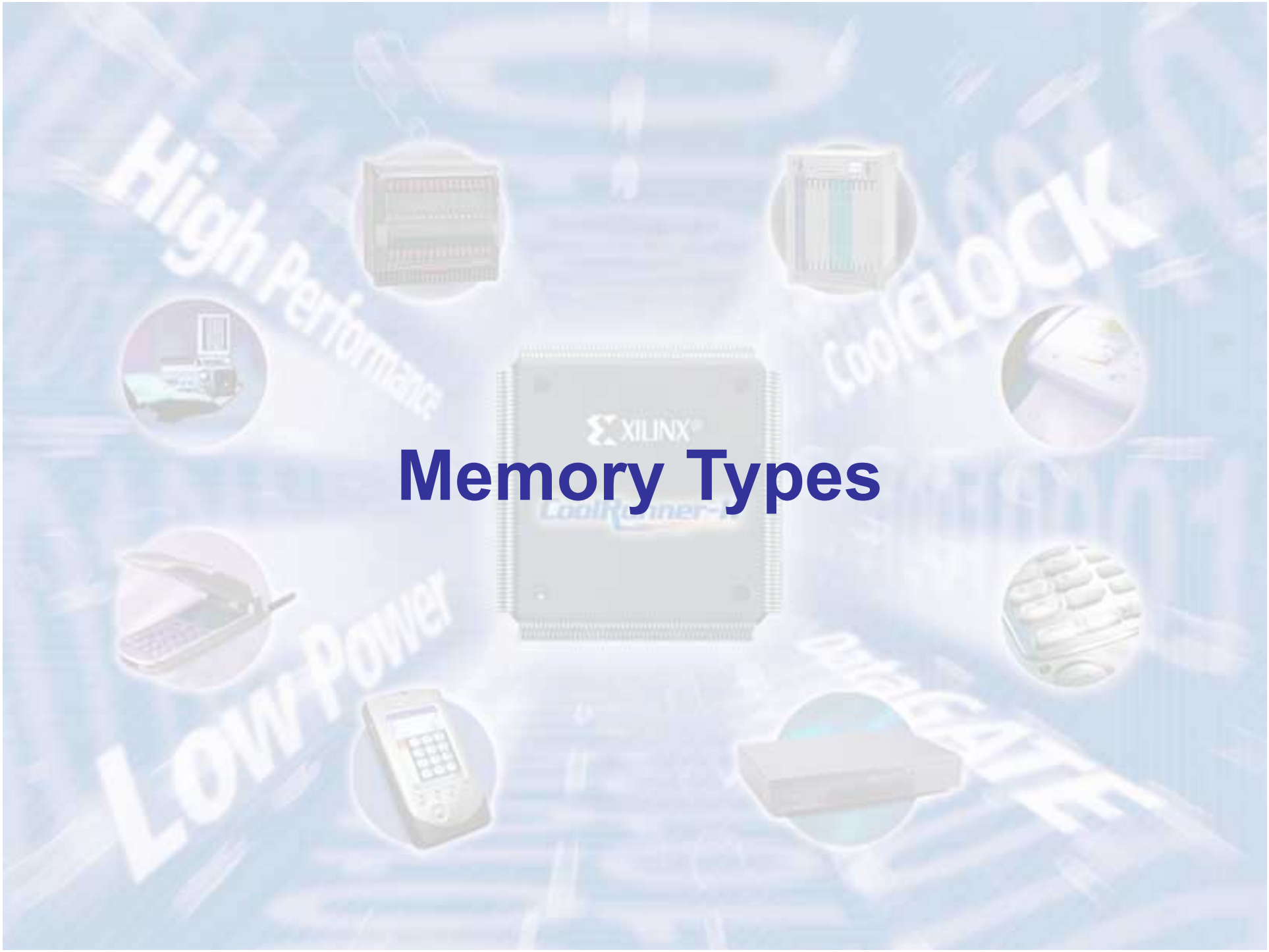
ECE 545
Lecture 17



RAM

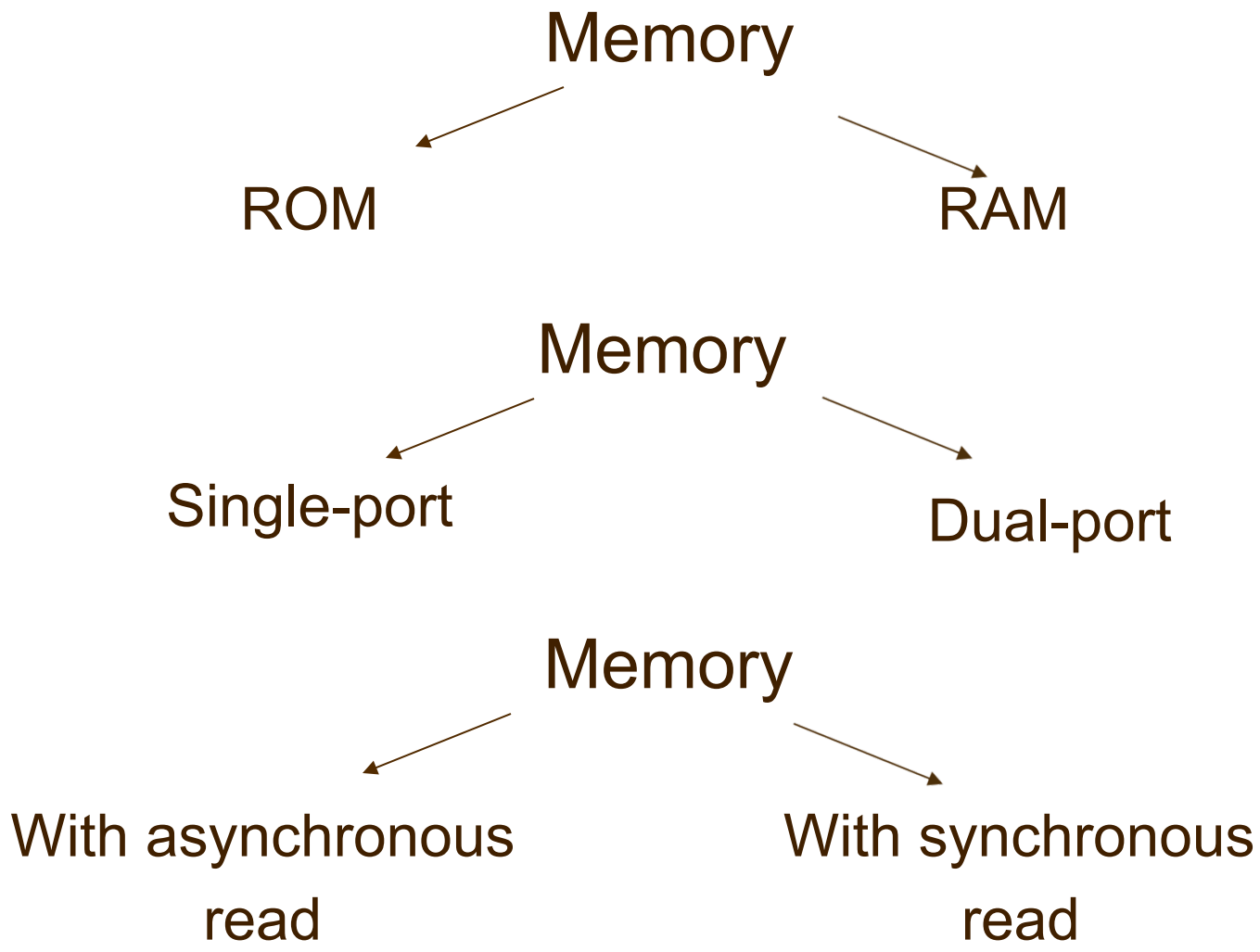
Recommended reading

- XST User Guide for Virtex-6, Spartan-6, and 7 Series Devices
  Chapter 7, HDL Coding Techniques
  [UG687 (v 14.5) March 20, 2013]
  Sections:
    - RAM HDL Coding Techniques
    - ROM HDL Coding Techniques
- ISE In-Depth Tutorial, Section: Creating a CORE Generator Tool Module
  [UG695 (v14.1) April 24, 2012]

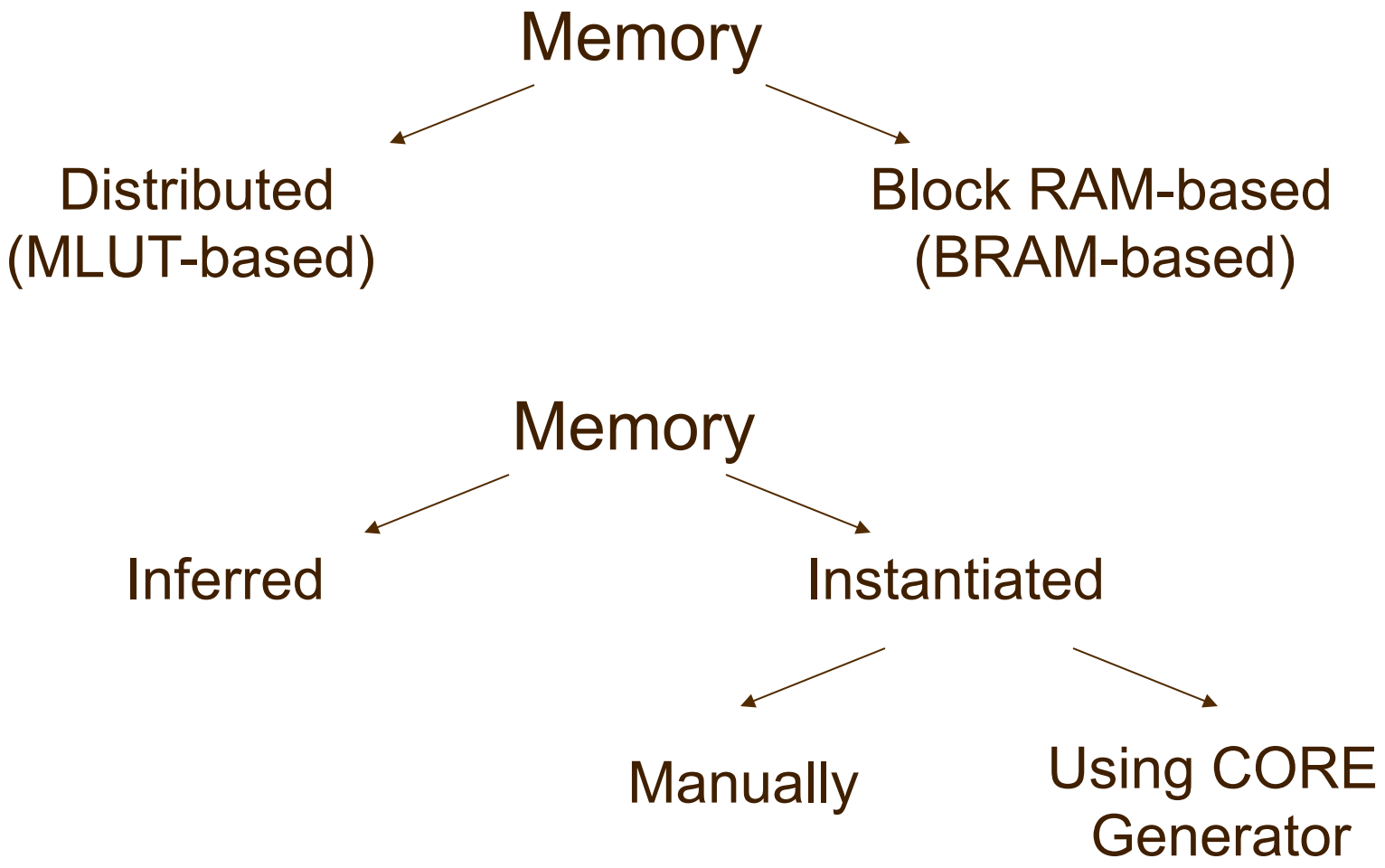


Memory Types

Memory Types



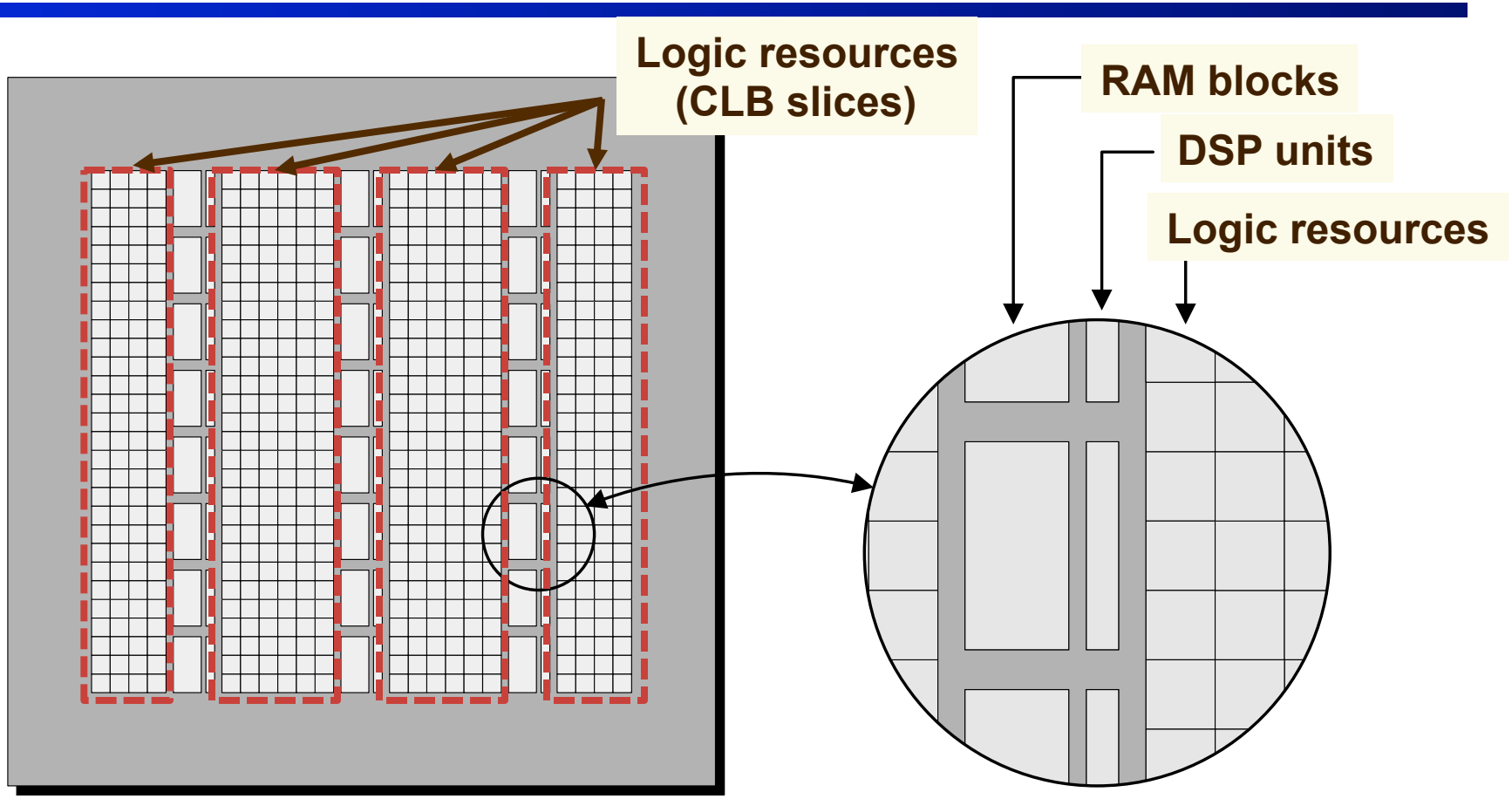
Memory Types specific to Xilinx FPGAs





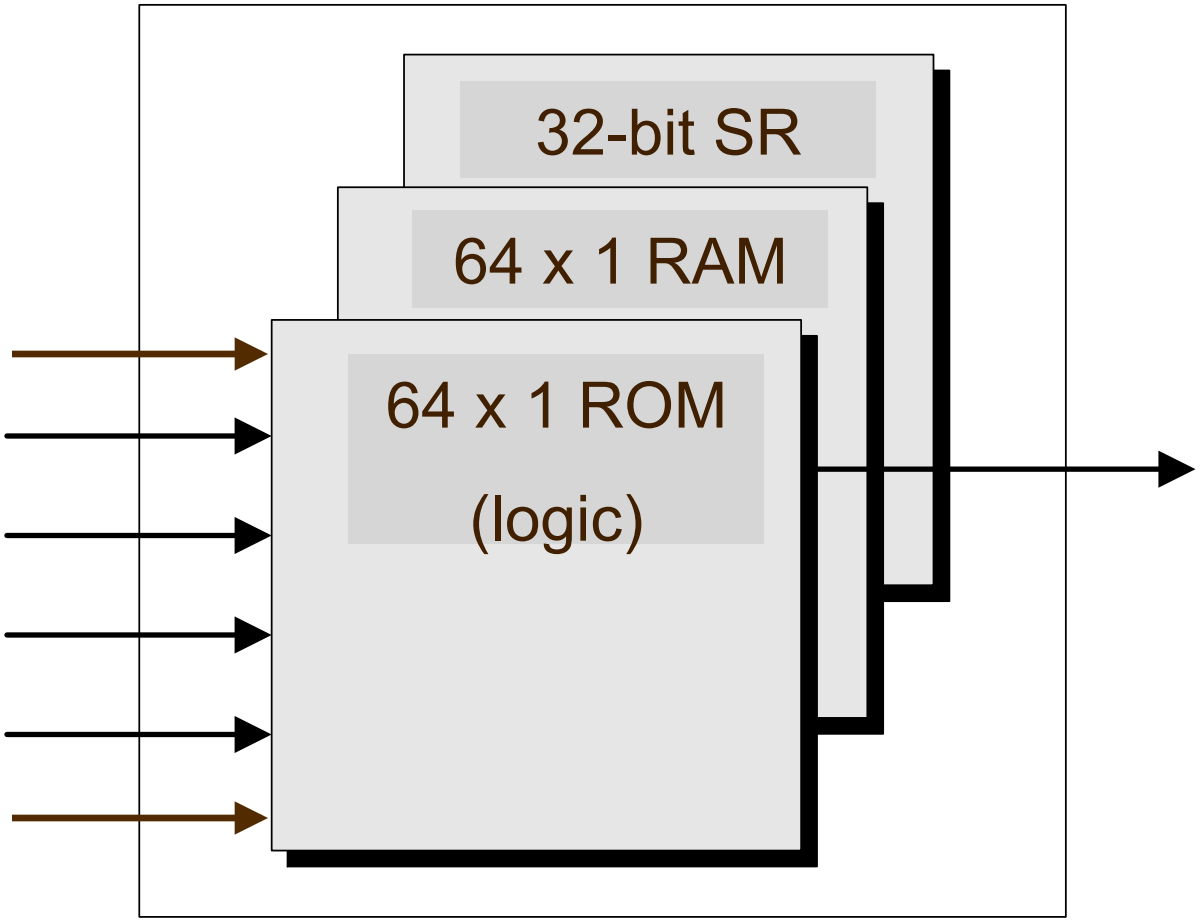
FPGA Distributed RAM

Location of Distributed RAM



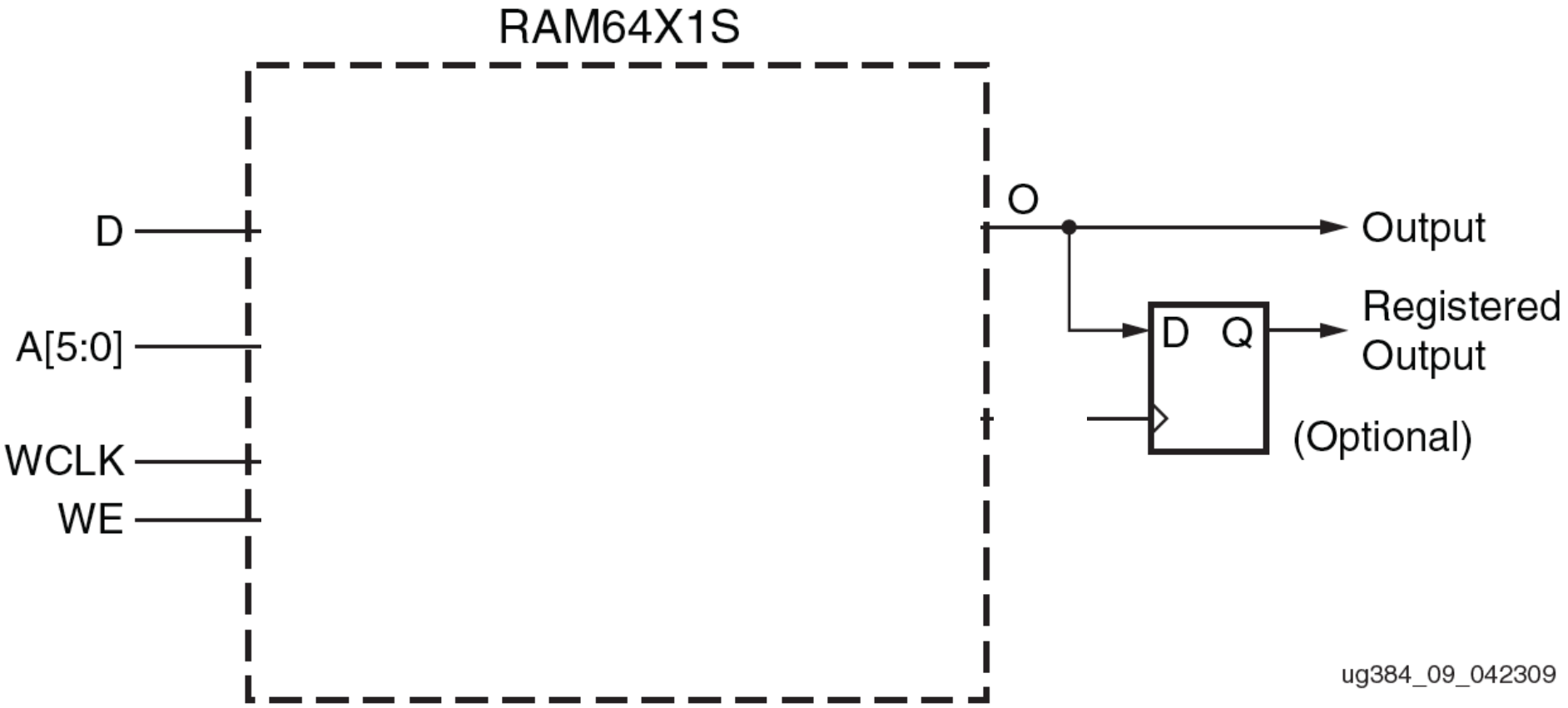
Graphics based on The Design Warrior's Guide to FPGAs
Devices, Tools, and Flows. ISBN 0750676043
Copyright © 2004 Mentor Graphics Corp. (www.mentor.com)

Spartan-6 Multipurpose LUT (MLUT)

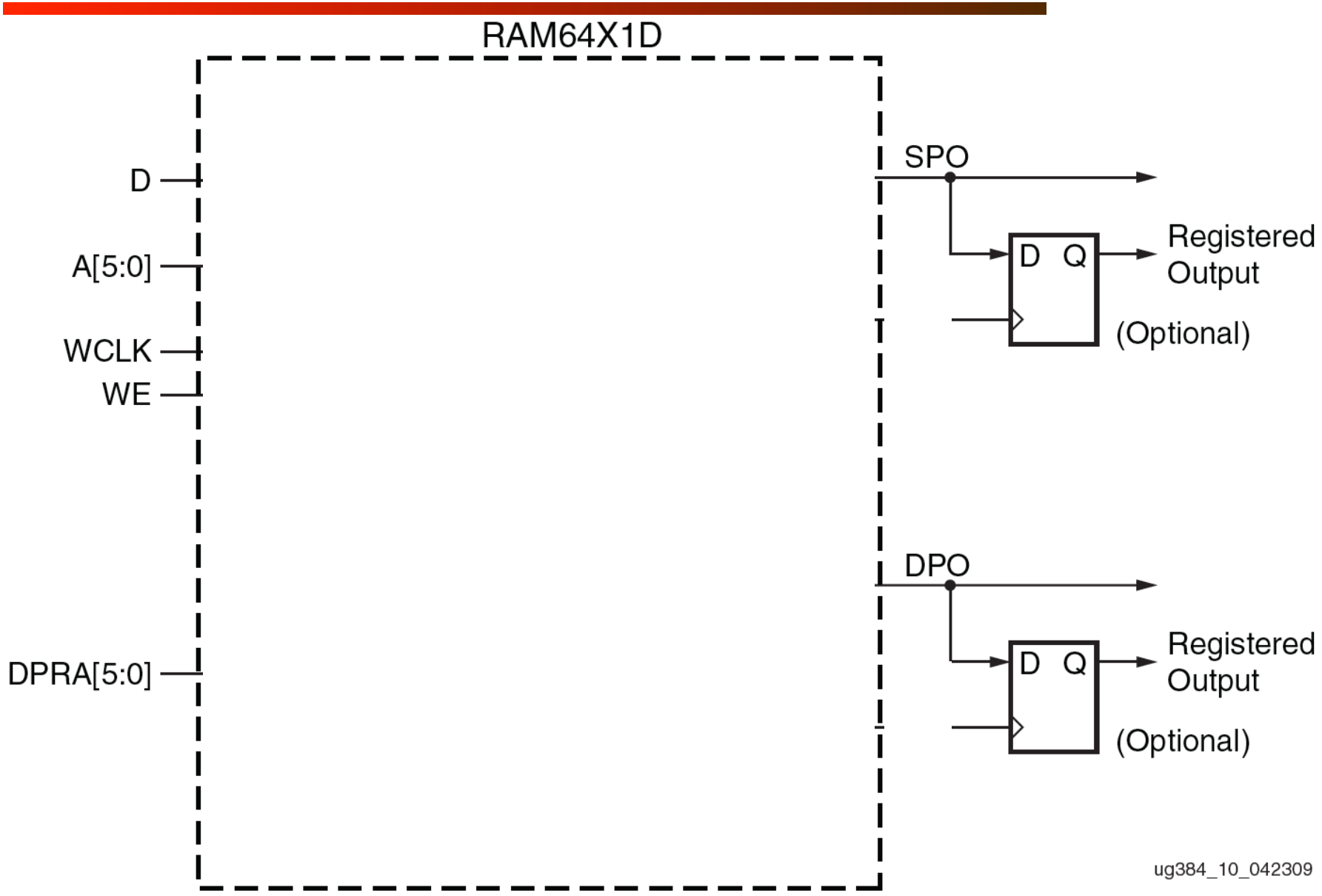


The Design Warrior's Guide to FPGAs
Devices, Tools, and Flows. ISBN 0750676043
Copyright © 2004 Mentor Graphics Corp. (www.mentor.com)

Single-port 64 x 1-bit RAM (1 MLUT)



Dual-port 64 x 1 RAM (2 MLUTs)





Generic Inferred Distributed RAM

Distributed single-port RAM with asynchronous read

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

entity raminfr is
  generic ( w : integer := 32;
            -- number of bits per RAM word
            r : integer := 6);
            -- 2^r = number of words in RAM
  port (clk : in std_logic;
        we  : in std_logic;
        a   : in std_logic_vector(r-1 downto 0);
        di  : in std_logic_vector(w-1 downto 0);
        do  : out std_logic_vector(w-1 downto 0));
end raminfr;
```

Distributed single-port RAM with asynchronous read

```
architecture behavioral of raminfr is
  type ram_type is array (0 to 2**r-1)
        of std_logic_vector (w-1 downto 0);
  signal RAM : ram_type := (others => (others => '0'));
begin
  process (clk)
  begin
    if rising_edge(clk) then
      if (we = '1') then
        RAM(to_integer(unsigned(a))) <= di;
      end if;
    end if;
  end process;
  do <= RAM(to_integer(unsigned(a)));
end behavioral;
```

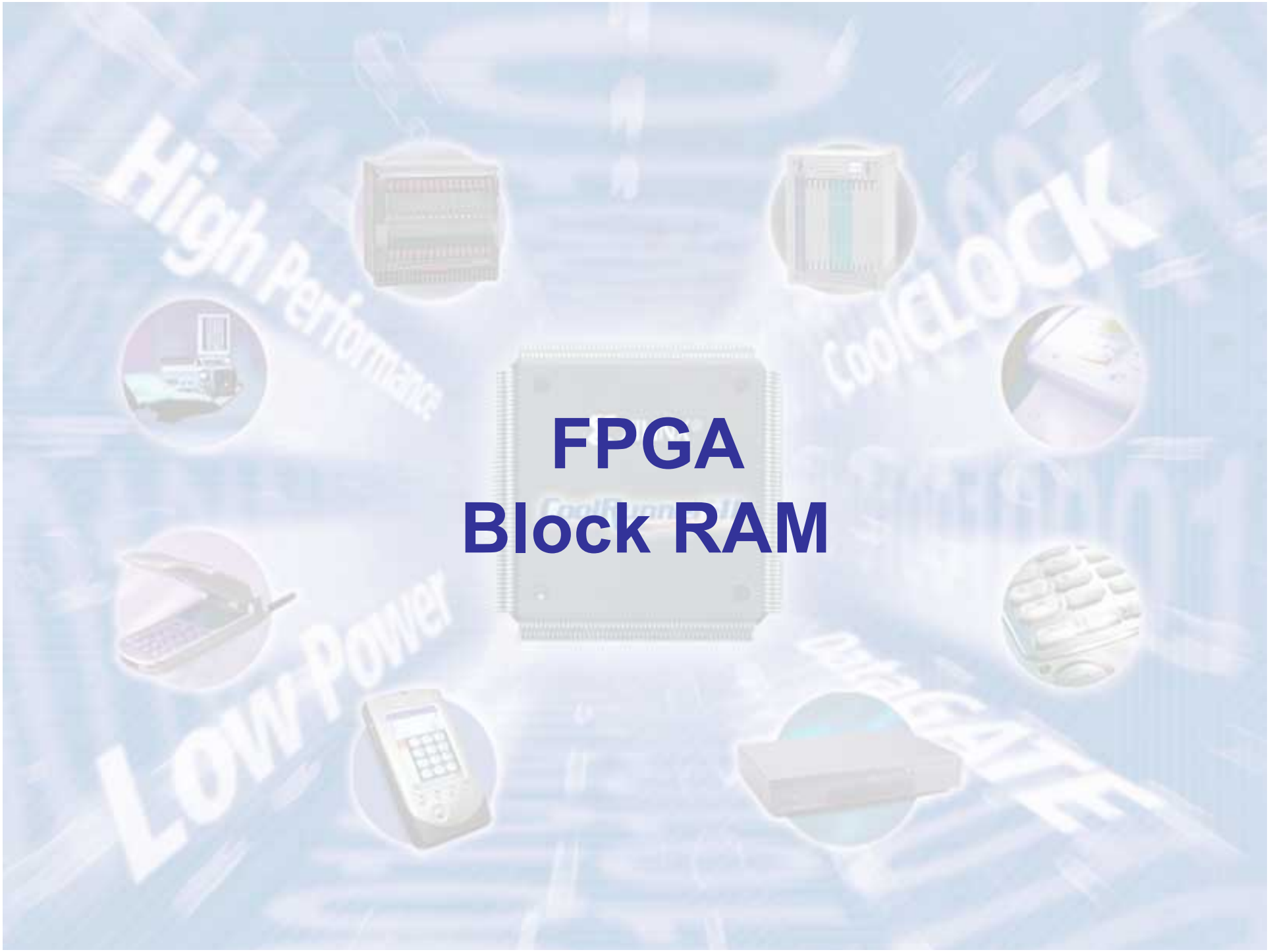
Distributed dual-port RAM with asynchronous read

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity raminfr is
 generic ( w : integer := 32;
             -- number of bits per RAM word
           r : integer := 6);
             -- 2^r = number of words in RAM
  port (clk  : in std_logic;
        we   : in std_logic;
        a    : in std_logic_vector(r-1 downto 0);
        dpra : in std_logic_vector(r-1 downto 0);
        di   : in std_logic_vector(w-1 downto 0);
        spo  : out std_logic_vector(w-1 downto 0);
        dpo  : out std_logic_vector(w-1 downto 0));
end raminfr;
```

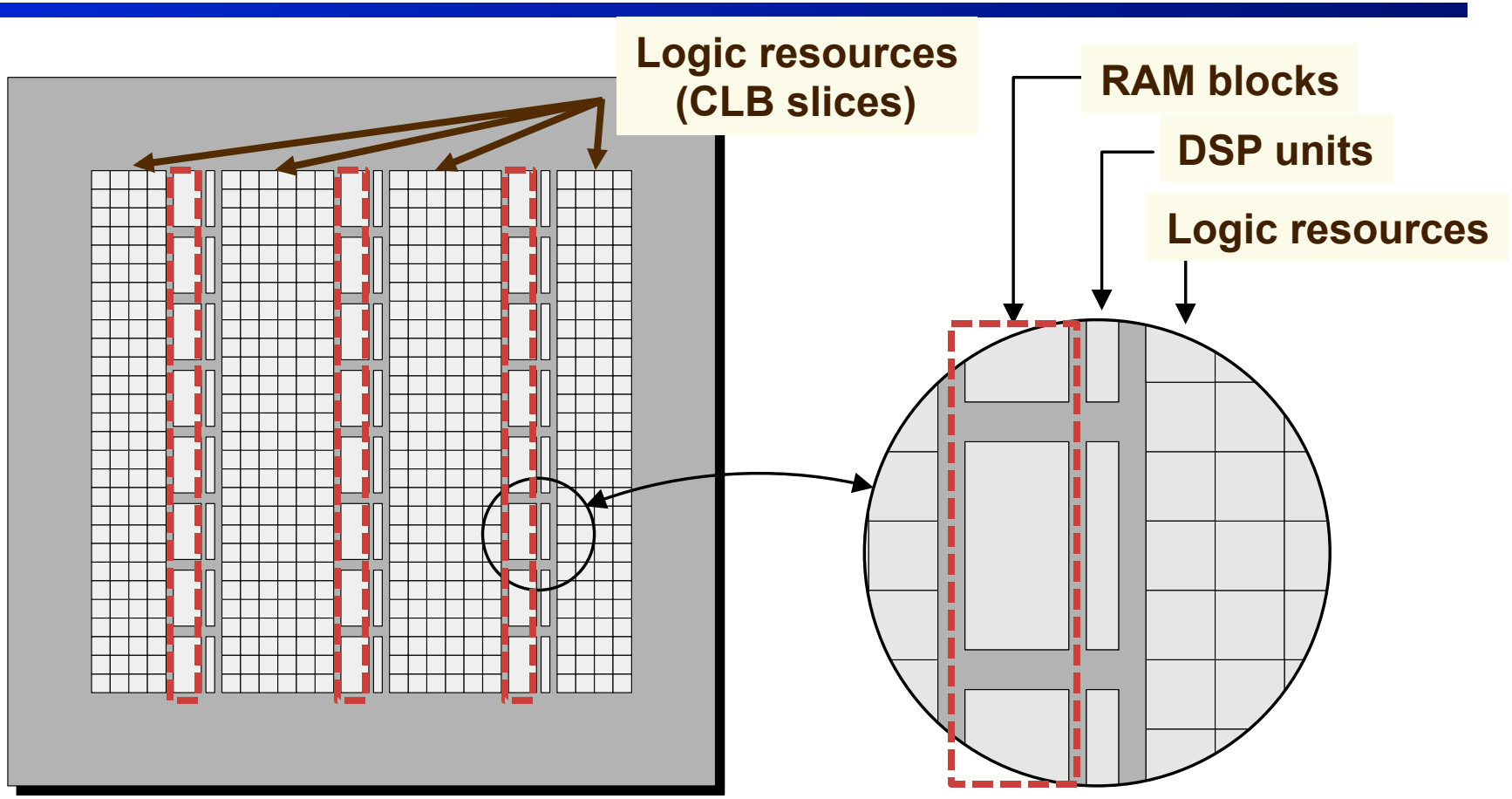
Distributed dual-port RAM with asynchronous read

```
architecture syn of raminfr is
  type ram_type is array (0 to 2**r-1) of
            std_logic_vector (w-1 downto 0);
  signal RAM : ram_type := (others => (others => '0'));
begin
  process (clk)
  begin
    if rising_edge(clk) then
      if (we = '1') then
        RAM(to_integer(unsigned(a))) <= di;
      end if;
    end if;
  end process;
  spo <= RAM(to_integer(unsigned(a)));
  dpo <= RAM(to_integer(unsigned(dpra)));
 end syn;
```



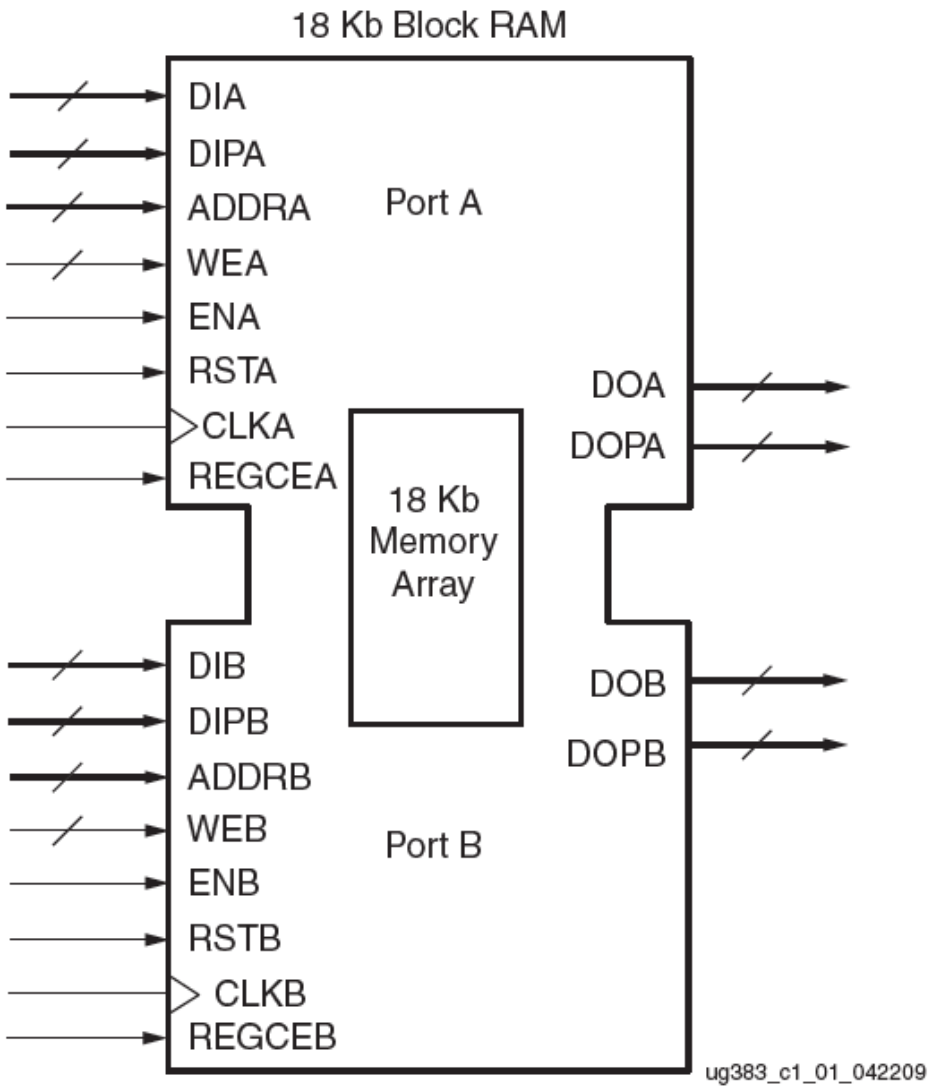
FPGA
Block RAM

Location of Block RAMs



Graphics based on The Design Warrior's Guide to FPGAs
Devices, Tools, and Flows. ISBN 0750676043
Copyright © 2004 Mentor Graphics Corp. (www.mentor.com)

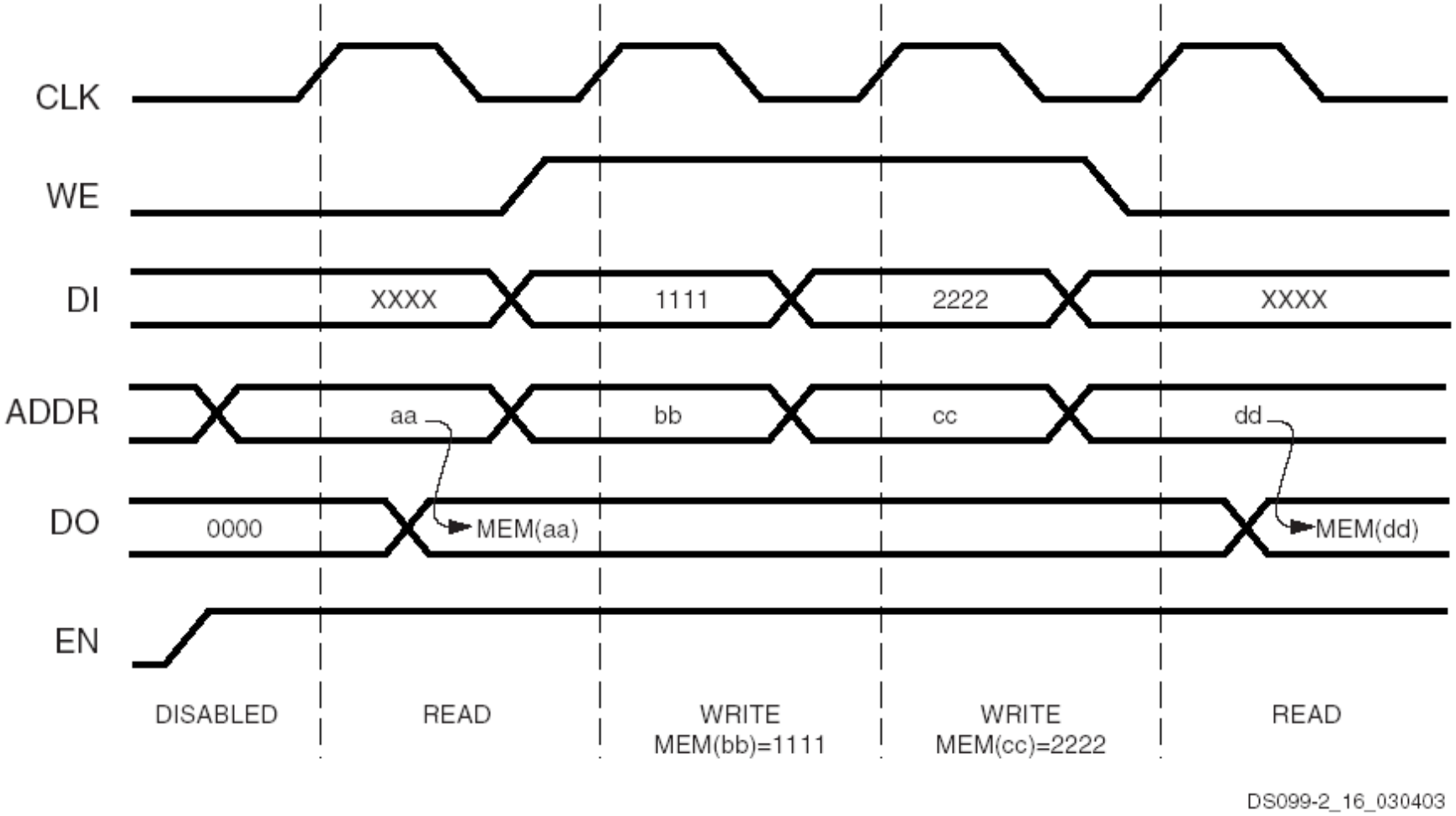
Block RAM Interface



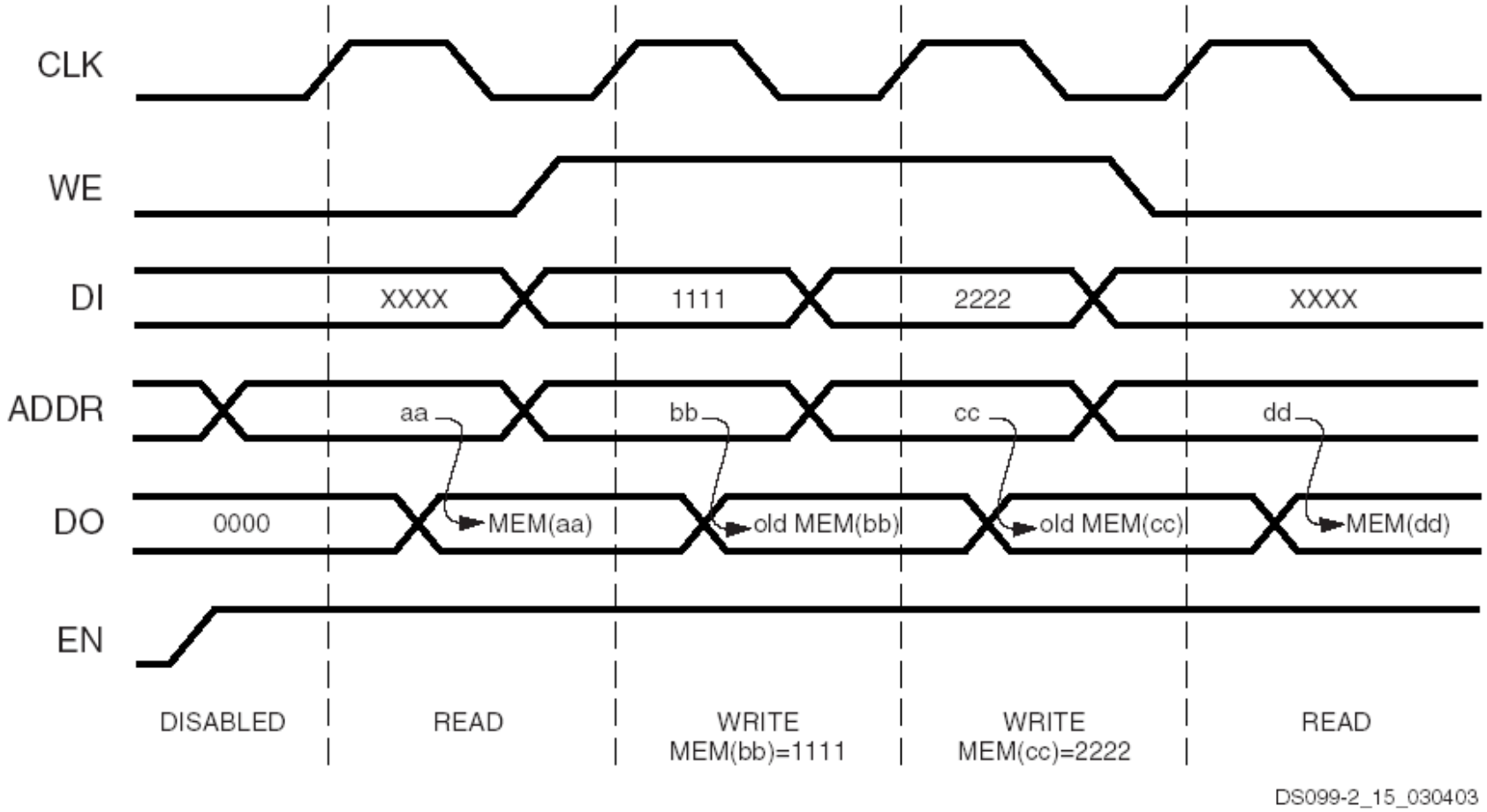
Block RAM Ports

Port Name	Description
DI[A | B](1)	Data Input Bus
DIP[A | B](1)	Data Input Parity Bus
ADDR[A | B]	Address Bus
WE[A | B]	Byte-wide Write Enable
EN[A | B]	When inactive no data is written to the block RAM and the output bus remains in its previous state.
RST[A | B]	Synchronous Set/Reset the output registers (DO_REG = 1).
CLK[A | B]	Clock Input
DO[A | B](1)	Data Output Bus
DOP[A | B](1)	Data Output Parity Bus
REGCE[A | B]	Output Register Clock Enable

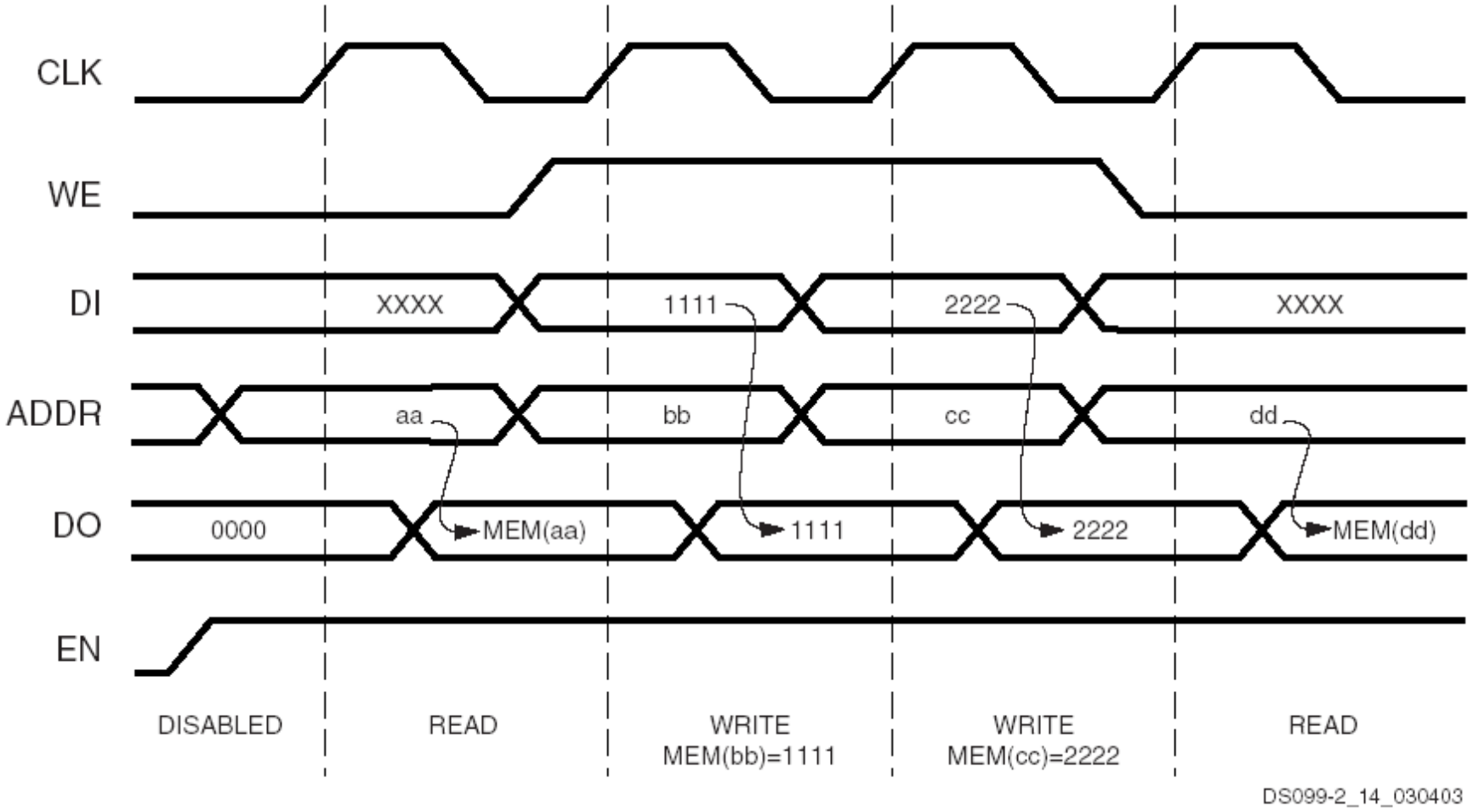
Block RAM Waveforms – NO_CHANGE mode

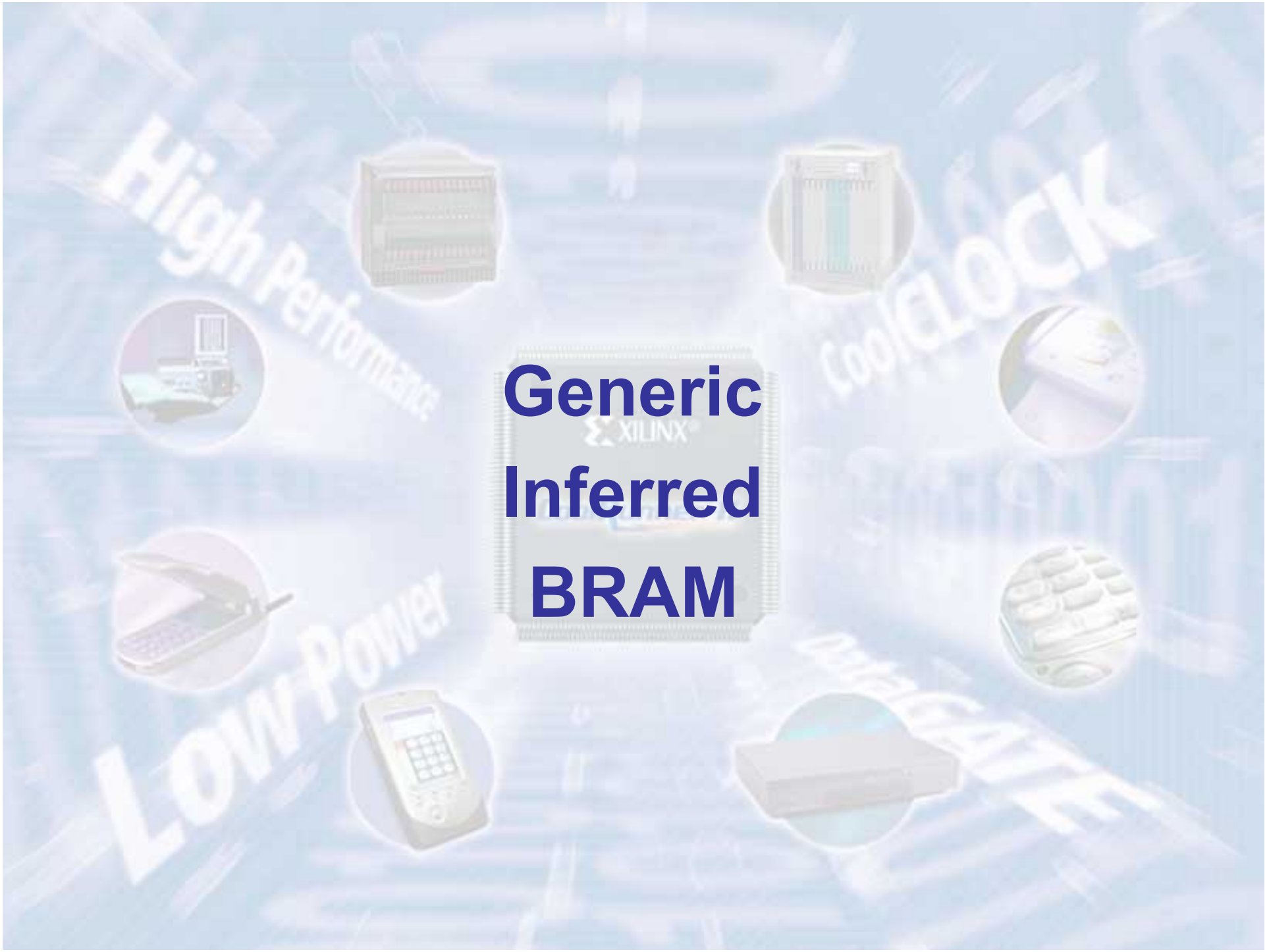


Block RAM Waveforms – READ_FIRST mode



Block RAM Waveforms – WRITE_FIRST mode



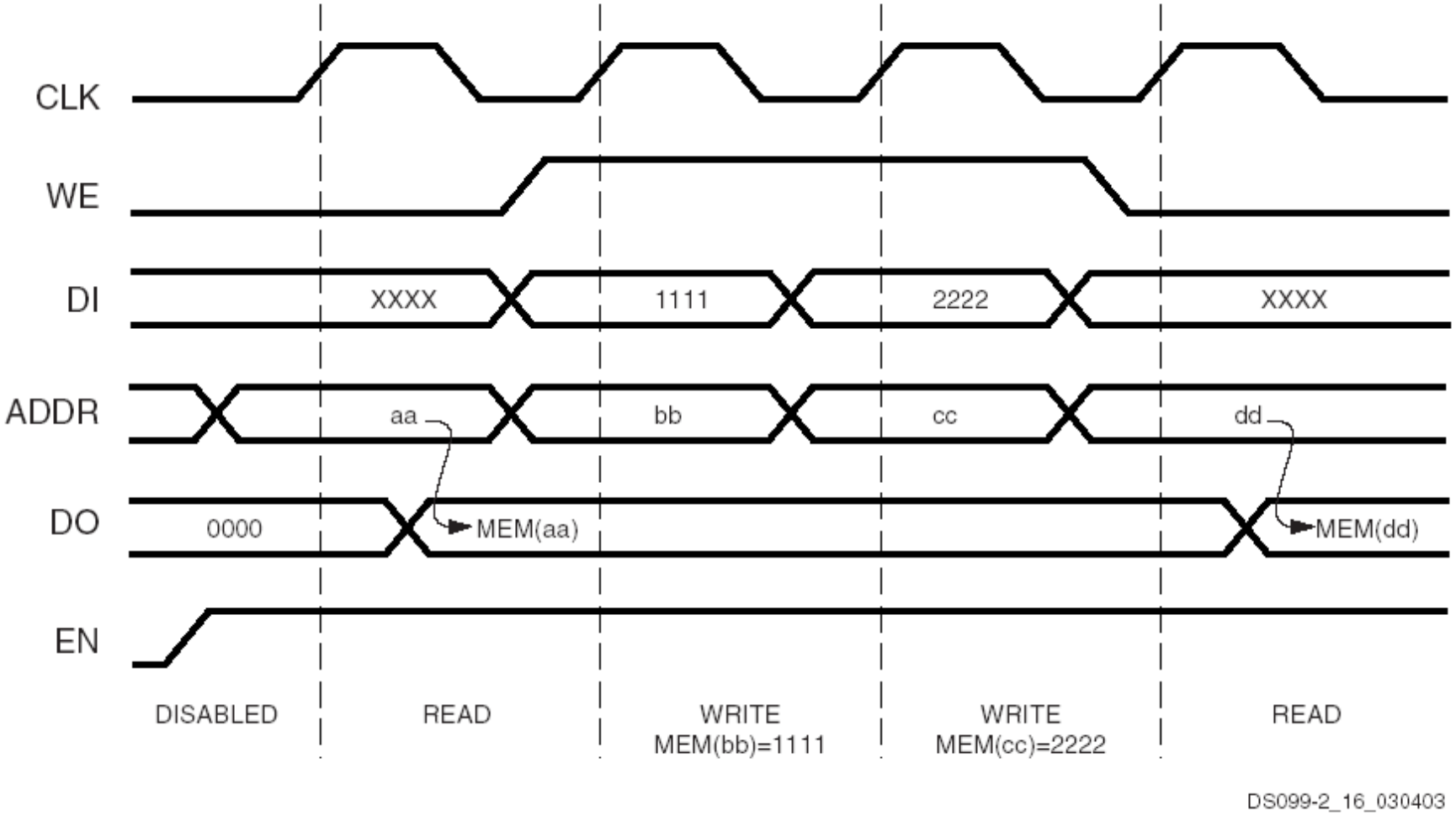

High Performance
Low Power
Generic
Inferred
BRAM

Single-port Block RAM with synchronous read

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

entity raminfr is
  generic ( w : integer := 32;
              -- number of bits per RAM word
            r : integer := 9);
              -- 2^r = number of words in RAM
  port (clk : in std_logic;
        we  : in std_logic;
        en  : in std_logic;
        addr   : in std_logic_vector(r-1 downto 0);
        di  : in std_logic_vector(w-1 downto 0);
        do  : out std_logic_vector(w-1 downto 0));
end raminfr;
```

Block RAM Waveforms – NO_CHANGE mode

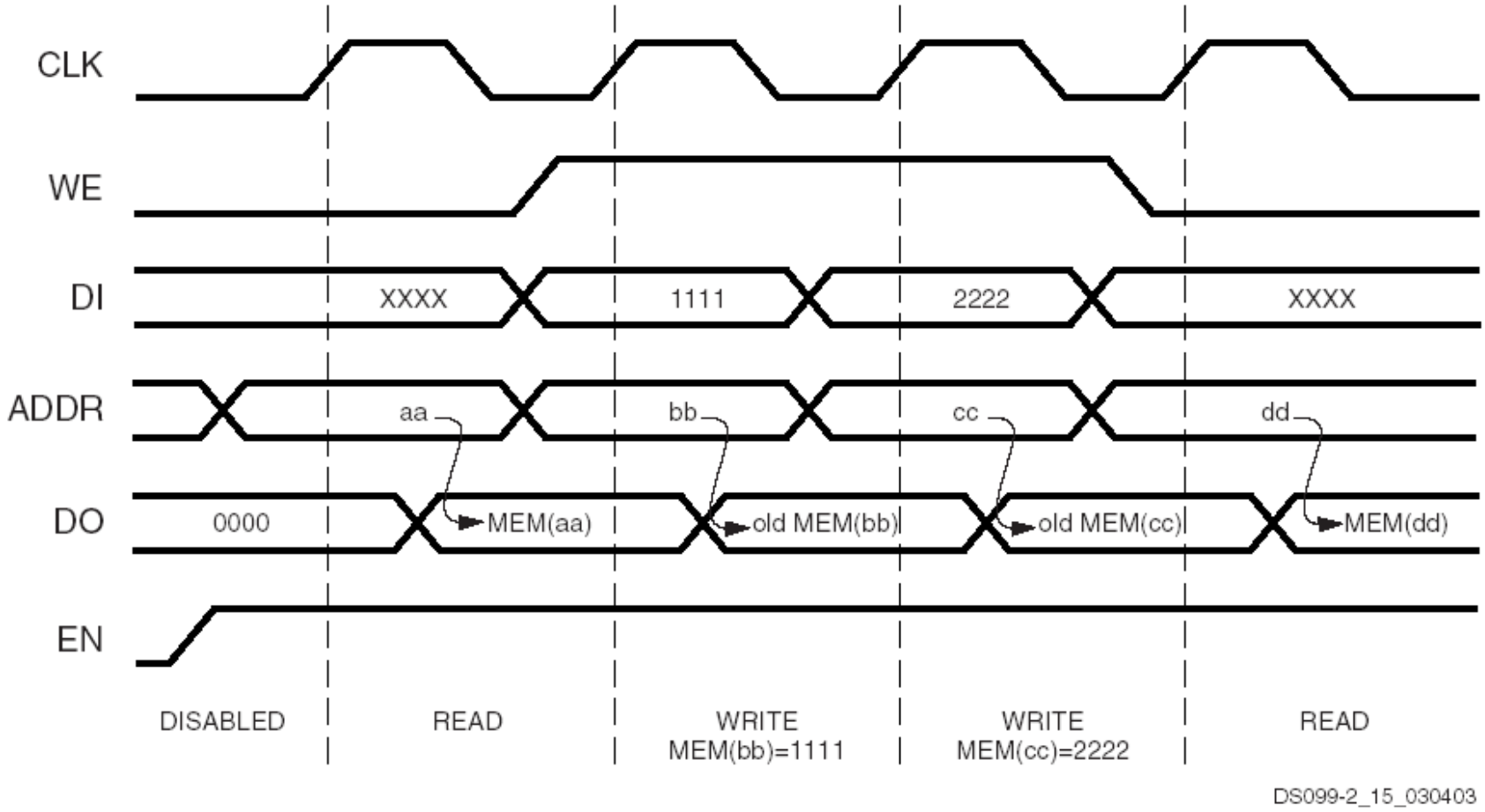


Single-port Block RAM with synchronous read No-Change Mode - cont'd

```
architecture behavioral of raminfr is
  type ram_type is array (0 to 2**r-1) of
                        std_logic_vector (w-1 downto 0);
  signal RAM : ram_type := (others => (others => '0'));

begin
  process (clk)
  begin
    if (clk'event and clk = '1') then
      if (en = '1') then
        if (we = '1') then
           RAM(to_integer(unsigned(addr))) <= di;
        else
           do <= RAM(to_integer(unsigned(addr)));
        end if;
      end if;
    end if;
  end process;
end behavioral;
```

Block RAM Waveforms – READ_FIRST mode

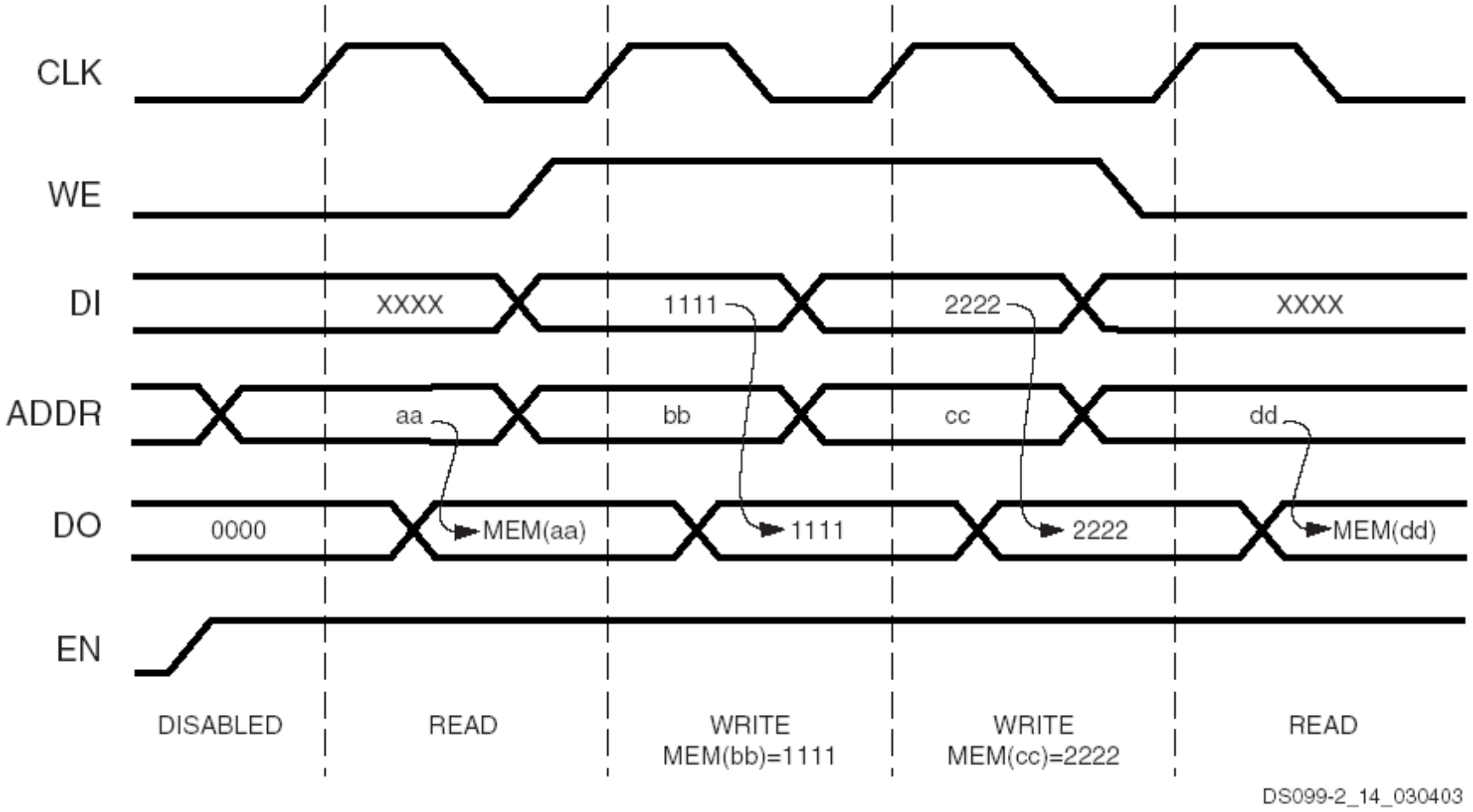


Single-port Block RAM with synchronous read Read-First Mode - cont'd

```
architecture behavioral of raminfr is
  type ram_type is array (0 to 2**r-1) of
                        std_logic_vector (w-1 downto 0);
  signal RAM : ram_type := (others => (others => '0'));

begin
  process (clk)
  begin
    if rising_edge(clk) then
      if (en = '1') then
        do <= RAM(to_integer(unsigned(addr)));
        if (we = '1') then
           RAM(to_integer(unsigned(addr))) <= di;
        end if;
      end if;
    end if;
  end process;
end behavioral;
```

Block RAM Waveforms – WRITE_FIRST mode



Single-port Block RAM with synchronous read Write-First Mode - cont'd

```
architecture behavioral of raminfr is
  type ram_type is array (0 to 2**r-1) of
                        std_logic_vector (w-1 downto 0);
  signal RAM : ram_type := (others => (others => '0'));

begin
  process (clk)
  begin
    if (clk'event and clk = '1') then
      if (en = '1') then
        if (we = '1') then
           RAM(to_integer(unsigned(addr))) <= di;
           do <= di;
        else
           do <= RAM(to_integer(unsigned(addr)));
        end if;
      end if;
    end if;
  end process;
end behavioral;
```

Simple Dual-port Block RAM with one port used for reading and one port used for writing

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

entity raminfr is
  generic ( w : integer := 32;
              -- number of bits per RAM word
            r : integer := 9);
              -- 2^r = number of words in RAM
  port (clk : in std_logic;
        we  : in std_logic;
        en  : in std_logic;
        addr_rd   : in std_logic_vector(r-1 downto 0);
        addr_wr   : in std_logic_vector(r-1 downto 0);
        di  : in std_logic_vector(w-1 downto 0);
        do  : out std_logic_vector(w-1 downto 0));
end raminfr;
```

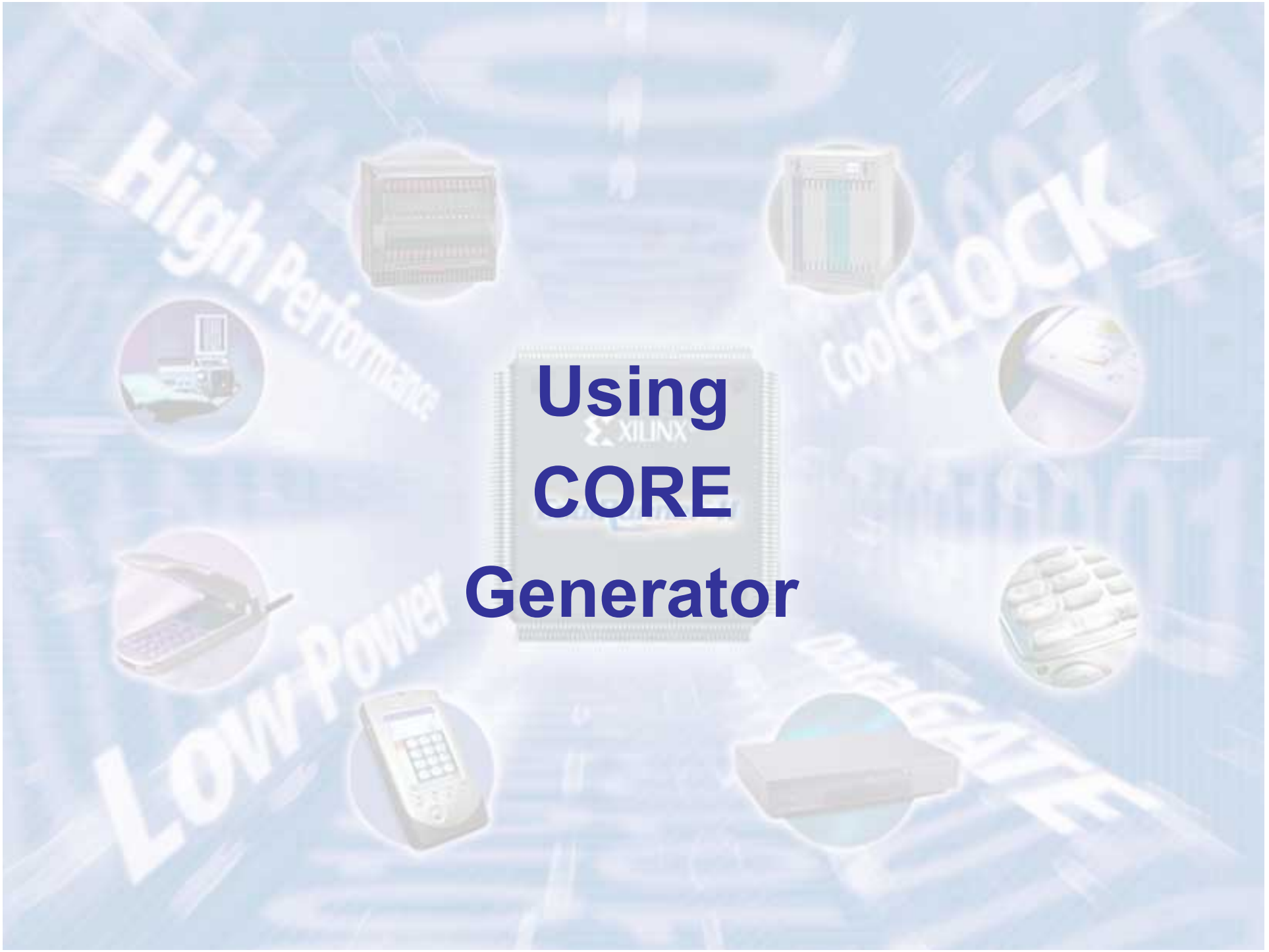
Simple Dual-port Block RAM with one port used for reading and one port used for writing

```
architecture behavioral of raminfr is
  type ram_type is array (0 to 2**r-1) of
                        std_logic_vector (w-1 downto 0);
  signal RAM : ram_type := (others => (others => '0'));

begin
  process (clk)
  begin
    if (clk'event and clk = '1') then
      if (en = '1') then
        if (we = '1') then
           RAM(to_integer(unsigned(addr_wr))) <= di;
        end if;
        do <= RAM(to_integer(unsigned(addr_rd)));
      end if;
    end if;
  end process;
end behavioral;
```

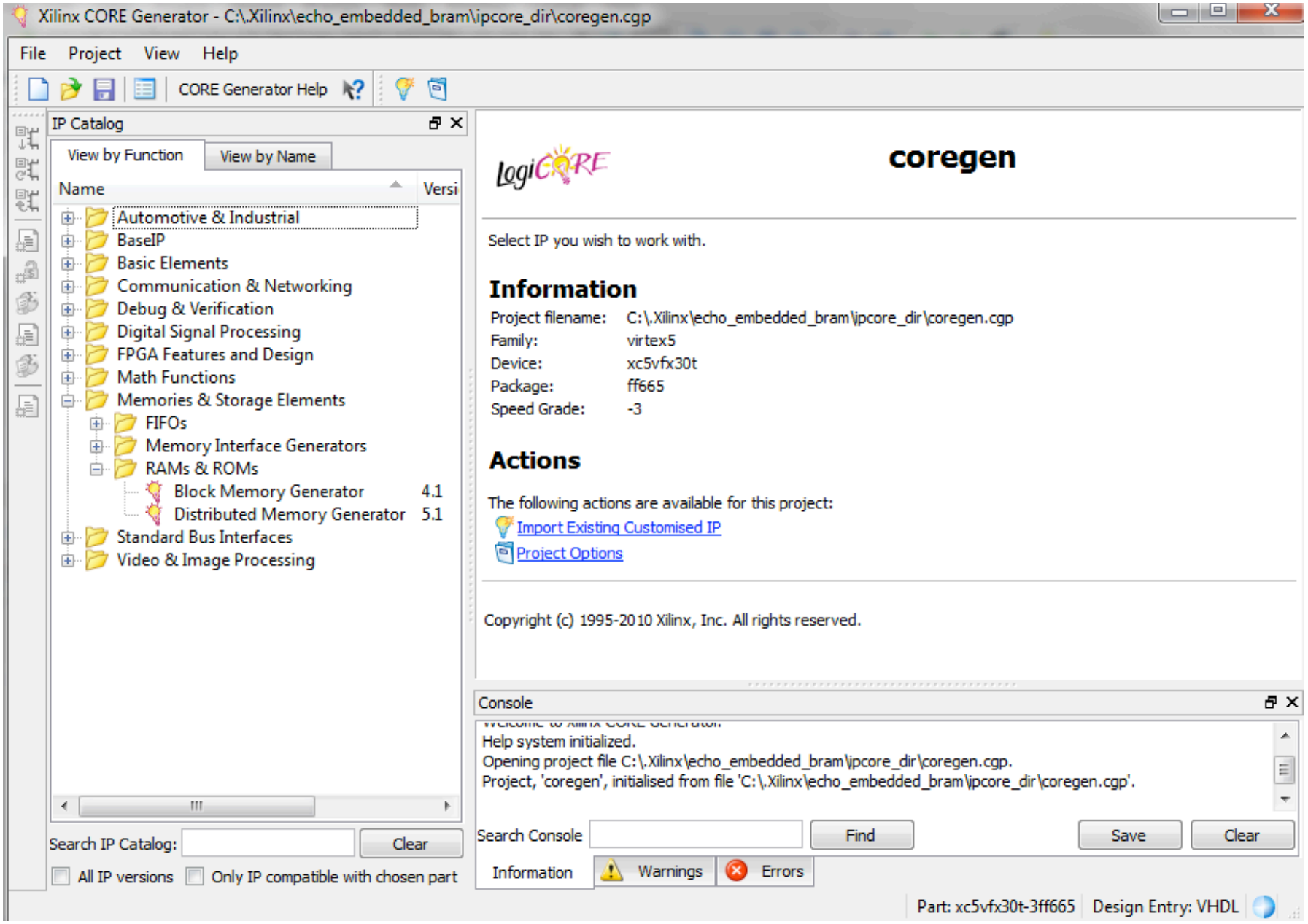
Criteria for Implementing Inferred RAM in BRAMs

Devices	Depth	Depth * Width
Spartan®-6	>= 127 words	> 512 bits
Virtex®-6	>= 127 words	> 512 bits
7 series	>= 127 words	> 512 bits



Using CORE Generator

CORE Generator



CORE Generator

