Memory Usage Guide for iCE40 Devices

Technical Note

FPGA-TN-02002-1.7

September 2020

Disclaimers

Lattice makes no warranty, representation, or guarantee regarding the accuracy of information contained in this document or the suitability of its products for any particular purpose. All information herein is provided AS IS and with all faults, and all risk associated with such information is entirely with Buyer. Buyer shall not rely on any data and performance specifications or parameters provided herein. Products sold by Lattice have been subject to limited testing and it is the Buyer's responsibility to independently determine the suitability of any products and to test and verify the same. No Lattice products should be used in conjunction with mission- or safety-critical or any other application in which the failure of Lattice's product could create a situation where personal injury, death, severe property or environmental damage may occur. The information provided in this document is proprietary to Lattice Semiconductor, and Lattice reserves the right to make any changes to the information in this document or to any products at any time without notice.

Contents

Figures

Tables

Acronyms in This Document

A list of acronyms used in this document.

Acronym	Definition
EBR	Embedded Block RAM
FIFO	First In First Out
FPGA	Field-Programmable Gate Array
HDL	Hardware Description Language
RAM	Random-Access Memory
ROM	Read-Only Memory
VHDL	Verilog Hardware Description Language

1. Introduction

This technical note discusses memory usage for the iCE40™ device family (iCE40 LP/HX, iCE40 LM, iCE40 Ultra™, iCE40 UltraLite™, iCE40 UltraPlus™). It is intended to be used as a guide to the high-speed synchronous RAM Blocks and the iCE40 sysMEM™ Embedded Block RAM (EBR). The EBR is the embedded block RAM of the device, each 4 Kbit in size.

The iCE40 device architecture provides resources for memory-intensive applications. Single-Port RAM, Dual-Port RAM and FIFO can be constructed using the EBRs. The EBRs can be utilized by instantiating software primitives as described later in this document. Apart from primitive instantiation, the iCEcube2™ design software infers generic codes as EBRs.

2. Memories in iCE40 Devices

iCE40 devices contain an array of EBRs.

Figure 2.1 shows the placement of EBRs in a typical iCE40 device (does not represent true numbers of design elements).

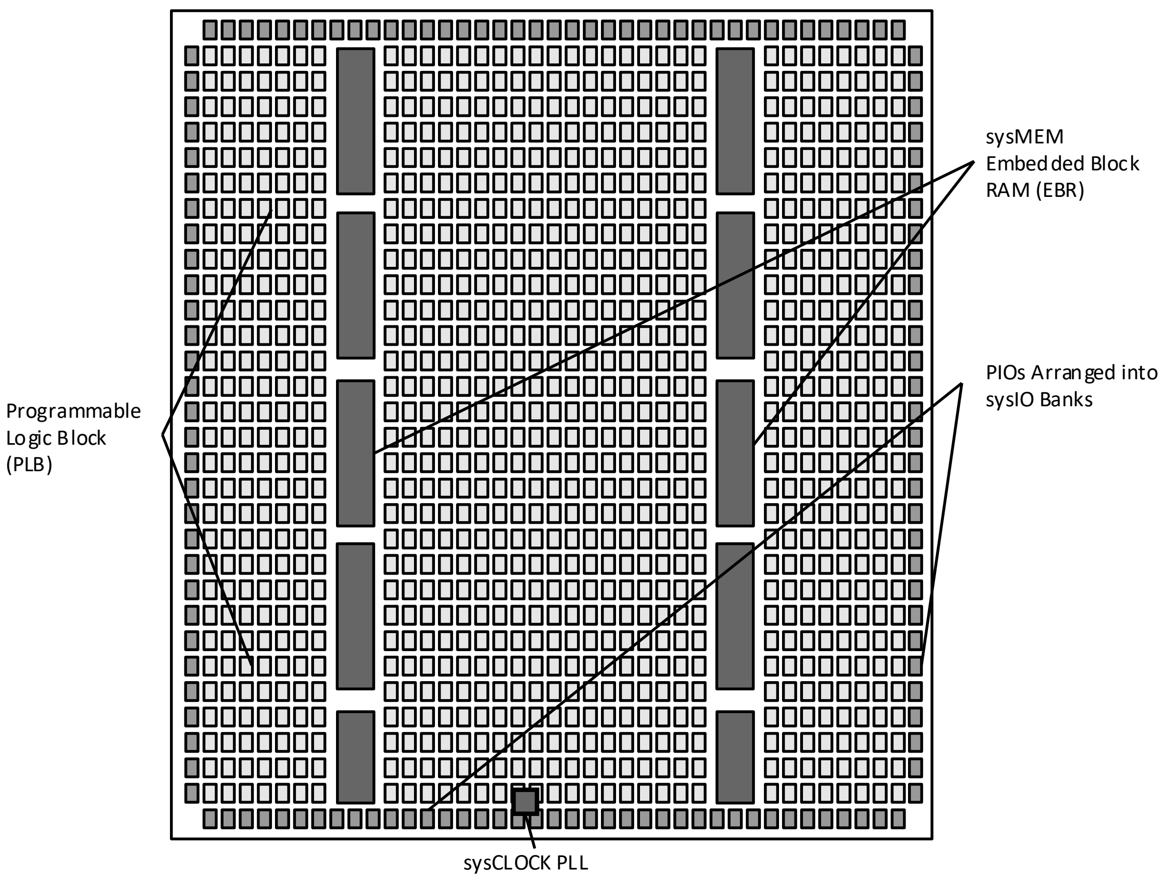

Figure 2.1. Typical Layout of an iCE40 Device

3. iCE40 sysMEM Embedded Block RAM

Each iCE40 device includes multiple high-speed synchronous EBRs, each 4Kbit in size. A single iCE40 device integrates between eight and 32 such blocks. Each EBR is a 256-word deep by 16-bit wide, two-port register file, as illustrated in Figure 3.1. The input and output connections to and from an EBR feed into the programmable interconnect resources.

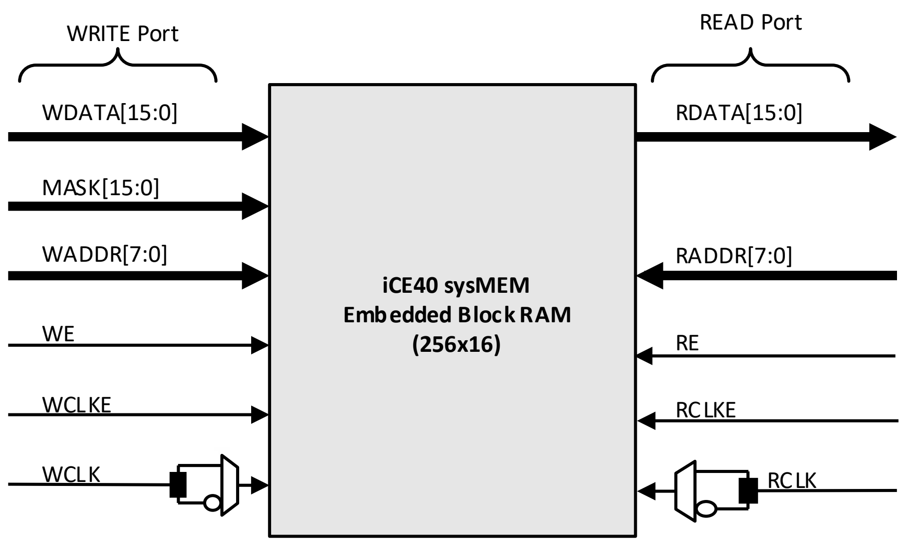

Figure 3.1. sysMEM Embedded Block RAM

Using programmable logic resources, an EBR implements a variety of logic functions, each with configurable input and output data widths.

- Random-access memory (RAM)
  - Single-port RAM with a common address, enable, and clock control lines
  - Two-port RAM with separate read and write control lines, address inputs, and enable
- Register file and scratchpad RAM
- First-In, First-Out (FIFO) memory for data buffering applications
- 256-deep by 16-wide ROM with registered outputs; contents loaded during configuration
- Counters, sequencers

As shown in Figure 3.1, an EBR has separate write and read ports, each with independent control signals. Table 3.1 lists the signals for both ports. Additionally, the write port has an active-low bit-line write-enable control; optionally mask write operations on individual bits. By default, input and output data is 16 bits wide, although the data width is configurable using programmable logic and, if needed, multiple EBRs.

The WCLK and RCLK inputs optionally connect to one of the following clock sources:

- The output from any one of the eight Global Buffers, or
- A connection from the general-purpose interconnect fabric

3.1. Signals

Table 3.1 lists the signal names, direction, and function of each connection to the EBR block.

Table 3.1. EBR Signal Descriptions

Signal Name	Direction	Description
WDATA[15:0]	Input	Write Data input.
MASK[15:0]	Input	Masks write operations for individual data bit-lines. 0 = write bit; 1 = don't write bit
WADDR[7:0]	Input	Write Address input. Selects one of 256 possible RAM locations.
WE	Input	Write Enable input.
WCLKE	Input	Write Clock Enable input.
WCLK	Input	Write Clock input. Default rising-edge, but with falling-edge option.
RDATA[15:0]	Output	Read Data output.
RADDR[7:0]	Input	Read Address input. Selects one of 256 possible RAM locations.
RE*	Input	Read Enable input. Only available for SB_RAM256x16 configurations.
RCLKE	Input	Read Clock Enable input.
RCLK	Input	Read Clock input. Default rising-edge, but with falling-edge option.

***Note**: Read Enable (RE) is available only for SB_RAM256x16 configuration. For other configurations (x2/ x4/ x8), RDATA output of SB_RAM40_4K can change, even if RE is active low. To use the RE functionality for other configurations, you can gate the Read Address RADDR with the RE signal when generating the ADDR signals.

3.2. Timing Diagram

Figure 3.2 shows the timing diagram for the EBR memory module.

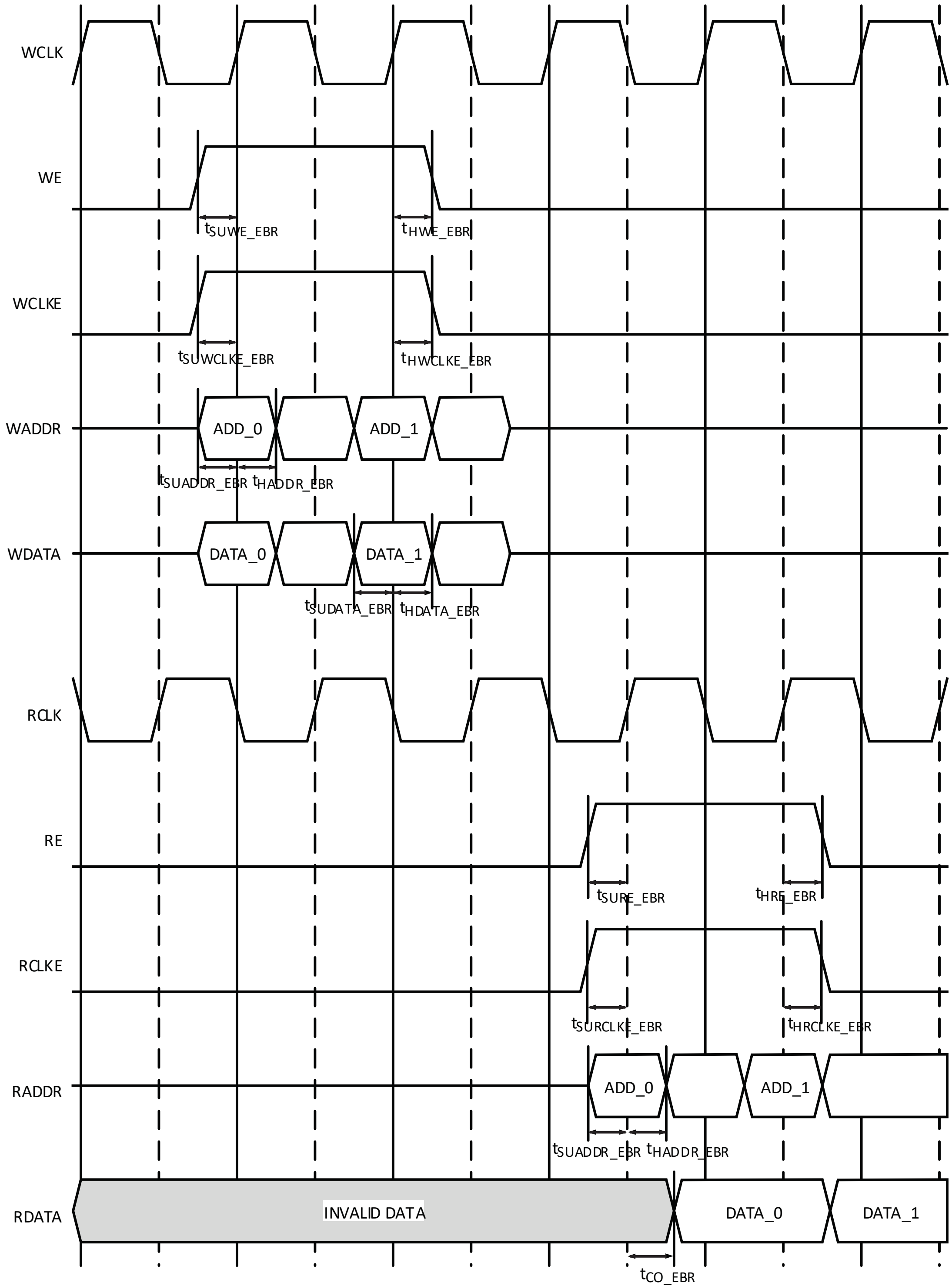

Figure 3.2. EBR Module Timing Diagram

Note: Internal timing values are considered in the iCEcube2 software's place and route. WE and WCLKE have to be valid for the address to be clocked in at the clock edge.

3.3. Memory Initialization

sysMEM memories can be initialized as needed. The initialization can be achieved through HDL (Verilog or VHDL) by specifying the initial values or through an initialization file (.mem file).

Refer to the iCEcube2 User Guide (under the Help menu) for more information on initializing memories. The Initializing Inferred RAM section covers the process of initializing memory by providing initial values or using mem file. The Memory Initializer section provides the DOS commands that can be used to initialize various memories using .mem files.

3.4. Write Operations

By default, all EBR write operations are synchronized to the rising edge of WCLK although the clock is invertible as shown in Figure 3.1. When the WCLKE signal is low, the clock to the EBR block is disabled, keeping the EBR in its lowest power mode.

To write data into the EBR block, perform the following operations:

- Supply a valid address on the WADDR[7:0] address input port
- Supply valid data on the WDATA[15:0] data input port
- To write or mask selected data bits, set the associated MASK input port accordingly. For example, write operations on data bit D[i] are controlled by the associated MASK[i] input.
    - MASK[i] = 0: Write operations are enabled for data line WDATA[i]
    - MASK[i] = 1: Mask write operations are disabled for data line WDATA[i]
- Enable the EBR write port (WE = 1)
- Enable the EBR write clock (WCLKE = 1)
- Apply a rising clock edge on WCLK (assuming that the clock is not inverted)

3.5. Read Operations

By default, all EBR read operations are synchronized to the rising edge of RCLK although the clock is invertible as shown in Figure 3.1.

To read data from the EBR block, perform the following operations:

- Supply a valid address on the RADDR[7:0] address input port
- Enable the EBR read port (RE = 1)
- Enable the EBR read clock (RCLKE = 1)
- Apply a rising clock edge on RCLK

After the clock edge, the EBR contents located at the specified address (RADDR) appear on the RDATA output port

3.6. EBR Considerations

3.6.1. Read Data Register Undefined Immediately After Configuration

Unlike the flip-flops in the Programmable Logic Blocks and Programmable I/O pins, the RDATA port is not automatically reset after configuration. Consequently, immediately following configuration and before the first valid Read Data operation, the initial RDATA read value is undefined.

3.6.2. Pre-loading EBR Data

The data contents for an EBR block can be optionally pre-loaded during iCE40 configuration. If not pre-loaded during configuration, then the EBR contents must be initialized by the iCE40 application before the EBR contents are valid. EBR initialization data can be done in the RTL code. Pre-loading the EBR data in the configuration bitstream increases the size of the configuration image accordingly.

3.6.3. EBR Contents Preserved During Configuration

EBR contents are preserved (write protected) during configuration, assuming that voltage supplies are maintained throughout. Consequently, data can be passed between multiple iCE40 configurations by leaving it in an EBR block and then skipping pre-loading during the subsequent reconfiguration.

3.6.4. Low-Power Setting

To place an EBR block in its lowest power mode, keep WCLKE = 0 and RCLKE = 0. In other words, when not actively using an EBR block, disable the clock inputs.

4. iCE40 sysMEM Embedded Block RAM Memory Primitives

This section lists the iCE40 sysMEM EBR software primitives that can be instantiated in the RTL. Different EBRs are used in different configurations. Each EBR has separate write and read ports, each with independent control signals. Each EBR can be configured into a RAM block of size 256x16, 512x8, 1024x4 or 2048x2. The data contents of the EBR can optionally be pre-loaded during iCE40 device configuration by specifying the initialization data in the primitive instantiation. Table 4.3 and Table 4.4 shows how the address and data connection are mapped according to RAM configuration.

Table 4.1 lists the supported dual port synchronous RAM configurations, each of 4 Kbits in size. The RAM blocks can be directly instantiated in the top module and taken through the iCEcube2 software flow.

Table 4.1. EBR Configurations and Primitive Names

Block RAM Configuration	Block RAM Configuration and Size	WADDR Port Size (Bits)	WDATA Port Size (Bits)	RADDR Port Size (Bits)	RDATA Port Size (Bits)	MASK Port Size (Bits)
SB_RAM256x16 SB_RAM256x16NR SB_RAM256x16NW SB_RAM256x16NRNW	256 x 16 (4K)	8 [7:0]	16 [15:0]	8 [7:0]	16 [15:0]	16 [15:0]
SB_RAM512x8 SB_RAM512x8NR SB_RAM512x8NW SB_RAM512x8NRNW	512 x 8 (4K)	9 [8:0]	8 [7:0]	9 [8:0]	8 [7:0]	No Mask Port
SB_RAM1024x4 SB_RAM1024x4NR SB_RAM1024x4NW SB_RAM1024x4NRNW	1024 x 4 (4K)	10 [9:0]	4 [3:0]	10 [9:0]	4 [3:0]	No Mask Port
SB_RAM2048x2 SB_RAM2048x2NR SB_RAM2048x2NW SB_RAM2048x2NRNW	2048 x 2 (4K)	11 [10:0]	2 [1:0]	11 [10:0]	2 [1:0]	No Mask Port

For iCE40 EBR primitives with a negative-edged read or write clock, the base primitive name is appended with a 'N' and a 'R' or 'W' depending on the clock that is affected (see Table 4.2 for the 256x16 RAM block configuration).

Table 4.2. Naming Convention for RAM Primitives

RAM Primitive Name	Description
SB_RAM256x16	Positive-edged read clock, positive-edged write clock
SB_RAM256x16NR	Negative-edged read clock, positive-edged write clock
SB_RAM256x16NW	Positive-edged read clock, negative-edged write clock
SB_RAM256x16NRNW	Negative-edged read clock, negative-edged write clock

Table 4.3. Address and Data Mapping for the Initialize RAM Configuration (Write Data)

Mode	WADDR	Configuration	Data Connection
0	NA	256 x 16	15,14,13,12,11,10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0
1		512 x 8	14,14,12,12,10,10, 8, 8, 6, 6, 4, 4, 2, 2, 0, 0
2		1024 x 4	13,13,13,13, 9, 9, 9, 9, 5, 5, 5, 5, 1, 1, 1, 1
3		2048 x 2	11,11,11,11,11,11,11,11, 3, 3, 3, 3, 3, 3, 3, 3

Table 4.4. Address and Data Mapping for the Initialize RAM Configuration (Read Data)

Mode	RADDR[10:8]	Configuration	Data Connection
0	xxx	256x16	15,14,13,12,11,10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0
1	xx0	512x8	14,12,10, 8, 6, 4, 2, 0
	xx1		15,13,11, 9, 7, 5, 3, 1
2	x00	1024x4	12, 8, 4, 0
	x01		13, 9, 5, 1
	x10		14,10, 6, 2
	x11		15,11, 7, 3
3	000	2048x2	8, 0
	001		9, 1
	010		10, 2
	011		11, 3
	100		12, 4
	101		13, 5
	110		14, 6
	111		15, 7

4.1. SB_RAM256x16

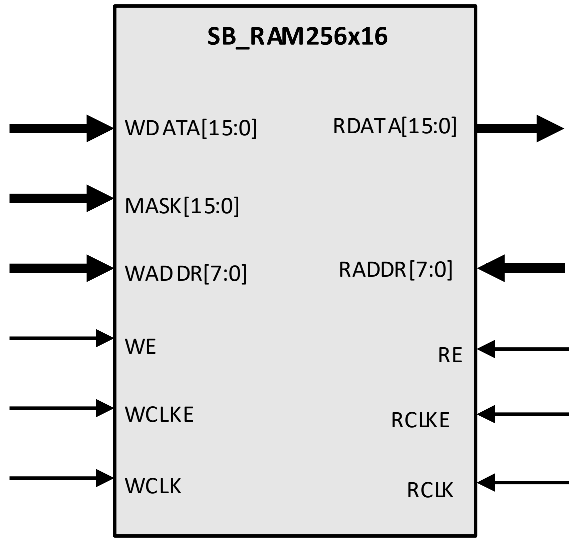

Figure 4.1. SB_RAM256x16 Primitive

Verilog Instantiation

```
SB_RAM256x16 ram256x16_inst (
    .RDATA(RDATA_c[15:0]),
    .RADDR(RADDR_c[7:0]),
    .RCLK(RCLK_c),
    .RCLKE(RCLKE_c),
    .RE(RE_c),
    .WADDR(WADDR_c[7:0]),
    .WCLK(WCLK_c),
    .WCLKE(WCLKE_c),
    .WDATA(WDATA_c[15:0]),
    .WE(WE_c),
    .MASK(MASK_c[15:0])
);

defparam ram256x16_inst.INIT_0 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram256x16_inst.INIT_1 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram256x16_inst.INIT_2 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram256x16_inst.INIT_3 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram256x16_inst.INIT_4 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram256x16_inst.INIT_5 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram256x16_inst.INIT_6 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
```

VHDL Instantiation

```
ram256x16_inst : SB_RAM256x16  generic map (
INIT_0 => X"0000000000000000000000000000000000000000000000000000000000000000",   INIT_1 =>
X"0000000000000000000000000000000000000000000000000000000000000000",   INIT_2 =>
X"0000000000000000000000000000000000000000000000000000000000000000",   INIT_3 =>
X"0000000000000000000000000000000000000000000000000000000000000000",   INIT_4 =>
X"0000000000000000000000000000000000000000000000000000000000000000",   INIT_5 =>
X"0000000000000000000000000000000000000000000000000000000000000000",   INIT_6 =>
X"0000000000000000000000000000000000000000000000000000000000000000",   INIT_7 =>
X"0000000000000000000000000000000000000000000000000000000000000000",   INIT_8 =>
X"0000000000000000000000000000000000000000000000000000000000000000",   INIT_9 =>
X"0000000000000000000000000000000000000000000000000000000000000000",   INIT_A =>
X"0000000000000000000000000000000000000000000000000000000000000000",   INIT_B =>
X"0000000000000000000000000000000000000000000000000000000000000000",   INIT_C =>
X"0000000000000000000000000000000000000000000000000000000000000000",   INIT_D =>
X"0000000000000000000000000000000000000000000000000000000000000000",   INIT_E =>
X"0000000000000000000000000000000000000000000000000000000000000000",   INIT_F =>
X"0000000000000000000000000000000000000000000000000000000000000000"
)

port map (
    RDATA => RDATA_c,
    RADDR => RADDR_c,
    RCLK => RCLK_c,
    RCLKE => RCLKE_c,
    RE => RE_c,
    WADDR => WADDR_c,
    WCLK=> WCLK_c,
    WCLKE => WCLKE_c,
    WDATA => WDATA_c,
    MASK => MASK_c,
    WE => WE_c
);
```

Table 4.5 is a complete list of SB_RAM256x16 based primitives.

Table 4.5. SB_RAM256x16 Based Primitives

Primitive	Description
SB_RAM256x16	SB_RAM256x16 //Positive edged clock RCLK WCLK (RDATA, RCLK, RCLKE, RE, RADDR, WCLK, WCLKE, WE, WADDR, MASK, WDATA);
SB_RAM256x16NR	SB_RAM256x16NR // Negative edged Read Clock – i.e. RCLKN (RDATA, RCLKN, RCLKE, RE, RADDR, WCLK, WCLKE, WE, WADDR, MASK, WDATA);
SB_RAM256x16NW	SB_RAM256x16NW // Negative edged Write Clock – i.e. WCLKN (RDATA, RCLK, RCLKE, RE, RADDR, WCLKN, WCLKE, WE, WADDR, MASK, WDATA);
SB_RAM256x16NRNW	SB_RAM256x16NRNW // Negative edged Read and Write – i.e. RCLKN WRCKLN (RDATA, RCLKN, RCLKE, RE, RADDR, WCLKN, WCLKE, WE, WADDR, MASK, WDATA);

4.2. SB_RAM512x8

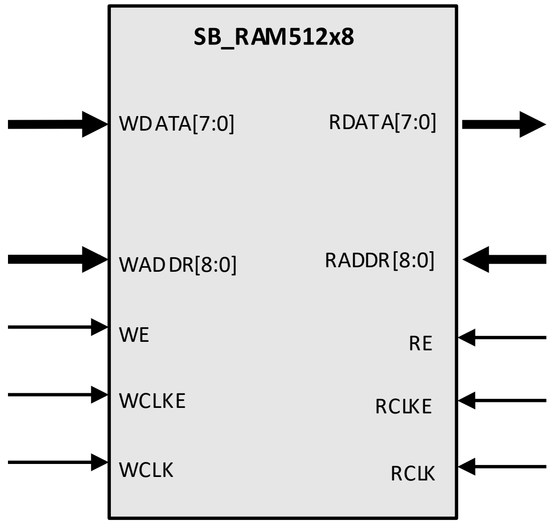

Figure 4.2. SB_RAM512x8 Primitive

Verilog Instantiation

```
SB_RAM512x8 ram512X8_inst (
    .RDATA(RDATA_c[7:0]),
    .RADDR(RADDR_c[8:0]),
    .RCLK(RCLK_c),
    .RCLKE(RCLKE_c),
    .RE(RE_c),
    .WADDR(WADDR_c[8:0]),
    .WCLK(WCLK_c),
    .WCLKE(WCLKE_c),
    .WDATA(WDATA_c[7:0]),
    .WE(WE_c)
);

defparam ram512x8_inst.INIT_0 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram512x8_inst.INIT_1 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram512x8_inst.INIT_2 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram512x8_inst.INIT_3 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram512x8_inst.INIT_4 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram512x8_inst.INIT_5 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram512x8_inst.INIT_6 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram512x8_inst.INIT_7 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram512x8_inst.INIT_8 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram512x8_inst.INIT_9 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram512x8_inst.INIT_A =
256'h0000000000000000000000000000000000000000000000000000000000000000;
```

```
defparam ram512x8_inst.INIT_B =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram512x8_inst.INIT_C =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram512x8_inst.INIT_D =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram512x8_inst.INIT_E =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram512x8_inst.INIT_F =
256'h0000000000000000000000000000000000000000000000000000000000000000;
```

VHDL Instantiation

```
ram512x8_inst : SB_RAM512x8  generic map (
INIT_0 => X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_1 =>
X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_2 =>
X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_3 =>
X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_4 =>
X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_5 =>
X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_6 =>
X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_7 =>
X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_8 =>
X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_9 =>
X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_A =>
X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_B =>
X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_C =>
X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_D =>
X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_E =>
X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_F =>
X"0000000000000000000000000000000000000000000000000000000000000000"
)

port map (
    RDATA => RDATA_c,
    RADDR => RADDR_c,
    RCLK => RCLK_c,
    RCLKE => RCLKE_c,
    RE => RE_c,
    WADDR => WADDR_c,
    WCLK=> WCLK_c,
    WCLKE => WCLKE_c,
    WDATA => WDATA_c,
    WE => WE_c
);
WE => WE_c
);
```

Table 4.6 is a complete list of SB_RAM512x8 based primitives.

Table 4.6. SB_RAM512x8 Based Primitives

Primitive	Description
SB_RAM512x8	SB_RAM512x8 //Positive edged clock RCLK WCLK (RDATA, RCLK, RCLKE, RE, RADDR, WCLK, WCLKE, WE, WADDR, MASK, WDATA);
SB_RAM512x8NR	SB_RAM512x8NR // Negative edged Read Clock – i.e. RCLKN (RDATA, RCLKN, RCLKE, RE, RADDR, WCLK, WCLKE, WE, WADDR, MASK, WDATA);
SB_RAM512x8NW	SB_RAM512x8NW // Negative edged Write Clock – i.e. WCLKN (RDATA, RCLK, RCLKE, RE, RADDR, WCLKN, WCLKE, WE, WADDR, MASK, WDATA);
SB_RAM512x8NRNW	SB_RAM512x8NRNW // Negative edged Read and Write – i.e. RCLKN WRCKLN (RDATA, RCLKN, RCLKE, RE, RADDR, WCLKN, WCLKE, WE, WADDR, MASK, WDATA);

4.3. SB_RAM1024x4

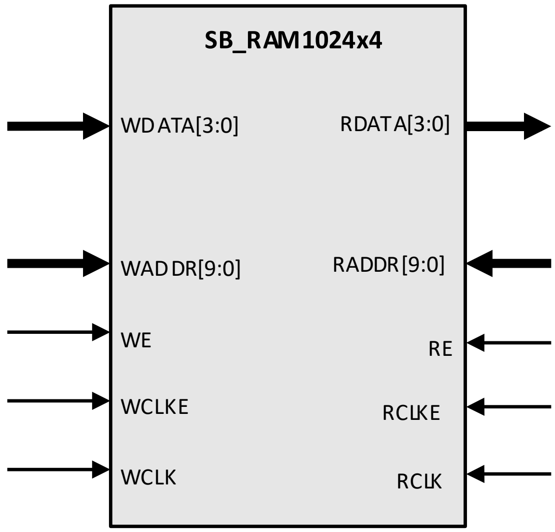

Figure 4.3. SB_RAM1024x4 Primitive

Verilog Instantiation

```
SB_RAM1024x4 ram1024x4_inst (
    .RDATA(RDATA_c[3:0]),
    .RADDR(RADDR_c[9:0]),
    .RCLK(RCLK_c),
    .RCLKE(RCLKE_c),
    .RE(RE_c),
    .WADDR(WADDR_c[9:0]),
    .WCLK(WCLK_c),
    .WCLKE(WCLKE_c),
    .WDATA(WDATA_c[3:0]),
    .WE(WE_c)
);

defparam ram1024x4_inst.INIT_0 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram1024x4_inst.INIT_1 =
256'h000000000000000000000000000000000000000000000000000000000000000000;
defparam ram1024x4_inst.INIT_2 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram1024x4_inst.INIT_3 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram1024x4_inst.INIT_4 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram1024x4_inst.INIT_5 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram1024x4_inst.INIT_6 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram1024x4_inst.INIT_7 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram1024x4_inst.INIT_8 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram1024x4_inst.INIT_9 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram1024x4_inst.INIT_A =
256'h0000000000000000000000000000000000000000000000000000000000000000;
```

```
defparam ram1024x4_inst.INIT_B =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram1024x4_inst.INIT_C =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram1024x4_inst.INIT_D =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram1024x4_inst.INIT_E =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram1024x4_inst.INIT_F =
256'h0000000000000000000000000000000000000000000000000000000000000000;
```

VHDL Instantiation

```
Ram1024X4_inst : SB_RAM1024x4  generic map (
INIT_0 => X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_1 =>
X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_2 =>
X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_3 =>
X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_4 =>
X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_5 =>
X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_6 =>
X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_7 =>
X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_8 =>
X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_9 =>
X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_A =>
X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_B =>
X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_C =>
X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_D =>
X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_E =>
X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_F =>
X"0000000000000000000000000000000000000000000000000000000000000000"
)

port map (
    RDATA => RDATA_c,
    RADDR => RADDR_c,
    RCLK => RCLK_c,
    RCLKE => RCLKE_c,
    RE => RE_c,
    WADDR => WADDR_c,
    WCLK=> WCLK_c,
    WCLKE => WCLKE_c,
    WDATA => WDATA_c,
    WE => WE_c
);
```

Table 4.7 is a complete list of SB_RAM1024x4 based primitives.

Table 4.7. SB_RAM1024x4 Based Primitives

Primitive	Description
SB_RAM1024x4	SB_RAM1024x4 //Positive edged clock RCLK WCLK (RDATA, RCLK, RCLKE, RE, RADDR, WCLK, WCLKE, WE, WADDR, MASK, WDATA);
SB_RAM1024x4NR	SB_RAM1024x4NR // Negative edged Read Clock – i.e. RCLKN (RDATA, RCLKN, RCLKE, RE, RADDR, WCLK, WCLKE, WE, WADDR, MASK, WDATA);
SB_RAM1024x4NW	SB_RAM1024x4NW // Negative edged Write Clock – i.e. WCLKN (RDATA, RCLK, RCLKE, RE, RADDR, WCLKN, WCLKE, WE, WADDR, MASK, WDATA);
SB_RAM1024x4NRNW	SB_RAM1024x4NRNW // Negative edged Read and Write – i.e. RCLKN WRCKLN (RDATA, RCLKN, RCLKE, RE, RADDR, WCLKN, WCLKE, WE, WADDR, MASK, WDATA);

4.4. SB_RAM2048x2

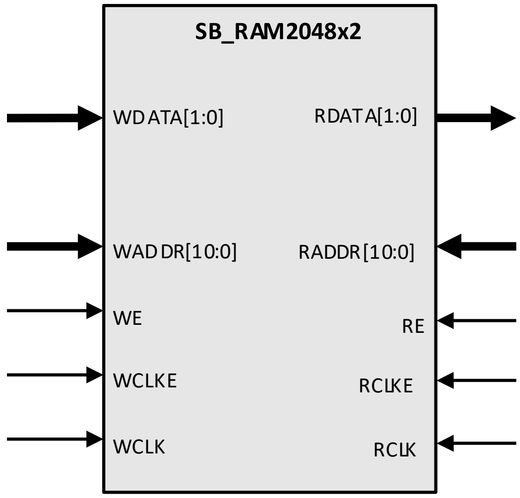

Figure 4.4. SB_RAM2048x2

Verilog Instantiation

```
SB_RAM2048x2 ram2048x2_inst (
    .RDATA(RDATA_c[1:0]),
    .RADDR(RADDR_c[10:0]),
    .RCLK(RCLK_c),
    .RCLKE(RCLKE_c),
    .RE(RE_c),
    .WADDR(WADDR_c[1:0]),
    .WCLK(WCLK_c),
    .WCLKE(WCLKE_c),
    .WDATA(WDATA_c[10:0]),
    .WE(WE_c)
);

defparam ram2048x2_inst.INIT_0 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram2048x2_inst .INIT_1 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram2048x2_inst .INIT_2 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram2048x2_inst .INIT_3 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram2048x2_inst .INIT_4 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram2048x2_inst .INIT_5 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram2048x2_inst .INIT_6 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram2048x2_inst .INIT_7 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram2048x2_inst .INIT_8 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram2048x2_inst .INIT_9 =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram2048x2_inst .INIT_A =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram2048x2_inst .INIT_B =
256'h0000000000000000000000000000000000000000000000000000000000000000;
```

```
defparam ram2048x2_inst .INIT_C =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram2048x2_inst .INIT_D =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram2048x2_inst .INIT_E =
256'h0000000000000000000000000000000000000000000000000000000000000000;
defparam ram2048x2_inst .INIT_F =
256'h0000000000000000000000000000000000000000000000000000000000000000;
```

VHDL Instantiation

```
Ram2048x2_inst : SB_RAM2048x2
generic map (
INIT_0 => X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_1 =>
X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_2 =>
X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_3 =>
X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_4 =>
X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_5 =>
X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_6 =>
X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_7 =>
X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_8 =>
X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_9 =>
X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_A =>
X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_B =>
X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_C =>
X"0000000000000000000000000000000000000000000000000000000000000000",
INIT_D => X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_E =>
X"0000000000000000000000000000000000000000000000000000000000000000",  INIT_F =>
X"0000000000000000000000000000000000000000000000000000000000000000"
)

port map (
    RDATA => RDATA_c,
    RADDR => RADDR_c,
    RCLK => RCLK_c,
    RCLKE => RCLKE_c,
    RE => RE_c,
    WADDR => WADDR_c,
    WCLK=> WCLK_c,
    WCLKE => WCLKE_c,
    WDATA => WDATA_c,
    WE => WE_c
);
```

Table 4.8 is a complete list of the SB_RAM2048x2 based primitives.

Table 4.8. SB_RAM2048x2 Based Primitives

Primitive	Description
SB_RAM2048x2	SB_RAM2048x2 //Positive edged clock RCLK WCLK (RDATA, RCLK, RCLKE, RE, RADDR, WCLK, WCLKE, WE, WADDR, MASK, WDATA);
SB_RAM2048x2NR	SB_RAM2048x2NR // Negative edged Read Clock – i.e. RCLKN (RDATA, RCLKN, RCLKE, RE, RADDR, WCLK, WCLKE, WE, WADDR, MASK, WDATA);
SB_RAM2048x2NW	SB_RAM2048x2NW // Negative edged Write Clock – i.e. WCLKN (RDATA, RCLK, RCLKE, RE, RADDR, WCLKN, WCLKE, WE, WADDR, MASK, WDATA);
SB_RAM2048x2NRNW	SB_RAM2048x2NRNW // Negative edged Read and Write – i.e. RCLKN WRCKLN (RDATA, RCLKN, RCLKE, RE, RADDR, WCLKN, WCLKE, WE, WADDR, MASK, WDATA);

4.5. SB_RAM40_4K

SB_RAM40_4K is the basic physical RAM primitive which can be instantiated and configured to different depths and data ports. The SB_RAM40_4K block has a size of 4 Kbits with separate write and read ports, each with independent control signals. By default, input and output data is 16 bits wide, although the data width is configurable using the READ_MODE and WRITE_MODE parameters. The data contents of the SB_RAM40_4K block are optionally pre-loaded during iCE device configuration.

Table 4.9. SB_RAM40_4K Naming Convention Rules

RAM Primitive Name	Description
SB_RAM40_4K	Positive-edged read clock, positive-edged write clock
SB_RAM40_4KNR	Negative-edged read clock, positive-edged write clock
SB_RAM40_4KNW	Positive-edged read clock, negative-edged write clock
SB_RAM40_4KNRNW	Negative-edged clock, negative-edged write clock

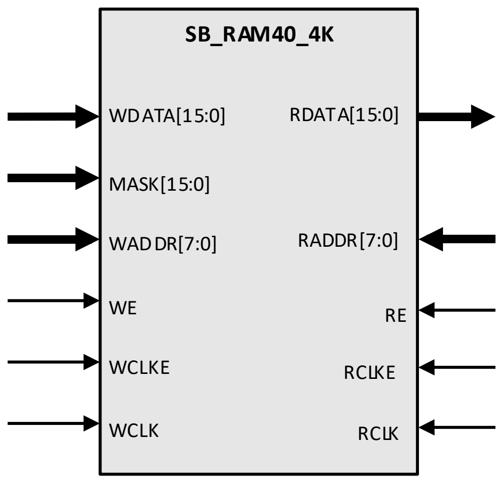

Figure 4.5. SB_RAM40_4K

Table 4.10 lists the signals for both ports.

Table 4.10. SB_RAM40_4K Signal Descriptions

Signal Name	Direction	Description
WDATA[15:0]	Input	Write Data input.
MASK[15:0]	Input	Bit-line Write Enable input, active low. Applicable only when WRITE_MODE parameter is set to '0'.
WADDR[7:0]	Input	Write Address input. Selects up to 256 possible locations.
WE	Input	Write Enable input, active high.
WCLK	Input	Write Clock input, rising-edge active.
WCLKE	Input	Write Clock Enable input.
RDATA[15:0]	Output	Read Data output.
RADDR[7:0]	Input	Read Address input. Selects one of 256 possible locations.
RE	Input	Read Enable input, active high.
RCLK	Input	Read Clock input, rising-edge active.
RCLKE	Input	Read Clock Enable input.

Table 4.11 on the next page describes the parameter values to infer the desired RAM configuration.

Table 4.11. SB_RAM40_4K Primitive Parameter Descriptions

Parameter Name	Description	Parameter	Configuration
INIT_0, … …,INIT_F	RAM Initialization Data. Passed using 16 parameter strings, each comprising 256 bits. (16 x 256=4096 total bits)	INIT_0 to INIT_F	Initialize the RAM with predefined value
WRITE_MODE	Sets the RAM block write port configuration	0	256 x 16
		1	512 x 18
		2	1024 x 4
		3	2048 x 2
READ_MODE		0	256 x 16
		1	512 x 8
		2	1024 x 4
		3	2048 x 2

Verilog Instantiation

```
// Physical RAM Instance without Pre Initialization
SB_RAM40_4K ram40_4kinst_physical (
    .RDATA(RDATA),
    .RADDR(RADDR),
    .WADDR(WADDR),
    .MASK(MASK),
    .WDATA(WDATA)
    .RCLKE(RCLKE),
    .RCLK(RCLK),
    .RE(RE),
    .WCLKE(WCLKE),
    .WCLK(WCLK),
    .WE(WE)
);

defparam ram40_4kinst_physical.READ_MODE=0;
defparam ram40_4kinst_physical.WRITE_MODE=0;
```

VHDL Instantiation

```
-- Physical RAM Instance without Pre Initialization
ram40_4kinst_physical : SB_RAM40_4K
generic map (
    READ_MODE => 0,
    WRITE_MODE= >0
)

port map (
    RDATA=>RDATA,
    RADDR=>RADDR,
    WADDR=>WADDR,
    MASK=>MASK,
    WDATA=>WDATA,
    RCLKE=>RCLKE,
    RCLK=>RCLK,
    RE=>RE,
    WCLKE=>WCLKE,
    WCLK=>WCLK,
    WE=>WE
);
```

5. EBR Utilization Summary in iCEcube2 Design Software

The placer.log file in the iCEcube2 design software shows the device utilization summary. The Final Design Statistics and Device Utilization Summary sections show the number of EBRs (or RAMs) used against the total number. Figure 5.1 shows the EBR usage when one SB_RAM256x16 was instantiated in the design.

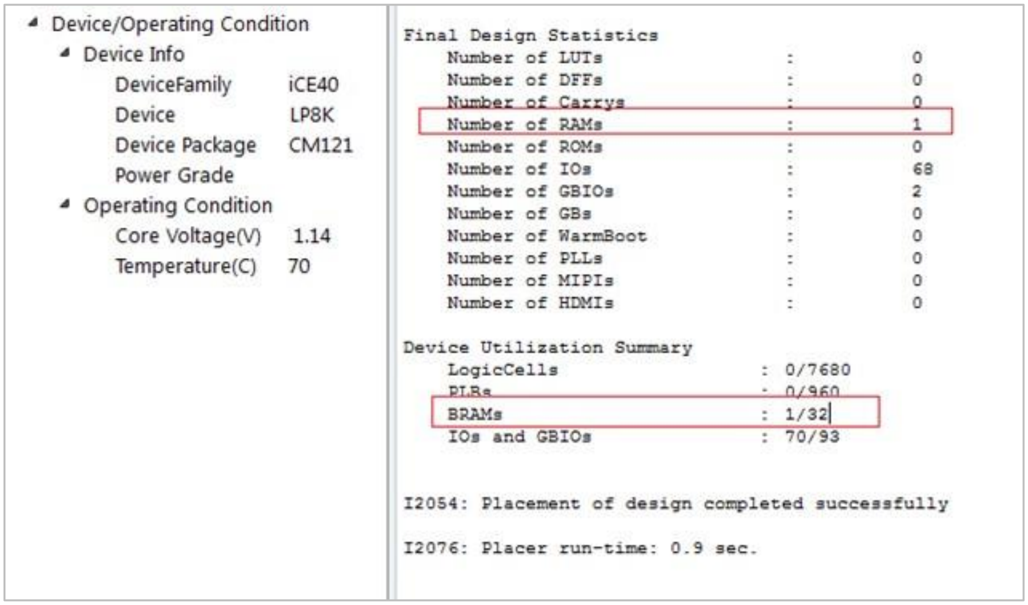

Figure 5.1. iCEcube2 Design Software Report File

Appendix A. Standard HDL Code References

This appendix contains standard HDL and VHDL codes for popular memory elements, which can be used to infer a sysMEM EBR automatically. Standard HDL coding techniques do not require you to know the details of the block RAMs of the device and are inferred automatically.

A.1. Single-Port RAM

Verilog

```
module ram (din, addr, write_en, clk, dout);// 512x8
    parameter addr_width = 9;
    parameter data_width = 8;
    input [addr_width-1:0] addr;
    input [data_width-1:0] din;
    input write_en, clk;
    output [data_width-1:0] dout;
    reg [data_width-1:0] dout; // Register for output.
    reg [data_width-1:0] mem [(1<<addr_width)-1:0];

    always @(posedge clk)
    begin
        if (write_en)
            mem[(addr)] <= din;
        dout = mem[addr]; // Output register controlled by clock.
    end
endmodule
```

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity ram is
generic (
    addr_width : natural := 9;--512x8
    data_width : natural := 8);
port (
    addr : in std_logic_vector (addr_width - 1 downto 0);
    write_en : in    std_logic;
    clk : in  std_logic;
    din : in  std_logic_vector (data_width - 1 downto 0);
    dout : out std_logic_vector (data_width - 1 downto 0));
end ram;

architecture rtl of ram is
    type mem_type is array ((2** addr_width) - 1 downto 0) of
        std_logic_vector(data_width - 1 downto 0);
    signal mem : mem_type;
begin
    process (clk)
    begin
        if (clk'event and clk = '1') then
            if (write_en = '1') then
                mem(conv_integer(addr)) <= din;
                   -- Using write address bus.
            end if;
        dout <= mem(conv_integer(addr));
    end if;
```

```
    -- Output register controlled by clock.
    end process;
end rtl;
```

A.2. Dual Port RAM

Verilog

```
module ram (din, write_en, waddr, wclk, raddr, rclk, dout);//512x8
    parameter addr_width = 9;
    parameter data_width = 8;
    input [addr_width-1:0] waddr, raddr;
    input [data_width-1:0] din;
    input write_en, wclk, rclk;
    output reg [data_width-1:0] dout;
    reg [data_width-1:0] mem [(1<<addr_width)-1:0]
        ;

    always @(posedge wclk) // Write memory.
    begin
        if (write_en)
            mem[waddr] <= din; // Using write address bus.
    end
    always @(posedge rclk) // Read memory.
    begin
        dout <= mem[raddr]; // Using read address bus.
    end
endmodule
```

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity ram is
generic (
    addr_width : natural := 9;--512x8
    data_width : natural := 8);
port (
    write_en : in    std_logic;
    waddr : in              std_logic_vector (addr_width - 1 downto 0);
    wclk : in std_logic;
    raddr : in              std_logic_vector (addr_width - 1 downto 0);
    rclk : in std_logic;
    din : in  std_logic_vector (data_width - 1 downto 0);
    dout : out std_logic_vector (data_width - 1 downto 0));
end ram;

architecture rtl of ram is
    type mem_type is array ((2** addr_width) - 1 downto 0) of
        std_logic_vector(data_width - 1 downto 0);
    signal mem : mem_type;
begin
    process (wclk)
    -- Write memory.
    begin
        if (wclk'event and wclk = '1') then
            if (write_en = '1') then
                mem(conv_integer(waddr)) <= din;
```

```
                    -- Using write address bus.
            end if;
        end if;
    end process;
    process (rclk) -- Read memory.
    begin
        if (rclk'event and rclk = '1') then
        dout <= mem(conv_integer(raddr));
            -- Using read address bus.
        end if;
    end process;
end rtl;
```

References

For more information, refer to the following documents:

- iCE40 LP/HX Family Data Sheet (FPGA-DS-02029)
- iCE40 LM Family Data Sheet (FPGA-DS-02043)
- Programming Cables User's Guide (FPGA-DS-02024)

Technical Support Assistance

Submit a technical support case via www.latticesemi.com/techsupport.

Revision History

Revision 1.7, September 2020

Section	Change Summary
Disclaimers	Added this section.
iCE40 sysMEM Embedded Block RAM	Update the note in Figure 3.2. EBR Module Timing Diagram.
iCE40 sysMEM Embedded Block RAM Memory Primitives	Added the following tables: • Table 4.3. Address and Data Mapping for the Initialize RAM Configuration (Write Data) • Table 4.4. Address and Data Mapping for the Initialize RAM Configuration (Read Data)
References	Updated document numbers of referenced data sheets and user guide.
—	Updated Revision History and back cover format.

Revision 1.6, August 2017

Section	Change Summary
All	• Changed document number from TN1250 to FPGA-TN-02002. • Updated document template.
Acronyms in This Document	Added this section.

Revision 1.5, June 2016

Section	Change Summary
Introduction	Updated Introduction section. Added iCE40 UltraPlus to introductory paragraph.
Signals	Updated Signals section. Added footnote to RE signal in Table 3.1. EBR Signal Descriptions.
Timing Diagram	Updated Timing Diagram section. Revised Figure 3.2. EBR Module Timing Diagram.
Memory Initialization	Added Memory Initialization section.
iCE40 sysMEM Embedded Block RAM Memory Primitives	• Updated SB_RAM512x8 section. Minor correction in Verilog Instantiation and VHDL Instantiation. • Updated SB_RAM1024x4 section. Minor correction in VHDL Instantiation. Revised SB_RAM512x8 Verilog Instantiation and VHDL Instantiation. • Revised SB_RAM1024x4 Verilog Instantiation and VHDL Instantiation. • Revised SB_RAM2048x2 Verilog Instantiation.
Technical Support Assistance	Updated Technical Support Assistance section.

Revision 1.4, January 2015

Section	Change Summary
All	Added support for iCE40 UltraLite.

Revision 1.3, June 2014

Section	Change Summary
All	Added support for iCE40 Ultra.

Revision 1.2, December 2013

Section	Change Summary
iCE40 sysMEM Embedded Block RAM	Added information to Table 3.1. EBR Signal Descriptions

Revision 1.1, October 2013

Section	Change Summary
All	Removed iCE40 Family EBRs table.
Technical Support Assistance	Updated Technical Support Assistance information.

Revision 1.0, September 2012

Section	Change Summary
All	Initial release.

www.latticesemi.com