Lecture 11

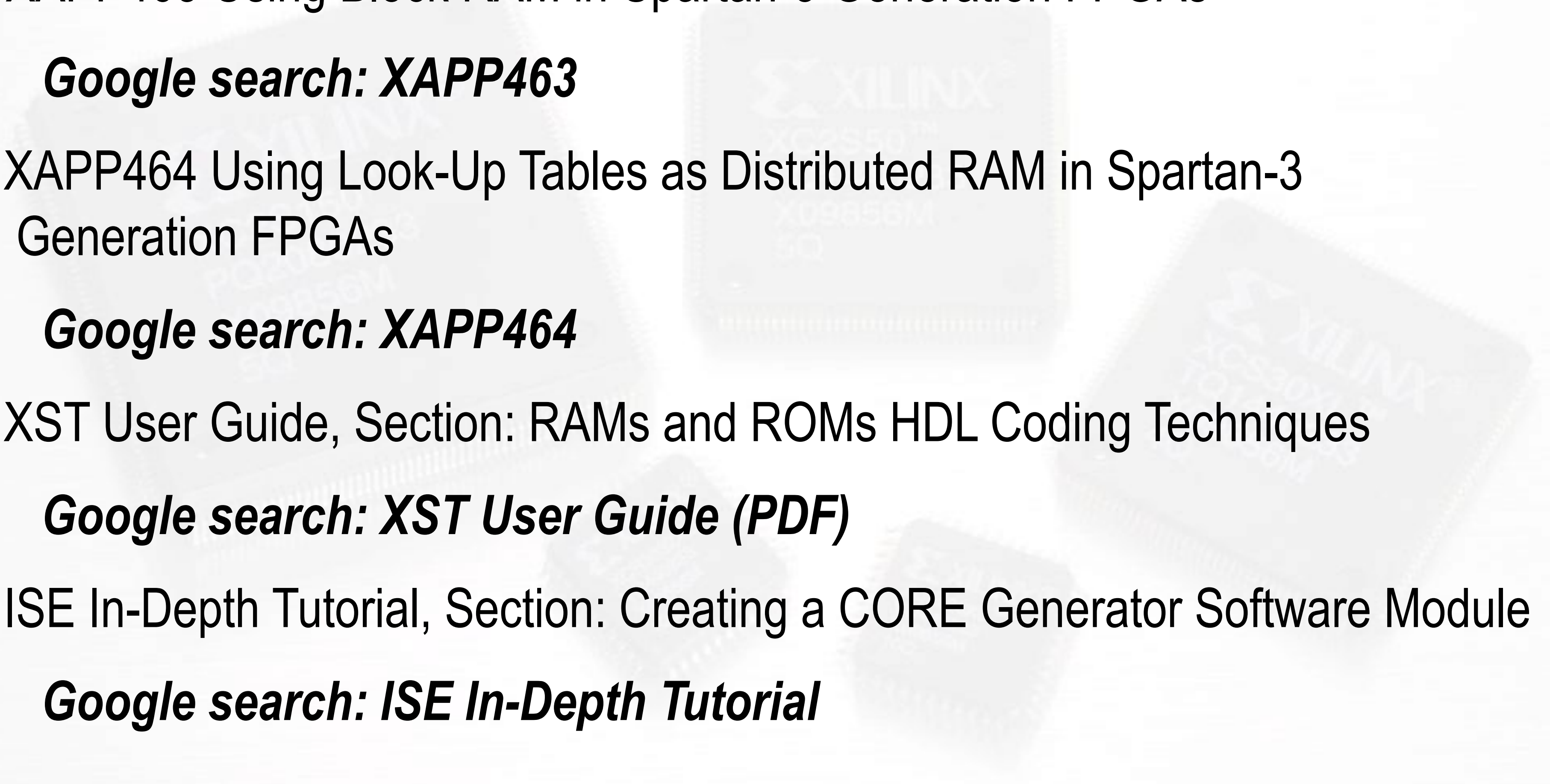
Xilinx FPGA Memories

Required reading

- P. Chu, *FPGA Prototyping by VHDL Examples*

  Chapter 11, Xilinx Spartan-3 Specific Memory

Recommended reading



- XAPP463 Using Block RAM in Spartan-3 Generation FPGAs

  Google search: XAPP463

- XAPP464 Using Look-Up Tables as Distributed RAM in Spartan-3 Generation FPGAs

  Google search: XAPP464

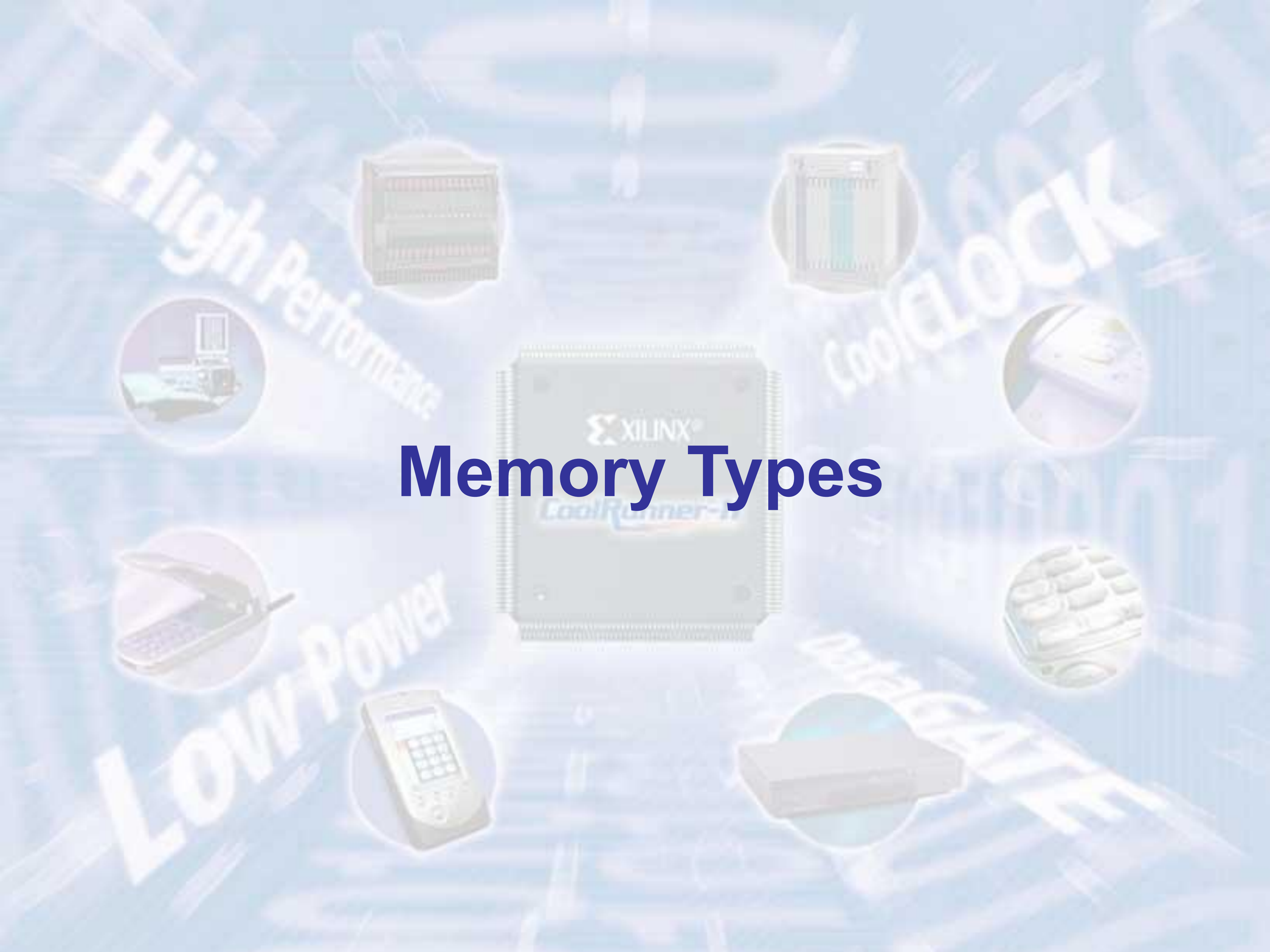
- XST User Guide, Section: RAMs and ROMs HDL Coding Techniques

  Google search: XST User Guide (PDF)

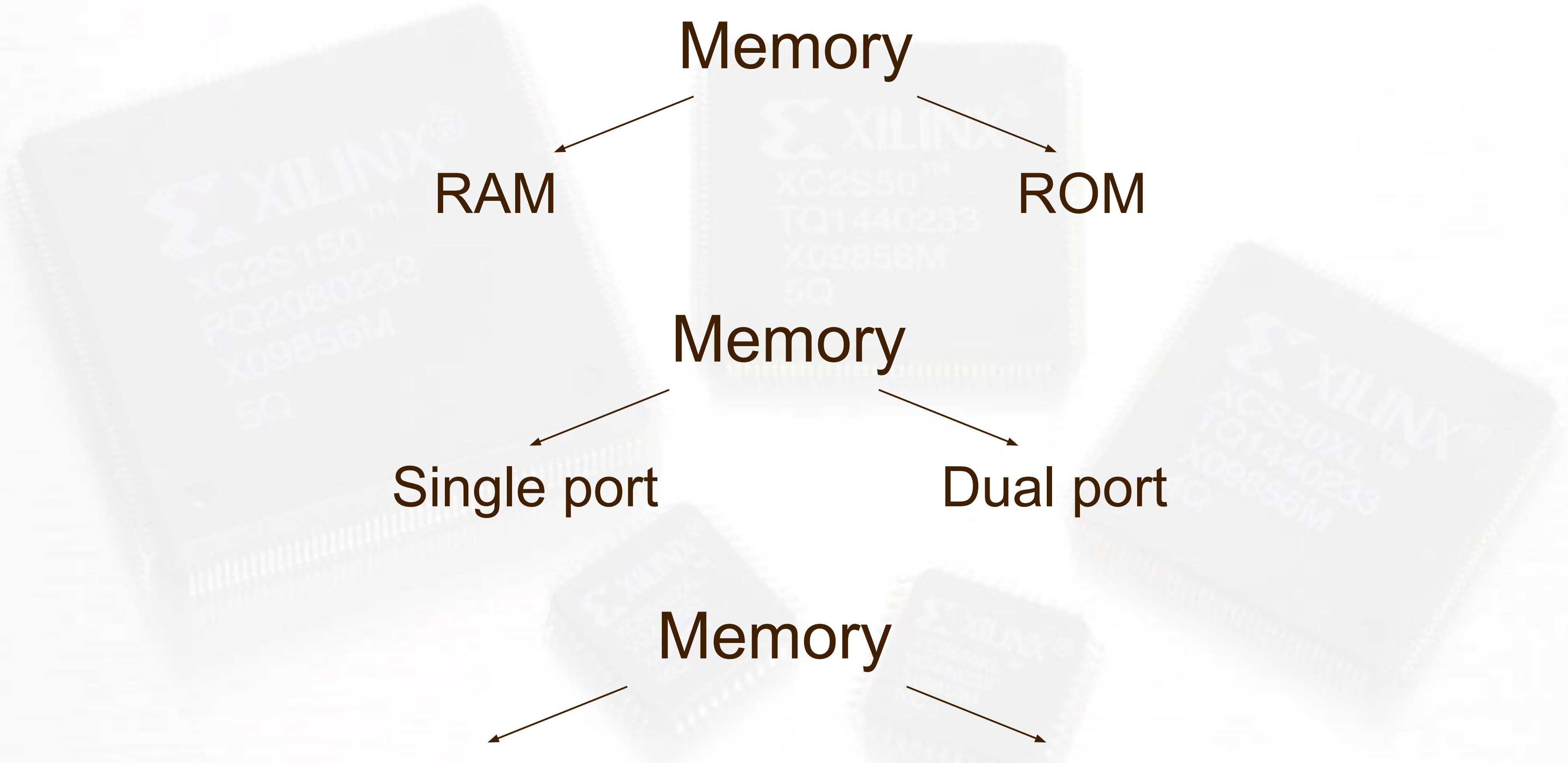
- ISE In-Depth Tutorial, Section: Creating a CORE Generator Software Module

  Google search: ISE In-Depth Tutorial

Memory Types



Memory Types

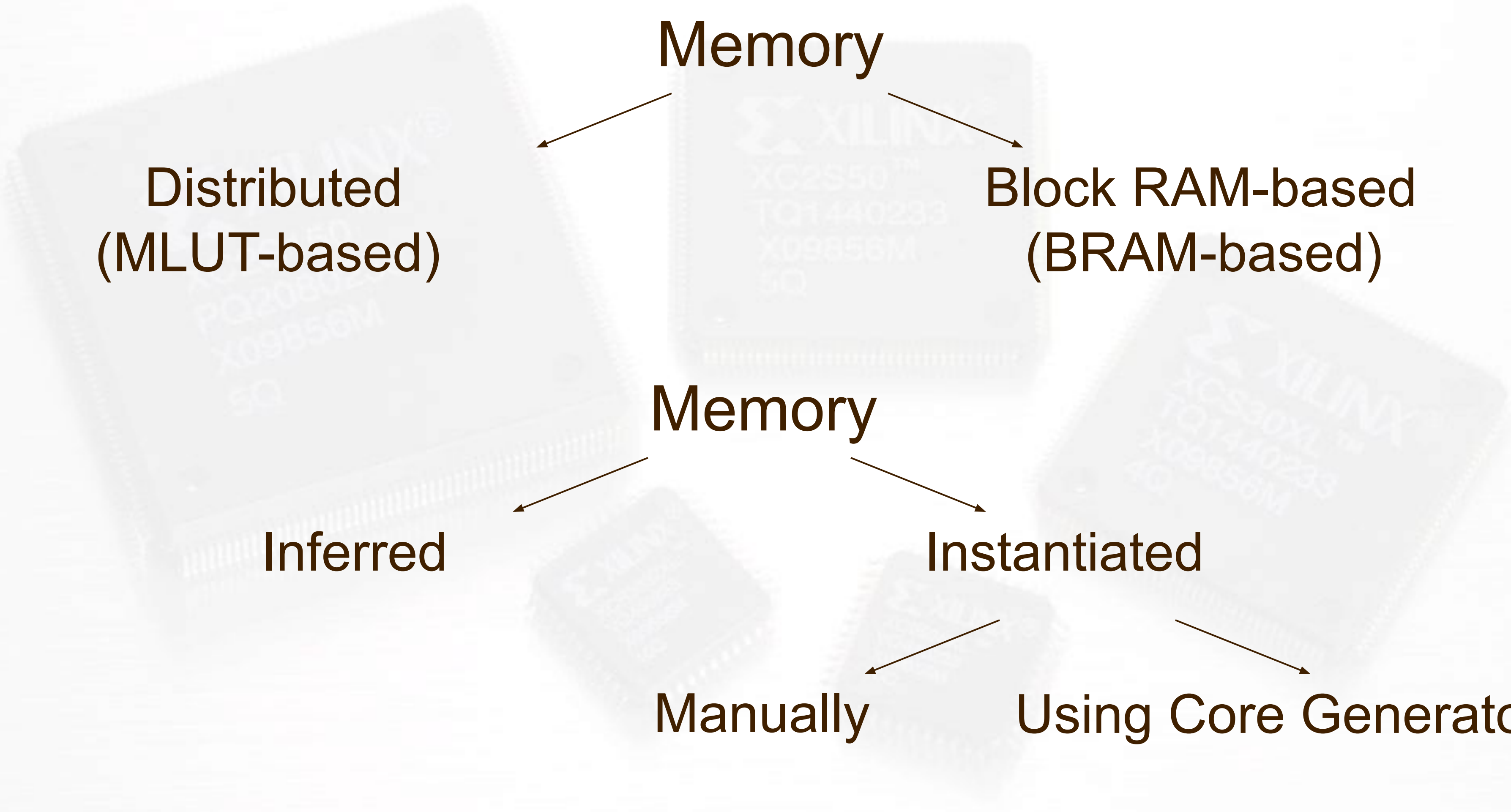


$$\begin{tikzcd}
 & \text{Memory} \arrow[dl] \arrow[dr] & \\
\text{RAM} & & \text{ROM}
\end{tikzcd}$$

$$\begin{tikzcd}
 & \text{Memory} \arrow[dl] \arrow[dr] & \\
\text{Single port} & & \text{Dual port}
\end{tikzcd}$$

$$\begin{tikzcd}
 & \text{Memory} \arrow[dl] \arrow[dr] & \\
\text{With asynchronous read} & & \text{With synchronous read}
\end{tikzcd}$$

Memory Types

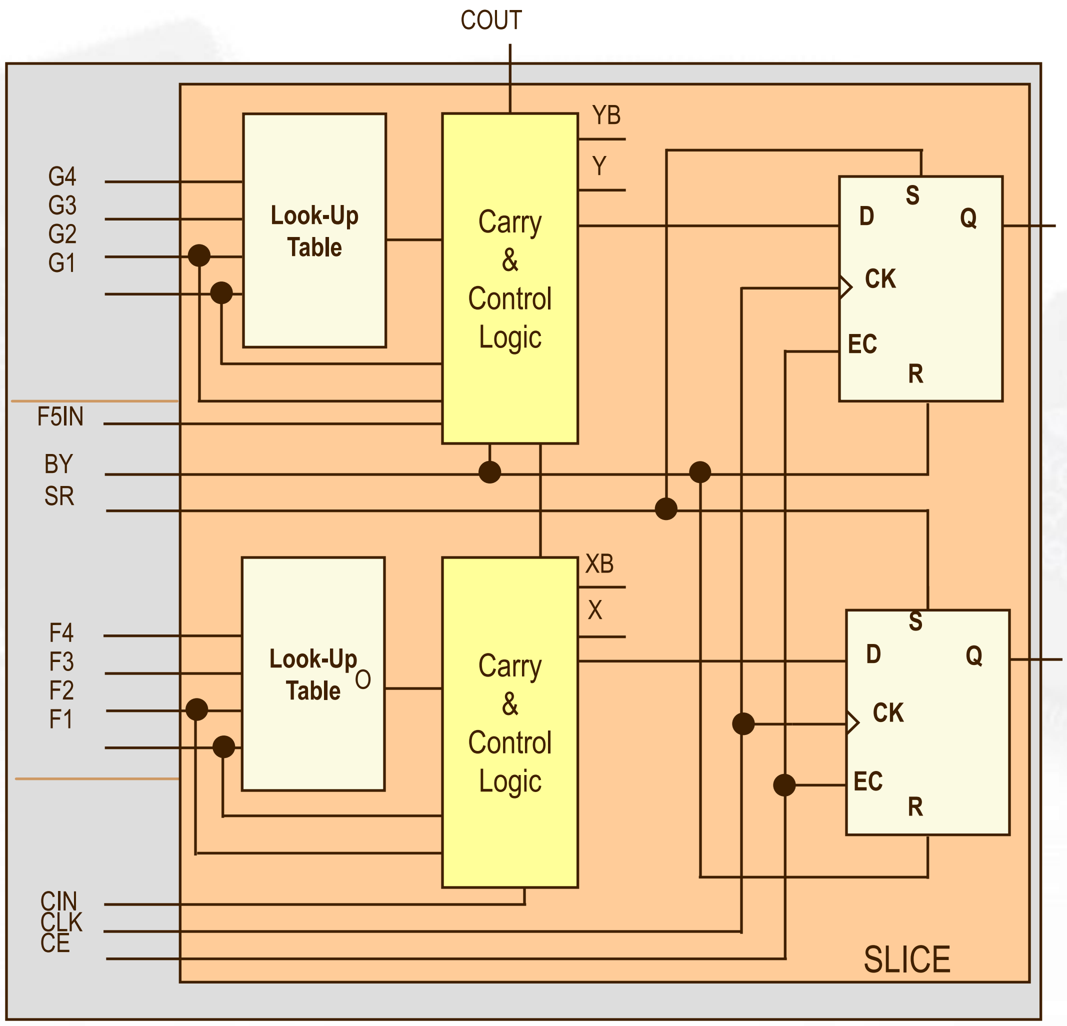


$$\begin{tikzcd}
 & \text{Memory} \arrow[dl] \arrow[dr] & \\
\text{Distributed (MLUT-based)} & & \text{Block RAM-based (BRAM-based)}
\end{tikzcd}$$

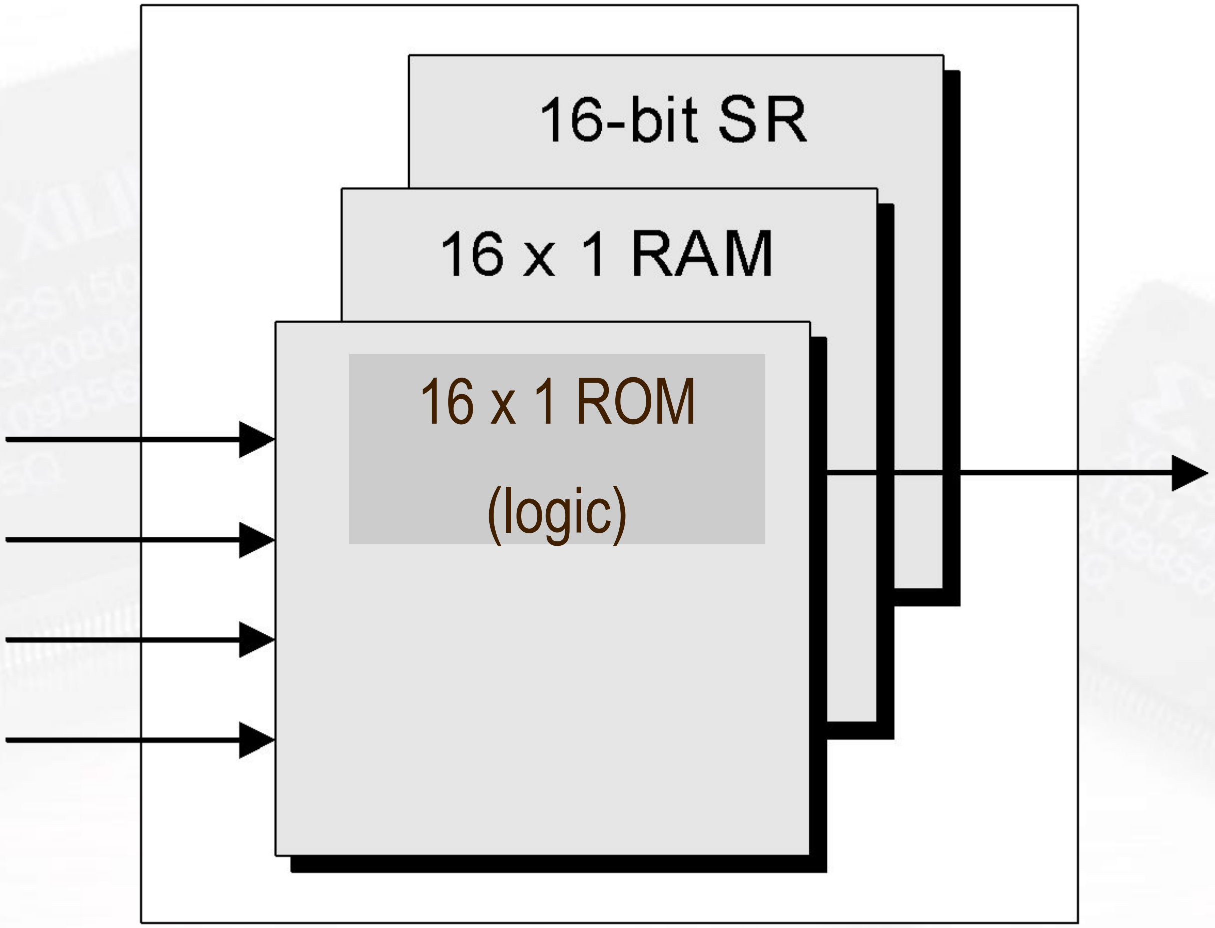
$$\begin{tikzcd}
 & \text{Memory} \arrow[dl] \arrow[dr] & & \\
\text{Inferred} & & \text{Instantiated} \arrow[dl] \arrow[dr] & \\
 & \text{Manually} & & \text{Using Core Generator}
\end{tikzcd}$$

FPGA Distributed Memory

CLB Slice

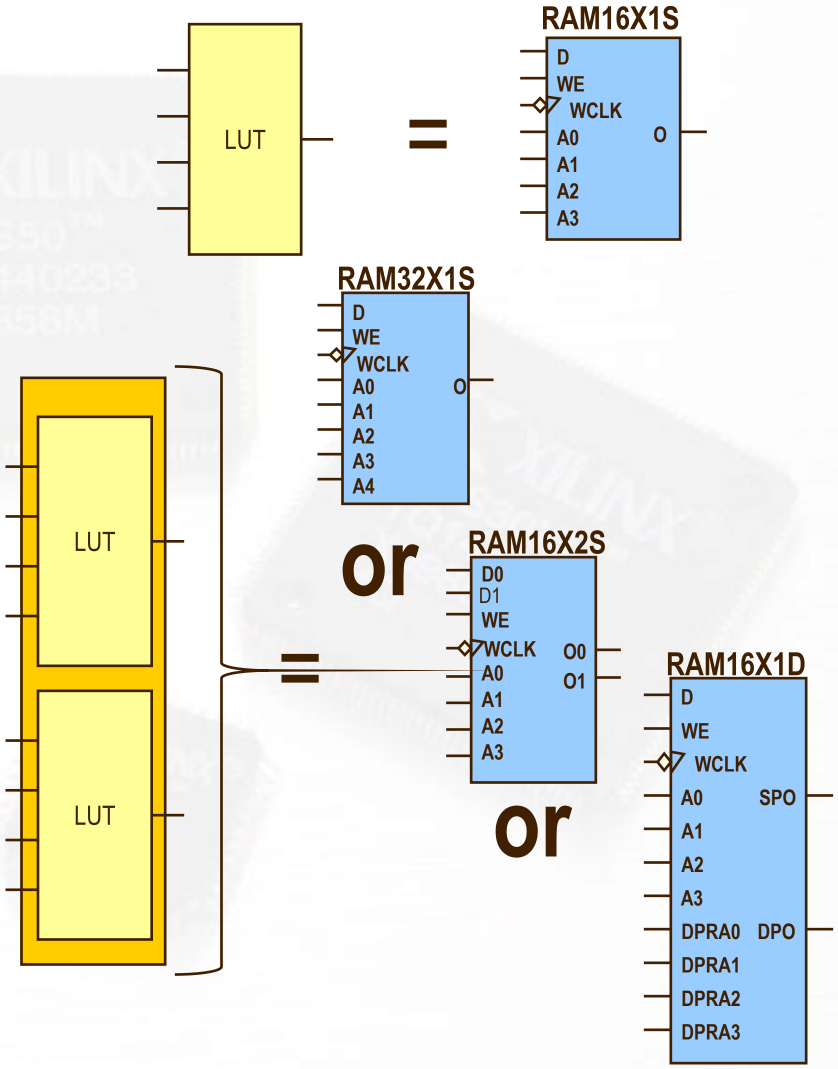

COUT
YB
Y
G4
G3
G2
G1
Look-Up Table
Carry & Control Logic
D
S
Q
CK
EC
R
F5IN
BY
SR
XB
X
F4
F3
F2
F1
Look-Up Table
O
Carry & Control Logic
S
D
Q
CK
EC
R
CIN
CLK
CE
SLICE

Xilinx Multipurpose LUT (MLUT)



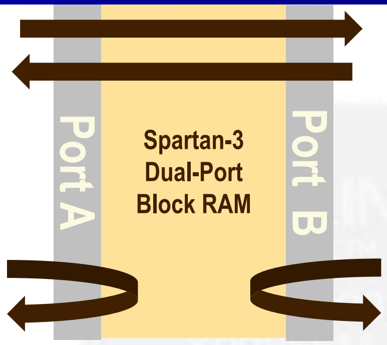
Distributed RAM

- CLB LUT configurable as Distributed RAM
  - An LUT equals 16x1 RAM
  - Cascade LUTs to increase RAM size
- Synchronous write
- Asynchronous read
  - Can create a synchronous read by using extra flip-flops
  - Naturally, distributed RAM read is asynchronous
- Two LUTs can make
  - 32 x 1 single-port RAM
  - 16 x 2 single-port RAM
  - 16 x 1 dual-port RAM



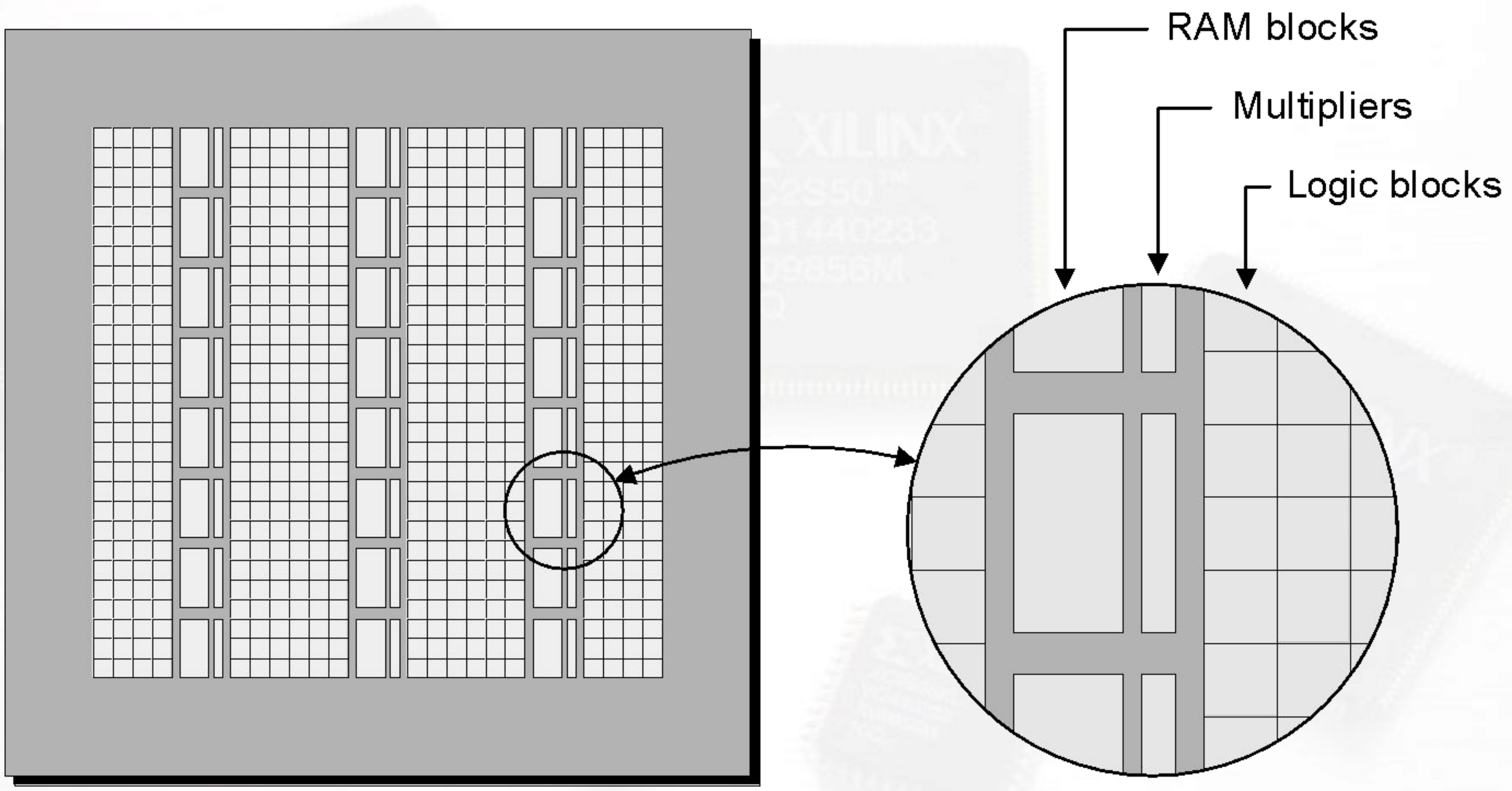
FPGA Block RAM

Block RAM



- Most efficient memory implementation
  - Dedicated blocks of memory
- Ideal for most memory requirements
  - 4 to 104 memory blocks
    - **18 kbits = 18,432 bits per block (16 k without parity bits)**
  - Use multiple blocks for larger memories
- Builds both single and true dual-port RAMs
- Synchronous write and read (different from distributed RAM)

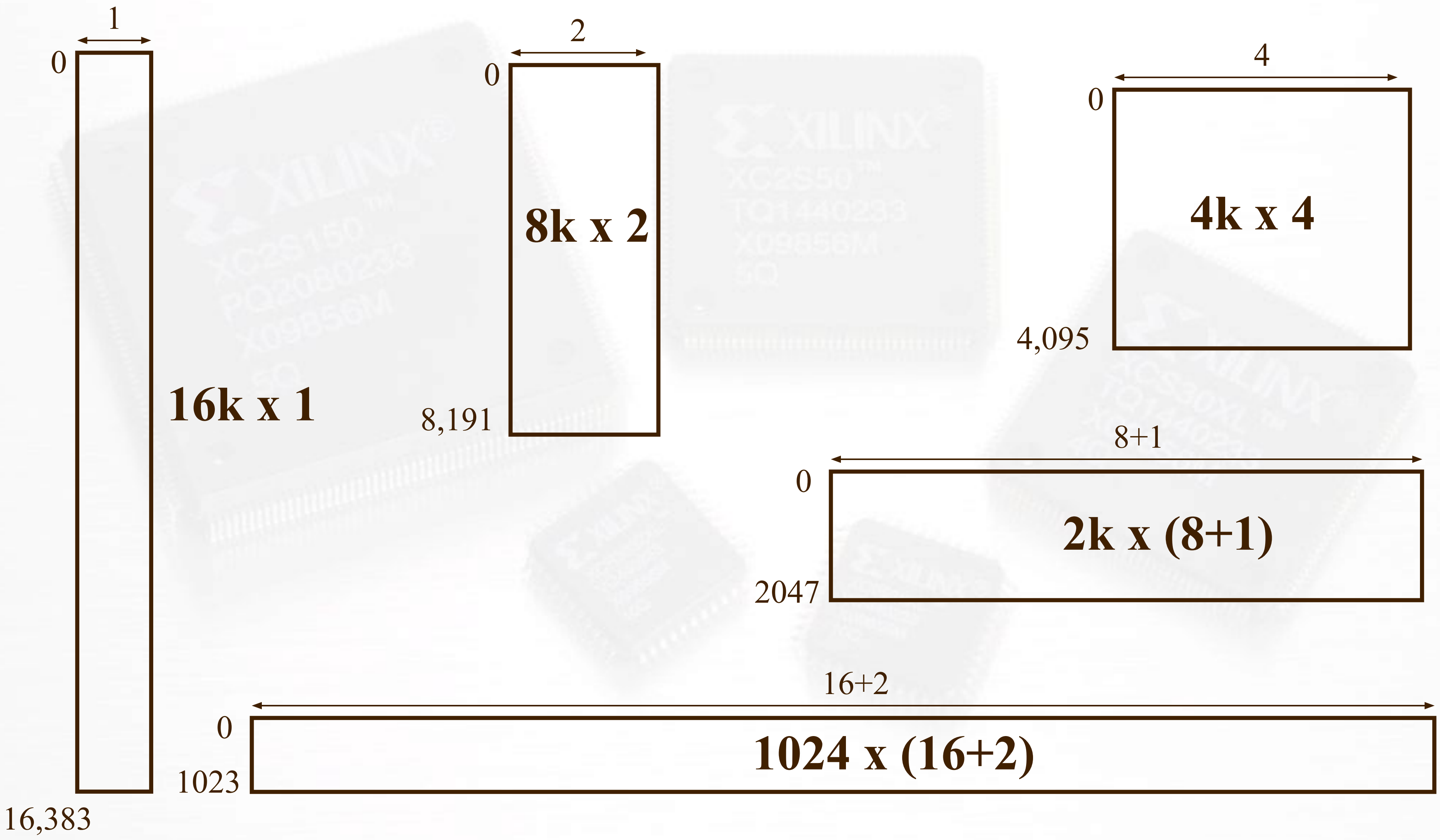
RAM Blocks and Multipliers in Xilinx FPGAs



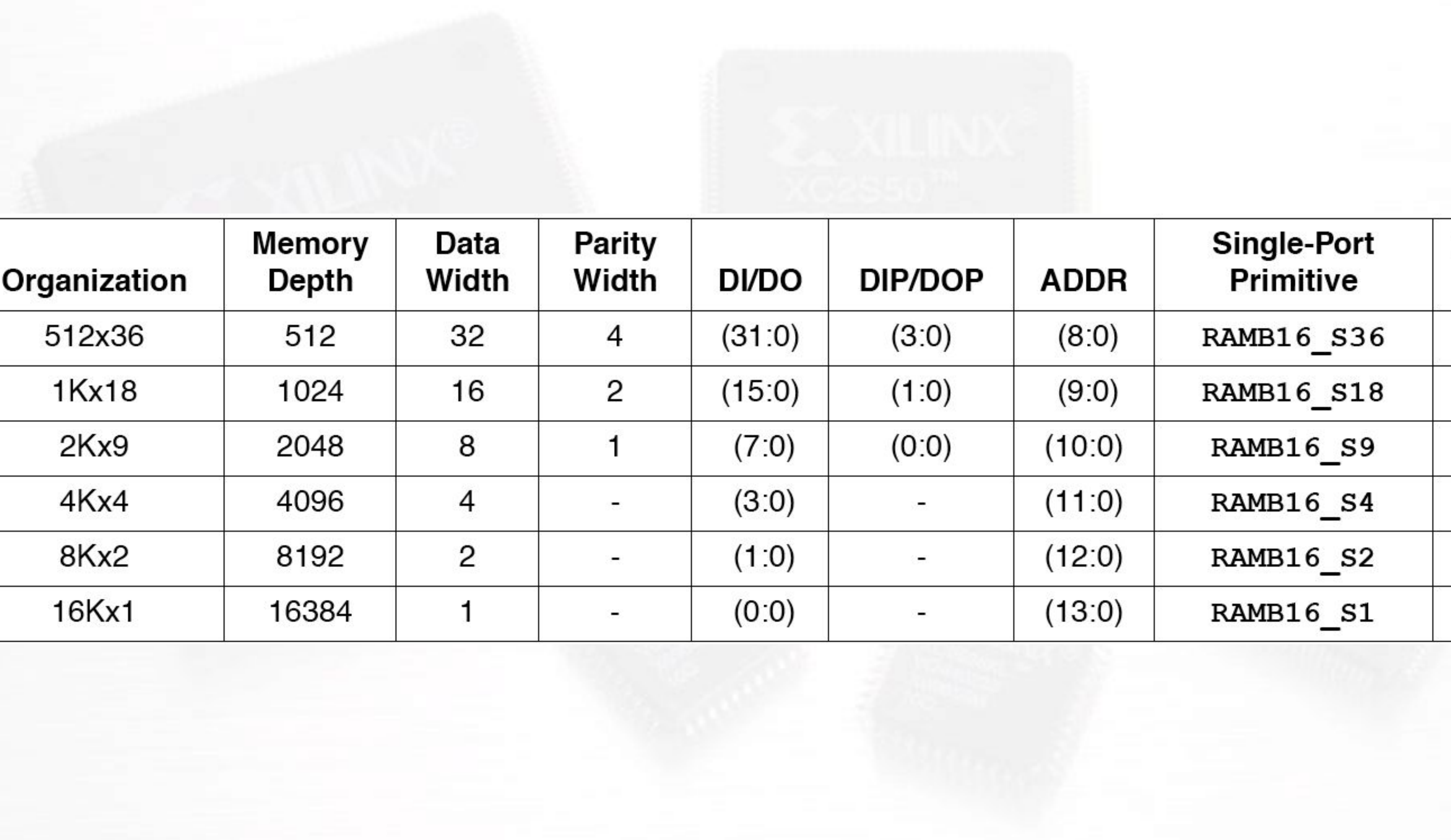
Spartan-3E Block RAM Amounts

Device	RAM Columns	RAM Blocks per Column	Total RAM Blocks	Total RAM Bits	Total RAM Kbits
XC3S100E	1	4	4	73728	72
XC3S250E	2	6	12	221184	216
XC3S500E	2	10	20	368640	360
XC3S1200E	2	14	28	516096	504
XC3S1600E	2	18	36	663552	648

Block RAM can have various configurations (port aspect ratios)

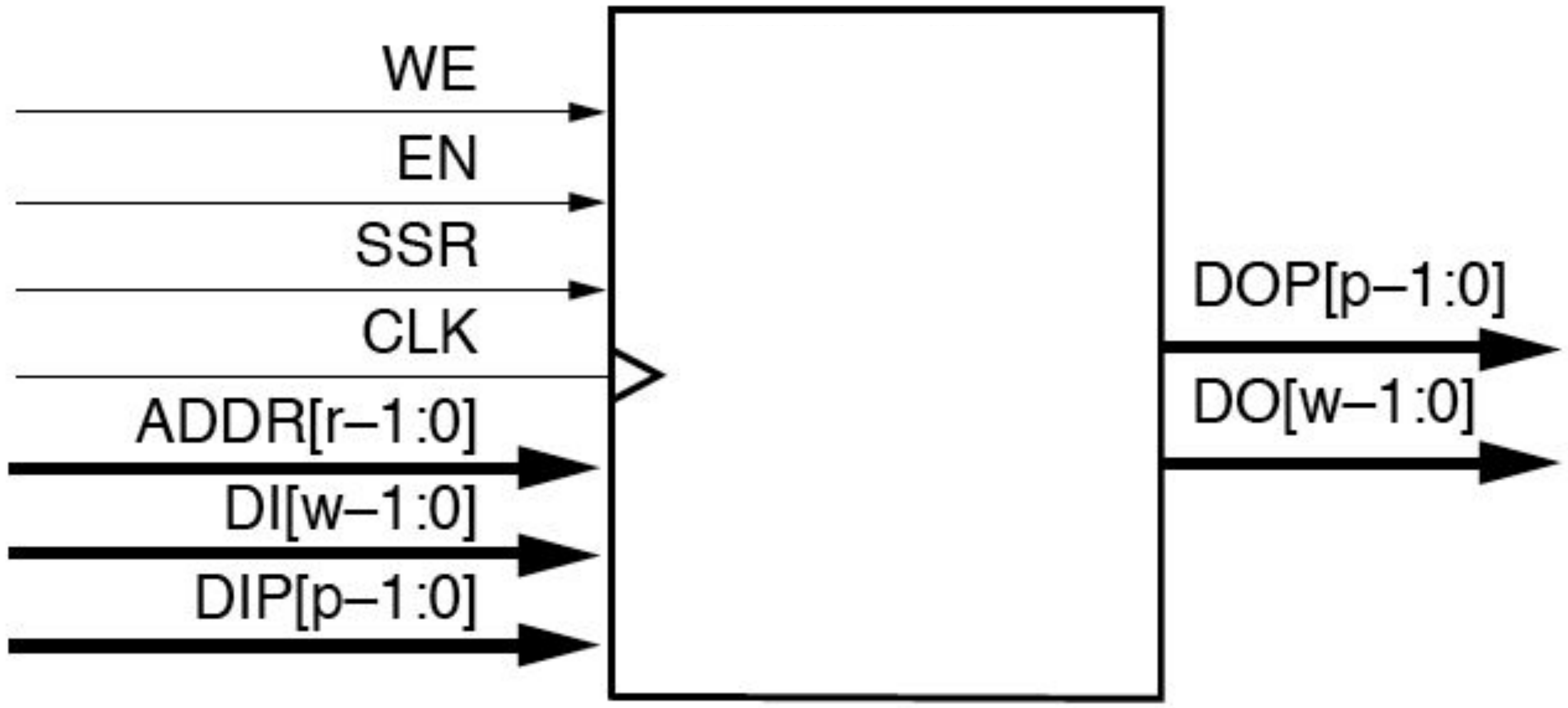


Block RAM Port Aspect Ratios



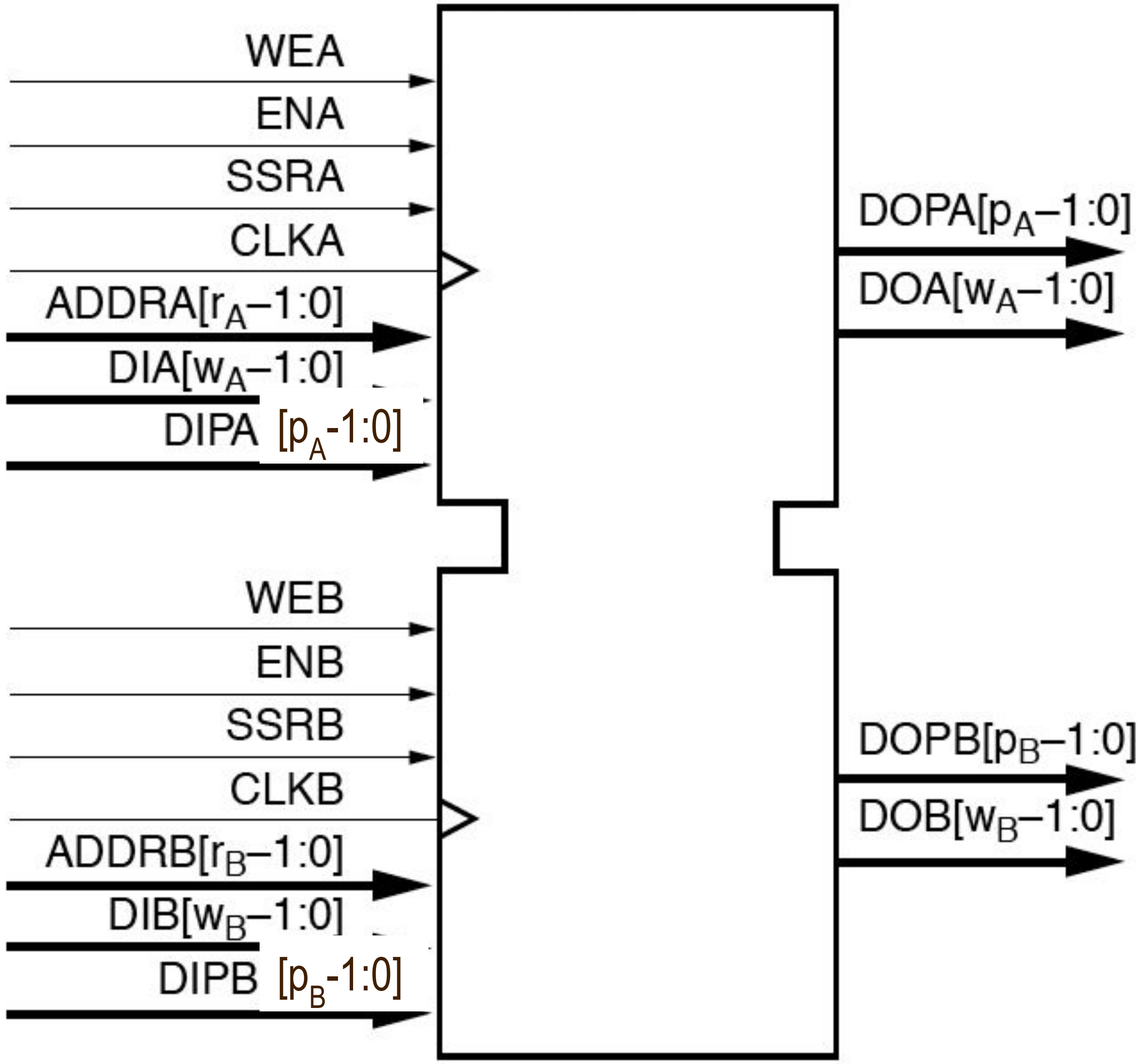
Organization	Memory Depth	Data Width	Parity Width	DI/DO	DIP/DOP	ADDR	Single-Port Primitive	Total RAM Kbits
512x36	512	32	4	(31:0)	(3:0)	(8:0)	`RAMB16_S36`	18K
1Kx18	1024	16	2	(15:0)	(1:0)	(9:0)	`RAMB16_S18`	18K
2Kx9	2048	8	1	(7:0)	(0:0)	(10:0)	`RAMB16_S9`	18K
4Kx4	4096	4	-	(3:0)	-	(11:0)	`RAMB16_S4`	16K
8Kx2	8192	2	-	(1:0)	-	(12:0)	`RAMB16_S2`	16K
16Kx1	16384	1	-	(0:0)	-	(13:0)	`RAMB16_S1`	16K

Single-Port Block RAM



(b) Single-Port

Dual-Port Block RAM

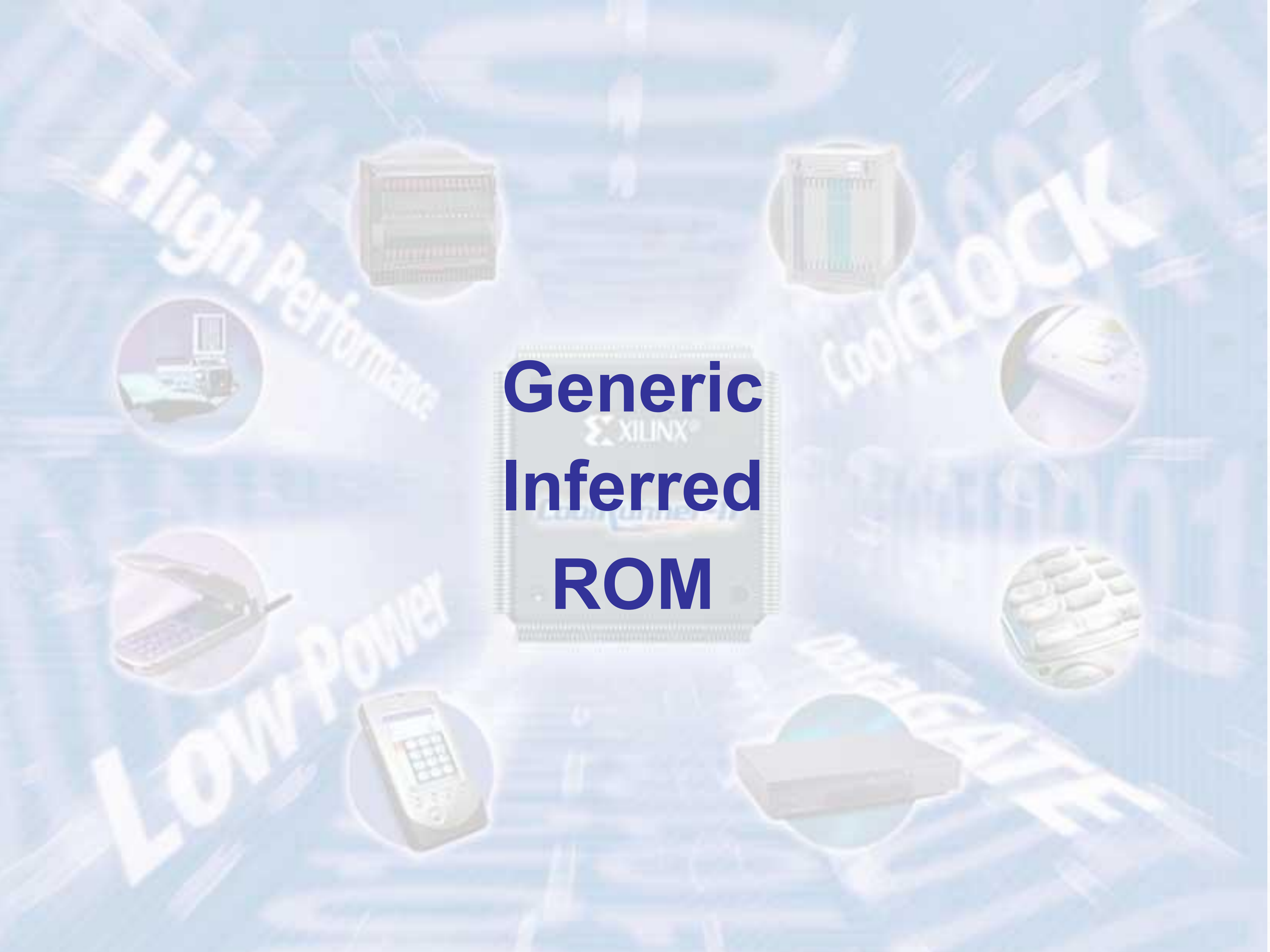


Inference
vs.
Instantiation

Inference vs. Instantiation

There are two methods to handle RAMs: instantiation and inference. Many FPGA families provide technology-specific RAMs that you can instantiate in your HDL source code. The software supports instantiation, but you can also set up your source code so that it infers the RAMs. The following table sums up the pros and cons of the two approaches.

Inference in Synthesis	Instantiation
Advantages	**Advantages**
Portable coding style	Most efficient use of the RAM primitives of a specific technology
Automatic timing-driven synthesis	Supports all kinds of RAMs
No additional tool dependencies	

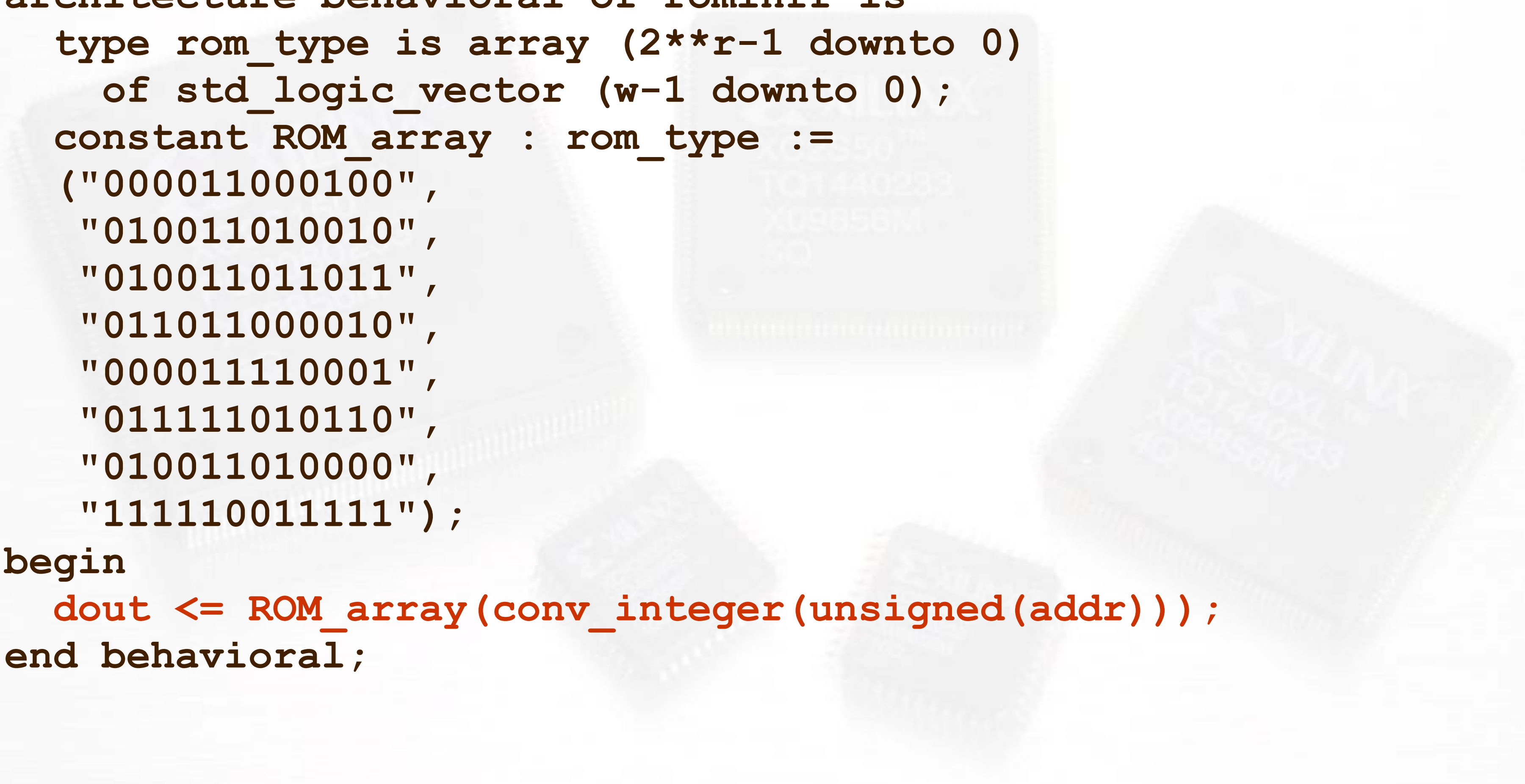


Generic Inferred ROM

Distributed ROM with asynchronous read

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

Entity ROM is
  generic ( w : integer := 12;
             -- number of bits per ROM word
            r : integer := 3);
             -- 2^r = number of words in ROM
  port (addr    : in std_logic_vector(r-1 downto 0);
        dout   : out std_logic_vector(w-1 downto 0));
end ROM;
```

Distributed ROM with asynchronous read



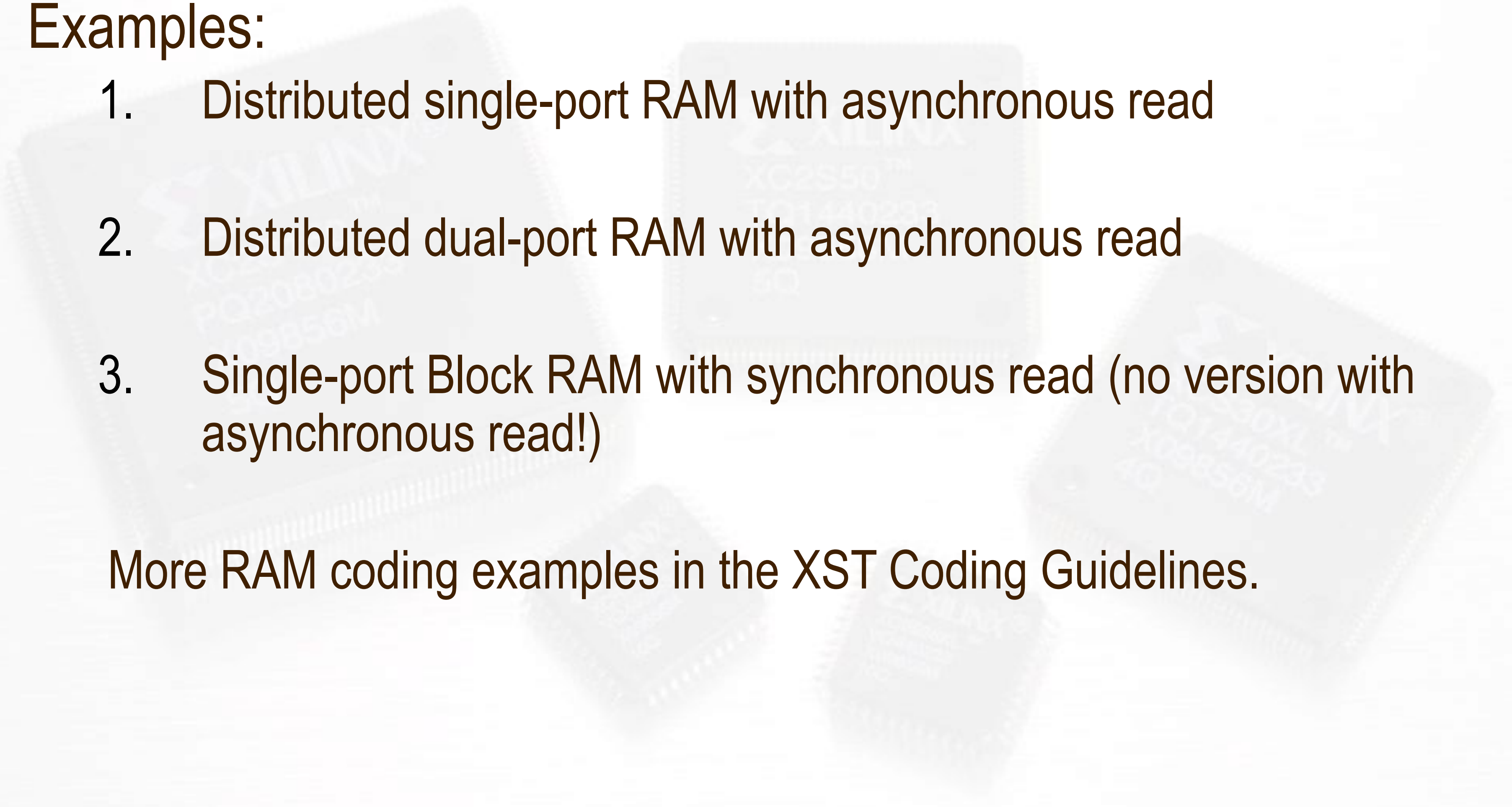
```
architecture behavioral of rominfr is
   type rom_type is array (2**r-1 downto 0)
     of std_logic_vector (w-1 downto 0);
   constant ROM_array : rom_type :=
   ("000011000100",
    "010011010010",
    "010011011011",
    "011011000010",
    "000011110001",
    "011111010110",
    "010011010000",
    "111110011111");
begin
   dout <= ROM_array(conv_integer(unsigned(addr)));
end behavioral;
```

Distributed ROM with asynchronous read

```
architecture behavioral of rominfr is
   type rom_type is array (2**r-1 downto 0)
     of std_logic_vector (w-1 downto 0);
   constant ROM_array : rom_type :=
   ("0C4",
    "4D2",
    "4DB",
    "6C2",
    "0F1",
    "7D6",
    "4D0",
    "F9F");
begin
   dout <= ROM_array(conv_integer(unsigned(addr)));
end behavioral;
```

Generic Inferred RAM

Distributed versus Block RAM Inference

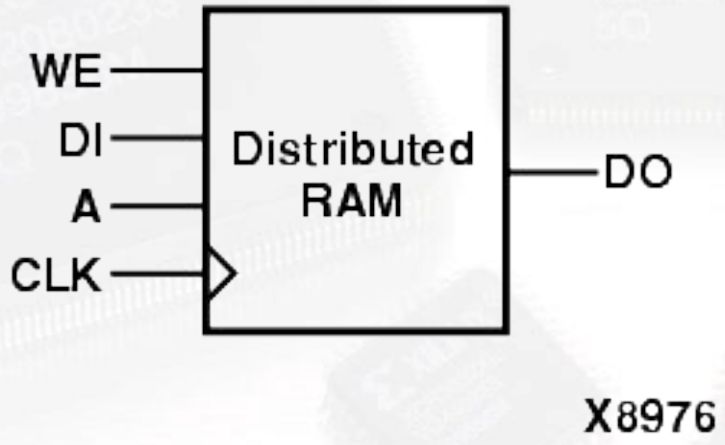


Examples:

1. Distributed single-port RAM with asynchronous read
2. Distributed dual-port RAM with asynchronous read
3. Single-port Block RAM with synchronous read (no version with asynchronous read!)

More RAM coding examples in the XST Coding Guidelines.

Distributed RAM with asynchronous read



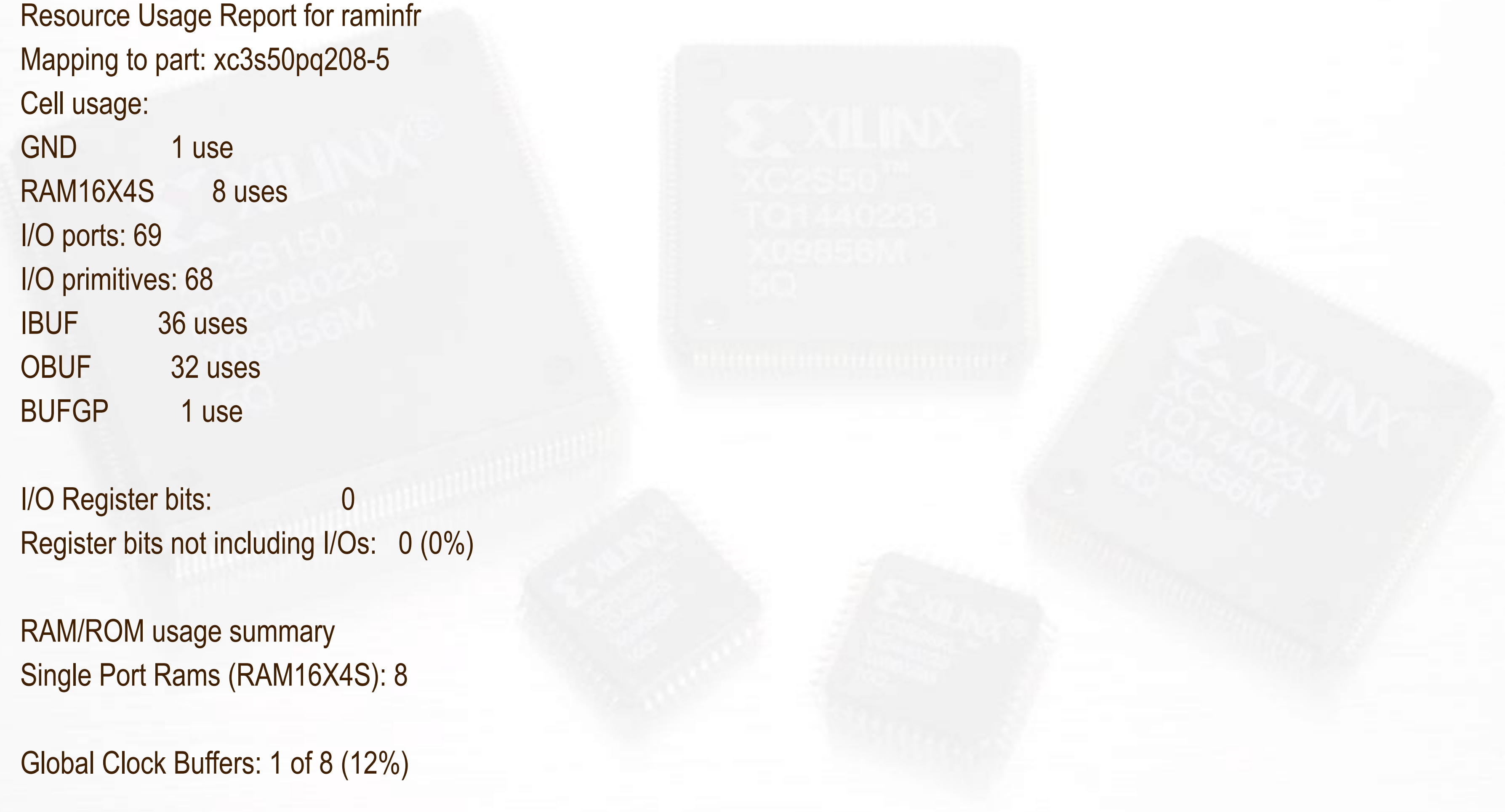
Distributed single-port RAM with asynchronous read

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

entity raminfr is
  generic ( w : integer := 32;
            -- number of bits per RAM word
            r : integer := 3);
            -- 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 (2**r-1 downto 0)
        of std_logic_vector (w-1 downto 0);
  signal RAM : ram_type;
begin
  process (clk)
  begin
    if (clk'event and clk = '1') then
      if (we = '1') then
        RAM(conv_integer(unsigned(a))) <= di;
      end if;
    end if;
  end process;
  do <= RAM(conv_integer(unsigned(a)));
end behavioral;
```

Report from Synthesis

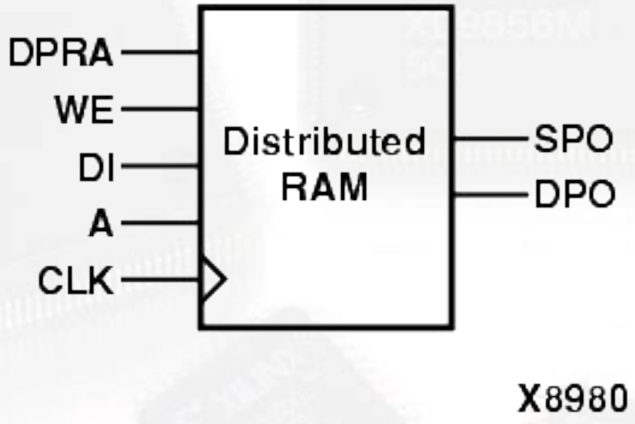


```
Resource Usage Report for raminfr
Mapping to part: xc3s50pq208-5
Cell usage:
GND           1 use
RAM16X4S      8 uses
I/O ports: 69
I/O primitives: 68
IBUF         36 uses
OBUF         32 uses
BUFGP         1 use

I/O Register bits:                0
Register bits not including I/Os:  0 (0%)

RAM/ROM usage summary
Single Port Rams (RAM16X4S): 8

Global Clock Buffers: 1 of 8 (12%)

Mapping Summary:
Total  LUTs: 32 (2%)
```

Report from Implementation

```
Design Summary:
   Number of errors:      0
   Number of warnings:    0
   Logic Utilization:
   Logic Distribution:
    Number of occupied Slices:                    16 out of    768    2%
      Number of Slices containing only related logic:    16 out of     16  100%
      Number of Slices containing unrelated logic:        0 out of     16    0%
        *See NOTES below for an explanation of the effects of unrelated logic
   Total Number of 4 input LUTs:            32 out of   1,536    2%
    Number used as 16x1 RAMs:               32
    Number of bonded IOBs:                69 out of     124   55%
    Number of GCLKs:                     1 out of       8   12%
```

Distributed dual-port RAM with asynchronous read



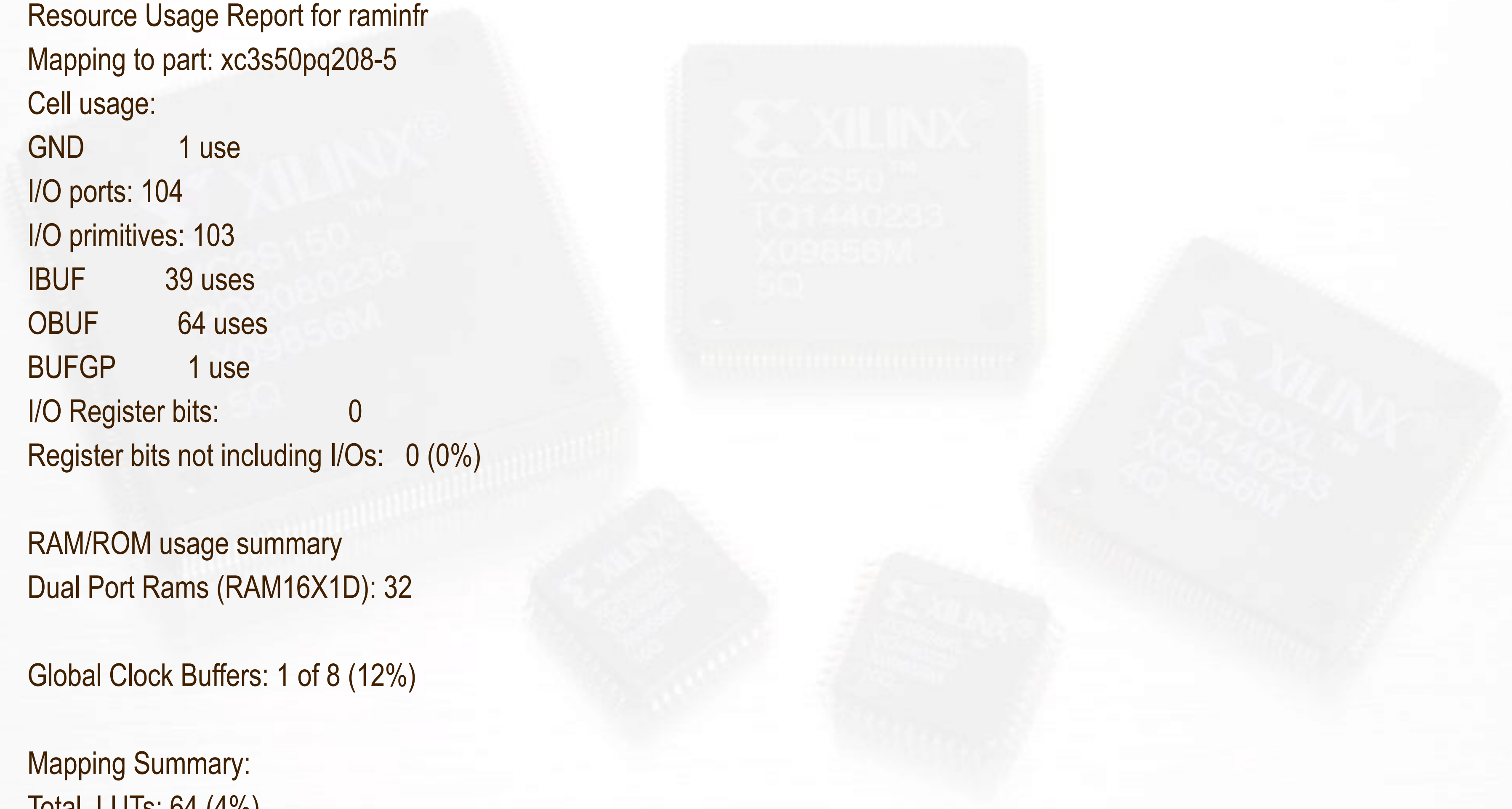
Distributed dual-port RAM with asynchronous read

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity raminfr is
 generic ( w : integer := 32;
             -- number of bits per RAM word
            r : integer := 3);
             -- 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 (2**r-1 downto 0) of std_logic_vector
   (w-1 downto 0);
  signal RAM : ram_type;
begin
  process (clk)
  begin
    if (clk'event and clk = '1') then
      if (we = '1') then
        RAM(conv_integer(unsigned(a))) <= di;
      end if;
    end if;
  end process;
  spo <= RAM(conv_integer(unsigned(a)));
  dpo <= RAM(conv_integer(unsigned(dpra)));
 end syn;
```

Report from Synthesis

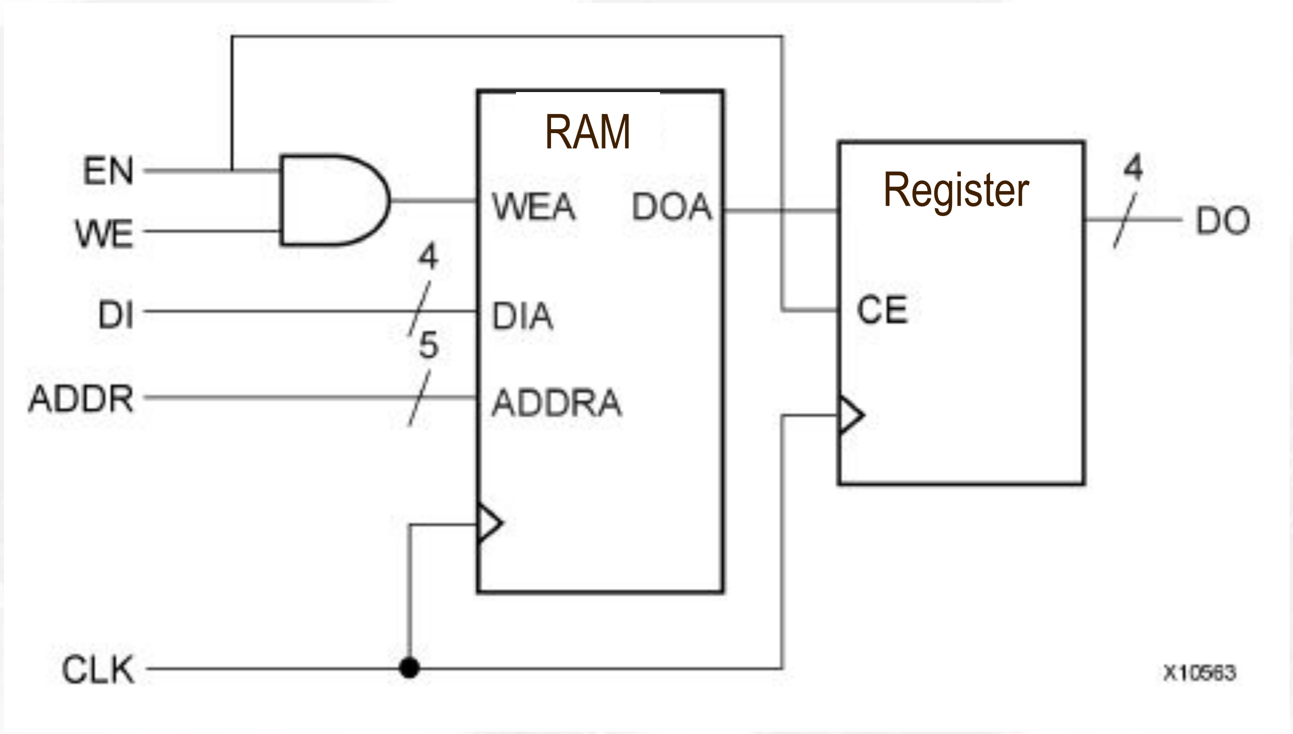


```
Resource Usage Report for raminfr
Mapping to part: xc3s50pq208-5
Cell usage:
GND           1 use
I/O ports: 104
I/O primitives: 103
IBUF         39 uses
OBUF         64 uses
BUFGP         1 use
I/O Register bits:              0
Register bits not including I/Os:   0 (0%)

RAM/ROM usage summary
Dual Port Rams (RAM16X1D): 32

Global Clock Buffers: 1 of 8 (12%)

Mapping Summary:
Total  LUTs: 64 (4%)
```

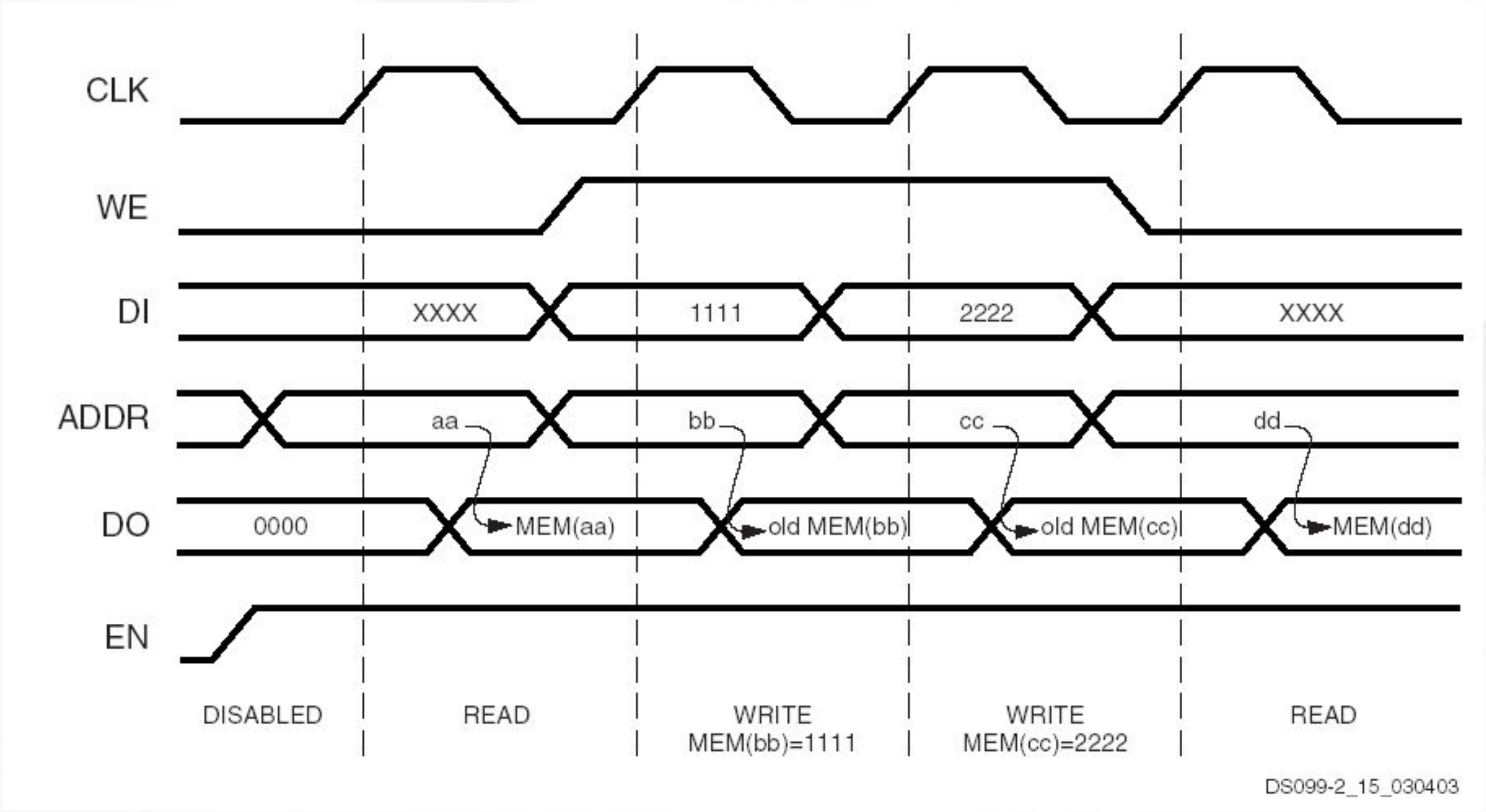
Report from Implementation

```
Design Summary:
Number of errors:      0
Number of warnings:    0
Logic Utilization:
Logic Distribution:
  Number of occupied Slices:                          32 out of    768    4%
    Number of Slices containing only related logic:      32 out of     32  100%
    Number of Slices containing unrelated logic:          0 out of     32    0%
      *See NOTES below for an explanation of the effects of unrelated logic
Total Number of 4 input LUTs:             64 out of   1,536    4%
  Number used for Dual Port RAMs:        64
    (Two LUTs used per Dual Port RAM)
  Number of bonded IOBs:               104 out of     124   83%
  Number of GCLKs:                       1 out of       8   12%
```

Block RAM with synchronous read in Read-First Mode



Block RAM Waveforms – READ_FIRST mode



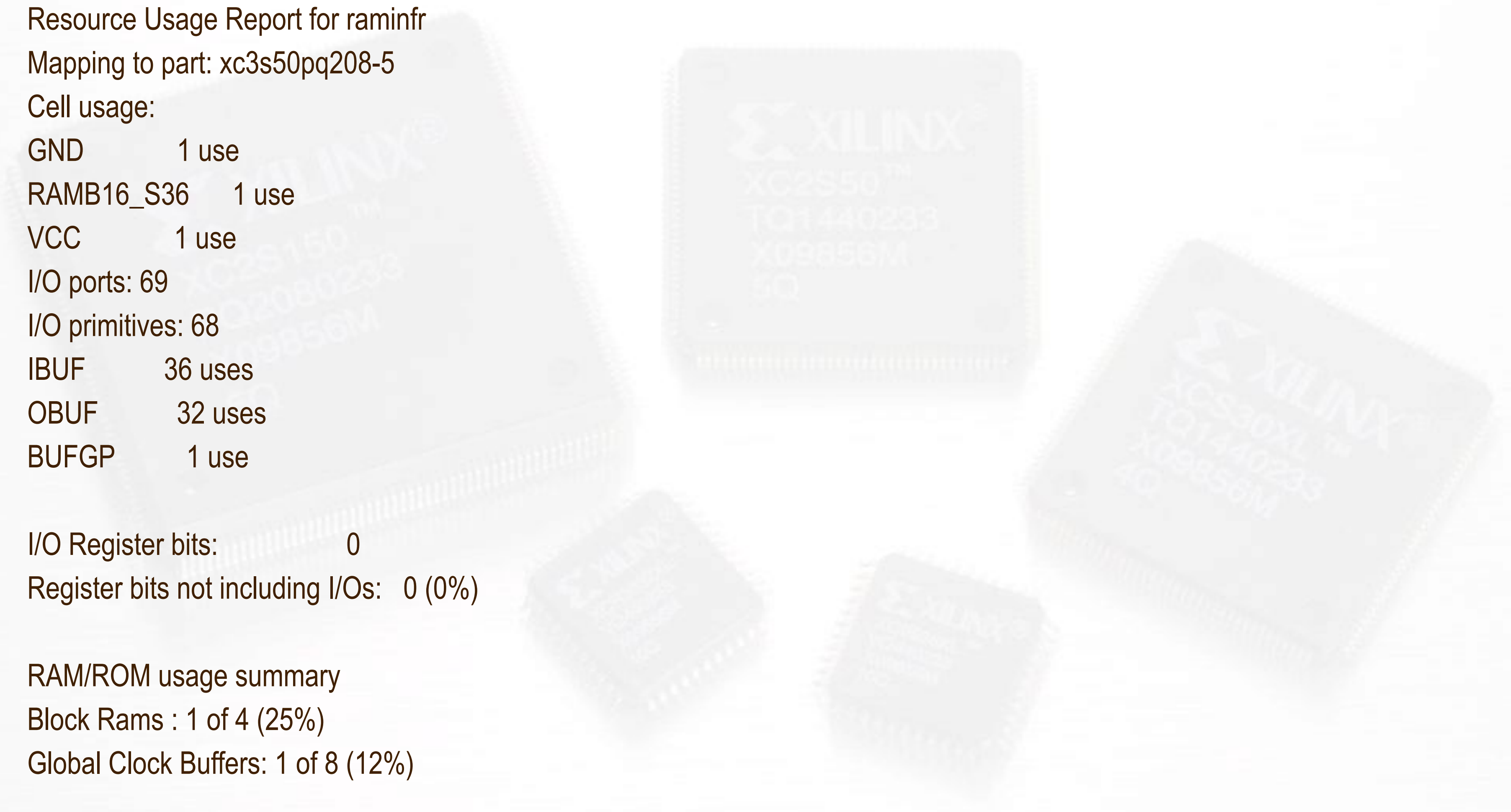
Block RAM with synchronous read Read-First Mode

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.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 with synchronous read Read First Mode - cont'd

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

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

Report from Synthesis

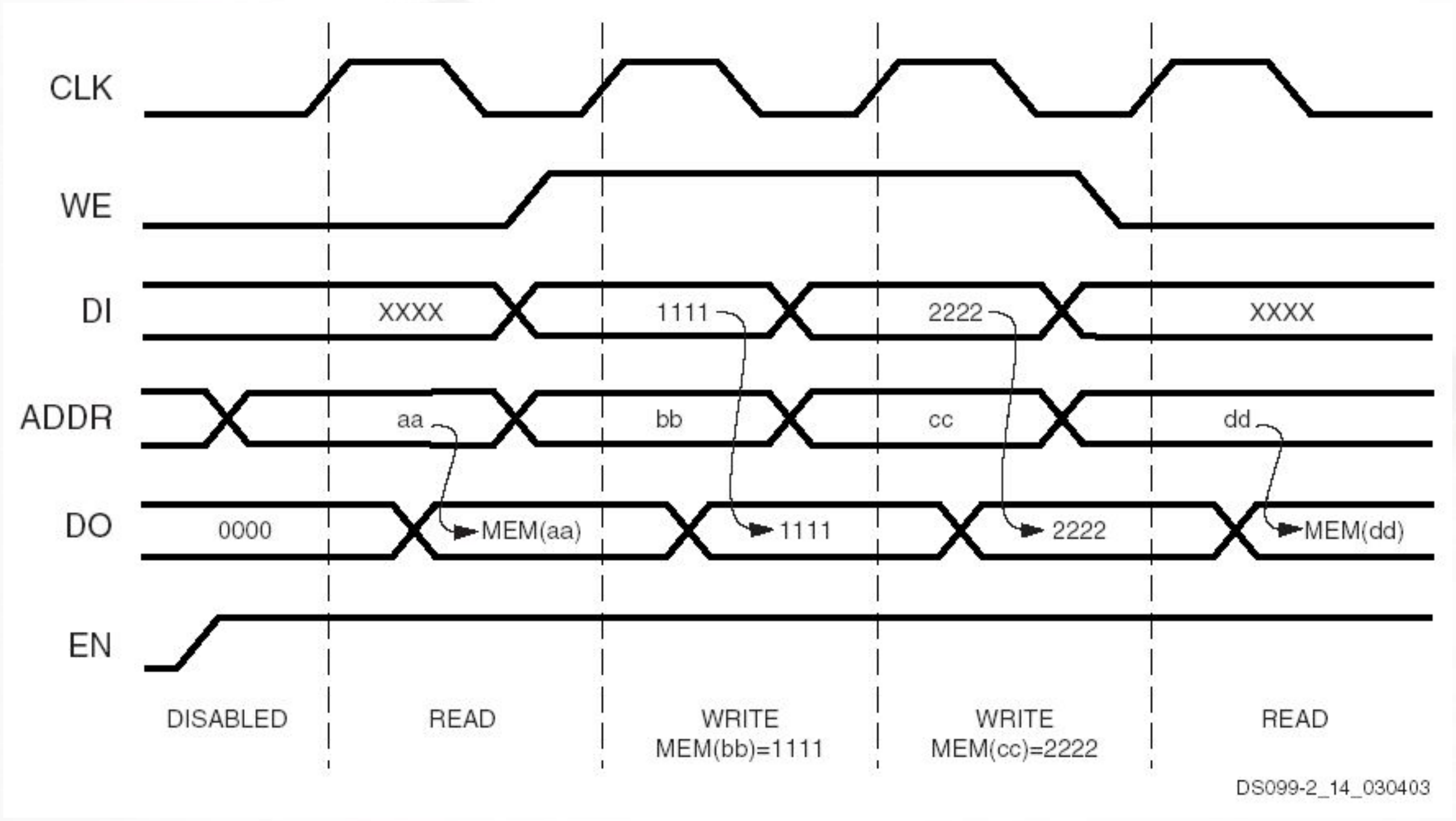


```
Resource Usage Report for raminfr
Mapping to part: xc3s50pq208-5
Cell usage:
GND            1 use
RAMB16_S36       1 use
VCC            1 use
I/O ports: 69
I/O primitives: 68
IBUF          36 uses
OBUF          32 uses
BUFGP          1 use

I/O Register bits:                0
Register bits not including I/Os:   0 (0%)

RAM/ROM usage summary
Block Rams : 1 of 4 (25%)
Global Clock Buffers: 1 of 8 (12%)

Mapping Summary:
Total  LUTs: 0 (0%)
```

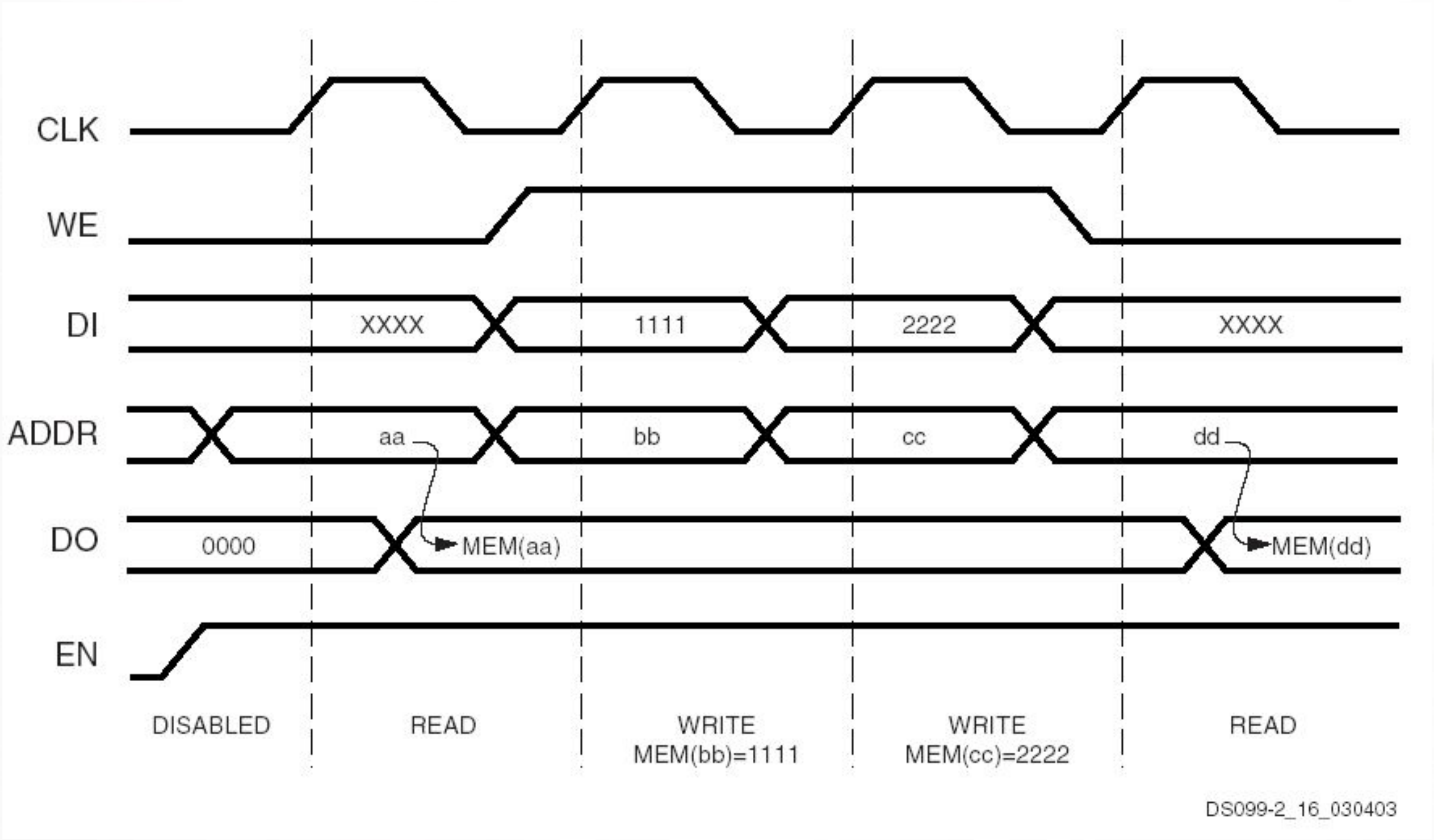
Report from Implementation

```
Design Summary:
Number of errors:      0
Number of warnings:    0
Logic Utilization:
Logic Distribution:
   Number of Slices containing only related logic:      0 out of      0    0%
   Number of Slices containing unrelated logic:         0 out of      0    0%
     *See NOTES below for an explanation of the effects of unrelated logic
  Number of bonded IOBs:              69 out of    124   55%
  Number of Block RAMs:                1 out of      4   25%
  Number of GCLKs:                     1 out of      8   12%
```

Block RAM Waveforms – WRITE_FIRST mode



Block RAM Waveforms – NO_CHANGE mode



FPGA specific memories: Instantiation

Genaral template of BRAM instantiation (1)

```
-- Component Attribute Specification for RAMB16_{S1 | S2 | S4}
-- Should be placed after architecture declaration but before the begin
-- Put attributes, if necessary
-- Component Instantiation for RAMB16_{S1 | S2 | S4}
-- Should be placed in architecture after the begin keyword
RAMB16_{S1 | S2 | S4}_INSTANCE_NAME : RAMB16_S1
-- synthesis translate_off
generic map (
       INIT => bit_value,
       INIT_00 => vector_value,
       INIT_01 => vector_value,
       ……………………………..
       INIT_3F => vector_value,
       SRVAL=> bit_value,
       WRITE_MODE => user_WRITE_MODE)
   -- synopsys translate_on
   port map (DO => user_DO,
             ADDR => user_ADDR,
             CLK => user_CLK,
             DI => user_DI,
             EN => user_EN,
             SSR => user_SSR,
             WE => user_WE);
```

Initializing Block RAMs 1024x16

```
INIT_00 : BIT_VECTOR := X"014A0C0F09170A04076802A800260205002A01C5020A0917006A006800060040";
INIT_01 : BIT_VECTOR := X"000000000000000008000A1907070A1706070A020026014A0C0F03AA09170026";
INIT_02 : BIT_VECTOR := X"0000000000000000000000000000000000000000000000000000000000000000";
INIT_03 : BIT_VECTOR := X"0000000000000000000000000000000000000000000000000000000000000000";
..............................................................................................
INIT_3F : BIT_VECTOR := X"0000000000000000000000000000000000000000000000000000000000000000")
```

DATA
ADDRESS

INIT_00 ADDRESS	014A 0F	0C0F 0E			0917 04	006A 03	0068 02	0006 01	0040 00
INIT_01 ADDRESS	0000 1F	0000 1E			014A 14	0C0F 13	03AA 12	0917 11	0026 10
INIT_3F ADDRESS	0000 FF	0000 FE	Addresses are shown in **red** and data corresponding to the same memory location is shown in **black**		0000 F4	0000 F3	0000 F2	0000 F1	0000 F0

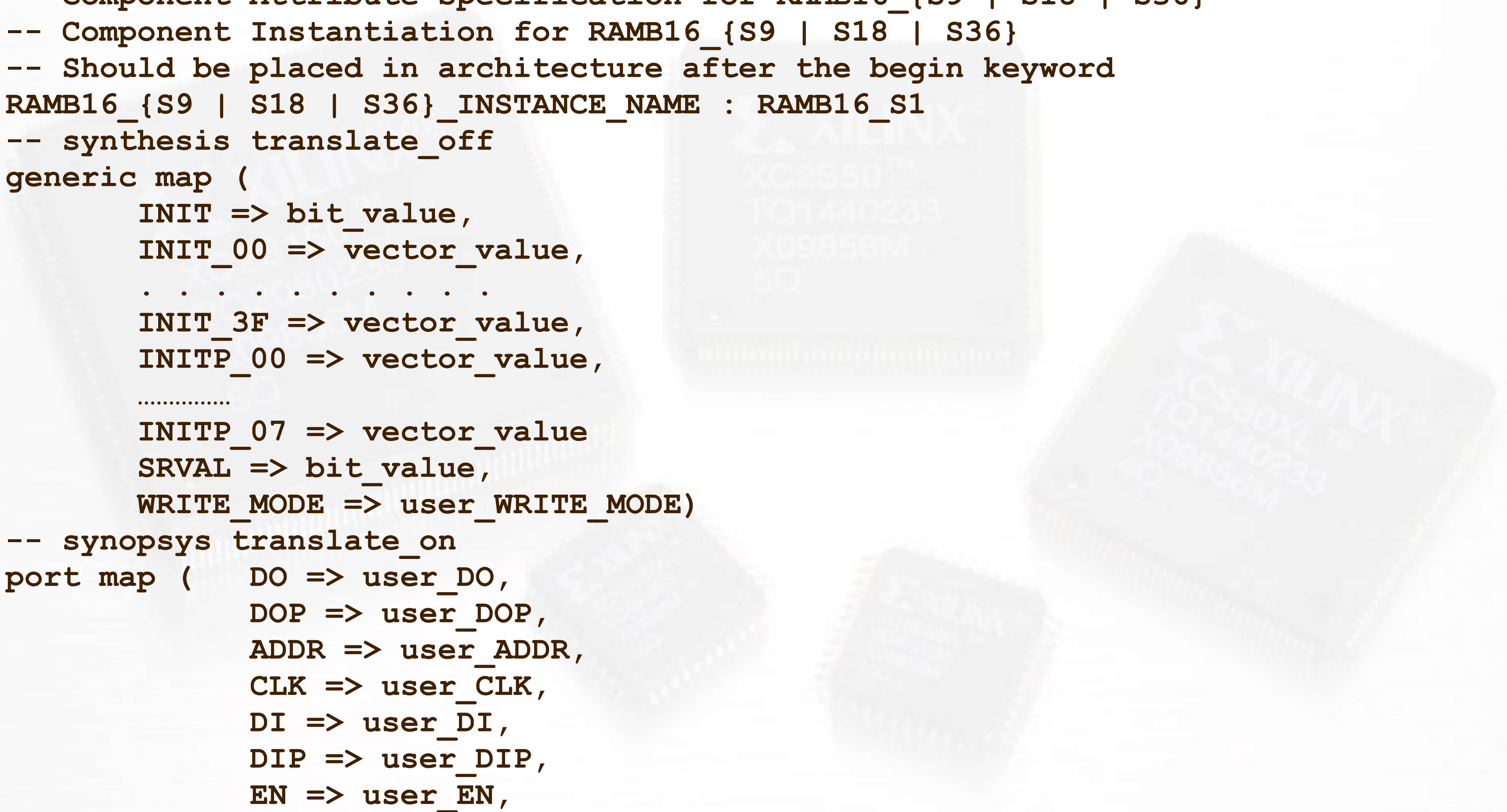
Component declaration for BRAM (2)

```
VHDL Instantiation Template for RAMB16_S9, S18 and S36
-- Component Declaration for RAMB16_{S9 | S18 | S36}
component RAMB16_{S9 | S18 | S36}
-- synthesis translate_off
generic (
       INIT : bit_vector := X"0";
       INIT_00 : bit_vector :=
   X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_3E : bit_vector :=
   X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_3F : bit_vector :=
   X"0000000000000000000000000000000000000000000000000000000000000000";
       INITP_00 : bit_vector :=
   X"0000000000000000000000000000000000000000000000000000000000000000";
       INITP_07 : bit_vector :=
   X"0000000000000000000000000000000000000000000000000000000000000000";
       SRVAL : bit_vector := X"0";
       WRITE_MODE : string := "READ_FIRST"; );
```

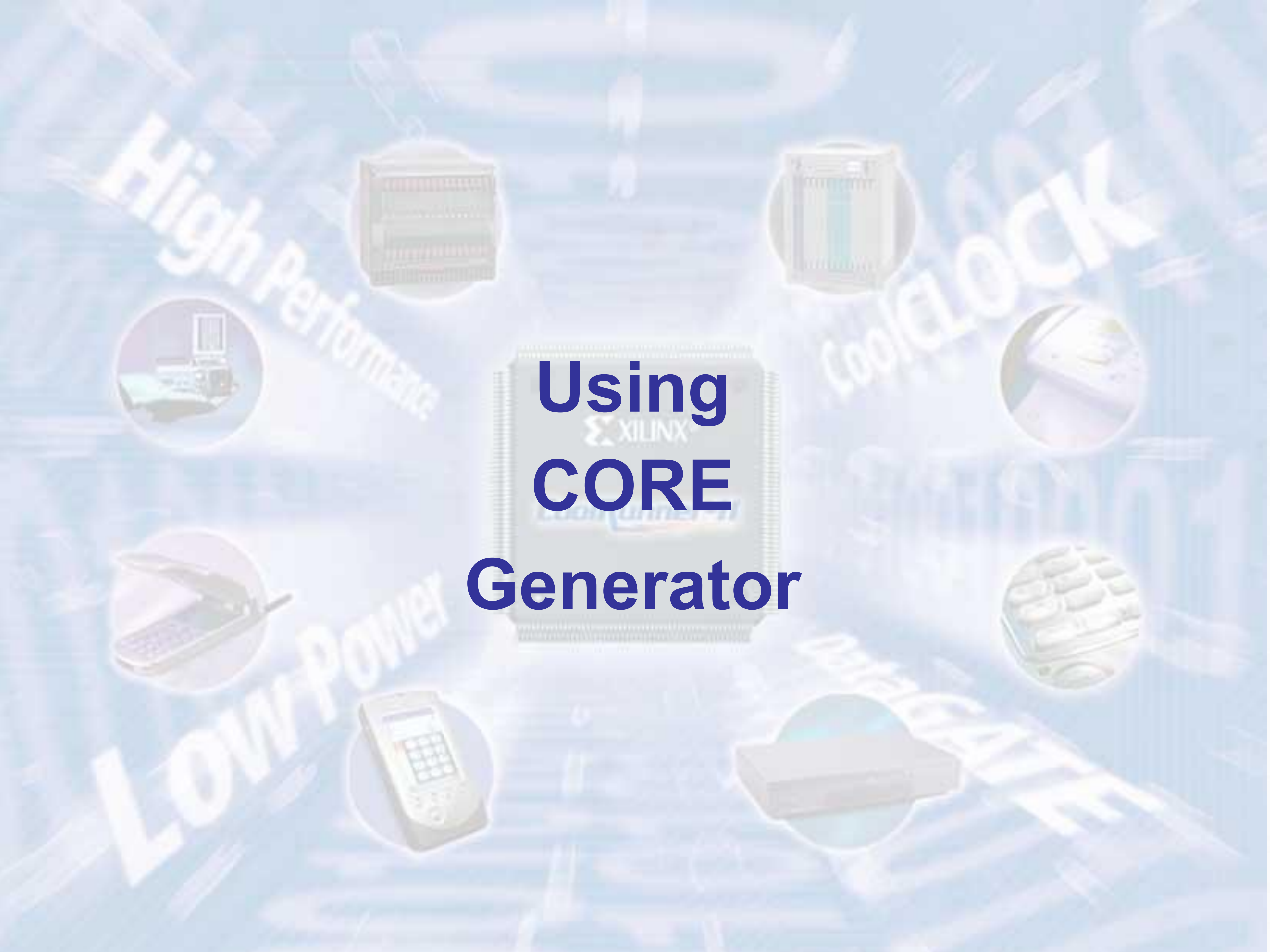
Component declaration for BRAM (2)

```
-- synthesis translate_on
   port (DO : out STD_LOGIC_VECTOR (31 downto 0);
         DOP : out STD_LOGIC_VECTOR (3 downto 0);
         ADDR : in STD_LOGIC_VECTOR (8 downto 0);
         CLK : in STD_ULOGIC;
         DI : in STD_LOGIC_VECTOR (31 downto 0);
         DIP : in STD_LOGIC_VECTOR (3 downto 0);
         EN : in STD_ULOGIC;
         SSR : in STD_ULOGIC;
         WE : in STD_ULOGIC);
end component;
```

Genaral template of BRAM instantiation (2)

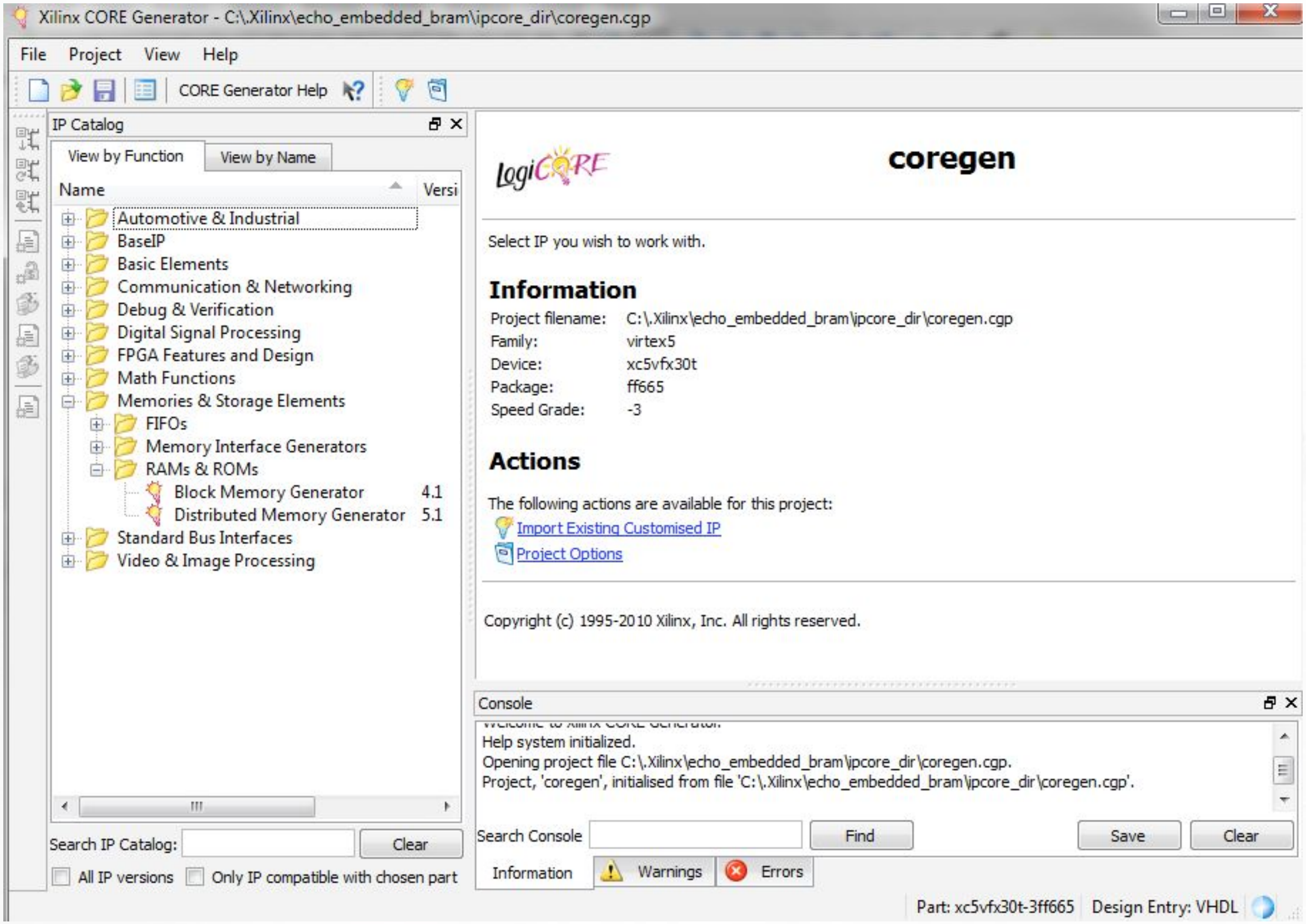


```
-- Component Attribute Specification for RAMB16_{S9 | S18 | S36}
-- Component Instantiation for RAMB16_{S9 | S18 | S36}
-- Should be placed in architecture after the begin keyword
RAMB16_{S9 | S18 | S36}_INSTANCE_NAME : RAMB16_S1
-- synthesis translate_off
generic map (
       INIT => bit_value,
       INIT_00 => vector_value,
       . . . . . . . . . .
       INIT_3F => vector_value,
       INITP_00 => vector_value,
       ................
       INITP_07 => vector_value
       SRVAL => bit_value,
       WRITE_MODE => user_WRITE_MODE)
-- synopsys translate_on
port map (    DO => user_DO,
             DOP => user_DOP,
             ADDR => user_ADDR,
             CLK => user_CLK,
             DI => user_DI,
             DIP => user_DIP,
             EN => user_EN,
             SSR => user_SSR,
             WE => user_WE);
```



Using CORE Generator

CORE Generator



Select IP you wish to work with.

Information

Project filename:	C:\.Xilinx\echo_embedded_bram\ipcore_dir\coregen.cgp
Family:	virtex5
Device:	xc5vfx30t
Package:	ff665
Speed Grade:	-3

Actions

The following actions are available for this project:

Import Existing Customised IP

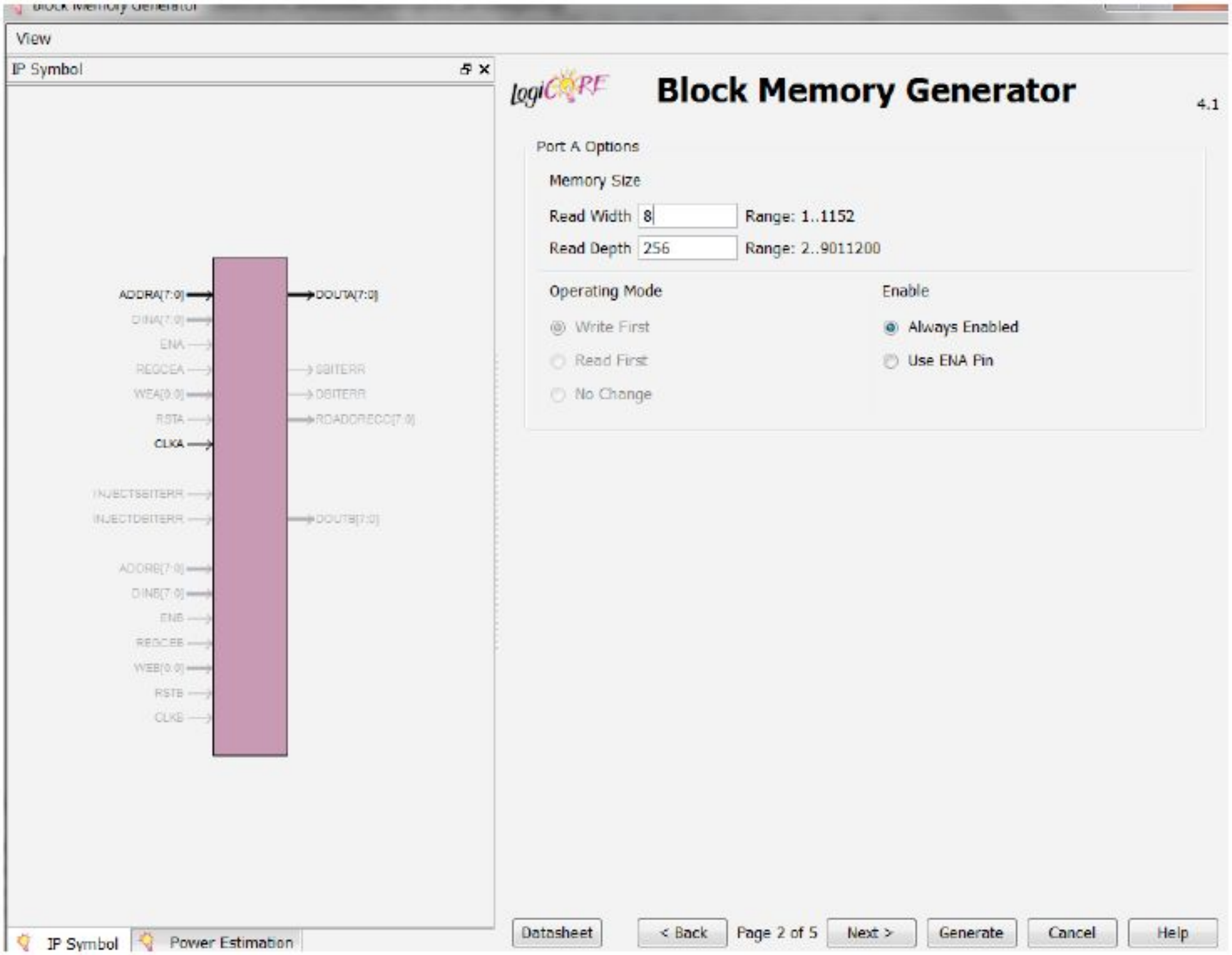
Project Options

Copyright (c) 1995-2010 Xilinx, Inc. All rights reserved.

```
Welcome to Xilinx CORE Generator.
Help system initialized.
Opening project file C:\.Xilinx\echo_embedded_bram\ipcore_dir\coregen.cgp.
Project, 'coregen', initialised from file 'C:\.Xilinx\echo_embedded_bram\ipcore_dir\coregen.cgp'.
```

Part: xc5vfx30t-3ff665 Design Entry: VHDL

CORE Generator



Block Memory Generator 4.1

Port A Options

Memory Size

Read Width: 8 — Range: 1..1152

Read Depth: 256 — Range: 2..9011200

Operating Mode

- [✓ Write First]
- Read First
- No Change

Enable

- [✓ Always Enabled]
- Use ENA Pin