Ternary Content
Addressable Memory
(TCAM) Search IP for
SDNet

SmartCORE IP Product Guide

PG190 (v1.0) November 10, 2017
# Table of Contents

**Chapter 1: Overview**
- Licensing and Ordering ......................................................... 6

**Chapter 2: Product Specification**
- Performance ................................................................. 8
- Resource Utilization ....................................................... 10
- Port Descriptions .......................................................... 10
- Register Space .............................................................. 13

**Chapter 3: Designing with the Core**
- General Design Guidelines ............................................... 16
- Clocking ................................................................. 16
- Resets ................................................................. 16
- Protocol Description ..................................................... 17

**Chapter 4: Design Flow Steps**
- Configuration ............................................................. 19
- Constraining the Core .................................................... 20
- Simulation .............................................................. 21
- Synthesis and Implementation ........................................... 21

**Appendix A: Upgrading**

**Appendix B: Debugging**
- Finding Help on Xilinx.com ............................................... 23
- Debug Tools ............................................................ 24
- Hardware Debug .......................................................... 24
- Interface Debug ............................................................ 25

**Appendix C: Application Software Development**
- Data Structures ........................................................... 27
- Device Initialization ....................................................... 28
- Device Management API Functions .................................... 31
Appendix D: Additional Resources and Legal Notices

Xilinx Resources ................................................................. 34
Documentation Navigator and Design Hubs ........................................ 34
References ........................................................................... 35
Revision History ..................................................................... 35
Notice of Disclaimer ............................................................. 36
## Introduction

The TCAM SmartCORE™ IP is a ternary content-addressable memory that stores (data, mask, value) entries with arbitrary data, mask and value bit strings. The mask is used to mark certain bits in the key field as “don’t care”. The retrieval of ‘value’ is based on a ternary match of ‘key’ with (data, mask).

## Features

- Associative array containing arbitrary {key, mask, value} sets
- Ternary match key lookup returns hit/miss result and associated data and address on hit
- High throughput: one lookup per clock cycle.
- One- and two-range comparisons in each rule supported efficiently without rule expansion
- Scalable, supporting a wide range of key, mask, and value field widths
- Supports both distributed RAM and block RAM implementations

### SmartCORE IP Facts Table

<table>
<thead>
<tr>
<th>Core Specifics</th>
<th>Supported Device Family</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Kintex® UltraScale™</td>
</tr>
<tr>
<td></td>
<td>Virtex® UltraScale</td>
</tr>
<tr>
<td></td>
<td>Virtex UltraScale+</td>
</tr>
<tr>
<td></td>
<td>Zynq® UltraScale+ MPSoC</td>
</tr>
<tr>
<td></td>
<td>Kintex-7, Virtex-7</td>
</tr>
<tr>
<td></td>
<td>Spartan®-7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Supported User Interfaces</th>
<th>Lookup and AXI4-Lite Interfaces</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Resources</th>
<th>Provided with Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encrypted register transfer level (RTL) or Netlist</td>
<td>Design Files</td>
</tr>
<tr>
<td>Not Provided</td>
<td>Example Design</td>
</tr>
<tr>
<td>System Verilog</td>
<td>Test Bench</td>
</tr>
<tr>
<td>Not Provided</td>
<td>Constraints File</td>
</tr>
<tr>
<td>Verilog</td>
<td>Simulation Model</td>
</tr>
<tr>
<td>Standalone</td>
<td>Supported S/W Driver(1)</td>
</tr>
</tbody>
</table>

### Tested Design Flows

<table>
<thead>
<tr>
<th>Design Entry</th>
<th>SDNet(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation</td>
<td>Vivado® Simulator</td>
</tr>
<tr>
<td>Mentor Graphics Questa Advanced Simulator</td>
<td>Synthesis</td>
</tr>
</tbody>
</table>

### 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 4].
Overview

The TCAM stores {data, mask, value} entries in RAM which can be either distributed RAM or block RAM, based on the configuration. The TCAM provides efficient use of Xilinx® FPGA resources, in contrast with basic TCAM implementations that store the keys in flip-flops and use logic resources for parallel key comparison.

The Lookup interface of the TCAM receives a lookup key and outputs a result that contains a match flag indicating whether the key matches any entry in the TCAM. If any TCAM entry is matched, the address and the associated value of the entry with the highest priority is output. The address of an entry in the TCAM determines its priority, i.e., an entry with a lower address has a higher priority than one with a higher address. The TCAM is pipelined so that it can process a lookup request every clock cycle.

In addition to ternary matching, the TCAM supports range comparisons, e.g., for TCP/UDP port range checking. Up to two sub-fields of the key can optionally be designated for numerical range checking ([min:max]), as opposed to the data/mask matching. The position and the width of these sub-fields of the key are statically configured according to user specifications.

The rule entries are read and written using a set of high-level Application Programming Interface (API) functions. The API functions are written in C that is compliant with the ISO/IEC 9899:1999 standard, and delivered as part of the IP. The API encapsulates the details of configuration register access and lookup/update scheduling semantics and provides you with a simple and efficient interface. An AXI4-Lite interface provides a communication link between the API software and the hardware configuration and status registers.

Examples of rules that can be programmed directly into the TCAM using the provided API functions are shown in Table 1-1.

Table 1-1: Example of TCAM Rules

<table>
<thead>
<tr>
<th>IPv4 Src Addr/Mask</th>
<th>IPv4 Dst Addr/Mask</th>
<th>TCP Src Port</th>
<th>TCP Dst Port</th>
<th>Protocol/ Mask</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>198.238.184.78/32</td>
<td>142.205.117.12/30</td>
<td>0:65535</td>
<td>0:65535</td>
<td>0x00/0x00</td>
<td>0</td>
</tr>
<tr>
<td>0.0.0.0/0</td>
<td>198.238.186.122/32</td>
<td>0:65535</td>
<td>25:25</td>
<td>0x11/0xFF</td>
<td>2</td>
</tr>
<tr>
<td>0.0.0.0/0</td>
<td>198.238.190.245/32</td>
<td>0:65535</td>
<td>8080:8080</td>
<td>0x11/0xFF</td>
<td>3</td>
</tr>
<tr>
<td>0.0.0.0/0</td>
<td>198.238.190.174/32</td>
<td>0:65535</td>
<td>67:67</td>
<td>0x11/0xFF</td>
<td>0</td>
</tr>
</tbody>
</table>
The rules in the table consist of five fields as follows:

- IPv4 Source Address and mask
- IPv4 Destination Address and mask
- TCP source port [min:max] range
- TCP destination port [min:max] range
- IPv4 protocol number and mask

---

**Licensing and Ordering**

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

For more information, visit the [SDNet Development Environment page](#).
Chapter 2

Product Specification

This chapter includes information about the performance, resource utilization, and the port descriptions of the TCAM.

The TCAM (see Figure 2-1) contains the following sub-blocks:

- Rule Database
  Storage for rules and algorithmic lookup matching
- Lookup Request FIFO
  Buffering queue for input lookup requests
- Update FSM
  Rule database access management and FIFO dequeue control
- Register Map
  Configuration and status registers
- AXI4-Lite interface slave
  Protocol handling for accepting read/write requests and generating responses
Note: FSM stands for Finite State Machine.

The rule database stores the \(\{\text{data, mask, value}\}\) entries and performs lookup matching. The block is fully-pipelined and can accept a new lookup request on each clock cycle. However when the rule database is being accessed by the API for reading or writing rules, the rule database is not available for lookup matching. In order to fully decouple the lookup interface from the API database management, the design uses a FIFO for buffering the input lookup requests while the database is not accessible during updates.

The duration of the update phase depends on the storage configuration option:

- For entries stored in distributed RAM: 33 clock cycles
- For entries stored in block RAM: 513 clock cycles

The Update FSM stops the de-queuing of the Lookup Request FIFO on a new API read/write request for the duration of the operation. The FIFO is sized to provide enough buffering in order not to overflow during the update operation. The Update FSM re-starts the lookup request dequeue operation after the completion of the update. The Update FSM will not schedule another update operation prior to the draining of the Lookup Request FIFO, which is guaranteed when the operating clock frequency exceeds the lookup request rate.

For example, for a 100 Gb/s Ethernet interface that has a maximum packet rate of 150 Mp/s, if each packet requires a single lookup in the TCAM, this corresponds to a minimum TCAM frequency of 150 MHz. A higher clock frequency allows for faster draining of the FIFO and increases the rate at which the TCAM can process the update requests from the API. The Throughput section shows the update rate calculation as a function of the difference between the clock frequency and the lookup rate.

---

### Performance

#### Maximum Frequencies

The TCAM is designed to run at 170 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 TCAM latency depends on both the configuration and the fill level of the Lookup Request FIFO. The latency is constant when no updates requests are active and is shown in Table 2-1 in terms of clock cycles.
Chapter 2: Product Specification

Table 2-1: TCAM Latency

<table>
<thead>
<tr>
<th>Width (bits)</th>
<th>20</th>
<th>40</th>
<th>80</th>
<th>160</th>
<th>320</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;=256 Depth</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>512 Depth</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>1024 Depth</td>
<td>6</td>
<td>6</td>
<td>10</td>
<td>14</td>
<td>22</td>
</tr>
</tbody>
</table>

Note: If updates are active, there is an additional latency due to the Lookup Request FIFO fill level, which can be between 0 and 33 cycles for a distributed RAM implementation and between 0 and 513 cycles for a block RAM implementation.

Throughput

One new search request can be issued per cycle. The lookup and update throughput of the TCAM can be expressed as follows, assuming one lookup per packet.

The example frequency of 170 MHz is suggested for over-clocking beyond the required 150 MLPS search rate; this creates "bubble" cycles available for performing updates.

Lookup Rate [M/s] = Clk Frequency [MHz]

For distributed RAM:

- Update Rate [M/s] = (Clk Frequency [MHz] - Packet Rate [Mp/s]) / 33

For block Ram:

- Update Rate [M/s] = (Clk Frequency [MHz] - Packet Rate [Mp/s]) / 513

Note: The Clk Frequency [MHz] must be higher than the Packet Rate [Mp/s] to accommodate updates. With a clock frequency of 170 MHz, TCAM throughput for 100Gb/s Ethernet is shown in Table 2-2.

Table 2-2: TCAM Throughput at Clk = 170 MHz

<table>
<thead>
<tr>
<th>Memory</th>
<th>Lookup Rate (M/s)</th>
<th>Update Rate (K/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributed RAM</td>
<td>150</td>
<td>600</td>
</tr>
<tr>
<td>Block Ram</td>
<td>150</td>
<td>39</td>
</tr>
</tbody>
</table>
Chapter 2: Product Specification

Resource Utilization

Virtex-7 Devices

Table 2-3 provides approximate resource counts for the various core options on Virtex®-7 devices.

Table 2-3: Device Resources and Maximum Frequency in Virtex-7 FPGAs

<table>
<thead>
<tr>
<th>Key width (K), Value width (V), Depth (D), block RAM (B), N_RANGE (R), RANGE_SIZE (S), RANGE_OFFSET (O)</th>
<th>Slices</th>
<th>LUTs</th>
<th>FFs</th>
<th>Block RAMs</th>
<th>Max Freq (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>K=64 V=16 D=256 B=0 R=0 S=X O=X</td>
<td>2562</td>
<td>8207</td>
<td>4459</td>
<td>2</td>
<td>173</td>
</tr>
<tr>
<td>K=128 V=16 D=512 B=0 R=1 S=16 O=0</td>
<td>12011</td>
<td>36434</td>
<td>35068</td>
<td>3</td>
<td>171</td>
</tr>
<tr>
<td>K=256 V=16 D=1024 B=0 R=2 S=16 O=0</td>
<td>44272</td>
<td>141674</td>
<td>138691</td>
<td>5</td>
<td>170</td>
</tr>
</tbody>
</table>

Notes:
1. See Chapter 4 for a detailed description of the parameters.
2. Maximum frequency when targeting a -2 speed grade.

Port Descriptions

The following parameters are a subset of the synthesis-time configuration parameters. They are listed here because they affect the bus width of some of the interface signals. A complete list of the synthesis-time configuration parameters is given in Chapter 4.
System Interface

Table 2-4: System Interface

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Direction</th>
<th>Width</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rst</td>
<td>Input</td>
<td>1</td>
<td>Synchronous active-high reset</td>
</tr>
<tr>
<td>Clk</td>
<td>Input</td>
<td>1</td>
<td>System clock</td>
</tr>
</tbody>
</table>

Lookup Interface

The Lookup interface is used to issue search requests and receive the corresponding results.

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

Table 2-5: Lookup Interface

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Direction</th>
<th>Width</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LookupReqValid</td>
<td>Input</td>
<td>1</td>
<td>Lookup Request Valid: When 1, indicates that LookupKey is valid.</td>
</tr>
<tr>
<td>LookupReqKey</td>
<td>Input</td>
<td>K</td>
<td>Lookup Request Key: Valid when LookupReqValid is asserted.</td>
</tr>
<tr>
<td>LookupRespValid</td>
<td>Output</td>
<td>1</td>
<td>Lookup Response Valid: Asserted after a variable number of clock cycles following the assertion of LookupReqValid to indicate that LookupRespMatch is valid.</td>
</tr>
<tr>
<td>LookupRespMatch</td>
<td>Output</td>
<td>1</td>
<td>Lookup Response Match: Indicates the input key is matching at least one entry in the TCAM. Valid when LookupRespValid = 1.</td>
</tr>
<tr>
<td>LookupRespKey</td>
<td>Output</td>
<td>K</td>
<td>Lookup Response Key: A copy of the key requested via LookupReqKey, valid when LookupRespValid=1.</td>
</tr>
<tr>
<td>LookupRespAddr</td>
<td>Output</td>
<td>Log2(D)</td>
<td>The address of the matching entry with the highest priority. Valid when LookupRespMatch = 1.</td>
</tr>
<tr>
<td>LookupRespValue</td>
<td>Output</td>
<td>V</td>
<td>Response Value: Outputs the value associated with the matching entry. Valid when LookupRespMatch = 1.</td>
</tr>
</tbody>
</table>
## AXI4-Lite Interface

The AXI4-Lite interface provides the register read/write interface for the CPU running the API software. Refer to the *AMBA AXI4-Stream Protocol Specification* [Ref 1] for the detailed description of the interface signals and the details of protocol operation.

Table 2-6: AXI4-Lite Interface

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Direction</th>
<th>Width</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AXI Global System Signals</strong> (1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S_AXI_ACLK</td>
<td>Input</td>
<td>1</td>
<td>AXI Clock</td>
</tr>
<tr>
<td>S_AXI_ARESETN</td>
<td>Input</td>
<td>-</td>
<td>AXI Reset, active-Low</td>
</tr>
<tr>
<td><strong>AXI Write Address Channel Signals</strong> (1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S_AXI_AWADDR</td>
<td>Input</td>
<td>32</td>
<td>AXI Write address. The write address bus gives the address of the write transaction.</td>
</tr>
<tr>
<td>S_AXI_AWVALID</td>
<td>Input</td>
<td>1</td>
<td>Write address valid. This signal indicates that valid write address and control information are available.</td>
</tr>
<tr>
<td>S_AXI_AWREADY</td>
<td>Output</td>
<td>1</td>
<td>Write address ready. This signal indicates that the slave is ready to accept an address and associated control signals.</td>
</tr>
<tr>
<td><strong>AXI Write Data Channel Signals</strong> (1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S_AXI_WDATA</td>
<td>Input</td>
<td>32</td>
<td>Write data</td>
</tr>
<tr>
<td>S_AXI_WSTB</td>
<td>Input</td>
<td>4</td>
<td>Write strobes. This signal indicates which byte lanes to update in memory. Note that individual byte masking is not supported. During a write, all four write strobe must be active.</td>
</tr>
<tr>
<td>S_AXI_WVALID</td>
<td>Input</td>
<td>1</td>
<td>Write valid. This signal indicates that valid write data and strobes are available.</td>
</tr>
<tr>
<td>S_AXI_WREADY</td>
<td>Output</td>
<td>1</td>
<td>Write ready. This signal indicates that the slave can accept the write data.</td>
</tr>
<tr>
<td><strong>AXI Write Response Channel Signals</strong> (1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S_AXI_BRESP</td>
<td>Output</td>
<td>2</td>
<td>Write response. This signal indicates the status of the write transaction. 00 - OKAY 10 - SLVERR</td>
</tr>
<tr>
<td>S_AXI_BVALID</td>
<td>Output</td>
<td>1</td>
<td>Write response valid. This signal indicates that a valid write response is available.</td>
</tr>
<tr>
<td>S_AXI_BREADY</td>
<td>Input</td>
<td>1</td>
<td>Response ready. This signal indicates that the master can accept the response information.</td>
</tr>
</tbody>
</table>
Chapter 2: Product Specification

Register Space

The register space is used for software control, monitoring and management of the TCAM IP. This information is provided for reference; the provided API functions should be used for controlling the TCAM IP instead of accessing the register space directly. The register space is documented in relative offsets. During AXI system initialization, the TCAM IP is assigned an absolute memory address range.

Table 2-7: Offset = 0X00 Unique Device ID

<table>
<thead>
<tr>
<th>Field name</th>
<th>Bits</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unique Device ID</td>
<td>31:0</td>
<td>Read-only</td>
<td>Unique Device ID. Used by the API functions to ensure compatibility.</td>
</tr>
</tbody>
</table>
The number of data/mask registers \( KN \) is equal to \( \text{ceil}(K / 32) \).

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

Data registers occupy the offset range of \([0x50: 0x50 + (KN\times4) - 1]\). Because \( K \) never exceeds 512, the maximum range of the data register offset is within \([0x50:0x8F]\).


Table 2-11:  Offset = 0x50 + 4*n: Data[n]

<table>
<thead>
<tr>
<th>Field name</th>
<th>Bits</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data[n]</td>
<td>31:0</td>
<td>Read-write</td>
<td>Bits [max(K, 31+32<em>n): 32</em>n] of the data or a min portion of the numeric range for [min:max] comparison. Loaded by software prior to issuing a write or verify operation. Not used for the initialization operation.</td>
</tr>
</tbody>
</table>

Mask registers occupy the offset range of [0x90: 0x90 + (KN*4) – 1]. Because K never exceeds 512, the maximum range of the mask register offset is within [0x90:0xCF].

Table 2-12:  Offset = 0x90 + 4*n: Mask[n]

<table>
<thead>
<tr>
<th>Field name</th>
<th>Bits</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mask[n]</td>
<td>31:0</td>
<td>Read-write</td>
<td>Bits [max(K, 31+32<em>n): 32</em>n] of the mask, or a max portion of the numeric range for [min:max] comparison. Loaded by software prior to issuing a write or verify operation. Not used for the initialization operation.</td>
</tr>
</tbody>
</table>

The number of value registers VN is equal to ceil(V /32).

Using integer division, VN = (V – 1) / 32 + 1.

Value registers occupy the offset range of [0xD0: 0xD0 + (VN*4) – 1]. Because V never exceeds 256, the maximum range of offset value register offsets is within [0xD0:0xEF].

Table 2-13:  Offset = 0xD0 + 4*N: Value[n]

<table>
<thead>
<tr>
<th>Field name</th>
<th>Bits</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value[n]</td>
<td>31:0</td>
<td>Read-write</td>
<td>Bits [max(V, 31+32<em>n): 32</em>n] of the value. Loaded by software prior to issuing a write or verify operation. Not used for the initialization operation.</td>
</tr>
</tbody>
</table>
Chapter 3

Designing with the Core

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

General Design Guidelines

Implementation in Distributed RAM vs. Block RAM

Implementation in distributed RAM provides higher memory usage efficiency than in block Ram because distributed RAM has smaller granularity. However, if a particular design has more block RAM spare capacity in the target device, block RAM can be used even if it is less efficient than distributed RAM because it will provide better overall device utilization.

Clocking

In general, the clock frequency should be higher than the packet rate to allow for background updates. See Throughput in Chapter 2 for details.

Resets

At startup, both the AXI reset and the main reset must be asserted simultaneously for 100 cycles of the slower of the two clocks (AXI clock and main clock). As long as the reset assertion time is met, either reset can be asserted or negated first.
Protocol Description

Interface Operation

This section describes the operation of the TCAM Lookup interface. The API functions are used for reading and writing the TCAM entries.

Reset and clock

The TCAM operates the Lookup interface synchronously to the main clock. The clock period must satisfy the constraints described in Throughput in Chapter 2.

The TCAM uses a synchronous active-High reset. Reset must have a minimum assertion pulse width of 100 cycles to fully clear internal pipelines. Reset does not have a maximum assertion time, and can be held asserted for any duration longer than minimum assertion time as required by other blocks in the system. Reset negation must meet setup timing relative to the rising edge of the clock.

The AXI4-Lite interface includes additional clock and reset inputs. Both the AXI reset and the main reset must be asserted simultaneously for 100 cycles of the slower of the two clocks (AXI clock and main clock.) As long as the reset assertion time is met, either reset can be asserted or negated first.

Lookup Interface

The Lookup interface is used to check for the presence of a particular key in the TCAM data structure. If the key is present, the corresponding value is output.

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 keys and returns the corresponding addresses and values.

Lookup Request

The Lookup Request interface is fully-pipelined and is able to process a lookup request every clock cycle. LookupReqKey must be valid when LookupReqValid is asserted and must contain the key being looked up (See Appendix C).
**Lookup Response**

The Lookup Response interface reports the result of looking up the key scheduled on the Lookup Request interface. LookupRespValid is asserted with a variable latency with respect to LookupReqValid. See Latency in Chapter 2 for a description of the latency calculation.

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 LookupReqKey in the TCAM IP when asserted. When LookupRespHit is asserted, it also validates the LookupRespValue output, which contains the value portion of the (data, mask, value) entry stored in the TCAM. LookupRespAddr is also validated by the LookupRespValid and contains the address of the matching rule.

When LookupRespHit is negated, it means LookupRespKey was not found and the LookupRespValue/LookupRespAddr signals are invalid.

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

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

**Note:** If more than one entry matches the (data, mask) being looked up, the entry with the highest priority (lowest address) is output.
Chapter 4

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 TCAM design is highly configurable at compile time to make it suitable for a large variety of applications. Table 4-1 lists available configuration parameters. Note that changing parameters requires design synthesis and generation of a new FPGA bitstream.

Table 4-1: Configuration Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Valid Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>1-512 bits</td>
<td>Key/Data/Mask width.</td>
</tr>
<tr>
<td>V</td>
<td>1-256 bits</td>
<td>Value width.</td>
</tr>
<tr>
<td>D</td>
<td>1-4096 entries</td>
<td>Maximum number of entries.</td>
</tr>
<tr>
<td>BRAM</td>
<td>0-1</td>
<td>Specify whether RAM is implemented using distributed RAM (0) or block Ram (1).</td>
</tr>
<tr>
<td>N_RANGE</td>
<td>0-2 ranges</td>
<td>Number of range comparisons.</td>
</tr>
<tr>
<td>RANGE_SIZE</td>
<td>2-16 bits</td>
<td>Width the range(s). If two range comparisons are configured, both will have the same width.</td>
</tr>
<tr>
<td>RANGE_OFFSET</td>
<td>0-(K – N_RANGE*RANGE_SIZE – 1)</td>
<td>Bit offset of the first range in the key. A value of 0 places the Range1 MSB at bit 0 of the key. Both range bit fields must be adjacent, e.g., (KEY_MSBs, RANGE2, RANGE1, KEY_LSBs).</td>
</tr>
<tr>
<td>INSTANCE_NAME</td>
<td>Text</td>
<td>Instance name used for the top level instance, e.g., “TCAM_1”</td>
</tr>
</tbody>
</table>

Notes:
1. The range comparison feature for specifying up to two integer ranges is not enabled in the current version of SDNet. This feature will be enabled in a future version of SDNet.

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 the number of keys that must be stored in the TCAM.
• BRAM must be set to 0 if using distributed RAM or 1 if using block RAM.
• N_RANGE must be set to the number of range comparisons in the rules (1 or 2). It must be set to 0 if the rules do not include range comparisons.
• RANGE_SIZE must be set to the width of the range(s) in the key, if any. Valid values must be between 2 and 16.
• RANGE_OFFSET must be set to the bit position of the RANGE1 LSB in the key.

Output Generation

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

<table>
<thead>
<tr>
<th>Filenames</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;INSTANCE_NAME&gt;/rtl/&lt;INSTANCE_NAME&gt;.v</code></td>
<td>Encrypted Verilog source code</td>
</tr>
<tr>
<td><code>&lt;INSTANCE_NAME&gt;/model/model.sv</code></td>
<td>Behavioral model of the TCAM</td>
</tr>
<tr>
<td><code>&lt;INSTANCE_NAME&gt;/sim/run.do</code></td>
<td>Questa Advanced Simulator command file</td>
</tr>
<tr>
<td><code>&lt;INSTANCE_NAME&gt;/tb/tb.sv</code></td>
<td>Demonstration test bench</td>
</tr>
<tr>
<td><code>&lt;INSTANCE_NAME&gt;/api/&lt;INSTANCE_NAME&gt;.h</code></td>
<td>Software driver files.</td>
</tr>
<tr>
<td><code>&lt;INSTANCE_NAME&gt;/api/xilinx_tcam.h</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;INSTANCE_NAME&gt;/api/xilinx_tcam.c</code></td>
<td></td>
</tr>
</tbody>
</table>

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 TCAM. An example for constraining a primary clock net called clk170 that drives the Clk input at 170 MHz follows.

```
create_clock -period 5.882 -name clk170 get_ports clk1
```
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 6].

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.
Appendix B

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 TCAM, 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 TCAM. 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 TCAM 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 managers (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 the Xilinx AMBA AXI4 Interface Protocol page [Ref 3] 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_arreset 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.
Appendix C

Application Software Development

This appendix describes the Application Programming Interface (API) functions for the TCAM SmartCORE™ IP. The API functions provide an easy-to-use interface between the user application and the TCAM hardware control and status registers. The API functions are implemented in the C programming language and conform to the ISO/IEC 9899:1999 standard, which is commonly referred to as “C99”.

The API library does not have any external dependencies and does not perform file access, I/O access, or memory allocation/deallocation.

In order to use the API functions, the user application includes the appropriate header file in the 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.

- `<INSTANCE_NAME>.h` – instance-specific macro definitions
- `TCAM.h` – API library header file
- `TCAM.c` – API library functions
Data Structures

This section describes the data structures used by the TCAM 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    num_ranges;
    uint32_t    range_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_t log_level;
} TCAM_CONTEXT;

The TCAM_CONTEXT structure contains device context information used to uniquely refer to a TCAM instance and pointers to system utility functions used by the TCAM API functions. The system can contain multiple TCAM instances. Each TCAM instance must be initialized separately and assigned a unique TCAM_CONTEXT data structure validated via the TCAM_Init_ValidateContext() API function call.

- **max_depth** contains the maximum number of entries that can be stored in the TCAM.
- **key_size** contains the width of the entry key in bits.
- **value_size** contains the width of the entry value in bits.
- **num_ranges** is the number of numeric min:max range subfields.
- **range_width** is the width of the range subfields.
- **range_offset** is the bit position of the LSB of the first numeric subfield within the key.
- **base** is the starting address of the memory-mapped address space allocated to the TCAM instance. The TCAM API functions will add the base address to an internal offset when calling the provided register_read and register_write function pointers.
- **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.
- **log_message** is a function pointer that provides the platform-specific wrapper for reporting an informational message for logging. The TCAM 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 using the following settings.
  - `TCAM_LOG_DISABLE = 0`
    No log messages are printed, `log_message` can be NULL.
  - `TCAM_LOG_ERROR`
    `log_message` is called for error messages.
  - `TCAM_LOG_WARNING`
    `log_message` is called for warning and error messages.
  - `TCAM_LOG_INFO`
    `log_message` is called for informational, warning, and error messages.

---

**Device Initialization**

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

**TCAM_ADDR_SIZE**

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

*Note:* TCAM_ADDR_SIZE is tied to the version of the API and might change in future versions of the TCAM.

**TCAM_Init_GetAddrSize()**

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

*Note:* TCAM_ADDR_SIZE is tied to the version of the API and might change in future versions of the TCAM.

**Prototype**

```c
uint32_t TCAM_Init_GetAddrSize();
```

**Arguments**

N/A
Appendix C: Application Software Development

Return value

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

**TCAM_Init_ValidateContext()**

This function creates the instance context data structure.

Prototype

```c
int TCAM_Init_ValidateContext(
    TCAM_CONTEXT* cx,
    uint32_t base,
    uint32_t size,
    uint32_t max_depth,
    uint32_t key_width,
    uint32_t value_width,
    uint32_t num_ranges,
    uint32_t range_width,
    uint32_t range_offset,
    void (*register_write)(uint32_t, uint32_t),
    uint32_t(*register_read)(uint32_t addr),
    int (*log_message)(const char *, ...),
    uint32_t log_level
);
```

Arguments

- **cx.** A pointer to the TCAM_CONTEXT structure to be initialized.
- **base.** Starting offset of the configuration memory address range assigned to this TCAM instance.
- **size.** Size in bytes of the configuration memory address range assigned to this TCAM instance.

**Note:** Must be equal to or greater than TCAM_ADDR_SIZE.
- **max_depth.** Maximum number of entries that can be stored in the TCAM.
- **key_width.** Width of the entry key in bits.
- **value_width.** Width of the entry value in bits.
- **num_ranges.** Number of numeric [min:max] range sub-fields within the key.
- **range_width.** Width of the numeric sub-fields.
- **range_offset.** Bit position of the LSB of the first range sub-field within the key.
- **register_write.** Pointer to the register write function.
- **register_read.** Pointer to the register read function.
- **log_message.** Pointer to the formatted output log function.
- **log_level.** Verbosity level.
Return value

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

TCAM_Init_SetLogLevel()

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

Prototype

```c
int TCAM_Init_SetLogLevel(TCAM_CONTEXT* cx, uint32 log_level);
```

Arguments

- `cx`. A pointer to the TCAM_CONTEXT structure to be updated.
- `log_level`. A new logging level.

Return value

An integer indicating success or an error code. Zero (TCAM_SUCCESS) indicates successful firmware image loading and activation. Non-zero value indicates an error. See the list of TCAM error codes for details.

TCAM_Init_Activate()

This function performs device self-initialization, erasing and invalidating all stored rules.

Prototype

```c
int TCAM_Init_Activate(TCAM_CONTEXT* cx);
```

Arguments

- `cx`. A pointer to the TCAM_CONTEXT structure.

Return value

An integer indicating success or an error code. Zero (TCAM_SUCCESS) indicates successful device initialization and activation. A non-zero value indicates an error. See the list of TCAM error codes for details.
Device Management API Functions

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

**TCAM\_Mgt\_WriteEntry()**

This function writes a valid entry into the TCAM.

**Prototype**

```c
int TCAM_Mgt_WriteEntry(TCAM_CONTEXT* cx, uint_32 addr, char* data, char* mask, char* value);
```

**Arguments**

- `cx`. A pointer to the TCAM\_CONTEXT structure.
- `addr`. TCAM location to overwrite.
- `data`. A pointer to a null-terminated C string encoding the key in hexadecimal.
- `mask`. A pointer to a null-terminated C string encoding the mask in hexadecimal.
- `value`. A pointer to a null-terminated C string encoding the value in hexadecimal.

**Return value**

An integer indicating success or an error code. Zero (TCAM\_SUCCESS) indicates successful key insertion or in-place update. A non-zero value indicates an error. See the list of TCAM error codes for details.

**TCAM\_Mgt\_EraseEntry()**

This function invalidates an entry in the TCAM.

**Prototype**

```c
int TCAM_Mgt_EraseEntry(TCAM_CONTEXT* cx, uint32_t addr);
```

**Arguments**

- `cx`. A pointer to the TCAM\_CONTEXT structure
- `addr`. TCAM location to overwrite.
Return value

An integer indicating success or an error code. Zero (TCAM_SUCCESS) indicates successful key deletion. A non-zero value indicates an error. See the list of TCAM error codes for details.

**TCAM_Mgt_VerifyEntry()**

This function performs a “verify” operation for an entry.

**Prototype**

```c
int TCAM_Mgt_VerifyEntry(TCAM_CONTEXT* cx, uint_32 addr, char* data, char* mask, char* value);
```

**Arguments**

- `cx`. A pointer to the TCAM_CONTEXT structure.
- `addr`. TCAM location to verify.
- `data`. A pointer to a null-terminated C string encoding the key in hexadecimal.
- `mask`. A pointer to a null-terminated C string encoding the mask in hexadecimal.
- `value`. A pointer to a null-terminated C string encoding the value in hexadecimal.

**Return value**

An integer indicating success or an error code. Zero (TCAM_SUCCESS) indicates successful key verification. A non-zero value indicates an error. See the list of TCAM error codes for details.

**Errors**

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

**TCAM_Error_Decode()**

This function provides the runtime description of each error code.

**Prototype**

```c
const char* TCAM_Error_Decode( int error);
```

**Arguments**

- `error`. A non-zero error code returned by the TCAM API function.
Appendix C: Application Software Development

Return value

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

Error Codes

- **TCAM_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.

- **TCAM_ERROR_INIT_SIZE**
  The size argument is not less than TCAM_ADDR_SIZE.

- **TCAM_ERROR_INIT_NULL_FUNCPTR:**
  *register_write, register_read, or log_message* function pointer is NULL.

- **TCAM_ERROR_INIT_READ_MISMATCH**
  Register reads to TCAM instance did not return the expected data written via register writes.

- **TCAM_ERROR_INIT_LOG**
  *log_message* did not return the expected number of arguments on output of init messages.

- **TCAM_ERROR_NULL_CONTEXT_PTR**
  Context pointer is NULL.

- **TCAM_ERROR_INVALID_CONTEXT**
  Context magic number mismatch.

- **TCAM_ERROR_INIT_SIZE**
  Context size too small for this instance.

- **TCAM_ERROR_KEY_NULL**
  Key string pointer is NULL.

- **TCAM_ERROR_VALUE_NULL**
  Value string pointer is NULL.

- **TCAM_ERROR_KEY_FORMAT**
  Invalid key string format.

- **TCAM_ERROR_VALUE_FORMAT**
  Invalid value string format.

- **TCAM_ERROR_MASK_NULL**
  Mask string pointer is NULL.

- **TCAM_ERROR_MASK_FORMAT**
  Invalid mask string format.

- **TCAM_ERROR_VERIFY_NO_MATCH**
  Entry verification failed (did not match).

- **TCAM_ERROR_ACC**
  Miscellaneous device access error.
Appendix D

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. Xilinx AMBA AXI4 Interface Protocol page
3. PCI-SIG Documentation (www.pcisig.com/specifications)
   - PCI Express Base Specification 1.1
   - PCI Express Card Electromechanical (CEM) Specification 1.1
6. SDNet Packet Processor User Guide (UG1012)

Revision History

The following table shows the revision history for this document.

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>11/10/2017</td>
<td>1.0</td>
<td>Initial public release. The document title has been changed from Ternary Content Addressable Memory SmartCORE IP to Ternary Content Addressable Memory (TCAM) Search IP for SDNet SmartCORE IP Product Guide.</td>
</tr>
</tbody>
</table>
Appendix D: Additional Resources and Legal Notices

Notice of Disclaimer

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos.

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

© Copyright 2013 - 2017 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.