



XAPP1151 (v1.0) March 1, 2011

# Parameterizable Content-Addressable Memory

Author: Kyle Locke

## Summary

This application note describes a parameterizable content-addressable memory (CAM), and is accompanied by a reference design that replaces the CAM core previously delivered through the CORE Generator™ software. The CAM reference design should be used for all new FPGA designs targeting Virtex®-6, Virtex-5, Virtex-4, Spartan®-6, Spartan-3, Spartan-3E, Spartan-3A, Spartan-3A DSP FPGAs, and newer architectures. All the features and interfaces included in the reference design are backward compatible with the LogiCORE™ IP CAM v6.1 core. In addition, because the reference design is provided in plain-text VHDL format, the implementation of the function is fully visible, allowing for easy debug and modification of the code.

## Introduction

A CAM performs content matching rather than the address matching performed by standard memory cores. The content matching approach enables faster data searches than can be achieved by sequentially checking each address location in a standard memory for a particular value. The higher speed searches are achieved by using content values as an index into a database of address values. The additional ability to perform content compares in parallel enables even higher speed searches. A set of scripts is included with the CAM reference design that allow the customization of width, depth, memory type, and optional features.

## Features

The CAM reference design has these features:

- Memory types: The CAM can be configured using one of two memory implementations:
  - SRL16E-based CAM with a 16 clock cycle write operation and a one clock cycle search operation.
  - Block RAM-based CAM with only a two clock cycle write operation and a one clock cycle search operation. The block RAM-based CAM also supports an optional additional output register that adds a one clock cycle latency to all read operations.
- Ternary modes: The CAM supports two ternary modes for both write and search operations in the SRL16E implementation:
  - Standard ternary mode: Bit X matches either 1, 0, or X (1010 = 1X1X = 10XX) and is referred to as a don't care bit.
  - Enhanced ternary mode: Bit X also matches either 1, 0, or X (1010 = 1X1X = 10XX), also referred to as a don't care bit. Bit U does not match any of the possible bit values: 1, 0, X, or U, and is referred to as an unmatchable bit in this document.
- Encoded/unencoded address: The match address can be in binary encoded, single-match unencoded (one-hot), or multi-match unencoded (many-hot) form.
- Multiple match resolution: Whenever the data being read matches data from more than one location in the CAM, a multiple match condition exists. The CAM supports this situation.
  - Single/multiple match flags: These two optional outputs can inform the user whether a single or multiple match situation exists.

- Multiple match address resolution: Depending on the parameter set by the user, the CAM can return either the highest or lowest matching address when a multiple match condition exists. This is only available when the address is converted to binary encoding or one-hot encoding.
- Initialization: The CAM supports initialization of binary and standard ternary CAMs with data from a MIF file, which is an ASCII file that contains the initial contents of the CAM. For CAMs using initialization, data entries in the MIF file must be in binary form. Standard ternary CAMs can be initialized with 0s, 1s, and Xs. Enhanced ternary CAMs cannot be initialized.
- Simultaneous read/write: The CAM supports optional simultaneous write and search operations, with an output to warn the user of possible collisions.
  - Read warning flag: This flag indicates that the data applied to the CAM for a read operation matches the data that is currently being written into the CAM by the unfinished write operation. This flag works in both single- and multiple-match scenarios.

## Interface

The CAM input and output ports are shown in [Figure 1](#).

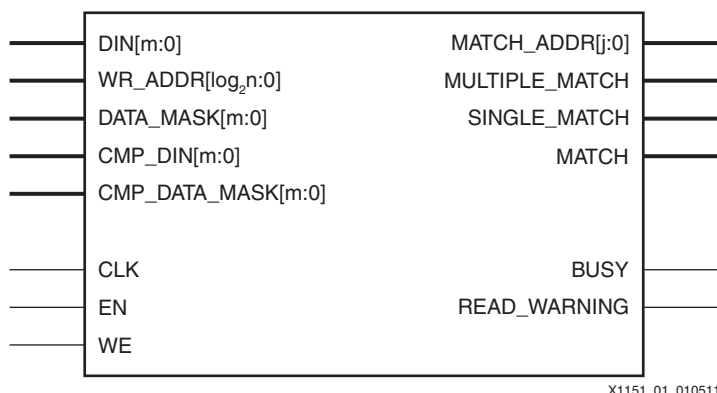


Figure 1: CAM Schematic

The CAM core signals are listed in [Table 1](#).

Table 1: CAM Core Signals

Port Name	Direction	Description
CLK	Input	<b>Clock:</b> All CAM operations are synchronous to the rising-edge of the clock input.
EN (Optional)	Input	<b>Enable:</b> This is the control signal used to enable both write and read operations.
DIN[m <sup>(1)</sup> :0]	Input	<b>Data In:</b> This is the data to be written to the CAM during a write operation. It is also the data to look up from the CAM during a read operation when simultaneous read/write option is not selected.
DATA_MASK[m:0] (Optional)	Input	<b>Data Mask:</b> This signal interacts with the DIN bus to create new bit values in ternary mode.
CMP_DIN[m:0] (Optional)	Input	<b>Compare Data In:</b> This is the data to look up from the CAM during a read operation if the simultaneous read/write option is selected.

Table 1: CAM Core Signals (Cont'd)

Port Name	Direction	Description
CMP_DATA_MASK[m:0] (Optional)	Input	<b>Compare Data Mask:</b> This bus interacts with the CMP_DIN bus to create new bit values in ternary mode if the simultaneous read/write option is selected.
WE (Optional)	Input	<b>Write Enable:</b> This is the control signal used to enable transfer of data into the CAM from the DIN bus.
WR_ADDR[log <sub>2</sub> n <sup>(2)</sup> :0]	Input	<b>Write Address:</b> This is the location to which the data on DIN is written into the CAM.
BUSY	Output	<b>Busy:</b> This signal Indicates that a write operation is currently being executed.
MATCH_ADDR[j:0]	Output	<b>Match Address:</b> This is the CAM address where matching data resides.
MATCH	Output	<b>Match:</b> This signal indicates that at least one location in the CAM contains the same data as the DIN bus (or CMP_DIN if in simultaneous read/write mode).
MULTIPLE_MATCH (Optional)	Output	<b>Multiple Match:</b> This signal indicates the existence of matching data in more than one location of the CAM.
SINGLE_MATCH (Optional)	Output	<b>Single Match:</b> This signal indicates the existence of matching data in only one location of the CAM.
READ_WARNING (Optional)	Output	<b>Read Warning:</b> This signal indicates that the data applied to the CAM for a read operation matches the data that is currently being written into the CAM by the unfinished write operation.

**Notes:**

1. m = CAM width.
2. n = CAM depth.

The CAM core signals are discussed in more detail here:

- **CLK (Clock):** The CAM module is fully synchronous with the rising edge of the clock input. All input pins have the setup time referenced to the CLK signal. All output ports have clock-to-out times referenced to the CLK signal.
- **EN (Enable):** When active, the optional enable signal allows the CAM to execute write and read operations. If the enable is inactive during normal operation of the core, the output pins hold their previous state and all internal states freeze. Any new input signal is ignored until the enable is driven active, at which time the CAM resumes all of its halted operations.
- **DIN[m:0] (Data In Bus):** The DIN bus provides the data to be written into or read from the CAM core, depending on the operation. If the simultaneous read/write option is selected, this bus is used only for the write operation, and the CMP\_DIN bus is used exclusively for the read operation.

In ternary modes, this bus becomes one of the two input buses used to determine the bit value. In standard ternary mode, a 0 on both DIN and DATA\_MASK designates a 0. A 1 on DIN and a 0 on DATA\_MASK designates a 1. A 0 or a 1 on DIN and a 1 on DATA\_MASK designates an x.

In enhanced ternary mode, a 0 on both DIN and DATA\_MASK designates an X. A 1 on both buses designates a U. A 1 on DIN and a 0 on DATA\_MASK designates a 1. A 0 on DIN and a 1 on DATA\_MASK designates a 0.

- **DATA\_MASK[m:0] (Data In Mask Bus):** This optional input bus is available when one of the ternary modes is selected. In standard ternary mode, this signal masks the DIN bus to create the don't care bits. Bits that are 1 on DATA\_MASK indicate the locations of the don't care bits on the DIN bus.

In ternary modes, this signal becomes one of the two input buses used to determine the bit value. For further information, see the description of the DIN bus defined earlier in this section. In this application note, the DATA\_MASK bus is treated as part of the DIN bus when a ternary CAM is selected.

- **CMP\_DIN[m:0] (Compare Data In Bus):** When the simultaneous read/write option is selected, this optional input bus provides the data for the read operation of the CAM. When the simultaneous read/write option is not selected, this bus is not available.

In ternary modes, this bus becomes one of the two input buses used to determine the bit value during a read operation. For further information, see the description of DIN bus defined earlier in this section.

- **CMP\_DATA\_MASK[m:0] (Compare Data Mask):** This optional input bus is available when the CAM core is configured to support both simultaneous read and write operations and ternary mode. In standard ternary mode, this signal masks the CMP\_DIN bus to create don't care bits. Bits that are 1 on this bus indicate the locations of don't care bits on the CMP\_DIN bus.

In ternary modes, this bus becomes one of the two input buses used to determine the bit value. For more information, see the descriptions of the DIN and DATA\_MASK buses defined earlier in this section. In this application note, the CMP\_DATA\_MASK bus is treated as part of the CMP\_DIN bus when a ternary CAM is selected.

- **WE (Write Enable):** The optional write enable signal allows data on the DIN bus to be written into the CAM. When this signal is asserted, the contents on the DIN bus are written into the location selected by the write address bus WR\_ADDR. This signal is not present if the read-only CAM option is selected. This signal is optional when the CAM initialization option is selected.
- **WR\_ADDR[log<sub>2</sub>n:0] (Write Address Bus):** The optional write address bus determines the memory location to be written to during the CAM's write operation. This bus is not present if the read-only CAM option is selected. This bus is optional when the CAM initialization option is selected.
- **BUSY (Busy):** The busy signal indicates that the write operation is currently being executed. It remains asserted until the multiple clock cycle write operation is completed. A new write operation cannot be started while this signal is active.
- **MATCH\_ADDR[j:0] (Match Address Bus):** This output bus indicates the address that matches the contents of the DIN bus, or the CMP\_DIN bus if the simultaneous read/write option is selected. The match address can be encoded (binary), single-match unencoded (one-hot), or multiple-match unencoded. The width j depends on the encoding type selected.
- **MATCH (Match):** The match signal is asserted for one clock cycle when data on the DIN bus matches data in one or more locations in the CAM. If the simultaneous read/write option is selected, data on the CMP\_DIN bus is used to search for a match instead of the DIN bus.
- **READ\_WARNING (Read Warning):** The optional read warning signal is asserted when data for the write in progress of the CAM is the same as data for the read initiated for the CAM. Because write operations take multiple cycles, writes performed prior to reads might not have been completed when the read is executed. READ\_WARNING is asserted to let the user know that the match address and match signals do not reflect the results of the most recent write operation being executed.

- **MULTIPLE\_MATCH (Multiple Match):** The optional multiple match signal is asserted for one clock cycle when more than one match is present in the CAM. It remains inactive if there is one or no matches.
- **SINGLE\_MATCH (Single Match):** The optional single match signal is asserted for one clock cycle when there is only one match in the CAM. This signal remains inactive if there is more than one match, or if there are no matches.

## Functional Description

### Operating Modes

The CAM has two operating modes: read operation and write operation.

#### Read Operation

The read operation of the CAM is synchronous to the rising edge of the clock. In a read operation, the CAM's contents are searched for the data present on the DIN bus or the CMP\_DIN bus (if the simultaneous read/write option is selected) at the rising edge of the clock. The enable (EN) signal must be asserted for the entire duration of the read operation.

If a read operation is applied to the CAM while the busy signal is asserted (which means a write operation is still being executed), the CAM location currently being written into appears to be empty. This location does not match any data that the user places on the DIN or CMP\_DIN bus for the read operation. If the user applies both read and write operations on the same rising edge of the clock, the write operation starts its execution before the read operation, meaning that write addresses in the CAM are cleared before the CAM is searched for a read match.

The match address bus behaves differently, depending on the selections made in the `CustomizeWrapper.pl` script. If there is one match in the core, multi-match unencoded and single-match unencoded behave identically by setting the bit corresponding to the location of that match in the MATCH\_ADDR bus active. Similarly, the binary encoded MATCH\_ADDR contains the encoded version of the active bits. When there are multiple matches in the core, single-match unencoded and binary encoded returns the match of the highest priority location, which can be selected as the lowest or highest address. Multi-match unencoded has every bit corresponding to the location of the matches in the MATCH\_ADDR bus active.

#### Write Operation

The write operation for the CAM is synchronous to the rising edge of the clock. The data on the DIN port is written into the memory location selected by the WR\_ADDR port when both WE and EN signals are active. The WE signal is required to be asserted for the initial clock cycle of the write operation applied to the CAM by the user.

On the first clock cycle of a write operation, the old data at the WR\_ADDR location is removed from the memory, and on the last clock cycle of a write operation, new data is written to the same location in the memory. During the first and middle clock cycles of a write operation, the memory location being written to behaves as an empty memory location.

During a write operation, the enable signal must remain active for the entire write cycle. If enable is deactivated at any time during the write operation, the write cycle stops and remains at that stage until enable is activated again.

### Block RAM-Based Implementation

A CAM implemented with block SelectRAM™ memory primitives has a single clock cycle latency on its read operation, and two clock cycle latency on its write operation.

#### Read Operation

Figure 2 shows consecutive read operations of a block SelectRAM memory CAM with the second operation not having a match. Three of the possible configurations for the MATCH\_ADDR and MATCH signals are displayed. On the second rising edge of CLK, a read of

data 01 is performed. On the third rising edge of CLK, a MATCH is reported at MATCH\_ADDR 01 (unencoded), or MATCH\_ADDR 00 (encoded), for the input data 01. At the same time, another read is performed for the data 11. On the fourth rising edge of CLK, no MATCH is reported for the input data of 11. The remaining CLK edges show multiple reads of the input data 10, with the CAM core reporting a MATCH for this data at MATCH\_ADDR 10 (unencoded), or MATCH\_ADDR 01 (encoded).

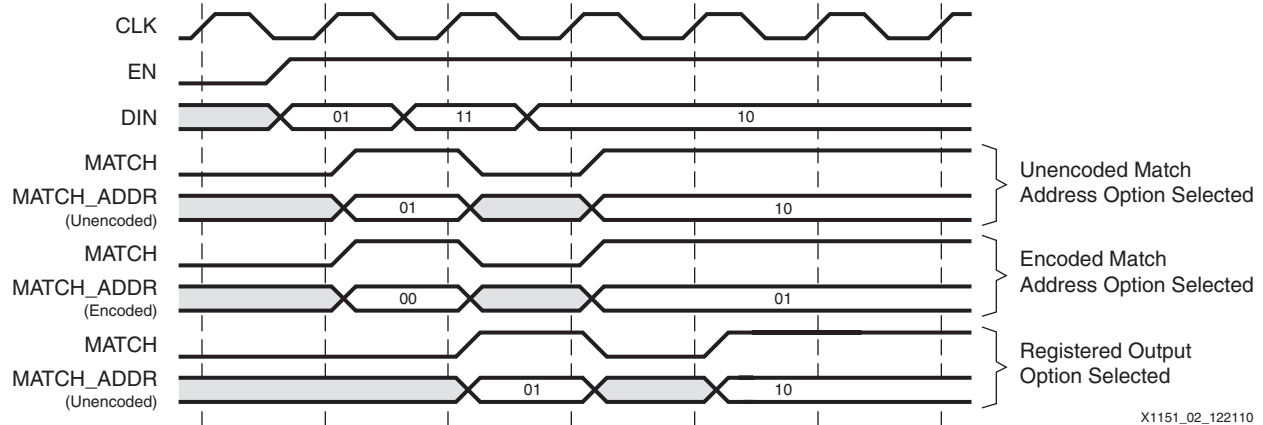


Figure 2: Block SelectRAM Memory Read Operation

By default, the block SelectRAM memory CAM has a single-clock read latency. However, an extra clock cycle can be added to the read latency by selecting the Register Outputs option in the `CustomizeWrapper.pl` script. New data written into the CAM is available to be read on the second rising edge of the clock after a write operation begins.

### Write Operation

Figure 3 shows three consecutive write operations of a block SelectRAM memory CAM with the simultaneous read/write option enabled. The figure also shows when the new data is available to be read by the read operation. The block SelectRAM Memory CAM has a two clock cycle write latency. When executing consecutive write operations, each write operation must be two clock cycles apart.

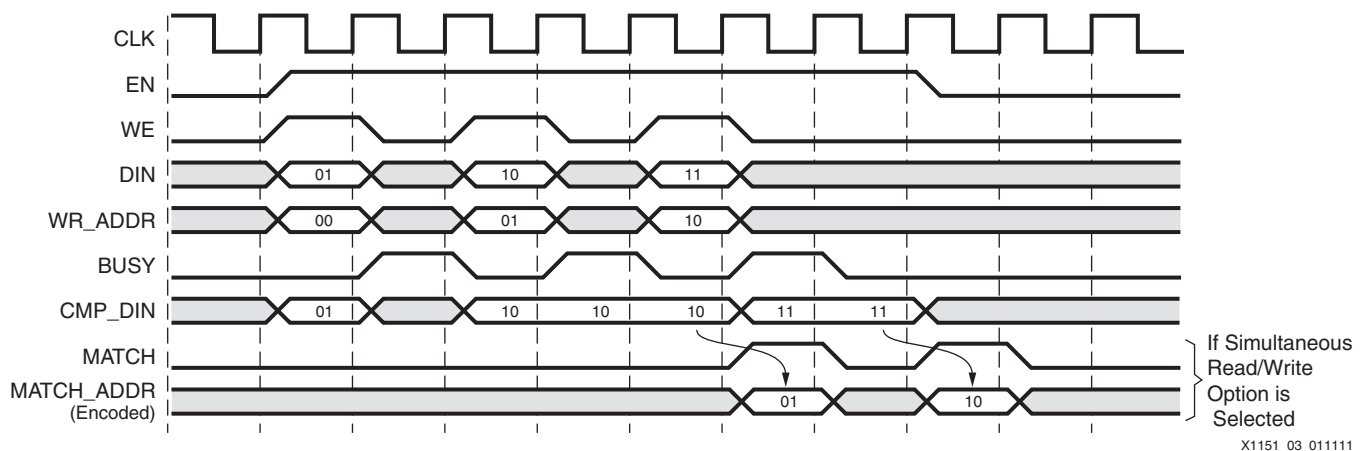


Figure 3: Block SelectRAM Memory Write Operation

The following describes the events shown in Figure 3:

1. A write of data 01 to WR\_ADDR of 00 is performed on the second rising edge of CLK. On the same CLK edge, a read of the same data 01 is attempted. Because the write operation takes precedence, no MATCH is reported for the read of data 01 on the next CLK cycle.

2. On the third rising edge of CLK, the BUSY signal is asserted, indicating a write is in progress (from the previous clock cycle) and no write operation can occur.
3. The fourth rising edge of CLK shows a write of data 10 to WR\_ADDR 01, with a read attempted on CMP\_DIN of the same data at the same time. Again, no MATCH is reported for the read of data 10 on the next cycle.
4. The fifth rising edge of CLK shows another read attempt of data 10, with no MATCH reported the next clock cycle because the write operation from the previous cycle is still in progress.
5. On the sixth rising edge of CLK, a write of data 11 is performed to WR\_ADDR 10. At the same time, a third read attempt of data 10 is performed. This time, a MATCH for the read data is reported on the next clock edge because the simultaneous write was not the same data that was read.
6. The seventh rising edge of CLK shows a read attempt of data 11. Because the write operation for the same data is still in progress (BUSY is asserted), no MATCH is reported.
7. On the eighth rising edge of CLK, a read is again attempted of data 11. This time, a MATCH is reported on the next clock edge at MATCH\_ADDR 10.

## SRL16E-Based Implementation

A CAM implemented with SRL16E primitives has a single clock cycle latency on its read operation and 16 clock cycle latency on its write operation.

### Read Operation

Figure 4 illustrates three consecutive read operations of an SRL16E-based CAM with the second operation not having a match. Two of the possible configurations for the MATCH\_ADDR and MATCH signals are displayed.

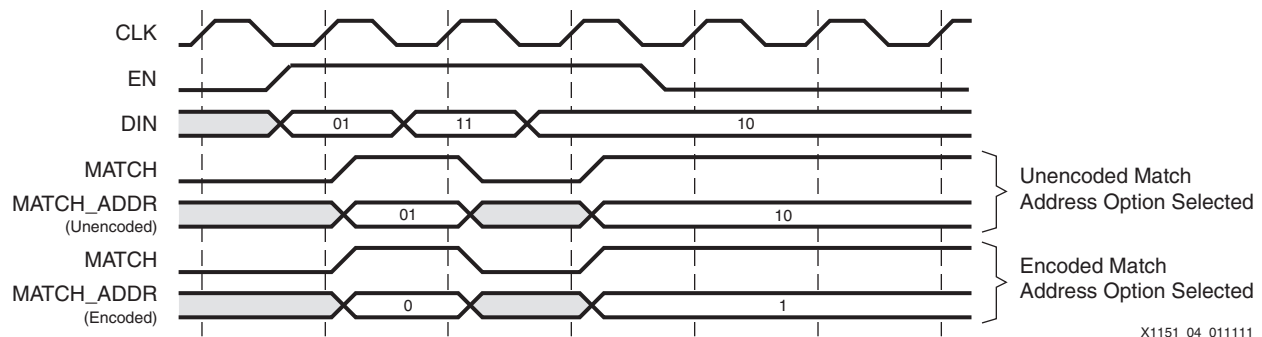


Figure 4: SRL16E Read Operation

The SRL16E-based CAM asserts the MATCH signal on the first rising clock edge after data is placed on the DIN bus by the user if there is at least one location in the CAM with matching data. New data written into the CAM is available to be read on the 17th rising edge of the clock after write operation begins.

### Write Operation

Figure 5 shows two consecutive write operations of an SRL16E-based CAM. The figure also shows when the new data is available to be read by the read operation. The SRL16E-based CAM has a 16 clock cycle write latency. When executing consecutive write operations, each write operation must be 16 clock cycles apart.

On CLK edge 1, the data 01 is written to address 00. Beginning on CLK edge 2, a read attempt of the same data 01 is begun. No MATCH for this data is reported until CLK edge 17 because it takes 16 clock cycles for the write to complete. The MATCH for the read data is reported on CLK edge 17 with an unencoded MATCH\_ADDR of 0001. The original WR\_ADDR was 00



(binary). Because this is the first address index of the CAM, the unencoded (one-hot) MATCH\_ADDR for that location is 0001.

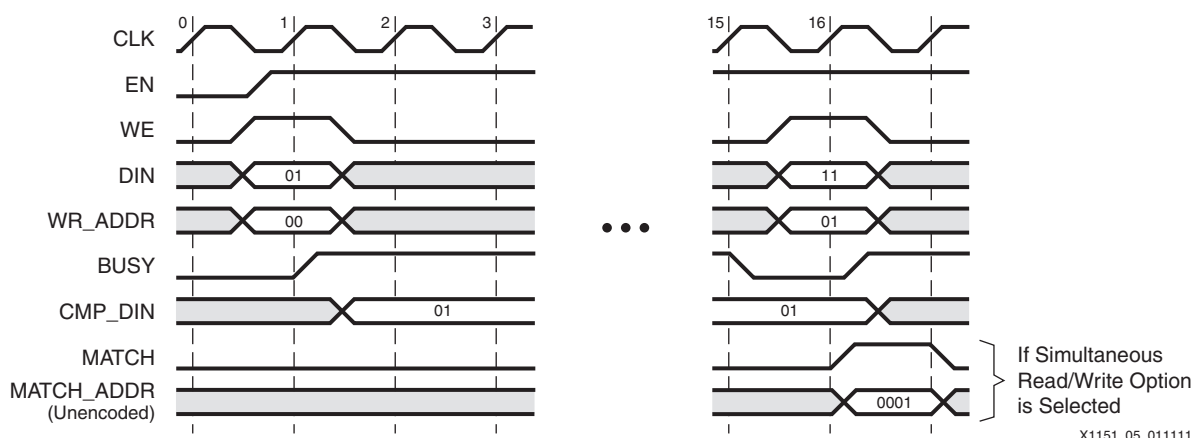


Figure 5: SRL16E Write Operation

When writing and reading the CAM, two ternary mode options are supported using ternary bits 0, 1, x, and optionally U:

- **Standard ternary mode:** In this mode, bit x matches either 1, 0, or x (1010 = 1x1x = 10xx) and is referred to as a don't care bit. This example shows how to write and search for ternary values (xs):
  - To write 1x1x, DIN = 1010, 1110, 1011, or 1111 and DATA\_MASK = 0101.
  - To search for 1x1x, DIN and DATA\_MASK use the same values as above.

**Note:** If the CAM already has a ternary value written to it, that ternary bit (x) matches 0, 1, and x in the input buses.

- **Enhanced Ternary Mode:** In this mode, bit x also matches either 1, 0, or x (1010 = 1x1x = 10xx) and is also referred to as a don't care bit. Bit U does not match any of the four possible bit values 1, 0, x, or U, and is referred to as an unmatchable bit.

## Hardware Implementation

### General Overview

The CAM design uses the same FPGA memory blocks as a traditional memory, but there are key differences between the two. For a traditional memory, the user provides the input data and address for a write operation. For a read operation, an address is provided, and the data stored at that address is read out. For a CAM, the write operation is the same as traditional memory. However, the read operation differs in that the user provides a data input to look up the address where that data is stored.

Just as a traditional memory only stores one data word at each address, each CAM address can only store one unique data. However, a particular data can be stored at multiple addresses.

To accomplish this functionality, a number of functional blocks are required in addition to the standard memory (SRL16E or block RAM) blocks. Before reading or writing to the memory, the CAM must process the input data and address to map to the appropriate memory block, and perform certain operations like ternary encoding, if required. At the memory output, the CAM must interpret which address(es) contain the data, generate the MATCH flags, and register the outputs. All of these functions are managed by a control block. The basic data flow through each block in the CAM is shown in Figure 6.



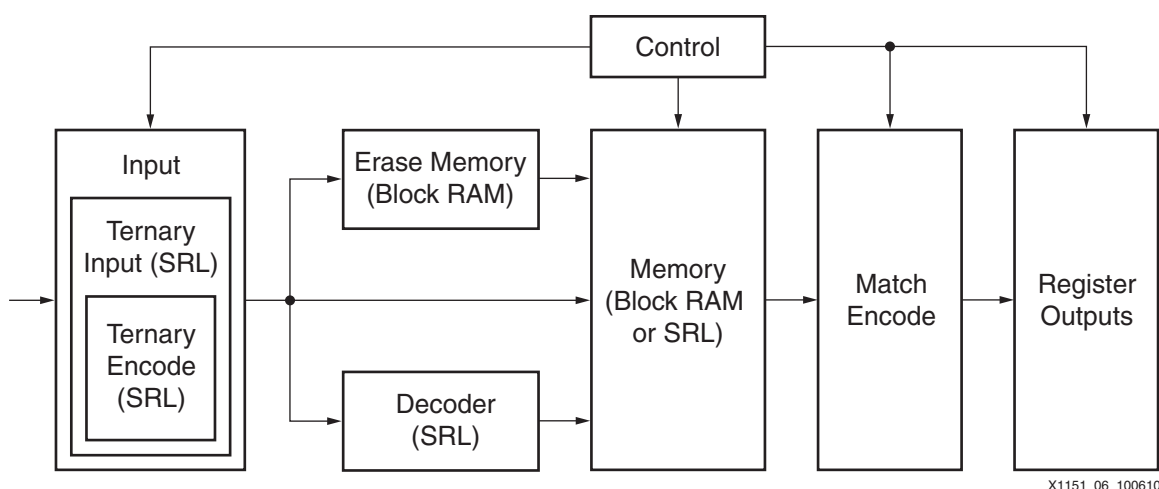


Figure 6: CAM Data Flow

## RAM-Based Implementation

### Introduction

When using RAM for the CAM implementation, the block memory behaves like a large grid where each element in the grid represents a particular mapping of a CAM address to a particular data value. In other words, every possible data/address combination is represented by one RAM bit. A simple example of an 8 x 3 ( $n = 8$  words deep,  $m = 3$  bits wide) CAM is shown in Table 2.

Table 2: RAM Match Grid

	RAM Data/CAM Address								
		0	1	2	3	4	5	6	7
RAM Address/ CAM Data	000	0	1	0	0	0	0	0	0
	001	0	0	0	0	0	1	0	0
	010	0	0	0	0	0	0	0	0
	011	0	0	0	1	0	0	0	0
	100	1	0	0	0	0	0	0	1
	101	0	0	0	0	0	0	0	0
	110	0	0	1	0	0	0	0	0
	111	0	0	0	0	0	0	0	0

A CAM whose data input is  $m$  bits wide requires a memory that is  $2^m$  deep to accommodate all  $2^m$  possible values of that data input. An 8 x 3 ( $n \times m$ ) CAM therefore requires a RAM that is  $2^3 = 8$  words deep. This same CAM requires a RAM of width  $n = 8$  to accommodate 8 possible address values. To summarize, a CAM that is  $m$  data input bits wide requires a memory that is  $2^m$  deep, and a CAM that has  $n$  addresses requires a memory that is  $n$  bits wide.

Each row in the RAM shown in Table 2 represents one possible mapping of the input data bits to the CAM contents. A CAM with a 3-bit data input has 8 possible unique data bit combinations and thus requires 8 rows (or “words”) in a RAM (in other words, the required CAM depth is 8). Similarly, a CAM with a depth of 8 addresses requires 8 columns (or data bits) in the RAM. If the CAM had a depth of 16 addresses, 16 columns would be required.

The value or bit at each grid location is set to 1 if the data is stored at that address; otherwise, it is set to 0. The example in [Table 2](#) shows a CAM that already has data stored within it, either by writing to the CAM or by using a MIF file to initialize the contents during implementation (see [Creating a MIF File \(Specifying CAM Contents\)](#), [page 27](#)). These are the contents of the CAM described in [Table 2](#):

- CAM data 000 is stored at address 1.
- CAM data 001 is stored at address 5.
- CAM data 011 is stored at address 3.
- CAM data 100 is stored at address 0 and 7.
- CAM data 110 is stored at address 2.
- CAM data 010, 111, and 101 is *not* stored at any addresses.
- CAM addresses 4 and 6 do not store any data.

As discussed in [General Overview](#), [page 8](#), a single data can be stored at multiple addresses, but a single address can only store a single data value.

### **Write Operation Overview**

The design for the two clock cycle write operation is from *Using Block RAM for High Performance Read/Write CAMs* [Ref 1]. The first clock cycle is the erase operation, which removes the data previously stored at the write address by clearing the bit at that location in the grid. The second cycle performs the write, setting to 1 the bit at the intersection of the address and data being written.

The initial erase cycle before the write is necessary to avoid the scenario of multiple data at the same address. Using [Table 2](#) as an example, consider that address 7 previously had data 100 and the new data 010 needs to be written to this address. The intersection of 100 and the address currently has a 1. If this is not erased, after writing data 010, the intersection of data 010 and this address will also have a 1, indicating the impossible scenario of both data 100 and 010 at this address.

### **Read Operation Overview**

The read operation reads the CAM address bits from the memory stored under the input read data. The output represents one bit for each possible address. For every location that contains or matches the data that is presented, this bit is set to 1. If a match is found, the MATCH flag is asserted and the matching address is presented on the MATCH\_ADDR port.

It is possible for more than one 1 to exist in the bits read from the memory. If this occurs, the data that is being searched is stored at more than one address. If the MULTIPLE\_MATCH flag option is enabled, the flag is asserted.

### **Initialization**

A text file with the MIF extension is used to initialize the block RAM primitives. The contents of this file are in binary form and are parsed into CAM data-width words in multiple stages in the RTL. Initialization is explained in detail within the VHDL code itself.

## **Basic CAM Configurations Using Block RAM Primitives (Virtex-6 FPGAs)**

Many different CAM configurations are possible using a dual-port block RAM, depending on the size of the block RAM primitive available in the FPGA. In a Virtex-6 device, the RAMB36 has 32 Kbits of memory available for a CAM. Using this available space, a 32 x 10 CAM is the most efficient CAM primitive that can be built from a single RAMB36 block RAM. The sizing and connections for each port of the block RAM for this size CAM are shown in [Figure 7](#). Port A of the block RAM is used for writes (if enabled), and port B is used for reads.

**Note:** The Virtex-6/Virtex-6L FPGA case using RAMB36 primitives is provided as an example. Virtex-5 devices also contain RAMB36 primitives, so larger CAMs are built in the same way as in

Virtex-6 devices, with multiple blocks of 32 x 10 CAMs. Spartan-6, Spartan-3, and Virtex-4 devices contain RAMB18 or RAMB16 primitives. Larger CAMs for these devices use multiple blocks of 32 x 9 CAMs.

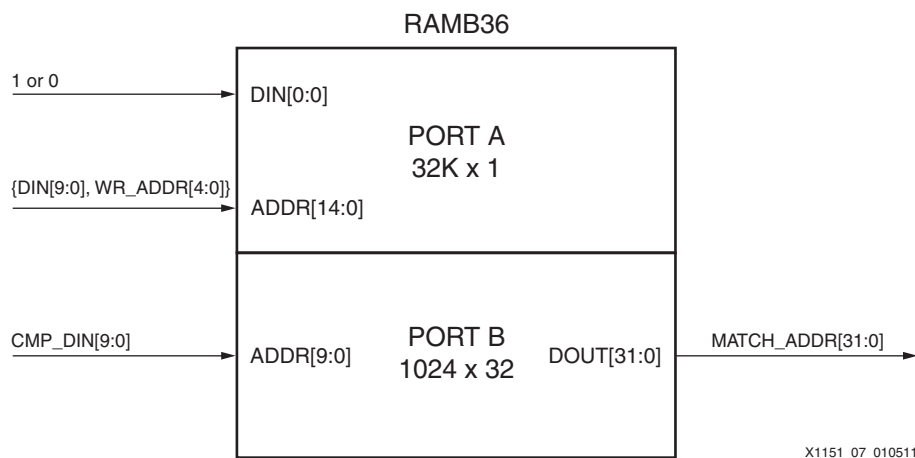


Figure 7: RAMB36 Connection for 32 x 10 CAM

### Larger CAM Depths and Widths

To achieve deeper and wider CAMs, multiple 32 x 10 CAM blocks are concatenated in width and depth, respectively.

#### Deeper CAMs

To achieve a CAM depth greater than 32 words, multiple basic CAM blocks are combined in parallel. Figure 8 shows the configuration of the basic CAM for this purpose. The CAM size shown in Figure 8 is 128 bits deep by 10 bits wide.

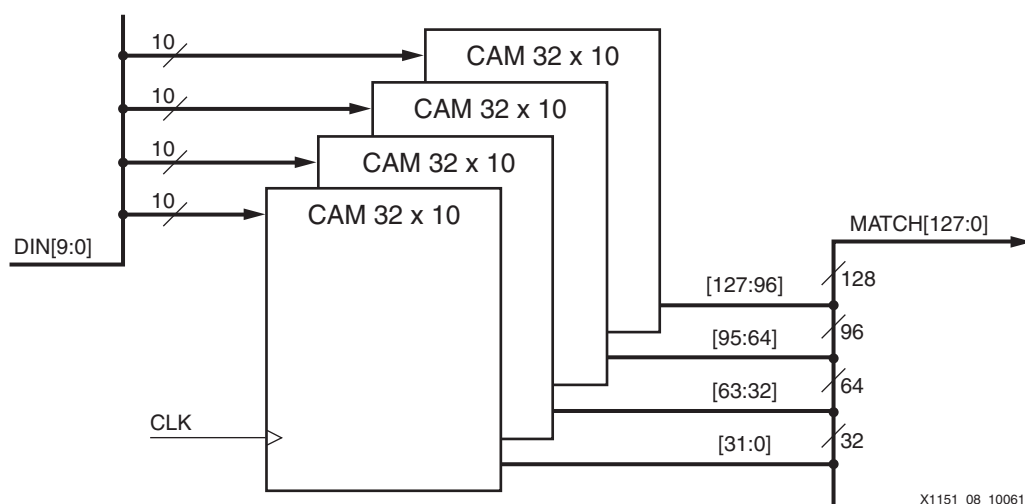


Figure 8: CAM Depth Expansion (128 x 10 CAM)

#### Wider CAMs

To achieve a CAM width greater than 10 bits wide for Virtex-6 FPGAs, multiple basic CAM block outputs are combined using AND gates. Figure 9 shows the configuration of the basic CAM for this purpose. The AND gates on each MATCH bit output are required because a match here is defined as both the lower data bits AND the upper data bits matching the read input on the same address. The CAM size shown in Figure 9 is 32 bits deep by 20 bits wide.

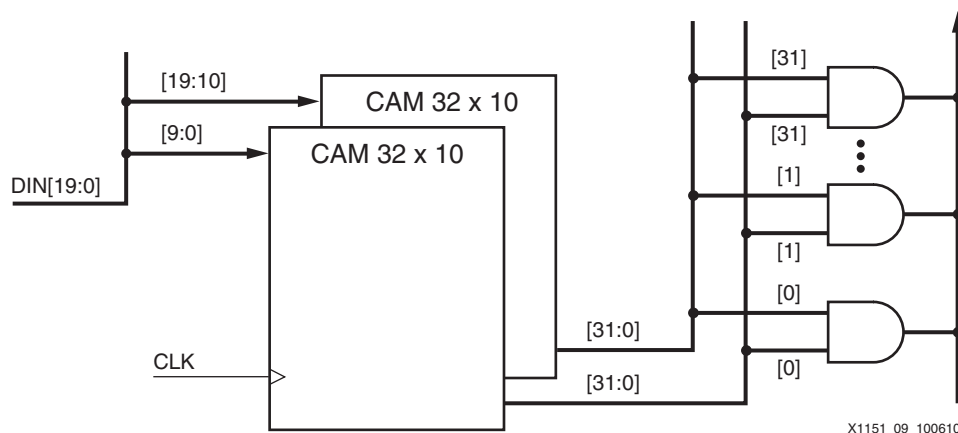
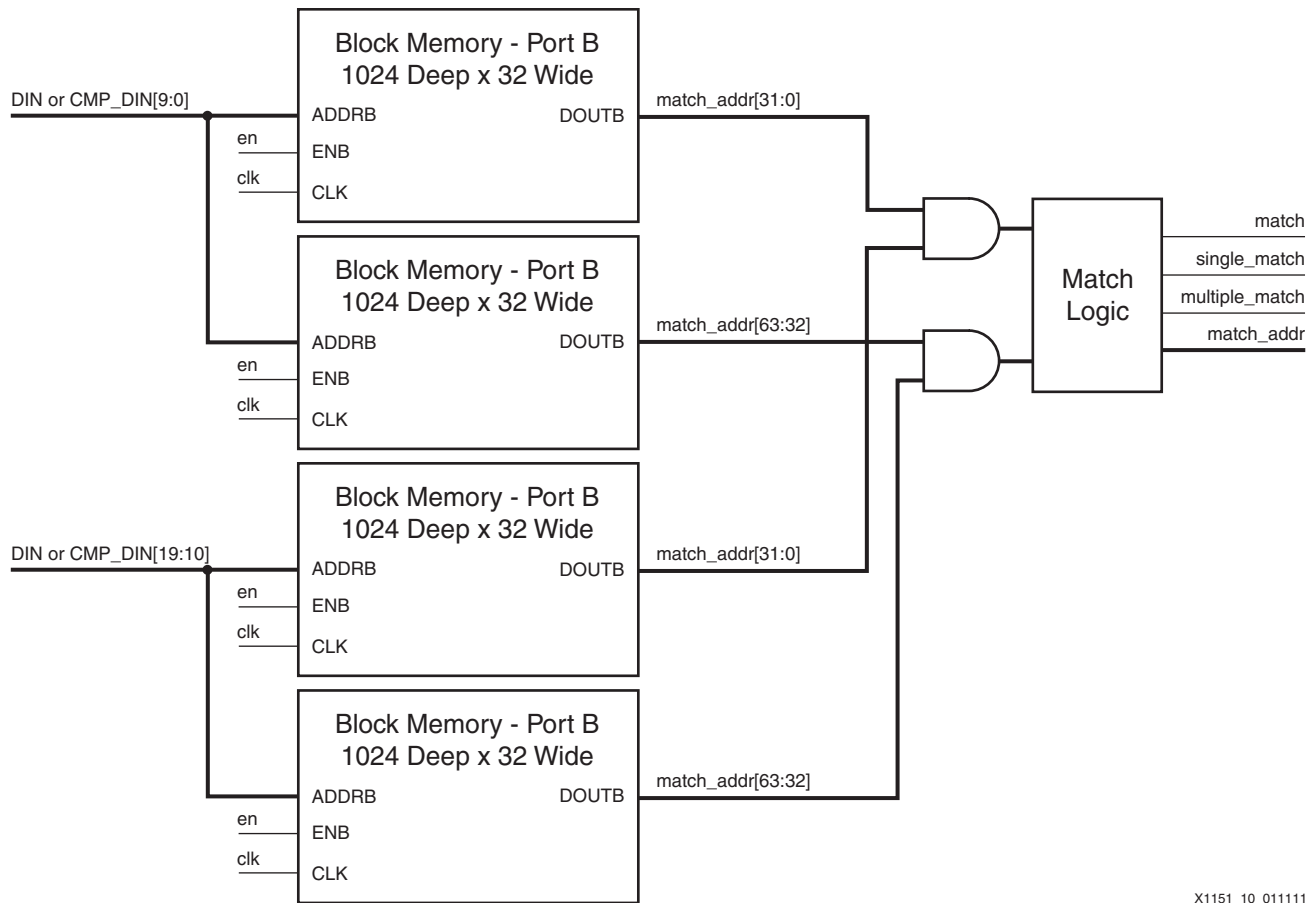


Figure 9: CAM Width Expansion (32 x 20 CAM)

### Read Implementation

For a simple CAM using a single block RAM primitive, the read data is presented on the address input of port B (ADDRB) of the block RAM. For a more complex CAM using multiple block RAM primitives, all port Bs of the block memory are addressed by the CAM read data (DIN or CMP\_DIN[9:0] in Figure 10). Figure 10 shows the logic connection for a CAM using more than one block RAM primitive (the CAM size shown in Figure 10 is 64 bits deep by 20 bits wide). The output of the block RAM is a series of 1s and 0s that indicate the address(es) which that data matches. The match logic uses this information to calculate how many CAM addresses matched the data, and the lowest/highest address that matched.

**Note:** If the simultaneous read/write option (parameter `c_has_cmp_din`) is selected, CMP\_DIN provides the read address (ADDRB) to the read port and the DIN bus is used for write data; otherwise, the DIN input bus is connected to ADDR B.



X1151\_10\_011111

Figure 10: CAM Width and Depth Expansion (64 x 20 CAM)

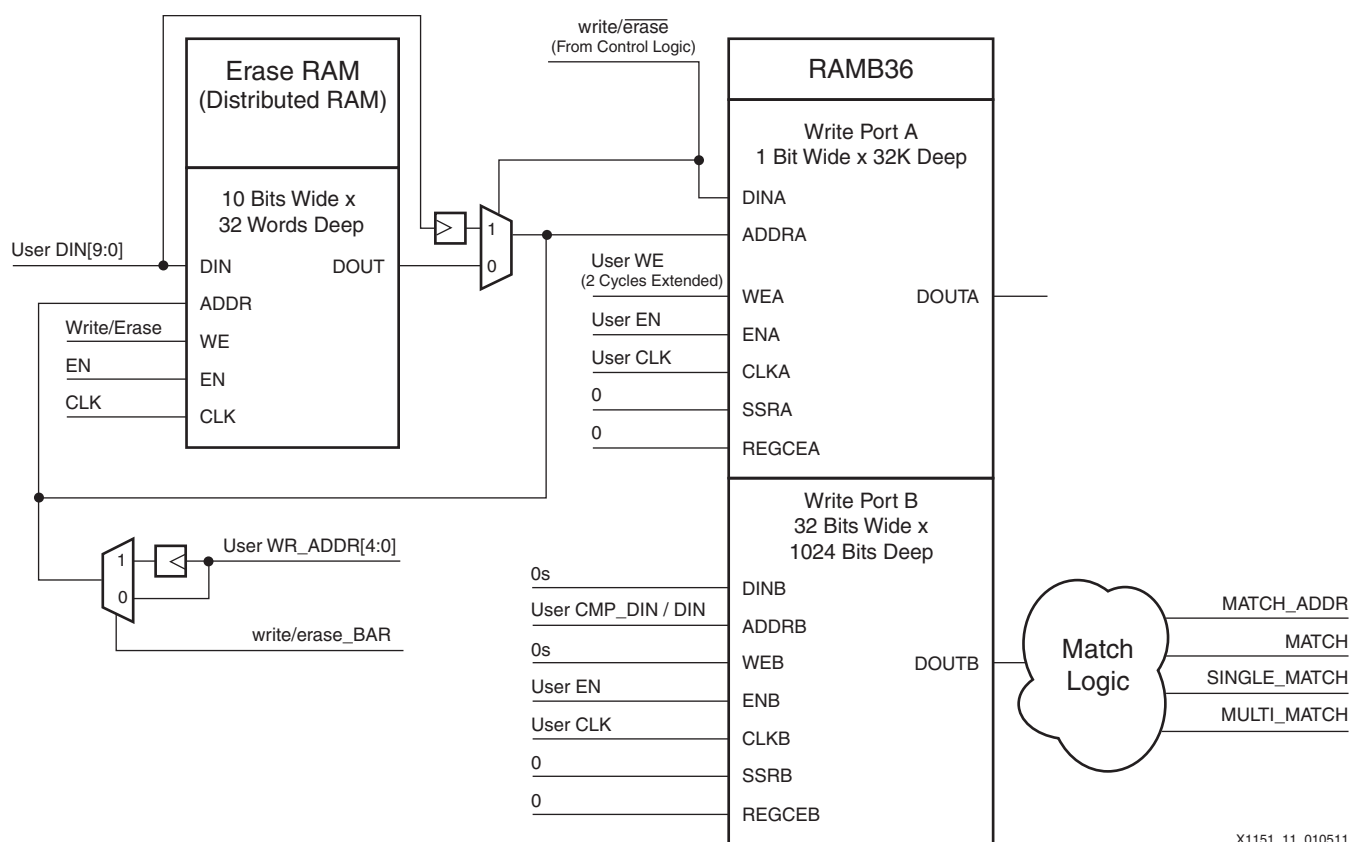
### Erase/Write Implementation

As described in [Write Operation Overview](#), page 10, the write operation is performed in two clock cycles: the erase cycle and the write cycle. [Figure 11](#) shows the write logic to implement this. On the first clock cycle, the previous contents of the memory location are erased. On the second clock cycle, the new contents are written to the memory location.

As shown in [Figure 11](#), the DIN data input of the CAM passes through a distributed RAM block called the Erase RAM. The data input also bypasses the Erase RAM and is used as an input to a multiplexer. The multiplexer selects between the DIN input and the DOUT output of the Erase RAM.

For a write operation, the data currently stored at the write address must be erased. During the first clock cycle, the value from the RAM ERASE memory is read combinatorially from the specified address. The data accessed from this RAM is the last data stored in the CAM at that address. The write/erase signal is then cleared to 0, and the output of the erase RAM is combined with the address and used to index a single bit in the block RAM. That location is then cleared to 0. This effectively removes the previous data stored at that address.

On the second clock cycle, the new data and address input is passed through directly to the block memory. At the new data/address location specified, a 1 is written by setting a 1 on the write/erase input shown in [Figure 11](#). On the same clock cycle, the data is also stored in the RAM Erase memory in preparation for the next erase cycle. The basic implementation of this logic for a 32 x 10 CAM is shown in [Figure 11](#). The Erase RAM size is always the same width and depth as the CAM core.



X1151\_11\_010511

Figure 11: CAM Core with Erase RAM Connection

## SRL16E-Based Implementation

### Introduction

Using an SRL16E as a CAM is in most ways the same as using a block RAM as a CAM. The SRL16E can be thought of as a 16-bit deep by 1-bit wide RAM, which translates into a 4-bit wide by 1-bit deep CAM. Because one SRL16E produces a CAM of 1 address deep, each SRL16E can only store one matchable data value. Table 3 shows an example in which the data value 0110 is stored in the single address space represented by this particular SRL16E block.

Table 3: SRL CAM Storage

CAM Data (SRL Address)	Match?
0000	0
0001	0
0010	0
0011	0
0100	0
0101	0
0110	1
0111	0
1000	0

Table 3: SRL CAM Storage (Cont'd)

CAM Data (SRL Address)	Match?
1001	0
1010	0
1011	0
1100	0
1101	0
1110	0
1111	0

**Write Operation**

As a shift register, an SRL16E is written by shifting data in one bit at a time. Because an SRL16E contains 16 bits of data, a write operation takes 16 cycles to complete.

In an SRL16E implementation, each bit of the SRL16E is used to encode 4 bits of input data with a depth of 1 word. Because each SRL16E represents one address space, only one bit in each of the SRL16E can be a 1 at any time. Thus, a write to the SRL16E must rewrite all 16 bits to ensure that only one match is present. If the input data matches the index of one of the words in the SRL16E, a 1 is written into the SRL16E; otherwise, a 0 is written. Because any previous match data stored in the SRL16E must be overwritten with the new data, all 16 bits of the SRL16E must be shifted out by writing 16 bits of new data. Therefore, a write operation always takes 16 clock cycles.

**Read Operation**

For a read, the read data is placed on the address lines of the SRL16E. If a 1 is stored at that address, the data presented is stored at the single address space represented by that SRL16E, and thus a match occurs. Because there is only one 1 bit stored in the SRL16E at a time, the match signal is active only when the location addressed by the input data contains a 1. The read from the SRL16E takes one clock cycle.

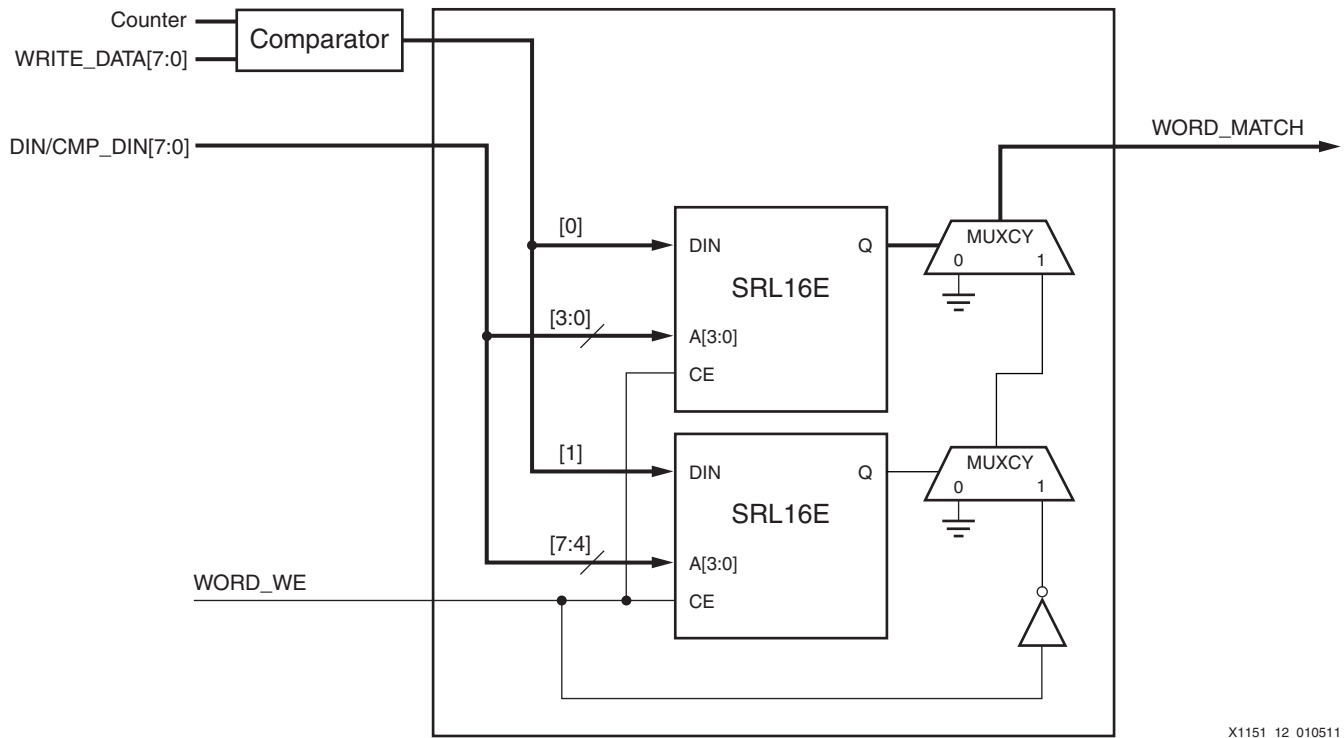
**Initialization**

A text file with a .mif extension is used to initialize the SRL16Es. The contents of the file are parsed into CAM data-width words. These words are then encoded, 4 bits at a time, into the 16-bit value to be stored in each SRL16E.

**Building Wider and Deeper CAMs using SRL16E primitives**

Implementing a CAM using SRL16s makes use of many components found within each FPGA slice. As shown in Figure 12, an 8 x 1 CAM is built using two SRL16E primitives and the MUXCYs located in the slice. Adding more SRL16E/MUXCY pairs allows for extension of the CAM width.

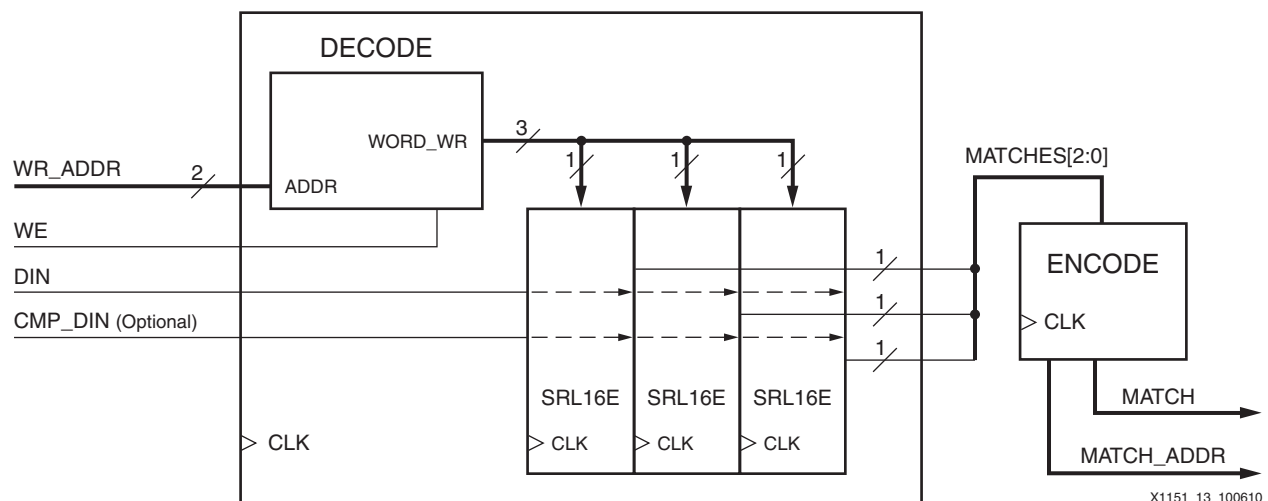




X1151\_12\_010511

Figure 12: 8-Bit Wide by 1-Bit Deep CAM in FPGA Slice

To extend the depth of a CAM built with SRL16Es, each group of 4 input data bits must connect to multiple SRL16Es in parallel. The `WR_ADDR` input is decoded to one-hot form to drive the individual write enables of each SRL16E. Each `Q` output of the SRL16Es creates a `MATCHES` bus that is passed to the Match Encoder block for processing. Figure 13 shows an example of this for a 4-bit wide by 3-bit deep CAM.



X1151\_13\_100610

Figure 13: 4-Bit Wide by 3-Bit Deep CAM Using SRL16E

### Details of Read Implementation

For a read, the input data is the address of the SRL16E as shown in Figure 14. If the content of the SRL16E at the address is 1, there is a match. The output of the SRL16E is used to select either a 1 or 0 using the select line of a MUXCY. This effectively uses the multiplexer as an AND

gate that is disabled during a write. The value out of the last MUXCY indicates whether or not there was a match of the input data on this 8-bit word.

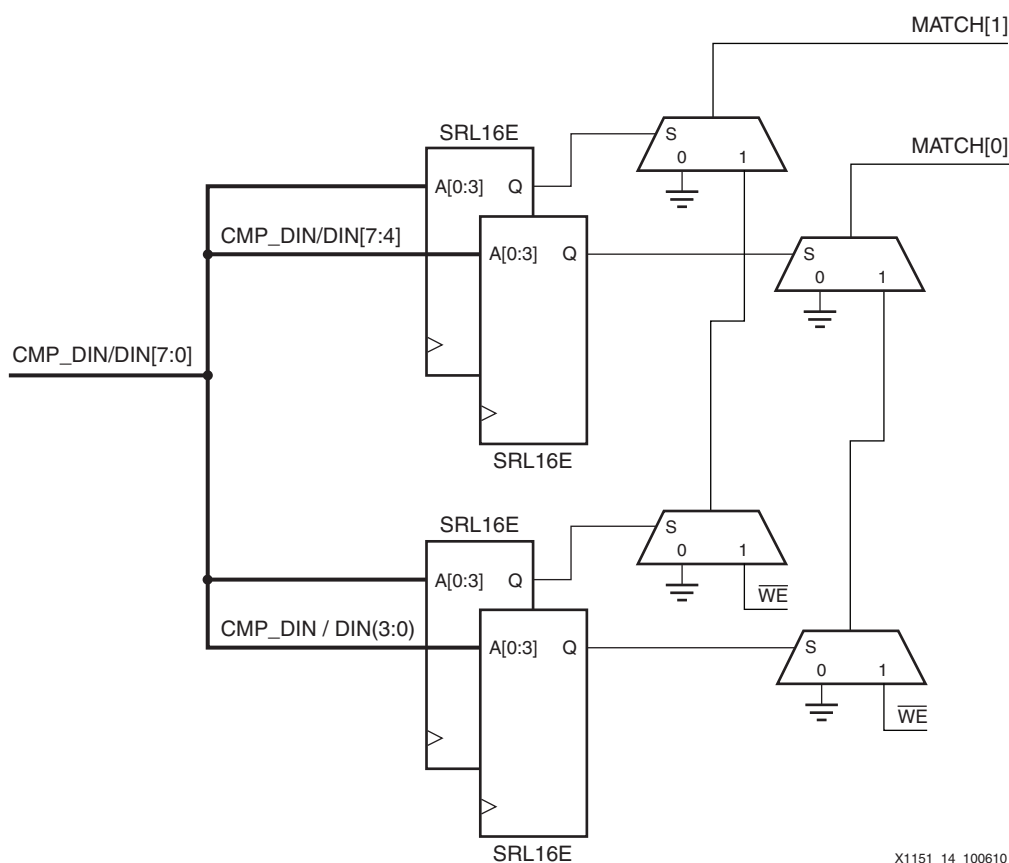


Figure 14: 8-Bit Wide by 1-Bit Deep CAM Read Operation

**Note:** With WE connected to the input of MUXCY, the user is able to read while waiting for a write operation to finish. Therefore, the address being written to is not included in the match operation because the read and write can occur simultaneously.

### Details of Write Implementation

The CAM address is decoded into a one-hot write enable bus that enables each CAM word for writing (Figure 15). It takes 16 clock cycles to shift in the result of the comparator into the SRL16E. The data input is compared with the value of the counter. When a match occurs, a 1 is shifted into the SRL16E. If there is no match, a 0 is shifted in. Each SRL16E has its own comparator to determine when to write a 1; this allows all SRL16s that make up one CAM address space to be written in parallel.

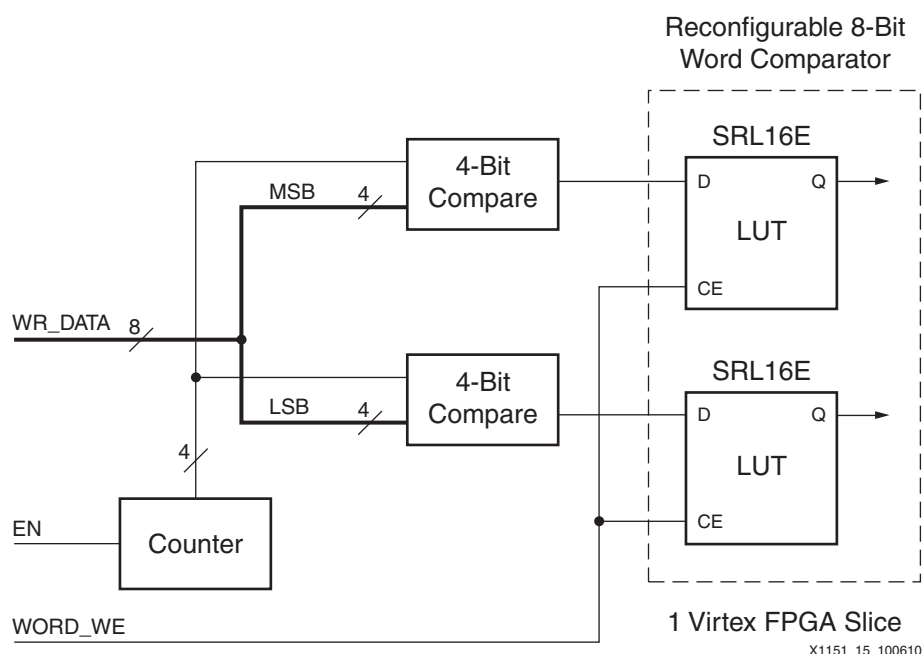


Figure 15: 8-Bit CAM Word Write Operation (16 Clock Cycles)

The timing diagram for a typical write operation is shown in Figure 16. The WR\_ADDR and DIN inputs are always used in a write operation. After WE is asserted, a write operation takes 16 rising clock edges to complete. The BUSY flag is asserted after a write operation has begun, and is deasserted on completion of the write operation. Whenever BUSY is Low, a write operation can begin on the next rising edge of the clock.

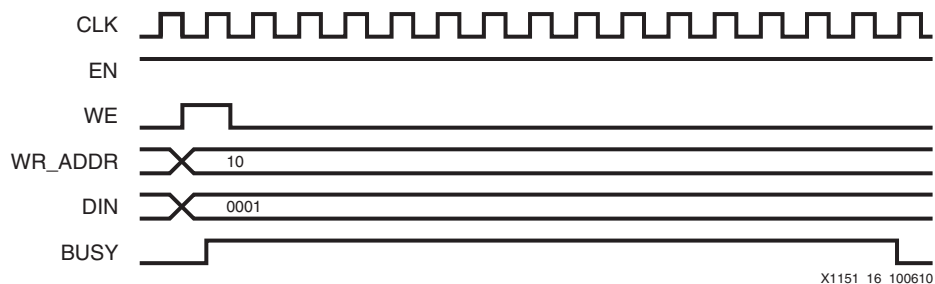


Figure 16: Timing Diagram for Write Operation

The WE input is ignored while BUSY is High. This means that it is impossible to interrupt a write operation after it has begun. Figure 17 demonstrates this. The behavior in this case is identical to that in Figure 16. Because of this feature, WE can remain High and a new write operation can begin immediately after this one concludes.

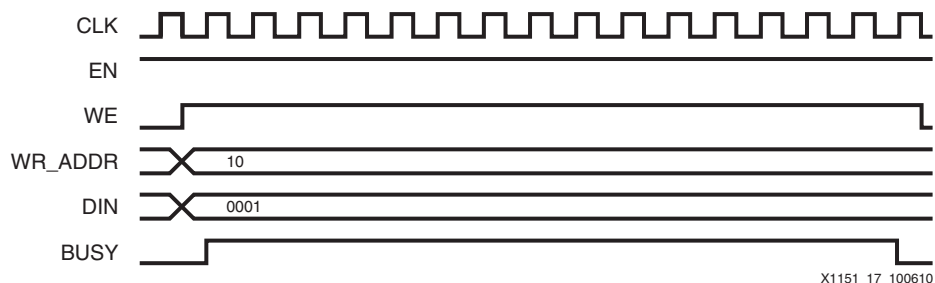


Figure 17: Timing Diagram for Write Operation with WE Held High

Like WE, the WR\_ADDR and DIN values are stored internally, so they too are ignored after a write operation has begun. Therefore, Figure 18 shows a third scenario that would produce identical results to Figure 16 and Figure 17. Because DIN can change after the write operation has begun, DIN can be used to read from the CAM while the write operation is in progress.

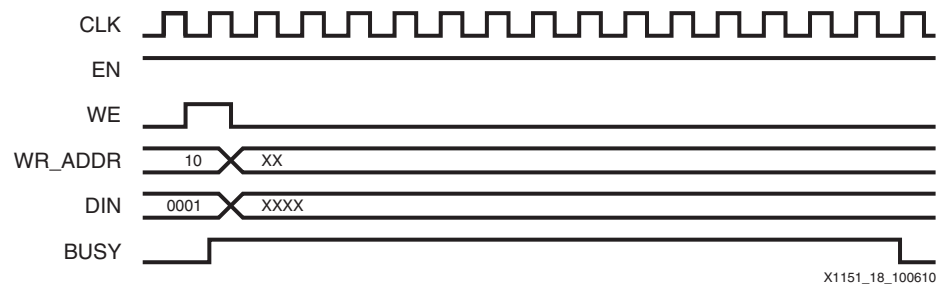


Figure 18: Timing Diagram for Write Operation with Changing WR\_ADDR and DIN

### Ternary Modes

A CAM with ternary modes enabled allows the use of X and U values in the CAM. Because more input combinations must be supported, the largest ternary CAM that can be implemented in a single SRL16E is smaller than for a non-ternary CAM. Standard ternary mode allows each bit to be in one of three states: 0, 1, and X. Each data bit in an enhanced ternary mode CAM can be in one of four states: 0, 1, X, and U. These values are encoded at the ternary CAM input using a combination of data bits (0 or 1) and mask bits (0 or 1).

For a 4-bit wide by 1-word deep non-ternary CAM, a single SRL16E can be used to store 4 bits of data at a single address. For ternary modes, a two-bit data input in combination with the two-bit mask value is encoded as a 4-bit word (see Table 4 and Table 5). This means that a single SRL16E can only be used to build a 2-bit wide by 1-word deep ternary mode CAM. Furthermore, unlike the non-ternary CAMs that can only represent a single 1 (or match) value in each CAM address location (i.e., each SRL16E), each ternary CAM address location can represent multiple input data matches. This is because a data input of 1X, for example, can match multiple two-bit values (10 and 11 in this case). The ternary mode CAM (using SRL16Es) performs both ternary reads and ternary writes.

#### Standard Ternary Encoder

The ternary encoder outputs four bits. Each of these four bits indicates whether or not a particular 2-bit encoded value can match the 2-bit ternary value being input. For example, for a ternary CAM implemented in a single SRL16E, the A bit is High (logic 1) whenever the ternary value can match 00. The ternary values that match 00 would be 00, 0X, X0, and XX.

**Note:** Ternary CAMs do not use all the possible addresses in an SRL16E.

The ternary encoder map is shown in Table 4.

Table 4: Standard Ternary Encoder Mapping

Input			Output (Address Inputs to SRL16E)				
Ternary Value	Data Input (d <sub>1</sub> d <sub>0</sub> )	Data Mask (m <sub>1</sub> m <sub>0</sub> )	A (00)	B (01)	C (10)	D (11)	ABCD (Addr[3:0])
00	00	00	1	0	0	0	1000
01	01	00	0	1	0	0	0100
0X	00, 01	01	1	1	0	0	1100
10	10	00	0	0	1	0	0010
11	11	00	0	0	0	1	0001

Table 4: Standard Ternary Encoder Mapping (Cont'd)

Input			Output (Address Inputs to SRL16E)				
1X	10, 11	01	0	0	1	1	0011
X0	00, 10	10	1	0	1	0	1010
X1	01, 11	10	0	1	0	1	0101
XX	00, 01, 10, 11	11	1	1	1	1	1111

The standard ternary encoder is built using these equations.

$$A = \bar{d}_1 \bar{d}_0 + m_1 m_0 + \bar{d}_1 m_0 + \bar{d}_0 m_1 \quad \text{Equation 1}$$

$$B = \bar{d}_1 d_0 + m_1 m_0 + \bar{d}_1 m_0 + d_0 m_1 \quad \text{Equation 2}$$

$$C = d_1 \bar{d}_0 + m_1 m_0 + d_1 m_0 + \bar{d}_0 m_1 \quad \text{Equation 3}$$

$$D = d_1 d_0 + m_1 m_0 + d_1 m_0 + d_0 m_1 \quad \text{Equation 4}$$

### Enhanced Ternary Encoder

The ternary encoder outputs four bits. Each of these four bits indicates whether or not a particular 2-bit encoded value matches the 2-bit enhanced ternary value being input. For example, the A bit is High (logic 1) whenever the ternary value can match 00. Any input with a value of U is mapped to SRL16E address 0000, and this address is never set to 1 because it is defined as unmatchable.

The enhanced ternary encoder map is shown in [Table 5](#).

Table 5: Enhanced Ternary Encoder Mapping

Input			Output				
Ternary Value	Data Input (d <sub>1</sub> d <sub>0</sub> )	Data Mask (m <sub>1</sub> m <sub>0</sub> )	A (00)	B (01)	C (10)	D (11)	ABCD (Addr[3:0])
00	00	11	1	0	0	0	1000
01	01	10	0	1	0	0	0100
0X	00	10	1	1	0	0	1100
0U	01	11	0	0	0	0	0000
10	10	01	0	0	1	0	0010
11	11	00	0	0	0	1	0001
1X	10	00	0	0	1	1	0011
1U	11	01	0	0	0	0	0000
X0	00	01	1	0	1	0	1010
X1	01	00	0	1	0	1	0101
XX	00	00	1	1	1	1	1111
XU	01	01	0	0	0	0	0000
U0	10	11	0	0	0	0	0000
U1	11	10	0	0	0	0	0000
UX	10	10	0	0	0	0	0000
UU	11	11	0	0	0	0	0000

The enhanced ternary encoder can be built using these equations.

$$A = \bar{d}_1 \bar{d}_0 \quad \text{Equation 5}$$

$$B = \bar{d}_1 \bar{m}_0 \quad \text{Equation 6}$$

$$C = \bar{d}_0 \bar{m}_1 \quad \text{Equation 7}$$

$$D = \bar{m}_1 \bar{m}_0 \quad \text{Equation 8}$$

### Ternary Write Cycle

Just like a standard SRL16E-based CAM, a ternary CAM write operation is performed in 16 clock cycles. As with the standard CAM, the ternary CAM uses a counter that counts from 15 down to 0, and comparator logic that determines the clock cycles on which to write a 1 or 0 into the SRL16E. Figure 19 shows an enhanced ternary CAM write operation.

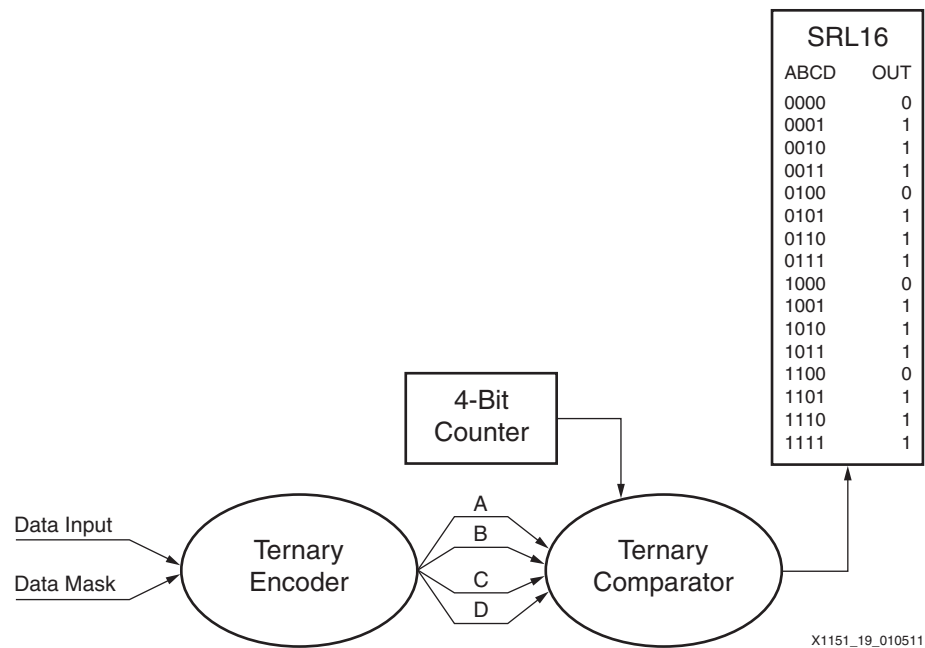


Figure 19: SRL16E Enhanced Ternary Write

The logic for the comparator is:

$$Out = (cnt_3 \cdot A) + (cnt_2 \cdot B) + (cnt_1 \cdot C) + (cnt_0 \cdot D) \quad \text{Equation 9}$$

The comparator logic compares the value of the counter to the vector ABCD from the ternary encoder in bitwise fashion and asserts a 1 when there is a match between the two. This ensures that a value of 1 is set on each data value represented by the SRL16E that is equivalent to the write input, as shown in Table 4 and Table 5. In Figure 19, the ternary data input is 1X, which can be represented by a data input of 10 and a data mask of 01—essentially the vector ABCD = 0011. When either of the two least significant bits of the counter are High (logic 1), the output bit is asserted. The comparator logic is the same for both standard ternary and enhanced ternary CAMs—only the encoder differs between the two modes.

### Ternary Read Operation

The read operation for a ternary CAM is essentially the same as the standard SRL16E-based CAM, except for the use of the ternary encoder. As shown in Figure 20, the ternary value (data Input and data mask) is converted into bits A, B, C, and D. ABCD are used as the address into the SRL16E. The output of the SRL16E is High (logic 1) if the ternary value (data input and data mask) are a match with the data stored at the CAM address served by this SRL16E. In this

example, the SRL16E was initially written with a value of 01. Thus, a read of value 01, 0X, X1, or XX creates a match at the address represented by this SRL16E. This logic is repeated for each CAM address.

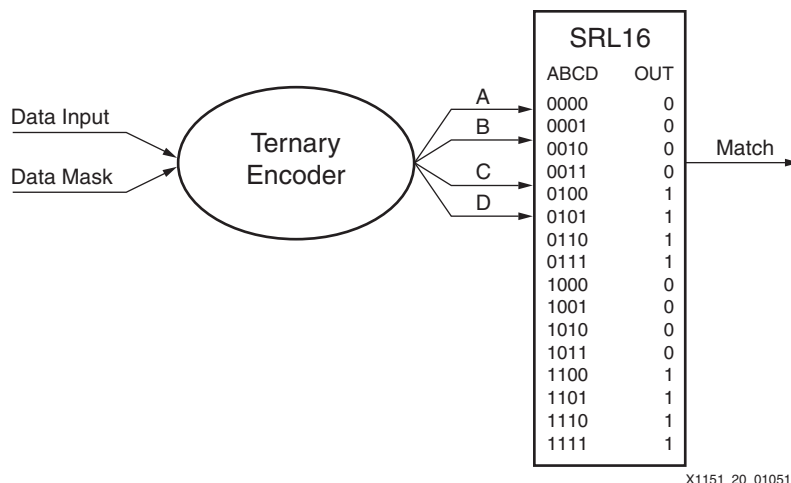


Figure 20: SRL16E Standard Ternary Read

## Reference Design

The reference design files for this application note can be downloaded from:

<https://secure.xilinx.com/webreg/clickthrough.do?cid=154257>.

The reference design checklist is shown in Table 6.

Table 6: Reference Design Checklist

Parameter	Description
<b>General</b>	
Developer Name	Xilinx
Target Devices (Stepping Level, ES, Production, Speed Grades)	Spartan-3, Xilinx Automotive (XA) Spartan-3, Spartan-3E, XA Spartan-3E, Spartan-3A, Spartan-3A DSP, Spartan-6, Virtex-4, Virtex-5, and Virtex-6/6L FPGAs
Source Code Provided	Yes
Source Code Format	VHDL
Design Uses Code/IP from Existing Application Note, Reference Designs, Third Party, or CORE Generator™ Software	Yes
<b>Simulation</b>	
Functional Simulation Performed	Yes
Timing Simulation Performed	Yes
Testbench Used for Functional and Timing Simulations Provided	No
Testbench Format	N/A
Simulator Software/Version Used	ModelSim, version 6.5d
SPICE/IBIS Simulations	No
<b>Implementation</b>	



Table 6: Reference Design Checklist (Cont'd)

Parameter	Description
Synthesis Software Tools/Version Used	XST, version 13.1
Implementation Software Tools/Versions Used	ISE software, version 13.1
Static Timing Analysis Performed	No
<b>Hardware Verification</b>	
Hardware Verified	No
Hardware Platform Used for Verification	N/A

The reference design contains VHDL source code and Perl scripts to customize the design, synthesize it in XST, and implement it through NGDBuild, MAP, and PAR. [Table 7](#) defines the reference design source files, and [Table 8](#) defines the scripts, project files, and documentation.

Table 7: CAM Design Files

Filename	Description
cam_wrapper.vhd	This is a customizable VHDL top-level core wrapper file with a simplified set of 15 generics.
cam_top.vhd	This core wrapper file translates the 15 simplified generics in the top-level wrapper file (cam_wrapper.vhd) to the full set of 27 generics in the top-level core file (cam_rtl.vhd).
cam_rtl.vhd	This top-level synthesizable core file instantiates all other submodules and uses an expanded set of generics.
cam_pkg.vhd	This is a package file containing commonly used constants and functions. <sup>(1)</sup>
cam_init_file_pack_xst.vhd	This file contains procedures for memory initialization, and reading and writing files.
init.mif	This text file contains a CAM-width x CAM-depth table for initializing the CAM, if applicable.
cam_regouts.vhd	This file registers the CAM outputs.
cam_control.vhd	This file generates control signals for the CAM, including an internal write enable and write counter, and user BUSY and READ_WARNING signals. <sup>(1)</sup>
cam_match_enc.vhd	This file contains the address match logic.
cam_mem.vhd	This file instantiates either block RAM or SRL16E memory based on code customization.
cam_mem_blk.vhd	For block RAM-based CAM, this file cascades multiple block RAM columns into rows for the final CAM width. <sup>(1)</sup>
cam_mem_blk_extdepth.vhd	For block RAM-based CAM, this file cascades multiple block RAMs into columns for the final CAM depth.
cam_mem_blk_extdepth_prim.vhd	For block RAM-based CAM, this file instantiates individual block RAM primitives depending on the FPGA architecture. <sup>(1)</sup>
dmem.vhd	For block RAM-based CAM, this file infers distributed memory for the erase RAM.

Table 7: CAM Design Files (Cont'd)

Filename	Description
cam_input.vhd	This file registers data input buses and instantiates ternary encoders for both read and write ports as necessary.
cam_input_ternary.vhd	This file pads the DIN and DATA_MASK inputs and instantiates the ternary encoder.
cam_input_ternary_ternenc.vhd	This file encodes DIN and DATA_MASK input buses as ternary encoded outputs for storage in memory.
cam_decoder.vhd	For SRL16E-based CAM, this file selects which 256-word block to write.
cam_mem_srl16_wrcomp.vhd	For SRL16E-based CAM, this is the write comparator for standard (non-ternary) mode.
cam_mem_srl16_ternwrcomp.vhd	For SRL16E-based CAM, this is the write comparator for ternary mode.
cam_mem_srl16.vhd	For SRL16E-based CAM, this file cascades c_width x 256-word deep blocks up to the final depth of the CAM, plus comparators and block decoder.
cam_mem_srl16_block.vhd	For SRL16E-based CAM, this file cascades multiple c_width x 1 word deep blocks into blocks of up to 256 words deep.
cam_mem_srl16_block_word.vhd	For SRL16E-based CAM, this file instantiates and cascades SRL16E primitives for c_width x 1 word of the CAM. <sup>(1)</sup>

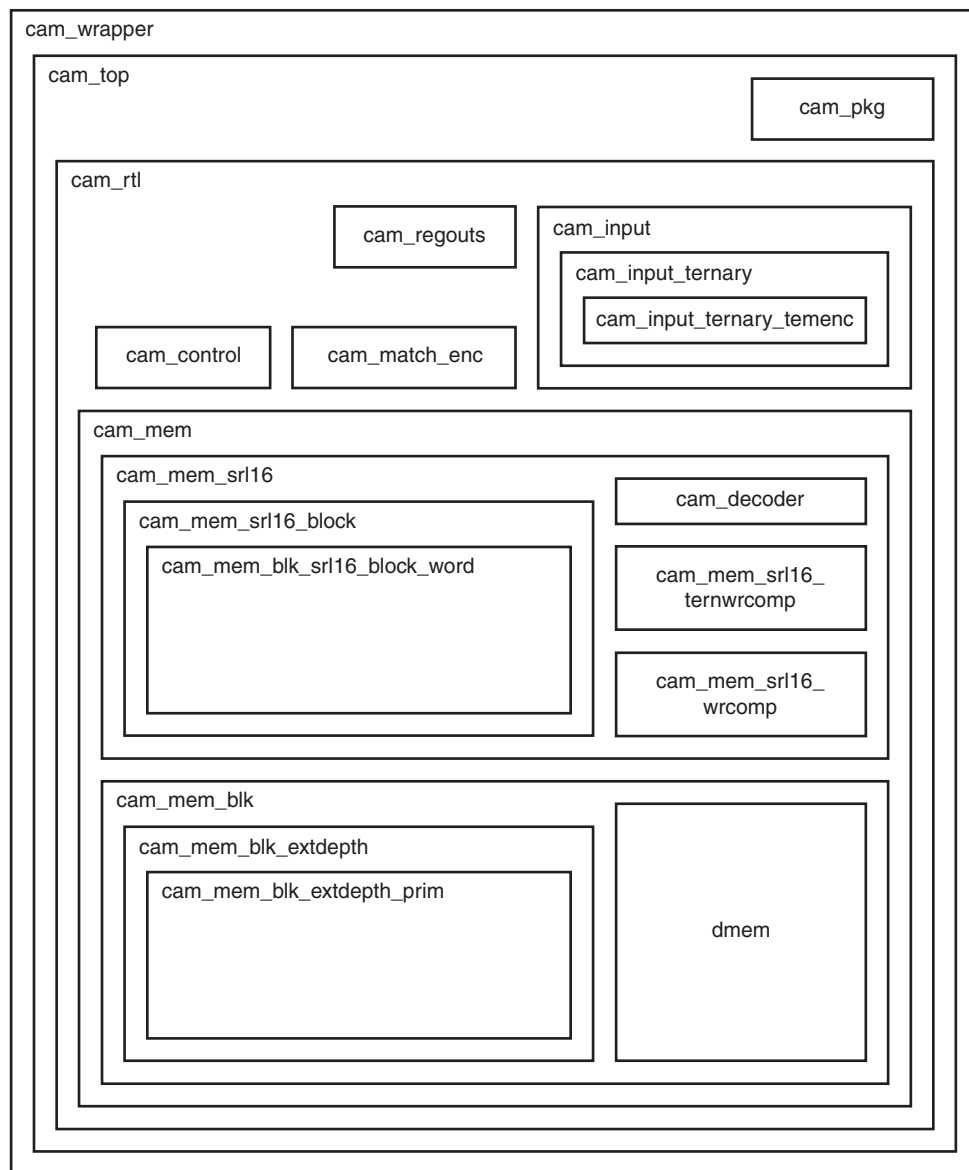
**Notes:**

1. The HDL source file contains architecture-specific component instantiations and/or coding and might require modification to support newer architectures than those listed in [Table 6, page 22](#).

Table 8: CAM Documentation and Script Files

Filename	Description
README_XAPP1151.txt	This file describes the reference design files and script files, and contains instructions for executing the provided scripts.
CustomizeWrapper.pl	This interactive Perl script is used to customize the top-level core wrapper file <code>cam_wrapper.vhd</code> .
WrapperTemplate.txt	This is the template for the customizable VHDL top-level core wrapper file <code>cam_wrapper.vhd</code> used by the script.
RunXST.pl	This Perl script synthesizes the wrapper and source files using XST.
vhdl_xst.scr	This XST script file contains the XST options for synthesis, including the target part.
vhdl_xst.prj	This XST project file contains the relative paths to the wrappers and source files to be synthesized.
Implement.pl	This Perl script runs NGDBuild, MAP, and PAR on the synthesized netlist.

Figure 21 shows the RTL hierarchy.



X1151\_21\_100610

Figure 21: RTL Hierarchy

## Compilation Parameters

The CAM reference design is parameterizable with a set of 15 simplified generics. An interactive command-line Perl script called `CustomizeWrapper.pl` is provided to facilitate configuration of these parameters. This script creates a top-level core wrapper file with the desired configuration. For additional control over the CAM implementation, the user can manually edit the 27 complete generics in the top-level core file (`cam_rtl.vhd`) and implement the design without the wrapper files (`cam_wrapper.vhd` and `cam_top.vhd`). Table 9 describes the full set of generics present in the top-level core file and the simplified set present in the top-level core wrapper file.

Table 9: RTL Parameters

Generic	Simplified Generics	Description
C_ADDR_TYPE	C_ADDR_TYPE	0 = Binary encoded. 1 = Single match unencoded. 2 = Multi-match unencoded.
C_CMP_DATA_MASK_WIDTH	C_WIDTH	Range: 1–512.
C_CMP_DIN_WIDTH	C_WIDTH	Range: 1–512.
C_DATA_MASK_WIDTH	C_WIDTH	Range: 1–512.
C_DEPTH	C_DEPTH	Range: 16–4096.
C_DIN_WIDTH	C_WIDTH	Range: 1–512.
C_FAMILY	C_FAMILY	Allowed values: virtex4, virtex5, virtex6, virtex6l, spartan3, spartan3e, spartan3a, spartan3adsp, aspartan3, aspartan3e, spartan6.
C_HAS_CMP_DATA_MASK	If C_TERNARY_MODE = 1 or 2 and C_HAS_CMP_DIN = 1, set to 1	1 indicates that the CMP_DATA_MASK port is present.
C_HAS_CMP_DIN	C_HAS_CMP_DIN	1 indicates that the CMP_DIN port is present.
C_HAS_DATA_MASK	If C_TERNARY_MODE = 1 or 2, set to 1	1 indicates that the DATA_MASK port is present.
C_HAS_EN	C_HAS_EN	1 indicates that the EN port is present.
C_HAS_MULTIPLE_MATCH	C_HAS_MULTIPLE_MATCH	1 indicates MULTIPLE_MATCH port present.
C_HAS_READ_WARNING	C_HAS_READ_WARNING	1 indicates that the READ_WARNING port is present.
C_HAS_SINGLE_MATCH	C_HAS_SINGLE_MATCH	1 indicates that the SINGLE_MATCH port is present.
C_HAS_WE	C_HAS_WE	1 indicates that the CAM is writeable (WE port present). 0 indicates that it is read only.
C_HAS_WR_ADDR	C_HAS_WE	1 indicates that the WR_ADDR port is present.
C_MATCH_ADDR_WIDTH	If C_ADDR_TYPE = 0, $\log_2\text{roundup}(C\_DEPTH)$ If C_ADDR_TYPE = 1 or 2, C_DEPTH	If C_ADDR_TYPE = 0, range: 4–12. If C_ADDR_TYPE = 1 or 2, range: 16–4096.

Table 9: RTL Parameters (Cont'd)

Generic	Simplified Generics	Description
C_MATCH_RESOLUTION_TYPE	C_MATCH_RESOLUTION_TYPE	1 indicates the highest matched address output. 0 indicates the lowest matched address output.
C_MEM_INIT	C_MEM_INIT	1 indicates MIF file is used to initialize CAM contents.
C_MEM_INIT_FILE	None	This is the hard-coded path to the <code>init.mif</code> file location.
C_ELABORATION_DIR	None	This is the hard-coded path to the <code>init.mif</code> file location.
C_MEM_TYPE	C_MEM_TYPE	1 indicates block RAM implementation. 0 indicates SRL16E implementation.
C_READ_CYCLES	Tied to 1	Not currently used.
C_REG_OUTPUTS	C_REG_OUTPUTS	1 indicates that the CAM outputs are registered (block RAM type only).
C_TERNARY_MODE	C_TERNARY_MODE	0 = No ternary mode. 1 = Standard ternary. 2 = Enhanced ternary.
C_WIDTH	C_WIDTH	Range: 1–512.
C_WR_ADDR_WIDTH	If C_HAS_WE = 1, equals $\log_2\text{roundup}(C\_DEPTH)$	Range: 4–12 (applies to writeable CAMs only).

## Creating a MIF File (Specifying CAM Contents)

The CAM reference design provides the option of initializing the memory contents during synthesis of the core in XST. For a read-only CAM, a MIF file is required during synthesis of the core.

The MIF file is a plain-text ASCII file written in binary with each line corresponding to an address in the CAM. The MIF file is written in the same way as a MIF file for a standard RAM. Because of this, the MIF file must have a number of lines equal to the depth of the CAM being used ( $n$ ), and each line must have a number of binary digits equal to the data width ( $DIN$  or  $m$ ) of the CAM. The `init.mif` file provided with the CAM reference design shows an initialization file for an 8-bit wide ( $m = 8$ ) by 16-word deep ( $n = 16$ ) CAM. These are the contents of the example `init.mif` file:

- 00100001
- 01010101
- 10101010
- 11110000
- 00001111

- 11001100
- 00110011
- 11100011
- 00011100
- 01000010
- 10000100
- 10011111
- 11101110
- 11111111
- 00000000
- 01100110

Each line of the MIF file specifies the match data for that address. For example, if line 3 (where the file starts at line 0) contained the value 11110000, the CAM signals a MATCH with MATCH\_ADDR = 3 when the DIN port has the value 11110000.

Additionally, for standard ternary CAMs, the MIF file can also specify don't-care bit entries as Xs. For example, if line 0 of the MIF file contains the value 000X0100, the CAM signals a MATCH with MATCH\_ADDR = 0 when the DIN port has either the value 00010100 or 00000100.

## Supported Design Tools

The design tools supported by this reference design are:

- ISE® Design Suite 13.1 (including XST 13.1 and xilperl)
- Mentor Graphics ModelSim 6.5d

## Resource Utilization and Performance

This reference design utilizes these resources:

- Block SelectRAM memory implementation: The number of block SelectRAM primitives required depends on the CAM depth and width selected in the `CustomizeWrapper.pl` script.
- SRL16E implementation: The number of SRL16E primitives required depends on the depth and width of the CAM and on whether a ternary CAM is selected in the `CustomizeWrapper.pl` script.
- Resource utilization and performance benchmarking: To maximize clock frequency for CAMs deeper than 256 words, a CAM with multi-match unencoded match address type and no additional match flags should be used. This corresponds to the following option settings:
  - Single-match flag = FALSE
  - Multiple-match flag = FALSE
  - Match address type = Multi-match unencoded

Table 10 and Table 11 show resource utilization and performance values for a 32-bit wide, 256-bit deep CAM for various match address types and storage element types implemented in a Virtex-5 device and a Spartan-3A device, respectively. These designs only contain a CAM and some glue logic, and are therefore representative of ideal performance rather than typical performance. To ensure timing closure, PERIOD constraints should be included in the user design. The benchmark designs do not include any implementation constraints besides a PERIOD constraint on the CAM clock. Virtex-6 devices have roughly the same resource utilization as Virtex-5 devices for the same CAM design, but with an estimated performance

gain of 15–20% over Virtex-5 devices. Spartan-6 devices use fewer resources than Spartan-3A devices, with an estimated performance gain of 10–15% over Spartan-3A devices.

In the benchmark designs described in [Table 10](#) and [Table 11](#), the core was encased in a wrapper with input and output registers to remove the effects of I/O delays from the results. Performance might vary depending on the design. Benchmarks were performed targeting the slowest and fastest speed grade devices in the Virtex-5 and Spartan-3A FPGAs.

- Virtex-5 FPGAs: XC5VLX220-1 (slowest) and XC5VLX220-2 (fastest)
- Spartan-3A FPGAs: XC3S1400A-4 (slowest) and XC3S1400A-5 (fastest)

**Table 10: Virtex-5 FPGA CAM Implementation: Resource Utilization and Performance**

Match Address Type	SRL16E					Block RAM				
	SRL16s	LUTs	Flip-Flops	Perf. (MHz) <sup>(1)</sup>	Perf. (MHz) <sup>(2)</sup>	Block RAMs	LUTs	Flip-Flops	Perf. (MHz) <sup>(1)</sup>	Perf. (MHz) <sup>(2)</sup>
Binary Encoded	2048	3438	60	100	110	32	1263	302	110	130
Single Match Unencoded	2048	3604	308	100	110	32	1405	302	100	120
Multi Match Unencoded	2048	3259	308	100	120	32	1144	303	110	130

**Notes:**

1. XC5VLX220, speed grade -1 (slowest).
2. XC5VLX220, speed grade -2 (fastest).

**Table 11: Spartan-3A FPGA CAM Implementation: Resource Utilization and Performance**

Match Address Type	SRL16E					Block RAM				
	SRL16s	LUTs	Flip-Flops	Perf. (MHz) <sup>(1)</sup>	Perf. (MHz) <sup>(2)</sup>	Block RAMs	LUTs	Flip-Flops	Perf. (MHz) <sup>(1)</sup>	Perf. (MHz) <sup>(2)</sup>
Binary Encoded	2048	3333	60	50	60	32	2236	303	60	70
Single Match Unencoded	2048	3430	309	50	60	32	2372	303	50	60
Multi Match Unencoded	2048	3336	309	50	60	32	2176	303	50	60

**Notes:**

1. XC3S1400A, speed grade -4 (slowest).
2. XC3S1400A, speed grade -5 (fastest).

[Table 12](#) and [Table 13](#) show how CAM content size affects the resource utilization and performance in Virtex-5 and Spartan-3A devices.

**Table 12: Virtex-5 FPGA SRL-Based CAM Implementation: Resource Utilization and Performance**

CAM Width	CAM Depth														
	16					256					1024				
	SRL16s	LUTs	Flip-Flops	Perf. (MHz) <sup>(1)</sup>	Perf. (MHz) <sup>(2)</sup>	SRL16s	LUTs	Flip-Flops	Perf. (MHz) <sup>(1)</sup>	Perf. (MHz) <sup>(2)</sup>	SRL16s	LUTs	Flip-Flops	Perf. (MHz) <sup>(1)</sup>	Perf. (MHz) <sup>(2)</sup>
8	32	148	27	190	200	512	1848	36	110	120	2048	7200	39	80	100
32	128	288	52	170	200	2048	3438	60	100	110	8192	13493	63	70	80
64	256	474	84	160	190	4096	5561	93	90	100	16348	21899	101	60	70

**Notes:**

1. XC5VLX220, speed grade -1 (slowest).
2. XC5VLX220, speed grade -2 (fastest).



Table 13: Spartan-3A FPGA SRL-Based CAM Implementation: Resource Utilization and Performance

CAM Width	CAM Depth														
	16					256					1024				
	SRL16s	LUTs	Flip-Flops	Perf. (MHz) <sup>(1)</sup>	Perf. (MHz) <sup>(2)</sup>	SRL16s	LUTs	Flip-Flops	Perf. (MHz) <sup>(1)</sup>	Perf. (MHz) <sup>(2)</sup>	SRL16s	LUTs	Flip-Flops	Perf. (MHz) <sup>(1)</sup>	Perf. (MHz) <sup>(2)</sup>
8	32	151	27	80	100	512	1727	36	50	70	2048	6646	42	40	50
32	128	299	52	80	100	2048	3333	60	50	60	8192	12892	66	40	40
64	256	499	84	80	90	4096	5464	92	40	50	- <sup>(3)</sup>	- <sup>(3)</sup>	- <sup>(3)</sup>	- <sup>(3)</sup>	- <sup>(3)</sup>

**Notes:**

1. XC3S1400A, speed grade -4 (slowest).
2. XC3S1400A, speed grade -5 (fastest).
3. Does not fit in an XC3S1400A device.

Table 14 and Table 15 show how CAM content size affects resource usage and performance clock periods for block SelectRAM memory in Virtex-5 and Spartan-3A devices. These devices only contain a CAM and some glue logic and for this reason represent ideal performance rather than typical performance.

Table 14: Virtex-5 FPGA Block RAM-Based CAM Implementation: Resource Utilization and Performance

CAM Width	CAM Depth														
	16					256					1024				
	SRL16s	LUTs	Flip-Flops	Perf. (MHz) <sup>(1)</sup>	Perf. (MHz) <sup>(2)</sup>	SRL16s	LUTs	Flip-Flops	Perf. (MHz) <sup>(1)</sup>	Perf. (MHz) <sup>(2)</sup>	SRL16s	LUTs	Flip-Flops	Perf. (MHz) <sup>(1)</sup>	Perf. (MHz) <sup>(2)</sup>
8	1	95	34	200	210	8	1077	277	130	160	32	4138	1048	100	110
32	4	170	57	170	180	32	1263	302	110	130	128	4677	1072	60	80
64	7	281	91	140	160	56	1721	335	90	110	- <sup>(3)</sup>	- <sup>(3)</sup>	- <sup>(3)</sup>	- <sup>(3)</sup>	- <sup>(3)</sup>

**Notes:**

1. XC5VLX220, speed grade -1 (slowest).
2. XC5VLX220, speed grade -2 (fastest).
3. Does not fit in the available block RAMs for an XC5VLX220 device.

Table 15: Spartan-3A FPGA SRL-Based CAM Implementation: Resource Utilization and Performance

CAM Width	CAM Depth														
	16					256					1024				
	SRL16s	LUTs	Flip-Flops	Perf. (MHz) <sup>(1)</sup>	Perf. (MHz) <sup>(2)</sup>	SRL16s	LUTs	Flip-Flops	Perf. (MHz) <sup>(1)</sup>	Perf. (MHz) <sup>(2)</sup>	SRL16s	LUTs	Flip-Flops	Perf. (MHz) <sup>(1)</sup>	Perf. (MHz) <sup>(2)</sup>
8	1	124	35	110	130	8	1368	278	70	80	32	5623	1079	50	- <sup>(3)</sup>
32	4	219	59	90	120	32	2236	303	60	70	- <sup>(3)</sup>	- <sup>(3)</sup>	- <sup>(3)</sup>	- <sup>(3)</sup>	- <sup>(3)</sup>
64	8	357	92	90	110	- <sup>(3)</sup>	- <sup>(3)</sup>	- <sup>(3)</sup>	- <sup>(3)</sup>	- <sup>(3)</sup>	- <sup>(3)</sup>	- <sup>(3)</sup>	- <sup>(3)</sup>	- <sup>(3)</sup>	- <sup>(3)</sup>

**Notes:**

1. XC3S1400A, speed grade -4 (slowest).
2. XC3S1400A, speed grade -5 (fastest).
3. Does not fit in the available block RAMs for an XC3S1400A device.

Table 16 and Table 17 show the resource usage and performance in Virtex-5 and Spartan-3A devices for a 32-bit wide and 256-word deep CAM using a multiple match unencoded match address type with other configuration options turned on.

Table 16: Virtex-5 FPGA CAM Implementation: Resource Utilization and Performance

Options	SRL16E					Block RAM				
	SRL16s	LUTs	Flip-Flops	Perf. (MHz) <sup>(1)</sup>	Perf. (MHz) <sup>(2)</sup>	Block RAMs	LUTs	Flip-Flops	Perf. (MHz) <sup>(1)</sup>	Perf. (MHz) <sup>(2)</sup>
Single Match Multiple Match Flags On	2048	3259	308	100	120	32	1144	303	110	130
Ternary Mode On	4096	5623	341	80	100	N/A <sup>(3)</sup>	N/A <sup>(3)</sup>	N/A <sup>(3)</sup>	N/A <sup>(3)</sup>	N/A <sup>(4)</sup>
Registered Outputs	N/A <sup>(4)</sup>	N/A <sup>(4)</sup>	N/A <sup>(4)</sup>	N/A <sup>(4)</sup>	N/A <sup>(4)</sup>	32	1144	562	110	130

**Notes:**

1. XC5VLX220, speed grade -1 (slowest).
2. XC5VLX220, speed grade -2 (fastest).
3. Ternary mode is not supported for block RAM-based CAMs.
4. Registered outputs are not supported for SRL-based CAMs.

Table 17: Spartan-3A FPGA CAM Implementation: Resource Utilization and Performance

Options	SRL16E					Block RAM				
	SRL16s	LUTs	Flip-Flops	Perf. (MHz) <sup>(1)</sup>	Perf. (MHz) <sup>(2)</sup>	Block RAMs	LUTs	Flip-Flops	Perf. (MHz) <sup>(1)</sup>	Perf. (MHz) <sup>(2)</sup>
Single Match Multiple Match Flags On	2048	3336	309	50	60	32	2176	303	50	60
Ternary Mode On	4096	5676	341	40	50	N/A <sup>(3)</sup>	N/A <sup>(3)</sup>	N/A <sup>(3)</sup>	N/A <sup>(3)</sup>	N/A <sup>(3)</sup>
Registered Outputs	N/A <sup>(4)</sup>	N/A <sup>(4)</sup>	N/A <sup>(4)</sup>	N/A <sup>(4)</sup>	N/A <sup>(4)</sup>	32	2265	561	60	70

**Notes:**

1. XC3S1400A, speed grade -4 (slowest).
2. XC3S1400A, speed grade -5 (fastest).
3. Ternary mode is not supported for block RAM-based CAMs.
4. Registered outputs are not supported for SRL-based CAMs.

## References

This document uses the following references:

1. [XAPP204](#), *Using Block RAM for High Performance Read/Write CAMs*.
2. [XAPP201](#), *An Overview of Multiple CAM Designs in Virtex Family Devices*.
3. [XAPP260](#), *Using Virtex-II Block RAM for High Performance Read/Write CAMs*.
4. [DS253](#), *Content-Addressable Memory v6.1*.

## Revision History

The following table shows the revision history for this document.

Date	Version	Description of Revisions
03/01/11	1.0	Initial Xilinx release.

## Notice of Disclaimer

Xilinx is disclosing this Application Note to you “AS-IS” with no warranty of any kind. This Application Note is one possible implementation of this feature, application, or standard, and is subject to change without further notice from Xilinx. You are responsible for obtaining any rights you may require in connection with your use or implementation of this Application Note. XILINX MAKES NO REPRESENTATIONS OR WARRANTIES, WHETHER EXPRESS OR IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, IMPLIED WARRANTIES OF MERCHANTABILITY, NONINFRINGEMENT, OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL XILINX BE LIABLE FOR ANY LOSS OF DATA, LOST PROFITS, OR FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, OR INDIRECT DAMAGES ARISING FROM YOUR USE OF THIS APPLICATION NOTE.