CrossLink Memory Usage Guide

Technical Note

FPGA-TN-02017-1.2

October 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

Acronyms in This Document 6
1. Introduction 7
2. Memory Generation 7
2.1. Clarity Designer Flow 8
2.2. Utilizing PMI 10
2.3. Utilizing Direct Instantiation of Memory Primitives 11
3. Memory Features 12
3.1. ECC in Memory Modules 12
3.2. Byte Enable 12
4. Memory Modules 12
4.1. Memory Cascading 13
4.1.1. Input and Output Register 13
4.1.2. Reset 13
4.1.3. Timing 13
4.2. Single Port RAM (RAM_DQ) – EBR Based 13
4.3. True Dual-Port RAM (RAM_DP_TRUE) – EBR Based 19
4.4. Pseudo Dual-Port RAM (RAM_DP) – EBR Based 27
4.5. Read Only Memory (ROM) – EBR Based 32
4.6. First In First Out (FIFO) Memory 34
4.6.1. Single Clock FIFO (FIFO) – EBR and LUT 34
4.6.2. Dual Clock First-In-First-Out (FIFO_DC) – EBR or LUT Based 44
4.6.3. FIFO_DC Flags 45
4.7. Distributed Single-Port RAM (Distributed_SPRAM) – PFU-Based 54
4.8. Distributed Dual-Port RAM (4.8. Distributed_DPRAM) – PFU-Based 57
4.9. Distributed ROM (Distributed_ROM) – PFU-Based 60
5. Initializing Memory 63
5.1. Initialization File Formats 63
5.1.1. Binary File 64
5.1.2. Hex File 64
5.1.3. Addressed Hex 64
Appendix A. Attribute Definitions 66
A.1. DATA_WIDTH 66
A.2. REGMODE 66
A.3. CSDECODE 66
A.4. WRITEMODE 66
References 67
Technical Support Assistance 68
Revision History 69

Figures

Figure 2.1. Memory Modules in Clarity Designer 8
Figure 2.2. Clarity Designer in Lattice Diamond Software 8
Figure 2.3. Example: Generating Pseudo Dual Port RAM (RAM_DP) Using Clarity Designer 9
Figure 2.4. Example: Generating Pseudo Dual Port RAM (RAM_DP) Module Customization – General Options 10
Figure 4.1. Single-Port Memory Module Generated by Clarity Designer 13
Figure 4.2. Single Port RAM Timing Waveform in Normal (NORMAL) Mode without Output Registers 16
Figure 4.3. Single Port RAM Timing Waveform in Normal (NORMAL) Mode with Output Registers 16
Figure 4.4. Single Port RAM Timing Waveform in Write through (WRITETHROUGH) Mode without Output Registers 17
Figure 4.5. Single Port RAM Timing Waveform in Write through (WRITETHROUGH) Mode with Output Registers 17
Figure 4.6. Single Port RAM Timing Waveform in Read before Write (READBEFOREWRITE) Mode without Output Registers 18
Figure 4.7. Single Port RAM Timing Waveform in Read before Write (READBEFOREWRITE) Mode with Output Registers 18
Figure 4.8. True Dual-Port Memory Module Generated by Clarity Designer 19
Figure 4.9. True Dual Port RAM Primitive for CrossLink Devices 19
Figure 4.10. True Dual Port RAM Timing Waveform in Normal (NORMAL) Mode without Output Registers 22
Figure 4.11. True Dual Port RAM Timing Waveform in Normal (NORMAL) Mode with Output Registers 23
Figure 4.12. True Dual Port RAM Timing Waveform in Write through (WRITETHROUGH) Mode without Output Registers 24
Figure 4.13. True Dual Port RAM Timing Waveform in Write through (WRITETHROUGH) Mode with Output Registers 25
Figure 4.14. True Dual Port RAM Timing Waveform in Read before Write (READBEFOREWRITE) Mode without Output Registers 26
Figure 4.15. True Dual Port RAM Timing Waveform in Read before Write (READBEFOREWRITE) Mode with Output Registers 27
Figure 4.16. Pseudo Dual-Port Memory Module Generated by Clarity Designer 28
Figure 4.17. Pseudo-Dual Port RAM Primitive for CrossLink Devices 28
Figure 4.18. Pseudo Dual Port RAM Timing Diagram without Output Registers 31
Figure 4.19. Pseudo Dual Port RAM Timing Diagram with Output Registers 31
Figure 4.20. Read Only Memory Module Generated by Clarity Designer 32
Figure 4.21. ROM Timing Waveform without Output Registers 33
Figure 4.22. ROM Timing Waveform with Output Registers 34
Figure 4.23. FIFO Module Generated by Clarity Designer 34
Figure 4.24. FIFO without Output Registers, Start of Data Write Cycle 36
Figure 4.25. FIFO without Output Registers, End of Data Write Cycle 37
Figure 4.26. FIFO without Output Registers, Start of Data Read Cycle 38
Figure 4.27. FIFO without Output Registers, End of Data Read Cycle 39
Figure 4.28. FIFO with Output Registers, Start of Data Write Cycle 40
Figure 4.29. FIFO with Output Registers, End of Data Write Cycle 41
Figure 4.30. FIFO with Output Registers, Start of Data Read Cycle 42
Figure 4.31. FIFO with Output Registers, End of Data Read Cycle 43
Figure 4.32. FIFO with Output Registers and RdEn on Output Registers 44
Figure 4.33. FIFO_DC Module generated by the Clarity Designer 44
Figure 4.34. FIFO_DC without Output Registers, Start of Data Write Cycle 46
Figure 4.35. FIFO_DC without Output Registers, End of Data Write Cycle 47
Figure 4.36. FIFO_DC without Output Registers, Start of Data Read Cycle 48
Figure 4.37. FIFO_DC without Output Registers, End of Data Read Cycle 49
Figure 4.38. FIFO_DC with Output Registers, Start of Data Write Cycle 50
Figure 4.39. FIFO_DC with Output Registers, End of Data Write Cycle 51
Figure 4.40. FIFO_DC with Output Registers, Start of Data Read Cycle 52
Figure 4.41. FIFO_DC with Output Registers, End of Data Read Cycle 53
Figure 4.42. FIFO_DC with Output Registers and RdEn on Output Registers 54
Figure 4.43. Distributed Single-Port RAM Module Generated by Clarity Designer 55

Figure 4.44. Single Port Distributed RAM Primitive for CrossLink Devices 55
Figure 4.45. PFU Based Distributed Single Port RAM Timing Waveform without Output Registers 56
Figure 4.46. PFU Based Distributed Single Port RAM Timing Waveform with Output Registers 57
Figure 4.47. Distributed Dual-Port RAM Module Generated by Clarity Designer 57
Figure 4.48. Dual Port Distributed RAM Primitive for CrossLink Devices 58
Figure 4.49. PFU Based Distributed Dual Port RAM Timing Waveform without Output Registers 59
Figure 4.50. PFU Based Distributed Dual Port RAM Timing Waveform with Output Registers 60
Figure 4.51. Distributed ROM Generated by Clarity Designer 60
Figure 4.52. Distributed ROM Primitive for CrossLink Devices 61
Figure 4.53. PFU Based Distributed Dual Port ROM Timing Waveform without Output Registers 62
Figure 4.54. PFU Based Distributed Dual Port ROM Timing Waveform with Output Registers 63

Tables

Table 3.1. Masked Data in Bits for an 8-Bit or 9-Bit Byte Size 12
Table 4.1. EBR-Based Single-Port Memory Port Definitions 14
Table 4.2. Single-Port Memory Sizes for 9 KB Memories in CrossLink Devices 14
Table 4.3. Single-Port Memory Attribute Definitions for CrossLink Devices 15
Table 4.4. EBR-Based True Dual-Port Memory Port Definitions 20
Table 4.5. Dual Port Memory Sizes for 9K Memory for CrossLink 20
Table 4.6. True Dual-Port RAM Attributes for CrossLink Devices 20
Table 4.7. EBR-Based Pseudo Dual-Port Memory Port Definitions 29
Table 4.8. Pseudo-Dual Port Memory Sizes for 9K Memory for CrossLink Devices 29
Table 4.9. Pseudo Dual-Port RAM Attributes for CrossLink Devices 30
Table 4.10. EBR-Based ROM Port Definitions 32
Table 4.11. ROM Memory Sizes for 9K Memory for CrossLink Devices 32
Table 4.12. ROM Attributes for CrossLink Devices 33
Table 4.13. Port Names and Definitions for FIFO 35
Table 4.14. Port Names and Definitions for FIFO_DC 45
Table 4.15. PFU-Based Distributed Single Port RAM Port Definitions 56
Table 4.16. PFU-Based Distributed Dual-Port RAM Port Definitions 58
Table 4.17. PFU-Based Distributed ROM Port Definitions 61

Acronyms in This Document

A list of acronyms used in this document.

Acronym	Definition
EBR	Embedded Block RAM
ECC	Error-Correcting Code
EDIF	Electronic Design Interchange Format
FIFO	First In First Out
GUI	Graphical User Interface
HDL	Hardware Description Language
LSB	Least Significant Bit
LUT	Look-up Tables
MSB	Most Significant Bit
PFU	Programmable Functional Unit
PMI	Parameterizable Module Inferencing
RAM	Random Access Memory
ROM	Read Only Memory
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit

1. Introduction

This technical note discusses memory usage for the CrossLink™ family of FPGA devices. It is intended to be used by design engineers as a guide to integrating the Embedded Block RAM (EBR) and Logic Block (PFU) based memories for this device family in the Lattice Diamond® design software.

The architecture of these devices includes a number of options for applications that require memories. The sysMEM™ EBR compliments its distributed PFU-based memory. Single-Port RAM, Dual-Port RAM, Pseudo Dual-Port RAM and ROM memories can be constructed using the EBR. The LUTs and PFUs can implement Distributed Single-Port RAM, Dual-Port RAM and ROM. Look-up Tables (LUTs) within PFUs can implement Distributed Single-Port RAM, Dual-Port RAM and ROM.

The capabilities of the EBR and PFU RAM are discussed in this document. Designers can utilize the memory primitives in three separate ways:

- Via **Clarity Designer** – The Clarity Designer GUI allows users to specify the memory type and size requirement. Clarity Designer takes this specification and constructs a netlist to implement the desired memory by using one or more of the memory primitives.
- Via **Parameterizable Module Inferencing (PMI)** – PMI allows experienced users to skip the graphical user interface and utilize the configurable memory primitives on-the-fly from the Lattice Diamond project navigator. The parameters and the control signals can be set up either in Verilog or VHDL. The top-level design has the parameters defined and signals declared for a functional model for a memory block. The synthesis can automatically utilize the memories in the device to realize these blocks.
- Via the **Instantiation of Memory Primitives** – Memory primitives are called directly by the top-level module and instantiated in the user's design. This is an advanced method and requires a thorough understanding of memory hook-ups and design interfaces.

2. Memory Generation

CrossLink devices provide a number of options to generate memories of different sizes. Each EBR block can accommodate a memory of 9 KB, and a number of these blocks can cascade together to generate larger memories. The type of memory that can fit in each block is referred to as primitive. There are both EBR as well as Distributed Primitives available for CrossLink devices.

Designers can utilize Clarity Designer to easily configure a variety of memories. The tool can generate modules using one or more memory primitives. Cascading requires connecting the memories and Clarity Designer can likewise generate all the connections as required.

Below is a list of available primitives in Clarity Designer. These are the types of memories that can be generated. Two soft FIFO modules that can be generated are also included. These are the single clock FIFO and the dual clock FIFO_DC.

- Distributed Memory Modules
  - Distributed Dual Port RAM (Distributed_DPRAM)
  - Distributed ROM (Distributed_ROM)
  - Distributed Single Port RAM (Distributed_SPRAM)
- EBR Components (or EBR based Modules)
  - Dual PORT RAM (RAM_DP_TRUE)
  - Pseudo Dual Port RAM (RAM_DP)
  - Single Port RAM (RAM_DQ)
  - Read Only Memory (ROM)
- First In First Out Memory (FIFO and FIFO_DC)

Figure 2.1 shows the memory modules that are available under Clarity Designer in Lattice Diamond software.

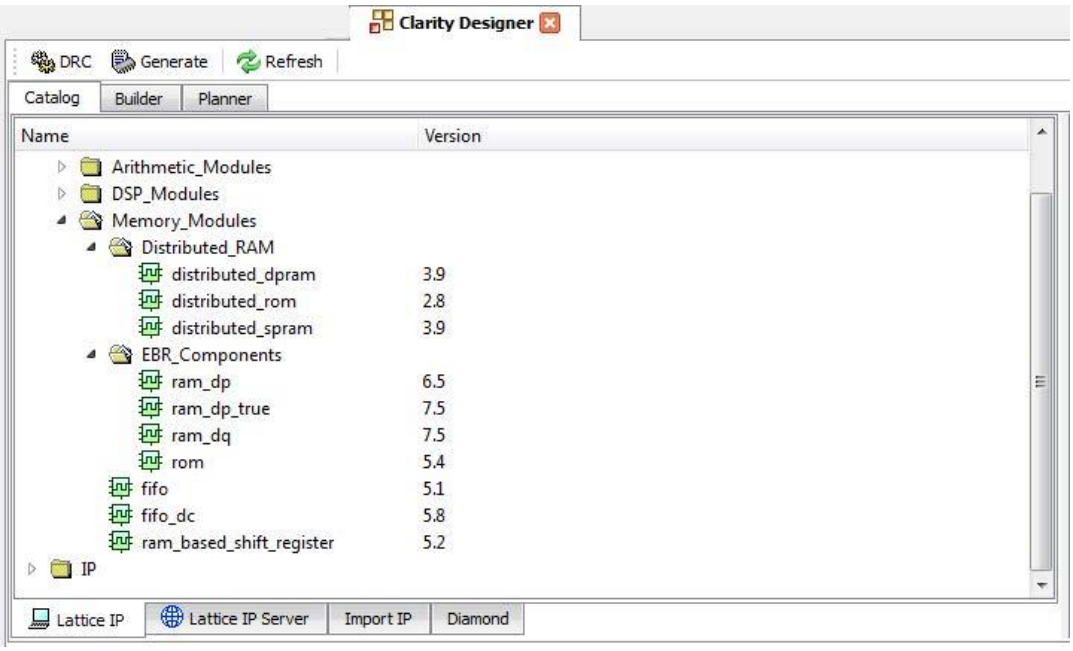

Figure 2.1. Memory Modules in Clarity Designer

2.1. Clarity Designer Flow

Clarity Designer allows you to generate, create (or open) the modules for CrossLink devices.

To open Clarity Designer from the Lattice Diamond software, as shown in Figure 2.2, select **Tools > Clarity Designer**.

Alternatively, click the  button in the toolbar.

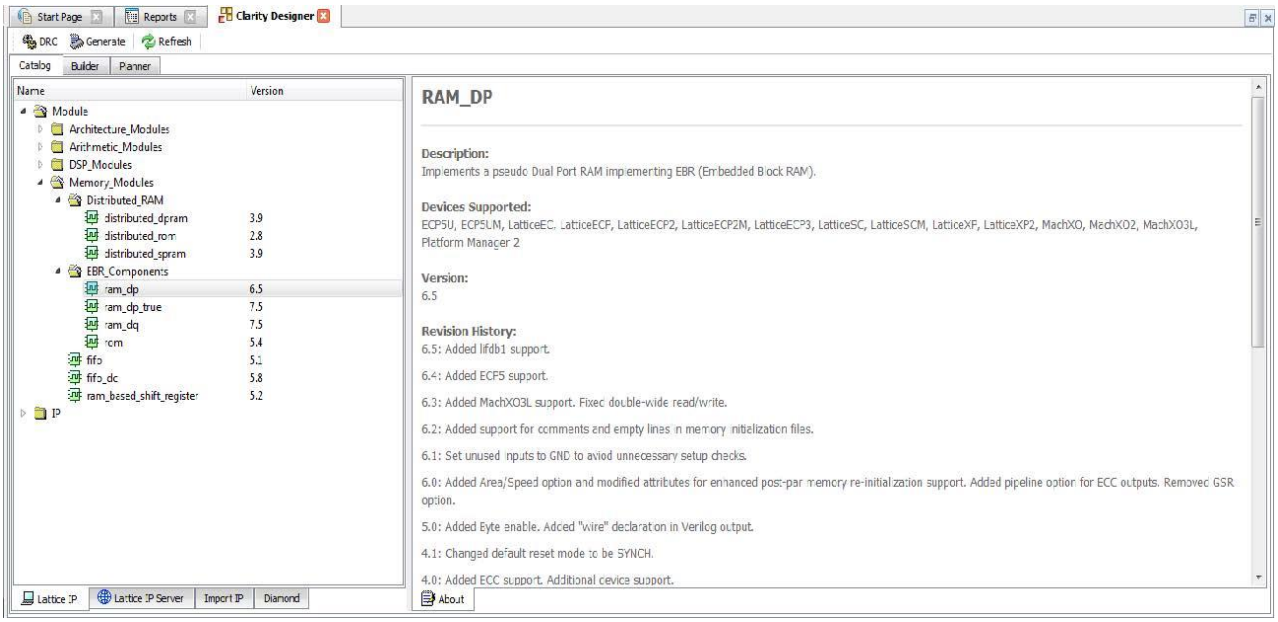

Figure 2.2. Clarity Designer in Lattice Diamond Software

The left section of the Clarity Designer window includes the Module Tree. The Memory Modules are categorized as **Distributed RAM, EBR Components** and **FIFOs**. The right section of the window shows the description of the selected module, supported devices, version, and other detailed information.

The following procedure shows an example of generating an EBR based 512 x 18 Pseudo Dual Port RAM.

Double-click **ram_dp** under **EBR_ Components**. In the **ram_dp** dialog box, shown in Figure 2.3, fill out the information regarding the module to generate, and click **Customize**.

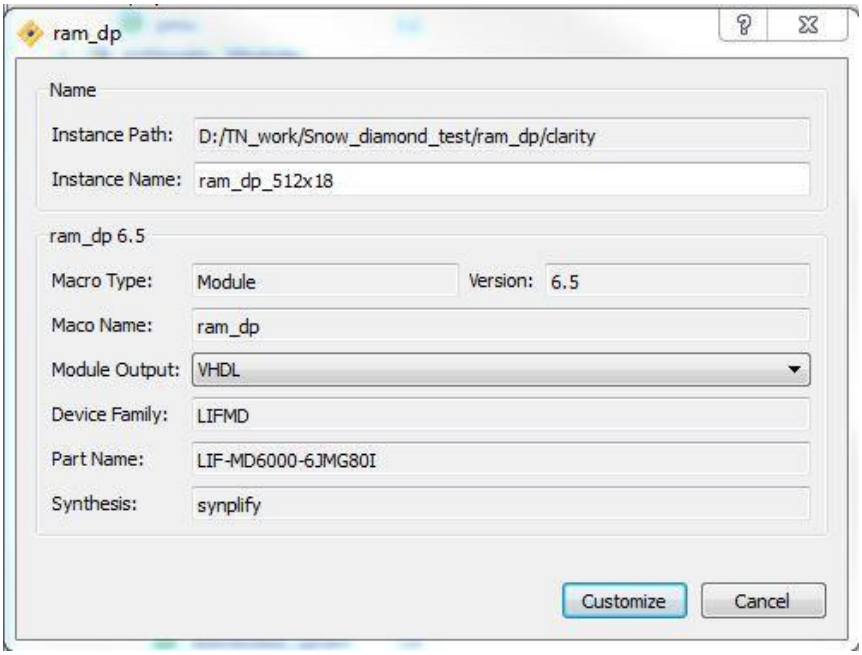

Figure 2.3. Example: Generating Pseudo Dual Port RAM (RAM_DP) Using Clarity Designer

Clicking **Customize** opens the **Lattice FPGA Module – RAM_DP** dialog box, shown in Figure 2.4, where you can customize the Distributed DPRAM.

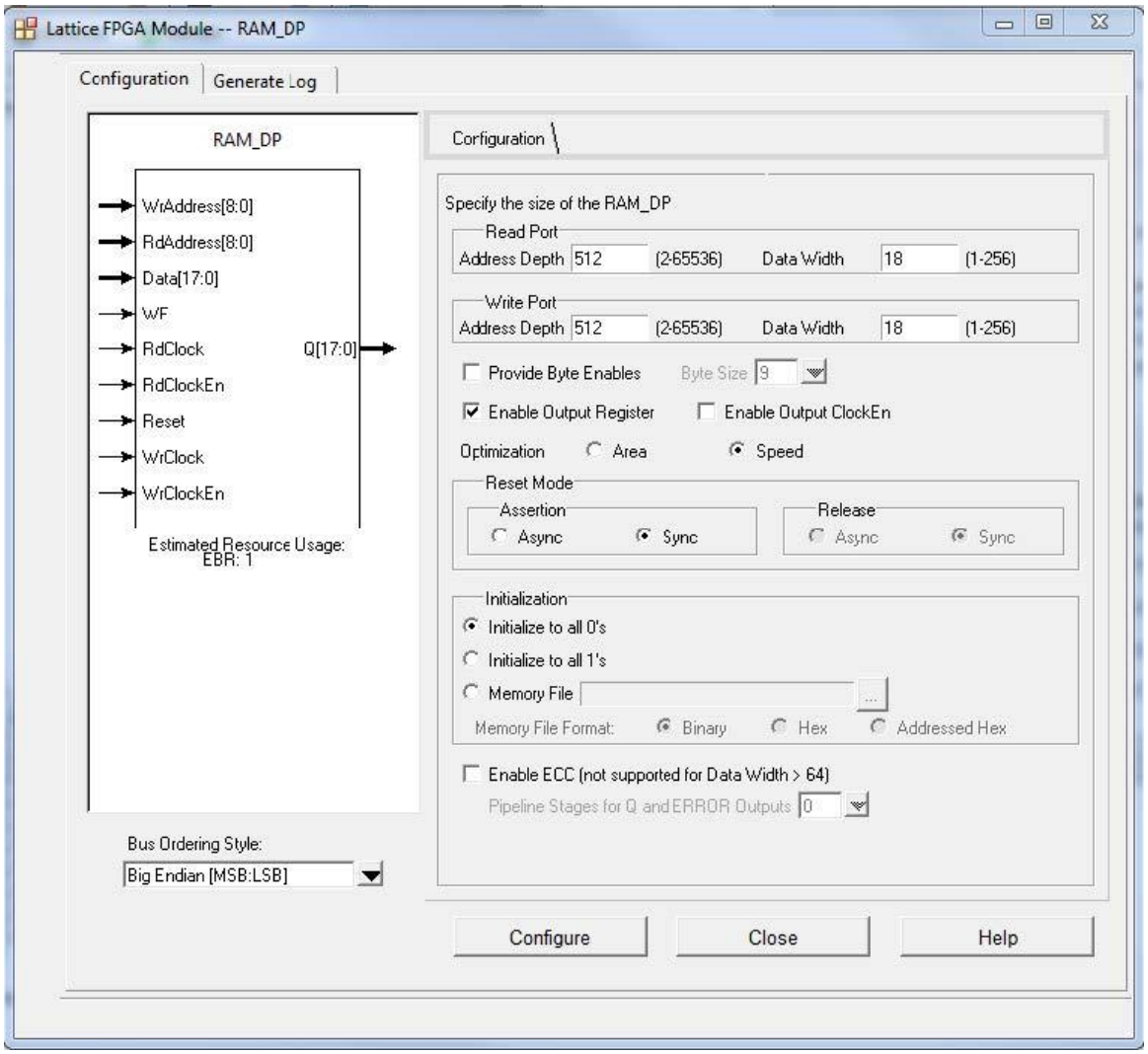

Figure 2.4. Example: Generating Pseudo Dual Port RAM (RAM_DP) Module Customization – General Options

Fill out the options and click **Configure**. When this module is generated in the Lattice Diamond project, it can be instantiated within other modules.

2.2. Utilizing PMI

PMI or the Parametrizable Module Inferencing is a method of using the memory blocks by writing behavior code for the memories. The parameters and control signals needed are set in either Verilog or VHDL. The top-level design includes the defined memory parameters and declared signals. The inference generates a memory using the primitives during synthesis and Lattice Diamond generates the netlist on-the-fly.

To do this, the user creates a Verilog or VHDL behavior code for the memory and the synthesis tool automatically identifies it as memory and synthesizes it as a distributed or EBR memory.

Memory sizes smaller than 2 KB are automatically mapped to Distributed mode and those larger than 2 KB are implemented using EBRs. This default option can be over-ridden using the RAM_STYLE attribute in Synopsys Synplify Pro®. Refer to the Synplify Pro Reference Manual on the Lattice Diamond Start page.

Synthesis Inferred RAMs

The user can create a Verilog or VHDL behavior code for the memory and the synthesis tool automatically identifies the RAM. Below is an example of a single port RAM inference:

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

entity sync_ram_singleport is
generic (data_width : natural := 8 ;
addr_width : natural := 8);

port ( clk : in std_logic;
we : in std_logic ;
addr : in std_logic_vector( addr_width - 1 downto 0) ; data_in : in
std_logic_vector( data_width - 1 downto 0) ; data_out : out
std_logic_vector( data_width - 1 downto 0) );
end sync_ram_singleport ;

architecture rtl of sync_ram_singleport is

_type is array (2** addr_width downto 0) of std_logic_vector(
data_width - 1 downto 0) ;

signal mem : mem_type ;
signal addr_reg : std_logic_vector( addr_width - 1 downto 0) ;

begin singleport : process (clk)

begin
 (clk'event and clk = '1') then if
(we = '1') then
   mem( conv_integer( addr)) <= data_in ; end
if ;
  addr_reg <= addr ;
end if ;

end process singleport;
data_out <= mem( conv_integer( addr_reg)) ;

end rtl;
```

Note that some of the unique features of a RAM such as ECC and Byte Enable, among others, are very difficult to realize using inferencing. The general behavior of the RAM is easily translated onto the EBR or Distributed RAM blocks. The Clarity Designer flow is recommended to access and utilize these features.

2.3. Utilizing Direct Instantiation of Memory Primitives

You can directly instantiate the memory primitives for the CrossLink devices. When instantiating the primitives, you have to work at the single block level – EBR or Distributed RAM Block.

For memories that span beyond a single primitive, cascading connections should be made during instantiation. This creates the cascading memory on its own. A detailed list of all the primitives in a VHDL/ Verilog file is available in the cae_library/synthesis folder in the Lattice Diamond software installation package. Search for the primitive names in the lifmd.v or lifmd.vhd file.

3. Memory Features

The RAMs can be generated with Error Correction and Byte Enables that mask selective bits. These features are available in the EBR based RAM modules.

3.1. ECC in Memory Modules

An error-correcting code (ECC) is a system of adding redundant data, or parity data, to a message, such that it can be recovered by a receiver even when a number of errors are introduced, either during the process of transmission, or on storage.

In Clarity Designer, you can implement ECC in EBR based memory modules by selecting Enable ECC in the Configuration tab.

Enabling ECC allows error correction of single errors and detection of 2-bit errors. This is not supported for data widths higher than 64 bits.

The two bits indicate the error, if any, and the following describes what each of these bits mean:

- Error[1:0]= 00 Indicates there is no error
- Error[0]= 1 Indicates there is a 1-bit error which is fixed
- Error[1]= 1 Indicates there is a 2-bit error which cannot be corrected.

ECC is adding 0~2 cycle of delay to both the data and ERROR outputs base on the Clarity Designer GUI setting. The data and ERROR output are always in sync.

One of the things to note is that the ECC is added in the PFU logic. As such, the logic is implemented outside the EBR block and can impact speed. The effective speed at which the memory runs should be determined by running the Trace report.

3.2. Byte Enable

Byte Enable is a feature available in the selected RAM modules to mask the bytes written in the RAM. While generating the module in clarity Designer, you can configure the Byte Enable control to be either 8 bits or 9 bits.

Note that the Byte Enable option is available for memory widths higher than 8 bits (1 byte).

Each bit of the BE signal corresponds to the 8-bit or 9-bit selection, starting from LSB side. Table 3.1 shows how written data (Data In) is masked for an 8-bit or 9-bit data size when you add Byte Enable to an 18-bit wide RAM.

Table 3.1. Masked Data in Bits for an 8-Bit or 9-Bit Byte Size

Byte Enable Bit	Data In Bits that Get Masked (8-Bit Size)	Data In Bits that Get Masked (9-Bit Size)
ByteEn(0)	Data(7:0)	Data(8:0)
ByteEn(1)	Data(13:8)	Data(15:9)

Note that the ByteEn and ECC are exclusive and cannot be used together.

4. Memory Modules

The following sections discuss the different modules, the memory size supported by each EBR block or Distributed primitive, and other special options for the module.

When you specify the width and depth of the memory in Clarity Designer, the tool generates the memory by depth cascading and/or width cascading, EBR blocks or Distributed RAM primitives.

Clarity Designer automatically allows you to create memories larger than the width and depth supported for each primitive.

4.1. Memory Cascading

For memory sizes smaller than what can fit in a single EBR block or Distributed primitive, the module utilizes the complete block or primitive.

For memory sizes larger than a single module, the multiple modules are cascaded (either in depth or width) to create a larger module.

4.1.1. Input and Output Register

The architecture of the EBR blocks in CrossLink devices are designed such that the inputs that go into the memory are always registered. This means that the input data and address are always registered at the input of the memory array. The output data of the memory is optionally registered at the output. You can enable this by selecting Enable Output Register in Clarity Designer while customizing the module.

Control signals such as WE and Byte Enable are also registered going into the EBR block.

4.1.2. Reset

The EBRs also support the Reset signal. The Reset (or RST) signal only resets input and output registers of the RAM. It does not reset the contents of the memory.

4.1.3. Timing

In order to correctly write into a memory cell in the EBR block, the correct address should be registered by the logic. Hence, it is important to note that while running the trace on the EBR blocks, there should be no setup and hold time violations on the address registers (address). Failing to meet these requirements can result in incorrect addressing and corruption of memory contents.

A similar issue can occur During a read cycle. The correct contents are not read if the address is not correctly registered in the memory.

Run a Post Place and Route timing report in Lattice Diamond design software to verify that no such timing errors occur. Refer to the timing preferences in the Online Help documents.

4.2. Single Port RAM (RAM_DQ) – EBR Based

CrossLink FPGA supports all the features of Single Port Memory Module or RAM_DQ. Clarity Designer allows you to generate the Verilog-HDL, VHDL or EDIF netlists for the memory size as per design requirement.

Clarity Designer generates the memory module, as shown in Figure 4.1.

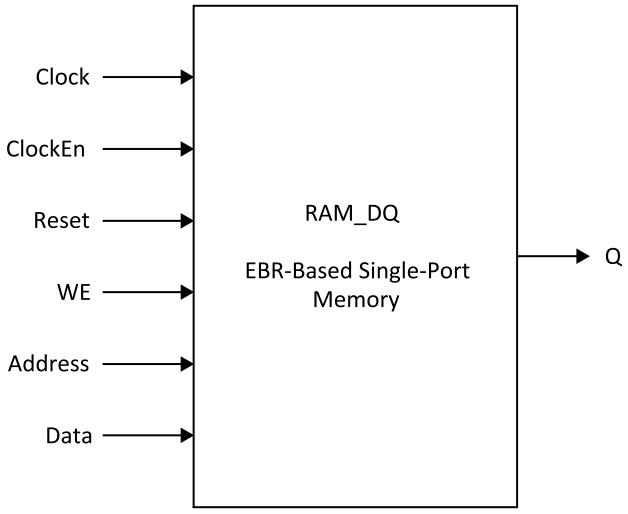

Figure 4.1. Single-Port Memory Module Generated by Clarity Designer

Note that each EBR can accommodate 9 KB of memory. If the memory required is larger than 9 KB, cascading can be done using the CS port (CSA and CSB in this case). See the A.3. CSDECODE section of Appendix A for information on cascading multiple EBR primitives.

The various ports and their definitions for Single-Port Memory are listed in Table 4.1. The table lists the corresponding ports for the module generated by Clarity Designer and for the EBR RAM_DQ primitive.

Table 4.1. EBR-Based Single-Port Memory Port Definitions

Port Name in the Generated Module	Port Name in the EBR Block Primitive	Description
Clock	CLK	Clock
ClockEn	CE	Clock Enable
Address	AD[x:0]	Address Bus
Data	DI[y:0]	Data In
Q	DO[y:0]	Data Out
WE	WE	Write Enable
Reset	RST	Reset
—	CS[2:0]	Chip Select

Each EBR block consists of 18,432 bits of RAM. The values for x (address) and y (data) of each EBR block are listed in Table 4.2.

Table 4.2. Single-Port Memory Sizes for 9 KB Memories in CrossLink Devices

Single Port Memory Size	Input Data	Output Data	Address [MSB:LSB]
8,192 x 1	DI	DO	AD[12:0]
4,096 x 2	DI[1:0]	DO[1:0]	AD[11:0]
2,048 x 4	DI[3:0]	DO[3:0]	AD[10:0]
1,024 x 9	DI[8:0]	DO[8:0]	AD[9:0]
512 x 18	DI[17:0]	DO[17:0]	AD[8:0]

Table 4.3 describes the various attributes available for the Single-Port Memory (RAM_DQ). Some of these attributes are user-selectable through the Clarity Designer GUI.

The attributes without selectable options in Clarity Designer are handled by the engine. However, users working with the direct primitive instantiation can access these options.

Table 4.3. Single-Port Memory Attribute Definitions for CrossLink Devices

Configuration Tab Attributes	Description	Values	Default Value
Address Depth	Address depth of the read and write port	2 – Max that can fit in the device	512
Data Width	Data word width of the read and write port	1 – 256	36
Enable Output Register	Data Out port (Q) can or cannot be registered using this selection.	TRUE, FALSE	TRUE
Enable Output ClockEn	Clock Enable for the output clock (this option requires Enabling Output Register)	TRUE, FALSE	FALSE
Byte Enables	Allows users to select Byte Enable options	TRUE, FALSE	FALSE
Byte Size	Byte Size selection when Byte Enable option is selected	9, 8	9
Reset Mode	Selection for the Reset to be synchronous or asynchronous to the Clock	Async, Sync	Sync
Reset Release	Selection for the release of Asynchronous Reset to be synchronous or asynchronous to the Clock	ASYNC, SYNC	SYNC
Optimization	Design optimizations to configure EBR for Speed or Area	Area, Speed	Speed
Initialization	Allows users to initialize their memories to all 1s, 0s or provide a custom initialization using a memory file.	0s, 1s, File	0s
Memory File	When Memory File is selected, users can browse to the memory file for custom initialization of RAM.	—	—
Memory File Format	Allows users to select if the memory file is formatted as Binary, Hex or address Hex. (See Appendix A for details on different formats.)	Binary, Hex, Addressed Hex	Binary
Enable ECC	Allows users to enable Error Correction Codes. This option is not available for memory that is wider than 64 bits.	TRUE, FALSE	FALSE
Advanced Tab Attributes	**Description**	**Values**	**Default Value**
Write Mode	Allows users to select different write modes. Check Timing waveforms for the behavior of RAMs in these modes.	NORMAL, WRITETHROUGH, READBEFOREWRITE	NORMAL

The Single-Port RAM (RAM_DQ) can be configured as NORMAL, WRITE THROUGH or READBEFOREWRITE mode. Each of these modes affects the data coming out of port Q of the memory during the write operation followed by the read operation at the same memory location.

Additionally, you can select to enable the output registers for RAM_DQ. The waveforms in the figures in the following pages show the internal timing waveforms for the Single Port RAM (RAM_DQ) with these options.

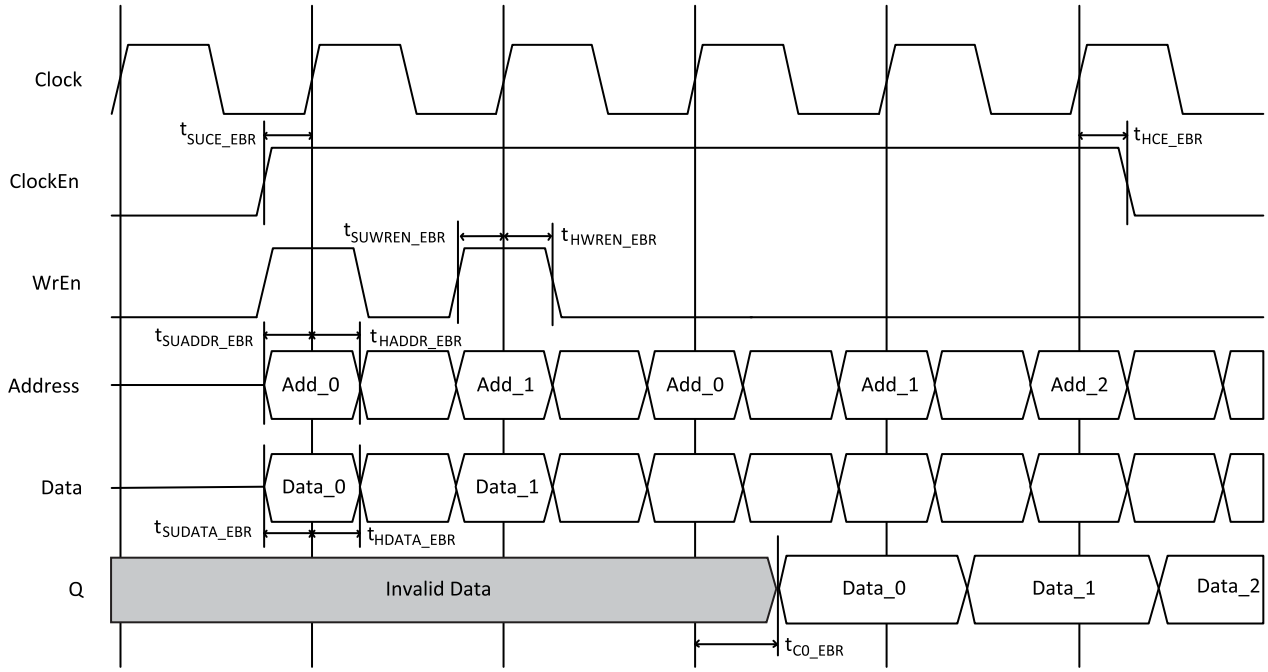

Figure 4.2. Single Port RAM Timing Waveform in Normal (NORMAL) Mode without Output Registers

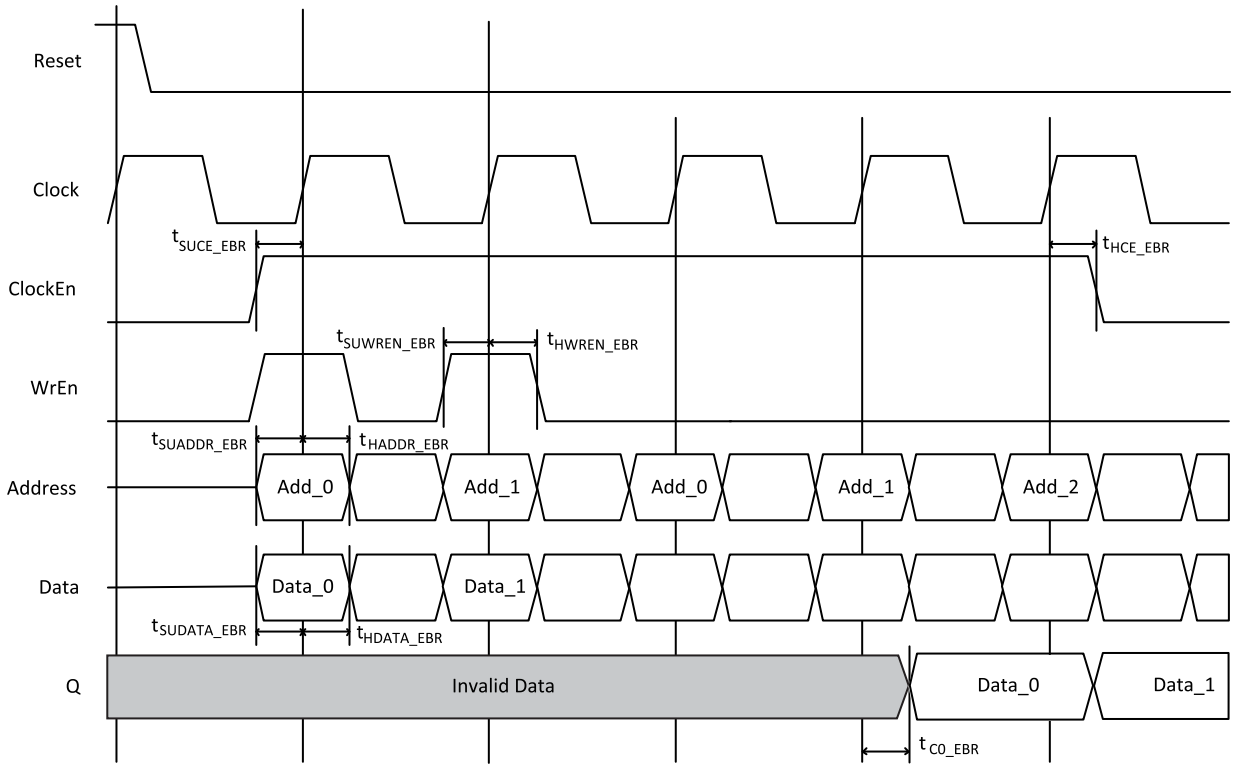

Figure 4.3. Single Port RAM Timing Waveform in Normal (NORMAL) Mode with Output Registers

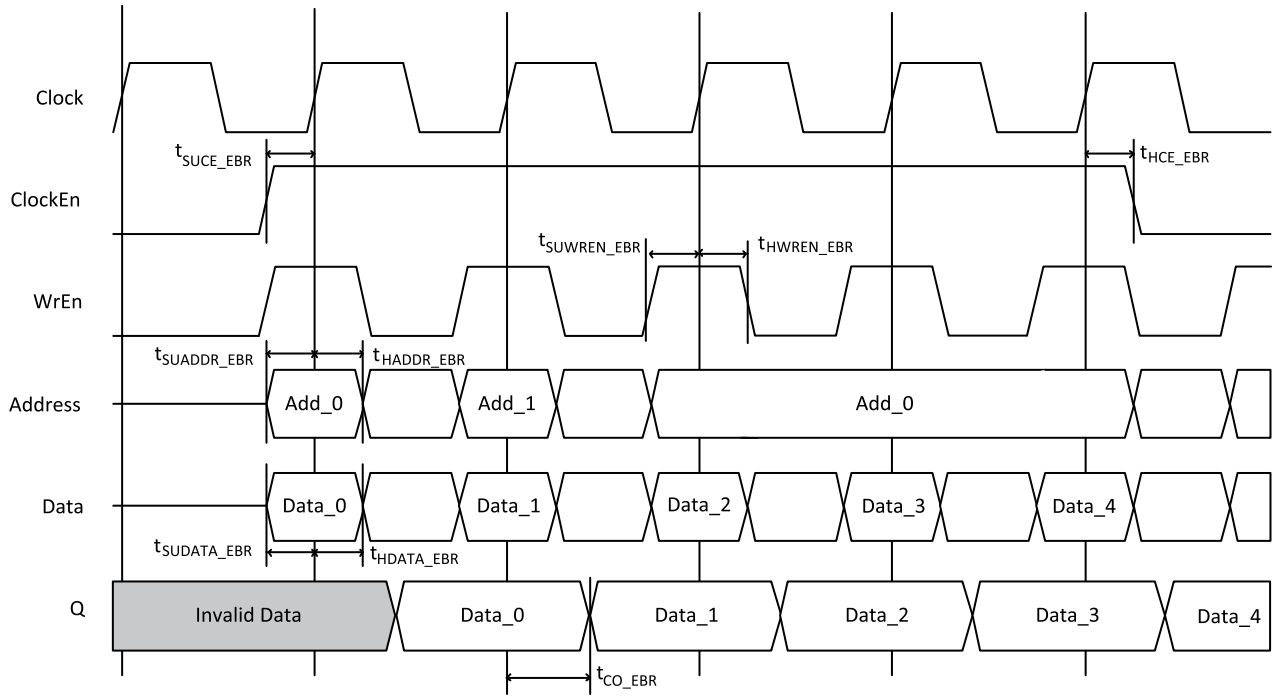

Figure 4.4. Single Port RAM Timing Waveform in Write through (WRITETHROUGH) Mode without Output Registers

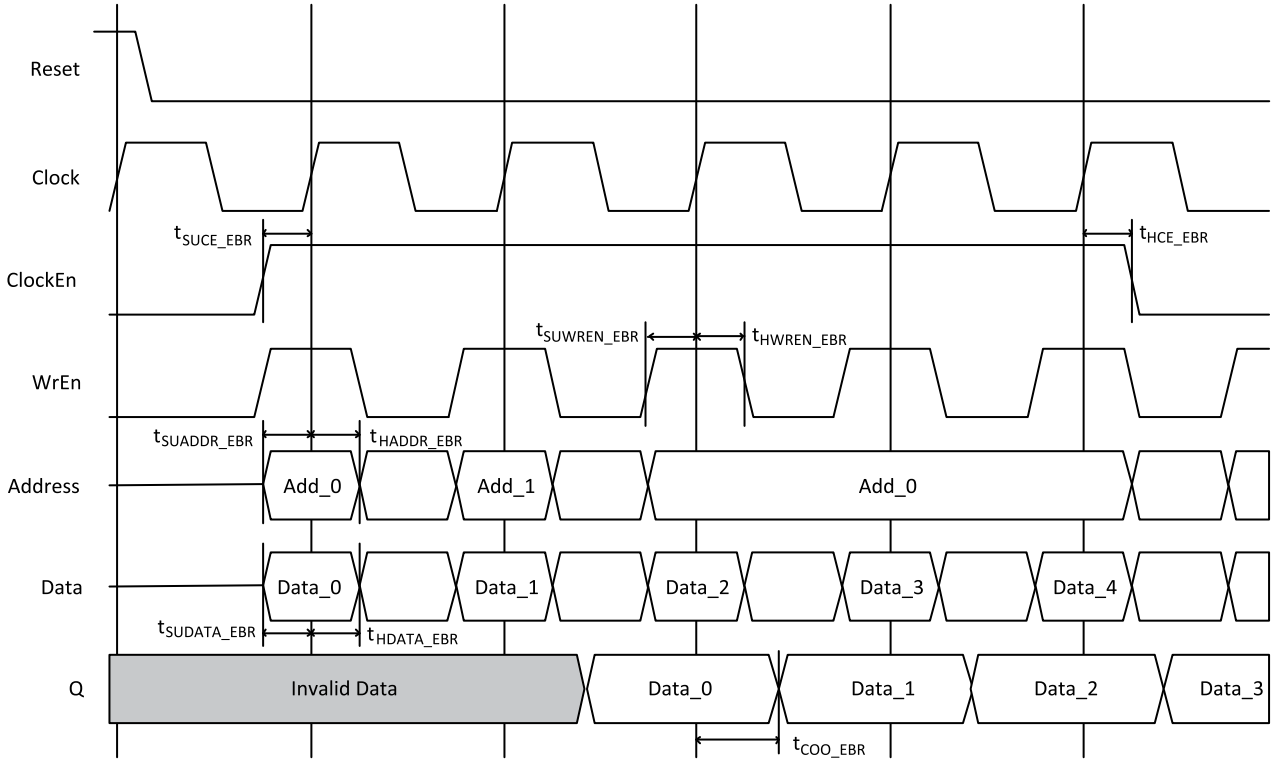

Figure 4.5. Single Port RAM Timing Waveform in Write through (WRITETHROUGH) Mode with Output Registers

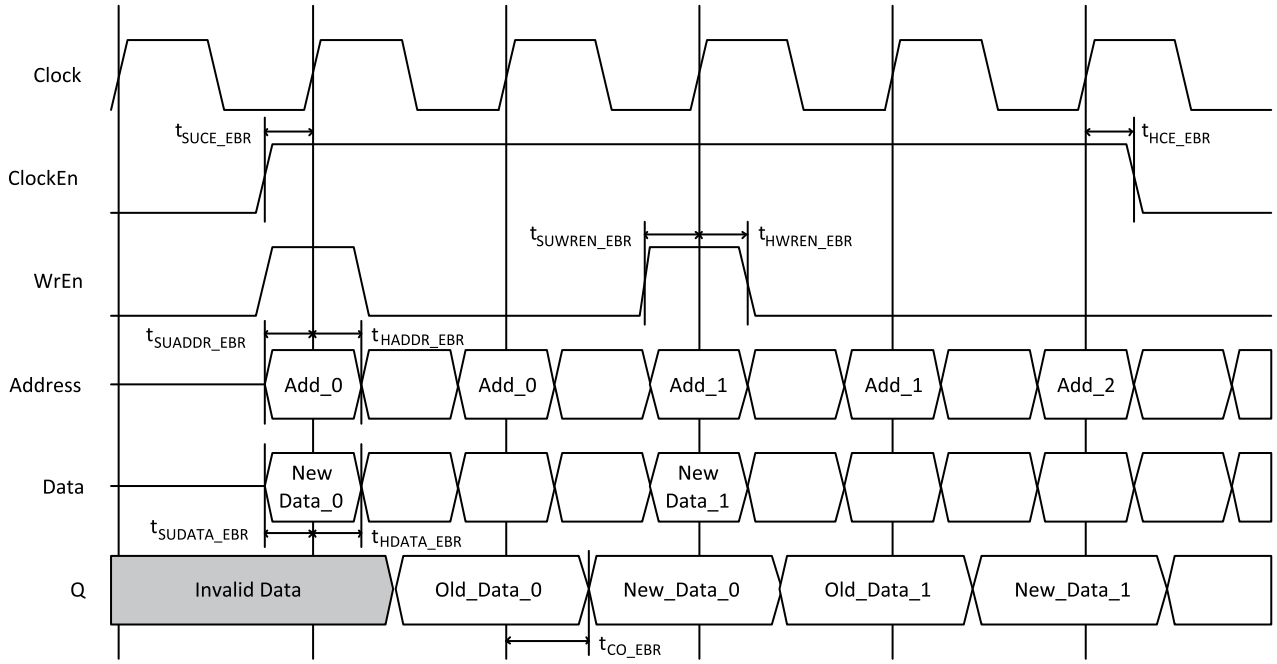

Figure 4.6. Single Port RAM Timing Waveform in Read before Write (READBEFOREWRITE) Mode without Output Registers

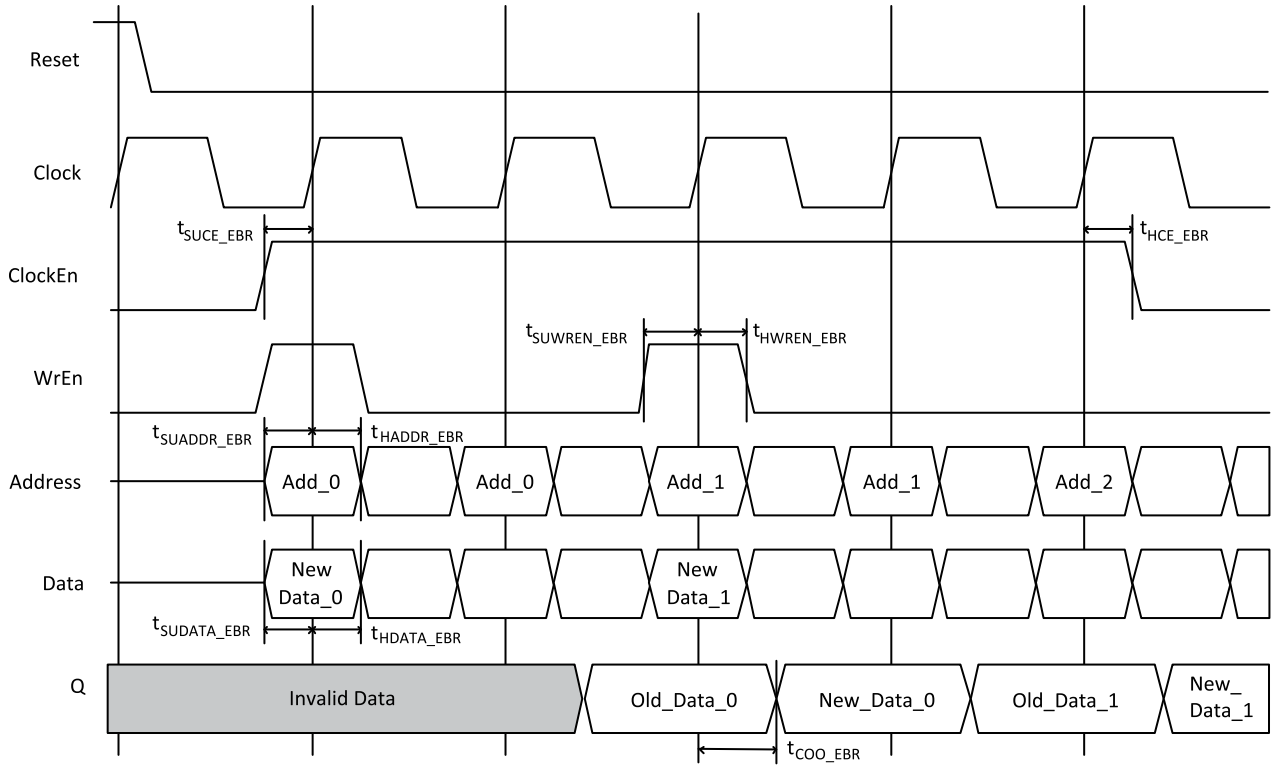

Figure 4.7. Single Port RAM Timing Waveform in Read before Write (READBEFOREWRITE) Mode with Output Registers

4.3. True Dual-Port RAM (RAM_DP_TRUE) – EBR Based

The EBR blocks in CrossLink devices can be configured as True-Dual Port RAM or RAM_DP_TRUE. Clarity Designer allows you to generate the Verilog-HDL, VHDL or EDIF netlists for the memory size as per design requirements.

Clarity Designer generates the memory module, as shown in Figure 4.8.

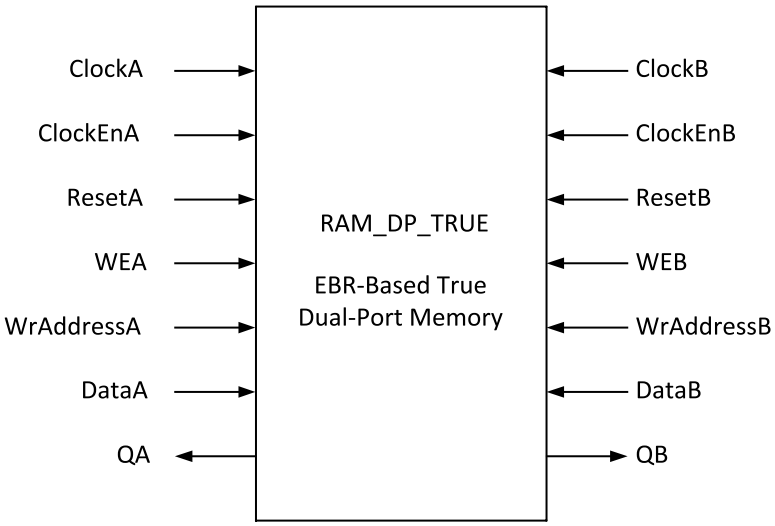

Figure 4.8. True Dual-Port Memory Module Generated by Clarity Designer

Figure 4.9 provides the primitive that can be instantiated for the True Dual Port RAM. The primitive name is DP8KE and it can be directly instantiated in the code. Check the details on the port and port names under the primitives available in the cae_library/synthesis folder in Lattice Diamond software installation package.

Note that each EBR can accommodate 9 KB of memory. If the memory required is larger than 9 KB, cascading can be done using the CS port (CSA and CSB in this case). See the A.3. CSDECODE section of Appendix A on how to perform cascading of multiple EBR primitives.

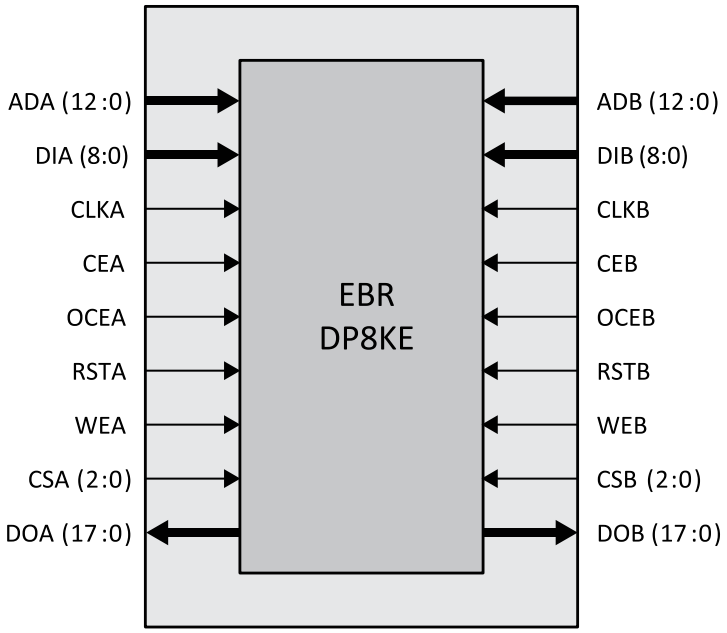

Figure 4.9. True Dual Port RAM Primitive for CrossLink Devices

The various ports and their definitions for True Dual-Port RAM are listed in Table 4.4 The table lists the corresponding ports for the module generated by Clarity Designer and for the EBR RAM_DP_TRUE primitive.

Table 4.4. EBR-Based True Dual-Port Memory Port Definitions

Port Name in the Generated Module	Port Name in the EBR Block Primitive	Description
ClockA, ClockB	CLKA, CLKB	Clock for PortA and PortB
ClockEnA, ClockEnB	CEA, CEB	Clock Enables for Port CLKA and CLKB
AddressA, AddressB	ADA[w:0], ADB[x:0]	Address Bus port A and port B
DataA, DataB	DIA[y:0], DIB[z:0]	Input Data port A and port B
QA, QB	DOA[y:0], DOB[z:0]	Output Data port A and port B
WEA, WEB	WEA, WEB	Write enable port A and port B
ResetA, ResetB	RSTA, RSTB	Reset for PortA and PortB
—	CSA[2:0], CSB[2:0]	Chip Selects for each port

Each EBR block consists of 9 KB of RAM. The values for address (w and x) and data (y and z) of each EBR block are listed in Table 4.5.

Table 4.5. Dual Port Memory Sizes for 9K Memory for CrossLink

Dual Port Memory Size	Input Data Port A	Input Data Port B	Output Data Port A	Output Data Port B	Address Port A	Address Port B
8192 x 1	DataInA	DataInB	QA	QB	AddressA(12:0)	AddressB(12:0)
4096 x 2	DataInA(1:0)	DataInB(1:0)	QA(1:0)	QB(1:0)	AddressA(11:0)	AddressB(11:0)
2049 x 4	DataInA(3:0)	DataInB(3:0)	QA(3:0)	QB(3:0)	AddressA(10:0)	AddressB(10:0)
1024 x 9	DataInA(8:0)	DataInB(8:0)	QA(8:0)	QB(8:0)	AddressA(9:0)	AddressB(9:0)

Table 4.6 describes the various attributes available for True Dual-Port Memory (RAM_DQ). Some of these attributes are user-selectable through the Clarity Designer GUI.

Table 4.6. True Dual-Port RAM Attributes for CrossLink Devices

Configuration Tab Attributes	Description	Values	Default Value
Port A Address Depth	Port A Address depth of the read and write port	2 – Max that can fit in the device	512
Port A Data Width	Port A Data word width of the read and write port	1 – 256	36
Port B Address Depth	Port B Address depth of the read and write port	2 – Max that can fit in the device	512
Port B Data Width	Port B Data word width of the read and write port	1 – 256	36
Port A Enable Output Register	Port A Data Out port (QA) can or cannot be registered using this selection.	TRUE, FALSE	TRUE
Port A Enable Output ClockEn	Port A Clock Enable for the output clock (this option requires Enabling Output Register)	TRUE, FALSE	FALSE
Port B Enable Output Register	Port B Data Out port (QB) can be registered or not using this selection.	TRUE, FALSE	TRUE
Port B Enable Output ClockEn	Port B Clock Enable for the output clock (this option requires Enabling Output Register)	TRUE, FALSE	FALSE
Byte Enables	Allows users to select Byte Enable options	TRUE, FALSE	FALSE
Byte Size	Byte size selection when Byte Enable option is selected	9, 8	9
Reset Mode	Selection for the Reset to be synchronous or asynchronous to the Clock	Async, Sync	Sync
Reset Release	Selection for the release of Asynchronous Reset to be synchronous or asynchronous to the Clock	ASYNC, SYNC	SYNC
Optimization	Design optimizations to configure EBR for Speed or Area	Area, Speed	Speed

Configuration Tab Attributes	Description	Values	Default Value
Initialization	Allows users to initialize their memories to all 1s, 0s or providing a custom initialization by providing a memory file	0s, 1s, File	0s
Memory File	When Memory File is selected, user can browse to the memory file for custom initialization of RAM.	—	—
Memory File Format	This option allows users to select if the memory file is formatted as Binary, Hex or address Hex. (See Appendix A for details on different formats.)	Binary, Hex, Addressed Hex	Binary
Enable ECC	Allows users to enable Error Correction Codes. This option is not available for memory that is wider than 64 bits.	TRUE, FALSE	FALSE
Advanced Tab Attributes	**Description**	**Values**	**Default Value**
Port A Write Mode	Option to select different write modes for Port A. Check Timing waveforms for the behavior of RAMs in these modes.	NORMAL, WRITETHROUGH, READBEFOREWRITE	NORMAL
Port B Write Mode	Option to select different write modes for Port B. Check Timing waveforms for the behavior of RAMs in these modes.	NORMAL, WRITETHROUGH, READBEFOREWRITE	NORMAL

The True Dual Port RAM (RAM_DP_TRUE) can be configured as NORMAL, WRITE THROUGH or READBEFOREWRITE modes. Each of these modes affects what data comes out of the port Q of the memory during the write operation followed by the read operation at the same memory location. For details on WRITE modes and the constraints of the True Dual Port see Appendix A. Attribute Definitions.

Additionally, users can select to enable the output registers for RAM_DP_TRUE. Waveforms in the following figures show the internal timing waveforms for the True Dual Port RAM (RAM_DP_TRUE) with these options.

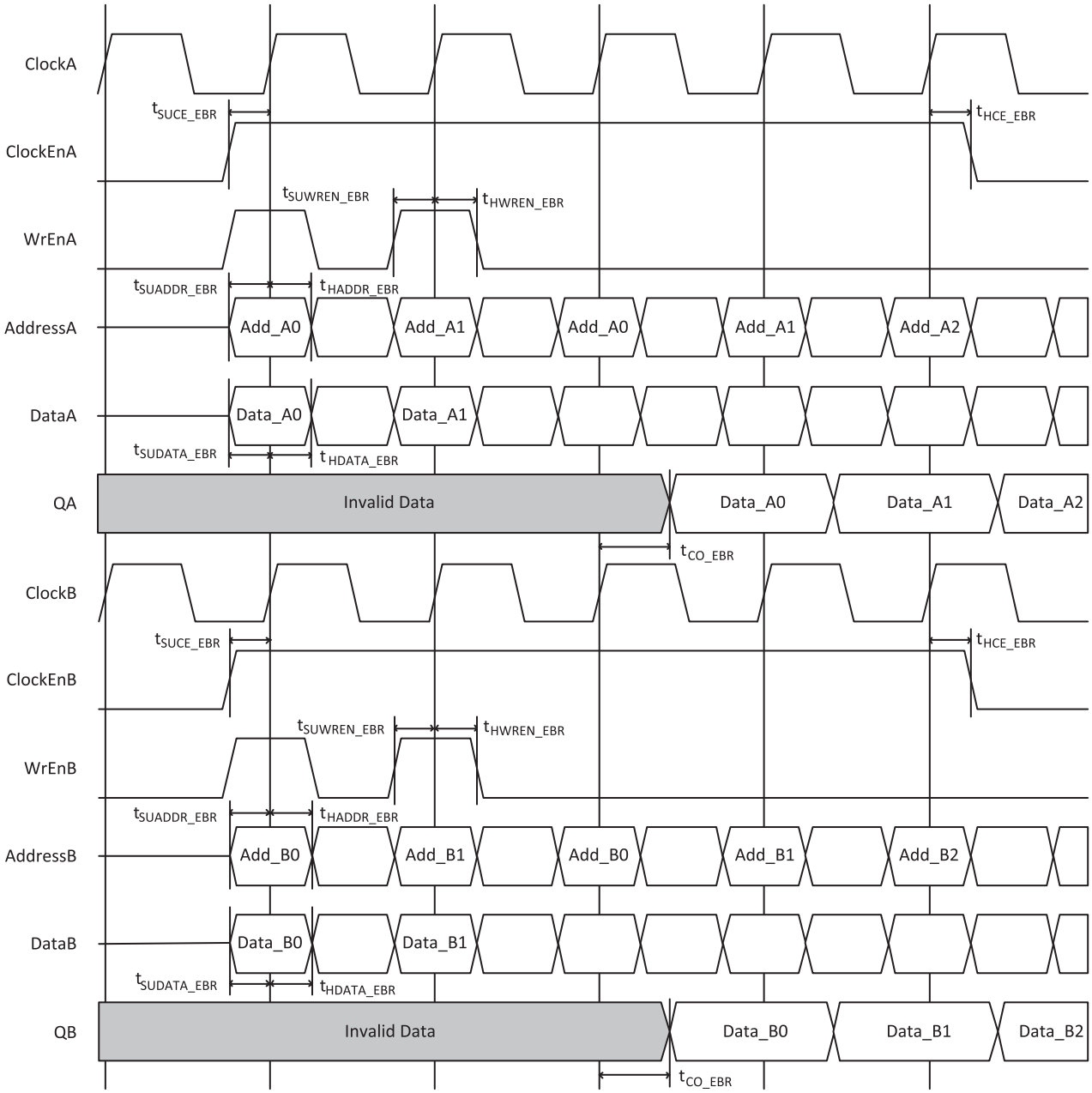

Figure 4.10. True Dual Port RAM Timing Waveform in Normal (NORMAL) Mode without Output Registers

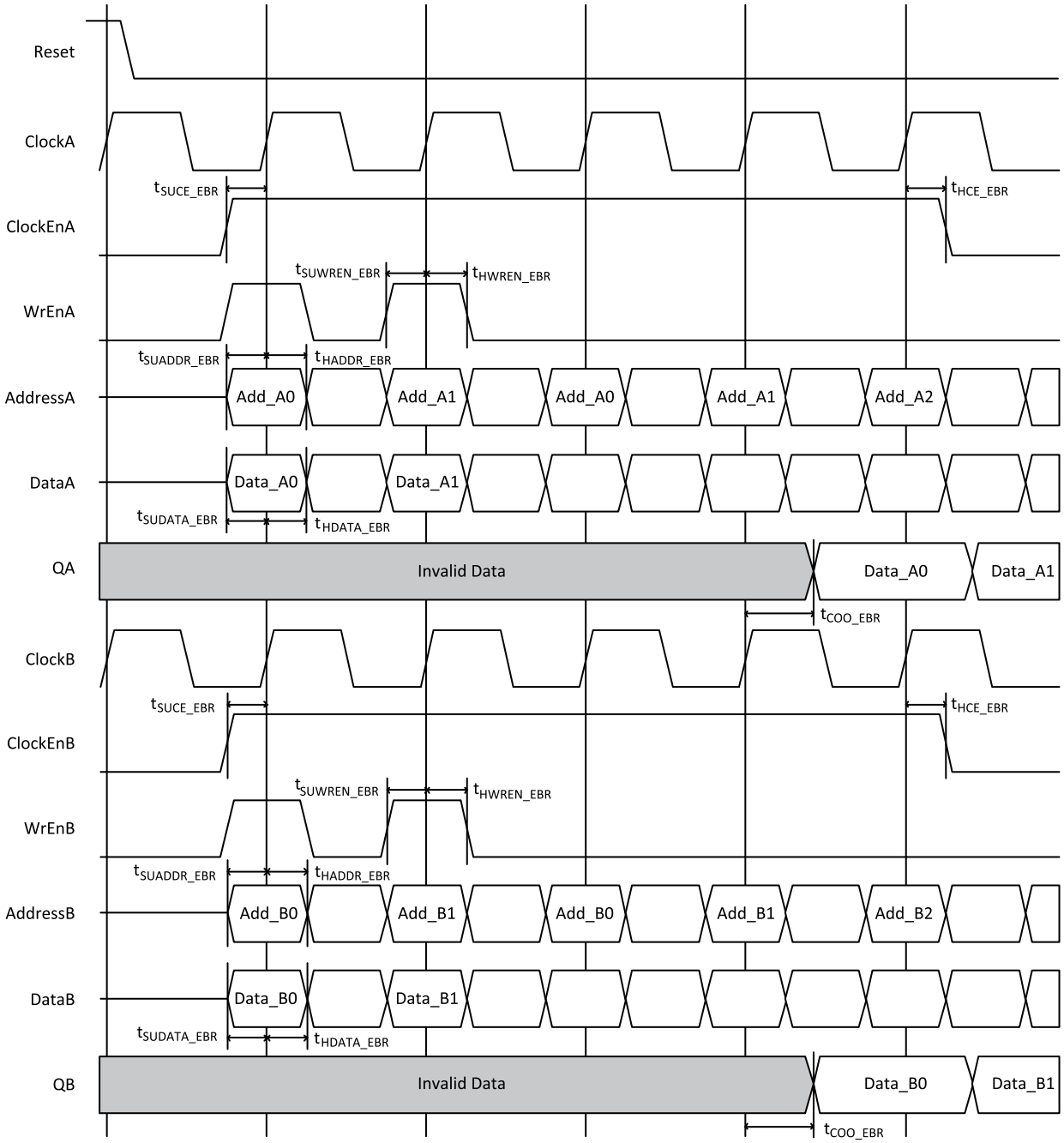

Figure 4.11. True Dual Port RAM Timing Waveform in Normal (NORMAL) Mode with Output Registers

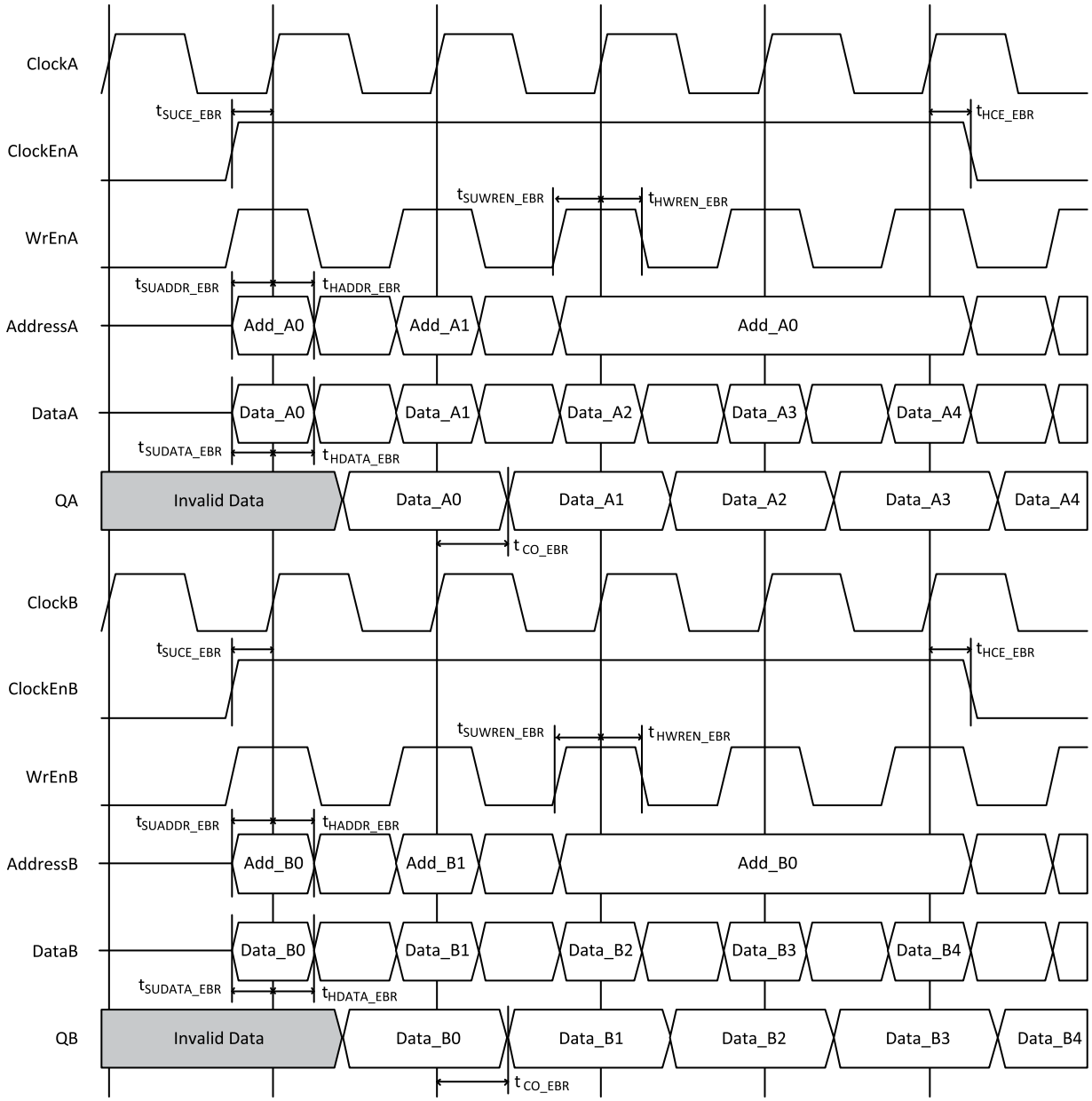

Figure 4.12. True Dual Port RAM Timing Waveform in Write through (WRITETHROUGH) Mode without Output Registers

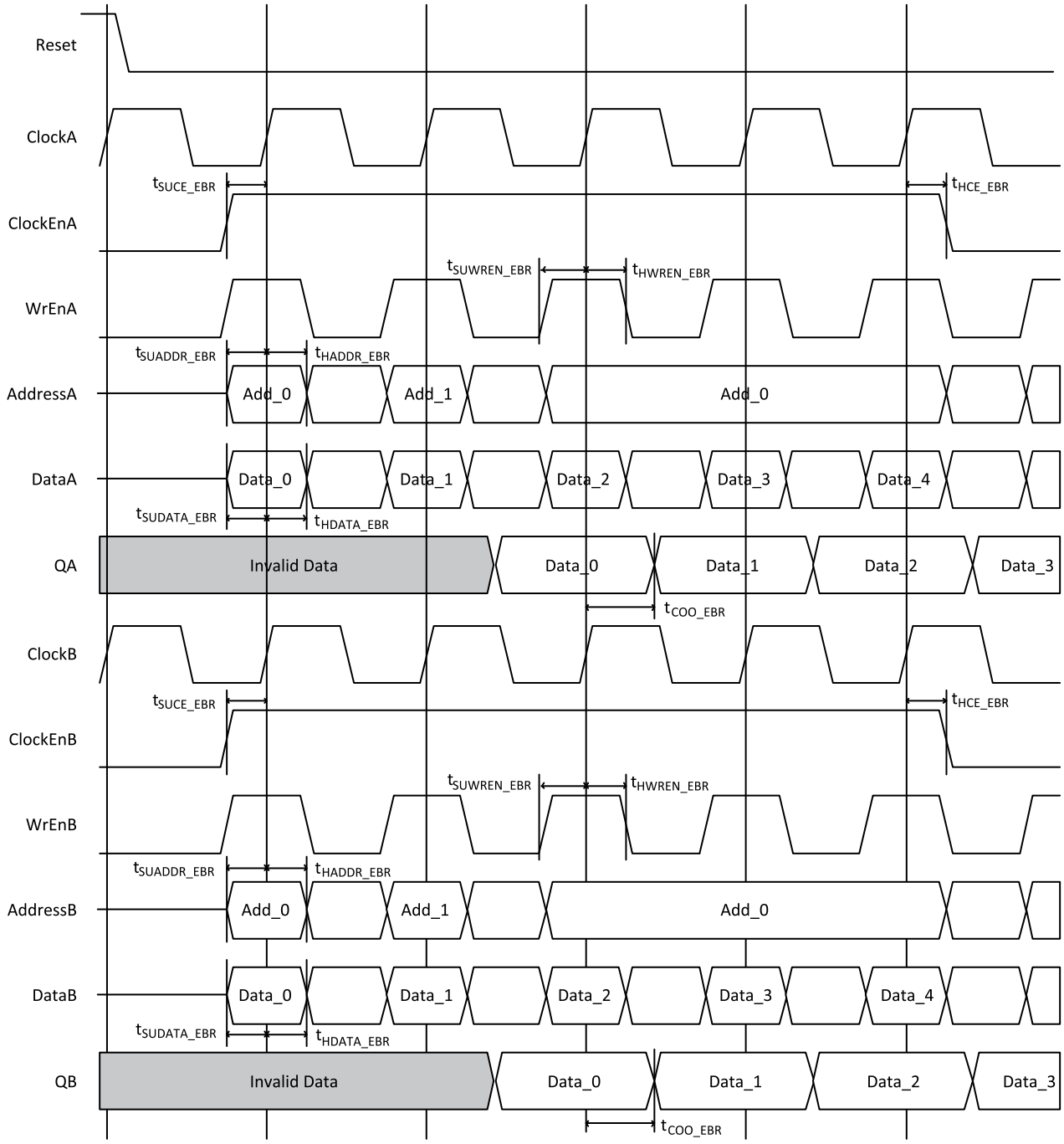

Figure 4.13. True Dual Port RAM Timing Waveform in Write through (WRITETHROUGH) Mode with Output Registers

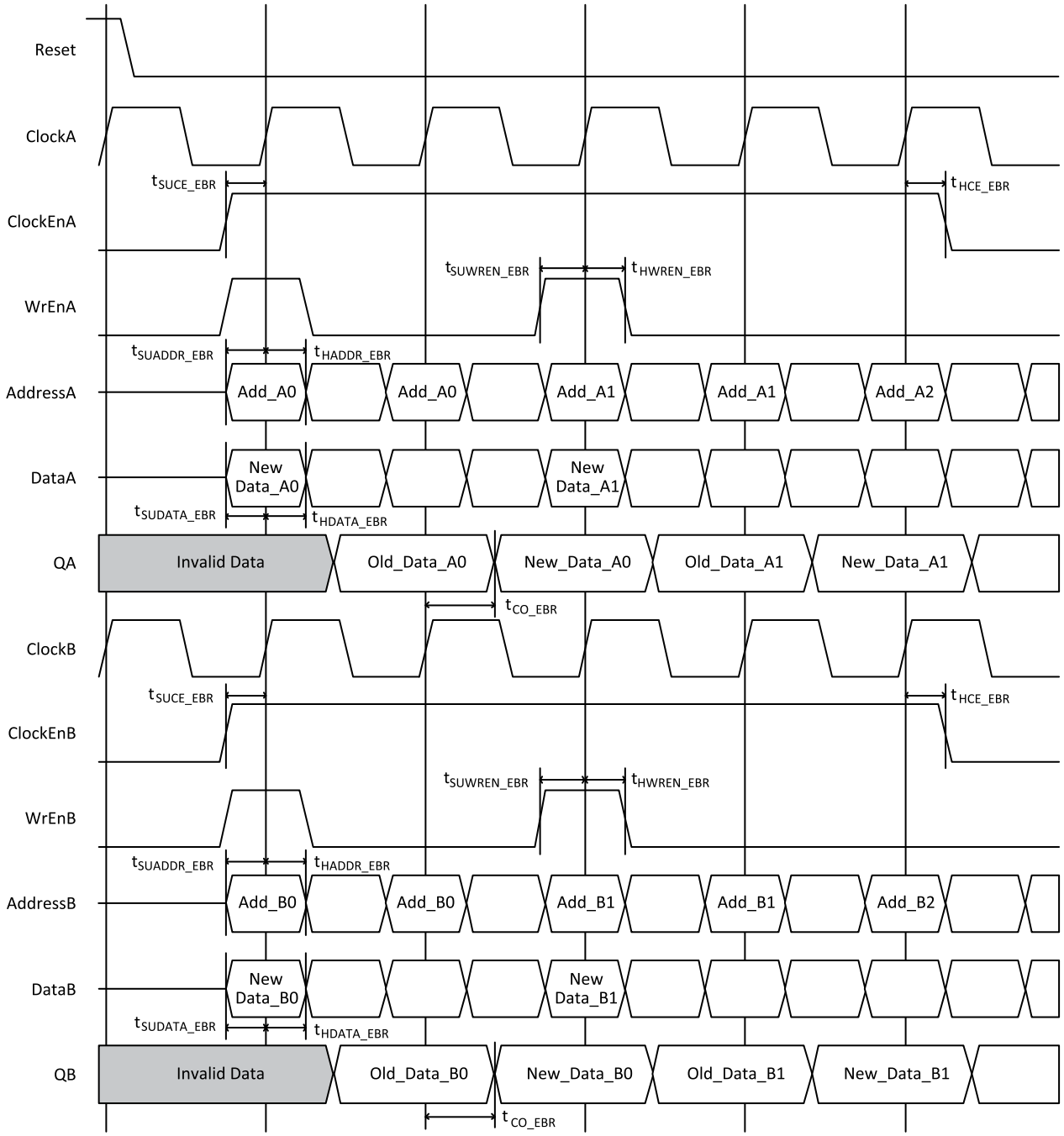

Figure 4.14. True Dual Port RAM Timing Waveform in Read before Write (READBEFOREWRITE) Mode without Output Registers

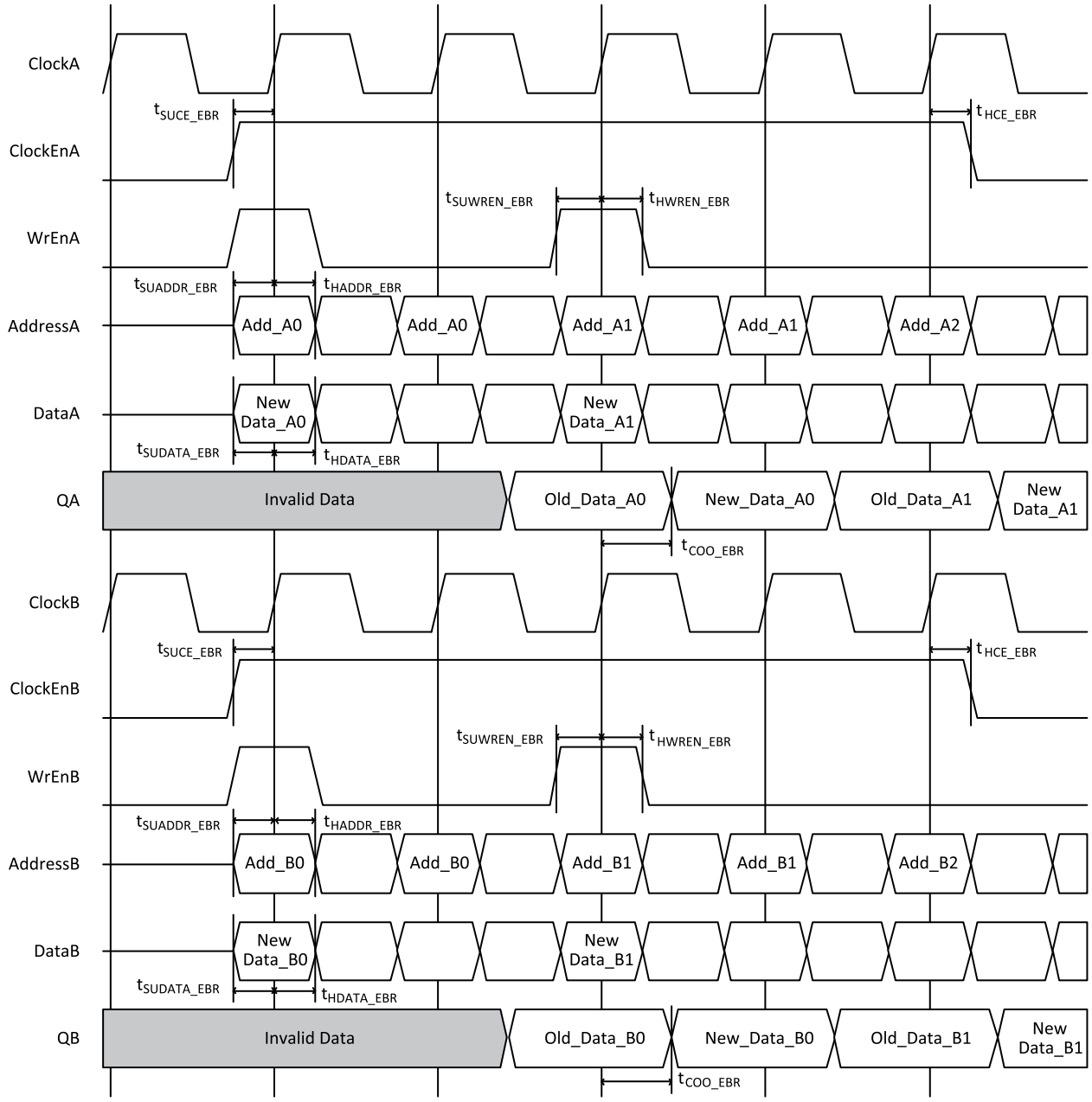

Figure 4.15. True Dual Port RAM Timing Waveform in Read before Write (READBEFOREWRITE) Mode with Output Registers

4.4. Pseudo Dual-Port RAM (RAM_DP) – EBR Based

CrossLink FPGA supports all the features of Pseudo-Dual Port Memory Module or RAM_DP. Clarity Designer allows you to generate the Verilog-HDL or VHDL along EDIF netlist for the memory size as per design requirement.

Clarity Designer generates the memory module shown in Figure 4.16.

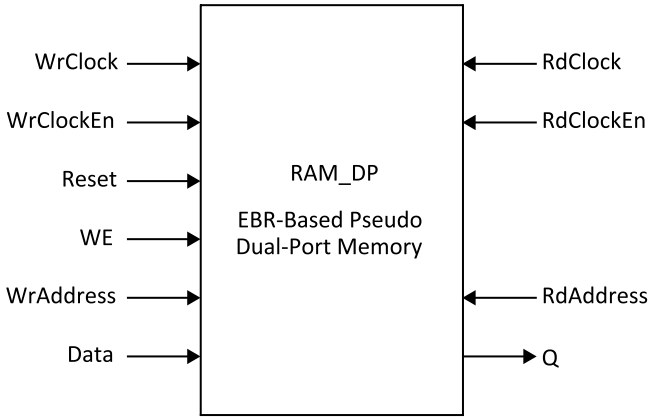

Figure 4.16. Pseudo Dual-Port Memory Module Generated by Clarity Designer

Figure 4.17 provides the primitive that can be instantiated for the Pseudo-Dual Port RAM. The primitive name is PDPW8KE and it can be directly instantiated in the code. Check the details on the port and port names under the primitives available in the cae_library/synthesis folder in Lattice Diamond software installation package.

Note that each EBR can accommodate 9 KB of memory; if the memory required is larger than 9 KB, then cascading can be done using the CS port (CSA and CSB in this case). See the A.3. CSDECODE section of Appendix A on how to perform cascading of multiple EBR primitives.

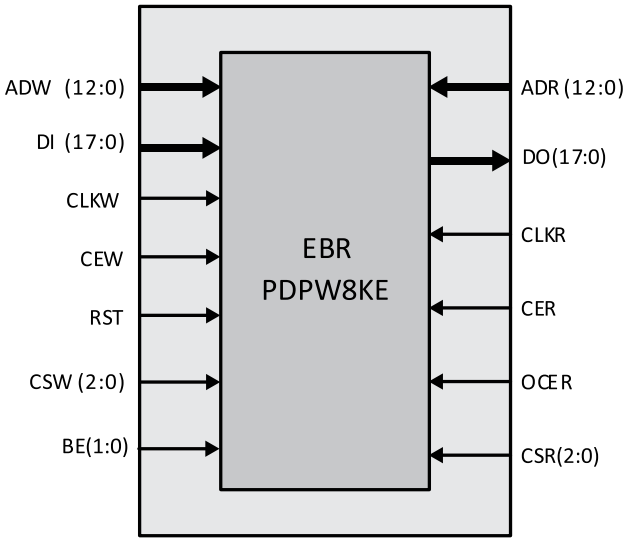

Figure 4.17. Pseudo-Dual Port RAM Primitive for CrossLink Devices

The various ports and their definitions for Pseudo Dual-Port memory are listed in Table 4.7. The table lists the corresponding ports for the module generated by Clarity Designer and for the EBR RAM_DP primitive.

Table 4.7. EBR-Based Pseudo Dual-Port Memory Port Definitions

Port Name in the Generated Module	Port Name in the EBR Block Primitive	Description
RdAddress	ADR[w:0]	Read Address
WrAddress	ADW[x:0]	Write Address
RdClock	CLKR	Read Clock
WrClock	CLKW	Write Clock
RdClockEn	CER	Read Clock Enable
WrClockEn	CEW	Write Clock Enable
Q	DO[y:0]	Read Data
Data	DI[z:0]	Write Data
WE	WE	Write Enable
Reset	RST	Reset
—	CS[2:0]	Chip Select
ByteEn	BE[1:0]	Byte Enable

Each EBR block consists of 18,432 bits of RAM. The values for address (w and x) and data (y and z) of each EBR block are listed in Table 4.8.

Table 4.8. Pseudo-Dual Port Memory Sizes for 9K Memory for CrossLink Devices

Dual Port Memory Size	Input Data Write Port	Output Data Read Port	Address Write Port	Address Read Port
8192 x 1	Data	Q	WrAddress(12:0)	RdAddress(12:0)
4096 x 2	Data(1:0)	Q(1:0)	WrAddress(11:0)	RdAddress(11:0)
2049 x 4	Data(3:0)	Q(3:0)	WrAddress(10:0)	RdAddress(10:0)
1024 x 9	Data(8:0)	Q(8:0)	WrAddress(9:0)	RdAddress(9:0)
512 x 18	Data(17:0)	Q(17:0)	WrAddress(8:0)	RdAddress(8:0)

Table 4.9 lists the various attributes available for the Pseudo Dual-Port Memory (RAM_DP). Some of these attributes are user-selectable through the Clarity Designer GUI.

Table 4.9. Pseudo Dual-Port RAM Attributes for CrossLink Devices

Configuration Tab	Description	Values	Default Value
Read Port Address Depth	Read Port Address depth of the read and write port	2 – 65536	512
Read Port Data Width	Read Port Data word width of the read and write port	1 – 256	18
Write Port Address Depth	Write Port Address depth of the read and write port	2 – 65536	512
Write Port Data Width	Write Port Data word width of the read and write port	1 – 256	18
Enable Output Register	Data Out port (Q) can or cannot be registered using this selection.	TRUE, FALSE	TRUE
Enable Output ClockEn	Clock Enable for the output clock (this option requires enabling Output Register)	TRUE, FALSE	FALSE
Byte Enables	Allows users to select Byte Enable options	TRUE, FALSE	FALSE
Byte Size	Byte Size selection when Byte Enable option is selected	9, 8	9
Reset Mode	Selection for the Reset to be synchronous or asynchronous to the Clock	Async, Sync	Sync
Reset Release	Selection for the release of Asynchronous Reset to be synchronous or asynchronous to the Clock	ASYNC, SYNC	SYNC
Optimization	Design optimizations to configure EBR for Speed or Area	Area, Speed	Speed
Initialization	Allows users to initialize their memories to all 1s, 0s or providing a custom initialization by providing a memory file	0s, 1s, File	0s
Memory File	When Memory file is selected, user can browse to the memory file for custom initialization of RAM.	—	—
Memory File Format	Allows users to select if the memory file is formatted as Binary, Hex or address Hex. (See Appendix A for details on different formats)	Binary, Hex, Addressed Hex	Binary
Enable ECC	Allows users to enable Error Correction Codes. This option is not available for memory that is wider than 64 bits.	TRUE, FALSE	FALSE

Users have the option to enable the output registers for Pseudo-Dual Port RAM (RAM_DP). Waveforms in the following figures show the internal timing waveforms for Pseudo-Dual Port RAM (RAM_DP) with these options.

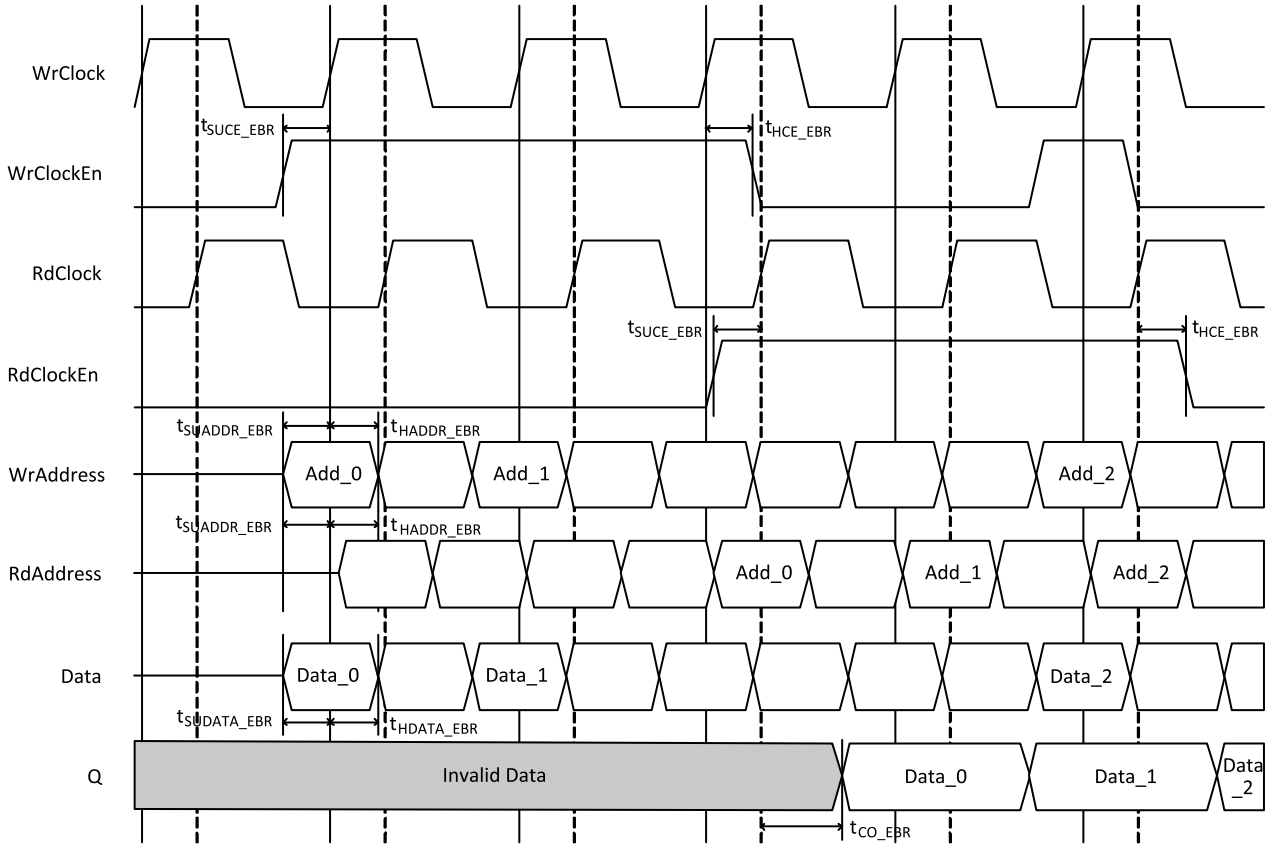

Figure 4.18. Pseudo Dual Port RAM Timing Diagram without Output Registers

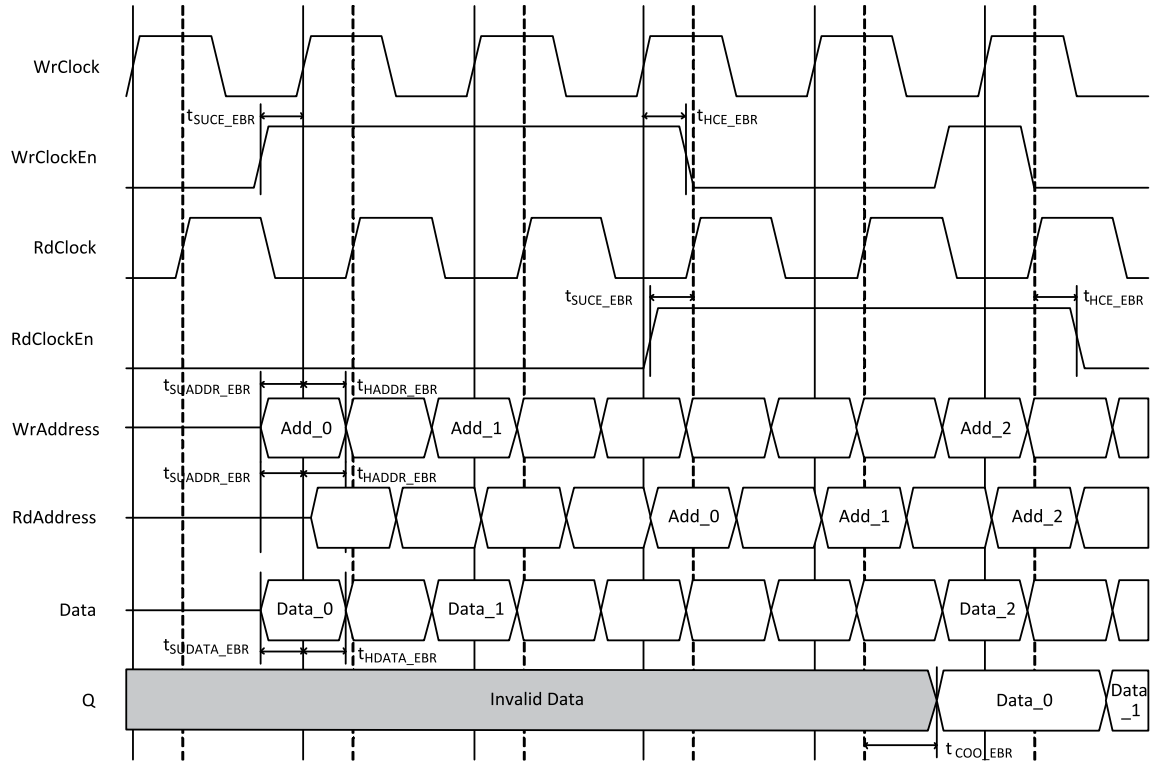

Figure 4.19. Pseudo Dual Port RAM Timing Diagram with Output Registers

4.5. Read Only Memory (ROM) – EBR Based

CrossLink FPGA supports all the features of ROM Memory Module or ROM. Clarity Designer allows you to generate the Verilog-HDL, VHDL or EDIF netlist for the memory size as per design requirement. You are required to provide the ROM memory content in the form of an initialization file.

Clarity Designer generates the memory module shown in Figure 4.20.

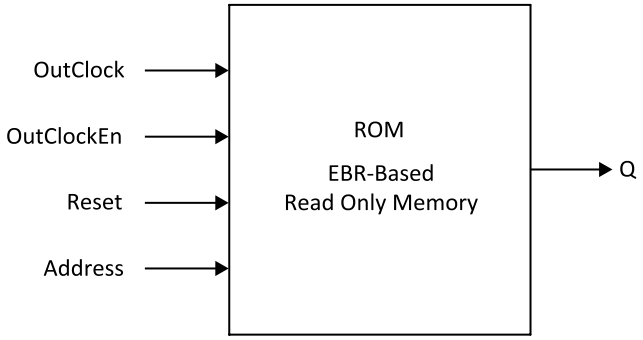

Figure 4.20. Read Only Memory Module Generated by Clarity Designer

The various ports and their definitions are listed in Table 4.10. The table lists the corresponding ports for the module generated by Clarity Designer and for the ROM primitive.

Table 4.10. EBR-Based ROM Port Definitions

Port Name in Generated Module	Port Name in the EBR Block Primitive	Description
Address	AD[x:0]	Read Address
OutClock	CLK	Clock
OutClockEn	CE	Clock Enable
Reset	RST	Reset
—	CS[2:0]	Chip Select

When generating ROM using Clarity Designer, the designer must provide the initialization file to pre-initialize the contents of the ROM. These files are the *.mem files and they can be of binary, hex or addressed hex formats. The initialization files are discussed in detail in the Initializing Memory section on page 63.

Each EBR block consists of 18,432 bits of RAM. The values for x (for address) and y (data) for each EBR block of the devices are included in Table 4.11.

Table 4.11. ROM Memory Sizes for 9K Memory for CrossLink Devices

Dual Port Memory Size	Output Data Read Port	Address Port
8192 x 1	Q	Address(12:0)
4096 x 2	Q(1:0)	Address(11:0)
2049 x 4	Q(3:0)	Address(10:0)
1024 x 9	Q(8:0)	Address(9:0)
512 x 18	Q(17:0)	Address(9:0)

Table 4.12 describes the various attributes available for the Read Only Memory (ROM). Some of these attributes are user-selectable through the Clarity Designer GUI. For detailed attribute definitions, see Appendix A. Attribute Definitions.

Table 4.12. ROM Attributes for CrossLink Devices

Configuration Tab Attributes	Description	Values	Default Value
Address Depth	Address depth of the read and write port	2 – 65536	512
Data Width	Data word width of the Read and write port	1 – 256	36
Enable Output Register	Data Out port (Q) can be registered or not using this selection.	TRUE, FALSE	TRUE
Optimization	Design optimizations to configure EBR for Speed or Area.	Area, Speed	Speed
Reset Mode	Selection for the Reset to be Synchronous or Asynchronous to the Clock.	Async, Sync	Sync
Reset Release	Selection for the release of Asynchronous Reset to be Synchronous or Asynchronous to the Clock.	ASYNC, SYNC	SYNC
Initialization	Allows users to initialize their memories to all 1s, 0s or providing a custom initialization by providing a memory file.	0s, 1s, File	0s
Memory File	When Memory file is selected, used can browse to the memory file for custom initialization of RAM.	—	—
Memory File Format	This option allows users to select if the memory file is formatted as Binary, Hex or address Hex. (See Appendix A for details on different formats).	Binary, Hex, Addressed Hex	Binary
Enable ECC	Option allows users to enable Error Correction Codes. This option is not available for memory that is wider than 64 bits.	TRUE, FALSE	FALSE

Users have the option to enable the output registers for Read Only Memory (ROM). Figure 4.21 and Figure 4.22 show the internal timing waveforms for ROM with these options.

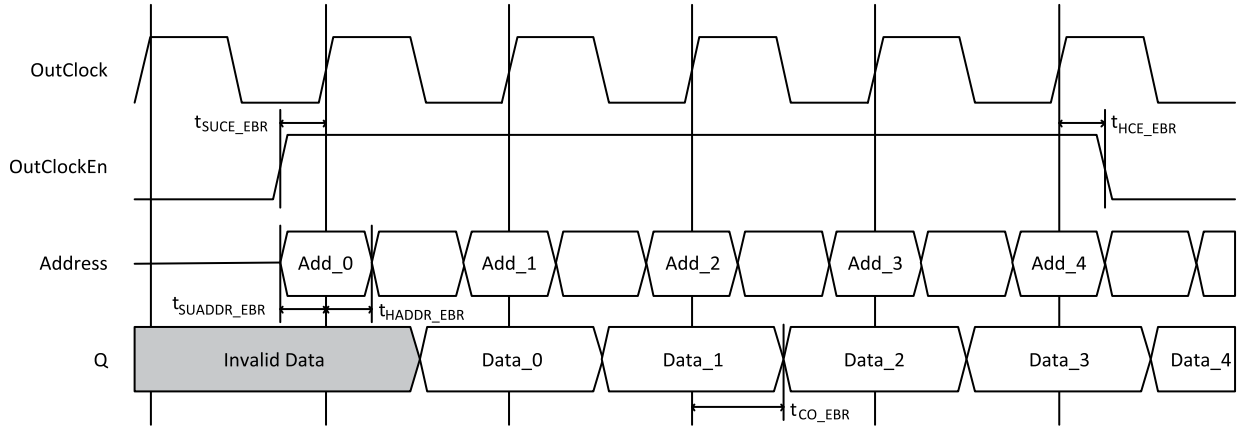

Figure 4.21. ROM Timing Waveform without Output Registers

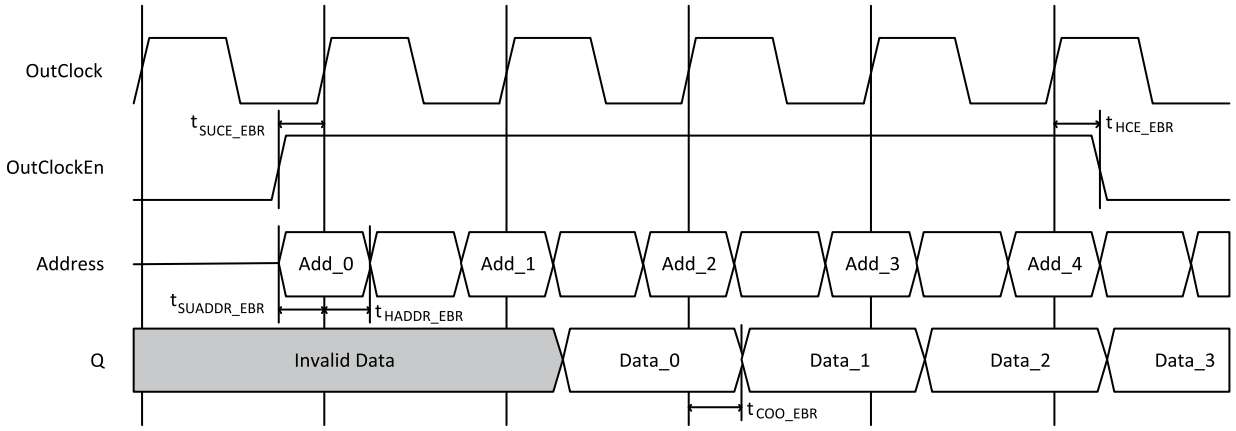

Figure 4.22. ROM Timing Waveform with Output Registers

4.6. First In First Out (FIFO) Memory

CrossLink devices support two different types of FIFOs:

- Single Clock FIFO (FIFO)
- Dual Clock FIFO (FIFO_DC)

The FIFOs in CrossLink devices are emulated such that they are built using Dual Port RAMs with additional logic around them. The additional logic (address counters, flag logic and pointer comparators) are implemented using PFUs around the Dual Port RAMs.

The EBR blocks in CrossLink devices can be configured as LUT based or EBR based, as well as Single Clock First-In First-Out Memory (FIFO) or Dual-Clock First-In First-Out Memory (FIFO_DC). Clarity Designer allows you to generate the Verilog-HDL or VHDL netlist for various memory sizes depending on design requirements.

Clarity Designer generated FIFO modules and their operations are discussed in detail in the following pages.

4.6.1. Single Clock FIFO (FIFO) – EBR and LUT

Figure 4.23 shows the module generated by Clarity Designer for FIFO.

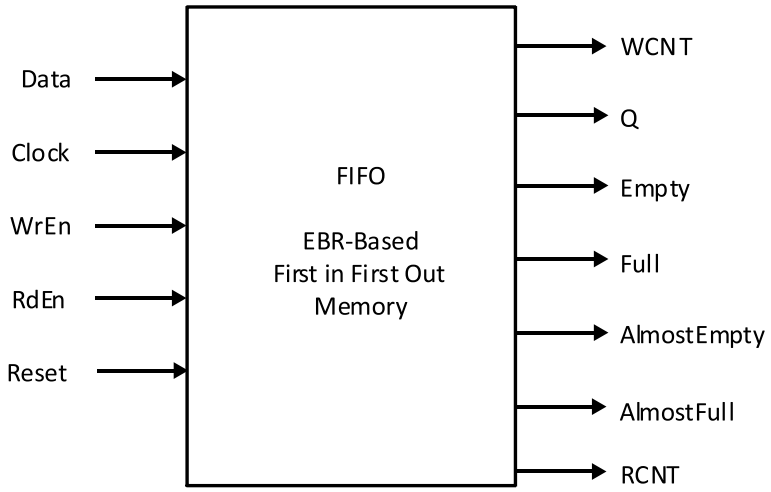

Figure 4.23. FIFO Module Generated by Clarity Designer

The various ports and their definitions for the FIFO are listed in Table 4.13.

Table 4.13. Port Names and Definitions for FIFO

Port Name in Generated Module	Description	Active State
Data	Input Data	—
Clock	Clock	Rising Clock Edge
WrEn	Write Enable	Active High
RdEn	Read Enable	Active High
Reset[1]	Reset	Active High
Q	Data Output	—
Empty	Empty Flag Active High	
Full	Full Flag Active High	
AlmostEmpty	Almost Empty Flag	Active High
AlmostFull	Almost Full Flag	Active High
ERROR[2]	Error Check Code	Active High
WCNT	Data count synchronized with Write Clock	—
RCNT	Data count synchronized with Read Clock	—

Notes:

1. Resets only the optional output registers, pointer circuitry and flags (except Empty, which is set to '1') of the FIFO. It does not reset the input registers or the contents of memory.
2. Optional port.

Let us first discuss the non-pipelined or the FIFO without output registers. Figure 4.24 shows the operation of the FIFO when it is empty and data starts to be written into it.

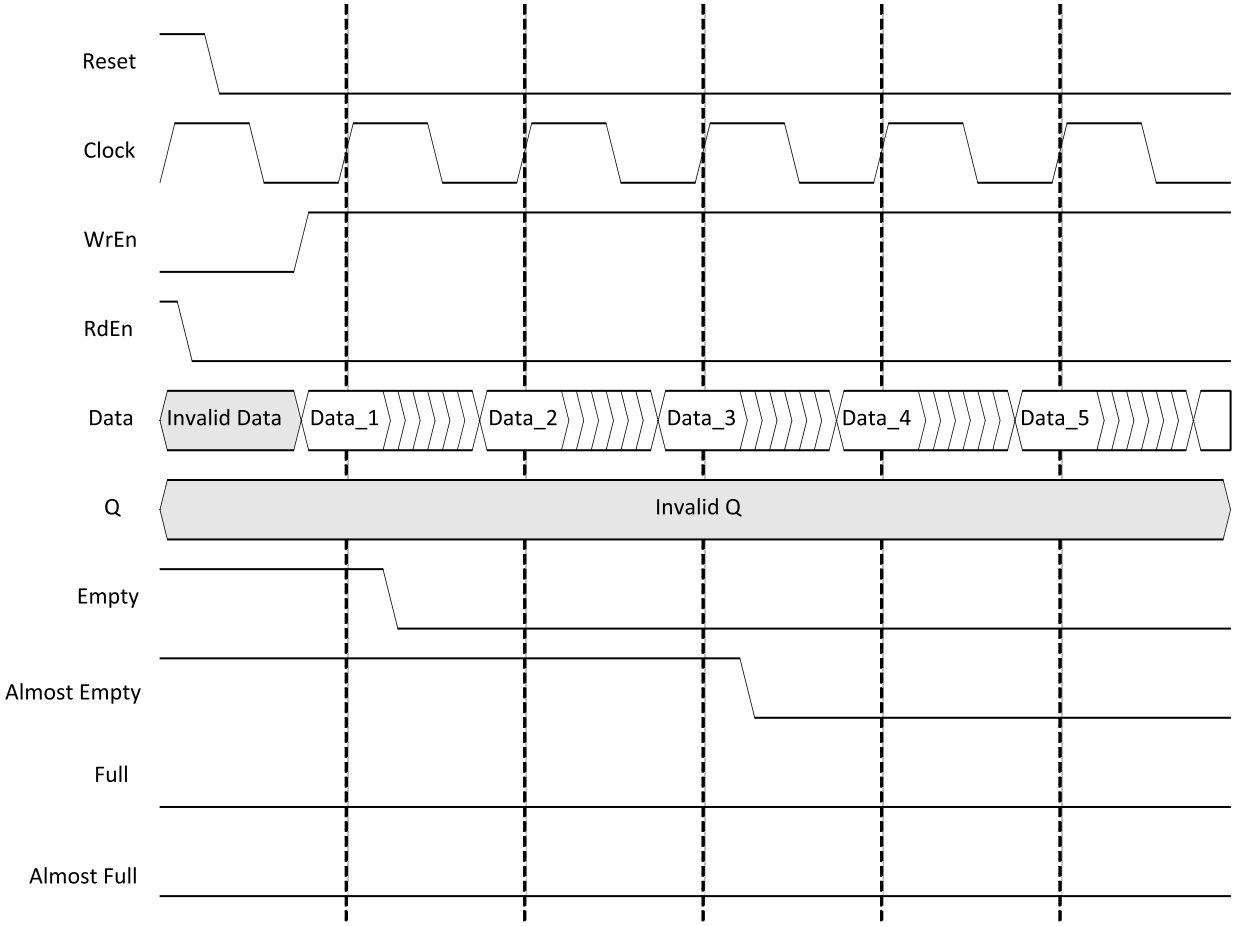

Figure 4.24. FIFO without Output Registers, Start of Data Write Cycle

When writing into the FIFO starts, the WrEn signal must be high. The Empty and Almost Empty flags are likewise high while the Full and Almost Full are low.

When the first data is written into the FIFO, the Empty flag de-asserts (or goes low) since the FIFO is no longer empty. In this figure, the assumed Almost Empty flag setting is 3 (address location 3). The Almost Empty flag is therefore deasserted when the third address location is filled.

If writing into the FIFO is continued, the FIFO is filled and the Almost Full and Full flags are asserted. Figure 4.25 shows the behavior of these flags. In this figure, it is assumed that the FIFO depth is 'N'.

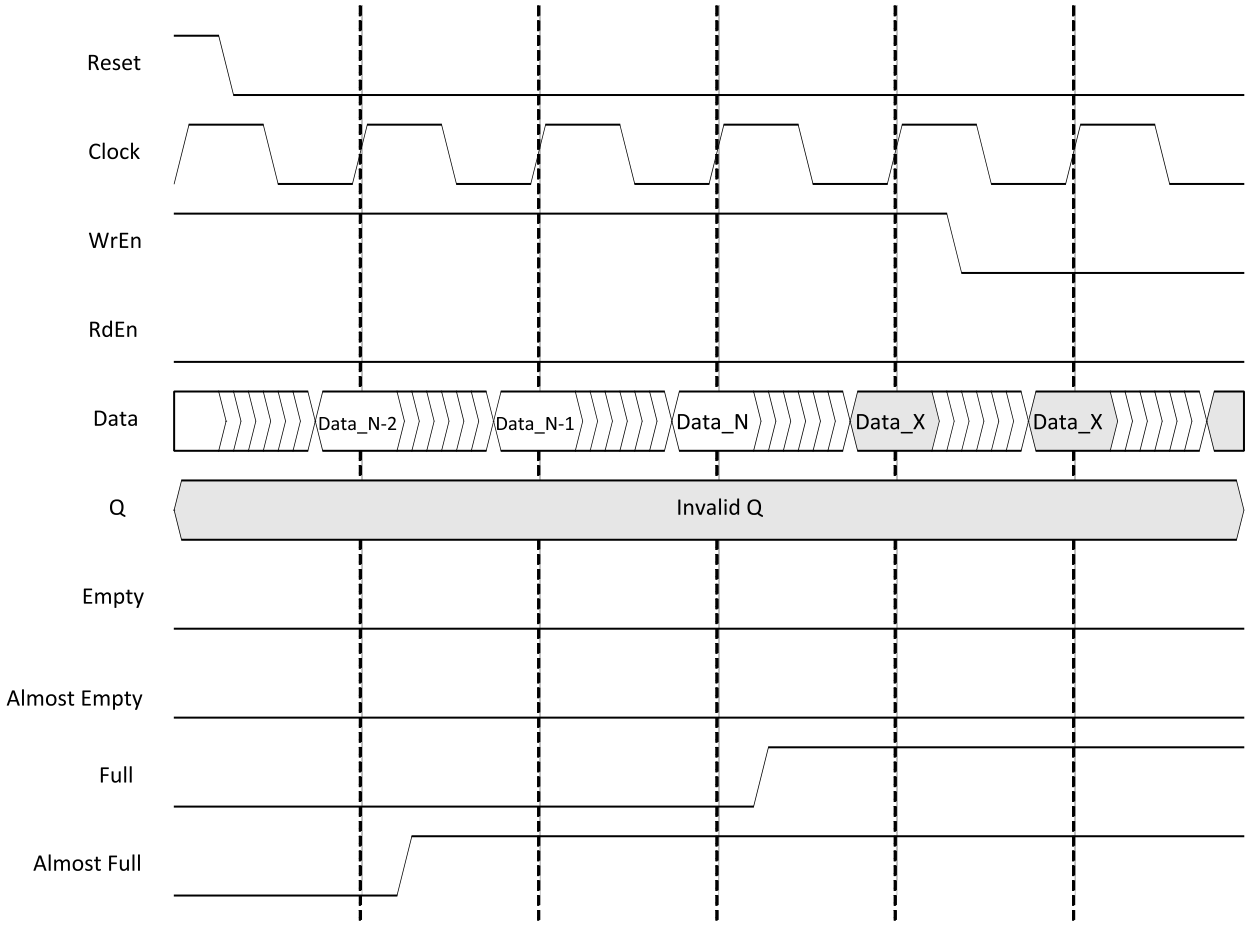

Figure 4.25. FIFO without Output Registers, End of Data Write Cycle

In Figure 4.25, the Almost Full flag is two locations before the FIFO is filled. The Almost Full flag is asserted when the N-2 location is written, and the Full flag is asserted when the last word is written into the FIFO.

Data_X data inputs are not written since the FIFO is full (Full flag is high).

When the contents of the FIFO are read out, the waveforms are shown in Figure 4.26. At the start of the read cycle, RdEn goes high and the data read starts. The Full and Almost Full flags are de-asserted.

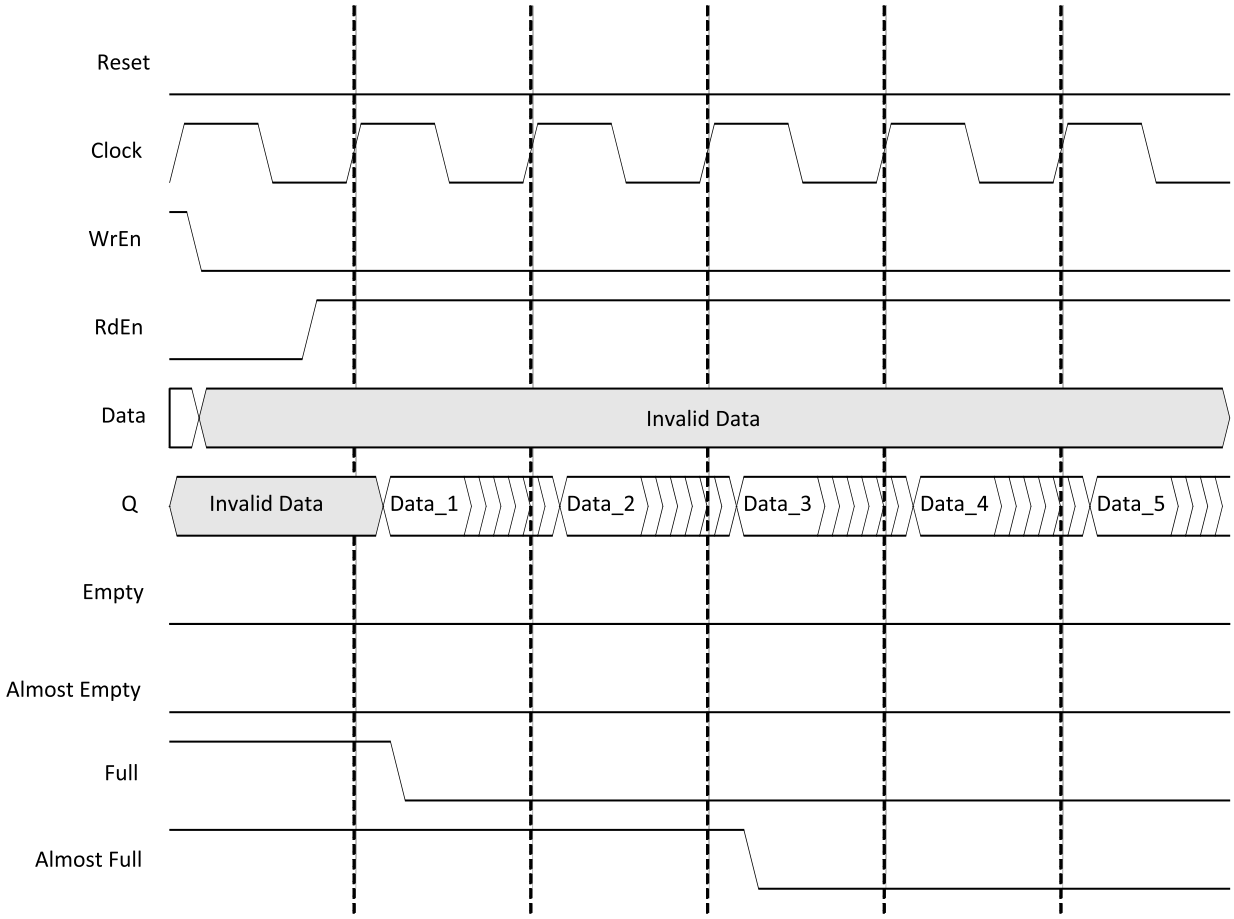

Figure 4.26. FIFO without Output Registers, Start of Data Read Cycle

Similarly, as the data is read out and FIFO is emptied, the Almost Empty and Empty flags are asserted. See Figure 4.27.

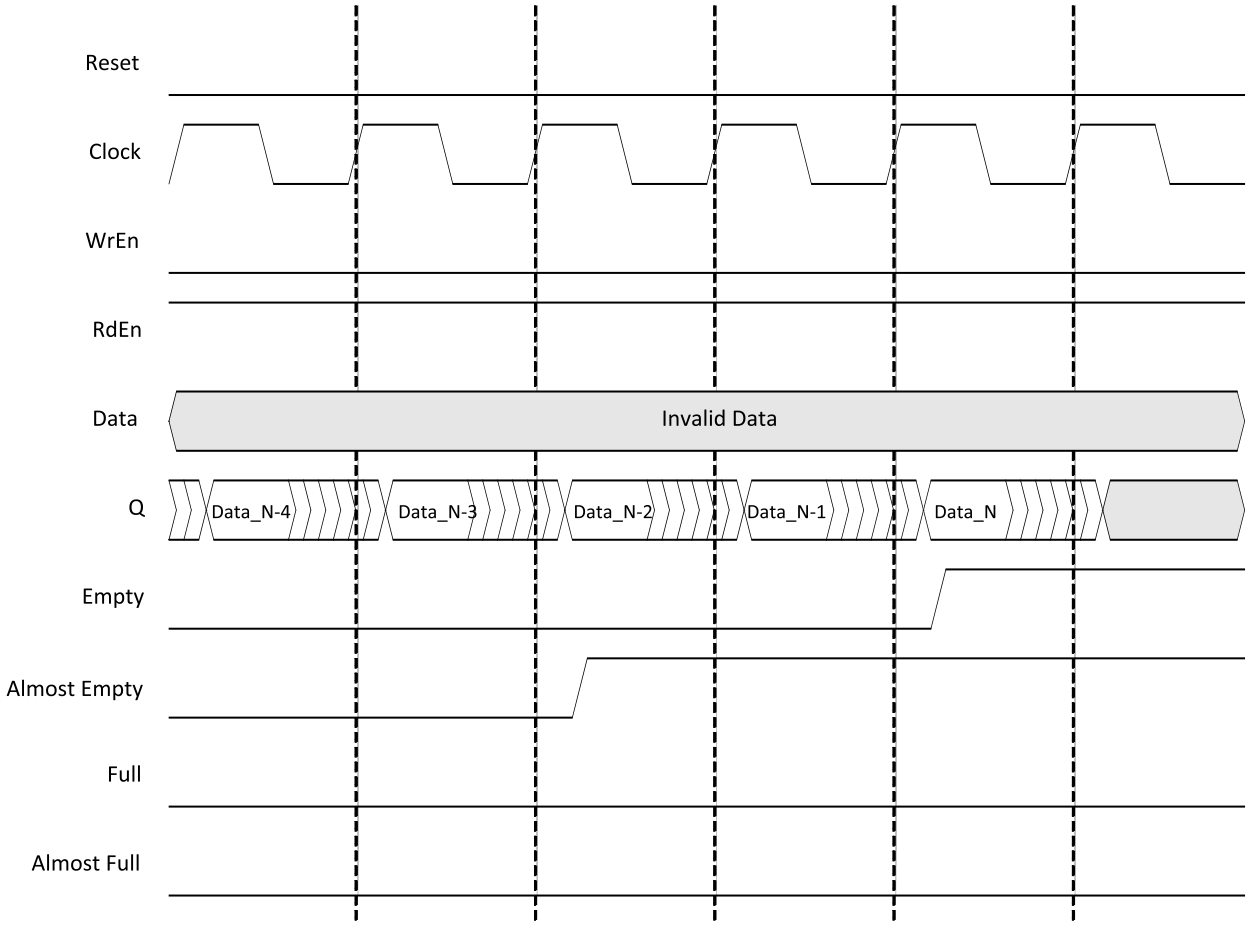

Figure 4.27. FIFO without Output Registers, End of Data Read Cycle

Figure 4.23 to Figure 4.27 show the behavior of non-pipelined FIFO or FIFO without output registers. When the registers are pipelined, the output data is delayed by one clock cycle. There is also an option for output registers to be enabled by the RdEn signal.

Figure 4.28 to Figure 4.31 show similar waveforms for the FIFO with an output register and an output register enable with RdEn. Note that flags are asserted and de-asserted with similar timing to the FIFO without output registers. Only the data out 'Q' is delayed by one clock cycle.

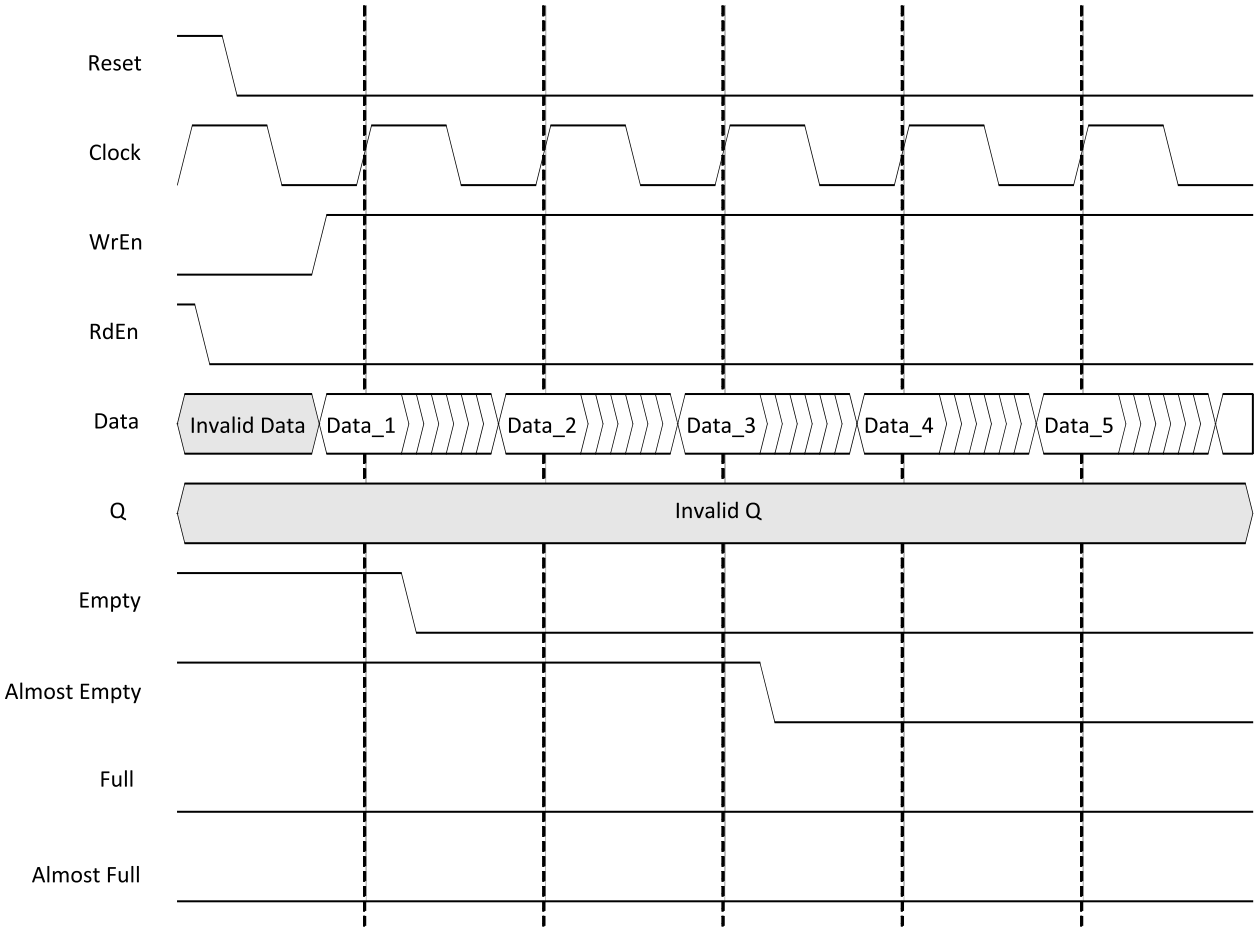

Figure 4.28. FIFO with Output Registers, Start of Data Write Cycle

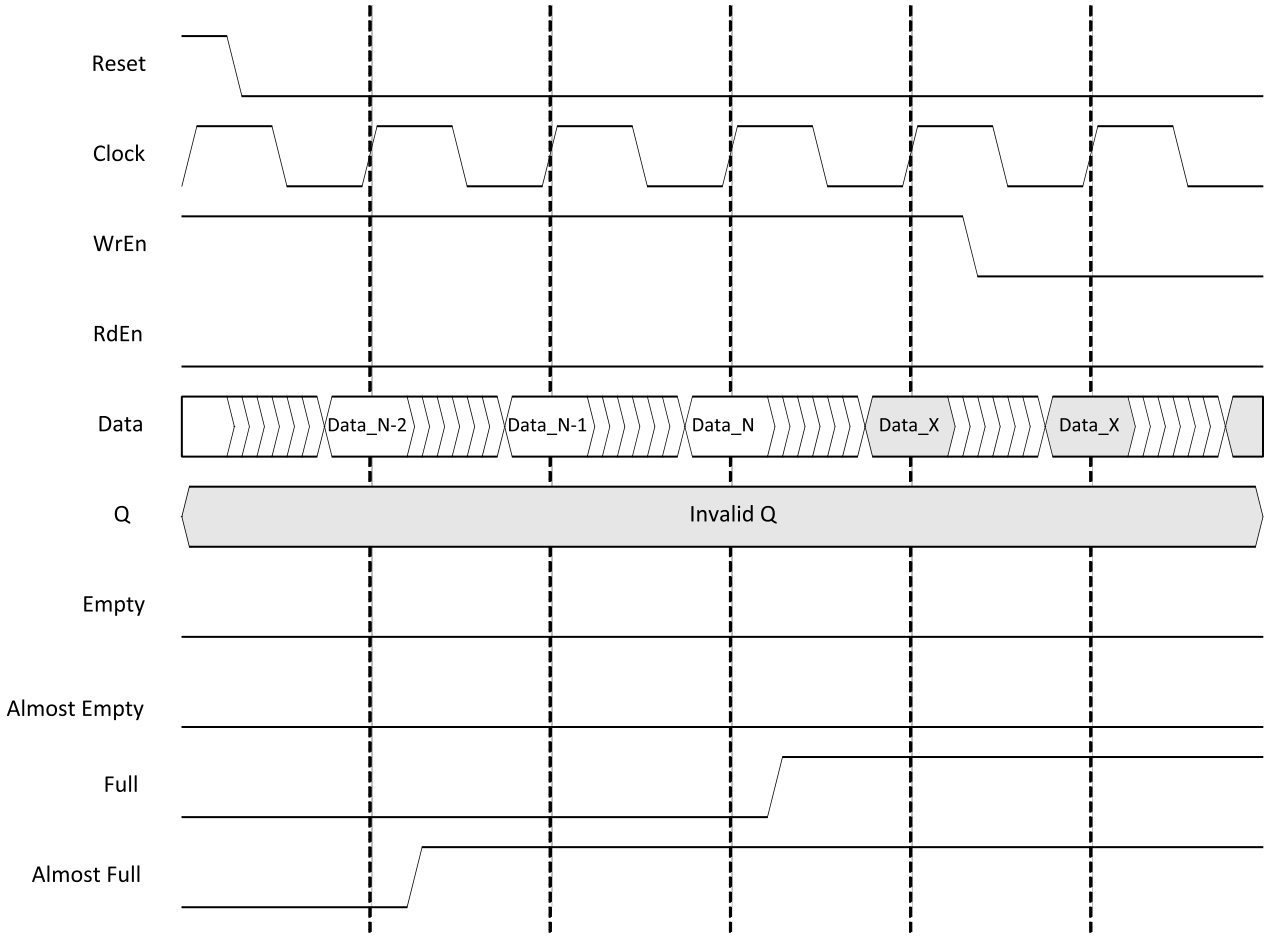

Figure 4.29. FIFO with Output Registers, End of Data Write Cycle

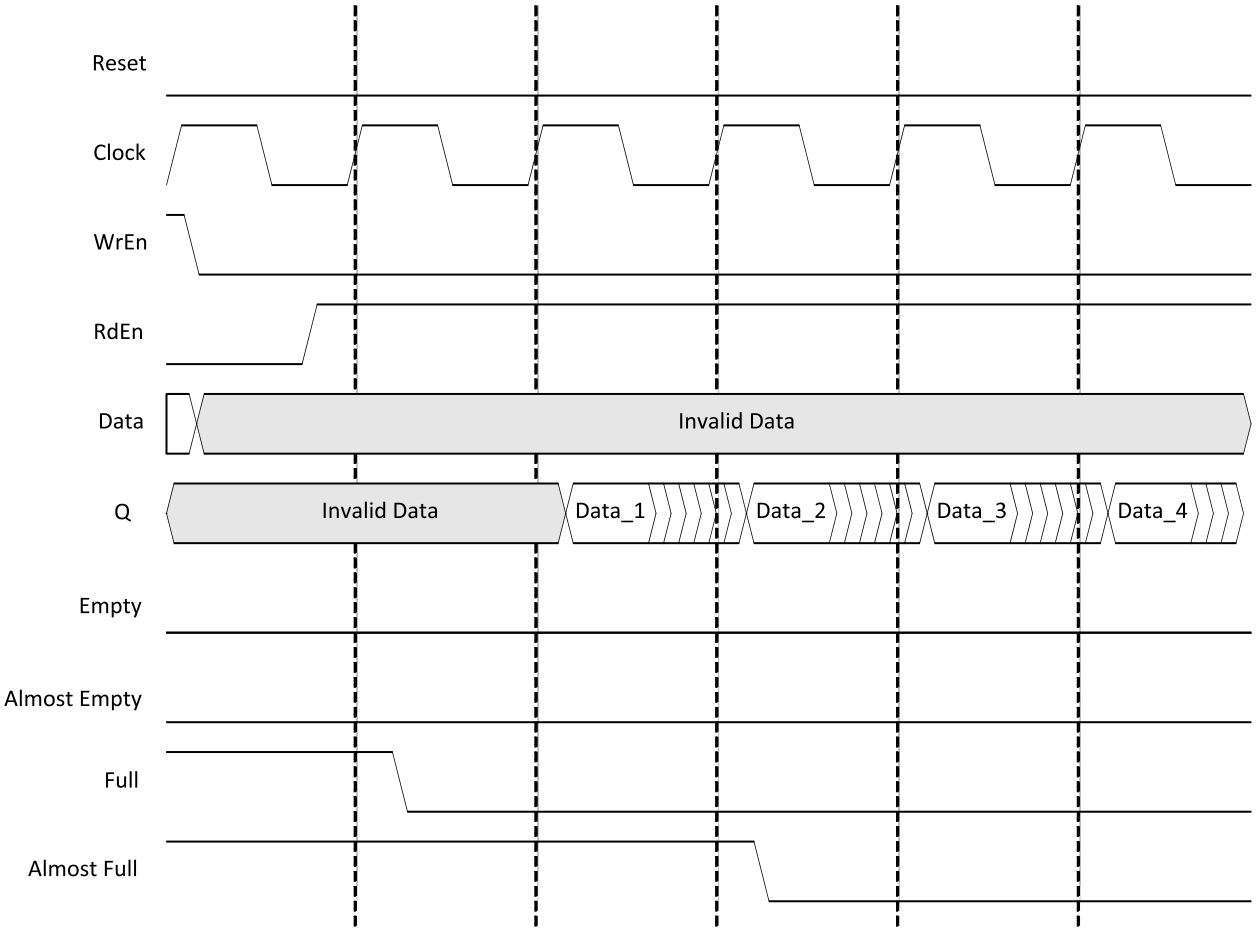

Figure 4.30. FIFO with Output Registers, Start of Data Read Cycle

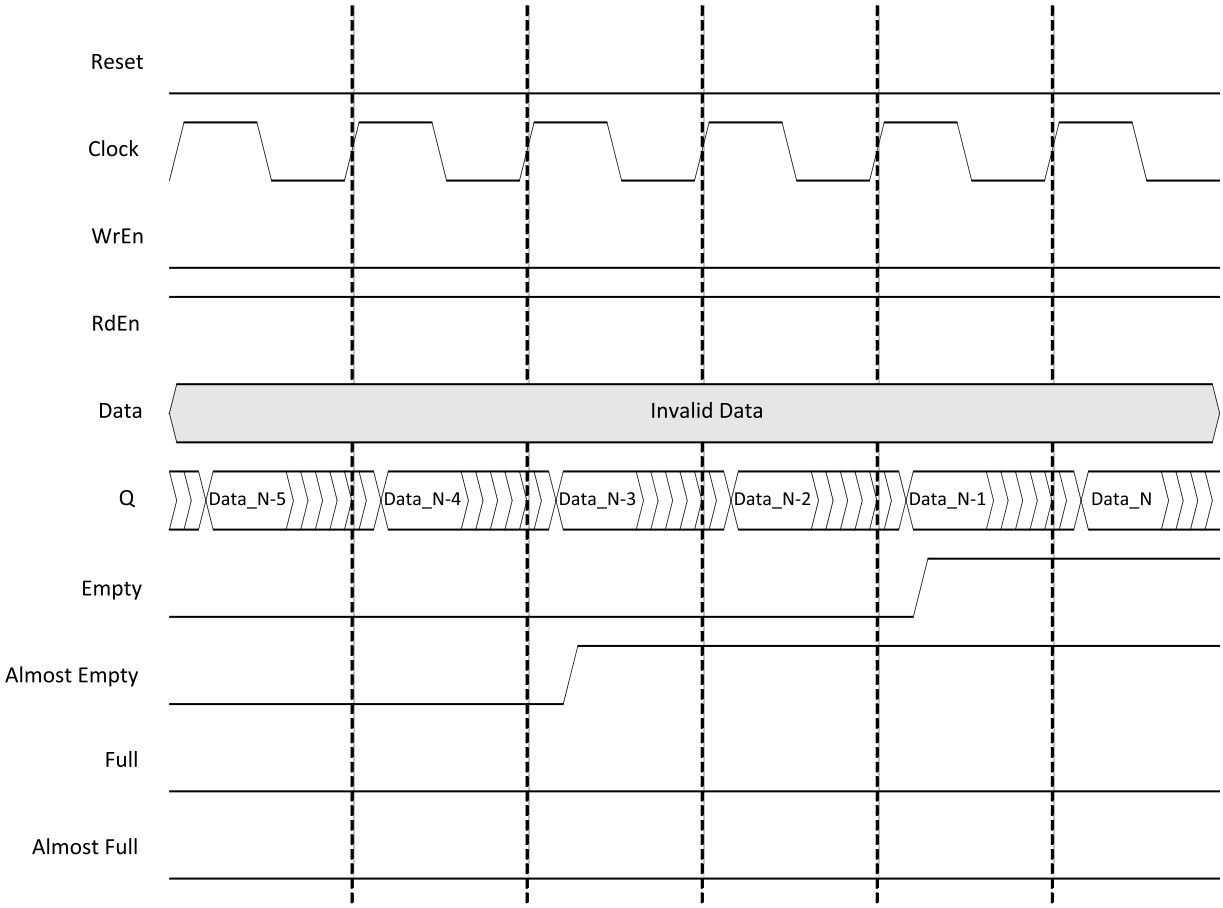

Figure 4.31. FIFO with Output Registers, End of Data Read Cycle

If the enable output register with RdEn option is selected, the data out is still delayed by one clock cycle (as compared to the non-pipelined FIFO). The RdEn should also be high during that clock cycle, otherwise the data takes an extra clock cycle when the RdEn goes true.

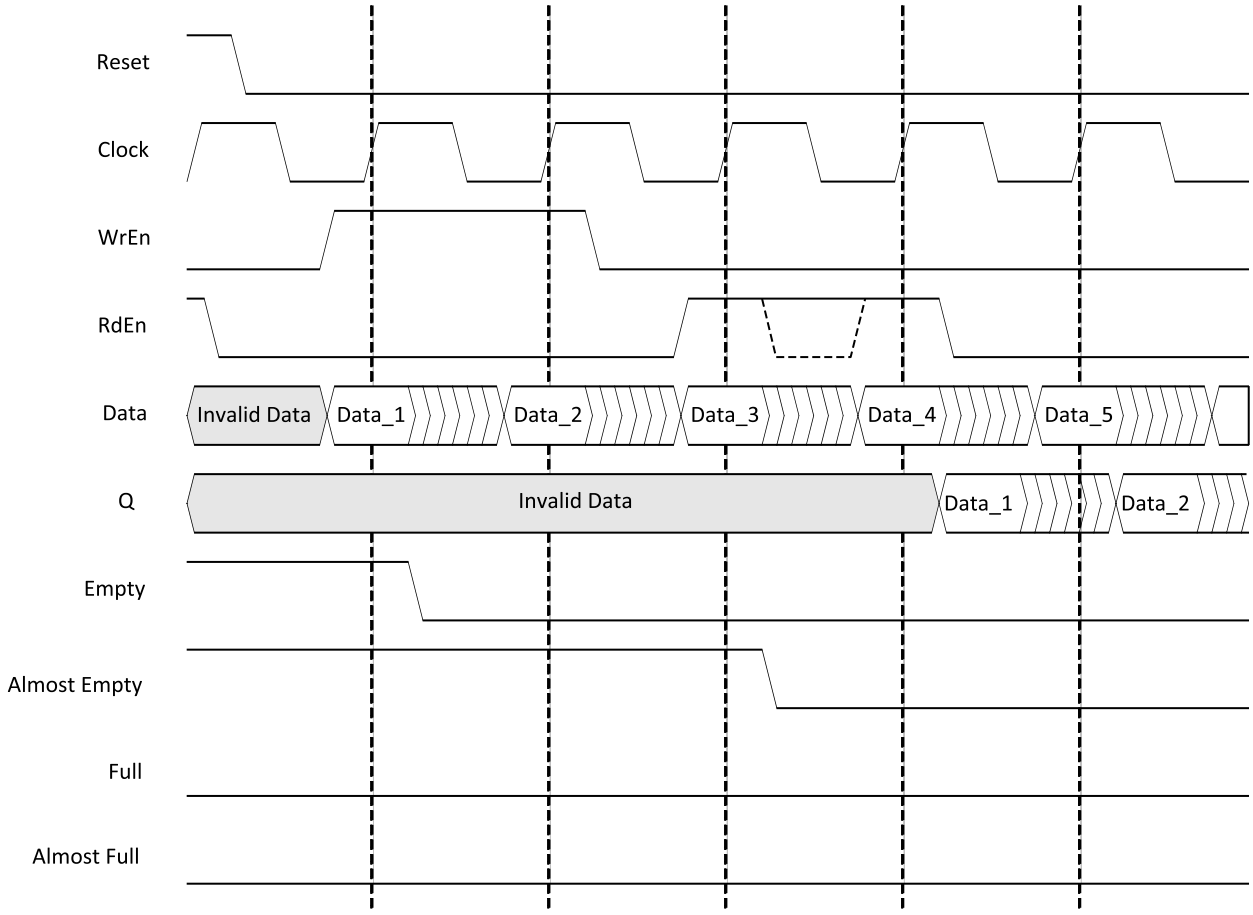

Figure 4.32. FIFO with Output Registers and RdEn on Output Registers

4.6.2. Dual Clock First-In-First-Out (FIFO_DC) – EBR or LUT Based

Figure 4.33 shows the module that is generated by the Clarity Designer for FIFO.

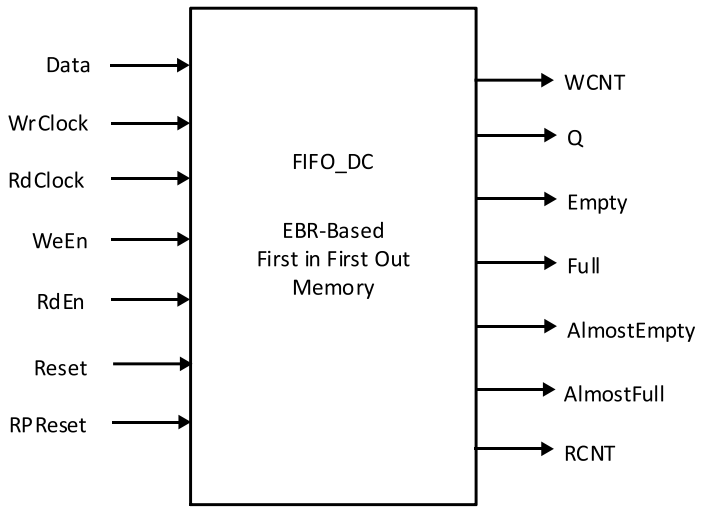

Figure 4.33. FIFO_DC Module generated by the Clarity Designer

The various ports and their definitions for the FIFO_DC are listed in Table 4.14.

Table 4.14. Port Names and Definitions for FIFO_DC

Port Name in Generated Module	Description	Active State
Data	Input Data	—
WrClock	Write Port Clock	Rising Clock Edge
RdClock	Read Port Clock	Rising Clock Edge
WrEn	Write Enable	Active High
RdEn	Read Enable	Active High
Reset[2]	Reset	Active High
RPReset[3]	Read Pointer Reset	Active High
Q	Data Output	—
Empty	Empty Flag Active High	
Full	Full Flag Active High	
AlmostEmpty	Almost Empty Flag	Active High
AlmostFull	Almost Full Flag	Active High
ERROR[1]	Error Check Code	Active High
WCNT	Data count synchronized with Write Clock	—
RCNT	Data count synchronized with Read Clock	—

Notes:

1. Optional port
2. Resets only the optional output registers, pointer circuitry and flags of the FIFO. It does not reset the input registers or the contents of memory.
3. Resets only the read pointer. See the sections below for more information.

4.6.3. FIFO_DC Flags

As an emulated FIFO, FIFO_DC requires the flags to be implemented in the FPGA logic around the block RAM. Because of the two clocks, the flags are required to change clock domains from read clock to write clock and vice versa. This adds latency to the flags either during assertion or de-assertion. Latency can be avoided only in one of the cases (either assertion or de-assertion) or distributed between the two.

In the emulated FIFO_DC, there is no latency during assertion of the flags. Thus, when the flag goes true, there is no latency. However, due to the design of the flag logic running on two clock domains, there is latency during de-assertion.

If we start to write into the FIFO_DC to fill it, the write operation is controlled by WrClock and WrEn. However, it takes extra RdClock cycles for the de-assertion of the Empty and Almost Empty flags.

De-assertion of Full and Almost Full although result of reading out the data from the FIFO_DC, takes extra WrClock cycles after reading the data for these flags to come out.

The waveforms for FIFO_DC without output registers shown in Figure 4.34. The waveforms show the operation of the FIFO_DC when it is empty and the data starts to be written into it.

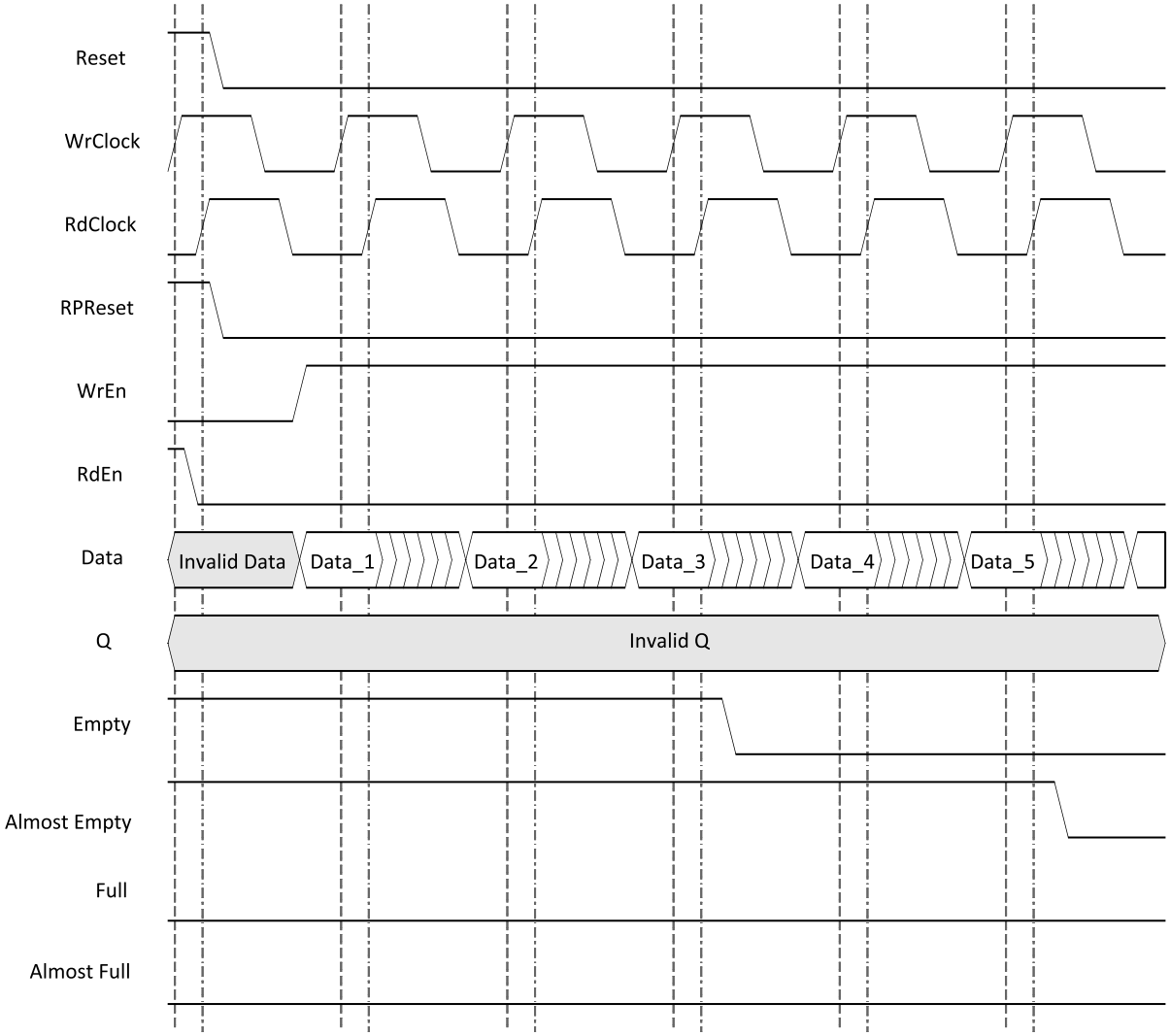

Figure 4.34. FIFO_DC without Output Registers, Start of Data Write Cycle

When writing into the FIFO_DC starts, the WrEn signal must be high. The Empty and Almost Empty flags are likewise high while the Full and Almost Full are low.

When the first data is written into the FIFO_DC, the Empty flag de-asserts (or goes low) since the FIFO_DC is no longer empty. In this figure, the assumed Almost Empty flag setting is 3 (address location 3). The Almost Empty flag is therefore de-asserted when the third address location is filled.

If writing into the FIFO is continued, the FIFO_DC is filled and the Almost Full and Full flags are asserted. Figure 4.35 shows the behavior of these flags. In this figure, it is assumed that the FIFO_DC depth is 'N'.

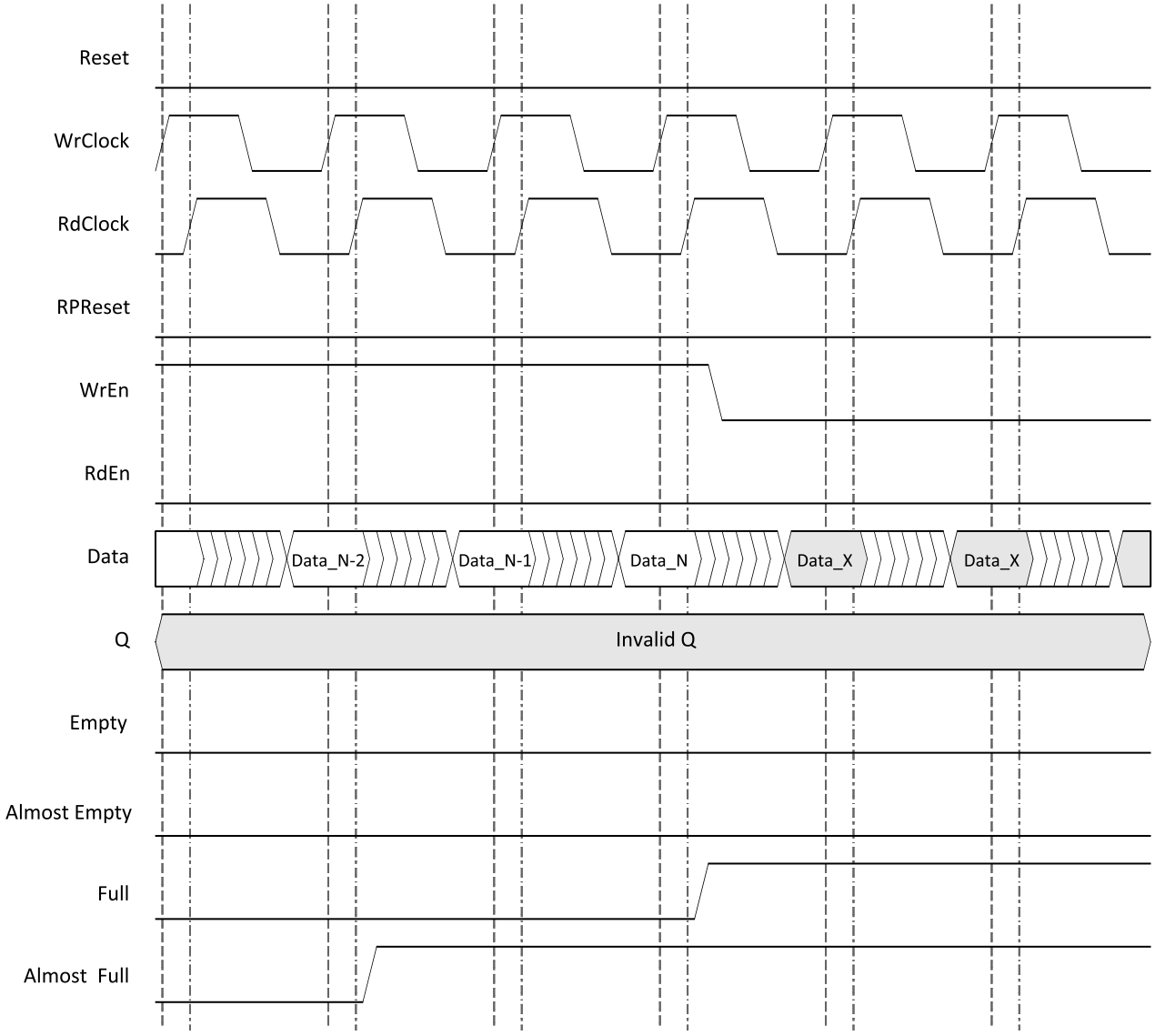

Figure 4.35. FIFO_DC without Output Registers, End of Data Write Cycle

In Figure 4.35, the Almost Full flag is two locations before the FIFO_DC is filled. The Almost Full flag is asserted when the N-2 location is written, and the Full flag is asserted when the last word is written into the FIFO_DC.

Data_X data inputs are not written since the FIFO_DC is full (Full flag is high).

Note that the assertion of these flags is immediate and there is no latency when they go true.

The waveforms when the contents of the FIFO_DC are read out are shown in Figure 4.36. The waveforms show the start of the read cycle. RdEn goes high and the data read starts. The Full and Almost Full flags are de-asserted.

Note that the de-assertion is delayed by two clock cycles.

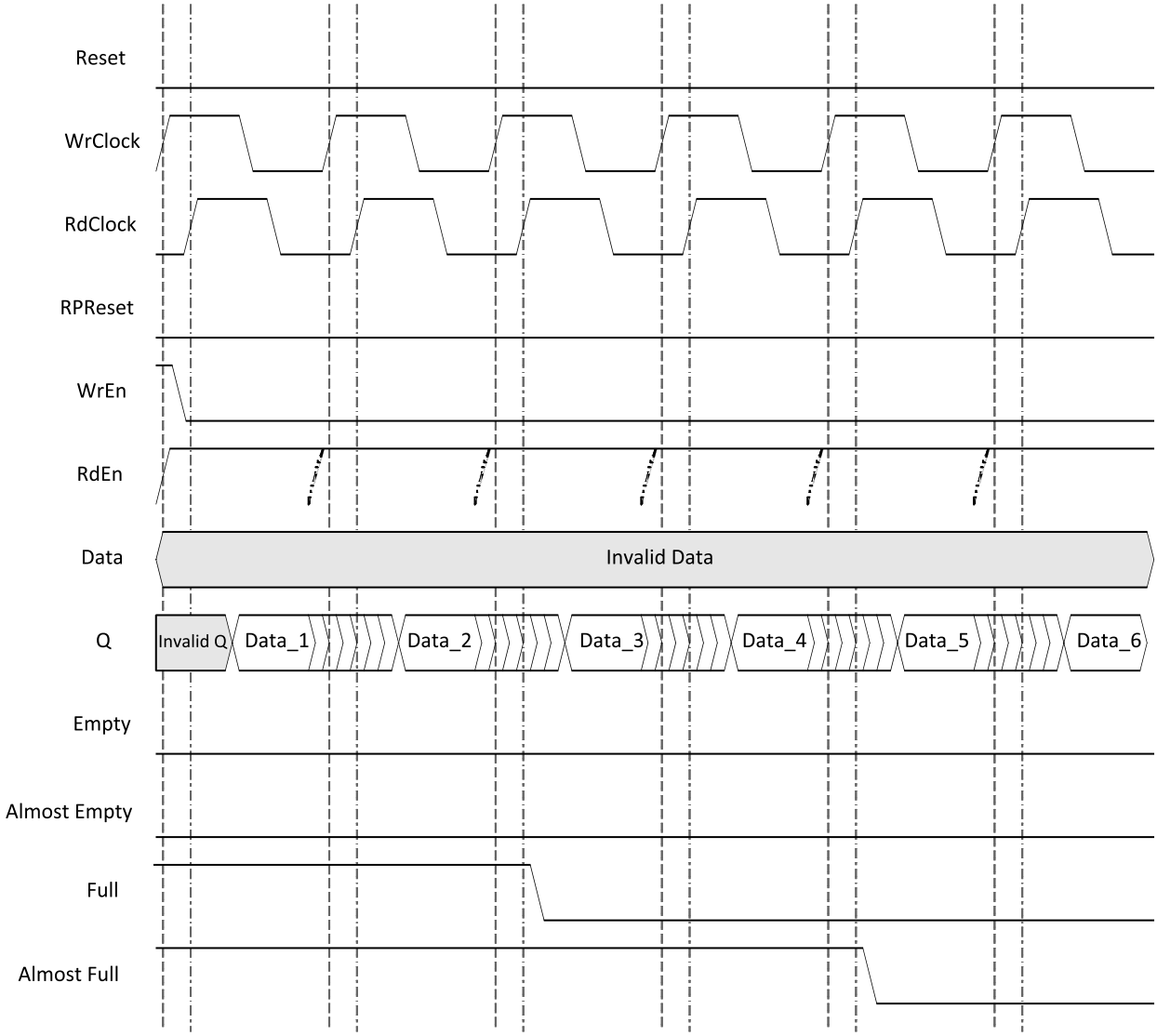

Figure 4.36. FIFO_DC without Output Registers, Start of Data Read Cycle

Similarly, as the data is read out and FIFO_DC is emptied, the Almost Empty and Empty flags are asserted. See Figure 4.37.

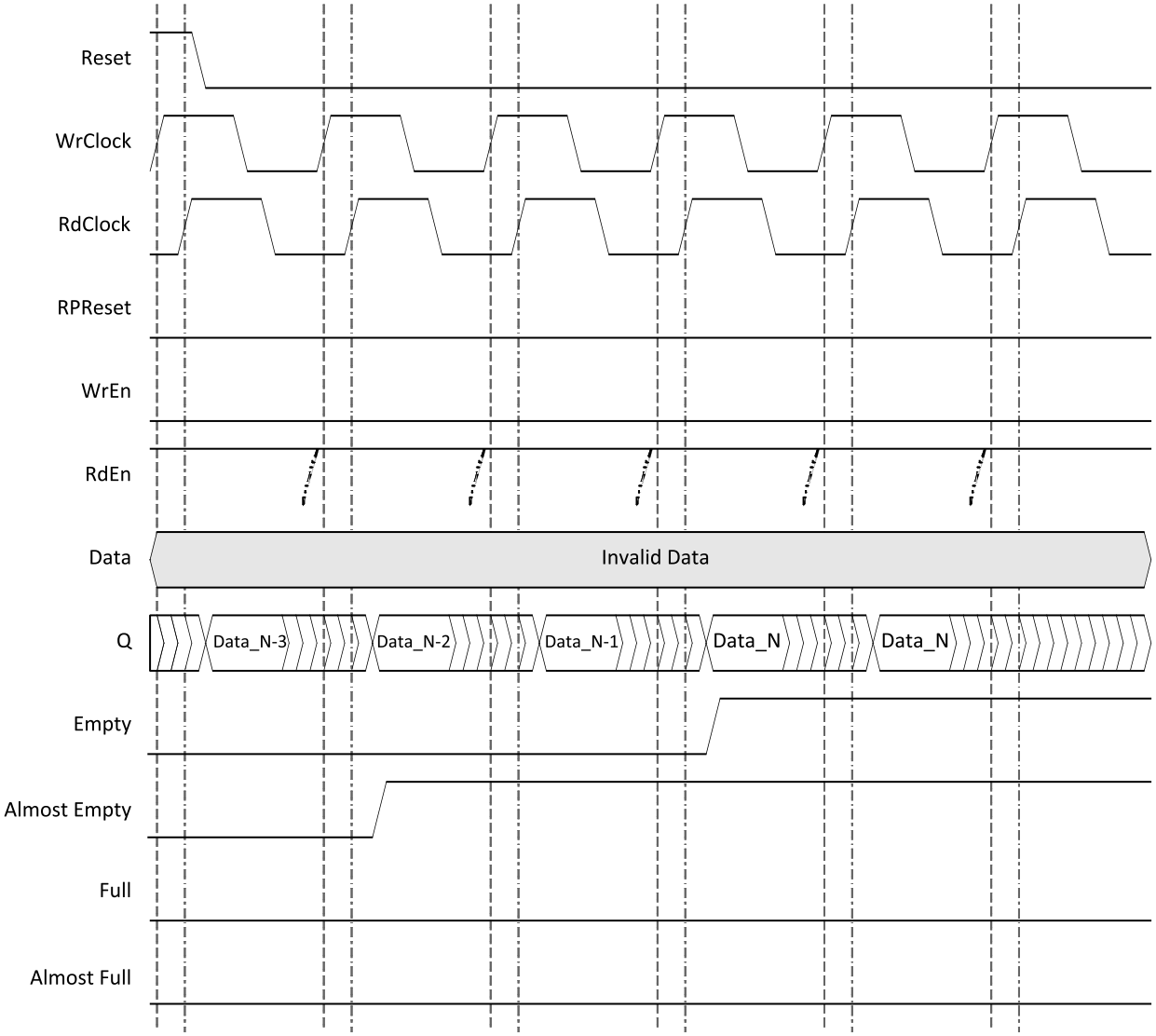

Figure 4.37. FIFO_DC without Output Registers, End of Data Read Cycle

Figure 4.34 to Figure 4.37 show the behavior of the non-pipelined FIFO_DC or FIFO_DC without output registers. When we pipeline the registers, the output data is delayed by one clock cycle. An extra option to enable output registers with the RdEn signal is available.

Figure 4.38 to Figure 4.41 show similar waveforms for the FIFO_DC with output register and without output register enable with RdEn. Note that flags are asserted and de-asserted with timing similar to the FIFO_DC without output registers. However, only the data out 'Q' is delayed by one clock cycle.

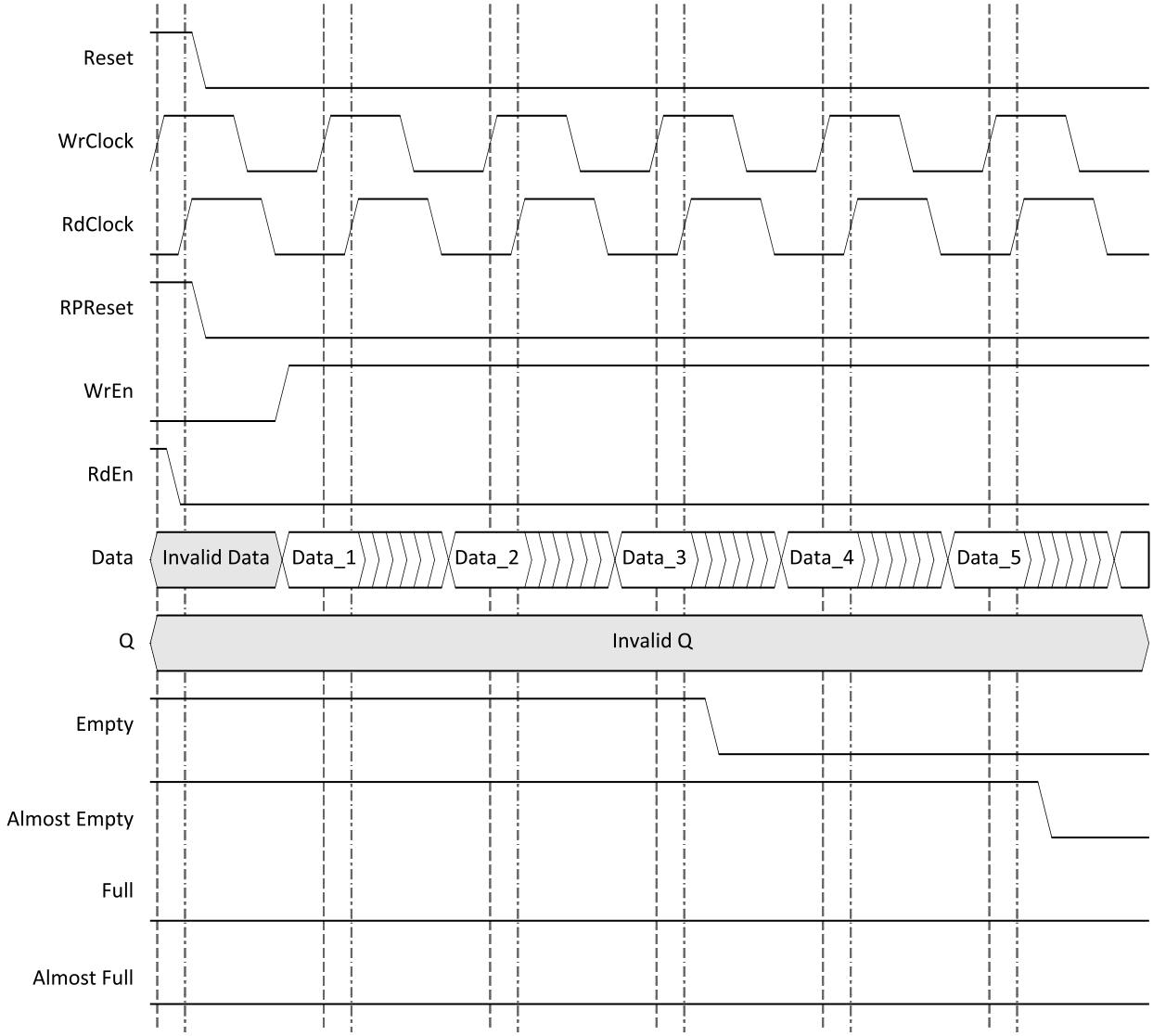

Figure 4.38. FIFO_DC with Output Registers, Start of Data Write Cycle

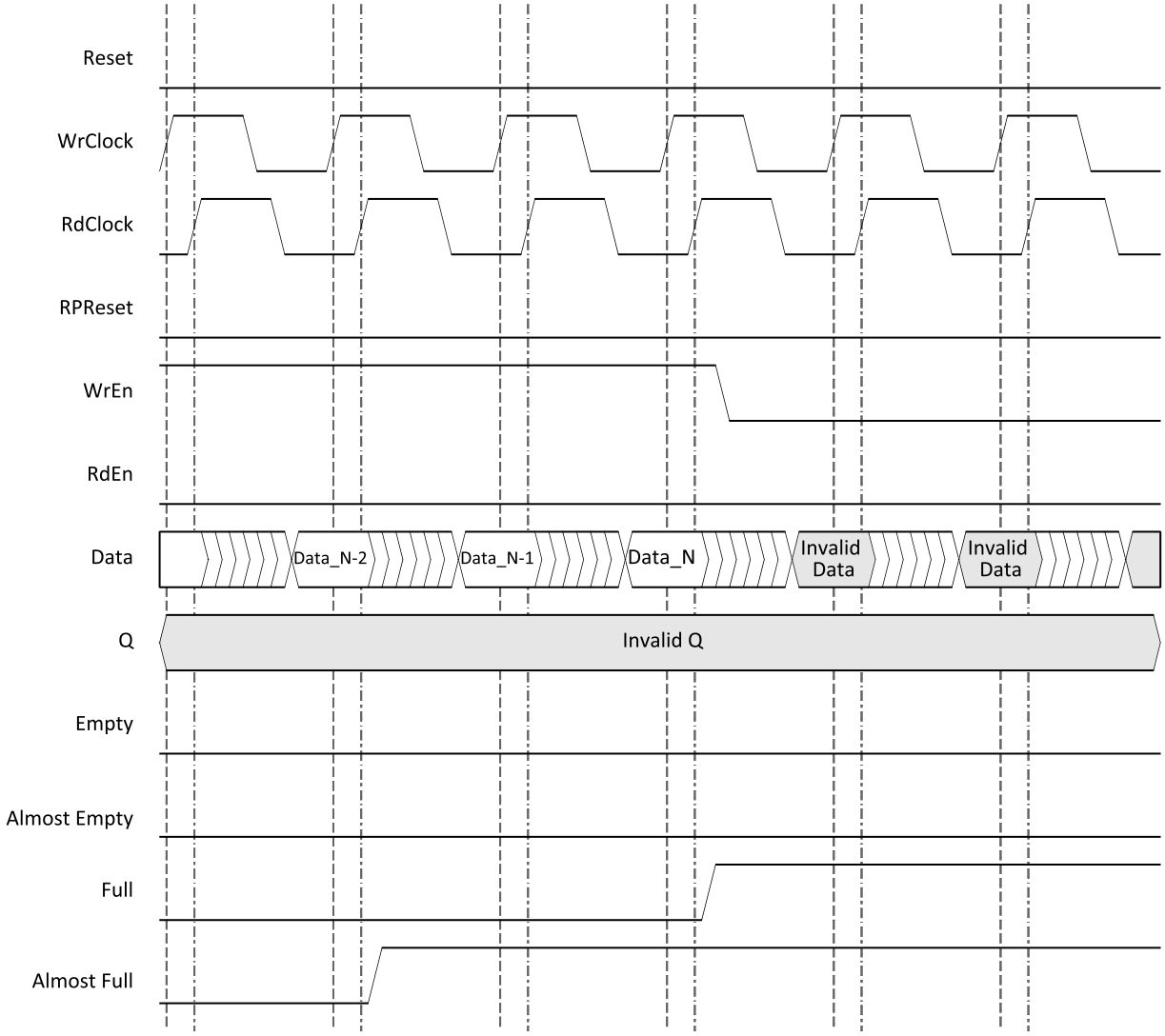

Figure 4.39. FIFO_DC with Output Registers, End of Data Write Cycle

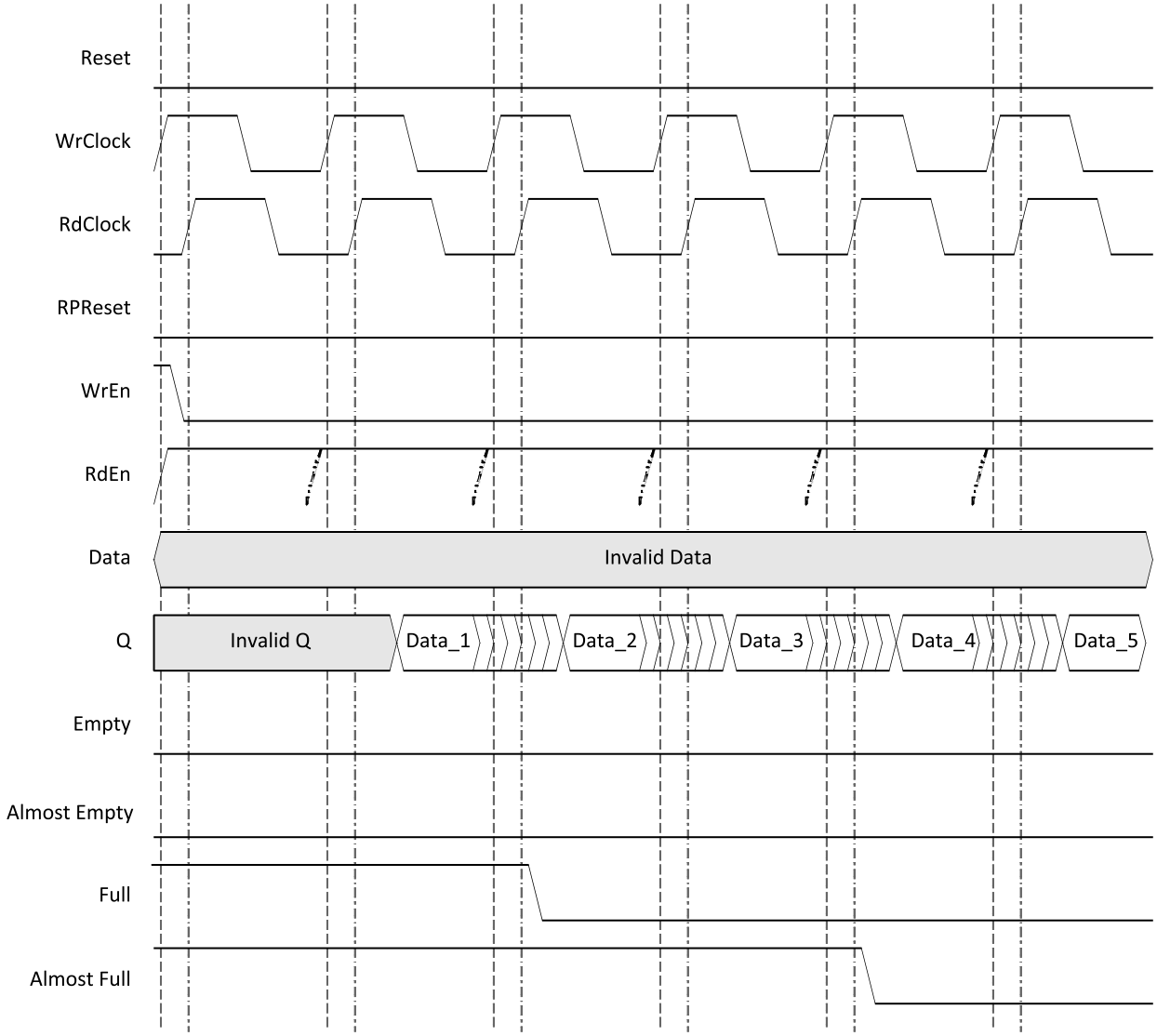

Figure 4.40. FIFO_DC with Output Registers, Start of Data Read Cycle

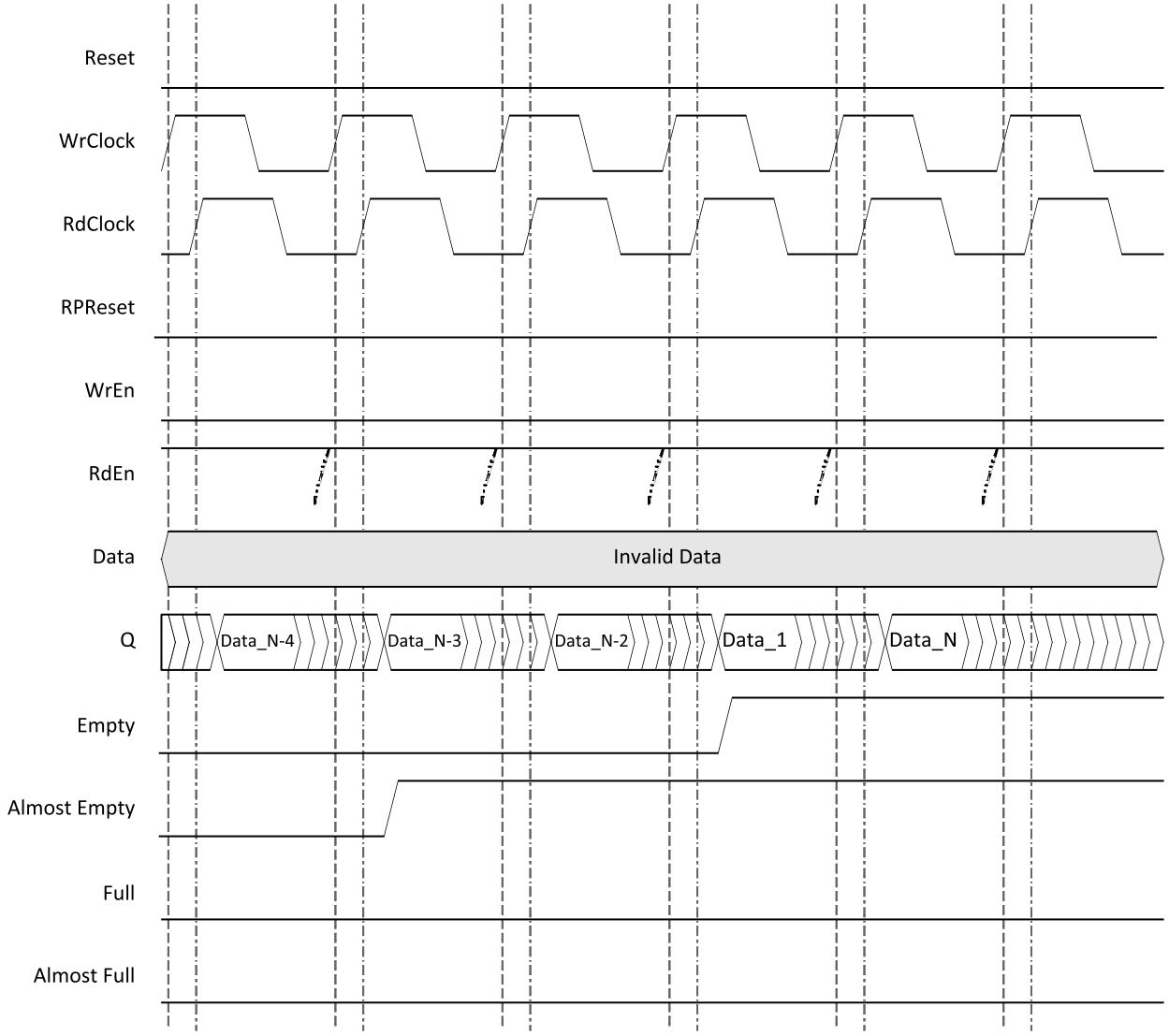

Figure 4.41. FIFO_DC with Output Registers, End of Data Read Cycle

If the enable output register with RdEn option is selected, the data out is still delayed by one clock cycle (as compared to the non-pipelined FIFO_DC). The RdEn should also be high during that clock cycle, otherwise the data takes an extra clock cycle when the RdEn goes true.

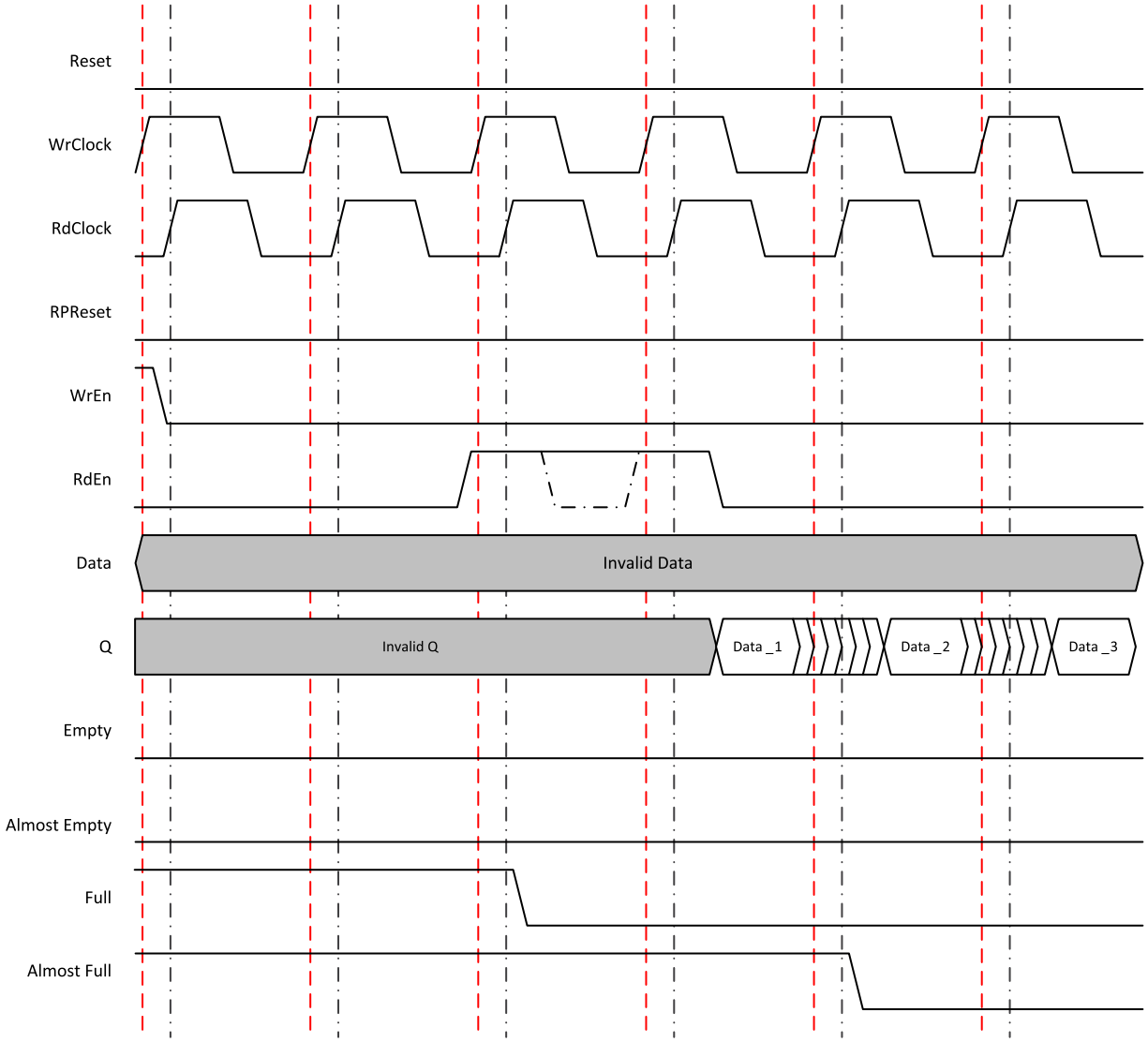

Figure 4.42. FIFO_DC with Output Registers and RdEn on Output Registers

When using FIFO_DC with different data widths on read and write ports, care should be taken to make sure that the wider data width should be the multiple of the smaller one. In addition, the words written or read out should follow the same relationship. For example, if the DataIn (write port) width is 8 bits and DataOut (read port) is 16 bits, there is a factor of 2 between the two. For every 2 words written in the FIFO_DC, one word is read out. When odd number of words are written, for example 7 words, the read port reads three complete words and one half word. The other half of the incomplete word will either be all zeroes (0s) or prior data written at the 8th location.

If we reverse the number of bits on DataIn and DataOut, for every written word, two words are read out. Once again, in order to completely read the FIFO_DC, we need twice the number of clock cycles on the write port.

FIFO_DC does not include any arbitration logic. It has to be implemented outside of the FIFO_DC. Read and Write Count pointers can be used to aid in counting the number of written or read words.

4.7. Distributed Single-Port RAM (Distributed_SPRAM) – PFU-Based

PFU-based Distributed Single-Port RAM is created using the 4-input LUTs available in the PFU. These LUTs can be cascaded to create larger distributed memory sizes.

Figure 4.43 shows the Distributed Single-Port RAM module as generated by Clarity Designer.

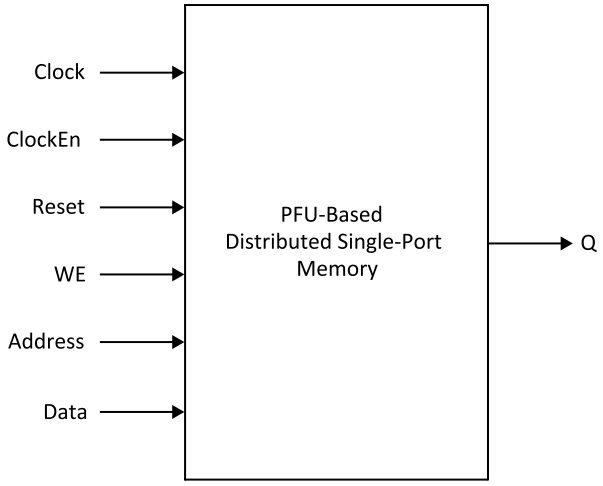

Figure 4.43. Distributed Single-Port RAM Module Generated by Clarity Designer

The generated module makes use of the 4-input LUTs available in the PFU. Additional logic such as a clock or reset is generated by utilizing the resources available in the PFU.

Ports such as Read Clock (RdClock) and Read Clock Enable (RdClockEn) are not available in the hardware primitive. These are generated by Clarity Designer when the user wants to enable the output registers in the Clarity Designer configuration.

Figure 4.44 shows the primitive that can be instantiated for the Single Port Distributed RAM. The primitive name is SPR16X4C and it can be directly instantiated in the code. Check the details on the port and port names under the primitives available under cae_library/synthesis folder in Lattice Diamond software installation folder.

Note that each EBR can accommodate 64 bits of memory; if the memory required is larger than 64 bits, then cascading can be used. Further, the ports can be registered by using external PFU registers.

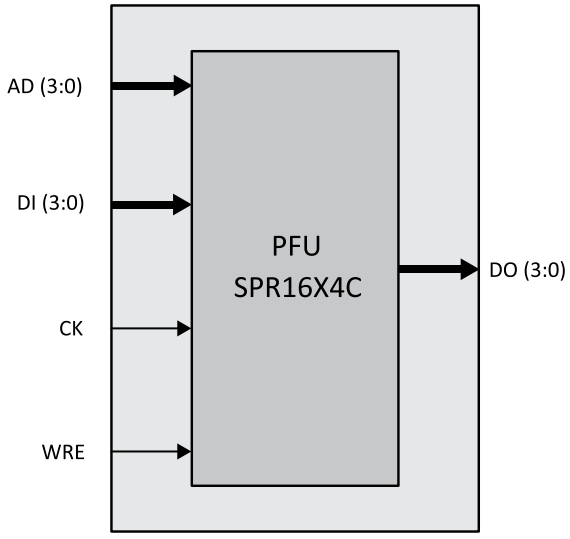

Figure 4.44. Single Port Distributed RAM Primitive for CrossLink Devices

The various ports and their definitions are listed in Table 4.15. The table lists the corresponding ports for the module generated by Clarity Designer and for the primitive.

Table 4.15. PFU-Based Distributed Single Port RAM Port Definitions

Port Name in Generated Module	Port Name in the EBR block Primitive	Description
Clock	CK	Clock
ClockEn	—	Clock Enable
Reset	—	Reset
WE	WRE	Write Enable
Address	AD[3:0]	Address
Data	DI[1:0]	Data In
Q	DO[1:0]	Data Out

Ports such as Read Clock (RdClock) and Read Clock Enable (RdClockEn) are not available in the hardware primitive. These are generated by Clarity Designer when the user wants to enable the output registers in their Clarity Designer configuration.

The various ports and their definitions for memory are included in Table 4.11 on page 32. The table lists the corresponding ports for the module generated by Clarity Designer and for the primitive.

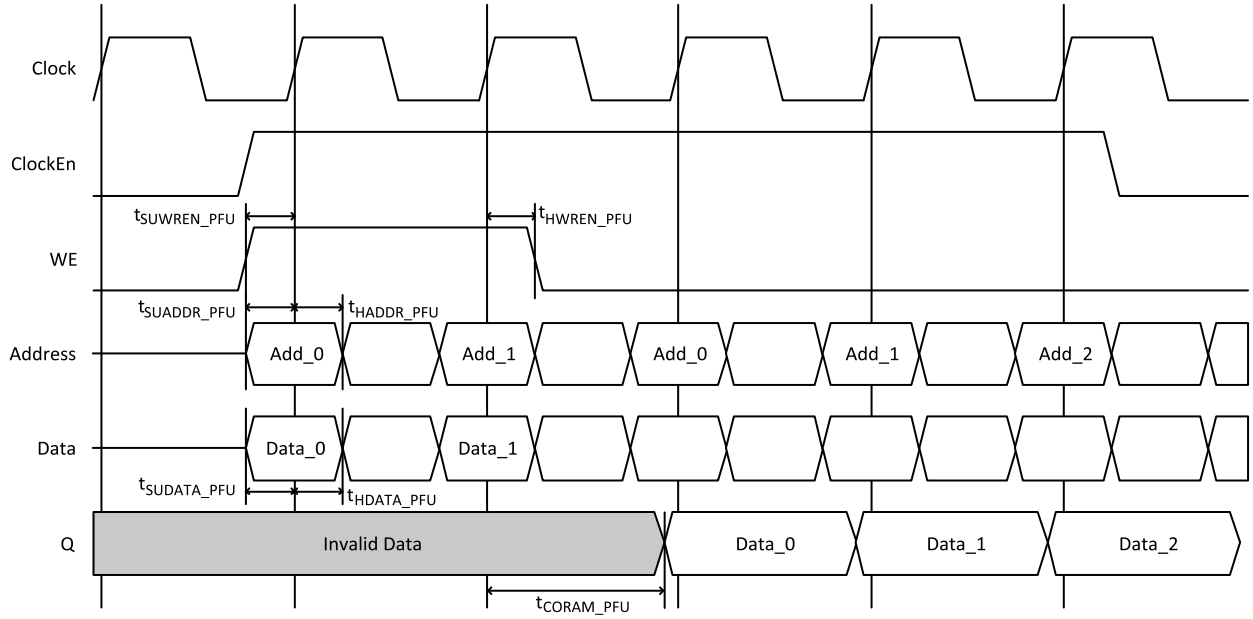

Figure 4.45. PFU Based Distributed Single Port RAM Timing Waveform without Output Registers

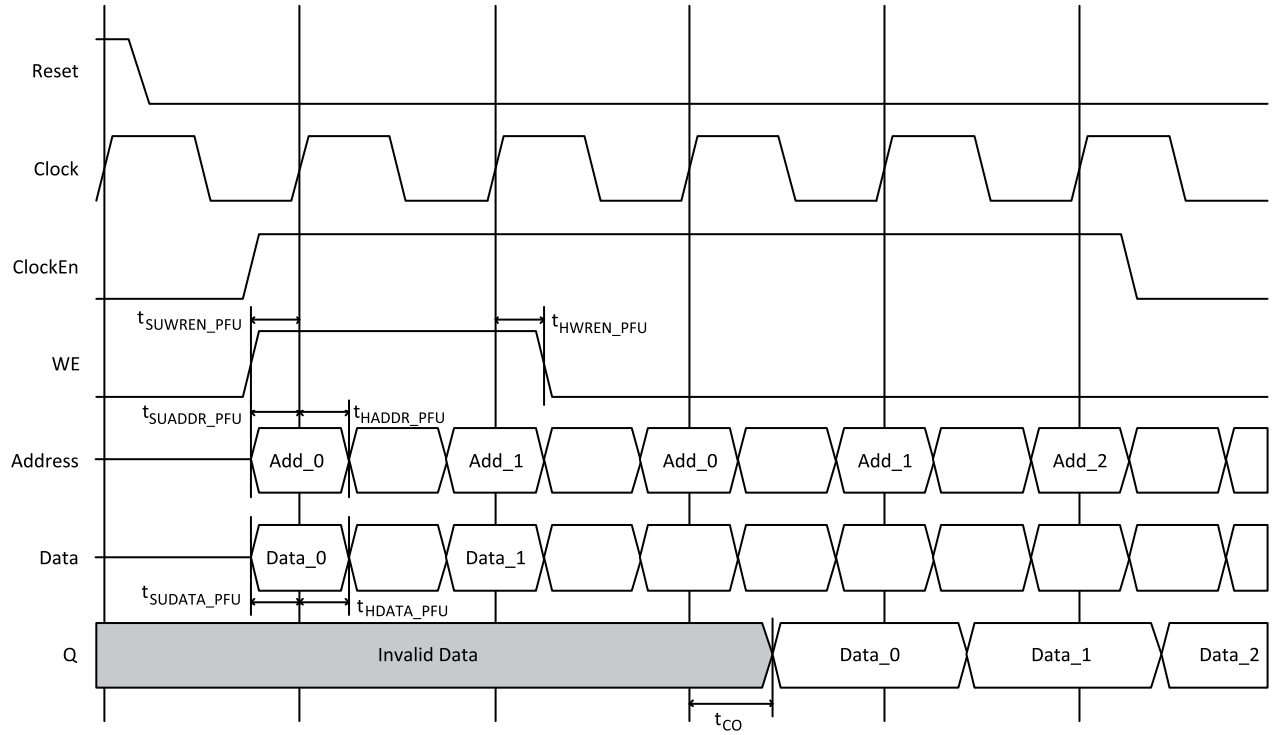

Figure 4.46. PFU Based Distributed Single Port RAM Timing Waveform with Output Registers

4.8. Distributed Dual-Port RAM (4.8. Distributed_DPRAM) – PFU-Based

PFU-based Distributed Dual-Port RAM is also created using the 4-input LUTs available in the PFU. These LUTs can be cascaded to create larger distributed memory sizes.

Figure 4.47 shows the Distributed Single-Port RAM module as generated by Clarity Designer.

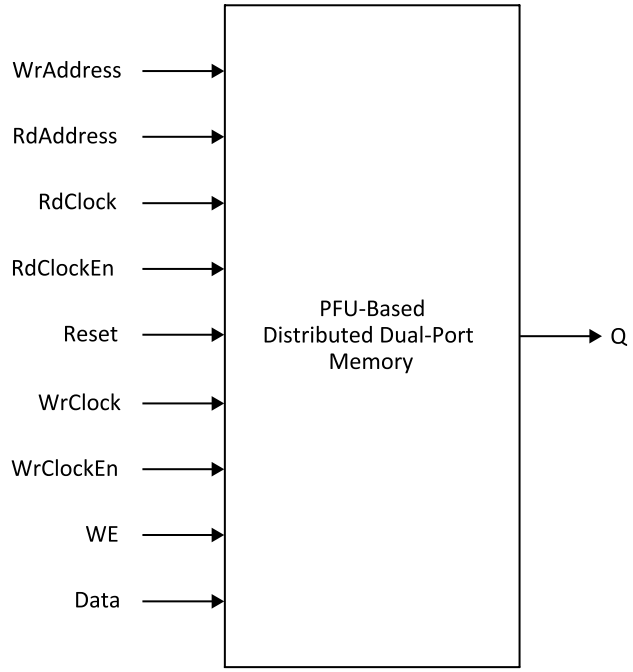

Figure 4.47. Distributed Dual-Port RAM Module Generated by Clarity Designer

The generated module makes use of the 4-input LUTs available in the PFU. Additional logic such as a clock or reset is generated by utilizing the resources available in the PFU.

Ports such as the Read Clock (RdClock) and Read Clock Enable (RdClockEn) are not available in the hardware primitive. These are generated by Clarity Designer when the user wants to enable the output registers in the Clarity Designer configuration.

Figure 4.48 shows the primitive that can be instantiated for the Dual Port Distributed RAM. T h e primitive name is DPR16X4C and it can be directly instantiated in the code. Check the details on the port and port names under the primitives available in the cae_library/synthesis folder in Lattice Diamond software installation package.

Note that each DPR16X4C (or PFU) can accommodate 64 bits of memory; if the memory required is larger than 64 bits, cascading can be used. Further, the ports can be registered by using external PFU registers.

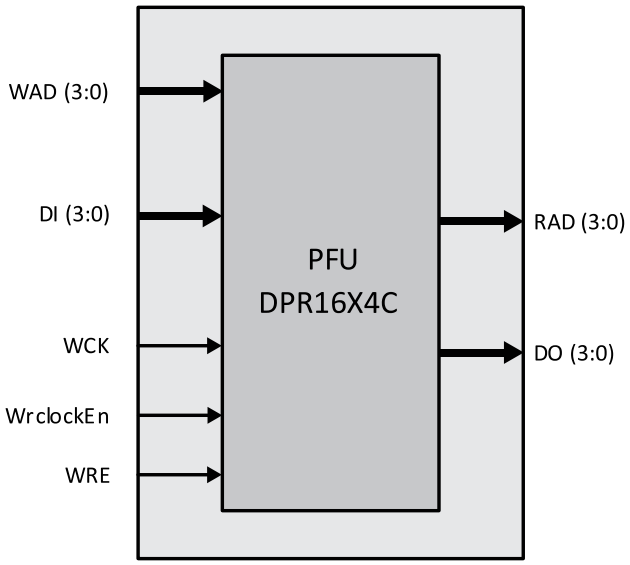

Figure 4.48. Dual Port Distributed RAM Primitive for CrossLink Devices

The various ports and their definitions are listed in Table 4.16. The table lists the corresponding ports for the module generated by Clarity Designer and for the primitive.

Table 4.16. PFU-Based Distributed Dual-Port RAM Port Definitions

Port Name in the Generated Module	Port Name in the Block Primitive	Description
WrAddress	WAD[3:0]	Write Address
RdAddress	RAD[3:0]	Read Address
RdClock	—	Read Clock
RdClockEn	—	Read Clock Enable
WrClock	WCK	Write Clock
WrClockEn	—	Write Clock Enable
WE	WRE	Write Enable
Data	DI[1:0]	Data Input
Q	RDO[1:0]	Data Out

Ports such as Read Clock (RdClock) and Read Clock Enable (RdClockEn) are not available in the hardware primitive. These are generated by Clarity Designer when the user wants to enable the output registers in the Clarity Designer configuration.

Users have the option of enabling the output registers for Distributed Dual Port RAM (Distributed_DPRAM). Figure Figure 4.49 and Figure 4.50 show the internal timing waveforms for the Distributed Dual Port RAM (Distributed_DPRAM) with these options.

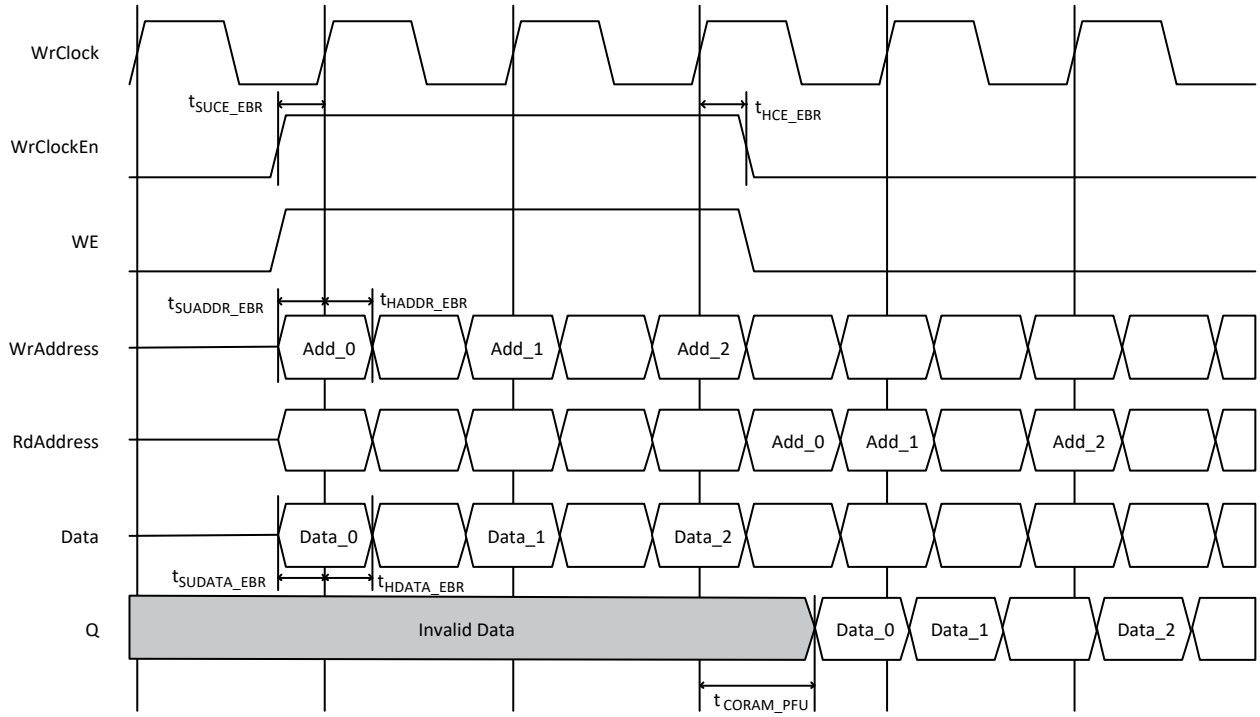

Figure 4.49. PFU Based Distributed Dual Port RAM Timing Waveform without Output Registers

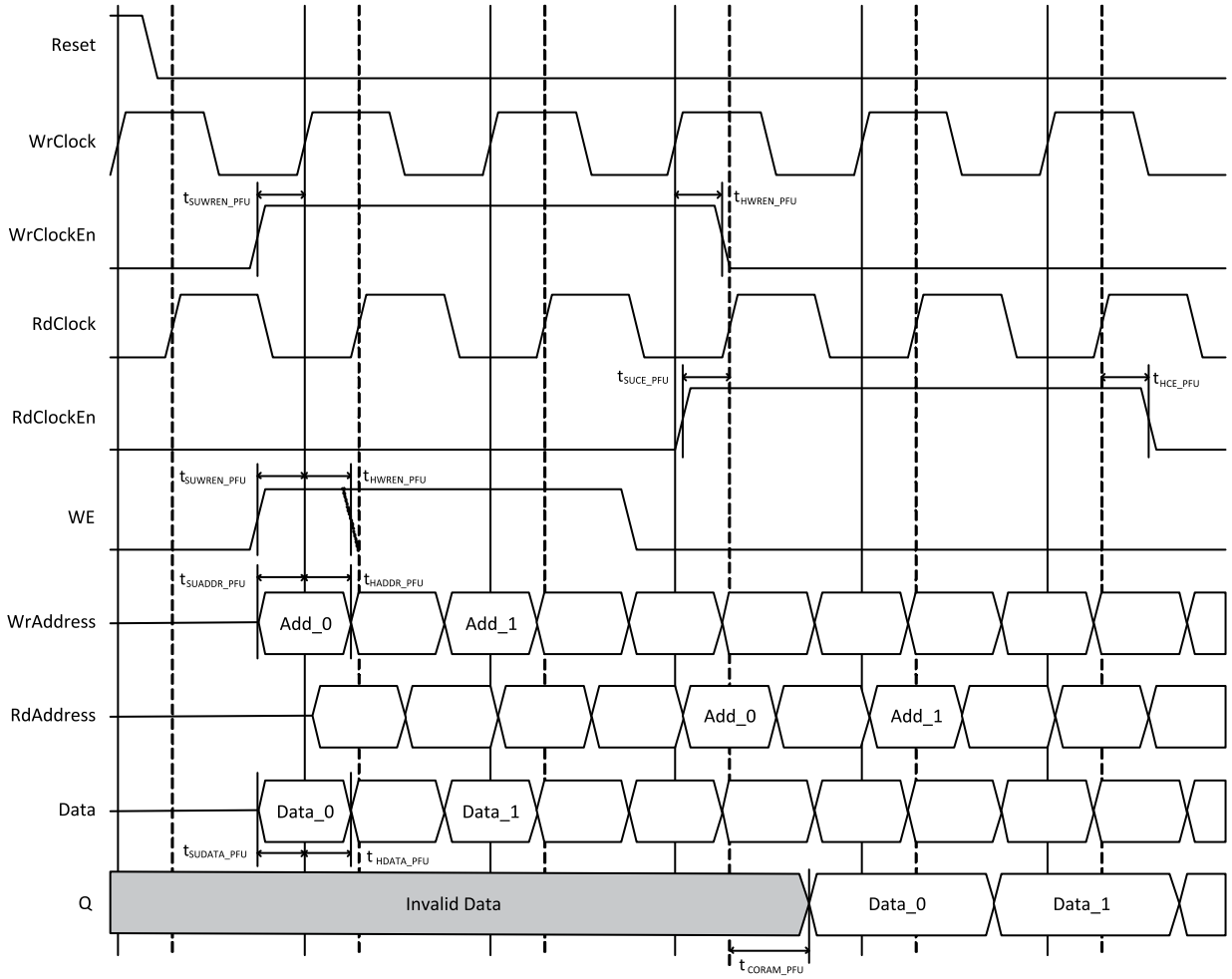

Figure 4.50. PFU Based Distributed Dual Port RAM Timing Waveform with Output Registers

4.9. Distributed ROM (Distributed_ROM) – PFU-Based

PFU-based Distributed ROM is created using the 4-input LUTs available in the PFU. These LUTs can be cascaded to create larger distributed memory sizes.

Figure 4.51 shows the Distributed ROM module generated by Clarity Designer.

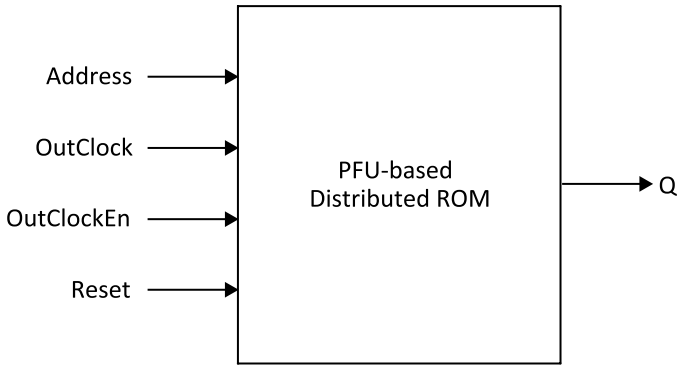

Figure 4.51. Distributed ROM Generated by Clarity Designer

The generated module makes use of the 4-input LUTs available in the PFU. Additional logic such as a clock or reset is generated by utilizing the resources available in the PFU.

Ports such as Out Clock (OutClock) and Out Clock Enable (OutClockEn) are not available in the hardware primitive. These are generated by Clarity Designer when the user wants to enable the output registers in the Clarity Designer configuration.

Figure 4.52 provides the primitive that can be instantiated for the Distributed ROM. Primitive name is DPRnX1a and it can be directly instantiated in the code. 'n' can be 3, 4, 5, 6 or 7 depending upon the size of the ROM. Check the details on the port and port names under the primitives available in the cae_library/synthesis folder in Lattice Diamond software installation package.

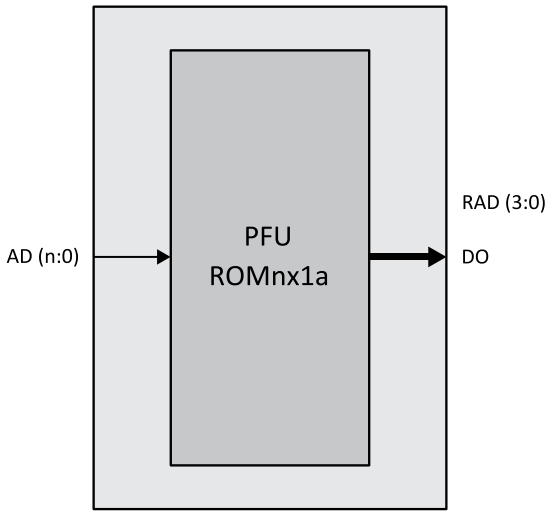

Figure 4.52. Distributed ROM Primitive for CrossLink Devices

If the memory required is larger than what can fit in the primitive bits, then cascading can be used. Further, the ports can be registered by using external PFU registers.

The various ports and their definitions are listed in Table 4.17. The table lists the corresponding ports for the module generated by Clarity Designer and for the primitive.

Table 4.17. PFU-Based Distributed ROM Port Definitions

Port Name in the Generated Module	Port Name in the EBR Block Primitive	Description
Address	AD[3:0]	Address
OutClock	—	Out Clock
OutClockEn	—	Out Clock Enable
Reset	—	Reset
Q	DO	Data Out

Users have the option to enable the output registers for Distributed ROM (Distributed_ROM). Figure 4.53 and Figure 4.54 show the internal timing waveforms for the Distributed ROM with these options.

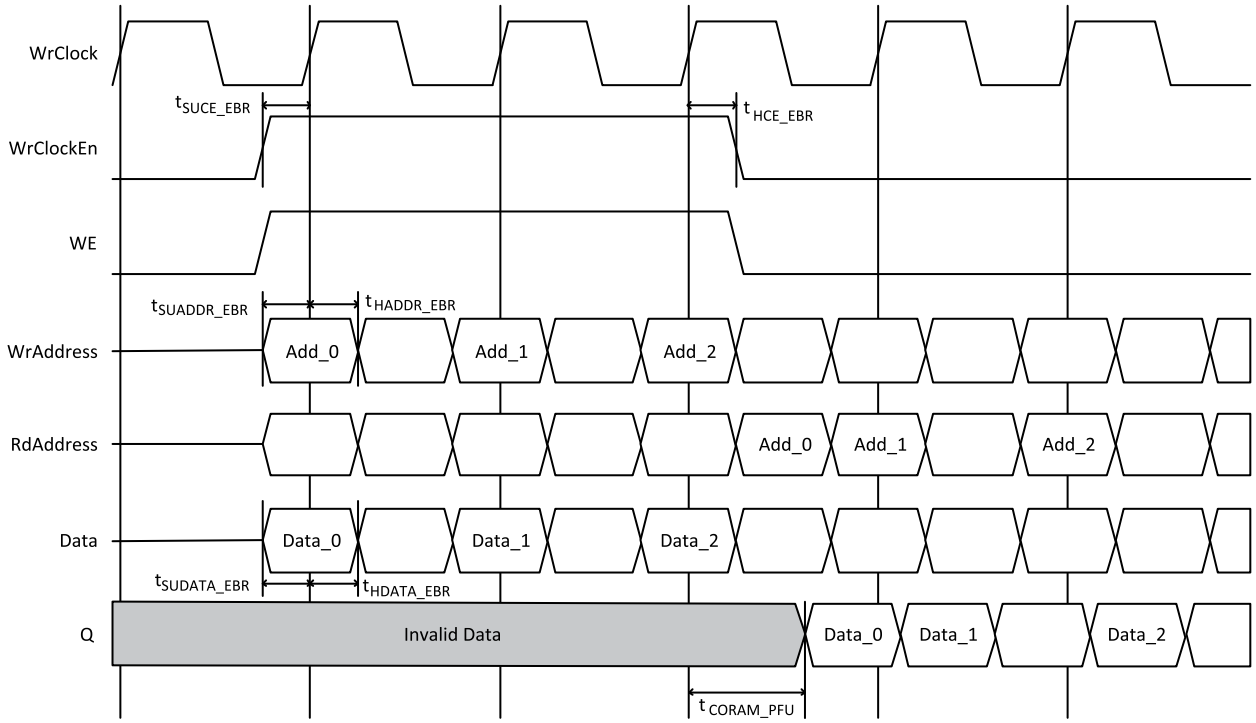

Figure 4.53. PFU Based Distributed Dual Port ROM Timing Waveform without Output Registers

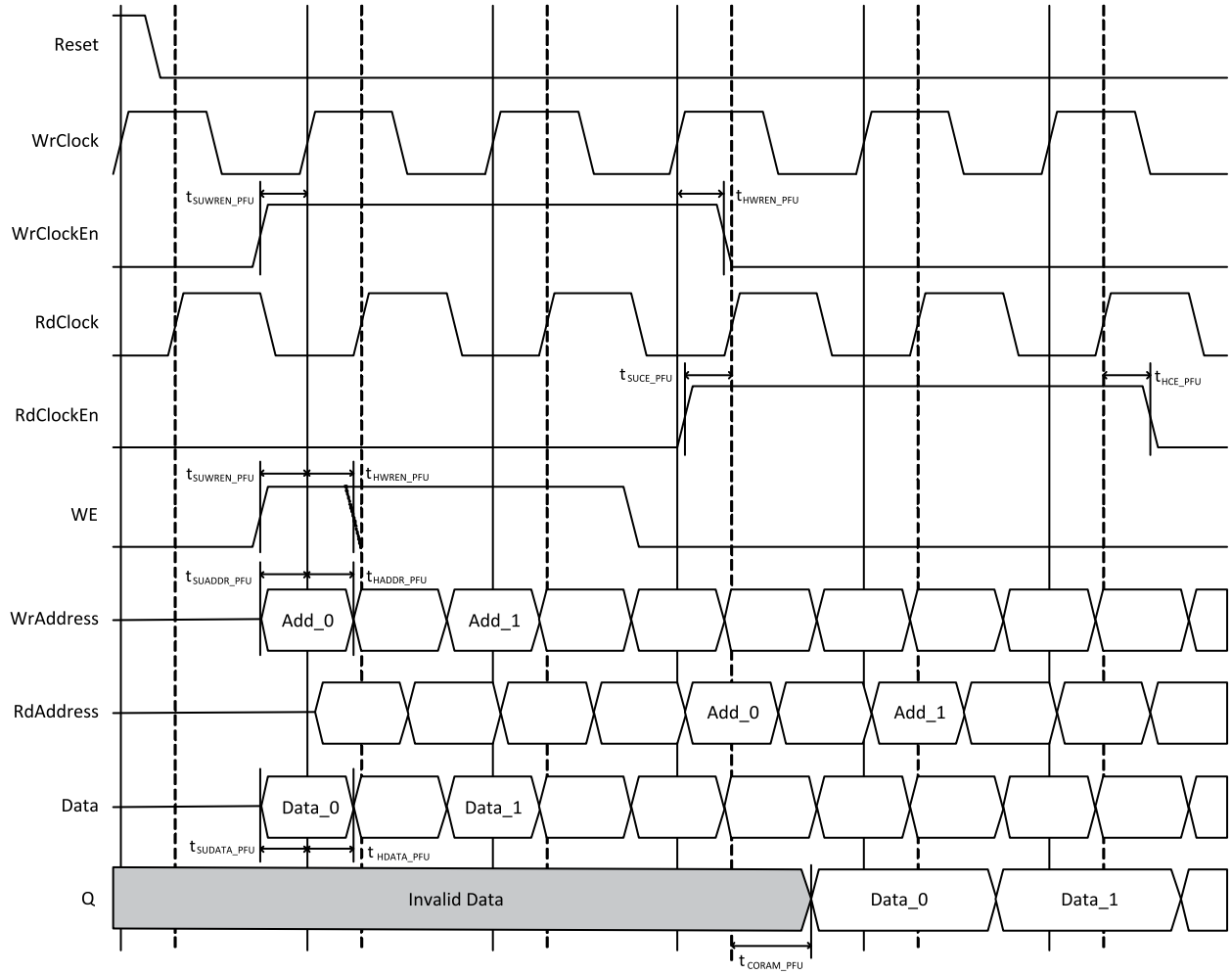

Figure 4.54. PFU Based Distributed Dual Port ROM Timing Waveform with Output Registers

5. Initializing Memory

In each memory mode, it is possible to specify the power-on state of each bit in the memory array. This allows the memory to be used as ROM. Each bit in the memory array can have a value of 0 or 1.

5.1. Initialization File Formats

The initialization file is an ASCII file, which the designer can create or edit using any ASCII editor. Clarity Designer supports three memory file formats:

- Binary file
- Hex File
- Addressed Hex

The file name for the memory initialization file is *.mem (<file_name>.mem). Each row includes the value to be stored in a particular memory location. The number of characters (or the number of columns) represents the number of bits for each address (or the width of the memory module).

The memory initialization can be static or dynamic. In case of static initialization, the memory values are stored in the bitstream. Dynamic initialization of memories, involve memory values stored in the external flash and can be updated by user logic knowing the EBR address locations. The size of the bitstream (bit or rbt file) is larger due to static values stored in them.

The initialization file is primarily used for configuring the ROMs. RAMs can also use the initialization file to preload memory contents.

5.1.1. Binary File

The binary file is a text file of 0s and 1s. The rows indicate the number of words and the columns indicate the width of the memory.

Memory Size 20x32

```
00100000010000000010000001000000
00000001000000010000000100000001
00000010000000100000001000000010
00000011000000110000001100000011
00000100000001000000010000000100
00000101000001010000010100000101
00000110000001100000011000000110
00000111000001110000011100000111
00001000010010000000100001001000
00001001010010010000100101001001
00001010010010100000101001001010
00001011010010110000101101001011
00001100000011000000110000001100
00001101001011010000110100101101
00001110001111100000111000111110
00001111001111110000111100111111
00010000000100000001000000010000
00010001000100010001000100010001
00010010000100100001001000010010
00010011000100110001001100010011
```

5.1.2. Hex File

The hex file is a text file of hexadecimal characters arranged in a similar row-column arrangement. The number of rows in the file is the same as the number of address locations, with each row indicating the content of the memory location.

Memory Size 8x16

```
A00
1
0B0
3
1004
CE0
6
000
7
040A
0017
02A4
```

5.1.3. Addressed Hex

Addressed hex consists of lines of addresses and data. Each line starts with an address, followed by a colon, and any number of data. The format of the file is “address: data data data data” where the address and data are hexadecimal numbers.

```
A0 : 03 F3 3E 4F
B2 : 3B 9F
```

In the example above, the first line shows 03 at address A0, F3 at address A1, 3E at address A2,and 4F at address A3. The second line shows 3B at address B2 and 9F at address B3.

There is no limitation on the address and data values. The value range is automatically checked based on the values of addr_width and data_width. If there is an error in an address or data value, an error message is printed. It is not necessary to specify data at all address locations. If data is not specified at a certain address, the data at that location is initialized to 0. SCUBA makes memory initialization possible both through the synthesis and simulation flows.

Appendix A. Attribute Definitions

A.1. DATA_WIDTH

Data width is associated with the RAM and FIFO elements. The DATA_WIDTH attribute defines the number of bits in each word. It uses the values defined in the RAM size tables in each memory module.

A.2. REGMODE

REGMODE, or the Register mode attribute, is used to enable pipelining in the memory. This attribute is associated with the RAM and FIFO elements. The REGMODE attribute takes the NOREG or OUTREG mode parameter that disables and enables the output pipeline registers.

A.3. CSDECODE

CSDECODE or the Chip Select Decode attributes are associated with block RAM elements. CS, or Chip Select, is the port available in the EBR primitive that is useful when memory requires multiple EBR blocks to be cascaded. The CS signal forms the MSB for the address when multiple EBR blocks are cascaded. CS is a 3-bit bus, so it can cascade eight memories easily.

CSDECODE takes the following parameters: “000”, “001”, “010”, “011”, “100”, “101”, “110”, and “111”. CSDECODE values determine the decoding value of CS[2:0]. CSDECODE_W is chip select decode for write and CSDECODE_R is chip select decode for read for Pseudo Dual Port RAM. CSDECODE_A and CSDECODE_B are used for True Dual-Port RAM elements and refer to the A and B ports.

A.4. WRITEMODE

The WRITEMODE attribute is associated with the block RAM elements. It takes the NORMAL, WRITETHROUGH, and READBEFOREWRITE mode parameters.

In NORMAL mode, the output data does not change or get updated, during the write operation. This mode is supported for all data widths.

In WRITETHROUGH mode, the output data is updated with the input data during the write cycle. This mode is supported for all data widths.

WRITEMODE_A and WRITEMODE_B are used for Dual-Port RAM elements and refer to the A and B ports in True Dual-Port RAM.

In READBEFOREWRITE mode, the output data port is updated with the existing data stored in the write address, during a write cycle. This mode is supported for x9, x18 and x36 data widths.

For all modes of the True Dual-Port module, simultaneous read access from one port and write access from the other port to the same memory address is not recommended. The read data may be unknown in this situation.

Also, simultaneous write access to the same address from both ports is not recommended. When this occurs, the data stored in the address becomes undetermined when one port tries to write an ‘H’ and the other tries to write an ‘L’. It is recommended that control logic be implemented to identify this situation if it occurs and do one of the following:

- Implement status signals to flag the read data as possibly invalid, or
- Implement control logic to prevent simultaneous access from both ports.

References

For more information, refer to the following documents:

- CrossLink Family Data Sheet (FPGA-DS-02007)
- CrossLink High-Speed I/O Interface (FPGA-TN-02012)
- CrossLink Hardware Checklist (FPGA-TN-02013)
- CrossLink Programming and Configuration Usage Guide (FPGA-TN-02014)
- CrossLink sysCLOCK PLL/DLL Design and Usage Guide (FPGA-TN-02015)
- CrossLink sysI/O Usage Guide (FPGA-TN-02016)
- Power Management and Calculation for CrossLink Devices (FPGA-TN-02018)
- CrossLink I2C Hardened IP Usage Guide (FPGA-TN-02019)
- Advanced CrossLink I2C Hardened IP Reference Guide (FPGA-TN-02020)

Technical Support Assistance

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

Revision History

Revision 1.2, October 2020

Section	Change Summary
All	Changed document status from Preliminary to final.

Revision 1.1, December 2019

Section	Change Summary
Disclaimers	Added this section.
Revision History	Updated format.

Revision 1.0, August 2016

Section	Change Summary
All	Updated document numbers.

Revision 1.0, May 2016

Section	Change Summary
All	First preliminary release.

www.latticesemi.com