cādence®

Virtuoso Liberate MX Reference Manual

Product Version 15.1
November 2015

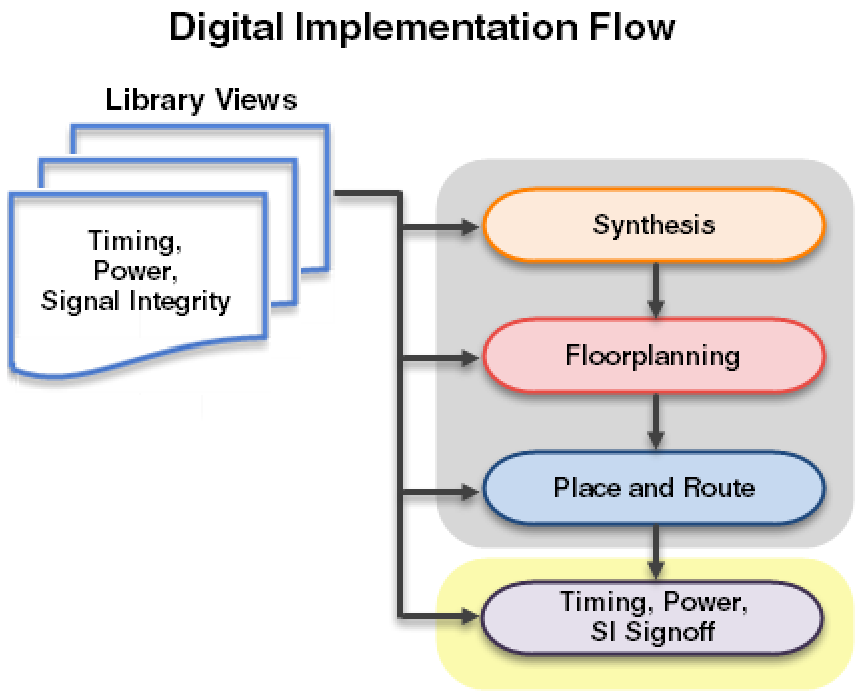
Contents

Preface

Introduction to Characterization

The Role and Importance of Libraries

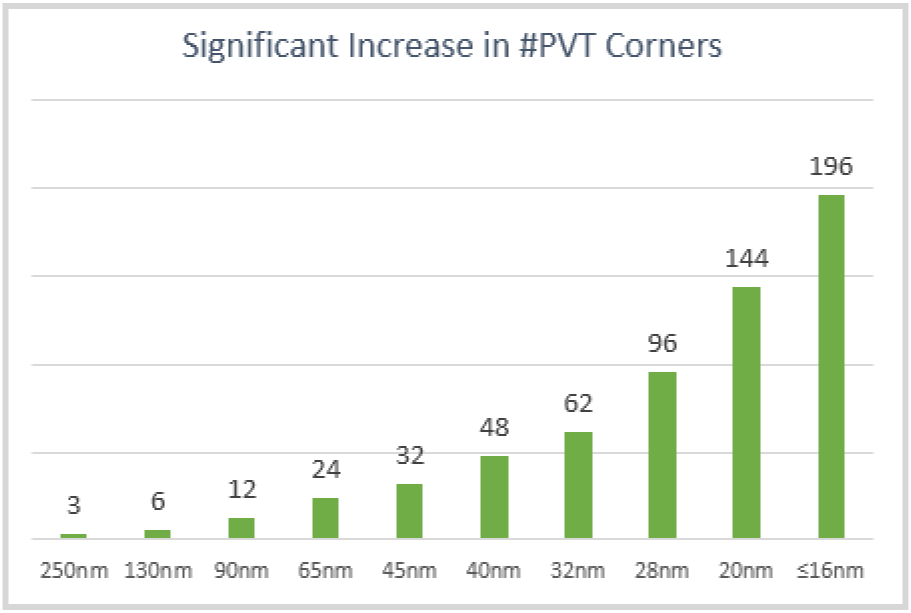
Creation of electrical views is a prerequisite for any digital design flow. The electrical information stored in the library views is used throughout design implementation from logic synthesis, through design optimization to final signoff verification. Accurate library view creation is essential to ensure close correlation between the design intent and the final silicon.


Digital Implementation Flow
Library Views
Timing, Power, Signal Integrity
Synthesis
Floorplanning
Place and Route
Timing, Power, SI Signoff

A Growing Problem

In nanometer geometries (65nm or below), the required number of library views is growing dramatically because of issues related to power leakage and process variation. To minimize power leakage at deep submicron nodes, we see process variations such as LVT, RVT, and HVT (low/regular/high voltage) being utilized. For example, to manage power at 65nm, it is common to have library cells with two or three different threshold values (high threshold to reduce leakage power, low thresholds to improve performance), and to use two or more on-chip supply voltages. In this scenario, the number of views needed for 65nm will be six times greater than what is needed for 130nm.

The figure below shows the growing trend that requires PVT corners to accurately model the circuit behavior:

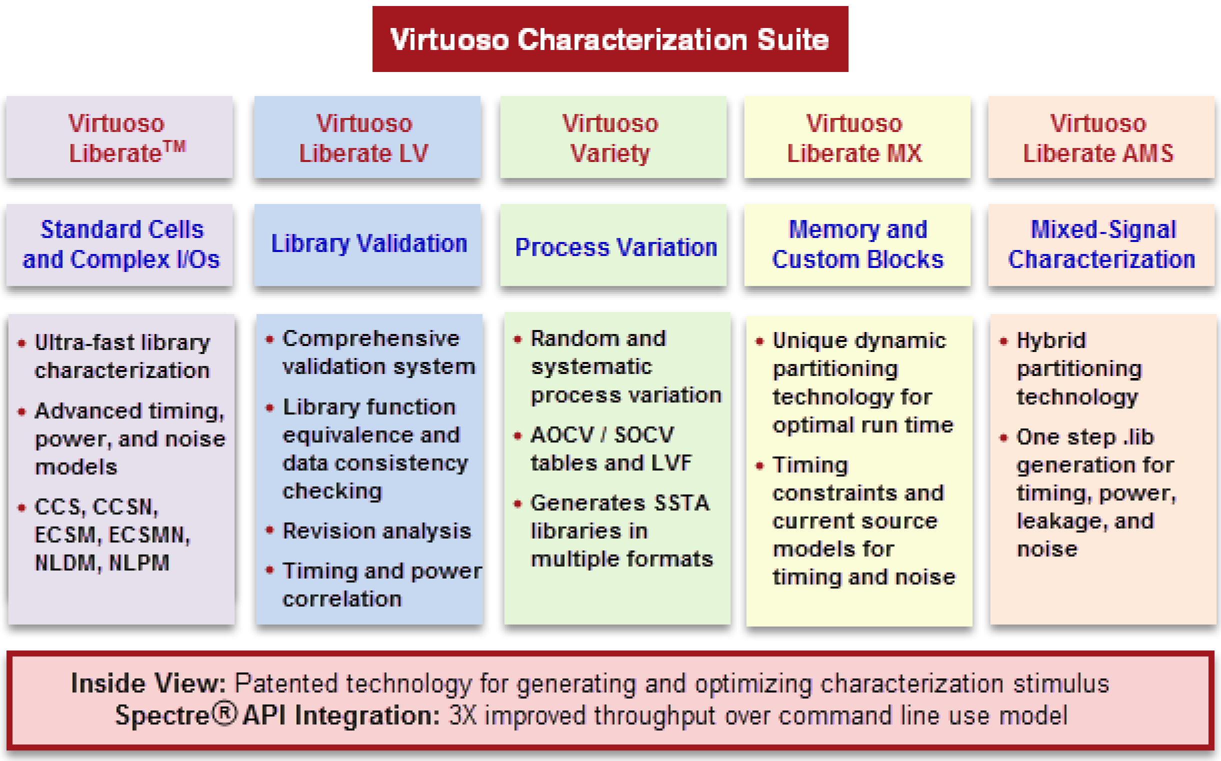


In addition, library views require more advanced models like:

- Current source models CCS and ECSM
- Statistical models – AOCV/SOCV/LVF
- Netlist extraction at various temperatures for Nanometer Process Nodes
- Support multiple foundries to assure flexibility for yield issues
- Support for many more functional designs – 1000+ STD cell, I/O, custom datapath, memory and Analog IP

Virtuoso Characterization Suite

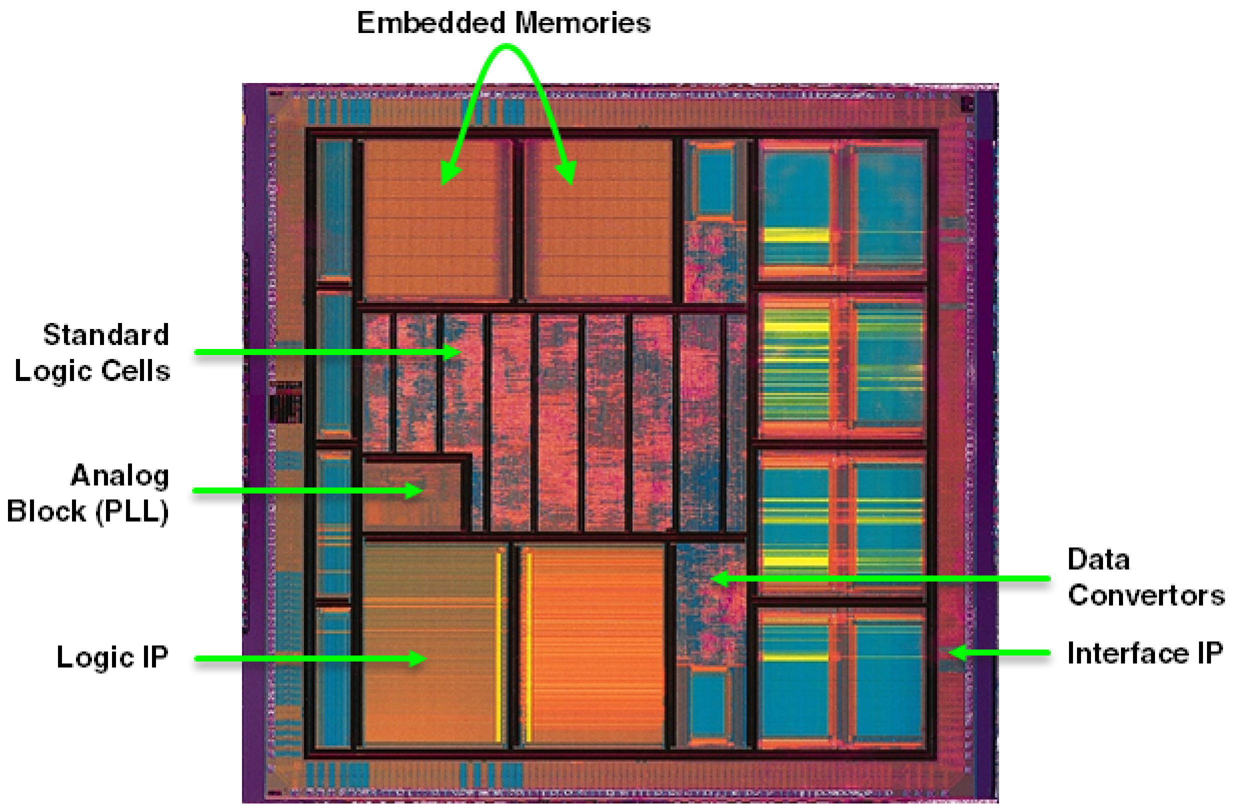
To address all the challenges, Cadence offers Virtuoso® Characterization Suite that covers the complete portfolio of characterization solutions given below:


Virtuoso Characterization Suite
Virtuoso Liberate™
Virtuoso Liberate LV
Virtuoso Variety
Virtuoso Liberate MX
Virtuoso Liberate AMS
Standard Cells and Complex I/Os
Library Validation
Process Variation
Memory and Custom Blocks
Mixed-Signal Characterization
• Ultra-fast library characterization
• Advanced timing, power, and noise models
• CCS, CCSN, ECSM, ECSMN, NLDM, NLPM
• Comprehensive validation system
• Library function equivalence and data consistency checking
• Revision analysis
• Timing and power correlation
• Random and systematic process variation
• AOCV / SOCV tables and LVF
• Generates SSTA libraries in multiple formats
• Unique dynamic partitioning technology for optimal run time
• Timing constraints and current source models for timing and noise
• Hybrid partitioning technology
• One step .lib generation for timing, power, leakage, and noise
Inside View: Patented technology for generating and optimizing characterization stimulus
Spectre® API Integration: 3X improved throughput over command line use model

The Virtuoso Characterization Suite intends to provide highly efficient and automated electrical view creation and validation for all IP blocks that include the following:

- Logic and I/O cells (GPIO, PCI, SSTL, PECL, and so on)
- Embedded memory (SRAM, ROM, Register files, CAM, and so on)
- Custom digital blocks (custom cells, datapath, cores, and so on)

- Interface IP and analog blocks (USB, Serdes, DDR, and so on)



System Requirements

Liberate, Liberate MX, Liberate LV, Liberate AMS, Variety, and ALAPI run exclusively on Linux operating system. The following table lists the supported platforms:

Architecture	Development OS	Supported Environments
x86_64 (32/64)	RHEL 5.5	RHEL 6
		SLES10
		SLES11

For detailed information about the requirements, see Computing Platforms.

Software and Licensing Requirements

- LIBERATE 15.1

  The following table lists the required server and client product numbers for each product in the Virtuoso Characterization Suite:

Product Name	Server Product Number	Client Product Number
Liberate	ALT110	ALT111
Variety	ALT210	ALT211
Liberate MX	ALT410	ALT411
Liberate LV	ALT610	ALT611
Liberate AMS	ALT810	ALT811 or ALT812

- MMSIM 14.1 or MMSIM 15.1

Product Name	Product Number
Spectre XPS	91600 or 90004
Spectre APS	3500 (restricted for characterization), 91050, or 90004

About This Manual

The *Virtuoso Liberate MX Reference Manual* describes the Cadence® Virtuoso® Liberate MX tool. The document includes opening chapters that describe what Liberate MX does and how to get started with the tool. Later chapters discuss the commands and variables that can be used with Liberate MX.

Audience Profile

This manual is aimed at developers and designers who want to work on library creation for memory instances. It assumes that you are familiar with:

- SPICE simulations
- Basic expected behavior of the design being used

Additional Documents for Reference

For information about known problems and solutions, see *Virtuoso Characterization Suite Known Problems and Solutions*.

For a list of new features in a release, see *Virtuoso Characterization Suite What's New*.

For information about other products in Virtuoso Characterization Suite, refer to the following manuals:

- *Virtuoso Liberate Reference Manual* describes the Liberate tool—an accurate, highly efficient and easy-to-use library characterizer that creates electrical views (timing, power, and signal integrity) in formats such as the Synopsys Liberty (`.lib`) format.
- *Virtuoso Liberate LV Reference Manual* describes the Liberate LV library validator—a tool that provides a collection of capabilities used to validate and verify the data consistency, accuracy, and completeness of cell libraries.
- *Virtuoso Variety Reference Manual* describes the Virtuoso Variety process variation cell characterizer—a tool that characterizes process variation aware timing models and generates libraries for multiple statistical static timing analyzers (SSTA) without requiring re-characterization for each unique format.
- *Virtuoso Liberate AMS Reference Manual* describes Liberate AMS—a tool that provides library creation capabilities for Analog Mixed Signal (AMS) macro blocks.
- *Virtuoso Liberate API Reference Manual* describes a Tcl interface that allows access to the Liberate characterized Library DataBase (LDB).

Rapid Adoption Kits

Cadence provides Rapid Adoption Kits that demonstrate how to use Virtuoso applications in your design flows. These kits contain design databases and instructions on how to run the design flow.

Application Notes

The following application notes are available on the Cadence Online Support website:

- Setting up Liberate MX for Various Usage Models
- Liberate MX: Debugging Netlist Issues
- Using and Debugging the Liberate MX Validation Flow

- Characterizing Minimum Period and Minimum Pulse Width Using Liberate MX

Typographic and Syntax Conventions

This section describes the typographic and syntax conventions used in this manual.

`literal`	Non-italic words indicate keywords that you must enter literally. These keywords represent command or option names.
argument	Words in italics indicate text that you must replace with an appropriate value.
`< >`	Angle brackets indicate text that you must replace with a single appropriate value. When used with vertical bars, they enclose a list of choices from which you must choose one.
`|`	Vertical bars separate a choice of values. They take precedence over any other character.
`-`	Hyphens denote arguments of commands or variables. Usually arguments denoted in this way are optional but, as noted in the syntax, some are required. The hyphen is part of the name and must be included when the argument is used.
`{ }`	Braces indicate values that must be denoted as a list. When used with vertical bars, braces enclose a set of values from which you must choose one or more. When you specify a list, the values must be enclosed by either quotation marks or braces. For example, {val1 val2 val3} and "val1 val2 val3" are legal lists.

Customer Support

For assistance with Cadence products:

- Contact Cadence Customer Support

  Cadence is committed to keeping your design teams productive by providing answers to technical questions and to any queries about the latest software updates and training needs. For more information, visit: http://www.cadence.com/support

- Log on to Cadence Online Support

Customers with a maintenance contract with Cadence can obtain the latest information about various tools at: http://support.cadence.com

Feedback about Documentation

You can contact Cadence Customer Support to open a service request if you:

- Find erroneous information in a product manual
- Cannot find in a product manual the information you are looking for
- Face an issue while accessing documentation by using Cadence Help

You can also submit feedback by using the following methods:

- In the Cadence Help window, click the *Feedback* button and follow instructions.
- On the Cadence Online Support Product Manuals page, select the required product and submit your feedback by using the *Provide Feedback* box.

1 Overview of Memory Characterization

The Digital Implementation flow requires the characterization of the sub blocks of a full chip. The timing, power, and pin capacitance information is captured in a Liberty format file.

Contents of Liberty Format File

The characterization information captured in a Liberty format file includes the following:

- **Pin Information**

  The Liberty file contains information about the external characteristics of the input and output pins, that is,

  - Pin capacitance for input pins
  - Noise immunity for input pins
  - Holding strength for output pins

- **Timing Information**

  The Liberty file contains information about the timing relationships of the pins of an instance. Within the Liberty file, the following timing arcs are captured:

  - Delay and retain arcs for input to output timing
  - Setup and hold arcs of input pins related to clock
  - Minimum period and minimum pulse width (MPW) for clock pins

- **Power Information**

  The Liberty file contains information about the static and dynamic power consumed by the memory instance.

  - Static leakage for each power supply
  - Dynamic power for each power supply resulting from events on each input

Types of Memory to be Characterized

In some cases, the characterization of the memory instance is dependent on the design of the instance.

Memory instances can be divided into the following two categories:

- Those that are fully timed by the external clock
- Those that generate an internal clock to time the cycles

The internal cycle of the memory instance is timed to provide enough time for all operations to complete. The end of this internal cycle results from termination of the external clock or from completion of an internal delay loop.

These two memory types are referred to as externally-timed (only dependent on external clock) or self-timed (with a delay loop controlled internal clock).

Pin Characterization

As part of the Liberty file, each input pin is characterized for pin capacitance. Optionally, the pins can also be characterized for noise characteristics.

Pin Capacitance Characterization

Each input pin should be characterized for pin capacitance. This will be used to determine the total loading of the driving signals. Simulations should be run to determine the equivalent capacitance value based on the early stages of the circuit.

Noise Characterization

In certain advanced Liberty models such as CCSN and ECSMN, it is necessary to model the noise characteristics of the input and the output pins. This includes noise immunity of the input pins and holding strength of output pins.

Methods and techniques of noise characterization will not be covered in this manual.

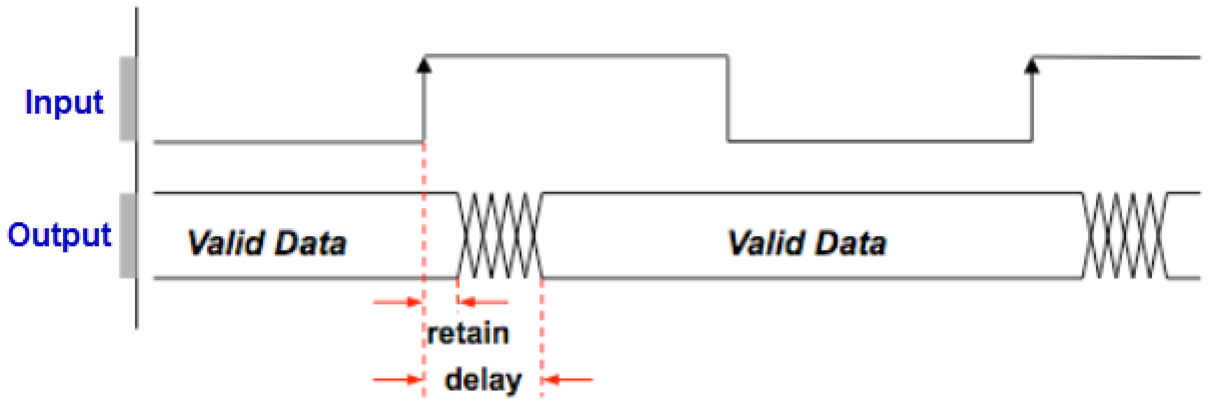
Timing Characterization

The purpose of the timing characterization of any embedded memory instance is to capture the timing relationships of the instance pins to other pins and themselves.

Delay and Retain Arcs Characterization

The delay arcs describe the timing relationship between the input and output pins. These timing arcs are used when an event on an input pin results in an event on the output pin.

The delay arc describes the worst case time for the new output data to be available. The retain arc describes the interval of time for which the previous data will be available after the input event. Retain is sometimes referred to as a minimum delay arc. The delay and retain arcs together describe the interval of time for which the output can be expected to be valid.



The use of delay and retain is especially important in a memory instance due to the range of times that the output event can occur. This is the result of multiple internal timing paths leading to the output and the bus nature of the output.

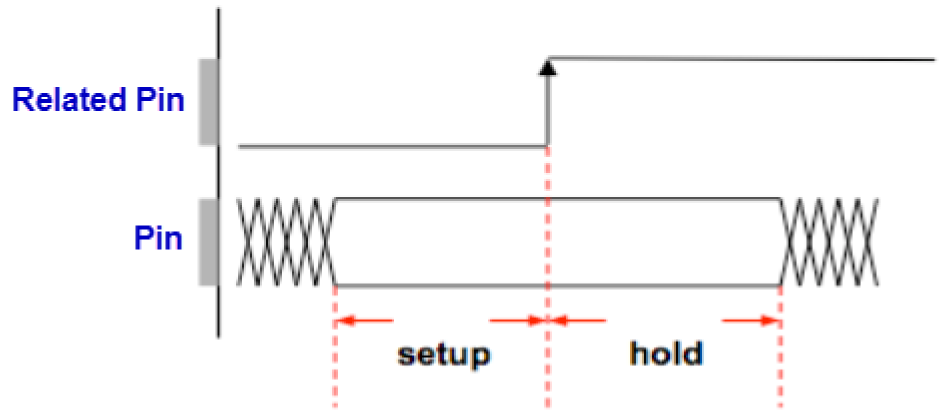
When creating input stimulus for a memory instance, it is important to capture the fastest paths and the slowest paths to ensure the correct data for both delay and retain. This means that for each output direction, rise and fall, there should be vectors that cover the following:

- Near and far output switching
- Near and far addressing
- All functional modes that can result in an output event such as read, write, bypass modes, and pipeline modes

The delay and retain values should be characterized for all input slew values and output load values.

Constraint Arcs Characterization

The constraint (setup and hold) arcs describe the required timing relationship of a pin against a related pin, usually a clock. The setup time describes the duration for which a related pin must be stable ***before*** triggering the clock edge to transition a pin. The hold time is the duration for which the related pin should be stable ***after*** triggering the clock edge to transition a pin. Both the setup and hold times are allowed to be negative; in this case, the specified transition is allowed to occur on the opposite side of the related pin event. The setup and hold together describe an interval of time surrounding the related pin event for which the pin is required to be stable.

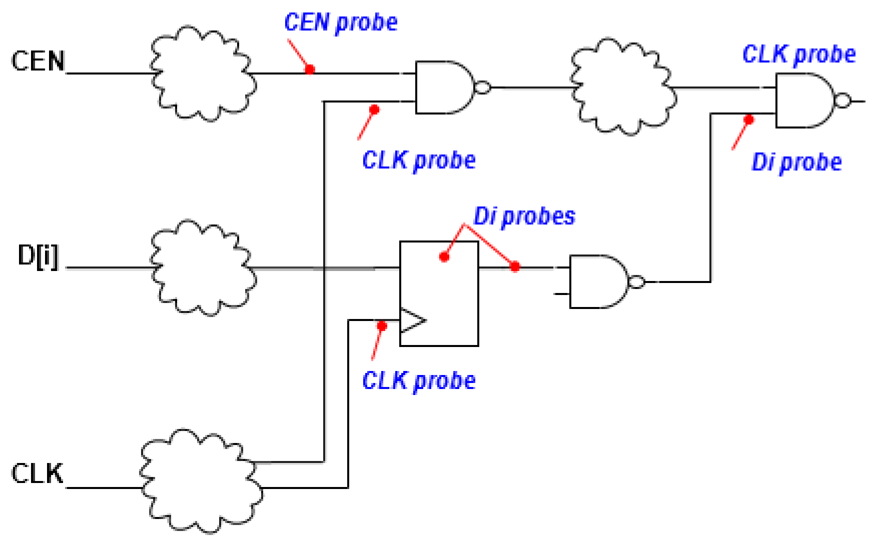


The setup and hold times need to be characterized for all values of pin and related pin input slew.

A valid setup time ensures that a pin's proper value for a cycle arrives before the clock at all locations where the clock propagation is dependent on the state of the input pins. In addition, it ensures that the proper value is stored in the input latches. The clock must not be allowed to propagate for a path that is reliant on a different state of the input.

A valid hold time ensures that a new value for the input pin will not corrupt the existing cycle. In this case, if the hold time is satisfied, the clock will prevent the propagation of the input and prevent it from corrupting the cycle.

The most common production method for characterizing setup and hold time of a memory is through the path delay method. In this method, the delay is measured for each pin and related pin, and this information is used to compute the required relationship of these two pins at the pin level. The circuit locations where the pin and related pin intersect are identified and the delay is measured from the pin and related pin to their corresponding probe locations. The difference of these delays is the setup or hold time.



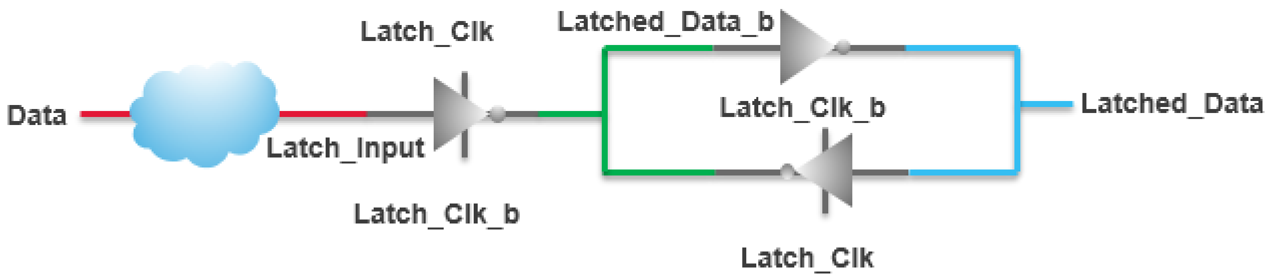
If the first stage is transparent during the situation that is being characterized for, then the following stage should be characterized as well. Typically, the first stage will be a latch and subsequent stages will be some form of combinational logic. In the case of setup time, the latch will be transparent when the setup is exercised, so the next stage should be considered as well. For hold time, when hold is exercised, the latch should be closed. So, if hold is satisfied at the latch it is not necessary to monitor later stages.

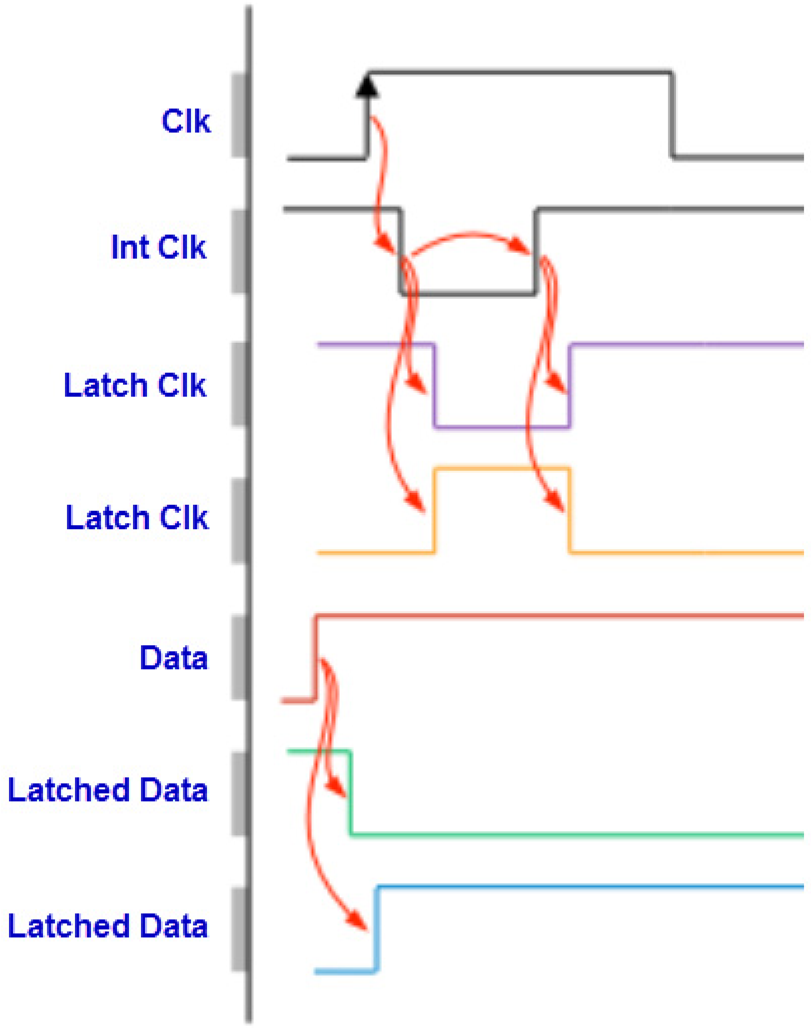
All valid probing locations should be measured and the worst case value should be captured in the Liberty file.

For pins that influence the clock propagation it is important to use the clock propagation for the correct state of the input pin. The clock propagation for signal high should be used for setup rising and hold falling. The clock propagation for signal low should be used for setup falling and hold rising. This is most important for signals that can change the clock propagation, such as, a chip enable pin.

Computing Setup and Hold Times using the Path Delay Method

As an example, let us review the computing of setup and hold times at the input latch.





For setup time, you need to ensure that the latched data is fully transitioned before the latch closes.

```
Setup Time = Max (Data -> Latched_Data_b, Data -> Latched_Data)
               -  Min(Clk -> Latch_Clk_b, Clk -> Latch_Clk)
```

For hold time, you need to ensure that the input to the latch does not switch until the latch is closed.

```
Hold Time = Max (Clk -> Latch_Clk_b, Clk -> Latch_Clk)  -  Latch_Input
```

For hold time, if the timing is satisfied at the input latch, the new data will not propagate further into the design. This means that it is sufficient to probe for hold only at the latch.

After the input latches, there can be other combinational logic where the internal clock is used to synchronize the signals that are used in the memory operation. These synchronized signals include wordlines that depend on the address value and bitlines that depend on inputs like address, data in, and write enable. These dependencies require that the internal clock and the input signals should be gated together. These locations where they meet require monitoring for setup time.

Setup and Hold Times for Bus Inputs

In many designs, buses are the input pins of an instance.

Within the Liberty syntax, the timing information is allowed under the bus or the individual bits. When the timing information is under individual bits, all bits can have the same data or each individual bit can have specific data. There can be limitations in the downstream tools to fully optimize the individual bit timing. Optimizing data by bits can be costly in terms of high simulation time. Therefore, the most efficient solution is to use the same data for all bits of the bus.

If the same data is to be used for all bits of the bus, it is necessary to identify the worst case of each bit and use that data for all bits. When doing this it is important to use the clock and data paths from the same probing location.

Tip

Do not use the clock path from input[x] and the data path from input[y].

Bisection Method

A less common approach to memory characterization is through the use of bisection methods. In this case, the pin and related pin timing relationship will be adjusted until the point of failure to determine the worst case arc value.

There are two possible criteria for determining whether an external value is valid. Delay push-out is used for cases where the output of the gate is expected to switch. This states that the output of the pin and related pin intersection cannot change by more than a specified amount when comparing a minimum timing to a relaxed timing. Glitch is used when the output of the intersecting gate is not expected to switch. In this case, the disturbance on the output signal is measured and compared to a specified pass criteria.

Bisection method tends to lead to significantly longer runtimes than path delay. This is because every characterized number will need several iterations to achieve an optimal result. For this reason, most production memory characterization is done with path delay method.

Path delay numbers will tend to be more conservative than bisection numbers because they require certain signal arrival relationships at the input of the gate rather than a non-failing output of the gate.

Bisection methods are sometimes used in memory characterization to quantify margin found in production path delay characterization.

Minimum Pulse Width Arcs Characterization

The minimum pulse width (MPW) specifies the minimum time between two consecutive opposite edges of an input signal. MPW is characterized for clock inputs and certain other asynchronous inputs.

The Liberty file will contain values for MPW varying with the input slew of the signal.

The characterization of the MPW will be very different in case of the externally timed memory and the self-timed memory.

Minimum Pulse Width in the Externally-Timed Memory

As the externally timed memory has the external clock determining the start and end of the internal operations, it is necessary to ensure that those operations complete within the clock pulse.

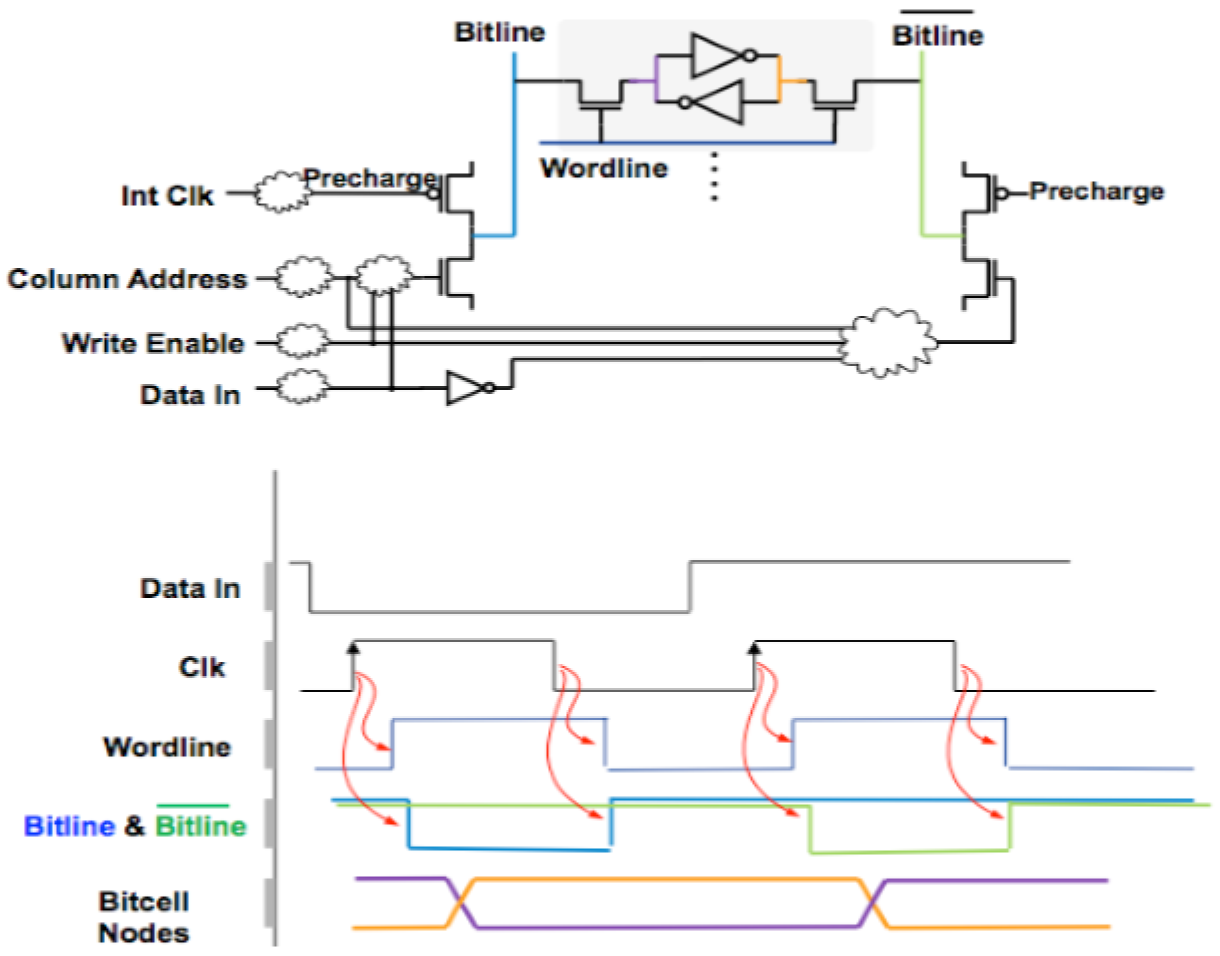
During the active portion (*usually high*) of the clock cycle, the following events must complete:

- Cell needs to reliably written for write cycles.
- Data must be reliably read for read cycles.

The inactive portion (*usually low*) of the clock cycle requires the following events to complete:

- Bitlines must be restored to their beginning of cycle state.
- Latches must open and propagate new values of input signals.

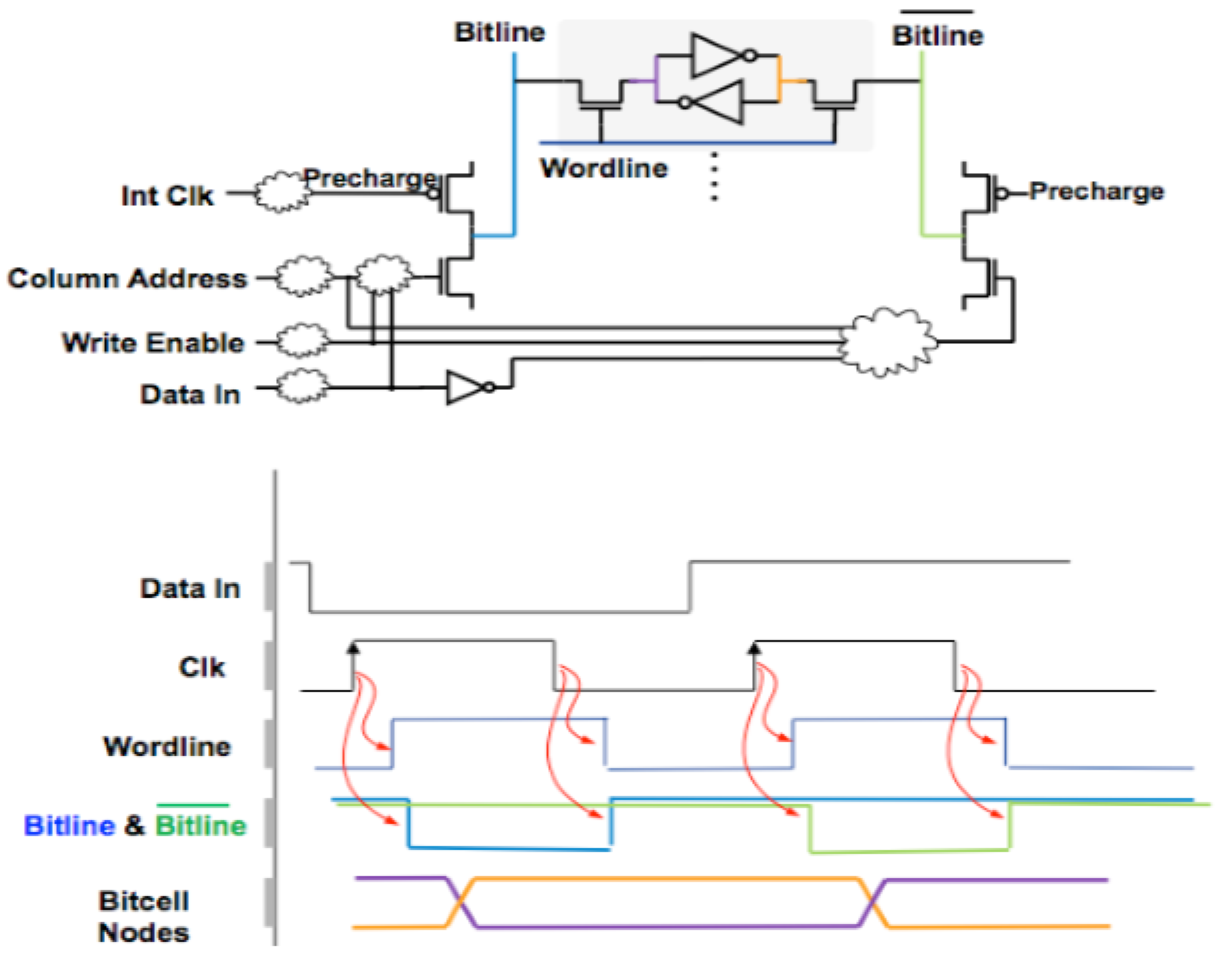
The following figure shows MPW during write:



In the figure above, the wordline and the bitline are controlled by an external clock. The wordline should stay high and the bitline should stay low, but long enough to reliably write data into the bitcell. As this is a critical operation, some margin should usually be added beyond the actual time to write the cell.

```
Minimum Pulse Width High = (Clk rise -> cell flip) - Min( ( Clk fall -> Bitline
rise) , ( Clk fall -> Wordline fall ) )
```

The following figure shows MPW during read:



In the figure above, the wordline and enabling of the sense amplifier are controlled by an external clock. To ensure a reliable read operation, the `Bitline` must sufficiently discharge before the `Wordline` falls and the `Sense Amp Enables` signal rises, as given in the equation below. The voltage requirement of the `Bitline` discharge will be design dependent.

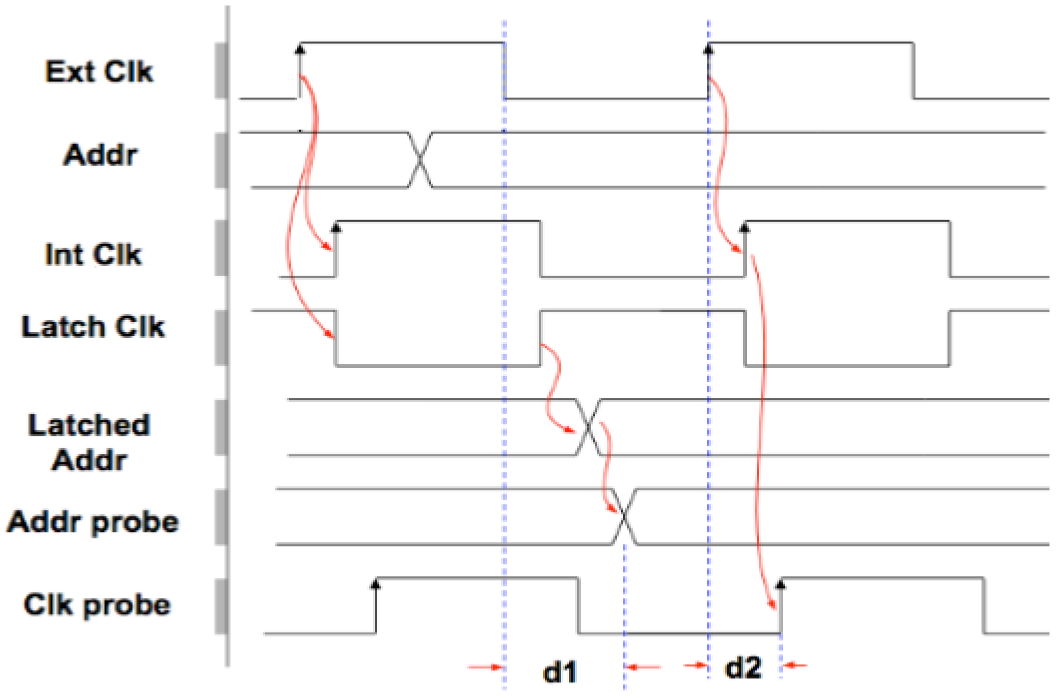
```
Minimum Pulse Width High = (Clk rise -> Bitline discharged) - Min( ( Clk fall ->
Sense Amp Enable rise) , ( Clk fall -> Wordline fall ) )
```

During the inactive portion of the cycle, it will be required that the bitlines return to their initial value before the next clock uses the bitlines. This should be measured for both the write and the read operation.

```
Minimum Pulse Width Low = (Clk fall -> Bitline returned to initial value) - ( Clk
rise -> Precharge rise )
```

The threshold used for the bitline value will be different from the threshold used for other internal signals and will usually be greater than 95%.

It will also be required that the latches open and propagate new data through before that data is needed by the clock of the next cycle. The probing locations for this component will be similar to the probing needed for the setup time of those signals. The difference will be that the probe will switch with clock instead of signal because it propagates as a result of clock opening the latch.



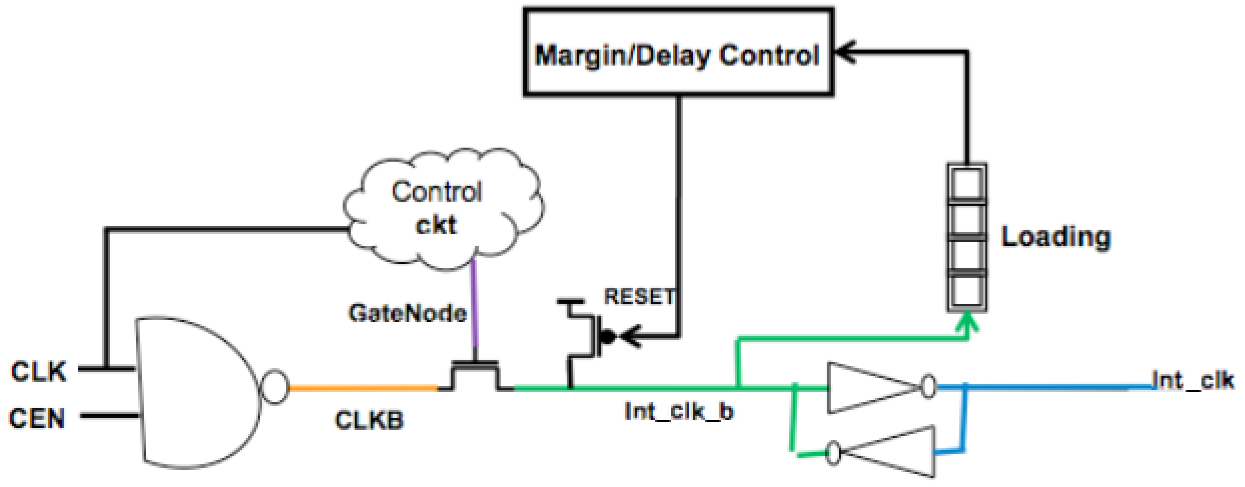
In the figure above, `Addr` probe must switch before `Clk` probe. The distance between these two events is dictated by the low pulse width (LPW) given to the `Ext Clk` signal, so that it determines the MPW.

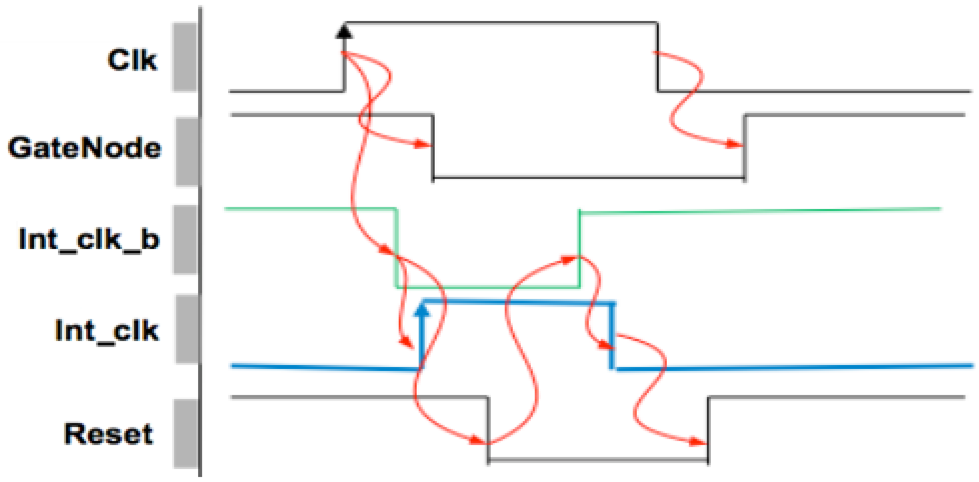
```
Minimum Pulse Width Low = (Clk fall -> Addr Probe) - ( Clk rise -> Clk probe rise )
```

The thresholds used for the internal nodes of MPW should also follow the same internal thresholds used for setup and hold characterization.

Minimum Pulse Width in the Self-Timed Memory

In the self-timed memory, the active edge of the clock is used to start the internal clock. Control over the internal clock then passes to the internal clock return. Once the internal clock returns, the internal clock terminates. The inactive edge of the external clock, combined with the internal clock return, passes control of the internal clock generation back to the external clock for the next cycle.





In the example above, the external clock needs to stay high until `GateNode` goes low. If the external clock were to go low before `GateNode`, `Int_clk` would be corrupted.

```
Minimum Pulse Width High = (Clk rise -> GateNode fall) - (Clk fall -> ClkB rise)
```

At the end of the cycle, `GateNode` needs to be high before the next `Clk` rising makes `ClkB` fall. If this is not satisfied, the generation of `Int_clk` would be delayed.

```
Minimum Pulse Width High = (Clk fall -> GateNode rise) - (Clk rise -> ClkB fall)
```

As with other internal measurements, the appropriate thresholds should be used.

Minimum Period Arcs

The minimum period defines the time that is needed between two of the same direction edges of an input signal, usually clock.

The measuring of the minimum period is significantly different for the externally timed memory compared to the self-timed memory.

Minimum Period in the Externally-Timed Memory

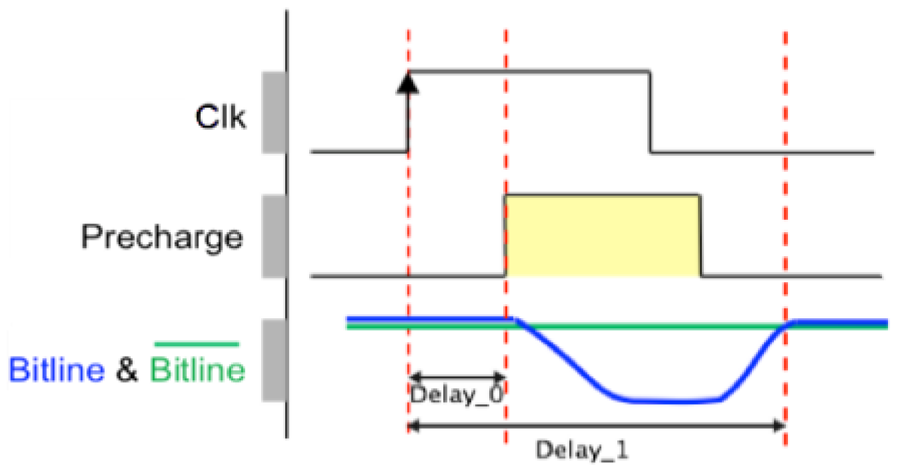
In the externally-timed memory, the two phases of a clock separately control the different portions of the active cycle. As there is no further dependence on the spacing between two like edges, the following equation is used:

```
Minimum Period = (Minimum Pulse Width High) + (Minimum Pulse Width Low)
```

Minimum Period in the Self-Timed Memory

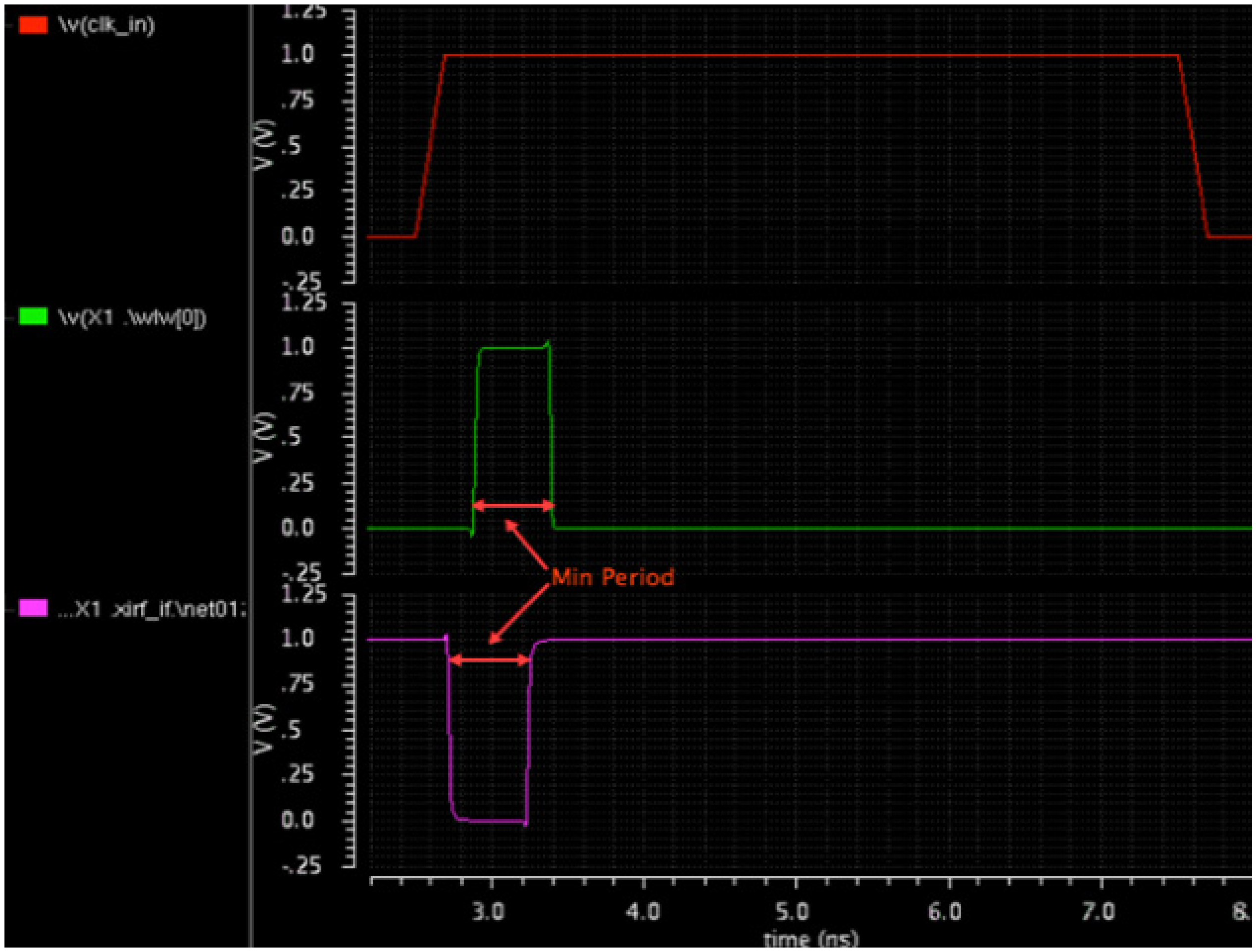
In the self-timed memory, the duration of the cycle is not determined by the values of the minimum pulse widths. These should be accounted for, but there are other requirements, listed below, for the clock cycle that need to also be satisfied.

- The sum of the MPW high and MPW low
- The bitlines must be returned to their initial state before the next cycle
- Any internal pulsing signals need to return to their initial state
- Latches must open in enough time before the next cycle to propagate the new input values



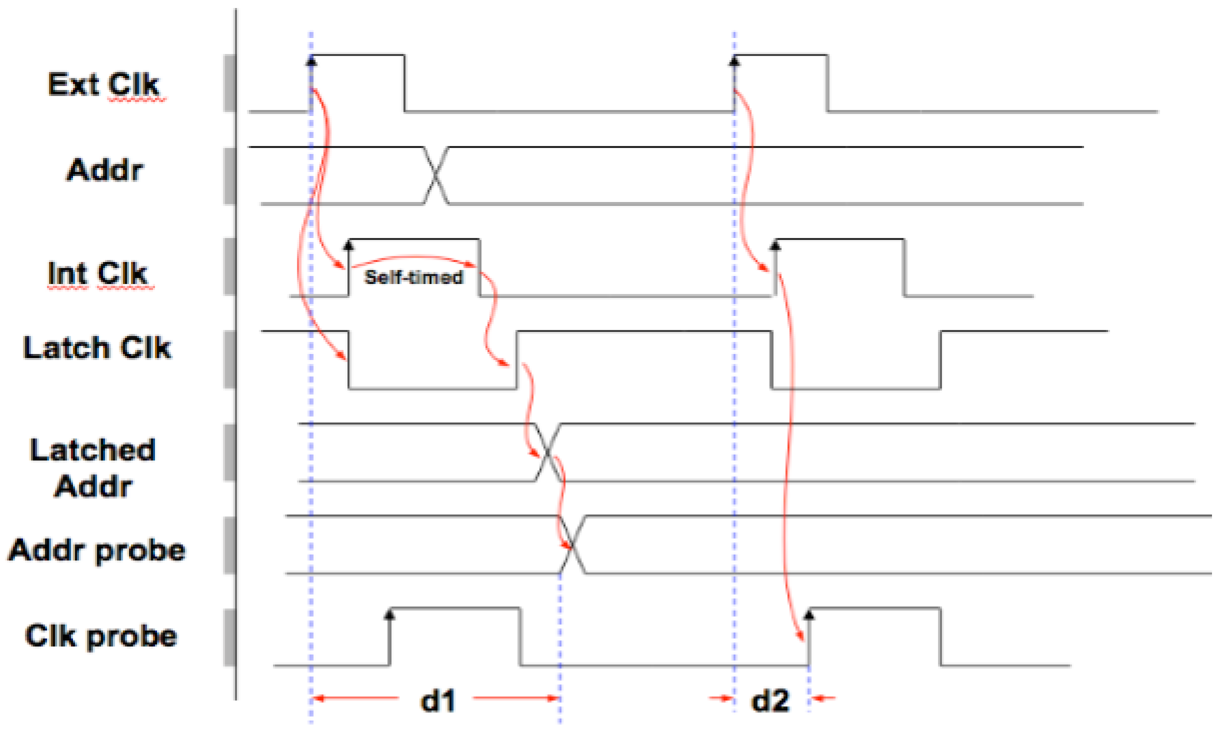
The bitline must return to its initial state (usually high) before the precharge turns off in the next cycle. This dictates the spacing between the two active edges of the clock. The threshold used for the bitline voltage will be higher than thresholds used for other internal signals and will usually be greater than 95%.

```
Minimum Period = (Clk rise -> Bitline high) - (Clk rise -> Precharge fall)
```



All internal pulsing signals must finish pulsing before they are activated again in the next cycle. This means that the minimum period must be greater than the width of any of these pulses.

The threshold used for this measurement is usually closer to the rails than the threshold used for setup and hold characterization.



The latch component of minimum period requires that the latches must open in enough time for the new data to propagate through the latch and arrive before the clock at the setup probing locations. If the external input switches while the latch is closed, the output of the latch switches with clock when the latch opens. This is the earliest the latch output can switch at the end of the cycle.

Basically, this means that even if the setup time of the input signal is satisfied, the latch needs to be open when the signal arrives there or it will be a violation. This is specified in minimum period to allow for enough time.

```
Minimum Period = (Ext Clk rise -> Addr probe) - ( Ext Clk rise -> Clk probe rise)
```

The thresholds that should be used here are the same as those used in setup time characterization.

Power Characterization

The purpose of the power characterization of an embedded memory instance is to capture the power effect of each of the pins as well as the standby leakage. This information is then used to compute the total power consumed by the memory instance.

Static Power (Leakage) Characterization

The static power or leakage is the nominal amount of power consumed by the instance, even if there is no activity. To measure this, the design should be configured idle for a long period of time to eliminate any transient effects. The current should be measured for each power supply.

The static power can be affected by the state of leakage reduction modes in the instance. These modes reduce the static power by placing the circuit into a lower leakage state. These modes would usually be captured in the `.lib` file as `when` conditions on the static leakage.

Dynamic Power Characterization

The dynamic power is the power consumed by the memory during active operation. The power is characterized separately for each input and output pin. Depending on which pins are active in a cycle, the power is summed to arrive at the total power consumption.

Dynamic power is measured separately for all power supply pins.

Power for clock pins is characterized by leaving all other pins stable and toggling the clock. This measurement should be done for the read and write operations separately. During the read operation, the output should not change to ensure that the output power is not counted. This is usually done by reading the same address location on consecutive cycles. For the write operation, there should be a predictable number of switching core cells, usually half. This is accomplished by writing a known pattern to the memory and then writing again to measure power.

The power for the other input pins should be characterized by toggling that pin separately from other pins and measuring the power consumed.

Power for output pins switching is the difference between a similar cycle with the output switching and a cycle without the output switching. This isolates the power contribution of the output pin and can be used to determine total power.

2

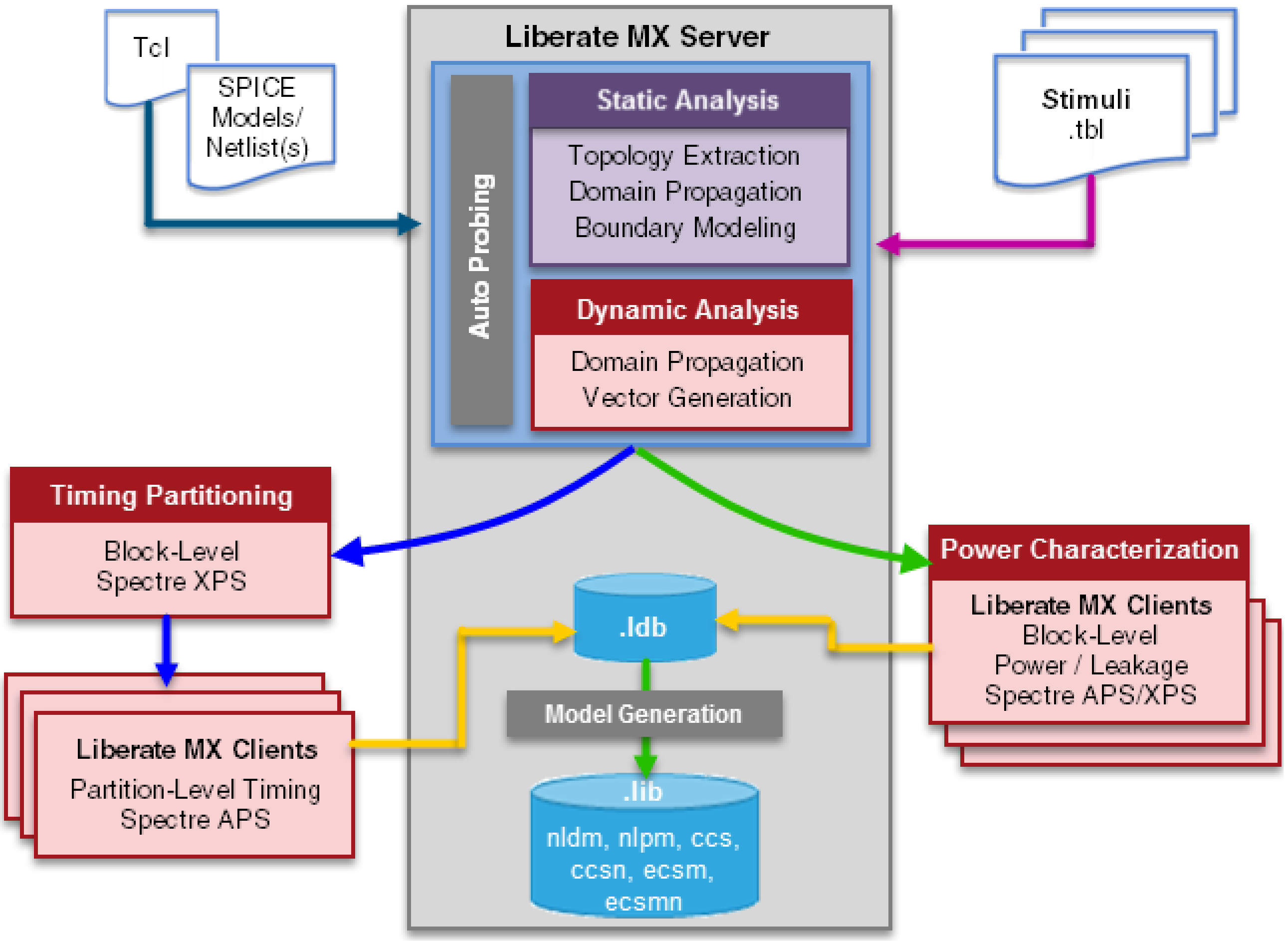
Overview of Liberate MX

What is Liberate MX?

Cadence® Virtuoso® Liberate™ MX provides library creation capabilities to cover the memory cores. Embedded and custom memories comprise a large percentage of silicon area on most chips and consequently, can often be major contributors to chip performance and power consumption. To validate a design's electrical performance, it is essential to have a highly accurate electrical model for each memory equivalent in accuracy of the electrical models used for standard cells and I/Os. Using pre-packaged models from an IP provider or memory compiler might not be sufficiently accurate, especially because the exact context of the memory is not known until it is placed on the chip. It is common, for example, to operate a memory block at a lower voltage to save power. To get an accurate electrical model that reflects the exact usage of the memory a design-specific, instance-specific characterization of each block is required.

Liberate MX implements additional techniques of spatial analysis, automatic probing, and temporal partitioning to make accurate characterization feasible. Using the divide and conquer algorithm, memories and cores circuits are reduced to manageable sizes and used by the unique Inside View technology of Liberate that optimizes the characterization run time. Consequently, the memories and cores are characterized with the same accuracy and techniques as standard cells. For detailed command and parameter description of standard cell library generation, refer to *Virtuoso Liberate Reference Manual*.

The completed libraries, which Liberate MX produces in industry-standard formats such as the Synopsys Liberty (.lib) format, can then be used to analyze the static timing and power performance for full chips that include the characterized memory instance.



Liberate MX works in server and client model, where the server takes the primary inputs and does the pre-processing (netlist processing, topology extraction, boundary modeling) and creates the database. Liberate MX clients are used for simulations and creating partitions.

The Liberate MX flow consists of two main steps: preprocessing and characterization.

In the preprocessing step, spatial partitioning techniques are used to identify the following basic building blocks:

- nand, inverter, latch, flop, arrays
- the clock trees
- internal probes of interest, such as latch/flop data

- clock nodes for constraint checks
- input and output nodes of combinational clock gates

User-provided stimuli—or truth tables—are then used to drive an activity-based temporal partitioning step to extract the transistor sets to be sent to characterization.

In the characterization step, partitions created from the first stage are characterized using a full SPICE simulator such as Spectre APS.

3

Getting Started with Liberate MX

This chapter describes briefly how to start using Cadence® Virtuoso® Liberate™ MX. A systematic approach to tool setup is covered here with the intent to help new users of the tool. Once you are familiar with the tool, these can be refined.

Tool Installation and Setup

Installing Liberate MX

To install Liberate MX:

1. Familiarize yourself with the installation tools, InstallScape and the license manager.

   To find guidance materials for these tools, see http://support.cadence.com.

2. To obtain the Liberate MX software, see http://downloads.cadence.com.
3. Select the *LINUX* tab.
4. Select the appropriate release (for example, `LIBERATE151`).

   The first two numbers in the release name designate the year of the release and remaining numbers begin with one and increment with each additional release during that year. Therefore, `LIBERATE151` is the first LIBERATE base release of 2015.

5. Download and install the product.

   This step utilizes the tools, InstallScape and the license manager, that you learned about in step 1.

6. Use commands such as the following to include Liberate MX in your software path.

```
% setenv ALTOSHOME <install_dir>/<liberate_release_name>
% set path=($path $ALTOSHOME/bin)
```

7. Set the following to include integrated Spectre in your executable path:

```
% set path ($path $ALTOSHOME/tools.lnx86/spectre/bin)
```

Managing Licenses

Liberate MX uses a server-client licensing scheme. The tool uses a server license for memory preprocessing and for starting and monitoring the characterization run on the server machine. It uses the client licenses for running characterization on the client machines and for any postprocessing of the library database. Each Liberate MX server can access all the available client licenses. For example, with two server licenses and forty client licenses the following configurations are all valid:

- A single characterization run using 40 client processes
- Two simultaneous characterization runs, each with 20 client processes
- Two simultaneous characterization runs, one with 30 client processes and one with 10 client processes

Important

Ensure that the license daemon (`cdslmd`) and the license server (`lmgrd`) have the same version and that this version is the same as that required for a release. For example, v11.11.1 is required for the Liberate 15.1 release. If a mismatch is detected, unexpected license behavior might be observed. For example, the license search path can be reset to `<none>` after a failed license check out request. This can result in incorrect license checking in process.

On a 64-bit license host, the 64-bit `cdslmd` and `lmgrd` must be used instead of the default 32-bit ones.

Waiting for Available License

When you submit a Liberate MX job, a request is made for a license. To request that Liberate MX wait until a license becomes available, set the following environment variable:

```
setenv ALTOS_QUEUE 1
```

How Liberate MX Uses Licenses

Liberate MX clients run using different types of client license features, depending on the product names. Some product licenses can be mixed and matched together. Liberate MX clients can run using the `Liberate_MX_Client` product license.

When a Liberate MX server starts, it checks out the `Liberate_MX_Server` license. Later, when simulations are ready to begin, Liberate MX tries to check out N clients, where N is the number of threads specified in the `char_macro -thread` command option.

When `ALTOS_QUEUE` is set to 1,

- If Liberate MX acquires fewer than N licenses, it waits for up to the value of the lic_max_timeout variable, trying to get all N licenses. After `lic_max_timeout` is reached, if Liberate MX acquires M licenses and M > 0, it starts M threads.
- If M is 0, Liberate MX again waits for another `lic_max_timeout` seconds to acquire client licenses. This process is repeated until at least one client license can be checked out.

When `ALTOS_QUEUE` is set to 0,

- If Liberate MX acquires all N licenses, it starts simulations using N threads.
- If Liberate MX acquires M licenses, 0 < M < N, it starts M threads.
- If Liberate MX does not acquire a license, it quits.

Environment Variables for Controlling Licensing Checks

- ALTOS_LIC_MAX_TIMEOUT

  ```
  setenv ALTOS_LIC_MAX_TIMEOUT <value>
  ```

  where;

  `value` is duration in seconds.

  This shell variable specifies the duration, in seconds, for which Liberate MX (both server and client) should wait for licenses.

  For a server process, if the `ALTOS_QUEUE` variable is enabled, Liberate will attempt to check out one server license. If the maximum timeout is reached, and no server license has been checked out, then Liberate will reset the timer and loop back to continue waiting for a license. For a client, when the maximum timeout is reached and at least one license was checked out, the Liberate client will start to run with the licenses it has. No additional licenses are checked out.

- ALTOS_LIC_CHECK_ALT_TIMEOUT

  ```
  setenv ALTOS_LIC_CHECK_ALT_TIMEOUT <value>
  ```

  where;

  `value` is duration in seconds.

  Some Cadence characterization products can run using more than one product license. This variable controls both the server and client timeout before trying to check out an alternative license feature if there are any such licenses in the license pool.

System Libraries

Liberate MX is shipped with dynamic-linked system libraries. To verify Liberate MX is capable of running on your system, just try executing it. If Liberate MX fails to start properly, it may be possible that you have an old system and that there are missing or incorrect system libraries. If this occurs and you have already checked your environment setup is correct, you can try using static-linked binaries by setting the following environment variable:

```
setenv ALTOS_USE_STATIC_BINARIES 1
```

Inputs Required for Liberate MX Characterization

Various types of data are required to run Liberate MX:

1. Extracted memory block netlists in SPICE format.
2. Foundry device models in SPICE format.
3. A Liberate MX command file in Tcl format.
4. An input stimuli file.

The files required to get started with Liberate MX characterization can be grouped into the following two categories:

1. **Simulation setup files**: These files are typically available by the time an instance needs to be characterized. This set of files consists of:

   a. **Netlist for the instance (Specifying extracted memory block netlists)**

      The transistors, diodes, resistors, capacitors, and extracted parasitic elements (RCs) that compose the memory top-level netlists are passed to Liberate MX in SPICE format. Spectre, Berkeley SPICE, and HSpice® netlist formats are supported. The information passed as files to Liberate MX must include a `.subckt` definition for the block to be characterized. To specify the extracted netlists, use the `read_spice` Tcl command, as shown below:

      ```
      read_spice {sram.lpe}
      ```

   b. **Model files (Specifying and using foundry device models)**

      Device models, which represent the electrical parameters of the devices, are supplied by foundries. The device models include P- and N-channel transistors, diodes, capacitors, and resistors. Most device model files include different parameters for different process corners, such as, a typical corner, a fast corner, and a slow corner. To prepare for a Liberate MX run, you must include the device model files and specify the operating conditions.

c. Vector or testbench

Vector is needed to set up the circuit in wanted state and transition the output and/ or internal nodes to exercise desired arc. The importance of vector is to help the tool to understand what input transition causes what output or internal node transitions.

❍ ***Truth table format***: User can write the stimulus in truth table format. For more information, see Appendix A, "Truth Table Format."

Note: In automatic flow, truth tables are created by Liberate MX.

2. **Files needed to setup characterization**: This group of files needs to be created. It contains:

a. Template file

A template file is used to define or specify the arcs needed from a characterization run. There are three primary sections in this file that you need to create.

❍ ***Template definitions***: input/clock slew, output load (`define_template`)

❍ ***Cell definition***: details of input/output pins in design

❍ ***Arc definitions***: timing, power, and leakage arcs

A template file can be created in the following two ways:

i) Manually create a `template.tcl` file where content can be written in the three sections listed above.

ii) If an existing library (`.lib`) is available for the block (maybe from an older process node), it can be read in and used to create the template file.

Use the `write_template` Tcl command to read in the reference library that you provided and generate a template or macro (`template.tcl`), which is used as an input file for the next script.

```
read_library [pwd]/ref.lib
write_template -verbose [pwd]/tmpl/template.tcl
```

See also Setting Up a Template File: An Example.

b. Primary setup Tcl file

The commands that run in Liberate MX are Tcl commands, typically embedded in Tcl scripts. It is convenient to create a shell file to run the various Tcl scripts in the proper sequence. The Tcl commands available for Liberate AMS are detailed in Chapter 5, "Liberate MX Commands."

Tcl scripts are used to specify the memory instance netlist, SPICE models, and operating conditions. Tcl scripts also define the range of data, such as, input slew and output-loading conditions, for the characterization. Liberate MX simulates and measures the MX netlist using each of the specified input slews and loads and generates the appropriate delay tables, timing checks (setup, hold, and so on) and power information (switching power, hidden power, state-dependent leakage). Liberate MX can also generate library information for the composite current source (CCS) model and the effective current source model (ECSM), for both timing and noise.

For an example of a setup Tcl file, see the Setting Up Manual Characterization Flow (Full Custom Flow) section in Liberate MX Flows.

Setting Up a Template File: An Example

The template file, `template.tcl`, contains the characterization information about the slews and loads to be characterized, the pinout of the instance specified using the define_cell command, and the define_arc statements for all the arcs that need to be in the final library file. The example `template.tcl` in this section shows how you can manually define arcs for a simple Input/Output (I/O) test case:

```
#%%%%%%%%%%%%%%%%%%%%%%%% template.tcl starts %%%%%%%%%%%%%%%%%%%
 ##################################################
# defining slews and measurement thresholds
##################################################
set_var slew_lower_rise 0.1
set_var slew_lower_fall 0.1
set_var slew_upper_rise 0.9
set_var slew_upper_fall 0.9

set_var measure_slew_lower_rise 0.3
set_var measure_slew_lower_fall 0.3
set_var measure_slew_upper_rise 0.7
set_var measure_slew_upper_fall 0.7

set_var delay_inp_rise 0.5
set_var delay_inp_fall 0.5
set_var delay_out_rise 0.5
set_var delay_out_fall 0.5

set_var max_transition 6e-09
set_var min_transition 2e-11
set_var min_output_cap 5e-15

set cells { \
    rf_top \
}

##################################################
# Template definitions
##################################################
define_template -type delay \
    -index_1 {0.16 0.5 1.6 } \
    -index_2 {0.04 0.08 0.2 } \
    delay_template

define_template -type constraint \
    -index_1 {0.16 0.5 1.6 } \
    -index_2 {0.16 0.5 1.6 } \
    constraint_template

define_template -type power \
    -index_1 {0.16 0.5 1.6 } \
    -index_2 {0.04 0.08 0.2 } \
    power_template

if {[ALAPI_active_cell "rf_top"]} {

##################################################
# Top-level design definition
##################################################
define_cell \
    -clock { clk_in } \
    -input { add_in[6:0] chip_en data_in[31:0] wr_in } \
    -output { data_out[31:0] } \
    -delay delay_template \
    -power power_template \
    -constraint constraint_template \
    rf_top

##################################################
# Overriding pin-wise slew indexes (if needed)
##################################################
```

```
define_index -pin {clk_in} -type power \
    -index_1 {0.2 0.3 0.4} \
    -index_2 {0.04 0.08 0.2 } \
    rf_top

###################################################
# ARC definitions
###################################################
#--------------
# Leakage
#--------------
define_leakage rf_top

#--------------
# Delay arc
#--------------
define_arc \
    -related_pin_dir R -pin_dir F  \
    -related_pin {clk_in} \
    -pin {data_out[31:0]} \
    rf_top

#--------------
# Retain arc
#--------------
define_arc \
    -type retain \
    -related_pin_dir R -pin_dir F  \
    -related_pin {clk_in} \
    -pin {data_out[31:0]} \
    rf_top

#--------------
# Setup arc
#--------------
define_arc \
    -type setup \
    -related_pin_dir R -pin_dir R  \
    -related_pin {clk_in} \
    -pin {add_in[6:0]} \
    rf_top
```

```
#--------------
# Hold arc
#--------------
define_arc \
    -type hold \
    -related_pin_dir R -pin_dir R  \
    -related_pin {clk_in} \
    -pin {add_in[6:0]} \
    rf_top

#--------------
# MPW arc
#--------------
define_arc \
    -type mpw \
    -pin_dir R  \
    -related_pin {clk_in} \
    -pin {clk_in} \
    rf_top

#--------------
# minimum_period arc
#--------------
define_arc \
    -type min_period \
    -pin_dir R  \
    -related_pin {clk_in} \
    -pin {clk_in} \
    rf_top

#--------------
# power arcs
#--------------
define_arc \
    -when "wr_in" \
    -type power \
    -pin_dir R  \
    -pin {clk_in} \
    rf_top
```

```
define_arc \
    -type power \
    -pin_dir R  \
    -pin {add_in[6:0]} \
    rf_top

define_arc \
    -type power \
    -pin_dir R  \
    -pin {data_out[31:0]} \
    rf_top
}
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% template.tcl ends %%%%%%%%%%%%%%%%%%
```

Running Liberate MX

Before using Liberate MX, make sure that it is installed correctly and that all the necessary prerequisite data are available.

To use the 64-bit port of Liberate AMS, set the `ALTOS_64` environment variable prior to running the tool, as shown below:

```
% setenv ALTOS_64 1
```

Start the characterization by typing `liberate_mx` followed by the Tcl command file, as shown below:

```
% liberate_mx mx.tcl
```

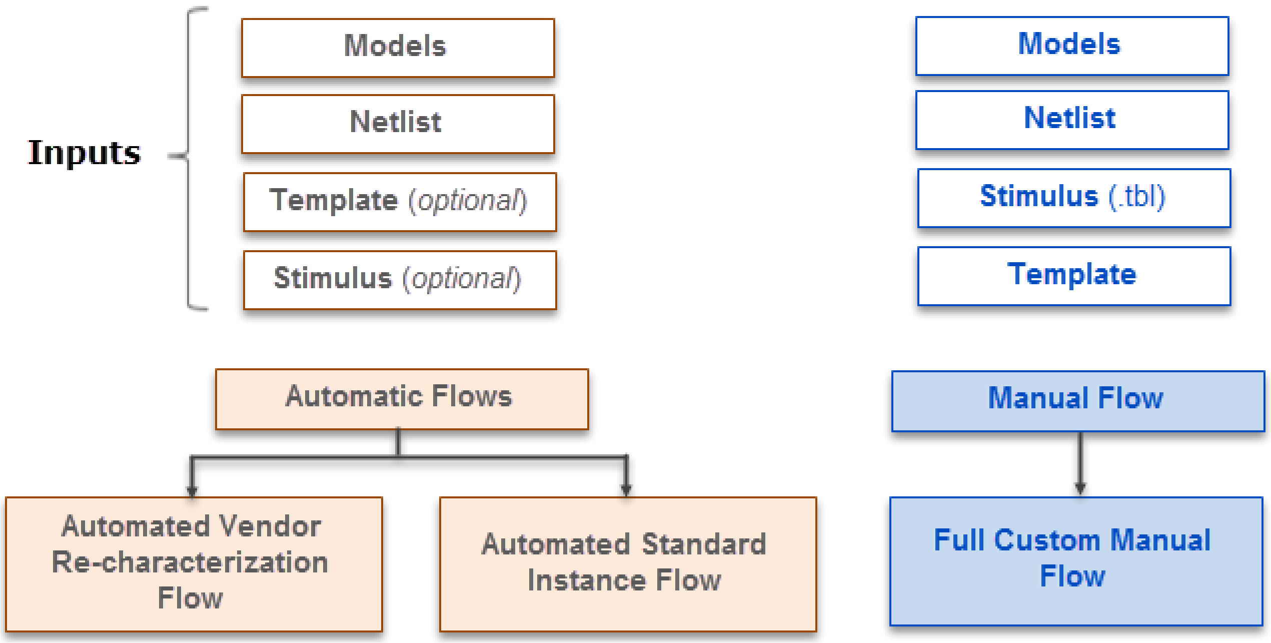
By default, Liberate MX utilizes `stdout` and `stderr` for messages and does not create a log file. However, to run Liberate MX so that it uses a log file, you can use a command such as following:

```
% liberate_mx mx.tcl |& tee mx.log
```

4

Liberate MX Flows

Liberate MX supports multiple use models for characterization that are classified as automatic flow and manual flow. The use model that is selected is determined by the instances that need to be characterized.



To validate whether a memory instance functions as per your requirements, Liberate MX provides the validation flow where you can simulate the memory instance and study the timing arcs. The validation flow can also be classified as automatic and manual.

This chapter discusses the following flows and how Liberate MX functions in them:

- Characterization Flows
    - Automatic Characterization Flow
        - Automated Vendor Re-characterization Flow
        - Automated Standard Instance Flow

- Manual Characterization Flow
- Validation Flow

Characterization Flows

This section covers information about the automatic and manual characterization flows available in Liberate MX.

Automatic Characterization Flow

The Liberate MX automatic flow (also called the define_memory flow) is intended to provide a simplified interface for characterizing standard instances and instances obtained from commercial IP vendors. There are certain criteria that an instance should satisfy in order to use the automatic flow. This section briefly discusses these requirements. For detailed usage and debugging information, refer to Appendix C, "Using Liberate MX Automatic Flow."

Automated Vendor Re-characterization Flow

In case of the vendor instances, the requirements are as follows:

- The instance should be from one of the main IP vendors (Virage, ARM, or TSMC)
- The design is known and supported by the flow. The designs that are supported by the flow are the standard available designs. These include single and dual port, SRAM, register files, and ROM designs.

When the automated vendor re-characterization flow is invoked, Liberate MX internally creates a custom setup base for the requested design requiring minimal input beyond the existing files generated from the compiler.

Automated Standard Instance Flow

For the non-vendor instances, you can describe the instance in terms of the functions of its pins. To ensure that the automated standard instance flow is an appropriate solution, the instance must satisfy the following criteria:

- The behavior of instance should be such that if the instance is enabled and in write mode, the contents of data in are written to the location specified by the address when the clock is activated.

- The behavior of instance should be such that if the instance is enabled and in read mode, the contents stored in the location specified by the address are reflected on data out when the clock is activated.
- Input signals are all latched by the clock signal
- Output signals are latched
- There are no disallowed combinations of address or data in. If the upper address locations do not exist, the location can be specified.
- The design is a SRAM or a ROM based design. This flow does not support CAM, CAMRAM, or flash.

Setting Up Automatic Flow

The Liberate MX automatic flow requires the following files:

- Tcl file containing the define_memory and char_memory commands
- Instance netlist
- SPICE or Spectre model files
- Reference library file or template file (*optional*)
- Tcl file containing any additional needed settings (*optional*)
- Additional table files (*optional*)

Primary Tcl File

The primary Tcl file of the automatic flow contains the `define_memory` and `char_memory` commands. The purpose of the `define_memory` command is to define the instance and the characterization conditions. The `char_memory` command runs the characterization.

The `define_memory` command is used for both vendor instance re-characterization and standard instance characterization.

- **Automated Vendor Re-characterization Flow**

  In case of automated vendor re-characterization flow, you need to use the `define_memory` flow to define the vendor, the instance netlist, and the Process, Voltage, and Temperature (PVT) conditions. The pin function and rail definitions are not required because the naming conventions of the pins and rails are known.

  ```
  define_memory \
  ```

```
    -ref_lib [pwd]/ref_lib/SRAM_2048x16.lib \
    -netlist [pwd]/netlist/SRAM_2048x16.spf \
    -vendor "TSMC" \
    -global_voltage 1.1 \
    -temp 25 \
    -models [pwd]/models/include_tt_model.sp \
    -mx_setting settings.tcl \
    SRAM_2048x16
```

Some common options available in the `define_memory` command to define the design that is being re-characterized are listed below:

`-vendor`	The IP vendor who designed the instance. Currently, the following IP vendors are supported: TSMC, ARM and Virage.
`-process_node`	The process node of the instance. This is used to identify process node-specific design characteristics for the instance.
`-cfg_file`	The configuration file generated by the compiler can sometimes be useful to define `process_node` and other design characteristics.
`-compiler_name`	The name of the generating compiler.

- **Automated Standard Instance Flow**

In case of automated standard instance flow, you need to define the netlist, the pins and their functions, and the PVT information.

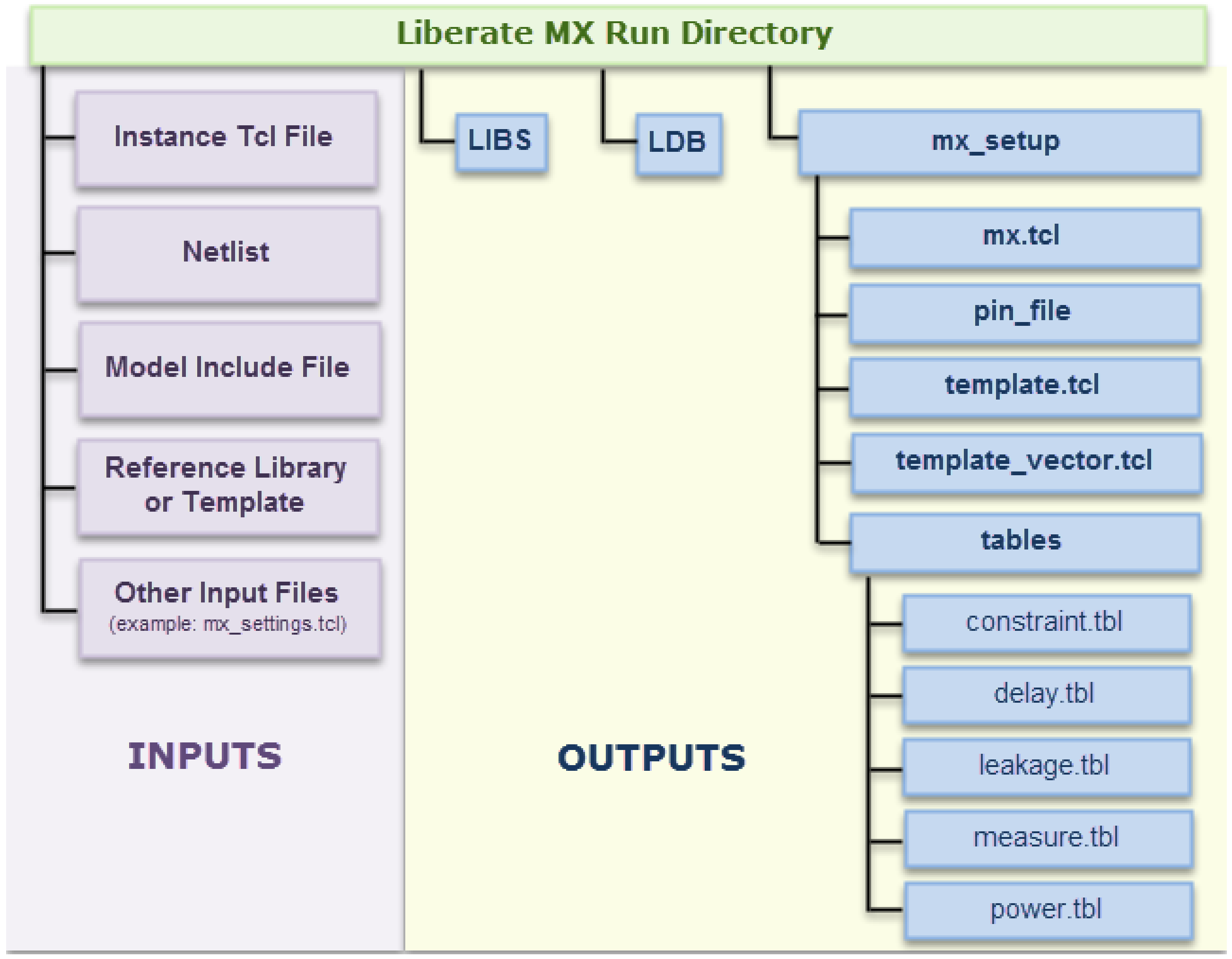
```
define_memory \
    -netlist SRAM_2048x16.spf \
    -ref_lib SRAM_2048x16.lib \
    -clock CLK \
    -address ADR \
    -data_in DIN \
    -data_out Q \
    -chip_enable {CEN L} \
    -write_enable {WEN L} \
    -rail {VDD 1.0 VSS 0} \
    -temp 25 \
    -foundry TSMC \
    -mx_setting settings.tcl \
    SRAM_2048x16
```

In both the automatic flows, the arc definitions can be passed in the form of the reference library using the `-ref_lib` option. In this case, Liberate MX characterizes the arcs exactly as defined in the reference library file. If it is necessary to modify the arc definitions, it is best to use the `read_library` and `write_template` commands to create the template and modify it. The template can then be included using the `-template` option. If there is no template or reference library available, Liberate MX creates arcs and `when` conditions based on the pins described in `define_memory`.

The temperature will always need to be defined using the `-temp` option.

Additional Liberate MX commands and options can be specified in a Tcl `include` file. This file needs to be sourced immediately before you run characterization to override all previous definitions. Use the `-mx_setting` option to specify the Tcl file containing Liberate MX commands and options.

When you run the Liberate MX automatic flow, a directory structure as illustrated below is created with all of the files needed for the characterization:



Manual Characterization Flow

The Liberate MX manual characterization flow (also called the full custom flow) provides you the flexibility to customize the steps involved in characterization as per your own requirements.

Setting Up Manual Characterization Flow (Full Custom Flow)

In the full custom flow, all the settings, commands, and vectors are provided to stimulate the memory instance and configure Liberate MX. This data is provided in the form of table files for the vectors and `.tcl` control files to configure the Liberate MX variables.

For setting the full custom flow, the required vectors, commands, and configuration settings should be provided to Liberate MX. The `mx.tcl` file is the main file in this flow. In this file, you specify the Liberate MX options and the path to the other needed files such as the template, netlist, models, and table files.

Note: The commands highlighted in **`bold`** typeface are mandatory or minimum requirements to get correct behavior from the tool.

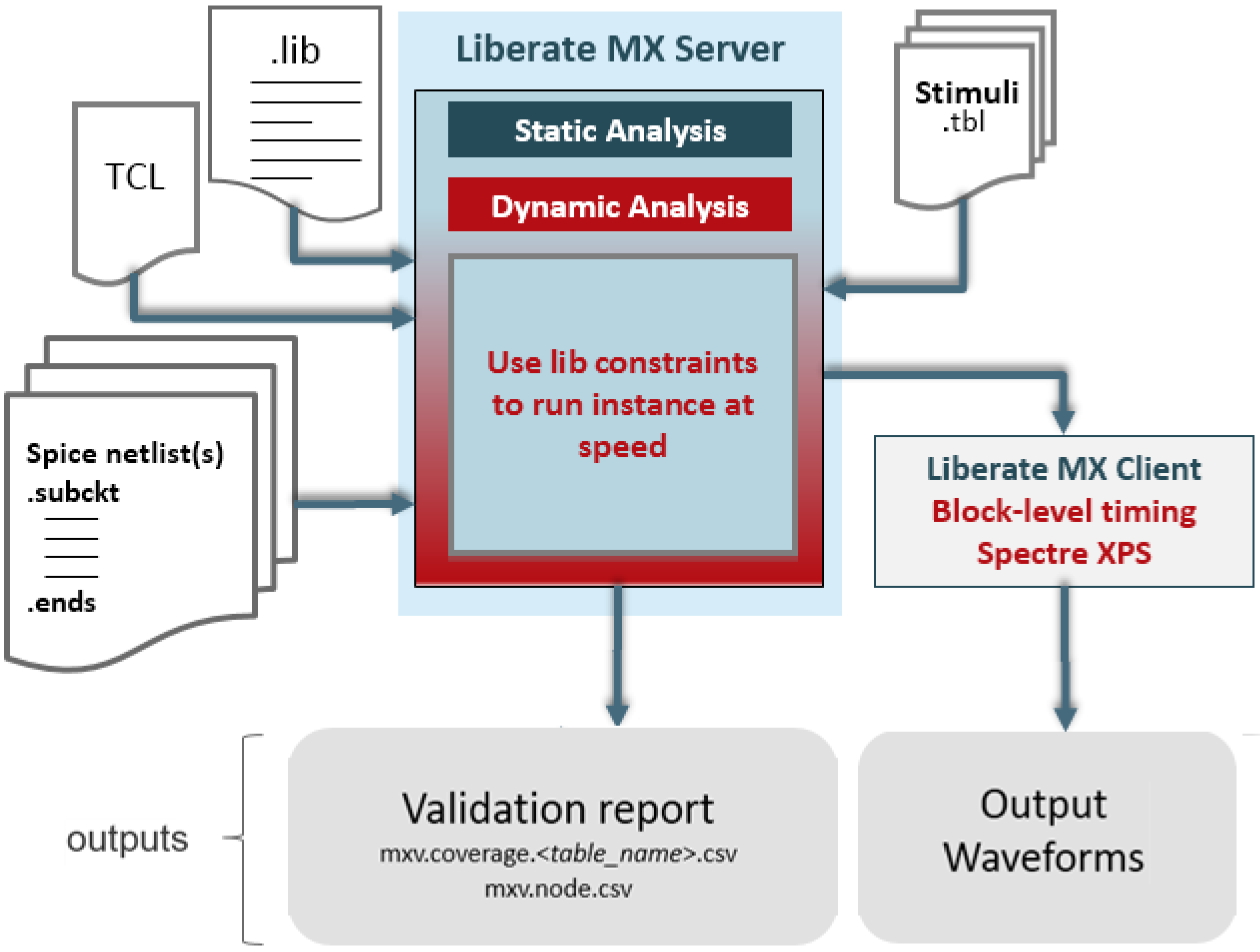
```
# SETUP DIRECTORY STRUCTURE
    set cell sram
    set datadir [pwd]

# INPUT FILES
    set tcldir   ${datadir}/tcl
    set tmpldir  ${datadir}/tmpl
    set tbldir   ${datadir}/tbl
    set spicedir ${datadir}/spice

# SET CORNER
    set_operating_condition -temp 25 -voltage 1.2

# SPECIFY RAILS
    set_vdd VDD 1.2
    set_gnd VSS 0

# LEAFCELL
    define_leafcell -type nmos -pins {0 1 2 3} {xmn xmn_sram}
    define_leafcell -type pmos -pins {0 1 2 3} {xmp xmp_sram}

# READ IN SPICE
    set modelfile ${spicedir}/include_${process}
    set_var extsim_model_include $modelfile

    set netlistfile ${spicedir}/${cell}.spf
    set_var extsim_use_node_name 1
    set_var extsim_deck_include 1
    read_spice $netlistfile

# DEFINE MACRO
    source ${tmpldir}/template.tcl

# DESCRIBE FUNCTIONALITY
    set tbls [list ${tbldir}/delay.tbl ${tbldir}/const.tbl ${tbldir}/power.tbl
    ${tbldir}/leakage.tbl ${tbldir}/mpw.tbl ${tbldir}/minp.tbl ]
    define_table $tbls $cell

# SETUP PARTITION/PROBING/CHAR PARAMETERS
    set_var mx_dynamic_include_full_core 1
    set_var sim_init_condition "ic"
    set_var mx_remove_rc_timing "rail"
    set_var mx_remove_rc_pincap "rail"

# MAIN COMMAND
    set part_sim "xps"
    set char_sim "aps"
    char_macro -extsim [list $part_sim $char_sim] -ccs

# WRITE MODELS
    write_ldb ${ldbdir}/${cell}.ldb
    write_library -overwrite -filename ${libdir}/${cell}.lib ${cell}
```

The netlist and model files are included by using the `read_spice` command. The successful reading of the netlist is always required, but the models only need to be read if the netlist contains MOS devices, as opposed to macro model devices. The models should always be specified using `extsim_model`.

The template file is included along with the `mx.tcl` file. This file contains the information about the slews and loads to be characterized, the pinout of the instance specified using the define_cell command, and the define_arc statements for all the arcs that need to be in the final library file. If there is a reference library file, the template file can be automatically generated by using the `read_library` and `write_template` commands.

Validation Flow

Liberate MX validation flow demonstrates how the memory will function with different timing values specified in the library file. This is done by simulating the memory with minimum setup, hold, and period. A passing result demonstrates the accuracy of the library file numbers. This is particularly useful to determine if a number is not sufficiently conservative.



The validation run is a top-level simulation and it is a single step process with no partitioning.

The validation flow runs two simulations per vector table. The first simulation is run with relaxed timing and is run to determine the correct functionality of the vectors. The second simulation is run as a stress with minimum timing. It is recommended to run the validation flow using a FastSPICE simulator such as Spectre XPS. Waveforms for both the simulations are provided to you. You can compare them by using the standard waveform viewers. For details, see Appendix D, "Liberate MX Timing Validation Flow."

Most problems can be found by running a single slew. It is recommended to run a single slew first. This can be followed by minimum and maximum of other slews. Output load is not expected to have a significant effect for this test.

Setting Up Validation Flow

The Liberate MX validation flow is similar to running characterization with the following exceptions:

- The template file is written to include the timing values from the library file.
- The run command is validate_macro instead of char_macro.
- The tables are written to ensure that an arc failure will be observable.

The validation flow can also be setup using the automatic validation flow or the full custom manual validation flow.

- In automatic flow, the setup from conventional `define_memory` flow can be used as it is with the last command changed to `validate_macro` (that is, replace `char_macro` with `validate_macro`). Here, the flow automatically creates the validation tables.
- In full custom flow, you need to manually create the validation table and validation template.

Automatic Validation Flow

The automatic validation flow is supported in the `define_memory` flow when you do the following:

- Code the define_memory command as usual. For example:

```
define_memory \
-ref_lib instance_name.lib \
....
....
....
instance_name
```

- Add the validate argument to the validate_memory command.

By doing this, Liberate MX takes the library generated template for validation, creates validate tables, and runs the relax and stress runs.

Manual Validation Flow

This flow involves the steps discussed in the sections below:

- Writing the Validation Template
- Writing the Validation Table Files
- Determining the Completeness of the Tables

See also Appendix E, “Basic Flow for Validating User-Defined Criteria.”

Writing the Validation Template

To provide the necessary timing information for the validation run, the validation flow uses the following command for template generation:

```
write_template -mx_validate <file_name>
```

The resulting template contains the timing information for the arcs:

```
define_arc \
    -type setup \
    -related_pin_dir R -pin_dir R  \
    -related_pin {CLK} \
    -pin {CEN} \
    -validation_values {0.219269 0.219272 0.219271 0.219274 0.219267 0.219268
    0.219268 0.229949 0.229953 0.229952 0.229954 0.229947 0.229948 0.229949
    0.244105 0.244108 0.244108 0.24411 0.244103 0.244104 0.244105 0.267319 0.267322
    0.267322 0.267324 0.267317 0.267318 0.267319 0.310398 0.310402 0.310401
    0.310404 0.310397 0.310398 0.310398 0.334074 0.334078 0.334077 0.33408 0.334073
    0.334074 0.334074 0.353111 0.353115 0.353114 0.353117 0.35311 0.353111 0.353111
    } \
    SRAM512x8
```

Writing the Validation Table Files

The tables for the validation flow should conform to the following criteria:

- Every arc should be exercised by the vectors. If an input transitions on the same line as the active edge of clock, it will be a setup stress. If an input transitions on the line after the active edge of clock, it will be a hold stress.
- The vectors should be written such that any single arc failure should lead to a difference in output.
- Input pins should have only one transition between clock active edges. This is done to ensure that there is enough time to satisfy setup, hold, and transitions within the clock period.
- Every signal should also contain a minimum pulse vector, where the setup and hold are both stressed around the same clock edge. This ensures that the latches are getting enough time to transition.
- It is recommended to use the relaxed simulation waveforms to verify the vectors before debugging the stress vectors.

Following is an example of the contents of a table file:

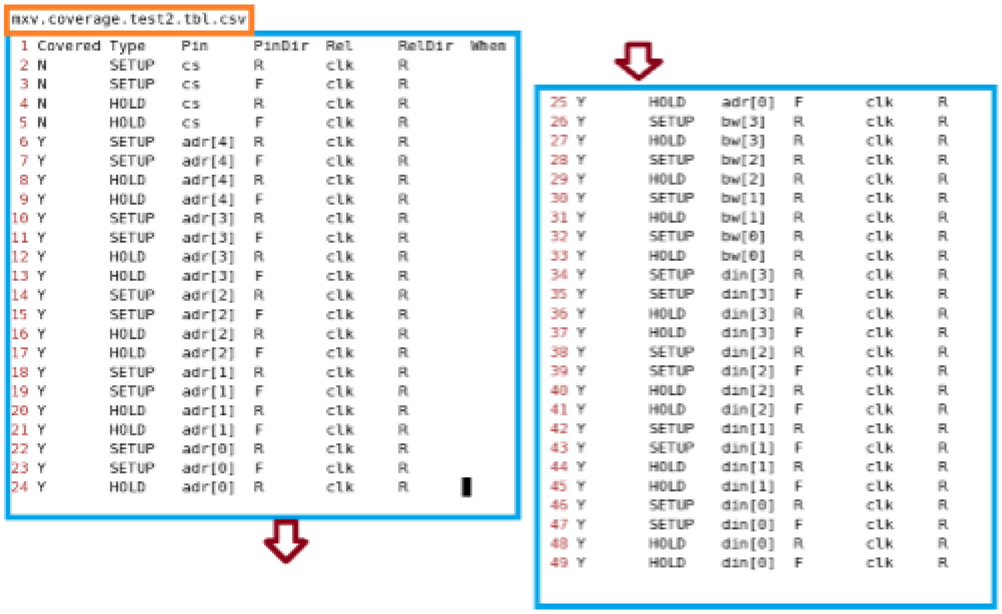
```
table write_read
pins    clk    addr      din  bw     cs        dout
W_a_A0  0       H        0x5  L      L         X
Clock1  1       L        0xa  H      H         D
disable 0       L        0x5  H      H         X
Clock2  1       L        0x5  H      L         D
W_5_A1  0       L        0x5  H      H         X
Clock3  1       H        0x5  H      H         D
disable 0       H        0xa  H      L         X
Clock4  1       H        0xa  H      L         D
R_a_A0  0       H        0xa  H      L         X
Clock5  1       L        0x5  L      H         D
endtable
```

Determining the Completeness of the Tables

To determine if all arcs have been tested, Liberate MX writes out a coverage report. This report is a summary of all arcs that existed in the template and whether they were tested by the tables.

The generated report is saved with a name of the following format:
`mxv.coverage.<`*`table_name`*`>.csv`

This file that has content presented in the following format can be read in Microsoft Excel:



mxv.coverage.test2.tbl.csv

Covered	Type	Pin	PinDir	Rel	RelDir	When
N	SETUP	cs	R	clk	R	
N	SETUP	cs	F	clk	R	
N	HOLD	cs	R	clk	R	
N	HOLD	cs	F	clk	R	
Y	SETUP	adr[4]	R	clk	R	
Y	SETUP	adr[4]	F	clk	R	
Y	HOLD	adr[4]	R	clk	R	
Y	HOLD	adr[4]	F	clk	R	
Y	SETUP	adr[3]	R	clk	R	
Y	SETUP	adr[3]	F	clk	R	
Y	HOLD	adr[3]	R	clk	R	
Y	HOLD	adr[3]	F	clk	R	
Y	SETUP	adr[2]	R	clk	R	
Y	SETUP	adr[2]	F	clk	R	
Y	HOLD	adr[2]	R	clk	R	
Y	HOLD	adr[2]	F	clk	R	
Y	SETUP	adr[1]	R	clk	R	
Y	SETUP	adr[1]	F	clk	R	
Y	HOLD	adr[1]	R	clk	R	
Y	HOLD	adr[1]	F	clk	R	
Y	SETUP	adr[0]	R	clk	R	
Y	SETUP	adr[0]	F	clk	R	
Y	HOLD	adr[0]	R	clk	R	
Y	HOLD	adr[0]	F	clk	R	
Y	SETUP	bw[3]	R	clk	R	
Y	HOLD	bw[3]	R	clk	R	
Y	SETUP	bw[2]	R	clk	R	
Y	HOLD	bw[2]	R	clk	R	
Y	SETUP	bw[1]	R	clk	R	
Y	HOLD	bw[1]	R	clk	R	
Y	SETUP	bw[0]	R	clk	R	
Y	HOLD	bw[0]	R	clk	R	
Y	SETUP	din[3]	R	clk	R	
Y	SETUP	din[3]	F	clk	R	
Y	HOLD	din[3]	R	clk	R	
Y	HOLD	din[3]	F	clk	R	
Y	SETUP	din[2]	R	clk	R	
Y	SETUP	din[2]	F	clk	R	
Y	HOLD	din[2]	R	clk	R	
Y	HOLD	din[2]	F	clk	R	
Y	SETUP	din[1]	R	clk	R	
Y	SETUP	din[1]	F	clk	R	
Y	HOLD	din[1]	R	clk	R	
Y	HOLD	din[1]	F	clk	R	
Y	SETUP	din[0]	R	clk	R	
Y	SETUP	din[0]	F	clk	R	
Y	HOLD	din[0]	R	clk	R	
Y	HOLD	din[0]	F	clk	R	

Determining Whether the Run is Passing

There are multiple criteria that can be used to consider a run to be passing. Some criteria are given below:

- All outputs switch in the expected interval with no push-out.
- All internal race conditions should not be worsened by running 'at speed' test.
- There should be no internal illegal states as a result of running 'at speed' test.
- All critical signals should switch full rail.

Checking Switching Activity

The `mx_reuse` or `mx_fastsim` directory has the following two waveform directories for each table file:

- The relaxed timing simulation:

  `<inst_name>``_`*`<table_name>`*`_relax_`*`<number>`*`_`
- The stress timing simulation:

  `<inst_name>``_`*`<table_name>`*`_stress_`*`<number>`*`_`

These waveform files can be viewed in a standard waveform viewer. The user should compare the results of the stress run against the results of the corresponding relaxed run. The output delays in the stress run should be very close to the delays in the relaxed run. Any differences should be investigated by the user.

Generating Node Comparison Report

A report can be generated to compare the activity of specified nodes between the relax run and the stress run. Use the `validate_node` and `report_node` commands to generate this comparison.

The first step is to run the `validate_node` command in the Tcl file, as shown below:

```
validate_node "bitline" instance_name
```

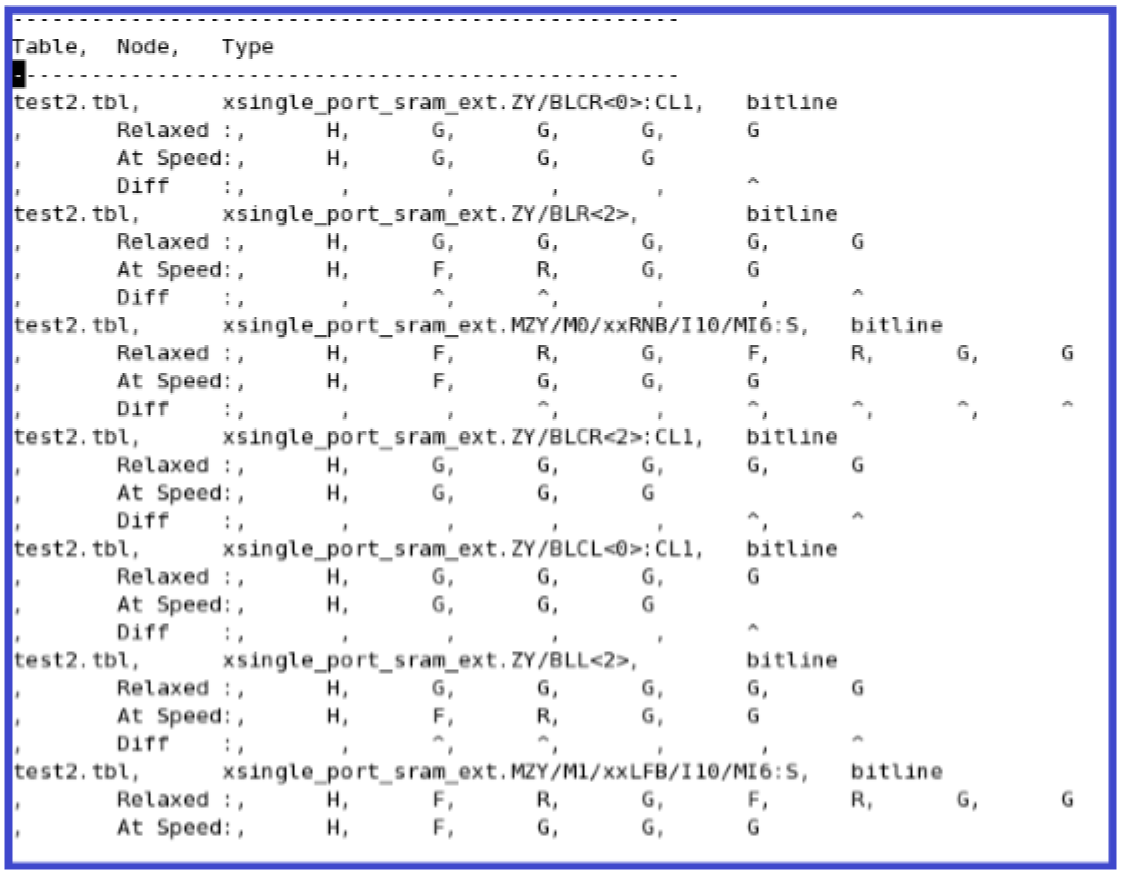
In this case, activities of all bitlines in the memory instance are compared. Other supported keywords are `wordline`, `core`, `bitline_precharger`, `senseamp_precharger`, `senseamp_enable`, and `output`. You can also specify a regular expression.

To report the node comparison, use the `report_node` command.

```
report_node -all "bitline" rf_top
```

By default, this command prints the values of only the failing comparisons. Using the `-all` option enables the printing of the passing and failing nodes.

These commands create a report named mxv.node.csv that can be opened in Microsoft Excel to view the node activity comparison, as shown below.



```
---------------------------------------------------
Table,  Node,   Type
---------------------------------------------------
test2.tbl,      xsingle_port_sram_ext.ZY/BLCR<0>:CL1,   bitline
,       Relaxed :,      H,      G,      G,      G,      G
,       At Speed:,      H,      G,      G,      G
,       Diff    :,       ,       ,       ,       ,      ^
test2.tbl,      xsingle_port_sram_ext.ZY/BLR<2>,        bitline
,       Relaxed :,      H,      G,      G,      G,      G,      G
,       At Speed:,      H,      F,      R,      G,      G
,       Diff    :,       ,      ^,      ^,       ,       ,      ^
test2.tbl,      xsingle_port_sram_ext.MZY/M0/xxRNB/I10/MI6:S,   bitline
,       Relaxed :,      H,      F,      R,      G,      F,      R,      G,      G
,       At Speed:,      H,      F,      G,      G,      G
,       Diff    :,       ,       ,      ^,       ,      ^,      ^,      ^,      ^
test2.tbl,      xsingle_port_sram_ext.ZY/BLCR<2>:CL1,   bitline
,       Relaxed :,      H,      G,      G,      G,      G,      G
,       At Speed:,      H,      G,      G,      G
,       Diff    :,       ,       ,       ,       ,      ^,      ^
test2.tbl,      xsingle_port_sram_ext.ZY/BLCL<0>:CL1,   bitline
,       Relaxed :,      H,      G,      G,      G,      G
,       At Speed:,      H,      G,      G,      G
,       Diff    :,       ,       ,       ,       ,      ^
test2.tbl,      xsingle_port_sram_ext.ZY/BLL<2>,        bitline
,       Relaxed :,      H,      G,      G,      G,      G,      G
,       At Speed:,      H,      F,      R,      G,      G
,       Diff    :,       ,      ^,      ^,       ,       ,      ^
test2.tbl,      xsingle_port_sram_ext.MZY/M1/xxLFB/I10/MI6:S,   bitline
,       Relaxed :,      H,      F,      R,      G,      F,      R,      G,      G
,       At Speed:,      H,      F,      G,      G,      G
```

Checking Internal Margins

Running the 'at speed' test can sometimes cause some internal signals to push out, affecting their relationship with other internal signals (setup time violation). These margins can also be compromised by the circuit not being fully reset at the end of the previous cycle (minimum period violation). Some examples of these margins that should be checked are:

- Write Margin

  In a typical design, the bitcell can write when the wordline is high and the bitline is low. The write window is defined as the time between the later of wordline rise and bitline fall and the earlier of wordline fall and bitline rise. This value should not be compromised by running the 'at speed' test.

- Read Margin

The read margin defines how much bitline differential has developed before the sense amplifier is enabled to amplify the difference in the read cycle. This is quantified by measuring the voltage difference between the two complimentary bitlines when the sense amplifier enabled signal becomes active. This value should not be compromised by running the simulation 'at speed' test.

Depending on the design, there might be other important margins that will need to be measured. It is important to measure these margins because it is possible that they might be reduced even if the correct output behavior is maintained.

Checking For Illegal States

There are several internal states within the memory instance that are illegal. These states usually are the result of the address decoding circuitry. An example of this can be the internal address lines that should have only one active line decoded. It is possible that under the 'at speed' test, multiple active lines overlap. These sort of conflicts might be temporary in nature, but they impact the robustness of the memory instance. To check this, all wordlines should be plotted and the user should ensure that only the intended wordlines go high during the clock cycle. Likewise, the column select lines at the column multiplexor should also be checked to ensure that only the intended lines are active on the cycle.

Checking For Full Rail Switching

All critical signals should switch full rail inside the memory instance to ensure robustness. There are two different causes for signals to not switch full rail in an 'at speed' test.

- The start of the internal signal is delayed at the beginning of the active cycle and the termination occurs before the signal can switch full rail. This situation is usually caused by setup time violations.
- The termination of the internal signal at the end of the active cycle does not go full rail before the next cycle starts and reactivates the line. This situation is usually caused by minimum period or minimum pulse width violations.

The signals that should be checked for full rail transition include the wordlines, bitlines during write, sense amp enable, and internal clocks.

Debugging the Failing Arcs

Once it is determined that the 'at speed' test failed, the next step is to determine which arc caused the failure. A circuit failure is usually a result of one or more arcs having incorrect values. There are two methods that can be used to determine such faulty arcs:

- Selectively increase the arc values to get a passing result.
- Debug the circuit to determine if a measurement location was not included in the original library characterization run.

Increasing Arc Values

To get a failing 'at speed' test to pass, one can employ the technique of selectively increasing the arc values. This is best done by modifying the values in the template file. It is generally best to increase one value at a time to isolate which arc is causing the failure. It can also be a combination of two arcs where either would pass by itself, but together they lead to a failure. It is recommended to increase minimum period first, followed by other arcs. Increasing other values before increasing minimum period can lead to illegal time sequences. After the minimum period has been increased, compare the failure traits with the expected effects of the arc failures to make an educated guess on which arcs might be causing the failure.

Using the Waveforms to Isolate the Failing Arc

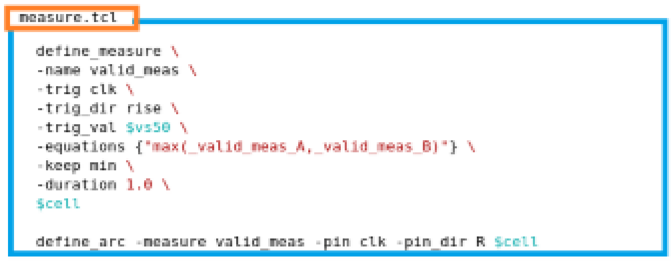
If the user has some familiarity of the design, it is possible to use the waveform results to determine what part of the circuit is failing and therefore which arc does not have a correct value. The user can compare the circuit behavior between the relax waveforms and the stress waveforms. It is useful to first compare the failure against the expected failures based on the vectors to narrow down the expected location of the problem.

Using the User-defined Criteria

The validation flow supports user-defined measurement criteria. To do this, you can use the following commands:

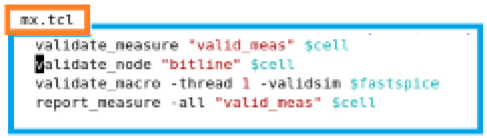
Inputs: define_measure -> validate_measure -> report_measure

Results: `mxv.meas.csv`, `mxv.meas.rpt`



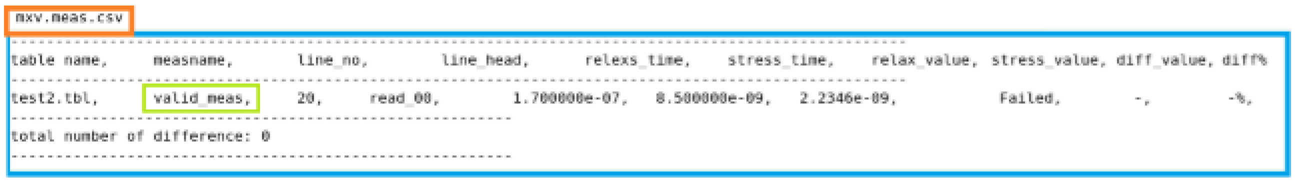
measure.tcl

```
define_measure \
-name valid_meas \
-trig clk \
-trig_dir rise \
-trig_val $vs50 \
-equations {"max(_valid_meas_A,_valid_meas_B)"} \
-keep min \
-duration 1.0 \
$cell

define_arc -measure valid_meas -pin clk -pin_dir R $cell
```



mx.tcl

```
validate_measure "valid_meas" $cell
validate_node "bitline" $cell
validate_macro -thread 1 -validsim $fastspice
report_measure -all "valid_meas" $cell
```



mxv.meas.csv

```
--------------------------------------------------------------------------------------------------------------------------
table name,     measname,       line_no,        line_head,      relexs_time,    stress_time,    relax_value, stress_value, diff_value, diff%
--------------------------------------------------------------------------------------------------------------------------
test2.tbl,      valid_meas,     20,      read_00,        1.700000e-07,   8.500000e-09,   2.2346e-09,          Failed,        -,          -%,
---------------------------------------------------------------------
total number of difference: 0
---------------------------------------------------------------------
```

5

Liberate MX Commands

This chapter describes the Tcl commands that control memory library creation.

Note: The command arguments that are prefixed with a hyphen (`-`) are optional except where explicitly indicated. All commands have a `-help` option. When this option is included with a command name, a help message is printed out that lists all currently available arguments for that command. There may be arguments that get printed out when help is used but are not officially supported. The only supported arguments are those that are documented in this manual. When `-help` is used, all other command arguments are ignored.

a...	
add_margin	
c...	
char_macro	compare_path
char_memory	
d...	
define_arc	define_leakage
define_cell	define_measure
define_duplicate_pins	define_memory
define_index	define_template
define_leafcell	define_table
h...	
hspice_lis_2_waves	
m...	
mx_match_node	mx_report
mx_recover_clean	mx_set_clockprop
mx_recover_info	mx_set_constprop

mx_recover_merge	mx_set_domainprop
mx_recover_setup	mx_set_finesim_param mx_set_ultrasim_param
r...	
report_measure	
s...	
set_gnd	set_virtual
set_vdd	spv_tcl_2_waves
v...	
validate_macro	validate_memory
validate_measure	
w...	
write_library	write_verilog

add_margin

-abs <value>	Amount of margin to add. Default 0.0 (no margin)
-cells {list}	List of cells
-direction <rise | fall | both>	Specify direction of data to add margin. Default: "both"
-pin {list}	List of pins
-related {list}	List of related pins
-rel <value>	Relative amount of margin to add, e.g. use 0.05 for 5%. Default 0.0 (0%)
-type {list}	Valid types include: cap, constraint, delay, delay_ccs, hidden, hold, leakage, mpw, power, recovery, removal, retain, retain_ccs, retain_trans, setup, trans. Default: apply margin to **all** types.
-when "string"	State dependent arc.

This command adds margin (padding) to values in the library. Margin is always added to all cells in a library.

type specifies the type of data to be modified. If the type option is not specified, the requested margin will be applied to all data types. Supported types are:

- cap
- constraint
- delay
- delay_ccs *(only accepts positive "abs" margin)*
- hidden
- hold
- leakage
- mpw
- power
- recovery
- removal
- retain
- retain_ccs *(only accepts positive "abs" margin)*
- retain_trans
- setup
- trans

constraint applies the same margin to all constraint types: setup, hold, recovery, removal and mpw.

abs (absolute) specifies the amount of margin to add in standard units. Default: 0.0 (no margin).

rel (relative) specifies a relative ratio amount of margin to add. Note that this option always makes the library value larger, whether the original value was positive or negative. Default 0.0 (0%). Example: 0.05 = 5% margin.

Types **power** and **hidden**: Margin may be added to the power and hidden types on individual cells by specifying a cell name with the -cells option. In this case the arc must be completely specified, that is, the pin, related, and when must also be specified. Example:

```
add_margin \
        -type {power|hidden} \
        -cells {celllists} \
        -pin {pin lists} \
        -related {related_pin list} \
        -when "when_condition" \
        -rel rel \
        -abs abs
```

This command can be used after **char_macro**, **read_ldb** and **read_library**, but it must be used before model generation such as with **write_library**. Multiple `add_margin` commands can be specified.

Examples:

```
read_ldb test.ldb.gz
write_library no_margin.lib

# Add 10% to power
add_margin -type power -rel 0.1

# Add 50ps to delay
add_margin -type delay -abs 50e-12
write_library margin.lib
```

char_macro

-ccs	Characterize CCS (delay) data
-ccsn	Characterize CCSN (noise) data
-char_params {*parameters*}	Parameters for Characterization. Default: none.
-char_script <*script_name*>	Extra Script for Characterization. This option is for backward compatibility for setups created prior to Libearte 13.1 release. Default: none.
-charsim "*simulator_name*"	Spice simulator to use for characterization. Default: Spectre APS.
-charthread <*number*>	How to multithread characterization. Default: 0.
-ecsm	Characterize ECSM (delay) data.

-ecsmn	Characterize ECSM (noise) data.
-part_arc_thread <*number*>	How to multithread arcs update in partitioning; overwrites -partthread value). Default: 1.
-partsim "*simulator_name*"	SPICE simulator to use for partitioning. Default: Spectre XPS.
-part_fastsim_thread <*number*>	How to multithread partsim during partitioning - (overwrites `-partthread` value only related to fastsim functionality). Default: 0.
-partthread <*number*>	How to multithread partitioning. Default: 0.
-part_wave_thread <*number*>	How to multithread waveforms update in partitioning. Default: 1.
-part_write_thread <*number*>	How to multithread components writing in partitioning; overwrites `-partthread` value. Default: 0.
-user_arcs_only	Only characterize user specified arcs.

Note: Following is a list of deprecated arguments for the **char_macro** command. These arguments are used only for backward compatibility.

-arc_thread <*number*>	How to multithread arcs update in preprocessing; overwrites -thread value. Default: 0 (Deprecated. Use `-part_arc_thread`)
-extsim "*simulator_name*"	External SPICE simulator program. (deprecated. Use `-charsim`).
-fastsim_thread <*number*>	How to multithread FastSPICE in preprocessing; overwrites -thread value. Default: 0 (Deprecated. Use `-part_fastsim_thread`)
-thread <*number*>	Maximum number of threads to use on the current machine. Default: 0 (deprecated. Use `-partthread`)
-wave_thread <*number*>	How to multithread waveforms update in preprocessing; overwrites -thread value. Default: 0 (Deprecated. Use `-part_wave_thread`)

-write_thread <*number*> How to multithread components writing in preprocessing; overwrites -thread value. Default: 0 (Deprecated. Use –`part_write_thread`)

The **char_macro** command performs memory characterization. Each memory listed in a **define_cell** command will be characterized providing the SPICE **subckt** definition for that memory is defined in the netlists passed to the **read_spice** command.

The **-partsim** argument specifies the external simulators of choice. Valid values are:

- "spectre"
- "finesim"
- "xa"

The **thread** argument defines the maximum number of threads to use on the current machine. Even if the **thread** argument is not specified Liberate MX will automatically use multiple threads based on the available CPUs. Steps that are multithread are FastSPICE simulation, results acquisition, arcs selection and file IO operations.

Note: The number of parallel FastSPICE runs is determined by the max among number of different mxtables files provided and number specified for **thread** argument – see Specifying Input Stimuli.The **char_params** argument specifies a list of options that can be provided to Liberate characterization. Valid arguments are **–ccs, -ccsn, -ecsm, -ecsmn, -thread, -extsim**. Please refer to the **char_macro** command in the latest Liberate Reference Manual for an explanation of these options.

The **char_script** argument specifies the name of a Tcl script to be sourced prior to characterization. Specify the complete path to the Tcl script. It is possible to specify commands that apply only to a specific characterization step – rather than all – by using variables **g_timing_char**, **g_inputcap_char**, **g_noise_char**, and **g_power_char**. Such variables are automatically set during the corresponding characterization run and can therefore be used to selectively apply settings to a specific step.

Examples:

- Example 1:

```
# Spectre XPS for dynamic partitioning with 2 threads and spectre APS for
#characterization (partition) runs with 5 threads
    char_macro -partsim spectre -partthread 2 -charsim spectre -charthread 5
```

- Example 2

```
# Characterize memory(s) defined via a previous define_cell
# command. Use Spectre-XPS for dynamic partitioning and
# use Spectre for delay and constraint characterization on
# dynamic partitions.
# Partition using 2 threads and characterize using 4
# threads. Generate CCS timing and CCS noise models
# Use distributed runs but only for timing characterization
    char_macro -extsim "spectre" -thread 2 \
        -char_params {-extsim "spectre" -thread 4 -ccs -ccsn} \
```

- Example 3

  Refer the scenarios below. The `-partthread` argument (earlier referred to as `-thread`) globally sets several sub-options. However, you can overwrite each individual option with corresponding sub-command. For example:

```
-partthread 10
automatically sets:
-part_fastsim_thread 10
-part_wave_thread 10
-part_arc_thread 10
-part_write_thread 10


-partthread 1
automatically sets:
-part_fastsim_thread 1
-part_wave_thread 1
-part_arc_thread 1
-part_write_thread 1


-partthread 1
-part_fastsim_thread 10
automatically sets:
-part_wave_thread 1
-part_arc_thread 1
-part_write_thread 1
```

char_memory

-ccs	Enables CCS characterization and library generation.
-ccsn	Enables CCSN characterization and library generation.
-ecsm	Enables ECSM characterization and library generation.

-ecsmn	Enables ECSMN characterization and library generation.
-work_dir "string"	The path where the `mx_setup` directory containing the reuseable setup files is created. The default for this is `mx_setup`.

The **char_memory** command is used along with the **define_memory** command to create all needed characterization files and run the characterization. It is required that **define_memory** is executed before the **char_memory** command. This command can be used to run the characterization of an instance of standard functionality (Standard Custom Instance Flow) or an instance designed by a third-party IP vendor (Vendor Recharacterization Flow).

compare_path

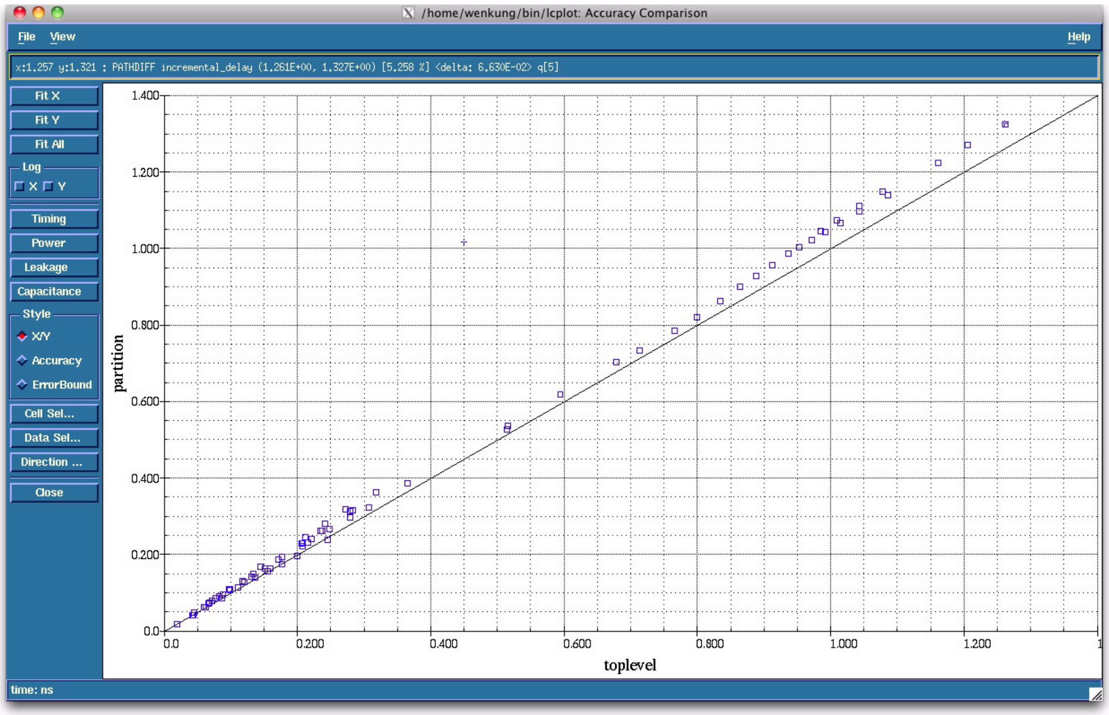
-abstol <double>	Specifies the error flag tolerate in nanoseconds (Default: 1e-2)
-debug < 0 | 1 >	Enables debug mode (Default: 0)
-gui <string>	Output the comparison result in graphical and lwave formats (Default: diff)
-lcplot < 0 | 1 >	Automatically output lcplot on result (Default: 0)
-lwave < 0 | 1 >	Automatically output lwave on result (Default: 0)
-part_report <string>	A report of the cumulative delay path as seen from the partition level (REALSPICE numbers). (Default: sim.rpt)
-report <string>	Output the comparison result in text format (Default: diff.rpt)
-top_report <string>	A report of the cumulative delay path as seen from the top level (FASTSPICE numbers). (Default: sim.top.rpt)

The **compare_path** command automatically compares the top-level and partition-level incremental delay path files to quickly identify where the top level and partition diverge.

The sim.top.rpt file is automatically generated at each run and for each delay partition. It contains the cumulative delay along the path as seen from the top level. The sim.rpt is automatically generated at each run and for each partition. It contains the cumulative delay along the path, as seen from the partition level. These reports help determine potential issues in the partitioning process itself. The command must be run in a delay, constraint, or measure partition directory.

The comparison report is output into both a graphical format (`diff.lcplot`) and a text format (diff.rpt).

The following is an example of a graphical version of the comparison report output by the **compare_path** command:



```
x:1.257 y:1.321 : PATHDIFF incremental_delay (1.261E+00, 1.327E+00) [5.258 %] <delta: 6.630E-02> q[5]
```

define_arc

-attribute {`altos_clone_arcs` **<*string*>}**

Instructs MX to duplicate the characterization results across equivalent arcs involving a bus.
Also can clone specific data types from a pin to a completely unrelated pin. These may be bus pins, or non-bus pins.

-attribute {`altos_mx_bisection 1`**}**

Instructs MX to use bisection for constraint (setup/hold) characterization.

attribute {`altos_mx_bundle <bundle_name>::<min|max>`**}**
Instructs MX on how to choose a representative of a specific measurement group before going to full SPICE characterization. All arc definitions of the same arc, with the same string in the `altos_mx_bundle`, are considered for the characterization of the arc. The `min` or `max` specifies whether the minimum or the maximum of all **define_arcs** is used in characterization. The `bundle_name` can be any string, but should be the same for all definitions of the same arc and not conflicting with the names chosen for different arcs. This is usually used along with `-measure` to consider multiple measurement definitions for a single arc.

-attribute {`altos_mx_existence` **<*string*>}**
Identifies the existence of specified switching

-attribute {`altos_mx_force_timing_type` **<*type*>}**
Forces a timing_type for a specific arc.

-attribute {`altos_mx_simulation_interval` **<*value*>}**
Specifies a simulation interval on an arc-basis.

-attribute {`altos_mx_autoprobing_skip_probe "`*<list_of_nodes>*`"`**}**
Specifies nodes to skip when considering probes.

-attribute {`altos_mx_autoprobing_skip_probe_regexp "`*<list_of_regular_expressions>*`"`**}**
Using regular expressions, specifies nodes to skip when considering probes.

-attribute {`altos_mx_autoprobing_skip_rel_probe "`*<list_of_nodes>*`"`**}**
Specifies related nodes to skip when considering probes.

-attribute {`altos_mx_autoprobing_skip_rel_probe_regexp "`*<list_of_regular_expressions>*`"`**}**
Using regular expressions, specifies related nodes to skip when considering probes.

-attribute {`altos_autoprobing_level <value>`**}**
Overwrites the values of both `mx_autoprobing_hold_level` and `mx_autoprobing_setup_level`.

-delay_threshold {`list`**}** Four delay measurement thresholds (in_rise, in_fall, out_rise, out_fall).

-equation <"`equation`">	Allows a SPICE equation to be used in place of a characterized value.
-extsim_deck_header	Allows to provide external simulator commands directly to the external simulator on an individual arc basis without using the Liberate process or reviewing them. This argument is intended to be used when an external simulator is used (refer to the `-extsim` argument of the **char_macro** command). It is a local arc specific version of the variable `extsim_deck_header`. As Liberate does not parse the string specified by this argument, ensure that the contents are valid and consistent with the arc simulation. The value string can contain the return character ("\n"). The value string is included at the top of simulation deck. For example: `define_arc -extsim_deck_header ".ic n128 0" -related_pin ck -pin Q ...`
-force	Allows only the probes specified by the `-probe` and `-related_probe` arguments. **Note**: The use of `-probe` and `-related_probe` arguments adds probes to the list of considered probes. The worst case of the Liberate MX determined probes and the **define_arc** declared probes are used in the partition simulations.
-measure "`name`"	Name of associated `define_measure` command.
-name "`name`"	Allows a name to be given to a value produced by characterization.
-pin {*pins*}	List of pin names (REQUIRED)
-pin_dir <`R` | `F`>	Transition direction of pin(s)
-pin_probe_threshold {*list*}	List of pin monitor nodes thresholds
-probe {*names*}	List of names of nodes to monitor for setup/hold characterization
-probe_dir <`R` | `F`>	Transition direction of probe pin(s)
-related_pin {*pins*}	List of related pin names
-related_pin_dir <`R` | `F`>	Transition direction of related pin(s)

-related_probe {*names*} List of names of clock nodes to monitor for setup/hold characterization

-related_probe_dir <`R` | `F`>
Transition direction of related probe pin(s)

-related_probe_threshold {*list*} List of related monitor nodes thresholds

-slew_threshold {*list*} Four slew measurement thresholds (lower_rise, upper_rise, lower_fall, upper_fall).

-type <`combinational|hold|max_clock_tree_path|min_clock_tree_path|minperiod|mpw|non_seq_hold|non_seq_setup|power|retain|setup`**>**
Type of arc (Default: `combinational`)

-when <*expression*> Conditional expression to be associated with the arc

{*cell_names*} List of cells

Liberate MX automatically extracts arcs from the provided table files. The **define_arc** command though allows you to:

- ❑ Override Liberate MX auto probing/partitioning
- ❑ Specify a **when** condition for an arc
- ❑ Specify retain arcs that should be characterized

type defines the type of arc. Possible values are *combinational, hold, max_clock_tree_path*, *min_clock_tree_path*, *minperiod*, *mpw*, *non_seq_hold, non_seq_setup*, *power, retain* and *setup*. The following are examples of how this argument would be used with *min_clock_tree_path* and *max_clock_tree_path*:

```
define_arc -type min_clock_tree_path -pin CKLA -pin_dir R $cell
define_arc -type min_clock_tree_path -pin CKLA -pin_dir F $cell
define_arc -type max_clock_tree_path -pin CKLA -pin_dir R $cell
define_arc -type max_clock_tree_path -pin CKLA -pin_dir F $cell
```

when defines the logic conditions of the other pins of the cell to enable this arc using the Liberty™ when syntax. It corresponds to the Liberty when attribute.

pin specifies a list of destination pins or buses for the arc.

related_pin specifies a list of related pins or buses for the arc.

probe defines the data node(s) to monitor when determining the constraint.

related_probe defines the clock node(s) to monitor when determining the constraint.

pin_dir, **related_pin_dir**, **probe_dir**, **related_probe_dir**, specify the direction of the transition (R = rise, F = fall). If directions are not specified, arcs are created only for R.

equation provides for specifying an equation to calculate a value. Any valid equation may be used. The calculated value is used instead of running simulation to generate a value. (See also -name option.)

Example:

```
define_arc -type mpw -pin CLK -pin_dir rise -name "mpwh" $cell
define_arc -type mpw -pin CLK -pin_dir fall -name "mpwl" $cell
define_arc -type min_period -pin CLK -equation "mpwl+mpwh" $cell
```

name allows a name (identifier) to be given to a value produced by characterization. This name may be used as part of an equation to calculate a value.

attribute passes an attribute name and value to the `define_arc` command. There are a number of applications of this:

1. **altos_mx_simulation_interval <value>** is used to specify a simulation interval on an arc-basis. *(Note: there are two other methods for specifying a simulation interval: globally with the variable mx_simulation_interval, and within a table, using the* simulation_interval *command in section* Specifying Input Stimuli*)* See mx_simulation_interval for a listing of precedence when there are multiple definitions of the simulation interval. Example:

   ```
   define_arc -attribute {altos_mx_simulation_interval 20e-9}
   ```

2. **altos_mx_existence <string>** uses the string as a cycle identifier in the table to check if it actually does switch as dictated by the arc. Example:

   ```
   altos_mx_existence <identifier>
   ```

Where <identifier> is <table_id>: :<mode_id>: :<line_id> and:

- **<table_id>** is an existing table file name (used in the run)
- **<mode_id>** is an existing table name (used in the run inside table_id)
- **<cycle_id>** is the table cycle name for the specific cycle that needs to be verified

The existence of specified switching of all involved measure points will be checked, starting at the simulation time identified by the attribute and for a number of sub-intervals, as indicated by the corresponding measure **–duration** option. *(See code example #3 below.)*

- **altos_mx_bundle <identifier>** instructs the tool to choose a representative of a specific measurement group before going to full SPICE characterization, where:

  ```
  <identifier> == <att_name>::<att_criterium>
  <att_name> == any string
  <att_criterium> == <min/max/absmin/absmax/average>
  ```

All measurement arcs with matching <**`att_name`**> will be performed an <**`att_criterium`**> operation on and only resulting one will generate a partition and be characterized with full SPICE. <**`att_criterium`**> is applied to the value of the first (if more than one) equation evaluated by the arc. For example, the following 3 measurement arcs:

```
define_arc -measure ABC_0
define_arc -measure ABC_1
define_arc -measure ABC_2
```

…will generate a partition and each be characterized with full SPICE.

However, for the following ones:

```
define_arc -measure ABC_0 -attribute {altos_mx_bundle ABC::min}
define_arc -measure ABC_1 -attribute {altos_mx_bundle ABC::min}
define_arc -measure ABC_2 -attribute {altos_mx_bundle ABC::min}
```

…only the minimum one will generate a partition and be characterized with full SPICE.

The two different `min_period` components are getting measured using **define_measure** and using **altos_mx_bundle** to take maximum of both. In the below example, both components get measured in multiple cycles during fastsim. At the end of fastsim `ck_min_period` gets one worst (max) value for which partition is created and then the partition is run with true SPICE.

Example:

```
set vc50 [expr $voltage * 0.5]
set vs90 [expr $voltage * 0.9]
set vc10 [expr $voltage * 0.1]

define_measure -name tcyc1_1 -trig CKR -trig_val $vc50 -targ $probe1_1 -
targ_type delay -targ_val $vc90 $cell
define_measure -name tcyc1_2 -trig CKR -trig_val $vc50 -targ $probe1_2 -
targ_type delay -targ_val $vc10 $cell
define_measure -name tcyc1 -trig CKR -trig_val $vc50 -targ_type delay -
equations {"tcyc1_1-tcyc1_2"} $cell
define_arc -type min_period -measure tcyc1 -pin CKR -pin_dir $dir_r -attribute
[list altos_mx_bundle ck_minperiod::max] $cell
```

```
define_measure -name tcyc3 -trig CKR -trig_val $vc50 -targ $probe3 -targ_type
delay -targ_val $vc90 $cell
```

```
define_arc -type min_period -measure tcyc3 -pin CKR -pin_dir $dir_r -attribute
[list altos_mx_bundle ck_minperiod::max] $cell
```

- **altos_clone_arcs <string>** is used to clone (duplicate) characterization results to other arcs. These may be bit of a bus; whole busses; or even completely unrelated pins. **<string>** is a list of {*related_pin / pin*} pairs that will receive the characterization results.

Note – how it works internally: The attribute altos_clone_arcs is usually derived internally by MX to instruct the tool on how to duplicate the characterization results across equivalent arcs involving a bus. For example, for an access-time characterization - `CLK` to bus `Q[31:0]` - MX will, by default, only characterize the worst case among all possible `CLK` to pin `Q[i]` - where `Q[i]` is a pin of bus `Q`. The result will then be "cloned" or copied to the remaining bits of the bus.

You can overwrite/augment the automatic setting of this attribute in the following ways:

Determine the ranges for cloning. It may be necessary to model arcs involving a bus not as a whole, but separated in to different groups. For example, for constraint arcs on an address bus A[7:0], it may be necessary to group column and row addresses in to separate groups, such as A[0:2], A[3:6] and A[7]. The specific grouping can then be specified in the `define_arc` itself and it will be respected when the `clone_attribute` is generated. For this specific example, you would issue three separate `define_arc` commands as shown below:

```
define_arc -type setup A[2:0] ...
define_arc -type setup A[6:3] ...
define_arc -type setup A[7] ...
```

and the tool will infer that 3 rather than 1 representative partitions and characterization runs will be needed to model these arcs in the final library.

Note: Having different values for different bits of the same bus will require the library to be written in a bit-blasted form (you would need to use the **-expand_buses** with the **write_library** command).

Use one arc characterization value for another arc

This will clone the hold arc from pin ena1 to ena2:

```
define_arc
    -type hold \
    -pin {ena1} \
    -related_pin {clk} \
    -attribute {altos_clone_arcs "clk ena2"} \
    myCell
```

This will clone the setup arcs from bus addrA to bus addrB:

```
set clone_arcs_str ""
for {set i 0} {$i < 8} {incr i} {
    set clone_arcs_str [concat $clone_arcs_str "clk addrB<$i>"]
}

define_arc
    -type setup \
    -pin {addrA<7:0>} \
    -pin_dir R \
    -when "!en" \
    -related_pin {clk} \
    -related_pin_dir R \
    -attribute [list altos_clone_arcs $clone_arcs_str] \
    myCell
```

1. **autoprobing_skip_probe <*list_of_nodes*>**
   autoprobing_skip_rel_probe <*list_of_nodes*>
   These are used to specify nodes and related nodes (respectively) to skip when considering probes during automatic probing for setup / hold characterization.

2. **autoprobing_skip_probe_regexp <*list_of_regular_expressions*>**
   autoprobing_skip_rel_probe_regexp <*list_of_regular_expressions*>
   These are used to specify nodes and related nodes using regular expressions that will be skipped when considering probes during automatic probing for setup / hold characterization.

Example 1: Do not use CLK as related probe for CLK->EZ setup:

```
define_arc \
    -type setup \
    -pin EZ \
    -related_pin CLK \
    -attribute {autoprobing_skip_rel_probe CLK} \
    $cell
```

Example 2: Do not use nodes that begin with "BL" or "WORD" for probes:

```
define_arc \
    -type setup \
    -pin EZ \
    -related_pin CLK \
    -attribute {autoprobing_skip_rel_probe_regexp BL* WORD*} \
    $cell
```

1. **altos_mx_bisection 1**
   Specifies that bisection should be used for constraint (setup/hold) characterization (instead of path-delay method.)

Example:

```
# Use bisection method to characterize setup constraint
define_arc \
    -type setup \
    -pin {addr[0]} \
    -related_pin {clk} \
    -attribute {altos_mx_bisection 1} \
    $cell
```

1. **altos_mx_force_timing_type <*type*>**
   Forces a timing_type for a specific arc.

Example: For a delay arc that models a dynamic output, force the timing_type to be "rising_edge" instead of the automatically found "combinational".

```
define_arc \
    -pin {dataout} \
    -related_pin {clk} \
    -attribute {altos_mx_force_timing_type rising_edge} \
    $cell
```

define_arc Examples:

Example 1:

```
# force when conditions on bypass arcs
define_arc -type combinational -pin dout[63:0] \
          -related_pin clk -when "BYP" $cell
define_arc -type combinational -pin dout[63:0] \
          -related_pin clk -when "!BYP" $cell

# force when conditions on memory's leakage power
define_arc -type leakage -when "ME" -cell $cell
define_arc -type leakage -when "!ME" -cell $cell

#define_arc to enable measurement flow to generate min period timing groups,
# define_arcs for minperiod
```

Example 2:

```
# add define_arc -type minperiod -pin {clk} <cell> to your run
# a new timing group will be generated in the library as:

  pin (CLK) {
      clock : true;
```

```
        direction : input;
        capacitance : 0.0100244;
        rise_capacitance : 0.0101743;
        rise_capacitance_range (0.00756358, 0.0116889);
        fall_capacitance : 0.00987448;
        fall_capacitance_range (0.00711549, 0.0117947);
        timing () {
          related_pin : "CLK";
          timing_type : minimum_period;
          rise_constraint (mpw_constraint_template_7x7) {
            index_1 ("0.004, 0.015, 0.038, 0.083, 0.174, 0.355, 0.717");
            values ( \
              "4.66069e-10, 4.69605e-10, 4.75735e-10, 4.83523e-10, 4.93672e-10,
5.06612e-10, 5.23597e-10" \
            );
          }
          fall_constraint (mpw_constraint_template_7x7) {
            index_1 ("0.004, 0.015, 0.038, 0.083, 0.174, 0.355, 0.717");
            values ( \
              "4.66069e-10, 4.69605e-10, 4.75735e-10, 4.83523e-10, 4.93672e-10,
5.06612e-10, 5.23597e-10" \
            );
          }
        }
```

Example 3:

The following syntax will trigger a check for *wgbt_r<7>* rising and *Xg/Xgcolr/Xgc<3>/phi1wx* falling at the simulation time corresponding to cycle *write00* in table functional of table file *timing.tbl*.

```
define_measure \
    -name      MCS_GWD_CLK_DIF_RR1_sig1 \
    -trig      {clk} \
    -trig_dir  rise \
    -trig_val  0.45 \
    -trig_d    "" \
    -targ      {"wgbt_r<7>" ""} \
    -targ_type "delay" \
    -targ_dir  rise \
    -targ_val  0.72 \
    -targ_d    "" \
    -failed_val "" \
    G40SP16384X4R3F1VTHSBSIR


define_measure \
    -name      MCS_GWD_CLK_DIF_RR1_sig2 \
    -trig      {clk} \
    -trig_dir  rise \
    -trig_val  0.45 \
    -trig_d    "" \
    -targ      "Xg/Xgcolr/Xgc<3>/phi1wx" \
    -targ_type "delay" \
    -targ_dir  fall \
    -targ_val  0.72 \
    -targ_d    "" \
    -failed_val "" \
    G40SP16384X4R3F1VTHSBSIR

define_measure \
    -name      MCS_GWD_CLK_DIF_RR1 \
    -keep      max \
    -duration  0.25 \
    -trig_d    "" \
    -targ      {"wgbt_r<7>" ""} \
    -targ_type "delay" \
    -targ_dir  rise \
    -targ_val  0.72 \
    -targ_d    "" \
    -failed_val "" \
    G40SP16384X4R3F1VTHSBSIR

define_measure \
    -name      MCS_GWD_CLK_DIF_RR1_sig2 \
    -trig      {clk} \
    -trig_dir  rise \
    -trig_val  0.45 \
    -trig_d    "" \
    -targ      "Xg/Xgcolr/Xgc<3>/phi1wx" \
    -targ_type "delay" \
    -targ_dir  fall \
    -targ_val  0.72 \
    -targ_d    "" \
    -failed_val "" \
    G40SP16384X4R3F1VTHSBSIR

define_measure \
    -name      MCS_GWD_CLK_DIF_RR1 \
    -keep      max \
    -duration  0.25 \
    -trig      {clk} \
    -trig_dir  rise \
    -trig_val  0.45 \
    -equations { \
                "MCS_GWD_CLK_DIF_RR1_sig1 - MCS_GWD_CLK_DIF_RR1_sig2" \
                "MCS_GWD_CLK_DIF_RR1_sig1 * 0.9 - MCS_GWD_CLK_DIF_RR1_sig2 *
1.1" \
               "100 * (MCS_GWD_CLK_DIF_RR1_sig1 - MCS_GWD_CLK_DIF_RR1_sig2) /
(MCS_GWD_CLK_DIF_RR1_sig1 + MCS_GWD_CLK_DIF_RR1_sig2) " \
                "100 * (MCS_GWD_CLK_DIF_RR1_sig1 * 0.9 -
MCS_GWD_CLK_DIF_RR1_sig2 * 1.1) / (MCS_GWD_CLK_DIF_RR1_sig1 +
MCS_GWD_CLK_DIF_RR1_sig2) " \
               } \
    G40SP16384X4R3F1VTHSBSIR

define_arc -pin {clk} -measure MCS_GWD_CLK_DIF_RR1 -attribute
{altos_mx_existence timing.tbl::functional::write00} G40SP16384X4R3F1VTHSBSIR
```

Example 4:

This will characterize output power for combinational delays:

```
define_arc \
    -when {DFTRAMP & X10} \
    -type power \
    -related_pin_dir R  \
    -pin_dir R  \
    -related_pin {A[4:0]} \
    -pin {AY[4:0]} \
    G40SP16384X4R3F1VTHSBSIR
```

define_cell

-async {pin_names}	List of asynchronous pin names. The async argument specifies pins that require recovery, removal, non_seq_setup, or non_seq_hold timing arcs.
-clock {pin_names}	List of clock pin names
-constraint <name>	Name of template for constraint tables
-delay <name>	Name of template for delay tables
-ignore_input_for_auto_cap {pin_names}	Ignore arcs originating from this pin when calculating auto_index and auto_max_capacitance.
-ignore_output_for_auto_cap {pin_names}	Ignore arcs to this output pin when calculating auto_index and auto_max_capacitance.
-ignore_pin_for_ccsn {pin_names}	List of input pin names not to have CCSN data.
-input {pin_names}	List of input pin names.
-mxcore {core_files}	List of one or more core cell description files.
-output {pin_names}	List of output pin names.
-power <name>	Name of template for power tables.
{cell_names}	List of cell names to be characterized.

The **define_cell** command defines how a memory is to be characterized.

The **input/output/clock** arguments define the pin type for the given list of pin names. All pins of a cell must have a defined pin type. The same pin name cannot appear in multiple pin types within a single **define_cell** command.

I/O bundles – buses – can be specified in a compressed form using any of the following delimiters: **||**, **<>**, **[]**, **{}**, **()**, _ enclosing a range specification in the form **N:M**, where N and M are integers.

In MX it is possible to use the following *bus* notation for a pin: *<name><left_delimiter>msb:lsb<?<right_delimiter>>* where *name* is the name of the bus, *left_delimiter* indicates what to use as left delimiter when expanding the bus, *msb:lsb* indicates the bit range the expansion must be done on and *right_delimiter* indicates what to use as right delimiter when expanding the bus. Supported delimiters are: *[]*, *<>*, *{}*, _, *()*. When the character / is specified, no delimiter is used. Examples are: *A[5:0]* will be expanded in to *A[5], A[4], … , A[0]. B/1:0/* will be expanded in to *B1*, *B0*; *C_0:3* will be expanded in to *C_0*, … , *C_3*.

The remaining arguments define which template to use for characterizing each library construct. If a template is specified then the appropriate construct is characterized for the given set of cells. If a template is omitted then this construct is not characterized.

The **delay** argument enables characterization of cell delay and output slew for the non-linear delay model (NLDM). The range of input slews and output loads to use for this construct is defined by the given template name where the template is pre-defined using the **define_template** command.

The **power** argument enables characterization of switching power. The range of input slews and output loads to use for this construct is defined by the given template name where the template is pre-defined using the **define_template** command.

The **constraint** argument enables characterization of timing constraints (setup, hold). The range of input slews to use for the data and clock signals is defined by the given template name where the template is pre-defined using the **define_template** command.

The **ignore_pin_for_ccsn** argument accepts a list of input/bidi pins. No CCSN data will be characterized for the specified pins.

The **mxcore** argument specifies the list of core cell description files used to match memory core cells in the design during automatic probing. For detailed information, see Appendix B, "Specifying Memory Core Cells."

define_duplicate_pins

<cellname >	The cell name to apply the duplication to.
<pin>	The pin to be duplicated.
{ duplicate_pins }	List of pin names to be duplicated.

The `define_duplicate_pins` command specifies a list of pins for a cell that will not be directly characterized, but instead will be given duplicate data from a characterized pin. This command will copy all the pin data from the <pin> to each of the duplicated pins including any data where the <pin> is a related_pin. If the pin is an input pin, the duplicate input pin will include pin cap, hidden power, and constraints. In addition, all input to output arcs where the pin is a related_pin will be duplicated for each `duplicate_pin`. Example:

```
    define_duplicate_pin mux A { B C D E F G H I }
```

This command also supports duplicating a complete bus. Example:

```
    define_duplicate_pins myCell addrA addrB
```

...where addrA and addrB are buses. The following restrictions apply:

1. addrA and addrB must have the same "direction" (cannot duplicate an input to an output.)
2. addrB cannot be of a lower range than addrA, but addrB can be of a larger range than addrA. In this case, only the 1st n bits of addrB will be duplicates of A and the rest will remain unique. *(In other words, if addrA is 16 bits, and addrB is 8 bits, you can clone addrA[7:0] to addB[7:0]. But if addrA is 8 bits, and addrB is 16 bits, you cannot clone anything into addrB[15:8] from addA, because addA doesn't contain that bit range.)*
3. addrB doesn't need to exist - it can be created on the fly during write_library.
4. Duplicating bundles is not supported.

define_index

-index_1 {indices}	List indices to use as index_1.
-index_2 {indices}	List indices to use as index_2.
-pin {pins}	List of pin names (REQUIRED). Accepts wildcards.
-related_pin {pins}	List of related pin names. Accepts wildcards. **Note**: `-related_pin` is required for most arcs, except for arcs that need only one pin, such as hidden power arcs and MPW or `min_period` arcs.
-type {data_types}	List of data types: *constraint, delay*, *mpw*, *power*, or *retain.*
-when <"function">	Logic state of side inputs.
{cell_names}	List of cell names. Accepts wildcards.

The **define_index** command overrides the indices specified in the templates referenced by **define_cell**. Valid table types are **constraint**, **delay**, **retain**, **mpw**, **power**, and **si_immunity**. The overrides apply only to the cells listed in the **define_index** command.

pin, **related_pin** and at least one of **index_1** or **index_2** must be specified.

Note: pin and related pin accept the asterisk "*" wildcard.

Example:

```
define_index -pin D*  # All pins beginning with "D"
define_index -pin  *  # All pins
```

The size of the **index_1** and **index_2** lists must be equal to the equivalent template type specified by **define_cell**. The **when** option helps define unique indexes by using the Liberty `when` syntax.

Multiple **define_index** commands can be used to specify different overrides for different arcs for the same set of cells, or for different cells.

The **define_index** command must follow the **define_cell** command and precede the **char_macro** command.

Examples:

```
define_template -type delay \
         -index_1 {0.16 0.5 1.6 } \
         -index_2 {0.04 0.08 0.2 } \
         delay_template

define_cell \
    -clock { clk_in } \
    -input { add_in<6:0> chip_en data_in<31:0> wr_in } \
    -output { data_out<31:0> } \
    -delay delay_template \
    rf_cell

### Here the load values getting overwritten by define_index below
define_index \
   -pin {data_out<31:0>} \
   -related_pin {clk_in} \
   -type delay \
    -index_2 {0.05 0.01 0.5 } \
   rf_cell
```

define_leafcell

-area <"string">	Name of area diode parameter (default 'area')
-element	Enables circuit element-based leafcells. Model name(s) must be specified.
-length "string"	Name of the length MOS parameter in the cell. (default 'l')
-multiple "string"	Name of multiple MOS parameter in the cell. (default 'm')
-nfin "parameter_name"	Name of the nfin parameter for FinFET devices. (default 'NFIN')
-pin_position {list of pin positions}	List of pin positions (REQUIRED)
-pj <"string">	Name of pj diode parameter (default 'pj')
-scale <"value">	MOS param scale factor (default '1.0')
-type "string"	Type of cell: [nmos | pmos | diode | r | c]
-width "string"	Name of width MOS parameter (default 'w')
{cell_names}	List of leafcell names

Use this command to define the level of hierarchy that resides at the bottom of a cell level netlist. This allows Liberate MX to correctly identify devices in the cell netlist even when the process model file cannot be parsed. This command can be used in combination with the **extsim_model_include** control variable to enable external simulation with the process models and the compiled netlist. This command supports identification of mosfets, diodes, resistors, and capacitors.

Use the **type** argument to specify the type of the cell. Supported settings are **nmos**. **pmos**, **diode**, **r**, and **c**.

Use the **pins** argument to provide the pin positions. There should be one number for each pin in the cell. The pins usually start from 0. The pin_positions usually start from 0 <drain gate source [bulk(s)]> | <terminal_p terminal_n[bulks]>.

Use the **length** argument to provide the name of the MOS length parameter in the cell. The default name is 'l'.

Use the **width** argument to provide the name of the MOS width parameter in the cell. The default name is 'w'.

Use the **area** argument to provide the name of the diode area parameter in the cell. The default name is 'area'.

Use the **pj** argument to provide the name of the diode pj parameter in the cell. The default name is 'pj'.

Use the **scale** argument to provide the scale in the cell. The default is 1.0.

This scale factor is used only by the Liberate "Inside View" to determine device sizes, and is not applied to the device sizes in the simulation netlist.

This command must be used before **read_spice**.

Examples:

```
# Define the cell NCH_MAC as a leafcell
    define_leafcell \
    -type nmos \
    -pins { 0 1 2 3 } NCH_MAC

# Define the cell PCH_map as a leafcell.
# first node (gate) in netlist must be swapped with the
# second node (drain) to match drain,gate,source,bulk order
    define_leafcell \
    -type pmos \
    -pins { 1 0 2 3 } PCH_map
```

define_leakage

-when <"function">	User-specified logic conditions
{cell_names}	List of cell names

The `define_leakage` command defines the logic conditions to use for calculating leakage power for the given cellname.

The `-when` argument specifies the logic conditions using the Liberty when format syntax.

The `-vector` argument allows for partial when conditions for leakage in the output library, while having secondary pins set to a mix of 0s and 1s.

Note: Enhanced leakage characterization is also achievable by the tool honoring the `-when` condition specified in the **define_cell** command.

This command must be used before **read_spice**, **char_macro**, and **write_library**.

Examples:

```
# Define the leakage conditions of a RF instance
define_leakage -when " !CEN" rf128
define_leakage -when " CEN" rf128
```

define_measure

-duration <# of cycles>	Measurement duration in number of cycles, starting at this trigger (Default: 1)
-equations <list>	Set of related equations consisting measure names in Spice syntax. Default: none
-failed_val <string>	Use this value for failed measurements. Default: none
-failed_val_reuse_factor <value>	Multiplies failed_val in partitioning step. Default: 1
-keep <none | min | max>	Keep the measure value in LDB. Default: none
-max_val_reuse_factor <value>	Multiplies max_val in partitioning step. Default: 1
-min_val_reuse_factor <value>	Multiplies min_val in partitioning step. Default: 1
-name <"name">	Measurement name. (Required). Default: none
-trig {signal_name}	Trigger signal name. (Required). Default: none
-trig_dir <rise | fall>	Trigger signal direction. (Required). Default: none
-trig_val <voltage>	Trigger voltage. (Required). Default: -100
-trig_val_reuse_factor <value>	Multiplies trig_val in partitioning step. Default: 1
-trig_d <named_measure | # of cycles >	Delay trigger after named measure or number of cycles. Default: none

-trig_e <first | last | # > Trigger edge specifier. Default: "`first`"

-trig_s <named_measure | # in sec.>
Shift trigger crossing time by result of named measure or number specified in seconds.

-targ {signal_name(s)} One or two target signal names. Default: none

-targ_type <delay | voltage> Target measurement type. Default: delay

-targ_dir <rise | fall> Target signal direction. Default: none

-targ_val <voltage> Target voltage. Default: -100

-targ_val_reuse_factor <value> Multiplies targ_val in partitioning step. Default: 1

-targ_d <named_measure | # of cycles >
Delay target after named measure or number of cycles. Default: none

-targ_e <first | last | # > Target edge specifier. Default: "`last`"

-targ_s <named_measure | # in sec.>
Shift target crossing time by result of named measure or number specified in seconds.

-trig_start {signal_name} Start trigger signal name (when specifying a window for a voltage measurement)

-trig_start_dir <rise | fall> Start trigger signal direction. Default: none

-trig_start_val <voltage> Start trigger voltage. Default: -100

-trig_start_val_reuse_factor <value>
Multiplies trig_start_val in partitioning step. Default: -1 (Not applied)

-trig_start_d <named_measure | # of cycles>
Delay start trigger after named measure or number of cycles

-trig_start_e <first | last | # > Start trigger edge specifier. Default: first

-trig_start_s <named_measure | # in sec.> Shift start trigger crossing time by result of named measure or number specified (in sec.)

-trig_end {signal_name} End trigger signal name (when specifying a window for a voltage measurement)

-trig_end_dir <rise|fall> End trigger signal direction.

-trig_end_val <voltage> End trigger voltage

-trig_end_val_reuse_factor <value>
Multiplies trig_end_val in partitioning step. Default: 1

-trig_end_d <named_measure | # of cycles>
Delay End trigger after named measure or number of cycles

-trig_end_e <first |last | # >End trigger edge specifier.

-trig_end_s <named_measure | # in sec.> Shift End trigger crossing time by result of named measure or number specified (in sec.)

-when <string> When to evaluate the measure. Default: none

-val_reuse_factor <value> Sets all "reuse_factors". Default: -1 (Not used)

{cell_names} List of cell names. Default: none

Use the define_measure command to specify a measurement in a form similar to the .meas command in HSpice. The same process of FastSPICE top-level evaluation and subsequent partitioning and full SPICE characterization – normally used for delay and constraint characterization – is used for the `define_measure` command as well. The results of the measurement is reported to files, user specified by the parameter `mx_margin_report`, for both the fast and real SPICE simulations.

Each measurement identified by a `define_measure` command will be evaluated in each simulation interval - as dictated by the define_table used for the measurement - and across as many intervals as specified by the duration option. For a measure that evaluates in more than a simulation interval, the maximum value will be used, unless otherwise specified by the keep option.

It is possible to specify one or more equations associated to a measurement via the equations option; note that if multiple equations are specified, they are evaluated independently one from the other.

In case a user defines `define_arc` followed by `define_measure`, then Liberate MX generates partition for that path and reports fastsim top-level evaluation as well as partition values. The results of the measurement is reported to files `measure.rpt.fastsim` and `measure.rpt`.

For example:

```
define_measure -name N -when "W"
define_arc -measure N -when "W"
```

If the user only defines `define_measure`, then Liberate MX evaluates it in fastsim and the measurement results is reported in the `measure.rpt.fastsim` file.

For example:

```
define_measure -name N -when "W"
```

keep instructs MX to keep the result of the measurement as a group in the LDB, and what to keep in case the measurement evaluates to different values in different simulation periods.

name uniquely identifies the measurement and is a mandatory option. Since the name can potentially be used as argument of other measurements equations, it should *not* contain characters that can be interpreted as mathematical operators:

```
    - + * / , ( ) { } > < % : ^ .
```

duration specifies how many cycles - starting at the triggering event - should a measurement be evaluated for. A cycle in MX is equivalent to 2 * mx_simulation_interval.

trig specifies the trigger signal name. (Required)

trig_dir specifies the trigger signal direction. (Required.) Valid values are rise and fall.

trig_val is the voltage at which the triggering event should be considered. (Required.) It is an absolute value in volts (V).

trig_d specifies the duration after which the trigger should be considered and can be either in terms of cycles or set to the name of a different measurement.

trig_e specifies what edge of the triggering signal should be considered. Valid values are first, last or any positive integer.

trig_s specifies the duration to shift the resulting trigger time point. Value can be an absolute number (in seconds) or the result of another measurement.

targ specifies the target signal name. (Required)

targ_type specifies the type of measurement, which can be either *delay* or *voltage*.

targ_dir specifies the target signal direction. (Required) Valid values are rise and fall.

targ_val is the voltage at which the target event should be considered. (Required) It is an absolute value in volts (V).

targ_d specifies the duration after which the target should be considered and can be either in terms of cycles or set to the name of a different measurement.

targ_e specifies what edge of the target signal should be considered. Valid values are first, last or any positive integer.

targ_s specifies the duration to shift the resulting target time point. Value can be an absolute number (in seconds) or the result of another measurement.

trig_start and **trig_end** allow for a specific window of observation to be specified. They are used with the following options:

trig_start_dir	trig_end_dir
trig_start_val	trig_end_val
trig_start_d	trig_end_d
trig_start_e	trig_end_e
trig_start_s	trig_end_s

Restrictions using `trig_start` and `trig_end`:

- `They` must be used together.
- `They` cannot be used with `trig` (if so, `trig` will be ignored.)
- The only supported measurement types (`targ_type`) are voltage *min* and *max*

failed_val sets a value for a measurement that does not evaluate.

equations specifies one or more equations to be evaluated for the specific measurement; equations are in terms of previous measurement names and must be specified in SPICE syntax. In the current release, only a 2-level nesting and a 2-level redirection are supported. Example:

```
define_measure -name A
define_measure -name B
define_measure -name C -equations "A+B" <- OK
define_measure -name D -equations A     <- OK
define_measure -name E -equations D     <- NOT SUPPORTED
```

"**reuse_factor**" options:

- failed_val_reuse_factor
- max_val_reuse_factor
- min_val_reuse_factor
- targ_val_reuse_factor
- trig_end_val_reuse_factor
- trig_start_val_reuse_factor
- trig_val_reuse_factor
- val_reuse_factor

These options are associated with the Reuse Flow, which requires that during the partition phase, waveforms are scaled from the old to the current values of rails. This is done automatically for automatic types of measurements. However, for user specified measurement - with user specified voltage levels - (i.e. define_measure commands) it's unknown whether a specified level is to be considered a voltage level (to be scaled) or an absolute value (to be left untouched). These options are used to explicitly specify scaling. The option **val_reuse_factor** is a "master" scaling factor that sets all the reuse scaling factors at once.

The typical usage of the `define_measure` command is the specification of one or more raw measurements followed by a second-level measurement which will use the raw ones in its specified equations.

Below is an example showing how HPICE .meas statements (in # comment) are converted into equivalent `define_measure` commands. In this particular example, measurement are such that the final (second level equation) value needs to be evaluated across 2 clock cycles, whereas the individual (raw) measurement need to be taken only in the first (sig1 and sigb) or in the second (sig2) cycle:

```
set cell SRAM
#.param cyc = 20n
#.param tc1 = 85n
#.param tc2 = tc1+cyc
#.param tc3 = tc2+cyc
#.param tc4 = tc3+cyc
#.meas tran sig1 trig v(clk) val=0.45 td=tc4 rise=1
#+targ v(x1.Xr/Xrblk<0>/reset_phi:1) val=0.18 td=tc4 fall = 1

define_measure \
    -name sig1 \
    -trig {clk} \
    -trig_dir rise \
    -trig_val 0.45 \
    -targ "reset_clk:1" \
    -targ_dir fall \
    -targ_val 0.18 \
    $cell

#.meas tran sig2 trig v(clk) val=0.45 td='tc4+cyc' rise=1
#+targ v(buf:1) val=0.18 td='tc4+cyc' rise=1
define_measure \
    -name sig2 \
    -trig {clk} \
    -trig_dir rise \
    -trig_val 0.45 \
    -trig_d 1 \
    -targ "buf:1" \
    -targ_dir rise \
    -targ_val 0.18 \
    -targ_d 1 \
    $cell

#.meas tran sigb trig v(st:1) val=0.45 td=tc4 fall=1
#+targ v(buf:1) val=0.45 td=tc4 fall=1
#.meas tran sigb_final param='max(sigb,0)'
# sigb actually switches in both cycles; force it to
# take the first falling transition for the target

define_measure \
    -name sigb \
    -trig {st:1} \
    -trig_dir fall \
    -trig_val 0.45 \
    -targ "buf:1" \
    -targ_dir fall \
    -targ_val 0.45 \
    -targ_edge first \
    $cell

#.meas tran reset param='sig1-sigb_final-sig2'
```

```
define_measure \
    -name reset \
    -keep min \
    -duration 2 \
    -trig {clk} \
    -trig_dir rise \
    -trig_val 0.45 \
    -targ_d 1 \
    -equations { \
        "sig1 - sigb - sig2" \
        "(sig1 - sigb)*0.9 - sig2 *1.1" \
        "100 * (sig1 - sigb - sig2)/ (sig1 - sigb + sig2) " \
        "100 * ((sig1 - sigb)*0.9 - sig2 *1.1) /(sig1 - sigb + sig2) " \
        } \
    $cell

define_arc -pin {clk} -measure reset $cell
```

define_memory

-additional_arcs list ()

List of one or more arcs to be included in the characterization.

-additional_tables {list_of_table_files}

List of one or more behavioral tables to be included in the characterization. These specified table files supplements the tables automatically generated by the **define_memory** flow.

-address <address_base_name>

Base Address bus name. This is the root name, not the full bus name and without any port or test prefix or suffix.
For example, if a dual port memory has addresses named `ADRA[9:0]` and `ADRB[9:0]`, then the root name would be `ADR` and `{A B}` would be `-port_suffix`. (Standard Custom Instance flow).

-autoprobing_hold_level list ()

Level value followed by base pin names.

-autoprobing_setup_level list ()

Level value followed by base pin names.

-banks int (1) Number of banks

-bisection <0 | 1> `Enable bisection for constraints. Set to 0 for path delay flow or 1 for bisection flow. Default: 0.`

-bist_prefix "string" Signal prefix for test mode. This specifies the prefix to be added to the base signal names to specify the signal names to be used for the test mode operation. This affects any defined address, data_in, data_out, chip_enable, write_enable, bit_write, or test_enable. (Standard Custom Instance flow)

-bist_suffix "string" Signal suffix for test mode. This specifies the suffix to be added to the base signal names to specify the signal names to be used for the test mode operation. This affects any defined address, data_in, data_out, chip_enable, write_enable, bit_write, or test_enable. (Standard Custom Instance flow)

-bits It is the width of one word in memory.

-bit_mask list() Base write enable bus name, followed by its state for bit is masked (L or H).

-bit_write {base_name masked_state}
Base Write Enable bus name, followed by its state for bit is masked (L or H). For example, if a dual port memory has bit write enables named `BWENA` and `BWENB`, then the root name would be `BWEN` and `{A B}` would be `-port_suffix`. If the bit is masked when the state is low, the correct argument would be `{BWEN L}`. (Standard Custom Instance flow)

-bitcell <rom|single_port|dual_port|2prf|10t>
Specify the type of bitcell. Available values are `rom`, `single_port`, `dual_port`, `2prf`, or `10t`. Default is to determine the probable bitcell based on the pinout of the instance. (Standard Custom Instance flow)

-bypass_enable list ()
Base Bypass Enable pin followed by its state for chip is in bypass mode (L or H).

-bypass_suffix list () Signal suffix for bypass mode.

-cfg_file <cfg_file_name>

The cfg file generated from the compiler along with the instance (Virage and TSMC recharacterization flows)

-char_script <string>

File containing the characterization commands

-char_spice <simulator_name>

Characterization SPICE simulator to be used for characterization. The allowed values are `aps`, `spectre`, `hspice`, or `finesim`. Default: `aps`.

-char_thread <number>

Number of threads to be used for characterization simulations. Default is to use maximum available threads or licenses. Overrides the value specified in `-thread`.

-chip_enable {base_name active_state}

Base Chip Enable pin name, followed by its state for chip is active (L or H). For example, if a dual port memory has chip enables named `CENA` and `CENB`, then the root name would be `CEN` and `{A B}` would be `-port_suffix`. If the chip enable is active low in this case, the correct argument would be `{CEN L}`. (Standard Custom Instance flow)

-clk_bist_prefix "string"

Clock prefix for test mode. This specifies the prefix to be added to the base clock name to specify the clock name to be used for the test mode operation. (Standard Custom Instance flow)

-clk_bist_suffix "string"

Clock suffix for test mode. This specifies the suffix to be added to the base clock name to specify the clock name to be used for the test mode operation. (Standard Custom Instance flow)

-clk_port_suffix {list}

List of clock port suffixes. This option is needed for instances with more than one port. These are the suffixes added to the base names to differentiate between the signals that are used for the various ports. This affects any defined clock. (Standard Custom Instance flow)

-clock <clock_base_name>

Base Clock pin name. This is the root name without any port or test prefix or suffix. For example, if a dual port memory has `CLKA` and `CLKB`, the `-clock` would be `CLK` and `{A B]` would be the `-clock_bist_suffix`. (Standard Custom Instance flow)

-clock_active_edge <string>

Active edge of clock for edge triggered designs [valid values rise/fall, defaults: rise]

-column_mux

Column MUX is used to optimize designs (Power/Performance/Area). A MUX number shows a muxing done for a column and controlled by address bits that is the column address. Higher the column address lower is the number of physical rows in designs.

-compiler_name string ()

Name of the generating compiler.

-const_arc_attr list ()

List of constraint arc attribute.

-data_in <data_in_base_name>

Base Data In bus name. This is the root name, not the full bus name and without any port or test prefix or suffix. For example, if a dual port memory has addresses named `DA[9:0]` and `DB[9:0]`, then the root name would be `D` and `{A B}` would be `-port_suffix`. (Standard Custom Instance flow)

-data_out <data_out_base_name>

Base Data Out bus name. This is the root name, not the full bus name and without any port or test prefix or suffix. For example, if a dual port memory has addresses named `QA[9:0]` and `QB[9:0]`, then the root name would be `Q` and `{A B}` would be `-port_suffix`. (Standard Custom Instance flow)

-debug int (0)

Print Verbose debug information. The valid values are 0 and 1. Default: 0.

-derive_table_from_tmpl int (1)

Derive MX tables from template.

-design <sram|rf|uhdrf|rom>

Name of design. Allowed values are `sram`, `rf`, `uhdrf`, or `rom`. (Standard Custom Instance flow)

-expand_buses <string (no)>

Controls whether the buses are written out expanded in the library file. The valid values are yes or no, Default is no.

-extsim_cmd_option <string>

Specify circuit simulation options. This option is used to specify simulator control options for the extsim simulator. Default is " ". The input option string is automatically added into the `mx.tcl` file.

-fastsim_cmd_option <string>

Specify fast simulator options. This option is used to specify simulator control options for the fastsim simulator. Default is " ". The input option string is automatically added into all mx table files.

-foundry <TSMC | SMIC>

Foundry for determination of device names. Allowed foundries are TSMC and SMIC. For all other foundries, it will be necessary to define the device leafcells in the `mx_setting` file.

-global_voltage <voltage_value>

Global Power supply voltage value in volts. Specifies the power supply value for vendor re-characterization instance. These instances do not use the `-rail` option since the power supply names are defined internally. For custom instances with `-rail` defined, it overrides the maximum rail voltage to define the instance operating condition.

-greybox <0|1> Enables Grey Box Mode (no partitioning). Set to `0` for standard flow or `1` for grey box mode. Default: `0`.

-internal_threshold {lower_threshold upper_threshold}

List of two values to be used for the lower and upper internal probing thresholds in percent. Default is `{20 80}`.

-latch_probing list ()

Override for latch internal probing option. Default is Input, State, Internal, Output.

-loads {list_of_loads}

List of loads to be used for characterization in pf. Overrides internal or template or library definition.

-model_format <spice | spectre>

Model Format. Allowed values are `spice` or `spectre`. Default is `spice`.

-models <model_include_file>

SPICE Model include file.

-model_leafcell

Used to confirm if `-element` is necessary for the specialized foundry. Default is `false.`
If `-foundry` is used for determining device names (for example, `mos` is defined as `.model` in model file and as `M` in netlist), then there is a need for defining the `-element` option of the **define_leafcell** command. For example:
`define_memory -foundry TSMC -model_leafcell 1`

-models_leakage <model_leakage_include_file>

Spice Model include file for leakage characterization. If not defined, then `-models` is used.

-models_power <model_power_include_file>

Spice Model include file for power characterization. If not defined, then `-models` is used.

-mx_setting <mx_tcl_file>

Name of user-provided Tcl file containing Liberate MX settings and commands. This is used to override the internal Liberate MX settings that are defined in the `define_memory` flow.

-netlist <netlist_file> The instance netlist to be used for the characterization.

-netlist_format <spice | spectre>

Netlist Format. Allowed values are `spice` or `spectre`. Default: `spice`.

-number_of_ports int ()

Number of ports in the bitcell, For dual port memory it is “2”.

-other_input_pins list ()

List of Input Pins of other functions.

-output_enable list ()
Base Output Enable pin name, followed by its state (L or H).

-part_spice <simulator_name>
Partition SPICE simulator to be used for characterization. Allowed values are `xps`, `aps`, `finesimpro`, or `xa`. Default: `xps`.

-part_thread <number>
Number of threads to be used for partitioning simulations. Default is to use maximum available threads or licenses. Overrides the value specified in *-thread*.

-part_waveform_format <string>
Format of partition simulation result. Allowed values are `sst2` or `fsdb`. Default is `sst2`. For example, to use `fsdb` as the waveform format for Liberate MX top-level simulation runs, specify:

```
part_waveform_format -string "fsdb"
```

-pin_file <pin_file> File containing the pin name information. For more information on usage of -pin_file, see Guidelines for Usage of -vendor and -pin_file in define_memory Flow.

-port_suffix {list_of_q_loads}
List of port suffixes. This option is needed for instances with more than one port. These are the suffixes added to the base names to differentiate between the signals that are used for the various ports. This affects any `defined address`, `data_in`, `data_out`, `chip_enable`, `write_enable`, `bit_write`, or `test_enable`. (Standard Custom Instance flow)

-process_node <process_node>
Process Node like 65nm, 55nm, 45nm, 40nm, 32nm, or 28nm. (Vendor Recharacterization flow if cfg file is not provided).

-qloads {list} List of loads to be used for characterization for the Data out pin in pf. Overrides internal, template or library, or `-load` definition.

-rail {paired_list_of_supplies_and_values}
Name and value pairs for each Voltage supply. Value is in volts. Format is `{<supply1> <supply1_voltage> <supply2> <supply2_voltage> ...}` (Standard Custom Instance flow)

-rcdb <0|1>	Enable the RCDB flow. Set to `0` for standard flow or `1` for rcdb flow. Default: `0`.
-read_enable list ()	Base Read Enable pin name, followed by its state for chip is reading (L or H)
-read_port list ()	List of read ports pins.
-read_timer list ()	Base Read Timer pin name.
-read_timer_enable list ()	Base Read Timer Enable pin name, followed by its state (L or H).
-redundancy_address list ()	Redundancy Address bus name.
-redundancy_enable list ()	Base Redundancy Enable pin followed by its state (L or H).
-ref_lib <string>	Name of reference library. Do not use if `-template` is specified.
-remove_tables {list_of_tables}	List of automatically generated tables to be removed from the run. Available values are: `leakage`, `delay`, `constraint`, `power`, `measure`, `bist`, `sleep`, `bist_pwr`.
-scan_enable list ()	Base Scan Enable pin name, followed by its state for chip is in scan mode (L or H)
-scan_in list ()	Base Scan In bus name
-scan_out list ()	Base Scan Out bus name
-setup_reuse <0|1>	Reuse setup files from the previous run if available. Allowed values are `0` (do not reuse) and `1` (reuse). Default is not to reuse.
-shutdown list ()	Base Shut Down Enable pin name, followed by its state (L or H).
-simulation_interval <number>	Value for mx_simulation interval in seconds. Default is `20e-9`.

-skip_read_model int (0)
Skip reading SPICE models.

-sleep list () Base Sleep Enable pin name, followed by its state (L or H).

-slews {list_of_slews}
List of slews to be used for characterization in ps. Overrides internal or template or library definition.

-targ_thresh <value> Defines target (end point) threshold as percentage of VDD.

-tbl_file_limit int (200)
Table file line number limitation. Default is 200 lines.

-temp <temperature> Temperature in degrees C for characterization.

-template <string> Name of template file. Do not use if `-ref_lib` is specified.

-template_vec <string> ()
Name of template vector.

-test1 list () Base Test1 Enable pin followed by its state for chip is in test1 mode (L or H).

-test_enable {base_name active_state}
Base Test Enable pin followed by its state for chip is in test mode (L or H). This is for specifying the a test mode that controls whether regular inputs or test inputs are active. For example, if a dual port memory has test enables named `TENA` and `TENB`, then the root name would be `TEN` and `{A B}` would be `-port_suffix`. If the instance is in test mode when the pin is low, the correct argument would be `{TEN L}`. (Standard Custom Instance flow)

-thread <number> Number of threads to be used for all aspects of characterization. Default is to use maximum available threads or licenses.

-trig_thresh <value> Defines trigger (starting point) threshold as percentage of VDD.

-unique_power int (0)
Enable -unique power in write_template.

-vendor <ARM|Virage|TSMC>
Name of Vendor. (Vendor Recharacterization flow). For more information on usage of -vendor, see Guidelines for Usage of -vendor and -pin_file in define_memory Flow.

-virtual_rails {paired_list_of_supplies_and_values}
Paired list of Virtual Rails and their expected voltage level. Value is in volts. Format is `{<supply1> <supply1_voltage> <supply2> <supply2_voltage> ...}`.

-words <number_of_words>
Number of Words in the memory instance. (Standard Custom Instance Flow and Vendor Recharacterization flow if cfg file is not provided).

-write_enable {base_name active_state}
Base Write Enable pin name, followed by its state for chip is writing (L or H). For example, if a dual port memory has write enables named `WENA` and `WENB`, then the root name would be `WEN` and `{A B}` would be `-port_suffix`. If the write enable is active low, the correct argument would be `{WEN L}`. (Standard Custom Instance flow).

-write_port list () List of write ports pins

-write_timer list () Base Write Timer pin name

-xps_smode_power <string>
Use XPS s-mode for power. XPS must be selected as FastSPICE to use this option.

{cell_names} List of Cells (required). Default: `none`.

The **define_memory** command describes the attributes and characterization settings of a memory instance to be characterized. This command is used along with the **char_memory** command to create all needed files for characterization and run the characterization. It is required that the **define_memory** command must be executed before the **char_memory** command. This command can be used to describe an instance of standard functionality (Standard Custom Instance Flow) or an instance designed by a third-party IP vendor (Vendor Recharacterization Flow). In this flow a directory called `mx_setup` is created containing the files needed to characterize the memory instance. Once a memory instance has been run through the flow, the user has the ability to edit the gerneated files and rerrun by setting the `setup_reuse` flag to `1`.

The define_memory command supports different models for timing, power, and leakage. For example:

```
define_memory \
-models [pwd]/DATA/include_models \
-models_power [pwd]/DATA/include_models_power \
-models_leakage [pwd]/DATA/include_models_leakage \
... ...
```

Note: If there is no `-models_power` or `-models_leakage`, the **define_memory** command uses `-models` as the default model files.

Usage of define_memory flow

- Automated vendor re-characterization flow: when you are using memories from a third-party vendor, you might be adding an uncharacterized PVT or verifying the compiler accuracy. An example of the usage of **define_memory** for the recharacterization of a Vendor Instance:

  Example 1

  ```
  define_memory \
      -ref_lib [pwd]/ref_lib/SRAM_2048x16.lib \
      -netlist [pwd]/netlist/SRAM_2048x16.spf \
      -vendor "TSMC" \
      -global_voltage 1.1 \
      -temp 25 \
      -models [pwd]/models/include_tt_model.sp \
      -mx_setting [pwd]/mx_settings.tcl \
      SRAM_2048x16
  ```

  Example 2: Two port Register file: Memory with 2 dedicated ports (1R 1W)

  ```
  CLKR: clock pin for read port
  CLKW: clock pin for write port B
  AA: address bus for port A [read]
  AB: address bus for port B [write]
  D : data bus  [write port only]
  Q : data out bus [read port only]
  WEB: write enable (active low)

  define_memory \
          -netlist [pwd]/regfile.sp \
          -template [pwd]/template.tcl \
          -mx_setting [pwd]/mx_setting.tcl \
          -additional_tables {[pwd]/new_table.tbl} \
  ```

```
        -vendor tsmc \
        -foundry tsmc \
        -design rf \
        -bitcell 2prf \
        -number_of_ports 2 \
        -global_voltage 0.9 \
        -temp 0 \
        -models [pwd]/models/include_ss \
        -process_node 45nm \
        -virtual_rails [list \
                    VDDI 0.9 \
                    VSSI 0 \
               ] \
        regFile
```

- Automated standard instance flow: the design is of standard functionality and can be characterized using define_memory flow. Anexample of the usage of **define_memory** for the characterization of a Standard Custom Instance:

  Example 1 pinout (Single port SRAM: Standard SRAM without test mode):

```
CLK:        clock pin
ADR[10:0]:  address bus
DIN[15:0]: data bus
Q[15:0]:   data out bus
CEN:        chip enable (active low)
WEN:        write enable (active low)

define_memory \
        -netlist SRAM_2048x16.spf \
        -clock CLK \
        -address ADR \
        -data_in DIN \
        -data_out Q \
        -chip_enable {CEN L} \
        -write_enable {WEN L} \
        -rail {VDD 1.0 VSS 0} \
        -temp 25 \
        -foundry TSMC \
        -models [pwd]/models/include_tt_model.sp \
        -template [pwd]/template.tcl \
        -mx_setting [pwd]/mx_settings.tcl \
        SRAM_2048x16
```

Example 2 pinout (Dual port SRAM (DPSRAM) with test mode)

Dual port SRAM, Standard SRAM with two ports which are symmetric and capable of reading and writing.

```
CLKA:         clock pin for port A
CLKB:         clock pin for port B
TCLKA:        test mode clock pin for port A
TCLKB:        test mode clock pin for port B
ADRA[10:0]:   address bus for port A
ADRB[10:0]:   address bus for port B
TADRA[10:0]: test address bus for port A
TADRB[10:0]: test address bus for port B
DINA[15:0]:   data bus for port A
DINB[15:0]:   data bus for port B
TDINA[15:0]: test mode data bus for port A
TDINB[15:0]: test mode data bus for port B
QA[15:0]:     data out bus for port A
QB[15:0]:     data out bus for port B
CENA:         chip enable for port A (active low)
CENB:         chip enable for port B (active low)
TCENA:        test mode chip enable for port A (active low)
TCENB:        test mode chip enable for port B (active low)
WENA:         write enable for port A  (active low)
WENB:         write enable for port B  (active low)
TWENA:        test mode write enable for port A  (active low)
TWENB:        test mode write enable for port B  (active low)
TENA:         test mode select for port A  (active low)
TENB:         test mode select for port B  (active low)

define_memory \
        -netlist SRAM_2048x16.spf \
        -clock CLK \
        -address ADR \
        -data_in DIN \
        -data_out Q \
        -chip_enable {CEN L} \
        -write_enable {WEN L} \
        -test_enable {TEN L} \
        -port_suffix {A B} \
        -clk_bist_prefix T \
        -bist_prefix T \
        -rail {VDD 1.0 VSS 0} \
        -temp 25 \
        -foundry TSMC \
        -models [pwd]/models/include_tt_model.sp \
        -template [pwd]/template.tcl \
        -mx_setting [pwd]/mx_settings.tcl \
        SRAM_2048x16
```

Example 3: Read Only Memory (ROM) with standard functionality where each read operation finishes in single clock cycle

```
    define_memory \
        -netlist [pwd]/rom.sp \
        -template [pwd]/rom_template.tcl
        -mx_setting [pwd]/mx_setting.tcl
        -bitcell rom \
        -design rom \
        -global_voltage 1.08 \
        -temp 125 \
        -words 4096 \
        -bits 32 \
        -models [pwd]/include_ss \
        -rail {VDD 1.08 VSS 0} \
        -clock CLK \
        -data_out Q \
        -address A \
        -chip_enable {CEN L} \
        -unique_power 1 \
        rom
```

define_template

-type {delay | power | ccs | ccsn_dc | constraint | ecsm | mpw}
Template type (REQUIRED)

-index_1 {values}	List values to be used as the first index (REQUIRED)
-index_2 {values}	List values to be used as the second index
-index_3 {values}	List values to be used as the third index
<template>	Name of template

The **define_template** command defines a template to be used for characterization. The **type** argument specifies the type of template being defined. How each cell is to be characterized is defined by associating the defined template with the appropriate argument of the **define_cell** command. Multiple **define_cell** commands can reference a single template.

Note: Internally **define_template** uses a fixed set of units (listed below). These cannot be changed, *however*, units may be changed when writing a library with the **set_units** command.

current	1mA (milliamps)
power	1nW (nano watts)
resistance	1kohm (kilohm)
time	1ns (nano seconds)
voltage	1V (volts)
capacitence	1pf (pico farad)

The **delay** template type is used for delay characterization using input slew and output load. It requires both **index_1** and **index_2** to be specified, where **index_1** represents the range of input slews and **index_2** represents the range of output loads.

The **power** template type is used for switching and hidden (internal) power characterization using input slew and output load. It requires both **index_1** and **index_2** to be specified, where **index_1** represents the range of input slews and **index_2** represents the range of output loads.

The **ccs** template type can be used for composite current source model (CCS) delay characterization. It requires **index_1** to be specified where **index_1** represents the range of the normalized voltage values to measure.

The **ccsn_dc** template type can be used for composite current source DC noise model characterization. It requires **index_1** and **index_2** to be specified where they represent a range of input/output voltages. If not specified Liberate uses a range of 29 voltage points from -Vdd to 2*Vdd. DC simulations are very fast, especially on a small CCB group of transistors extracted for noise stage simulations and therefore does not use much CPU time compared to the transient CCS noise models. It usually is not necessary to change the size of these tables from the default 29x29. It can be useful to optimize the size of the tables to avoid non-convergence errors at the extremes of the voltage range. The ccsn_dc template is global to all cells and is not included in the define_cell command.

The **constraint** template type can be used for timing constraint (setup, hold, removal, recovery) characterization. It requires both **index_1** and **index_2** to be specified, where **index_1** represents the range of input slews of the data signal and **index_2** represents the range of input slews of the reference signal (clock, reset etc.).

The **ecsm** template type can be used for effective current source model (ECSM) characterization. It requires **index_1** to be specified, where **index_1** represents the range of the normalized voltage values to measure.

The **mpw** template type is used when a two dimensional mpw table is required. The user must provide both **index_1** and **index_2**. In addition, the **define_cell -mpw** option must reference the mpw template. By default, if the user does not provide an **mpw** template, the **mpw** timing constraint will be characterized as a single attribute. If the variable **mpw_table** is set to a **1**, then Liberate will output a one dimensional mpw table using the **define_cell** constraint index_1 list of values. Use the **define_template type mpw** only when a 2 dimensional mpw table is required.

All **index_*** entries for all the library constructs should be monotonically increasing.

This command must be used before **char_macro**.

Examples:

```
# Delay template for 3 input slews, 3 output loads
define_template -type delay \
-index_1 {0.025 0.1 0.25} \
-index_2 {0.0010 0.015 0.100} \
delay_3x3
# Power Template for 3 input slews, 3 output loads
define_template -type power \
-index_1 {0.025 0.1 0.25} \
-index_2 {0.0010 0.015 0.100} \
power_3x3

# Timing constraint template for 2 input slews
define_template -type constraint \
-index_1 {0.025 0.25} \
-index_2 {0.025 0.25} \
constraint_2x2

# ECSM  template for 5 intervals
define_template -type ecsm \
-index_1 {0.05 0.2 0.5 0.8 .95} \
ecsm_5

# CCS Noise DC Curve with 11 input and 11 output voltages.
define_template -type ccsn_dc \
-index_1 {-1.0 -0.5 -0.2 -0.1 0.0 \
 0.1 0.2 0.5 1.0 1.2 1.5} \
-index_2 {-1.0 -0.5 -0.2 -0.1 0.0 \
 0.1 0.2 0.5 1.0 1.2 1.5} \
ccsn_dc_template
```

Guidelines for Usage of -vendor and -pin_file in define_memory Flow

- If vendor information is unknown and unsupported, specify `-data_in`, `-data_out`, `-clock`, or any other information.
- If vendor information is known and supported, specify `-vendor`. The **define_memory** command generates a `pin_file` [`%outdir/mx_setup/pin_file`].
- If `pin_file` is correct, use it and also use the table that **define_memory** generates.
- If `pin_file` has an error, modify it manually (`pinfile.modified`). Also, modify the removing `-vendor` from the **define_memory** command and instead add `-pin_file [pwd]/pinfile.modified`. This implies that the `-vendor` and `-pin_file` options should not be used simultaneously.

define_table

Specifies the vector table files. (See Specifying Input Stimuli.)

mxtables {table_files}	List of vector table files. (Required)
cell <name>	Cell name. (Required)

Example:

```
define_table delay.tbl myMemCell
```

hspice_lis_2_waves

lis <file>	The HSpice .lis filename.
-nets <list>	List of nodes to show.

Use this command to have Liberate_MX convert HSpice waveforms from a .lis file into a data file format that the Liberate MX lwave program can read.

Specify the -nets list to limit the waveform display to a select number of nets If no '-nets' option is specified, all nodes available for display in the lwave program. Example:

```
hspice_lis_2_waves sim.lis
```

mx_match_node

-bitline <bitline_name>
Filters result of the command by returning only nodes that intersect the specified bitline.

-wordline <wordline_name>
Filters result of the command by returning only nodes that intersect the specified wordline.

-core <core_node_name>
Filters result of the command by returning only nodes that intersect the specified core node.

-data_in <data_in_name>
Filter result of the command by returning only nodes that propagate from the specified data input.

-senseamp <sense_amp_node_name>
Filter result of the command by returning only nodes that propagate into the senseamp containing the specified node.

{pattern_or_keyword}	Pattern or keyword to use when matching nodes. Pattern is any TCL regexp pattern. Valid keywords are: bitline, wordline, core, bitline_precharger, senseamp_precharger, senseamp_enable
{cell name}	List of cells.

The **mx_match_node** command searches the netlist which has been read by **read_spice** and return the nodes satisfied by the filtering options and the pattern or keyword argument. Intersection criteria are handled as nodes that are connected through a corecell (in the case of bitline, wordline or core) or through physical connection (for precharge and senseamp).

Examples:

- To return all bitlines in the instance:

  ```
  mx_match_node bitline <cellname>
  ```

- To return all bitlines that propagate from D<0>:

  ```
  mx_match_node -data_in D<0> bitline <cellname>
  ```

- To return all nodes that contain “BLB”:

```
mx_match_node *BLB* <instname>
```

mx_recover_clean

-debug	Prints debug info on cleanup steps.
{remove_double_groups}	List of errors to look for / correct.

Analyzes characterization data for issues and corrects them, if possible. *(See mx_recover_info for full description of flow.)*

mx_recover_info

<no options>	Outputs a usage statement detailing the commands below.

Liberate MX provides for a recovery flow using the following three commands:

- mx_recover_setup: Generates a place holder LDB ready to receive characterization LDBs for subsequent merging.
- mx_recover_clean: Analyzes characterization data for issues and possibly corrects them.
- mx_recover_merge: Merges characterization LDBs in to final LDB.

The procedure for this flow is to create 3 separate Tcl scripts, each one containing a separate command (as shown in the examples below). To make a complete flow, create one more script that calls each of these scripts in turn:

```
# mx_recover_setup.tcl
mx_recover_setup <cell>.ldb.gz


# mx_recover_clean.tcl ... This is optional if cleaning is not needed.
mx_recover_clean {remove_double_groups}


# mx_recover_merge.tcl
mx_recover_merge
```

Note: The "clean" step is optional, and may be omitted if there's no need to correct existing characterization LDBs.

mx_recover_merge

-debug	Debug info on merging step and intermediate merging LDBs.
-merged_ldb <name>	User name for resultant LDB

Merges (possibly cleaned up) characterization LDBs in to final LDB. *(See mx_recover_info for full description of flow.)*

mx_recover_setup

<ldb>	Full path name to LDB that needs to be regenerated.

Generates a place holder LDB ready to receive characrization LDBs for subsequent merging. *(See mx_recover_info for full description of flow.)*

mx_report

-ldb {*list*}	Paths to mx reference/compare ldb's - as written by `write_ldb` command in respective runs
-lib {*list*}	Paths to MX reference/compare libraries - as written by `write_library` command in respective runs
-rpt <*file_name*>	Name of output report (do not include file name extension.)

This command generates a series of reports from a regression run. Output is an HTML file (and supporting directories) that can be viewed as-is or can be opened as a Microsoft Excel document containing a workbook with 6 separate sheets:

- Summary
- Lib Compare
- Measurement Compare
- Partitions Compare
- Fast-Spice vs. Full-Spice
- Statistics

mx_set_clockprop

This command deprecated. Please use **mx_set_domainprop**

mx_set_constprop

{<node> <value>}	Lists nodes and their logic values, where to start constant propagation from.

The **node** is the pin, bus or bus range that must be considered during constant propagation. Note that if a bus – or a partial bus – is specified, its constant value must be specified in hexadecimal notation - see example below. Constant propagation is not a required step but helps in various spatial analysis steps during the preprocessing step. Once a value is specified, via the **mx_set_constprop** command for a pin or a bus, that same value is used during the FastSPICE simulation run and therefore does not need to be specified again in the stimuli or truth table. Example:

```
# Set test mode pins to '1000' value
mx_set_constprop {{tm[3:0] 0x8}}
```

mx_set_domainprop

{ <node> <start | stop | enable | force> <domain>}
Lists nodes to start, stop, enable or force domain propagation.

The **node** is a pin or internal wire node name, **domain** is a primary input name. Using **start** will begin propagation for the given domain at the given node while **stop** will stop propagation for that domain at that node. Using **enable** will allow propagation through a gate that is controlled by the node while **force** will make the node a descendent of the given domain. By default, domain propagation starts at primary input pins and continues through combinational logic gates until it reaches a sequential element that has a non-clocked pin. Example:

```
# For primary clock CLK: stop it at node A, restart it at node B,
# force it at node C. Anytime a CLK derived node meets with D,
# let it go through

mx_set_domainprop {\
  {A stop CLK} {B start CLK} \
  {C force CLK} {D enable CLK}\
}
```

See also mx_domain_propagation.

mx_set_finesim_param
mx_set_ultrasim_param

<list>	List of FineSim and UltraSim parameters that will be used during FastSPICE simulation.

These commands are used to pass parameters to the appropriate external simulator. Parameters are passed as a list of name-value pairs.

Note: These parameters can also be specified in a table format using define_table. Parameters specified in a table will *override* parameters that are specified with a Tcl command. (See fastsim_deck.) For example:

```
px_set_ultrasim_param { \
    {simpreset 5} \
    {pn_level 5} \
    {cgnd 1e-15} \
    {sfe_compaction 0} \
    {keepparaname 0} \
    {rshort 2} \
    {hier_delimiter .} \
    {dc_turbo 3} \
    {rcr_fmax 1G} \
}
```

report_measure

-all	When set to `true`, reports all activities. When `false`, reports only failed activities. Default: `false`.
-abs_diff <value>	Reports absolute error bigger then `rel_diff`. Default is 1n.
-rel_diff <value>	Reports relative error bigger then `rel_diff`. Default is 0.05, which is equal to 5%.
measname <string>	measure name to be validated.
cell `<string>`	Cell names.

This command is used to report the difference between relax or stress simulation results. The `-all` shows all activities for both relax or stress run. The `abs_diff` and `rel_diff` are used to set the absolute and relative differences of measure values between relax/stress simulation results.

set_gnd

-ignore_power	Ignore the power contribution from this supply net.
-no_model	Request to not include this supply in the output .lib.
-virtual	Treat the net as logic 0 for static analysis, but as a regular node for dynamic simulation.
-waveform <name>	Provide a file containing a text description of a waveform as a time/value pair list.
<net_name>	Name of ground supply net
<voltage>	Voltage value (in Volts)

set_vdd

-ignore_power	Ignore the power contribution from this supply net.
-no_model	Request to not include this supply in the output .lib.
-virtual	Treat the net as logic 1 for static analysis, but as a regular node for dynamic simulation.
-waveform <name>	Provide a file (full path name) containing a text description of a waveform as a time/value pair list.
<net_name>	Name of power supply net
<voltage>	Voltage value (in Volts)

In Liberate MX you can specify power and ground nodes using the **set_gnd** and **set_vdd** commands.

The **set_gnd** and **set_vdd** commands define the names of ground nets and power supply nets respectively. Multiple **set_gnd** and **set_vdd** commands can be specified.

If the **ignore_power** option is set, the contribution of the specified supply net will be ignored. That is, the current in this supply net will not be summed into any power measurement.

If the **virtual** option is set, the node is treated as a logic 0 or logic 1 for the static analysis step, but as a regular node for dynamic simulation and characterization. No contribution to internal power or leakage will come from the node. See **mx_find_virtual_rails** to instruct the tool on how to report design nodes that should be defined as **virtual.**

Liberate MX will automatically identify the following net names (case insensitive) as ground supplies and will set them to zero volts:

- 0, GND, VSS

Use the **set_gnd** command to set them to alternative values. It is not recommended to attempt to change the voltage of the ground net **0** since this is considered the reference ground.

Liberate MX will automatically identify the following net names (case insensitive) as power supplies and will set them to the default voltage specified by the **set_operating_condition** command.

- VDD, VCC

Use the **set_vdd** command to set them to alternative values

At 45nm and below it is not uncommon to find power-gating structures used to generate internal rails. These virtual nodes can be considered as logic 0/1 when using static analysis techniques to identify clock trees, latches, etc., but should be considered as regular nodes during dynamic simulation. Option **-virtual**, can be used for such nodes. In case the names for such nodes are not easily identifiable, for example in a fully RC extracted net list, parameter **mx_find_virtual_rails** can be used – see parameter description in net chapter. Example:

```
set_vdd -no_model vss1 1 0.9
set_vdd -type back_up -cells cell1 vdd2 0.9
```

Using option **'–no_model'**: the vdd1 will not appear in cell1 since it is globally no_model and no local vdd1 set to cell1.

set_virtual

-vdd	Treat the net as logic 1 for static analysis.
-gnd	Treat the net as logic 0 for static analysis.
<net_name>	Name of virtual rails net.
<voltage>	Voltage value (in Volts).

This command specifies the virtual ground and power nodes. It is a command for virtual net definitions. The selection of a virtual net is controlled by the mx_virtual_rail_opposite_device_minimum_factor variable.

Note: In earlier releases, the virtual net definitions functionality was used by the `-ignore_power`, `-nomodel`, and `-virtual` arguments of the **set_vdd** and **set_gnd** commands.

spv_tcl_2_waves

-debug	Print debug info
waveforms_file <string>	Specify waveforms file name
nets { list of nets }	Instructs the tool to require activity information be available for memory core bits. Default: 0

Use the spv_tcl_2_waves command to help debug FastSPICE waveforms when tcl waveform file is too large to be loaded from TCL interpreter. The command can be used to display waveforms using the Liberate MX lwave utility for the list of nets as stored in file 'waveforms_file' that is output bye the FastSPICE simulation (when mx_spv_api is 1).

validate_macro

-validsim "simulator_name"	The circuit simulation program to be used for validation. Default: `“xps"`
-thread <number>	Specifies the maximum number of threads to be used on the host machine. Default: `0` (Let Liberate MX decide).

This command performs memory validation. Each memory that has a `define_cell` command and a netlist loaded using the `read_spice` command is validated.

-validsim this argument specifies which external simulator to use. Supported simulators are: `"aps″`, `finesim"`, `"xa"`, and `"xps"`. The default is `"xps"`.

-thread specifies the maximum number of threads to use on the host machine. The default value is `0`. When set to `0`, Liberate MX automatically uses multiple threads up to the total number of available CPUs on the host. The following steps support multithreading: FastSPICE simulation, results acquisition, arcs selection, and file IO operations. The number of parallel FastSPICE runs is determined by the maximum number from among the different tables files provided and the number specified by the thread argument. For more information, see Specifying Input Stimuli.

Examples:

```
# Validate memory(s) defined via a previous define_cell command.
# Use Spectre APS for validation.
# Validation using 2 threads
validate_macro -validsim "aps" -thread 2
```

For more information on library validation, see Appendix D, “Liberate MX Timing Validation Flow.”.

validate_measure

measname <string>	measure name to be validated.
cell <string>	Cell name.

This command validates the measure defined by **define_measure**.

validate_memory

Creates all needed validation files when used after the define_memory command in the Liberate MX validation flow.

-work_dir "string"	The path where the `mx_setup` directory containing the reusable setup files is created. Default: `mx_setup`

This command can be used to run the validation on an instance of standard functionality in automated standard instance flow. For more information, see Appendix E, “Basic Flow for Validating User-Defined Criteria.”

write_library

-bus_syntax "<>" | "[ ]" | "( )"
Controls the bus_syntax used when outputting the library and the `bus_naming_style` attribute.

-capacitance_only Omit *fall / rise_capacitance* attributes.

-capacitance_range	Output rise/fall_capacitance_range attributes.
-ccs	Include CCS data
-ccsn	Include CCSN (noise) data
-cells {*cell_names*}	List of cell names. Default: *all cells*
-dcnoise_abstol <*tolerance*>	Tolerance used to group similar DC templates. Default: 1e-6 Volts (1 uV)
-dcnoise_prefix <*prefix*>	Prefix used for writing DC noise templates. Default: "*DC_*"
-driver_waveform	request output of normalized driver waveform.
-driver_waveform_size <*value*>	Number of points in normalized driver waveform. Default: 500
-ecsm	Include ECSM data
-ecsmn	Include ECSM noise data
-em	Include electromigration data
-exclude	Exclude cells given by `-cells`
-expand_buses	Turns off the creation of buses and instead outputs individual pins.
-filename <*filename*>	Output file name
-gzip	Compress the output library using gzip
-indent <*number*>	Number of spaces to indent by. Default: 2
-overwrite	Overwrite existing `.lib` file
-precision <*precision*>	Format string to control the precision of the output values. Default: "*%g*"

-preserve_user_data_precision {*attributes***}**
List of attributes to have original precision preserved

-sdf_cond_equals {"==" | "===" | "== logical" | ""}
sdf_cond attribute style. Default: "" (none)

-sdf_edges Include *sdf_edges* attribute

-si Include SI data

-skip {*leakage* **|** *power* **|** *hidden_power* **|** *conditional_hidden_power***}**
List of data types to be filtered from the output library.
Default: do not skip any data

-swap_index_order Swap the index order for 2d tables.

-swap_index_order Swap the index order for 2d tables.

-user_data <*filename***>** User provided library data

<*libname***>** The output library name.

The **write_library** command outputs the library in Liberty format to a file given by the filename argument. If filename is not specified the library is written to `library_name.lib`. The gzip argument will compress the output file using gzip. If the output library file already exists, a warning will be given and a unique file name will be generated using the given name suffixed with a unique number. The overwrite argument will disable this automatic version control, and if the output library already exists, it will be overwritten.

cells controls which cells get written to the output library. If exclude is also set then only cells not listed in the `cells` list will be output. By default all cells get written. This option supports the use of a wildcard.

The `si, ecsm, ecsmn, ccs, ccsn`, and `em` arguments enable inclusion of Liberty SI, ECSM, ECSM noise, CCS timing, and CCS noise data in the output library if it exists in the characterized database (ldb). By default only leakage values and NLDM timing and power table data are written.

Example:

```
read_ldb <ldb>
write_library -ecsm  <ecsm.lib>
```

capacitance_only disables the output of `rise_capacitance` and `fall_capacitance` attributes. The output library will only have a single capacitance attribute. This option is useful for backward compatibility. Care should be taken when using this argument.

precision controls the precision used when writing out the library. The value for this argument must conform to standard Tcl formatting. The default value is "%g". The indent option specifies the number of space to indent, default 2. The `preserve_user_data_precision` will tell Liberate to preserve the precision of attributes in the user_data file and not to apply the precision to them.

`dcnoise_prefix` and `dcnoise_abstol` are used when writing DC noise templates. The `dcnoise_prefix` argument controls the prefix used when naming the templates. The `dcnoise_abstol` argument controls the merging of DC noise templates. If the noise values are less than this tolerance, then the templates will be merged into a single group.

The `sdf_edges` will enable the output of the `sdf_edges` attribute.

The `sdf_cond_equals` argument specifies how `sdf_cond` attributes are written. The following table explains supported values:

```
Value   Output
"==""a == 1'b1 && b == 1'b0 …"
"===""a === 1'b1 && b === 1'b0 …"
"== logical""a == 1 && b == 0 …"
default"a  && ~b …"
```

user_data file specifies a user provided library in Liberty format to be merged with the current library. This is useful to include non-characterized data such as wire-load models in the output library. Once this user-data is merged into the current library, all subsequent `write_library` commands will output the merged constructs as part of the output library. If this is not desired, then separate runs of Liberate consisting of `read_ldb` and `write_library` must be executed. Any valid construct that is present in the user-provided library that is not present in the current library database will be copied to the output library, with the following exceptions:

- Attribute `slew_derate_from_library` is not copied.
- Attributes `function`, `state_function` and `area` will override values in the current library.
- Groups `state_table`, ff and latch will override the equivalent groups in the current library.

The parameter **user_data_override** can be used to tell `write_library` to allow certain attributes in the `user_data` to override the characterized values.

preserve_user_data_precision specifies a list of attributes for Liberate to preserve the original precision in the `user_data` file. By default, Liberate will use the same precision as all other attributes.

capacitance_range requests the output of `rise/fall_capacitance_range` attributes into the library. Supported values are:

0: Omit

1: Include the rise and fall range spanning from the min of the *min_capacitance* values to the max of the *max_capacitance* values. This method has been reported to cause timing issues in PrimeTime. (Default)

2: Include the rise and fall capacitance ranges where both range limits are both set to the rise/fall capacitance attribute values:

```
rise_capacitance_range = "<rise_capacitance>, <rise_capacitance>"
fall_capacitance_range = "<fall_capacitance>, <fall_capacitance>"
```

swap_index_order swaps the index order for 2d tables. Default: Use the order specified by the `read_library`, `define_template` or `read_ldb` commands.

driver_waveform outputs normalized driver waveforms into the output library. For the output to include the driver waveform, the ldb/vdb must contain the driver waveform data. If the Tcl contains multiple `write_library` commands, the first command using this option will enable the waveform output for all subsequent `write_library` commands. Normalized driver waveforms will not be output for user defined PWLs which are incompletely specified or use wildcards.

driver_waveform_size sets the number of voltage points in the normalized driver waveform `index_2`. The normalized waveform currently uses an arbitrary number of voltage points uniformly distributed from gnd to vdd. The number of points can be controlled using this option. Default: 500.

ccs_compact outputs a library in the compact CCS format. This is currently a BETA feature.

skip disables the output of power arcs into the output .lib file. Supported values are: `power`, `hidden_power` and `conditional_hidden_power`. This capability is useful when characterizing a library using different SPICE models for timing & power. The characterization of power cannot be skipped when CCS data is desired since Liberate needs the hidden power simulations to generate the receiver pin caps. Specifying `hidden_power` would skip the output of `hidden_power` arcs. Specifying `conditional_hidden_power` would skip the output of `conditional_hidden_power` arcs.

Bus Support

write_library also supports buses. Buses can be defined by `define_cell`, in the ldb, or by the `define_bus` command. For each defined bus, a bus template is created in the library header (group name "type"). A `bus_naming_style` attribute is also created. For each bus, all the timing, power, CCSN data, etc., for that bus pin is represented once under the bus group with only `capacitance`, `min/max_transition` attributes given for each pin. Note that this can result in a loss in accuracy, as all the data is taken from the first bus bit (the from index). The `expand_buses` flag can be used to output a library with individual pins and no buses. The `bus_syntax` option can be used to change the bus syntax characters.

Bit-level Delay Modeling

By default, for an arc involving a bus, only the "worst case" representative is chosen for characterization. The worst case values are then applied to all elements of the bus. This is controlled by adding `-attribute altos_clone_arcs` to the command `define_arc`. These attributes can be set directly by the user, or can be controlled indirectly through the syntax shown below.

- ❑ **Worst-case determined using all elements of a bus** (default)

This is the default. The worst case will be determined from the full bus, whether it is specified as a full range of bits, ***or*** as individual bits.

Full range:

```
define_arc -type <...> -pin bus[MSB:0] ...
```

Single bits:

```
define_arc -type <...> -pin bus[MSB] ...
define_arc -type <...> -pin bus[MSB-1] ...
...
define_arc -type <...> -pin bus[0] ...
```

- ❑ **Worst-case determined within a range of bits**

The worst case will be calculated for the bits within the specified ranges, and applied to all bits in that range. In the example below, the worst case will be applied to bits in the range R0:R1 separately from bits in range R2:R3:

```
define_arc -type <...> -pin bus[R0:R1] ...
define_arc -type <...> -pin bus[R2:R3] ...
```

- ❑ **No worst-case; each bit considered separately**

All bits of the bus will characterized separately (no worst-casing). This is achieved by specifying "single bit ranges":

```
define_arc -type <...> -pin bus[MSB:MSB] ...
define_arc -type <...> -pin bus[1:1] ...
```

```
    define_arc -type <...> -pin bus[0:0] ...
```

Note:

When separate ranges are specified, only a fully expanded (bit-blasted) library is able to properly represent the different values for the different bits (write_library -expand_buses).

If the user instead choses to generate a fully compressed library (default in write_library) a worst-casing step will be done at the LDB level prior to generating the library.

Therefore if both an expanded and a bit blasted libraries need to be generated, the bit blasted should be generated first.

write_verilog

-cells {*cell_names***}**	List of cells to output. Default: all cells.
-delayed	Controls naming convention for creating "delayed" signals. Default: "delayed_%P" (where %P is the pin name.)
-exclude	Exclude cells from `-cells` list.
-fwire_prefix <"*prefix***">**	Prefix for internal wires for pin functions. Default "*int_fwire_*"
-indent <*number***>**	Number of characters to indent. Default: *use tab*
-merge	Merge in cell modules from `user_data`
-mpw_include_output_state	Include output pin logic state in mpw timing checks for "clear" or "preset" input signals.
-mux<*type***>**	Create MUX UDPs (user defined primitives) for mux functions, default use basic logic primitives
-no_edge	Exclude 'posedge' or 'negedge' on edge triggered arcs, default is to include edges.
-path<*path***>**	String used to denote a path, e.g "=>", "*>". Default "=>"
-sdf_version<*version***>**	SDF version, must be "**2.1**", or "**3.0**". Default: 3.0

-specparams	Output delay assignments to "specparams" rather than directly to delay values.
-split_notifier	When writing verilog modules for multi-bit cells, it is required to output separate notifier commands for each DFF. The -split_notifier option is used to output separate notifier commands for each DFF.
-timescale <*timescale*>	Verilog timescale. (1ns/10ps)
-twire_prefix	Prefix for internal wires of conditional timing constraint functions. Default "int_twire_"
-udp_prefix<*prefix*>	Prefix for built-in user defined primitives (UDPs). Set to "" to exclude UDPs. (Default: altos_)
-user_data<*filename*>	User Verilog file to merge timing info with.
<*verilog_filename*>	Output Verilog filename.

The **write_verilog** command creates a Verilog file for the current library. The Verilog is written to the given <*verilog_filename*>. A **.v** suffix will be added to filenames that do not end in **.v**. The `user_data` argument specifies a user provided Verilog file to merge with the generated Verilog data with. If a `user_data` file is provided, timing information (paths and any additional wires required to specify the conditions for those paths) are merged with the user file and written to the output file, replacing any existing user provided timing information. Without a `user_data` file, a complete Verilog file is written including function descriptions. The `write_verilog` command should not be used in the same run as the `char_macro`, `read_ldb` or `write_library` commands. Instead, it should be used in a separate Liberate run following a `read_library` command. This is because the Liberty file may have formatting that is required for the Verilog output to be properly formatted.

Example:

```
read_library my.lib
write_verilog my.v
```

cells controls which cells get written to the output. If `exclude` is also set then only cells not listed in the `cells` list will be output. By default all cells get written. This option supports the use of a wildcard.

delayed controls the naming convention for creating "delayed" signals. When using user-data with write_verilog it is necessary to match these delayed signals with the equivalent signals used in the user-provided functional description. By default delayed output signals are created for the signals passed to timing checks such as setup-hold and/or recrem in Verilog.

The delayed option uses a special variable "%P" to return the pin name and combine it with a user-defined string. Example:

```
write_verilog -delayed "delayed_%P"
```

For a pin named "myPin", this will produce a delayed signal name "delayed_myPin". Some examples are below:

Example 1:

```
module DFFSRN (QN, D, CP, RN, SN);
    output QN;
    input D, CP, RN, SN;
    reg notifier;
    wire delayed_D, delayed_CP, delayed_RN, delayed_SN;

    // Function
    ....
    // Timing
    specify
        ...
        $setuphold (posedge CP, posedge D, 0, 0, notifier,,, delayed_CP, delayed_D);
        ...
        $recrem (posedge RN, posedge CP, 0, 0, notifier,,, delayed_RN, delayed_CP);
        ...
    endspecify
endmodule
```

Example 2:

```
write_verilog -delayed "dly_%P"
```

... will produce:

```
    wire dly_D, dly_CP, dly_RN, dly_SN;
    ...
    $setuphold (posedge CP, posedge D, 0, 0, notifier,,, dly_CP, dly_D);
```

Example 3:

```
write_verilog -delayed "%P_d"
```

... will produce:

```
    wire D_d, CP_d, RN_d, SN_d;
    ...
    $setuphold (posedge CP, posedge D, 0, 0, notifier,,, CP_d, D_d);
```

The default name for these delayed signals is "**delayed_<pin_name>**" (delayed_%P) where <pin_name> is a pin that is involved in a timing check.

To <u>turn off</u> generating delayed signals, use "" (empty double quotes). Example:

```
module DFFSRN (QN, D, CP, RN, SN);
    output QN;
    input D, CP, RN, SN;
    reg notifier;

    // Function
    ....
    // Timing
    specify
        ...
        $setuphold (posedge CP, posedge D, 0, 0, notifier);
        ...
        $recrem (posedge RN, posedge CP, 0, 0, notifier);
        ...
    endspecify
endmodule
```

merge includes cells not specified in the library but present in the `user_data` file.

indent specifies the number of spaces to use for indentation. Default: tab.

specparams causes delay assignments in the Verilog to be assigned to specparam variables rather than directly to values. The path argument controls the delimiter used for delay assignments, either => or *>.

sdf_version controls the format of the output Verilog for use with SDF annotation. Set to **3.0** to generate a Verilog file that is compatible with SDF version 3.0. SDF version 3.0 permits *recrem* constructs in the Verilog to represent recovery and removal of timing constraints.

twire_prefix is the prefix used for internal wires created when generating additional functions for state dependent timing constraints. The `fwire_prefix` is the prefix used for internal wires created when generating logic functions. The `udp_prefix` is the prefix used for user defined primitives that are created for latches and/or flip-flops. Set `udp_prefix` to a null string to exclude generating user defined primitives.

mpw_include_output_state requires that the `sdf_cond_style` variable is set to 1. If not set, it will force the setting. Note that this variable should be set prior to creating an equivalent library (.lib) with `write_library`, to ensure consistency between the library and the Verilog. Otherwise, there may be warnings during SDF back-annotation. The `mpw_include_output_state` option should be used before `read_library`, `char_macro`, or `read_ldb`. When this variable is used, the mpw check ($width) will only be checked by Verilog when the output pin of the cell is high for clear inputs and low for preset inputs. The Verilog will contain extra "timing" gates to add the logic necessary to "and" the logic state of the output pin to logic representing the "when" condition given in the library for each `min_pulse_width` timing arc.

mux converts pins whose functions are a 2x1 or 4x1 mux into a pre-defined, user-defined primitive (UDP) named `altos_mux2` and `altos_mux4` respectively.

The following Tcl variables can also be used to control the format of the Verilog output.

verilog_delay_value	The delay value. Default: 0
verilog_delay_Zvalue	The delay value for tristates. Default: 0
verilog_delay_clk2q_value	The delay value for clock to Q arcs on sequential cells. Default: 0
verilog_IQ	The name map for the internal state of flip-flops (e.g. IQ) to the Verilog state function.
verilog_IQN	The name map for the internal state of flip-flops (e.g. IQN) to the Verilog state function.
verilog_start_skip	Line to mark the start of the timing section in the **user_data** file which is to be replaced. Must match exactly apart for leading or trailing white space. Default: "specify"
verilog_stop_skip	Line to mark the end of the timing section in the **user_data** file which is to be replaced. Must match exactly apart for leading or trailing white space. Default: "endspecify"

This command must be used after a database has been loaded.

Example:

```
read_library my.lib
# Output a Verilog file
write_verilog -user_data my_verilog my.v
```

6 Liberate MX Variables

This chapter describes the following Liberate MX specific variables that impacts memory library creation.

Note: Liberate MX specific variables are set using the `set_var` command.

c...	
constraint_glitch_peak	constraint_glitch_peak_mode
constraint_glitch_peak_max	
d...	
debug_flow	
e...	
extsim_deck_include	extsim_model_include
f...	
fastsim_cmd	fastsim_cmd_option
k...	
keep_empty_cells	
l...	
lic_max_timeout	lic_queue_timeout
mx...	
mxtable_dontcare_value	mxvw_min_glitch_peak
mx_a...	
mx_active_fanout_channel_include	mx_auto_char_params
mx_active_load	mx_autoprobing_hold_level
mx_active_load_thresh	mx_autoprobing_setup_level
mx_arc_report	

mx_b...	
mx_bisection	
mx_c...	
mx_char_bundle_size	mx_clone_if_uda
mx_char_virtual_as_rail	mx_const_prop
mx_check_arcs	mx_constraint_ocv_factor
mx_check_arcs_exit_on_missing	mx_corecell
mx_clock2clock_constraints	mx_create_if_dynamic
mx_clock_tree_report	mx_create_if_uda
mx_d...	
mx_debug	mx_domain_propagation
mx_delay_ocv_factor	mx_dpartition_inactive_tie
mx_dir	mx_dynamic_include_full_core
mx_distributed_sim	
mx_f...	
mx_failed_char_report	mx_find_memcore_numbit_threshold
mx_fastsim_auto_ic	mx_find_memcores
mx_fastsim_clock_skew	mx_find_stack_loads
mx_fastsim_input_slew	mx_find_virtual_rails
mx_fastsim_load	mx_fix_pin_vdd
mx_fastsim_reuse	mx_full_rail_tol
mx_find_arrays	mx_fullsim_measurement
mx_g...	
mx_greybox	mx_greybox_constraint_method
mx_i...	
mx_inputcap_ldb_reuse	
mx_l...	
mx_ldbs_reuse	

mx_m...	
mx_margin_report	mx_mpw_false_probe_delay_threshold
mx_mcf	mx_mpw_measurement _duration
mx_min_period_latch_component_mode	mx_mpw_mode
mx_min_period_mode	mx_mpw_probe
mx_monitor_memcore	mx_mpw_probe_lower_fall mx_mpw_probe_lower_rise mx_mpw_probe_upper_fall mx_mpw_probe_upper_rise
mx_mpw_allow_same_probe_on_both_rise_and_fall_clock_tree	mx_mxtable_interpret_read_write_cycle_keywords
mx_n...	
mx_negedge_clock	mx_noise_ldb_reuse
mx_o...	
mx_output_require_fullrail_switch	
mx_p...	
mx_partition_name_use_arc	mx_power_assign
mx_pathdelay_hold_clock_margin	mx_power_ldb_reuse
mx_pathdelay_hold_data_margin	mx_power_single_point
mx_pathdelay_setup_clock_margin	mx_preprocess
mx_pathdelay_setup_data_margin	mx_probe_peak_currents
mx_pincap_char	mx_probes_report
mx_posedge_clock	
mx_r...	
mx_read_spice_exit_on_missing_file	mx_remove_rc_timing
mx_remove_false_ic_group	mx_retaining_time
mx_remove_rc_pincap	mx_ring_model_fold

mx_s...	
mx_seq_probing	mx_skip_autoprobing
mx_setup_seq mx_hold_seq mx_setup_comb mx_hold_comb	mx_skip_print
mx_simulation_interval	mx_spv_api
mx_t...	
mx_timing_ldb_reuse	mx_timing_report_debug
mx_timing_report	
mx_v...	
mx_verbose	mx_virtual_rail_opposite_device_minimum_factor
mx_virtual_rail_auto_mode	mx_virtual_rail_minimum_xtrs
mx_w...	
mx_whitebox_active_coupling_threshold	mx_whitebox_model_file
mx_whitebox_active_wire_threshold	mx_whitebox_monitor_memcores
mx_w...	
mx_zip_partition_deck	

constraint_glitch_peak

<value>	Glitch height as a ratio of supply used in characterizing timing constraints (setup, hold, recovery, removal). Default: 0.1 (10%)

This parameter is used to specify the maximum size of logic glitch permitted on the constraint output pin before an arriving signal is deemed to fail a timing constraint.

The set_constraint_criteria command can also be used to set this variable. If both this variable and the set_constraint_criteria are used, the last one executed will set the value to be used by MX.

This variable must be used before **char_macro**.

constraint_glitch_peak_max

<value>	Specifies the maximum threshold for constraint_glitch_peak_mode. Default: 0.5

If the new threshold from using constraint_glitch_peak_mode exceeds the ratio of this variable times vdd , then the threshold will be limited to the voltage represented by the ratio of vdd specified by this variable. This can result in a search bound error.

This variable must be used before **char_macro**.

constraint_glitch_peak_mode

< 0 l 1 l 2 >	Apply `constraint_glitch_peak` on top of inherent glitch. Default: 0 (Recommended: 1)

Many cells exhibit inherent glitches immediately upon clock transition. This is a glitch that occurs on a node as a direct result of the clock switching and is not related to any race condition between data and clock. If this inherent glitch occurs at a node that Liberate identifies as the probe node, then the logfile will contain the warning message "Too close to search bound". If the constraint measurement criteria is glitch, then it is possible that an inherent glitch occurs on the probe node. Setting `constraint_glitch_peak_mode` can work around this by accounting for the inherent glitch.

0: Don't apply `constraint_glitch_peak` on top of inherent glitch. (Default)

1: Liberate will measure the inherent glitch magnitude (noise on the net) and then add that to the `constraint_glitch_peak` to use as a new threshold. If the new threshold exceeds `constraint_glitch_peak_max`, the threshold will be limited to `constraint_glitch_peak_max`. This helps prevent warnings about "Too close to search bound" for glitch-based constraint measurements in the Liberate log file. (Recommended)

2: Operates the same as option 1, but pertains to internal nodes. Only clock-gater hold results will be impacted in an entire library. Non-sequential setup/hold are not affected as long as `constraint_async_probe_internal` = 0.

This variable must be used before **char_macro**.

debug_flow

< 1x1 l 2x2 >

This variable can be used to speed up the characterization by reducing the template as defined by the **define_template** command to either a `1x1` or a `2x2` data matrix. This can significantly decrease the runtime for a quick debug flow.

You can check selected points in the slew/load matrix by shrinking the template to a `2×2` or `1×1`:

A `2×2` data matrix uses the first and last values in each index.
A `1×1` data matrix uses the first value in each index.

Example:

```
set_var debug_flow 2x2
```

This variable must be set before the first **define_template** command.

extsim_deck_include

< 0 | 1 >	Controls how the FastSPICE simulation deck is written out. Default: 1

The **extsim_deck_include** command lets you control how the FastSPICE simulation deck is written out. When set to 1, only the original netlist is included via an .inc statement. To have the full netlist reported into the deck, set the **extsim_deck_include** variable to 0.

Example:

```
    #to .include original netlist use by FastSPICE
    #simulation.
    set_var extsim_deck_include 1
```

extsim_model_include

<value>	Specify full path to a file that will load the SPICE models. Default: Use flattened models.

Use this option to specify a full path to a file that will load the models when using an external SPICE simulation engine. If a full path is not provided, an error will result. Normally, Liberate MX will use flattened models in the external simulation input decks. When this variable is used, Liberate MX will use the file specified instead of the flattened models in the external simulation input deck. Liberate MX will place a statement in the extsim SPICE decks such as:

```
    .include <extsim_model_include_file>
```

Note that, for 40nm and below, the recommended flow is to use all three -- **extsim_model_include**, **extsim_deck_include**, and **define_leafcell** options.

This variable must be used before **char_macro.** Example:

```
set_var extsim_model_include "/home/user1/models/include_ff"
set_var extsim_deck_include 1
```

Where include_ff looks like:

```
.include '/home/user1/models/models.l' ff
```

fastsim_cmd

<path to executable>	Specifies the path to an executable to be used for simulating this table file.

fastsim_cmd_option

<command options>	Specifies command line options to be passed to the executable used for simulating this table file.

keep_empty_cells

< 0 I 1 >	Determines how to handle a cell when the netlist is empty. Default: 0 (Treat empty cells as an error condition.)

A cell is available for characterization when all of the following conditions are true:

- There is a **define_cell** command and a netlist that has been read in using the **read_spice** command.
- The **char_library** command does not have the `-cells` argument, or the `-cells` argument includes the cell name and the `-exclude` argument is not used.

0: Generates an error condition when an empty cell subcircuit is found. The cell will not be included in the .lib file. (Default)

1: Generates a warning condition when an empty cell subcircuit is found. The cell will be included in the .lib file, but will not have any data.

This variable must be used before **char_library**.

lic_max_timeout

<value>	Specifies the duration, in seconds, to wait for each license feature/token before skipping and checking with the next possible license feature/token. Default: `86400`

Note that this variable only works when the shell environment variable **ALTOS_QUEUE** is set to 1. See the Waiting for Available License section of Chapter 3, "Getting Started with Liberate MX" for more details about how this variable works.

The shell environment variable ALTOS_LIC_MAX_TIMEOUT will override the value set by this variable in the Tcl file. For more information, see ALTOS_LIC_MAX_TIMEOUT.

This variable must be used before **char_macro**.

lic_queue_timeout

<value>	Specifies the duration, in seconds, to wait for the required licenses to be acquired. Default: 60 (seconds)

The shell environment variable ALTOS_LIC_CHECK_ALT_TIMEOUT will override the value set by this variable in the Tcl file. For more information, see ALTOS_LIC_CHECK_ALT_TIMEOUT.

This variable must be used before **char_macro**.

mxtable_dontcare_value

{<bus_name | pin_name> < 0 | 1>}
Sets a default table entry for pin or bus. Default: none

This variable sets a default table entry value for a pin or bus. Variable entries are specified as a list of name-value pairs. The value specified for the given pin/bus will be used whenever there is no other value specified in the table. For example, if a pin is omitted from a table, or if there is a question-mark (?) entry in a table, the "dont_care" (default) value will be used. See also Appendix A, "Truth Table Format."

Example:

```
    set_var mxtable_dontcare_value {oe 0 ctrl 0xc}
```

mxvw_min_glitch_peak

< value>	The value range for this parameter is 0.0 to 1.0. Default: 0.1.

This variable is used in mx validation flow to identify a glitch in waveform comparison. It is used to define the glitch by specifying the ratio of vdd between 0 to100 percent. This variable is applied on glitch above gnd, glitch under vdd, undershoot under gnd, and overshoot over vdd.

mx_active_fanout_channel_include

< "none" | "!memcore" >	Determines how the probe node is loaded for modeling purposes. Default: "none".

Intended for backward compatibility only. This controls how the probe node is loaded for modeling purposes. Releases 3.1 and earlier employed a scheme that loaded the node in a recursive fashion, including the full active channel (except for memory core nodes.) This caused a degrade in performance for no appreciable gain in accuracy. This variable should be set to "none" to implement the more efficient modeling scheme of version 3.1p1.

none: Load the probe with the transistors directly driven, and model the channel connected terminals with equivalent caps. (Default)

!memcore: Set this to implement the probe loading scheme of release 3.1 and earlier. Caution: may cause a degrade in performance – use for backward compatibility only.

Example:
```
    set_var mx_active_fanout_channel "!memory"
```

mx_active_load

"value"	Controls loading on a per-net basis. Valid values listed below. Default: "none"

Control loading on a per-net basis. Note: this is really only useful to achieve correlation on internal margin measurements and should not be used for regular library characterization. Valid values:

all	Load any node with active devices
clock	Load clocks with active devices
none	Load any (non-probe) node with equivalent passive loads (Default.)
wordline	Load wordlines with active devices
net_name	Load specified net name with active devices

Example:
```
    # Set active load on clocks, wordlines, and signal "sig1"
    set_var mx_active_load "clock wordline sig1"
```

mx_active_load_thresh

<value>	Threshold for determining if the active load on a node can be replaced with a passive load. Default: 1.0 (farads)

MX simplifies a partition by substituting a passive load for active devices. If the passive load required is larger than the specified threshold, then don't perform this substitution. Default is 1.0 farads (a very large number.)

Example:

```
    set_var mx_active_load_thresh "1e-15"
```

mx_arc_report

<filename>	Specifies file where failed arcs are reported. Default: "mx_dir/arc.rpt"

Arcs that are found during partitioning but fail to check against user-defined arcs are reported to this file. Also reported are arcs found by partitioning but fail during characterization.

mx_auto_char_params

< 0 | 1 >	Tells MX to pass commands and variables specified in the main script directly to characterization phase. Default: 0

0: Don't pass through any commands or variables. (Default)

1: Pass-through commands and variables.

Commands passed through are:

- define_template
- define_leafcell

All variables are passed through except:

- variables specific to MX (variables that begin with "mx_")
- variables needed to be set to a specific value for the flow to work, such as extsim_deck_include (0), extsim_use_node_name (0)

mx_autoprobing_hold_level

<number>	Specifies where hold constraint probing will be inserted based on the intersection between a pin and related pin. Default: 0 (i.e. first latch/combinational; i.e. master)

Allows for probing to be intersection-level based: the choice of a specific logic region as a valid candidate for probing is based on the number of times pin and related pin intersect on any path propagating from the inputs to that logic. In this case, "logic region" means a channel connected region (i.e. NAND gate) or a strongly coupled component (i.e. latch, domino-stage, flip flop, memory array). See schematic below for an explanation of "Level 0" and" Level 1". (Default: 0) Example:

```
    set_var mx_autoprobing_hold_level 0;
```

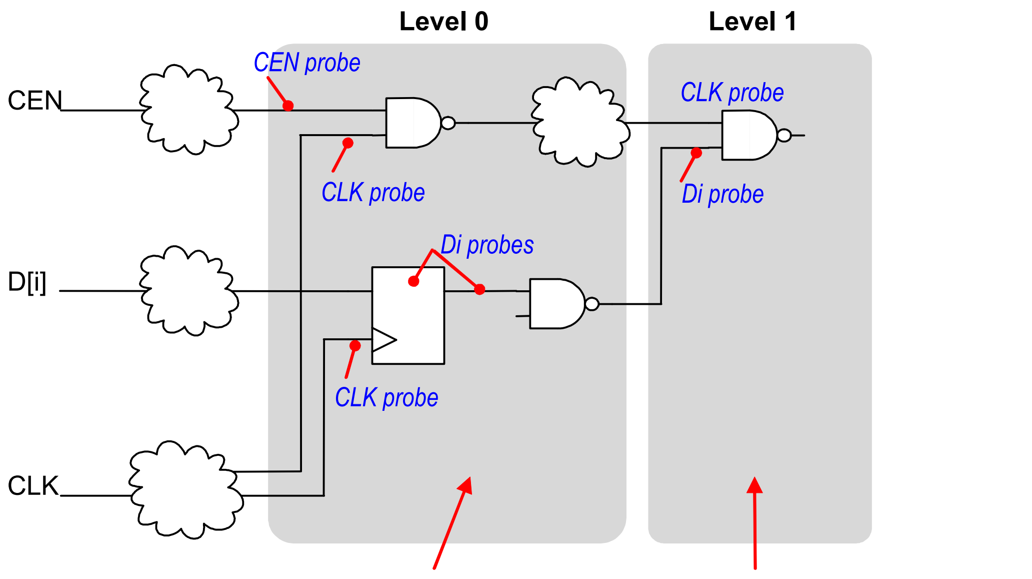
mx_autoprobing_setup_level

<number>	Specifies where setup constraint probing will be inserted based on the intersection between a pin and related pin. Default: 1 (i.e. second latch/combinational; i.e. slave)

Similar to mx_autoprobing_hold_level, except applies to setup constraint (see above). Default: 1

Note: for flip-flop based designs, this default will be **0**.

```
    set_var mx_autoprobing_setup_level 1;
```



Hold constraint probes placed here for mx_autoprobing_**hold**_level = **0**

Setup constraint probes placed here for mx_autoprobing_**setup**_level = **1**

mx_bisection

< 1 I 0 >	Specifies whether MX should use bisection to do characterization. Default: 0 (Do not perform bisection, use default path-delay method)

During the partitioning and automatic probing phase, worst case constraints arcs are identified using path-delay differences method. As for any other arc in the MX flow, the resulting worst case arcs are partitioned and characterized in full SPICE. The full SPICE characterization phase uses either bisection or path-delay depending on the value of the mx_bisection parameter.

If bisection is specified, the probes actually being measured may differ from the ones used when the path-delay method is used – usually probes are pushed at the outputs of slave stages (as opposed to inputs of slave stages when path-delay is used). The user has the ability to control automatic probing when in bisection mode by using the **-bis_probe** option in the corresponding define_arc command.

1: MX will use bisection to perform characterization.
0: MX will use the path-delay method to perform characterization. (Default)

Example:
```
    define_arc \
        -bis_probe myNode \
        -bis_probe_dir R \
        -pin data –pin_dir F \
        -related_pin clk -related_pin_dir R \
        ...
```

Note: The 3.2 version of MX can only accept single switching on a pin in tables to generate correct bisection. If the table has multiple switching vectors as inputs, then MX characterization will fail to perform bisection on that arc.

mx_char_bundle_size

<number>	Causes the characterization to group partitions and execute as separate runs. Default: 0 (Don't group partitions into separate runs.)

This variable instructs Liberate MX to split the characterization portion into separate runs in order to avoid excessive memory usage during read_spice (often caused by huge RC trees present in partitions.) Accordingly, this feature is useful for heavily extracted netlists, particularly those with extracted power rails or extracted virtual rails.

Note that regardless of this variable, 3 distinct characterization scripts are always generated:

- ❑ All **timing** (delay, constraint, measure) partitions are grouped and run at once (through script timing.tcl)
- ❑ All **pincap/noise** partitions are grouped and run at once (through script pincap.tcl)
- ❑ **Power** characterization is run separately

If `mx_char_bundle_size` is set to a number greater than zero (N > 0), then N partitions (or less) will be grouped as a set, and MX will be called sequentially on each set.

Example:

`mx_char_bundle_size = 20`, and a timing characterization contains 201 timing partitions, 10 pincap, and 1 power, the following scripts will be automatically generated:
```
    <cell>.timing.0.tcl# characterizes partitions 0 through 19
    <cell>.timing.1.tcl# characterizes partitions 20 through 29
    ...
    <cell>.timing.9.tcl# characterizes partitions 180 through 199
    <cell>.timing.10.tcl# characterizes partition 200

    <cell>.pincap.0.tcl
    <cell>.power.0.tcl
```

Notes:

- ❑ This bundling is independent from any LSF/queuing specified for the characterization run (queuing will happen independently from this bundling.)
- ❑ This is not related to the bundle or parallel bundle flow (which offers no advantage for runtime/memory in MX.)

mx_char_virtual_as_rail

<string>	Controls the behavior of virtual rails in dynamic partition and characterization. Default: all

This variable controls the behavior of virtual rails in dynamic partition and characterization. This variable works together with the Liberate-MX set_vdd command.

Example:
```
set_vdd -virtual VDD1 0.9
set_var mx_char_virtual_as_rail VDD1
```

mx_check_arcs

< 0 I 1 >	Check arcs found during partitioning against user defined arcs and report them to file. Default: 0

Set this to **1** to report arcs specified by the user that will not be present in the final library. This can occur because arcs are not found by partitioning (could be an issue in user-specified vector, automatic probing, or dynamic partitioning), or because of a failure during characterization (issue in deck, or simulation option.)

The variable mx_arc_report specifies the file where failed arcs are reported. (Default: mx_dir/arc.rpt). See also mx_check_arcs_exit_on_missing.

Example:
```
# Report user-specified arcs that will not be in final library
set_var mx_check_arcs 1
# File where arcs are reported
set_var mx_arc_report [pwd]/arc.info
# Exit if one or more user-specified arcs will not generate a partition
set_var mx_check_arcs_exit_on_missing 1
```

mx_check_arcs_exit_on_missing

< 0 I 1 >	Terminate execution if there’s a mismatch between arcs specified by the user and arcs found by partitioning. Default: 0 (don’t terminate execution)

Set this to **1** to force MX to terminate execution if there's a mismatch between user-specified arcs and arcs found by partitioning.

mx_clock2clock_constraints

< 0 | 1 >	Allows for CLK to CLK constraint characterization. Default: 0

mx_clock_tree_report

< 0 | 1 >	Requests output of a clock tree report. Default: 0

0: Does not generate a clock tree report. (Default)
1: Generates a clock tree report, `clock_tree.rpt,` in the Liberate MX directory. This report lists the active nets in the clock tree path for each table.

Example:

```
TABLE: /xxx/xxx/xxx/constraint.tbl
clk
->clkb
-->clk_buf_
---->clk_gate
```

This variable must be set before **char_macro**.

mx_clone_if_uda

< 0 | 1 >	Allows a worst-case arc to be cloned, only if a user-defined arc for the cloned one is available. Default: 1

This variable allows a worst-case arc to be cloned, only if a user-defined arc for the cloned one is available. Example:

```
    # Turn off cloning
    set_var mx_clone_if_uda 0
```

mx_const_prop

< 0 | 1 >	Performs/Skips constant propagation at the top level. Default: 0

mx_constraint_ocv_factor

<value>	Factor by which to correct constraint measurements. Default: 0

For HOLD, maximum measured clock path delay is increased (multiplied) and minimum measured data path delay is decreased (divided) by this factor. For SETUP, maximum measured data path delay is increased (multiplied) and minimum measured clock path delay is decreased (divided) by this factor. The default is 0, i.e. NO OCV correction.

Example:
```
# set 3% variation
set_var mx_constraint_ocv_factor 0.03
```

mx_corecell

<identifier>	Where identifier is one of the following: *single_port*, *dual_port*, *rom*. Default: *single_port*

The **mx_corecell** variable offers a way of specifying the type of core cell in a more concise way than with the **-mxcore** option to **define_cell**. It accepts as value the most common configuration of SRAM cells – single or dual port, i.e. 6T or 8T – or just the keyword *rom*. When *rom* is specified, the tool doesn't look for an SRAM structure at all. Example:
```
# look for a dual_port sram 8T core cell …
set_var mx_corecell "dual_port"
# Command has the same effect of mxcore example command
```

mx_create_if_dynamic

< 0 | 1 >	Adjusts internal MX algorithms to move dynamic filtering of statically-found arcs to occur early in the process. Default: 0 (Off)

Used when the number of statically found arcs becomes very large, which may cause excessive memory usage. This can occur when **mx_setup_comb** or **mx_hold_comb** are set to "**all**". Symptoms may be that the process size grows too large, or the program stalls at this point:
```
(MX-info) - Auto-probing ...
```

If this occurs, please terminate the process, set this variable, and start again.

mx_create_if_uda

< 0 | 1 >	Prevents MX from generating extra arcs from simulation results. Default: 0

If set to 1, MX will not generate extra-arcs from simulation results if those arcs are not defined in **template.tcl**. Default: 0, which means MX will generate extra arcs even if they are not defined.

mx_debug

< clock | memory | pattern | power | sim | arc>

clock	Generates information on clock propagation.
memory	Generates information on the system memory footprint (memory consumed during the run).
pattern	Generates information on mxtable expansion results.
power	Generates information about power characterization.
sim	Generates information on FastSPICE simulation decks and commands being generated during the run.
arc	Generates information on arc probing.

Example:

```
set_var mx_debug arc :
```

It generates files `debug_arc.csv` and `debug_arc.log`, which contain all valid candidates considered to get worst case for a given arc.

- **debug_arc.csv**: This file contains a comma seperated list of all arcs. It can be directly opened in Excel.
- **debug_arc.log**: This file contains various debug information.

Each line gives detail about a valid path pair for a given arc (setup/hold).

For any given arc, MX starts with the default value `-1.0`, and the moment a valid value is found, the database gets updated with the new value. You can see the updated value in the column `New` of the xls file (debug_arc.csv).

Delay/Retain: For delay and retain arcs, values mapped to `t1` are for delay measurement and to `t2` are for transitions.

```
DELAY = t1_max

TRANS = t2_max

RETAIN = t1_min
```

```
RETAIN_SLEW = t2_min
```

Constraints: MX performs a total of four measurements in order to find setup/hold and these are mapped to `t1_min/t1_max/t2_min/t2_max`. MX takes measurements on both lower and upper thresholds, for example, 30-70 or 20-80. Smaller delays are mapped to min and larger ones are mapped to max. Hence final setup/hold is calculated as given below.

```
SETUP = t1_max-t2_min
HOLD = t2_max-t1_min
```

You can specify any combination of the values listed.

mx_delay_ocv_factor

<value>	Factor by which to correct delay measurements. Default: 0

Example:

```
    # set 3% variation
    set_var mx_delay_ocv_factor 0.03
```

mx_dir

<string>	Specifies the directory where MX temporary files should be stored. Default: ./mx

Example:

```
    # write mx files in to specified dir
    set_var mx_dir /home/user/test_macro/mx
```

mx_distributed_sim

<"options">	Passes switches and options to an external program. Default "" (none).

This variable passes switches and options to external programs such as a simulator or job management system. This allows the user to run a program with the same options they use standalone. Examples:

```
    # Pass info to job queue
    set_var mx_distributed_sim "bsub -q <queue_name> -o <log> -I spectre"
```

mx_domain_propagation

<"static" | "dynamic" | "static dynamic">	Controls behavior of domain propagation. Default: "static dynamic"

Set to "static": Use static information to propagate the domain through the circuit, using these heuristics:
Combinational logic (mix of clock and data domain among its inputs)
- clock will propagate to the output
- data will stop

Sequential logic (mix of clock and data domain among its inputs)
- clock will stop
- data will propagate through

Set to "dynamic": Use dynamic information coming from FastSPICE simulation to propagate the domain through the circuit; i.e. a domain on the input of a gate will propagate to the output if they both switch in at least one common simulation interval.

Set to "static dynamic": Use a combination of static and dynamic information to propagate the domain through the circuit. Heuristics used in the static propagation are augmented or corrected with dynamic information available from the FastSPICE simulation. (Default.)

A domain force or stop on a domain node will overwrite the results of either method.

mx_dpartition_inactive_tie

< all | inter | ? >	Specifies how to transfer the steady state region of the circuit from the top-level fastsim run to the partition-level fullspice run for a specific vector. Default: `inter`

If set to **all**, all inactive notes are connected.

mx_dynamic_include_full_core

< 0 | 1 >	Instructs the tool to include the full core cell in dynamic partitioning independently from activity. Default: 1

This command should be used in conjunction with **mx_monitor_memcore 1**. In that case, activity would be available for memory cores, just like any other node, so only the active portion of the memory core would be included in the partition when traversed in. For example, a memory access traversal.

Using this command alone forces the whole core to be included, which is normally needed in the measurement flow to improve accuracy for measurements that involve core nodes as probes.

mx_failed_char_report

<string>	Specifies the file name to print detailed information about partitions that fail characterization – if any. Default: ./failed_char_arcs.rpt

mx_fastsim_auto_ic

< 0 | 1 >	Forces on/off the settings of initial condition on memory cores in the FastSPICE decks. Default: 1

mx_fastsim_clock_skew

<double>	Specifies what clock slew to use in FastSPICE simulation. When not specified, slew is chosen as the first entry in the clock slew table. Default: -1

mx_fastsim_input_slew

<double>	Specifies what input slew to use in FastSPICE simulation. When not specified, slew is chosen as the first entry in the input slew table. Default: -1

mx_fastsim_load

<double>	Specifies what load to use in FastSPICE simulation. When not specified, slew is chosen as the first entry of the input slew table. Default: -1

mx_fastsim_reuse

< 0 | 1 >	Uses FastSPICE simulation results from a previous run. Default: 1

Results of FastSPICE runs are stored by MX in directory *./mx_*fastsim under file name *<macro_name>_<table_file_name>.alwf*. If the sensitization (i.e. the mxtables) does not change from one run to the next, it is advisable to reuse previous run results by setting this flag to 1. Note that Liberate MX will automatically re-invoke the FastSPICE simulator for the expected result files that cannot be found. See also table-based option *fastsim_reuse*.

Example:

```
    # No table has changed from previous run - reuse
    # FastSPICE results for current run
    set_var mx_fastsim_reuse 1
```

mx_find_arrays

< 0 I 1 >	Identifies large channel-connected components as possible candidates for memory arrays. Default: 0

It is recommended that this be turned off for very small memories, if memory cores are not properly identified.

mx_find_memcore_numbit_threshold

<integer>	Sets a lower bound for the number of memory cores expected to be found in a channel connected or strongly-coupled region. Default: 1

Using the **mx_find_memcore_numbit_threshold** variable helps avoid identifying a regular latch as a memory array when the latch's structure is identical to that of a memory core cell, which could affect the automatic probing and, ultimately, the constraint characterization.

mx_find_memcores

< 0 I 1 >	Forces Liberate-MX to stop searching for memory cores recognition when set to zero. Default: 1

This variable allows forcing Liberate-MX to stop the searching of memory cores recognition when set to zero.

Example:

```
    #to stop memory core search.
    set_var mx_find_memcores 0
```

mx_find_stack_loads

< 0 | 1 >	Instructs the tool to identify active loads typically used on tracking lines. Default: 0

Because they usually come in large numbers, active loads can introduce a large number of errors if modeled as passive caps in partitioning. The **mx_find_stack_loads** command identifies them and forces partitioning to keep them as active loads rather than equivalent passive caps.

mx_find_virtual_rails

< 0 | 1 | 2>	Identifies nodes that should be treated as logic 1, 0 during static analysis. Default: 0

When set to 1, the command triggers reporting of power switched nodes that should be considered as logic constant during static analysis. Reported nodes should be then set as vdds, gnds by using the **set_vdd** or **set_gnd** commands with the **-virtual** option for subsequent runs. Use this flag when static partitioning step takes an unusually long time, that is, more than 10 minutes on a 10M xtrs block. The usage is as follows:

1. Set the variable and run to have a first set of virtual rail candidates reported
2. Modify the tcl script to include **set_vdd/set_gnd –virtual** commands on the reported nodes and rerun, checking whether static partitioning runtimes are as expected
3. Repeat as needed. Usually one iteration is sufficient.

Example:
```
# Tool seems to be hanging in static partitioning:
# (MX-info) - Partitioning - static - start ...
#    (MX-info) - Partitioning  24/1693326 xtrs
#

# Stop execution and add to your tcl script:
set_var mx_find_virtual_rails 1
#

# Rerun; following will be reported:
| (MX-info) - Finding virtual rails ...   wall clock time +=    0 sec
  |  | (MX-info) - xtop/net112:1 - was no set as virtual rail. If the next
partitioning steps take longer than expected, check the node and add 'set_vdd
(set_gnd) -virtual xtop/net112:1 $vdd ($gnd)' to your script
#

# Stop execution and add to your tcl script:
set_vdd -virtual xtop/net112:1 0.99
#
# Rerun.
```

```
when set to "2":
Liberate_mx could auto recognize all the virtual rails, which drive pmos/nmos
number is larger than mx_virtual_rail_minimum_xtrs(default is 50) and match the
virtual rail topology. This value reports which wire is set as virtual rail,
and some other wires may be virtual rail [mx/virtual.rpt].

Example: Mx/virtual.rpt
*** wire_name                      drive_pmos      drive_nmos      virtual_vdd
N_XI0-Q_3_47_c_1031883_n         2204              0                   set
(has been set as virtual_vdd)

*** wire_name                     drive_pmos      drive_nmos       virtual_gnd
N_XI0-VSS_c_697803_n              0                   870                  set
(has been set as virtual_gnd)

*** wire_name                 drive_pmos     drive_nmos     other_unset_wires
N_XI0-DBL_c_1280305_n          10                       68                unset
(not set as virtual rail)
```

mx_fix_pin_vdd

< 0 I 1>	Turns on a fix to correct an issue where the automatically generate pin_vdd value may be wrong for an IO. Default: 0

When MX used in reuse mode and in a different corner than what FastSPICE simulation was run at, the pin_vdd value automatically generated for IO pins may be wrong and use the previous PVT value rather than the current one.
Set to 1 to implement this fix. (Recommended.)

mx_full_rail_tol

< tolerance >	Controls the tolerance that defines what is considered full-rail, as a percentage of VDD. Default: 0.05 (5%)

If Vmax and Vmin are maximum and minimum voltage levels reached by an output in a transition, the transition is considered to be full-rail if **mx_output_require_fullrail_switch** is set true, and (Vmax - Vmin) < mx_full_rail_tol * VDD. Example:

```
# Outputs can transition to 91% of VDD and still be considered switching
set_var mx_output_require_fullrail_switch 1
set_var mx_full_rail_tol 0.1
```

mx_fullsim_measurement

< 0 I 1 >	Generates dynamic partitions out of **define_measurement** commands. Default: 0

mx_greybox

< 0 I 1 >	Generate a library directly out of FastSPICE. Default: 0

Set this to generate a library directly out of FastSPICE without partitioning, or a full SPICE run. NOTE: This must be a standalone and separate run. This means that either a library will be generated directly from FastSPICE (mx_greybox=1) or the software will proceed with the normal flow. Default: 0

mx_greybox_constraint_method

< 0 I 1 >	Controls the number of runs to be done on a constraint table. Default: 1

Variable controls the number of runs to be done on a constraint table and sets it to max(i1,i2) where i1 and i2 are number of constrained and related slew indexes.

mx_inputcap_ldb_reuse

<0 I 1>	Reuse pin cap LDB from previous run. Default: 0

See mx_ldbs_reuse for description of all LDB reuse variables.

mx_ldbs_reuse

<0 I 1>	Reuse timing, power, pin-cap, noise, power and leakage LDBs from previous run. Default: 0

Liberate MX can reuse all or selective LDBs from the previous run by using the following variables:

mx_inputcap_ldb_reuse
mx_ldbs_reuse
mx_power_ldb_reuse
mx_noise_ldb_reuse
mx_timing_ldb_reuse

It will also re-characterize the arcs that failed characterization in previous run, if any.

mx_margin_report

{filename}	Generates a report by the **define_measure** commands. Default: “measure.rpt”

To generate this report, specify a filename. (You may specify this with a full path.) To omit this report, set this variable to “” (empty quotes.)

mx_mcf

<double>	Miller Cap Factor; multiplies each coupling cap in the netlist by the specified amount. Default: 1

mx_min_period_latch_component_mode

< 0 I 1 >	Specifies an enhanced min period latch component calculation. Default: 0 (off)

Set this to specify an enhanced minimum period latch component calculation for designs where the latch clock is gated by both an external and internal clock.

0: For designs where the latch clock is controlled only by internal clock (Default)
1: For designs where the latch clock is controlled by both an internal and external clock.

mx_min_period_mode

<internal_pulse I user_spec I latch I bitline_precharge >
Controls various minimum period components. Default: `internal_pulse`

Liberate MX supports all measurements contributing to minimum period for self-timed memories. This variable is used to control the components that should be considered for minimum period calculation. The valid values are:

internal_pulse: Internal pulse width in design is considered. (Default)
user_spec: Only user-defined measurements.
latch: Use latch component.
bitline_precharge: Use bitline precharge component.

Note: When using automatic flows (using the **define_memory** command), this variable is set to `latch`, `user_spec`, `bitline_precharge`, or `internal_pulse`.

You can use any combination of valid values for minimum period calculation. For example:

```
set_var mx_min_period_mode "latch bitline_precharge"
```

mx_monitor_memcore

<value>	Special debug flag. Default: "inter"

Valid values are:
none: No memory core node is monitored; can lead to slightly larger partition sizes in register files, but to significant reduction in FastSPICE run time and MX footprint for larger testcases.
inter: Only memory core nodes that cross CCC are monitored. (Default.)
intra: Only memory core nodes that do not cross CCC are monitored. (Should be used for debugging purposes only.)
all: All memory core nodes are monitored. (Should be used for debugging purposes only and on very small cases.)

mx_mpw_allow_same_probe_on_both_rise_and_fall_clock_tree

< 0 | 1 >	Allows a node to be associated with both rising and falling clock trees. Default: 1

Set to 0: Requires a node to ONLY switch with clock rising(falling) to be considered on the rise(fall) clock tree.
Set to 1: A node that switches with both clk:R and clk:F can belong to both trees. (Default)

mx_mpw_false_probe_delay_threshold

<value>	Filters out nodes that are internal return dignals. Default: 1e-9

Filters out nodes that are just internal return signals and do not contribute to the characterization of MPW. If the delay between primary input and probe is greater than mx_mpw_false_probe_delay_threshold, the probe is discarded and not used during MPW calculation.

mx_mpw_measurement _duration

<value>	Sets duration of the MPW measurement as a fraction of `mx_simulation_interval`. Default: 0.75 (assumes clock period of 1).

Sets the duration of the MPW measurement as a fraction of mx_simulation_interval. Usually the period is two-times the simulation interval (clk period == 2 * mx_simulation_interval) so a default of 0.75 ensures the duration of the measurement covers a rising and a falling edge.

mx_mpw_mode

< 0 | 1 >	Selects method for calculating MPW. Default: 1

0: Selects calculation method used with version 3.0p3 and earlier.
1: (Default) Selects method described below:

1. Clock trees are traced to identify the rising-edge and falling-edge clock trees. Tracing is done using both static and dynamic methods.
2. Identification of probe nodes as nodes that lie on the intersection between rising and falling clock tree.
3. Definition of MPW HIGH as setup between CLK:R nodes and CLK:F nodes, and MPW LOW as setup between CLK:F nodes and CLK:R nodes.

mx_mpw_probe

<"seq" | "seq comb array">	Specifies what type of logic should be probed. Default: "seq"

When looking for MPW probes, specifies what type of logic should be probed. For minimum and maximum clock tree probe selection, Liberate MX considers first level sequential logic, that is, hold latches.

mx_mpw_probe_lower_fall
mx_mpw_probe_lower_rise
mx_mpw_probe_upper_fall
mx_mpw_probe_upper_rise

<value>	Specifies the thresholds for MPW probe measurements.

Defaults are as follows:
mx_mpw_probe_**lower_fall**: 0.3
mx_mpw_probe_**lower_rise**: 0.3
mx_mpw_probe_**upper_fall**: 0.7
mx_mpw_probe_**upper_rise**: 0.7

mx_mxtable_interpret_read_write_cycle_keywords

< 0 I 1 >	Control interpretation of read & write cycle keywords. Default: 0

Use this variable to turn on and off (default OFF) the capability to interpret 'read' and 'write' cycle keywords in a table. These keywords were originally intended to allow for a more concise table functional description of a memory, and they should be used only for very simple cases. In general, it is recommended not to use these keywords, particularly for the more complex designs - and instead give a fully expanded description of the write and read operations.

mx_negedge_clock

<string>	Specifies the active edge(s) of a clock, otherwise Liberate MX assumes positive active edge(s). Default: none
-clock	Specifies the name of negative clock edge(s) clock names

This variable allows specifying negative active edge(s) of a clock otherwise, Liberate-MX assume positive active edge(s). Example:

```
# to specify clk1 & clk2 being negative active edge.
set_var mx_negedge_clock -clock clk1 clk2
```

mx_noise_ldb_reuse

<0 I 1>	Reuse noise LDB from previous run. Default: 0

See mx_ldbs_reuse for description of all LDB reuse variables.

mx_output_require_fullrail_switch

< 0 I 1 >	Requires an output transition from FastSPICE to be a fullrail in order to be considered as valid for partitioning. Default: 1

mx_partition_name_use_arc

< 0 I 1 >	Controls the naming of MX partitions. Default: 0

If the **mx_partition_name_use_arc** variable is set to 1, MX uses information on the arc the partition represents. Examples:

Constraint partition:

```
single_port_sram_ext_constraint_adr0_hold_f_553/
```

Delay partition:

```
single_port_sram_ext_delay_dout0_r_clk_r_558/
```

Without setting this variable, "`adr0_hold_f`" and "`dout0_r_clk_r`" would be missing from above partitions respectively.

mx_pathdelay_hold_clock_margin

<double>	The percent margin to add to hold on clock path. Default: 0

mx_pathdelay_hold_data_margin

<double>	The percent margin to add to hold on data path. Default: 0

mx_pathdelay_setup_clock_margin

<double>	The percent margin to add to setup on clock path. Default: 0

mx_pathdelay_setup_data_margin

<double>	The percent margin to add to setup on data path. Default: 0

mx_pincap_char

< 0 | 1 >	Performs/skips pin cap characterization. Default: 1

mx_posedge_clock

<string>	Specifies rising edge(s) clocks. Default: all

This variable allows specifying positive active edge clocks. This is useful when a deck has both positive and negative active edge(s).

Example:

```
    # to specify clk1 & clk2 being negative active edge and
    # clk3 and clk4 being positive edge.
    Set_var mx_negedge_clock clk1 clk2
    Set_var mx_posedge_clock clk3 clk4
```

mx_power_assign

all | clock | input | output <list of pins>	Specifies how to distribute internal power among switching pins. Default: all

Selecting **all** distributes internal power contribution equally among switching pins. Selecting ***<list of pins>,*** internal power contribution is distributed among listed pins when switching. If you use the default setting of **clock**, MX assigns all switching power to clock, independently from other inputs switching.

Selecting **input** distributes internal power contribution equally among switching input and clock pins. Selecting **output** calculates the output switching power and assigns it to the output. Selecting **all** is equivalent to input and output.

If **<list of pins>** is selected, internal power contribution is distributed among listed pins when switching. If you use the default setting of **clock**, MX assigns all switching power to clock, independently from other inputs switching.

mx_power_ldb_reuse

<0 | 1>	Reuse power and leakage LDB from previous run. Default: 0

See mx_ldbs_reuse for description of all LDB reuse variables.

mx_power_single_point

< 0 | 1 >	Allows power characterization to be performed for multiple slews/ loads. Default: 1

Allows power characterization to be performed for multiple input slews and output loads, based on the power template that is provided.

0: Characterize multiple points for slew/load table.

1: Populate slew/load table with a single point. (Default)

This variable must be set before **char_macro**.

mx_preprocess

< 0 l 1 >	Enable preprocess speedup. Default: 1

The Preprocess Flow prunes FastSim decks to provide initial conditions for better performance and capacity. It needs the RCDB flow enabled. *(Please see RCDB flow Application Note.)* Only UltraSim is supported as the partition simulator.

mx_probe_peak_currents

<0l1>	Default is `0`.

Setting this variable will trigger calculation for inrush and peak current as well as attribute settings in the final library.

mx_probes_report

<filename>	Specifies file(s) to report the results of automatic probing. User may specify a full path name.

MX produces the following two reports:
<filename>: Contains the results of all possible probes found statically.
<filename>.red: Less-verbose ("reduced") report filtered down based on switching activity.

mx_read_spice_exit_on_missing_file

<0 l 1>	Tells MX to issue an error and exit if there is a file missing during read_spice. Default: 0

0: Don’t exit if there is a missing file. (Default)

1: Issue an error and exit if there is a missing or unreadable file during the `read_spice` phase.

Example:

```
    set_var mx_read_spice_exit_on_missing_file 1
```

This parameter *must* be set before **read_spice**.

mx_remove_false_ic_group

< 0 | 1 >	Controls whether measurement results generated from false initial condition arcs will be included in the data base. For backward compatibility only. Default: 1 (Always remove false initial condition groups.)

Because dynamic information for memory cores is usually unknown, partitions that cover memory core nodes will have multiple define_arc commands to ensure that all possible initial conditions for the memory cores are characterized. The normal behavior is for one – and only one – of these arcs to generate successful measurements, and all others to fail.

However, it may happen that some of the bad initial condition arcs will generate results. In this case, proper results are identified as the one being the closest to the FastSPICE result; all others are removed from the database.

0: Allows (potentially) false initial condition groups. For backward compatibility only.
1: Always remove false initial condition groups. (Default)

An informational message is always printed to the standard output when results for false initial condition arcs are removed from the data base.

mx_remove_rc_pincap

{"all" | "none" | "rail" | net_name}

Controls removing parasitic RC networks on a per-net basis.
Default: "rail"

This controls removing parasitic networks for a partition. This may be applied on a per-net basis, for example supply rails, or specific nets. User may use the keywords "all" or "none" or supply a list of nets. Note: "all" or "none" will take precedence if used together with net names.

all: Remove parasitic RC networks for all nets.
rail: Remove parasitic RC networks for rails (vdd, vss)
none: Do not remove any parasitic RC networks. (Default)
net_name: Remove parasitic RC network for the specified net.

Example:

```
# Remove parasitic RC network for rails
set_var mx_remove_rc_pincap "rail"
```

mx_remove_rc_timing

{"all" | "none" | "rail" | net_name}	Controls removing parasitic RC networks on a per-net basis. Default: "none"

This controls removing parasitic networks for a partition. This may be applied on a per-net basis, for example supply rails, or specific nets. User may use the keywords "all" or "none" or supply a list of nets. Note: "all" or "none" will take precedence if used together with net names.

all: Remove parasitic RC networks for all nets.
rail: Remove parasitic RC networks for rails (vdd, vss)
none: Do not remove any parasitic RC networks. (Default)
net_name: Remove parasitic RC network for the specified net.

Example:
```
    # Remove parasitic RC network for specified nets
    set_var mx_remove_rc_timing {myNet123 myNet456}
```

mx_retaining_time

< 0 | 1 >	Generates – and characterizes – partitions for retaining_rise / retaining_fall arcs. Default: 1

Note that retaining (a.k.a. valid time) arcs are characterized only when user-defined.

Example:
```
    # Do not generate / characterize retaining time arcs
    set_var mx_retaining_time 0
```

mx_ring_model_fold

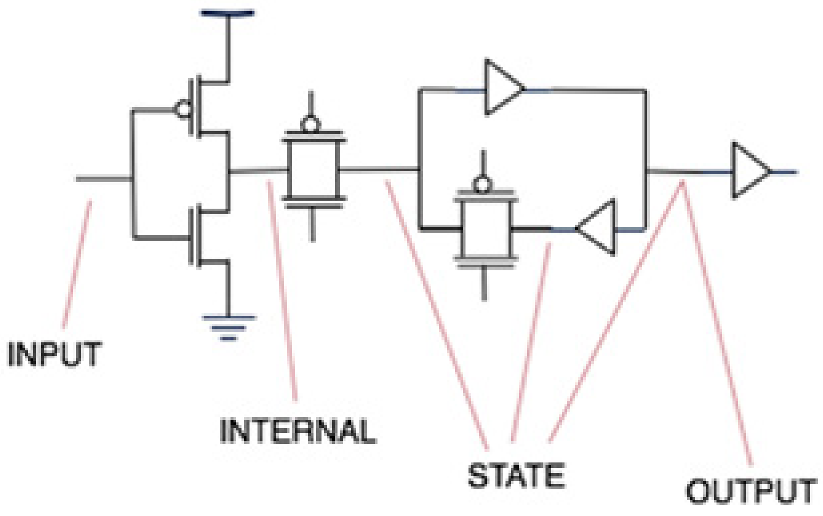
< 0 | 1 >	Turns on topological matching of logic. Default :1

This variable **mx_ring_model_fold** can be use to turn on topological matching of logic driven by primary inputs / driving primary outputs belonging to the same bus, to speed up pincap and CCSN characterization

mx_seq_probing

<string>	Sets the node type to consider on a sequential component as possible candidates for data path probes in setup/hold

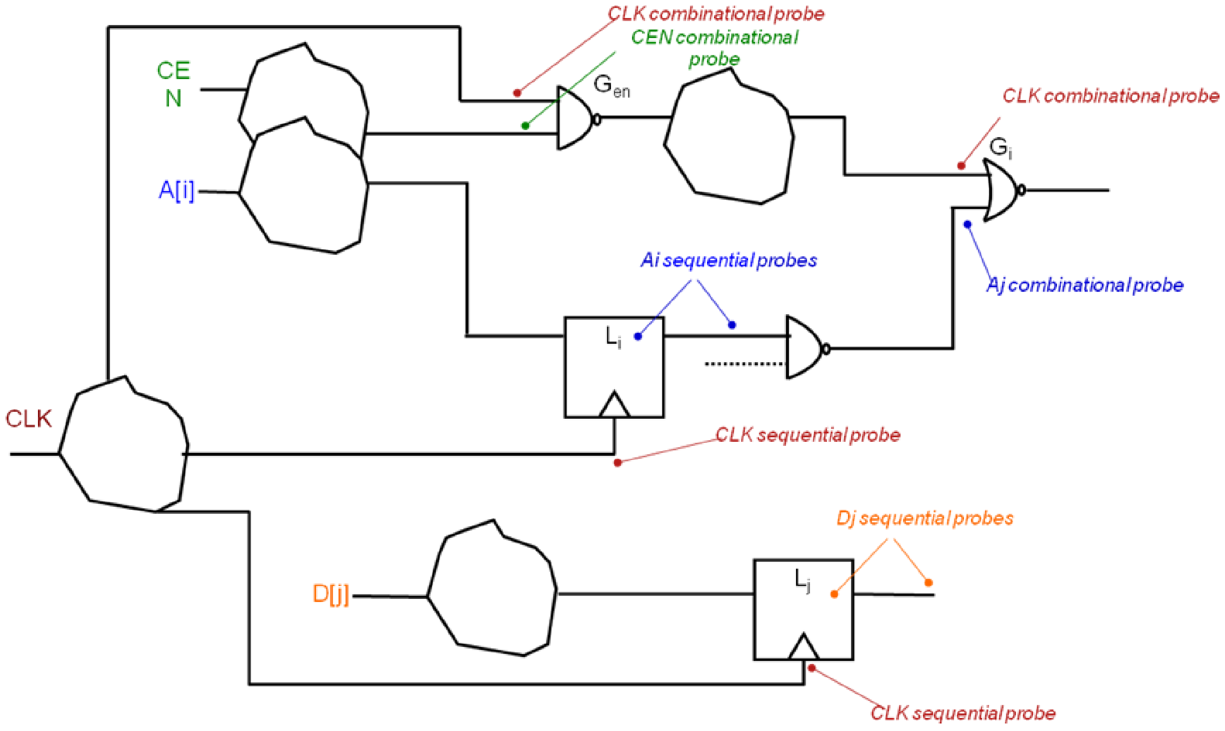
characterization.
Default: state output internal input.



mx_setup_seq
mx_hold_seq
mx_setup_comb
mx_hold_comb

{<identifier> }	Flags drive the automatic probe identification step in identifying candidates for setup and hold constraints probes.

When probing for constraint characterization, Liberate MX automatically probes the following structures (please refer to the figure below): clock gated combinational logic on non-clock inputs (G_{en}); first level latches (L_i, L_j); clock gated combinational logic on outputs of first level latches (G_i)



By default, Liberate MX uses first-level probes for hold characterization for all pins and both first-level and second-level probes for setup characterization for all pins. Internal and output nodes of sequential elements are used as non-clock probes and clock inputs to the sequential element are used as clock probes. This corresponds to the following values for the flags:

```
    set_var mx_setup_seq  "all"
    set_var mx_hold_seq   "all"
    set_var mx_setup_comb "all"
    set_var mx_hold_comb  "all"
```

It is often the case that probing for constraint characterization is a function of the input pin for which constraints are to be characterized. For this reason flags can take pin-dependent values. Referring to the picture above, the user would specify:

set_var **mx_setup_comb** *"CEN A[i]"*, to allow for combinational probes on CEN and A[i] pins (and those pins only) to be used for setup constraint calculation for that pin.

Moreover, you can specify that the non-clock inputs to sequential elements also be used as probes – using keyword "*:probe_inputs"* modifier after pin name. Referring to the picture above, you would specify:

set_var **mx_setup_seq** *"all Dj: probe_inputs"*, to use inputs to sequential gate L_j as probes for D_j, but leave the default for other pins (i.e. internal and output nodes only).

You can also specify whether sequential slave (i.e. second level) nodes should be considered when probing – usually for setup. Using set var mx_setup_seq and post-fixing he :slave keyword to the name allows for a pin or set of pins to be probed at second-level sequential elements when characterizing setup constraint.

In general, to allow for both first (master) and second (slave) level sequential probing for all inputs, you would specify the syntax as follows:

```
    set_var mx_setup_seq "all all:slave"
```

This example uses first-level sequential for all pins' setup characterization, except for pin "wtz", which uses second-level sequential as well:

```
    set_var mx_setup_seq "all wtz:slave"
```

mx_simulation_interval

<value>	Value, in seconds, of the simulation interval used by the FastSPICE step. Default: 10ns

Set the value of this parameter to at least twice the value of the maximum delay that needs to be measured during the run. This is a global attribute and is applied to all tables and arcs.

Note: there are two other methods of specifying the simulation interval:
Table-based, using simulation_interval (See Specifying Input Stimuli.)
Arc-based: using an -attribute to define_arc.

If there are multiple definitions of the simulation interval, the precedence order is as follows (1st in list means highest precedence)

1. Arc-based(highest)
2. Table-based
3. Global - i.e. set_var(lowest)

Example:

```
    # if max clk->out delay is expected to be around 5.2 ns …
    set_var mx_simulation_interval 12e-9
```

mx_skip_autoprobing

< identifier | "array">	Skips autoprobing for channel connected regions. Default: none

This command forces MX to avoid automatic probing on specific channel connected regions (CCC). The command accepts either a string describing the type of logic to avoid (for example: my_cell_25) or the keyword "array". Example:

```
set_var mx_skip_autoprobing "array"
```

mx_skip_print

<regular expression>	Used to reduce FastSPICE runtime by filtering out nodes that should not get monitored. Default: none.

Variable accepts one or more regular expression to filter nodes that should not get monitored in the FastSPICE simulation - this is to reduce FastSPICE runtime / disk space / MX footprint when reading results back. Regular expression are in TCL style (see regexp TCL command documentation).

Issuing the command multiple times causes concatenation of patterns rather than overwrite. Command must be issued before char_macro. Node names / patterns should refer to pre-layout names. Example:

```
# No activity needed (and therefore skipping them in FastSPICE) on
# nodes that match pattern below:

set_var mx_skip_print \
"XLPCAM/XLPCAM/XMEMARRAY_(.*)/XCG64MEMARRAY(.*)/XCG64ROW(.*)/XSRCH_B/NET200"
```

mx_spv_api

**< 0	1 >**	Turns on/off generation of API scripts and data to be used in conjunction with the SpiceVision GUI from Concept Eng (third-party product). Default: 0

mx_timing_ldb_reuse

**<0	1>**	Reuse timing LDB from previous run. Default: 0

See mx_ldbs_reuse for description of all LDB reuse variables.

mx_timing_report

**<0	1>**	Controls generation of detailed timing reports. Default: 0

When set to `1`, Liberate MX generates the following two detailed timing reports:

- `sim.top.rpt` that reports the data from fastsim.
- `sim.rpt` that reports the data from characterization runs.

mx_timing_report_debug

<0l1>	Generates timing reports with extra columns. Default: 0

When set to `1`, Liberate MX generates the `sim.top.rpt` and `sim.rpt` timing reports with two extra columns, `Incr` and `Tran`.

Incr: From startpoint to current net incremental time (50% ~ 50%).
Tran: Current net transition time (30% - 70%).

For example:

```
PointPath PulseWidthIncrTran
clk 0.02500 r 0 0.02000
MZY/I0/cgr/I2/Mn1_G 0.06577 f 0.04077 0.01894
MZY/I0/cgr/I1/MI6_G 0.06714 f 0.00137 0.02042
MZY/I0/cgr/I4/Mn1_G 0.13050 r 0.06336 0.04266
```

mx_verbose

< 0 l 1 >	Prints basic flow and runtime information. Default: 1

Example:

```
    #report basic flow info
    set_var mx_verbose 1
```

mx_virtual_rail_auto_mode

<0 l 1 l 2>	Automatically sets the virtual power supplies in a design. Default: 2

This variable enables you to find (mx_find_virtual_rails) and define virtual power nodes in a design. The valid values are:

0: There are no virtual rail findings. It is equivalent to `mx_find_virtual_rail` set to `0`.
1: The `virtual.rpt` report is created. However, there is no automatic setting. User can refer to the report and set the virtuals accordingly.
2: Automatic settings along with the `virtual.rpt` report. (Default)

This variable works when `mx_find_virtual_rails` is set to `2`.

mx_virtual_rail_opposite_device_minimum_factor

Sets a threshold value for the count of maximum opposite devices. The net below the maximum opposite device counts is not considered as a virtual net. Default: 1000

For example:

wire_name drive_pmosdrive_nmosother_unset_wires

VDD_1130388unset

VSS_25081038unset

`VDD_1` is set as a virtual vdd because of connections to pmos. Note that eight opposite nmos are connected to the same net. Similarly, for `VSS_2` where 508 pmos are connected, which is below the default value and because of 1038 nmos connections, `VSS_2` is set as a virtual net.

mx_virtual_rail_minimum_xtrs

<integer>	Sets a lower limit for the number of wire drive MOS. Default: 50

The wire that has more drive MOS than `mx_virtual_rail_minimun_xtrs` is a virtual rail.

Example:

```
# Defining minimum number of mos to "70"
set_var mx_virtual_rail_minimum_xtrs 70
```

mx_whitebox_active_coupling_threshold

double	Sets the threshold coupling between victim/aggressor, which will be considered in dynamic partitioning. Default: -1

For example, if total couple between aggressor A and victim V is > threshold, aggressor A will be transversed during dynamic transversal of victim V.

mx_whitebox_active_wire_threshold

double	Sets the threshold for a wire to be considered active. Default: 0.01 volts

mx_whitebox_model_file

< 0 I 1 >	Forces the model file used in both partitioning and characterization to point to the specified file. Default: 0
{filename}	Specifies the full path name for the file.

mx_whitebox_monitor_memcores

Deprecated. Use mx_monitor_memcore instead.

mx_zip_partition_deck

< 0 I 1 >	Controls generation of partition decks in gzip format. Default: 0

When set to `1`, it generates partition decks in gzip format. You can use this variable when characterizing large extracted netlists.

This variable must be used before **char_macro** command.

A

Truth Table Format

Specifying Input Stimuli

In Liberate MX, you can specify input stimuli by passing one or more truth table files to the `define_table` command.

define_table command

The `define_table` command specifies a list of truth table files. The number of specified table files determines the number of FastSPICE runs that will be performed in parallel—use the `-thread` option of the char_macro command to control the number of threads.

Following is the file syntax for the `define_table` command:

```
arctypes <arc_type>
<fastsim_option>
# <comment_string>
table <table_id>
pins <bus_id> <bus_id> … <bus_id>
    <cycle_id> <value> <value> … <value>
endtable
```

Where:

`<arc_type>`	Vectors specified in the table file will be used for *<arc_type>* characterization. It accepts one or more of the following values: `delay`, `retain`, `setup`, `hold`, `minperiod`, mpw, `power`, `leakage`, measure, `timing`. **Note:** Timing equals delay retain setup hold. If *<arc_type>* is not specified, the default is `timing` measure.

`<fastsim_option>`	It accepts one the following values: `fastsim`, `fastsim_reuse`, `fastsim_deck_include`, `fastsim_deck`, `simulation_interval`, `fastsim_auto_ic`, `fastsim_model_include`, `fastsim_cmd`, `fastsim_cmd_option`.

fastsim

`<fastsim_indentifier>`	One of the following: `spectre`, `aps`, `finesim`, `hspice`, `xa`

Specifies which FastSPICE engine will be used to simulate the table file. Overwrites the output of the `char_ams -charsim` command.

This example specifies Spectre as the FastSPICE simulator:

```
fastsim spectre
fastsim_deck .option rawfmt=fsdb
```

fastsim_auto_ic

`<value>`	Either `1` or 0 (May also be specified as `true`, `on`, `false`, `off`)

Instructs Liberate MX to add or not add `.ic` statements to bit-lines and core nodes, as recognized in the static topology recognition step. It overwrites the global variable `mx_fastsim_auto_ic` for the table file it is specified in. If not specified, the default value is given by the value of `mx_fastsim_auto_ic`, which has a default of `true`.

The `simulation_interval` command overwrites `mx_simulation_interval` for the table it is specified in. This can be used in a table of any type: power, leakage, timing, measure, etc.

Syntax:

```
simulation_interval <time>
<time> == <value><?unit>
<value> == a number (in any notation)
<?unit> == one of f,p,n,u,m,fs,ps,ns,us,ms
```

Note: The following are equivalent:

```
simulation_interval 20e-9
simulation_interval 20n
simulation_interval 20ns
```

Example:

```
simulation_interval 20e-9
```

The command above if specified in the `timing.tbl` table ensures that vectors listed in that table will use a simulation interval of 20ns rather than the default 10ns (which will be used for any other table that does not have a `simulation_interval` command).

fastsim_cmd

`<path_to_executable>`	
	Specifies the path to an executable to be used for simulating this table file.

fastsim_cmd_option

`<command_options>`	Specifies command line options to be passed to the executable used for simulating this table file.

Example:

```
fastsim_cmd /home/tools/mmsimcm_v4/lnx86/latest/bin/spectre
fastsim_cmd_option +spice -64
```

Note: When Spectre XPS is used, the output format defaults to SST2 during partitioning. This overwrites any value coming from `fastsim_deck` options in tables ***or*** the `extsim_cmd_options` command in scripts.

Re-using fastsim results:

```
fastsim_reuse
```

When present, instructs Liberate MX to look for previous fastsim results on this table file. fastsim_reuse results are stored for each run inside the directory:

```
./mx_fastsim - with name <cellname>.<tablename>.alwf
```

This overwrites `mx_fastsim_reuse`.

fastsim_deck

<fastsim_deck_string>

A valid text (for the engine specified in fastsim) that will be included (as is) to the deck being simulated in FastSPICE.

This variable is used to specify truth table files FastSPICE simulator options in Liberate MX. This optimizes how options are passed to FastSPICE and unifies it into a single "`fastsim_deck`" line. You can then specify anything that needs to go to the fastsim deck, meaning there will be no interpretation of the option/command as previously done. Example:

```
# Using timing table to pass FastSPICE simulator specific options
arctypes leakage
fastsim finesim
fastsim_deck .param simpreset=5
fastsim_deck .param pn_level=5
fastsim_deck .param cgnd=1e-15
fastsim_deck .param sfe_compaction=0
fastsim_deck .param keepparaname=0
fastsim_deck .param rshort=2
fastsim_deck .param hier_delimiter=.
fastsim_deck .param dc_turbo=3
fastsim_deck .param rcr_fmax=1G

table...
  ...
endtable
```

fastsim_deck_include

<netlist_path_name>

Full path name of a SPICE netlist

Specifies what netlist to run fastsim on for this table, when different than the main netlist read into the database through the `read_spice` command. It is usually used to run power characterization on a different netlist than the one used for timing. It assumes:

- ❑ The netlist specified has the exact same .subckt interface as the main one and, if used for other than the power table, all nodes contained in this netlist must be contained in the main one as well.

- That `extsim_model_include` is specified, otherwise the option will be ignored.

The name of the file must be a full path, else the option will be ignored.

fastsim_model_include

<model_path_name>	Specify the full path name of a SPICE model file for power or leakage characterizations. Default: none

This specifies a full path to a model file to be used with fastsim for power or leakage characterizations.

Liberate MX will only use this model file if it is different from the file read into the database through the `read_spice` command. The name of the file must be a full path or the option will be ignored. Example:

```
fastsim_model_include "/home/users/models/XYZ/power.mod"

table...
...
endtable
```

Guidelines for Writing Truth Table for Multiport Memories

Following are some of the guidelines for writing truth table for multiport memories.

- When there are more output buses and need measures/arcs terminating to both outputs by “assign values”

  For example, there are 2 output buses `output_0` and `output_1` and you would like to measure delay/retain on both buses.

  # only on one of the output bus

  ```
  table measure_output0_only
  pins        input   ouput0
  write0      R       X
  write1      R       X
  read0       R       X
  read1       R       D
  read0       R       D
  endtable
  ```

  # both output buses in same cycles

```
table measure_both_output0_and_output1
pins        input     ouput0    output1
write0      R           X           X
write1      R           X           X
read0       R           X           X
read1       R           D           D
read0       R           D           D
endtable
```

- For `clk hidden power` arc ensure only one of the output switches at a time. This is mostly required to be taken care with multiport memories where multiple outputs can switch due to different read and write operations on ports simultaneously.
- It is recommended to start with one of the port and try to get most of the arcs (Delay/ power) correctly. Once you are sure that the results are fine, you can just replicate the same table by changing bus names for other ports.

Required Keywords

The table, pins, and endtable entries are required keywords.

```
<table_id> : <string>
<cycle_id> : <string>
<bus_id> : <string>
<value_string> : R F ? 0 1 B C D - X H L A P N onehot onecold
```

The `table_id` is an arbitrary string which is used to identify the table. This ID must be unique for each table.

The `cycle_id` is any string and is used as a placeholder to position the pin data. If the `cycle_id` is set to `minperiod` or `output_period`, special functionality is enabled.

The `bus_id` is the name used in the define_cell command to refer to a circuit port.

Valid characters that can be used in the `value_string` are:

Inputs

Value	Description
R F	Rising (Falling) edge; expands the table to generate a 0->1 (1->0) sequence while modifying the other values on the same line. Specify 1 (one) edge per row.
1	One, bus-size independent; expands in to 00...00->11...11 when on the same line with an edge; to …1111 otherwise.
0	Zero, bus-size independent; expands in to 11...11->00...0 when on the same line with an edge; to …0000 otherwise.
B	Binary; expands in to …0000->…1111->…1111->…0000 when on the same line with an edge; to …0000->…1111 otherwise.
I	Inverted binary; expands in to …1111->…0000->…0000->…1111 when on the same line with an edge; Opposite of B.
A	a, bus-size independent; expands in to …0101->…1010 when on the same line with an edge; to …1010 otherwise.
L	Low, bus-size independent; tied to 00...00 independently from any expansion it's involved in.
H	High, bus-size independent; tied to 11...11 independently from any expansion it's involved in.
?	Don't care; expanded to ??...?? and tied to 00...00 independently from any expansion it's involved in. Does not affect other measurement options.
X	Overwrite any other output letter on the line
5	Five, bus-size independent; expands in to …1010->…0101 when on the same line with an edge; to …0101 otherwise.
P N	Positive (negative) pulse; expands the table to generate a 0->1->0 (1->0->1) sequence while keeping the other values on the same line untouched.
onehot onecold	Expands the table to generate a 00…01->00…10->...->01…00->10…00 (11…10->11…01->…->10…11->01…11) sequence while keeping the other values on the same line untouched.

Outputs

Value	Measurement
B	All *(constraints, delay, measure, and power)*
C *or* -	Constraints *(setup & hold)*
D *or* **delay**	Delay
H *or* **hold**	Hold constraint

J *or* **power**	Switching & hidden power	
M	"define_measure" + mpw + minperiod + min & max clock tree paths	
These offer further granularity over "M"	mpw	Minimum pulse width (mpw) only.
	min_period *or* **minperiod**	Minimum period only.
	min_clock_tree_path	Minimum clock tree path only.
	max_clock_tree_path	Maximum clock tree path only.
S *or* **setup**	Setup constraint	
W *or* **leakage**	Leakage power	
X	- nothing - (Do not measure constraints, delay, or power)	
Z	Tri-state arcs	

In case the output is a bus, it is possible to instruct MX to measure only specific bits, using hex notation as bit-mask:

```
### only look at bit 0 of output 0…
pinsCLK AWT TME ME  WE  WEM D   ADR OE  Q
readA   0   0   1   0   0   ?   b   1   0x1
```

Values are automatically expanded to the proper size of the bus they are specified for.

Note: The first line in the table must not perform any measurements (in other words, it acts like a dummy line.) This allows the memory cell to achieve a stable condition in simulation before measurements are taken. In the code example below, measurement on the Q pin is set to "don't care" in the first line of the table.

Default Values

Pins or busses may be given a default value, which can simplify entering data into a table. Default values may be specified with the Tcl variable, mxtable_dontcare_value, or within a table file. Within a table file, use the keyword `dontcare_value`.

Within a table file, `dontcare_value` can be specified globally for all tables, or on a table-by-table basis. To specify for all tables, place within the file after the “arctypes” statement.

Example:

```
arctypes delay enable disable
dontcare_value din 1
```

To specify on a table-by-table basis, place within a table immediately following the "pins" identifier inside the table. The `dontcare_value` will be used if there is no explicit value assigned, or wherever the table has a question mark (?) for a value.

Example:

```
table seldesel
pins           clk    cs      bw      din    adr    dout
dontcare_value oe 1
# oe=1 will be used throughout this table
deselect        R      1      L      ?      ?      X
select          R      0      L      ?      ?      X
deselect        R      1      L      ?      ?      X
endtable
```

Precedence of `dontcare_value` directives is:

```
(table level) > (table file level) > tcl command level > default (=0)
```

Three-State Enable or Disable Arcs

Three state characterization is performed in MX if both following conditions are met:

- ❑ Three state enable and disable lines exist in `define_table`
- ❑ User-defined arc of type enable/disable are available (`-type enable` or `-type disable`)

A table line is considered an enable line for a specific output when the line contains the identifier "enable" or "E" or "e" in the column corresponding to that output. A table line is considered a disable line for a specific output when the line contains the identifier "disable or "Z" or "z" in the column corresponding to that output.

Note: The table file containing enable and disable lines must also contain `arctypes` identifier to cover `enable` and `disable` arcs.

Example:

```
arctypes delay enable disable
table en_dis
pins    oe clk bw adr dout
disable F  ?  H  ?  Z
set_1   L  R  H  H  X
read_1  R  H  H  H  E
disable F  ?  H  ?  Z
set_0   L  R  H  L  X
```

```
read_0  R  H  H  L  E
endtable
```

Truth Table Example

```
# Specify a full table for a sram with bist and asynch. In particular, table
# delay describes basic write/read operations controlled by rising edge of CLK:
# write D = 11111111111 in to address ADR = 111111111
# write D = 00000000000 in to address ADR = 000000000
# read from address ADR = 000000000 and measure delay
# read from address ADR = 111111111 and measure delay
# write D = 00000000000 in to address ADR = 111111111
# write D = 11111111111 in to address ADR = 000000000
# read from address ADR = 000000000
# read from address ADR = 111111111

arctypes timing power
fastsim spectre
fastsim_cmd_option -64 +spice +xps=s3 +cktpreset=sram_pwr -format fsdb


table delay
pins    BISTE   ME      WE      OE      CLK     WEM     ADR     D       Q
write11 l      h       h       l       R       h       1       1       x  #Dummy
write00 l       h       h       l       R       h       0       0       -
read0_  l       h       l       h       R       l       0       ?       b
read1_  l       h       l       h       R       l       1       ?       b
write10 l       h       h       l       R       h       1       0       -
write01 l       h       h       l       R       h       0       1       -
read0_  l       h       l       h       R       l       0       ?       b
read1_  l       h       l       h       R       l       1       ?       b
endtable

table delay_bist
pins    BISTE   TME     TWE     TOE     CLK     TWEM    TADR    TD      Q
write11 h       h       h       l       R       h       1       1       -
write00 h       h       h       l       R       h       0       0       -
read0_  h       h       l       h       R       l       0       ?       b
read1_  h       h       l       h       R       l       1       ?       b
write10 h       h       h       l       R       h       1       0       -
write01 h       h       h       l       R       h       0       1       -
```

```
read0_  h       h       l       h       R       l       0       ?       b
read1_  h       h       l       h       R       l       1       ?       b
endtable

table constraint
pins    CLK     ADR     D       WEM     WE      OE      ME      AWT    BISTE     Q
const   R       b       l       l       l       l       h       l       l        -
const   R       l       b       l       l       l       h       l       l        -
const   R       l       l       b       b       l       h       l       l        -
const   R       l       l       l       l       l       b       l       l        -
const   R       l       l       l       l       l       h       l       0        -
endtable

table constraint_bist
pins    CLK     TADR    TD      TWEM    TWE     TOE     TME     AWT    BISTE     Q
const   R       b       l       l       l       l       h       l       h        -
const   R       l       b       l       l       l       h       l       h        -
const   R       l       l       b       b       l       h       l       h        -
const   R       l       l       l       l       l       b       l       h        -
const   R       l       l       l       l       l       h       l       1        -
endtable

table asynch
pins    BISTE   AWT     D       WEM     TD      TWEM    OE      TOE     Q
asynch  l       R       b       b       ?       ?       h       l       b
asynch  h       R       ?       ?       b       b       l       h       b
endtable
```

B

Specifying Memory Core Cells

In Liberate MX, memory core nodes need to be identified during automatic probing step. By default, the tool looks for a 6T SRAM cell. The default can be overwritten by specifying one or more core cell description files. Each file - typically just one per macro - contains the description of a core cell. Each cell is defined by one storage group and one or more port groups. In turn, each group is a collection of devices - transistors (for SRAMs and DRAMs) and capacitors (for DRAMs). Refer to the syntax and examples that follow.

Note: This step only effects automatic probing – i.e. understanding of what's a memory core node, what is a bit line pair, etc. - and not partitioning, which is based on switching information only.

Syntax:

```
mxcore <core_name> {
    <group_id> {
        <subgroup_id> {
            device {
                model:<model_id>
                side:<side_id>
                ctrl:<ctrl_id>
                use:<use_id>
                sizel:<l_id>
                sizew:<w_id>
                sizec:<c_id>
    }}}}
```

where:

<core_name>	<any string> Identifies the core cell.
<group_id>	storage, port Identifies the group type.

<subgroup_id>	sram, dram, rom, write, read, writeread Identifies the subgroup type
<model_id>	n, p, c Identifies the device type
<side_id>	1, 0 Identifies the "side" of the core cell the device is in. This is used whenever there are back-to-back drivers forming the storage group of the cell and is useful to refer to a right and a left side of the cell. Note that choice of 0, 1 is fully arbitrary - but must be consistent among different devices. See examples below.
<ctrl_id>	word, core Identifies device's gate terminal connectivity: controlled by wordline, controlled by mem core node.
<use_id>	pass, pull-up, pull-down, storage Identifies how the device is used; as a pass transistor, a pull up one, a pull down one, a storage.
<l_id>	double Specifies L for device - when <model_id> is n or p
<w_id>	double Specifies W for device - when <model_id> is n or p
<c_id>	double Specifies C for device - when <model_id> is c

Example:

8T SRAM - dual port – see Figure 2 – 8T core cell2 below

```
mxcore 8T_SRAM  {
    storage {
        sram {
            device {
                model:nmos
                side:1
                ctrl:core
                use:storage
            }
            device {
```

```
            model:pmos
            side:1
            ctrl:core
            use:storage
        }
        device {
            model:nmos
            side:0
            ctrl:core
            use:storage
        }
        device {
            model:pmos
            side:0
            ctrl:core
            use:storage
        }
    }
}
port {
    writeread {
        device {
            model:nmos
            side:1
            ctrl:wordbit
            use:pass
        }
        device {
            model:nmos
            side:0
            ctrl:wordbit
            use:pass
        }
    }
}
port {
    writeread {
        device {
            model:nmos
            side:1
            ctrl:wordbit
            use:pass
        }
        device {
            model:nmos
            side:0
            ctrl:wordbit
            use:pass
        }}}}
```

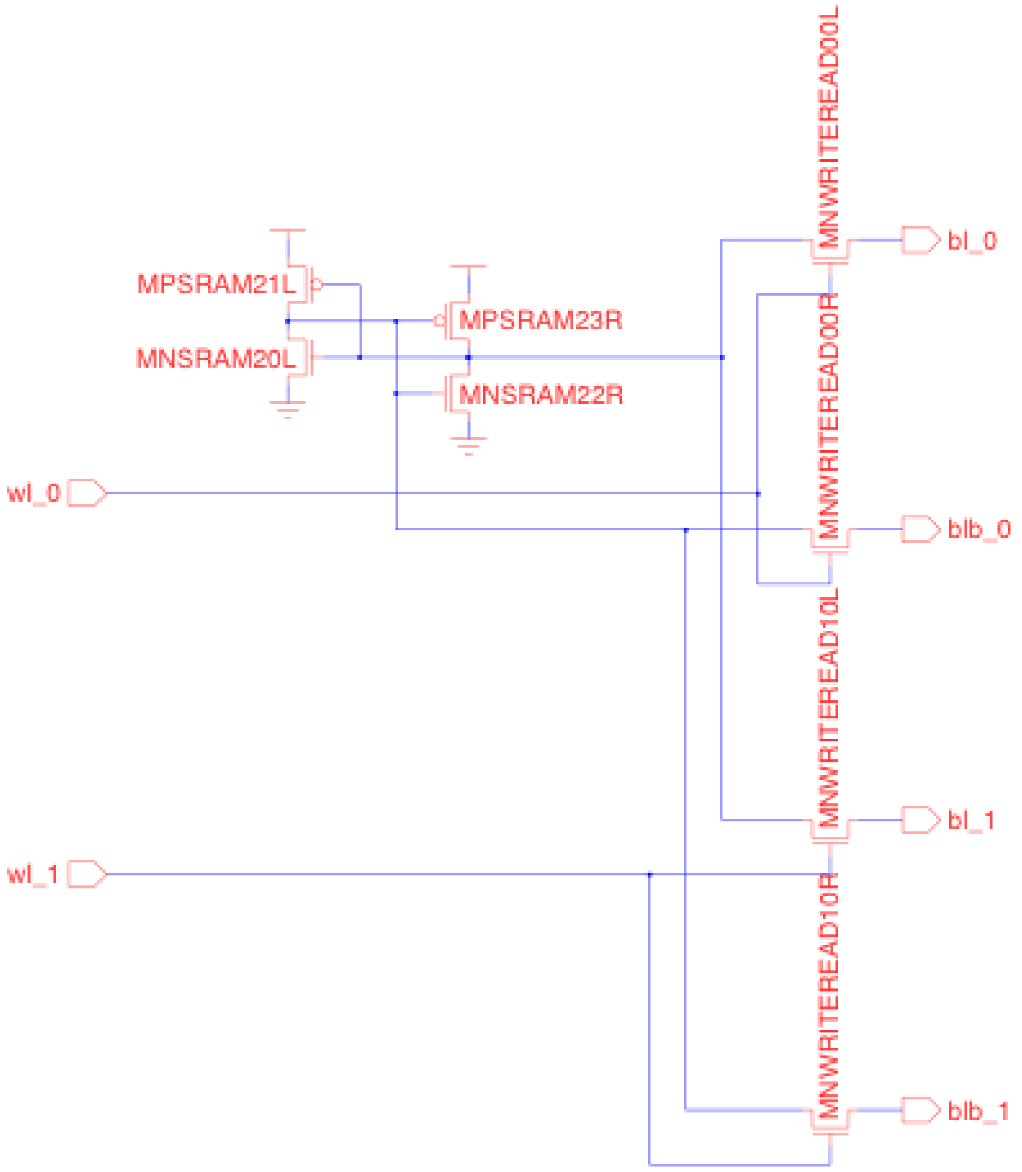


Figure 2 – 8T core cell

Example:

```
10T SRAM - dual port - pass write, pull-down read - see Figure 3 – 10T core
cell3.

mxcore 10T  {
    storage {
        sram {
            device {
                model:nmos
                side:1
                ctrl:core
                use:storage
            }
            device {
                model:pmos
                side:1
                ctrl:core
                use:storage
            }
            device {
                model:nmos
                side:0
                ctrl:core
                use:storage
            }
            device {
                model:pmos
                side:0
                ctrl:core
                use:storage
            }
        }
    }
    port {
        write {
            device {
                model:nmos
                side:1
                ctrl:word
                use:pass
            }
            device {
                model:nmos
                side:0
                ctrl:word
                use:pass
            }
        }
    }
    port {
        read {
            device {
                model:nmos
                side:1
                ctrl:core
                use:pulldown
            }
            device {
                model:nmos
                side:1
                ctrl:word
                use:pass
            }
            device {
                model:nmos
                side:0
                ctrl:core
                use:pulldown
            }
            device {
                model:nmos
                side:0
                ctrl:word
                use:pass
            }}}}
```

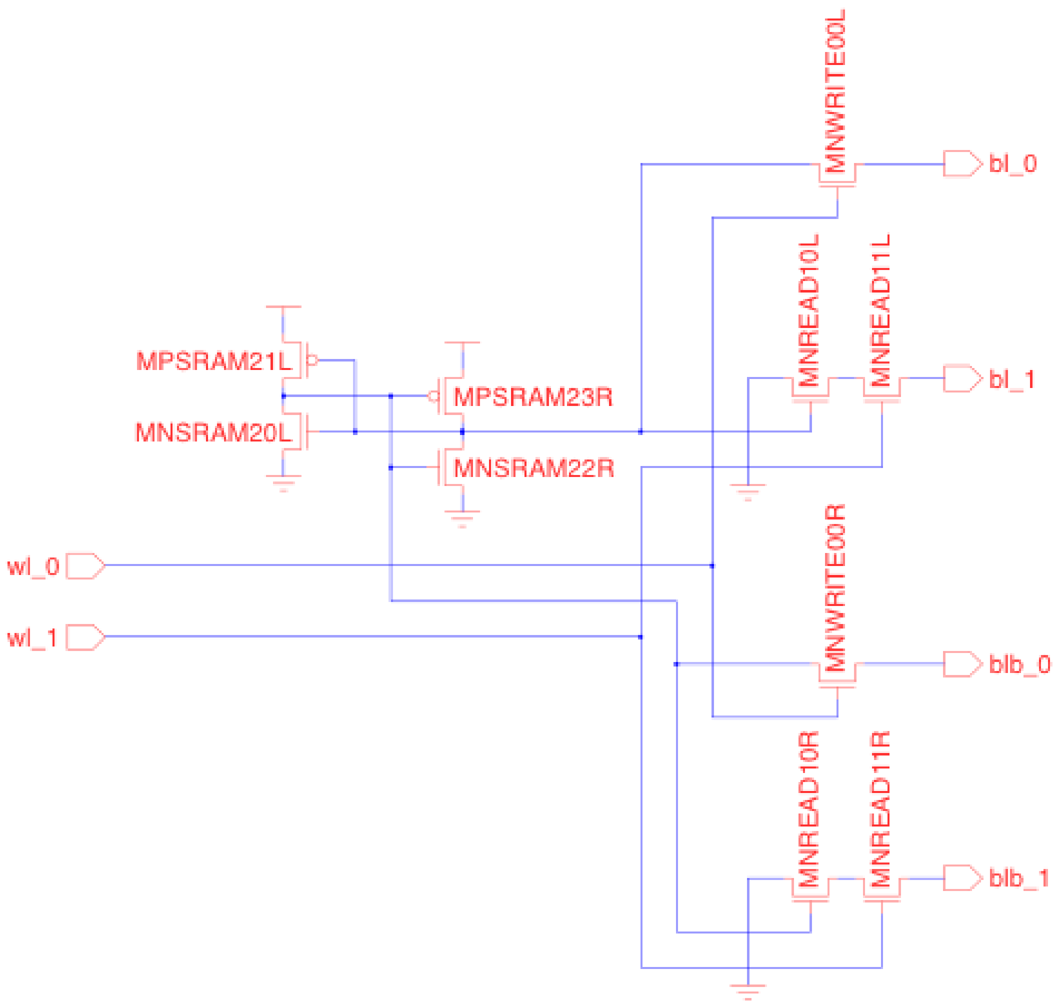


Figure 3 – 10T core cell

C

Using Liberate MX Automatic Flow

This appendix discusses the commands and their options that enable you to run the Liberate MX automatic flow (also known as the define_memory flow). For information about the criteria that an instance should satisfy to use the automatic flow, refer to the Automatic Characterization Flow section in Chapter 4, “Liberate MX Flows.”

This appendix also covers information about Debugging Common Issues with Automatic Flow.

Defining Liberate MX Settings for Automatic Flow

The Liberate MX automatic flow requires the following files:

- Tcl file containing the `define_memory` and char_memory commands
- Instance netlist
- SPICE or Spectre model files
- Reference library file or template file (*optional*)
- Tcl file containing any additional needed settings (*optional*)
- Additional table files (*optional*)

Primary Tcl File

The primary Tcl file of the automatic flow should contain the `define_memory` and `char_memory` commands. The sections below explain the various options of these commands that you can use to provide the Liberate MX settings and run characterization.

define_memory

The define_memory command is used to fully describe the run to be performed. The main information covered includes netlist and models location and format, PVT, pin name, functionality and active level, simulator control options and threading, and high-level settings to control the algorithm (activate the graybox mode, enable the rcdb flow, and so on).

Arc Definition

The arc definitions can be passed in the form of the reference library using the `-ref_lib` option. In this case, Liberate MX characterizes the arcs exactly as defined in the reference library file.

If it is necessary to modify the arc definitions, it is best to use the `read_library` and `write_template` commands to create the template and modify it (take care to generate the template with Liberate MX). The template can then be included using the `-template` option.

If there is no template or reference lib available, Liberate MX will create arcs and `when` conditions based on the pins described in `define_memory`.

`-ref_lib`	Reference Liberty file, containing all arcs and conditions to be used during the characterization.
`-template`	Template file, written in Tcl, containing all arcs, cell description and measurement conditions.

In case of re-characterization, you might want to modify the lists of slews and characterize loads without editing the template file. This is enabled using the following options:

`-slews`	List of slews to be characterized. Overrides template or reference lib definition.
`-loads`	List of loads to be characterized. Overrides template of reference lib definition.
`-qloads`	List of loads to be characterized for the data out pin. Overrides template, reference lib or `-loads` definition.

Netlist and Models

Once arcs and look up tables are defined, you need to point to instance's netlist and models to be used for the different measurements, using the netlist and model arguments below:

`-netlist`	The instance netlist file.
`-netlist_format`	The format of the netlist file. Default is `spice`.
`-models`	The model include file.

`-models_leakage`	Specific models to be used for leakage simulations.
`-models_power`	Specific models to be used for dynamic power simulations.
`-model_format`	The format of the models. Default is `spice`.
`-skip_read_model`	Controls whether the models will be read in with read_spice. Default is not to read in the models.

The `-foundry` argument can be used to automatically set the define_leafcell commands. The available foundries are TSMC and SMIC. All other foundries will need to `define_leafcell` in the `mx_settings` file (described later in this section).

Corner Definition

The temperature will always need to be defined using the `-temp` option.

The instance voltage supplies can be defined as follows:

`-rail`	A paired value list of voltage supplies and values.
`-global_voltage`	Defines the nominal supply value when the supply names do not need to be defined.
`-virtual_rails`	The names of the virtual rails of the instance followed by their normal operation value. Supplements the virtual rails that are automatically located.

Port and Physical Implementation of an Instance

To help the tool to properly recognize memory point and efficiently partition the design, the following options can be used to describe the physical implementation of the instance:

`-words`	Describes the number of address locations in the memory. The default of this will be 2^(number of address bits). It is essential to define the number of words if the number of words is less than 2^(number of address bits) to ensure that the vectors will not try to access an address that does not exist.
`-bits`	The number of bits in the instance. This is usually not needed if the reference lib or template is provided.
`-column_mux`	The number of physical columns in the array corresponding to a single data bit.

`-banks`	The number of separate banks in the instance.
`-number_of_ports`	The number of ports in the instance.
`-bitcell`	The type of bitcell used in the instance. Default is a 6t single port cell. Available values are rom, single_port, dual_port, 2prf and 10t.

In the automated standard instance flow, the instance pins need to be defined with their functions. These functions will then be used to create the vectors to be used for the characterization. The following options are used to define the basename of their respective pins:

`-clock`	`-address`	`-data_in`	`-data_out`
`-scan_in`	`-scan_out`	`-read_timer`	`-write_timer`

Some other pins will need to be defined along with their active state:

`-chip_enable`	The base name of the chip enable pin along with the state for chip is enabled.
`-write_enable`	The base name of the write enable pin along with the state for chip is in write mode.
`-bit_mask`	The base name of the bit mask or bit write bus along with the state for bit is in masked mode.
`-output_enable`	The base name of the output enable pin along with the state for output is enabled.
`-sleep`	The base name of the sleep pin and its state for the instance is in sleep mode.
`-shutdown`	The base name of the shutdown pin and the state for the memory is in shutdown mode.
`-bypass_enable`	The base name of the bypass mode enable pin and the state for the instance is in bypass mode.
`-read_timer_enable`	The base name of the pin to select between and externally specified read timer setting and a default internal value.
`-test_enable`	The signal that would control the input MUXes and select between regular mode inputs and test mode inputs.

There are other options that will control the prefix or suffix to be appended to the base signal names to determine the final names. This is particularly useful in the case of dual port devices or BIST control. The base name of the input signals is used with the defined prefixes and suffixes to map the pins from the subcircuit in the netlist file to their function.

`-clock_bist_prefix`	Prefix applied to the test version of the clock.
`-clock_bist_suffix`	Suffix applied to the test version of the clock.
`-clock_port_suffix`	Suffix applied to the clock signals to denote port.
`-bist_prefix`	Prefix applied to the test version of input pins.
`-bist_suffix`	Suffix applied to the test version of input pins.
`-port_suffix`	Suffix applied to the input signals to denote port.
`-bypass_suffix`	Suffix applied to a base input name to get the corresponding bypass mode output name.

You might also need to describe other pins used to control the device. In that case, the `-other_input_pins` option allows you to add them. Related non-standard arcs will have to be described in the template files or reference library, and you might have to provide additional tables (see later) to have arcs fully exercised during the fastsim simulations.

`-other_input_pins`	List of input pins of other functions.

Skipped or Additional Tables

You have the ability to override the tables that were created by the Liberate MX define_memory command. This allows the `define_memory` command to be used for instances that do not fit the exact definition of a standard instance.

`-additional_tables`	A list of tables to be added to the characterization.
`-remove_tables`	A list of tables created by define_memory that are not to be used for the characterization. The allowed values are: `leakage`, `delay`, `constraint`, `power`, `measure`, `bist`, `sleep`, and `bist_pwr`.

Simulator Options

The memory is now properly defined. Next step is to set simulator options and threading. Options below allow you to control speed of the characterization simulations:

`-part_spice`	The simulator to be used during the full instance portion of the Liberate MX run. The default is Spectre XPS.
`-char_spice`	The simulator to be used for the partition characterization portion of the Liberate MX run. The default is Spectre APS.
`-xps_smode_power`	Disables the use of s-mode for dynamic power simulations when using Spectre XPS. The default is 0, use s-mode for power.
`-fastsim_cmd_option`	Options to be used for the full instance simulation.
`-extsim_cmd_option`	Options to be used for the partition characterization simulations.
`-thread`	Number of threads to be used for the Liberate MX run. Default is 0, use all available.
`-part_thread`	Number of parallel jobs to be run during the full instance simulation portion. Overrides `-thread`.
`-char_thread`	Number of parallel jobs to be run during the partition characterization portion. Overrides `-thread`.
`-part_waveform_format`	
	Specifies the waveform format to be used for the full instance simulations. Default is `fsdb` format. This option can also be set to `sst2`.

Probing, Method Control and Additional Setup

Depending on your memory implementation, you might need to control where constraint probing will be inserted, based on the intersection between a pin and related pin. Probing and internal thresholds can also be controlled with the following options:

`-internal_threshold`	Lower and upper thresholds to be used for internal probing, in percent. Default is 20 80.
`-latch_probing`	Controls the nodes within a sequential element to be considered for probing. Default is input, state, internal, output.
`-autoprobing_hold_level`	

	List of pins and values for autoprobing hold level. Default is level 0.
`-autoprobing_setup_level`	
	List of pins and values for autoprobing setup value. Default is level 1.

The Liberate MX settings file contains the additional Liberate MX commands and options to be used for the characterization. This is used to override the internal Liberate MX settings that are defined, and instead define specific options, like define_leafcell commands or debug options.

`-mx_setting`	Name of the user-provided Tcl file containing the Liberate MX settings and commands.

The last step is to define master characterization methods, and specify the name of your memory, with the following:

`-greybox`	Enables the greybox (non-partitioning) flow. Default is 0.
`-bisection`	Enables bisection characterization for constraints. Default is 0.
`-rcdb`	Enables the RCDB flow. In this flow, the Liberate MX database size is reduced by leveraging information in the dspf netlist. This should only be used for large instances in dspf format. Default is 0.
`user_memory_name`	Specify the name of your memory.

char_memory

Once the define_memory command is completed, launch the characterization using the char_memory command. This command accepts very few options that control where the run is to be performed and which format has to be generated.

The options of the `char_memory` command are:

`-work_dir`	Specify the work directory name. It will be created as a subdirectory of the current directory if it does not already exist.
`-ecsm`	Enables ECSM characterization and lib generation.
`-ecsmn`	Enables ECSMN characterization and lib generation.

`-ccs`	Enables CCS characterization and library generation.
`-ccsn`	Enables CCSN characterization and library generation.

Additional Table Files

It might be necessary to provide additional tables to the Liberate MX `define_memory` flow. This could be needed if the tables created by Liberate MX are either incorrect or not sufficient to complete the characterization.

To learn more about tables, see the Guidelines for Writing Truth Table for Multiport Memories section.

Example

As an example, look at the following instance with `bist` inputs:

Pin Name	Function	Pin Name	Function
`CLOCK`	Clock pin for all modes	`BIST`	Bist control selects between input and test input - active high
`CE`	Chip Enable - active high	`TCE`	Test Chip Enable
`WE`	Write Enable - active high	`TWE`	Test Write Enable
`A[6:0]`	Address	`TA[6:0]`	Test Address
`D[31:0]`	Data	`TD[31:0]`	Test Data In
`Q[31:0]`	Data Out		

This would be the corresponding define_memory command:

```
define_memory \
    -netlist ./Netlist/rf_top_bist.cir \
    -models ./Models/include_models.scs \
    -model_format spectre \
    -global_voltage 1 \
    -temp 25 \
    -template ./Templates/template.tcl \
    -mx_setting ./Tcl/mx_settings.tcl \
    -bits 32 \
    -words 128 \
    -clock {CLOCK} \
    -address {A} \
    -data_in {D} \
    -data_out {Q} \
    -write_enable {WE H} \
    -chip_enable {CE H} \
    -test_enable {BIST H} \
    -bist_prefix {T} \
    -rail {  VDD 1.0 \
             VSS 0.0 } \
    rf_top_bist

char_memory
```

Defining the `bist` prefix as `T` instructs Liberate MX to prepend a `T` to the input pin names to derive the test input name. For example, a `T` is prepended to the chip enable name, `"CE"` to get the test chip enable name of `"TCE"`.

Debugging Common Issues with Automatic Flow

The Liberate MX automatic flow is intended as a fully automated solution for characterization of standard and vendor sourced memory instances. There are some cases, such as incorrect setup or a non-conforming instance, that can require some user debugging to understand what went wrong.

Pin Name Recognition Issues

The most common sources of debug in the `define_memory` flow are pin name recognition issues. These can result in incorrect stimulus vectors and missing arcs.

You are provided with a summary of the final pin mapping as `mx_setup`/`pin_file`:

```
pin_name            function            base_name
A                   address             A
BIST                bist                BIST
CE                  chip_enable         CE
CLOCK               clock               CLOCK
D                   data_in             D
Q                   data_out            Q
TA                  address             A
TCE                 chip_enable         CE
TD                  data_in             D
TWE                 write_enable        WE
WE                  write_enable        WE
```

If there is a mistake in the pin names defined, there will be question marks inserted into the `pin_file` report:

```
pin_name            function            base_name
```

```
A                    address              A
BIST                 bist                 BIST
CE                   chip_enable          CE
CLOCK                clock                CLOCK
D                    ?                    ?
Q                    data_out             Q
TA                   address              A
TCE                  chip_enable          CE
TD                   ?                    ?
TWE                  write_enable         WE
WE                   write_enable         WE
```

The report has all of the names from the instance netlist subcircuit in the first column. In this example, the user should investigate the name defined for the pins `D` and `TD`.

Missing Arcs

After running the `define_memory` flow, the user may find that some arcs are missing from the lib file. Some possible sources of these missing arcs include:

- Missing arc definitions
- Simulation issues
- Incorrect vectors

As in all Liberate MX flows, in order to characterize and arc, it is necessary to have an arc definition as well as a proper stimulus. If an arc is missing, the user should verify that both the stimulus and the definition are present. This can be corrected by the user by using the `-template` flow and modifying the template file as needed to contain the missing arc. Missing arc definitions in the template file can affect the stimulus created by the `define_memory` flow.

You should delete all reuse results if the template is modified.

If the arc is properly defined, but the arc is still missing from the lib file, the user should review the waveforms from the top level simulations to ensure that the activity is correct. If it is found to be incorrect, the user should review the `define_memory` command used to make sure that all side pins are properly defined.

If the stimulus cannot be corrected by modifying the `define_memory` command. You might need to create a new table file and include it using the `-additional_tables` command.

D

Liberate MX Timing Validation Flow

The Liberate MX timing validation flow performs validation of the timing arcs of the memory instance by running at speed simulation. The memory is simulated with user provided vectors with the minimum required setup, hold, and clock period found in the lib file to ensure that functionality is maintained.

The output of the MX timing validation flow is the two simulation waveform files. One simulation uses a deck with relaxed timing without any stressed waveforms. The second simulation uses a deck with stressed timing with realignments of signal waveforms against clocks to reflect the minimum timing that is provided. You can check the waveform differences between these two simulations to determine if the functionality and marginality of the instance is preserved under the stress conditions. For example, if the customer provides a table called `validation.tbl`, then two tables, `validation_relax.tbl` and `validation_stress.tbl` are generated. These two tables (vectors) are run and the results are stored.

The procedure for this is as given below.

1. Prepare a reference template file called `ref_template.tcl`:

   Input: A library with setup or hold time constraints of signals relative to clock.

   Output: `ref_template.tcl` file with `define_arc -validation_values {<values>}`.

   Command flow:
   ```
   read_library to_be_verified.lib
   write_template -mx_validate ref_template.tcl
   ```

2. Prepare a Liberate MX control file called `mxv.tcl` using the `ref_template.tcl` from step 1 above, as shown below:
   ```
   source ref_template.tcl
   validate_macro -validsim "xps" -thread 2
   ```

3. Check the results.

   The output from `validate_macro` will have waveforms. The waveform data is stored in two directories under `mx_reuse/mx_fastsim/`. If a table called `validation.tbl` was provided, two tables `validation_relax.tbl` and `validation_stree.tbl` are generated. The two simulation results are stored in:

```
mx_reuse/mx_fastsim/SRAM512x8_validation_relax_0_/transient1.tran.trn
mx_reuse/mx_fastsim/SRAM512x8_validation_stress_1_/transient1.tran.trn
```

4. About the simulation points:

The default data point chosen for fastsim simulation will be the minimum number of the delay/constraint indexes. If users want to change the simulation slew/load, they can use:

```
set_var mx_fastsim_input_slew  <input_slew_wanted>
set_var mx_fastsim_clock_slew  <clock_slew_wanted>
set_var mx_fastsim_load        <load_wanted>
```

E

Basic Flow for Validating User-Defined Criteria

To verify user-defined criteria by using the Liberate MX validation flow, perform the following steps:

1. define_measure <options>

   In this step, specify a measurement based on the requirement.

2. validate_measure <options>

   In this step, provide measure name to be validated from `define_measure`.

3. validate_macro <options>

   In this step, Liberate MX runs the validate feature and simulates the design.

4. report_measure <options>

   In this step, report the differences between relax and stress simulation results.

Example:

The example provided below demonstrates the steps that have been explained above. Here, the `define_measure` command is specified to provide the measurement details. Once it is specified, the measurement is done for measure named, `valid_meas`. Next, `validate_measure` is used to specify `valid_meas` to confirm the measurement that is of interest. Then, the `validate_macro` command runs the simulation considering the vectors provided by the user in the table (`.tbl`) file. In the final step, the `report_measure` command is specified to report of error(s), if any.

```
----------------
set cell sram
set voltage 1.2
set tskew 30e-12
set vc50 [expr $voltage * 0.5]
set vs50 [expr $voltage * 0.5]

define_measure \
-name _valid_meas_A \
-trig clk  -trig_dir rise -trig_val $vs50 \
-targ { dout[0] } -targ_dir fall -targ_val $vs50 \
-failed_val "0" \
$cell

define_measure \
-name _valid_meas_B \
-trig clk -trig_dir rise -trig_val $vs50 \
-targ { dout[0] } -targ_dir rise -targ_val $vs50 \
-failed_val "0" \
$cell

define_measure \
-name valid_meas \
-trig clk -trig_dir rise -trig_val $vs50 \
-equations {"max(_valid_meas_A,_valid_meas_B)"} \
-keep min -duration 1.0 \
$cell

define_arc -measure valid_meas -pin clk -pin_dir R $cell

validate_measure "valid_meas" $cell
validate_macro -thread 1 -validsim $fastspice
report_measure -all "valid_meas" $cell
```

F

Legacy Commands and Variables

This chapter lists all the commands and variables that are either deprecated, or included for backward compatibility. We would like to discourage users from relying on these commands and variables. Instead, find the alternate command or variable along with its settings to achieve the best results from Liberate MX.

Deprecated Commands

The following commands are being phased out, and have been replaced by either new commands (or related options), new variables, or new behaviors of the tool.

mx_set_hsim_param

`<list>`	Passes a list of HSIM parameters that is used during FastSPICE simulation.

This variable is used to pass parameters to the HSIM simulator. Parameters are passed as a list of name-value pairs.

Parameters can also be passed by specifying them in a table and using the define_table command. Parameters specified in a table override parameters that are specified with a Tcl command.

mx_set_nanosim_param

`<list>`	Passes a list of NanoSim parameters that is used during FastSPICE simulation.

This variable is used to pass parameters to the NanoSim simulator. Parameters are passed as a list of name-value pairs.

Parameters can also be passed by specifying them in a table and using the `define_table` command. Parameters specified in a table override parameters that are specified with a Tcl command.

Deprecated Variables

mx_remove_rc

Use the mx_remove_rc_pincap and mx_remove_rc_timing variables instead.

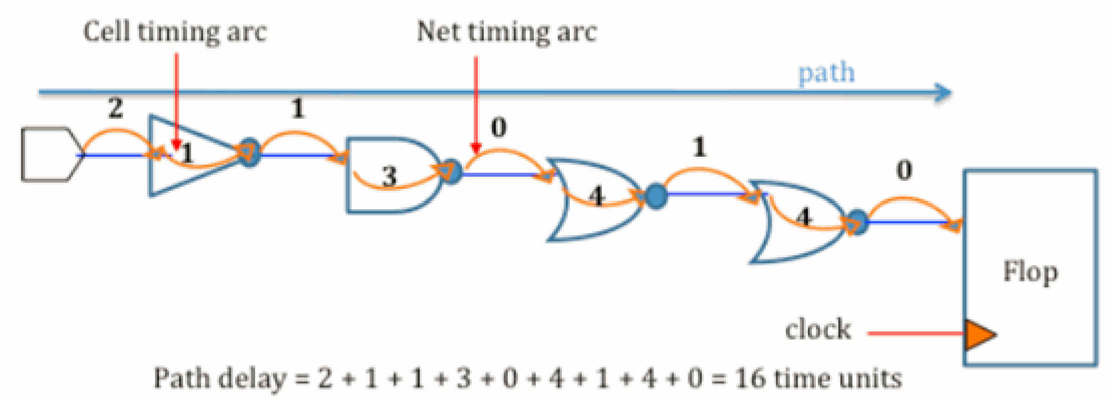
G
Glossary

Inside View

All characterization solutions available in the Virtuoso Characterization Suite use a unique “inside view” pre-characterization circuit analysis technique to perform vector generation and binning, automatic indices selection, and optimization of timing constraint characterization. This enables fully-automated library creation mechanism.

Static Timing Analysis (STA)

It is an approach to verify timing of digital designs that uses cell delays and net delays to obtain path delays which is used to validate timing specification. It validates if the design can operate at the rated speed. The figure below shows a simple example in which the cell and net delays are used to obtain the path delay.


Cell timing arc
Net timing arc
path
2
1
1
3
0
4
1
4
0
Flop
clock
Path delay = 2 + 1 + 1 + 3 + 0 + 4 + 1 + 4 + 0 = 16 time units

Timing Libraries

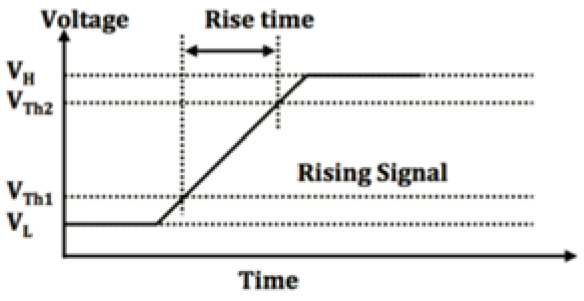
The Liberty format (`.lib`) is the industry standard for specifying the timing information. It is an ASCII file containing timing and power numbers associated with a cell. These are obtained by running SPICE simulations on the cell under a range of conditions.

Timing Arcs

A timing arc is a construct used to represent a single causal (change in input causes change in output) relationship. These arcs form the building blocks for STA.

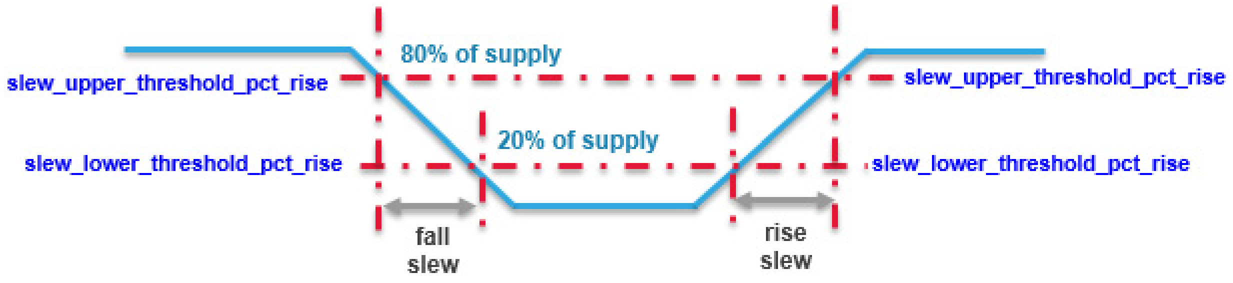
Transition Time

It is defined as the time it takes for a signal to change states between two specific levels. Rise and Fall transitions times, as shown in the figure below, are properties of a timing arc.



Slew

As slew rate is the rate of change, slew is typically measured in terms of transition time. Thresholds of signal transition times are used to measure slew. In the figure below, the threshold setting specifies that the falling slew is the difference between time points when the falling edge reaches 80% and 20% of supply value.


80% of supply
slew_upper_threshold_pct_rise
slew_upper_threshold_pct_rise
20% of supply
slew_lower_threshold_pct_rise
slew_lower_threshold_pct_rise
fall slew
rise slew

The slew thresholds are typically chosen to correspond to the part of the waveform that is linear. In newer technologies, the timing libraries will have these thresholds set to 30% and 70% of supply.

Cell Delay

Propagation delay through a cell is commonly known as cell delay. The slew of the input waveform and the load connected to the cell influence the delay value. The delay values are characterized for different values of input slew and output load. Typical delay measurements are done from 50% of input signal to 50% of output signal.

Related Pin

`Related_pin` attribute defines pin(s) that represent the beginning point of the timing arc. This attribute is required in all timing groups.

Timing Arc Types

Following types of timing arc can be used:

- Combinational timing arcs:

  These are used to describe the timing arcs for combinational element. The timing arc will be attached to an output pin and the related_pin will be an input or an output. Types of combinational timing arcs:

  - combinational
  - combinational_rise
  - combinational_fall
  - three_state_disable
  - three_state_disable_rise
  - three_state_disable_fall
  - three_state_enable
  - three_state_enable_rise
  - three_state_enable_fall

- Sequential timing arcs:

  These describe timing arcs for sequential elements. It can be a delay arc (if it describes relation between clock transition to data output i.e. input to output) or a constraint arc (if it describes relation between clock transition and data input i.e. input to input). Sequential timing arcs can be one of the following:

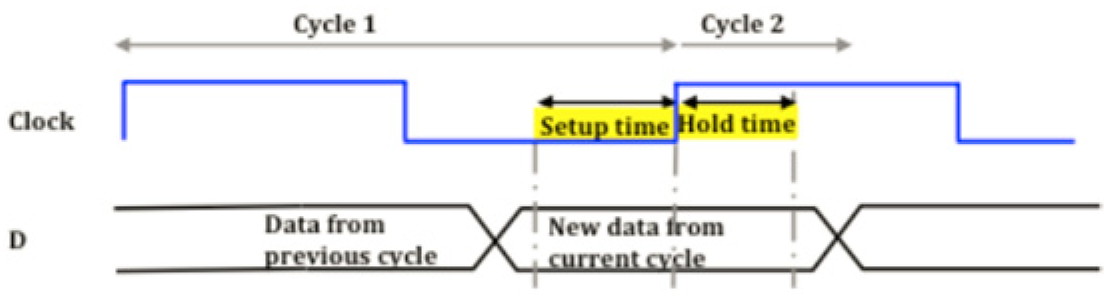
  - Edge-sensitive (rising_edge or falling_edge)
  - Preset or clear
  - Setup or hold (setup_rising, setup_falling, hold_rising, or hold_falling)
  - Nonsequential setup or hold (non_seq_setup_rising, non_seq_setup_falling, non_seq_hold_rising, non_seq_hold_falling)
  - Recovery or removal (recovery_rising, recovery_falling, removal_rising, or removal_falling)
  - No change (nochange_high_high, nochange_high_low, nochange_low_high, nochange_low_low)

Timing Sense

This attribute is used in the library to specify unateness in the .lib file.

Setup and Hold

These are synchronous timing checks that ensure proper propagation of data through sequential cells. Setup time is the duration for which input data must be stable before the triggering edge of clock. Hold time is the duration for which the synchronous input should be stable after the triggering edge of clock.


Cycle 1
Cycle 2
Clock
Setup time
Hold time
D
Data from previous cycle
New data from current cycle