on a single bit port results in std_logic_vector(0 downto 0).

- **Default Value**

  The default value port setting indicates what value should be driven on an interface/operator input port when the interface/operator is not being read/written. This setting is used for ports like memory write enables, resets, and so on.
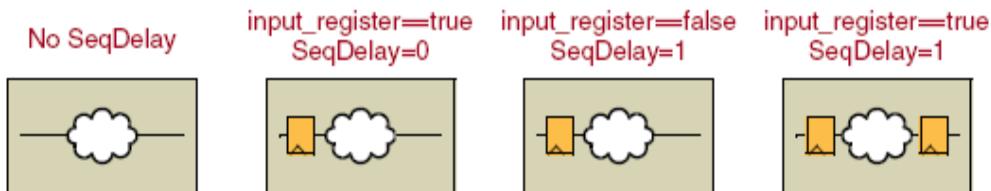

Illustration 32: Port Settings Overview

4.4. Editing Module Bindings

Bindings tell Catapult how to connect to the RTL interface/operator. They also allow timing and area information to be specified. There are usually at least two bindings for every interface/operator, the "All" binding, and the operator binding.

Operator bindings typically consist of read_port/write_port bindings for interfaces, read_ram/write_ram bindings for memories, built-in operators (add, mul, etc.), and user-defined for custom operators.

All binding is typically used to set area and timing information. This is done via a property mapping. Property mappings allow module parameters to be set, or to map operator parameters to module parameters. The value on the properties on the binding can use any of the PARAMETERS as variables (eg. RAM 'area' may be determined by number of words and the bitwidth control).


Illustration 33: Effect of Input Register on SeqDelay

The user selectable property can be set to true (1=true, 0=false) on individual parameters to make them editable by designers in the Catapult Constraint Editor. In a Catapult session, when editing the architectural constraints of a resource, any user selectable parameters of the library component is editable in the Resource Options field of the Constraint Editor. Each parameter field has a drop-down list of valid values to choose from.

Note that for a base library the bindings for each module are set in the template and should not be edited. The only exception is the property templates under the property mapping. Editing the property templates affects the interpolation quality and interpolation error. You will notice that every time you change the interpolation template the corresponding property is re-interpolated.

You can specify any properties you would like but the following properties are recognized by Catapult:

- **Area**: Total area of the module.
- **AreaSeq**: Sequential area of the module.
- **Delay**: Combinatorial delay, or clock-to-out time of sequential components.
- **InputDelay**: Input-to-clock delay of sequential components (excluding setup time).
- **InputSetup**: Setup time of the first row of registers for sequential components.
- **R2RDelay**: Clock-to-clock delay of the registers for sequential components (excluding setup time).
- **R2RSetup**: Setup time of the registers for sequential components except for the first row specified by InputSetup.
- **MinClkPrd**: Minimum clock period is computed as max(InputSetup+InputDelay, R2RSetup+R2RDelay, Delay).
- **SeqDelay**: Sequential delay. Indicates the number of clock cycles required to complete the operation. If the ports associated with that operation have input registers, this property is needed. If the ports are asynchronous, this property is not needed.
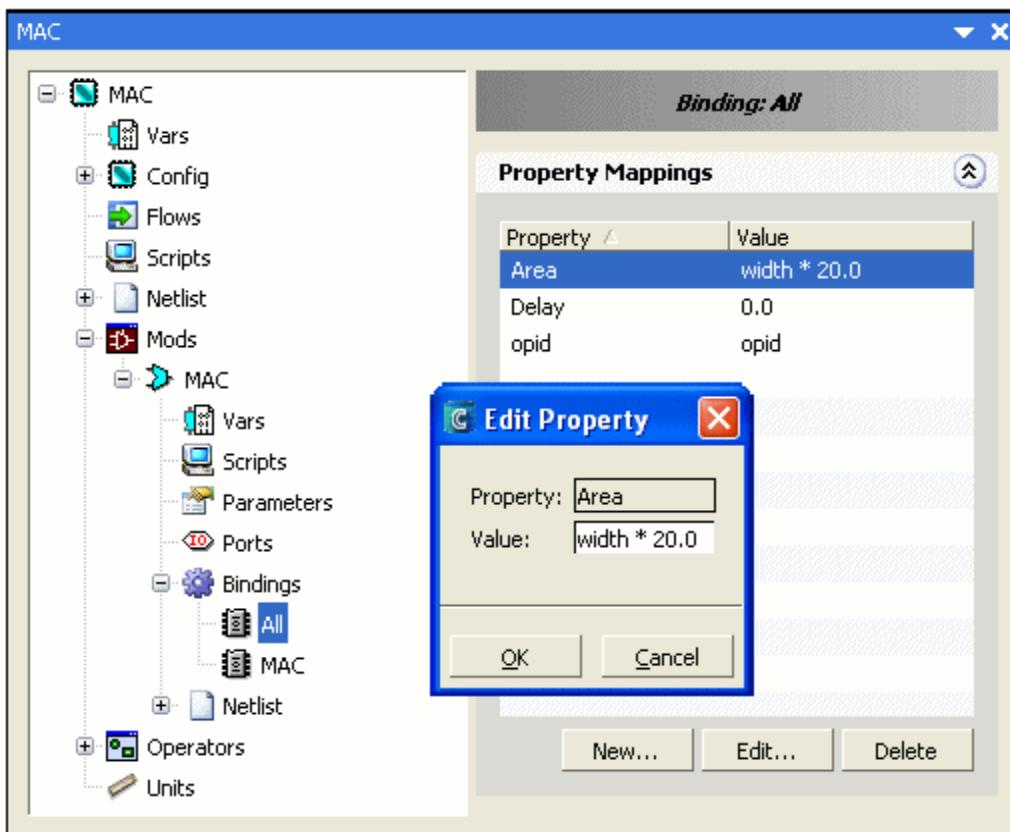
  For example:
  - SeqDelay = Number of Component registers in any IO path less 1
  - For components with input_reg == true, SeqDelay = 0. That means that Data available after first clock edge.
  - No sequential delay property means the component is combinatorial.

- **InitDelay**: Number of cycles between accesses to this port. Default is *SeqDelay* + 1, but it can be set smaller for pipelined operation. For example, *InitDelay* = 2 means that the component can be pipelined downto *II*=2, but not *II*=1.

- **Always**: Must evaluate to true for all valid parameter configurations (QMODS). Used to limit the range of valid QMODS.

Property names are case sensitive. They can be set from the command line (example below) or from the GUI as shown in the following illustration.

```
library add \
/LIBS/<library_name>/MODS/<module_name>/BINDINGS/all/PROPERTY_MAPPING \
-- -Area width*20 -Delay 1.2 -SeqDelay 1 -InitDelay 1
```

The following illustration shows how to set the property in the GUI:

Illustration 34: Setting Property on the ALL Binding

4.5. Editing Module Pin Associations

Note that for a base library the pin associations for each module are set in the template and should not be edited.

Pin associations are used to connect operator ports to RTL ports and to tell Catapult how to bind the RTL component when netlisting the final VHDL or Verilog output. The pin associations are set on the operator binding. The following table describes the pin association types.

Table 8: Pin Associations

Pin	Description
Unbound	Left unconnected
OPERATOR_PIN	Binds to port on operator
CLOCK	Clock from Catapult process.
ENABLE	Clock enable from Catapult process.
S_RST	Synchronous reset from Catapult process
A_RST	Asynchronous reset from Catapult process.
[EXTERNAL]	Connects the port of a bound internal or interface resource to an external port. Interface resources with EXTERNAL bindings must be bound.
DIRECT	Connects the port of a bound internal resource to an external port. DIRECT bindings are removed for interface resources.
GLOBAL	Connects the port of bound internal resources to a single external port when the internal port names are the same. GLOBAL bindings are removed for interface resources.
CONSTANT	Drives a constant value to the component port.
WAITON	Used for handshaking. The Catapult process will wait for the component port to be driven high.

When the RTL netlist is imported, the appropriate binding is automatically added based on the module type setting. If the module type is set to userop the operator ports must be defined before the pin associations can be made. If the inport, outport, ram, or rom module type is set, the operator binding is added and you must specify the pin associations. The following table describes the pins on each operator.

Table 9: Operator Pin Bindings

Binding	Data Port	Address
read_port	D	n/a
write_port	D	n/a
read_ram	D	I
write_ram	D	I

4.6. Editing Module Property Mappings

Property mappings allow operator parameters to be mapped to Module parameters which in turn may correspond to generics on the RTL. This allows an operator to control things like RTL port width, number of words in a memory, and so on. The library import command automatically maps the necessary operator parameters to their respective module parameters (such as width, id, etc.). If you are creating a library component manually, and that component uses built-in operators such as read_port or write_port, you must manually map the operator parameters to the module parameter in the bindings. The built-in interface and

memory operators have the following parameters.

Table 10: Property Mapping

Param	read_port	write_port	read_ram	write_ram	Custom operator	Comment
width	X	X	X	X		Set based on interface or memory width arch constraint
size			X	X		Set based on number of array elements
ramid			X	X		Set for each ram instance
id	X	X				Set for each interface instance
opid					X	Set for operators with state

Chapter 5:   Third-Party Information

The *Third Party Software for Catapult Products* document provides information on third-party software that may be included in the Catapult product, including any additional license terms. This document is located at the following location within the Catapult install tree:

```
$MGC_HOME/shared/legal/oss_thirdparty.pdf
```

Chapter 6:   End-User License Agreement

Use of software (including any updates) and/or hardware is subject to the End-User License Agreement together with the Mentor Graphics EDA Software Supplement Terms. You can view and print a copy of this agreement at:

http://www.mentor.com/eula