



TECHNICAL REPORT	EIA-IS-103™	REV. A
	Issued 1996-10 Stabilized 2015-07	
Library of Parameterized Modules (LPM) Version 210		

NOTICE

This document has been taken directly from the original TechAmerica document and contains only minor editorial and format changes required to bring it into conformance with the publishing requirements of SAE Technical Standards. The release of this document is intended to replace the original with the SAE International document. Any numbers established by the original document remain unchanged.

The original document was adopted as an SAE publication under the provisions of the SAE Technical Standards Board (TSB) Rules and Regulations (TSB 001) pertaining to accelerated adoption of specifications and standards. TSB rules provide for (a) the publication of portions of unrevised specifications and standards without consensus voting at the SAE committee level, and (b) the use of the existing specification or standard format.

SAE Technical Standards Board Rules provide that: "This report is published by SAE to advance the state of technical and engineering sciences. The use of this report is entirely voluntary, and its applicability and suitability for any particular use, including any patent infringement arising therefrom, is the sole responsibility of the user."

SAE reviews each technical report at least every five years at which time it may be revised, reaffirmed, stabilized, or cancelled. SAE invites your written comments and suggestions.

Copyright © 2016 SAE International

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of SAE.

TO PLACE A DOCUMENT ORDER: Tel: 877-606-7323 (inside USA and Canada)
Tel: +1 724-776-4970 (outside USA)
Fax: 724-776-0790
Email: CustomerService@sae.org
<http://www.sae.org>

SAE WEB ADDRESS:

Full document details: [SAE EIAIS103A](#)

SAE values your input. To provide feedback on this Technical Report, please visit <http://www.sae.org/technical/standards/EIAIS103A>

Documents

Document Sharing Hub



Electronic Design Interchange Format
A DIVISION OF ELECTRONIC INDUSTRIES ASSOCIATION

Library of
Parameterized Modules
(LPM)
Version 2 1 0

Contents

1. INTRODUCTION TO LPM.....	1-1
1.1 ABOUT THIS DOCUMENT	1-1
1.2 BACKGROUND OF LPM	1-1
1.3 OBJECTIVES OF LPM	1-1
1.4 LPM DESIGN FLOW	1-3
1.5 CHOICE OF LIBRARY ELEMENTS	1-4
1.6 TIMING DRIVEN SYNTHESIS FOR LPM	1-4
1.7 SIMPLIFIED SUPPORT OF LPM	1-5
2. FORMAT OF LPM MODULES.....	2-1
2.1 PARAMETERIZATION OF MODULES	2-1
2.1.1 Parameterizing Size	2-1
2.1.2 Parameterizing Functionality	2-1
2.1.2.1 No Package Pin Placement Information	2-1
2.2 DESCRIPTION OF PROPERTIES	2-1
2.2.1 Legal Values for Properties	2-2
2.2.1.1 Legal Values for LPM_HINT Property	2-2
2.2.2 Universal Properties	2-2
2.2.2.1 Signal Polarities (LPM_Polarity Property)	2-2
2.2.2.2 Basic Module type (LPM_TYPE Property)	2-2
2.2.2.3 Implementation Suggestions (LPM_Hint Property)	2-2
2.3 DATA REPRESENTATION IN MODULES	2-2
2.3.1 Bit-Ordering.....	2-2
2.3.2 Arithmetic.....	2-3
2.3.3 LPM Values	2-3
2.3.3.1 Integer	2-3
2.3.3.2 String	2-4
2.3.3.3 Don't Care.....	2-5
3. DESIGN FOR TESTABILITY IN LPM	3-1
3.1 REQUIREMENTS	3-1
3.2 TESTING PHILOSOPHY	3-1
3.2.1 How LSSD works	3-1
3.3 LPM IMPLEMENTATION OF LSSD	3-2
3.4 TEST OPTIONS	3-2
3.5 USAGE: CLOCKING AND PRECEDENCE	3-2
4. DESCRIPTION OF LPM MODULES	4-1
4.1 MODULE CATEGORIES.....	4-1
4.1.1 Logic Conventions.....	4-2
4.1.2 Drawing Conventions	4-3
4.1.3 Scan Test Conventions.....	4-3
4.2 GATES.....	4-4
4.2.1 LPM_CONSTANT.....	4-4
4.2.1.1 Ports	4-4
4.2.1.2 Properties	4-4
4.2.1.3 Function	4-4
4.2.2 LPM_INV	4-5

4.2.2.1 Ports	4-5
4.2.2.2 Properties	4-5
4.2.2.3 Function	4-5
4.2.3 LPM_AND	4-6
4.2.3.1 Ports	4-6
4.2.3.2 Properties	4-6
4.2.3.3 Function	4-6
4.2.3.4 Example	4-6
4.2.4 LPM_OR	4-8
4.2.4.1 Ports	4-8
4.2.4.2 Properties	4-8
4.2.4.3 Function	4-8
4.2.4.4 Example	4-9
4.2.5 LPM_XOR	4-10
4.2.5.1 Ports	4-10
4.2.5.2 Properties	4-10
4.2.5.3 Function	4-10
4.2.5.4 Example	4-11
4.2.6 LPM_BUSTRI	4-12
4.2.6.1 Ports	4-12
4.2.6.2 Properties	4-12
4.2.6.3 Functions	4-13
4.2.7 LPM_MUX	4-14
4.2.7.1 Ports	4-14
4.2.7.2 Properties	4-15
4.2.7.3 Functions	4-15
4.2.7.4 Example	4-15
4.2.8 LPM_DECODE	4-17
4.2.8.1 Ports	4-17
4.2.8.2 Properties	4-17
4.2.8.3 Functions	4-18
4.2.9 LPM_CLSHIFT	4-19
4.2.9.1 Ports	4-19
4.2.9.2 Properties	4-20
4.2.9.3 Functions	4-20
4.3 ARITHMETIC COMPONENTS	4-21
4.3.1 LPM_ADD_SUB	4-21
4.3.1.1 Ports	4-21
4.3.1.2 Properties	4-22
4.3.1.3 Functions	4-23
4.3.2 LPM_COMPARE	4-24
4.3.2.1 Ports	4-24
4.3.2.2 Properties	4-24
4.3.2.3 Functions	4-25
4.3.3 LPM_MULT	4-26
4.3.3.1 Ports	4-26
4.3.3.2 Properties	4-27
4.3.3.3 Function	4-27
4.3.3.4 Example	4-28
4.3.4 LPM_ABS	4-29
4.3.4.1 Ports	4-29
4.3.4.2 Properties	4-29
4.3.4.3 Function	4-29
4.3.5 LPM_COUNTER	4-30
4.3.5.1 Ports	4-30
4.3.5.2 Properties	4-32
4.3.5.3 Functions	4-33
4.4 STORAGE COMPONENTS	4-34
4.4.1 LPM_LATCH	4-34

4.4.1.1 Ports	4-34
4.4.1.2 Properties	4-35
4.4.1.3 Functions	4-35
4.4.2 LPM_FF	4-36
4.4.2.1 Ports	4-36
4.4.2.2 Properties	4-37
4.4.2.3 Functions	4-38
4.4.3 LPM_SHIFTREG	4-39
4.4.3.1 Ports	4-39
4.4.3.2 Properties	4-40
4.4.3.3 Functions	4-40
4.4.4 LPM_RAM_DQ	4-42
4.4.4.1 Ports	4-42
4.4.4.2 Properties	4-43
4.4.4.3 Functions	4-43
4.4.5 LPM_RAM_IO	4-45
4.4.5.1 Ports	4-45
4.4.5.2 Properties	4-46
4.4.5.3 Functions	4-47
4.4.6 LPM_ROM	4-49
4.4.6.1 Ports	4-49
4.4.6.2 Properties	4-49
4.4.6.3 Functions	4-50
4.4.6.4 ROM Contents	4-51
4.5 TABLE PRIMITIVES	4-54
4.5.1 TABLE FORMATS	4-54
4.5.2 LPM_TTABLE	4-56
4.5.2.1 Ports	4-56
4.5.2.2 Properties	4-56
4.5.2.3 Function	4-56
4.5.2.4 Example	4-57
4.5.3 LPM_FSM	4-58
4.5.3.1 Ports	4-58
4.5.3.2 Properties	4-59
4.5.3.3 Functions	4-59
4.5.3.4 Example	4-60
4.6 PAD PRIMITIVES	4-62
4.6.1 LPM_INPAD	4-62
4.6.1.1 Ports	4-62
4.6.1.2 Properties	4-62
4.6.1.3 Function	4-62
4.6.2 LPM_OUTPAD	4-63
4.6.2.1 Ports	4-63
4.6.2.2 Properties	4-63
4.6.2.3 Function	4-63
4.6.3 LPM_BIPAD	4-64
4.6.3.1 Ports	4-64
4.6.3.2 Properties	4-64
4.6.3.3 Function	4-64
5. INSTANTIATING LPM IN EDIF	5-1
5.1 EDIF VERSIONS	5-1
5.2 MODULE DECLARATION	5-1
5.2.1 CELL	5-1
5.2.2 PORTS	5-2
5.2.2.1 Single Ports	5-2
5.2.2.2 Bus Port Names	5-2
5.2.3 PROPERTIES	5-3

5.2.3.1 LPM_TYPE Property	5-3
5.2.3.2 LPM_Polarity Property	5-4
5.2.3.3 LPM_Hint Property	5-4
5.3 EXAMPLE	5-5
5.3.1 Schematic Description	5-5
5.3.2 EDIF 2 0 0 Description	5-5
5.3.3 EDIF 3 0 0 Description	5-9
6. INSTANTIATING LPM IN VHDL	6-1
6.1 COMPONENT DECLARATION	6-1
6.2 COMPONENT INSTANTIATION	6-1
6.2.1 Port	6-1
6.2.2 Property	6-2
6.3 EXAMPLE	6-2
6.3.1 Schematic description	6-2
6.3.2 VHDL description	6-2
7. INSTANTIATING LPM IN VERILOG	7-1
7.1 MODULE DECLARATION	7-1
7.2 MODULE INSTANTIATION	7-1
7.2.1 Port	7-1
7.2.2 Property	7-2
7.3 EXAMPLE	7-2
7.3.1 Schematic description	7-2
7.3.2 Verilog description	7-2
8. SUPPORTING FILES	8-1
8.1 MEMORY INITIALIZATION FILES	8-1
8.2 TRUTH TABLE AND FINITE STATE MACHINE FILES	8-1
8.3 LPM VHDL PACKAGE FILE	8-1
8.4 LPM VERILOG LIBRARY FILE	8-9
8.5 SIMPLIFIED SUPPORT THROUGH THE EXCEPTION LIST	8-17
9. TIMING DRIVEN SYNTHESIS QUERY/RESPONSE FOR LPM	9-1
9.1 RESOURCE ACCOUNTING	9-1
9.2 INVOCATION OF FITTER	9-2
9.3 CONSTRAINTS QUERY REQUESTS	9-2
9.3.1 TPD Property	9-3
9.3.2 THOLD Property	9-3
9.3.3 TSETUP Property	9-4
9.3.4 TPERIOD Property	9-4
9.3.5 Resources	9-4
9.3.6 Iterations	9-4
9.3.7 Query Example	9-4
9.4 RESPONSE FILE	9-7
9.5 ESTIMATOR/FITTER TOOL INVOCATION	9-8
10. FUTURE DIRECTIONS	10-1
11. BNF OF SUPPORTING FILES	11-1
11.1 BNF DESCRIPTION OF QUERY FILE	11-1
11.2 BNF DESCRIPTION OF RESPONSE FILE	11-2
11.3 BNF DESCRIPTION OF EXCEPTION LIST FILE	11-3
11.4 HEX OBJECT FILE SPECIFICATION	11-4
11.4.1 BNF Description of the Hex File	11-4

11.4.2 Semantics of Hex File.....	11-4
11.5 BNF DESCRIPTION OF TRUTH TABLE FILE	11-5
11.6 BNF DESCRIPTION OF FSM FORMAT	11-6
12. INDEX.....	12-1

1. INTRODUCTION TO LPM

This introduction provides an overview of the Library of Parameterized Modules (LPM), including an insight into the history and motivations for developing LPM as well as a description of the functionality and benefits of LPM.

1.1 ABOUT THIS DOCUMENT

This document is the authoritative reference for the Library of Parameterized Modules (LPM) standard version 2 1 0. This specification defines the semantics of each module along with a complete description of the module's functionality as well as the syntax for instantiating LPM modules in an EDIF netlist.

The intended audience for this document is CAE vendors, silicon vendors, and only secondarily, logic designers. It is assumed that the reader is familiar with logic design, design automation tools and silicon implementation options. What the reader should get out of this document is an understanding of the benefits of the LPM standard as well as the details required to implement an LPM interface.

1.2 BACKGROUND OF LPM

The LPM standard was proposed in 1990 as a means to enable efficient mapping of digital designs into divergent technologies such as programmable logic, Gate Arrays, and Standard Cells. The standard was accepted as the Electronic Industries Association (EIA) Interim standard number IS-103 in April 1993 as an adjunct standard to the Electronic Design Interface Format (EDIF, EIA-548-A).

EDIF is the preferred method for transferring designs between the tools of different EDA vendors and from the EDA tools to the Integrated Circuit (IC) vendors. EDIF describes the syntax that represents a logical netlist, and LPM adds a set of functions that describe the logical operation of the netlist. Before LPM, each EDIF netlist would typically contain technology-specific logic functions, making technology-independent design impossible.

Although LPM is an adjunct standard to EDIF, it is compatible with any text or graphic design entry tool. In particular, LPM is a welcome addition to Verilog HDL or VHDL designs.

1.3 OBJECTIVES OF LPM

The primary objective of LPM is to enable technology-independent design, without sacrificing efficiency. By using LPM, the designer is freed from deciding the target technology until late in the design flow. All design entry and simulation tools remain technology-independent and rely

on the synthesis or fitting tools to efficiently map the design to various technologies. Efficiency is guaranteed because the technology mapping is defined by the technology vendors.

To be effective, LPM meets the following key criteria:

- **Allow Technology-Independent Design Entry**

The primary goal of LPM is to enable technology-independent design. Designers can work with the LPM modules during design entry and verification without specifying the target technology.

- **Allow Efficient Design Mapping**

Technology-independent design historically meant inefficient design. LPM allows designers to use technology-independent design without sacrificing efficiency. The technology mapping of LPM modules is specified by the technology-vendor, so that the most optimum solutions are guaranteed.

- **Allow Tool-Independent Design Entry**

Designers require the ability to migrate a design from one EDA vendor's tool to another. Many designers, for example, use one vendor for logic synthesis and another vendor for logic simulation. LPM enables designers to migrate designs between EDA vendors while maintaining a high-level logic description of the functions.

- **Allow specification of a complete design**

The LPM set of modules can completely specify the digital logic for any design. Any function that is not included in the initial set of modules, can be created out of the modules.

- **Allow separation of logical design from physical design**

Because physical design is currently tied very closely to the implementation technology, the standard cannot contain physical information and remain technology independent. The LPM standard describes only the logical content of a design.

- **Allow timing-driven synthesis**

Because physical design is currently tied very closely to the implementation technology, the standard cannot contain physical information and remain technology independent. The LPM standard describes only the logical content of a design and the hints (loose constraints) needed to insure proper timing.

1.4 LPM DESIGN FLOW

LPM fits into any standard design flow used for designing programmable logic, Gate Arrays, or Standard Cells. The library works equally well with HDLs (Verilog HDL or VHDL), schematics, or block diagrams, and can be used during functional or pre-route simulation.

Figure 1 shows how LPM fits into a standard design flow while providing technology-independent design entry. Each EDA supplier will provide the symbol and functional library. For the technology mapping phase, each technology vendor will provide optimized implementations of each LPM function. These optimized functions can be used to by any EDA vendor to map to any technology.

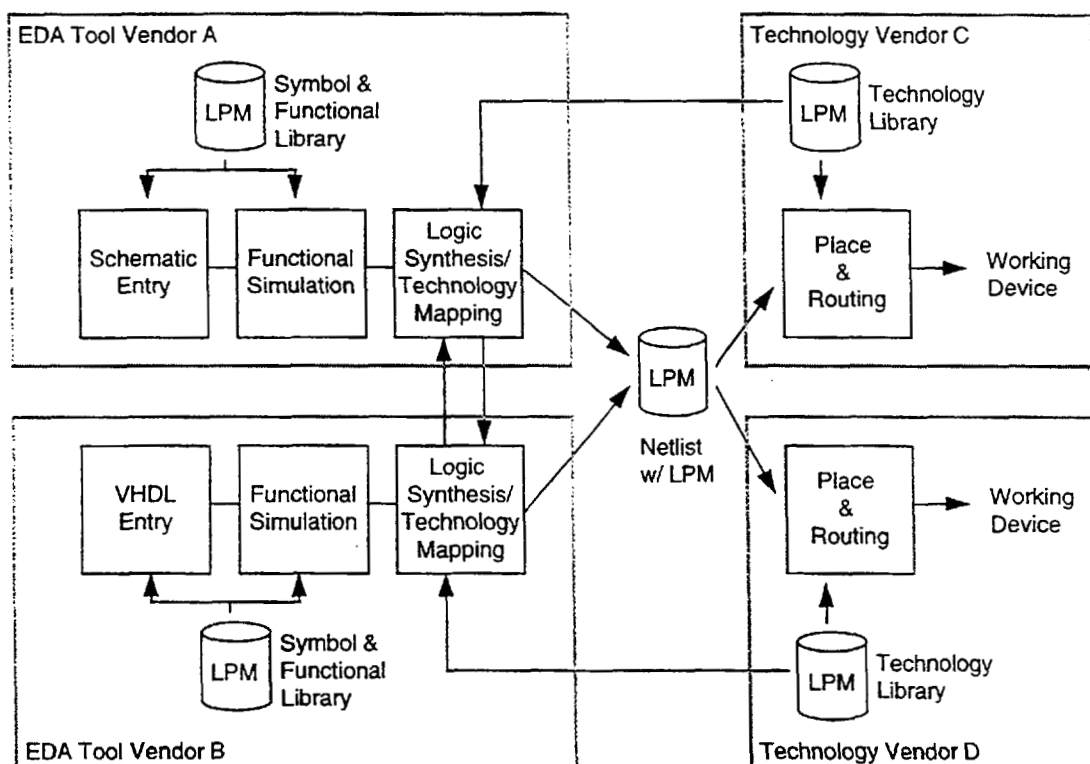


Figure 1 LPM Design Flow

1.5 CHOICE OF LIBRARY ELEMENTS

The LPM library is chosen to provide a complete yet compact set of digital design building blocks. There is no unique way to specify such a set so the following criteria were used to *guide* the establishment of the set of modules for LPM:

- **Completeness**
The LPM standard supplies a complete set of logic functions. Any Boolean function can be implemented using the primitives.
- **Orthogonality**
Frequently, functions can be implemented in a number of different ways. The LPM modules try to reduce such duplication. For example, both synchronous and asynchronous types of RAM can be implemented by the LPM_RAM_DQ and LPM_RAM_IO modules. This goal was loosely followed so as not to conflict with the next two criteria.
- **Popularity**
In order to keep the standard to a reasonable number of modules, only the most common functions are defined as separate modules.
- **Difficulty of De-Compiling**
If a logic function is difficult to recognize once it has been decomposed into low level gates, then it is a candidate for inclusion.
- **Representation**
The representation of data processed by the LPM modules is an attribute of each module. A minimum set of representations was chosen to be expressive and cover the bulk of existing designs.

1.6 TIMING DRIVEN SYNTHESIS FOR LPM

The LPM library offers significant circuit area and timing improvements by allowing custom gate-level implementations of high-level design building blocks. LPM also works well with logic optimization algorithms, including timing-driven synthesis to achieve additional improvements. Logic optimization requires timing and area information about LPM modules to select the optimal gate-level implementation for each LPM module and to correctly optimize the surrounding logic.

The logic optimizer has two options for finding this information. The first option would allow the optimizer to build a netlist from a vendor specific set of primitives for each module. The logic optimizer could calculate the delay and area of the module from this netlist. The second option would allow the optimizer to query the fitter for an estimation of the speed and area of the LPM module.

1.7 SIMPLIFIED SUPPORT OF LPM

The LPM library contains a complete yet compact set of modules that can be used to describe any digital design. However, a silicon vendor is not required to support every module nor every feature of each module to use LPM. Other libraries or design formats can be used in conjunction with LPM to provide a complete design flow. A silicon vendor may choose to support a subset of the LPM library for any of the following reasons:

- Existing synthesis tools provide acceptable results for simpler modules
- A specific module cannot be implemented in the target technology, e.g. some technologies cannot support RAM
- A specific feature of a particular module cannot be implemented in the target technology, e.g. some technologies cannot support a D-type flip-flop with both asynchronous clear and preset
- Other methodologies are preferred for a particular module, e.g. a RAM compiler may be preferred over the LPM RAM module

Each silicon vendor can provide an exception list which identifies modules and features that are not supported. This list allows design entry and synthesis tools to generate only netlists that can be implemented in the target technology.

2. FORMAT OF LPM MODULES

The LPM standard currently describes 25 configurable modules. Each module has options that allow customization of the size and functionality of that module. For example the counter module can be customized to have a width of 14 bits, use up/down and enable controls, and have an active low asynchronous clear.

2.1 PARAMETERIZATION OF MODULES

2.1.1 Parameterizing Size

All the LPM components have configurable width i.e., they can provide the same function to multiple signals of a vector. An example of this is the LPM_INV (inverter) component which has input ports DATA₀, DATA₁ ... DATA_{n-1} and corresponding output ports RESULT₀, RESULT₁ ... RESULT_{n-1}. Thus they need a minimum of one parameter, the width "n". Some components also have a variable size. The AND component can have different sizes (e.g., it can have 2,3...m inputs per AND). Each of the vector inputs and outputs will then have this size and will thus have input ports DATA₀X₀, DATA₀X₁,... DATA₀X_{n-1}, DATA₁X₀, DATA₁X₁,... DATA₁X_{n-1}, ..., DATA_{m-1}X₀, DATA_{m-1}X₁, .. DATA_{m-1}X_{n-1}.

2.1.2 Parameterizing Functionality

In addition to having a varying width and size, LPM components have a parameterized function. Most of the components provide a wide variety of functions (e.g., the counter can include a comparator operator to provide a signal when the counter reaches a specified value). However, many applications do not need this function and would not like to pay for the incorporation of unused logic. Aside from the basic primitive function, the additional functions incorporated in LPM modules can be deleted by simply not using the ports that are needed solely for the unnecessary function.

The description of each module thus describes which ports are required and which are optional. The LPM_COUNTER, for instance, has a required port, the Clock port, but its Data outputs are optional as the LPM_COUNTER can be count modulo 2^N where N is the width of the counter.

Some functional combinations are mutually exclusive and lead to the definition of an illegal combination of ports. If illegal combinations of ports are used in a cell definition then it is considered to be an error.

2.1.2.1 No Package Pin Placement Information

Pin placement information is tied closely to the chip package and therefore is tied too tightly to the physical design. While this is an important issue, it is outside the scope of the LPM standard, which deals with the logical design.

2.2 DESCRIPTION OF PROPERTIES

2.2.1 Legal Values for Properties

All of the LPM modules have some properties associated with them. For the storage primitives it might be a constant load value; for the compare primitive it is the assumed representation of the input (unsigned or signed). The LPM standard specifies what values are valid for all recognized properties. Any non-standard value for a standard property is considered an error.

2.2.1.1 Legal Values for LPM_HINT Property

The only exception to the above is the LPM_HINT property. This optional property, which is attached to instances of cell use, contains additional technology-specific information for use by the silicon vendor tools. There are no standard values for this field and it is there for the design tool vendors and the silicon tool vendors to utilize as needed. This property can always be ignored and the correct logic will be constructed, although the logic may not be constructed in the most efficient way.

2.2.2 Universal Properties

Some properties apply to all LPM modules. Some of these properties are optional and some are required. Although these properties can be used with any module, they are not described as one of the required or optional properties in the individual module descriptions.

2.2.2.1 Signal Polarities (LPM_Polarity Property)

This property is optional and can be applied to any port on any module. By default, all signals are assumed to use a true (active high) representation. Use of the LPM_Polarity property will make the signal active low. The only valid value for this property is "INVERT".

2.2.2.2 Basic Module type (LPM_TYPE Property)

This property is required on all modules. This property is a part of the cell definition in EDIF and identifies which LPM module was the prototype for the specific instance in the library. This is the only property that is ALL CAPS. For all other properties only the first letter of each word is capitalized.

2.2.2.3 Implementation Suggestions (LPM_Hint Property)

This is an optional property on all modules. It contains additional technology-specific information for use by the silicon vendor's tools. There are no standard values for this field and it is up to the design tool vendors and the silicon tool vendors to utilize it as needed. The LPM_Hint property can always be ignored and the correct logic is still constructed, although it may not be constructed efficiently.

2.3 DATA REPRESENTATION IN MODULES

2.3.1 Bit-Ordering

Bits in buses are ordered (and therefore indexed) in the big-endian style. That is, the most-significant bit is on the left and has the highest bit index. The least-significant bit is the right-most, and is numbered 0. Fractional bits are not supported with this notation.

2.3.2 Arithmetic

The representation of an LPM module refers to the encoding of the data input and output from the module on the bused ports. Valid representations are UNSIGNED and SIGNED. When numeric quantities are being represented in one of these forms, then the encoding is two's complement (binary). When logical quantities are being represented, then the UNSIGNED representation should be used.

The representation can affect the manner in which an LPM module operates as follows:

- **Sign Extension**
A right shift in the LPM_CLSHIFT module operates differently for SIGNED and UNSIGNED representations. With the SIGNED representation, sign extension is performed on right shifts, while zero fill is performed on the UNSIGNED representation.
- **Overflow**
Indicates if the precision of the operation has been exceeded. In SIGNED arithmetic, this output is usually the exclusive-OR of the two most-significant bits.
- **Carry-Out**
This indicates that a carry/borrow should be communicated to another stage.

LPM does not enforce the consistency of data representation between interconnected modules. Therefore, it is possible to connect the output of a SIGNED LPM_ADD_SUB to the input of an UNSIGNED LPM_ADD_SUB.

2.3.3 LPM Values

“LPM Values” are used to express quantity in LPM. The following two methods can be used to express LPM Values:

2.3.3.1 Integer

In any situation where LPM specifies that an **LPM Value** can be used, it is possible to use a simple base ten integer. These are limited to values in the range -2,147,483,648 to +2,147,483,647, that is, using 32 bits. In some cases, only whole number (integers > 0) are appropriate: LPM_Width, LPM_WidthAd, LPM_WidthS, LPM_WidthA, LPM_WidthB, LPM_Size, LPM_Decodes, LPM_Modulus and LPM_Numwords, and it will be noted.

Examples:

273 +73 -2147483625

The regular expression for integer LPM Values:

```
[-+]?([0]|[1-9][0-9]*)
```

2.3.3.2 String

In addition to using an integer to specify a property value, a string of containing an ASCII representation of a number in base ten, base two (B) or base sixteen (H) can be used to specified a LPM value if the quantity is larger than $(2^{31}-1)$ or the vector oriented representation is commonly used.

The string representation usage is limited to LPM_Avalue, LPM_Svalue, LPM_Pvalue, LPM_Cvalue and LPM_Modulus. For example,

LPM_Avalue	Binary String representation	Hex String representation
2^{34}	"B10 0010 0100111101010001010100100111"	"H2_24F5_1527"
2^7	"B1000011"	"H43"

In some cases, the LPM Value associated with a property can be interpreted as "a binary pattern LPM_Width bits wide". When this value is translated into binary, if it becomes a pattern with less than n bits, it should be right justified in the field and sign extended to fill the undefined bits of the n bit wide vector. If the binary equivalent of the value needs more than n bits, the n least significant bits should be used to fill the binary pattern.

Here is an example where the LPM_Avalue is specified for the LPM_FF. The LPM Value and the corresponding value in the register after the Aset signal has been asserted are shown for various possibilities.

LPM_Width = 3	LPM_Avalue = "B1111"	Register = "111"	Truncated
LPM_Width = 5	LPM_Avalue = "B1111"	Register = "01111"	Sign Extended
LPM_Width = 5	LPM_Avalue = -10	Register = "10110"	Sign Extended
LPM_Width = 5	LPM_Avalue = 10	Register = "01010"	Sign Extended
LPM_Width = 5	LPM_Avalue = -50	Register = "01110"	Truncated. Note 1

Note 1: The value in the register is "corrupted" since the LPM_Width is not sufficient to represent the intended value.

2.3.3.3 Don't Care

Where appropriate, "don't cares" are allowed in the binary and hexadecimal string values, represented by a '-'. The width of the "don't care" is the width of a digit in the appropriate base; one bit for binary and four bits for hexadecimal.

Examples:

"B1111_----"	"-B----_1111"
"HEAD--"	"-HE--"

The "don't cares" can take on any value for simulation. That is: undefined, one or zero.

The regular expressions for string values with don't cares in LPM:

"[-+]?B[-01_]*"	"[-+]?H[-0-9a-fA-F_]*"
-----------------	------------------------

3. DESIGN FOR TESTABILITY IN LPM

LPM provides an optional feature that allows test functions to be built into a design without constraining all silicon architectures to use this test feature. The standard allows the insertion of test ports and nets within a design. An overview of the test strategy is given below. Module-specific information is given in the description of each module.

3.1 Requirements

It is important to note that test ports and nets in an LPM netlist can be ignored by silicon vendors with a built-in ability to test the functionality of their parts. The testing feature is most commonly used with ASIC designs, which frequently use scan testing strategies.

3.2 Testing Philosophy

The testing strategy is assumed to be compatible with the Level Sensitive Scan Design (LSSD) testing strategy developed by IBM. This provides capabilities for testing "stuck at" faults within a design. A complete implementation of LSSD that achieves a suitably high coverage is a complex process and will not be fully discussed here. The LPM modules provide the options to work within this framework.

3.2.1 How LSSD works

LSSD is a method of designing a circuit so that you can easily test for "stuck at" faults within a design. LSSD does this by turning a sequential design into a combinatorial design for which it is relatively easy to obtain automatic test patterns. To transform the sequential design into a combinatorial one, all of the storage elements in the design must be connected via one or more shift register chains. By shifting data in and out of this chain, intervening combinatorial logic can be tested for stuck at faults. Note that it is still possible to design untestable logic between the storage elements.

LSSD requires that there be an external method of loading and later reading data in the shift register chain without disturbing the rest of the circuit. This loading and reading is called scanning. This usually requires two different clocks (one for normal operation and one for the scan operation) on the storage elements, and two methods of controlling them externally. There must also exist a clean path from an external input to an external output; the path must be routed through each storage element (or some of them, anyway) with no intervening logic.

The scan testing strategy proceeds as follows:

1. Disable system clocks (this includes any asynchronous sets or resets).
2. Apply the external test pattern on the Scan-In port and scan (shift) the test pattern in via the externally generated test clock (this clock can use the same wires as the system

clock via a multiplexer and a mode port on the storage modules - the method used in LPM).

3. Switch back to functional mode and issue a single clock pulse.
4. Switch to test mode and scan out the resulting vector for comparison with the expected contents of the scan shift register.

3.3 LPM Implementation of LSSD

There are many ways to implement the LSSD strategy and the LPM method does not completely dictate how the designer would do it, but the LPM method does limit some choices. Instead of having two clocks each with its own data input, LPM has a single clock and an enable port which controls the mode of the module. Of course, you can view this as two separate clocks, with the system clock tied off when the circuit is in scan mode.

3.4 Test Options

Obviously the testing options pertain only to the clocked modules of LPM which include the latch, flip-flop, counter, and the state machine. The options for each of these module vary a bit, but these rules apply:

- All of the clocked modules have a test input (TestIn) port.
- All of the edge-triggered modules have test enable (TestEnab) and test output (TestOut) ports.
- All of the level-sensitive modules have a test gate (TestGate) port.

3.5 Usage: Clocking and Precedence

LPM uses only a single clock for both functional and testing modes, with a mode port selecting which to use. When in test mode, synchronous loads, sets and resets must be disabled within the LPM module. Asynchronous loads, sets, and resets are not required to be disabled within the module since these signals act more like clocks and will be assumed to be disabled external to the LPM module (Note: these signals must be disabled during test clocking mode or the scan testing strategy will not function correctly.)

When a clock occurs while in test mode, the test input will be clocked (or latched) into the storage element. There is no need for a test output port since this is exactly the same as the functional output port. A test output port would only be required if no functional output pin is placed at the end of the scan chain.

4. DESCRIPTION OF LPM MODULES

This section describes in detail the functionality and semantics of each module in LPM.

4.1 MODULE CATEGORIES

The LPM modules fall into five major categories:

CATEGORY	DESCRIPTION
GATES	
LPM_CONSTANT	Constant value
LPM_INV, LPM_AND, LPM_OR, LPM_XOR	Basic combinatorial gates
LPM_BUSTRI	Tri-State buffer
LPM_MUX	Multiplexer
LPM_DECODE	Decoder
LPM_CLSHIFT	Combinatorial shifter
ARITHMETIC COMPONENTS	
LPM_COUNTER	Counter
LPM_ADD_SUB	Adder/Subtractor
LPM_COMPARE	Comparator
LPM_MULT	Multiplier
LPM_ABS	Absolute Value
STORAGE COMPONENTS	
LPM_ROM	Read Only Memory
LPM_LATCH	Transparent latch
LPM_FF	D-type or T-type flip-flop
LPM_SHIFTREG	Shift Register
LPM_RAM_DQ, LPM_RAM_IO	Random Access Memory
TABLE PRIMITIVES	
LPM_FSM	Finite state machine
LPM_TTABLE	Truth table
PAD PRIMITIVES	
LPM_INPAD, LPM_OUTPAD, LPM_BIPAD	Input/Output/Bidirectional pads

Note that truth table, finite state machine, RAM and ROM modules require more information than can be contained in the LPM netlist to define their function. These modules use supporting files to describe their function. These supporting files use the standard Intel HEX, Berkeley PLA and KISS formats.

4.1.1 Logic Conventions

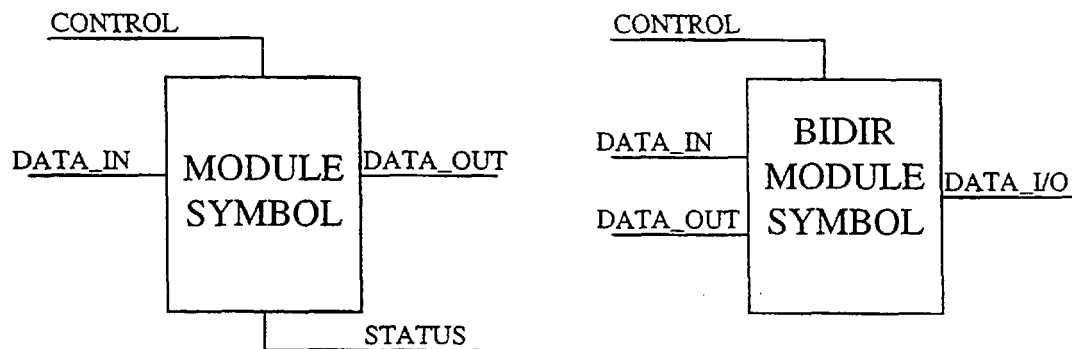
Where logic equations or logic models are used, the following symbols are used for both single bit operations and bit-wise vector operations:

AND	&	
OR		
XOR	^	
NOT or INVERT	~	on vectors this is 1's complement
NAND	~&	
NOR	~	
XNOR	~^	
LEFT SHIFT	<<	vector only
RIGHT SHIFT	>>	vector only
Two's Complement or Unsigned Add	+	
Two's Complement or Unsigned Subtract	-	
Two's Complement or Unsigned Multiply	×	

4.1.2 Drawing Conventions

The drawings of all modules in this document use the following conventions (shown in the figure below):

Data Inputs	Are shown going into the left of the module symbol.
Data Outputs	Are shown coming out of the right side of the module symbol except on bidir module symbols that have bi-directional port where they are on the left side.
Control Inputs	Are shown going into the top of the module symbol.
Status Outputs	Are shown coming out of the bottom of the module symbol.
Data I/Os	Are shown on the right side of the bidir module symbol.
Width & Size	Width refers to LPM_Width and Size refers to LPM_Size

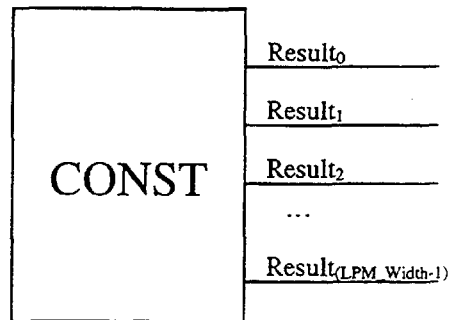


4.1.3 Scan Test Conventions

All of the modules that have scan-test ports (LPM_COUNTER, LPM_LATCH, LPM_FF, LPM_TFF, LPM_FSM) use the same convention. The **TestOut** port always has the same value as the most significant bit of the output (Q or State). When **TestEnab** is high, the data on **TestIn** is shifted into the least significant bit of the associated register as the contents of the register are shifted towards the most significant bit.

4.2 GATES

4.2.1 LPM_CONSTANT



4.2.1.1 Ports

Port Name	Type	Usage	Description	Comments
Result	O	Required	Value specified by CValue	Output vector LPM_Width wide LPM_Cvalue is truncated or sign extended to LPM_Width bits

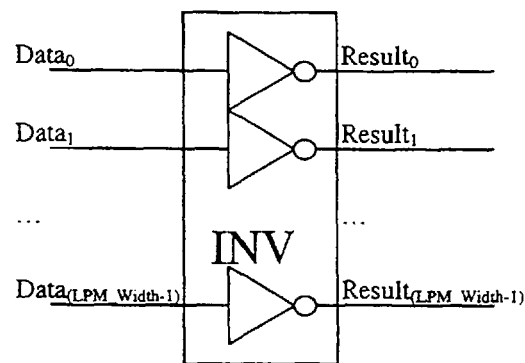
4.2.1.2 Properties

Property	Usage	Value	Comments
LPM_Width	Required	LPM Value > 0	Width of output vector
LPM_CValue	Required	LPM Value	Value of constant
LPM_Strength	Optional	WEAK	If present, this indicates a pullup or pulldown strength

4.2.1.3 Function

Result = LPM_Cvalue

4.2.2 LPM_INV



4.2.2.1 Ports

Port Name	Type	Usage	Description	Comments
Data	I	Required	Data input	Vector, LPM_Width wide
Result	O	Required	Inverted Result	Vector, LPM_Width wide

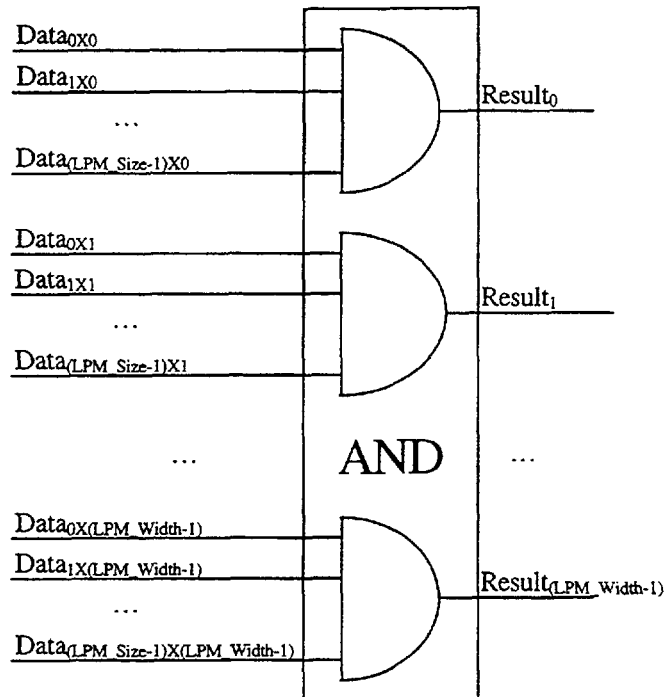
4.2.2.2 Properties

Property	Usage	Value	Comments
LPM_Width	Required	LPM Value > 0	Width of input and output vectors

4.2.2.3 Function

$$\text{Result} = \sim \text{Data}$$

4.2.3 LPM_AND



4.2.3.1 Ports

Port Name	Type	Usage	Description	Comments
Data	I	Required	Data input	Vector, LPM_Size times LPM_Width wide
Result	O	Required	Result of AND operators	Vector, LPM_Width wide

4.2.3.2 Properties

Property	Usage	Value	Comments
LPM_Width	Required	LPM Value > 0	Width of output vector. Number of AND gates.
LPM_Size	Required	LPM Value > 0	Number of inputs to each AND gate. Number of input buses.

4.2.3.3 Function

$$\text{Result}_0 = \text{Data}_{0X0} \& \text{Data}_{1X0} \& \text{Data}_{2X0} \& \dots \& \text{Data}_{\text{LPM_Size}-1X0}$$

$$\text{Result}_1 = \text{Data}_{0X1} \& \text{Data}_{1X1} \& \text{Data}_{2X1} \& \dots \& \text{Data}_{\text{LPM_Size}-1X1}$$

$$\text{Result}_2 = \text{Data}_{0X2} \& \text{Data}_{1X2} \& \text{Data}_{2X2} \& \dots \& \text{Data}_{\text{LPM_Size}-1X2}$$

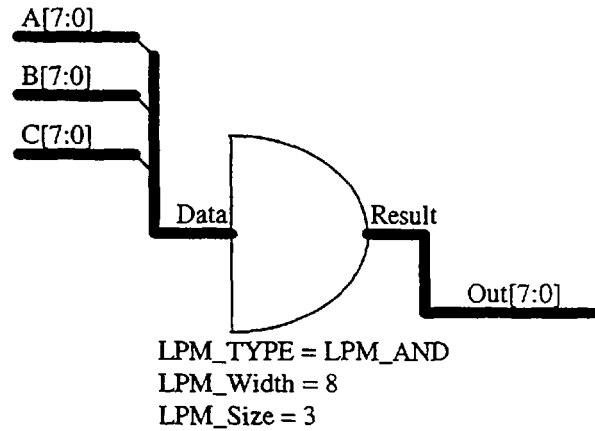
...

$$\text{Result}_i = \text{Data}_{0Xi} \& \text{Data}_{1Xi} \& \text{Data}_{2Xi} \& \dots \& \text{Data}_{\text{LPM_Size}-1Xi}$$

Where i goes from 0 to (LPM_Width - 1).

4.2.3.4 Example

Suppose the designers have three 8-bit buses and they want to AND the corresponding bits of the three buses. This is done using an LPM_AND with an LPM_Width = 8 and an LPM_Size = 3. The LPM_Width of eight indicates that there are eight AND gates, and the LPM_Size of three indicates that each AND gate has three inputs.



This diagram is for illustrative purposes only and is not intended to specify any implementation details.

The function performed by the LPM_AND gate in this case is:

$$Out[0] = Result_0 = Data_{0X0} \& Data_{1X0} \& Data_{2X0} = C[0] \& B[0] \& A[0]$$

$$Out[1] = Result_1 = Data_{0X1} \& Data_{1X1} \& Data_{2X1} = C[1] \& B[1] \& A[1]$$

$$Out[2] = Result_2 = Data_{0X2} \& Data_{1X2} \& Data_{2X2} = C[2] \& B[2] \& A[2]$$

$$Out[3] = Result_3 = Data_{0X3} \& Data_{1X3} \& Data_{2X3} = C[3] \& B[3] \& A[3]$$

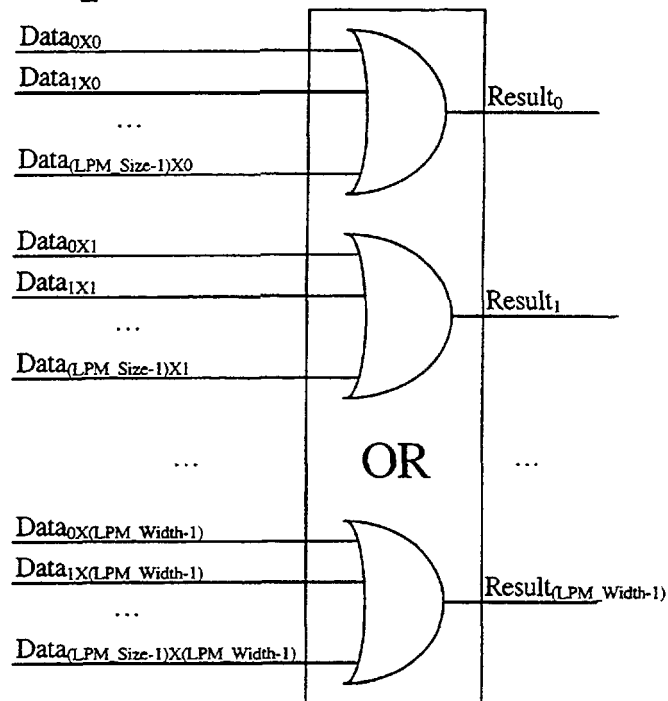
$$Out[4] = Result_4 = Data_{0X4} \& Data_{1X4} \& Data_{2X4} = C[4] \& B[4] \& A[4]$$

$$Out[5] = Result_5 = Data_{0X5} \& Data_{1X5} \& Data_{2X5} = C[5] \& B[5] \& A[5]$$

$$Out[6] = Result_6 = Data_{0X6} \& Data_{1X6} \& Data_{2X6} = C[6] \& B[6] \& A[6]$$

$$Out[7] = Result_7 = Data_{0X7} \& Data_{1X7} \& Data_{2X7} = C[7] \& B[7] \& A[7]$$

4.2.4 LPM_OR



4.2.4.1 Ports

Port Name	Type	Usage	Description	Comments
Data	I	Required	Data input	Vector, LPM_Size times LPM_Width wide
Result	O	Required	Result of OR operators	Vector, LPM_Width wide

4.2.4.2 Properties

Property	Usage	Value	Comments
LPM_Width	Required	LPM Value > 0	Width of output vector. Number of OR gates.
LPM_Size	Required	LPM Value > 0	Number of inputs to each OR gate. Number of input buses.

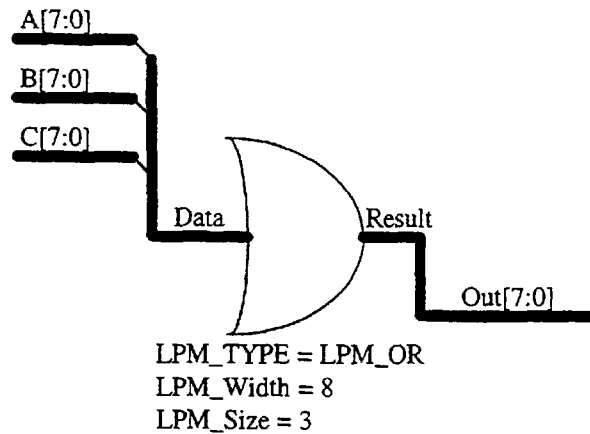
4.2.4.3 Function

$$\text{Result}_i = \text{Data}_{0Xi} \mid \text{Data}_{1Xi} \mid \text{Data}_{2Xi} \mid \dots \mid \text{Data}_{LPM_Size-1Xi}$$

Where i goes from 0 to $(LPM_Width - 1)$.

4.2.4.4 Example

Suppose the designers have three 8-bit buses and they want to OR the corresponding bits of the three buses. This is done using an LPM_OR with an LPM_Width of 8 and an LPM_Size of three. The LPM_Width of eight indicates that there are eight OR gates, and the LPM_Size of three indicates that each OR gate has three inputs.



This diagram is for illustrative purposes only and is not intended to specify any implementation details.

The function performed by the LPM_OR gate in this case is:

$$Out[0] = Result_0 = Data_{2X0} \mid Data_{1X0} \mid Data_{0X0} = A[0] \mid B[0] \mid C[0]$$

$$Out[1] = Result_1 = Data_{2X1} \mid Data_{1X1} \mid Data_{0X1} = A[1] \mid B[1] \mid C[1]$$

$$Out[2] = Result_2 = Data_{2X2} \mid Data_{1X2} \mid Data_{0X2} = A[2] \mid B[2] \mid C[2]$$

$$Out[3] = Result_3 = Data_{2X3} \mid Data_{1X3} \mid Data_{0X3} = A[3] \mid B[3] \mid C[3]$$

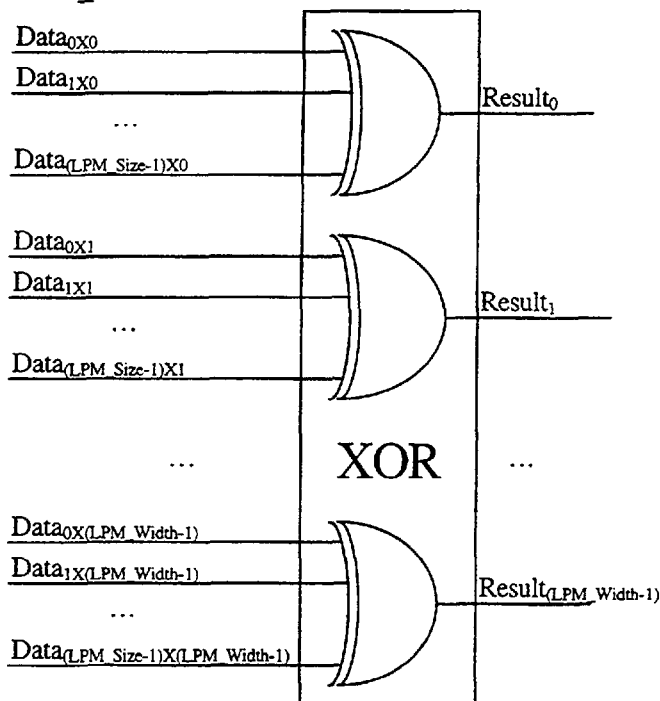
$$Out[4] = Result_4 = Data_{2X4} \mid Data_{1X4} \mid Data_{0X4} = A[4] \mid B[4] \mid C[4]$$

$$Out[5] = Result_5 = Data_{2X5} \mid Data_{1X5} \mid Data_{0X5} = A[5] \mid B[5] \mid C[5]$$

$$Out[6] = Result_6 = Data_{2X6} \mid Data_{1X6} \mid Data_{0X6} = A[6] \mid B[6] \mid C[6]$$

$$Out[7] = Result_7 = Data_{2X7} \mid Data_{1X7} \mid Data_{0X7} = A[7] \mid B[7] \mid C[7]$$

4.2.5 LPM_XOR



4.2.5.1 Ports

Port Name	Type	Usage	Description	Comments
Data	I	Required	Data input	Vector, LPM_Size times LPM_Width wide
Result	O	Required	Result of XOR operators	Vector, LPM_Width wide

4.2.5.2 Properties

Property	Usage	Value	Comments
LPM_Width	Required	LPM Value > 0	Width of output vector. Number of XOR gates.
LPM_Size	Required	LPM Value > 0	Number of inputs to each XOR gate. Number of input buses.

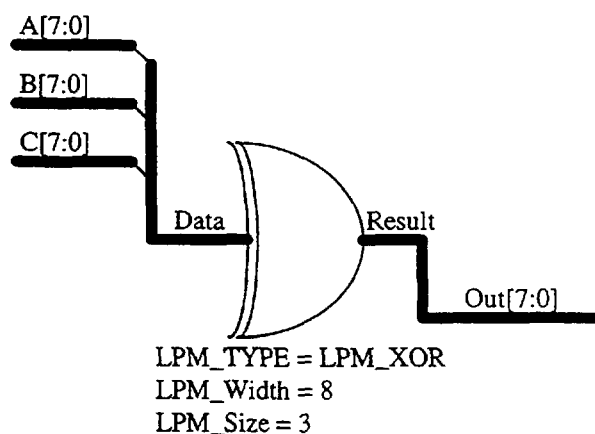
4.2.5.3 Function

$$Result_i = Data_{0Xi} \wedge Data_{1Xi} \wedge Data_{2Xi} \wedge \dots \wedge Data_{LPM_Size-1Xi}$$

Where i goes from 0 to $(LPM_Width - 1)$.

4.2.5.4 Example

Suppose the designers have three 8-bit buses and they want to XOR the corresponding bits of the three buses. This is done using an LPM_XOR with an LPM_Width of 8 and an LPM_Size of three. The LPM_Width of eight indicates that there are eight XOR gates, and the LPM_Size of three indicates that each XOR gate has three inputs.



This diagram is for illustrative purposes only and is not intended to specify any implementation details.

The function performed by the LPM_XOR gate in this case is:

$$\text{Out}[0] = \text{Result}_0 = \text{Data}_{2X0} \wedge \text{Data}_{1X0} \wedge \text{Data}_{0X0} = A[0] \wedge B[0] \wedge C[0]$$

$$\text{Out}[1] = \text{Result}_1 = \text{Data}_{2X1} \wedge \text{Data}_{1X1} \wedge \text{Data}_{0X1} = A[1] \wedge B[1] \wedge C[1]$$

$$\text{Out}[2] = \text{Result}_2 = \text{Data}_{2X2} \wedge \text{Data}_{1X2} \wedge \text{Data}_{0X2} = A[2] \wedge B[2] \wedge C[2]$$

$$\text{Out}[3] = \text{Result}_3 = \text{Data}_{2X3} \wedge \text{Data}_{1X3} \wedge \text{Data}_{0X3} = A[3] \wedge B[3] \wedge C[3]$$

$$\text{Out}[4] = \text{Result}_4 = \text{Data}_{2X4} \wedge \text{Data}_{1X4} \wedge \text{Data}_{0X4} = A[4] \wedge B[4] \wedge C[4]$$

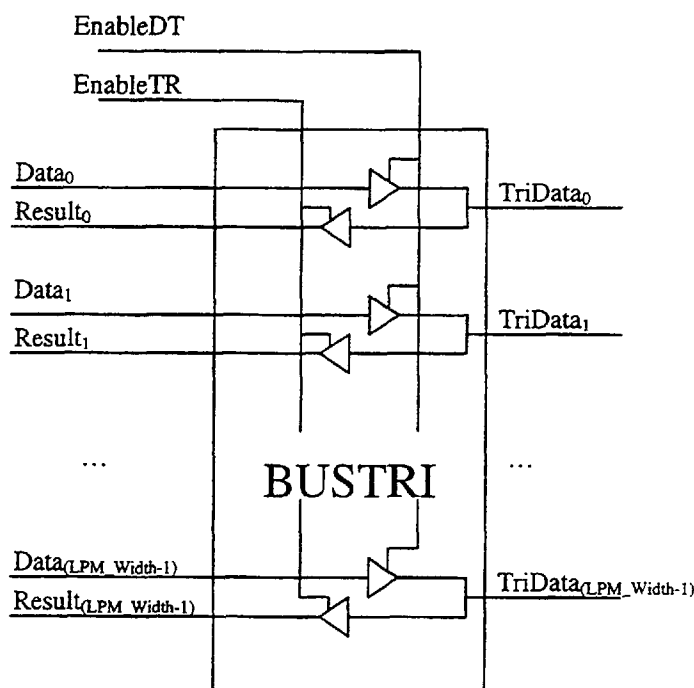
$$\text{Out}[5] = \text{Result}_5 = \text{Data}_{2X5} \wedge \text{Data}_{1X5} \wedge \text{Data}_{0X5} = A[5] \wedge B[5] \wedge C[5]$$

$$\text{Out}[6] = \text{Result}_6 = \text{Data}_{2X6} \wedge \text{Data}_{1X6} \wedge \text{Data}_{0X6} = A[6] \wedge B[6] \wedge C[6]$$

$$\text{Out}[7] = \text{Result}_7 = \text{Data}_{2X7} \wedge \text{Data}_{1X7} \wedge \text{Data}_{0X7} = A[7] \wedge B[7] \wedge C[7]$$

4.2.6 LPM_BUSTRI

Connection to a Tri-State Bus.



4.2.6.1 Ports

Port Name	Type	Usage	Description	Comments
TriData	IO	Required	Bi-directional bus signal	Vector, LPM_Width wide
Data	I	Note 1	Data input to TriData bus	Vector, LPM_Width wide. One of Data or Result must be used.
EnabledT	I	Optional	If HIGH, enables Data onto the TriData bus.	Default value is Low. Required if Data is used.
Result	O	Note 1	Output from TriData bus.	Vector, LPM_Width wide. One of Data or Result must be used.
EnableTR	I	Optional	If HIGH, enable TriData onto the Result bus.	Default value is Low. Required if Result is used

Note 1: Either the **Result** or **Data** port is required. Both may be used.

4.2.6.2 Properties

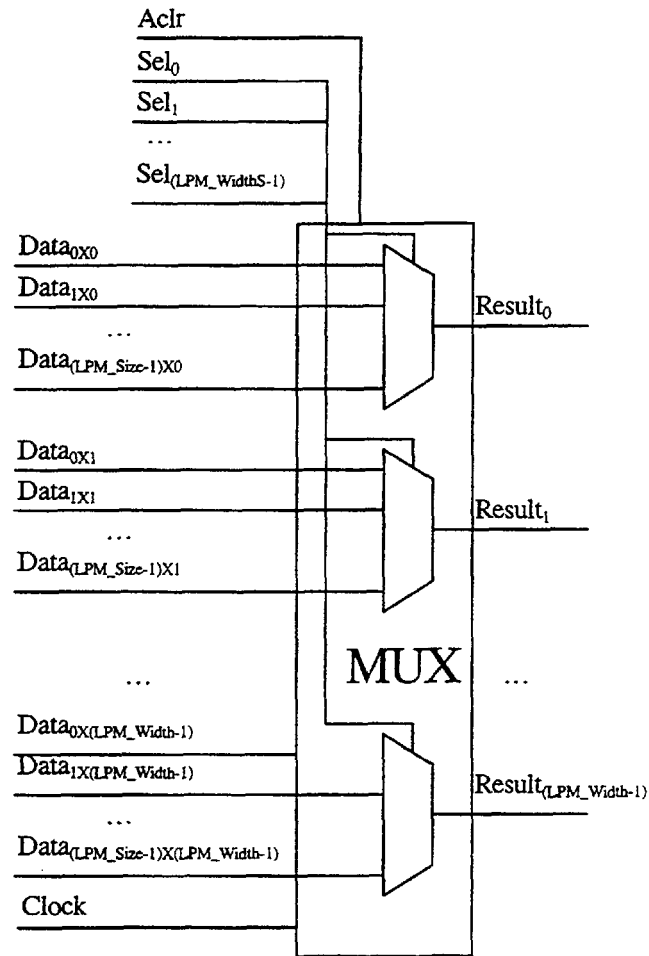
Property	Usage	Value	Comments
LPM_Width	Required	LPM Value > 0	Width of input and output vectors

4.2.6.3 Functions

EnabledT	EnableTR	Data	Result	TriData
L	L	X	Hi-Z	Hi-Z Note 1
L	H	X	VALUE (From TriData)	VALUE
H	L	VALUE	Hi-Z	VALUE (From Data)
H	H	VALUE	VALUE (From Data)	VALUE (From Data)

Note 1: When both control ports (EnabledT and EnableTR) are inactive (LOW) the Result port is high impedance, and the TriData port will take its value from the attached net (i.e. it is not driven by the LPM_Bustri).

4.2.7 LPM_MUX



4.2.7.1 Ports

Port Name	Type	Usage	Description	Comments
Data	I	Required	Data input	Vector, LPM_Size times LPM_Width wide
Result	O	Required	Selected input vector	Vector, LPM_Width wide
Sel	I	Required	Selects one of the input vectors	Vector, LPM_WidthS wide
Clock	I	Optional	Clock for pipelined usage	Note 1
Aclr	I	Optional	Asynchronous Clear	Note 2

Note 1: The clock port provides for pipelined operation of the LPM_MUX. If a lpm_pipeline other than 0 (default value) is specified, then the clock port must be connected.

Note 2: The pipelined initializes to undefined. The Aclr port may be used at any time to reset the pipeline to all 0's asynchronously to clock.

4.2.7.2 Properties

Property	Usage	Value	Comments
LPM_Width	Required	LPM Value > 0	Width of output vector. Number of Multiplexers.
LPM_Size	Required	LPM Value > 0	Number of inputs to each Multiplexer. Number of input buses.
LPM_WidthS	Required	LPM Value > 0	WidthS should be the next integer greater than or equal to $\log_2(\text{LPM_Size})$ or there will be unselectable input vectors.
LPM_Pipeline	Optional	LPM Value ≥ 0	Default is 0 - non-pipelined

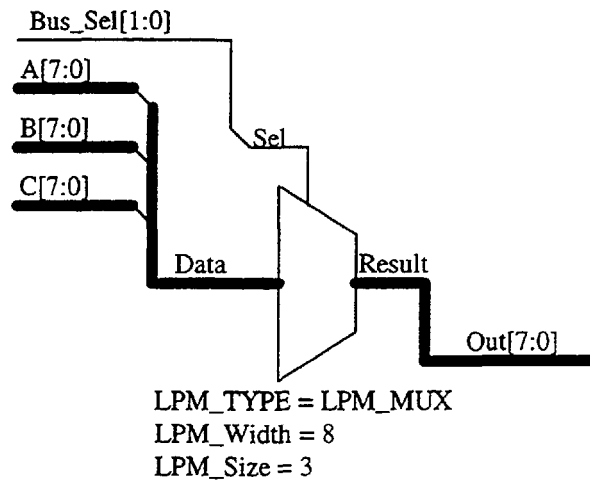
4.2.7.3 Functions

Sel vector	Sel Value	Result
0000...000	0	Data ₀ [LPM_Width-1:0]
0000...001	1	Data ₁ [LPM_Width-1:0]
0000...010	2	Data ₂ [LPM_Width-1:0]
...	...	...
1111...110	LPM_Size-2	Data _{LPM_Size-2} [LPM_Width-1:0]
1111...111	LPM_Size-1	Data _{LPM_Size-1} [LPM_Width-1:0]

This table assumes that LPM_Size is a power of two, but that is not required. If there is no **Data** vector that corresponds to the 'Sel Value'; that is, if **Data**_{Sel_Value} is not connected or is greater than LPM_Size, the selection of 'Sel Value' will produce an undefined **Result**.

4.2.7.4 Example

Suppose the designers have three 8-bit buses and they want to select one of the three buses. This is done using an LPM_MUX with an LPM_Width of 8 and an LPM_Size of three. The LPM_Width of eight indicates that there are eight multiplexers, and the LPM_Size of three indicates that each multiplexer has three inputs.



This diagram is for illustrative purposes only and is not intended to specify any implementation details.

Supposing that bus A becomes **Data_{2Xi}**, bus B becomes **Data_{1Xi}**, and bus C becomes **Data_{0Xi}**, the function performed by the LPM_MUX gate in this case is:

$\text{Out}[0] = \text{Result}_0 = \text{Data}_{\text{SelX}0} = (\text{UNDEFINED if Sel} = 3, \text{A}[0] \text{ if Sel} = 2, \text{B}[0] \text{ if Sel} = 1, \text{C}[0] \text{ if Sel} = 0)$

$\text{Out}[1] = \text{Result}_1 = \text{Data}_{\text{SelX}1} = (\text{UNDEFINED if Sel} = 3, \text{A}[1] \text{ if Sel} = 2, \text{B}[1] \text{ if Sel} = 1, \text{C}[1] \text{ if Sel} = 0)$

$\text{Out}[2] = \text{Result}_2 = \text{Data}_{\text{SelX}2} = (\text{UNDEFINED if Sel} = 3, \text{A}[2] \text{ if Sel} = 2, \text{B}[2] \text{ if Sel} = 1, \text{C}[2] \text{ if Sel} = 0)$

$\text{Out}[3] = \text{Result}_3 = \text{Data}_{\text{SelX}3} = (\text{UNDEFINED if Sel} = 3, \text{A}[3] \text{ if Sel} = 2, \text{B}[3] \text{ if Sel} = 1, \text{C}[3] \text{ if Sel} = 0)$

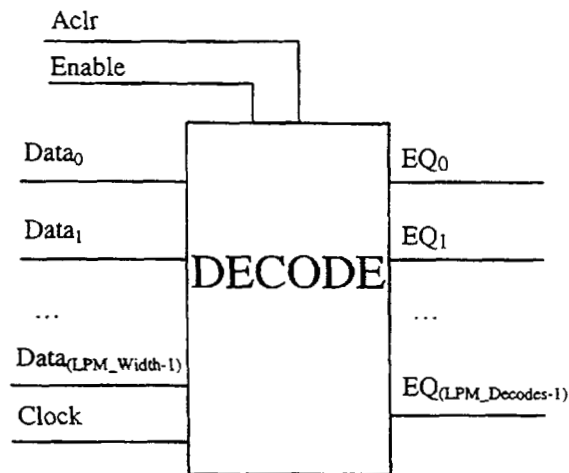
$\text{Out}[4] = \text{Result}_4 = \text{Data}_{\text{SelX}4} = (\text{UNDEFINED if Sel} = 3, \text{A}[4] \text{ if Sel} = 2, \text{B}[4] \text{ if Sel} = 1, \text{C}[4] \text{ if Sel} = 0)$

$\text{Out}[5] = \text{Result}_5 = \text{Data}_{\text{SelX}5} = (\text{UNDEFINED if Sel} = 3, \text{A}[5] \text{ if Sel} = 2, \text{B}[5] \text{ if Sel} = 1, \text{C}[5] \text{ if Sel} = 0)$

$\text{Out}[6] = \text{Result}_6 = \text{Data}_{\text{SelX}6} = (\text{UNDEFINED if Sel} = 3, \text{A}[6] \text{ if Sel} = 2, \text{B}[6] \text{ if Sel} = 1, \text{C}[6] \text{ if Sel} = 0)$

$\text{Out}[7] = \text{Result}_7 = \text{Data}_{\text{SelX}7} = (\text{UNDEFINED if Sel} = 3, \text{A}[7] \text{ if Sel} = 2, \text{B}[7] \text{ if Sel} = 1, \text{C}[7] \text{ if Sel} = 0)$

4.2.8 LPM_DECODE



4.2.8.1 Ports

Port Name	Type	Usage	Description	Comments
Data	I	Required	Data input. Treated as unsigned binary number.	Vector, LPM_Width wide
Enable	I	Optional	Enable. All outputs low when not active	Default value is Active (High) if absent.
EQ	O	Required	For $i = 0$ to LPM_Decodes if ($i = \text{Data}$) $Eq_i = 1$ else $Eq_i = 0$	Vector, LPM_Decodes wide. If $\text{Data} \geq \text{LPM_Decodes}$ then all Eq_i are 0
Clock	I	Optional	Clock for pipelined usage	Note 1
Aclr	I	Optional	Asynchronous Clear	Note 2

Note 1: The clock port provides for pipelined operation of the LPM_DECODE. If a lpm_pipeline other than 0 (default value) is specified, then the clock port must be connected.

Note 2: The pipelined initializes to undefined. The Aclr port may be used at any time to reset the pipeline to all 0's asynchronously to clock.

4.2.8.2 Properties

Property	Usage	Value	Comments
LPM_Width	Required	LPM Value > 0	Width of input vector
LPM_Decodes	Required	$0 < \text{LPM Value} \leq 2^{\text{LPM_Width}}$	Number of explicit decodes
LPM_Pipeline	Optional	LPM Value ≥ 0	Default is 0 - non-pipelined

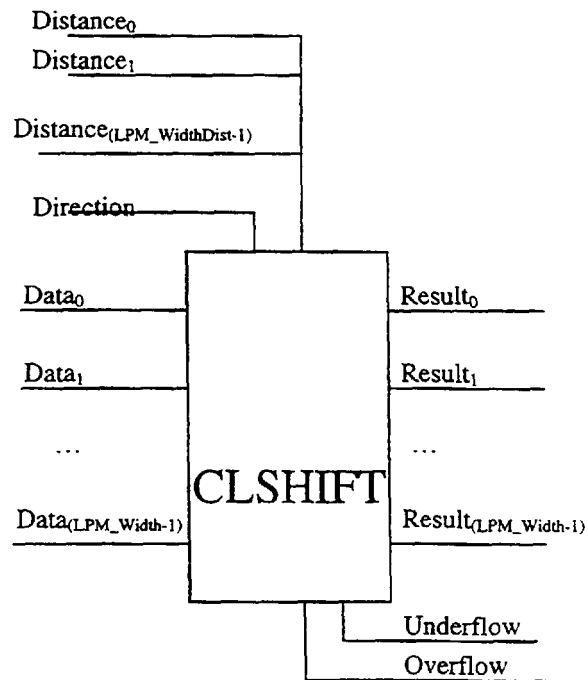
4.2.8.3 Functions

Enable	Data vector	Eq_i that is 1 (high) all other $Eq_i = 0$ (low)
L	X	NONE
H	0000...000	Eq_0
H	0000...001	Eq_1
...	...	...
H	000...0101	Eq_5
...	...	...
H	Data = LPM_Decodes - 1	$Eq_{(LPM_Decodes-1)}$
H	Data = LPM_Decodes	NONE
H	Data > LPM_Decodes	NONE

Note 1: If **Data** = i and Eq_i is not connected or does not appear in the symbol then all outputs will be low.

4.2.9 LPM_CLSHIFT

Combinatorial Logic shifter. Barrel Shifter.



4.2.9.1 Ports

Port Name	Type	Usage	Description	Comments
Data	I	Required	Data to be shifted.	Vector, LPM_Width wide
Distance	I	Required	Number of positions to shift Data.	Vector, LPM_WidthDist wide
Direction	I	Optional	Direction of shift. Low = Left (toward MSB) High = Right (toward LSB)	Default value is 0 (Low) = Left (toward the MSB)
Result	O	Required	Shifted Data	Vector, LPM_Width wide
Overflow	O	Optional	Logical or Arithmetic Overflow	Note 1
Underflow	O	Optional	Logical or Arithmetic Underflow	Note 1

Note 1: If the LPM_ShiftType is **ROTATE** and Overflow or Underflow are connected, the output of those ports will be undefined.

4.2.9.2 Properties

Property	Usage	Value	Comments
LPM_Width	Required	LPM Value > 0	Width of input vector
LPM_WidthDist	Optional	LPM Value > 0	Width of the Distance Port Note 1
LPM_ShiftType	Optional	LOGICAL ROTATE ARITHMETIC	Default is LOGICAL Note 2

Note 1: LPM_WidthDist specifies the width of the Distance port. The values on the Distance port would normally range from 0 which would mean “no shift” to (LPM_Width-1) which would be the maximum meaningful shift. The typical value assigned to LPM_WidthDist would be “the smallest integer not less than $\log_2(\text{LPM_Width})$ ” or $\lceil \log_2 \text{LPM_Width} \rceil$. Any value on the Distance port greater than LPM_Width-1 results in an UNDEFINED output.

Note 2: The sign bit is extended for ARITHMETIC. For a LOGICAL right shift 0’s are always shifted into the MSB.

4.2.9.3 Functions

The LPM_CLSHIFT module acts like a barrel-shifter. It is entirely combinational logic.

Overflow occurs when the shifted result exceeds the precision of the **Result** bus. For LOGICAL values, overflow occurs when all ones have been shifted out. For ARITHMETIC value, overflow occurs a significant digit is shifted into or past the sign bit.

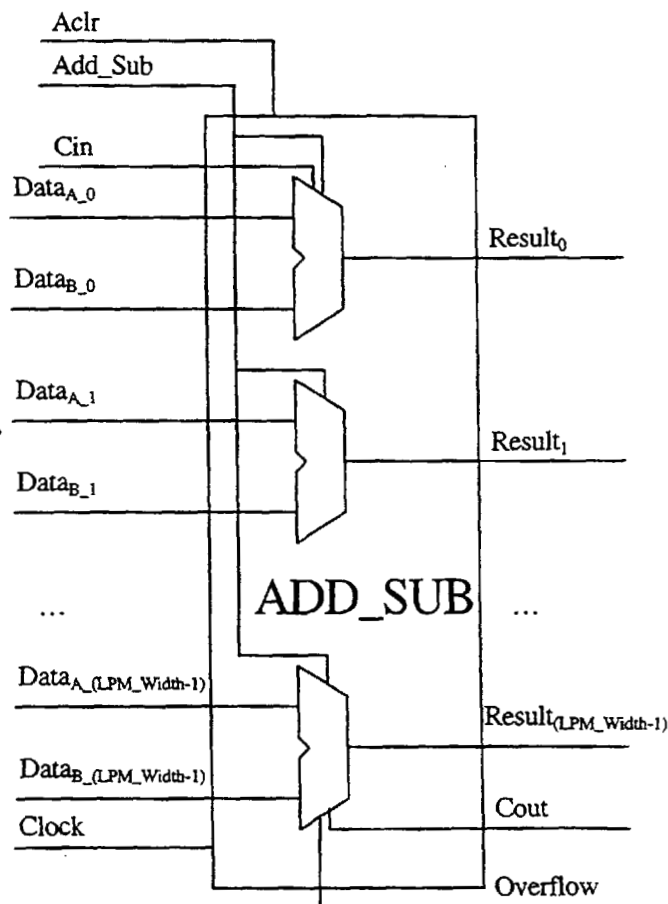
Underflow occurs when the shifted result contains no significant digits.

LPM_ShiftType	Direction	Function
LOGICAL	0 = Left	Result = Data << Distance
LOGICAL	1 = Right	Result = Data >> Distance
ROTATE	0 = Left	Result _i = Data _x where x is ((Distance + i) mod LPM_Width)
ROTATE	1 = Right	Result _i = Data _x where x is ((Distance - i) mod LPM_Width)
ARITHMETIC	0 = Left	Result = DATA * 2 ^{LPM_Distance}
ARITHMETIC	1 = Right	Result = DATA ÷ 2 ^{LPM_Distance} (integer divide)

Values larger than (LPM_Width - 1) result in an UNDEFINED output.

4.3 ARITHMETIC COMPONENTS

4.3.1 LPM_ADD_SUB



4.3.1.1 Ports

Port Name	Type	Usage	Description	Comments
DataA	I	Required	Augend/Minuend	Vector, LPM_Width wide
DataB	I	Required	Addend/Subtrahend	Vector, LPM_Width wide
Cin	I	Optional	Carry in to the low order bit OP=ADD Low= 0 High= +1 OP=SUB Low = -1 High= 0	If not connected, default value is LOW.
Add_Sub	I	Optional	The Operation to be performed =H: OP = ADD =L: OP = SUB	Cannot be used if LPM_Direction property is used. If not connected, defaults value is ADD.
Result	O	Required	$\text{DataA} \pm \text{DataB} \pm \text{Cin}$	Vector, LPM_Width wide
Cout	O	Optional	Carry-out (~Borrow-in) of Most Significant Bit (MSB)	Note 1
Overflow	O	Optional	Result exceeds available precision.	Note 2

Clock	I	Optional	Clock for pipelined usage	Note 3
Aclr	I	Optional	Asynchronous Clear	Note 4

Note 1: **Cout** has a physical interpretation as the carry-out (~borrow-in) of the most significant bit. Cout is most meaningful for detecting overflow in **unsigned** numbers. See Table 1 for the arithmetic interpretation of **Cout** = 1.

Note 2: Overflow has a physical interpretation as the XOR (exclusive or) of the carry into the MSB with the carry out of the MSB. Overflow is only meaningful when the LPM_Representation is **signed**. It indicates that the **Result** has exceeded the available precision. See Table 2 for the arithmetic interpretation of **Overflow** = 1.

Note 3: The clock port provides for pipelined operation of the LPM_ADD_SUB. If a lpm_pipeline other than 0 (default value) is specified, then the clock port must be connected.

Note 4: The pipelined initializes to undefined. The Aclr port may be used at any time to reset the pipeline to all 0's asynchronously to clock.

Table 1: Arithmetic interpretation of Cout = 1

	OP = ADD	OP = SUB
Unsigned	$(\text{DataA} + \text{DataB} + \text{Cin}) > 2^{\text{LPM_Width}-1}$	Normal Subtract. However , if Cout = 0, then $(\text{DataA} - \text{DataB} - \text{Cin}) < 0$
Signed	Normal result of adding two negative numbers, or possible overflow.	Normal result when subtracting a positive number from a negative number, or possible overflow.

Table 2: Arithmetic interpretation of Overflow = 1

	OP=ADD	OP=SUB
Unsigned	Not meaningful	Not meaningful
Signed	$(\text{DataA} + \text{DataB} + \text{Cin}) > 2^{\text{LPM_Width}-1}$ or $(\text{DataA} + \text{DataB} + \text{Cin}) < -2^{\text{LPM_Width}-1}$	$(\text{DataA} - \text{DataB} - \text{Cin}) > 2^{\text{LPM_Width}-1}$ or $(\text{DataA} - \text{DataB} - \text{Cin}) < -2^{\text{LPM_Width}-1}$

4.3.1.2 Properties

Property	Usage	Value	Comments
LPM_Width	Required	LPM Value > 0	Width of DataA , DataB and Result
LPM_Direction	Optional	ADD SUB	Default is ADD. Add_Sub port may not be used if this property is used.
LPM_Representation	Optional	UNSIGNED or SIGNED	Default is SIGNED
LPM_Pipeline	Optional	LPM Value ≥ 0	Default is 0 - non-pipelined

4.3.1.3 Functions

$$\text{Result}_i = \text{DataA}_i \wedge \text{DataB}_i \wedge \text{Cin}_i \wedge (\sim \text{Add_Sub})$$

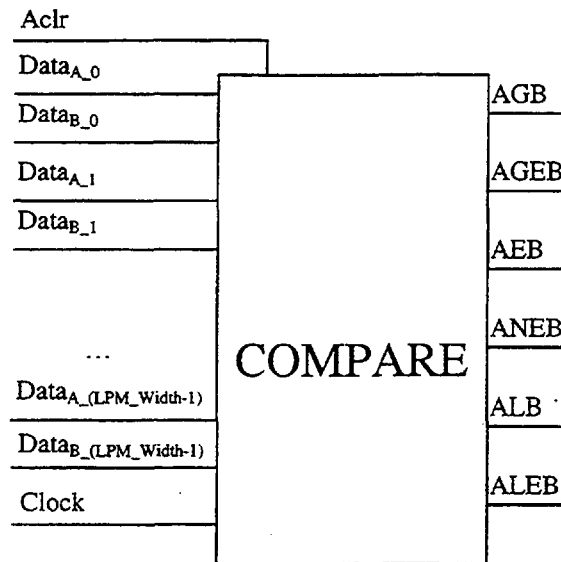
Cout = carry out of the MSB

Overflow = the XOR of the carry into the MSB and **Cout**

When **Cout** is prepended to the **Result**, the result is a vector that always has sufficient precision to represent the result of the operation.

$$\{\text{Cout}, \text{Result}\} = \text{DataA} + \text{Cin} \pm \text{DataB}$$

4.3.2 LPM_COMPARE



4.3.2.1 Ports

Port Name	Type	Usage	Description	Comments
DataA	I	Required	DataB is compared to this.	Vector, LPM_Width wide
DataB	I	Required	This is compared to DataA	Vector, LPM_Width wide
AGB	O	Note 1	High (1) if DataA > DataB	
AGEB	O	Note 1	High (1) if DataA ≥ DataB	
AEB	O	Note 1	High (1) if DataA = DataB	
ANEB	O	Note 1	High (1) if DataA ≠ DataB	
ALB	O	Note 1	High (1) if DataA < DataB	
ALEB	O	Note 1	High (1) if DataA ≤ DataB	
Clock	I	Optional	Clock for pipelined usage	Note 2
Aclr	I	Optional	Asynchronous Clear	Note 3

Note 1: At least one of the 6 output ports must be connected.

Note 2: The clock port provides for pipelined operation of the LPM_COMPARE. If a lpm_pipeline other than 0 (default value) is specified, then the clock port must be connected.

Note 3: The pipelined initializes to undefined. The Aclr port may be used at any time to reset the pipeline to all 0's asynchronously to clock.

4.3.2.2 Properties

Property	Usage	Value	Comments
LPM_Width	Required	LPM Value > 0	Width of DataA and DataB
LPM_Representation	Optional	UNSIGNED or SIGNED	Default is UNSIGNED.