Longest Prefix Match (LPM) Search IP for SDNet

SmartCORE IP Product Guide

PG191 (v1.0) November 10, 2017

XILINX®

XILINX

Table of Contents

Appendix D: Additional Resources and Legal Notices

Introduction

The Longest Prefix Match (LPM) SmartCORE™ IP compares a search key against a table of rules to find the best match, which in the case of Internet Protocol addresses is determined by the longest matching prefix. The LPM is programmed with arbitrary {pattern/length, value} entries and allows the retrieval of the value associated with the longest matching rule for a given search key.

Features

- Rule database storing {pattern/length, value} sets
- High throughput: one lookup per clock cycle
- Scalable, supporting a wide range of key, mask, and value field widths
- Efficient implementation using two block RAM bits per entry bit

SmartCORE IP Facts Table	
Core Specifics	
Supported Device Family	Kintex® UltraScale™ Virtex® UltraScale Kintex-7 Virtex-7 Spartan®-7
Supported User Interfaces	Lookup and AXI4-Lite Interfaces
Resources	See Table 2-1
Provided with Core	
Design Files	Encrypted register transfer level (RTL) or Netlist
Example Design	Verilog
Test Bench	Verilog
Constraints File	Not Provided
Simulation Model	Verilog
Supported S/W Driver(1)	Standalone
Tested Design Flows	
Design Entry	SDNet(2)
Simulation	Vivado® Simulator Mentor Graphics Questa Advanced Simulator
Synthesis	Not Applicable
Support	
Provided by Xilinx at the Xilinx Support web page	

Notes:

1. Stand-alone driver details can be found in the software development kit (SDK) directory (<install_directory>/SDK/<release>/data/embeddedsw/doc/xilinx_drivers.htm). Linux OS and driver support information is available from the Xilinx Wiki page.
2. See the *SDNet Packet Processor User Guide* (UG1012) [Ref 3].

Overview

The LPM SmartCORE™ IP implements a pipelined search tree with tree data stored in embedded block RAM. The LPM is generated according to parameters that control the table dimensions, and whether or not the implementation includes a shadow memory for updates (doubling the memory requirements).

The Lookup interface of the LPM receives a search key and outputs a result with a fixed latency that is a function of the capacity of the LPM. The result contains a hit/miss flag, indicating whether the key is found. If the key is found (hit condition), the associated value is output as well.

The table contents are initialized and can be updated by writing data to the AXI4 configuration interface. Mapping software is used to translate rules (prefix/length patterns and associated values) into an information sequence that is stored in memory. This information is loaded into the device using the Application Programming Interface (API) functions.

Examples of rules that can be programmed directly into the LPM using the provided Mapping and API software are shown in Table 1-1.

The rules in Table 1-1 consist of IPv4 Destination Address and mask pairs.

Note: The mask length is the value after the "/" and indicates the number of valid address bits in the address starting from the MSB. A length value of 32 indicates that all bits in the IPv4 Source or Destination Address are valid and the mask value in decimal is expressed as 255.255.255.255 (or 0xFF:FF:FF:FF in hexadecimal). Likewise, a mask length value of 30 indicates that the first 30 bits of the address are valid and the decimal mask value is 255.255.255.252 (or 0xFF:FF:FF:FC in hexadecimal)

Table 1-1: **Example of LPM Rules**

IPv4 Src Addr/Mask	Value
198.238.184.78/32	0
198.238.184.00/24	2
100.238.0.0/16	3
100.0.0.0/8	0

Licensing and Ordering

Note: Customers who pay a license fee for SDNet are then are able to use the IP such as this CAM as they want for free.

For more information, visit the SDNet Development Environment page.

Product Specification

This chapter includes information about the performance, resource utilization, and the port descriptions of the LPM. Figure 2-1 shows a high-level block diagram of the LPM.

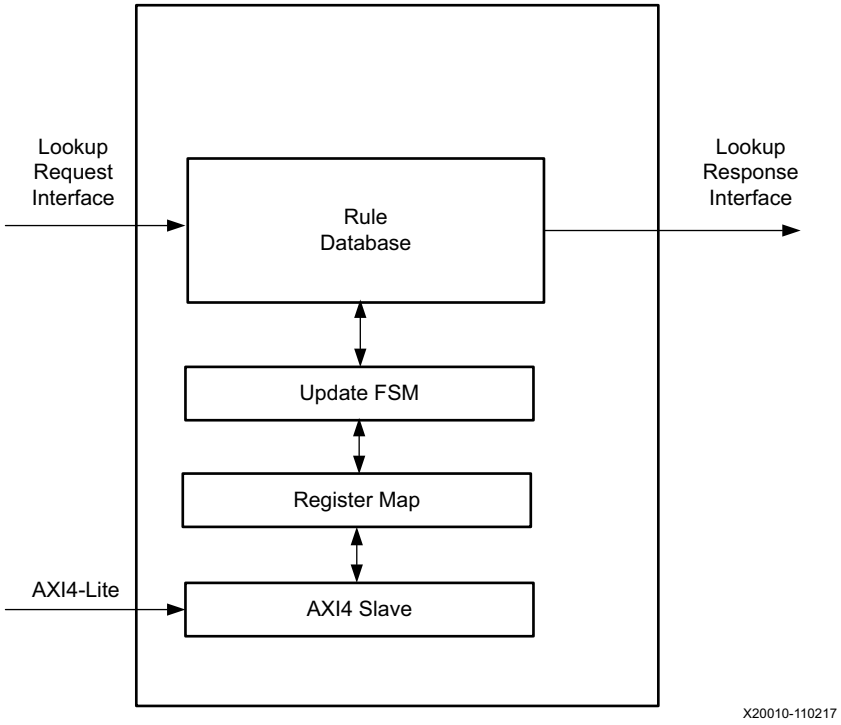

Figure 2-1: **High-Level LPM Block Diagram**

Performance

The LPM can continuously sustain a rate of one lookup per clock cycle.

Maximum Frequencies

The LPM is designed to run at 150 MHz in Virtex®-7 -2 speed grade devices. Higher frequencies can be achieved depending on the configuration and the target device. See Table 2-1.

Latency

The LPM latency is a function of the depth parameter (D) and is expressed as follows.

```
If D < 512
   Latency = log2(D) + 2
Else
   Latency = log2(D) + 9
```

If the SHADOW_MEM configuration parameter is set to one, two additional cycles are added to the above latency calculation.

Throughput

Because the LPM processes one lookup per cycle on the Lookup interface, for a clock frequency of 150 MHz, the LPM performs 150 million lookups per second. Throughput can be expressed as follows:

Throughput [Mp/s] = Clock Frequency [MHz]

Resource Utilization

Virtex-7 Devices

Table 2-1 provides approximate resource counts for the various core options on Virtex-7 devices.

Table 2-1: **Device Utilization and Maximum Frequency: Virtex-7 FPGAs**

Parameter Values	Device Resources				
Key width (K), Value width (V), Depth (D), Shadow Memory (SM)	**Slices**	**LUTs**	**FFs**	**Block RAMs**	**Max Freq (MHz)**
K=32 V=16 D=4095 SM=0	861	813	2,466	23	223
K=64 V=32 D=16383 SM=1	2,953	3,424	8,841	234	191
K=128 V=63 D=32767 SM=0	4,388	3,387	10,216	420	170

Port Descriptions

Note: For a complete list of the synthesis-time configuration parameters, see Configuration Parameters in Chapter 4.

Table 2-2: **Clock and Reset**

Signal Name	Direction	Width	Description
Rst	Input	1	Synchronous active-High reset
Clk	Input	1	Lookup interface clock

Lookup Interface

The Lookup interface is used to request a lookup and receive the response.

IMPORTANT: *SDNet exposes these ports through a wrapped interface as described in the SDNet Packet Processor User Guide (UG1012) [Ref 4].*

Table 2-3: **Lookup Request Interface**

Signal Name	Direction	Width	Description
LookupReqValid	Input	1	Lookup Request Valid.
LookupReqKey	Input	K	Lookup Request Key: Valid when LookupReqValid is asserted.

Table 2-4: **Lookup Response Interface**

Signal Name	Direction	Width	Description
LookupRespValid	Output	1	Lookup Response Valid, asserted for a fixed number of cycles following LookupReqValid assertion. Validates the rest of the LookupResp* signals.
LookupRespKey	Output	K	Valid when LookupRespValid=1. A copy of the key requested via LookupReqKey.
LookupRespHit	Output	1	Valid when LookupRespValid=1. Indicates LookupRespKey was matched in the LPM rule database and validates LookupRespValue signal.
LookupRespValue	Output	V	Valid when (LookupRespValid and LookupRespHit) = 1. Outputs the value word associated with the longest pattern matching LookupRespKey.

AXI4-Lite Configuration Interface

The AXI4-Lite interface is used for receiving register access commands from the CPU.

Table 2-5: **AXI4-Lite Configuration Interface**

Signal Name	Direction	Width	Description
AXI Global System Signals			
S_AXI_ACLK	Input	1	AXI Clock
S_AXI_ARESETN	Input	-	AXI Reset, active-Low
AXI Write Address Channel Signals			
S_AXI_AWADDR	Input	32	AXI Write address. The write address bus gives the address of the write transaction.
S_AXI_AWVALID	Input	1	Write address valid. This signal indicates that valid write address and control information are available.
S_AXI_AWREADY	Output	1	Write address ready. This signal indicates that the slave is ready to accept an address and associated control signals.
AXI Write Data Channel Signals (1)			
S_AXI_WDATA	Input	32	Write data
S_AXI_WSTB	Input	4	Write strobes. This signal indicates which byte lanes to update in memory.
S_AXI_WVALID	Input	1	Write valid. This signal indicates that valid write data and strobes are available.
S_AXI_WREADY	Output	1	Write ready. This signal indicates that the slave can accept the write data.
AXI Write Response Channel Signals (1)			
S_AXI_BRESP	Output	2	Write response. This signal indicates the status of the write transaction. 00 - OKAY 10 - SLVERR
S_AXI_BVALID	Output	1	Write response valid. This signal indicates that a valid write response is available.
S_AXI_BREADY	Input	1	Response ready. This signal indicates that the master can accept the response information.

Table 2-5: **AXI4-Lite Configuration Interface *(Cont'd)***

Signal Name	Direction	Width	Description
AXI Read Address Channel Signals[1]			
S_AXI_ARADDR	Input	32	Read address. The read address bus gives the address of a read transaction.
S_AXI_ARVALID[2]	Input	1	Read address valid. This signal indicates, when high, that the read address and control information is valid and will remain stable until the address acknowledgment signal, S_AXI_ARREADY, is High.
S_AXI_ARREADY	Output	1	Read address ready. This signal indicates that the slave is ready to accept an address and associated control signals.
AXI Read Data Channel Signals[1]			
S_AXI_RDATA	Output	32	Read data
S_AXI_RRESP	Output	2	Read response. This signal indicates the status of the read transfer.
S_AXI_RVALID	Output	1	Read valid. This signal indicates that the required read data is available and the read transfer can complete.
S_AXI_RREADY	Input	1	Read ready. This signal indicates that the master can accept the read data and response information.

Notes:

1. The function and timing of these signals is defined in the *AMBA AXI Protocol Version: 2.0 Specification* [Ref 1].
2. Read transactions have higher priority than write transactions.

Register Space

The register space is used for software control, monitoring and management of the LPM. This information is provided for reference; the provided API functions should be used for controlling the LPM instead of accessing the register space directly.

The register space is documented in relative offsets. During AXI system initialization, the LPM is assigned an absolute memory address range.

Table 2-6: **Offset = 0X00 Unique Device ID**

Field name	Bits	Access	Description
Unique Device ID	31:0	Read-only	Unique Device ID. Used by the API functions to ensure compatibility.

Table 2-7: **Offset = 0X40 Update Request**

Field name	Bits	Access	Description
Operation	28	Read-write	Writing this register offset triggers a new command request. This type of command is encoded in the value written into this field. Valid encodings are: • 0 = read the contents of the LPM data structure at the address referenced by the Addr field and store it in the Data register. • 1 = write the contents of the Data register into the LPM data structure at the address referenced by the Addr field.
Addr	27:0	Read-write	Controls the address being accessed by the read and write operations.

Table 2-8: **Offset = 0X44: Update Response**

Field name	Bits	Access	Description
Ack	0	Read-only	Acknowledge. This register is reset by hardware immediately upon detecting a write to the Update Request register. It is set by hardware after the Update Request is completed. Software should ensure that this register is set before scheduling a new request or reading the value register after an update command.

Table 2-9: **Offset = 0X4C: Data Status**

Field name	Bits	Access	Description
Status	31:0	Read-only	Data Status. Written by software prior to issuing a write operation. Written by hardware during read operation.

Table 2-10: **Offset = 0X50: Data Key[N]**

Field name	Bits	Access	Description
Data Key[n]	31:0	Read-write	Data Key. Bits [max(K, 31+32*n): 32*n] of the key. Written by software prior to issuing a write operation. Written by hardware during read operation.

The number of data/mask registers KN is equal to ceil(K /32).

Using integer division, KN = (K – 1) / 32 + 1.

Data registers occupy the offset range of [0x50: 0x50 + (KN*4) – 1]. Since K never exceeds 512, the maximum range of the data register offset is within [0x50:0x8F].

Table 2-11: **Offset = 0XA0: Data Value[N]**

Field name	Bits	Access	Description
Data Value[n]	31:0	Read-write	Data value. Bits [max(V, 31+32*n): 32*n] of the value. Written by software prior to issuing a write operation. Written by hardware during read operation

The number of data value registers VN is equal to ceil(V /32).
Using integer division, VN = V / 32 + 1.

Data registers occupy the offset range of [0xA0: 0xA0 + (VN*4) - 1]. Because V never exceeds 128, the maximum range of the data register offset is within [0xA0:0xAF].

Table 2-12: **Offset = 0XF0: Bank Select**

Field name	Bits	Access	Description
Bank Select	31:0	Read-write	Bank Select for SmartCORE IP cores generated with the SHADOW parameter. This selects which of logical banks is active for performing lookups, and the alternate bank can be configured in the background (accepting reads and writes).

Designing with the Core

This chapter includes guidelines and additional information to facilitate designing with the core.

Clocking

In general, the clock frequency should be equal to the packet rate. For example, for 100Gb/s Ethernet, the packet rate is approximately 150 Mp/s and therefore, the clock frequency must be 150 MHz.

Resets

Reset must be asserted for at least 100 cycles at startup.

Protocol Description

Lookup Interface

The Lookup interface is used to check for the longest matching prefix rule for a given search key in the LPM data structure. If a matching prefix rule is present, the LPM value corresponding to the longest matching prefix is fetched.

The Lookup interface should not be accessed during initialization or during updates scheduled on the AXI4-Lite interface.

The Lookup Interface is composed of two sub-interfaces: Lookup Request and Lookup Response. The Lookup Request interface is used by the application to drive the keys that need to be checked, and the Lookup response interface reports the presence of the matching rules and returns the values corresponding to the longest match.

Lookup Response

The Lookup Response interface reports the result of looking up the key scheduled on the Lookup Request interface. `LookupRespValid` is asserted with respect to `LookupReqValid`. The latency is fixed for a given configuration and depends on the size of the data structure. Please refer to the Latency in Chapter 2 for details.

`LookupRespValid` is asserted for one cycle during which the `LookupRespKey` duplicates the value present on the `LookupReqKey` during the corresponding `LookupReqValid` assertion cycle. `LookupRespHit` is valid together with `LookupRespKey` and indicates the presence of a matching rule in the LPM. When `LookupRespHit` is asserted, it also validates `LookupRespValue` output, which contains the value portion of the rule. When `LookupRespHit` is negated, it means a matching rule was not found and `LookupRespValue` is invalid.

Figure 3-1 shows the timing waveforms of the Lookup Interface operation. The first and third lookup requests result in misses. The second and fourth requests result in hits and output the corresponding values on the response interface.

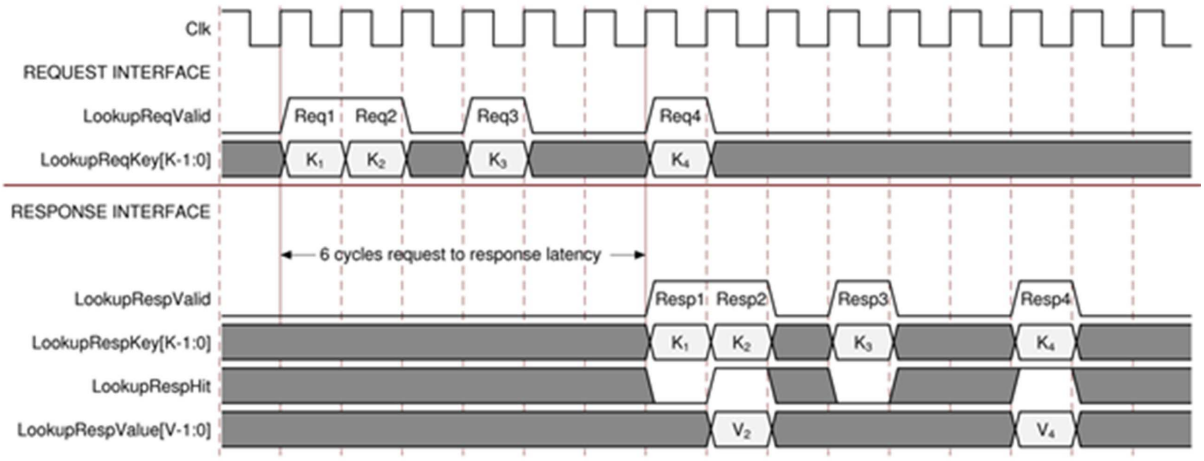

Figure 3-1: **Lookup Interface Timing**

Updates

The LPM is generated with or without a shadow memory according to the SHADOW_MEM configuration parameter. The shadow memory capability enables in-service updates of the LPM (that is, allowing lookups to take place during updates) by providing two memory banks: an active one used for lookups and another (non-active) used for updates. This capability essentially doubles the memory utilization of the LPM.

The Rule Mapping Software (see Rule Mapping Software in Appendix C for more details) converts a set of input rules into a firmware image used to program the LPM. The firmware image is loaded into the LPM by using the `LPM_Mgt_LoadDataset()` API function (see Appendix C, Application Software Development for more detail). In the case of instances having a shadow memory, the firmware image is loaded into the non-active bank of the LPM. The non-active bank must then be made active by using the `LPM_Mgt_SetActiveLookupBank()` API function (see Appendix C, for more detail) before it can be used for lookups.

LPM instances without a shadow memory should not be updated while lookups are in-flight, in order to ensure consistent lookup results. In addition, lookups should be deferred until the update of the LPM is complete.

Design Flow Steps

This chapter describes customizing and generating the core, constraining the core, and the simulation, synthesis and implementation steps that are specific to this IP core.

Configuration

The LPM SmartCORE™ IP design is highly configurable at compile time to make it suitable for a large variety of applications. Table 4-1 lists available configuration parameters. Please note that changing parameters requires design synthesis and generation of a new FPGA bitstream.

Table 4-1: **Configuration Parameters**

Parameter Name	Description
K	Key width, 8 to 128 bits
V	Value width, 1 to 32 bits
D	Number of entries
SHADOW_MEM	Creates a second copy of the table to support updates without interrupting the search operation. This feature doubles the storage requirements.
INSTANCE_NAME	Instance name used for the top level instance, e.g., "LPM_1"

To configure the core properly for a given application, the following must be considered.

- K (Key width) must be equal to the number of bits in the search key.
- V (Value width) must be equal to the number of bits in the value associated with the key.
- D (Depth) must be equal to number of keys that must be stored in the LPM.
- SHADOW_MEM must be set to 1 if an in-service update capability is required; otherwise it must be set to 0.

  Setting this parameter doubles the memory requirement by creating two tables. One table is active. The active table can be used for lookups, while the inactive table can be updated in the background. These tables can be swapped by selecting which table is active.

Output Generation

Table 4-2, shows the files associated with the core.

Table 4-2: **Core Files**

Filenames	Description
<INSTANCE_NAME>/rtl/<INSTANCE_NAME>.v	Encrypted Verilog source code
<INSTANCE_NAME>/model/model.sv <INSTANCE_NAME>/sim/run.do <INSTANCE_NAME>/tb/tb.sv	Behavioral model of the LPM Questa Advanced Simulator command file Demonstration test bench
<INSTANCE_NAME>/api/<INSTANCE_NAME>.h <INSTANCE_NAME>/api/xilinx_lpm.h <INSTANCE_NAME>/api/xilinx_lpm.c	Software driver files

Constraining the Core

This section contains information about constraining the core.

Required Constraints

This section is not applicable for this IP core.

Device, Package, and Speed Grade Selections

This section is not applicable for this IP core.

Clock Frequencies

The design must contain a clock frequency constraint for the primary clock driving the Clk input of the LPM. An example for constraining a primary clock net called clk150 that drives the Clk input at 150 MHz is shown below.

```
create_clock -period 6.667 -name clk150 [get_ports clk150]
```

Clock Management

This section is not applicable for this IP core.

Clock Placement

This section is not applicable for this IP core.

Banking

This section is not applicable for this IP core.

Transceiver Placement

This section is not applicable for this IP core.

I/O Standard and Placement

This section is not applicable for this IP core.

Simulation

For comprehensive information about Vivado simulation components, as well as information about using supported third-party tools, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 4]. Please also refer to the *SDNet Packet Processor User Guide* (UG1012) [Ref 3].

IMPORTANT: *For cores targeting 7 series or Zynq-7000 devices, UNIFAST libraries are not supported. Xilinx IP is tested and qualified with UNISIM libraries only.*

Synthesis and Implementation

For details about synthesis and implementation, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 5].

Appendix A

Upgrading

This appendix is not applicable for the first release of the core.

Debugging

This appendix includes details about resources available on the Xilinx Support website and debugging tools.

Finding Help on Xilinx.com

To help in the design and debug process when using the LPM SmartCORE, the Xilinx Support web page contains key resources such as product documentation, release notes, answer records, information about known issues, and links for obtaining further product support.

Documentation

This product guide is the main document associated with the LPM SmartCORE™ IP. This guide, along with documentation related to all products that aid in the design process, can be found on the Xilinx Support web page or by using the Xilinx Documentation Navigator.

Download the Xilinx Documentation Navigator from the Downloads page. For more information about this tool and the features available, open the online help after installation.

Answer Records

Answer Records include information about commonly encountered problems, helpful information on how to resolve these problems, and any known issues with a Xilinx product. Answer Records are created and maintained daily ensuring that users have access to the most accurate information available.

Answer Records for this core can also be located by using the Search Support box on the main Xilinx support web page. To maximize your search results, use proper keywords such as:

- Product name
- Tool message(s)
- Summary of the issue encountered

A filter search is available after results are returned to further target the results.

The Master Answer Record for this core is AR 59718.

Technical Support

Xilinx provides technical support at the Xilinx Support web page for this SmartCORE IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support if you do any of the following:

- Implement the solution in devices that are not defined in the documentation.
- Customize the solution beyond that allowed in the product documentation.
- Change any section of the design labeled DO NOT MODIFY.

To contact Xilinx Technical Support, navigate to the Xilinx Support web page.

Debug Tools

There are many tools available to address LPM design issues. It is important to know which tools are useful for debugging various situations.

Hardware Debug

Hardware issues can range from link bring-up to problems seen after hours of testing. This section provides debug steps for common issues.

General Checks

Ensure that all the timing constraints for the core were properly incorporated and that all constraints were met during implementation.

- Does it work in post-place and route timing simulation? If problems are seen in hardware but not in timing simulation, this could indicate a PCB issue. Ensure that all clock sources are active and clean.
- If using mixed-mode clock manager (MMCMs) in the design, ensure that all MMCMs have obtained lock by monitoring the LOCKED port.
- If your outputs go to 0, check your licensing.

Interface Debug

AXI4-Lite Interfaces

Read from a register that does not have all 0s as a default to verify that the interface is functional. See *AMBA AXI Protocol Version: 2.0 Specification* [Ref 1] for a read timing diagram. Output `s_axi_arready` asserts when the read address is valid, and output `s_axi_rvalid` asserts when the read data/response is valid. If the interface is unresponsive, ensure that the following conditions are met:

- The `s_axi_aclk` and `aclk` inputs are connected and toggling.
- The interface is not being held in reset, and `s_axi_areset` is an active-Low reset.
- The interface is enabled, and `s_axi_aclken` is active-High (if used).
- The main core clocks are toggling and that the enables are also asserted.

If the simulation has been run, verify in simulation that the waveform is correct for accessing the AXI4-Lite interface.

Application Software Development

This Appendix describes the software provided together with the LPM SmartCORE™ IP.

The LPM SmartCORE IP solution consists of the Rule Mapping software and Application Programming Interface (API) functions for initializing the managing of the hardware device.

The Rule Mapping software converts prefix rules from text representation to the internal data structure format that can be loaded into the LPM using the API functions.

Rule Mapping Software

The Java-based Rule Mapping software generates the configuration data that is written to the LPM block. The software accepts the following command line parameters.

Table C-1: **Command Line Parameters**

Argument switch	Description	Default
-i <input_fn>	the input filename	read from stdin
-o <output_fn>	the output filename	print to stdout
-n <table_name>	name of the prefix table	"lpm_table"
-k <key_len>	number of bits in input key	32
-r <res_len>	number of bits in output result	16
-d <yes|no>	whether to allow duplicate entries	Yes
-l <max_lvl>	the maximum tree level number	Calculated based on input
-w <waddrlen>	length (in bits) of word address within a node	
-I <full|single|batch>	use {full, single, batch} insert mode	Full

For the number of levels (-l), a default a minimum guaranteed value will be calculated and used. This also determines the number of level and node address bits.

If the input is completely valid, the configuration information is written to `Output.tbl` and consists of a list of address and data (both hex) pairs that the `LPM_Mgt_LoadDataset()` API function (described below) writes to the LPM.

The control software returns EXIT_FAILURE if the input is invalid, EXIT_SUCCESS otherwise.

To run the LPM generator:

```
java -jar lpm_cfg.jar <parameters>
```

As a simple example, following is a 10-entry table, consisting of 32-bit keys and explicit 8-bit results:

Prefix_value (dotted decimal or colon-separated-hex) /Length (decimal)

Stored_result (hex)

```
4.17.225.0/24 1
4.17.251.0/24 2
4.17.252.0/23 3
4.36.200.0/21 4
6.1.0.0/16 5
6.2.0.0/22 6
6.3.0.0/18 7
6.9.0.0/20 8
8.6.220.0/22 9
8.6.240.0/23 A
```

Device Management API

The API is a set of functions that provides an easy-to-use interface between the user application and the LPM hardware control and status registers. The API functions are implemented in the C programming language and conform to the C99 ANSI standard.

In order to use the API, the user application includes the appropriate header file in the user source code and calls the required functions. The API library is delivered as source code and can be compiled and linked together with the user application or as a separate static or dynamic library. The build system or the compilation scripts should ensure the correct setup of the 'include' and library paths.

The API is comprised of the following files.

- LPM.h –header file
- LPM.c – function implementation file

Data Structures

This section describes the data structures used by the LPM API functions to interface with the user application.

Device Context

```
typedef struct {
  uint32_t max_depth;
  uint32_t key_width;
  uint32_t value_width;
  uint32_t base;
  void(*register_write)(uint32_t addr, uint32_t data);
  uint32_t(*register_read)(uint32_t addr);
  int (*log_message)(const char * format, ...);
  uint32_tlog_level;
} LPM_CONTEXT;
```

The `LPM_CONTEXT` structure contains device context used to uniquely refer to an LPM instance and provides LPM API functions with pointers to system utility functions. The system can contain multiple LPM instances. Each LPM instance must be initialized separately and assigned a unique `LPM_CONTEXT` data structure validated via the `LPM_Init_ValidateContext()` API function call.

max_depth contains the maximum number of entries that can be stored in the CAM.

key_width contains the width of the entry key in bits.

value_width contains the width of the entry value in bits.

base is the starting address of the memory-mapped address space allocated to the LPM instance. The LPM API will add base to an internal offset when calling the provided *register_read* and *register_write* function pointers.

The *register_read* and *register_write* function pointers provide platform-specific access to the device memory space and must be initialized by the user code to point to implemented function entry points.

The *log_message* function pointer provides the platform-specific wrapper for reporting an informational message for logging. The LPM calls this function when an event occurs, passing a printf-formatted string argument, and a variadic list of arguments. Only %s, %d and %x format specifiers are used. The final string never exceeds 255 characters.

log_level sets the logging verbosity threshold as shown below.

- LPM_LOG_DISABLE = 0
  No log messages are printed, *log_message* can be NULL
- LPM_LOG_ERROR
  log_message is called for error messages
- LPM_LOG_WARNING
  log_message is called for warning and error messages
- LPM_LOG_INFO
  log_message is called for informational, warning, and error messages

Device Initialization

The initialization API provides constants and function for correct LPM instance and software initialization.

LPM_ADDR_SIZE

This define specifies the size of the LPM address space in bytes. Each LPM instance must be allocated `LPM_ADDR_SIZE` bytes in the system memory map. This define can be used to statically configure the memory map.

Note: `LPM_ADDR_SIZE` is tied to the version of the API and might change in future versions of the LPM.

LPM_Init_GetAddrSize()

This function returns the `LPM_ADDR_SIZE` and can be called at runtime to dynamically allocate or resize the configuration memory mapping of the LPM instance.

Note: `LPM_ADDR_SIZE` is tied to the version of the API and may change in future versions of the LPM.

Prototype

```
uint32_t LPM_Init_GetAddrSize();
```

Arguments

N/A

Return value

An integer indicating the size of the memory space in bytes.

LPM_Init_ValidateContext()

This function creates the instance context data structure.

Prototype

```
int LPM_Init_ValidateContext(
  LPM_CONTEXT* cx,
  uint32_t base,
  uint32_t size,
  uint32_t max_depth,
  uint32_t key_width,
  uint32_t value_width,
  uint32_t shadow_mem,
  void (*register_write)(uint32_t, uint32_t),
  uint32_t(*register_read)(uint32_t addr),
  int(*log_message)(const char *, ...),
  uint32_tlog_level
);
```

Arguments

- `cx` is a pointer to the `LPM_CONTEXT` structure to be initialized.
- `base` is the starting offset of the configuration memory address range assigned to this LPM instance.
- `size` is the size in bytes of the configuration memory address range assigned to this LPM instance.

  Note: `size` must be equal to or greater than `LPM_ADDR_SIZE`.
- `max_depth` contains the maximum number of entries that can be stored in the LPM.
- `key_width` contains the width of the entry key in bits.
- `value_width` contains the width of the entry value in bits.
- `shadow_mem` indicates whether the table uses a shadow memory for configuration and updates.
- `register_write` is a pointer to the register write function.
- `register_read` is a pointer to the register read function.
- `log_message` is a pointer to the formatted output log function.
- `log_level` is the verbosity level.

Return value

An integer indicating success or an error code. Zero (`LPM_SUCCESS`) indicates successful firmware image loading and activation. A non-zero value indicates an error. See the list of LPM error codes for details.

LPM_Init_SetLogLevel()

This function updates the logging level in the device context data structure.

Prototype

```
int LPM_Init_SetLogLevel(LPM_CONTEXT* cx, uint32 log_level);
```

Arguments

- `cx` is a pointer to the `LPM_CONTEXT` structure to be updated
- `log_level` is the new logging level.

Return value

An integer indicating success or an error code. Zero (`LPM_SUCCESS`) indicates successful firmware image loading and activation. A non-zero value indicates an error. See the list of LPM error codes for details.

LPM_Init_Activate()

This function performs device self-initialization and activation.

Prototype

```
int LPM_Init_Activate(LPM_CONTEXT* cx);
```

Arguments

`cx` is the pointer to the `LPM_CONTEXT` structure.

Return value

An integer indicating success or an error code. Zero (`LPM_SUCCESS`) indicates successful device initialization and activation. A non-zero value indicates an error. See the list of LPM error codes for details.

Device Management API

This API provides a set of functions for inserting and removing LPM rules and for configuring operational parameters.

LPM_Mgt_LoadDataset()

This function loads a new dataset into the LPM. In the case of versions having a shadow memory, the new dataset is loaded into the non-active bank, which must be made active before it can be used for lookups.

Prototype

```
int LPM_Mgt_LoadDataset(LPM_CONTEXT* cx, char* filename);
```

Arguments

- cx is a pointer to the `LPM_CONTEXT` structure
- `filename` is a pointer to a null-terminated C string containing the dataset filename.

Return value

An integer indicating success or an error code. Zero (`LPM_SUCCESS`) indicates successful dataset loading and activation. A non-zero value indicates an error. See the list of LPM error codes for details.

LPM_Mgt_VerifyDataset()

This function verifies the integrity of a dataset stored in the LPM against a file. In the case of versions having a shadow memory, the integrity of a dataset stored in the non-active bank of the LPM is verified against the file.

Prototype

```
int LPM_Mgt_VerifyDataset(LPM_CONTEXT* cx, char* filename);
```

Arguments

- `cx` is a pointer to the `LPM_CONTEXT` structure
- `filename` is a pointer to a null-terminated C string containing the dataset filename.

Return value

An integer indicating success or an error code. Zero (`LPM_SUCCESS`) indicates successful verification of the dataset. A non-zero value indicates an error. See the list of LPM error codes for details.

LPM_Mgt_SetActiveLookupBank()

This function sets which bank is active for versions having a shadow memory.

Prototype

```
Int LPM_Mgt_SetActiveLookupBank(LPM_CONTEXT* cx, uint32_t bank);
```

Arguments

`bank` is an integer identifying the logical bank number (0 or 1) to be active, It is used for lookup operations.

Return value

An integer indicating the success or an error code. Zero (`LPM_SUCCESS`) indicates successful switching of the bank. A non-zero value indicates an error. See the list of LPM error codes for details.

Errors

This section details the error condition, error code values, and the utility functions.

LPM_Error_Decode()

This function provides the runtime description for each error code.

Prototype

```
const char* LPM_Error_Decode(int error);
```

Arguments

`error` is a non-zero error code returned by the LPM API function.

Return value

A null-terminated string pointer containing a short description of the error code.

Error Codes

- LPM_SUCCESS = 0

  Successful completion of the operation. The value of this code is zero and is guaranteed to be fixed in all future versions of the API.

- LPM_ERROR_INIT_SIZE

  The size argument is not less than `LPM_ADDR_SIZE`.

- LPM_ERROR_INIT_NULL_FUNCPTR

  `register_write`, `register_read`, or `log_message` function pointer is NULL.

- LPM_ERROR_INIT_READ_MISMATCH

  Register reads to the LPM instance that it did not return the expected data written via register writes.

- LPM_ERROR_BANK_ACTIVATE_ERROR

  There was an error setting the new active bank.

- LPM_ERROR_BANK_ACTIVATE_INVALID_BANK

  The specified bank is out of range (valid bank options are 0 or 1).

- LPM_ERROR_INIT_LOG

  `log_message` did not return the expected number of arguments on output of initialization messages.

- LPM_ERROR_NULL_CONTEXT_PTR

  Context pointer is NULL.

- LPM_ERROR_INVALID_CONTEXT

  Context magic number mismatch.

- LPM_ERROR_INIT_SIZE

  Context size is too small for this instance version.

- LPM_ERROR_INIT_NULL_FUNC_PTR

  Context function pointer is NULL.

- PM_ERROR_INIT_READ_MISMATCH

  Register read data does not match previously written value.

- LPM_ERROR_INIT_LOG

  `log_message` function return value does not match expected value.

- LPM_ERROR_KEY_NULL

  `key` string pointer is NULL.

- PM_ERROR_VALUE_NULL

  `value` string pointer is NULL.

- LPM_ERROR_KEY_FORMAT

  `Invalid` key string format.

- LPM_ERROR_VALUE_FORMAT

  `Invalid` value string format.

- LPM_ERROR_ACC

  Miscellaneous device access error.

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

Documentation Navigator and Design Hubs

Xilinx® Documentation Navigator provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open the Xilinx Documentation Navigator (DocNav):

- On Windows, select **Start > All Programs > Xilinx Design Tools > DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In the Xilinx Documentation Navigator, click the **Design Hubs View** tab.
- On the Xilinx website, see the Design Hubs page.

Note: For more information on Documentation Navigator, see the Documentation Navigator page on the Xilinx website.

References

These documents provide supplemental material useful with this product guide:

1. *AMBA AXI4-Stream Protocol Specification*
2. PCI-SIG Documentation (www.pcisig.com/specifications)
   - *PCI Express Base Specification 1.1*
   - *PCI Express Card Electromechanical (CEM) Specification 1.1*
3. *SDNet Packet Processor User Guide* (UG1012)
4. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
5. *Vivado Design Suite User Guide: Designing with IP* (UG896)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
11/10/2017	1.0	Initial public release. The document title has been changed from *Longest Prefix Match SmartCORE IP Product Guide* to *Longest Prefix Match (LPM) Search IP for SDNet SmartCORE IP Product Guide*.

Notice of Disclaimer