SYNOPSYS®

Module Compiler Express

User Manual

Version 1998.02

February 1998

Printed in the U.S.A.

Document Order Number: 32902-000 EA

Module Compiler Express User Manual, Version 1998.02

About this User Guide

The Module Compiler Express User Guide describes the functionality of the Module Compiler Express product. This document includes instructions for installing the product and for using it from either the graphical user interface or from the command line. A complete list of functions and their uses is included in Chapter 5.

Audience

This reference manual is for logic designers or electronics engineers using Module Compiler to design complex datapath modules.

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
Courier	Indicates command syntax. In command syntax and examples, shows system prompts, text from files, error messages, and reports printed by the system.
Courier italic	Indicates a user specification, such as *`object_name`*
Courier bold	In command syntax and examples, indicates user input (text the user types verbatim).
[ ]	Denotes optional parameters, such as *`pin1 [pin2, . . pinN]`*
|	Indicates a choice among alternatives, such as `low | medium | high` This example indicates that you can enter one of three possible values for an option: low, medium, or high.
_	Connects two terms that are read as a single term by the system. For example, design_space.
(Control-c)	Indicates a keyboard combination, such as holding down the Control key and pressing c.
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Edit > Copy	Shows a menu selection. *Edit* is the menu name and *Copy* is the item on the menu.

Table of Contents

List of Figures

List of Tables

1 MCE Basics

This chapter provides an introduction to Module Compiler Express and lists the available functions.

Topics in this chapter include:

- A comparison of Module Compiler and Module Compiler Express (MCE)
- A description of the contents and functionality of MCE
- An in-context synthesis example
- A list of functions provided by MCE

Introduction

Module Compiler Express (MCE) is the baseline package for Module Compiler (MC). MC is a high-performance datapath synthesis tool used to build the complex datapath modules that are used in demanding applications such as multimedia, graphics, digital signal processing, and communications.

Module Compiler provides access to an HDL that you, as an ASIC datapath designer, can use to author new datapath modules that are specifically tailored to your particular application. Module Compiler Express provides a graphical interface to a set of standard, pre-coded, commonly used datapath blocks. You get Module Compiler's high-performance synthesis and optimization engine, but instead of writing HDL code to describe your own modules, you use graphical panels to specify parameters for pre-coded modules.

In summary, Module Compiler is an authoring tool, while Module Compiler Express enables you to use predesigned modules. Module Compiler Express is an excellent synthesis tool as long as you do not require the HDL authoring capabilities of MC for demanding applications with high-performance datapaths.

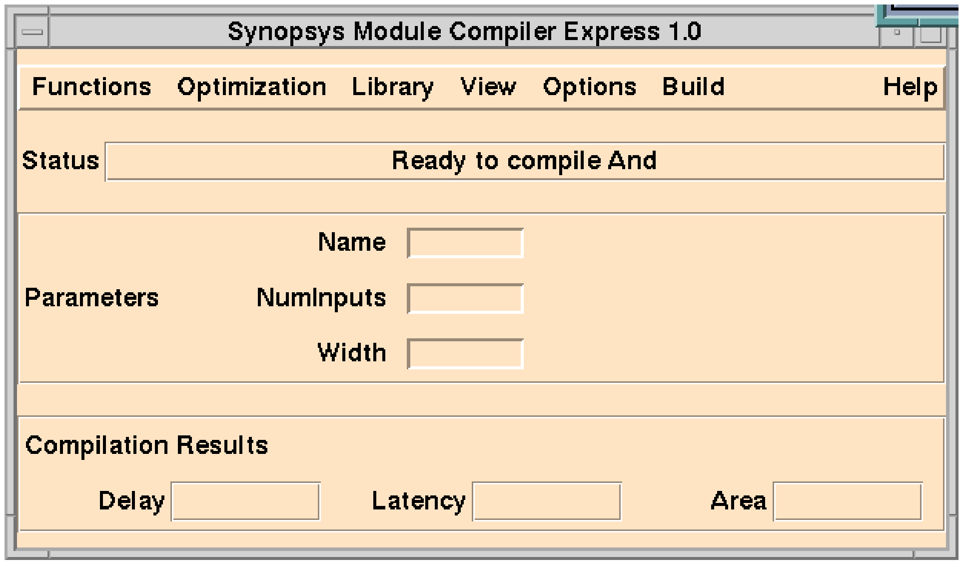
What's in MCE

Module Compiler Express provides over 30 commonly used datapath blocks ranging from incrementors to FIR filters. Each block has a graphical panel in which you enter parameters such as width or format. Module Compiler Express builds the chosen block to your specification.

The outputs of Module Compiler Express are:

- Gate-level netlist (Verilog or VHDL)
- RTL model (Verilog or VHDL)
- Analysis report

Figure 1-1 The Module Compiler Express Graphical User Interface



You can analyze the generated results and integrate the block into your larger design. If you want to pursue higher performance, lower area, or lower power, you can easily regenerate the results for a different set of parameters, perhaps using different operator architectures. In this manner, you can achieve high-quality results with a fraction of the cost associated with hand-crafting.

Moving Up to Module Compiler

As you gain experience with Module Compiler Express and as your datapath application grows in complexity, you will want to upgrade to the full functionality of Module Compiler. To achieve truly hand-crafted performance, you need to go beyond generating individual components singly and then instantiating them in the parent module. You will want to author a datapath module that fits your application needs precisely. Module Compiler brings to bear several techniques that cannot come into play in Module Compiler Express because of the inherent nature of using precoded modules.

Module Compiler's primary technique is *in-context architectural synthesis*. This is a powerful concept that provides substantial improvements in timing and area for designs that have significant operator interaction. Each operator in these designs is generated in the exact context of its use and can therefore be precisely optimized. This is in strong contrast to methodologies in which the operators are generated out of context and then stitched together.

If you have a module that contains several datapath blocks of the sort available in MCE, you can generate each block individually in MCE and then interconnect them. Or, you can code the module in Module Compiler, making function calls to the MCE blocks. In the latter case, Module Compiler is able to apply in-context synthesis to all the operators. Not only do you benefit from easier integration, you get a higher quality of results because Module Compiler applies operator-level optimization and propagate delays, constants, and loading constraints.

In addition, many techniques such as carrysave, automatic pipelining, pipeline loaning, constants, and operator merging make sense only in the context of a larger design. Module Compiler provides additional built-in functions, analysis capabilities, support for automatically iterating through different versions of your module, and relative placement for driving layout.

An In-Context Synthesis Example

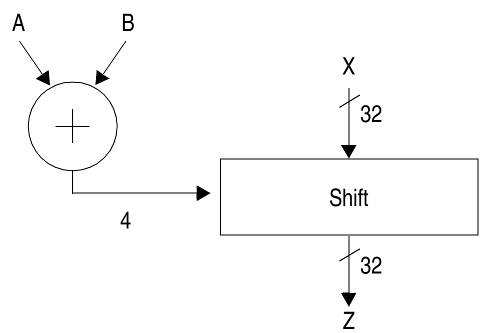
The following design can be built using either MC or MCE. Using MCE, you generate the adder and the barrel shifter and instantiate them in the parent module. If you want the fastest design, you choose to build a fast-carry-lookahead adder (which sacrifices area to get speed) and a fast barrel shifter.

Using MC, you generate the entire module and let Module Compiler build the barrel shifter in the context of its use. Module Compiler will stage the select MUXs to take advantage of the skew in the select lines. A ripple-add structure will do nicely here, even though the adder itself is not a high-speed architecture. The overall implementation in Module Compiler will have smaller area and smaller delay.

The examples below contrast the work flows in MC and MCE.

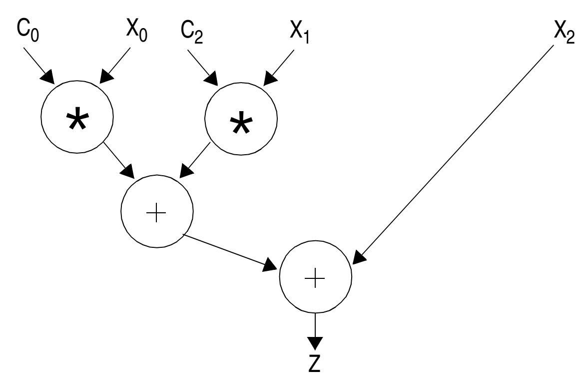
Method using MCE	**Method using MC**

• Generate add
• Generate shift
• Connect
	• Write code
	`S=A+B;`
	`Z=X >> S;`
	• Generate



The following design can be built easily using MCE.

Method using MCE	**Method using MC**
• Generate multiply • Generate add • Connect	• Write code: $Z = C_0 * X_0 + C_1 * X_1 + X_2$ • Generate



Similar design styles apply to structures such as constant coefficient FIRs or color space converters.

Functions

The five categories of functions that can be synthesized are shown below. This grouping of the functions matches that found within the GUI menu system. Please note that the complete reference for these functions is provided in Chapter 5, “MCE Basics”.

Logic Functions

Logic functions are commonly used functions that are needed in almost every design.

Table 1-1 Logic Functions

Function	Description
And	Variable width, variable number of input, AND tree
Counter	Variable width up counter
CountUpDown	Variable width up/down counter
Equal	Variable width equality comparison
Mux	Variable width, variable number of inputs multiplexor
Or	Variable width, variable number of input, OR tree
Xor	Variable width, variable number of input, XOR tree
Shift	Left/right signed/unsigned barrel shifter
Rotate	Left/right signed/unsigned rotator
Sat	Saturate input to smaller output

Adder Functions

The adder functions are based on addition.

Table 1-2 Addition-Based Function

Function	Description
Abs	Absolute value
ACC	Variable width accumulator
Adder	2 input adder with optional carry input
Addsub	2 input Adder/Subtractor
Compare	Variable width signed/unsigned magnitude comparator
CompGE	Comparator with both greater than and equal-to outputs
Decrement	Variable width decrementer
Increment	Variable width incrementor
Neg	Conditionally negate the input
Subtractor	2 input subtractor
Sum	Variable width, variable number of input, sum tree

Multiplier Functions

The multiplier functions are based on multiplication and on multiplication-addition.

Table 1-3 Multiplier Functions

Function	Description
DSPMAC	Computes C+/-A*B, supports extended precision
FIR	N-point FIR (finite-impulse-response) filter
Inner	N-point inner product (sum of products)
MAC	Multiplier-accumulator
MACCS	High-speed multiplier-accumulator
Multiplier	Two input multiplier
MultConst	Multiplies a signal by a constant
Square	Computes X*X
Divide	Unsigned Divider

Alu Functions

The Alu category contains a set of programmable functions. Unlike the functions in the other categories, these are programmable at runtime rather than at compile time.

Table 1-4 Alu Functions

Function	Description
Alu	16-function variable width ALU
Norm	Leading 0/1 normalization
ShiftLR	Arithmetic/Logical Left/Right Shifter
LeadingZero	Leading 0, Leading 1 detector

Memory Functions

The memory functions are functions that store data. These functions are logic implemented.

Table 1-5 Memory Functions

Memory	Description
nR1W_Latch	Netlist-based RAM with only 1 write port
nRmW_FF	Netlist-based multi-ported synchronous RAM

Installation and Setup

2

This chapter describes how to install the Module Compiler Express software and how to set up the user and group environments.

Chapter 2 discusses the following topics:

- Platform requirements
- User Quick Start
- Installation instructions
- System administration information
- Instructions for building pseudo-cell libraries

Platform Requirements

The graphical user interface for MCE requires the X Window system. The command-line interface for MCE can be used in any terminal environment. Other specific platform requirements are outlined below.

- SUN-Sparc workstation, SunOS 4.1.3, Solaris, HP, SGI
- Main Memory: 64MB
- Swap Space: 250MB
- Disk Space: 20MB

User Quick Start

To get started as an end user of Module Compiler Express, follow the steps in this section. If you are functioning as the administrator and need to install and maintain MCE, follow the steps given in the “Installing Module Compiler Express” and “System Administration” sections.

First Run

The following steps assume that Module Compiler Express has been properly installed. To run MCE for the first time:

1. Create a clean directory. In the example below, the directory is `mcproj`:

   ```
   % mkdir mcproj
   ```

   ```
   % cd mcproj
   ```

2. Initialize your UNIX environment. In most cases, the administrator will have put the necessary path information into the `setup.csh` file, so you type the path to that file; for example:

   ```
   % source /mc1.2/localadm/setup.csh
   ```

   This sets the variables that point to the MCE program and to the directories containing the technology libraries.

3. Start MCE using the -tech switch to specify the technology library you want to use. MCE will not run without a technology library.

   ```
   % mce -tech XYZ
   ```

   This loads the specified library and runs MCE in GUI mode.

4. To test the MCE installation, fill in the following values in the opening screen

Field	Value
Name	Test
NumInputs	2
Width	8

   Click the Build button. MCE should build a bitwise AND circuit, showing its progress in the Status field. See Chapter 3 for information about using MCE’s graphical user interface.

Subsequent Runs

When you run MCE for the first time, it creates an `mc.env` file in the directory where you started it and stores all your settings in it, both those set from the command line and those set in the GUI. When you start MCE again *in that directory*, MCE reads the settings and starts up in the previous configuration. You can override these settings with command-line switches.

Installing Module Compiler Express

This section and the following sections of this chapter are intended for use only by system administrators who need to install and maintain MCE. End users can ignore the remainder of this chapter.

1. During installation, set umask to

   ```
   % umask 22
   ```

   or

   ```
   % umask 2
   ```

2. Determine where you would like to install the software and create a new directory in that location. These instructions assume that the location is `/mc1.2`. To create the new directory, type:

   ```
   % mkdir /mc1.2
   ```

3. Change directory to the new location:

   ```
   % cd /mc1.2
   ```

4. Load the contents of the tape into this directory. The following is an example `tar` command for loading the tape:

   ```
   % tar xvf /dev/rst0
   ```

5. Execute the following commands:

   ```
   % set x = (‘\ls | grep -v localadm‘)
   % chmod -R a-w $x
   % unset x
   % mkdir tech
   ```

6. `localadm/setup.csh` is a source script that sets up the UNIX environment for users of the “csh” shell. Change the MCDIR variable in this file to reflect the actual location chosen in Step 2 above. Type in a complete path: do not use the “~” character in the path.

7. `localadm/setup.csh` assumes that the UNIX script `/bin/arch` correctly returns the platform name, such as sun4. If this is not the case, change the character sequence `‘/bin/arch‘` to the name of the platform (sun4, for example).

8. Execute the following command:

   ```
   % \ls -lF
   ```

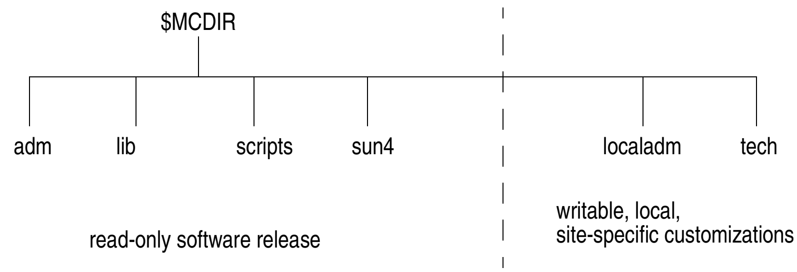
   A successful installation should look something like this:

   ```
   total 6
   dr-xr-xr-x  2 root  512 Oct 10 20:14 adm/
   dr-xr-xr-x  6 root  512 Oct 10 20:14 lib/
   drwxrwxr-x  2 root  512 Nov  5 01:53 localadm/
   dr-xr-xr-x  2 root  512 Oct 10 20:14 scripts/
   dr-xr-xr-x  3 root  512 Oct 10 20:14 sun4/
   drwxrwxr-x  2 root  512 Nov  5 14:39 tech/
   ```

System Administration

Once installation has been successfully completed, you should see a directory structure similar to the following one:

Figure 2-1 MCE Directory Structure



The read-only portion of the directory tree should be preserved in its original state without any modifications.

- The `localadm` directory is the place for local administration and setup files.
- `localadm/setup.csh` is a source script that you can use to initialize your UNIX C shell environment.
- `localadm/mc.env` is a file that contains site-specific settings for MC environment variables.
- The `tech` directory is where technology-specific library files should be kept, including the Synopsys “db” libraries. This is not required, but is convenient. When MC creates pseudo-cell libraries, they are written into the `tech` directory.

UNIX Environment Variables

Module Compiler and Module Compiler Express use a few UNIX environment variables. Most of these variables can be initialized using the `localadm/setup.csh` source script. Table 2-1 identifies and defines the various UNIX environment variables used by MC and MCE.

Table 2-1 UNIX Environment Variables

Type	Variable	Description
MC	MCDIR	The path name of the software installation point. This is the Module Compiler root directory location. *Required.*
	MCLIBDIR	The path name of the technology library directory. This directory holds all the technology libraries for MC including any pseudo-cell libraries. *Required.*
	MCENVDIR	In addition to the Unix environment variables, there are a number of Module Compiler environment variables that are specified in `mc.env` files. These are not Unix variables. The MC environment variables are described in the *Module Compiler Reference Manual*. MCENVDIR is a list of path names to directories that contain `mc.env` files. The directory "./" is implied at the beginning of the list. The priority is decreasing from left to right so that the variables set in the working directory have the highest priority, followed by the other directories given in the list. *Optional. MC uses $MCDIR/adm if MCENVDIR is not defined.*
License	SYNOPSYS	See Design Compiler installation.

Technology-Specific MC Variables

Module Compiler and Module Compiler Express have a number of environment variables that are specified in `mc.env` files. These MC environment variables are described in the *Module Compiler Reference Manual*. Some of these variables have technology-specific versions. A technology-specific MC environment variable has the technology name appended to the normal variable name. Table 2-2 identifies and defines the technology-specific MC variables. "XYZ" is used as a placeholder for the technology name.

Note: When a variable has both technology-independent and technology-specific versions, the technology-specific version has the highest priority.

Table 2-2 Technology-Specific MC Environment Variables

Variable	Description
dp_tech_lib dp_tech_lib_XYZ	A comma separated list of db files. The file names must include the full path name. These db files comprise the technology library.
dp_dc_wireload dp_dc_wireload_XYZ	This variable is the named wire load model from the technology library.
derate_slow_named_opcond derate_slow_named_opcond_XYZ	The named operating condition from the technology library that is used when the operating condition is **slow**.
derate_typ_named_opcond derate_typ_named_opcond_XYZ	The named operating condition from the technology library that is used when the operating condition is **typ**.
derate_fast_named_opcond derate_fast_named_opcond_XYZ	The named operating condition from the technology library that is used when the operating condition is **fast**.

Using the `mcenv` Program

You can use the `mcenv` program to set and query MC environment variables. When you set an MC variable, `mcenv` stores the value in the `./mc.env` file. When you query the value of a variable, `mcenv` first checks the `./mc.env` file for the MC variable. This ensures that the working directory has the highest priority. Next, it check directories in the `MCENVDIR` list from left to right until the variable is located. This is the same mechanism that MC and MCE use when they check for variable values.

To set an MC variable, specify the variable name and its new value as shown in the following example:

```
% mcenv dp_opcond slow
```

This sets the value of the *dp_opcond* variable to **slow**.

To query the value of an MC variable, specify the variable name as shown in the following example:

```
% mcenv dp_opcond
```

This returns the value of the *dp_opcond* variable.

To query the value of an MC variable that has a technology-specific version, use the **-tech** switch:

```
% mcenv -tech dp_tech_lib
```

This returns the value of the *dp_tech_lib* variable with the highest priority. Since all technology-specific variables have higher priority than technology-independent variables, `mcenv` returns the technology-specific version for the current technology if one exists.

Customizing MC Environment Variables for All Users

Module Compiler and Module Compiler Express have a number of environment variables that are specified in `mc.env` files.

Note: In practice, you need to set only a few of these variables. In fact, you may never need to set any of these variables. The software installation stores default values for all of these variables in the `$MCDIR/adm/mc.env` file. The system administrator can override these default values for all users by setting MC environment variables in the `$MCDIR/localadm/mc.env` file. This is a convenient way to set preferences for the entire group.

The technology-specific MC variables are the variables that the system administrator most commonly needs to manage in the `$MCDIR/localadm/mc.env` file. To initialize the technology-specific MC environment variables, follow the steps below. These instructions assume that the software installation point is `/mc1.2`, and that the technology name is “XYZ”.

1. Initialize your UNIX environment. For example:

```
% source /mc1.2/localadm/setup.csh
```

2. Change directory to the `localadm` directory:

```
% cd $MCDIR/localadm
```

3. Execute the following commands (where “XYZ” is the technology):

```
% mcenv dp_dc_wireload_XYZ <my_wireload>
% mcenv derate_slow_named_opcond_XYZ <my_worst>
% mcenv derate_typ_named_opcond_XYZ <my_typical>
% mcenv derate_fast_named_opcond_XYZ <my_best>
```

4. Set the *dp_tech_lib_XYZ* variable based on the location of the library files for the XYZ technology. Assume that the XYZ technology has two db library files. If these files are located in the `$MCDIR/tech` directory, execute this command:

```
% mcenv dp_tech_lib_XYZ ’(MCLIBDIR)/XYZ.db,(MCLIBDIR)/XYZ_wires.db’
```

If the db files for the XYZ technology are located in the `/my/dbs/go/here` directory, execute this command:

```
% mcenv dp_tech_lib_XYZ /my/dbs/go/here/XYZ.db,/my/dbs/go/here/XYZ_wires.db
```

Licensing

This program requires an MCE version 1.0 license key. Module Compiler Express uses the standard Synopsys floating license manager. In addition, a separate key is needed for each MCE library. These keys are listed in Table 2-3.

The environment variables *SYNOPSYS* and/or *SYNOPSYS_KEYFILE* need to be set. Refer to the *Synopsys Installation User Guide* for more information.

Table 2-3 Required MCE Library Keys

Key Name	Function Categories
MCE-Lib-Logic	Logic
MCE-Lib-Arith1	Adders
MCE-Lib-Arith2	Multipliers: all except FIR and Inner
MCE-Lib-Arith3	Multipliers: FIR and Inner
MCE-Lib-Alu	Alu
MCE-Lib-Memory	Memory

Building Pseudo-Cell Libraries

To build a pseudo-cell library for a given technology, the system administrator should follow these steps. These instructions assume that the software installation point is `/mc1.2` and that the technology name is "XYZ."

1. Initialize your UNIX environment. For example:

   ```
   % source /mc1.2/localadm/setup.csh
   ```

2. Execute the following commands:

   ```
   % cd $MCLIBDIR
   % makeMcLib
   ```

   The `makeMcLib` program displays its usage as follows:

   ```
   usage: makeMcLib <tech> [<wireload>]
   by default 2.5 load per fanout is used
   ```

3. Determine the most commonly used wire load model. The pseudo-cell library will be built and characterized using this wire load model.

4. Execute `makeMcLib` for your technology and wire load model. This example builds the pseudo-cell library for the XYZ technology, using the "wires_15K_used" wire load model.

   ```
   % makeMcLib XYZ wires_15K_used
   ```

Running MCE

3

This chapter describes how to operate the Module Compiler Express product.

Topics in this chapter include:

- Using the graphical user interface
- Using the command-line interface

Using the Graphical User Interface

Module Compiler Express has a very simple graphical user interface (GUI), consisting of a menu bar, the status display, the parameter entry area and the output result area.

To start MCE in GUI mode, type the following command:

% mce -tech <*technology name*> -gm +

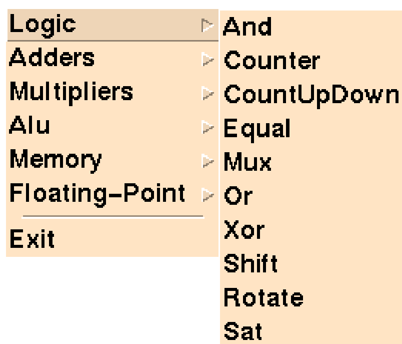
The Menu Bar

The menu bar items allow you to set options and to perform actions such as building a circuit and viewing output files.

Figure 3-1 The MCE Menu Bar

The following sections describe the choices in each menu and provide an overview of how each item is used in MCE.

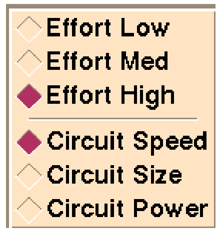
Figure 3-2 The Function Menu



Functions

Select an available function or quit MCE.

Figure 3-3 The Optimization Menu



Optimization

Select how long optimization can take and whether to optimize for speed, size or power

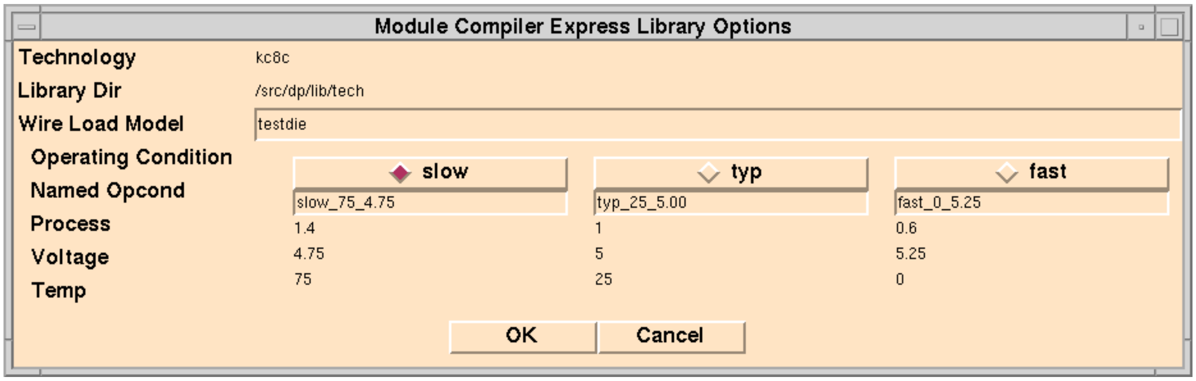
Effort Low
Runs fast and produces the poorest results.

Effort High
Runs the slowest and produces the best results.

For small circuits and when generating final results, choose Effort High. For quick estimates of large circuits, choose Effort Med or Effort Low.

Also, select the type of circuit optimization (Circuit Speed, Circuit Size or Circuit Power). The corresponding optimization criterion is used during logic optimization. Note that some functions may override the value set here.

Figure 3-4 The Library Menu



Library Dir

Displays the name of the directory that contains the technology library files.

Technology

Displays the name of the currently loaded technology library.

Wire Load Model

Shows the name of the current wire load model. To change the model, type in a new name. If you type in a model name that is not in the current vendor library, MC pops up a list of available models. You can find detailed information about what's available in the current vendor's technology library by choosing Library Report from the View Menu.

Operating Condition

Selects the conditions under which the chip is likely to be used. These are radio buttons, so only one can be selected.

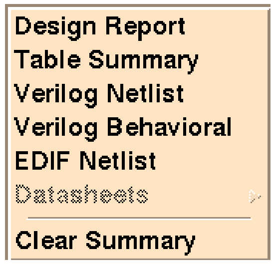
Named Opcond

Specifies which model in the technology library is associated with each of the Operating Condition radio buttons.

Process, Voltage, and Temp

These three items display the values assigned to these items by the associated Named Opcond.

Figure 3-5 The View Menu



View

Select an output file to view. All files except the Table Summary are available only after a successful compilation.

Design Report
View a single module summary including statistics, critical path and cell usage.

Table Summary
View multiple module summaries for comparing the results of different compilations. (Clear with the ‘Clear Summary’ option.)

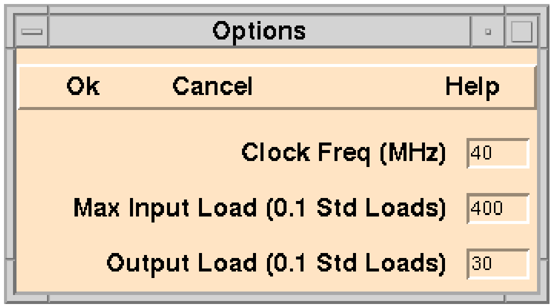
Verilog/VHDL Netlist
The logic level netlist in Verilog or VHDL (as specified by the starting command line option).

Verilog/VHDL Behavioral
The behavioral model in Verilog or VHDL (as specified by the starting command line option).

EDIF Netlist

Clear Summary
Clear the summary file

Figure 3-6 MCE Options



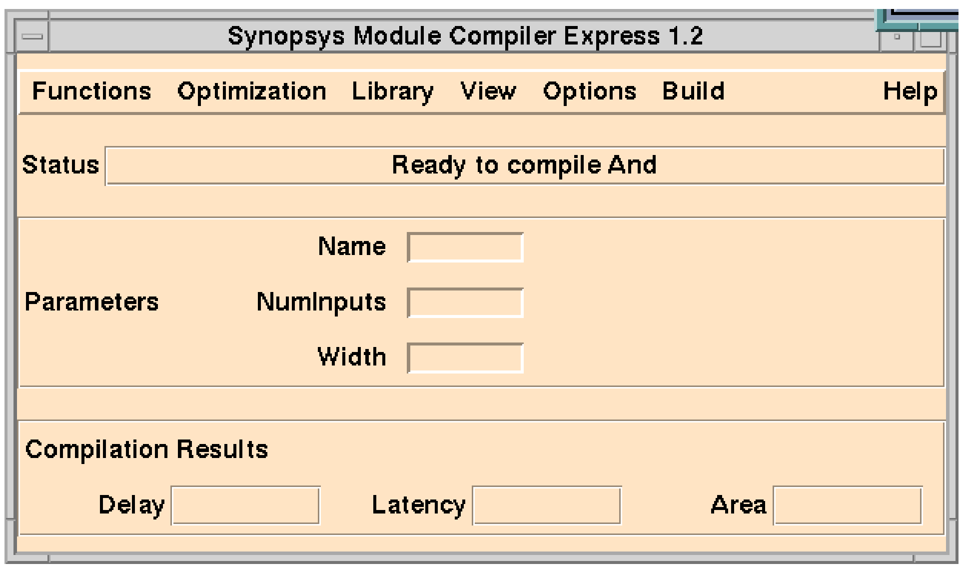
Options

View and change secondary options: clock frequency, input and output loading.

Build

Choose Build to start the compilation.

Figure 3-7 The Main MCE Window



The Main Window Status Area

The status window indicates the current status of Module Compiler Express and can indicate simple problems with short diagnostic messages. Check this window to see if the compilation succeeded. The messages are summarized in Table 3-1.

Table 3-1 Status Messages

Message	Meaning
Building <name>, please wait ...	Compilation is in progress.
Compilation failed for <name>	Something went wrong. In general, a parameter is out of range. Module Compiler Express will pop up an error window with more information to help correct the problem.
Compilation succeeded for <name>	Everything worked. You can now view the output files.
Please select a function first	You must select a function from the Functions menu before proceeding.
Please set a value for <name> first	One of the parameters has no value. Set it before building.
Ready, choose a function to compile	Module Compiler Express has been initialized. You should now choose a function from the function menu.
Ready to Compile: <name>	The function you selected has been initialized. You may compile it after all parameters have been entered.

The Parameter Entry Fields

Use these fields to set the parameters for the current function. The parameters change for each new function. All values entered are saved in the MC environment when Build is pressed and are automatically reloaded the next time the function is selected. See Help/Module for details of the parameter meanings.

Module Compiler Express uses three types of parameters to customize each function: string, integer and Boolean. The Parameters section of the GUI is reconfigured when a new function is selected. All of the parameter values must be entered before the function can be compiled.

Toggle *Boolean parameters* by clicking the appropriate button. The Boolean parameter has a TRUE value when the small square at the left of the button is filled.

To select an integer or string field to edit, click the left mouse button in the entry area. Type the desired value of the parameter or use the special keystrokes shown in Table 3-2 for editing.

Table 3-2 Keyboard Shortcuts for Parameter Entry

Key	Use
Left-Arrow	move cursor one character left
Right-Arrow	move cursor one character right
Ctrl-h	destructive backspace
Ctrl-d	delete character to the right of cursor
Ctrl-u	clear the entry field

The Compilation Results Area

Delay
The critical path delay.

Latency
The number of pipeline cycles inserted.

Area
The area of the circuit. The area of the logic is reported; wire area is not included.

After a successful compilation, the delay, latency, and area are displayed in the Compilation Results area. The delay in nanoseconds indicates the critical path delay of the circuit. This number may not be meaningful for some functions such as the netlist RAMs.

The Compilation Process

To compile a function, follow these steps:

1. *Set the options*. The operating condition, optimization effort, clock frequency and input/output loading constraints should be set if desired.

   Generally, the operating condition is set only once using the “Library” menu, but it can be set at any time. Always run with the “slow” option, since this provides the most realistic delay estimates.

   In general, set the optimization effort to Effort High to provide the best circuit area and delay. However, it may be best to lower the optimization effort for some large circuits to produce estimates more quickly.

   Select “Speed,” “Area,” or “Power” as the primary optimization criterion for the circuit, as appropriate for your design constraints. Note that if a function allows you to set the delay goal for pipelining, the function sets a new optimization criterion that overrides the one you set in the GUI.

   Finally, the clock frequency and input and output loading options can be set by from the Options menu. Each entry field is edited in the same manner as described for parameters.

   - Set the clock frequency to the desired value in MHz.
   - Loading constraints are entered as integers with units of 0.1 std loads. Note that raising the output loading or reducing the maximum input loading increases the area and delay of the circuit.
   - When all of the options have been set, choose OK to accept and save the changes, or choose Cancel to discard the changes.
2. *Choose a function*. Pick one of the functions from the Functions menu. The summary provided earlier or the reference pages are a good source of information to help in choosing the best function for your current need.
3. *Enter or modify the parameters.* After a function has been selected, the desired parameters can be entered. If you are unsure what a particular parameter value should be, check the reference or the Module help from the Help menu.
4. *Compile the function* by pressing “Build.” The cursor changes to a watch while the function is being compiled.
5. *Check the outputs.* The Compilation Results area provides quick information regarding the area and delay of the circuit. To view one of the five output files, select the appropriate item from the View menu. Menu entries are dim when the corresponding file is not available. Note that if any of the output file viewers are left open, each is automatically updated at the successful completion of a compilation.

History

When you choose Build from the GUI menu bar, MCE saves the current function and the parameter values of each function in the MCE environment. The next time you start MCE, the current function and all parameters are automatically initialized to the previous values. When you select a function for which you have previously entered parameters, the previous parameter values are used as initial values. This history mechanism works only in the directory from which MCE was started. If the you change to another directory and start MCE, no history values are available from the previous run.

Using the Command-Line Interface

To use MCE from the command line for batch operation, start MCE with the **-gm -** option.

% mce -gm - [*other options*]

In this mode, you can set all of the parameters from the command line that can be set from the GUI. All parameters are optional. If a parameter is not specified, MCE obtains its value from the MCE environment.

To see a complete list of available command-line options, type the following command:

% mce -h

Every specified parameter is saved in the environment. Table 3-3 lists the available command line options.

Table 3-3 Command-Line Options for MCE

Option	Meaning	Values
-cf	Clock frequency in MHz	
-ds	Slew derating control	- = underrated, + = derated
-eval	Library evaluation mode	
-f	function name	
-fp	Flatten pseudo-cells	- = don't flatten; + = flatten
-gm	Graphical interface mode	+ = GUI; - = text
-h	list available options	
-il	Default maximum input loading in 0.1 std load	
-l	log file name	- for stdout
-o	Optimization effort	0=low; 1=med; 2=high
-oc	Operating condition	fast; typ; slow
-ol	Default output loading in 0.1 std load	
-opt	Optimize mode	speed; area; power
-p	Parameters	name=value, name=value,...
-pl	Use pseudo-cell library	-=don't use; +=use
-sp	Enable spinner	-=disable; +=enable
-tech	Name of technology library	
-ver	Print full version	
-vm	Verbose mode	+=verbose; -=normal
-vhdl	VHDL mode	+=VHDL; -=Verilog

Setting Functions and Parameters

The **-f** option specifies a function and the **-p** option sets the parameters for the function. There is a list of available function names in Chapter 1 and a complete description of each function in Chapter 5. All parameters for a given function must be entered with a single **-p** in the form *name=value*. Use commas to separate the parameters, with no spaces around the commas. For example, to construct a 10-bit, cla-based subtractor named "sub," enter the following:

```
% mce -gm - -f Subtractor -p Name=sub,Width=10,AdderType=cla
```

Loading Constraints

The **-il** and **-ol** options set the input and output loading constraints, respectively. Each takes an integer value with units of 0.1 standard loads.

Clock Frequency

The the **-f** option sets the clock frequency for power computations. This option does not affect the delay goal of the compiled circuit.

Operating Condition

The **-oc** option sets the operating condition to **fast**, **typ**, or **slow**.

Optimization Level

The **-o** option sets the optimization level. A value of 0 gives minimum optimization but run fast, 1 is a medium setting, and 2 gives maximum optimization to produce the best results.

MCE Output Files

4

This chapter discusses the MCE output files. Use these result files to analyze your design and plan future design changes.

This chapter discusses:

- Information provided in the output files
- Structural and behavioral models
- Datasheets

The Output Files

MCE generates five output files that you can view from the GUI. Two are used to examine the performance and characteristics of the circuit; three provide behavioral and structural information. Other files are created, but cannot be viewed from the GUI.

Statistics

MCE provides two types of statistical reports: the Design Report and the Summary. Together, they provide a wealth of useful information to help you analyze your design and plan future designs.

The Design Report

The Design Report provides detailed information for the last compiled function. Its name is *name.report* where *name* is the name you supplied in the Parameters section. The Design Report File contains the following sections:

- The name of the function being compiled
- The parameters supplied by the user
- Characteristics of the function
- The critical path
- Information for each input and output bit, including: delay, slack and loading information
- A list of cells used in the circuit

The **critical path information** shows the longest path from beginning (bottom of the list) to end (top of the list). For each point on the path, the critical input to the cell is shown followed by the net name. Table 4-1 describes the columns in the Critical Path portion of the Statistics Report.

Table 4-1 Columns in the Critical Path Summary

Column	Contents
delta	The change in delay between each net and the previous net on the path
delay	The maximum delay at the net
rise	The maximum rise delay at the net
fall	The maximum fall delay at the net
load	The total load on the net using an estimated wire load model
gload	The total cell input loading
pins	The number of pins on the net

If the critical path ends at a flip-flop, the setup time of the flip-flop is shown at the end of the critical path summary.

A sample Design Report file is shown in Example 4-1.

Example 4-1 A Sample Module Compiler Express Design Report File

```
Synopsys Module Compiler Express Report
MCE Version: 1.0
Input File: /src/dp/lib/dp//dplite//Adder.comp.dpa
Function: Adder
Module Name: adder8
Parameters: Name=adder8,Width=8,AdderType=clsa,CarryIn=0,CarryOut=0
Date: Fri Nov 22 17:57:53 1996


Options
     Technology Lib Dir: /src/dp/lib/tech/
     Technology: lca500k
     Operating Condition: WCCOM
     Operating Temperature: 70
     Operating Voltage:  3.13
     Wireload Model: B5X5

     Optimization Criterion: speed
     Optimization Effort: High

     Clock Frequency: 40

     Default Input Max Load: 400
     Default Output Load: 30
     Behavioral Model File: ./adder8.bvrl
     Logic Model File: ./adder8.vrl
     EDIF Output File: ./adder8.edif
```

```
Summary

                   DESIGN   TIMING (ns)   POWER        AREA       LATENCY
                     name  final internal  (W)    ff  inst   area   cycles
--------------------------------------------------------------------------
                   adder8    2.2     2.2  0.00     0    46    108     0


Design: adder8
-------------------------------------------------------------------------------
               Number of instances:       46
                      Number of ff:        0
                    Number of nets:       62
                    Number of pins:      156
                     pin/net ratio:      2.5
                              Area:      108
         Longest final path (nS):       2.16
      Longest internal path (nS):       2.16
                           Latency:        0
                         Power (W):    0.000
-------------------------------------------------------------------------------

    Critical Path Summary ...
    Path Ends at: Z1[7] Z_1_[7] Z[7]
    Endpoint is in group: misc, slack:  -2.155, delay goal:   0.001
                                 delta  delay   rise   fall   load  gload   pins
                       setup:    0.00   2.16   2.16   2.16
  /Z1[7]/I49/EO3P(C->Z)/N146:    0.48   2.16   2.16   2.16    4.3    3.0     2
  /Z1[5]/I47/AO7P(A->Z)/N180:    0.45   1.67   1.49   1.68    3.3    2.0     2
   /Z1[1]/I19/AO6(B->Z)/N154:    0.36   1.23   1.23   1.18    5.7    3.2     3
   /Z1[1]/I18/NR2(A->Z)/N153:    0.42   0.87   0.70   0.86    4.4    1.9     3
  /Z1[2]/I69/NR2P(B->Z)/N116:    0.44   0.44   0.44   0.19    7.4    3.7     4
                      /X[2]:    0.00   0.00   0.00   0.00    9.0    5.3     4
```

```
Input Summary
--------------------------------------------------------------------------------
    CLK[0:0] (signed)
          Bit   Load    Fanout    Delay  Relative Slack
            0    0.0       0       0.00   100000.00

    X[7:0] (unsigned)
          Bit   Load    Fanout    Delay  Relative Slack
            0    5.5       2       0.00       0.18
            1    9.0       3       0.00       0.12
            2    9.0       3       0.00       0.00
            3    9.0       3       0.00       0.05
            4    9.0       3       0.00       0.01
            5    9.0       3       0.00       0.21
            6    8.0       3       0.00       0.06
            7    2.4       1       0.00       1.26

    Y[7:0] (unsigned)
          Bit   Load    Fanout    Delay  Relative Slack
            0    4.5       2       0.00       0.18
            1    7.6       3       0.00       0.12
            2    7.6       3       0.00       0.00
            3    7.6       3       0.00       0.05
            4    7.6       3       0.00       0.01
            5    7.6       3       0.00       0.21
            6    6.6       3       0.00       0.06
            7    3.6       1       0.00       1.58


Output Summary
--------------------------------------------------------------------------------
    Z[7:0] (unsigned)
         Bit    Load    Int   Total  Slack   Latency Path
                       Delay  Delay                  Start
           0     4.3    0.64   0.64  -0.64    0       Y[0]
           1     4.3    1.51   1.51  -1.51    0       X[0]
           2     4.3    1.86   1.86  -1.85    0       X[1]
           3     4.3    1.96   1.96  -1.96    0       X[2]
           4     4.3    1.98   1.98  -1.98    0       X[1]
           5     4.3    2.14   2.14  -2.14    0       X[4]
           6     4.3    2.08   2.08  -2.08    0       X[4]
           7     4.3    2.16   2.16  -2.15    0       X[2]
```

```
================================================================================

Cell Use Summary

     By Group:
     count (  %)       Group        total (  %)
        46 (100)        Comb          108 (100)

     Core Sorted by die area:
     count (  %)        cell area  total (  %)
         4 (  9)        EN3P    6     24 ( 22)
         3 (  7)        EO3P    6     18 ( 17)
         5 ( 11)        AO7P    3     15 ( 14)
         6 ( 13)        NR2P    2     12 ( 11)
         5 ( 11)         AO6    2     10 (  9)
        10 ( 22)         ND2    1     10 (  9)
         4 (  9)         AO7    2      8 (  7)
         5 ( 11)         NR2    1      5 (  5)
         1 (  2)          EO    3      3 (  3)
         2 (  4)          IV    1      2 (  2)
         1 (  2)         IVP    1      1 (  1)

     Core Sorted by name:
     count (  %)        cell area  total (  %)
         5 ( 11)         AO6    2     10 (  9)
         4 (  9)         AO7    2      8 (  7)
         5 ( 11)        AO7P    3     15 ( 14)
         4 (  9)        EN3P    6     24 ( 22)
         1 (  2)          EO    3      3 (  3)
         3 (  7)        EO3P    6     18 ( 17)
         2 (  4)          IV    1      2 (  2)
         1 (  2)         IVP    1      1 (  1)
        10 ( 22)         ND2    1     10 (  9)
         5 ( 11)         NR2    1      5 (  5)
         6 ( 13)        NR2P    2     12 ( 11)
```

The Table Summary File

The Table Summary file lets you compare the results of two or more compilations. Each line contains the statistics and parameters of one compiled function, and there is a line for each compiled function. To clear this file, choose Clear Summary from the View menu.

Example 4-2 shows an example of a Table Summary file after three compilations of the And function. The most recent data is at displayed at the top of the GUI window. If you view the file manually, the most recent data will be at the end of the file.

Example 4-2 A Table Summary File after Three Compilations of the **`AND`** Function

```
Module  Sections  Delay  Latency  Power Parameters
    and       45    1.97       0    0.047 Name=and,NumInputs=10,Width=5
    and      100    2.53       0    0.099 Name=and,NumInputs=20,Width=5
    and       72    1.97       0    0.075 Name=and,NumInputs=10,Width=8
```

Structural and Behavioral Models

MCE can generate output in Verilog or VHDL format. You can specify your desired format using command line argument when you start up MCE.

The netlists are provided as a convenience in the GUI file viewing menu although they should rarely need to be viewed from the GUI. The structural file contains a gate-level netlist. The behavioral file provides the high-level behavioral simulation model that can be used to accelerate simulations. The EDIF structural file is provided for reading the compiled function into the design system for integration with the rest of the design.

When a function containing a physical RAM is compiled, the detailed functional model with timing is included in another file. The RAM model file cannot be viewed from the GUI.

MCE Functions

5

This chapter provides an exhaustive listing of the functions available in Module Compiler Express. For each function, it provides the following information:

- A detailed description of the functionality
- The underlying function
- Input and output signals and their parameters
- Verilog/VHDL usage

Table 5-1 List of MCE Functions

Function	Description
ACC	Variable width accumulator
Abs	Absolute value
Add	2 input adder with optional carry input
Addsub	2 input Adder/Subtractor
Alu	Programmable ALU with up to 16 instructions
And	Variable width, variable number of input, AND tree
CompGE	Variable width greater than and equal to comparator
Compare	Variable width signed/unsigned magnitude comparator
CountUpDown	Variable width up/down counter
Count	Variable width up counter
DSPMAC	Multiplier-accumulator that can accumulate both positive and negative products
Decrement	Variable width decrementer
Divide	Variable width unsigned divider
Equal	Variable width equality comparison
FIR	N-point FIR (finite-impulse-response) filter
Increment	Variable width Incrementers
Inner	N-point inner product (sum of products)
LeadingZero	Leading 0's or 1's detector.
MAC	Multiplier-accumulator
MACCS	High-speed multiplier-accumulator
MultConst	Multiply a variable input by a constant
multiplier	Two input multiplier
Mux	Variable width, variable number of inputs multiplexor
Neg	Conditionally negate the input
Norm	Leading 0/1 normalization
nR1W_Latch	Latch based asynchronous RAM
nRmW_FF	Flip-Flop based synchronous RAM
Or	Variable width, variable number of input, OR tree
Rotate	Left/right signed/unsigned rotator
Sat	Saturate input to smaller output
Shift	Left/right signed/unsigned barrel shifter
ShiftLR	Left/right arithmetic/logical shifter
Square	Variable width Square (X*X)
Subtract	2 input subtractor
Sum	Variable width, variable number of input, sum tree
Xor	Variable width, variable number of input, XOR tree

Abs

This function computes the absolute value or magnitude of X to form Z. The width of X and Z is controlled by Width. The type of the adder is controlled by AdderType. The adder is always optimized for minimum possible delay.

Function

Z=abs(X) or Z = X<0 ? –X : X

Signals

Z	width-wide output
X	width-wide signed input

Parameters

Name	actual module name
Width	width of the X and Z signals
AdderType	type of incrementer to build (cla, fastcla, clsa, csa, ripple)

Verilog/VHDL Usage

Name(X,Z);

ACC

The ACC function accumulates an input X to form the result Z. The width of the X input can be controlled with the Xwidth parameter. The format (signed or unsigned) of X is controlled with Xsigned. The output Z and the accumulator have a width equal to Awidth. When rounding is selected (Round>0), only the bits of Z with indices from Round to Awidth–1 are valid. The accumulator is reset when R is LOW and CLK goes HIGH.

Function

$Z(n)=X(n-1) + (R(n-1)\ ?\ Z(n-1) : 1<<(\mathrm{Round}-1))$

Signals

X	Xwidth-wide input (Xsigned sets format)
Z	Awidth-wide output
R	synchronous 1-bit reset control, active LOW
CLK	positive edge-triggered clock

Parameters

Name	actual module name
Xwidth	width of input X
Awidth	width of the accumulator Z and output
Xsigned	format of X (signed when selected, unsigned otherwise)
Round	select the rounding position (0 for no rounding)
AdderType	type of final adder to use in building this function (cla, fastcla, clsa, csa, ripple)

Verilog/VHDL Usage

Name(X,Z,R,CLK);

Adder, Subtractor

These functions add or subtract two inputs, X and Y, to form the sum Z. The width of all three signals is controlled by Width. The type of the adder is controlled by AdderType. The adder is always optimized for minimum possible delay. The Adder function optionally allows a 1-bit carry input (CI) to added to X and Y.

Function

Z=X+Y (Adder, CarryIn not selected)

Z=X+Y+CI (Adder, CarryIn selected)

Z=X−Y (Subtractor)

Signals

Z	width-wide output
X,Y	width-wide inputs
CI	1-bit carry input (Adder with CarryIn selected)
CO	1-bit carry output (Adder with CarryOut selected)

Parameters

Name	actual module name
Width	width of the X,Y and Z signals
AdderType	type of final adder to use in building this function (cla, fastcla, clsa, csa, ripple)
CarryIn	compute X+Y+CI when selected, X+Y when deselected (Adder only)
CarryOut	generate carry output selected, X+Y when deselected (Adder only)

Verilog/VHDL Usage

Name(X,Y,Z);	(not Adder or CarryIn and CarryOut not selected)
Name(X,Y,CI,CO,Z);	(Adder, CarryIn and CarryOut selected)
Name(X,Y,CI,Z);	(Adder, CarryIn only selected)
Name(X,Y,CO,Z);	(Adder, CarryOut only selected)

Addsub

This function adds or subtracts two inputs, X and Y, to form Z. The input S controls whether addition or subtraction is performed. The width of all three signals is controlled by Width. The type of the adder is controlled by AdderType. The adder is always optimized for minimum possible delay.

Function

Z=X+Y (S=0)

Z=X-Y (S=1)

Signals

Z	width-wide output
X,Y	width-wide inputs
S	1-bit subtract control (1 for subtract, 0 for add)

Parameters

Name	actual module name
Width	width of the X,Y and Z signals
AdderType	type of final adder to use in building this function (cla, fastcla, clsa, csa, ripple)

Verilog/VHDL Usage

Name(X,Y,S,Z);

Alu

The Alu function provides a programmable ALU (arithmetic-logic unit) with up to 16 instructions. During normal operation, the A and B inputs are used to compute the output Z using INST as the number of the instruction to execute. Several flags are available for indicating special conditions.

Unneeded instructions and flag bits can be disabled individually during compilation to reduce the delay and area of the ALU. When an instruction or flag bit is disabled, the Z output or the flag bit, respectively, are undefined when the instruction is selected.

The arithmetic instructions (0-7) use an adder with a type specified by AdderType. The logical instructions (8-15) do not require the adder.

A four-quadrant division algorithm can be implemented serially with one quotient bit computed in each step. At the beginning of the operation, the dividend is supplied on the DI inputs (B is set to zero if DI>=0 or –1 if DI<0) and the divisor on the A input. After the first cycle, the output Z is fed back to the B input and the output DO is fed back to the DI output. After all iterations (Width cycles) are completed, Z is the remainder and DO is the quotient. Two final processing steps are required to complete the division. If the divisor is negative, the quotient must be negated. If the remainder is less than zero, the absolute value of the divisor must be added to the remainder.

Extended precision operation is supported for the IncA, DecA, Add, Sub, NegA, and NegB instructions. The data is processed from LSB to MSB, Width bits at a time. In the first cycle, the signal FirstCyc is set HIGH and the CI input is ignored. On subsequent cycles, FirstCyc is set LOW and the FLAG[1] output (carry out) is fed back to the CI input. For other operations or when using single precision operation, you can set FirstCyc to HIGH and CI is ignored. It is also possible to set FirstCyc LOW for all operations, in which case you must supply the correct carry input value on the CI input.

Three flags are available. When the OVF flag is set HIGH, it indicates that the 2's-complement overflow has occurred. If the inputs and outputs are unsigned, this bit has no meaning. The Carry flag is connected to the carry output of the adder. It is used to indicate overflow for unsigned operations and to provide the CI input value for extended precision operations. The OVF and Carry flags are undefined for logical instructions. To force these flags to zero, the outputs should be ANDed with ~INST[0]. The Zero flag is valid for all instructions and indicates that Z is zero.

Function (assuming no disabled functions or flag bits)

```
switch (INST) {
case 0 : Z=A+1
case 1 : Z=A-1
case 2 : Z=A+B
case 3 : Z=A-B
case 4 : Z=abs(A)
case 5 : Z=-A
case 6 : Z= (A[msb]==B[msb] ? (B-A) : B+A)<<1 (division iteration)
case 7 : Z=-B
case 8 : Z=A&B
case 9 : Z=A|B
case 10: Z=A^B
case 11: Z=~B
case 12: Z=A
case 13: Z=~A
case 14: Z=0
case 15: Z=1
}
FLAGS[0]= OVF (output has incorrect sign, 2’s-complement overflow)
FLAGS[1]= Carry (adder carry output, unsigned overflow)
FLAGS[2]= Zero (Z is zero)
FLAGS[3]= reserved
```

Signals

A,B	width-wide primary inputs
CI	1-bit-wide carry input signal, ignored if firstCyc is HIGH
Z	width-wide output
DI	dividend input (divide only)
DO	dividend output (divide only)
INST	4-bit-wide instruction input
FLAG	4-bit-wide condition code (flag) output
FirstCyc	HIGH during first cycle of extended precision operations. CI is ignored if FirstCyc is HIGH

Parameters

Name	actual module name
Width	width of the A,B and Z signals
AdderType	type of final adder to use in building this function (cla, fastcla, clsa, csa, ripple)
IncA	select to enable A+1 instruction
DecA	select to enable A–1 instruction
Add	select to enable A+B instruction
Sub	select to enable A–B instruction
Abs	select to enable abs(A) instruction
NegA	select to enable –A instruction
Divide	select to enable division instruction
NegB	select to enable –B instruction
And	elect to enable A|B instruction
XOR	select to enable A^B instruction
InvB	select to enable ~B instruction
PassA	select to enable A instruction
InvA	select to enable A instruction
Zero	select to enable the zero instruction
One	select to enable the one instruction
OvfFlag	select to enable the overflow flag output
CarryFlag	select to enable the carry flag output
ZeroFlag	select to enable the zero flag output

Verilog/VHDL Usage

Name(Z,A,B,DI,DO,CI,INST,FLAGS,FirstCyc);

And, Or, Sum, Xor

These functions perform logical AND, OR, or XOR or sum on two or more inputs, X0 to X*n*–1, to form the result Z. AND, OR, and XOR are bitwise computations across all inputs. The width of all signals is controlled by Width, except for Sum, which generates the full precision result. The number of inputs *n* is controlled by NumInputs. The logic is always optimized for minimum possible delay.

Function

Z=X0&X1& .. X*n*–1	(bitwise And)
Z=X0|X1| .. X*n*–1	(bitwise Or)
Z=X0+X1+ .. X*n*–1	(Sum)
Z=X0^X1^ .. X*n*–1	(bitwise Xor)

Signals

Z	width-wide output (for Sum the output has full precision width)
Xi	width-wide inputs (0 <= i < *n*)

Parameters

Name	actual module name
Signed	the format of the inputs (Sum only)
NumInputs	the number *n* of input signals (2-256)
Width	width of the signals (1-256)
AdderType	type of final adder to use in building the Sum function (cla, fastcla, clsa, csa, ripple)

Verilog/VHDL Usage

Name(X0, .. X*n*–1,Z);

Compare

The Compare function performs a magnitude comparison of two inputs, X and Y, to form the Boolean result Z. Z has a value of 1 when X>Y or X>=Y depending on X_GR_Y. The width of the input signals is controlled by Width. The comparator is always optimized for minimum possible delay.

Function

Z=(X>Y) (X_GR_Y selected)

Z=(X>=Y) (X_GR_Y deselected)

Signals

X,Y	width-wide inputs
Z	1-bit output (true if X>Y or X>=Y)

Parameters

Name	actual module name
Width	width of the X,Y and Z signals (1-256)
Signed	X and Y are signed when selected, unsigned otherwise
X_GR_Y	compute X>Y when selected, X>=Y otherwise

Verilog/VHDL Usage

Name(X,Y,Z);

CompGE

The CompGE function performs a numerical comparison of two inputs, X and Y, to form the 2-bit output, Z. If Z[0] is set, the two inputs are equal. If Z[1] is set, X is greater than Y.

Note: It is possible for two signals to have the same bit pattern but not be equal. For example,

```
wire signed [7:0] X = -1;

wire [7:0] Y = 255;
```

These two inputs have the same bit pattern, but they are not equal. The Equal bit of the output, Z[0], is 0.

The width of the X input signals is controlled by Xwidth. The width of the Y input signals is controlled by Ywidth.

When Xsigned is set, X input is a 2's complement number. When Ysigned is set, Y input is a 2's complement number.

CompGE is always optimized for minimum possible delay.

Function

Z[1]=(X>Y)

Z[0]=(X==Y)

Signals

X	Xwidth wide input (signed if Xsigned is selected)
Y	Ywidth wide input (signed if Ysigned is selected)
Z	2-bit output (Z[1] is true if X>Y, Z[0] is true if X==Y)

Parameters

Name	actual module name
Xwidth	width of the X signal (1–256)
Ywidth	width of the Y signal (1–256)
Xsigned	format of X (signed when selected, unsigned otherwise)
Ysigned	format of Y (signed when selected, unsigned otherwise)

Verilog/VHDL Usage

```
Name(X,Y,Z);
```

Counter, CountUpDown

These functions are synchronous up and up/down counters, respectively. Both are positive-edge triggered. The width of the counter is controlled by Width. The type of adder used in the Incrementers is controlled by AdderType. The adder is always optimized for minimum possible delay.

Counter is an up-only counter, with a 1-bit X input (count up 0 or 1). CountUpDown has a 2-bit signed X input to allow counting by -2, –1, 0, 1 Both counters can be held by setting the X input to zero and can be loaded to the value on the START bus by holding RESET LOW while CLK goes HIGH.

Function

$Z(n) = RESET(n-1)\ ?\ Z(n-1)+X(n-1) : START(n-1)$

Signals

Z	width-wide output
X	1-bit unsigned input (Counter), 2-bit signed input (CountUpDown)
RESET	1-bit synchronous reset input, active LOW
START	width-wide input, initial counter value
CLK	positive edge-triggered clock

Parameters

Name	actual module name
Width	width of the START and Z signals
AdderType	type of final adder to use in building this function (cla, fastcla, clsa, csa, ripple)

Verilog/VHDL Usage

Name (RESET,START,X,Z,CLK);

Decrement, Increment

These functions increment and decrement X by Y to form Z. The width of X and Z is controlled by Width. When Y is HIGH, the input is incremented (or decremented). When Y is LOW, the input is passed to the output. The type of the adder is controlled by AdderType. The adder is always optimized for minimum possible delay.

Function

Z=X+Y	(Increment)
Z=X-Y	(Decrement)

Signals

Z	width-wide output
X	width-wide input
Y	1-bit input (HIGH to increment/decrement)

Parameters

Name	actual module name
Width	width of the X,Y and Z signals
AdderType	type of final adder to use in building this function (cla, fastcla, clsa, csa, ripple)

Verilog/VHDL Usage

Name(X,Y,Z);

Divide

The Divide function divides dividend X by divisor Y, to get quotient Q and remainder R. All of X, Y, Q, and R are unsigned inputs/outputs. The width of the X and Y inputs can be controlled independently with the Xwidth and Ywidth parameters, respectively. Consequently, the width of Q and R are Xwidth and Ywidth, respectively.

Note that Q and R are not defined when divisor, Y, is 0. The equation, X = Q*Y+R, is true only when Arch=1, Round is not selected, and Y != 0. There are two architectures in Divide function controlled by “Arch.”

If Arch=1, a shift-subtract divider is produced, which is generally smaller than Arch=2. The remainder (R=X%Y) is only meaningful at Arch=1 and Round is not selected.

If Arch=2, X is multiplied by 1/Y. A ROM generates 1/Y; therefore, this architecture is good for small Ywidth and is generally faster than Arch=1. The type of the multiplier (Booth or non-Booth) is controlled by “Booth”. There is no remainder, R, for both rounding modes in this architecture.

For both architectures, there are two rounding modes for the quotient Q, controlled by “Round”. When Round is selected, Q is the nearest integer of quotient X/Y. When Round is not selected, the quotient, X/Y, is truncated to form Q.

The Divide desired delay is controlled via the Delay parameter. When Delay is greater than zero, a pipelined Divider is constructed; otherwise, an unpipelined Divider is constructed.

Function

Q = nint(X/Y) (if Round is selected)

: nint(x) which converts a float x into the nearest integer. The halfway cases are rounded up.

Q = floor(X/Y) (if Round is not selected)

: floor(x) returns the greatest integral value less than or equal to x.

This corresponds to IEEE rounding toward negative infinity.

R = X%Y (Exists only at Arch=1 and Round not selected)

Signals

X Xwidth wide input (unsigned only)

Y: Ywidth wide input (unsigned only)

Q Xwidth wide output (unsigned only)

R Ywidth wide output (unsigned only)

CLK: positive edge triggered clock (if pipelined)

Parameters

Name	actual module name
Xwidth	width of input
X Ywidth	width of input
Y Delay	select the cycle time in picoseconds (pipelined), or 0 for the fastest unpipelined design.
Arch	1 for architecture with smaller size, 2 for faster speed
Round	select for biased rounding, no rounding otherwise
Booth	select for a Booth multiplier, non-Booth otherwise (only for Arch=2)
AdderType	type of final adder to use in building this function (cla, fastcla, clsa, csa, ripple)

Verilog/VHDL Usage

Arch=1 and Round not selected:

Name(X,Y,Q,R); (not pipelined)

Name(X,Y,Q,R,CLK); (pipelined)

Otherwise:

Name(X,Y,Q); (not pipelined)

Name(X,Y,Q,CLK); (pipelined)

DSPMAC

The DSPMAC function is designed to be used in applications requiring a MAC that can accumulate both positive and negative products (for example, Z=C +/– X*Y). The Add input controls whether the positive or negative product is accumulated. Note that the accumulator and reset circuits must be supplied externally. The Xwidth and Ywidth parameters control width of the X and Y inputs. The XS and YS signal control the formats (signed or unsigned) of X and Y. The formats of X and Y can be changed dynamically to accommodate extended precision calculations. The multiplier is always Booth-encoded.

The Z output has the same precision as the C input. When rounding is selected (Round>0), only the bits of Z with indices from Round to Awidth–1 are valid.

Function

Z=C+X*Y (Add HIGH)

Z=C-X*Y (Add LOW)

Signals

X	Xwidth-wide input (XS sets format)
Y	Ywidth-wide input (YS sets format)
Z	Cwidth-wide output
C	Cwidth-wide accumulator input
Add	HIGH to add X*Y, LOW to subtract X*Y
XS	HIGH for X signed, LOW for unsigned
YS	HIGH for Y signed, LOW for unsigned

Parameters

Name	actual module name
Xwidth	width of input X
Ywidth	width of input Y
Cwidth	width of the accumulator input, output
AdderType	type of final adder to use in building this function (cla, fastcla, clsa, csa, ripple)
Round	select the rounding position (0 for no rounding)

Verilog/VHDL Usage

Name(C,X,Y,Add,XS,YS,Z);

Equal

The Equal function performs an equality comparison of two inputs, X and Y, to form the Boolean result Z. Z has a value of 1 when X==Y. The width of the input signals is controlled by Width. The comparator is always optimized for minimum possible delay.

Function

Z=(X==Y)

Signals

X,Y	width-wide inputs
Z	1-bit output (true if X==Y)

Parameters

Name	actual module name
Width	width of the X,Y and Z signals (1-256)

Verilog/VHDL Usage

Name(X,Y,Z);

FIR

The FIR function filters the data input X with the filter specified by the coefficients on the Yi inputs. The number of filter taps and multipliers is specified with the Taps parameter. The filter output is provided at Z. The width of the X and Y inputs can be controlled independently with the Xwidth and Ywidth parameters, respectively. The formats, signed or unsigned, of X and Y are also controlled independently with Xsigned and Ysigned, respectively. The Delay parameter controls the desired delay of the filter. When Delay is greater than zero, a pipelined filter is constructed; otherwise, an unpipelined filter is constructed.

The Z output is always a full precision value, with width equal to Zwidth. When rounding is selected (Round>0), only the bits of Z with indices from Round to Zwidth−1 are valid.

The type of the multipliers used (Booth or non-Booth) is controlled by Booth. Booth multipliers are most efficient when X and Y are large (greater than 8 bits) and when X and Y have an even number of bits and are signed.

Non-Booth multipliers are most efficient when X and Y are unsigned and/or the widths are small.

Function

$Z(n)=Y0*X(n)+Y1*X(n-1)+ \ldots\ YTaps-1*X(n\text{-}Taps-1) + 1<<(Round-1)$

Signals

X	Xwidth-wide data input (Xsigned sets format)
Yi	Ywidth-wide coefficient inputs (Ysigned sets format, 0<=i<Taps)
Z	Zwidth-wide output
CLK	positive edge-triggered clock

Parameters

Name	actual module name
Taps	number of taps
Xwidth	width of input X
Ywidth	width of input Y
Zwidth	width of the output, Z
Xsigned	format of X (signed when selected, unsigned otherwise)
Ysigned	format of all Yi inputs (signed when selected, unsigned otherwise)
Delay	select the pipelined cycle time in picoseconds, or 0 for unpipelined
Round	select the rounding position (0 for no rounding)
AdderType	type of final adder to use in building this function (cla, fastcla, clsa, csa, ripple)
Booth	select for a Booth multiplier, non-Booth otherwise

Verilog/VHDL Usage

Name(X,Y0, ... YTaps–1,Z,CLK);

Inner

The Inner function computes the inner product of the two input vectors, X and Y. The dimension of vectors and the number of multipliers is specified with the Taps parameter. The inner product result is provided at Z. The width of the X and Y inputs can be controlled independently with the Xwidth and Ywidth parameters, respectively. The formats (signed or unsigned) of X and Y are also controlled independently with Xsigned and Ysigned, respectively. The inner product desired delay is controlled via the Delay parameter. When Delay is greater than zero, a pipelined inner product is constructed; otherwise, an unpipelined inner product is constructed.

The Z output is always a full precision value, with width equal to Zwidth. When rounding is selected (Round>0), only the bits of Z with indices from Round to Zwidth–1 are valid.

The type of the multipliers used (Booth or non-Booth) is controlled by Booth. Booth multipliers are most efficient when X and Y are large (greater than 8 bits) and when X and Y have an even number of bits and are signed. Non-Booth multipliers are most efficient when X and Y are unsigned and/or the widths are small.

Function

$Z(n)=Y0*X0+Y1*X1+ \ldots YTaps-1*XTaps-1 + 1<<(Round-1)$

Signals

Xi	Xwidth-wide inputs (Xsigned sets format, 0<=i<Taps)
Yi	Ywidth-wide inputs (Ysigned sets format, 0<=i<Taps)
Z	Zwidth-wide output
CLK	positive edge triggered clock (if pipelined)

Parameters

Name	actual module name
Taps	number of taps
Xwidth	width of input X
Ywidth	width of input Y
Width	width of the output
Xsigned	format of all Xi inputs (signed when selected, unsigned otherwise)
Ysigned	format of all Yi inputs (signed when selected, unsigned otherwise)
Delay	select the pipelined cycle time in picoseconds, or 0 for unpipelined
Round	select the rounding position (0 for no rounding)
AdderType	type of final adder to use in building this function (cla, fastcla, clsa, csa, ripple)
Booth	select for Booth multipliers, non-Booth otherwise

Verilog/VHDL Usage

Name(X0, ... XTaps–1,Y0, ... YTaps–1,Z,CLK); (if pipelined)

Name(X0, ... XTaps–1,Y0, ... YTaps–1,Z); (not pipelined)

LeadingZero

The LeadingZero function counts leading zeros or leading ones from the input, X, to form the output, Z. Counting starts from the MSB of X.

The LSBs of Z indicates the number of leading zeros or ones, while the MSB indicates whether X is all zeros or all ones. Note that when the MSB of Z is set, the LSBs of Z are set to all ones for leading 0 detection but are set to all zeros for leading 1 detection.

LeadingZero uses a very fast architecture compared to Norm. Its behavior is similar to Norm, but it does not normalize the input as Norm does.

If the width of X is not a power of 2, X is left shifted to make its width a power of 2. Because the padding for left shift is 0, the behaviors of leading 0 and leading 1 detection are not symmetrical.

Following are a few examples showing the operation of the LeadingZero function. The width of output Z is ceil(log2(width(X)))+1.

Leading 0

```
Input X       Left shift    Output Z
-------------------------------------------------------
001100        00110000      0010
000000        00000000      1111
00000000      00000000      1111
```

Leading 1

```
Input X       Left shift    Output Z
-------------------------------------------------------
110011        11001100      0010
111111        11111100      0110
11111111      11111111      1000
```

Note that for the X=111111 case, the MSB of Z is not set because it is not all ones after the left shift.

Function

Z[log2(width(X))-1:0]	number of leading zeros or ones
Z[log2(width(X))]	all zeros or ones, depending on Leading0

Signals

Z	output width chosen automatically=ceil(log2(width(X)))+1
X	width-wide input

Parameters

Name	actual module name
Width	width of the X input
Leading0	counts leading zeros when selected, counts leading ones otherwise

Verilog/VHDL Usage

Name(X,Z);

MAC, MACCS

The MAC and MACCS functions multiply two inputs, X and Y, and sum the product with an accumulator to form the result Z. The width of the X and Y inputs can be controlled independently with the Xwidth and Ywidth parameters, respectively. The signed or unsigned formats of X and Y are also controlled independently with Xsigned and Ysigned, respectively. The type of the multiplier (Booth or non-Booth) is controlled by Booth. The multiplier desired delay is controlled using the Delay parameter. When Delay is greater than 0, a pipelined multiplier is constructed; otherwise, an unpipelined multiplier is constructed.

MAC provides a direct multiplier-accumulator in which the carry-propagate adder is inside the accumulator loop. The MACCS function moves the carry-propagate adder outside the accumulator loop. In general, MACCS will achieve higher clock frequencies with a greater area.

The Z output is always a full precision value, with width equal to Awidth. When rounding is selected (Round>0), only the bits of Z with indices from Round to Awidth−1 are valid.

Booth multipliers are most efficient when X and Y are large (greater than 8 bits) and when X and Y have an even number of bits and are signed. Non-Booth multipliers are most efficient when X and Y are unsigned and/or the widths are small.

Function

$Z(n)=X(n-1)*Y(n-1) + (R(n-1)\ ?\ Z(n-1) : 1<<(Round-1))$

Signals

X	Xwidth-wide input (Xsigned sets format)
Y	Ywidth-wide input (Ysigned sets format)
Z	Awidth-wide output
R	synchronous 1-bit reset control, active LOW
CLK	positive-edge triggered clock

Parameters

Name	actual module name
Xwidth	width of input X
Width	width of input Y
Awidth	width of the accumulator
Xsigned	format of X (signed when selected, unsigned otherwise)
Ysigned	format of Y (signed when selected, unsigned otherwise)
Delay	the pipelined cycle time in picoseconds, or 0 for unpipelined
AdderType	type of final adder to use in building this function (cla, fastcla, clsa, csa, ripple)
Booth	select for a Booth multiplier, non-Booth otherwise
Round	the rounding position (0 for no rounding)

Verilog/VHDL Usage

Name(X,Y,Z,R,CLK);

MultConst

The MultConst function multiplies a variable input, X by a constant Yval, to form the product Z. The width of the X input can be controlled with the Xwidth parameters. The format (signed or unsigned) of X is controlled with Xsigned. The multiplier desired delay is controlled via the Delay parameter. When Delay is greater than 0, a pipelined multiplier is constructed; otherwise, an unpipelined multiplier is constructed.

The Z output is always a full precision value, with width equal to Xwidth+width(Yval). When rounding is selected (Round>0), only the bits of Z with indices from Round to Xwidth+width(Yval)–1 are valid.

Function

Z=X*Yval

Signals

X	Xwidth wide input (Xsigned sets format)
Z	Xwidth+width(Yval)-wide output
CLK	positive edge triggered clock (if pipelined)

Parameters

Name	actual module name
Xwidth	width of input X
Yval	32-bit signed decimal constant value of Y (signed integer)
Xsigned	format of X (signed when selected, unsigned otherwise)
Delay	select the pipelined cycle time in picoseconds, or 0 for unpipelined
Round	select the rounding position (0 for no rounding)
AdderType	type of final adder to use in building this function (cla, fastcla, clsa, csa, ripple)

Verilog/VHDL Usage

Name(X,Z); (not pipelined)

Name(X,Z,CLK); (pipelined)

Multiplier

The Multiplier function multiplies two inputs, X and Y, to form the product Z. The width of the X and Y inputs can be controlled independently with the Xwidth and Ywidth parameters, respectively. The formats (signed or unsigned) of X and Y are also controlled independently with Xsigned and Ysigned, respectively. The Delay parameter controls the desired delay of the multiplier. When Delay is greater than zero, a pipelined multiplier is constructed; otherwise, an unpipelined multiplier is constructed.

The Z output is a full precision value with width equal to Xwidth+Ywidth. When Round is greater than zero, only the bits of Z with indexes from Round to Xwidth+Ywidth−1 are valid.

The type of the multiplier (Booth or non-Booth) is controlled by Booth. Booth multipliers are most efficient when X and Y are large (greater than 8 bits) and when X and Y have an even number of bits and are signed. Non-Booth multipliers are most efficient when X and Y are unsigned and/or the widths are small.

Function

Z=X*Y

Signals

X	Xwidth-wide input (Xsigned sets format)
Y	Ywidth-wide input (Ysigned sets format)
Z	Xwidth+Ywidth-wide output
CLK	positive-edge triggered clock (if pipelined)

Parameters

Name	actual module name
Xwidth	width of input X
Ywidth	width of input Y
Xsigned	format of X (signed when selected, unsigned otherwise)
Ysigned	format of Y (signed when selected, unsigned otherwise)
Delay	the pipelined cycle time in picoseconds, or 0 for unpipelined
AdderType	type of final adder to use in building this function (cla, fastcla, clsa, csa, ripple)
Booth	select for a Booth multiplier, non-Booth otherwise
Round	the rounding position (0 for no rounding)

Verilog/VHDL Usage

Name(X,Y,Z);	(not pipelined)
Name(X,Y,Z,CLK);	(pipelined)

Mux

The Mux function selects one input from the set of two or more inputs, X0 to X*n*–1 to form the result Z. The width of all data input and output signals is controlled by Width. MC chooses the width of the select input S automatically. The number of inputs *n* is controlled by NumInputs. MuxType selects the multiplexor type. The logic is always optimized for minimum possible delay.

Function

Z=S ? X*n*–1 : ... : X1 : X0

Signals

Z	width-wide output
S	selects input (width chosen automatically to allow full selection)
Xi	width-wide inputs (0 <= i < *n*)

Parameters

Name	actual module name
NumInputs	the number of input signals (2-256)
Width	width of the signals (1-256)
MuxType	the architecture (mux, and/or, tristate)

Verilog/VHDL Usage

Name(X0, .. X*n*–1,S,Z);

Neg

This function conditionally negates the input X based on the control signal S to form Z. The width of X and Z is controlled by Width. The type of the adder is controlled by AdderType. The adder is always optimized for minimum possible delay.

Function

Z= S ? −X : X

Signals

Z	width-wide output
X	width-wide signed input
S	1-bit-wide negation control input (HIGH for −X)

Parameters

Name	actual module name
Width	width of the X,Y and Z signals
AdderType	type of final adder to use in building this function (cla, fastcla, clsa, csa, ripple)

Verilog/VHDL Usage

Name(X,S,Z);

Norm

The Norm function removes leading zeros or ones from the input X to form the mantissa Z and exponent E. E indicates the number of leading 0s or 1s removed from the input. If the input is zero, then E has its maximum value. The width of X and Z is controlled by Width. The width of E is chosen automatically to allow full normalization of X. The logic is always optimized for minimum possible delay.

See also LeadingZero for a faster detector for leading ones and zeros.

Function

Z=mantissa(X)

E=exp(X)

Signals

Z	width-wide mantissa output
E	exponent output (width chosen automatically=ceil(log2(width(X))))
X	width-wide input

Parameters

Name	actual module name
Width	width of the X and Z signals
Leading0	removes leading zeros when selected, remove leading ones otherwise

Verilog/VHDL Usage

Name(X,E,Z);

nR1W_Latch

These functions provide netlist-based low-power asynchronous memories (RAMs) with 1 to 8 ports. All RAMs have only one write port and have a behavior very similar to the NR1W_RAM memories, with the exception that CSN does not directly affect the output.

The write port is always controlled by the highest numbered address (or A for 1-port).

> ***Note: Note: the delay reported by MC is the longest path through the register file. In general, this is a write through delay. For more meaningful delay information, choose Design Report from the View menu and look for the estimated timing parameters.***

Function

Asynchronous RAM

Signals

DOi	bits-wide data outputs for port i
DIi	bits-wide data input for port i
Ai	address input for port i
WEN	1-bit write enable (LOW to write)
CSN	1-bit chip select (LOW to enable, HIGH to disable)

Parameters

Name	actual module name
Ports	the number of total ports (1-8)
Words	the number of words (2-128)
Bits	the number of bits (1-128)

Verilog/VHDL Usage

Name(DI,A,WEN,CSN,DO); (1-port)

Name(DI*n*,A1,..,A*n*,WEN,CSN,DO1,..,DO*n*–1); (2 or more ports)

nRmW_FF

These functions provide netlist-based low-power synchronous memories (RAMs) with 1 to 8 read ports and 1 to 3 write ports. The number of words and bits for each RAM can be specified. To resolve write conflicts (more than one port writing to the same address), the write ports are each assigned a priority that decreases with the port number. The port with the highest priority attempting to write into the memory succeeds. All lower-priority ports writing to the same address are ignored.

All write operations are synchronous, with CLK rising edge. All read operations are asynchronous. To make read operations synchronous, the read address can be latched with a flip-flop.

Choose Design Report from the View menu and look in the timing parameters section to see some of the timing parameters for the RAM.

Function

Multi-write port synchronous RAM (See the *Memory Architect User Manual* for details.)

Signals

DOi	bits-wide data outputs for port i
DIi	bits-wide data input for port i
Ai	address input for port i
WENi	1-bit write enable (LOW to write)
CSN	1-bit chip select (LOW to enable, HIGH to disable)

Parameters

Name	actual module name
PortsW	the number of write ports, *n* (1-3)
PortsP	the number of read ports, *m* (1-8)
Words	the number of words (2-128)
Bits	the number of bits (1-128)

Verilog/VHDL Usage

Name(DI*m*+1,..,DIn+*m*,A1,...,An+*m*,WEN*m*+1,...,WEN*m*+*n*,CSN,DO1,..DO*m*,CLK);

Rotate, Shift

These functions shift or rotate the input X to form the result Z. The input S specifies the number of bits to shift or rotate X. The width of X and Z is controlled by Width. The width of S is chosen automatically to allow full shifting or rotating of X (width[S]=ceil[log2(width[X])]). The logic is always optimized for minimum possible delay.

Function

Z=X<<S (Shift, Left selected)

Z=X>>S (Shift, Left deselected)

Z=X<<<S (Rotate, Left selected)

Z=X>>>S (Rotate, Left deselected)

Signals

Z	width-wide mantissa output
S	shift input (width chosen automatically)
X	width-wide input

Parameters

Name	actual module name
Width	width of the X and Z signals
Signed	X is signed if selected, unsigned otherwise
Left	X is shifted or rotated left if selected, right otherwise

Verilog/VHDL Usage

Name(X,S,Z);

Sat

The Sat function clips the input X to the nearest value allowed in the output Z. The width and format of each is controlled by the Xwidth, Zwidth, Xsigned, and Zsigned parameters. The width of Z must be less than the width of X.

Function

Z= X<Zmin ? Zmin : (X>Zmax ? Zmax :X)

Zsigned	Zmin	Zmax
1	-2^(Zwidth–1)	2^(Zwidth-1)-1
0	0	2^Zwidth-1

Signals

Z	Zwidth-wide output (signed if Zsigned is selected)
X	Xwidth-wide input (signed if Xsigned is selected)

Parameters

Name	actual module name
Xwidth	width of the X (2-256)
Xsigned	X is signed if selected, unsigned otherwise
Zwidth	width of the Z (must be less than Xwidth)
Zsigned	Z is signed if selected, unsigned otherwise

Verilog/VHDL Usage

Name(X,Z);

ShiftLR

The ShiftLR function shifts the input X to form the result Z. The type of shift performed is determined by the LEFT (left or right) and LOG (arithmetic or logical) inputs. Arithmetic right shifts are sign extended while logical shifts are zero extended. The input S specifies the number of bits to shift X. The width of X and Z is controlled by Width. The width of S is chosen automatically to allow full shifting of X (width(S)=ceil(log2(width(X)))). The logic is always optimized for minimum possible delay.

Function

```
Z=LEFT ? X<<S : X>>S;
```

Signals

Z	width-wide output
S	shift input (width chosen automatically)
X	width-wide input
LEFT	1-bit direction input (TRUE for left)
LOG	1-bit mode input (TRUE for logical shift, FALSE for arithmetic shift)

Parameters

Name	actual module name
Width	width of the X and Z signals

Verilog/VHDL Usage

```
Name(X,S,Z,LEFT,LOG);
```

Square

The Square function squares the input, X, to form the product Z. The width of the X input can be controlled with the Xwidth parameter. The format (signed or unsigned) of X is controlled with Xsigned. The desired delay is controlled via the Delay parameter. When Delay is greater than zero, a pipelined squaring circuit is constructed; otherwise, it is unpipelined.

The Z output is always a full precision value, with width equal to 2*Xwidth. When rounding is selected (Round>0), only the bits of Z with indices from Round to 2*Xwidth–1 are valid.

Function

Z=X*X

Signals

X	Xwidth-wide input (Xsigned sets format)
Z	2*Xwidth wide output
CLK	positive edge triggered clock (if pipelined)

Parameters

Name	actual module name
Xwidth	width of input X
Xsigned	format of X (signed when selected, unsigned otherwise)
Delay	select the cycle time in picoseconds (pipelined), or 0 for unpipelined
AdderType	type of final adder to use in building this function (cla, fastcla, clsa, csa, ripple)
Round	select the rounding position (0 for no rounding)

Verilog/VHDL Usage

Name(X,Z); (not pipelined)

Name(X,Z,CLK); (pipelined)

Arithmetic Options

A

This appendix discusses multiplier and adder structures.

Multiplication

There are two options for multiplier implementation available in the multiplication-based functions such as Multiplier, MAC, MACCS, and FIR. If you select the Booth option, MCE constructs a Booth multiplier. Otherwise, the multiplier is non-Booth.

The non-Booth-encoded multiplier generates addends using only simple logic. This type of multiplier generates N partial products of M bits each, where N is the width of the Y input and M is the width of the X input. The non-Booth multiplier is most efficient when N and M are small numbers.

The Booth-encoded multiplier utilizes special library cells to encode the Y inputs and to generate partial products. The number and width of the partial products (PP) are summarized in Table A-0.

Table A-0 Partial Products Produced by Booth-encoded Multipliers

Y input with width N	Num PP
Signed-even N	N/2
Signed-odd N	(N+1)/2
Unsigned-even N	N/2+1
Unsigned-odd N	(N+1)/2

X input with width M	Width PP
Signed	M+1
Unsigned	M+2

Booth multipliers are most efficient for signed X and Y and especially so for signed and even Y. These multipliers are not as efficient for narrow and/or unsigned X or Y.

The Adders

For many addition-based functions such as Adder, Sum, and Increment, you can select one of five different adder architectures for the AdderType parameter. Each has its own trade-off between area and delay, so you should choose an architecture that is appropriate for the circuit constraints. MCE chooses the adder architecture automatically for the multiplier-based circuits. Table A-0 shows the basic characteristics of each adder type for a given input width, *n*.

Table A-0 Characteristics of Adder Types

Type	Description	Area	Delay
csa	carry-select	$O(n)$	$O(\sqrt{n})$
cla	carry-lookahead	$O(n)$	$O(2 \log_2(n))$
fastcla	fastcarry-lookahead	$O(n \log_2(n))$	$O(\log_2(n))$
clsa	carry-look ahead-select	Variable ripple-> fastcla	Variable ripple-> fastcla
ripple	ripple	$O(n)$	$O(n)$

When you are optimizing for speed, the **clsa** adder is similar to the **fastcla** adder. The only difference is that **clsa** adders use fewer and larger cells than **fastcla** adders.

Error and Warning Messages

B

There are only a few error conditions that can be generated within Module Compiler Express. Most error conditions indicate to input parameters that are out of range or an incorrect setup. Occasionally, the synthesis and optimization routines generate errors.

Parameter Errors

`<param name> is out of range (<lower> to <upper>)`

The specified parameter is outside of the legal range shown. Change the parameter to be within the range.

`Illegal delay value (<delay>)`

The delay value provided is illegal because it is less than zero. Provide zero or a positive value.

`Delay value too small for pipelining (<delay>). Min value is <min delay>`

Non-zero positive delay values correspond to pipelined circuits. Module Compiler Express enforces a minimum value of cycle time to prevent excessive pipelining. Make sure the delay value is greater than the minimum value given. Also note that delay values are given in units of picoseconds.

`Illegal mux type (<type>). Use ’mux’, ’andor’ or ’tristate’`

The MUX type provided is not legal. Use one of the three legal types.

`Illegal Function ...`

The function specified is not legal. Choose a legal function (command line mode only).

`Illegal Optimization value ...`

The optimization value must be an integer between 0 and 2 (command line mode only).

`Illegal adder type (<type>). Use ’fastcla’, ’cla’, ’clsa’ ’csa’ or ’ripple’.`

The adder type provided is not legal. Only the five types shown in the message are allowed. A table indicating the different attributes of the types is provided in Appendix A.