cadence®

Virtuoso Liberate MX Reference Manual

Product Version 13.1
March 2014

Cadence Design Systems, Inc. (Cadence), 2655 Seely Ave., San Jose, CA 95134, USA.

Contents

Preface

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.

Related Documents

For related characterization commands applicable to Liberate MX, see:

Virtuoso Liberate Reference Manual

For information about known problems and solutions, see:

Virtuoso Foundation IC Characterization Known Problems and Solutions

For a list of new features in this release, see:

Virtuoso Foundation IC Characterization What's New

For information about other Cadence products associated with Liberate MX, refer to the following manuals:

- *ALAPI Reference Manual* describes a Tcl interface that allows access to the Liberate characterized Library DataBase (LDB)..
- *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.

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.

1

Liberate MX Introduction

This section gives an overview of the Liberate MX memory characterization.

Liberate MX provides library creation capabilities to cover 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 may not be sufficiently accurate especially as 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 in order to make accurate characterization feasible. Using a divide and conquer approach, memories and cores circuit are reduced to manageable sizes so that Liberates' unique "inside view" technology for optimizing characterization runtime can then be utilized, allowing them to be characterized with the same accuracy and techniques as standard cells. Please refer to the latest Liberate User Guide for a detailed command/ parameter description of standard cells libraries generation.

The Liberate-MX (memory characterization) product requires its own license to run, which is separate from Liberate (cell characterization).

Liberate MX Flows

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

In **Preprocessing**, spatial partitioning techniques are used to identify the basic building blocks: nand, inverter, latch, flop, arrays; the clock trees; internal probes of interest, such as latch/flop data and clock nodes for constraint checks, inputs and outputs nodes of combinational clock gaters. 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.

During **Characterization**, dynamic partitions are characterized, as in the regular Liberate flow, enabling all the characterization and modeling features, and techniques available in Liberate. See ***Figure 1***.

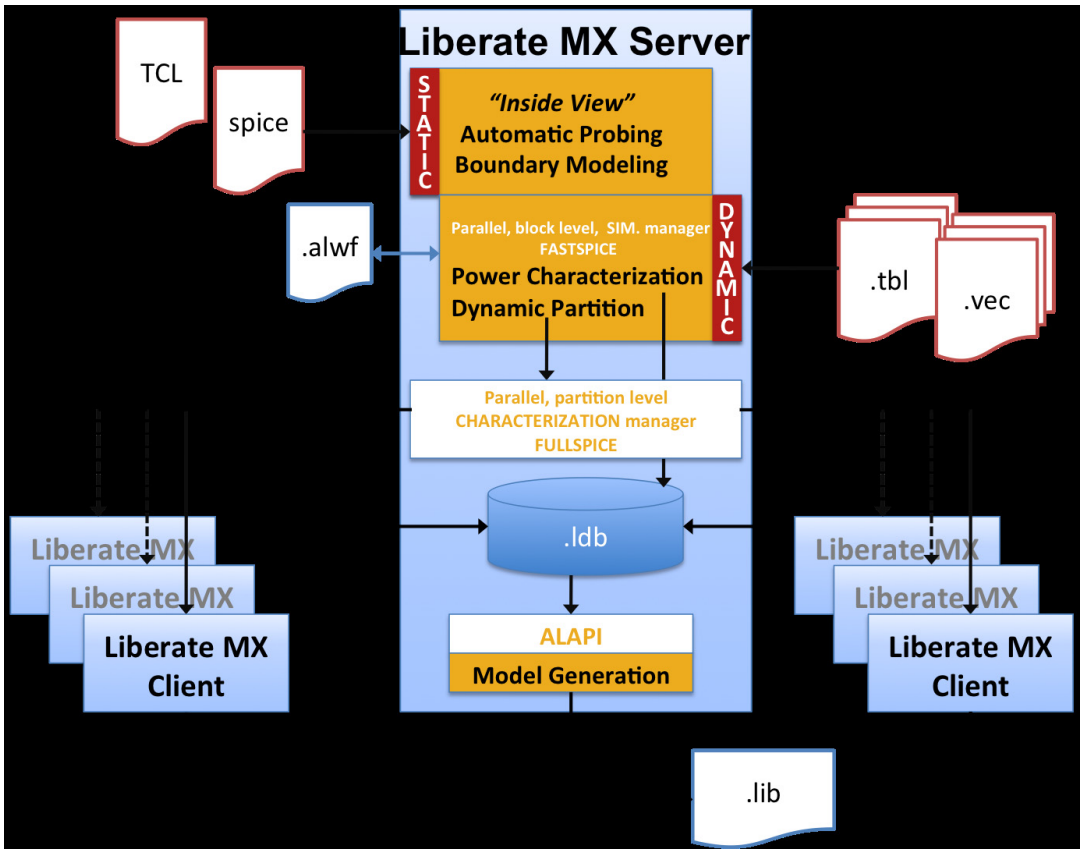


Figure 1: Liberate MX flow

There are three usage models that are available for Liberate MX. The flows are: Vendor Recharacterization Flow, Standard Custom Instance Flow, and Full Custom Instance Flow.

- Vendor Recharacterization Flow

  It is suggested to use the Vendor Recharacterization flow when there is a need to recharacterize memory instances generated by a third party IP vendor compiler. In the Vendor Recharacterization flow, all settings and vectors are automatically generated by

Liberate MX through the use of the **define_memory** and **char_memory** commands. In this flow, there is no need to provide any additional settings or vectors to generate the required liberty file.

- Standard Custom Instance Flow

  It is possible to use the Standard Custom Instance flow when characterizing custom instances which are of standard functionality. In order to use this flow, the instance must meet certain criteria. The instance should be an SRAM or ROM based design and have latched inputs and outputs. The operation of the memory can be either self timed or externally timed, however all operation must complete within the clock cycle. In the Standard Custom Instance flow, the functional information about the pinout of the instance is provided. The **define_memory** and **char_memory** commands are then used to generate all settings and vectors for the characterization.

- Full Custom Instance Flow

  The third available flow is the Full Custom Instance flow. This is for instance that do not meet the criteria of the Vendor Recharacterization flow or the Standard Custom Instance flow or situations where the greater amount of control over the generation of the settings and vectors to be used in the characterization is required. In this flow, all settings, commands, and vectors are provided to Liberate MX.

2

Getting Started

This chapter describes how to start using Liberate MX.

Before using Liberate MX, make sure that it is installed correctly and that all the necessary pre-requisite data is available. (See the **Cadence Installation Guide**, and **Cadence License Manager** manuals.)

Environment Variables

Path to Executable

Set the following environment variables to include Liberate MX in your executable path:

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

64-bit Machine Support

Liberate MX also ships with support for a 64 bit machines. To use the 64 bit port, set the environment variable **ALTOS_64** prior to calling Liberate MX:

```
% setenv ALTOS_64 1
```

Environment Variables for Controlling Licensing Checks

- ALTOS_LIC_MAX_TIMEOUT
  `setenv ALTOS_LIC_MAX_TIMEOUT <value>`

  where;

  Value is time in seconds

  This shell variable specifies how long Liberate (both Server and Client) will wait to obtain a license.

For a server process, if the ALTOS _QUEUE variable is enabled, Liberate will attempt to check out 1 Server license. If the max 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 max timeout is reached and at least one license was checked out, then 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 time 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.

Server-Client Licensing

Liberate MX uses a server-client licensing scheme. A server license is used for memory preprocessing and for invoking and monitoring the characterization run on the server machine, while the client licenses are used for running characterization on the client machines and for any post-processing 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:

- 1 characterization run using 40 client processes
- 2 simultaneous characterization runs, each with 20 client processes
- 2 simultaneous characterization runs, 1 with 30 client processes, 1 with 10 client processes

Wait for Available License

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

```
setenv ALTOS_QUEUE 1
```

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 following product licenses: Liberate_MX_Client and Variety_MX_Client. For example, to run Liberate MX with 4 threads, two Liberate_MX Client and 2 Variety_MX_Client features could be utilized.

When a Liberate server starts, it checks out 1 server license – Liberate_MX Server or Variery_MX_Server. 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_library -thread command option. Liberate MX tries to check out a combination of:

- Liberate_MX Client + Variety_MX_Client
- If Liberate MX acquires all N licenses, then it starts simulations using N threads.
- If Liberate MX acquires M licenses, 0 < M < N, and **ALTOS_QUEUE** is not set to 1, then it starts M thread of jobs.
- If Liberate MX does not acquire a license and **ALTOS_QUEUE** is not set to 1, then it quits.
- If Liberate MX acquires fewer than N licenses and ALTOS_QUEUE is set to 1, then 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, then it starts M threads.
- If M is 0, then it 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 (since **ALTOS_QUEUE** is set to 1).

Invoking Liberate MX

Liberate MX uses stdout and stderr for all messages. By default, no log file is created. To invoke Liberate MX while creating a log file:

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

Run fastsim and exit.

To start a fastsim job and exit, invoke Liberate MX with this command-line option:

```
% liberate_mx -start_fastsim_and_exit
```

This will accomplish three things: 1) MX starts fastsim jobs and exits, 2) Generate a script to be sourced at the next run, 3) Inform the user about what to do next.

System Libraries

Liberate MX ships enabled 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
```

Preparing For Characterization

Four pieces 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.

Extracted Memory Block Netlist

The transistors, diodes, resistors, capacitors and extracted parasitic elements (RCs) that comprise the memory are passed to Liberate MX in SPICE format. Extracted SPICE netlists can be created directly from the memory layout by device and interconnect parameter extraction tools. Standard SPICE and Hspice® netlist formats are currently supported. The memory to be characterized must have a **.subckt** definition in the files passed to Liberate MX. To specify the memory netlists, use the **read_spice** Tcl command:

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

Device Models

The device models are supplied by the foundry and represent the electrical parameters of the target process. The device models include models from transistors (P and N channel), diodes, capacitors and resistors. Most device model files include different parameters for different process corners such as a typical corner, fast corner and a slow corner. To read device model into Liberate MX, use the **read_spice** Tcl command:

```
read_spice {models.spi sram.lpe}
```

To specify the voltage and temperature to use for characterization, use the **set_operating_condition** command:

```
set_operating_condition -voltage 1.2 -temp 25
```

Tcl Command File

Liberate MX uses the Tcl scripting language to control the characterization process. The Tcl script is used to specify the memory netlist, SPICE models, and operating conditions. In addition, the Tcl script defines the range of data that the characterization is to be performed over, such as input slew and output-loading conditions. Liberate MX will simulate and measure the memory using each of the specified input slews and loads and will generate the appropriate delay tables, timing checks (setup, hold etc) 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). The Tcl commands available for controlling Liberate MX are detailed in Chapter 3 of this manual.

A sample Tcl script for running Liberate MX is shown below. This script will characterize the memory block named 'sram'.

```
set pwd [exec pwd]

# Define operating conditions ##
set vdd 1.2
set gnd 0.0
set temp 25
set_operating_condition -voltage $vdd -temp $temp

# define templates
define_template -type delay \
    -index_1 { 0.010 0.050 0.200 0.400 1.000 } \
    -index_2 { 0.0009 0.060 0.250 0.500 1.000 } \
    SRAM_delay_template

define_template -type constraint \
    -index_1 { 0.010 0.050 0.200 0.400 1.000 } \
    -index_2 { 0.010 0.050 0.200 0.400 1.000 } \
    SRAM_constraint_template

define_template -type power \
    -index_1 { 0.010 0.050 0.200 0.400 1.000 } \
    -index_2 { 0.0009 0.060 0.250 0.500 1.000 } \
```

```
    SRAM_power_template
set delay_table "SIG2SRAM_delay_template"
set constraint_table "SIG2SRAM_constraint_template"
set power_template "SIG2SRAM_power_template"

# define cell, pins, truth tables
set cell { sram }
set mxtables {}
lappend mxtables $pwd/tables/sram.tbl

define_cell \
    -clock { CLK } \
    -input { ADR[9:0] D[11:0] WEM[11:0] \
             WE OE ME \
             AWT \
             TADR[9:0] TD[11:0] TWEM[11:0] \
             TWE TOE TME \
             BISTE \
             } \
    -output { Q[11:0] } \
    -delay $delay_table \
    -constraint $constraint_table \
    -power $power_template \
    -mxtable $mxtables \
    $cell

# read spice
set spicein {}
lappend spicein $pwd/data/cl_models.sp
lappend spicein $pwd/data/sram.sp
read_spice $spicein

# clock prop
mx_set_clockprop {{ME enable CLK} {TME enable CLK}}

# setup partition simulator
set partition_simulator "spectre"
set_var fastsim_cmd "spectre"
set_var fastsim_cmd_option " +xps +spice +cktpreset=sram -64 -format spice"

# set characterization simulator
```

```
# if none specified, internal alspice used
set characterization_simulator "spectre"

# specify models the memory should be characterized for
# partition (with fast spice) and characterize (with real spice)
set models [list -ccs -ccsn -ecsm -ecsmn]
char_macro \
    -extsim $partition_simulator \
    -char_params [concat $models -extsim $characterization_simulator]
# write models
write_library -overwrite sram.lib
foreach model $models {
    write_library -overwrite $model sram.${model}.lib
}
```

Input Stimuli File

Please refer to Specifying Input Stimuli

3

Liberate MX Commands

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

All command arguments that are preceded with a - 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 output that lists all currently-available options for that command. There may be options that will print out when help is used that are not officially supported. The only supported options are those that are documented in this manual. When help is used, all other command options are ignored.

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_library**, **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

-char_params {*parameters***}**Parameters for Characterization. Default: none.

-char_script <*script_name***>**Extra Script for Characterization. Default: none.

-extsim "*simulator_name***"** External SPICE simulator program. Default: none

-thread <*number***>** Maximum number of threads to use on the current machine. Default: 0

-arc_thread <*number***>** How to multithread arcs update in preprocessing; overwrites -thread value. Default: 0

-fastsim_thread <*number***>** How to multithread fast spice in preprocessing; overwrites -thread value. Default: 0

-skip {data_type} Skip characterization of selected categories. See below for supported types. Default: do not skip any types.

-wave_thread <*number***>** How to multithread waveforms update in preprocessing; overwrites -thread value. Default: 0

-write_thread <*number***>** How to multithread components writing in preprocessing; overwrites -thread value. Default: 0

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

- "finesim"
- "hsim"
- "nanosim"
- "ultrasim"
- "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 fast-spice simulation, results acquisition, arcs selection and file IO operations. Note that the number of parallel fast-spice runs is determined by the max among number of different mxtables files provided and number specified for **thread** argument – see **Specifying Input Stimuli** chapter.

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_library** 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.

skip disables characterization of specific categories of data. Supported types are: delay, power, leakage, constraint, mpw, cin, setup, hold. A list containing multiple categories is supported. We recommend this option only be used to improve runtime while studying the characterization output from Liberate for a specific category. For example, while tuning the setup for constraint characterization, use **-skip** {delay power leakage cin hold} to speed up the runtime for constraints.

Examples:

```
# Characterize memory(s) defined via a previous define_cell
# command. Use Synopsis' HSIM for dynamic partitioning and
# use Synopsis' HSPICE 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 "hsim" -thread 2 \
    -char_params {-extsim "hspice" -thread 4 -ccs -ccsn} \
    -char_script bsub.tcl

# where bsub.tcl is a tcl file containing:
if {![info exists mx_g_timing_char]} { return }
if {$mx_g_timing_char != 1 } { return }
set_var rsh_cmd "qsub -b y -q linux64 %C"
```

char_memory

-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 comparision result in text format (Default: diff.rpt)

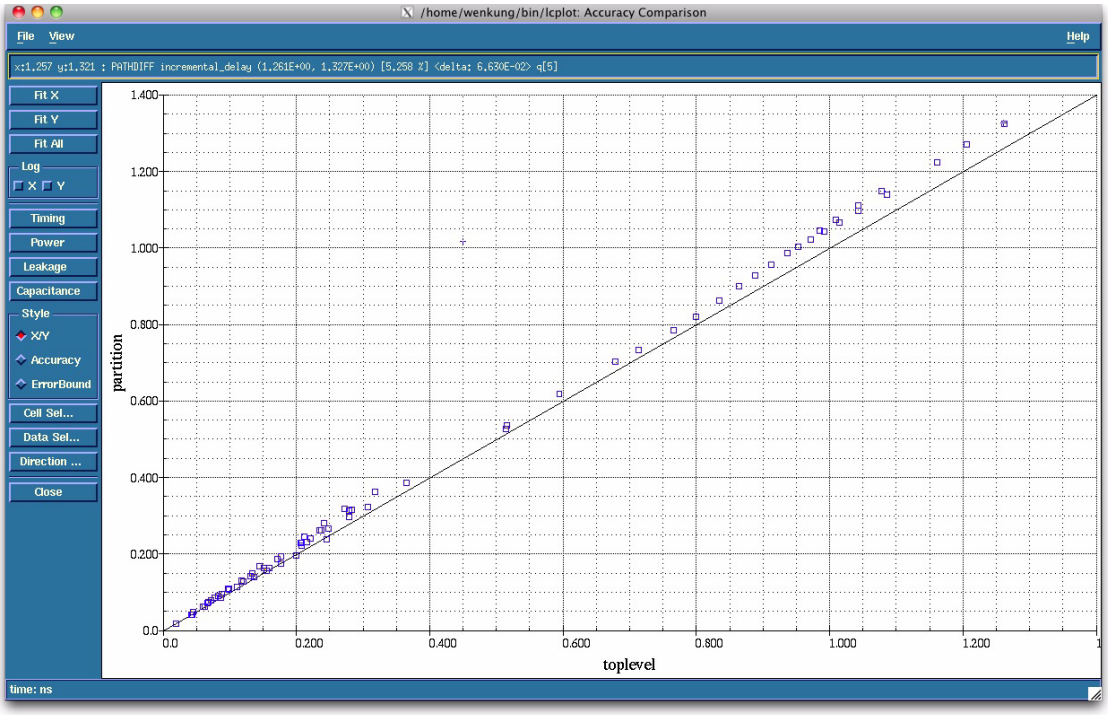
-top_report <string> A report of the cumulative delay path as seen from the top level (FASTSPICE numbers). (Default: sim.top.rpt)

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 comparion report output by the **compare_path** command:



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` **<***string***>}**
Instructs MX on how to choose a representative of a specific measurement group before going to full spice characterization.

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

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

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

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

-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.

-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 transistion (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 & 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. But 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 min one will generate a partition and be characterized with full spice.

- **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 3 separate define_arc commands as in:

```
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 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_exppressions***>**
   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

-clock {pin_names}	List of clock pin names
-constraint <name>	Name of template for constraint tables
-delay <name>	Name of template for delay tables
-input {pin_names}	List of input pin names
-mxcore {core_files}	List of one or more core cell description files
-mxtable {table_files}	List of one or more behavioral tables
-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: II, **<>**, **[]**, **{}**, **()**, _ 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 **mxtable** argument specifies the list of behavioral truth table files used to generate stimuli for Liberate MX. (See Truth Table Format.)
Note: This can also be accomplished with the command define_table.

The **mxcore** argument specifies the list of core cell description files used to match memory core cells in the design during automatic probing. Please refer to "Specifying memory core cells in MX" for a detailed description.

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_leafcell

-area <"string"> Name of area diode parameter (default 'area')

-length "string" Name of the Length mos parameter (default 'l')

-multiple "string" Name of Multiple mos parameter (default 'm')

-pin_position {list of pin positions}
Specify 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 leaf cell 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_measure

-duration <# of cycles> Measurement duration in number of cycles, starting at this trigger (Default: 1)

-equations <list> Set of related equations consisting of 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 mx 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.

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 amount of time 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 amount of time 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 amount of time 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 amount of time 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 which 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 Resuse 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_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).

-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)

-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)

-cfg_file <cfg_file_name>
The cfg file generated from the compiler along with the instance (Virage and TSMC recharacterization flows)

-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`.

-char_spice "string" Characterization spice simulator to be used for characterization. Allowed values are aps, spectre, hspice, or finesim. Default: `aps`

-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)

-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)

-design <sram|rf|uhdrf|rom>

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

-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. (Vendor Recharacterization flow)

-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}`.

-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: `spice`.

-models <model_include_file>
Spice Model include file.

-mx_setting <mx_tcl_file>
Name of user provided tcl file containing `mx setting` settings and commands. This is used to override the internal 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`.

-part_spice <simulator_name>
Partition spice simulator to be used for characterization. Allowed values are `xps`, `ultrasim`, `aps`, `hsim`, `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*.

-pin_file <pin_file> File containing the pin name information.

-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`.

-ref_lib <ref_lib_file> Name of reference library. Do not use if `-template` is specified.

-remove_tables {list_of_tables}
List of automatically generated tables to be skipped in the run. Available values are: `leakage`, `delay`, `constraint`, `power`, `measure`, `bist`, `sleep`, `bist_pwr`.

-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.

-simulation_interval <number>
Value for mx_simulation interval in seconds. Default is `20e-9`.

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

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

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

-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.

-vendor <ARM|Virage|TSMC>
Name of Vendor. (Vendor Recharacterization 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).

{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`.

Example of the usage of **define_memory** for the recharacterization of a Vendor Instance:

```
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 \
    SRAM_2048x16
```

Example of the usage of define_memory for the characterization of a Standard Custom Instance:

Example 1 pinout (Single port 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 \
    SRAM_2048x16
```

Example 2 pinout (Dual port with test mode):

```
    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 \
    SRAM_2048x16
```

define_table

cell <name> Cell name. (Required)

mxtables {table_files} List of vector table files. (Required)

Allows vector table files to be specified without using the **-mxtable** option to the **define_cell** command. (See Specifying Input Stimuli.)
Example:

```
define_table myMemCell delay.tbl
```

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_mode

-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 names} List of cells.

The **mx_match_mode** 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).

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 fast spice 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_hsim_param
mx_set_nanosim_param
mx_set_ultrasim_param

<list>	List of FineSim / HSIM / NanoSim / UltraSim parameters that will be used during Fast Spice simulation.

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

```
mx_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} \
}
```

Note: These parameters may also be specified in a table format using the **-mxtable** option to define_cell. Parameters specified in a table will *override* parameters that are specified with a Tcl command. (See fastsim_deck)

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.

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. Suppored simulators are: `"aps"`, `finesim"`, `"hsim"`, `"nanosim"`, `"ultrasim"`, `"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: fast-spice simulation, results acquisition, arcs selection, and file IO operations. The number of parallel fastspice runs is determined by the max number from among the different mxtables 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 MX Timing Validation Flow.

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 -ccsp  <ccsp.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 (see `char_library -skip {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: 2.1
-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_library, 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 turn off 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_library`, 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
```

4

Liberate MX Variables

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 | 1 | 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**.

extsim_deck_include

< 0 | 1 >	Controls how the fast-spice simulation deck is written out. Default: 0

The **extsim_deck_include** command lets you control how the fast-spice 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 fast-spice
#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_library.** 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.

lic_max_timeout

<value>	Specifies the amount of time, in seconds, to wait for each license feature/token before skipping and checking with the next possible license feature/token. Default: 600s

Note that this variable only works when the shell environment variable **ALTOS_QUEUE** is set to 1. See the Liberate MX Licensing section in Chapter 2 of this manual 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_library**.

lic_queue_timeout

<value>	Specifies the amount of time, 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_library**.

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:

```
setvar mx_autoprobing_hold_level 0;
```

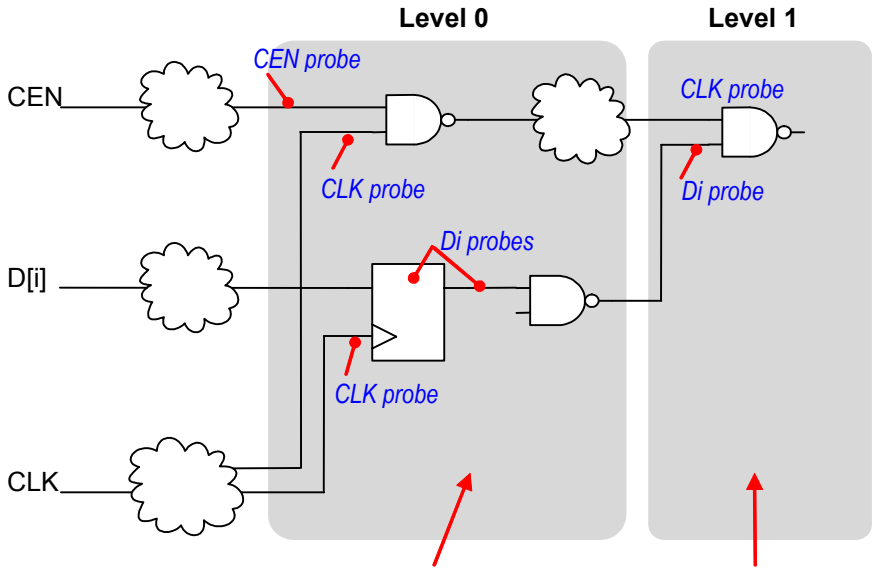
mx_autoprobing_setup_level

<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**.

```
setvar mx_autoprobing_hold_level 1;
```



mx_bisection

< 1 | 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 | 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 | 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_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: 1

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 >

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.

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 (such as HSIM) with the same options they use standalone. Examples:

```
# Pass switch to HSIM
set_var mx_distributed_sim "hsim -mt"

# Pass info to job queue
set_var mx_distributed_sim "bsub -q <queue_name> -o <log> -I hsim"
```

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 Fast Spice 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 fast spice 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 fast spice simulation results from a previous run. Default: 1

Results of fast spice 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 fast spice 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
# fast-spice results for current run
set_var mx_fastsim_reuse 1
```

mx_find_arrays

< 0 | 1 >	Identifies large channel-connected components as possible candidates for memory arrays. Default: 1

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 | 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 > Identifies nodes that should be treated as logic 1, 0 during static analysis. Default: 0

When set, 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 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 - i.e. > 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 1 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.
```

mx_fix_pin_vdd

< 0 | 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 | 1 > Generates dynamic partitions out of **define_measurement** commands. Default: 0

mx_greybox

< 0 | 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 | 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 | 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 | 1>	Reuse timing, power, pin-cap, noise, power and leakage LDBs from previous run. Default: 0

MX is able to reuse all the LDBs or selective LDB from previous run with 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 | 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_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 runtime 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.

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 | 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 | 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 | 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 | 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 | 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_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 | 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 fast spice 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

This variable has been deprecated.
Use variables **mx_remove_rc_pincap** & **mx_remove_rc_timing**.

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

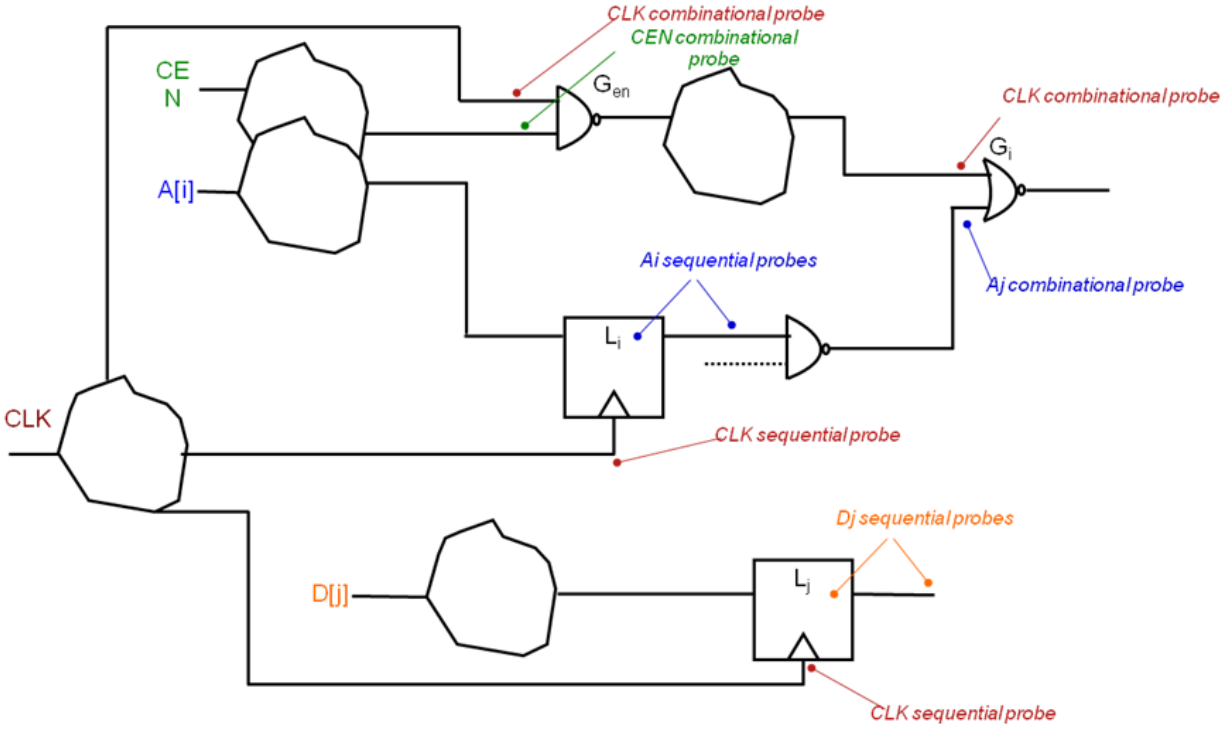
<string>	Sets the node type to consider on a sequential component as possible candidates for probes in setup/hold characterization Default: state output internal.

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 only first-level sequential probes for both setup and hold constraint 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 "none"
set_var mx_hold_comb  "none"
```

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 fast spice 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.

mxtable_dontcare_value

{<bus_name | pin_name> < 0 | 1 | hex_number>}
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 Truth Table Format

Example:

```
set_var mxtable_dontcare_value {oe 0 ctrl 0xc}
```

mx_verbose

< 0 | 1 > Prints basic flow and runtime information. Default: 1

Example:

```
#report basic flow info
set_var mx_verbose 1
```

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 | 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.

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_cell** command via the **mxtable** option.

-mxtable option

The **-mxtable** option for the **define_cell** command specifies a list of truth table files. The number of specified table files determines the number of fast-spice runs that will be performed in parallel – use **thread** option of **char_macro** command to control.

The file syntax for **-mxtable option** is as follows:

```
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> <arc_type_string>

Vectors specified in the table file will be used for <arc_type_string> characterization.

<arc_type_string> *One or more of:*
delay, retain, setup, hold, minperiod, mpw, power, leakage, measure, timing

Note that timing equals delay retain setup hold. If **<arc_type>** is not specified, the default is timing measure.

<fastsim_option>	fastsim, fastsim_reuse, fastsim_deck_include, fastsim_deck, simulation_interval, fastsim_auto_ic, fastsim_model_include, fastsim_cmd, fastsim_cmd_option

fastsim <fastsim_identifier>

<fastsim_indentifier>	hsim xa nanosim ultrasim adit raser spectre

Specifies what fastspice engine will be used to simulate the table file. Overwrites **char_macro -extsim**

This example specifies Spectre as the fast-spice simulator:

```
fastsim spectre
fastsim_deck .option rawfmt=fsdb
```

fastsim_auto_ic <value>

<value>	Either 1 or 0 (May also be specified as **true**, **on**, **false**, **off**)

Instructs 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** option 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
```

Specified in table timing.tbl will ensure that vectors listed in that table will use a simulation interval of 20 ns rather than the default 10ns (which will be used for any other table that doesn't have a simulation_interval keyword).

Use this to specify a particular simulator executable and its options:

fastsim_cmd <path to executable>

<path to executable> Specifies the path to an executable to be used for simulating this table file.

fastsim_cmd_option <command options>

<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 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>

<fastsim_deck_string> A valid text (for the engine specified in fastsim) that will be included (as is) to the deck being simulated in fast spice.

This variable is used to specify truth table files fastspice simulator options in 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 ultrasim
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 <netist_path_name>

<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 or the option will be ignored.

fastsim_model_include <model_path_name>

<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.

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
```

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** 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	Descripton
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.
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 measurments (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, dont_care may 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 **dont_care** 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 / disable arcs

Three state characterization is performed in MX if both following conditions are met:

- Three state enable and disable lines exist in an mxtable
- User defined arc of type enable/disable are available (-type enable / -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. Please note that the table file containing enable/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
# Use the table for both timing and power and run it with HSIM speed 3

arctypes timing power
fastsim hsim
.option hsimspeed 3

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
    }}}}
```

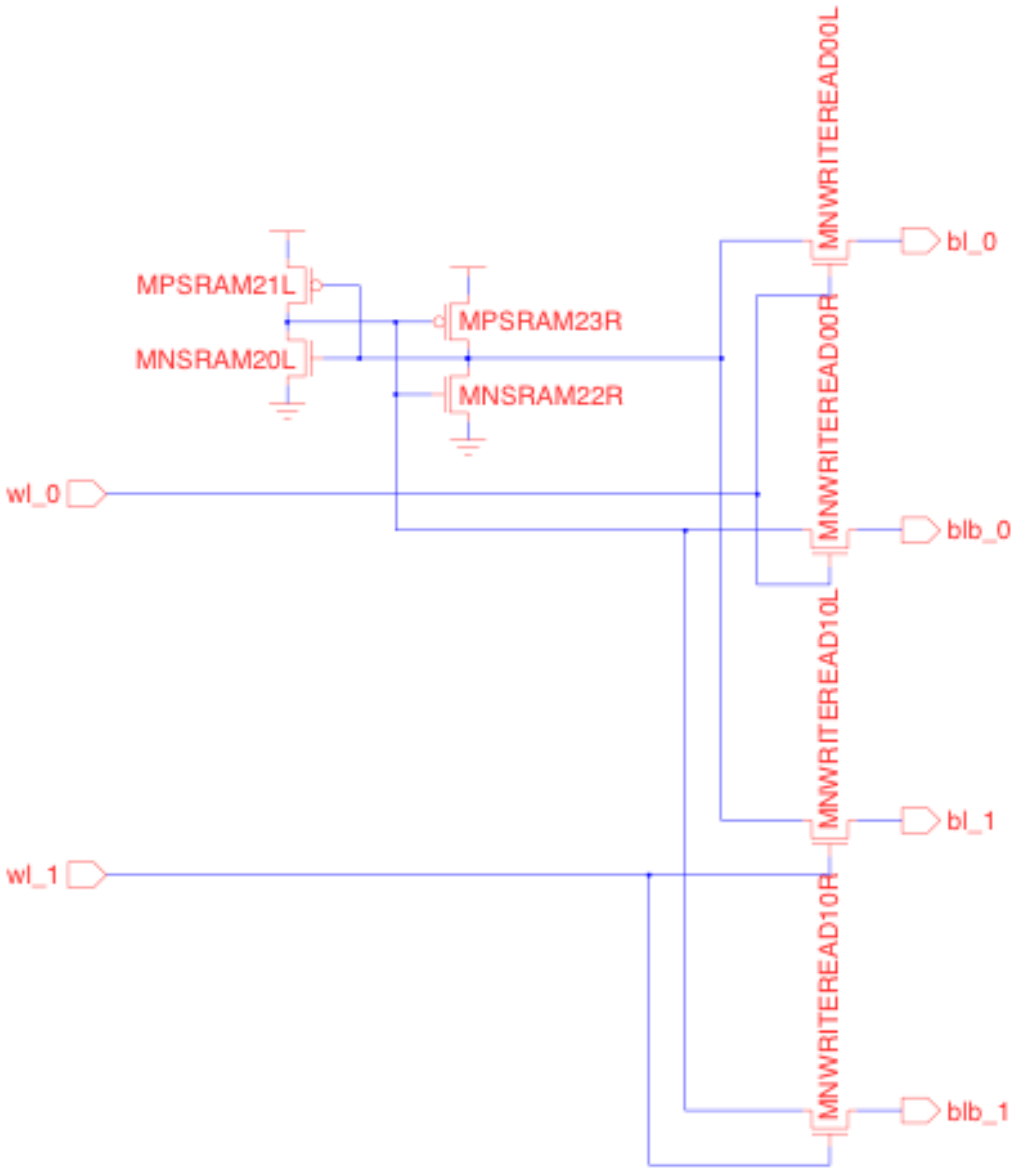


Figure 2 – 8T core cell

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
        }}}}
```

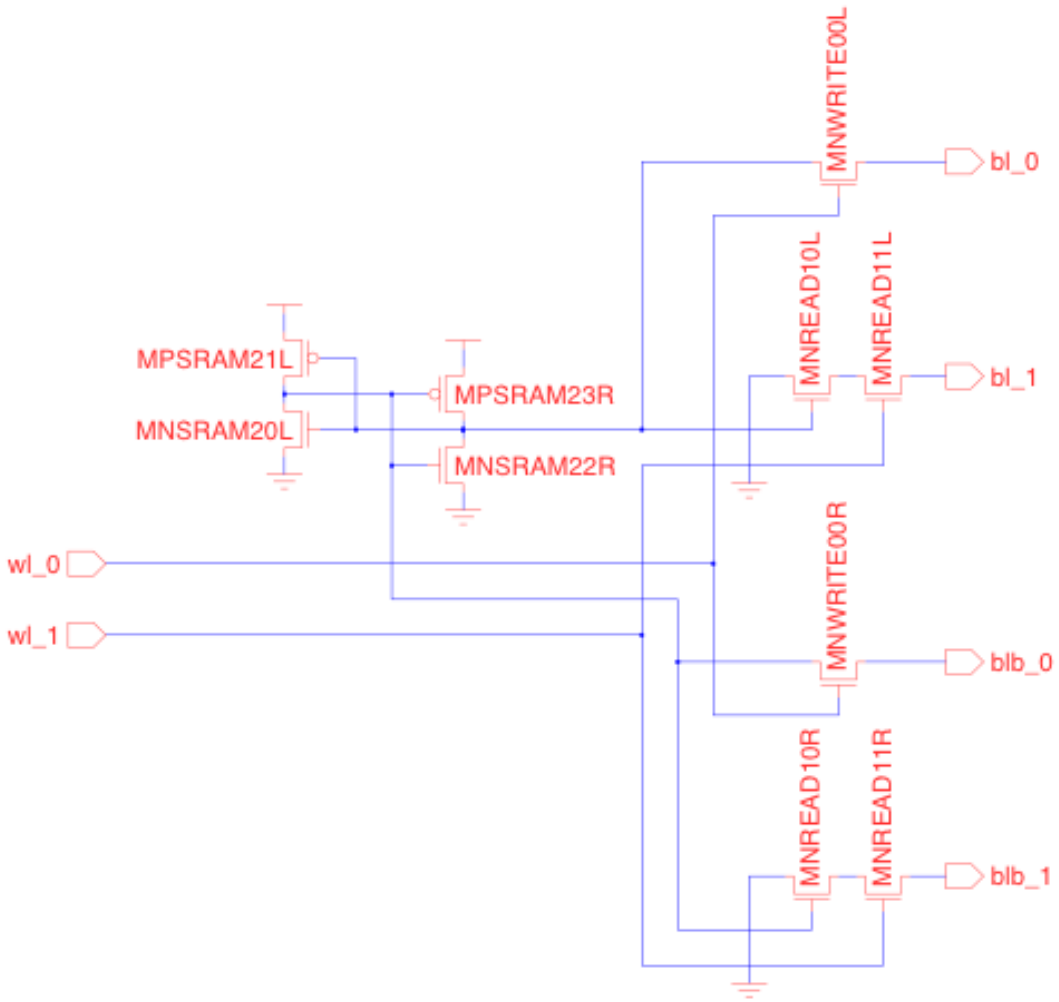


Figure 3 – 10T core cell

C

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 1) above.

   Example mxv.tcl:

   ```
   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, then 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/constrant 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>
```