Delta39K™ And Quantum38K™ Dual-Port RAM

Introduction

The purpose of this application note is to provide information and instruction in implementing synchronous/asynchronous Dual-Port Random Access Memory (DPRAM) in Delta39K™ and Quantum38K™ Complex Programmable Logic Devices (CPLDs). Delta39K is a family of high density CPLDs with features such as PLL, SRAM, and true dual-port RAM. Quantum38K is a high-density, low-cost CPLD that features abundant logic resources and the dual-port RAM. The discussion of dual-ports in this application note apply to Delta39K and Quantum38K CPLDs. The Delta39K and the Quantum38K device families are fully supported by Cypress's software *Warp*™ R6.0 or later.

Dual-port memory is used in any system that requires transferring data from one system to another where the systems operate at different speeds. Dual-ports are widely used in communication equipment (Ethernet, LAN, switches, etc.) and multiprocessor systems.

There are two types of memory in Delta39K: cluster memory and channel memory. Cluster memory is located inside the Logic Block Cluster and channel memory is located at the intersection of horizontal and vertical routing channels **(***Figure 1*). Of these available memory blocks, only the channel memory blocks can be configured as dual-port RAM. In Quantum38K, there is no cluster memory. Only channel memory is available and this memory can be configured as dual-port RAM as in Delta39K.

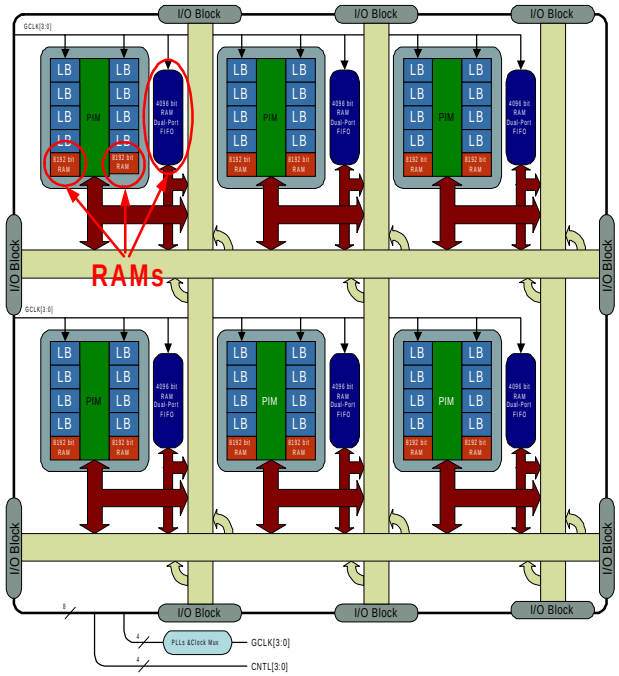


Figure 1. Delta39K Channel and Cluster RAM

This application note will discuss a general overview of the channel memory, configuration of the dual-port modules, expansion of the dual-port module across multiple blocks of channel memory, software support and a design example.

Total Available Memory

Table 1 lists the total amount of RAM available in each member of the Delta39K family and *Table 2* lists the total amount of RAM available in Quantum38K family. Since the dual-port module requires a channel memory block, the total available dual-port memory is the same as the total channel memory in the device.

Table 1. Available RAM (bits) in Delta39K

Part	Total RAM	Total Cluster RAM	Total Channel RAM (Available DPRAM)
Delta39K15	40K	32K	8K
Delta39K30	80K	64K	16K
Delta39K50	120K	96K	24K
Delta39K100	240K	192K	48K
Delta39K165	400K	320K	80K
Delta39K200	480K	384K	96K
Delta39K250	600K	480K	120K
Delta39K350	840K	672K	168K

Table 2. Available RAM (bits) in Quantum38K

Part	Total RAM	Total Channel RAM (Available DPRAM)
Quantum38K15	8K	8K
Quantum38K30	16K	16K
Quantum38K50	24K	24K
Quantum38K100	48K	48K

Overview of the Channel Memory

Each channel memory block is 4096 bits in size configurable as asynchronous dual-port, synchronous dual-port, or synchronous FIFO memory. The data path for the channel memory can also be configured for four different block sizes: 4096x1, 2048x2, 1024x4, 512x8.

All data, address, and control inputs to the channel memory are driven from horizontal and vertical channel Programmable Interconnect Matrices (PIM). All data and FIFO flag outputs drive dedicated tracks in the horizontal and vertical routing channels. These tracks are available as sources for all other PIMs located within the horizontal or vertical routing channels.

The clocks for the channel memory blocks are selected from four global clocks and the signals in the channel each from the horizontal and vertical channels. The clock muxes also include a polarity mux for each clock so that the user can choose an inverted clock signal. *Figure 2* shows the block diagram of the channel memory.

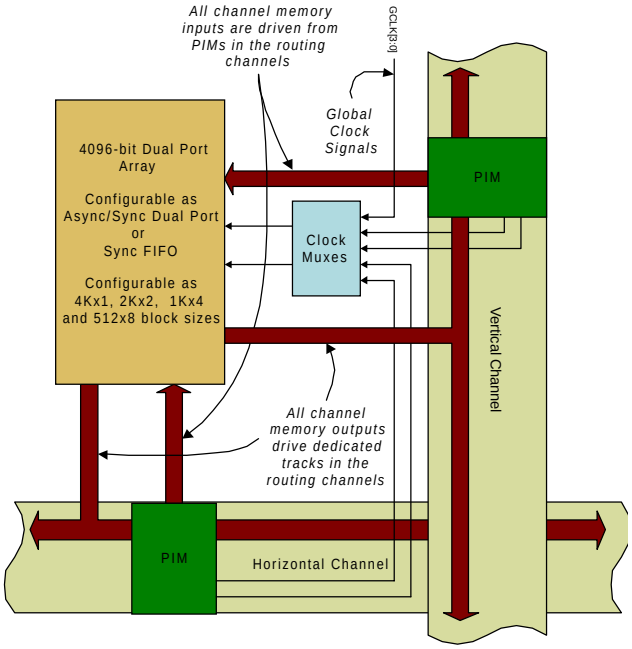


Figure 2. Block Diagram Of Channel Memory Block

Overview of the Dual-Port RAM

Each port of the module has separate address, data and control signals that can be accessed simultaneously. Both of the ports have to be of the same type (asynchronous or synchronous) and of same block size. Each port also has the ability to register the output data, which can be used for a synchronous pipelined mode of operation, or to register the output of an asynchronous memory access. The output register also includes an asynchronous RESET capability. Port A mainly interfaces with the horizontal routing channel while Port B with the vertical routing channel.

Inputs and Outputs

Inputs from Routing Channels

All inputs to the dual-port are driven from PIMs in the horizontal and vertical routing channels. The data inputs to Port A and Port B can be driven from either channel. For example, if the data input to Port A is driven from horizontal channel then the Port B data input will be driven from vertical channel. The address and the reset signals for Port A are driven the horizontal channel and the address and rest signals for Port B from the vertical channel. The write enable and clocks for both ports can be driven from either the horizontal or vertical routing channels.

Output to Routing Channels

The data outputs of the ports can be driven onto either the horizontal or vertical routing channels. But they have to be routed to different channels. For example, if output data from Port A is routed to horizontal channel then the output from Port B should be routed to vertical channel. The address match signal is routed to both channels.

Registering Output Data

Each data output port has an optional data output register. This allows each port to be configured for synchronous pipelined operation. The output registers can also be used to simply register the output of an asynchronous dual-port. This is useful when the channel memory is being used as a look-up-table for logic. In addition, the output registers have an asynchronous register RESET. Each port can individually be configured for registered or non-registered data outputs.

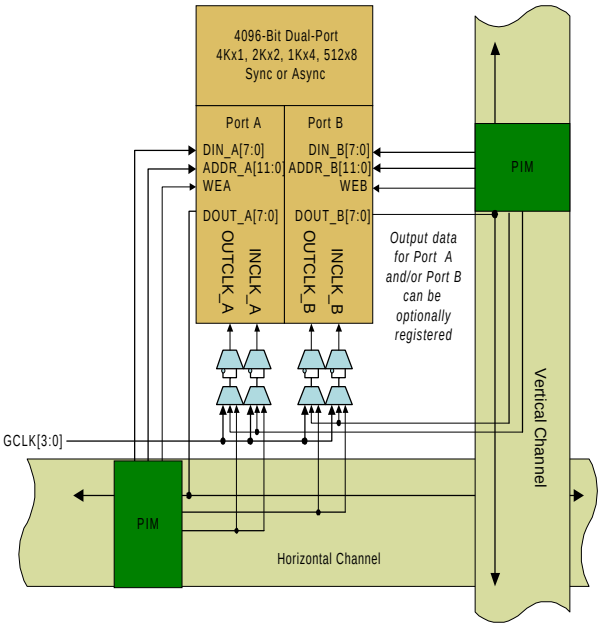


Figure 3. Dual-Port Memory Routing Interface

Input and Output Clock Selection

The module uses four independent clock input signals. Each input port uses a separate clock and each output port has a separate clock for the associated data output register. These clocks can select the input signals from any of four global clock signals and the PIM signals of the horizontal and vertical routing channels.

Figure 3 shows the routing interface of the dual-port memory with the horizontal and vertical channels.

Read/Write Collisions

There are 4 possible collision scenarios: Port A and Port B reading the same memory location, Port A reading while Port B writing from/to the same address, Port A writing while Port B reading to/from the same address and Port A and Port B writing to the same memory location. No arbitration is needed when both Port A and Port B are trying to read from the same address. Port A will always win when both Port A and Port B try to write to the same address. Port B will be blocked and have to wait until Port A completes the write to the memory location. When one port tries to read and the other port tries to write, the write always wins and the other port will read the new data. When one port tries to read first, as soon as the request by the other port to write to the same address is received, the control is switched to the port trying to write. After writing is completed, the reading port reads the new data. Whenever the addresses on the two ports match, the addr_matchb signal transitions low. *Table 3* summarizes how address collisions are resolved by the dual-port memory.

Table 3. Dual-Port Behavior During Collisions

Port A	Port B	Result Of Arbitration	Comment
Read	Read	No arbitration required	Both ports read at the same time
Write	Read	Port A gets priority	If port B requests first then it will read the current data. The output will then change to the newly written data by port A
Read	Write	Port B gets priority	If port A requests first then it will read the current data. The output will then change to the newly written data by Port B
Write	Write	Port A gets priority	Port B is blocked until Port A is finished writing

Expandability

Dual-port block size can be expanded to form deeper or wider dual-port blocks. Multiple memory blocks will be used to perform the expansion. Additional logic may be needed for the expansion. In the case of width expansion, the data path is split evenly between the channel blocks being utilized. The same address, clock, write enable and reset signals are given to all the blocks. This is illustrated in the *Figure 4*. In the depth expansion the same data, clock and reset signals are given to all the blocks. The lower address bits are used by all the blocks but the MSB of the address is combined with the write enable to determine which block it is being written to. The MSB of the address will also be used to select the data via an output mux. Both the output mux and the write enable have to be implemented in a logic cluster and routed to and from the appropriate channel memory blocks. This expansion is demonstrated in the *Figure 5*.

The required number of blocks to be used for these expansions is determined by *Warp* based on the user requirement of the size of the memory.

Thus we classify the dual-port RAM into two distinct categories: depths less than or equal to 4096 and depths greater than 4096.

Depths Less Than or Equal To 4096

Expanded blocks that fall into this classification can be easily implemented by simply utilizing the proper number of channel memory blocks in parallel. This configuration doesn't require depth expansion. *Warp* never uses depth expansion when the depth is less than 4096. This is to avoid the additional logic delays involved in depth expansion.

For example, two channel memory blocks, each configured for 1024x4, can implement a 1024x8 dual-port RAM. This can be achieved by width expansion. Two 1024X4 configured blocks are width expanded to form a 1024X8 memory configuration. No additional circuitry or multiplexing is needed. One of the ports will incur a routing penalty because common control signals will have to be switched between routing channels.

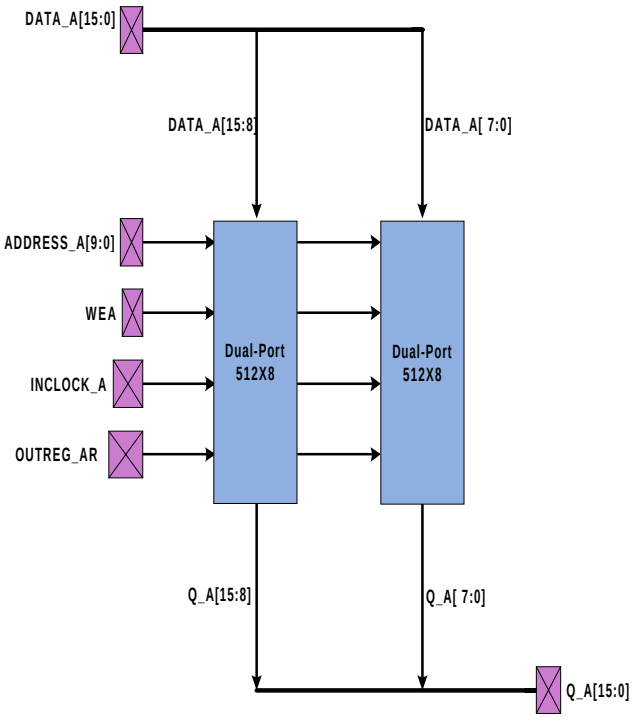


Figure 4. Width Expansion for 512X16 Memory

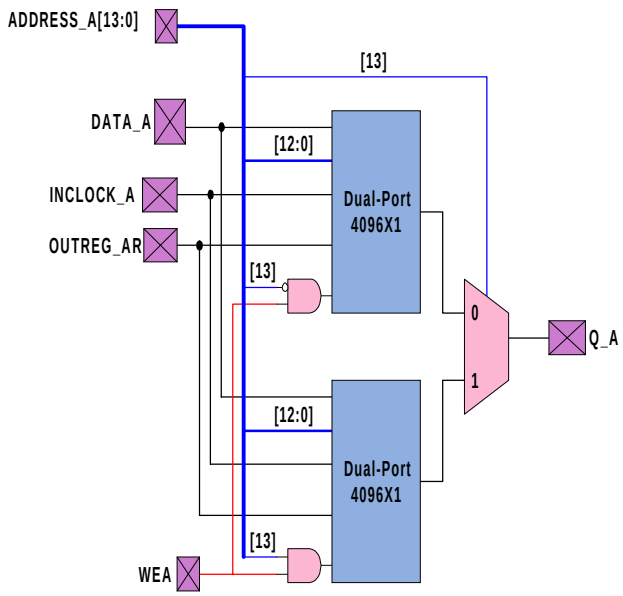


Figure 5. Depth Expansion for 8192X1 memory

Depths Greater Than 4096

Memories of these depths are more difficult to implement. There will be additional logic needed to handle both the address and the data paths. In these cases, *Warp* uses 4096X1 configured blocks and appropriately arranges them based on the required memory size. In other words, both width and depth expansion techniques are utilized to implement the required memory size. For example, a 8192x2 dual-port RAM will need four channel memory blocks configured as 4096x1 blocks. Width expansion is used to generate two 4096X2 memory blocks which are then combined with external logic to generate a 8192X2 memory block by depth expansion.

The additional logic that is involved in the implementing a depth expansion make the access times slower and place a larger burden on the routing resources.

Software Support

All the architectural details discussed in the earlier sections of this application note are provided for better understanding of the Delta39K and Quantum38K dual-ports by the user. However, the user does not have to be concerned about these details in using the dual-port. The dual-port can be implemented by instantiating *CY_RAM_DP* module in their HDL design. The Delta39K and the Quantum38K device families are fully supported by Cypress's *Warp* R6.0 or later. *Warp* is a software package that allows users to enter designs in either VHDL or Verilog. Once the design's description is written, the HDL is synthesized and fit into a target device.

LPM Modules

There are five types of RAM-based LPM modules which are supported by *Warp* R6.0 for implementation in Delta39K. *Table 4* lists all the memory components supported by Cypress for Delta39K. Of these the *CY_FIFO* is not supported for Quantum38K. Only the dual-port RAM module will be covered in this application note.

The dual-port LPM Module *CY_RAM_DP* declaration with the generic variables, ports, and default values is described in *Appendix A*. The lpm_file value should be set to a text file with rom extension containing the initialization data in Intel Hex format if you want to initialize the module. It is not possible to reinitialize the module during operation using this file.

Instantiation

Warp users will specify dual-port modules through instantiations of the *CY_RAM_DP* component in their VHDL or Verilog code. The *CY_RAM_DP* component is parameterized to support all possible size configurations and all of the dual-port features. *Table 1* of *Appendix A* lists the names, types and characteristics of all the ports in the *CY_RAM_DP* component. *Table 2* of *Appendix A* lists the parameters used to configure the size and features of the *CY_RAM_DP* component.

To use the *CY_RAM_DP* component in VHDL designs, an instance of the *CY_RAM_DP* component must be declared and the required ports connected. Additionally, required generic parameters must be mapped to a value. Optional ports or generics of the *CY_RAM_DP* component can also be mapped to take advantage of additional features.

To use *CY_RAM_DP* in Verilog designs, the *CY_RAM_DP* module must be instantiated with the required ports connected. In Verilog, the properties of the *CY_RAM_DP* are specified by default parameters. These default parameters can be changed by using the *defparam* override technique supported by *Warp*. Unspecified parameters are assigned their default values. Additional features of the *CY_RAM_DP* module can be used by connecting the appropriate ports and specifying the necessary parameters.

By using the LPM Template Menu Option in *Warp* R6.0, you can insert the *CY_RAM_DP* instantiation template.

Table 4. LPM Modules for Delta39K/ Quantum38K

Library Component	Implementation	Description
MRAM_IO	Cluster/Channel	Single port RAM with bidirectional data in and data out
MRAM_DQ	Cluster/Channel	Single port RAM with separate data in and data out
MROM	Cluster/Channel	ROM
CY_FIFO	Channel only	Clocked FIFO (not supported in Quantum38K)
CY_RAM_DP	Channel only	Dual-Port RAM

Initialization

Memory can be initialized using a ROM file written in Intel Hex Format. This file can then be included in the project using the add ROM files menu when targeting the Delta39K or Quantum38K.

Figure 6 shows the data record format for the Intel Hex record. All records start with a colon. The rest of the data uses 2 ASCII characters (0-9 and A-F) to represent 1 byte of data

Record Mark	reclen	Load Offset	Rec-types	Info /Data	Check-sum
1 char	2 chars	4 chars	2 chars	2n chars	2 chars
	1 byte	2 bytes	1 byte	n bytes	1 byte

Figure 6. Intel HEX Data Record Format

RECORD MARK is always the colon character.

RECLEN is the number of bytes contained in the data field (0 to 255 bytes).

LOAD OFFSET is 16-bit starting load offset of the data bytes.

There are 2 RECTYPES used: ‘00’ for Data record and ‘01’ for End of File record.

DATA contains a pair of ASCII digits to represent each byte of data.

CHECKSUM is the two’s complement of the 8-bit addition of the bytes in RECLEN, LOAD OFFSET, RECTYPE, and DATA, not including the RECORD MARK

For example, to write a Data record that contains four bytes of data: 0x05, 0xFA, 0x39, and 0x4D, the record line will be

: 0400000005FA394D77

The 0x77 is obtained by doing 8-bit addition:

0x04 + 0x05 = 0x09

0x09 + 0xFA = 0x03

0x03 + 0x39 = 0x3C

0x3C + 0x4D = 0x89

-(0x89) = 0x77

Then, to add another Data record containing 2 bytes of data: 0x55 and 0xAA, the next record line will be:

:0200040055AAFB

Note that the LOAD OFFSET is 0004.

Once all the Data records are added, add the End of File record line as shown below:

:00000001FF

All memory components can be initialized with a ROM file with the exception of the FIFO module. A ROM file to be used may only contain initialization data for the memory in Intel Hex format and has a .rom extension. In the source code, the ROM file is specified by mapping the “lpm_file” generic to the name of the desired ROM file. Each byte of the data initializes one word when word size is less than or equal to 8 bits (for example: 4096x1, 2048x2, 1024x4, 512x8). Each byte in Hex file will only initialize part of one word when word size is greater than 8 bits. You will need to use more words as necessary to initialize the rest of the bits. *Warp* ignores any bits that are out of range for the word size. So, a 1 bit word initialized to 0x55 is set to 0x01. Any memory not initialized will be set to 0.

An example of the HEX file to initialize the first 256 bytes of a 512X8 dual-port is given in the *Appendix B*. In this example, the first 256 bytes of a 512X8 dual-port are written to initial values using the ROM file "Example.rom". Each line of represents a record. As discussed earlier, each record line starts with a colon. The first record line in the example is writing 3 bytes of data starting at the address 0000 and the rest of the lines are writing different number of bytes of data but at consecutive addresses. The last line is the end of file record line indicating the end of the ROM file.

Design Example

Let us consider an example. A designer needs a 16-bit wide dual-port. To instantiate and connect the dual-port, the LPM module component has to be instantiated. The VHDL and Verilog codes for this example are given in the *Appendix C*. When the addresses of both the ports match, the match signal becomes low (the addr_matchb is an active LOW signal). When the match signal is LOW, if both the write enables are HIGH (both ports are writing), then data on Port B is not written and the error indicator of Port B (invb) becomes HIGH. When either port A or Port B are reading while the other port is writing, data on the reading port may be corrupt or invalid. Thus that particular error indicator becomes high. When both the ports are reading, then there are no potential data hazards. Both ports receive the same data. When match signal is high, each port is accessing a separate address without error.

Compiling the given code will instantiate the dual-port RAM. To initialize the above RAM, a HEX file, which was discussed in the earlier section, has to be created and this file is mapped to the “*lpm_file”* in the instantiation code. The HEX file can be written to initialize the contents of the RAM. In this example, the file "Example.rom" is mapped which is discussed in the Initialization section.

Switching Characteristics

The Timing diagrams and their parameters are discussed in *Appendix D*. These diagrams show the performance of both the ports and their characteristics in different modes of performance. All of these timing diagrams are given for one port (Port A) except for the Address match signal diagrams.

The Switching parameters and their characteristics for the asynchronous dual-port are shown in *Figure 1* of *Appendix D.* The dual-port is always in read mode when the Write enable is LOW and is in the write mode only when the Write enable goes HIGH. The Write enable determines the Set-up and Hold time as there is no Global clock. The dual-port cannot be run faster than the access time. The maximum speed of the dual-port should be the t_{CHMAA} which is the channel memory access time. The parameter t_{CHMOHA} determines the period when the previous data is available to be sampled after the address is changed.

The timing diagrams for the synchronous dual-port for both the Pipeline and Flow through modes is shown in *Figure 2* and *Figure 3* of *Appendix D.* These timing parameters are

measured with respect to a global clock. In the flow through mode, the outputs are asynchronous. Thus the output can be accessed after time t_{CHMDV1}, the Global clock to data valid, from the clock edge where the inputs are clocked. In the case of pipeline mode, the output is also synchronous and this timing model is achieved assuming that the speed of the clocks of both the input and output registers is the same. In this case, the output can be accessed after the time t_{CHMDV2} from the next clock edge.

The timing diagrams for the Address match signal for both the asynchronous and synchronous modes are shown in *Figure 4* and *Figure 5* of *Appendix D.* When both the addresses match, the addr_matchb signal goes LOW after time, t_{CHMBA} in asynchronous dual-port and t_{CHMBDV} in the case of synchronous dual-port. Then the write enable signals determine which port is written to. In the case of synchronous dual-port, the address match signal goes low after the addresses are clocked.

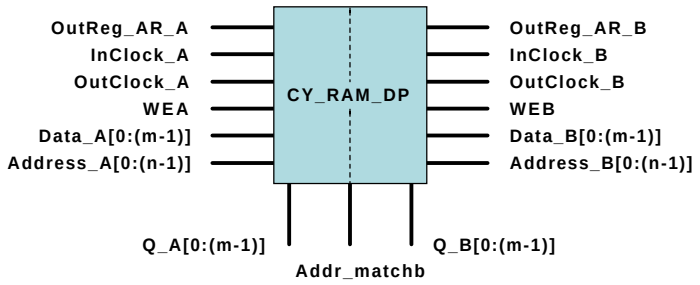
Additional timing diagrams and the time values for the parameters in this application note can be obtained from the Delta39K and Quantum38K data sheets located at:

http://www.cypress.com/pld/datasheets.html

Conclusion

This application note will be useful in understanding and implementing dual-port RAM in Delta39K and Quantum38K. The dual-port RAM in both these devices is both width and depth expandable and support both synchronous and asynchronous implementations. Dual-port memories can be instantiated in user designs through the use of the *CY_RAM_DP* module in Cypress *Warp* R6.0 or later software. Delta39K features the speed, ease of use and predictability of CPLDs, and adds a programmable PLL, true FIFO and Dual-Port Memories, abundant I/Os, a unique self-boot solution and more embedded RAM than any other CPLD.The combination of abundant logic resources and true dual-port memory make Delta39K and Quantum38K CPLDs an ideal solution for many Data Communications Systems.

Appendix A: LPM Module



LPM PROPERTIES

LPM_WIDTH (m)	**LPM_ADDRESS_CONTROL**
LPM_WIDTHAD (n)	**LPM_OUTDATA_A**
LPM_NUMWORDS	**LPM_OUTDATA_B**
LPM_INDATA	**LPM_FILE**
	LPM_HINT

Table 1. CY_RAM_DP Port Description

Port Name	Type	Usage	Description	Comments
Data A	In	Required	Data inputs for Port A	Vector, LPM_WIDTH wide
Data B	In	Required	Data inputs for Port B	Vector, LPM_WIDTH wide
Address A	In	Required	Address inputs for Port A	Vector, LPM_WIDTHAD wide
Address B	In	Required	Address inputs for Port B	Vector, LPM_WIDTHAD wide
Q_A	Out	Required	Data outputs for Port A	Vector, LPM_WIDTH wide
Q_B	Out	Required	Data outputs for Port B	Vector, LPM_WIDTH wide
Addr_matchb	Out	Required	Does the address match (busy)	
WEA	In	Required	Read/write control for Port A	
WEB	In	Required	Read/write control for Port B	
InClock_A	In	Optional	Clock signal for Port A	If defined, InClock_B must also be specified
InClock_B	In	Optional	Clock signal for Port B	If defined, InClock_A must also be specified
OutClock_A	In	Optional	Clock signal for Port A output	
OutClock_B	In	Optional	Clock signal for Port B output	
OutRegA_AR	In	Optional	Port A Output register asynchronous reset	Reset output registers when HIGH
OutRegB_AR	In	Optional	Port B Output register asynchronous reset	Reset output registers when HIGH

Table 2. CY_RAM_DP Property Description

LPM Property	Usage	Value	Comments
LPM_WIDTH	Required	LPM value > 0	Width of input and output vectors of both ports
LPM_WIDTHAD	Required	LPM value > 0	Width of address of both ports
LPM_NUMWORDS	Optional	0 < LPM value <= $2^{LPM\_WIDTHAD}$	Specifies the depth of the memory. Default is $2^{LPM\_WIDTHAD}$
LPM_INDATA	Optional	LPM_REGISTERED or LPM_UNREGISTERED	Indicates if Data port is registered. Default is LPM_REGISTERED.
LPM_ADDRESS_CONTROL	Optional	LPM_REGISTERED or LPM_UNREGISTERED	Indicates if Address and WE ports are registered. Default is LPM_REGISTERED
LPM_OUTDATA_A	Optional	LPM_REGISTERED or LPM_UNREGISTERED	Indicates if Q of port A is registered. Default is LPM_REGISTERED
LPM_OUTDATA_B	Optional	LPM_REGISTERED or LPM_UNREGISTERED	Indicates if Q of port B is registered. Default is LPM_REGISTERED
LPM_FILE	Optional	File name	File for RAM initialization.
LPM_HINT	Optional	Not used	Not used

Appendix B: Example of an Intel HEX File to initialize the first 256 bytes of a 512X8 Dual-Port

Example.rom

```
:03000000020023D8
:10000300E50B250DF509E50A350CF5081200132259
:10001300AC12AD13AE10AF1112002F8E0E8F0F2244
:0C002300787FE4F6D8FD7581130200031D
:10002F00EFF88DF0A4FFEDC5F0CEA42EFEEC88F016
:04003F00A42EFE22CB
:0400430005FA394D34
:0200470055AAB8
:100490005090389FA77001017192021443F3B2F2E
:10005900464C5549442050524F46494C4500464C60
:1000690033354445465F5C55BAACAF1F2F3F1F2F50
:200079001A2A3A4A5A6A7A8A9A1B2B3B4B5B6B7B8B9B1C2C3C4C5C6C7C8C9C1D2D3D4D20DA
:200099005D6D7D8D9D1E2E3E4E5E6E7E8E9E1F2F3F4F5F6F7F8F9FA1A2A3A4A5A6A7A821EC
:2000B900A9B1B2B3B4B5B6B7B8B9C1C2C3C4C5C6C7C8C9D1D2D3D4D5D6D7D8D9E1E2E322EF
:2700D900E4E5E6E7E8E9F1F2F3F4F5F6F7F8F91122334455667788 99AABBCCDDEEFF12131415161718192335
:00000001FF
```

Appendix C: The Source code for Design Example

VHDL Example

--Delta39K and Quantum38K sample code showing use of internal Dual-Port

--File: Example.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use work.lpmpkg.all;

ENTITY arbitrator IS PORT ( a, b: in std_logic_vector (15 downto 0);          -- Port A and B data in
            adda, addb: in std_logic_vector (7 downto 0);                    -- Port A and B address busses
            wena, wenb: in std_logic;                                         -- Port A and B write enables
            clka, clkb: in std_logic;                                         -- Port A and B clocks
            inva, invb: out std_logic;                                        -- Port A and B invalid (used for arbitration)
            a_out, b_out: out std_logic_vector (15 downto 0)  );             -- Port A and B output
END arbitrator;

ARCHITECTURE arch_arbitrator OF arbitrator IS
signal match: std_logic;

BEGIN

arbitrate:process (wena, wenb, match)
begin
        if (match = '0') then
            if (wena = '1' and wenb = '1') then
                inva <= '0';
                invb <= '1';
            elsif (wena = '1' and wenb = '0') then
                inva <= '0';
                invb <= '1';
            elsif (wena = '0' and wenb = '1') then
                inva <= '1';
                invb <= '0';
            else
                inva <= '0';
                invb <= '0';
            end if;
        else
            inva <= '0';
            invb <= '0';
        end if;
end process;

U1: cy_ram_dp                                                    -- Dual-port RAM

generic map(
  lpm_width              => 16,
  lpm_widthad            => 8,
  lpm_numwords           => 0,                                   -- optional
  lpm_indata             => LPM_REGISTERED,                      -- optional
  lpm_address_control    => LPM_REGISTERED,                      -- optional
  lpm_outdata_a          => LPM_REGISTERED,                      -- optional
  lpm_outdata_b          => LPM_REGISTERED,                      -- optional
  lpm_file               => "example.rom",                       -- optional
  lpm_hint               => speed                                -- optional
   )

port map(

   data_a                => a,
   data_b                => b,
   address_a             => adda,
   address_b             => addb,
   q_a                   => a_out,
   q_b                   => b_out,
   addr_matchb           => match,
   wea                   => wena,
   web                   => wenb,
   inclock_a             => clka,                                -- optional
   inclock_b             => clkb,                                -- optional
   outclock_a            => clka,                                -- optional
   outclock_b            => clkb,                                -- optional
   outrega_ar            => zero,                                -- optional
   outregb_ar            => zero                                 -- optional

 );

END arch_arbitrator;
```

Verilog Example

//Delta39K and Quantum38K sample code showing use of internal Dual-Port
// File: Example.v

```
`include "rtl.v"
`include "lpm.v"
 module arbitrator(a, b, adda, addb, wena, wenb, clka, clkb, inva, invb, a_out, b_out);

   input[7:0]adda, addb;
   input[15:0] a, b;
   input wena, wenb;
   input clka, clkb;
   output[15:0] a_out, b_out;
   output inva, invb;
   reg inva, invb;
   wire match;

   defparam U0.lpm_width= 16;
   defparam U0.lpm_widthad= 8;
   defparam U0.lpm_numwords= 0;                                    // optional
 //defparam U0.lpm_indata = `LPM_REGISTERED;                       // optional
 //defparam U0.lpm_address_control = `LPM_REGISTERED;              // optional
 //defparam U0.lpm_outdata_a = `LPM_REGISTERED;                    // optional
 //defparam U0.lpm_outdata_b = `LPM_REGISTERED;                    // optional
 //defparam U0.lpm_file = "example.rom";                           // optional
 //defparam U0.lpm_hint = `SPEED;                                  // optional

cy_ram_dp  U0(

  .data_a            ( a ),
  .data_b            ( b ),
  .address_a      ( adda ),
  .address_b      ( addb ),
  .q_a               ( a_out ),
  .q_b               ( b_out ),
  .addr_matchb ( match ),
  .wea               ( wena ),
  .web               ( wenb ),
  .inclock_a      ( clka ),                                        // optional
  .inclock_b      ( clkb ),                                        // optional
  .outclock_a    ( clka ),                                        // optional
  .outclock_b    ( clkb ),                                        // optional
  .outrega_ar    ( 1'b0 ),                                       // optional
  .outregb_ar    ( 1'b0 ));                                      // optional

    always@(match or wena or wenb)
       if(match == 1'b0)
          begin
            if(wena == 1'b1 && wenb == 1'b1)
               begin
                  inva <= 1'b0;
                  invb <= 1'b1;
              end
            else if(wena == 1'b1 && wenb == 1'b0)
              begin
                  inva <= 1'b0;
                  invb <= 1'b1;
              end
            else if(wena == 1'b0 && wenb == 1'b1)
              begin
                  inva <= 1'b1;
                  invb <= 1'b0;
              end
           else
              begin
                  inva <= 1'b0;
                  invb <= 1'b0;
             end
         end
      else
         begin
              inva <= 1'b0;
              invb <= 1'b0;
        end

 endmodule
```

Appendix D: Timing Diagrams and their Parameters

Channel Memory DP Asynchronous Timing

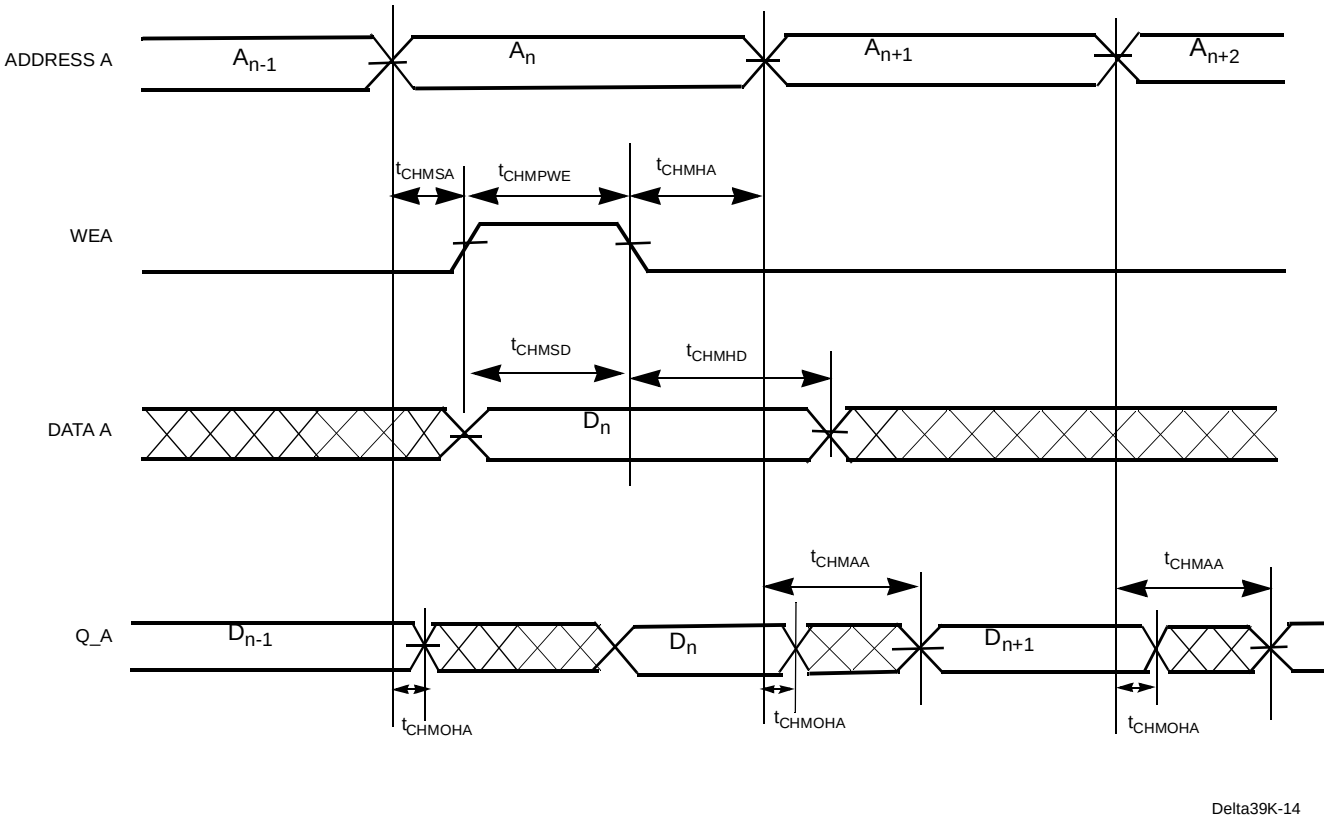
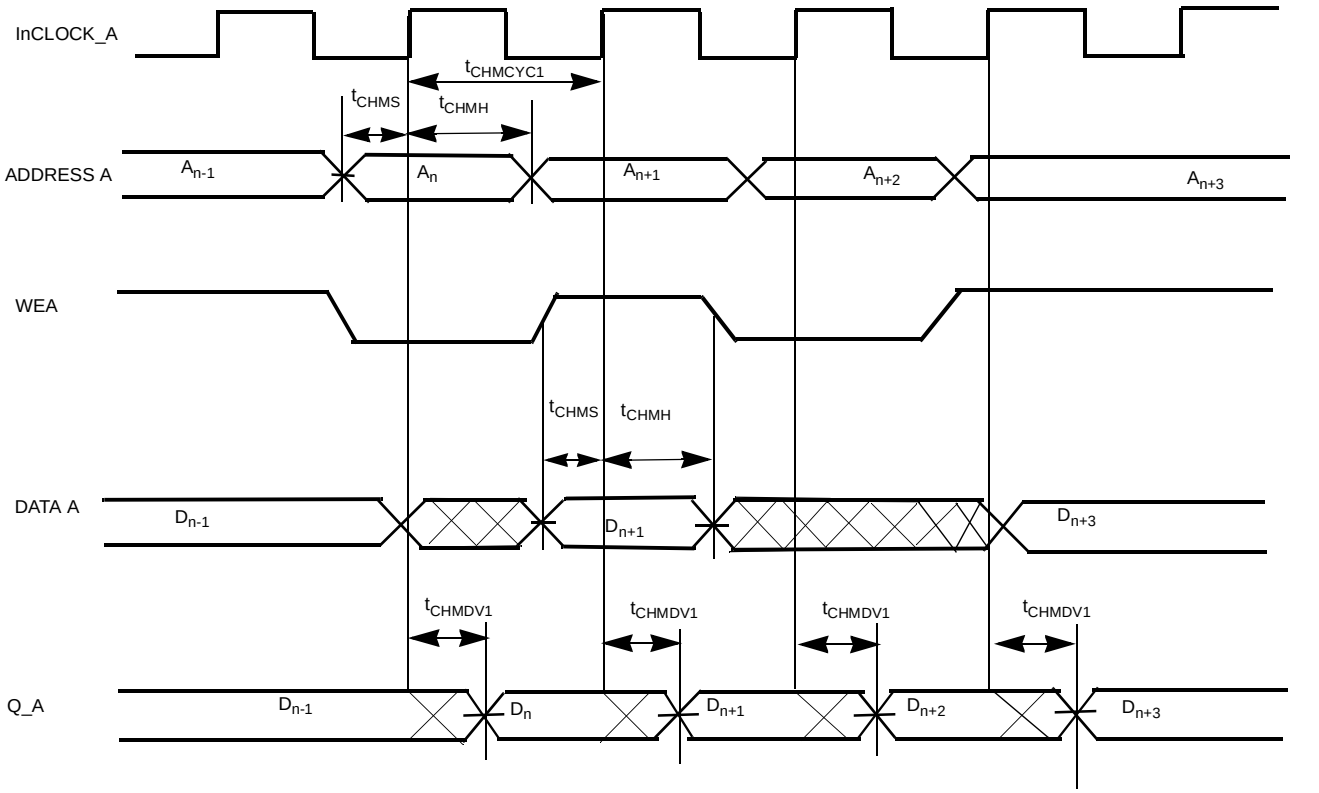


Figure 1. Dual-Port Asynchronous Timing Diagram and Parameters

Table 1. Dual-Port Asynchronous Mode Parameters

Parameter	Description
t_{CHMAA}	Channel memory access time. Delay from address to read data out
t_{CHMOHA}	Output hold from address change
t_{CHMPWE}	Write enable pulse width
t_{CHMSA}	Address set-up to the beginning of the write
t_{CHMHA}	Address hold to the end of the write
t_{CHMSD}	Data set-up to the end of the write
t_{CHMHD}	Data hold to the end of the write

Channel Memory DP SRAM Flow Through R/W Timing



Delta39K-17

Figure 2. Dual-Port Synchronous Flow through R/W Timing Diagram

Table 2. Dual-Port Synchronous Flow through Mode Parameters

Parameters	Description
$t_{CHMCYC1}$	Clock cycle time for flow through read and write operations
t_{CHMS}	Address, data, and WE set-up time fro pin inputs, relative to a global clock
t_{CHMH}	Address, data, and WE hold time of pin inputs, relative to a global clock
t_{CHMDV1}	Global clock to data valid on output pins for flow through data

Channel Memory DP SRAM Pipeline R/W Timing

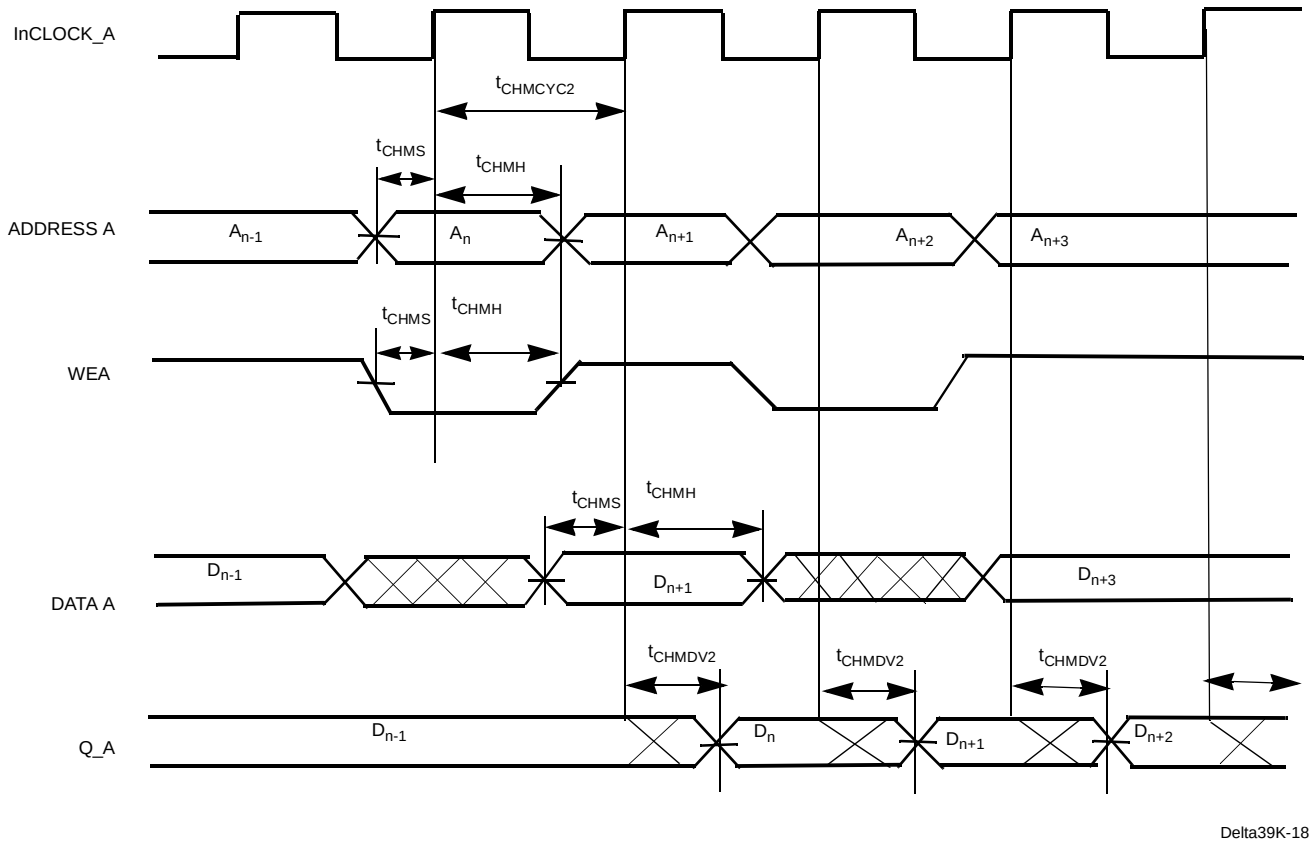


Figure 3. Dual-Port Synchronous Pipeline R/W Timing Diagram

Table 3. Dual-Port Synchronous Pipeline R/W Mode Parameters

Parameters	Description
$t_{CHMCYC2}$	Clock cycle time for pipelined read and write operations
t_{CHMS}	Address, data, and WE set-up time of pin inputs, relative to a global clock
t_{CHMH}	Address, data, and WE hold time of pin inputs, relative to a global clock
t_{CHMDV2}	Global clock to data valid on output pins for pipelined data

Dual-Port Asynchronous Address Match Busy Signal

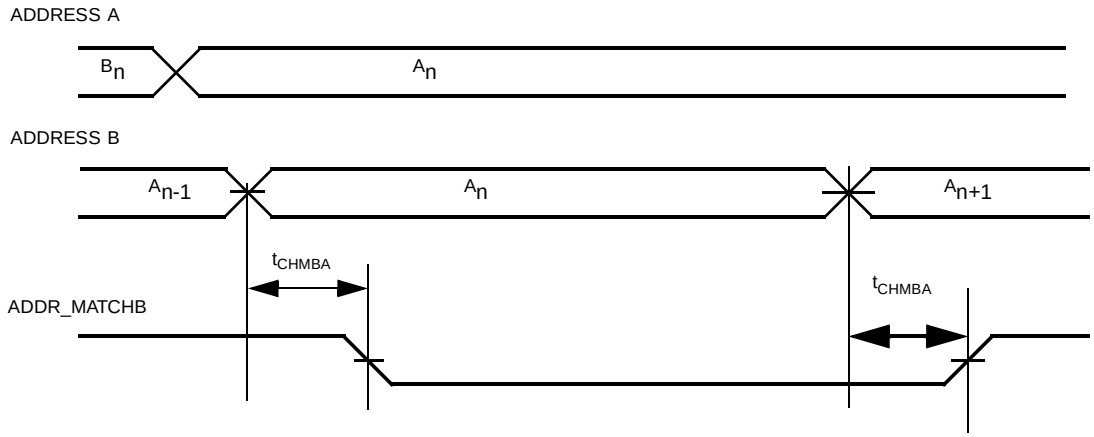


Figure 4. Dual-Port Asynchronous Address match Signal characteristics

Table 4. Dual-Port Asynchronous Address Match Busy Signal Parameters

Parameters	Description
t_{CHMBA}	Channel memory asynchronous Dual-Port address match (busy access time)

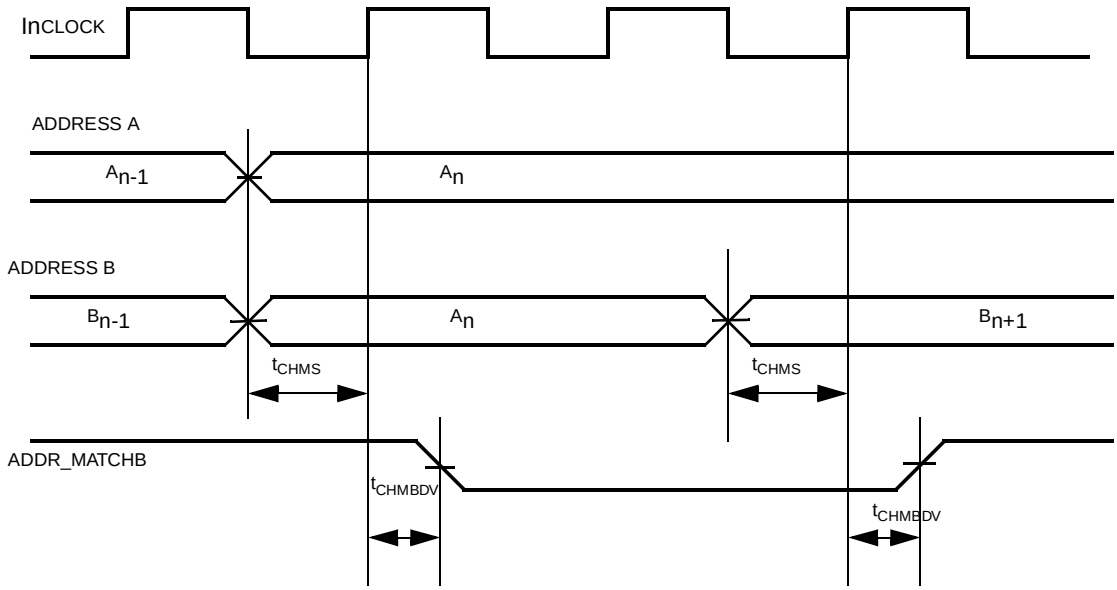


Figure 5. Dual-Port Synchronous Address Match Signal Characteristics

Table 5. Dual-Port Synchronous Address Match Busy Signal Parameters

Parameters	Description
t_{CHMBDV}	Channel memory synchronous Dual-Port address match (busy, clock to data valid)
t_{CHMS}	Address, data, and WE set-up time of pin inputs, relative to a global clock