Speedster22i
Macro Cell Library

UG021 v1.10 – April 21, 2015

achronix
SEMICONDUCTOR CORPORATION

Copyright Info

Achronix Semiconductor Corporation

2953 Bunker Hill Lane, Suite 101
Santa Clara, CA, 95054
USA

Phone : 877.GHZ.FPGA (877.449.3742)
Fax : 408.286.3645
E-mail : info@achronix.com

Table of Contents

Preface ---------- xiv

Introduction ---------- xiv

Cell Naming Conventions ---------- xv

Register Naming Conventions ---------- xv

I/O Cell Naming Conventions ---------- xv

Conventions Used in this Guide ---------- xvi

Chapter 1 – I/O Cells ---------- 1

IOPAD ---------- 2

Bidirectional I/O Pad ---------- 2

Verilog Instantiation Template ---------- 3

VHDL Instantiation Template ---------- 3

IOPAD_D ---------- 4

Bidirectional Registered I/O Pad with Asynchronous or Synchronous Set/Reset ---------- 4

Verilog Instantiation Template ---------- 7

VHDL Instantiation Template ---------- 7

IOPAD_D2 ---------- 9

Bidirectional DDR I/O Pad with Asynchronous or Synchronous Set/Reset ---------- 9

Verilog Instantiation Template ---------- 12

VHDL Instantiation Template ---------- 12

IPAD ---------- 13

Non-Registered Input Pad ---------- 13

Verilog Instantiation Template ---------- 13

VHDL Instantiation Template ---------- 14

IPAD_D ---------- 15

Registered Input Pad with Asynchronous or Synchronous Set/Reset ---------- 15

Verilog Instantiation Template ---------- 16

VHDL Instantiation Template ---------- 16

IPAD_D2 ---------- 17

DDR Input Pad with Asynchronous or Synchronous Set/Reset ---------- 17

Verilog Instantiation Template ---------- 18

VHDL Instantiation Template ---------- 18

IPAD_DIFF ---------- 20

Non-Registered Differential Input Pad ---------- 20

Verilog Instantiation Template ---------- 21

VHDL Instantiation Template ---------- 21

IPAD_DIFFD ---------- 22

Registered Differential Input Pad with Asynchronous or Synchronous Set/Reset ---------- 22
Verilog Instantiation Template ---------- 23
VHDL Instantiation Template ---------- 23
IPAD_DIFFD2 ---------- 24
DDR Differential Input Pad with Asynchronous or Synchronous Set/Reset ---------- 24
Verilog Instantiation Template ---------- 25
VHDL Instantiation Template ---------- 25
OPAD ---------- 27
Non-Registered Output Pad ---------- 27
Verilog Instantiation Template ---------- 27
VHDL Instantiation Template ---------- 28
OPAD_D ---------- 29
Registered Output Pad with Asynchronous or Synchronous Set/Reset ---------- 29
Verilog Instantiation Template ---------- 30
VHDL Instantiation Template ---------- 31
OPAD_D2 ---------- 32
DDR Output Pad with Asynchronous or Synchronous Set/Reset ---------- 32
Verilog Instantiation Template ---------- 33
VHDL Instantiation Template ---------- 34
OPAD_DIFF ---------- 35
Non-Registered Differential Output Pad ---------- 35
Verilog Instantiation Template ---------- 35
VHDL Instantiation Template ---------- 36
OPAD_DIFFD ---------- 37
Registered Differential Output Pad with Asynchronous or Synchronous Set/Reset ---------- 37
Verilog Instantiation Template ---------- 38
VHDL Instantiation Template ---------- 39
OPAD_DIFFD2 ---------- 40
DDR Differenctial Output Pad with Asynchronous or Synchronous Set/Reset ---------- 40
Verilog Instantiation Template ---------- 41
VHDL Instantiation Template ---------- 42
TPAD ---------- 43
Non-Registered Tristate Output Pad ---------- 43
Verilog Instantiation Template ---------- 43
VHDL Instantiation Template ---------- 44
TPAD_D ---------- 45
Registered Tristate Output Pad with Asynchronous or Synchronous Set/Reset ---------- 45
Verilog Instantiation Template ---------- 46
VHDL Instantiation Template ---------- 47
Chapter 2 – Registers ---------- 48
DFF ---------- 48
Positive Clock Edge D-Type Register ---------- 48

Pins ---- 48
Parameters ---- 48
init ---- 48
Verilog Instantiation Template ---- 49
VHDL Instantiation Template ---- 49
DFFE ---- 50
Positive Clock Edge D-Type Register with Clock Enable ---- 50
Pins ---- 50
Parameters ---- 50
init ---- 50
Verilog Instantiation Template ---- 51
VHDL Instantiation Template ---- 51
DFFEC ---- 52
Positive Clock Edge D-Type Register with Clock Enable and Synchronous Clear ---- 52
Pins ---- 52
Parameters ---- 52
init ---- 52
Verilog Instantiation Template ---- 53
VHDL Instantiation Template ---- 53
DFFEP ---- 54
Positive Clock Edge D-Type Register with Clock Enable and Synchronous Preset ---- 54
Pins ---- 54
Parameters ---- 54
init ---- 54
Verilog Instantiation Template ---- 55
VHDL Instantiation Template ---- 55
DFFER ---- 56
Positive Clock Edge D-Type Register with Clock Enable and Asynchronous/Synchronous Reset 56
Pins ---- 56
Parameters ---- 56
init ---- 56
sr_assertion ---- 57
Verilog Instantiation Template ---- 57
VHDL Instantiation Template ---- 58
DFFES ---- 59
Positive Clock Edge D-Type Register with Clock Enable and Asynchronous/Synchronous Set 59
Pins ---- 59
Parameters ---- 59
init ---- 59
sr_assertion ---- 60
Verilog Instantiation Template ---- 60
VHDL Instantiation Template ---- 61

DFFN - 62
Negative Clock Edge D-Type Register - 62
Pins - 62
Parameters - 62
init - 62
Verilog Instantiation Template - 63
VHDL Instantiation Template - 63
DFFNEC - 64
Negative Clock Edge D-Type Register with Clock Enable and Synchronous Clear - 64
Pins - 64
Parameters - 64
init - 64
Verilog Instantiation Template - 65
VHDL Instantiation Template - 65
DFFNEP - 66
Negative Clock Edge D-Type Register with Clock Enable and Synchronous Preset - 66
Pins - 66
Parameters - 66
init - 66
Verilog Instantiation Template - 67
VHDL Instantiation Template - 67
DFFNER - 68
Negative Clock Edge D-Type Register with Clock Enable and Asynchronous/Synchronous Reset 68
Pins - 68
Parameters - 68
init - 68
sr_assertion - 69
Verilog Instantiation Template - 69
VHDL Instantiation Template - 70
DFFNES - 71
Negative Clock Edge D-Type Register with Clock Enable and Asynchronous/Synchronous Set - 71
Pins - 71
Parameters - 71
init - 71
sr_assertion - 72
Verilog Instantiation Template - 72
VHDL Instantiation Template - 73
DFFNR - 74
Negative Clock Edge D-Type Register with Asynchronous Reset - 74
Pins - 74
Parameters - 74

init ---- 74
Verilog Instantiation Template ---- 75
VHDL Instantiation Template ---- 75
DFFNS ---- 76
Negative Clock Edge D-Type Register with Asynchronous Set ---- 76
Pins ---- 76
Parameters ---- 76
init ---- 76
Verilog Instantiation Template ---- 77
VHDL Instantiation Template ---- 77
DFFR ---- 78
Positive Clock Edge D-Type Register with Asynchronous Reset ---- 78
Pins ---- 78
Parameters ---- 78
init ---- 78
Verilog Instantiation Template ---- 79
VHDL Instantiation Template ---- 79
DFFS ---- 80
Positive Clock Edge D-Type Register with Asynchronous Set ---- 80
Pins ---- 80
Parameters ---- 80
init ---- 80
Verilog Instantiation Template ---- 81
VHDL Instantiation Template ---- 81
Chapter 3 – Logic Functions ---- 82
MUX2 ---- 82
Two Input Multiplexer Gate ---- 82
Pins ---- 82
Verilog Instantiation Template ---- 82
VHDL Instantiation Template ---- 83
Chapter 4 – Lookup Table (LUT) Functions ---- 84
LUT4 ---- 84
Four Input Lookup Table ---- 84
Pins ---- 84
Parameters ---- 84
lut_function ---- 84
Verilog Instantiation Template ---- 85
VHDL Instantiation Template ---- 85
Chapter 5 – Arithmetic Functions ---- 86
ALU ---- 86

Two Input Adder / Subtractor with Programmable Load - 86
Pins - 86
Parameters - 87
invert_b - 87
Verilog Instantiation Template - 87
VHDL Instantiation Template - 88

Chapter 6 – Memories - 89

BRAM80K - 89
80k-bit Dual-Port Memory - 89
BRAM80K Pins - 90
Parameters - 91
porta_read_width(portb_read_width) - 92
porta_write_width(portb_write_width) - 92
porta_write_mode(portb_write_mode) - 92
porta_clock_polarity(portb_clock_polarity) - 92
porta_peval(portb_peval) - 92
porta_latch_rstval(portb_latch_rstval) - 92
porta_en_out_reg(portb_en_out_reg) - 92
porta_reg_rstval(portb_reg_rstval) - 93
porta_regce_priority(portb_regce_priority) - 93
porta_initval(portb_initval) - 93
porta_srval(portb_srval) - 94
mem_init_file - 94
initd_000 – initd_255 - 94
initp_00 – initp_31 - 94
initpx_00 – initpx_31 - 95
Memory Organization and Data Input / Output Pin Assignments - 95
Read and Write Operations - 98
Read Operation - 98
Write Operation - 98
Simultaneous Memory Operations - 99
Timing Diagrams - 100
Support for Read-First (Read-Before-Write) Memory Operations - 101
Memory Initialization - 101
BRAM80K Verilog Instantiation Template - 103
BRAM80K VHDL Instantiation Template - 111
BRAM80KFIFO - 119
80k-bit FIFO Memory - 119
Parameters - 121
sync_mode - 121
write_width - 121
read_width - 122
fwft - 124

en_out_reg ---- 124
reg_initval ---- 125
reg_srval ---- 125
reg_rstval ---- 126
wrrst_rstval ---- 126
wrrst_input_mode ---- 126
wrrst_sync_stages ---- 128
wrptr_sync_stages ---- 128
rdrst_rstval ---- 129
rdrst_input_mode ---- 129
rdrst_sync_stages ---- 130
rdptr_sync_stages ---- 130
wrcount_sync_mode ---- 130
rdcount_sync_mode ---- 131
afull_offset ---- 131
aempty_offset ---- 131
wren_polarity_sel ---- 132
rden_polarity_sel ---- 132
en_rd_when_empty ---- 132
en_wr_when_full ---- 132
Read and Write Count Outputs ---- 133
Write Count Output ---- 133
Read Count Output ---- 133
Status Flags ---- 134
Empty Flag ---- 134
Almost Empty Flag ---- 134
Full Flag ---- 134
Almost Full Flag ---- 134
Write Error Flag ---- 134
Read Error Flag ---- 135
Flag Latency ---- 135
Optional Output Register ---- 136
FIFO Operational Modes ---- 137
Asynchronous FIFO Operation ---- 137
Synchronous FIFO Operation ---- 137
FIFO Operations ---- 138
Basic Mode FIFO Reset Operation ---- 138
Advanced Mode FIFO Reset Operation ---- 139
Writing an Empty Asynchronous FIFO (sync_mode = 1'b0) ---- 142
Writing an Empty Synchronous FIFO (sync_mode = 1'b1) ---- 142
Writing to an Almost Full FIFO (en_wr_when_full = 1'b0) ---- 143
Writing to an Almost Full FIFO (en_wr_when_full = 1'b1) ---- 143
Reading from an Almost Empty FIFO (en_rd_when_empty = 1'b0, fwft = 1'b0) ---- 144
Reading from an Almost Empty FIFO (en_rd_when_empty = 1'b0, fwft = 1'b1) ---- 145
Reading from an Almost Empty FIFO (en_rd_when_empty = 1'b1) ---- 145
Limitations of Concurrent Read/Write Operations ---- 146

BRAM80KECC ---- 147
80k-bit Simple Dual-Port Memory with Error Correction ---- 147
BRAM80KECC Pins ---- 148
Parameters ---- 148
en_out_reg ---- 149
reg_initval ---- 149
reg_srval ---- 149
reg_rstval ---- 149
encoder_enable ---- 149
decoder_enable ---- 150
mem_init_file ---- 150
initd_000 – initd_255 ---- 150
initp_0 – initp_31 ---- 150
initpx_0 – initpx_31 ---- 150
BRAM80KECC Modes of Operation ---- 151
ECC Encode/Decode Operation Mode ---- 151
ECC Encode-Only Operation Mode ---- 151
ECC Decode-Only Operation Mode ---- 151
BRAM80KECCFIFO ---- 153
80k-bit FIFO Memory with Error Correction ---- 153
Parameters ---- 155
LRAM640 ---- 156
640-bit (64x10) Simple-Dual-Port Memory ---- 156
LRAM640 Pins ---- 157
Parameters ---- 157
write_clock_polarity ---- 158
read_clock_polarity ---- 158
reg_dout ---- 158
reg_initval ---- 158
reg_rstval ---- 158
regce_priority ---- 158
mem_init ---- 158
mem_init_file ---- 158
Simultaneous Memory Operations ---- 159
LRAM640 Memory Initialization ---- 159
LRAMFIFO ---- 160
LRAM-Based 64-Word FIFO Memory ---- 160
Parameters ---- 161
width ---- 161
depth ---- 161
ptr_sync_mode ---- 162
wrptr_sync_stages ---- 162
rdptr_sync_stages ---- 162

rst_sync_mode ---- 163
wrrst_sync_stages ---- 163
rdrst_sync_stages ---- 164
afull_offset ---- 164
aempty_offset ---- 165
Status Flags ---- 165
Empty Flag ---- 165
Almost Empty Flag ---- 165
Full Flag ---- 165
Almost Full Flag ---- 165
Write Error Flag ---- 166
Read Error Flag ---- 166
Flag Latency in Asynchronous Mode (ptr_sync_mode = 1'b0) ---- 166
Flag Latency in Synchronous Mode (ptr_sync_mode = 1'b1) ---- 167
FIFO Operational Modes ---- 167
Asychronous FIFO Mode (ptr_sync_mode = 1'b0) ---- 167
Synchronous FIFO Mode (ptr_sync_mode = 1'b1) ---- 168
FIFO Operations ---- 168
Asynchronous FIFO Mode Reset Operation ---- 168
Writing an Empty Asynchronous FIFO (ptr_sync_mode = 1'b0) ---- 169
Writing an Empty Synchronous FIFO (ptr_sync_mode = 1'b1) ---- 169
Writing to an Almost Full FIFO ---- 170
Reading from an Almost Empty FIFO ---- 170

Chapter 7 – Multipliers ---- 171

BMACC56 ---- 171
28 x 28 Multiplier / Accumulator ---- 171
BMACC56 Pins ---- 173
Parameters ---- 175
init_a ---- 176
init_b ---- 176
init_sub ---- 176
init_cin ---- 176
init_mask_adda ---- 176
init_dout ---- 176
init_cout ---- 176
rst_value_a ---- 176
rst_value_b ---- 176
rst_value_sub ---- 176
rst_value_cin ---- 176
rst_value_mask_adda ---- 177
rst_value_dout ---- 177
rst_value_cout ---- 177
regce_priority_a ---- 177
regce_priority_b ---- 177

regce_priority_sub ---- 177
regce_priority_cin ---- 177
regce_priority_dout ---- 178
reg_a ---- 178
reg_b ---- 178
reg_addb ---- 178
reg_mask_adda ---- 178
reg_dout ---- 178
reg_cout ---- 178
sel_cascade_in ---- 178
sel_cascade_out ---- 178
sel_cin ---- 179
sel_sub ---- 179
mult_bypass ---- 179
clock_edge ---- 179
BMACC56 Verilog Instantiation Template ---- 180
BMACC56 VHDL Instantiation Template ---- 181
BMULT28X28 ---- 183
28 × 28 Signed Multiplier ---- 183
Verilog Instantiation Template ---- 183
VHDL Instantiation Template ---- 183
Chapter 8 – Special Functions ---- 184
ACX_DESERIALIZE (Speedster22iHP Only) ---- 184
1:N Serial-to-Parallel Converter ---- 184
Verilog Instantiation Template ---- 184
VHDL Instantiation Template ---- 185
ACX_SERIALIZE (Speedster22iHP Only) ---- 186
N:1 Parallel-to-Serial Converter ---- 186
Verilog Instantiation Template ---- 186
VHDL Instantiation Template ---- 186
Chapter 9 – PLL/DLL Clock Generators ---- 188
ACX_CLKGEN ---- 188
Phase-Locked Loop Clock Generator ---- 188
ACX_CLKGEN Pins ---- 189
Parameters ---- 190
ACX_CLKGEN Components ---- 194
Reference Divider ---- 194
Voltage Controlled Oscillator (VCO) ---- 194
Phase Rotator with Output Divider ---- 194
Output Synthesizer ---- 194
Phase Frequency Detector (PFD) ---- 195
Feedback Divider ---- 195
Clock Feedback Selection ---- 195

PLL Control ---------- 196
Serial Control Bus (SCB) ---------- 196
Control Status Registers (CSR) Register Description ---------- 197
Verilog Instantiation Template ---------- 203
VHDL Instantiation Template ---------- 205

Revision History---------- 207

Preface

Introduction

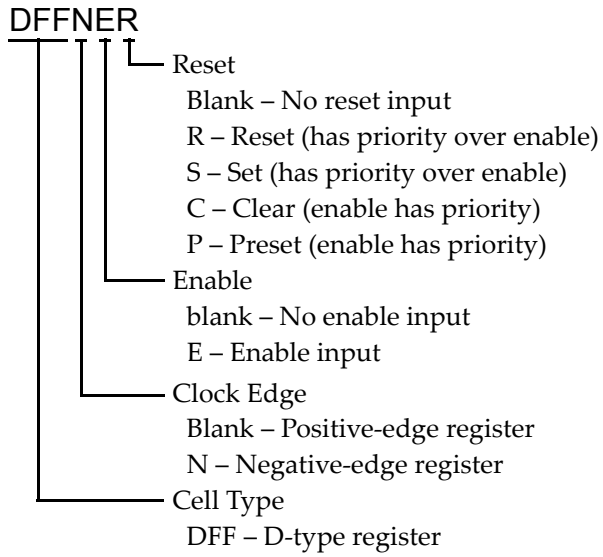
The Achronix Macro Cell Library provides the user with building blocks that may be instantiated into the user's design. These macros provide access to low-level fabric primitives, complex I/O block, and higher level design components. Each library element entry describes the operation of the macro as well as any parameters that must be initialized. Verilog and VHDL templates are also provided to aide in the implementation of the user's design.

The Speedster Macro Cell Library is divided into nine separate chapters:

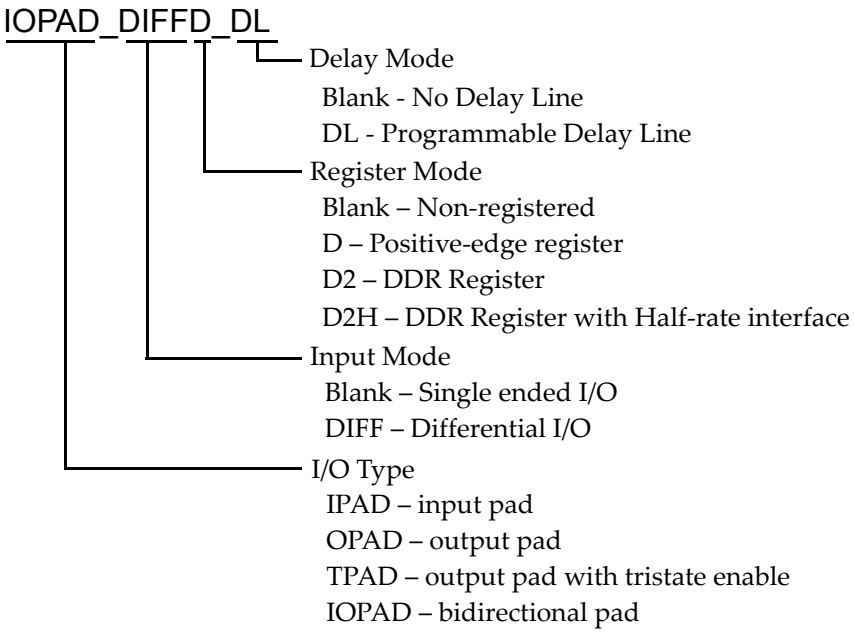
- **Chapter 1 – "I/O Cells"**
- **Chapter 2 – "Registers"**
- **Chapter 3 – "Logic Functions"**
- **Chapter 4 – "Lookup Table (LUT) Functions"**
- **Chapter 5 – "Arithmetic Functions"**
- **Chapter 6 – "Memories"**
- **Chapter 7 – "Multipliers"**
- **Chapter 8 – "Special Functions"**
- **Chapter 9 – "PLL/DLL Clock Generators"**

Cell Naming Conventions

Register Naming Conventions



I/O Cell Naming Conventions



Conventions Used in this Guide

Item	Format	Examples
Command-line entries	Formatted with the Courier bold font face.	$ **Open *top_level_name*.log**
File Names	Formatted with the Courier font face.	filename.ext
GUI buttons, menus and radio buttons	Formatted with the Helvetica bold font face.	Click **OK** to continue. **File → Open**
Variables	Formatted with italic emphasis.	*design_dir*/output.log
Window and dialog box headings and sub-headings	Heading formatted in quotation marks.	Under "Output Files," select ...
Window and dialog box names	Initial caps.	From the Add Files dialog box, ...

Chapter 1 – I/O Cells

Table 1-1: *Supported Single-Ended Voltage Standards*

I/O Standard Parameter	Output V_{DDO} (Volts)	Input V_{DDI} (Volts)	V_{REF} (Volts)[1]	Description
HSTL15_I	1.5	1.5	0.75	1.5V HSTL, type I
HSTL15_II	1.5	1.5	0.75	1.5V HSTL, type II
HSTL18_I	1.8	1.8	0.90	1.8V HSTL, type I
HSTL18_II	1.8	1.8	0.90	1.8V HSTL, type II
LVCMOS12	1.2	1.2	0.60	1.2V LVCMOS
LVCMOS15	1.5	1.5	0.75	1.5V LVCMOS
LVCMOS18	1.8	1.8	0.90	1.8V LVCMOS
SSTL15_I	1.5	1.5	0.75	1.5V SSTL, type I
SSTL15_II	1.5	1.5	0.75	1.5V SSTL, type II
SSTL18_I	1.8	1.8	0.90	1.8V SSTL, type I
SSTL18_II	1.8	1.8	0.90	1.8V SSTL, type II
POD18	1.8	1.8	0.90	1.8V POD
POD15	1.5	1.5	0.75	1.5V POD

Notes:
1. Set by V_{REF} Pin.

Table 1-2: *Supported Differential Voltage Standards*

I/O Standard Parameter	Output V_{DDO} (Volts)	Input V_{DDI} (Volts)	Description
DIFF_HSTL15_I	***1.50***	***1.50***	***Differential 1.5V HSTL, type I***
DIFF_HSTL15_II	***1.50***	***1.50***	***Differential 1.5V HSTL, type II***
DIFF_HSTL18_I	1.80	1.80	Differential 1.8V HSTL, type I
DIFF_HSTL18_II	1.80	1.80	Differential 1.8V HSTL, type II
DIFF_SSTL15_I	1.50	1.50	Differential 1.5V SSTL, type I
DIFF_SSTL15_II	1.50	1.50	Differential 1.5V SSTL, type II
DIFF_SSTL18_I	1.80	1.80	Differential 1.8V SSTL, type I
DIFF_SSTL18_II	1.80	1.80	Differential 1.8V SSTL, type II
HT1.0	1.20	1.20	Differential HyperTransport 1.0
LVDS	1.8	1.8	LVDS (Low Voltage Differential Signalling)

IOPAD

Bidirectional I/O Pad

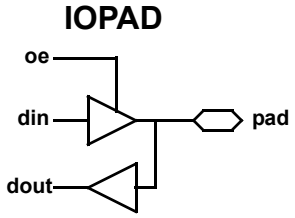


Figure 1-1: *IOPAD Logic Symbol*

IOPAD is an asynchronous I/O pad with active-high clock enable.

Table 1-3: *Ports*

Name	Type	Description
pad	inout	**Bidirectional device pad.**
din	input	**Data input.**
dout	output	**Data output.** The dout pin is driven with the value present on the pad pin.

Table 1-4: *Parameters*

Parameter	Defined Values	Default Value
location	“<pad_location>”	“”
iostandard	See **Table 1-1**	“LVCMOS18”
drive	"2", "4", "6", "8", "12", "16"	"16"
slew	“fast”, “slow”	“slow”
keepmode	"pullup", "pulldown", "none"	“none”
hysteresis	"none", "schmitt"	“none”
open_drain	“true”, “false”	“false”
pvt_comp	“none”, “own”	“none”
odt	“off”, “on”	“off”
termination	“50”, “60”, “75”, “100”, “120”, “240”	“50”

Table 1-5: *Output Function Table*

din	oe	pad
X	0	Z
0	1	0
1	1	1

Table 1-6: *Input Function Table*

pad	dout
0	0
1	1
X	X
Z	X

Verilog Instantiation Template

```
IOPAD #(.location(""),
     .iostandard("LVCMOS18"),
     .drive("16"),
     .slew("slow"),
     .keepmode("none"),
     .open_drain("false"),
     .hysteresis("none"),
     .pvt_comp("none"),
     .termination("50"),
     .odt("off"))
instance_name (.din(user_din), .dout(user_dout), .oe(user_oe),
.pad(user_pad));
```

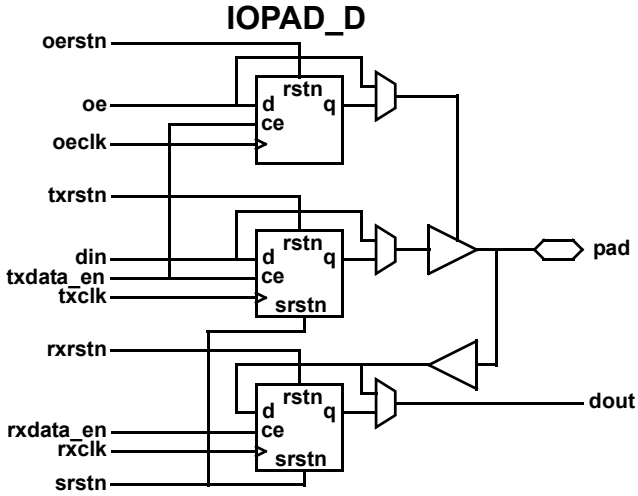
VHDL Instantiation Template

```
------------- ACHRONIX LIBRARY ------------
library speedster22i;
use speedster22i.components.all;
------------- DONE ACHRONIX LIBRARY ---------

-- Component Instantiation
IOPAD_instance_name : IOPAD
  generic map (location => "",
    iostandard => "LVCMOS18",
    drive => "16",
    slew => "slow",
    keepmode => "none",
    open_drain => "false",
    hysteresis => "none",
    pvt_comp => "none",
    termination => "50",
    odt => "off")
  port map ( din => user_din ,
           dout => user_dout ,
           oe => user_oe ,
           pad => user_pad);
```

IOPAD_D

Bidirectional Registered I/O Pad with Asynchronous or Synchronous Set/Reset



Note: For Speedster22iHP, txdata_en and rxdata_en are shared.
For Speedster22iHD, txdata_en and rxdata_en may be driven separately.

Figure 1-2: *IOPAD_D Logic Symbol*

IOPAD_D is an I/O pad where each of the transmit, receive, or output enable registers may be bypassed. The input, output, and output enable registers are all clocked on the rising edge of their respective clocks. The active-high oe register is asynchronously cleared upon a low on the oerstn input. Driving rxrstn low performs an asynchronous initialization of the input register. The value initialized into the input register is determined by the value of the rstvalue parameter. Similarly, driving txrstn low performs an asynchronous initialization of the output register. The value initialized into the output register is also determined by the value of the rstvalue parameter. Driving srstn low performs a synchronous initialization of both the input and output registers if the rstmode parameter is set to "sync". The value initialized into the input and output registers is also determined by the value of the rstvalue parameter. When the rstmode parameter is set to "sync", the input and output registers have their synchronous reset inputs active. The synchronous and asynchronous reset capability of the input and output registers is mutually exclusive. The input and output registers can both have a synchronous or both have asychronous resets as selected by the rstmode parameter.

Table 1-7: *Ports*

Name	Type	Description
pad	inout	**Bidirectional device pad.**
din	input	**Positive-edge based data input.** If parameter txregmode="reg", data is clocked into the din register upon the rising edge of the clk input, and is driven to the pad if the oe input was high before the rising edge of the clk input. If parameter txregmode="nonreg", din is driven to the pad when the output is enabled (oe=1).
dout	output	**Positive-edge based data output.** If parameter rxregmode="reg", data is clocked from the pad to dout on the rising edge of clk. If parameter rxregmode="nonreg", the input register is bypassed and the pad is driven onto output dout.
oe	input	**Output Enable.** The output enable register transitions upon the rising edge of the oeclk clock. A low value on the rstn input performs an asynchronous clear to disable the output when the output enable is set to registered mode by setting the oeregmode parameter to "reg". A high value on oe enables the output pad upon the next rising edge of the clock when the Output Enable register is enabled. A low value on oe disables the pad upon the next rising edge of the clock and places the pad in high impedance mode when the Output Enable register is enabled. If the output enable register is bypassed by setting the txregmode parameter to "nonreg", driving oe high immediately enables the pad.
txdata_en	input	**Output Register Clock Enable.** A high value on txdata_en enables the Output Register to clock the din input to the output at the next rising edge of the txclk. A low value on txdata_en allows the Output Register to retain its current value.
rxdata_en	input	**Input Register Clock Enable.** A high value on rxdata_en enables the Input Register to clock the value on pad into the Input Register at the next rising edge of the rxclk. A low value on rxdata_en allows the Input Register to retain its current value.
txrstn	input	**Output Register Asynchronous Reset.** A low value on txrstn performs an asynchronous initialization of the Output Register. The value initialized into the Output Register is determined by the value of the rstvalue parameter.
rxrstn	input	**Input Register Asynchronous Reset.** A low value on rxrstn performs an asynchronous initialization of the Input Register. The value initialized into the Input Register is determined by the value of the rstvalue parameter.
oerstn	input	**Output Enable Register Asynchronous Reset.** A low value on oerstn performs an asynchronous initialization of the Output Enable Register and initializes it to a low value to put the output in a high-impedance mode.
srstn	input	**Synchronous Reset.** A low value on srstn performs a synchronous initialization of the input register and output register if the rstmode parameter is set to "sync". Setting the rstmode parameter to "async" disables the synchronous reset.
txclk	input	**Output Register Clock input for transmit side.**
rxclk	input	**Input Register Clock input for receive side.**
oeclk	input	**Output Enable Register Clock input.**

Table 1-8: *Parameters*

Parameter	Defined Values	Default Value
location	"<pad_location>"	""
iostandard	See **Table 1-1**	"LVCMOS18"
drive	"2", "4", "6", "8", "12", "16"	"16"
txregmode	"reg", "nonreg"	"reg"
rxregmode	"reg", "nonreg"	"reg"
oeregmode	"reg", "nonreg"	"reg"
rstmode	"sync", "async"	"async"
rstvalue	"low", "high"	"low"
slew	"fast", "slow"	"slow"
keepmode	"pullup", "pulldown", "none"	"none"
hysteresis	"none", "schmitt"	"none"
open_drain	"true", "false"	"false"
pvt_comp	"none", "own"	"none"
odt	"off", "on"	"off"
termination	"50", "60", "75", "100", "120", "240"	"50"

Table 1-9: *Output Function Table (txregmode = "reg", oeregmode = "reg", rstmode = "async")*

din	txdata_en	oe	oerstn	txrstn	txclk	pad
X	X	X	0	X	X	Z
X	0	X	1	1	↑	Hold previous data
X	1	0	1	1	↑	Z
0	1	1	1	1	↑	0
1	1	1	1	1	↑	1
X	1	0	1	1	↑	Z

Table 1-10: *Output Function Table (txregmode = "reg", oeregmode = "reg", rstmode = "sync")*

din	txdata_en	oe	oerstn	txrstn	txclk	oeclk	pad
X	X	X	0	X	↑	↑	Z
X	0	X	1	1	↑	↑	Hold previous data
X	1	0	1	1	↑	↑	Z
0	1	1	1	1	↑	↑	0
1	1	1	1	1	↑	↑	1
X	1	0	1	1	↑	↑	Z

Table 1-11: *Input Function table (rxregmode = "reg")*

pad	rxdata_en	rxclk	dout
0	1	↑	0
1	1	↑	1
X	1	↑	X
Z	1	↑	X
X,Z	0	↑	Hold previous data

Verilog Instantiation Template

```
IOPAD_D #(.location(""),
     .iostandard("LVCMOS18"),
     .drive("16"),
     .txregmode("reg"),
     .rxregmode("reg"),
     .oeregmode("reg"),
     .rstmode("async"),
     .rstvalue("low"),
     .slew("slow"),
     .keepmode("none"),
     .hysteresis("none"),
     .open_drain("false"),
     .pvt_comp("none"),
     .termination("50"),
     .odt("off"))
instance_name (.pad(user_pad), .din(user_din), .dout(user_dout),
.oe(user_oe), .txdata_en(user_txdata_en), .rxdata_en(user_rxdata_en),
.txrstn(user_txrstn), .rxrstn(user_rxrstn), .oerstn(user_rxrxtn),
.srstn(user_txsrstn), ..txclk(user_txclk), .rxclk(user_rxclk),
.oeclk(user_oeclk));
```

VHDL Instantiation Template

```
------------- ACHRONIX LIBRARY ------------
library speedster22i;
use speedster22i.components.all;
------------- DONE ACHRONIX LIBRARY ---------
```

```
-- Component Instantiation
IOPAD_D_instance_name : IOPAD_D
  generic map (location => "",
               iostandard => "LVCMOS18",
               drive => "16",
               txregmode => "reg",
               rxregmode => "reg",
               oeregmode => "reg",
               rstmode => "async",
               rstvalue => "low",
               slew => "slow",
               keepmode => "none",
               hysteresis => "none",
               open_drain => "false",
               pvt_comp => "none,
               termination => "50",
               odt => "off")

  port map (pad => user_pad,
            din => user_din,
            dout => user_dout,
            oe => user_oe,
            txrstn => user_txrstn,
            srstn => user_srstn,
            rxrstn => user_rxrstn,
            txclk => user_txclk,
            rxclk => user_rxclk,
            oeclk => user_oeclk);
```

IOPAD_D2

Bidirectional DDR I/O Pad with Asynchronous or Synchronous Set/ Reset

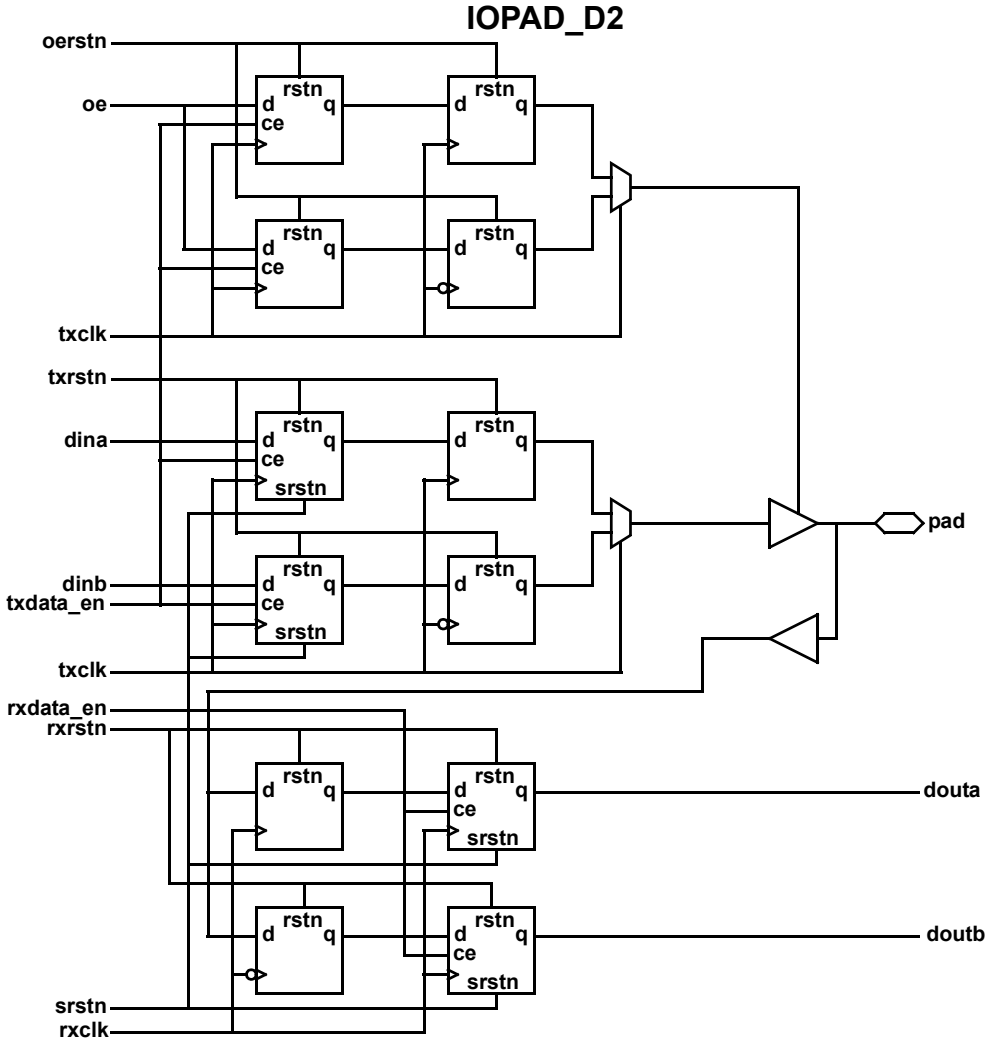


Figure 1-3: *IOPAD_D2 Logic Symbol*

IOPAD_D2 is a Double Data Rate (DDR) I/O pad with active-high registered output enable. There is an additional stage of registers on the inputs and outputs to allow the logic level on the pad to changes on both the rising and falling edges of the clock, but allow the interface signals to and from the FPGA core to change on the rising edge of the clock. This additional level of registers provides a full cycle to get into and out of the FPGA core.

P

Table 1-12: *Ports*

Name	Type	Description
pad	inout	**Bidirectional device pad.**
dina	input	**Positive-edge based data input.** Data is clocked into the dina register upon the rising edge of the txclk input when the txdata_en signal is high. It is routed to the pad on the following rising edge of the clock. If the oe input was high during the same clock period of the dina input, the pad will be actively driven with the dina data during the portion of the clock period when txclk is high.
dinb	input	**Negative-edge based data input.** Data is clocked into the dinb register upon the falling edge of the txclk input when the txdata_en signal is high. It is routed to the pad on the following rising edge of the clock. If the oe input was high during the same clock period of the dinb input, the pad will be actively driven with the dinb data during the portion of the clock period when txclk is low.
douta	output	**Positive-edge based data output.** Data is clocked from the pad to an internal register on the rising edge of the clock. On the next rising edge of the clock, if the rxdata_en input is high, the data will appear on the douta output.
doutb	output	**Negative-edge based data output.** Data is clocked from the pad to an internal register on the falling edge of the clock. On the next rising edge of the clock, if the rxdata_en input is high, the data will appear on the doutb output.
oe	input	**Output Enable** (active-high).The output enable input must be aligned with the dina and dinb inputs. The oe data is clocked into the oe register upon the rising edge of the txclk input when the txdata_en signal is high. It is routed to the enable of the pad on the following rising edge of the clock. The value of oe is applied to the interval of both the dina and dinb data at the pad. A high value on the oe input results in the dina and dinb data being driven onto the pad. If the value of oe at the enable intput of the pad driver is low, the pad will be placed in a high-impedance state.
txdata_en	input	**Transmit Data Enable** (active-high). A high value on txdata_en enables the dina, dinb, and oe inputs to be clocked into the transmit and output enable registers.
rxdata_en	input	Receive Data Enable (active-high). A high value on rxdata_en enables the received data to be clocked into the douta and doutb output registers.
txrstn	input	**Output Register Asynchronous Reset input.** A low value on txrstn performs an asynchronous initialization of the Output Register if the rstmode parameter is set to “async”. The value initialized into the Output Register is determined by the value of the rstvalue parameter. A low value on txrstn also asynchronously initializes the output enable register to a low value independent of the value of the rstmode parameter.
rxrstn	input	**Input Register Asynchronous Reset input.** A low value on rxrstn performs an asynchronous initialization of the Input Register if the rstmode parameter is set to “async”. The value initialized into the Input Register is determined by the value of the rstvalue parameter.
srstn	input	**Synchronous Reset Input.** A low value on srstn performs a synchronous initialization of the input registers and output registers if the rstmode parameter is set to "sync". Setting the rstmode parameter to "async" disables the synchronous reset.
txclk	input	**Output Register Clock Input.**
rxclk	input	**Input Register Clock Input.**

Table 1-12: *Ports (Continued)*

Name	Type	Description
oeclk	input	**Output Enable Register Clock Input.**

Table 1-13: *Parameters*

Parameter	Defined Values	Default Value
location	“<pad_location>”	“”
iostandard	See **Table 1-1**	“LVCMOS18”
drive	"2", "4", "6", "8", "12", "16"	"16"
rstmode	“sync”, “async”	“async”
rstvalue	“low”, “high”	“low”
slew	“fast”, “slow”	“slow”
keepmode	"pullup", "pulldown", "none"	“none”
open_drain	“true”, “false”	“false”
pvt_comp	“none”, “own”	“none”

Figure 1-4: *IOPAD_D2 Output Timing Diagram (assumes txdata_en = 1’b1)*

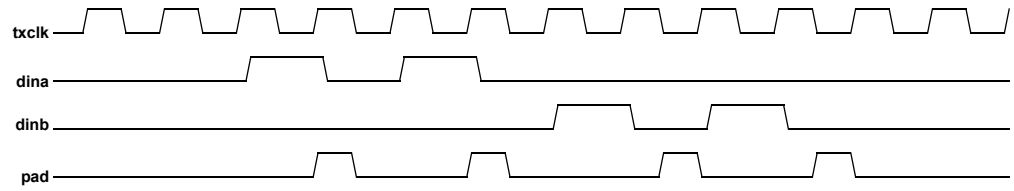
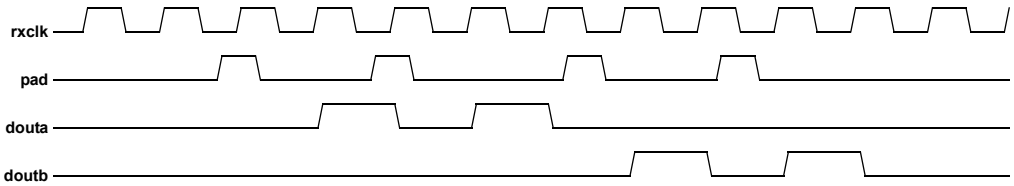


Figure 1-5: *IOPAD_D2 Input Timing Diagram (assumes rxdata_en = 1’b1)*



Verilog Instantiation Template

```
IOPAD_D2 #(.location(""),
     .iostandard("LVCMOS18"),
     .drive("16"),
     .rstmode("async"),
     .rstvalue("low"),
     .slew("slow"),
     .keepmode("none"),
     .hysteresis("none"),
     .open_drain("false"),
     .pvt_comp("none"),
     .termination("50"),
     .odt("off"))
instance_name (.pad(user_pad), .dina(user_dina), .dinb(user_dinb),
.douta(user_douta), .doutb(user_doutb), .oe(user_oe),
.txdata_en(user_txdata_en), .rxdata_en(user_rxdata_en),
.txrstn(user_txrstn), .rxrstn(user_rxrstn), .oerstn(user_rxrstn),
.srstn(user_txsrstn), ..txclk(user_txclk), .rxclk(user_rxclk),
.oeclk(user_oeclk));
```

VHDL Instantiation Template

```
------------ ACHRONIX LIBRARY ------------
library speedster22i;
use speedster22i.components.all;
------------ DONE ACHRONIX LIBRARY ---------

-- Component Instantiation
IOPAD_D2_instance_name : IOPAD_D2
  generic map (location => "",
               iostandard => "LVCMOS18",
               drive => "16",
               rstmode => "async",
               rstvalue => "low",
               slew => "slow",
               keepmode => "none",
               hysteresis => "none",
               open_drain => "false",
               pvt_comp => "none,
               termination => "50",
               odt => "off")

  port map (pad => user_pad,
            dina => user_dina,
            dinb => user_dinb,
            douta => user_douta,
            doutb => user_doutb,
            oe => user_oe,
            txrstn => user_txrstn,
            rxrstn => user_rxrstn,
            oerstn => user_oerstn,
            srstn => user_srstn,
            txclk => user_txclk,
            rxclk => user_rxclk,
            oeclk => user_oeclk);
```

IPAD

Non-Registered Input Pad

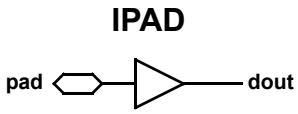


Figure 1-6: *IPAD Logic Symbol*

IPAD is an asynchronous input pad.

Table 1-14: *Ports*

Name	Type	Description
pad	input	**Device pad.**
dout	output	**Data output.**

Table 1-15: *Parameters*

Parameter	Defined Values	Default Value
location	“<pad_location>”	“”
iostandard	See **Table 1-1**	“LVCMOS18”
keepmode	"pullup", "pulldown", "none"	“none”
hysteresis	"none", "schmitt"	“none”
pvt_comp	“none”, “own”	“none”
odt	“off”, “on”	“off”
termination	“50”, “60”, “75”, “100”, “120”, “240”	“50”

Table 1-16: *Input Function Table*

pad	dout
0	0
1	1
X	X
Z	X

Verilog Instantiation Template

```
IPAD #(.location(""),
       .iostandard("LVCMOS18"),
       .keepmode("none"),
       .hysteresis("none"),
       .pvt_comp("none"),
       .termination("50"),
       .odt("off"))
instance_name (.dout(user_dout),
      .pad(user_pad));
```

VHDL Instantiation Template

```
------------- ACHRONIX LIBRARY ------------
library speedster22i;
use speedster22i.components.all;
------------- DONE ACHRONIX LIBRARY ---------

-- Component Instantiation
IPAD_instance_name : IPAD
  generic map (location => "",
               iostandard => "LVCMOS18",
               keepmode => "none",
               hysteresis => "none",
               pvt_comp => "none",
               termination => "50",
               odt => "off")
  port map (dout => user_dout,
            pad => user_pad);
```

IPAD_D

Registered Input Pad with Asynchronous or Synchronous Set/Reset

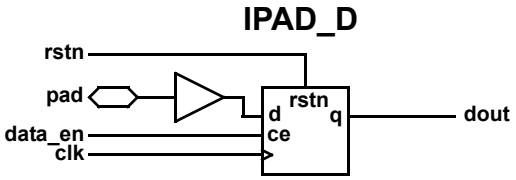


Figure 1-7: *IPAD_D Logic Symbol*

IPAD_D is a registered input pad. Driving rstn low performs either a synchronous or asynchronous reset of the input register as determined by the value of the rstmode parameter. Upon assertion of the rstn signal, the input register is initialized to the value determined by the rstvalue parameter.

Table 1-17: *Ports*

Name	Type	Description
pad	input	**Device pad.**
rstn	input	**Reset input.** The active-low rstn input performs either a synchronous or asynchronous set/reset operation as determined by the rstmode parameter. The value that is initialized into the register is determined by the value of the rstvalue parameter.
data_en	input	**Input Register Clock Enable.** A high value on data_en enables the Input Register to clock the value on pad into the Input Register at the next rising edge of clk. A low value on data_en allows the Input Register to retain its current value.
dout	output	**Positive-edge based data output.** Data is clocked from the pad to dout on the rising edge of clk.
clk	input	**Clock.**

Table 1-18: *Parameters*

Parameter	Defined Values	Default Value
location	“<pad_location>”	“”
iostandard	See **Table 1-1**	“LVCMOS18”
rstmode	“sync”, “async”	“async”
rstvalue	“low”, “high”	“low”
keepmode	"pullup", "pulldown", "none"	“none”
hysteresis	"none", "schmitt"	“none”
pvt_comp	“none”, “own”	“none”
odt	“off”, “on”	“off”
termination	“50”, “60”, “75”, “100”, “120”, “240”	“50”

Table 1-19: *Input Function table*

pad	rxdata_en	rxclk	dout
0	1	↑	0
1	1	↑	1
X	1	↑	X
Z	1	↑	X
X,Z	0	↑	Hold previous data

Verilog Instantiation Template

```
IPAD_D #( .location(""),
     .iostandard("LVCMOS18"),
     .rstmode("async"),
     .rstvalue("low"),
     .keepmode("none"),
     .hysteresis("none"),
     .pvt_comp("none"),
     .termination("50"),
     .odt("off"))
instance_name (.pad(user_pad),
               .dout(user_dout),
               .rstn(user_rstn),
               .data_en(user_data_en),
               .clk(user_clk));
```

VHDL Instantiation Template

```
------------- ACHRONIX LIBRARY ------------
library speedster22i;
use speedster22i.components.all;
------------- DONE ACHRONIX LIBRARY ---------

-- Component Instantiation
IPAD_D_instance_name : IPAD_D
  generic map (location => "",
               iostandard => "LVCMOS18",
               rstmode => "async",
               rstvalue => "low",
               keepmode => "none",
               hysteresis => "none",
               pvt_comp => "none",
               termination => "50",
               odt => "off")
  port map (dout => user_dout,
            pad => user_pad,
            clk => user_clk,
            data_en => user_data_en,
            rstn => user_rstn);
```

IPAD_D2

DDR Input Pad with Asynchronous or Synchronous Set/Reset

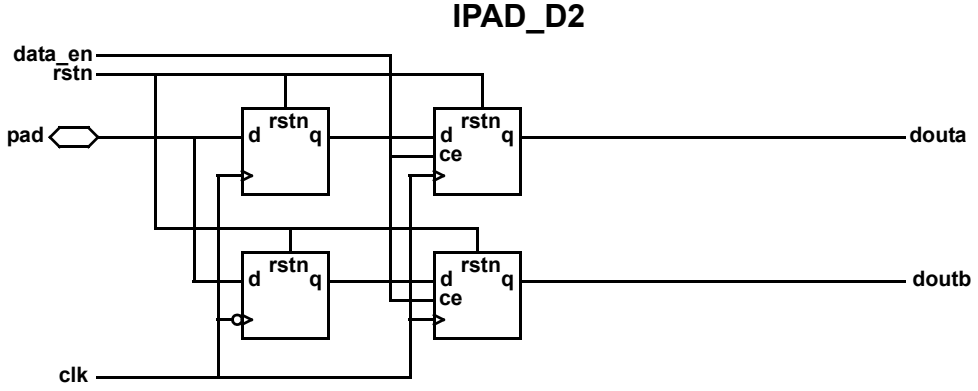


Figure 1-8: *IPAD_D2 Logic Symbol*

IPAD_D2 is a Double Data Rate (DDR) input pad. There is an additional register stage on the input to allow the logic level on the pad to changes on both the rising and falling edges of the clock, but allow the interface signals to and from the FPGA core to change on the rising edge of the clock. This additional level of registers provides a full cycle to get into and out of the FPGA core.

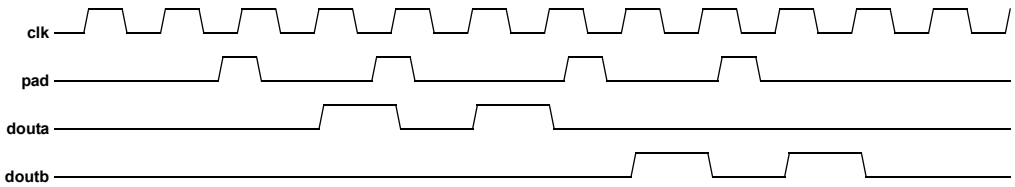
Table 1-20: *Ports*

Name	Type	Description
pad	input	**Bidirectional device pad.**
douta	output	**Positive-edge based data output.** Data is clocked from the pad to an internal register on the rising edge of the clock. On the next rising edge of the clock, if the data_en input is high, the data will appear on the douta output.
doutb	output	**Negative-edge based data output.** Data is clocked from the pad to an internal register on the falling edge of the clock. On the next rising edge of the clock, if the data_en input is high, the data will appear on the doutb output.
data_en	input	Receive Data Enable (active-high). A high value on rxdata_en enables the received data to be clocked into the douta and doutb output registers.
rstn	input	**Input Register Reset input.** A low value on rstn performs an asynchronous initialization of the Input Register if the rstmode parameter is set to “async”. A low value on rstn performs a synchronous initialization of the input registers if the rstmode parameter is set to "sync". The value initialized into the Input Register is determined by the value of the rstvalue parameter.
clk	input	**Input Register Clock Input.**

Table 1-21: *Parameters*

Parameter	Defined Values	Default Value
location	“<pad_location>”	“”
iostandard	See **Table 1-1**	“LVCMOS18”
drive	"2", "4", "6", "8", "12", "16"	"16"
rstmode	“sync”, “async”	“async”
rstvalue	“low”, “high”	“low”
slew	“fast”, “slow”	“slow”
keepmode	"pullup", "pulldown", "none"	“none”
hysteresis	"none", "schmitt"	“none”
open_drain	“true”, “false”	“false”
pvt_comp	“none”, “own”	“none”
odt	“off”, “on”	“off”
termination	“50”, “60”, “75”, “100”, “120”, “240”	“50”

Figure 1-9: *IPAD_D2 Input Timing Diagram (assumes data_en = 1’b1)*



Verilog Instantiation Template

```
IPAD_D2 #(.location(""),
     .iostandard("LVCMOS18"),
     .drive("16"),
     .rstmode("async"),
     .rstvalue("low"),
     .hysteresis("none"),
     .pvt_comp("none"),
     .termination("50"),
     .odt("off"))
instance_name (.pad(user_pad), .douta(user_douta), .doutb(user_doutb),
.data_en(user_data_en), .txrstn(user_txrstn), .rxrstn(user_rxrstn),
.rstn(user_rstn), .clk(user_clk));
```

VHDL Instantiation Template

```
------------ ACHRONIX LIBRARY ------------
library speedster22i;
use speedster22i.components.all;
------------ DONE ACHRONIX LIBRARY ---------

-- Component Instantiation
IPAD_D2_instance_name : IPAD_D2
  generic map (location => “”,
               iostandard => “LVCMOS18”,
               rstmode => “async”,
               rstvalue => “low”,
```

```
                    keepmode => "none",
                    hysteresis => "none",
                    pvt_comp => "none,
                    termination => "50",
                    odt => "off")

port map (pad => user_pad,
          douta => user_douta,
          doutb => user_doutb,
          oe => user_oe,
          rstn => user_rstn,
          clk => user_clk);
```

IPAD_DIFF

Non-Registered Differential Input Pad

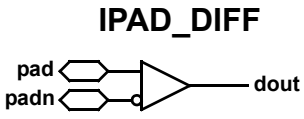


Figure 1-10: *IPAD_DIFF Logic Symbol*

IOPAD is a non-registered differential input pad with complementary input pads pad and padn with output dout.

Table 1-22: *Ports*

Name	Type	Description
pad	input	**Device pad.**
padn	input	**Complement Device pad.** The padn input must be driven with the logical complement of the pad input.
dout	output	**Data output.**

Table 1-23: *Parameters*

Parameter	Defined Values	Default Value
locationp	"<pad_location>"	""
locationn	"<pad_location>"	""
iostandard	See **Table 1-2**	"LVDS"
pvt_comp	"none", "own"	"none"
odt	"off", "on"	"off"
termination	"50", "60", "75", "100", "120", "240"	"50"

Table 1-24: *Input Function Table*

pad	padn	dout
0	0	X
0	1	0
1	0	1
1	1	X
Z	Z	X

Verilog Instantiation Template

```
IPAD_DIFF #(.locationp(""),
        .locationn(""),
        .iostandard("LVDS"),
        .pvt_comp("none"),
        .termination("50"),
        .odt("off"))
    instance_name (.dout(user_dout),
                   .pad(user_pad)
                   .padn(user_padn));
```

VHDL Instantiation Template

```
------------- ACHRONIX LIBRARY ------------
library speedster22i;
use speedster22i.components.all;
------------- DONE ACHRONIX LIBRARY ---------

-- Component Instantiation
IPAD_DIFF_instance_name : IPAD_DIFF
   generic map (locationp => "",
    locationn => "",
    iostandard => "LVDS",
    pvt_comp => "none",
    termination => "50",
    odt => "off")
  port map (dout => user_dout,
            pad => user_pad,
            padn => user_padn);
       .pad(user_pad));
```

IPAD_DIFFD

Registered Differential Input Pad with Asynchronous or Synchronous Set/Reset

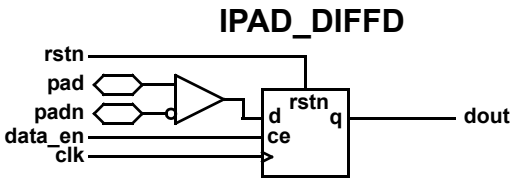


Figure 1-11: *IPAD_DIFFD Logic Symbol*

IPAD_DIFFD is a registered differential input pad. Driving rstn low performs either a synchronous or asynchronous reset of the input register as determined by the value of the rstmode parameter. Upon assertion of the rstn signal, the input register is initialized to the value determined by the rstvalue parameter.

Table 1-25: *Ports*

Name	Type	Description
pad	input	**Device pad.**
padn	input	**Comlement device pad.** The padn input must be driven with the logical complement of the pad input.
rstn	input	**Reset input.** The active-low rstn input performs either a synchronous or asynchronous set/reset operation as determined by the rstmode parameter. The value that is initialized into the register is determined by the value of the rstvalue parameter.
data_en	input	**Input Register Clock Enable.** A high value on data_en enables the Input Register to clock the value on pad into the Input Register at the next rising edge of clk. A low value on data_en allows the Input Register to retain its current value.
dout	output	**Positive-edge based data output.** Data is clocked from the differential pad to dout on the rising edge of clk.
clk	input	**Clock.**

Table 1-26: *Parameters*

Parameter	Defined Values	Default Value
locationp	"<pad_location>"	""
locationn	"<pad_location>"	""
iostandard	See Table 1-2	"LVDS"
rstmode	"sync", "async"	"async"
rstvalue	"low", "high"	"low"
pvt_comp	"none", "own"	"none"
odt	"off", "on"	"off"
termination	"50", "60", "75", "100", "120", "240"	"50"

Table 1-27: *Input Function table*

pad	padn	rxdata_en	rxclk	dout
0	1	1	↑	0
1	0	1	↑	1
X	X	1	↑	X
Z	Z	1	↑	X
X	X	0	↑	Hold previous data
Z	Z	0	↑	Hold previous data

Verilog Instantiation Template

```
IPAD_DIFFD #( .locationp(""),
     .locationn(""),
     .iostandard("LVDS"),
     .rstmode("async"),
     .rstvalue("low"),
     .pvt_comp("none"),
     .termination("50"),
     .odt("off"))
instance_name (.pad(user_pad),
               .padn(user_padn),
               .dout(user_dout),
               .rstn(user_rstn),
               .data_en(user_data_en),
               .clk(user_clk));
```

VHDL Instantiation Template

```
------------- ACHRONIX LIBRARY ------------
library speedster22i;
use speedster22i.components.all;
------------- DONE ACHRONIX LIBRARY ---------

-- Component Instantiation
IPAD_DIFFD_instance_name : IPAD_DIFFD
  generic map (location => "",
               iostandard => "LVDS",
               rstmode => "async",
               rstvalue => "low",
               pvt_comp => "none",
               termination => "50",
               odt => "off")
  port map (dout => user_dout,
            pad => user_pad,
            padn => user_padn,
            clk => user_clk,
            data_en => user_data_en,
            rstn => user_rstn);
```

IPAD_DIFFD2

DDR Differential Input Pad with Asynchronous or Synchronous Set/ Reset

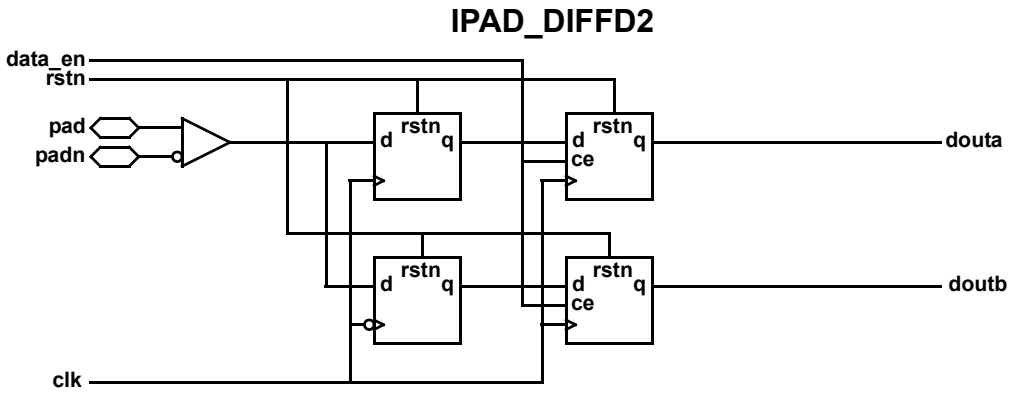


Figure 1-12: *IPAD_DIFFD2 Logic Symbol*

IPAD_D2 is a differential Double Data Rate (DDR) input pad. There is an additional register stage on the input to allow the logic level on the pad to changes on both the rising and falling edges of the clock, but allow the interface signals to and from the FPGA core to change on the rising edge of the clock. This additional level of registers provides a full cycle to get into and out of the FPGA core.

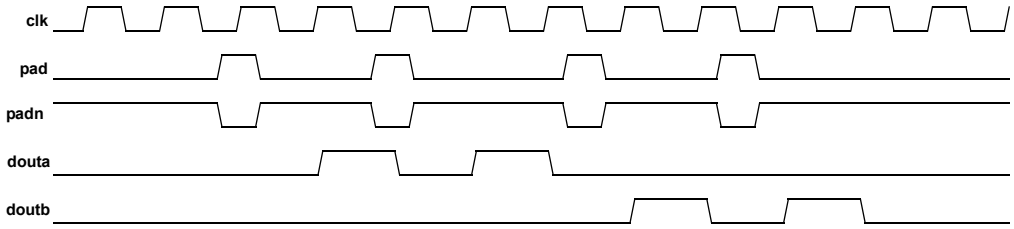
Table 1-28: *Ports*

Name	Type	Description
pad	input	**Device pad.**
padn	input	**Complement Device pad.**
douta	output	**Positive-edge based data output.** Data is clocked from the pad to an internal register on the rising edge of the clock. On the next rising edge of the clock, if the data_en input is high, the data will appear on the douta output.
doutb	output	**Negative-edge based data output.** Data is clocked from the pad to an internal register on the falling edge of the clock. On the next rising edge of the clock, if the data_en input is high, the data will appear on the doutb output.
data_en	input	Receive Data Enable (active-high). A high value on rxdata_en enables the received data to be clocked into the douta and doutb output registers.
rstn	input	**Input Register Reset input.** A low value on rstn performs an asynchronous initialization of the Input Register if the rstmode parameter is set to “async”. A low value on rstn performs a synchronous initialization of the input registers if the rstmode parameter is set to "sync". The value initialized into the Input Register is determined by the value of the rstvalue parameter.
clk	input	**Input Register Clock Input.**

Table 1-29: *Parameters*

Parameter	Defined Values	Default Value
locationp	“<pad_location>”	“”
locationn	“<padn_location>”	“”
iostandard	See **Table 1-1**	“LVCMOS18”
drive	"2", "4", "6", "8", "12", "16"	"16"
rstmode	“sync”, “async”	“async”
rstvalue	“low”, “high”	“low”
slew	“fast”, “slow”	“slow”
keepmode	"pullup", "pulldown", "none"	“none”
hysteresis	"none", "schmitt"	“none”
open_drain	“true”, “false”	“false”
pvt_comp	“none”, “own”	“none”
odt	“off”, “on”	“off”
termination	“50”, “60”, “75”, “100”, “120”, “240”	“50”

Figure 1-13: *IPAD_DIFFD2 Input Timing Diagram (assumes data_en = 1’b1)*



Verilog Instantiation Template

```
IPAD_DIFFD2 #(.locationp(""),
     .locationn(""),
     .iostandard("LVCMOS18"),
     .drive("16"),
     .rstmode("async"),
     .rstvalue("low"),
     .hysteresis("none"),
     .pvt_comp("none"),
     .termination("50"),
     .odt("off"))
instance_name (.pad(user_pad), .padn(user_padn), .douta(user_douta),
.doutb(user_doutb), .data_en(user_data_en), .txrstn(user_txrstn),
.rxrstn(user_rxrstn), .rstn(user_rstn), .clk(user_clk));
```

VHDL Instantiation Template

```
------------- ACHRONIX LIBRARY ------------
library speedster22i;
use speedster22i.components.all;
------------- DONE ACHRONIX LIBRARY ---------

-- Component Instantiation
IPAD_DIFFD2_instance_name : IPAD_DIFFD2
```

```
generic map (location => "",
             iostandard => "LVCMOS18",
             rstmode => "async",
             rstvalue => "low",
             keepmode => "none",
             hysteresis => "none",
             pvt_comp => "none,
             termination => "50",
             odt => "off")

port map (pad => user_pad,
          padn => user_padn,
          douta => user_douta,
          doutb => user_doutb,
          oe => user_oe,
          rstn => user_rstn,
          clk => user_clk);
```

OPAD

Non-Registered Output Pad

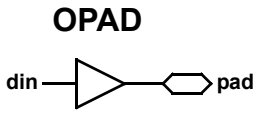


Figure 1-14: *OPAD Logic Symbol*

OPAD is an non-registered output pad.

Table 1-30: *Ports*

Name	Type	Description
din	input	**Data input.**
pad	output	**Device output pad.** The data at the din input is driven to the pad output.

Table 1-31: *Parameters*

Parameter	Defined Values	Default Value
location	“<pad_location>”	“”
iostandard	See **Table 1-1**	“LVCMOS18”
drive	"2", "4", "6", "8", "12", "16"	"16"
slew	“fast”, “slow”	“slow”
open_drain	“true”, “false”	“false”
pvt_comp	“none”, “own”	“none”

Table 1-32: *Output Function Table*

din	pad
0	0
1	1

Verilog Instantiation Template

```
OPAD #(.location(""),
 .iostandard("LVCMOS18"),
 .drive("16"),
 .slew("slow"),
 .pvt_comp("none"))
 instance_name (.din(user_din),
                .pad(user_pad));
```

VHDL Instantiation Template

```
------------- ACHRONIX LIBRARY ------------
library speedster22i;
use speedster22i.components.all;
------------- DONE ACHRONIX LIBRARY ---------

-- Component Instantiation
OPAD_instance_name : OPAD
    generic map (location => "",
    iostandard => "LVCMOS18",
    drive => "16",
    slew => "slow",
    pvt_comp => "none")
  port map (din => user_din,
            pad => user_pad);
```

OPAD_D

Registered Output Pad with Asynchronous or Synchronous Set/Reset

OPAD_D

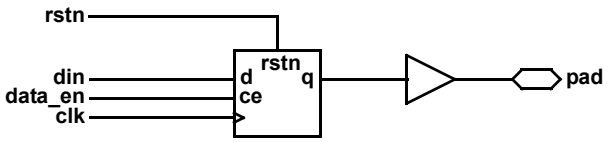


Figure 1-15: *OPAD_D Logic Symbol*

OPAD_D is a registered output pad. The output register is clocked on the rising edge of the clock. Driving rstn low performs an asynchronous initialization of the output register if the rstmode parameter is set to async and performs a sychronous initialization of the output register if the rstmode parameter is set to sync. The value initialized into the output register is determined by the value of the rstvalue parameter.

Table 1-33: *Ports*

Name	Type	Description
pad	output	**Device output pad.**
din	input	**Positive-edge based data input.** Data is clocked into the output register upon the rising edge of the clk input, and is driven to the pad.
data_en	input	**Output Register Clock Enable.** A high value on data_en enables the Output Register to clock the din input to the output at the next rising edge of the clock. A low value on data_en allows the Output Register to retain its current value.
rstn	input	**Output Register Reset.** If the value of the rstmode parameter is “async”, a low value on the rstn input performs an asynchronous initialization of the Output Register. If the value of the rstmode parameter is “sync”, a low value on the rstn input performs a synchronous initialization of the Output Register on the next rising edge of the clock. The value initialized into the Output Register is determined by the value of the rstvalue parameter.
clk	input	**Output Register Clock input.**

Table 1-34: *Parameters*

Parameter	Defined Values	Default Value
location	“<pad_location>”	“”
iostandard	See **Table 1-1**	“LVCMOS18”
drive	"2", "4", "6", "8", "12", "16"	"16"
rstmode	“sync”, “async”	“async”
rstvalue	“low”, “high”	“low”
slew	“fast”, “slow”	“slow”
open_drain	“true”, “false”	“false”
pvt_comp	“none”, “own”	“none”

Table 1-35: *Output Function Table (rstmode = “async”)*

din	data_en	rstn	clk	pad
0	1	1	↑	0
1	1	1	↑	1

Table 1-36: *Output Function Table (rstmode = “sync”)*

din	data_en	rstn	clk	pad
0	1	1	↑	0
1	1	1	↑	1

Verilog Instantiation Template

```
OPAD_D #(.location(""),
     .iostandard("LVCMOS18"),
     .drive("16"),
     .rstmode("async"),
     .rstvalue("low"),
     .slew("slow"),
     .open_drain("false"),
     .pvt_comp("none"))
instance_name (.pad(user_pad), .din(user_din), .data_en(user_data_en),
.rstn(user_rstn), .clk(user_clk));
```

VHDL Instantiation Template

```
------------- ACHRONIX LIBRARY ------------
library speedster22i;
use speedster22i.components.all;
------------- DONE ACHRONIX LIBRARY ---------

-- Component Instantiation
OPAD_D_instance_name : OPAD_D
  generic map (location => "",
               iostandard => "LVCMOS18",
               drive => "16",
               rstmode => "async",
               rstvalue => "low",
               slew => "slow",
               open_drain => "false",
               pvt_comp => "none")

  port map (pad => user_pad,
            din => user_din,
            rstn => user_rstn,
            clk => user_clk);
```

OPAD_D2

DDR Output Pad with Asynchronous or Synchronous Set/Reset

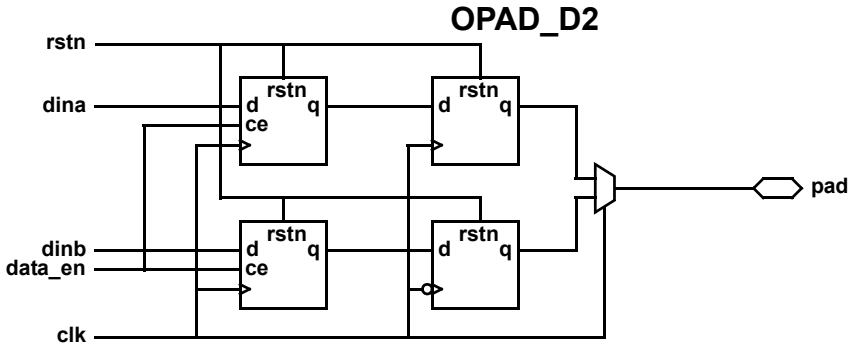


Figure 1-16: *OPAD_D2 Logic Symbol*

OPAD_D2 is a Double Data Rate (DDR) output pad with active-high registered output enable. There is an additional stage of registers on the outputs to allow the logic level on the pad to changes on both the rising and falling edges of the clock, but allow the interface signals to the FPGA core to change on the rising edge of the clock. This additional level of registers provides a full cycle to get into and out of the FPGA core.

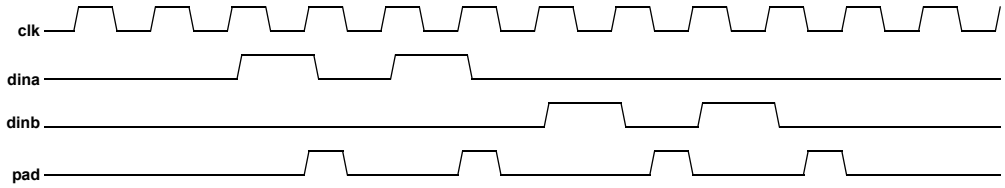
Table 1-37: *Ports*

Name	Type	Description
pad	output	**Device output pad.**
dina	input	**Positive-edge based data input.** Data is clocked into the dina register upon the rising edge of the txclk input when the txdata_en signal is high. It is routed to the pad on the following rising edge of the clock. If the oe input was high during the same clock period of the dina input, the pad will be actively driven with the dina data during the portion of the clock period when txclk is high.
dinb	input	**Negative-edge based data input.** Data is clocked into the dinb register upon the falling edge of the txclk input when the txdata_en signal is high. It is routed to the pad on the following rising edge of the clock. If the oe input was high during the same clock period of the dinb input, the pad will be actively driven with the dinb data during the portion of the clock period when txclk is low.
data_en	input	**Transmit Data Enable** (active-high). A high value on data_en enables the dina and dinb inputs to be clocked into the transmit registers.
rstn	input	**Asynchronous Reset input.** A low value on rstn performs an asynchronous initialization of the Output Register if the rstmode parameter is set to “async”. The value initialized into the Output Register is determined by the value of the rstvalue parameter.
clk	input	**Clock Input.**

Table 1-38: *Parameters*

Parameter	Defined Values	Default Value
location	“<pad_location>”	“”
iostandard	See **Table 1-1**	“LVCMOS18”
drive	"2", "4", "6", "8", "12", "16"	"16"
rstmode	“sync”, “async”	“async”
rstvalue	“low”, “high”	“low”
slew	“fast”, “slow”	“slow”
keepmode	"pullup", "pulldown", "none"	“none”
hysteresis	"none", "schmitt"	“none”
open_drain	“true”, “false”	“false”
pvt_comp	“none”, “own”	“none”
odt	“off”, “on”	“off”
termination	“50”, “60”, “75”, “100”, “120”, “240”	“50”

Figure 1-17: *OPAD_D2 Output Timing Diagram (assumes data_en = 1’b1)*



Verilog Instantiation Template

```
OPAD_D2 #(.location(""),
     .iostandard("LVCMOS18"),
     .drive("16"),
     .rstmode("async"),
     .rstvalue("low"),
     .slew("slow"),
     .open_drain("false"),
     .pvt_comp("none"))
instance_name (.pad(user_pad), .dina(user_dina), .dinb(user_dinb),
.data_en(user_data_en), .rstn(user_rstn), .clk(user_clk));
```

VHDL Instantiation Template

```
------------- ACHRONIX LIBRARY ------------
library speedster22i;
use speedster22i.components.all;
------------- DONE ACHRONIX LIBRARY ---------

-- Component Instantiation
OPAD_D2_instance_name : OPAD_D2
  generic map (location => "",
               iostandard => "LVCMOS18",
               drive => "16",
               rstmode => "async",
               rstvalue => "low",
               slew => "slow",
               open_drain => "false",
               pvt_comp => "none)

  port map (pad => user_pad,
            dina => user_dina,
            dinb => user_dinb,
            rstn => user_rstn,
            srstn => user_srstn,
            clk => user_clk);
```

OPAD_DIFF

Non-Registered Differential Output Pad

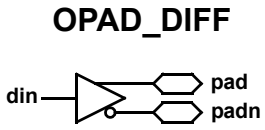


Figure 1-18: *OPAD_DIFF Logic Symbol*

OPAD_DIFF is an asynchronous differential output pad with input din and outputs pad and padn.

Table 1-39: *Ports*

Name	Type	Description
din	input	**Data input.**
pad	output	**Device output pad.** The data at the din input is driven to the pad output.
padn	output	**Device complement output pad.** The logical inverse of the data at the din input is driven to the padn output.

Table 1-40: *Parameters*

Parameter	Defined Values	Default Value
locationp	“<pad_location>”	“”
locationn	“<pad_location>”	“”
iostandard	See **Table 1-2**	“LVDS”
drive	"2", "4", "6", "8", "12", "16"	"16"
slew	“fast”, “slow”	“slow”
open_drain	“true”, “false”	“false”
pvt_comp	“none”, “own”	“none”
invert_out	“off”,”on”	“off”

Table 1-41: *Output Function Table*

din	pad	padn
0	0	1
1	1	0

Verilog Instantiation Template

```
OPAD_DIFF #(.locationp(""),
 .locationn(""),
 .iostandard("LVDS"),
 .drive("16"),
 .slew("slow"),
 .invert_out(“off”),
 .open_drain(“false”),
 .pvt_comp("none"))
```

```
 instance_name (.din(user_din), .pad(user_pad),
.padn(user_padn));
```

VHDL Instantiation Template

```
------------- ACHRONIX LIBRARY ------------
library speedster22i;
use speedster22i.components.all;
------------- DONE ACHRONIX LIBRARY ---------

-- Component Instantiation
OPAD_DIFF_instance_name : OPAD_DIFF
    generic map (locationp => "",
    locationn => "",
    iostandard => "LVDS",
    drive => "16",
    slew => "slow",
    pvt_comp => "none",
    open_drain => "false",
    invert_out => "off")
  port map (din => user_din,
            pad => user_pad,
            padn => user_padn);
```

OPAD_DIFFD

Registered Differential Output Pad with Asynchronous or Synchronous Set/Reset

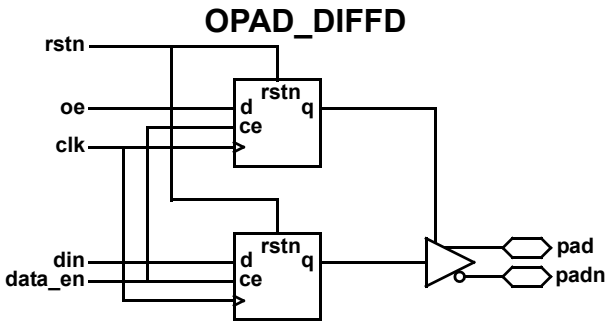


Figure 1-19: *OPAD_DIFFD Logic Symbol*

OPAD_DIFFD is a registered differential output pad. The output and output enable registers are clocked on the rising edge of the clock. The active-high oe register is asynchronously cleared upon a low on the rstn input. Driving rstn low performs an asynchronous initialization of the output register if the rstmode parameter is set to async and performs a sychronous initialization of the output register if the rstmode parameter is set to sync. The value initialized into the output register is determined by the value of the rstvalue parameter.

Table 1-42: *Ports*

Name	Type	Description
pad	output	**Device output pad.**
padn	output	**Device complement output pad.**
din	input	**Positive-edge based data input.** Data is clocked into the output register upon the rising edge of the clk input, and is driven to the pad if the oe input was high before the rising edge of the clk input.
oe	input	**Output Enable.** The output enable register transitions upon the rising edge of the clock. A low value on the rstn input performs an asynchronous clear to disable the output . A high value on oe enables the output pad upon the next rising edge of the clock. A low value on oe disables the pad upon the next rising edge of the clock and places the pad in high impedance mode.
data_en	input	**Output Register Clock Enable.** A high value on data_en enables the Output Register and Output Enable Register to clock the din input to the output at the next rising edge of the clock. A low value on data_en allows the Output Register to retain its current value.
rstn	input	**Output Register Asynchronous Reset.** A low value on the rstn input performs an asynchronous clear of the Output Enable Register to disable the output. If the value of the rstmode parameter is “async”, a low value on the rstn input performs an asynchronous initialization of the Output Register. If the value of the rstmode parameter is “sync”, a low value on the rstn input performs a synchronous initialization of the Output Register on the next rising edge of the clock. The value initialized into the Output Register is determined by the value of the rstvalue parameter.
clk	input	**Output Register / Output Enable Register Clock input.**

Table 1-43: *Parameters*

Parameter	Defined Values	Default Value
locationp	“<pad_location>”	“”
locationn	“<pad_location>”	“”
iostandard	See **Table 1-2**	“LVDS”
drive	"2", "4", "6", "8", "12", "16"	"16"
rstmode	“sync”, “async”	“async”
rstvalue	“low”, “high”	“low”
slew	“fast”, “slow”	“slow”
open_drain	“true”, “false”	“false”
pvt_comp	“none”, “own”	“none”
invert_out	“off”, ”on”	“off”

Table 1-44: *Output Function Table (rstmode = “async”)*

din	data_en	oe	rstn	clk	pad	padn
X	X	X	0	X	Z	Z
X	0	X	1	↑	Hold previous data	Hold previous data
X	1	0	1	↑	Z	Z
0	1	1	1	↑	0	1
1	1	1	1	↑	1	0
X	1	0	1	↑	Z	Z

Table 1-45: *Output Function Table (rstmode = “sync”)*

din	data_en	oe	rstn	clk	pad	padn
X	X	X	0	↑	Z	Z
X	0	X	1	↑	Hold previous data	Hold previous data
X	1	0	1	↑	Z	Z
0	1	1	1	↑	0	1
1	1	1	1	↑	1	0
X	1	0	1	↑	Z	Z

Verilog Instantiation Template

```
IOPAD_DIFFD #(.locationp(""),
     .locationn(""),
     .iostandard("LVDS"),
     .drive("16"),
     .rstmode("async"),
     .rstvalue("low"),
     .slew("slow"),
     .invert_out("off"),
     .open_drain("false"),
     .pvt_comp("none"))
instance_name (.pad(user_pad), .padn(user_padn)
.din(user_din),  .oe(user_oe), .data_en(user_data_en),
.rstn(user_rstn), .clk(user_clk));
```

VHDL Instantiation Template

```
------------- ACHRONIX LIBRARY ------------
library speedster22i;
use speedster22i.components.all;
------------- DONE ACHRONIX LIBRARY ---------

-- Component Instantiation
OPAD_D_instance_name : OPAD_DIFFD
  generic map (location => "",
               iostandard => "LVDS",
               drive => "16",
               rstmode => "async",
               rstvalue => "low",
               slew => "slow",
               invert_out => "off",
               open_drain => "false",
               pvt_comp => "none")

  port map (pad => user_pad,
            padn => user_padn,
            din => user_din,
            oe => user_oe,
            rstn => user_rstn,
            clk => user_clk);
```

OPAD_DIFFD2

DDR Differenctial Output Pad with Asynchronous or Synchronous Set/Reset

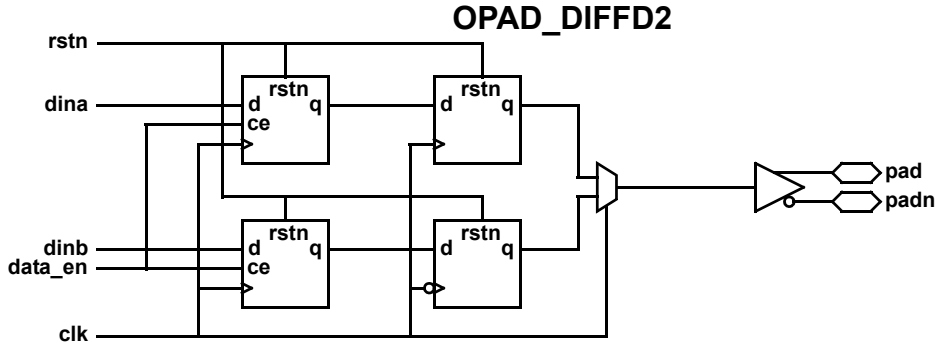


Figure 1-20: *OPAD_DIFFD2 Logic Symbol*

OPAD_DIFFD2 is a Double Data Rate (DDR) differential output pad with active-high registered output enable. There is an additional stage of registers on the outputs to allow the logic level on the pad to changes on both the rising and falling edges of the clock, but allow the interface signals from the FPGA core to change on the rising edge of the clock. This additional level of registers provides a full cycle to get into and out of the FPGA core.

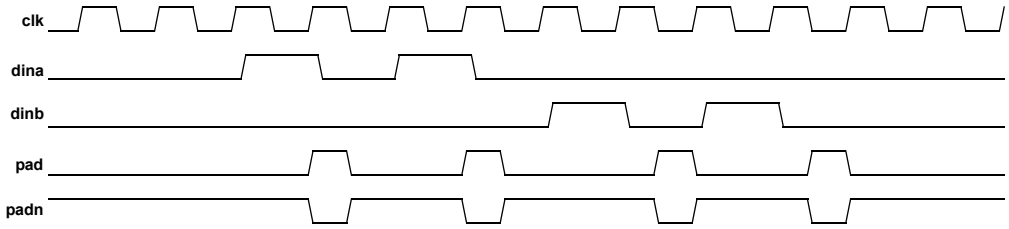
Table 1-46: *Ports*

Name	Type	Description
pad	output	**Device output pad.**
padn	output	**Device complement output pad.**
dina	input	**Positive-edge based data input.** Data is clocked into the dina register upon the rising edge of the txclk input when the txdata_en signal is high. It is routed to the pad on the following rising edge of the clock. If the oe input was high during the same clock period of the dina input, the pad will be actively driven with the dina data during the portion of the clock period when txclk is high.
dinb	input	**Negative-edge based data input.** Data is clocked into the dinb register upon the falling edge of the txclk input when the txdata_en signal is high. It is routed to the pad on the following rising edge of the clock. If the oe input was high during the same clock period of the dinb input, the pad will be actively driven with the dinb data during the portion of the clock period when txclk is low.
data_en	input	**Transmit Data Enable** (active-high). A high value on data_en enables the dina and dinb inputs to be clocked into the transmit registers.
rstn	input	**Asynchronous Reset input.** A low value on rstn performs an asynchronous initialization of the Output Register if the rstmode parameter is set to “async”. The value initialized into the Output Register is determined by the value of the rstvalue parameter.
clk	input	**Clock Input.**

Table 1-47: *Parameters*

Parameter	Defined Values	Default Value
locationp	“<pad_location>”	“”
locationn	“<padn_location>”	“”
iostandard	See **Table 1-1**	“LVCMOS18”
drive	"2", "4", "6", "8", "12", "16"	"16"
rstmode	“sync”, “async”	“async”
rstvalue	“low”, “high”	“low”
slew	“fast”, “slow”	“slow”
keepmode	"pullup", "pulldown", "none"	“none”
hysteresis	"none", "schmitt"	“none”
invert_out	“on”, “off”	“off”
open_drain	“true”, “false”	“false”
pvt_comp	“none”, “own”	“none”
odt	“off”, “on”	“off”
termination	“50”, “60”, “75”, “100”, “120”, “240”	“50”

Figure 1-21: *OPAD_DIFFD2 Output Timing Diagram (assumes data_en = 1’b1)*



Verilog Instantiation Template

```
OPAD_D2 #(.locationp(""),
     .locationn(""),
     .iostandard("LVCMOS18"),
     .drive("16"),
     .rstmode("async"),
     .rstvalue("low"),
     .slew("slow"),
     .open_drain("false"),
     .invert_out("off"),
     .pvt_comp("none"))
instance_name (.pad(user_pad), .dina(user_dina), .dinb(user_dinb),
.data_en(user_data_en), .rstn(user_rstn), .clk(user_clk));
```

VHDL Instantiation Template

```
------------- ACHRONIX LIBRARY ------------
library speedster22i;
use speedster22i.components.all;
------------- DONE ACHRONIX LIBRARY ---------

-- Component Instantiation
OPAD_D2_instance_name : OPAD_D2
  generic map (locationp => "",
               locationn => "",
               iostandard => "LVCMOS18",
               drive => "16",
               rstmode => "async",
               rstvalue => "low",
               slew => "slow",
               invert_out => "off",
               open_drain => "false",
               pvt_comp => "none)

  port map (pad => user_pad,
            dina => user_dina,
            dinb => user_dinb,
            rstn => user_rstn,
            srstn => user_srstn,
            clk => user_clk);
```

TPAD

Non-Registered Tristate Output Pad

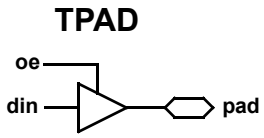


Figure 1-22: *TPAD Logic Symbol*

TPAD is an non-registered tristate output pad.

Table 1-48: *Ports*

Name	Type	Description
din	input	**Data input.**
oe	input	**Output Enbale.** The data at the din input is driven to the pad output when the oe input is driven high. The pad output will be driven into high-impedance mode when oe is low.
pad	output	**Device output pad.**

Table 1-49: *Parameters*

Parameter	Defined Values	Default Value
location	“<pad_location>”	“”
iostandard	See **Table 1-1**	“LVCMOS18”
drive	"2", "4", "6", "8", "12", "16"	"16"
slew	“fast”, “slow”	“slow”
open_drain	“true”, “false”	“false”
keepmode	“pullup”, “pulldown”, “none”	“none”
pvt_comp	“none”, “own”	“none”

Table 1-50: *Output Function Table*

din	oe	pad
0	1	0
1	1	1
X	0	Z

Verilog Instantiation Template

```
TPAD #(.location(""),
 .iostandard("LVCMOS18"),
 .drive("16"),
 .slew("slow"),
 .open_drain(“false”),
 .keepmode(“none”),
 .pvt_comp("none"))
 instance_name (.din(user_din), .oe(user_oe),
.pad(user_pad));
```

VHDL Instantiation Template

```
------------- ACHRONIX LIBRARY ------------
library speedster22i;
use speedster22i.components.all;
------------- DONE ACHRONIX LIBRARY ---------

-- Component Instantiation
TPAD_instance_name : TPAD
    generic map (location => "",
    iostandard => "LVCMOS18",
    drive => "16",
    slew => "slow",
    keepmode => “none”,
    open_drain => “false”,
    pvt_comp => "none")
  port map (din => user_din,
            oe => user_oe,
            pad => user_pad);
```

TPAD_D

Registered Tristate Output Pad with Asynchronous or Synchronous Set/Reset

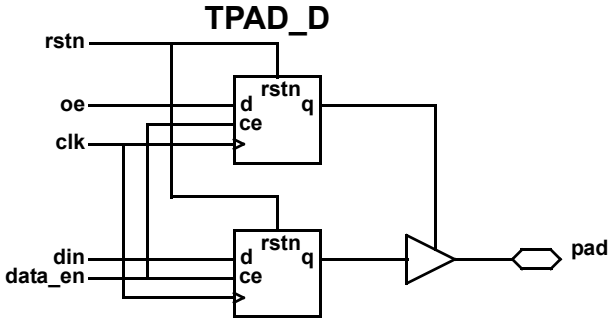


Figure 1-23: *TPAD_D Logic Symbol*

TPAD_D is a registered output pad. The output and output enable registers are clocked on the rising edge of the clock. The active-high oe register is asynchronously cleared upon a low on the rstn input. Driving rstn low performs an asynchronous initialization of the output register if the rstmode parameter is set to async and performs a sychronous initialization of the output register if the rstmode parameter is set to sync. The value initialized into the output register is determined by the value of the rstvalue parameter.

Table 1-51: *Ports*

Name	Type	Description
pad	output	**Device output pad.**
din	input	**Positive-edge based data input.** Data is clocked into the output register upon the rising edge of the clk input, and is driven to the pad if the oe input was high before the rising edge of the clk input.
oe	input	**Output Enable.** The output enable register transitions upon the rising edge of the clock. A low value on the rstn input performs an asynchronous clear to disable the output . A high value on oe enables the output pad upon the next rising edge of the clock. A low value on oe disables the pad upon the next rising edge of the clock and places the pad in high impedance mode.
data_en	input	**Output Register Clock Enable.** A high value on data_en enables the Output Register and Output Enable Register to clock the din input to the output at the next rising edge of the clock. A low value on data_en allows the Output Register to retain its current value.
rstn	input	**Output Register Asynchronous Reset.** A low value on the rstn input performs an asynchronous clear of the Output Enable Register to disable the output. If the value of the rstmode parameter is “async”, a low value on the rstn input performs an asynchronous initialization of the Output Register. If the value of the rstmode parameter is “sync”, a low value on the rstn input performs a synchronous initialization of the Output Register on the next rising edge of the clock. The value initialized into the Output Register is determined by the value of the rstvalue parameter.
clk	input	**Output Register / Output Enable Register Clock input.**

Table 1-52: *Parameters*

Parameter	Defined Values	Default Value
location	“<pad_location>”	“”
iostandard	See **Table 1-1**	“LVCMOS18”
drive	"2", "4", "6", "8", "12", "16"	"16"
rstmode	“sync”, “async”	“async”
rstvalue	“low”, “high”	“low”
slew	“fast”, “slow”	“slow”
open_drain	“true”, “false”	“false”
pvt_comp	“none”, “own”	“none”

Table 1-53: *Output Function Table (rstmode = “async”)*

din	data_en	oe	rstn	clk	pad
X	X	X	0	X	Z
X	0	X	1	↑	Hold previous data
X	1	0	1	↑	Z
0	1	1	1	↑	0
1	1	1	1	↑	1
X	1	0	1	↑	Z

Table 1-54: *Output Function Table (rstmode = “sync”)*

din	data_en	oe	rstn	clk	pad
X	X	X	0	↑	Z
X	0	X	1	↑	Hold previous data
X	1	0	1	↑	Z
0	1	1	1	↑	0
1	1	1	1	↑	1
X	1	0	1	↑	Z

Verilog Instantiation Template

```
TPAD_D #(.location(""),
     .iostandard("LVCMOS18"),
     .drive("16"),
     .rstmode("async"),
     .rstvalue("low"),
     .slew("slow"),
     .open_drain("false"),
     .pvt_comp("none"))
instance_name (.pad(user_pad), .din(user_din),  .oe(user_oe),
.data_en(user_data_en), .rstn(user_rstn), .clk(user_clk));
```

VHDL Instantiation Template

```
------------- ACHRONIX LIBRARY ------------
library speedster22i;
use speedster22i.components.all;
------------- DONE ACHRONIX LIBRARY ---------

-- Component Instantiation
TPAD_D_instance_name : TPAD_D
  generic map (location => "",
               iostandard => "LVCMOS18",
               drive => "16",
               rstmode => "async",
               rstvalue => "low",
               slew => "slow",
               open_drain => "false",
               pvt_comp => "none")

  port map (pad => user_pad,
            din => user_din,
            oe => user_oe,
            rstn => user_rstn,
            clk => user_clk);
```

Chapter 2 – Registers

Naming Convention

These Macros are named based upon their characteristics and behavior. In each case, the name begins with DFF for D-type Flip Flop. In addition to DFF each has one or more modifiers which indicates it's unique properties.

The modifiers are:

E - Enable

N - Negatively Clocked

R - Reset (has priority over enable)

S - Set (has priority over enable)

C - Clear (enable has priority)

P - Preset (enable has priority)

DFF

Positive Clock Edge D-Type Register

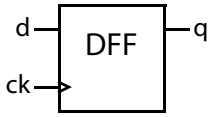


Figure 2-1: *Logic Symbol*

DFF is a single D-type register with data input (d) and clock (ck) inputs and data (q) output. The data output is set to the value on the data input upon the next rising edge of the clock.

Pins

Table 2-1: *Pin Descriptions*

Name	Type	Description
d	input	**Data input.**
ck	input	**Positive-edge clock input.**
q	output	**Data output.** The value present on the data input is transferred to the q output upon the rising edge of the clock.

Parameters

Table 2-2: *Parameters*

Parameter	Defined Values	Default Value
init	1'b0, 1'b1	1'b0

init

The init parameter defines the initial value of the output of the DFF register. This is the value the register takes upon the initial application of power to the FPGA. The default value of the init parameter is 1'b0.

Table 2-3: *Function Table*

Inputs		Output
d	ck	q
0	↑	0
1	↑	1

Verilog Instantiation Template

```
DFF #(.init(1'b0))
  instance_name
  (.q(user_out),
   .d(user_din),
   .ck(user_clock));
```

VHDL Instantiation Template

```
------------- ACHRONIX LIBRARY ------------
library speedster22i;
use speedster22i.components.all;
------------- DONE ACHRONIX LIBRARY ---------

-- Component Instantiation
DFF_instance_name : DFF
  generic map ( init => '0')
  port map (q => user_out,
            d => user_din,
            ck => user_clock);
```

DFFE

Positive Clock Edge D-Type Register with Clock Enable

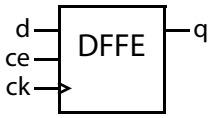


Figure 2-2: *Logic Symbol*

DFFE is a single D-type register with data input (d), clock enable (ce), and clock (ck) inputs and data (q) output. The data output is set to the value on the data input upon the next rising edge of the clock if the active-high clock enable input is asserted.

Pins

Table 2-4: *Pin Descriptions*

Name	Type	Description
d	input	**Data input.**
ce	input	**Active-high clock enable input**.
ck	input	**Positive-edge clock input.**
q	output	**Data output.** The value present on the data input is transferred to the q output upon the rising edge of the clock if the clock enable input is high.

Parameters

Table 2-5: *Parameters*

Parameter	Defined Values	Default Value
init	1'b0, 1'b1	1'b0

init

The init parameter defines the initial value of the output of the DFFE register. This is the value the register takes upon the initial application of power to the FPGA. The default value of the init parameter is 1'b0.

Table 2-6: *Function Table*

Inputs			Output
ce	d	ck	q
0	X	X	Hold
1	0	↑	0
1	1	↑	1

Verilog Instantiation Template

```
DFFE #(.init(1'b0))
  instance_name
  (.q(user_out),
   .d(user_din),
   .ce(user_clock_enable),
   .ck(user_clock));
```

VHDL Instantiation Template

```
------------- ACHRONIX LIBRARY ------------
library speedster22i;
use speedster22i.components.all;
------------- DONE ACHRONIX LIBRARY ---------

-- Component Instantiation
DFFE_instance_name : DFFE
  generic map ( init => '0')
  port map (q => user_out,
            d => user_din,
            ce => user_clock_enable,
            ck => user_clock);
```

DFFEC

Positive Clock Edge D-Type Register with Clock Enable and Synchronous Clear

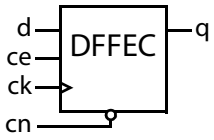


Figure 2-3: *Logic Symbol*

DFFEC is a single D-type register with data input (d), clock enable (ce), clock (ck), and active-low synchronous clear (cn) inputs and data (q) output. The active-low synchronous clear input sets the data output low upon the next rising edge of the clock if it is asserted low and the clock enable signal is asserted high. If the synchronout clear input is not asserted, the data output is set to the value on the data input upon the next rising edge of the clock if the active-high clock enable input is asserted.

Pins

Table 2-7: *Pin Descriptions*

Name	Type	Description
d	input	**Data input.**
cn	input	**Active-low synchronous clear input**. A low on cn sets the q output low upon the next rising edge of the clock if the clock enable is asserted high.
ce	input	**Active-high clock enable input**.
ck	input	**Positive-edge clock input.**
q	output	**Data output.** The value present on the data input is transferred to the q output upon the rising edge of the clock if the clock enable input is high and the synchronous clear input is high.

Parameters

Table 2-8: *Parameters*

Parameter	Defined Values	Default Value
init	1'b0, 1'b1	1'b0

init

The init parameter defines the initial value of the output of the DFFEC register. This is the value the register takes upon the initial application of power to the FPGA. The default value of the init parameter is 1'b0.

Table 2-9: *Function Table*

Inputs				Output
cn	**ce**	**d**	**ck**	**q**
X	0	X	X	Hold
0	1	X	↑	0
1	1	0	↑	0
1	1	1	↑	1

Verilog Instantiation Template

```
DFFEC #(.init(1'b0))
  instance_name
  (.q(user_out),
   .d(user_din),
   .cn(user_clear),
   .ce(user_clock_enable),
   .ck(user_clock));
```

VHDL Instantiation Template

```
------------- ACHRONIX LIBRARY ------------
library speedster22i;
use speedster22i.components.all;
------------- DONE ACHRONIX LIBRARY ---------

-- Component Instantiation
DFFEC_instance_name : DFFEC
  generic map ( init => '0')
  port map (q => user_out,
            d => user_din,
            cn => user_clear,
            ce => user_clock_enable,
            ck => user_clock);
```

DFFEP

Positive Clock Edge D-Type Register with Clock Enable and Synchronous Preset

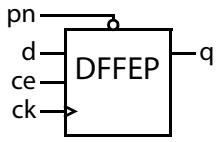


Figure 2-4: *Logic Symbol*

DFFEP is a single D-type register with data input (d), clock enable (ce), clock (ck), and active-low synchronous preset (pn) inputs and data (q) output. The active-low synchronous preset input sets the data output high upon the next rising edge of the clock if it is asserted low and the clock enable signal is asserted high. If the synchronous preset input is not asserted, the data output is set to the value on the data input upon the next rising edge of the clock if the active-high clock enable input is asserted.

Pins

Table 2-10: *Pin Descriptions*

Name	Type	Description
d	input	**Data input.**
pn	input	**Active-low synchronous preset input**. A low on pn sets the q output high upon the next rising edge of the clock if the clock enable is asserted high.
ce	input	**Active-high clock enable input**.
ck	input	**Positive-edge clock input.**
q	output	**Data output.** The value present on the data input is transferred to the q output upon the rising edge of the clock if the clock enable input is high and the synchronous preset input is high.

Parameters

Table 2-11: *Parameters*

Parameter	Defined Values	Default Value
init	1'b0, 1'b1	1'b1

init

The init parameter defines the initial value of the output of the DFFEP register. This is the value the register takes upon the initial application of power to the FPGA. The default value of the init parameter is 1'b1.

Table 2-12: *Function Table*

Inputs				Output
pn	**ce**	**d**	**ck**	**q**
X	0	X	X	Hold
0	1	X	↑	1
1	1	0	↑	0
1	1	1	↑	1

Verilog Instantiation Template

```
DFFEP #(.init(1'b1))
  instance_name
  (.q(user_out),
   .d(user_din),
   .pn(user_preset),
   .ce(user_clock_enable),
   .ck(user_clock));
```

VHDL Instantiation Template

```
------------- ACHRONIX LIBRARY ------------
library speedster22i;
use speedster22i.components.all;
------------- DONE ACHRONIX LIBRARY ---------

-- Component Instantiation
DFFEP_instance_name : DFFEP
  generic map ( init => '1')
  port map (q => user_out,
            d => user_din,
            pn => user_preset,
            ce => user_clock_enable,
            ck => user_clock);
```

DFFER

Positive Clock Edge D-Type Register with Clock Enable and Asynchronous/Synchronous Reset

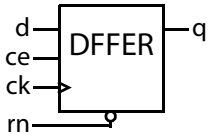


Figure 2-5: *Logic Symbol*

DFFER is a single D-type register with data input (d), clock enable (ce), clock (ck), and active-low reset (rn) inputs and data (q) output. The active-low reset input overrides all other inputs when it is asserted low and sets the data output low. The response of the q output in response to the asserted reset depends on the value of the sr_assertion parameter and is detailed in **Table 2-15: DFFER Function Table when sr_assertion = "unclocked"** and **Table 2-16: DFFER Function Table when sr_assertion = "clocked"**. If the reset input is not asserted, the data output is set to the value on the data input upon the next rising edge of the clock if the active-high clock enable input is asserted.

Pins

Table 2-13: *Pin Descriptions*

Name	Type	Description
d	input	**Data input.**
rn	input	**Active-low asynchronous/synchronous reset input**. A low on rn sets the q output low independent of the other inputs if the sr_assertion parameter is set to "unclocked". If the sr_assertion parameter is set to "clocked", a low on rn sets the q output low at the next rising edge of the clock.
ce	input	**Active-high clock enable input.**
ck	input	**Positive-edge clock input.**
q	output	**Data output.** The value present on the data input is transferred to the q output upon the rising edge of the clock if the clock enable input is high and the reset input is high.

Parameters

Table 2-14: *Parameters*

Parameter	Defined Values	Default Value
init	1'b0, 1'b1	1'b0
sr_assertion	"unclocked", "clocked"	"unclocked"

init

The init parameter defines the initial value of the output of the DFFER register. This is the value the register takes upon the initial application of power to the FPGA. The default value of the init parameter is 1'b0.

sr_assertion

The sr_assertion parameter defines the behavior of the output when the rn reset input is asserted. Assigning the sr_assertion to "unclocked" results in an asychronous assertion of the reset signal, where the q output is set to zero upon assertion of the active-low reset signal. Assigning the sr_assertion to "clocked" results in a synchronous assertion of the reset signal, where the q output is set to zero at the next rising edge of the clock. The default value of the sr_assertion parameter is "unclocked".

Table 2-15: *DFFER Function Table when sr_assertion = "unclocked"*

Inputs				Output
rn	**ce**	**d**	**ck**	**q**
0	X	X	X	0
1	0	X	X	Hold
1	1	0	↑	0
1	1	1	↑	1

Table 2-16: *DFFER Function Table when sr_assertion = "clocked"*

Inputs				Output
rn	**ce**	**d**	**ck**	**q**
0	X	X	↑	0
1	0	X	X	Hold
1	1	0	↑	0
1	1	1	↑	1

Verilog Instantiation Template

```
DFFER #(.init(1'b0),
        .sr_assertion("unclocked"))
  instance_name
   (.q(user_out),
    .d(user_din),
    .rn(user_reset),
    .ce(user_clock_enable),
    .ck(user_clock));
```

VHDL Instantiation Template

```
------------- ACHRONIX LIBRARY ------------
library speedster22i;
use speedster22i.components.all;
------------- DONE ACHRONIX LIBRARY ---------

-- Component Instantiation
DFFER_instance_name : DFFER
    generic map (
      init => '0',
      sr_assertion => "unclocked")
  port map (q => user_out,
            d => user_din,
            rn => user_reset,
            ce => user_clock_enable,
            ck => user_clock);
```

DFFES

Positive Clock Edge D-Type Register with Clock Enable and Asynchronous/Synchronous Set

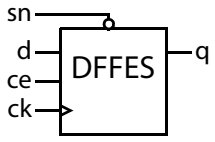


Figure 2-6: *Logic Symbol*

DFFES is a single D-type register with data input (d), clock enable (ce), clock (ck), and active-low set (sn) inputs and data (q) output. The active-low set input overrides all other inputs when it is asserted low and sets the data output high. The resonse of the q output in response to the asserted set depends on the value of the sr_assertion parameter and is detailed in **Table 2-19: DFFES Function Table when sr_assertion = "unclocked"** and **Table 2-20: DFFES Function Table when sr_assertion = "clocked"**. If the set input is not asserted, the data output is set to the value on the data input upon the next rising edge of the clock if the active-high clock enable input is asserted.

Pins

Table 2-17: *Pin Descriptions*

Name	Type	Description
d	input	**Data input.**
sn	input	**Active-low asynchronous/synchronous set input**. A low on sn sets the q output high independent of the other inputs if the sr_assertion parameter is set to "unclocked". If the sr_assertion parameter is set to "clocked", a low on rn sets the q output high at the next rising edge of the clock.
ce	input	**Active-high clock enable input.**
ck	input	**Positive-edge clock input.**
q	output	**Data output.** The value present on the data input is transferred to the q output upon the rising edge of the clock if the clock enable input is high and the reset input is high.

Parameters

Table 2-18: *Parameters*

Parameter	Defined Values	Default Value
init	1'b0, 1'b1	1'b1
sr_assertion	"unclocked", "clocked"	"unclocked"

init

The init parameter defines the initial value of the output of the DFFES register. This is the value the register takes upon the initial application of power to the FPGA. The default value of the init parameter is 1'b1.

sr_assertion

The sr_assertion parameter defines the behavior of the output when the sn set input is asserted. Assigning the sr_assertion to "unclocked" results in an asychronous assertion of the reset signal, where the q output is set to one upon assertion of the active-low reset signal. Assigning the sr_assertion to "clocked" results in a synchronous assertion of the reset signal, where the q output is set to one at the next rising edge of the clock. The default value of the sr_assertion parameter is "unclocked".

Table 2-19: *DFFES Function Table when sr_assertion = "unclocked"*

Inputs				Output
sn	**ce**	**d**	**ck**	**q**
0	X	X	X	1
1	0	X	X	Hold
1	1	0	↑	0
1	1	1	↑	1

Table 2-20: *DFFES Function Table when sr_assertion = "clocked"*

Inputs				Output
sn	**ce**	**d**	**ck**	**q**
0	X	X	↑	1
1	0	X	X	Hold
1	1	0	↑	0
1	1	1	↑	1

Verilog Instantiation Template

```
DFFES #(.init(1'b1),
        .sr_assertion("unclocked"))
  instance_name
   (.q(user_out),
    .d(user_din),
    .sn(user_set),
    .ce(user_clock_enable),
    .ck(user_clock));
```

VHDL Instantiation Template

```
------------- ACHRONIX LIBRARY ------------
library speedster22i;
use speedster22i.components.all;
------------- DONE ACHRONIX LIBRARY ---------

-- Component Instantiation
DFFES_instance_name : DFFES
    generic map (
      init => '1',
      sr_assertion => "unclocked")
  port map (q => user_out,
            d => user_din,
            sn => user_set,
            ce => user_clock_enable,
            ck => user_clock);
```

DFFN

Negative Clock Edge D-Type Register

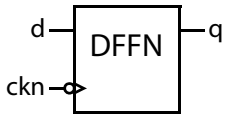


Figure 2-7: *Logic Symbol*

DFFN is a single D-type register with data input (d) and clock (ckn) inputs and data (q) output. The data output is set to the value on the data input upon the next falling edge of the clock.

Pins

Table 2-21: *Pin Descriptions*

Name	Type	Description
d	input	**Data input.**
ckn	input	**Negative-edge clock input.**
q	output	**Data output.** The value present on the data input is transferred to the q output upon the falling edge of the clock.

Parameters

Table 2-22: *Parameters*

Parameter	Defined Values	Default Value
init	1'b0, 1'b1	1'b0

init

The init parameter defines the initial value of the output of the DFFN register. This is the value the register takes upon the initial application of power to the FPGA. The default value of the init parameter is 1'b0.

Table 2-23: *Function Table*

Inputs		Output
d	ck	q
0	↓	0
1	↓	1

Verilog Instantiation Template

```
DFFN #(.init(1'b0))
  instance_name
  (.q(user_out),
   .d(user_din),
   .ckn(user_clock));
```

VHDL Instantiation Template

```
------------- ACHRONIX LIBRARY ------------
library speedster22i;
use speedster22i.components.all;
------------- DONE ACHRONIX LIBRARY ---------

-- Component Instantiation
DFFN_instance_name : DFFN
  generic map ( init => '0')
  port map (q => user_out,
            d => user_din,
            ckn => user_clock);
```

DFFNEC

Negative Clock Edge D-Type Register with Clock Enable and Synchronous Clear

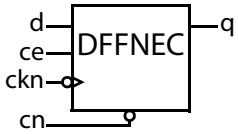


Figure 2-8: *Logic Symbol*

DFFNEC is a single D-type register with data input (d), clock enable (ce), clock (ckn), and active-low synchronous clear (cn) inputs and data (q) output. The active-low synchronous clear input sets the data output low upon the next falling edge of the clock if it is asserted low and the clock enable signal is asserted high. If the synchronous clear input is not asserted, the data output is set to the value on the data input upon the next falling edge of the clock if the active-high clock enable input is asserted.

Pins

Table 2-24: *Pin Descriptions*

Name	Type	Description
d	input	**Data input.**
cn	input	**Active-low synchronous clear input**. A low on cn sets the q output low upon the next falling edge of the clock if the clock enable is asserted high.
ce	input	**Active-high clock enable input**.
ckn	input	**Negative-edge clock input.**
q	output	**Data output.** The value present on the data input is transferred to the q output upon the falling edge of the clock if the clock enable input is high and the synchronous clear input is high.

Parameters

Table 2-25: *Parameters*

Parameter	Defined Values	Default Value
init	1'b0, 1'b1	1'b0

init

The init parameter defines the initial value of the output of the DFFNEC register. This is the value the register takes upon the initial application of power to the FPGA. The default value of the init parameter is 1'b0.

Table 2-26: *Function Table*

Inputs				Output
cn	**ce**	**d**	**ckn**	**q**
X	0	X	X	Hold
0	1	X	↓	0
1	1	0	↓	0
1	1	1	↓	1

Verilog Instantiation Template

```
DFFNEC #(.init(1'b0))
  instance_name
  (.q(user_out),
   .d(user_din),
   .cn(user_clear),
   .ce(user_clock_enable),
   .ckn(user_clock));
```

VHDL Instantiation Template

```
------------- ACHRONIX LIBRARY ------------
library speedster22i;
use speedster22i.components.all;
------------- DONE ACHRONIX LIBRARY ---------

-- Component Instantiation
DFFNEC_instance_name : DFFNEC
  generic map ( init => '0')
  port map (q => user_out,
            d => user_din,
            cn => user_clear,
            ce => user_clock_enable,
            ckn => user_clock);
```

DFFNEP

Negative Clock Edge D-Type Register with Clock Enable and Synchronous Preset

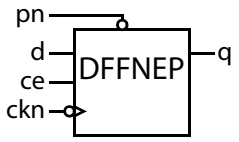


Figure 2-9: *Logic Symbol*

DFFNEP is a single D-type register with data input (d), clock enable (ce), clock (ckn), and active-low synchronous preset (pn) inputs and data (q) output. The active-low synchronous preset input sets the data output high upon the next falling edge of the clock if it is asserted low and the clock enable signal is asserted high. If the synchronous preset input is not asserted, the data output is set to the value on the data input upon the next falling edge of the clock if the active-high clock enable input is asserted.

Pins

Table 2-27: *Pin Descriptions*

Name	Type	Description
d	input	**Data input.**
pn	input	**Active-low synchronous preset input**. A low on pn sets the q output high upon the next falling edge of the clock if the clock enable is asserted high.
ce	input	**Active-high clock enable input**.
ckn	input	**Negative-edge clock input.**
q	output	**Data output.** The value present on the data input is transferred to the q output upon the falling edge of the clock if the clock enable input is high and the synchronous preset input is high.

Parameters

Table 2-28: *Parameters*

Parameter	Defined Values	Default Value
init	1'b0, 1'b1	1'b1

init

The init parameter defines the initial value of the output of the DFFNEP register. This is the value the register takes upon the initial application of power to the FPGA. The default value of the init parameter is 1'b1.

Table 2-29: *Function Table*

Inputs				Output
pn	**ce**	**d**	**ckn**	**q**
X	0	X	X	Hold
0	1	X	↓	1
1	1	0	↓	0
1	1	1	↓	1

Verilog Instantiation Template

```
DFFNEP #(.init(1'b1))
  instance_name
  (.q(user_out),
   .d(user_din),
   .pn(user_preset)
   .ce(user_clock_enable),
   .ckn(user_clock));
```

VHDL Instantiation Template

```
------------- ACHRONIX LIBRARY ------------
library speedster22i;
use speedster22i.components.all;
------------- DONE ACHRONIX LIBRARY ---------

-- Component Instantiation
DFFNEP_instance_name : DFFNEP
  generic map ( init => '1')
  port map (q => user_out,
            d => user_din,
            pn => user_preset,
            ce => user_clock_enable,
            ckn => user_clock);
```

DFFNER

Negative Clock Edge D-Type Register with Clock Enable and Asynchronous/Synchronous Reset

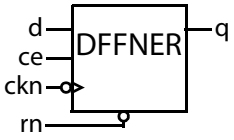


Figure 2-10: *Logic Symbol*

DFFNER is a single D-type register with data input (d), clock enable (ce), clock (ckn), and active-low reset (rn) inputs and data (q) output. The active-low reset input overrides all other inputs when it is asserted low and sets the data output low. The resonse of the q output in response to the asserted reset depends on the value of the sr_assertion parameter and is detailed in **Table 2-32: DFFNER Function Table when sr_assertion = "unclocked"** and **Table 2-33: DFFNER Function Table when sr_assertion = "clocked"**. If the reset input is not asserted, the data output is set to the value on the data input upon the next falling edge of the clock if the active-high clock enable input is asserted.

Pins

Table 2-30: *Pin Descriptions*

Name	Type	Description
d	input	**Data input.**
rn	input	**Active-low asynchronous/synchronous reset input**. A low on rn sets the q output low independent of the other inputs if the sr_assertion parameter is set to "unclocked". If the sr_assertion parameter is set to "clocked", a low on rn sets the q output low at the next falling edge of the clock.
ce	input	**Active-high clock enable input.**
ckn	input	**Negative-edge clock input.**
q	output	**Data output.** The value present on the data input is transferred to the q output upon the falling edge of the clock if the clock enable input is high and the reset input is high.

Parameters

Table 2-31: *Parameters*

Parameter	Defined Values	Default Value
init	1'b0, 1'b1	1'b0
sr_assertion	"unclocked", "clocked"	"unclocked"

init

The init parameter defines the initial value of the output of the DFFNER register. This is the value the register takes upon the initial application of power to the FPGA. The default value of the init parameter is 1'b0.

sr_assertion

The sr_assertion parameter defines the behavior of the output when the rn reset input is asserted. Assigning the sr_assertion to "unclocked" results in an asychronous assertion of the reset signal, where the q output is set to zero upon assertion of the active-low reset signal. Assigning the sr_assertion to "clocked" results in a synchronous assertion of the reset signal, where the q output is set to zero at the next falling edge of the clock. The default value of the sr_assertion parameter is "unclocked".

Table 2-32: *DFFNER Function Table when sr_assertion = "unclocked"*

Inputs				Output
rn	**ce**	**d**	**ckn**	**q**
0	X	X	X	0
1	0	X	X	Hold
1	1	0	↓	0
1	1	1	↓	1

Table 2-33: *DFFNER Function Table when sr_assertion = "clocked"*

Inputs				Output
rn	**ce**	**d**	**ckn**	**q**
0	X	X	↓	0
1	0	X	X	Hold
1	1	0	↓	0
1	1	1	↓	1

Verilog Instantiation Template

```
DFFNER #(.init(1'b0),
         .sr_assertion("unclocked"))
  instance_name
   (.q(user_out),
    .d(user_din),
    .rn(user_reset),
    .ce(user_clock_enable),
    .ckn(user_clock));
```

VHDL Instantiation Template

```
------------- ACHRONIX LIBRARY ------------
library speedster22i;
use speedster22i.components.all;
------------- DONE ACHRONIX LIBRARY ---------

-- Component Instantiation
DFFNER_instance_name : DFFNER
    generic map (
      init => '0',
      sr_assertion => "unclocked")
  port map (q => user_out,
            d => user_din,
            rn => user_reset,
            ce => user_clock_enable,
            ckn => user_clock);
```

DFFNES

Negative Clock Edge D-Type Register with Clock Enable and Asynchronous/Synchronous Set

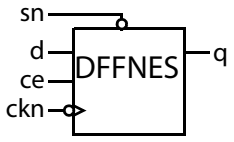


Figure 2-11: *Logic Symbol*

DFFNES is a single D-type register with data input (d), clock enable (ce), clock (ckn), and active-low set (sn) inputs and data (q) output. The active-low set input overrides all other inputs when it is asserted low and sets the data output high. The resonse of the q output in response to the asserted set depends on the value of the sr_assertion parameter and is detailed in **Table 2-36: DFFNES Function Table when sr_assertion = "unclocked"** and **Table 2-37: DFFNES Function Table when sr_assertion = "clocked"**. If the set input is not asserted, the data output is set to the value on the data input upon the next falling edge of the clock if the active-high clock enable input is asserted.

Pins

Table 2-34: *Pin Descriptions*

Name	Type	Description
d	input	**Data input.**
sn	input	**Active-low asynchronous/synchronous set input**. A low on sn sets the q output high independent of the other inputs if the sr_assertion parameter is set to "unclocked". If the sr_assertion parameter is set to "clocked", a low on sn sets the q output high at the next falling edge of the clock.
ce	input	**Active-high clock enable input.**
ckn	input	**Negative-edge clock input.**
q	output	**Data output.** The value present on the data input is transferred to the q output upon the falling edge of the clock if the clock enable input is high and the set input is high.

Parameters

Table 2-35: *Parameters*

Parameter	Defined Values	Default Value
init	1'b0, 1'b1	1'b1
sr_assertion	"unclocked", "clocked"	"unclocked"

init

The init parameter defines the initial value of the output of the DFFNES register. This is the value the register takes upon the initial application of power to the FPGA. The default value of the init parameter is 1'b1.

sr_assertion

The sr_assertion parameter defines the behavior of the output when the sn set input is asserted. Assigning the sr_assertion to "unclocked" results in an asychronous assertion of the set signal, where the q output is set to one upon assertion of the active-low set signal. Assigning the sr_assertion to "clocked" results in a synchronous assertion of the set signal, where the q output is set to one at the next falling edge of the clock. The default value of the sr_assertion parameter is "unclocked".

Table 2-36: *DFFNES Function Table when sr_assertion = "unclocked"*

Inputs				Output
sn	**ce**	**d**	**ckn**	**q**
0	X	X	X	1
1	0	X	X	Hold
1	1	0	↓	0
1	1	1	↓	1

Table 2-37: *DFFNES Function Table when sr_assertion = "clocked"*

Inputs				Output
sn	**ce**	**d**	**ckn**	**q**
0	X	X	↓	1
1	0	X	X	Hold
1	1	0	↓	0
1	1	1	↓	1

Verilog Instantiation Template

```
DFFNES #(.init(1'b1),
         .sr_assertion("unclocked"))
  instance_name
        (.q(user_out),
         .d(user_din),
         .sn(user_set),
         .ce(user_clock_enable),
         .ckn(user_clock));
```

VHDL Instantiation Template

```
------------- ACHRONIX LIBRARY ------------
library speedster22i;
use speedster22i.components.all;
------------- DONE ACHRONIX LIBRARY ---------

-- Component Instantiation
DFFNES_instance_name : DFFNES
    generic map (
      init => '1',
      sr_assertion => "unclocked")
  port map (q => user_out,
            d => user_din,
            sn => user_set,
            ce => user_clock_enable,
            ckn => user_clock);
```

DFFNR

Negative Clock Edge D-Type Register with Asynchronous Reset

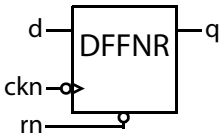


Figure 2-12: *Logic Symbol*

DFFNR is a single D-type register with data input (d), clock (ckn), and active-low reset (rn) inputs and data (q) output. The active-low reset input overrides all other inputs when it is asserted low and sets the data output low. If the asynchronous reset input is not asserted, the data output is set to the value on the data input upon the next falling edge of the clock.

Pins

Table 2-38: *Pin Descriptions*

Name	Type	Description
d	input	**Data input.**
rn	input	**Active-low asynchronous reset input**. A low on rn sets the q output low independent of the other inputs.
ckn	input	**Negative-edge clock input.**
q	output	**Data output.** The value present on the data input is transferred to the q output upon the falling edge of the clock if the asynchronous reset input is high.

Parameters

Table 2-39: *Parameters*

Parameter	Defined Values	Default Value
init	1'b0, 1'b1	1'b0
sr_assertion	"unclocked", "clocked"	"unclocked"

init

The init parameter defines the initial value of the output of the DFFNR register. This is the value the register takes upon the initial application of power to the FPGA. The default value of the init parameter is 1'b0.

sr_assertion

The sr_assertion parameter defines the behavior of the output when the sn set input is asserted. Assigning the sr_assertion to "unclocked" results in an asychronous assertion of the reset signal, where the q output is set to one upon assertion of the active-low reset signal. Assigning the sr_assertion to "clocked" results in a synchronous assertion of the reset signal, where the q output is set to one at the next rising edge of the clock. The default value of the sr_assertion parameter is "unclocked".

Table 2-40: *Function Table when sr_assertion = "unclocked'*

Inputs			Output
rn	**d**	**ckn**	**q**
0	X	X	0
1	X	X	Hold
1	0	↓	0
1	1	↓	1

Table 2-41: *Function Table when sr_assertion = "clocked'*

Inputs			Output
rn	**d**	**ckn**	**q**
0	X	↓	0
1	X	X	Hold
1	0	↓	0
1	1	↓	1

Verilog Instantiation Template

```
DFFNR #(.init(1'b0))
  instance_name
   (.q(user_out),
    .d(user_din),
    .rn(user_reset),
    .ckn(user_clock));
```

VHDL Instantiation Template

```
------------- ACHRONIX LIBRARY ------------
library speedster22i;
use speedster22i.components.all;
------------- DONE ACHRONIX LIBRARY ---------

-- Component Instantiation
DFFNR_instance_name : DFFNR
    generic map (
      init => '0')
  port map (q => user_out,
            d => user_din,
            rn => user_reset,
            ckn => user_clock);
```

DFFNS

Negative Clock Edge D-Type Register with Asynchronous Set

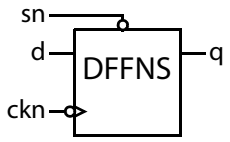


Figure 2-13: *Logic Symbol*

DFFNS is a single D-type register with data input (d), clock (ckn), and active-low set (sn) inputs and data (q) output. The active-low set input overrides all other inputs when it is asserted low and sets the data output high. If the asynchronous set input is not asserted, the data output is set to the value on the data input upon the next falling edge of the clock.

Pins

Table 2-42: *Pin Descriptions*

Name	Type	Description
d	input	**Data input.**
sn	input	**Active-low asynchronous set input.** A low on sn sets the q output high independent of the other inputs.
ckn	input	**Negative-edge clock input.**
q	output	**Data output.** The value present on the data input is transferred to the q output upon the falling edge of the clock if the asynchronous set input is high.

Parameters

Table 2-43: *Parameters*

Parameter	Defined Values	Default Value
init	1'b0, 1'b1	1'b1
sr_assertion	"unclocked", "clocked"	"unclocked"

init

The init parameter defines the initial value of the output of the DFFNS register. This is the value the register takes upon the initial application of power to the FPGA. The default value of the init parameter is 1'b1.

sr_assertion

The sr_assertion parameter defines the behavior of the output when the sn set input is asserted. Assigning the sr_assertion to "unclocked" results in an asychronous assertion of the reset signal, where the q output is set to one upon assertion of the active-low reset signal. Assigning the sr_assertion to "clocked" results in a synchronous assertion of the reset signal, where the q output is set to one at the next rising edge of the clock. The default value of the sr_assertion parameter is "unclocked".

Table 2-44: *Function Table when sr_assertion = "unclocked'*

Inputs			Output
sn	**d**	**ckn**	**q**
0	X	X	1
1	X	X	Hold
1	0	↓	0
1	1	↓	1

Table 2-45: *unction Table when sr_assertion = "clocked'*

Inputs			Output
sn	**d**	**ckn**	**q**
0	X	↓	1
1	X	X	Hold
1	0	↓	0
1	1	↓	1

Verilog Instantiation Template

```
DFFNS #(.init(1'b1))
  instance_name
   (.q(user_out),
    .d(user_din),
    .sn(user_set),
    .ckn(user_clock));
```

VHDL Instantiation Template

```
------------- ACHRONIX LIBRARY ------------
library speedster22i;
use speedster22i.components.all;
------------- DONE ACHRONIX LIBRARY ---------

-- Component Instantiation
DFFNS_instance_name : DFFNS
    generic map (
      init => '1')
  port map (q => user_out,
            d => user_din,
            sn => user_set,
            ckn => user_clock);
```

DFFR

Positive Clock Edge D-Type Register with Asynchronous Reset

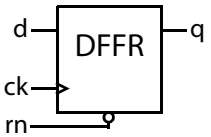


Figure 2-14: *Logic Symbol*

DFFR is a single D-type register with data input (d), clock (ck), and active-low reset (rn) inputs and data (q) output. The active-low reset input overrides all other inputs when it is asserted low and sets the data output low. If the asynchronous reset input is not asserted, the data output is set to the value on the data input upon the next rising edge of the clock.

Pins

Table 2-46: *Pin Descriptions*

Name	Type	Description
d	input	**Data input.**
rn	input	**Active-low asynchronous reset input**. A low on rn sets the q output low independent of the other inputs.
ck	input	**Positive-edge clock input.**
q	output	**Data output.** The value present on the data input is transferred to the q output upon the rising edge of the clock if the asynchronous reset input is high.

Parameters

Table 2-47: *Parameters*

Parameter	Defined Values	Default Value
init	1'b0, 1'b1	1'b0
sr_assertion	"unclocked", "clocked"	"unclocked"

init

The init parameter defines the initial value of the output of the DFFR register. This is the value the register takes upon the initial application of power to the FPGA. The default value of the init parameter is 1'b0.

sr_assertion

The sr_assertion parameter defines the behavior of the output when the sn set input is asserted. Assigning the sr_assertion to "unclocked" results in an asychronous assertion of the reset signal, where the q output is set to one upon assertion of the active-low reset signal. Assigning the sr_assertion to "clocked" results in a synchronous assertion of the reset signal, where the q output is set to one at the next rising edge of the clock. The default value of the sr_assertion parameter is "unclocked".

Table 2-48: *Function Table when sr_assertion = "unclocked'*

Inputs			Output
rn	**d**	**ck**	**q**
0	X	X	0
1	X	X	Hold
1	0	↑	0
1	1	↑	1

Table 2-49: *Function Table when sr_assertion = "clocked'*

Inputs			Output
rn	**d**	**ck**	**q**
0	X	↑	0
1	X	X	Hold
1	0	↑	0
1	1	↑	1

Verilog Instantiation Template

```
DFFR #(.init(1'b0))
  instance_name
   (.q(user_out),
    .d(user_din),
    .rn(user_reset),
    .ck(user_clock));
```

VHDL Instantiation Template

```
------------- ACHRONIX LIBRARY ------------
library speedster22i;
use speedster22i.components.all;
------------- DONE ACHRONIX LIBRARY ---------

-- Component Instantiation
DFFR_instance_name : DFFR
    generic map (
      init => '0')
  port map (q => user_out,
            d => user_din,
            rn => user_reset,
            ck => user_clock);
```

DFFS

Positive Clock Edge D-Type Register with Asynchronous Set

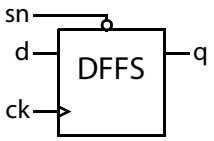


Figure 2-15: *Logic Symbol*

DFFS is a single D-type register with data input (d), clock (ck), and active-low set (sn) inputs and data (q) output. The active-low set input overrides all other inputs when it is asserted low and sets the data output high. If the asynchronous set input is not asserted, the data output is set to the value on the data input upon the next rising edge of the clock.

Pins

Table 2-50: *Pin Descriptions*

Name	Type	Description
d	input	**Data input.**
sn	input	**Active-low asynchronous set input.** A low on sn sets the q output high independent of the other inputs.
ck	input	**Positive-edge clock input.**
q	output	**Data output.** The value present on the data input is transferred to the q output upon the rising edge of the clock if the asynchronous set input is high.

Parameters

Table 2-51: *Parameters*

Parameter	Defined Values	Default Value
init	1'b0, 1'b1	1'b1
sr_assertion	"unclocked", "clocked"	"unclocked"

init

The init parameter defines the initial value of the output of the DFFS register. This is the value the register takes upon the initial application of power to the FPGA. The default value of the init parameter is 1'b1.

sr_assertion

The sr_assertion parameter defines the behavior of the output when the sn set input is asserted. Assigning the sr_assertion to "unclocked" results in an asychronous assertion of the reset signal, where the q output is set to one upon assertion of the active-low reset signal. Assigning the sr_assertion to "clocked" results in a synchronous assertion of the reset signal, where the q output is set to one at the next rising edge of the clock. The default value of the sr_assertion parameter is "unclocked".

Table 2-52: *Function Table when sr_assertion = "unclocked'*

Inputs			Output
sn	**d**	**ck**	**q**
0	X	↑	1
1	X	X	Hold
1	0	↑	0
1	1	↑	1

Table 2-53: *Function Table when sr_assertion = "clocked'*

Inputs			Output
sn	**d**	**ck**	**q**
0	X	X	1
1	X	X	Hold
1	0	↑	0
1	1	↑	1

Verilog Instantiation Template

```
DFFS #(.init(1'b1))
  instance_name
   (.q(user_out),
    .d(user_din),
    .sn(user_set),
    .ck(user_clock));
```

VHDL Instantiation Template

```
------------- ACHRONIX LIBRARY ------------
library speedster22i;
use speedster22i.components.all;
------------- DONE ACHRONIX LIBRARY ---------

-- Component Instantiation
DFFS_instance_name : DFFS
    generic map (
      init => '1')
  port map (q => user_out,
            d => user_din,
            sn => user_set,
            ck => user_clock);
```

Chapter 3 – Logic Functions

MUX2

Two Input Multiplexer Gate

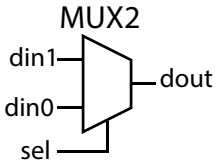


Figure 3-1: *Logic Symbol*

MUX2 implements a two-input multiplexer gate that has each of the two data inputs (din0, din1) connected directly to the outputs of a pair of LUT4 blocks in an RLB. The MUX2, when combined with two LUT4s, can be used as either a LUT5, a 4:1 multiplexer, or a function of up to nine inputs.

Pins

Table 3-1: *Pin Descriptions*

Name	Type	Description
din0, din1	input	**Data inputs.**
sel	input	**Data Selectinput.**
dout	output	**Data Output.**

Table 3-2: *Function Table*

sel	din0	din1	dout
0	0	X	0
0	1	X	1
1	X	0	0
1	X	1	1
X	0	0	0
X	1	1	1

Verilog Instantiation Template

```
MUX2 instance_name (.dout(user_out),
                    .sel(user_sel),
                    .din0(user_din0),
                    .din1(user_din1));
```

VHDL Instantiation Template

```
------------- ACHRONIX LIBRARY ------------
library speedster22i;
use speedster22i.components.all;
------------- DONE ACHRONIX LIBRARY ---------

-- Component Instantiation
MUX2_instance_name : MUX2
  port map (dout => user_out,
            sel => user_sel,
            din0 => user_din0,
            din1 => user_din1);
```

Chapter 4 – Lookup Table (LUT) Functions

LUT4

Four Input Lookup Table

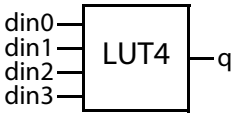


Figure 4-1: *Logic Symbol*

LUT4 implements a four-input lookup table with data inputs (din0 - din3) and data output (q), whose function is defined by the sixteen-bit lut_function parameter. The LUT4 is treated as a black-box by the synthesis tools. The user may instantiate this block to define specific structures of logic functions.

Pins

Table 4-1: *Pin Descriptions*

Name	Type	Description
din0 - din3	input	**Data inputs.**
q	output	**Data output.** The value on dout is the bit of the lut_function parameter indexed by the inputs {din3,din2,din1,din0}.

Parameters

Table 4-2: *Parameters*

Parameter	Defined Values	Default Value
lut_function	16-bit hexadecimal value	16'h0

lut_function

The lut_function parameter defines the value on the q output of the LUT4 as detailed in **Table 4-3: Function Table**. The default value of the lut_function parameter is 16'h0.

Table 4-3: *Function Table*

din3	din2	din1	din0	q
0	0	0	0	lut_function[0]
0	0	0	1	lut_function[1]
0	0	1	0	lut_function[2]
0	0	1	1	lut_function[3]
0	1	0	0	lut_function[4]
0	1	0	1	lut_function[5]
0	1	1	0	lut_function[6]
0	1	1	1	lut_function[7]
1	0	0	0	lut_function[8]
1	0	0	1	lut_function[9]
1	0	1	0	lut_function[10]
1	0	1	1	lut_function[11]
1	1	0	0	lut_function[12]
1	1	0	1	lut_function[13]
1	1	1	0	lut_function[14]
1	1	1	1	lut_function[15]

Verilog Instantiation Template

```
LUT4 #(.lut_function(16'h0123))
    instance_name(.dout(user_out),
                  .din0(user_in0),
                  .din1(user_in1),
                  .din2(user_in2),
                  .din3(user_in3));
```

VHDL Instantiation Template

```
------------ ACHRONIX LIBRARY ------------
library speedster22i;
use speedster22i.components.all;
------------ DONE ACHRONIX LIBRARY ---------

-- Component Instantiation
LUT4_instance_name : LUT4
  generic map (lut_function => X"0123")
  port map (dout => user_out,
            din0 => user_din0,
            din1 => user_din1,
            din2 => user_din2,
            din3 => user_din3);
```

Chapter 5 – Arithmetic Functions

ALU

Two Input Adder / Subtractor with Programmable Load

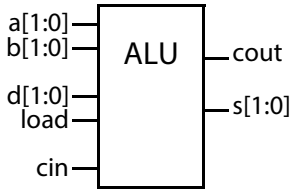


Figure 5-1: *Logic Symbol*

ALU implements either a two-bit adder or two-bit subtractor with adder/subtractor inputs (a[1:0], b[1:0]), load value (d[1:0]), Load Enable (load) and carry-in (cin) inputs. It generates the sum/difference (s[1:0]) and carry-out (cout) outputs. Asserting the load signal high overrides the s[1:0] output with the value of the Load Value d[1:0] input. Multiple ALU blocks may be combined by connecting the cout output of one slice to the cin input of the next significant two-bit slice. Selection of whether the ALU is configured as an adder or subtractor is determined by the value of the invert_b parameter.

Pins

Table 5-1: *Pin Descriptions*

Name	Type	Description
a[1:0]	input	**Data Input a.** Data Input a is a 2-bit two's complement signed input, where bit 1 is the most significant bit. In subtraction mode, Data Input a is the minuend.
b[1:0]		**Data Input b.** Data Inputs b is a 2-bit two's complement signed input, where bit 1 is the most significant bit. In subtraction mode, Data Input b is the subtrahend.
d[1:0]	input	**Load Value input.** Input d[1:0] is the value that is loaded onto the outputs s[1:0] upon the active-high assertion of the load input.
load	input	**Load input** (active-high). Asserting the load input high set the s[1:0] output equal to the d[1:0] input.
cin	input	**Carry In input**(active-high). The cin is the carry-in to the ALU. Note that for subtraction, cin should be tied high.
s[1:0]	output	**Sum/Difference output**. If the invert_b parameter is set to 1'b0, the s[1:0] output will reflect the sum of the a, b, and cin inputs if the load input is low. If the invert_b parameter is set to 1'b1, the s[1:0] output will reflect the difference of the a, b, cin inputs if the load input is low.
cout	output	**Carry-out output.** The cout is set high during an add when the s[1:0] output overflows.

Parameters

Table 5-2: *Parameters*

Parameter	Defined Values	Default Value
invert_b	1'b0, 1'b1	1'b0

invert_b

The invert_b parameter defines if the ALU functions as an adder or a subtractor. Setting the invert_b parameter to 1'b0 configures the ALU to perform two's complement addition of a[1:0] + b[1:0] + cin. Setting the invert_b parameter to 1'b1 configures the ALU to invert the b[1:0] input so that the two's complement subtraction of a[1:0] - b[1:0] is performed. Note that when subtractions is desired, the cin input must be tied high. When multiple ALUs are connected to perform higher resolution subtractors, only the cin of the LSB of the subtractor is to be tied high.

Table 5-3: *Function Table When invert_b = 1'b0.*

load	cin	s[1:0]	Note
1	X	d[1:0]	Load
0	-	a[1:0] + b[1:0] + cin	Add

Table 5-4: *Function Table When invert_b = 1'b1.*

load	cin	s[1:0]	Note
1	X	d[1:0]	Load
0	1	a[1:0] - b[1:0]	Subtract
0	0	a[1:0] - b[1:0] - 1	Subtract - 1

Verilog Instantiation Template

```
ALU #(.invert_b(1'b0))
    instance_name(.a(user_a),
                  .b(user_b),
                  .d(user_load_value),
                  .load(user_load),
                  .cin(user_carry_in),
                  .s(user_sum),
                  .cout(user_cout));
```

VHDL Instantiation Template

```
------------- ACHRONIX LIBRARY ------------
library speedster22i;
use speedster22i.components.all;
------------- DONE ACHRONIX LIBRARY ---------

-- Component Instantiation
LUT4_instance_name : LUT4
  generic map (invert_b => '0')
  port map (s => user_sum,
            cout => user_carry_out,
            a => user_a,
            b => user_b,
            d => user_d,
            load => user_load,
            cin => user_carry_in);
```

Chapter 6 – Memories

BRAM80K

80k-bit Dual-Port Memory

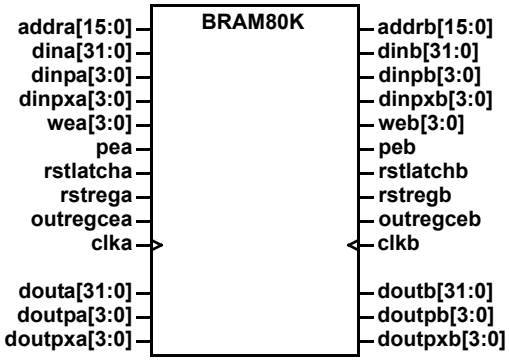
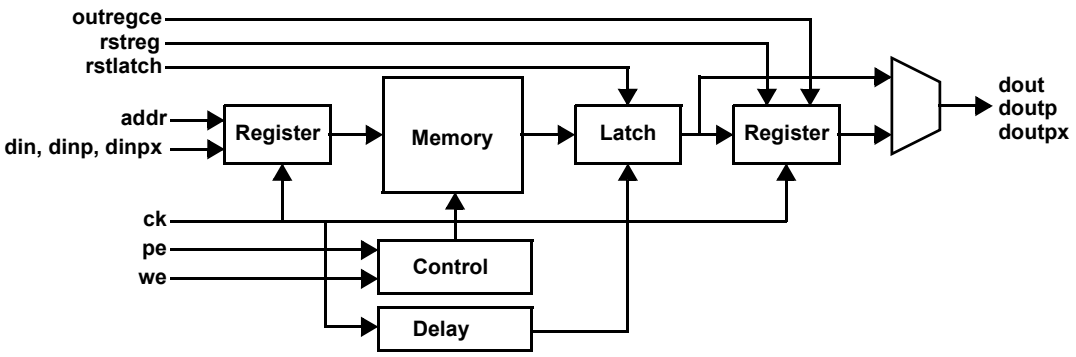


Figure 6-1: *Logic Symbol*

The block RAM (BRAM80K) implements a 80k-bit dual-ported memory block where each port can be independently configured with respect to size and function. The BRAM80K can be configured as a single-port (1 r/w port), dual-port (two r/w ports with independent clocks), or ROM memory. Each memory can be configured as a 2kx40, 2kx36, 2kx32, 4kx20, 4kx18, 4kx16, 8kx10, 8kx9, 8kx8, 16kx5, 16kx4, 32kx2, or 64kx1. The read and write operations are both synchronous. For higher performance operation, an addition output register can be enabled. Enabling the output register will require an additional cycle of read latency. Write Enable (wea/web) controls provide 10-bit enable control for port widths of 10, 20 or 40 bit. The initial value of the memory contents may be specified by the user from either parameters or a memory initialization file. The initial/reset values of the output registers may also be specified by the user. The porta_write_mode/portb_write_mode parameters define the behavior of the output data port during a write operation. When porta_write_mode/portb_write_mode is set to write_first, the douta/doutb is set to the value being written on the dina/dinb port during a write operation. Setting porta_write_mode/portb_write_mode to no_change keeps the douta/doutb port unchanged during a write operation to porta/portb. Conflict arises when the same memory cell is accessed by both ports within a narrow window and one or both ports are writing to memory. If this condition occurs, the contents of the memory for the colliding address will be undefined, but no damage will occur to the Speedster FPGA:

Figure 6-2: *BRAM80K Block Diagram (Per Port)*



BRAM80K Pins

Table 6-1: *BRAM80K Pin Descriptions*

Name	Type	Description
dina[31:0], dinb[31:0]	input	**Port A(B) data input.**
dinpa[3:0], dinpb[3:0]	input	**Port A(B) parity input (may be used for data).**
dinpxa[3:0], dinpxb[3:0]	input	**Port A(B) extended parity input (may be used for data).**
addra[15:0], addrb[15:0]	input	**Port A(B) read/write address input. Note that the addra, addrb inputs are top justified. See Table 6-11: BRAM80K Address Bus Mapping (Per Port)**
wea[3:0], web[3:0]	input	**Port A(B) byte-wide (10-bit) write enable.** Each bit of a port's write enable input enables a 10-bit byte to be written to the memory block as detailed below. Byte transactions are enabled on port A(B) when both the corresponding write enable input wea(web) is high and the port enable pea(peb) signal is active. For port widths of ten bits or less, all four write enables must be tied together. For port widths of 16, 18, or 20 bits, user_we[0] must be connected to both wea[0](web[0]) and wea[2](web[2]) and user_we[1] must be connected to both wea[1](web[1]) and wea[3](web[3]). If the write enable signal is inactive while the port enable line is active, the output is updated with the contents of the addressed memory cells regardless of the value of the porta_write_mode(portb_write_mode) parameter.
pea, peb	input	**Port A(B) enable** (programmable, default active-high = 1'b1). The pea(peb) signal must be asserted to enable a read, write, or reset (via rstlatcha(rstlatchb)) operation on port A(B).
rstlatcha, rstlatchb	input	**Port A(B) output latch synchronous set/reset** (programmable, default active-high = 1'b1). When rstlatcha(rstlatchab) is asserted, the value of the porta_srval(portb_srval) parameter is written to the port a(b) output latch upon the next active edge of clka(clkb).
rstrega, rstregb,	input	**Port A(B) output register synchronous set/reset** (programmable, default active-high = 1'b1). When rstrega(rstregb) is asserted, the value of the porta_srval(portb_srval) parameter is written to the port a(b) output register upon the next active edge of clka(clkb). The priority of rstrega(rstregb) relative to the clock enable input outregcea(outregceb) is determined by the value of the porta_regce_priority (portb_regce_priority parameter).
outregcea, outregceb	input	**Port A(B) output register clock enable** (active-high).
clka, clkb	input	**Port A(B) clock input.** Read and write operations are fully synchronous and occur upon the active edge of the clka(clkb) clock input when the pea(peb) signal is both high. The active edge of clka(clkb) is determined by the value of the value of the porta_clock_polarity (portb_clock_polarity) parameter.

Name	Type	Description
douta[31:0], doutb[31:0]	output	**Port A(B) data output.** For read operations, the douta(doutb) outputs are updated with the memory contents addressed by addra(addrb) if the pea(peb) port enable is active and wea(web) inputs are low. For write operations, if the porta_write_mode(portb_write_mode) parameter is set to write_first, the douta(doutb) outputs are updated with the value present on dina(dinb) if the pea(peb) port enable is active and the corresponding write enable input wea(web) is high.
doutpa[3:0], doutpb[3:0]	output	**Port A(B) parity output.** The port A(B) doutpa(doutpb) parity output behaves in the same manner as outputs douta(doutb) and is used when the porta_read_width(portb_read_width) is set to 10, 20, or 40 bits.
doutpxa[3:0], doutpxb[3:0]	output	**Port A(B) extended parity output.** The port A(B) doutpxa(doutpxb) extended parity output behaves in the same manner as outputs douta(doutb) and is used when the porta_read_width(portb_read_width) is set to 10, 20, or 40 bits.

Parameters

Table 6-2: *BRAM80K Parameters*

Parameter	Defined Values	Default Value
porta_read_width, portb_read_width	1, 2, 4, 5, 8, 9, 10, 16, 18, 20, 32, 36, 40	40
porta_write_width, portb_write_width	1, 2, 4, 5, 8, 9, 10, 16, 18, 20, 32, 36, 40	40
porta_write_mode, portb_write_mode	“write_first”, ”no_change”	“write_first”
porta_clock_polarity, portb_clock_polarity	“rise”, “fall”	“rise“
porta_peval, portb_peval	1’b0,1’b1	1’b1
porta_latch_rstval, portb_latch_rstval	1’b0,1’b1	1’b1
porta_en_out_reg, portb_en_out_reg	1’b0,1’b1	1’b0
porta_reg_rstval, portb_reg_rstval	1’b0,1’b1	1’b1
porta_regce_priority, portb_regce_priority	“rstreg”, ”regce”	“rstreg”
porta_initval, portb_initval	40-bit hexadecimal number	40’h0
porta_srval, portb_srval	40-bit hexadecimal number	40’h0
mem_init_file	<path to HEX file>	“”
initd_000 – initd_255	256 bit hexadecimal number	256’hx
initp_00 – initp_31	256 bit hexadecimal number	256’hx
initpx_00 – initpx_31	256 bit hexadecimal number	256’hx

porta_read_width(portb_read_width)

The porta_read_width(portb_read_width) parameter sets the read width for Port A(B). The read width may vary from the write port width, but it must be within the allowable combinations defined in the **Memory Organization and Data Input / Output Pin Assignments** section.

porta_write_width(portb_write_width)

The porta_write_width(porta_write_width) parameter sets the write width for Port A(B). The read width may vary from the write port width, but it must be within the allowable combinations defined in the **Memory Organization and Data Input / Output Pin Assignments** section.

porta_write_mode(portb_write_mode)

The porta_write_mode(portb_write_mode) parameter is used to define the response of the Port A(B) output to write operations. If porta_write_mode(portb_write_mode) is set to "no_change", douta(doutb) will remain unchanged during write operations. If porta_write_mode(portb_write_mode) is set to "write_first", the data present on the dina(dinb) input during the write operation will appear on the output of Port A(B) if the appropriate write enable bit, wea(web), is high. Note that the BRAM80K does not support a 'read-first' or 'read-before-write' mode. If this behavior is detected by synthesis, a warning will be issued in the synthesis log file and a register file will be synthesized. To implement a more efficient mapping of a 'read-first' memory, the user should update his code to use an Achronix BRAM80K_READ_FIRST soft macro. Please refer to **Support for Read-First (Read-Before-Write) Memory Operations** for a further explanation of read-first memory support.

porta_clock_polarity(portb_clock_polarity)

The porta_clock_polarity(portb_clock_polarity) parameter is used to set the active edge of the Port A(B) clock. A value of "rise" corresponds to an active rising edge assignment while "fall" corresponds to an active falling edge assignment. The default value of the porta_clock_polarity(portb_clock_polarity) is "rise".

porta_peval(portb_peval)

The porta_peval(portb_peval) parameter defines the active level of the Port A(B) pea(peb) port enable input. Assigning a value of 1'b0 to porta_peval(portb_peval) configures the Port A(B) pea(peb) port enable input to be active low, while an assignment to 1'b1 sets an active-high level. The default value of the porta_peval(portb_peval) parameter is 1'b1.

porta_latch_rstval(portb_latch_rstval)

The porta_latch_rstval(portb_latch_rstval) parameter defines the active level of the Port A(B) output latch reset input. Assigning a value of 1'b0 to porta_latch_rstval(portb_latch_rstval) configures the Port A(B) output latch to have an active-low synchronous reset, while assigning a value of 1'b1 configures the Port A(B) output latch to have an active-high synchronous reset. The default value of porta_latch_rstval(portb_latch_rstval) is 1'b1.

porta_en_out_reg(portb_en_out_reg)

The porta_en_out_reg(portb_en_out_reg) parameter determines whether the Port A(B) output register is enabled. A value of 1'b0 disables the output register and results in a read latency of one cycle, while a value of 1'b1 enables the output register and results in a read latency of two cycles. The default value of the porta_en_out_reg(portb_en_out_reg) parameter is 1'b0.

porta_reg_rstval(portb_reg_rstval)

The porta_reg_rstval(portb_reg_rstval) parameter defines the active level of the Port A(B) output register reset input. Assigning a value of 1'b0 to porta_reg_rstval(portb_reg_rstval) configures the Port A(B) output register to have an active-low synchronous reset, while assigning a value of 1'b1 configures the Port A(B) output register to have an active-high synchronous reset. The default value of the porta_reg_rstval(portb_reg_rstval) parameter is 1'b1.

porta_regce_priority(portb_regce_priority)

The porta_regce_priority(portb_regce_priority) parameter defines the priority of the outregcea(outregceb) clock enable input relative to the rstrega(rstregb) reset input during an assertion of the rstrega(rstregb) signal on the output register of Port A(B). Setting porta_regce_priority(portb_regce_priority) to "rstreg" allows the Port A(B) output register to be set/reset at the next active edge of the Port A(B) clock without requiring a specific value on the outregcea (outregceb) output register clock enable input. Setting porta_regce_priority (portb_regce_priority) to "regce" requires that the outregcea(outregceb) output register clock enable input is high for the output register set/reset operation to occur at the next active edge of the Port A(B) clock.

porta_initval(portb_initval)

The porta_initval(portb_initval) parameter defines the power-up default value of the data on the output of Port A(B) latch(register if porta_en_out_reg(portb_en_out_reg)=1'b1). The 40-bit porta_initval(portb_initval) parameter assignment is dependent on the porta_read_width (portb_read_width). The association of the of the porta_initval(portb_initval) parameter values to the douta,doutpa,doutpxa (doutb,doutpb,doutpxb) bits is assigned according to **Table 6-3: Relationship of porta_initval(portb_initval) bit positions to douta,doutpa,doutpxa (doutb,doutpb,doutpxb)**. The default value of porta_initval(portb_initval) is 40'h0.

Table 6-3: *Relationship of porta_initval(portb_initval) bit positions to douta,doutpa,doutpxa (doutb,doutpb,doutpxb)*

porta_read_width (portb_read_width)	doutpxa (doutpxb) porta_initval[39:36] (portb_initval[39:36])	doutpa (doutpb) porta_initval[35:32] (portb_initval[35:32])	douta (doutb) porta_initval[31:0] (portb_initval[31:0])
40	porta_initval[39:36]	porta_initval[35:32]	porta_initval[31:0]
36	4'hx	porta_initval[35:32]	porta_initval[31:0]
32	4'hx	4'hx	porta_initval[31:0]
20	2'bxx,porta_initval[37:36]	2'bxx,porta_initval[33:32]	16'hxxxx,porta_initval[15:0]
18	4'hx	2'bxx,porta_initval[33:32]	16'hxxxx,porta_initval[15:0]
16	4'hx	4'hx	16'hxxxx,porta_initval[15:0]
10	3'bxxx,porta_initval[36]	3'bxxx,porta_initval[32]	24'hxxxxxx,porta_initval[7:0]
9	4'hx	3'bxxx,porta_initval[32]	24'hxxxxxx,porta_initval[7:0]
8	4'hx	4'hx	24'hxxxxxx,porta_initval[7:0]
5	4'hx	3'bxxx,porta_initval[32]	28'hxxxxxxx,porta_initval[3:0]
4	4'hx	4'hx	28'hxxxxxxx,porta_initval[3:0]
2	4'hx	4'hx	30'hxxxxxxxx,porta_initval[1:0]
1	4'hx	4'hx	31'hxxxxxxxx,porta_initval[0]

porta_srval(portb_srval)

The porta_srval(portb_srval) parameter defines the value assigned to the Port A(B) output latch(register if porta_en_out_reg(portb_en_out_reg)=1) at the next active edge of the clock when the Port A(B) latch(register) reset conditions are met. The 40-bit porta_srval(portb_srval) parameter assignment is dependent on the porta_read_width(portb_read_width). The association of the of the porta_srval(portb_srval) parameter values to the douta,doutpa,doutpxa (doutb,doutpb,doutpxb) bits is assigned according to **Table 6-4: Relationship of porta_srval(portb_srval) bit positions to douta,doutpa,doutpxa (doutb,doutpb,doutpxb)**The default value of porta_srval (portb_srval) is 40'h0.

Table 6-4: *Relationship of porta_srval(portb_srval) bit positions to douta,doutpa,doutpxa (doutb,doutpb,doutpxb)*

porta_read_width (portb_read_width)	doutpxa (doutpxb) porta_srval[39:36] (portb_srval[39:36])	doutpa (doutpb) porta_srval[35:32] (portb_srval[35:32])	douta (doutb) porta_srval[15:0] (portb_srval[15:0])
40	porta_srval[39:36]	porta_srval[35:32]	porta_srval[31:0]
36	4'hx	porta_srval[35:32]	porta_srval[31:0]
32	4'hx	4'hx	porta_srval[31:0]
20	2'bxx,porta_srval[37:36]	2'bxx,porta_srval[33:32]	16'hxxxx,porta_srval[15:0]
18	4'hx	2'bxx,porta_srval[33:32]	16'hxxxx,porta_srval[15:0]
16	4'hx	4'hx	16'hxxxx,porta_srval[15:0]
10	3'bxxx,porta_srval[36]	3'bxxx,porta_srval[32]	24'hxxxxxx,porta_srval[7:0]
9	4'hx	3'bxxx,porta_srval[32]	24'hxxxxxx,porta_srval[7:0]
8	4'hx	4'hx	24'hxxxxxx,porta_srval[7:0]
5	4'hx	3'bxxx,porta_srval[32]	28'hxxxxxxx,porta_srval[3:0]
4	4'hx	4'hx	28'hxxxxxxx,porta_srval[3:0]
2	4'hx	4'hx	30'hxxxxxxxx,porta_srval[1:0]
1	4'hx	4'hx	31'hxxxxxxxx,porta_srval[0]

mem_init_file

The mem_init_file parameter provides a mechanism to set the initial contents of the BRAM80K memory. If the mem_init_file parameter is defined, the BRAM80K will be initialized with the values defined in the file pointed to by the mem_init_file parameter according to the format defined in the **Memory Initialization** section. If the mem_init_file is left at the default value of "", the initial contents will be defined by the values of the initd_000 - initd_255, initp_00 - initp_31, and the initpx_00 - initpx_31 parameters. If the memory initialization parameters and the mem_init_file parameters are not defined, the contents of the BRAM80K will not be initialized.

initd_000 – initd_255

The initd_000 through initd_255 parameters define the initial contents of the memory contents associated with douta[31:0] and doutb[31:0]. Each 256-bit parameter associated with the BRAM80K memory as defined in the **Memory Initialization** section.

initp_00 – initp_31

The initp_00 through initp_31 parameters define the initial contents of the memory contents associated with doutpa[3:0] and doutpb[3:0]. Each 256-bit parameter associated with the BRAM80K memory is defined in the **Memory Initialization** section.

initpx_00 – initpx_31

The initpx_00 through initpx_31 parameters define the initial contents of the memory contents associated with doutpxa[3:0] and doutpxb[3:0]. Each 256-bit parameter associated with the BRAM80K memory is defined in the **Memory Initialization** section.

Memory Organization and Data Input / Output Pin Assignments

The BRAM80K memory block supports memory widths from one to forty bits wide. The width of the dina(dinb) data input is determined by the porta_write_width(portb_write_width) parameter while the width of the douta(doutb) data output is determined by the porta_read_width(portb_read_width) parameter. The width of read port may be set to be different from the width of the write port on a per-port basis. Additionally, the width of the Port A read/write may be set to be different from the Port B read/write widths. There are, however, some limitations of the port width assignments between the read and write width assignments as well as the Port A and Port B width assignments. The valid port widths defined in **Table 6-2: BRAM80K Parameters** may be divided into three groups:

- Group1 (n x 5 widths): 40, 20, 10, 5
- Group2 (n x 9 widths): 36, 18, 9
- Group3 (n x 2 widths): 32, 16, 8, 4, 2, 1

Table 6-5: *Valid Port A Width Versus Port B Width Combinations per port for n x 5 width modes*

Port A Width	Port B Width				
	2kx40	4kx20	8kx10	16kx5	Other
40	✓	✓	✓	✓	–
20	✓	✓	✓	✓	–
10	✓	✓	✓	✓	–
5	✓	✓	✓	✓	–

Table 6-6: *Valid Port A Width Versus Port B Width Combinations per port for n x 9 width modes*

Port A Width	Port B Width			
	2kx36	4kx18	8kx9	Other
36	✓	✓	✓	–
18	✓	✓	✓	–
9	✓	✓	✓	–

Table 6-7: *Valid Port A Width Versus Write Port B Combinations per port for n x 2 width modes*

Port A Width	Port B Width						
	2kx32	4kx16	8kx8	16kx4	32kx2	64kx1	Other
32	✓	✓	✓	✓	✓	✓	–
16	✓	✓	✓	✓	✓	✓	–
8	✓	✓	✓	✓	✓	✓	–
4	✓	✓	✓	✓	✓	✓	–
2	✓	✓	✓	✓	✓	✓	–
1	✓	✓	✓	✓	✓	✓	–

Table 6-8: *dina(dinb) bit assignments per porta_write_width(portb_write_width) values*

porta_write_width, portb_write_width	dinpxa[3:0], dinpxb[3:0]	dinpa[3:0], dinpb[3:0]	dina[31:0], dinb[31:0]
40	user_din[39], user_din[29], user_din[19], user_din[9]	user_din[38], user_din[28], user_din[18], user_din[8]	user_din[37:30], user_din[27:20], user_din[17:10], user_din[7:0]
36	4'hx	user_din[35], user_din[26], user_din[17], user_din[8]	user_din[34:27], user_din[25:18], user_din[16:9], user_din[7:0]
32	4'hx	4'hx	user_din[31:0]
20	2'bxx,user_din[19], user_din[9]	2'bxx,user_din[18], user_din[8]	16'hxxxx,user_din[17:10], user_din[7:0]
18	4'hx	2'bxx,user_din[17], user_din[8]	16'hxxxx,user_din[16,9], user_din[7:0]
16	4'hx	4'hx	16'hxxxx,user_din[15:0]
10	3'bxxx,user_din[9]	3'bxxx,user_din[8]	24'hxxxxxx,user_din[7:0]
9	4'hx	3'bxxx,user_din[8]	24'hxxxxxx,user_din[7:0]
8	4'hx	4'hx	24'hxxxxxx,user_din[7:0]
5	4'hx	3'bxxx,user_din[4]	28'hxxxxxxx,user_din[3:0]
4	4'hx	4'hx	28'hxxxxxxx,user_din[3:0]
2	4'hx	4'hx	30'hxxxxxxxx,user_din[1:0]
1	4'hx	4'hx	31'hxxxxxxxx,user_din[0]

Table 6-9: *Write Enable to Data Input Bus Mapping per Write Port Width*

porta_write_width (portb_write_width)	wea[3] (web[3])	wea[2] (web[2])	wea[1] (web[1])	wea[0] (web[0])
40	dinpx[3],dinp[3], din[31:24]	dinpx[2],dinp[2], din[23:16]	dinpx[1],dinp[1], din[15:8]	dinpx[0],dinp[0], din[7:0]
36	dinp[3], din[31:24]	dinp[2], din[23:16]	dinp[1], din[15:8]	dinp[0], din[7:0]
32	din[31:24]	din[23:16]	din[15:8]	din[7:0]
20	dinpx[1],dinp[1], din[15:8]	dinpx[0],dinp[0], din[7:0]	dinpx[1],dinp[1], din[15:8]	dinpx[0],dinp[0], din[7:0]
18	dinp[1], din[15:8]	dinp[0], din[7:0]	dinp[1], din[15:8]	dinp[0], din[7:0]
16	din[15:8]	din[7:0]	din[15:8]	din[7:0]
10	dinpx[0],dinp[0], din[7:0]	dinpx[0],dinp[0], din[7:0]	dinpx[0],dinp[0], din[7:0]	dinpx[0],dinp[0], din[7:0]
9	dinp[0], din[7:0]	dinp[0], din[7:0]	dinp[0], din[7:0]	dinp[0], din[7:0]
8	din[7:0]	din[7:0]	din[7:0]	din[7:0]
5	dinp[0], din[3:0]	dinp[0], din[3:0]	dinp[0], din[3:0]	dinp[0], din[3:0]
4	din[3:0]	din[3:0]	din[3:0]	din[3:0]
2	din[1:0]	din[1:0]	din[1:0]	din[1:0]
1	din[0]	din[0]	din[0]	din[0]

Table 6-10: *douta(doutb) bit assignments per porta_read_width(portb_read_width) values*

porta_read_width, portb_read_width	doutpxa[3:0], doutpxb[3:0]	doutpa[3:0], doutpb[3:0]	douta[31:0], doutb[31:0]
40	user_dout[39], user_dout[29], user_dout[19], user_dout[9]	user_dout[38], user_dout[28], user_dout[18], user_dout[8]	user_dout[37:30], user_dout[27:20], user_dout[17:10], user_dout[7:0]
36	4'hx	user_dout[35], user_dout[26], user_dout[17], user_dout[8]	user_dout[34:27], user_dout[25:18], user_dout[16:9], user_dout[7:0]
32	4'hx	4'hx	user_dout[31:0]
20	2'bxx,user_dout[19], user_dout[9]	2'bxx,user_dout[18], user_dout[8]	16'hxxxx,user_dout[17:10], user_dout[7:0]
18	4'hx	user_dout[17], user_dout[8]	16'hxxxx,user_dout[16:9], user_dout[7:0]
16	4'hx	4'hx	16'hxxxx,user_dout[15:0]
10	3'bxxx,user_dout[9]	3'bxxx,user_dout[8]	24'hxxxxxx,user_dout[7:0]
9	4'hx	3'bxxx,user_dout[8]	24'hxxxxxx,user_dout[7:0]
8	4'hx	4'hx	24'hxxxxxx,user_dout[7:0]
5	4'hx	3'bxxx,user_dout[4]	28'hxxxxxxx,user_dout[3:0]
4	4'hx	4'hx	28'hxxxxxxx,user_dout[3:0]
2	4'hx	4'hx	30'hxxxxxxxx,user_dout[1:0]
1	4'hx	4'hx	31'hxxxxxxxx,user_dout[0]

Table 6-11: *BRAM80K Address Bus Mapping (Per Port)*

Memory Organization	Used Address addra/addrb Pins	Unconnected Address Pins
2kx40	15:5	4:0
2kx36	15:5	4:0
2kx32	15:5	4:0
4kx20	15:4	3:0
4kx18	15:4	3:0
4kx16	15:4	3:0
8kx10	15:3	2:0
8kx9	15:3	2:0
8kx8	15:3	2:0
16kx5	15:2	1:0
16kx4	15:2	1:0
32kx2	15:1	0
64kx1	15:0	-

Table 6-12: *Mapping of Word Sizes to the Native 2048x40 Memory Locations*

Port Width	Extended Parity Bits	Parity Bits	Data Bits
40	0	0	0
36	n/a	0	0
32	n/a		0
20	1 0	1 0	1 0
18	n/a	1 0	1 0
16	n/a		1 0
10	3 2 1 0	3 2 1 0	3 2 1 0
9	n/a	3 2 1 0	3 2 1 0
8	n/a		3 2 1 0
5	7 6 5 4	3 2 1 0	7 6 5 4 3 2 1 0
4	n/a		7 6 5 4 3 2 1 0
2	n/a		15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1	n/a		31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Read and Write Operations

The BRAM80K has four basic modes of read and write operations:

- Write-First, Latched Mode
- Write-First, Registered Mode
- No-Change, Latched Mode
- No-Change, Registered Mode

One of these four modes is selected for Port A(Port B) via the porta_write_mode(portb_write_mode) and porta_en_out_reg(portb_en_out_reg) parameters.

Read Operation

There are two modes of operation supported for reading data from the BRAM80K: latched mode and register mode. In latched mode, the read address is registered and the stored data is latched into the output latches. In the registered mode, the read operation results in one additional cycle of latency.

Write Operation

The write operation is a single clock edge operation. The addra(addrb) write address is registered and the dina(dinb) data input is stored in the BRAM80K at the next active edge of the write clock. The porta_write_mode(portb_write_mode) parameter controls what is output on the douta(doutb) output during a write operation. If the porta_write_mode(portb_write_mode) parameter is set to “no_change”, the douta(doutb) will remain unchanged during a write operation on port A(B). If the porta_write_mode(portb_write_mode) parameter is set to “write_first”, the value of douta(doutb) output will be updated with the dina(dinb) data if the appropriate byte enable input is enabled.

Table 6-13: *BRAM Output Function Table for Latched Mode (Assumes active-high clock, port enable, and latch reset value)*

Operation	pea (peb)	wea (web)	rstlatcha (rstlatchb)	porta_write_mode (portb_write_mode)	clka (clkb)	douta (doutb)
Hold	X	X	X	X	X	douta_previous (doutb_previous)
Hold	0	X	X	X	↑	douta_previous (doutb_previous)
Reset Output	1	X	1	X	↑	porta_srval (portb_srval)
Read	1	0	0	X	↑	mem[addra] (mem[addrb])
Write	1	1	0	"write-first"	↑	dina (dinb)
Write	1	1	0	"no_change"	↑	douta_previous (doutb_previous)

Table 6-14: *BRAM Output Function Table for Registered Mode (Assumes active-high clock, output register clock enable, and output register reset)*

Operation	porta_regce_priority (portb_regce_priority)	rstrega (rstregb)	outregcea (outregceb)	clka (clkb)	douta (doutb)
Hold	X	X	X	X	douta_previous (doutb_previous)
Hold	"rstreg"	0	0	↑	douta_previous (doutb_previous)
Update Output	"rstreg"	0	1	↑	latcha_output (latchb_output)
Reset Output	"rstreg"	1	X	↑	porta_srval (portb_srval)
Hold	"regce"	X	0	↑	douta_previous (doutb_previous)
Update Output	"regce"	0	1	↑	latcha_output (latchb_output)
Reset Output	"regce"	1	1	↑	porta_srval (portb_srval)

Simultaneous Memory Operations

Memory operations may be performed simultaneously from both sides of the memory, however there is a restriction with memory collisions. A memory collision is defined as the condition where both of the ports access the same memory address within the same clock cycle (both ports connected to the same clock), or within a TBD ps window (if each port is connected to a different clock). Simultaneous read operations to the same address by both ports is allowed and will produce valid data on each of the ports. If one of the ports is writing an address while the other port is reading the same address, the write operation will occur, but the read data will be invalid. The user may reliably read the data the next cycle if there is no longer a write collision. If both ports write the same address at the same time, the memory contents for that memory address will become invalid. While simultaneously writing the same address from both ports will invalidate the data, no damage to the hardware will occur.

Note that for the special case of the BRAM80K having both ports configured for write_first mode, a write-write collision will corrupt the memory contents, but the user will see the

correct data at both output ports. In this case, the data corruption will not be noticed by the circuit until the the corrupted memory location is later read.

Timing Diagrams

The timing diagrams for the four combinations of the porta_write_mode(portb_write_mode) and porta_en_out_reg(portb_en_out_reg) parameters is shown below

Figure 6-3: *Write-First, Latched Mode Timing Diagram*

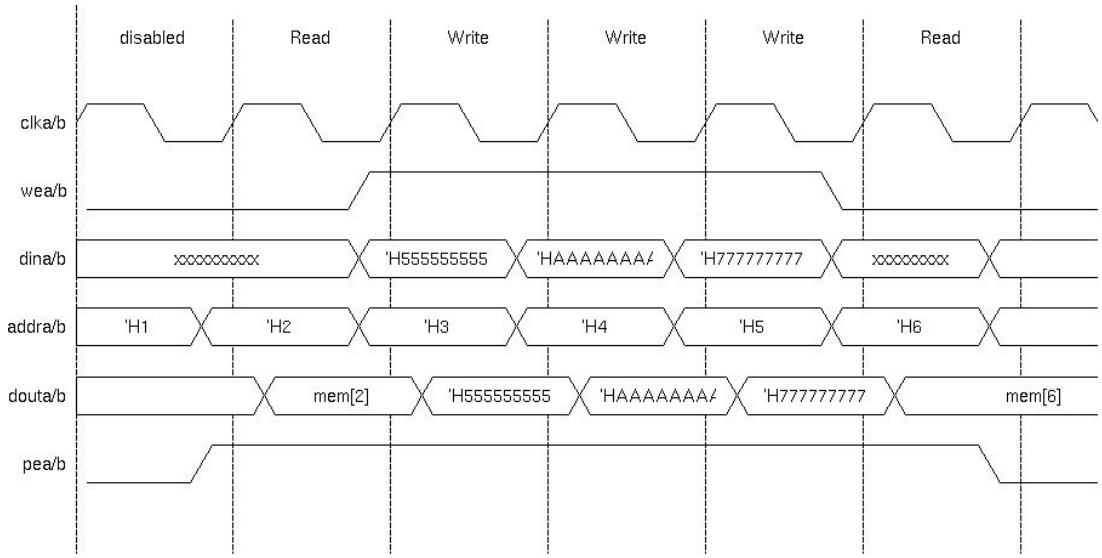


Figure 6-4: *Write-First, Registered Mode Timing Diagram*

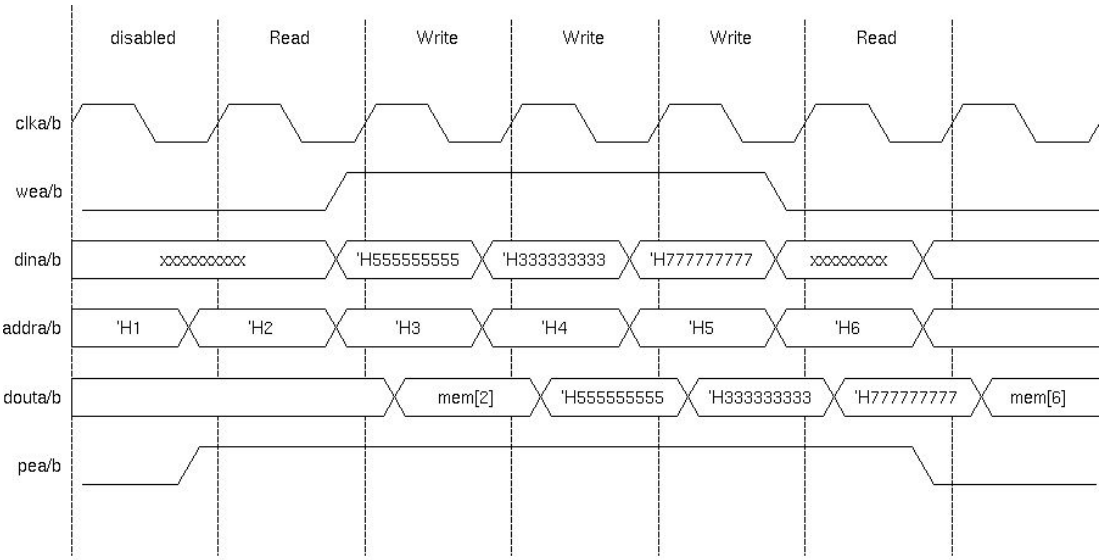


Figure 6-5: *No-Change, Latched Mode Timing Diagram*

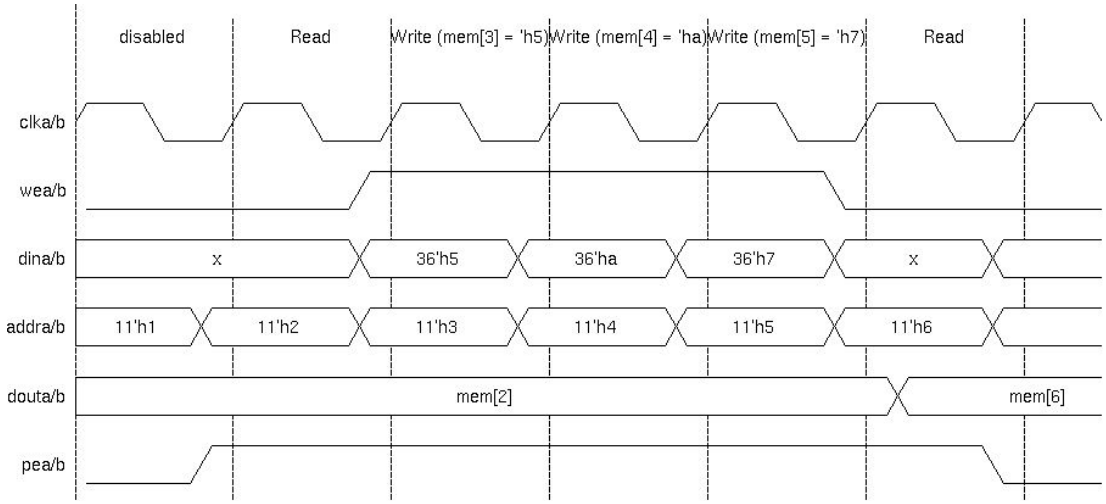
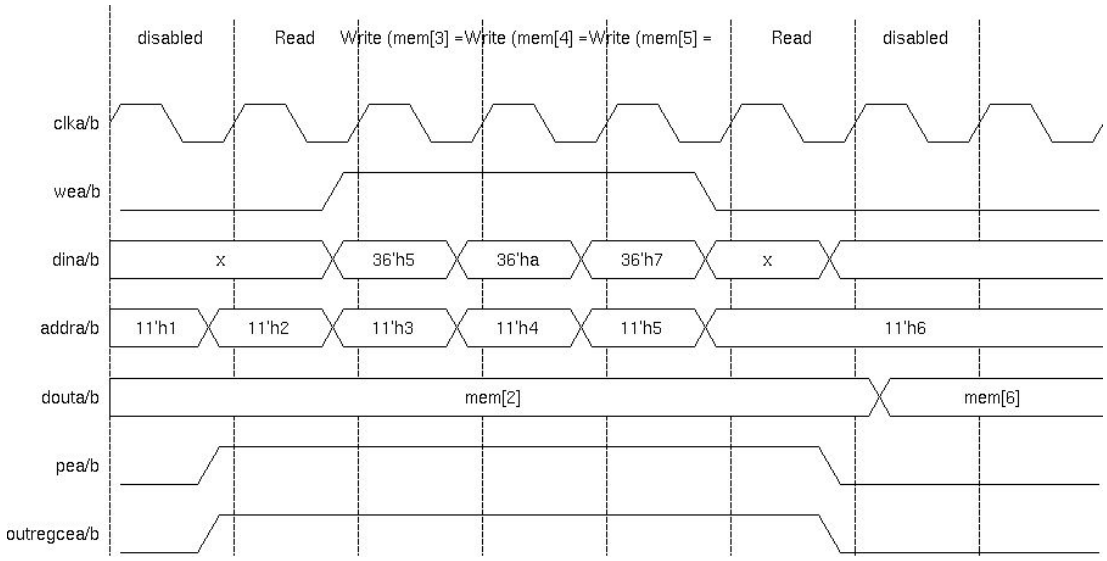


Figure 6-6: *No-Change, Registered Mode Timing Diagram*



Support for Read-First (Read-Before-Write) Memory Operations

The BRAM80K memory does not directly support read-first or read-before-write mode of operation. If this behavior is detected by synthesis, a warning will be issued in the synthesis log file and a register file will be synthesized. To implement a more efficient mapping of a 'read-first' memory, the user should update his code to use an Achronix BRAM80K_READ_FIRST soft macro. This soft macro block combines a BRAM80K memory block with LUT circuitry to convert the read-first memory access into a separate read operation followed by a write operation at twice the clock rate of the requested clock frequency. Note that the user will have to provide the BRAM80K_READ_FIRST macro with a 2x clock with one of the on-chip PLLs.

Memory Initialization

When the BRAM80K memory is configured with port widths of 1, 2, 4, 8, 16, or 32 bits wide, the initial memory contents may be defined by initializing the 256 256-bit parameters initd_000 through initd_255. The data memory is organized as little-endian with bit 0 mapped to bit zero of parameter initd_000 and bit 65535 mapped to bit 255 of parameter initd_255.

When the BRAM80K memory is configured with port widths of 9, 18, or 36 bits wide, the initial memory contents may be defined by initializing the 256 256-bit parameters initd_000 through initd_255 and the 32 256-bit parameters initp_00 through initp_31. Each nine-bit byte is configured by programming the bottom eight bits of the byte to consecutive bytes of the initd_000 through initd_255 parameters and programming the ninth bit to consecutive bits of the initp_00 through initp_32 parameters. The parity memory is also organized as little-endian with the first parity bit location mapped to bit 0 of initp_00 and the last parity bit mapped to the bit 255 of initp_31.

When the BRAM80K memory is configured with port widths of 10, 20, or 40 bits wide, the initial memory contents may be defined by initializing the 256 256-bit parameters initd_000 through initd_255, the 32 256-bit parameters initp_00 through initp_31, and the 32 256-bit parameters initpx_00 through initpx_31. Each ten-bit byte is configured by programming the bottom eight bits of the byte to consecutive bytes of the initd_000 through initd_255 parameters, programming the ninth bit to consecutive bits of the initp_00 through initp_31 parameters and programming the tenth bit to consecutive bits of the initpx_00 through initpx_31 parameters. The extended parity memory is also organized as little-endian with the first parity bit (tenth bit) location mapped to bit 0 of initpx_00 and the last parity bit mapped to the bit 255 of initpx_31.

The BRAM80K memory block may alternatively be initialized with a memory file by setting the mem_init_file to point to the path of a memory initialization file. The file format in the latter case is defined by hexadecimal entries separated by white space, where the white space is defined by spaces or line separation. Each number is hexadecimal of width equal to the maximum value amongst the parameters: porta_read_width, porta_write_width, portb_read_width, portb_write_width. A number entry may contain underscore (_) characters amongst the digits, for example, A234_4567_33. Commenting is allowed following a double-slash (//) through to the end of the line. C-like commenting is also allowed where the characters between the /* and */ are ignored. The memory is initialized starting with the first entry of the file initializing the memory array starting with address zero, moving upward.

If the mem_init_value is defined, the BRAM80K will be initialized with the values defined in the file pointed to by the mem_init_file parameter according to the format defined above. If the mem_init_file is left at the default value of “”, the initial contents will be defined by the values of the initd_000 - initd_255, initp_00 - initp_31, and the initpx_00 - initpx_31 parameters. If the memory initialization parameters and the mem_init_file parameters are not defined, the contents of the BRAM80K will not be initialized and the contents will be unknown until the memory locations are written.

BRAM80K Verilog Instantiation Template

```
BRAM80K #(
   .porta_read_width(40),
   .porta_write_width(40),
   .porta_write_mode("write_first"),
   .porta_clock_polarity("rise"),
   .porta_en_out_reg(1'b0),
   .porta_regce_priority("rstreg"),
   .porta_peval(1'b1),
   .porta_reg_rstval(1'b1),
   .porta_latch_rstval(1'b1),
   .porta_initval(40'h0),
   .porta_srval(40'h0),
   .portb_read_width(40),
   .portb_write_width(40),
   .portb_write_mode("write_first"),
   .portb_clock_polarity("rise"),
   .portb_en_out_reg(1'b0),
   .portb_regce_priority("rstreg"),
   .portb_peval(1'b1),
   .portb_reg_rstval(1'b1),
   .portb_latch_rstval(1'b1),
   .portb_initval(40'h0),
   .portb_srval(40'h0),
   .mem_init_file(""),

   .initd_000(256'h0),
   .initd_001(256'h0),
   .initd_002(256'h0),
   .initd_003(256'h0),
   .initd_004(256'h0),
   .initd_005(256'h0),
   .initd_006(256'h0),
   .initd_007(256'h0),
   .initd_008(256'h0),
   .initd_009(256'h0),
   .initd_010(256'h0),
   .initd_011(256'h0),
   .initd_012(256'h0),
   .initd_013(256'h0),
   .initd_014(256'h0),
   .initd_015(256'h0),
   .initd_016(256'h0),
   .initd_017(256'h0),
   .initd_018(256'h0),
```

```
.initd_019(256'h0),
.initd_020(256'h0),
.initd_021(256'h0),
.initd_022(256'h0),
.initd_023(256'h0),
.initd_024(256'h0),
.initd_025(256'h0),
.initd_026(256'h0),
.initd_027(256'h0),
.initd_028(256'h0),
.initd_029(256'h0),
.initd_030(256'h0),
.initd_031(256'h0),
.initd_032(256'h0),
.initd_033(256'h0),
.initd_034(256'h0),
.initd_035(256'h0),
.initd_036(256'h0),
.initd_037(256'h0),
.initd_038(256'h0),
.initd_039(256'h0),
.initd_040(256'h0),
.initd_041(256'h0),
.initd_042(256'h0),
.initd_043(256'h0),
.initd_044(256'h0),
.initd_045(256'h0),
.initd_046(256'h0),
.initd_047(256'h0),
.initd_048(256'h0),
.initd_049(256'h0),
.initd_050(256'h0),
.initd_051(256'h0),
.initd_052(256'h0),
.initd_053(256'h0),
.initd_054(256'h0),
.initd_055(256'h0),
.initd_056(256'h0),
.initd_057(256'h0),
.initd_058(256'h0),
.initd_059(256'h0),
.initd_060(256'h0),
.initd_061(256'h0),
.initd_062(256'h0),
.initd_063(256'h0),
.initd_064(256'h0),
```

```
.initd_065(256'h0),
.initd_066(256'h0),
.initd_067(256'h0),
.initd_068(256'h0),
.initd_069(256'h0),
.initd_070(256'h0),
.initd_071(256'h0),
.initd_072(256'h0),
.initd_073(256'h0),
.initd_074(256'h0),
.initd_075(256'h0),
.initd_076(256'h0),
.initd_077(256'h0),
.initd_078(256'h0),
.initd_079(256'h0),
.initd_080(256'h0),
.initd_081(256'h0),
.initd_082(256'h0),
.initd_083(256'h0),
.initd_084(256'h0),
.initd_085(256'h0),
.initd_086(256'h0),
.initd_087(256'h0),
.initd_088(256'h0),
.initd_089(256'h0),
.initd_090(256'h0),
.initd_091(256'h0),
.initd_092(256'h0),
.initd_093(256'h0),
.initd_094(256'h0),
.initd_095(256'h0),
.initd_096(256'h0),
.initd_097(256'h0),
.initd_098(256'h0),
.initd_099(256'h0),
.initd_100(256'h0),
.initd_101(256'h0),
.initd_102(256'h0),
.initd_103(256'h0),
.initd_104(256'h0),
.initd_105(256'h0),
.initd_106(256'h0),
.initd_107(256'h0),
.initd_108(256'h0),
.initd_109(256'h0),
.initd_110(256'h0),
```

```
.initd_111(256'h0),
.initd_112(256'h0),
.initd_113(256'h0),
.initd_114(256'h0),
.initd_115(256'h0),
.initd_116(256'h0),
.initd_117(256'h0),
.initd_118(256'h0),
.initd_119(256'h0),
.initd_120(256'h0),
.initd_121(256'h0),
.initd_122(256'h0),
.initd_123(256'h0),
.initd_124(256'h0),
.initd_125(256'h0),
.initd_126(256'h0),
.initd_127(256'h0),
.initd_128(256'h0),
.initd_129(256'h0),
.initd_130(256'h0),
.initd_131(256'h0),
.initd_132(256'h0),
.initd_133(256'h0),
.initd_134(256'h0),
.initd_135(256'h0),
.initd_136(256'h0),
.initd_137(256'h0),
.initd_138(256'h0),
.initd_139(256'h0),
.initd_140(256'h0),
.initd_141(256'h0),
.initd_142(256'h0),
.initd_143(256'h0),
.initd_144(256'h0),
.initd_145(256'h0),
.initd_146(256'h0),
.initd_147(256'h0),
.initd_148(256'h0),
.initd_149(256'h0),
.initd_150(256'h0),
.initd_151(256'h0),
.initd_152(256'h0),
.initd_153(256'h0),
.initd_154(256'h0),
.initd_155(256'h0),
.initd_156(256'h0),
```

```
.initd_157(256'h0),
.initd_158(256'h0),
.initd_159(256'h0),
.initd_160(256'h0),
.initd_161(256'h0),
.initd_162(256'h0),
.initd_163(256'h0),
.initd_164(256'h0),
.initd_165(256'h0),
.initd_166(256'h0),
.initd_167(256'h0),
.initd_168(256'h0),
.initd_169(256'h0),
.initd_170(256'h0),
.initd_171(256'h0),
.initd_172(256'h0),
.initd_173(256'h0),
.initd_174(256'h0),
.initd_175(256'h0),
.initd_176(256'h0),
.initd_177(256'h0),
.initd_178(256'h0),
.initd_179(256'h0),
.initd_180(256'h0),
.initd_181(256'h0),
.initd_182(256'h0),
.initd_183(256'h0),
.initd_184(256'h0),
.initd_185(256'h0),
.initd_186(256'h0),
.initd_187(256'h0),
.initd_188(256'h0),
.initd_189(256'h0),
.initd_190(256'h0),
.initd_191(256'h0),
.initd_192(256'h0),
.initd_193(256'h0),
.initd_194(256'h0),
.initd_195(256'h0),
.initd_196(256'h0),
.initd_197(256'h0),
.initd_198(256'h0),
.initd_199(256'h0),
.initd_200(256'h0),
.initd_201(256'h0),
.initd_202(256'h0),
```

```
.initd_203(256'h0),
.initd_204(256'h0),
.initd_205(256'h0),
.initd_206(256'h0),
.initd_207(256'h0),
.initd_208(256'h0),
.initd_209(256'h0),
.initd_210(256'h0),
.initd_211(256'h0),
.initd_212(256'h0),
.initd_213(256'h0),
.initd_214(256'h0),
.initd_215(256'h0),
.initd_216(256'h0),
.initd_217(256'h0),
.initd_218(256'h0),
.initd_219(256'h0),
.initd_220(256'h0),
.initd_221(256'h0),
.initd_222(256'h0),
.initd_223(256'h0),
.initd_224(256'h0),
.initd_225(256'h0),
.initd_226(256'h0),
.initd_227(256'h0),
.initd_228(256'h0),
.initd_229(256'h0),
.initd_230(256'h0),
.initd_231(256'h0),
.initd_232(256'h0),
.initd_233(256'h0),
.initd_234(256'h0),
.initd_235(256'h0),
.initd_236(256'h0),
.initd_237(256'h0),
.initd_238(256'h0),
.initd_239(256'h0),
.initd_240(256'h0),
.initd_241(256'h0),
.initd_242(256'h0),
.initd_243(256'h0),
.initd_244(256'h0),
.initd_245(256'h0),
.initd_246(256'h0),
.initd_247(256'h0),
.initd_248(256'h0),
```

```
.initd_249(256'h0),
.initd_250(256'h0),
.initd_251(256'h0),
.initd_252(256'h0),
.initd_253(256'h0),
.initd_254(256'h0),
.initd_255(256'h0),
.initp_00(256'h0),
.initp_01(256'h0),
.initp_02(256'h0),
.initp_03(256'h0),
.initp_04(256'h0),
.initp_05(256'h0),
.initp_06(256'h0),
.initp_07(256'h0),
.initp_08(256'h0),
.initp_09(256'h0),
.initp_10(256'h0),
.initp_11(256'h0),
.initp_12(256'h0),
.initp_13(256'h0),
.initp_14(256'h0),
.initp_15(256'h0),
.initp_16(256'h0),
.initp_17(256'h0),
.initp_18(256'h0),
.initp_19(256'h0),
.initp_20(256'h0),
.initp_21(256'h0),
.initp_22(256'h0),
.initp_23(256'h0),
.initp_24(256'h0),
.initp_25(256'h0),
.initp_26(256'h0),
.initp_27(256'h0),
.initp_28(256'h0),
.initp_29(256'h0),
.initp_30(256'h0),
.initp_31(256'h0),
.initpx_00(256'h0),
.initpx_01(256'h0),
.initpx_02(256'h0),
.initpx_03(256'h0),
.initpx_04(256'h0),
.initpx_05(256'h0),
.initpx_06(256'h0),
```

```
.initpx_07(256'h0),
.initpx_08(256'h0),
.initpx_09(256'h0),
.initpx_10(256'h0),
.initpx_11(256'h0),
.initpx_12(256'h0),
.initpx_13(256'h0),
.initpx_14(256'h0),
.initpx_15(256'h0),
.initpx_16(256'h0),
.initpx_17(256'h0),
.initpx_18(256'h0),
.initpx_19(256'h0),
.initpx_20(256'h0),
.initpx_21(256'h0),
.initpx_22(256'h0),
.initpx_23(256'h0),
.initpx_24(256'h0),
.initpx_25(256'h0),
.initpx_26(256'h0),
.initpx_27(256'h0),
.initpx_28(256'h0),
.initpx_29(256'h0),
.initpx_30(256'h0),
.initpx_31(256'h0))

instance_name (.addra(user_addra),
               .dina(user_dina),
               .dinpa(user_dinpa),
               .dinpxa(user_dinpxa),
               .wea(user_wea),
               .pea(user_pea),
               .rstlatcha(user_rstlatcha),
               .rstrega(user_rstrega),
               .outregcea(user_outregcea),
               .clka(user_clka),
               .douta(user_douta),
               .doutpa(user_doutpa),
               .doutpxa(user_doutpxa),
               .addrb(user_addrb),
               .dinb(user_dinb),
               .dinpb(user_dinpb),
               .dinpxb(user_dinpxb),
               .web(user_web),
               .peb(user_peb),
               .rstlatchb(user_rstlatchb),
```

```
                    .rstregb(user_rstregb),
                    .outregceb(user_outregceb),
                    .clkb(user_clkb),
                    .doutb(user_doutb),
                    .doutpb(user_doutpb),
                    .doutpxb(user_doutpxb));
```

BRAM80K VHDL Instantiation Template

```
------------ ACHRONIX LIBRARY ------------
library speedster22i;
use speedster22i.components.all;
------------- DONE ACHRONIX LIBRARY ---------

-- Component Instantiation
  BRAM80K_instance_name : BRAM80K
    generic map (
      porta_read_width => 40,
      porta_write_width => 40,
      porta_write_mode => "write_first",
      porta_clock_polarity => "rise",
      porta_en_out_reg => 0,
      porta_regce_priority => "rstreg",
      porta_peval => 1,
      porta_reg_rstval => 1,
      porta_latch_rstval => 1,
      porta_initval => X"0000000000",
      porta_srval => X"0000000000",
      portb_read_width => 40,
      portb_write_width => 40,
      portb_write_mode => "write_first",
      portb_clock_polarity => "rise",
      portb_en_out_reg => 0,
      portb_regce_priority => "rstreg",
      portb_peval => 1,
      portb_reg_rstval => 1,
      portb_latch_rstval => 1,
      portb_initval => X"0000000000",
      portb_srval => X"0000000000",
      mem_init_file => "",
     initd_000 => X"0000000000000000000000000000000000000000000000000000000000000000",
     initd_001 => X"0000000000000000000000000000000000000000000000000000000000000000",
     initd_002 => X"0000000000000000000000000000000000000000000000000000000000000000",
     initd_003 => X"0000000000000000000000000000000000000000000000000000000000000000",
     initd_004 => X"0000000000000000000000000000000000000000000000000000000000000000",
     initd_005 => X"0000000000000000000000000000000000000000000000000000000000000000",
     initd_006 => X"0000000000000000000000000000000000000000000000000000000000000000",
```

```
initd_007 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_008 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_009 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_010 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_011 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_012 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_013 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_014 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_015 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_016 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_017 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_018 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_019 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_020 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_021 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_022 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_023 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_024 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_025 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_026 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_027 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_028 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_029 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_030 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_031 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_032 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_033 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_034 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_035 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_036 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_037 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_038 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_039 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_040 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_041 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_042 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_043 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_044 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_045 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_046 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_047 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_048 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_049 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_050 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_051 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_052 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_053 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_054 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_055 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_056 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_057 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_058 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_059 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_060 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_061 => X"0000000000000000000000000000000000000000000000000000000000000000",
```

```
initd_062 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_063 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_064 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_065 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_066 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_067 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_068 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_069 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_070 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_071 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_072 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_073 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_074 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_075 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_076 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_077 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_078 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_079 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_080 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_081 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_082 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_083 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_084 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_085 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_086 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_087 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_088 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_089 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_090 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_091 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_092 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_093 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_094 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_095 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_096 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_097 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_098 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_099 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_100 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_101 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_102 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_103 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_104 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_105 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_106 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_107 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_108 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_109 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_110 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_111 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_112 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_113 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_114 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_115 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_116 => X"0000000000000000000000000000000000000000000000000000000000000000",
```

```
initd_117 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_118 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_119 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_120 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_121 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_122 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_123 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_124 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_125 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_126 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_127 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_128 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_129 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_130 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_131 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_132 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_133 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_134 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_135 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_136 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_137 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_138 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_139 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_140 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_141 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_142 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_143 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_144 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_145 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_146 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_147 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_148 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_149 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_150 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_151 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_152 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_153 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_154 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_155 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_156 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_157 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_158 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_159 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_160 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_161 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_162 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_163 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_164 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_165 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_166 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_167 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_168 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_169 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_170 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_171 => X"0000000000000000000000000000000000000000000000000000000000000000",
```

```
initd_172 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_173 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_174 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_175 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_176 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_177 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_178 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_179 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_180 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_181 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_182 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_183 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_184 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_185 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_186 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_187 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_188 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_189 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_190 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_191 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_192 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_193 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_194 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_195 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_196 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_197 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_198 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_199 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_200 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_201 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_202 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_203 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_204 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_205 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_206 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_207 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_208 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_209 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_210 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_211 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_212 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_213 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_214 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_215 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_216 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_217 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_218 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_219 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_220 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_221 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_222 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_223 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_224 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_225 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_226 => X"0000000000000000000000000000000000000000000000000000000000000000",
```

```
initd_227 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_228 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_229 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_230 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_231 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_232 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_233 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_234 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_235 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_236 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_237 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_238 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_239 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_240 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_241 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_242 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_243 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_244 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_245 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_246 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_247 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_248 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_249 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_250 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_251 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_252 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_253 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_254 => X"0000000000000000000000000000000000000000000000000000000000000000",
initd_255 => X"0000000000000000000000000000000000000000000000000000000000000000",
initp_00 => X"0000000000000000000000000000000000000000000000000000000000000000",
initp_01 => X"0000000000000000000000000000000000000000000000000000000000000000",
initp_02 => X"0000000000000000000000000000000000000000000000000000000000000000",
initp_03 => X"0000000000000000000000000000000000000000000000000000000000000000",
initp_04 => X"0000000000000000000000000000000000000000000000000000000000000000",
initp_05 => X"0000000000000000000000000000000000000000000000000000000000000000",
initp_06 => X"0000000000000000000000000000000000000000000000000000000000000000",
initp_07 => X"0000000000000000000000000000000000000000000000000000000000000000",
initp_08 => X"0000000000000000000000000000000000000000000000000000000000000000",
initp_09 => X"0000000000000000000000000000000000000000000000000000000000000000",
initp_10 => X"0000000000000000000000000000000000000000000000000000000000000000",
initp_11 => X"0000000000000000000000000000000000000000000000000000000000000000",
initp_12 => X"0000000000000000000000000000000000000000000000000000000000000000",
initp_13 => X"0000000000000000000000000000000000000000000000000000000000000000",
initp_14 => X"0000000000000000000000000000000000000000000000000000000000000000",
initp_15 => X"0000000000000000000000000000000000000000000000000000000000000000",
initp_16 => X"0000000000000000000000000000000000000000000000000000000000000000",
initp_17 => X"0000000000000000000000000000000000000000000000000000000000000000",
initp_18 => X"0000000000000000000000000000000000000000000000000000000000000000",
initp_19 => X"0000000000000000000000000000000000000000000000000000000000000000",
initp_20 => X"0000000000000000000000000000000000000000000000000000000000000000",
initp_21 => X"0000000000000000000000000000000000000000000000000000000000000000",
initp_22 => X"0000000000000000000000000000000000000000000000000000000000000000",
initp_23 => X"0000000000000000000000000000000000000000000000000000000000000000",
initp_24 => X"0000000000000000000000000000000000000000000000000000000000000000",
initp_25 => X"0000000000000000000000000000000000000000000000000000000000000000",
```

```
  initp_26 => X"0000000000000000000000000000000000000000000000000000000000000000",
  initp_27 => X"0000000000000000000000000000000000000000000000000000000000000000",
  initp_28 => X"0000000000000000000000000000000000000000000000000000000000000000",
  initp_29 => X"0000000000000000000000000000000000000000000000000000000000000000",
  initp_30 => X"0000000000000000000000000000000000000000000000000000000000000000",
  initp_31 => X"0000000000000000000000000000000000000000000000000000000000000000",
  initpx_00 => X"0000000000000000000000000000000000000000000000000000000000000000",
  initpx_01 => X"0000000000000000000000000000000000000000000000000000000000000000",
  initpx_02 => X"0000000000000000000000000000000000000000000000000000000000000000",
  initpx_03 => X"0000000000000000000000000000000000000000000000000000000000000000",
  initpx_04 => X"0000000000000000000000000000000000000000000000000000000000000000",
  initpx_05 => X"0000000000000000000000000000000000000000000000000000000000000000",
  initpx_06 => X"0000000000000000000000000000000000000000000000000000000000000000",
  initpx_07 => X"0000000000000000000000000000000000000000000000000000000000000000",
  initpx_08 => X"0000000000000000000000000000000000000000000000000000000000000000",
  initpx_09 => X"0000000000000000000000000000000000000000000000000000000000000000",
  initpx_10 => X"0000000000000000000000000000000000000000000000000000000000000000",
  initpx_11 => X"0000000000000000000000000000000000000000000000000000000000000000",
  initpx_12 => X"0000000000000000000000000000000000000000000000000000000000000000",
  initpx_13 => X"0000000000000000000000000000000000000000000000000000000000000000",
  initpx_14 => X"0000000000000000000000000000000000000000000000000000000000000000",
  initpx_15 => X"0000000000000000000000000000000000000000000000000000000000000000",
  initpx_16 => X"0000000000000000000000000000000000000000000000000000000000000000",
  initpx_17 => X"0000000000000000000000000000000000000000000000000000000000000000",
  initpx_18 => X"0000000000000000000000000000000000000000000000000000000000000000",
  initpx_19 => X"0000000000000000000000000000000000000000000000000000000000000000",
  initpx_20 => X"0000000000000000000000000000000000000000000000000000000000000000",
  initpx_21 => X"0000000000000000000000000000000000000000000000000000000000000000",
  initpx_22 => X"0000000000000000000000000000000000000000000000000000000000000000",
  initpx_23 => X"0000000000000000000000000000000000000000000000000000000000000000",
  initpx_24 => X"0000000000000000000000000000000000000000000000000000000000000000",
  initpx_25 => X"0000000000000000000000000000000000000000000000000000000000000000",
  initpx_26 => X"0000000000000000000000000000000000000000000000000000000000000000",
  initpx_27 => X"0000000000000000000000000000000000000000000000000000000000000000",
  initpx_28 => X"0000000000000000000000000000000000000000000000000000000000000000",
  initpx_29 => X"0000000000000000000000000000000000000000000000000000000000000000",
  initpx_30 => X"0000000000000000000000000000000000000000000000000000000000000000",
  initpx_31 => X"0000000000000000000000000000000000000000000000000000000000000000")
port map (
   addra => user_addra ,
   dina => user_dina ,
   dinpa => user_dinpa ,
   dinpxa => user_dinpxa ,
   wea => user_wea ,
   pea => user_pea ,
   rstlatcha => user_rstlatcha ,
   rstrega => user_rstrega ,
   outregcea => user_outregcea ,
   clka => user_clka ,
   douta => user_douta ,
   doutpa => user_doutpa ,
   doutpxa => user_doutpxa ,
```

```
addrb => user_addrb ,
dinb => user_dinb ,
dinpb => user_dinpb ,
dinpxb => user_dinpxb ,
web => user_web ,
peb => user_peb ,
rstlatchb => user_rstlatchb ,
rstregb => user_rstregb ,
outregceb => user_outregceb ,
clkb => user_clkb ,
doutb => user_doutb ,
doutpb => user_doutpb ,
doutpxb => user_doutpxb);
```

BRAM80KFIFO

80k-bit FIFO Memory

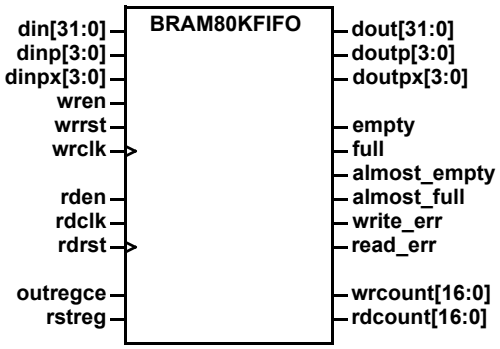


Figure 6-7: *Logic Symbol*

The BRAM80KFIFO implements a 80k-bit FIFO memory block utilizing the embedded BRAM80K blocks with dedicated pointer and flag circuitry. The BRAM80KFIFO can be configured to support a variety of widths and depths, ranging from 2k-depth with 40-bit data down to 64k-depth with 1-bit data. The read and write clocks may be either synchronous or asynchronous with respect to each other. If the user read and write clocks are the same clock, the user may set the sync_mode to 1'b1 to enable faster and synchronous generation of the status flags and FIFO pointer outputs.

Figure 6-8: *BRAM80KFIFO Block Diagram*

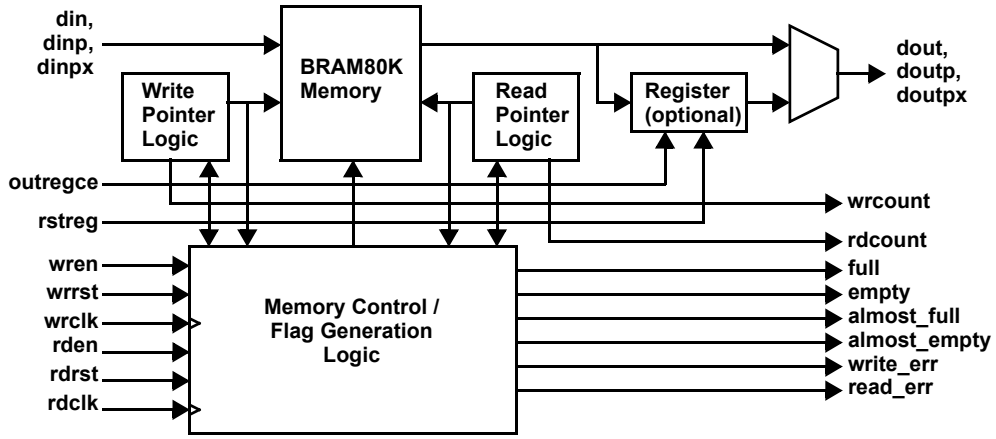


Table 6-15: *BRAM80KFIFO Pin Description*

Name	Type	Clock Domain	Description
din[31:0]	input	wrclk	**Write port data input.**
dinp[3:0]	input	wrclk	**Write port parity input (may be used for data).**
dinpx[3:0]	input	wrclk	**Write port extended parity input (may be used for data).**
wren	input	wrclk	**Write enable** (active-high). Data is written into the FIFO at the next active edge of the write clock when wren is driven high, as long as the full flag is not asserted.
wrrst	input	prog.	**Write port FIFO reset** (programmable, default active-high). Asserting the wrrst signal resets the FIFO to clear both the read and the write pointers and set the FIFO to the empty condition. The contents of the FIFO are not affected by the wrrst signal.
wrclk	input	wrclk	**Write clock.** (rising edge).
rden	input	rdclk	**Read enable** (active-high). Data is read from the FIFO at the next active edge of the clock when the rden is driven high, as long as the empty flag is not asserted.
rdrst	input	prog.	**Read port FIFO reset** (programmable, default active-high). Asserting the rdrst signal resets the FIFO to clear both the read and the write pointers and set the FIFO to the empty condition. The contents of the FIFO are not affected by the rdrst signal.
rdclk	input	rdclk	**Read clock** (rising edge).
outregce	input	rdclk	**Output register clock enable** (active-high).
rstreg	input	rdclk	**Output register reset** (programmable, default active-high).
dout[31:0]	output	rdclk	**Read port dout output.**
doutp[3:0]	output	rdclk	**Read port parity output (used for data).**
doutpx[3:0]	output	rdclk	**Read port extended parity output (used for data).**
empty	output	rdclk	**Empty flag** (active-high).
full	output	wrclk	**Full flag** (active-high).
almost_empty	output	rdclk	**Almost Empty flag** (active-high).
almost_full	output	wrclk	**Almost Full flag** (active-high).
write_err	output	wrclk	**Write Error flag** (active-high).
read_err	output	rdclk	**Read Error flag** (active-high).
wrcount[16:0]	output	prog.	**FIFO write pointer.**
rdcount[16:0]	output	prog.	**FIFO read pointer.**

Parameters

Table 6-16: *BRAM80KFIFO Parameters*

Parameter	Defined Values	Default Value
sync_mode	1'b0, 1'b1	1'b0
write_width	1, 2, 4, 5, 8, 9, 10, 16, 18, 20, 32, 36, 40	40
read_width	1, 2, 4, 5, 8, 9, 10, 16, 18, 20, 32, 36, 40	40
fwft	1'b0, 1'b1	1'b0
en_out_reg	1'b0, 1'b1	1'b1
reg_initval	40-bit hexadecimal number	40'h0
reg_srval	40-bit hexadecimal number	40'h0
reg_rstval	1'b0, 1'b1	1'b1
regce_priority	"rstreg", "regce"	"rstreg"
wrrst_rstval	1'b0, 1'b1	1'b1
wrrst_input_mode	2'b00, 2'b01, 2'b10, 2'b11	2'b10
wrrst_sync_stages	2'b00, 2'b01, 2'b10, 2'b11	2'b00
wrptr_sync_stages	2'b00, 2'b01, 2'b10, 2'b11	2'b00
rdrst_rstval	1'b0, 1'b1	1'b1
rdrst_input_mode	2'b00, 2'b01, 2'b10, 2'b11	2'b10
rdrst_sync_stages	2'b00, 2'b01, 2'b10, 2'b11	2'b00
rdptr_sync_stages	2'b00, 2'b01, 2'b10, 2'b11	2'b00
wrcount_sync_mode	1'b0, 1'b1	1'b1
rdcount_sync_mode	1'b0, 1'b1	1'b1
afull_offset	17-bit hexadecimal number	17'h00004
aempty_offset	17-bit hexadecimal number	17'h00004
wren_polarity_sel	1'b0, 1'b1	1'b1
rden_polarity_sel	1'b0, 1'b1	1'b1
en_rd_when_empty	1'b0, 1'b1	1'b0
en_wr_when_full	1'b0, 1'b1	1'b0

sync_mode

The sync_mode parameter is used to bypass the synchronization circutry between the read and write ports when both ports are connected to the same clock. If both the wrclk and rdclk clock inputs are connected to the same clock without any phase difference between the rdclk and wrclk inputs, the user may set the sync_mode parameter to 1'b1 to allow faster updates to the status flags (empty, full, etc.). If the read and write clocks are connected to different clocks, the synchronization circutry must be used and the sync_mode parameter must be set to 1'b0. The default value of the sync_mode parameter is 1'b0.

write_width

The write_width parameter defines the width of the data input bus of the FIFO from one to forty bits according to the allowed values defined in **Table 6-16: BRAM80KFIFO Parameters**. **Table 6-17: FIFO write_width versus Maximum Write Depth** defines the maximum depth of the FIFO for each of the valid values of the write_width parameter. The default value of the write_width parameter is 40'h0.

Table 6-17: *FIFO write_width versus Maximum Write Depth*

write_width	FIFO Write Depth fwft = 1'b0	FIFO Write Depth fwft = 1'b1
40	2048	2049
36	2048	2049
32	2048	2049
20	4096	4097
18	4096	4097
16	4096	4097
10	8192	8193
9	8192	8193
8	8192	8193
5	16384	16385
4	16384	16385
2	32768	32769
1	65536	65537

Table 6-18: *din, dinp, dinpx bit assignments per write_width values*

write_width	dinpx[3:0]	dinp[3:0]	din[31:0]
40	user_din[39], user_din[29], user_din[19], user_din[9]	user_din[38], user_din[28], user_din[18], user_din[8]	user_din[37:30], user_din[27:20], user_din[17-10], user_din[7:0]
36	4'hx	user_din[35], user_din[26], user_din[17], user_din[8]	user_din[34:27], user_din[25:18], user_din[16-9], user_din[7:0]
32	4'hx	4'hx	user_din[31:0]
20	2'bxx,user_din[19], user_din[9]	2'bxx,user_din[18], user_din[8]	16'hxxxx,user_din[17:10], user_din[7:0]
18	4'hx	2'bxx,user_din[17], user_din[8]	16'hxxxx,user_din[16:9], user_din[7:0]
16	4'hx	4'hx	16'hxxxx,user_din[15:0]
10	3'bxxx,user_din[9]	3'bxxx,user_din[8]	24'hxxxxxx,user_din[7:0]
9	4'hx	3'bxxx,user_din[8]	24'hxxxxxx,user_din[7:0]
8	4'hx	4'hx	24'hxxxxxx,user_din[7:0]
5	4'hx	3'bxxx,user_din[4]	28'hxxxxxxx,user_din[3:0]
4	4'hx	4'hx	28'hxxxxxxx,user_din[3:0]
2	4'hx	4'hx	30'hxxxxxxxx,user_din[1:0]
1	4'hx	4'hx	31'hxxxxxxxx,user_din[0]

read_width

The read_width parameter defines the width of the data output bus of the FIFO from one to forty bits. The allowed value of the read_width is subject to the combinations defined below

in **Table 6-19: FIFO read_width versus Maximum Read Depth**. **Table 6-17: FIFO write_width versus Maximum Write Depth** defines the maximum depth of the FIFO for each of the valid values of the write_width parameter. The default value of the read_width parameter is 40'h0.

Table 6-19: *FIFO read_width versus Maximum Read Depth*

read_width	FIFO Read Depth fwft = 1'b0	FIFO Read Depth fwft = 1'b1
40	2048	2049
36	2048	2049
32	2048	2049
20	4096	4097
18	4096	4097
16	4096	4097
10	8192	8193
9	8192	8193
8	8192	8193
5	16384	16385
4	16384	16385
2	32768	32769
1	65536	65537

Table 6-20: *dout bit assignments per read_width values*

read_width	doutpx[3:0]	doutp[3:0]	dout[31:0]
40	user_dout[39], user_dout[29], user_dout[19], user_dout[9]	user_dout[38], user_dout[28], user_dout[18], user_dout[8]	user_dout[37:30], user_dout[27:20], user_dout[17-10], user_dout[7:0]
36	4'hx	user_dout[35], user_dout[26], user_dout[17], user_dout[8]	user_dout[34:27], user_dout[25:18], user_dout[16-9], user_dout[7:0]
32	4'hx	4'hx	user_dout[31:0]
20	2'bxx,user_dout[19], user_dout[9]	2'bxx,user_dout[18], user_dout[8]	16'hxxxx,user_dout[17:10], user_dout[7:0]
18	4'hx	user_dout[17], user_dout[8]	16'hxxxx,user_dout[16:9], user_dout[7:0]
16	4'hx	4'hx	16'hxxxx,user_dout[15:0]
10	3'bxxx,user_dout[9]	3'bxxx,user_dout[8]	24'hxxxxxx,user_dout[7:0]
9	4'hx	3'bxxx,user_dout[8]	24'hxxxxxx,user_dout[7:0]
8	4'hx	4'hx	24'hxxxxxx,user_dout[7:0]
5	4'hx	3'bxxx,user_dout[4]	28'hxxxxxxx,user_dout[3:0]
4	4'hx	4'hx	28'hxxxxxxx,user_dout[3:0]
2	4'hx	4'hx	30'hxxxxxxxx,user_dout[1:0]
1	4'hx	4'hx	31'hxxxxxxxx,user_dout[0]

Required Relationship of write_width to read_width parameter assignments

The BRAM80KFIFO block supports memory widths from one to forty bits wide. The width of the din data input is determined by the write_width parameter while the width of the dout data output is determined by the read_width parameter. The width of read port may be set to be different from the width of the write port. There are, however, some limitations of the port width assignments between the read and write width assignments. The valid port widths defined in **Table 6-16: BRAM80KFIFO Parameters** may be divided into three groups:

- Group1 (n x 5 widths): 40, 20, 10, 5
- Group2 (n x 9 widths): 36, 18, 9
- Group3 (2^n widths): 32, 16, 8, 4, 2, 1

Table 6-21: *Valid Read Width Versus Write Width Combinations per port for n x 5 width modes*

Port Read Width	Port Write Width				
	2kx40	4kx20	8kx10	16kx5	Other
40	✓	✓	✓	✓	–
20	✓	✓	✓	✓	–
10	✓	✓	✓	✓	–
5	✓	✓	✓	✓	–

Table 6-22: *Valid Read Width Versus Write Width Combinations per port for n x 9 width modes*

Port Read Width	Port Write Width			
	2kx36	4kx18	8kx9	Other
36	✓	✓	✓	–
18	✓	✓	✓	–
9	✓	✓	✓	–

Table 6-23: *Valid Read Width Versus Write Width Combinations per port for 2^n width modes*

Port Read Width	Port Write Width						
	2kx32	4kx16	8kx8	16kx4	32kx2	64kx1	Other
32	✓	✓	✓	✓	✓	✓	–
16	✓	✓	✓	✓	✓	✓	–
8	✓	✓	✓	✓	✓	✓	–
4	✓	✓	✓	✓	✓	✓	–
2	✓	✓	✓	✓	✓	✓	–
1	✓	✓	✓	✓	✓	✓	–

fwft

The fwft parameter defines if the FIFO is in the First-Word-Fall_Through mode. When the fwft parameter is set to 1'b1, the first value written into the FIFO appears at the dout (and doutp, doutxp if applicable) output without having to perform a read operation. If the fwft parmaeter is set to 1'b0, the first data word written into the FIFO is available at the FIFO output one rdclk clock cycle after the first read operation. This parameter only effects the availability of the first word written into the FIFO after an empty condition. Operation of the two modes is the same after the first read operation is performed. The default value of the fwft parmeter is 1'b0. Note that the fwft parameter may only be set to 1'b1 when the sync_mode parameter is set to 1'b0.

en_out_reg

The en_out_reg parameter enables the register at the output of the FIFO. A value of 1'b0 disables the output register. When the output register is enabled by setting the en_out_reg to 1'b1, there is an additional cycle of latency for each read operation. The en_out_reg parameter is only set to its non-default value when the FIFO is in single clock mode (sync_mode = 1'b1). When the FIFO is in dual clock mode, the en_out_reg parameter must be set to 1'b1. The default value of the en_out_reg parameter is 1'b1.

reg_initval

The reg_initval parameter defines the 40-bit initial value on the output of the FIFO upon application of power to the device. The 40-bit reg_initval parameter assignment is dependent on the read_width parameter value. The association of the of the reg_initval parameter values to the dout,doutp,doutpx bits is assigned according to **Table 6-24: Relationship of reg_initval bit positions to dout,doutp,doutpx**. The default value of reg_initval is 40'h0.

Table 6-24: *Relationship of reg_initval bit positions to dout,doutp,doutpx*

read_width	doutpx reg_initval[39:36]	doutp reg_initval[35:32]	dout reg_initval[31:0]
40	user_initval[39:36]	user_initval[35:32]	user_initval[31:0]
36	4'hx	user_initval[35:32]	user_initval[31:0]
32	4'hx	4'hx	user_initval[31:0]
20	2'bxx,user_initval[37:36]	2'bxx,user_initval[33:32]	16'hxxxx,user_initval[15:0]
18	4'hx	2'bxx,user_initval[33:32]	16'hxxxx,user_initval[15:0]
16	4'hx	4'hx	16'hxxxx,user_initval[15:0]
10	3'bxxx,user_initval[36]	3'bxxx,user_initval[32]	24'hxxxxxx,user_initval[7:0]
9	4'hx	3'bxxx,user_initval[32]	24'hxxxxxx,user_initval[7:0]
8	4'hx	4'hx	24'hxxxxxx,user_initval[7:0]
5	4'hx	3'bxxx,user_initval[32]	28'hxxxxxxx,user_initval[3:0]
4	4'hx	4'hx	28'hxxxxxxx,user_initval[3:0]
2	4'hx	4'hx	30'hxxxxxxxx,user_initval[1:0]
1	4'hx	4'hx	31'hxxxxxxxx,user_initval[0]

reg_srval

The reg_srval parameter defines 40-bit value on the output of the FIFO after a synchronous reset of the output register. The 40-bit reg_srval parameter assignment is dependent on the read_width parameter value. The association of the of the reg_srval parameter values to the dout,doutp,doutpx bits is assigned according to **Table 6-24: Relationship of reg_initval bit positions to dout,doutp,doutpx** The default value of the reg_srval parameter is 40'h0. Note that this parameter is only relevant when the output register is enabled with the en_out_reg parameter.

Table 6-25: *Relationship of reg_srval bit positions to dout,doutp,doutpx*

read_width	doutpx reg_srval[39:36]	doutp reg_srval[35:32]	dout reg_srval[31:0]
40	user_srval[39:36]	user_srval[35:32]	user_srval[31:0]
36	4'hx	user_srval[35:32]	user_srval[31:0]
32	4'hx	4'hx	user_srval[31:0]
20	2'bxx,user_srval[37:36]	2'bxx,user_srval[33:32]	16'hxxxx,user_srval[15:0]
18	4'hx	2'bxx,user_srval[33:32]	16'hxxxx,user_srval[15:0]
16	4'hx	4'hx	16'hxxxx,user_srval[15:0]
10	3'bxxx,user_srval[36]	3'bxxx,user_srval[32]	24'hxxxxxx,user_srval[7:0]
9	4'hx	3'bxxx,user_srval[32]	24'hxxxxxx,user_srval[7:0]
8	4'hx	4'hx	24'hxxxxxx,user_srval[7:0]
5	4'hx	3'bxxx,user_srval[32]	28'hxxxxxxx,user_srval[3:0]
4	4'hx	4'hx	28'hxxxxxxx,user_srval[3:0]
2	4'hx	4'hx	30'hxxxxxxxx,user_srval[1:0]
1	4'hx	4'hx	31'hxxxxxxxx,user_srval[0]

reg_rstval

The reg_rstval parameter defines the active level of the output register rstreg input. Assigning a value of 1'b0 to reg_rstval configures the output register to have an active-low synchronous reset, while assigning a value of 1'b1 configures the output register to have an active-high synchronous reset. The default value of the reg_rstval parameter is 1'b1.

regce_priority

The regce_priority parameter defines the priority of the outregce clock enable input relative to the rstreg reset input during an assertion of the rstreg signal on the output register. Setting regce_priority to "rstreg" allows the output register to be set/reset at the next active edge of the rdclk without requiring a specific value on the outregce output register clock enable input. Setting regce_priority to "regce" requires that the outregce output register clock enable input is active for the output register set/reset operation to occur at the next active edge of the rdclk.

wrrst_rstval

The wrrst_rstval parameter defines the active level of the write port reset (wrrst) input. Assigning a value of 1'b0 to wrrst_rstval configures the write port reset input to have an active-low synchronous reset, while assigning a value of 1'b1 configures the write port reset input to have an active-high reset. The default value of the wrrst_rstval parameter is 1'b1.

wrrst_input_mode

The wrrst_input_mode parameter defines how the Write Pointer is reset. The FIFO macro provides the user with several options to reset the FIFO either sychronously or to synchronize the reset input to the appropriate clock domain within the FIFO without the need to implement separate synchronization circuitry in the FPGA fabric.

The Write Pointer Reset input of the Write Pointer must be synchronous to the wrclk clock domain. The user must either provide a synchronous reset via the wrrst or rdrst inputs or synchronize the rdrst input. The method to reset the Write Pointer is selected via the wrrst_input_mode paramter as defined in **Table 6-26: wrrst_input_mode Parameter Mapping.**

By configuring the wrrst_input_mode and rdrst_input_mode parameters, the user may choose to have the FIFO Write Pointer and Read Pointer reset by one or both of the wrrst/rdrst

inputs. Alternatively, the user may also program the reset of the Write Pointer and Read Pointer independently of each other. For example, the user may program the FIFO reset inputs to act independently of each other so that the Read Pointer is reset exclusively by the rdrst input to allow the contents of a previously written FIFO to be reread. Note that as a result of the Write Pointer reset, the flag outputs are also updated. A block diagram of the Write Pointer Reset selection circuitry is shown in **Figure 6-9: Read and Write Pointer Reset Input Selection Block Diagram**. The default value of the wrrst_input_mode parameter is 2'b10.

Table 6-26: *wrrst_input_mode Parameter Mapping.*

wrrst_input_mode	Selected Input for Write Pointer Reset	Write Pointer Reset Use Model
2'b00	wrrst input resets Write Pointer	Requires wrrst input is synchronous to wrclk clock domain
2'b01	rdrst input resets Write Pointer	Requires rdrst input is synchronous to wrclk clock domain
2'b10	wrrst or synchronized rdrst input resets Write Pointer	Write pointer may be reset by either the synchronous wrrst or synchonized rdrst inputs.
2'b11	Synchronized rdrst input resets Write Pointer	Write Pointer only reset by synchronized rdrst input.

Figure 6-9: *Read and Write Pointer Reset Input Selection Block Diagram*

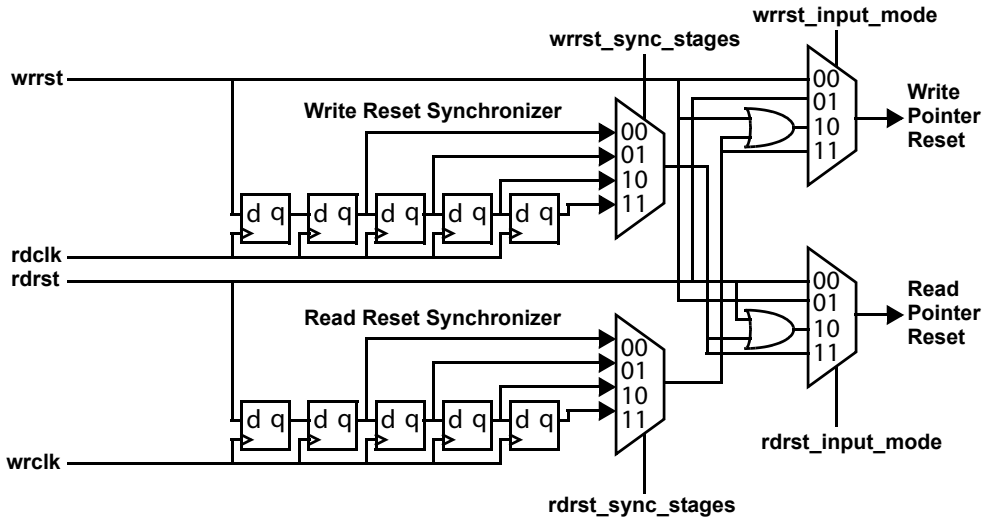


Table 6-27: *Reset Usage Model for wrrst and rdrst Inputs*

Use Model	Required wrrst and rdrst connections	Required wrrst_input_mode assignment	Required rdrst_input_mode assignment
A single reset in the rdclk domain resets both the read and write pointers.	The user reset is connected to the rdrst input. The wrrst signal is tied inactive.	2'b11	2'b00
A single reset in the wrclk domain resets both the read and write pointers.	The user reset is connected to the wrrst input. The rdrst signal is tied inactive.	2'b00	2'b11
A single asychronous reset resets both the read and write pointers.	The user reset is connected to both the wrrst and rdrst inputs.	2'b11	2'b11
The wrrst reset in the wrclk domain or the rdrst reset in the rdclk domain resets both the read and write pointers.	The user reset in the wrclk domain is connected to the wrrst input. The user reset in the rdclk domain is connected to the rdrst input.	2'b10	2'b10
The wrrst reset in the wrclk domain resets the write pointer. The rdrst reset in the rdclk domain resets the read pointer.	The user reset in the wrclk domain is connected to the wrrst input. The user reset in the rdclk domain is connected to the rdrst input.	2'b00	2'b00

wrrst_sync_stages

The wrrst_sync_stages parameter defines the number of stages of registers used to synchronize the wrrst input pin to the rdclk clock domain if the wrrst signal is used by the Read Pointer Reset. The value of the wrrst_sync_stages parameter is only used if the rdrst_input_mode is set to 2'b10 or 2'b11. The mapping of the wrrst_sync_stages parameter value to the number of sychronization stages is defined in **Table 6-28: Mapping wrrst_sync_stages Parameter Settings to Synchronization Stage Depth**, where each stage corresponds to a register in the Write Reset Synchronizer shown in **Figure 6-9: Read and Write Pointer Reset Input Selection Block Diagram**. For example, setting wrrst_sync_stages to 2'b00 configures the wrrst synchronization circuit to have two back-to-back registers in the Write Reset Synchonizer. The default value of the wrrst_sync_stages parameter is 2'b00.

Table 6-28: *Mapping wrrst_sync_stages Parameter Settings to Synchronization Stage Depth*

wrrst_sync_stages	Write Reset Synchronization Stage Depth
2'b00	2
2'b01	3
2'b10	4
2'b11	5

wrptr_sync_stages

The wrptr_sync_stages parameter defines the number of stages used in the Write Pointer Synchonizer circuit that synchronizes the Write Pointer to the rdclk clock domain. When the FIFO is in asynchronous mode, (sync_mode = 1'b0), the output of the synchonized Write Pointer is compared to the Read Pointer to generate the empty and almost_empty flags. The synchronized Write Pointer may also be routed to the wrcounter output (wrcount_sync_mode = 1'b0). The mapping of the wrptr_sync_stages parameter value to the number of sychronization stages is defined in **Table 6-29: Mapping wrptr_sync_stages Parameter Settings to Synchronization Stage Depth**, where each stage corresponds to a register in the Write Pointer Synchronizer circuit shown in **Figure 6-10: Write Pointer Synchronizer Block Diagram**. Higher values for the wrptr_sync_stages parameter reduce the possibility of a metastable

event when transferring the Write Pointer across clock domains. As an example, setting wrptr_sync_stages to 2'b00 configures the write pointer synchronization circuit to have two back-to-back registers in the Write Pointer Synchonizer. The default value of the wrptr_sync_stages parameter is 2'b00.

Table 6-29: *Mapping wrptr_sync_stages Parameter Settings to Synchronization Stage Depth*

wrptr_sync_stages	Write Pointer Synchronization Stage Depth
2'b00	2
2'b01	3
2'b10	4
2'b11	5

rdrst_rstval

The rdrst_rstval parameter defines the active level of the read port reset (rdrst) input. Assigning a value of 1'b0 to rdrst_rstval configures the read port reset input to have an active-low synchronous reset, while assigning a value of 1'b1 configures the read port reset input to have an active-high synchronous reset. The default value of the wrrst_rstval parameter is 1'b1.

rdrst_input_mode

The rdrst_input_mode parameter defines how the Read Pointer is reset. The FIFO macro provides the user with several options to reset the FIFO either sychronously or to synchronize the reset input to the appropriate clock domain within the FIFO without the need to implement separate synchronization circuitry in the FPGA fabric.

The Read Pointer Reset input of the Read Pointer must be synchronous to the rdclk clock domain. The user must either provide a synchronous reset via the wrrst or rdrst inputs or synchronize the wrrst input. The method to reset the Read Pointer is selected via the rdrst_input_mode paramter is defined in **Table 6-30: rdrst_input_mode Parameter Mapping.**

By configuring the wrrst_input_mode and rdrst_input_mode parameters, the user may choose to have the FIFO Write Pointer and Read Pointer reset by one or both of the wrrst/rdrst inputs. Alternatively, the user may also program the reset of the Write Pointer and Read Pointer independently of each other. For example, the user may program the FIFO reset inputs to act independently of each other so that the Read Pointer is reset exclusively by the rdrst input to allow the contents of a previously written FIFO to be reread. Note that as a result of the Read Pointer reset, the flag outputs are also updated. A block diagram of the Read Pointer Reset selection circuitry is shown in **Figure 6-9: Read and Write Pointer Reset Input Selection Block Diagram**. The default value of the rdrst_input_mode parameter is 2'b10.

Table 6-30: *rdrst_input_mode Parameter Mapping.*

rdrst_input_mode	Selected Input for Read Pointer Reset	Read Pointer Reset Use Model
2'b00	rdrst input resets Read Pointer	Requires rdrst input is synchronous to rdclk clock domain
2'b01	wrrst input resets Read Pointer	Requires wrrst input is synchronous to rdclk clock domain
2'b10	rdrst or synchronized wrrst input resets Read Pointer	Read pointer may be reset by either the synchronous rdrst or synchonized wrrst inputs.
2'b11	Synchronized wrrst input resets Read Pointer	Read Pointer only reset by synchronized wrrst input.

rdrst_sync_stages

The rdrst_sync_stages parameter defines the number of stages of registers used to synchronize the rdrst input pin to the wrclk clock domain if the rdrst signal is used by the Write Pointer Reset. The value of the rdrst_sync_stages parameter is only used if the wrrst_input_mode is set to 2'b10 or 2'b11. The mapping of the rdrst_sync_stages parameter value to the number of sychronization stages is defined in **Table 6-31: Mapping rdrst_sync_stages Parameter Settings to Synchronization Stage Depth**, where each stage corresponds to a register in the Read Reset Synchronizer shown in **Figure 6-9: Read and Write Pointer Reset Input Selection Block Diagram**. For example, setting rdrst_sync_stages to 2'b00 configures the rdrst synchronization circuit to have two back-to-back registers in the Write Reset Synchronizer. The default value of the rdrst_sync_stages parameter is 2'b00.

Table 6-31: *Mapping rdrst_sync_stages Parameter Settings to Synchronization Stage Depth*

rdrst_sync_stages	Read Reset Synchronization Stage Depth
2'b00	2
2'b01	3
2'b10	4
2'b11	5

rdptr_sync_stages

The rdptr_sync_stages parameter defines the number of stages used in the Read Pointer Synchonizer circuit that synchronizes the Read Pointer to the wrclk clock domain. When the FIFO is in asynchronous mode, (sync_mode = 1'b0), the output of the synchonized Read Pointer is compared to the Write Pointer to generate the full and almost_full flags. The synchronized Read Pointer may also be routed to the rdcounter output (rdcount_sync_mode = 1'b0). The mapping of the rdptr_sync_stages parameter value to the number of sychronization stages is defined in **Table 6-32: Mapping rdptr_sync_stages Parameter Settings to Synchronization Stage Depth**, where each stage corresponds to a register in the Read Pointer Synchronizer circuit shown in **Figure 6-11: Read Pointer Synchronizer Block Diagram**. Higher values for the rdptr_sync_stages parameter reduce the possibility of a metastable event when transferring the Read Pointer across clock domains. As an example, setting rdptr_sync_stages to 2'b00 configures the read pointer synchronization circuit to have two back-to-back registers in the Read Pointer Synchonizer. The default value of the rdptr_sync_stages parameter is 2'b00.

Table 6-32: *Mapping rdptr_sync_stages Parameter Settings to Synchronization Stage Depth*

rdptr_sync_stages	Read Pointer Synchronization Stage Depth
2'b00	2
2'b01	3
2'b10	4
2'b11	5

wrcount_sync_mode

The wrcount_sync_mode parameter defines whether the write counter (wrcount) output is synchonous to the wrclk clock input. Assigning a value of 1'b0 to wrcount_sync_mode configures the wrcount output to be synchonized to the rdclk clock. Assigning a value of 1'b1 to wrcount_sync_mode configures the wrcount output to be synchonized to the wrclk clock. The default value of the wrcount_sync_mode parameter is 1'b1.

rdcount_sync_mode

The rdcount_sync_mode parameter defines whether the read counter (rdcount) output is synchonous to the rdclk clock input. Assigning a value of 1'b0 to rdcount_sync_mode configures therdcount output to be synchonized to the wrclk clock. Assigning a value of 1'b1 to rdcount_sync_mode configures the rdcount output to be synchonized to the rdclk clock. The default value of the rdcount_sync_mode parameter is 1'b1.

afull_offset

The afull_offset parameter defines the word depth at which the FIFO almost_full signal changes. The almost_full flag may be used to determine the number of blind writes to the FIFO that can be made without monitoring the full flag. For example, if the afull_offset parameter is set to 17'h00004 and the almost_full flag is deasserted, the user is guaranteed that there are at least five empty locations in the FIFO. The user may write all five words without monitoring the full flag and be guaranteed that these words will be written to the FIFO and the write_err flag will not be asserted. The Maximum FIFO Depth is a function of the value assigned to the read_width parameter as shown in **Table 6-19: FIFO read_width versus Maximum Read Depth**. The default value of the afull_offset parameter is 17'h00004, corresponding to five or less available locations remain in the FIFO.

Table 6-33: *Condition to Assert almost_full Flag based on afull_offset Parameter Assignment*

Mode	Condition when almost_full flag is asserted	Condition when almost_full flag is deasserted
sync_mode = 1'b0, fwft = 1'b0	afull_offset or fewer empty locations remain in the FIFO.	There are at least (afull_offset +1) empty locations remaining in the FIFO.
sync_mode = 1'b0, fwft = 1'b1	afull_offset or fewer empty locations remain in the FIFO.	There are at least (afull_offset +1) empty locations remaining in the FIFO.
sync_mode = 1'b1, fwft = 1'b0	afull_offset or fewer empty locations remain in the FIFO.	There are at least (afull_offset +1) empty locations remaining in the FIFO.
sync_mode = 1'b1, fwft = 1'b1	Illegal parameter combination	Illegal parameter combination

aempty_offset

The aempty_offset parameter defines the word depth at which the FIFO almost_empty changes. The almost_empty flag may be used to determine the number of blind reads from the FIFO that can be perfromed without monitoring the empty flag. For example, if the aempty_offset parameter is set to 17'h0004 and the almost_empty flag is deasseted, the user is guaranteed that there are at least five words in the FIFO. The user may read all five words without monitoring the empty flag and be guaranteed that these words will be read from the FIFO and the read_err flag will not be asserted. The FIFO Depth is a function of the value assigned to the read_width parameter as shown in **Table 6-19: FIFO read_width versus Maximum Read Depth**. The default value of the aempty_offset parameter is 17'h00004, corresponding to four or less words remaining in the FIFO.

Table 6-34: *Condition to Assert almost_empty Flag based on afull_offset Parameter Assignment*

Mode	Condition when almost_empty flag is asserted	Condition when almost_empty flag is deasserted
sync_mode = 1'b0, fwft = 1'b0	aempty_offset or fewer words remain in the FIFO.	There are at least (aempty_offset +1) words in the FIFO.
sync_mode = 1'b0, fwft = 1'b1	(aempty_offset + 1) or fewer words remain in the FIFO.	There are at least (aempty_offset +1) words in the FIFO, plus the word fallen through to the output.
sync_mode = 1'b1, fwft = 1'b0	aempty_offset or fewer words remain in the FIFO.	There are at least (aempty_offset +1) words in the FIFO.
sync_mode = 1'b1, fwft = 1'b1	Illegal parameter combination	Illegal parameter combination

wren_polarity_sel

The wren_polarity_sel parameter allows the user to set the active level of the wren input signal. When wren_polarity_sel is set to 1'b0 the wren input is active-low. When wren_polarity is set to 1'b1 the wren input is active-high. The default value of the wren_polarity_sel parameter is 1'b1.

rden_polarity_sel

The rden_polarity_sel parameter allows the user to set the active level of the rden input signal. When rden_polarity_sel is set to 1'b0 the rden input is active-low. When rden_polarity is set to 1'b1 the rden input is active-high. The default value of the rden_polarity_sel parameter is 1'b1.

en_rd_when_empty

The en_rd_when_empty parameter allows the FIFO to be read when the empty flag is asserted. When en_rd_when_empty is set to 1'b0, read operations are not allowed when the empty flag is asserted. If the user tries to read the FIFO while the en_rd_when_empty parameter is set to 1'b0, the read_error signal will be asserted on the following clock cycle and the read operation will not occur. When en_rd_when_empty is set to 1'b1, read operations may be performed when the empty flag is asserted. If the user reads the FIFO while the en_rd_when_empty parameter is set to 1'b1, the read operation will occur and the read pointer will be incremented. The user should disregard the level of the read_error if the en_rd_when_empty parameter is set to 1'b1. The default value of the en_rd_when_empty parameter is 1'b0.

en_wr_when_full

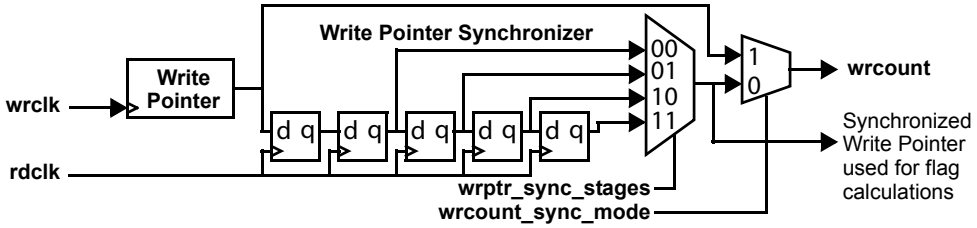
The en_wr_when_full parameter allows the FIFO to be written when the full flag is asserted. When en_wr_when_full is set to 1'b0, write operations are not allowed when the full flag is asserted. If the user tries to write the FIFO while the en_wr_when_full parameter is set to 1'b0, the write_error signal will be asserted on the following clock cycle and the write operation will not occur. When en_wr_when_full is set to 1'b1, write operations may be performed when the full flag is asserted. If the user writes the FIFO while en_wr_when_full parameter is set to 1'b1, the write operation will occur and the write pointer will be incremented. The user should disregard the level of the write_error if the en_wr_when_full parameter is set to 1'b1. The default value of the en_wr_when_full parameter is 1'b0.

Read and Write Count Outputs

Write Count Output

The Write Count (wrcount) output of the FIFO shows the value of the Write Pointer. The wrcount may be synchronized to either the wrclk or rdclk clock domains by setting the wrcount_sync_mode parameter bit. Setting the wrcount_sync_mode parameter to 1'b1 outputs the wrcount directly from the Write Pointer (synchronous to the wrclk clock). Setting the wrcount_sync_mode to 1'b0 outputs the wrcount synchronous to the rdclk output after it has been synchronized to the rdclk clock through a synchonization circuit that is wrptr_sync_stages + 2 registers deep. Note that if the Write Pointer is synchronized to the rdclk clock domain, the value of the Write Counter will be delayed and have a rate of change that will vary dependent on the rdclk clock rate. **Figure 6-10: Write Pointer Synchronizer Block Diagram** shows the Write Count Output Circuit.

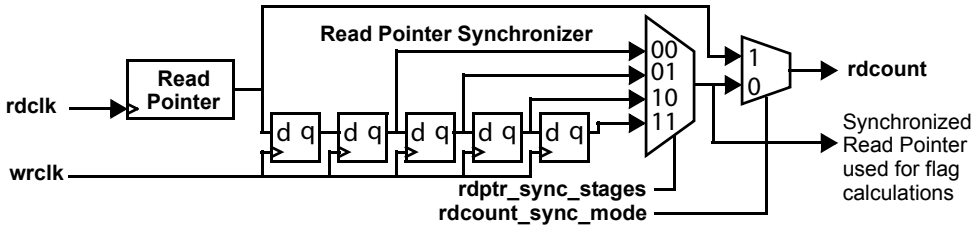
Figure 6-10: *Write Pointer Synchronizer Block Diagram*



Read Count Output

The Read Count (rdcount) output of the FIFO shows the value of the Read Pointer. The rdcount may be synchronized to either the wrclk or rdclk clock domains by setting the rdcount_sync_mode parameter bit. Setting the rdcount_sync_mode parameter to 1'b1 outputs the rdcount directly from the Read Pointer (synchronous to the rdclk clock). Setting the rdcount_sync_mode parameter to 1'b0 outputs the rdcount synchronous to the wrclk output after it has been synchronized to the wrclk clock through a synchonization circuit that is rdptr_sync_stages + 2 registers deep. Note that if the Read Pointer is synchronized to the wrclk clock domain, the value of the Read Counter will be delayed and have a rate of change that will vary dependent on the wrclk clock rate. **Figure 6-11: Read Pointer Synchronizer Block Diagram** shows the Read Count Output Circuit.

Figure 6-11: *Read Pointer Synchronizer Block Diagram*



Status Flags

Empty Flag

The Empty (empty) flag is asserted after the FIFO is reset or when all of the data has been read from the FIFO. The Empty flag is synchronous to the rdclk clock domain. Further attempts to read the FIFO when the Empty flag is asserted will depend upon the en_rd_when_empty parameter. If the en_rd_when_empty parameter is set to 1'b0, attempts to read the FIFO when the Empty flag is asserted will be blocked, the Read Error (read_err) flag will be set in the following rdclk clock cycle, and the Read Pointer will remain unchanged. If the en_rd_when_empty parameter is set to 1'b1, attempts to read the FIFO when the Empty flag is asserted will result in a valid read operation from the FIFO with the Read Pointer incrementing by one word value. The user should disregard the Read Error flag when the en_rd_when_empty parameter is set to 1'b1. The use may keep track of the Read Pointer value by monitoring the rdcount output pins.

Almost Empty Flag

The Almost Empty (almost_empty) flag is asserted when there are aempty_offset or fewer words remaining in the FIFO (See **Table 6-34: Condition to Assert almost_empty Flag based on afull_offset Parameter Assignment**). The almost_empty flag may be used to determine the number of blind reads from the FIFO that can be perfromed without monitoring the empty flag. For example, if the aempty_offset parameter is set to 17'h0004 and the almost_empty flag is deasseted, the user is guaranteed that there are at least five words in the FIFO. The user may read all five words without monitoring the empty flag and be guaranteed that these words will be read from the FIFO and the read_err flag will not be asserted. The Almost Empty flag is synchronous with the rdclk clock input.

Full Flag

The Full (full) flag is asserted when all of the available locations of the FIFO have been written. It is synchronous to the wrclk clock domain. Further attempts to write the FIFO when the Full flag is asserted will depend upon the en_wr_when_full parameter. If the en_wr_when_full parameter is set to 1'b0, attempts to write the FIFO when the Full flag is asserted will be blocked, the Write Error (write_err) flag will be set in the following wrclk clock cycle, and the Write Pointer will remain unchanged. If the en_wr_when_full parameter is set to 1'b1, attempts to write the FIFO when the Full flag is asserted will result in a valid write operation to the FIFO with the Write Pointer incrementing by one word value. The user should disregard the Write Error flag when the en_wr_when_full parameter is set to 1'b1. The use may keep track of the Write Pointer value by monitoring the wrcount output pins.

Almost Full Flag

The Almost Full (almost_full) flag is asserted when there are afull_offset or fewer available locations remaining in the FIFO. The almost_full flag may be used to determine the number of blind writes to the FIFO that can be made without monitoring the full flag. For example, if the afull_offset parameter is set to 17'h00004 and the almost_full flag is deasserted, the user is guaranteed that there are at least five empty locations in the FIFO. The user may write all five words without monitoring the full flag and be guaranteed that these words will be written to the FIFO and the write_err flag will not be asserted. The Almost Full flag is synchronous with the wrclk clock input.

Write Error Flag

The Write Error (write_err) flag is asserted in the following wrclk clock cycle when the user tries to write the FIFO while the Full Flag is high and the Enable Write When Full (en_wr_when_full) parameter is set to 1'b0. The Write Error flag is undefined when the Enable Write When Full parameter is set to 1'b1.

Read Error Flag

The Read Error (read_err) flag is asserted in the following rdclk clock cycle when the user tries to read the FIFO while the Empty Flag is high and the Enable Read When Empty (en_rd_when_empty) parameter is set to 1'b0. The Read Error flag is undefined when the Enable Read When Empty parameter is set to 1'b1.

Flag Latency

The empty, full, almost_empty, and almost_full flags are calculated based on comparisons between the FIFO write pointer and the FIFO read pointer. The write pointer is synchronous to the wrclk domain and the read pointer is synchronous to the rdclk domain. For the case of an asynchronous FIFO, the wrclk and rdclk clocks reside in different clock domains.

Table 6-35: *FIFO Pointers and Status Flag Clock Domain Assignments*

Flag	Associated Clock Domain
FIFO Write Pointer	wrclk
FIFO Read Pointer	rdclk
empty flag	rdclk
almost_empty flag	rdclk
full flag	wrclk
almost_full flag	wrclk

Before flag calculations can be made, circuitry has to make sure that both pointers are in the same clock domain as the flag for which the calculation is done. The Write Pointer Synchronizer **Figure 6-10: Write Pointer Synchronizer Block Diagram** and the Read Pointer Synchronizer **Figure 6-11: Read Pointer Synchronizer Block Diagram** are used to transfer each of the pointers into the other clock domain. In order to synchronize a given pointer to the opposite clock domain, a series of registers, whose depth is determined by the wrptr_sync_stages and rdptr_sync_stages parameters, is used. The transfer of a pointer through these registers adds additional delay to the flag calculation. The versions of the pointers used for flag calculations are shown below in **Table 6-36: Pointers Used for FIFO Flag Calculations**.

Table 6-36: *Pointers Used for FIFO Flag Calculations*

Flag	Write Pointer Used for Flag Calculation	Read Pointer Used for Flag Calculation
empty flag	Synchronized Write Pointer	Read Pointer
almost_empty flag	Synchronized Write Pointer	Read Pointer
full flag	Write Pointer	Synchronized Read Pointer
almost_full flag	Write Pointer	Synchronized Read Pointer

For example, the empty flag is computed from the Synchronized Write Pointer and the Read Pointer. The write pointer incurs an additional delay of two to five rdclk cyles (set by the wrptr_sync_stages parameter) before it is used to calculate the empty flag. Therefore, the empty flag will not transition from the empty to non-empty state for a minimum of two rdclk cycles after the first write to the FIFO occurs. A similar delay occurs for the almost_empty flag as well. Likewise, for the full and almost_full flags, there are two to five wrclk cycles of delay in the actual FIFO status due to the synchronized read pointer. For an asyncronous FIFO, the calculation of the flags does not immediately reflect the state of the FIFO. While this behavior is normal for asynchronous FIFOs, it should be noted. A synchronous FIFO has only a single clock, so there is no clock domain crossing required. A synchronous FIFO has the advantage that the flag calculation is immediate and precise. **Table 6-37: Empty and Almost Empty**

Flag Latency in Terms of Read Clock Cycles and **Table 6-38: Full and Almost Full Flag Latency in Terms of Write Clock Cycles** show the latency for the FIFO flag calculations.

Table 6-37: *Empty and Almost Empty Flag Latency in Terms of Read Clock Cycles*

FIFO Status Flag	Read Clock Cycle Latency (rdclk cycles)			
	Flag Assertion		Flag Deassertion	
	Standard Mode (fwft = 1'0)	FWFT Mode (fwft = 1'b1)	Standard Mode (fwft = 1'0)	FWFT Mode (fwft = 1'b1)
empty flag	0	0	3	4
almost empty flag	0	0	3	3

Table 6-38: *Full and Almost Full Flag Latency in Terms of Write Clock Cycles*

FIFO Status Flag	Write Clock Cycle Latency (wrclk cycles)			
	Flag Assertion		Flag Deassertion	
	Standard Mode (fwft = 1'0)	FWFT Mode (fwft = 1'b1)	Standard Mode (fwft = 1'0)	FWFT Mode (fwft = 1'b1)
full flag	0	0	3	3
almost full flag	0	0	3	3

Optional Output Register

An optional output register may be enabled at the output of the FIFO to improve the clock to out timing when in single clock mode (sync_mode = 1'b1). Enabling the output register adds an a additional cycle of latency to the output data for each read operation. It should be considered as an optional pipeline stage at the data output of the FIFO. The timing of the FIFO flags is not changed when the output register is enabled. The output register is enabled by setting the en_out_reg parameter to 1'b1. The output register is shown in **Figure 6-8: BRAM80KFIFO Block Diagram**. The output register has independent clock enable (outregce) and synchronous reset (rstreg) inputs. The output register may be configured to have an active-high or active-low reset input as determined by the reg_rstval parameter. When rstreg is asserted, the value of the reg_srval is placed on the output of the register at the next active edge of the rdclk clock. The initial power-up value of the output register is defined by the reg_initval parameter. The regce_priority parameter value determines if the reset operation is dependent on the outregce input. **Table 6-39: Function Table for Optional Output Register (Assumes active-high rdclk, active-high outregce, and active-high rstreg)** shows the functions of the optional output register.

Table 6-39: *Function Table for Optional Output Register (Assumes active-high rdclk, active-high outregce, and active-high rstreg)*

Operation	regce_priority	rstreg	outregce	rdclk	dout
Hold	X	X	X	X	dout_previous
Hold	"rstreg"	0	0	↑	dout_previous
Update Output	"rstreg"	0	1	↑	fifo_output
Reset Output	"rstreg"	1	X	↑	reg_srval
Hold	"regce"	X	0	↑	dout_previous
Update Output	"regce"	0	1	↑	fifo_output

Operation	regce_priority	rstreg	outregce	rdclk	dout
Reset Output	"regce"	1	1	↑	reg_srval

FIFO Operational Modes

The FIFO macro supports both single clock synchronous (same clock connected to wrclk and rdclk inputs without any phase offset between the two clocks) and dual clock asynchronous (two unrelated clocks or two related clocks) modes of operation. For synchronous operation, both the wrclk and rdclk inputs must be connected to the same clock net. Phase offsets between the two clocks in synchronous mode is not allowed. For asynchronous mode, the user may connect the wrclk and rdclk inputs to two different clocks. The FIFO will treat the two clocks as if they are unrelated.

Asynchronous FIFO Operation

When the FIFO is configured as an asynchronous FIFO (sync_mode = 1'b0), there are two modes of operation available. The two modes of the FIFO is controlled by the First Word Fall Through (fwft) parameter. This parameter controls what is on the output of the FIFO after the first word has been written to a FIFO that was previously in an empty state. For asynchronous operation, the user must set the en_out_reg parameter to 1'b1.

Asychronous FIFO Standard Mode (sync_mode = 1'b0, fwft = 1'b0)

After a reset operation, or after the last word has been read from the FIFO, the FIFO will be in an empty state as indicated by a high level on the empty flag. When the FIFO in set to standard mode (fwft = 1'b0), the output of the FIFO remains unchanged after the first write to a FIFO in the empty state. After the first write operation the empty flag will be deasserted indicating that there is data in the FIFO that may be read. The user must read the FIFO by setting the rden high at which time the first word written into the FIFO will be available at the FIFO outputs at the next rising edge of the rdclk input. Each subsequent read operation updates the FIFO outputs with the next stored data word if it is available (empty flag = false).

Asychronous FIFO First Word Fall Through Mode (sync_mode = 1'b0, fwft = 1'b1)

After a reset operation, or after the last word has been read from the FIFO, the FIFO will be in an empty state as indicated by a high level on the empty flag. When the FIFO in set to First Word Fall Through mode (fwft = 1'b1), the output of the FIFO will be updated after the first write to a FIFO in the empty state. After the first write operation the empty flag will be deasserted indicating that there is data in the FIFO that may be read. In First Word Fall Through mode, the first word written into the FIFO is placed on the FIFO outputs before the first read operation occurs. Each subsequent read operation updates the FIFO outputs with the next stored data word if it is available (empty flag = 1'b0). Note that when selecting First Word Fall Through mode, the FIFO must be configured for asynchronous operation (sync_mode = 1'b0). First Word Fall Through mode is not supported when the FIFO is configured for synchronous FIFO operation (sync_mode = 1'b1).

Synchronous FIFO Operation

The synchronous FIFO mode supports the standard mode. The First Word Fall Through mode is not supported when synchronous FIFO operation is selected. The user must not set the fwft parameter to 1'b1 when synchronous FIFO operation is selected. The user may either allow the FIFO to pick up the default value of the fwft parameter (1'b0) or set it to 1'b0.

Synchronous FIFO Standard Mode (sync_mode = 1'b1)

The synchronous FIFO standard mode has the advantage that there is no latency in the flag calculations, so the flags represent the exact state of the FIFO. In sycnchronous mode, the

FIFO may be configured with or without the output register enabled. When the output register is disabled, there is a single cycle of latency for the FIFO read operations, but it has a longer clock to output delay of the output data. To reduce the clock to out delay of the synchronous FIFO, the output register may be enabled (en_out_reg = 1'b1), but the FIFO data output has an additional cycle of latency. The latency of the flags is not changed by the setting of the en_out_reg parameter.

FIFO Operations

Basic Mode FIFO Reset Operation

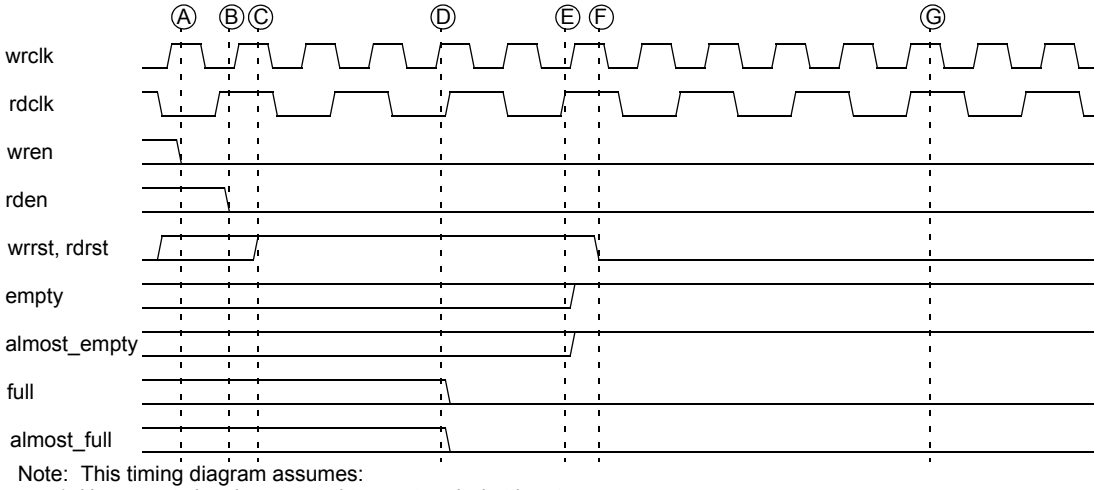
Several options are available to the user with regard to resetting the FIFO. The Basic FIFO reset allows the user to reset the FIFO without having a detailed knowledge of the FIFO sycnchronization circuit, where the reset timing is not critical to the FIFO operation. For users that require faster response time between the reset and normal FIFO operation, several advanced options are available as detailed below in **Advanced Mode FIFO Reset Operation**.

For basic FIFO Reset, the user should connect the user reset signal to both the wrrst and rdrst input pins. To reset the FIFO, the user will assert the reset signal for a minimum of three clock cycles of the slower clock cycle between the wrclk and rdclk. Asserting the reset signal clears both the Write Pointer and Read Pointer, sets the empty and almost_empty flags, and clears the full and almost_full flags. The user may then release the reset signal. The user should not attempt to read or write the FIFO while the reset is asserted or before three cycles after the deassertion of the reset signal. For Basic FIFO operation, the parameters associated with reset should be left with their default settings as shown in **Table 6-40: Parameter Settings Required for Basic Mode FIFO Reset Operation**.

Table 6-40: *Parameter Settings Required for Basic Mode FIFO Reset Operation*

Parameter	Parameter Setting for Basic Mode FIFO Reset Operation
wrrst_input_mode	2'b11
wrrst_sync_stages	2'b00
rdrst_input_mode	2'b11
rdrst_sync_stages	2'b00

Figure 6-12: *Basic Mode FIFO Reset Timing Diagram*



Note: This timing diagram assumes:
1. User reset signal connected to wrrst and rdrst inputs
2. wrrst and rdrst configured as active-high wrrst_rstval = 1'b1, rdrst_rstval = 1'b1.

Event Ⓐ: The user must disable the wren signal during the reset operation.
Event Ⓑ: The user must disable the rden signal during the reset operation.
Event Ⓒ: The user (wrrst and rdrst) reset signals asserted.
Event Ⓓ: The full and almost_full flags are deasserted (rdrst_sync_stages + 2) active wrclk edges after the reset inputs are asserted.
Event Ⓔ: The empty and almost_empty flags are asserted (wrrst_sync_stages + 2) active rdclk edges after the reset inputs are asserted.
Event Ⓕ: The user (wrrst and rdrst) reset signals are deasserted after a minimum of: max{(wrrst_sync_stages + 3) rdclk cycles, (rdrst_sync_stages + 3) wrclk cycles} after the wren and rden inputs are disabled.
Event Ⓖ: The first FIFO write operation may begin: max{(wrrst_sync_stages + 3) rdclk cycles, (rdrst_sync_stages + 3) wrclk cycles} after the user (wrrst and rdrst) reset signals are deasserted.

Advanced Mode FIFO Reset Operation

The FIFO provides the user with several options to reset the FIFO from either the read or write clock domains. The reset operation may either be sychronous with respect to the read and/or write clock domains or the user may enable the internal reset synchronization circuitry to synchronize the reset inputs. The capability for synchronous resets is provided to allow the user the fastest response between the reset operation and when the FIFO is ready to be written. Internal to the FPGA, the read and write pointers have synchronous reset inputs. The FIFO macro provides the necessary circuitry to perform the synchronization of the pointer resets without the need to implement synchronization circuitry within the FPGA fabric.

The Write Pointer reset circuitry is configured via the wrrst_sync_mode and wrsrt_sync_stages parameters and the Read Pointer reset circuitry is configured via the, rdrst_sync_mode and rdrst_sync_stages parameters. The reset operation of the read and write pointers is configured independently so that in addition to configuring the reset to be synchronous or synchonized, the resets can also be configured to respond to one or both of the wrrst or rdrst input signals. This independent configuration additionally allows the wrrst and rdrst inputs to operate separately on the write and read pointers so that the FIFO may be used as a building block for finite state machines or allow the read pointer to be reset for re-read operations.

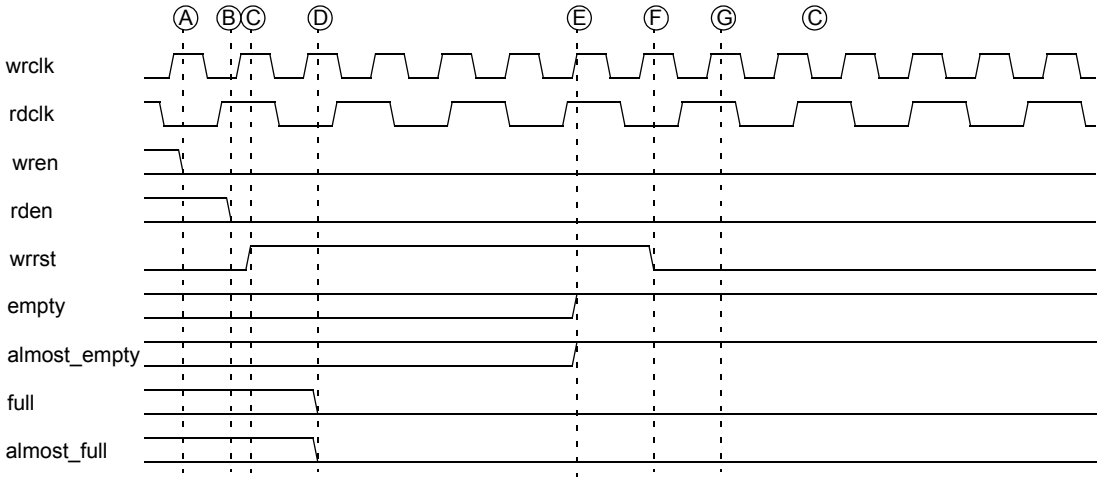
The user must take care to make sure the resets are synchronized to the proper clock domain. If the Read Pointer or Write Pointer synchonizers are bypassed, synchronization of the rdrst and wrrst will have to be performed within the FPGA fabric unless the user is already providing resets that are synchronous with the appropriate clock domain.

Figure 6-9: Read and Write Pointer Reset Input Selection Block Diagram shows the block diagram of the FIFO Reset Selection circuitry. The circuits to configure the Read Pointer and Write Pointer resets are identical. The wrrst_sync_stages(rdrst_sync_stages) parameter configures the depth of the wrrst(rdrst) synchronizer from two to five stages according to the definitions in **Table 6-28: Mapping wrrst_sync_stages Parameter Settings to Synchronization Stage Depth**.(**Table 6-31: Mapping rdrst_sync_stages Parameter Settings to Synchronization Stage Depth**). The wrrst_sync_stages(rdrst_sync_stages) parameter allows the user the capability to tune the synchronizer as a function of frequency. A higher value of the wrrst_sync_stages(rdrst_sync_stages) parameter setting is recommended for higher frequencies of FIFO operation while lower settings may be used for lower frequencies. Note that higher settings increase the number of cycles that the wrrst(rdrst) have to be asserted and increase the number of cycles between the deassertion of the wrrst(rdrst) input and the time that the user may begin to write(read) the FIFO.

For reliable operation, the user should not attempt to read or write the FIFO during a reset operation. If the wrrst synchronizer is used, the wrrst reset should be active for at least wrrst_sync_stages +3 rdclk cycles and if the rdrst synchronizer is used, the rdrst reset should be active for at least rdrst_sync_stages + 3 rdclk cycles. Write operations should not begin before wrrst_sync_stages + 3 rdclk cycles after the wrrst signal is inactive AND rdrst_sync_stages + 3 rdclk cycles after the rdrst signal is inactive to allow the deassertion of the resets to reach their synchronized clock domains.

The highest reset performance is achieved with a synchronous (single clock) FIFO. For an asynchronous FIFO, the highest reset performance is achieved by using a reset input that is synchronous with respect to the write clock domain. The Write Pointer should be configured with a synchronous reset input(wrrst_sync_mode = 2'b00) and the Read Pointer should be configure to be reset by the synchronized wrrst input(rdrst_sync_mode = 2'b11). The user should then connect the wrrst input to the FIFO reset signal. The user should tie the unused reset input to its inactive logic level.

Figure 6-13: *Reset Behavior Timing Diagram (Requires sync wrrst)*



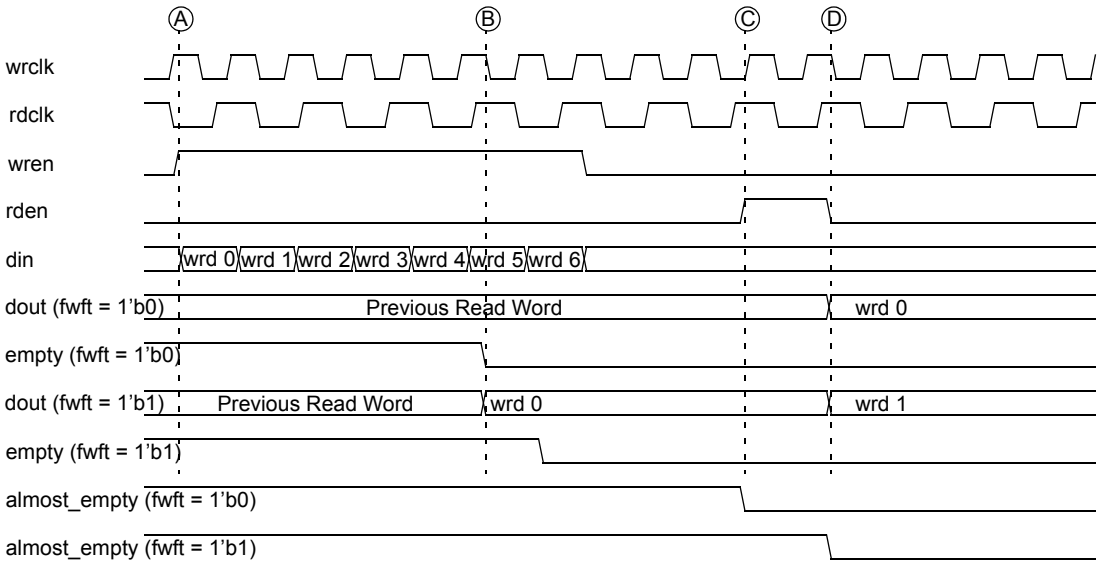
Note: This timing diagram assumes:

1. wrrst input resets both the Write Pointer and the Read Pointer
2. Write Pointer sychronously reset by wrrst input (wrrst_input_mode = 2'b00)
3. Reset Sychronizer sychronizes wrrst input with two stages of registers (rdrst_input_mode = 2'b11, wrrst_sync_stages = 2'b00)
4. wrrst and rdrst configured as active-high wrrst_rstval = 1'b1, rdrst_rstval = 1'b1

Event Ⓐ: The user must disable the wren signal during the reset operation.
Event Ⓑ: The user must disable the rden signal during the reset operation.
Event Ⓒ: The user wrrst reset signal is asserted.
Event Ⓓ: The full and almost_full flags are deasserted one active wrclk edge after wrrst is asserted.
Event Ⓔ: The empty and almost_empty flags are asserted (wrrst_sync_stages + 3) active rdclk edges after the wrrst input is asserted.
Event Ⓕ: The user (wrrst and rdrst) reset signals are deasserted after a minimum of:
max{(wrrst_sync_stages + 3) rdclk cycles, 1 wrclk cycle}
after the wren and rden inputs are disabled.
Event Ⓖ: The first FIFO write operation may begin one wrclk cycle after the user wrrst reset signal is deasserted.

Writing an Empty Asynchronous FIFO (sync_mode = 1'b0)

Figure 6-14: *Writing an Empty Asynchronous FIFO (sync_mode = 1'b0)*



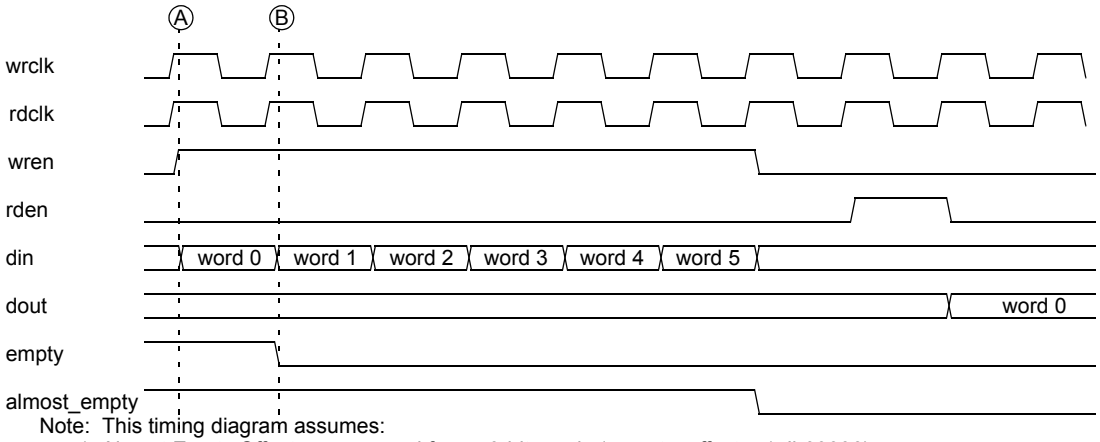
Note: This timing diagram assumes:

1. Almost Empty Offset programmed for 5 40-bit words (aempty_offset = 17'h00005)
2. wptr_sync_stages = 2'b00
3. sync_mode = 1'b0
4. In First Word Fall Through mode, the first word is guaranteed to be at the output no later than the cycle when the empty flag transitions from the empty to thenon-empty state. Due to the asynchronous transfer of the first written word from the write operation in the wrclk clock domain to the dout output in the rdclk clock domain, word 0 may arrive at the dout output before the empty flag transitions from the empty to the non-empty state.

Event Ⓐ: Begin writing 7 words to the FIFO.
Event Ⓑ: empty flag goes inactive one wrclk cycle plus (wrptr_sync_stages + 3) rdclk cycles for fwft = 1'b0. empty flag goes inactive one wrclk cycle plus (wrptr_sync_stages + 4) rdclk cycles for fwft = 1'b1. after wren becomes active. The first word (wrd0) appears at the dout output if fwft = 1'b1.
Event Ⓒ: almost_empty flag goes inactive one wrclk cycle plus (wrptr_sync_stages + 3) rdclk cycles after wrd4 (6th word) is written into the FIFO (7th word if fwft = 1'b1).
Event Ⓓ: The second word (wrd1) appears at the dout output if fwft = 1'b1. The first (wrd0) appears at the dout output if fwft - 1'b0.

Writing an Empty Synchronous FIFO (sync_mode = 1'b1)

Figure 6-15: *Writing an Empty Synchronous FIFO (sync_mode = 1'b1)*



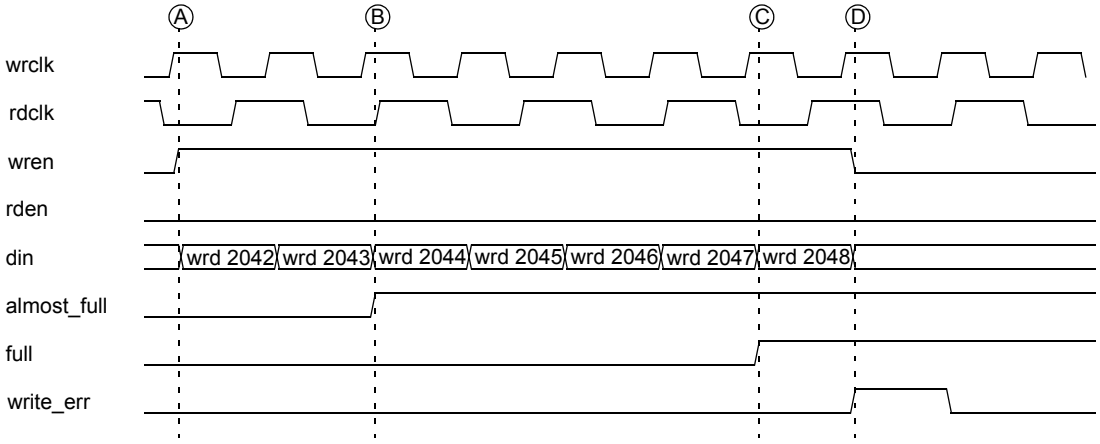
Note: This timing diagram assumes:

1. Almost Empty Offset programmed for 5 40-bit words (aempty_offset = 17'h00006)
2. sync_mode = 1'b1

Event Ⓐ: Begin writing 6 words to the FIFO.
Event Ⓑ: empty flag goes inactive one wrclk cycle after wren becomes active.

Writing to an Almost Full FIFO (en_wr_when_full = 1'b0)

Figure 6-16: *Writing to an Almost Full FIFO (en_wr_when_full = 1'b0)*

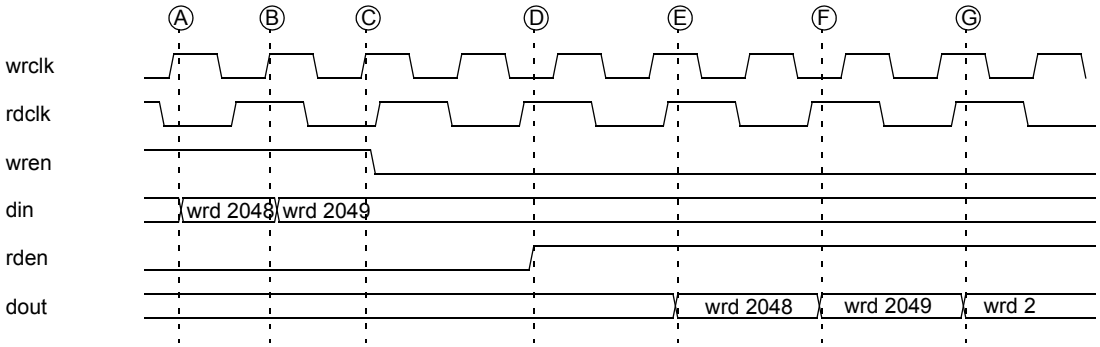


Note: This timing diagram assumes:
1. Almost Full Offset programmed for 5 40-bit words (aempty_offset = 17'h00004)
2. wptr_sync_stages = 2'b00
3. en_wr_when_empty = 1'b0

Event Ⓐ: Finish writing 2049 words to the FIFO.
Event Ⓑ: The almost_full flag is asserted.
Event Ⓒ: The full flag is asserted four write cycles after almost_full flag is asserted (assuming no reads).
Event Ⓓ: The write_err flag is asserted one wrclk cycle after attempting to write a full FIFO.

Writing to an Almost Full FIFO (en_wr_when_full = 1'b1)

Figure 6-17: *Writing to an Almost Full FIFO (en_wr_when_full = 1'b1)*

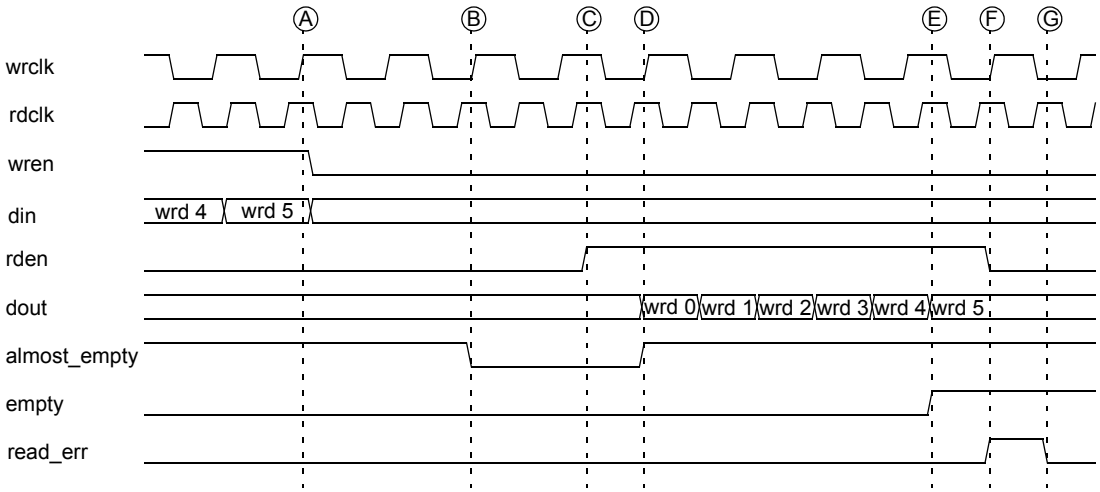


Note: This timing diagram assumes:
1. wptr_sync_stages = 2'b00
2. en_wr_when_empty = 1'b0
3. fwft = 1'b0
4. write_width = 40 and read_width = 40, implying a maximum FIFO depth of 2048 locations
5. After the FIFO was reset, a sequence of 2049 writes (wrd 0 through wrd 2048) to the FIFO occurs.

Event Ⓐ: In process of writing 2050 (wrd 0 through wrd 2049) words to the FIFO.
Event Ⓑ: The 2049th (wrd 2048) word is written to the FIFO. The first word (wrd 0) is overwritten by the 2049th word since en_wr_when_empty = 1'b1 and the maximum depth of the FIFO is 2048 words.
Event Ⓒ: The 2050th (wrd 2049) word is written to the FIFO. The second word (wrd 1) is overwritten by the 2050th word.
Event Ⓓ: Begin reading FIFO from location 0.
Event Ⓔ: First word read is wrd 2048 that overwrote wrd 0.
Event Ⓕ: Second word read is wrd 2049 that overwrote wrd 1.
Event Ⓖ: Third word read is wrd 3.

Reading from an Almost Empty FIFO (en_rd_when_empty = 1'b0, fwft = 1'b0)

Figure 6-18: *Reading From an Almost Empty FIFO (en_rd_when_empty = 1'b0, fwft = 1'b0)*



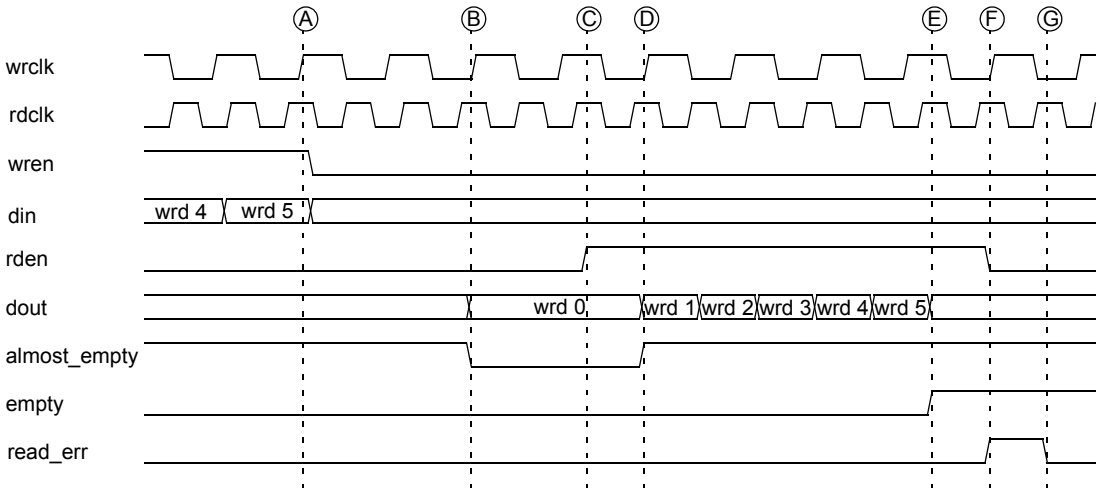
Note: This timing diagram assumes:

1. Almost Empty Offset programmed for 6 40-bit words (aempty_offset = 17'h00005)
2. wptr_sync_stages = 2'b00
3. en_rd_when_empty = 1'b0
4. fwft = 1'b0

Event Ⓐ: Finish writing 6 words to the FIFO.
Event Ⓑ: The almost_empty flag is deasserted one wrclk plus (wrptr_sync_stages + 3) rdclk active clock edges after the sixth (wrd 5) word is presented at the din input with wren high.
Event Ⓒ: Begin to read the six word from the FIFO.
Event Ⓓ: The almost_empty flag is asserted the cyle after the first read request, when five words remain in the FIFO.
Event Ⓔ: The empty flag is asserted after the last (sixth) word is read from the FIFO. The rden signal remains high, attempting to read an empty FIFO.
Event Ⓕ: The read_err signal is asserted the cycle after the attempt to read an empty FIFO. The sixth (wrd 5) word remains at the dout output.

Reading from an Almost Empty FIFO (en_rd_when_empty = 1'b0, fwft = 1'b1)

Figure 6-19: *Reading From an Almost Empty FIFO (en_rd_when_empty = 1'b0, fwft = 1'b1)*

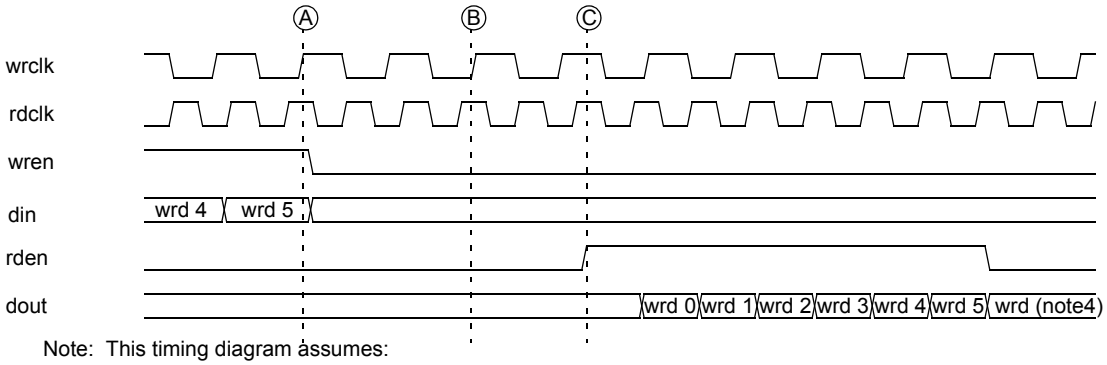


Note: This timing diagram assumes:
1. Almost Empty Offset programmed for 6 40-bit words (aempty_offset = 17'h00004)
2. wptr_sync_stages = 2'b00
3. en_rd_when_empty = 1'b0
4. fwft = 1'b1

Event Ⓐ: Finish writing 6 words to the FIFO.
Event Ⓑ: The almost_empty flag is deasserted one wrclk plus (wrptr_sync_stages + 3) rdclk active clock edges after the sixth (wrd 5) word is presented at the din input with wren high.
Event Ⓒ: Begin to read the six word from the FIFO.
Event Ⓓ: The almost_empty flag is asserted the cyle after the first read request, when five words (including the one on the FIFO output) remain in the FIFO.
Event Ⓔ: The empty flag is asserted after the last (sixth) word is taken from the FIFO output. The rden signal remains high, attempting to read an empty FIFO.
Event Ⓕ: The read_err signal is asserted the cycle after the attempt to read an empty FIFO. The sixth (wrd 5) word remains at the dout output.

Reading from an Almost Empty FIFO (en_rd_when_empty = 1'b1)

Figure 6-20: *Reading From an Almost Empty FIFO (en_rd_when_empty = 1'b1)*

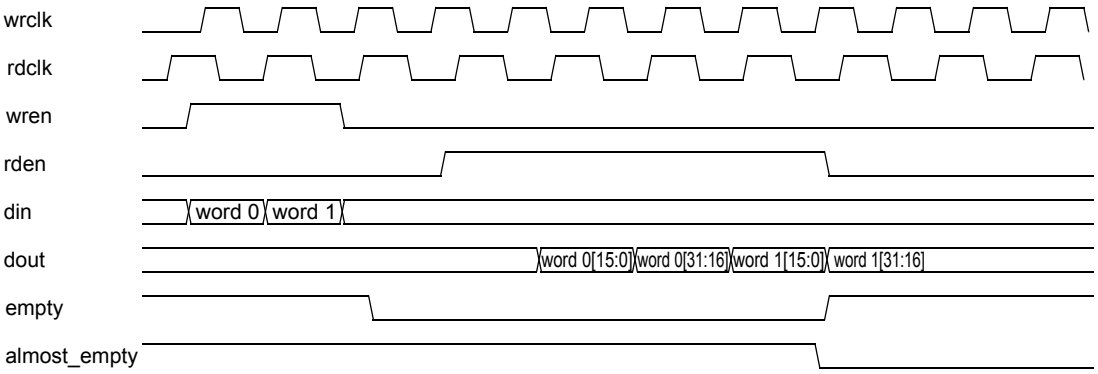


Note: This timing diagram assumes:
1. wptr_sync_stages = 2'b00
2. en_rd_when_empty = 1'b1
3. fwft = 1'b0
4. The word read after the sixth word (wrd 5) will be the last word written to the address of: (Address[wrd 5] + 1) modulo (maximum FIFO depth for programmed read_width)

Event Ⓐ: Finish writing 6 words to the FIFO.
Event Ⓑ: The almost_empty flag is deasserted one wrclk plus (wrptr_sync_stages + 3) rdclk active clock edges after the sixth (wrd 5) word is presented at the din input with wren high.
Event Ⓒ: Begin to read the six word from the FIFO.

Writing and Reading a Mixed-Width FIFO

Figure 6-21: *Writing and Reading a Mixed-Width FIFO*



Note: This timing diagram assumes:
1. write_width = 32, read_width = 16
2. wptr_sync_stages = 2'b00
3. sync_mode = 1'b0
4. fwft = 1'b0

Limitations of Concurrent Read/Write Operations

FIFO operations may be performed simultaneously from both sides of the memory, however there is a restriction with memory collisions. A memory collision is defined as the condition where both of the ports access the same memory address within the same clock cycle (both ports connected to the same clock), or within a TBD ps window (if each port is connected to a different clock). If one of the ports is writing an address while the other port is reading the same address, the write operation will occur, but the read data will be invalid. If the user programs either the en_wr_when_full or en_rd_when_empty parameters to 1'b1, it is possible for simultaneous memory operations to occur, resulting in corrupted reads from the FIFO. Even though memory collisions corrupt the read data, no damage to the hardware will occur.

BRAM80KECC

80k-bit Simple Dual-Port Memory with Error Correction

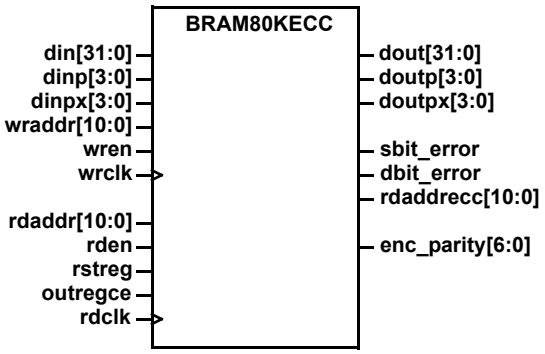


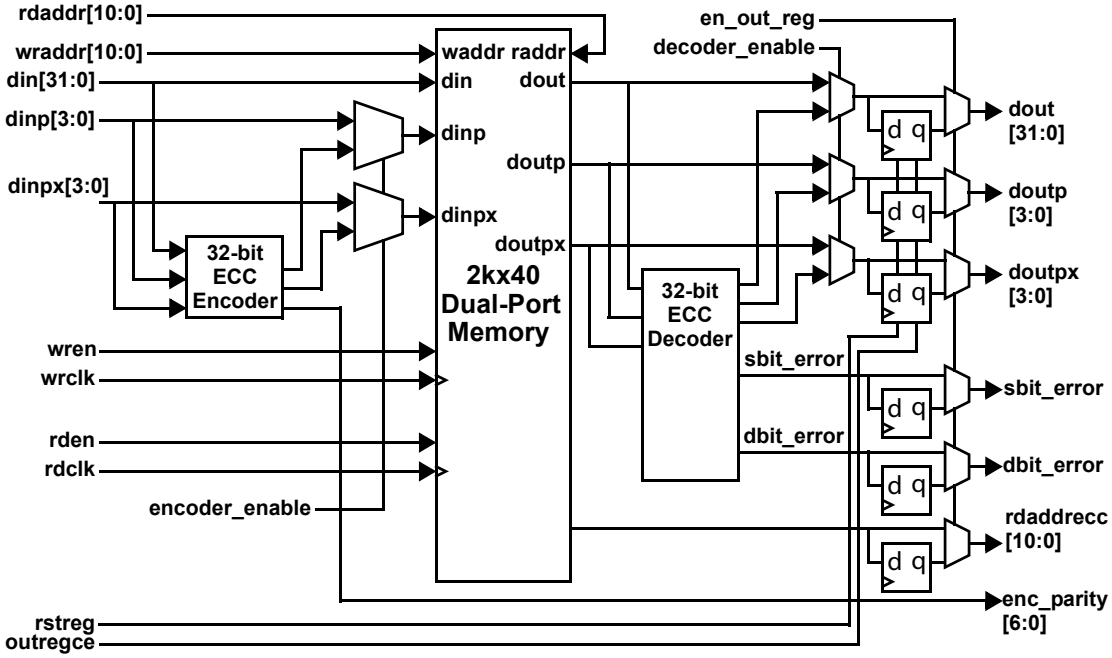
Figure 6-22: *Logic Symbol*

The BRAM80KECC implements an 2kx32 simple dual-ported memory block with error correction codes (ECC). The embedded error correction encoder generates seven parity bits and stores it alongside each word written into the memory. During the read operations, the error correction decoder reads the seven parity bits and the 32 data bits to provide error correction for all single-bit errors and error detection without correction for all two-bit errors. Both the error correction encoder and the error correction decoder may be enabled separtely to allow the ECC circuitry to be used outside of the BRAM memory block.

Although the embedded ECC circuitry corrects single-bit errors upon reading the data, the contents of the memory block do not get corrected. In order to provide the user a means to correct the memory contents, the read memory address is also available alongside the read data on the rdaddrecc outputs.

To increase the speed of the ECC memory accesses, an optional output register, complete with reset and clock enable inputs, is available to the user. The use of the output register will incur a single additional cycle of read latency.

Figure 6-23: *BRAM80KECC Block Diagram*



BRAM80KECC Pins

Table 6-41: *BRAM80KECC Pin Descriptions*

Name	Type	Description
din[31:0]	input	**Write Port Data Input.**
dinp[3:0]	input	**Write Port Parity Input (may be used for data).**
dinpx[3:0]	input	**Write Port Extended Parity input (may be used for data).**
wraddr[10:0]	input	**Write Address.**
wren	input	**Write Enable** (active-high).
wrclk	input	**Write Clock** (programmable, default rising edge = 1'b1).
rdaddr[10:0]	input	**Read Address.**
rden	input	**Read Enable** (active-high).
rstreg	input	**Output Register Reset** (programmable, default active-high).
outregce	input	**Output Register Clock Enable** (active-high).
rdclk	input	**Read Clock** (programmable, default rising edge = 1'b1).
dout[31:0]	output	**Read Port Data Output.**
doutp[3:0]	output	**Read Port Parity Output.**
doutpx[3:0]	output	**Read Port Extended Parity Output.**
sbit_error	output	**Single Bit Error.** (active-high). The sbit_error signal is asserted during a read operation when a single bit error is detected and the corrected word is output on the dout pins. The memory contents are not corrrected by the error correction circuitry. The sbit_error signal is aligned with the associated read data word.
dbit_error	output	**Double Bit Error.** (active-high). The dbit_error signal is asserted during a read operation when a two-bit error is detected. In the case of a two-bit error condition, the uncorrected read data word is output on the dout pins. The dbit_error signal is asserted one cycle after the associated read data word.
rdaddrecc[10:0]	output	**Read Address.** The rdaddrecc output is the 11-bit address of the read data word.
enc_parity[6:0]	output	**Encoder Parity.**

Parameters

Table 6-42: *BRAM80KECC Parameters*

Parameter	Defined Values	Default Value
en_out_reg	1'b0,1'b1	1'b0
reg_initval	40-bit hexadecimal number	40'h0
reg_srval	40-bit hexadecimal number	40'h0
reg_rstval	1'b0,1'b1	1'b1
regce_priority	"rstreg", "regce"	"rstreg"
encoder_enable	1'b0,1'b1	1'b1
decoder_enable	1'b0,1'b1	1'b1
mem_init_file	<path to HEX file>	""

en_out_reg

The en_out_reg parameter enables the register at the output of the BRAM80KECC block. A value of 1'b0 disables the output register. When the output register is enabled by setting the en_out_reg to 1'b1, there is an additional cycle of latency for each read operation. The default value of the en_out_reg parameter is 1'b0.

reg_initval

The reg_initval parameter defines the 40-bit initial value on the output of the BRAM80KECC upon application of power to the device. The association of the of the reg_initval parameter values to the dout,doutp,doutpx bits is assigned according to **Table 6-43: Relationship of reg_initval bit positions to dout,doutp,doutpx**. The default value of reg_initval is 40'h0.

Table 6-43: *Relationship of reg_initval bit positions to dout,doutp,doutpx*

doutpx reg_initval[39:36]	doutp reg_initval[35:32]	dout reg_initval[31:0]
user_initval[39:36]	user_initval[35:32]	user_initval[31:0]

reg_srval

The reg_srval parameter defines 40-bit value on the output of the FIFO after a synchronous reset of the output register. The association of the of the reg_srval parameter values to the dout,doutp,doutpx bits is assigned according to **Table 6-44: Relationship of reg_srval bit positions to dout,doutp,doutpx** The default value of the reg_srval parameter is 40'h0. Note that this parameter is only relevant when the output register is enabled with the en_out_reg parameter.

Table 6-44: *Relationship of reg_srval bit positions to dout,doutp,doutpx*

doutpx reg_srval[39:36]	doutp reg_srval[35:32]	dout reg_srval[31:0]
user_srval[39:36]	user_srval[35:32]	user_srval[31:0]

reg_rstval

The reg_rstval parameter defines the active level of the output register rstreg input. Assigning a value of 1'b0 to reg_rstval configures the output register to have an active-low synchronous reset, while assigning a value of 1'b1 configures the output register to have an active-high synchronous reset. The default value of the reg_rstval parameter is 1'b1.

regce_priority

The regce_priority parameter defines the priority of the outregce clock enable input relative to the rstreg reset input during an assertion of the rstreg signal on the output register. Setting regce_priority to "rstreg" allows the output register to be set/reset at the next active edge of the rdclk without requiring a specific value on the outregce output register clock enable input. Setting regce_priority to "regce" requires that the outregce output register clock enable input is active for the output register set/reset operation to occur at the next active edge of the rdclk.

encoder_enable

The encoder_enable parameter defines if the ECC encoder circuitry is selected or bypassed. Setting encoder_enable to 1'b1 enables the ECC encoder for normal operation. Setting encoder_enable to 1'b0 disables the ECC encoder circuitry and allows the din, dinp, and dinpx inputs to be connected directly to the memory write port. The default value of the encoder_enable parameter is 1'b1.

decoder_enable

The decoder_enable parameter defines if the ECC decoder circuitry is selected or bypassed. Setting decoder_enable to 1'b1 enables the ECC decoder for normal operation. Setting decoder_enable to 1'b0 disables the ECC decoder circuitry and allows the dout, doutp, and doutpx memory outputs to be routed to the output ports without error correction. The default value of the decoder_enable parameter is 1'b1.

mem_init_file

The mem_init_file parameter provides a mechanism to set the initial contents of the BRAM80KECC memory. If the mem_init_value is defined, the BRAM80KECC will be initialized with the values defined in the file pointed to by the mem_init_file parameter according to the format defined in the **Memory Initialization** section. If the mem_init_file is left at the default value of "", the initial contents will be defined by the values of the initd_000 - initd_255, initp_00 - initp_31, and the initpx_00 - initpx_31 parameters. If the memory initialization parameters and the mem_init_file parameters are not defined, the contents of the BRAM80K will be initialized to all zeros.

initd_000 – initd_255

The initd_000 through initd_255 parameters define the initial contents of the memory contents associated with dout[31:0]. Each 256-bit parameter associated with the BRAM80KECC memory as defined in the **Memory Initialization** section.

initp_0 – initp_31

The initp_00 through initp_31 parameters define the initial contents of the memory contents associated with doutp[3:0]. Each 256-bit parameter associated with the BRAM80KECC memory as defined in the **Memory Initialization** section.

initpx_0 – initpx_31

The initpx_00 through initpx_31 parameters define the initial contents of the memory contents associated with doutpx[3:0]. Each 256-bit parameter associated with the BRAM80KECC memory as defined in the **Memory Initialization** section.

BRAM80KECC Modes of Operation

There are three modes of operation for the BRAM80KECC block defined by the encoder_enable and decoder_enable parameter shown in **Table 6-45: BRAM80KECC Modes of Operation**.

Table 6-45: *BRAM80KECC Modes of Operation*

encoder_enable	decoder_enable	BRAM80KECC Operation Mode
1'b0	1'b0	Unsupported mode of operation
1'b0	1'b1	ECC decode-only mode
1'b1	1'b0	ECC encode-only mode
1'b1	1'b1	Normal ECC encode/decode mode

ECC Encode/Decode Operation Mode

The ECC Encode/Decode operation mode utilizes both the ECC Encoder and the ECC Decoder. The 32-bit user data is written into the memory via the din[31:0] inputs. The ECC Encoder generates the 7-bit error correction syndrome and writes it into the memory alongside the data word via the Parity (dinp) and Extended Parity (dinpx) inputs. During read operations, the ECC Decoder reads the 32-bit user data and the 7-bit syndrome data to generate an error correction mask. The ECC decoder will correct any single bit error and detect, but not correct, any two-bit error. If the ECC decoder detects a single bit error, it automatically corrects the error and places the corrected data on the dout[31:0] pins as well as asserts the sbit_error flag. The memory location containing the error is not corrected. If the ECC decoder detects a two-bit error, is will place the uncorrected data on the dout[31:0] pins and assert the dbit_error flag one cycle after the the data word is read.

ECC Encode-Only Operation Mode

The ECC Encode-Only operation has the ECC Encoder enabled and the ECC Decoder disabled. This mode allows the user to write the user-provided 32-bit data alongside the 7-bit error correction syndrome to the memory during write operations. Read operations allow the user to read the 32-bit user data alongside the error syndrome directly out of the memory without correcting the data. The Encode-Only can be used as a building block to have error correction for off-chip memories.

ECC Decode-Only Operation Mode

The ECC Decode-Only operation has the ECC Encoder disabled and the ECC Decoder enabled. This mode bypasses the ECC Encoder and allows the user to write 40-bit data directly into the BRAM80KECC memory during write operations. Read operations use the memory's doutpx[2:0] and doutp[3:0] bits as a 7-bit syndrome for error correction. The ECC decoder will correct any single bit error and detect, but not correct, any two-bit error. If the ECC decoder detects a single bit error, it automatically corrects the error and places the corrected data on the dout[31:0] pins as well as asserts the sbit_error flag. The memory location containing the error is not corrected. If the ECC decoder detects a two-bit error, is will place the uncorrected data on the dout[31:0] pins and assert the dbit_error flag one cycle after the the data word is read. The Encode-Only can be used as a building block to have error correction for off-chip memories.

Figure 6-24: *ECC Write Operation Timing Diagram*

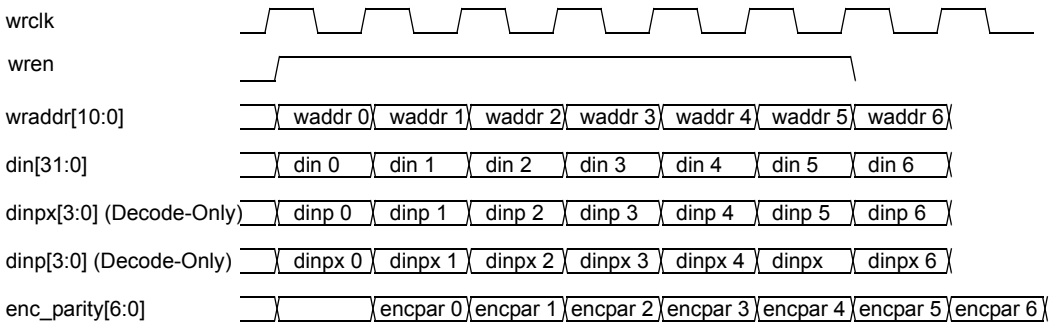


Figure 6-25: *ECC Read Operation Timing Diagram*

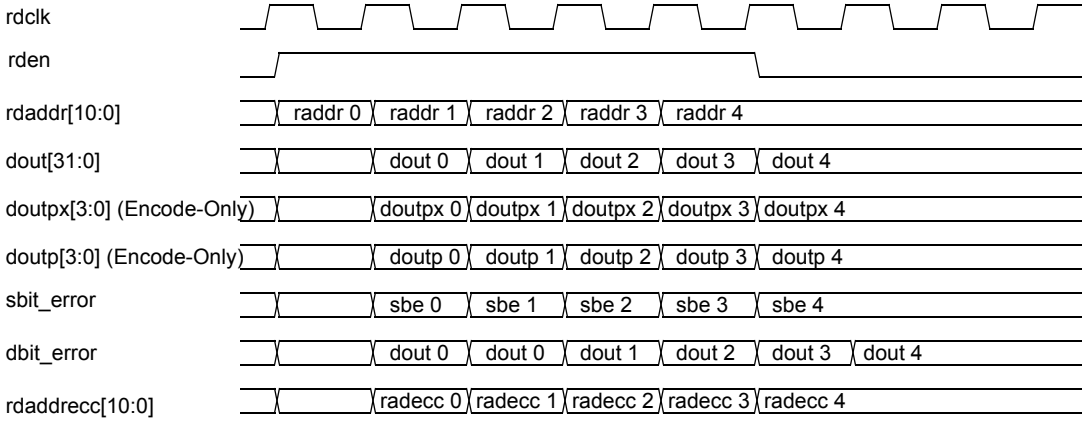


Figure 6-26: *ECC Read Operation Timing Diagram*

BRAM80KECCFIFO

80k-bit FIFO Memory with Error Correction

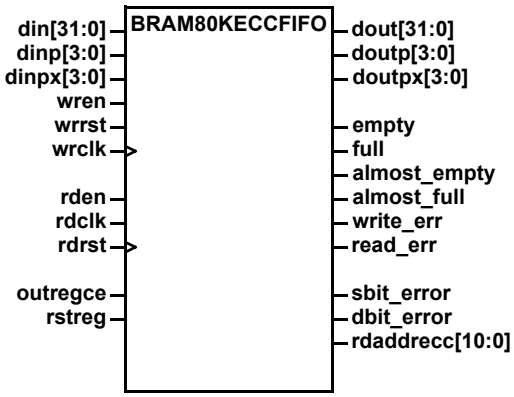


Figure 6-27: *Logic Symbol*

The BRAM80KECCFIFO implements a 2k deep, 32-bit wide FIFO memory block utilizing the embedded BRAM80K blocks with dedicated pointer and flag circuitry. The BRAM80KECCFIFO has 2k locations of 32-bit data. The read and write clocks may be either synchronous or asynchronous with respect to each other. If the user read and write clocks are the same clock, the user may set the sync_mode to 1'b1 to enable faster and synchronous generation of the status flag outputs.

Figure 6-28: *BRAM80KECCFIFO Block Diagram*

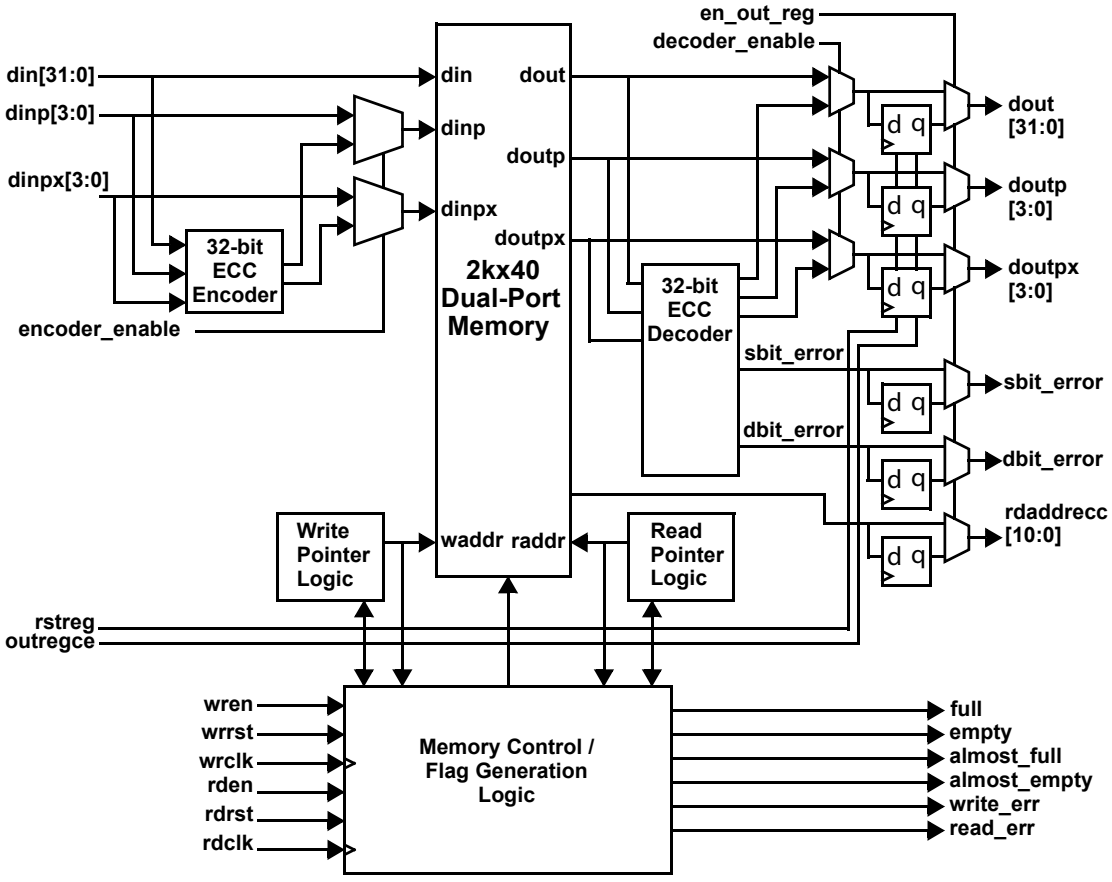


Table 6-46: *BRAM80KECCFIFO Pin Description*

Name	Type	Clock Domain	Description
din[31:0]	input	wrclk	**Write port data input.**
dinp[3:0]	input	wrclk	**Write port parity input (may be used for data).**
dinpx[3:0]	input	wrclk	**Write port extended parity input (may be used for data).**
wren	input	wrclk	**Write enable** (active-high). Data is written into the FIFO at the next active edge of the write clock when wren is driven high, as long as the full flag is not asserted.
wrrst	input	prog.	**Write port FIFO reset** (programmable, default active-high). Asserting the wrrst signal resets the FIFO to clear both read and write pointers and set the FIFO to the empty condition. The contents of the FIFO are not affected by the wrrst signal.
wrclk	input	wrclk	**Write clock.** (programmable, default rising edge = 1'b1).
rden	input	rdclk	**Read enable** (active-high). Data is read from the FIFO at the next active edge of the clock when the rden is driven high, as long as the empty flage is not asserted.
rdrst	input	prog.	**Read port FIFO reset** (programmable, default active-high). Asserting the rdrst signal resets the FIFO to clear both read and write pointers and set the FIFO to the empty condition. The contents of the FIFO are not affected by the rdrst signal.
rdclk	input	rdclk	**Read clock** (programmable, default rising edge = 1'b1).
outregce	input	rdclk	**Output register clock enable** (active-high).
rstreg	input	rdclk	**Output register reset** (programmable, default active-high).
dout[31:0]	output	rdclk	**Read port dout output.**
doutp[3:0]	output	rdclk	**Read port parity output (used for data).**
doutpx[3:0]	output	rdclk	**Read port extended parity output (used for data).**
empty	output	rdclk	**Empty flag** (active-high).
full	output	wrclk	**Full flag** (active-high).
almost_empty	output	rdclk	**Almost Empty flag** (active-high).
almost_full	output	wrclk	**Almost Full flag** (active-high).
write_err	output	wrclk	**Write Error flag** (active-high).
read_err	output	rdclk	**Read Error flag** (active-high).
sbit_error	output	rdclk	**Single Bit Error.** (active-high). The sbit_error signal is asserted during a read operation when a single bit error is detected and the corrected word is output on the dout pins. The memory contents are not corrrected by the error correction circuitry. The sbit_error signal is aligned with the associated read data word.
dbit_error	output	rdclk	**Double Bit Error.** (active-high). The dbit_error signal is asserted during a read operation when a two-bit error is detected. In the case of a two-bit error condition, the uncorrected read data word is output on the dout pins. The dbit_error signal is asserted one cycle after the associated read data word.
rdaddrecc[10:0]	output	rdclk	**Read Address.** The rdaddrecc output is the 11-bit address of the read data word.

Parameters

Table 6-47: *BRAM80KECCFIFO Parameters*

Parameter	Defined Values	Default Value
sync_mode	1'b0, 1'b1	1'b0
en_out_reg	1'b0, 1'b1	1'b1
reg_initval	40-bit hexadecimal number	40'h0
reg_srval	40-bit hexadecimal number	40'h0
reg_rstval	1'b0, 1'b1	1'b1
regce_priority	"rstreg", "regce"	"rstreg"
wrrst_rstval	1'b0, 1'b1	1'b1
wrrst_input_mode	2'b00, 2'b01, 2'b10, 2'b11	2'b10
wrrst_sync_stages	2'b00, 2'b01, 2'b10, 2'b11	2'b00
wrptr_sync_stages	2'b00, 2'b01, 2'b10, 2'b11	2'b00
rdrst_rstval	1'b0, 1'b1	1'b1
rdrst_input_mode	2'b00, 2'b01, 2'b10, 2'b11	2'b10
rdrst_sync_stages	2'b00, 2'b01, 2'b10, 2'b11	2'b00
rdptr_sync_stages	2'b00, 2'b01, 2'b10, 2'b11	2'b00
wrcount_sync_mode	1'b0, 1'b1	1'b1
rdcount_sync_mode	1'b0, 1'b1	1'b1
afull_offset	17-bit hexadecimal number	17'h00004
aempty_offset	17-bit hexadecimal number	17'h00004
encoder_enable	1'b0,1'b1	1'b1
decoder_enable	1'b0,1'b1	1'b1

For a detailed description of the BRAM80KECCFIFO parameters, refer to the like named parameter desicription given in the BRAM80KFIFO **Parameters** and the BRAM80KECC **Parameters** sections.

LRAM640

640-bit (64x10) Simple-Dual-Port Memory

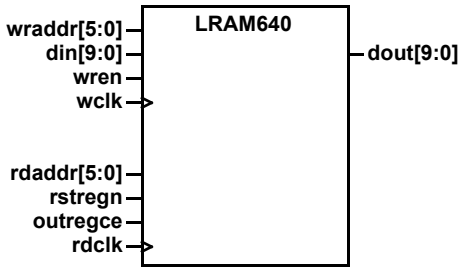
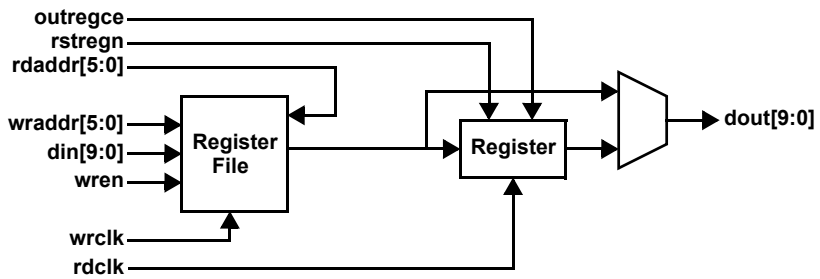


Figure 6-29: *Logic Symbol*

The Logic RAM (LRAM640) implements a 640-bit memory block with one write port and one read port. The LRAM640 can be configured as either a 64x10 simple dual-port (1 write port, 1 read port) RAM or a 64x10 single port (1 read/write port) RAM. The LRAM640 has a synchronous write port. The read port can be configured for either asynchronous or synchronous read operations. This memory block is distributed in the FPGA fabric occupying roughly five percent of the Reconfigurable Logic Blocks.

Figure 6-30: *LRAM640 Block Diagram*



LRAM640 Pins

Table 6-48: *LRAM640 Pin Descriptions*

Name	Type	Description
wraddr[5:0]	input	**Write Port Address Input.**
din[9:0]	input	**Write Port Data Input.**
wren	input	**Write Port Enable.** (active-high) When the Write Port Enable signal is asserted, the data present on the Write Port Data Input (din[9:0]) is written into the memory location addressed by the Write Port Address input (wraddr[5:0]) at the next active edge of wrclk.
wrclk	input	**Write Port Clock.** (programmable, default rising edge).
rdaddr[5:0]	input	**Read Port Address Input.**
rstregn	input	**Read Port Output Register Synchronous Set/Reset** (active-low). When rstregn is asserted, the value of the reg_rstval parameter is written to the Read Port Output Register upon the next active edge of the rdclk input. The priority of rstregn input relative to the Read Port Output Register clock enable (outregce) input is determined by the value of the regce_priority parameter. The rstregn signal only resets the Read Port Output register. It does not reset the memory contents.
outregce	input	**Read Port Output Register Clock Enable** (active-high).
rdclk	input	**Read Port Clock.** (programmable, default rising edge).
dout[9:0]	output	**Read Port Data Output.** The Read Port Data Output is configured to be either synchronous or asynchronous as determined by the reg_dout parameter. If the reg_dout parameter is set to 1'b0, the Read Port Data Output reads the contents of the memory addressed by the Read Port Address onto the dout[9:0] pins. If the reg_dout parameter is set to 1'b1, the output of the Read Port Data is driven by the contents of the memory addressed by the Read Port Address at the next active edge of rclk if the Read Port Output Clock Enable input is high.

Parameters

Table 6-49: *LRAM640 Parameters*

Parameter	Defined Values	Default Value
write_clock_polarity	"rise", "fall"	"rise"
read_clock_polarity	"rise", "fall"	"rise"
reg_dout	1'b0,1'b1	1'b0
reg_initval	10-bit binary or hexadecimal value	10'h0
reg_rstval	10-bit binary or hexadecimal value	10'h0
regce_priority	"rstreg", "regce"	"rstreg"
mem_init	640-bit hexadecimal value	640'hx
mem_init_file	<path to HEX file>	""

write_clock_polarity

The write_clock_polarity parameter is used to set the active edge of the wrclk clock. A value of "rise" corresponds to an active rising edge assignment while "fall" corresponds to an active falling edge assignment. The default value of the write_clock_polarity parameter is "rise".

read_clock_polarity

The read_clock_polarity parameter is used to set the active edge of the rdclk clock. A value of "rise" corresponds to an active rising edge assignment while "fall" corresponds to an active falling edge assignment. The default value of the read_clock_polarity paramter is "rise".

reg_dout

The reg_dout parameter defines if the Read Port Output Register is used or bypassed. Setting reg_dout to 1'b0 bypasses the register while setting reg_dout to 1'b1 enables the register. Enabling the output register incurrs an additional cycle of latency for the read operation. The reg_dout parameter defaults to the value 1'b0.

reg_initval

The reg_initval parameter defines the power-up default value of the Read Port Output Register. The reg_initval parameter defaults to the value 10'h0.

reg_rstval

The reg_rstval parameter defines the value assigned to the Read Port Output Register when the rstregn input is asserted low and there is active edge of the rdclk. The reg_rstval parameter defaults to the value 10'h0.

regce_priority

The regce_priority parameter defines the priority of the outregce clock enable input relative to the rstregn reset input during an assertion of the rstregn reset input on the Read Port Output Register. Setting regce_priority to "rstreg" allows the Read Port Output Register to be set/reset at the next active edge of rdclk without requiring the outregce clock enable input to be active. Setting regce_priority to "regce" requires that the regce clock enable input is high for the reset operation to occur at the next active edge of rdclk. The default value of the regce_priority_a parameter is "rstreg".

mem_init

The mem_init parameter defines the initial contents of the memory. The 640-bit parameter associated with the LRAM640 memory as defined in the **LRAM640 Memory Initialization** section.

mem_init_file

The mem_init_file parameter provides a mechanism to set the initial contents of the LRAM640 memory. If the mem_init_value is defined, the LRAM640 will be initialized with the values defined in the file pointed to by the mem_init_file parameter according to the format defined in the **LRAM640 Memory Initialization** section. If the mem_init_file is left at the default value of "", the initial contents will be defined by the value of the mem_init parameter. If the memory initialization parameter and the mem_init_file parameters are not defined, the contents of the LRAM640 will be undefined.

Simultaneous Memory Operations

Memory operations may be performed simultaneously from both sides of the memory, however there is a restriction with memory collisions. A memory collision is defined as the condition where both of the ports access the same memory address within the same clock cycle (both ports connected to the same clock), or within a TBD ps window (if each port is connected to a different clock). The definition of a memory collision depends on whether or not the Read Port Output Register is enabled.

If the Read Port Output Register in not enabled (reg_dout = 1'b0), a memory collision is defined by reading the same address the cycle after a write command has occurred. If the Read Port Output Reigister is enabled (reg_dout = 1'b1), a memory collision is defined by reading the same address two cycle after a write command has occurred. If a memory collision occurs, the write to the memory is valid, but the read data may be incorrect.

LRAM640 Memory Initialization

By default, the contents of the LRAM640 memory are undefined. If the user wants the initial contents to be defined, he may assign them from either a file pointed to by the mem_init_file parameter or assign them from the value of the mem_init parameter.

The memory is organized as little-endian with bit 0 mapped to bit zero of parameter mem_init and bit 639 mapped to bit 639 of parameter mem_init.

The LRAM640 memory block may alternatively be initialized with a memory file by setting the mem_init_file to point to the path of a memory initialization file. The file format in the latter case is defined by hexadecimal entries separated by white space, where the white space is defined by spaces or line separation. Each number is written as an 8-bit hexadecimal number. Commenting is allowed following a double-slash (//) through to the end of the line. C-like commenting is also allowed where the characters between the /* and */ are ignored. The memory is initialized starting with the first entry of the file initializing the memory array starting with address zero, moving upward. Each line consists as the number itself represented as a hexadecimal number.

LRAMFIFO

LRAM-Based 64-Word FIFO Memory

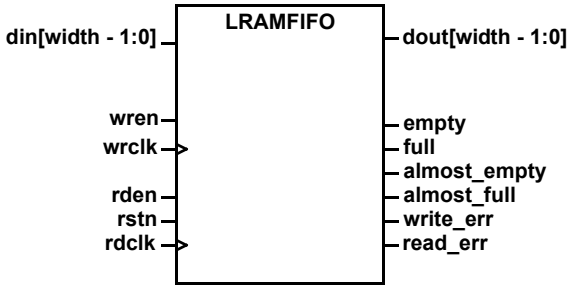


Figure 6-31: *Logic Symbol*

The LRAMFIFO implements a 64 word by n-bit FIFO memory block utilizing the embedded LRAM blocks and LUTs. The LRAMFIFO can be configured to support a variety of widths in increments of one bit. The read and write clocks may be either synchronous or asynchronous with respect to each other. If the user read and write clocks are the same clock, the user may set the sync_mode to 1'b1 to enable faster and synchronous generation of the status flags.

Figure 6-32: *LRAMFIFO Block Diagram*

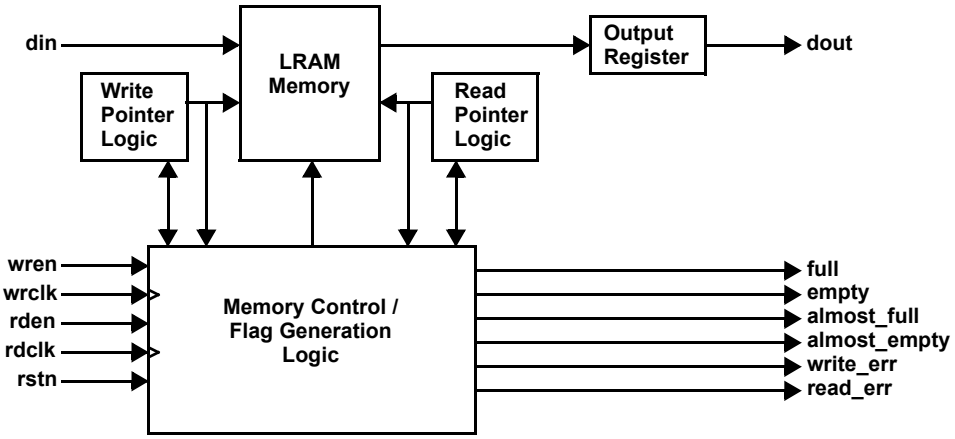


Table 6-50: *LRAMFIFO Pin Description*

Name	Type	Clock Domain	Description
din[width-1:0]	input	wrclk	**Write port data input.**
wren	input	wrclk	**Write enable** (active-high). Data is written into the FIFO at the next rising edge of the write clock when wren is driven high, as long as the full flag is not asserted.
wrclk	input	wrclk	**Write clock.** (rising edge based).
rstn	input	async.	**FIFO reset** (active-low). Asserting the rstn signal low resets the FIFO to clear both the read and the write pointers and set the FIFO to the empty condition.
rden	input	rdclk	**Read enable** (active-high). Data is read from the FIFO at the next active edge of the clock when the rden is driven high, as long as the empty flag is not asserted.
rdclk	input	rdclk	**Read clock** (rising edge based).
dout[width-1:0]	output	rdclk	**Read port dout output.**
empty	output	rdclk	**Empty flag** (active-high).
full	output	wrclk	**Full flag** (active-high).
almost_empty	output	rdclk	**Almost Empty flag** (active-high).
almost_full	output	wrclk	**Almost Full flag** (active-high).
write_err	output	wrclk	**Write Error flag** (active-high).
read_err	output	rdclk	**Read Error flag** (active-high).

Parameters

Table 6-51: *LRAMFIFO Parameters*

Parameter	Defined Values	Default Value
width	positive integers	10
depth	8-64	10
ptr_sync_mode	1'b0, 1'b1	1'b0
wrptr_sync_stages	2'b00, 2'b01, 2'b10, 2'b11	2'b00
rdptr_sync_stages	2'b00, 2'b01, 2'b10, 2'b11	2'b00
rst_sync_mode	1'b0, 1'b1	1'b0
wrrst_sync_stages	2'b00, 2'b01, 2'b10, 2'b11	2'b00
rdrst_sync_stages	2'b00, 2'b01, 2'b10, 2'b11	2'b00
afull_offset	7-bit hexadecimal number	7'h04
aempty_offset	7-bit hexadecimal number	7'h04

width

The width parameter defines the width of the data input and output busses of the FIFO. The width parameter must be a multiple of ten bits. The default value of the width parameter is 10'h0.

depth

The depth parameter defines the depth of the FIFO, which may be up to 64 locations. Choosing a depth of less than the full depth of 64 locations will allow a smaller implementation of the FIFO controller logic. The default value of the depth parameter is 64.

ptr_sync_mode

The ptr_sync_mode parameter is used to bypass the synchronization circuitry between the read and write ports when both ports are connected to the same clock. If both the wrclk and rdclk clock inputs are connected to the same clock, the user may set the ptr_sync_mode parameter to 1'b1 to allow faster updates to the status flags (empty, full, etc.). Note that when ptr_sync_mode is set to 1'b1, the two clocks have to be connected to the same clock net. There cannot be any phase difference between the two clocks in single clock mode (ptr_sync_mode = 1'b1). If the read and write clocks are connected to different clocks, the synchronization circuitry must be used and the ptr_sync_mode parameter must be set to 1'b0. The default value of the ptr_sync_mode parameter is 1'b0.

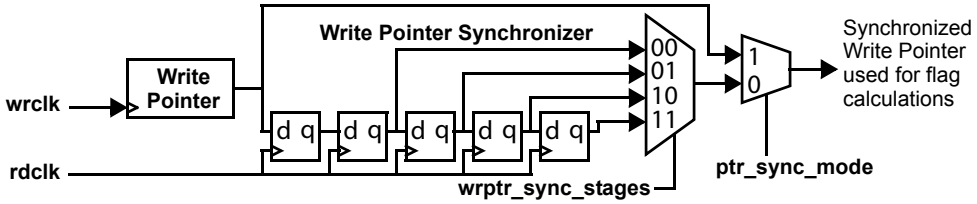
wrptr_sync_stages

The wrptr_sync_stages parameter defines the number of stages used in the Write Pointer Synchonizer circuit that synchronizes the Write Pointer to the rdclk clock domain. When the FIFO is in asynchronous mode, (ptr_sync_mode = 1'b0), the output of the synchonized Write Pointer is compared to the Read Pointer to generate the empty and almost_empty flags. The mapping of the wrptr_sync_stages parameter value to the number of sychronization stages is defined in **Table 6-52: Mapping wrptr_sync_stages Parameter Settings to Synchronization Stage Depth**, where each stage corresponds to a register in the Write Pointer Synchronizer circuit shown in **Figure 6-33: Write Pointer Synchronizer Block Diagram**. Higher values for the wrptr_sync_stages parameter reduce the possibility of a metastable event when transferring the Write Pointer across clock domains. As an example, setting wrptr_sync_stages to 2'b00 configures the write pointer synchronization circuit to have two back-to-back registers in the Write Pointer Synchonizer. The default value of the wrptr_sync_stages parameter is 2'b00.

Table 6-52: *Mapping wrptr_sync_stages Parameter Settings to Synchronization Stage Depth*

wrptr_sync_stages	Write Pointer Synchronization Stage Depth
2'b00	2
2'b01	3
2'b10	4
2'b11	5

Figure 6-33: *Write Pointer Synchronizer Block Diagram*



rdptr_sync_stages

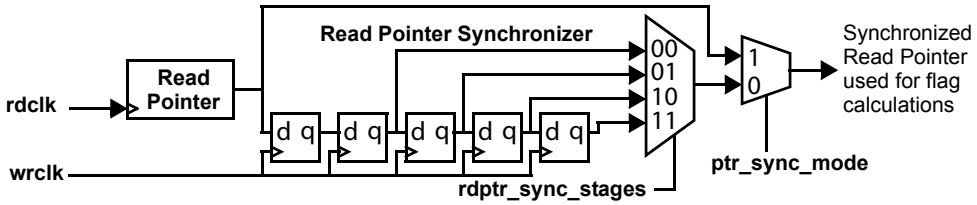
The rdptr_sync_stages parameter defines the number of stages used in the Read Pointer Synchonizer circuit that synchronizes the Read Pointer to the wrclk clock domain. When the FIFO is in asynchronous mode, (ptr_sync_mode = 1'b0), the output of the synchonized Read Pointer is compared to the Write Pointer to generate the full and almost_full flags. The mapping of the rdptr_sync_stages parameter value to the number of sychronization stages is defined in **Table 6-53: Mapping rdptr_sync_stages Parameter Settings to Synchronization Stage Depth**, where each stage corresponds to a register in the Read Pointer Synchronizer circuit shown in **Figure 6-34: Read Pointer Synchronizer Block Diagram**. Higher values for the rdptr_sync_stages parameter reduce the possibility of a metastable event when

transferring the Read Pointer across clock domains. As an example, setting rdptr_sync_stages to 2'b00 configures the read pointer synchronization circuit to have two back-to-back registers in the Read Pointer Synchonizer. The default value of the rdptr_sync_stages parameter is 2'b00.

Table 6-53: *Mapping rdptr_sync_stages Parameter Settings to Synchronization Stage Depth*

rdptr_sync_stages	Read Pointer Synchronization Stage Depth
2'b00	2
2'b01	3
2'b10	4
2'b11	5

Figure 6-34: *Read Pointer Synchronizer Block Diagram*



rst_sync_mode

The rst_sync_mode parameter affects the behavior of the reset operation of the write and read pointers. When the rst_sync_mode parameter is set to 1'b0, both the read and write pointers resets utilize the Reset Synchronizer circuitry. When the rst_sync_mode parameter is set to 1'b1 the rstn input must be synchronous to the wrclk/rdclk clock driving the FIFO. The default value of the rst_sync_mode parameter is 1'b0.

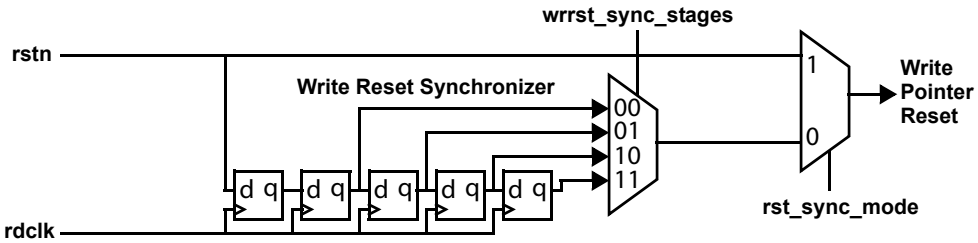
wrrst_sync_stages

The wrrst_sync_stages parameter defines the number of stages of registers used to synchronize the rstn input pin to the rdclk clock domain. The value of the wrrst_sync_stages parameter is only used if the rst_sync_mode is set to 1'b0. The mapping of the wrrst_sync_stages parameter value to the number of sychronization stages is defined in **Table 6-54: Mapping wrrst_sync_stages Parameter Settings to Synchronization Stage Depth**, where each stage corresponds to a register in **Figure 6-35: Write Pointer Reset Input Selection Block Diagram**. For example, setting wrrst_sync_stages to 2'b00 configures the rstn synchronization circuit to have two back-to-back registers in the Write Reset Synchonizer. The default value of the wrrst_sync_stages parameter is 2'b00.

Table 6-54: *Mapping wrrst_sync_stages Parameter Settings to Synchronization Stage Depth*

wrrst_sync_stages	Write Reset Synchronization Stage Depth
2'b00	2
2'b01	3
2'b10	4
2'b11	5

Figure 6-35: *Write Pointer Reset Input Selection Block Diagram*



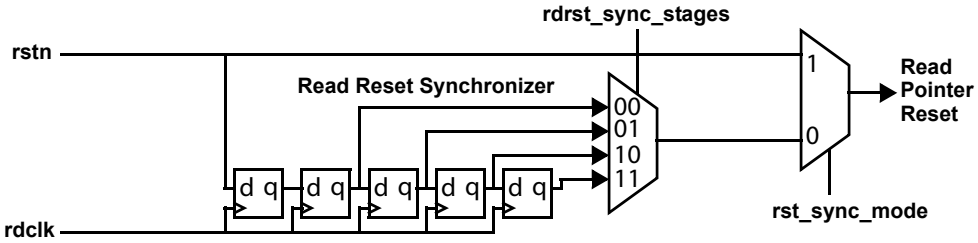
rdrst_sync_stages

The rdrst_sync_stages parameter defines the number of stages of registers used to synchronize the rstn input pin to the wrclk clock domain. The value of the rdrst_sync_stages parameter is only used if the rst_sync_mode is set to 1'b0. The mapping of the rdrst_sync_stages parameter value to the number of sychronization stages is defined in **Table 6-55: Mapping rdrst_sync_stages Parameter Settings to Synchronization Stage Depth**, where each stage corresponds to a register in the Read Reset Synchronizer shown in **Figure 6-36: Read Pointer Reset Input Selection Block Diagram**. For example, setting rdrst_sync_stages to 2'b00 configures the rstn synchronization circuit to have two back-to-back registers in the Write Reset Synchronizer. The default value of the rdrst_sync_stages parameter is 2'b00.

Table 6-55: *Mapping rdrst_sync_stages Parameter Settings to Synchronization Stage Depth*

rdrst_sync_stages	Read Reset Synchronization Stage Depth
2'b00	2
2'b01	3
2'b10	4
2'b11	5

Figure 6-36: *Read Pointer Reset Input Selection Block Diagram*



afull_offset

The afull_offset parameter defines the word depth at which the FIFO almost_full signal changes. The almost_full flag may be used to determine the number of blind writes to the FIFO that can be made without monitoring the full flag. For example, if the afull_offset parameter is set to 7'h04 and the almost_full flag is deasserted, the user is guaranteed that there are at least five empty locations in the FIFO. The user may write all five words without monitoring the full flag and be guaranteed that these words will be written to the FIFO and the write_err flag will not be asserted. The default value of the afull_offset parameter is 7'h04, corresponding to five or fewer available locations remain in the FIFO.

Table 6-56: *Condition to Assert almost_full Flag based on afull_offset Parameter Assignment*

Condition when almost_full flag is asserted	Condition when almost_full flag is deasserted
afull_offset or fewer empty locations remain in the FIFO.	There are at least (afull_offset +1) empty locations remaining in the FIFO.

aempty_offset

The aempty_offset parameter defines the word depth at which the FIFO almost_empty changes. The almost_empty flag may be used to determine the number of blind reads from the FIFO that can be performed without monitoring the empty flag. For example, if the aempty_offset parameter is set to 7'h04 and the almost_empty flag is deasseted, the user is guaranteed that there are at least five words in the FIFO. The user may read all five words without monitoring the empty flag and be guaranteed that these words will be read from the FIFO and the read_err flag will not be asserted. The default value of the aempty_offset parameter is 7'h04, corresponding to four or fewer words remaining in the FIFO.

Table 6-57: *Condition to Assert almost_empty Flag based on aempty_offset Parameter Assignment*

Condition when almost_empty flag is asserted	Condition when almost_empty flag is deasserted
aempty_offset or fewer words remain in the FIFO.	There are at least (aempty_offset +1) words in the FIFO.

Status Flags

Empty Flag

The Empty (empty) flag is asserted after the FIFO is reset or when all of the data has been read from the FIFO. The Empty flag is synchronous to the rdclk clock domain. Further attempts to read the FIFO when the Empty flag is asserted will be blocked, the Read Error (read_err) flag will be set in the following rdclk clock cycle, and the Read Pointer will remain unchanged.

Almost Empty Flag

The Almost Empty (almost_empty) flag is asserted when there are aempty_offset or fewer words remaining in the FIFO (See **Table 6-57: Condition to Assert almost_empty Flag based on aempty_offset Parameter Assignment**). The almost_empty flag may be used to determine the number of blind reads from the FIFO that can be performed without monitoring the empty flag. For example, if the aempty_offset parameter is set to 7'h04 and the almost_empty flag is deasserted, the user is guaranteed that there are at least five words in the FIFO. The user may read all five words without monitoring the empty flag and be guaranteed that these words will be read from the FIFO and the read_err flag will not be asserted. The Almost Empty flag is synchronous with the rdclk clock input.

Full Flag

The Full (full) flag is asserted when all of the available locations of the FIFO have been written. It is synchronous to the wrclk clock domain. Further attempts to write the FIFO when the Full flag is asserted will be blocked, the Write Error (write_err) flag will be set in the following wrclk clock cycle, and the Write Pointer will remain unchanged.

Almost Full Flag

The Almost Full (almost_full) flag is asserted when there are afull_offset or fewer available locations remaining in the FIFO. The almost_full flag may be used to determine the number

of blind writes to the FIFO that can be made without monitoring the full flag. For example, if the afull_offset parameter is set to 7'h04 and the almost_full flag is deasserted, the user is guaranteed that there are at least five empty locations in the FIFO. The user may write all five words without monitoring the full flag and be guaranteed that these words will be written to the FIFO and the write_err flag will not be asserted. The Almost Full flag is synchronous with the wrclk clock input.

Write Error Flag

The Write Error (write_err) flag is asserted in the following wrclk clock cycle when the user tries to write the FIFO while the Full Flag is high.

Read Error Flag

The Read Error (read_err) flag is asserted in the following rdclk clock cycle when the user tries to read the FIFO while the Empty Flag is high.

Flag Latency in Asynchronous Mode (ptr_sync_mode = 1'b0)

The empty, full, almost_empty, and almost_full flags are calculated based on comparisons between the FIFO write pointer and the FIFO read pointer. The write pointer is synchronous to the wrclk domain and the read pointer is synchronous to the rdclk domain.

For the case of an asynchronous FIFO, the wrclk and rdclk clocks reside in different clock domains.

Table 6-58: *FIFO Pointers and Status Flag Clock Domain Assignments*

Flag	Associated Clock Domain
FIFO Write Pointer	wrclk
FIFO Read Pointer	rdclk
empty flag	rdclk
almost_empty flag	rdclk
full flag	wrclk
almost_full flag	wrclk

Before flag calculations can be made, circuitry has to make sure that both pointers are in the same clock domain as the flag for which the calculation is done. The Write Pointer Synchronizer **Figure 6-33: Write Pointer Synchronizer Block Diagram** and the Read Pointer Synchronizer **Figure 6-34: Read Pointer Synchronizer Block Diagram** are used to transfer each of the pointers into the other clock domain. In order to synchronize a given pointer to the opposite clock domain, a series of registers, whose depth is determined by the wrptr_sync_stages and rdptr_sync_stages parameters, is used. The transfer of a pointer through these registers adds additional delay to the flag calculation. The versions of the pointers used for flag calculations are shown below in **Table 6-59: Pointers Used for FIFO Flag Calculations (ptr_sync_mode = 1'b0)**.

Table 6-59: *Pointers Used for FIFO Flag Calculations (ptr_sync_mode = 1'b0)*

Flag	Write Pointer Used for Flag Calculation	Read Pointer Used for Flag Calculation
empty flag	Synchronized Write Pointer	Read Pointer
almost_empty flag	Synchronized Write Pointer	Read Pointer
full flag	Write Pointer	Synchronized Read Pointer
almost_full flag	Write Pointer	Synchronized Read Pointer

For example, the empty flag is computed from the Synchronized Write Pointer and the Read Pointer. The write pointer incurs an additional delay of two to five rdclk cyles (set by the wrptr_sync_stages parameter) before it is used to calculate the empty flag. Therefore, the empty flag will not transition from the empty to non-empty state for a minimum of two rdclk cycles after the first write to the FIFO occurs. A similar delay occurs for the almost_empty flag as well. Likewise, for the full and almost_full flags, there are two to five wrclk cycles of delay in the actual FIFO status due to the synchronized read pointer. For an asyncronous FIFO, the calculation of the flags does not immediately reflect the state of the FIFO. While this behavior is normal for asynchronous FIFOs, it should be noted. **Table 6-60: Empty and Almost Empty Flag Latency in Terms of Read Clock Cycles(ptr_sync_mode =1'b0)** and **Table 6-61: Full and Almost Full Flag Latency in Terms of Write Clock Cycles(ptr_sync_mode = 1'b0)** show the latency for the FIFO flag calculations.

Table 6-60: *Empty and Almost Empty Flag Latency in Terms of Read Clock Cycles(ptr_sync_mode =1'b0)*

FIFO Status Flag	Read Clock Cycle Latency (rdclk cycles)	
	Flag Assertion	Flag Deassertion
empty flag	0	3 + rdptr_sync_stages
almost empty flag	0	3

Table 6-61: *Full and Almost Full Flag Latency in Terms of Write Clock Cycles(ptr_sync_mode = 1'b0)*

FIFO Status Flag	Write Clock Cycle Latency (wrclk cycles)	
	Flag Assertion	Flag Deassertion
full flag	0	3 + wrptr_sync_stages
almost full flag	0	3

Flag Latency in Synchronous Mode (ptr_sync_mode = 1'b1)

A synchronous FIFO has only a single clock, so there is no clock domain crossing required. A synchronous FIFO has the advantage that the flag calculations are immediate and precise.

FIFO Operational Modes

The FIFO macro supports both single clock synchronous (same clock connected to wrclk and rdclk inputs without any phase offset between the two clocks) and dual clock asynchronous (two unrelated clocks or two related clocks) modes of operation. For synchronous operation, both the wrclk and rdclk inputs must be connected to the same clock net. Phase offsets between the two clocks in synchronous mode is not allowed. For asynchronous mode, the user may connect the wrclk and rdclk inputs to two different clocks. The FIFO will treat the two clocks as if they are unrelated.

Asychronous FIFO Mode (ptr_sync_mode = 1'b0)

After a reset operation, or after the last word has been read from the FIFO, the FIFO will be in an empty state as indicated by a high level on the empty flag. When the FIFO is set to asynchronous mode (ptr_sync_mode = 1'b0), the output of the FIFO remains unchanged after the first write to a FIFO in the empty state. After the first write operation the empty flag will be deasserted indicating that there is data in the FIFO that may be read. The user must read the FIFO by setting the rden high at which time the first word written into the FIFO will be available at the FIFO outputs at the next rising edge of the rdclk input. Each subsequent read operation updates the FIFO outputs with the next stored data word if it is available (empty flag = false).

Synchronous FIFO Mode (ptr_sync_mode = 1'b1)

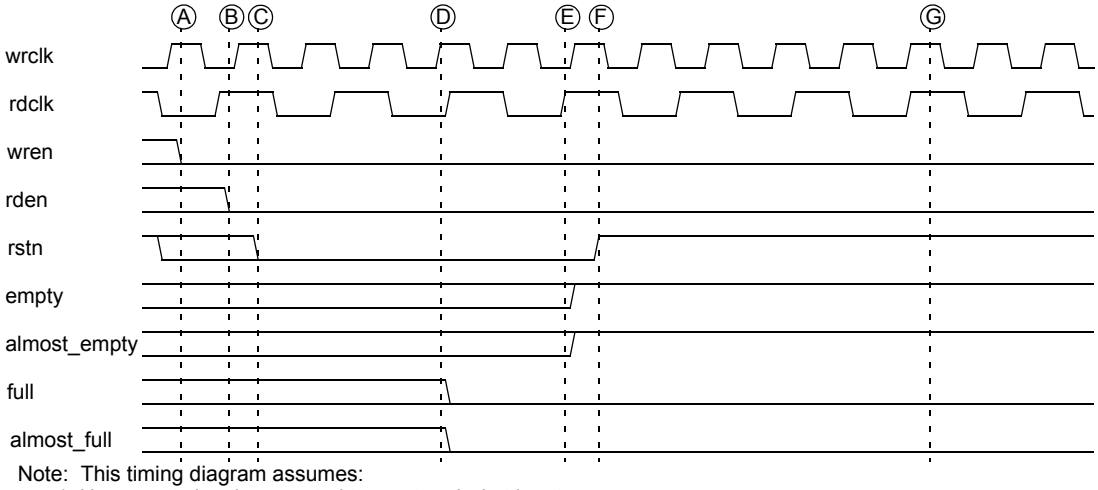
The synchronous FIFO standard mode has the advantage that there is no latency in the flag calculations, so the flags represent the exact state of the FIFO. For synchrous operation, both wrclk and rdclk must be tied to the same clock signal.

FIFO Operations

Asynchronous FIFO Mode Reset Operation

Two options are available to the user with regard to resetting the FIFO. For asychronous FIFO Reset, the user set rst_sync_mode to 1'b0. To reset the FIFO, the user will assert the rstn signal for a minimum of three clock cycles of the slower clock cycle between the wrclk and rdclk. Asserting the reset signal clears both the Write Pointer and Read Pointer, sets the empty and almost_empty flags, and clears the full and almost_full flags. The user may then release the rstn signal. The user should not attempt to read or write the FIFO while the rstn input is asserted or before three cycles after the deassertion of the rstn signal. **Figure 6-37: Asynchronous FIFO Mode Reset Timing Diagram** shows the timing for an asychronous reset.

Figure 6-37: *Asynchronous FIFO Mode Reset Timing Diagram*

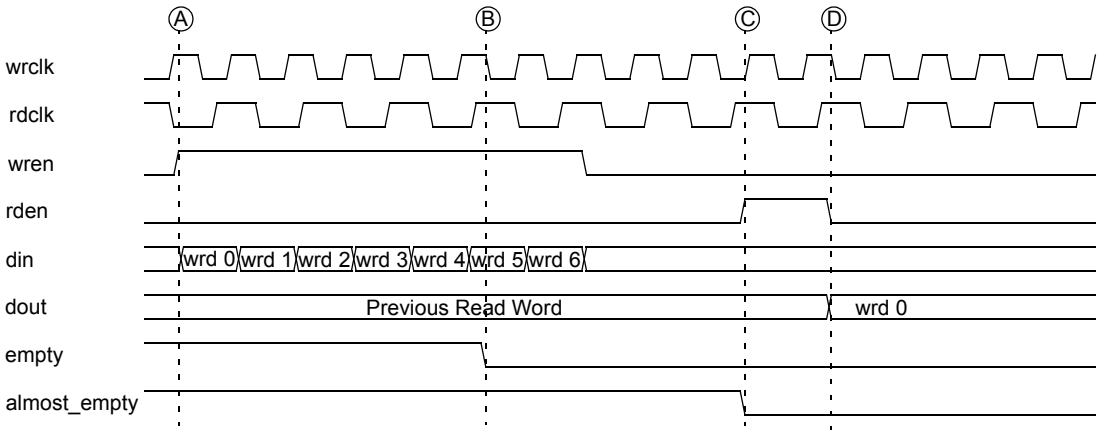


Note: This timing diagram assumes:
1. User reset signal connected to wrrst and rdrst inputs

Event Ⓐ: The user must disable the wren signal during the reset operation.
Event Ⓑ: The user must disable the rden signal during the reset operation.
Event Ⓒ: The user rstn reset signal is asserted.
Event Ⓓ: The full and almost_full flags are deasserted (rdrst_sync_stages + 2) active wrclk edges after the rstn input is asserted.
Event Ⓔ: The empty and almost_empty flags are asserted (wrrst_sync_stages + 2) active rdclk edges after the rstn input is asserted.
Event Ⓕ: The user rstn reset signal is deasserted after a minimum of:
max{(wrrst_sync_stages + 3) rdclk cycles, (rdrst_sync_stages + 3) wrclk cycles}
after the wren and rden inputs are disabled.
Event Ⓖ: The first FIFO write operation may begin:
max{(wrrst_sync_stages + 3) rdclk cycles, (rdrst_sync_stages + 3) wrclk cycles}
after the rstn signal is deasserted.

Writing an Empty Asynchronous FIFO (ptr_sync_mode = 1'b0)

Figure 6-38: *Writing an Empty Asynchronous FIFO (ptr_sync_mode = 1'b0)*

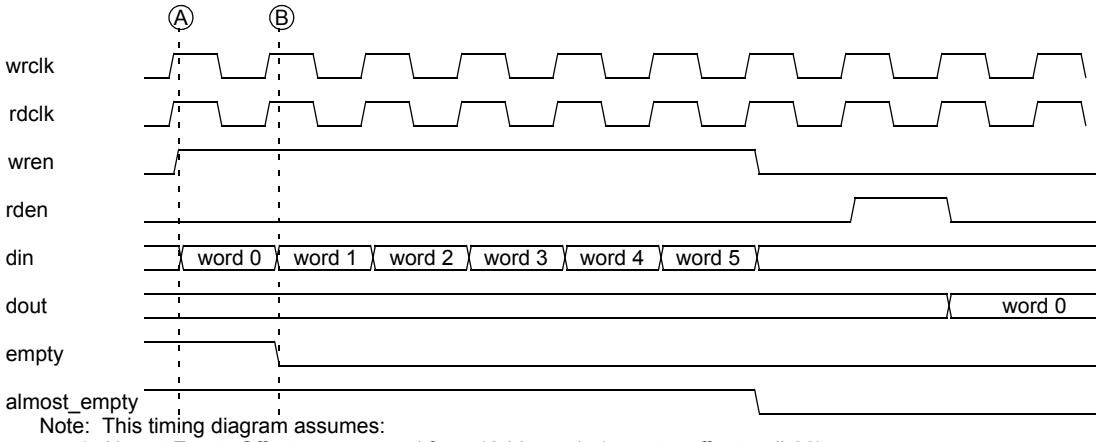


Note: This timing diagram assumes:
1. Almost Empty Offset programmed for 5 40-bit words (aempty_offset = 17'h00005)
2. wptr_sync_stages = 2'b00
3. ptrsync_mode = 1'b0

Event Ⓐ: Begin writing 7 words to the FIFO.
Event Ⓑ: empty flag goes inactive one wrclk cycle plus (wrptr_sync_stages + 3) rdclk cycles after wren becomes active.
Event Ⓒ: almost_empty flag goes inactive one wrclk cycle plus (wrptr_sync_stages + 3) rdclk cycles after wrd4 (6th word) is written into the FIFO.
Event Ⓓ: The first (wrd0) appears at the dout output.

Writing an Empty Synchronous FIFO (ptr_sync_mode = 1'b1)

Figure 6-39: *Writing an Empty Synchronous FIFO (ptr_sync_mode = 1'b1)*

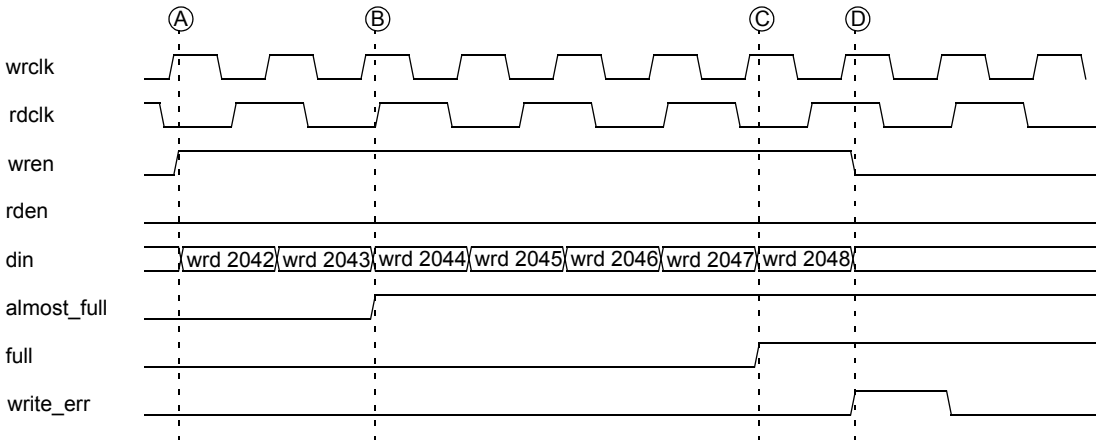


Note: This timing diagram assumes:
1. Almost Empty Offset programmed for 5 10-bit words (aempty_offset = 7'h06)
2. ptr_sync_mode = 1'b1

Event Ⓐ: Begin writing 6 words to the FIFO.
Event Ⓑ: empty flag goes inactive one wrclk cycle after wren becomes active.

Writing to an Almost Full FIFO

Figure 6-40: *Writing to an Almost Full FIFO*

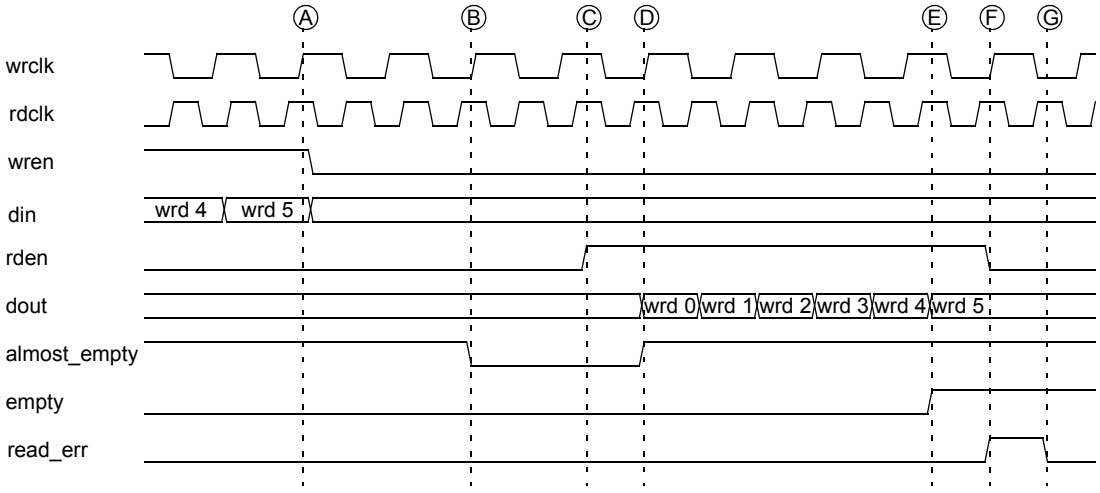


Note: This timing diagram assumes:
1. Almost Full Offset programmed for 5 10-bit words (aempty_offset = 7'h04)
2. wptr_sync_stages = 2'b00

Event Ⓐ: Finish writing 65 words to the FIFO.
Event Ⓑ: The almost_full flag is asserted.
Event Ⓒ: The full flag is asserted four write cycles after almost_full flag is asserted (assuming no reads).
Event Ⓓ: The write_err flag is asserted one wrclk cycle after attempting to write a full FIFO.

Reading from an Almost Empty FIFO

Figure 6-41: *Reading From an Almost Empty FIFO*



Note: This timing diagram assumes:
1. Almost Empty Offset programmed for 6 10-bit words (aempty_offset = 7'h05)
2. wptr_sync_stages = 2'b00

Event Ⓐ: Finish writing 6 words to the FIFO.
Event Ⓑ: The almost_empty flag is deasserted one wrclk plus (wrptr_sync_stages + 3) rdclk active clock edges after the sixth (wrd 5) word is presented at the din input with wren high.
Event Ⓒ: Begin to read the sixth word from the FIFO.
Event Ⓓ: The almost_empty flag is asserted the cyle after the first read request, when five words remain in the FIFO.
Event Ⓔ: The empty flag is asserted after the last (sixth) word is read from the FIFO. The rden signal remains high, attempting to read an empty FIFO.
Event Ⓕ: The read_err signal is asserted the cycle after the attempt to read an empty FIFO. The sixth (wrd 5) word remains at the dout output.

Chapter 7 – Multipliers

BMACC56

28 x 28 Multiplier / Accumulator

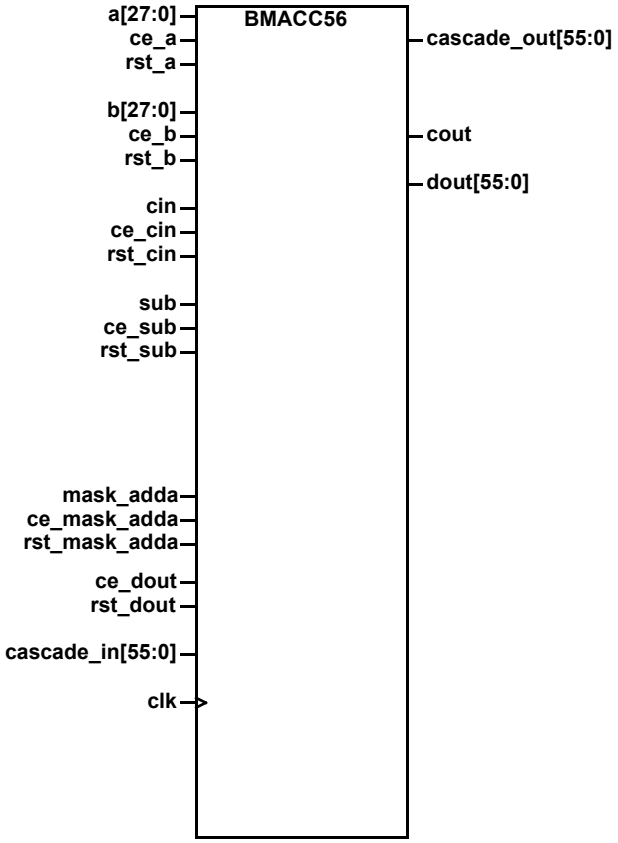


Figure 7-1: *Logic Symbol*

The Multiplier / Accumulator (BMACC56) block implements a signed 28x28 multiplier followed by an optional accumulator block. The multiplier produces a 56-bit adder result which is fed into the 56-bit accumulator.

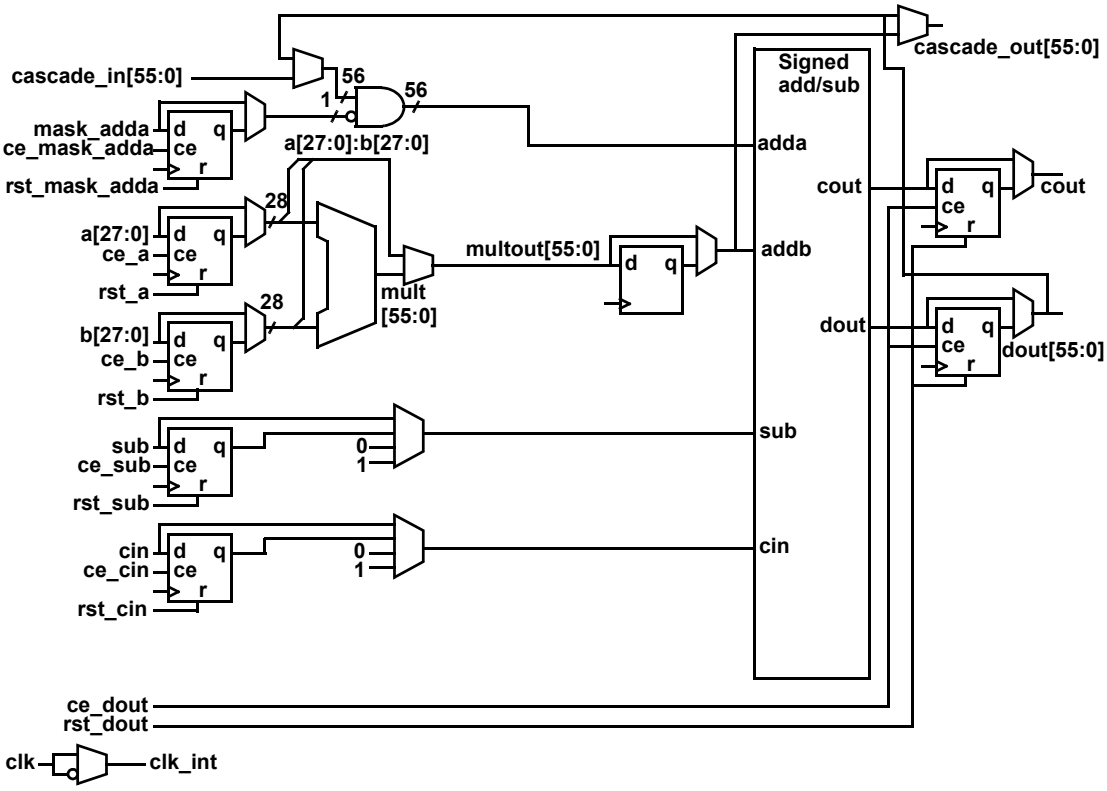


Figure 7-2: *BMACC56 Block Diagram*

Table 7-1: *I/O Count*

Inputs	Outputs
71	57

BMACC56 Pins

Table 7-2: *BMACC56 Pin Description*

Name	Type	Description
a[27:0]	input	**Data Input A.** Data Input A is a 28-bit two's complement signed input, where bit 27 is the most significant bit. In subtraction mode, Data Input A is the minuend.
b[27:0]	input	**Data Input B.** Data Input B is a 28-bit two's complement signed input, where bit 27 is the most significant bit. In subtraction mode, Data Input B is the subtrahend.
sub	input	**Subtract.** Setting the subtract input to one inverts the b input so that an a[27:0] - b[27:0] subtraction operation can be performed. Note that the cin input also has to be set to one during a subtract operation.
cin	input	**Carry In.** Carry input to the adder/subtractor.
mask_adda	input	**Adda Mask.** Setting mask_adda high clears the adda input and allows the add/sub block to pass the addb + cin value to the add/sub output. Setting mask_adda low enables the adda input to the add/sub block to be driven by either the dout output (sel_cascade_in = 1'b0) or the cascade_input from the previous BMACC56 block (sel_cascade_in = 1'b1).
ce_a	input	**Data Input A Input Register Clock Enable** (active-high). Setting ce_a to one allows Data Input A to be clocked into the input register when the rst_a signal is one.
ce_b	input	**Data Input B Input Register Clock Enable** (active-high). Setting ce_b to one allows Data Input B to be clocked into the input register when the rst_b signal is one.
ce_sub	input	**Subtract Input Register Clock Enable** (active-high). Setting ce_sub to one allows the sub input to be clocked into the input register when the rst_sub signal is one.
ce_cin	input	**Carry In Input Register Clock Enable** (active-high). Setting ce_cin to one allows the cin input to be clock into the cin input register. Setting ce_cind to zero allows the cin input register to hold its value at the next active edge of the clock.
ce_mask_adda	input	**Adda Mask Input Register Clock Enable** (active-high). Setting ce_mask_adda to one allows the mask_adda input to be clocked into the input register. Setting ce_mask_adda to zero allows the mask input register to hold its value at the next active edge of the clock.
ce_dout	input	**Data Out Output Register Clock Enable** (active-high). Setting ce_dout to one allows dout to be clocked into the Data Out Output Register and cout to be clocked into the Carry Out Output Register. Setting ce_dout to zero allows the Data Out Output Register to hold its value at the next active edge of the clock.
rst_a	input	**Data Input A Register Reset** (active-low). Asserting input rst_a performs a synchronous reset of the Data Input a register upon the next active edge of the clock, and will set the register to the value defined by the rst_value_a parameter.

Name	Type	Description
rst_b	input	**Data Input B Register Reset** (active-low). Asserting input rst_b performs a synchronous reset of the Data Input b register upon the next active edge of the clock, and will set the register to the value defined by the rst_value_b parameter.
rst_sub	input	**Subtract Input Register Reset** (active-low). Asserting input rst_sub performs a synchronous reset of the Subtract Input Register upon the next active edge of the clock, and will set the register to the value defined by the rst_level_sub parameter.
rst_cin	input	**Carry In Input Register Reset** (active-low). Asserting input rst_cin performs a synchronous reset of the Carry In Input Register upon the next active edge of the clock, and will set the register to the value defined by the rst_level_cin parameter.
rst_mask_adda	input	**Adda Mask Input Register Reset** (active-low). Asserting input rst_mask_adda performs a synchronous reset of the Adda Mask Input Register upon the next active edge of the clock, and will set the register to the value defined by the rst_level_mask_adda parameter.
rst_dout	input	**Dout Output Register Reset** (active-low). Asserting input rst_dout performs a synchronous reset of the Dout and Cout output registers upon the next active edge of the clock, and will set the dout register to the value defined by the reset_dout parameter. and will set the cout register to the value defined by the rst_level_cout parameter.
cascade_in[55:0]	input	**Cascade In.** The cascade input allows the 56-bit cascade_out output of the previous BMACC56 block to be connected to the adda input of the add/sub block. The cascade_out to cascade_in flows from the bottom of the Speedster22i device to the top.
clk	input	**Clock.** Data is clocked into the input and output registers at the active edge of the clock input. The active edge of the clock input is selected by setting the clock_edge parameter to the appropriate value.
dout[55:0]	output	**Data Out.** The dout[55:0] output is the 56-bit signed two's complement output, where bit 55 is the most significant bit.
cout	output	**Carry Out.** The cout bit is set high if a carry was generated out of the add/sub block.
cascade_out[55:0]	output	**Cascade Out.** Cascade_out may be fed to the next BMACC56 blocks adda input. This can be used to help in the formation of sum-of-products of multiple BMACC56 blocks when the sel_cascade_out parameter is set to zero.

Parameters

Table 7-3: *BMACC56 Parameters*

Parameter	Defined Values	Default Value
init_a	28-bit hexadecimal value	28'h0
init_b	28-bit hexadecimal value	28'h0
init_sub	1'b0,1'b1	1'b0
init_cin	1'b0,1'b1	1'b0
init_mask_adda	1'b0,1'b1	1'b0
init_dout	56-bit hexadecimal value	56'h0
init_cout	1'b0,1'b1	1'b0
rst_value_a	28-bit hexadecimal value	28'h0
rst_value_b	28-bit hexadecimal value	28'h0
rst_value_sub	1'b0,1'b1	1'b0
rst_value_cin	1'b0,1'b1	1'b0
rst_value_mask_adda	1'b0,1'b1	1'b0
rst_value_dout	56-bit hexadecimal value	56'h0
rst_value_cout	1'b0,1'b1	1'b0
regce_priority_a	"rstreg","regce"	"regce"
regce_priority_b	"rstreg","regce"	"regce"
regce_priority_sub	"rstreg","regce"	"regce"
regce_priority_cin	"rstreg","regce"	"regce"
regce_priority_mask_adda	"rstreg","regce"	"regce"
regce_priority_dout	"rstreg","regce"	"regce"
reg_a	1'b0,1'b1	1'b0
reg_b	1'b0,1'b1	1'b0
reg_addb	1'b0,1'b1	1'b0
reg_mask_adda	1'b0,1'b1	1'b0
reg_dout	1'b0,1'b1	1'b0
reg_cout	1'b0,1'b1	1'b0
sel_cascade_in	1'b0,1'b1	1'b0
sel_cascade_out	1'b0,1'b1	1'b0
sel_sub	2'b00, 2'b01, 2'b10, 2'b11	2'00
sel_cin	2'b00, 2'b01, 2'b10, 2'b11	2'00
mult_bypass	1'b0,1'b1	1'b0
clock_edge	1'b0,1'b1	1'b0

init_a

The init_a parameter defines the power-up default value of the Data Input A Input Register. The init_a parameter defaults to the value 28'h0.

init_b

The init_b parameter defines the power-up default value of the Data Input B Input Register. The init_b parameter defaults to the value 28'h0.

init_sub

The init_sub parameter defines the power-up default value of the Subtract Input Register. The init_sub parameter defaults to the value 1'b0.

init_cin

The init_cin parameter defines the power-up default value of the Carry In Input Register. The init_cin parameter defaults to the value 1'b0.

init_mask_adda

The init_mask_adda parameter defines the power-up default value of the Adda Mask Input Register. The init_mask_adda parameter defaults to the value 1'b0.

init_dout

The init_dout parameter defines the power-up default value of the Data Out Output Register. The init_dout parameter defaults to the value 56'h0.

init_cout

The init_cout parameter defines the power-up default value of the Carry Out Output Register. The init_cout parameter defaults to the value 1'b0.

rst_value_a

The rst_value_a parameter defines the value assigned to the Data Inputa A Input Register when the rst_a input is asserted and there is active edge of the clock. The rst_value_a parameter defaults to the value 28'h0.

rst_value_b

The rst_value_b parameter defines the value assigned to the Data Inputa B Input Register when the rst_b input is asserted and there is active edge of the clock. The rst_value_b parameter defaults to the value 28'h0.

rst_value_sub

The rst_value_sub parameter defines the value assigned to the Subtract Input Register when the rst_sub input is asserted and there is active edge of the clock. The rst_value_sub parameter defaults to the value 1'b0.

rst_value_cin

The rst_value_cin parameter defines the value assigned to the Carry In Input Register when the rst_cin input is asserted and there is active edge of the clock. The rst_value_cin parameter defaults to the value 1'b0.

rst_value_mask_adda

The rst_value_mask_adda parameter defines the value assigned to the Adda Mask Input Register when the rst_mask_adda input is asserted and there is active edge of the clock. The rst_value_mask_adda parameter defaults to the value 1'b0.

rst_value_dout

The rst_value_dout parameter defines the value assigned to the Data Out Output Register when the rst_dout input is asserted and there is active edge of the clock. The rst_value_dout parameter defaults to the value 56'h0.

rst_value_cout

The rst_value_cout parameter defines the value assigned to the Carry Out Output Register when the rst_dout input is asserted and there is active edge of the clock. The rst_value_cout parameter defaults to the value 1'b0.

regce_priority_a

The regce_priority_a parameter defines the priority of the ce_a clock enable input relative to the rst_a reset input during an assertion of the rst_a reset input on the Data Input A Input Register. Setting regce_priority_a to "rstreg" allows the Data Input A Input Register to be set/reset at the next active edge of the clock without requiring the ce_a clock enable input to be active. Setting regce_priority_a to "regce" requires that the ce_a clock enable input is high for the reset operation to occur at the next active edge of the clock. The default value of the regce_priority_a parameter is "regce".

regce_priority_b

The regce_priority_b parameter defines the priority of the ce_b clock enable input relative to the rst_b reset input during an assertion of the rst_b reset input on the Data Input B Input Register. Setting regce_priority_b to "rstreg" allows the Data Input B Input Register to be set/reset at the next active edge of the clock without requiring the ce_b clock enable input to be active. Setting regce_priority_b to "regce" requires that the ce_b clock enable input is high for the reset operation to occur at the next active edge of the clock. The default value of the regce_priority_b parameter is "regce".

regce_priority_sub

The regce_priority_sub parameter defines the priority of the ce_sub clock enable input relative to the rst_sub reset input during an assertion of the rst_sub reset input on the Sub Input Register. Setting regce_priority_sub to "rstreg" allows the Sub Input Register to be set/reset at the next active edge of the clock without requiring the ce_sub clock enable input to be active. Setting regce_priority_sub to "regce" requires that the ce_sub clock enable input is high for the reset operation to occur at the next active edge of the clock. The default value of the regce_priority_sub parameter is "regce".

regce_priority_cin

The regce_priority_cin parameter defines the priority of the ce_cin clock enable input relative to the rst_cin reset input during an assertion of the rst_cin reset input on the Cin Input Register. Setting regce_priority_cin to "rstreg" allows the Cin Input Register to be set/reset at the next active edge of the clock without requiring the ce_cin clock enable input to be active. Setting regce_priority_cin to "regce" requires that the ce_cin clock enable input is high for the reset operation to occur at the next active edge of the clock. The default value of the regce_priority_cin parameter is "regce".

regce_priority_dout

The regce_priority_dout parameter defines the priority of the ce_dout clock enable input relative to the rst_dout reset input during an assertion of the rst_dout reset input on the Dout Output Register and Cout Output Register. Setting regce_priority_dout to "rstreg" allows the Dout Output Register and Cout Output Register to be set/reset at the next active edge of the clock without requiring the ce_dout clock enable input to be active. Setting regce_priority_dout to "regce" requires that the ce_dout clock enable input is high for the reset operation to occur at the next active edge of the clock. The default value of the regce_priority_dout parameter is "regce".

reg_a

The reg_a parameter defines if the Data Input A Input Register is used or bypassed. Setting reg_a to 1'b0 bypasses the register while setting reg_a to 1'b1 enables the register. The reg_a parameter defaults to the value 1'b0.

reg_b

The reg_b parameter defines if the Data Input B Input Register is used or bypassed. Setting reg_b to 1'b0 bypasses the register while setting reg_b to 1'b1 enables the register. The reg_b parameter defaults to the value 1'b0.

reg_addb

The reg_addb parameter defines if the Add/Sub Addb Input Register is used or bypassed. Setting reg_addb to 1'b0 bypasses the register while setting reg_addb to 1'b1 enables the register. The reg_addb parameter defaults to the value 1'b0.

reg_mask_adda

The reg_mask_adda parameter defines if the Adda Mask Input Register is used or bypassed. Setting reg_mask_adda to 1'b0 bypasses the register while setting reg_mask_adda to 1'b1 enables the register. The reg_mask_adda parameter defaults to the value 1'b0.

reg_dout

The reg_dout parameter defines if the Data Out Output Register is used or bypassed. Setting reg_dout to 1'b0 bypasses the register while setting reg_dout to 1'b1 enables the register. The reg_dout parameter defaults to the value 1'b0. Note that for the Speedster22i HP devices that reg_dout and reg_cout must be set to the same value.

reg_cout

The reg_cout parameter defines if the Carry Out Output Register is used or bypassed. Setting reg_cout to 1'b0 bypasses the register while setting reg_cout to 1'b1 enables the register. The reg_cout parameter defaults to the value 1'b0.

sel_cascade_in

The sel_cascade_in parameter defines what is routed to the input of the adda input to the add/sub block. Setting sel_cascade_in to 1'b0 selects the the dout[55:0] output while setting sel_cascade_in to 1'b1 selects the cascade_in[55:0] input. The sel_cascade_in parameter defaults to the value 1'b0.

sel_cascade_out

The sel_cascade_out parameter defines what is routed to the cascade_out output. Setting sel_cascade_out to 1'b0 selects the the dout[55:0] output while setting sel_cascade_out to 1'b1 selects the output of the conditionally registered output of the multiplier output multiplexer. The sel_cascade_out parameter defaults to the value 1'b0.

sel_cin

The sel_cin parameter defines what is routed to the cin input of the add/sub block. The cin input may be selected from the registered or non-registered cin input, or forcing it to a constnat value of 0 or 1.The sel_cin parameter defaults to the value 2'b00.

Table 7-4: *Add/Sub Block Carry Input Assignment*

sel_cin	Add/Sub Block Cin Assignment
2'b00	Select the cin input to drive the add/sub cin input.
2'b01	Select the output of the Carry In Input Register to drive the add/sub cin input.
2'b10	Select 1'b0 to drive the add/sub cin input.
2'b11	Select 1'b1 to drive the add/sub cin input.

sel_sub

The sel_sub parameter defines what is routed to the sub input of the add/sub block. The sub input may be selected from the registered or non-registered sub input, or force it to a constant value of 0 or 1.The sel_sub parameter defaults to the value 2'b00.

Table 7-5: *Add/Sub Block Sub Input Assignment*

sel_sub	Add/Sub Block Cin Assignment
2'b00	Select the sub input to drive the add/sub sub input.
2'b01	Select the output of the Sub Input Register to drive the add/sub sub input.
2'b10	Select 1'b0 to drive the add/sub sub input.
2'b11	Select 1'b1 to drive the add/sub sub input.

mult_bypass

The mult_bypass parameter determines whether the 28x28 multiplier is used or bypassed. Setting mult_bypass to 1'b0 enables the multiplier while setting mult_bypass to 1'b1 bypasses the multiplier and routes the concatenated a[27:0],b[27:0] data the the multout[55:0] output. The mult_bypass parameter defaults to the value of 1'b0.

clock_edge

The clock_edge parameter defines which edge of the Clock input is used by all of the input and output registers. Setting clock_edge to 1'b0 selects the rising edge of the clock as the active edge while setting clock_edge to 1'b1 selects the falling edge of the clock as the active edge. The clock_edge parameter defaults to the value 1'b0.

BMACC56 Verilog Instantiation Template

```
BMACC56 #(
   .init_a(28'h0),
   .init_b(28'h0),
   .init_sub(1'b0),
   .init_cin(1'b0),
   .init_mask_adda(1'b0),
   .init_dout(56'h0),
   .init_cout(1'h0),
   .rst_value_a(28'h0),
   .rst_value_b(28'h0),
   .rst_value_sub(1'b0),
   .rst_value_cin(1'b0),
   .rst_value_mask_adda(1'b0),
   .rst_value_dout(56'h0),
   .rst_value_cout(1'h0),
   .regce_priority_a("regce"),
   .regce_priority_b("regce"),
   .regce_priority_sub("regce"),
   .regce_priority_cin("regce"),
   .regce_priority_mask_adda("regce"),
   .regce_priority_dout("regce"),
   .reg_a(1'b0),
   .reg_b(1'b0),
   .reg_addb(1'b0),
   .reg_mask_adda(1'b0),
   .reg_dout(1'b0),
   .reg_cout(1'b0),
   .sel_cascade_in(1'b0),
   .sel_cascade_out(1'b0),
   .sel_cin(2'b00),
   .sel_sub(2'b00),
   .mult_bypass(1'b0),
   .clock_edge(1'b0));
instance_name(
  .a(user_a),
  .b(user_b),
  .sub(user_sub),
  .cin(user_cin),
  .mask_adda(user_mask_adda),
  .ce_a(user_ce_a),
  .ce_b(user_ce_b),
  .ce_sub(user_ce_sub),
  .ce_cin(user_ce_cin),
```

```
  .ce_mask_adda(user_ce_mask_adda),
  .ce_dout(user_ce_dout),
  .rst_a(user_rst_a),
  .rst_b(user_rst_b),
  .rst_sub(user_rst_sub),
  .rst_cin(user_rst_cin),
  .rst_mask_adda(user_rst_mask_adda),
  .rst_dout(user_rst_dout),
  .cascade_in(user_cascade_in),
  .clk(user_clk),
  .dout(user_dout),
  .cout(user_cout),
  .cascade_out(user_cascade_out));
```

BMACC56 VHDL Instantiation Template

```
------------ ACHRONIX LIBRARY ------------
library speedster22i;
use speedster22i.components.all;
------------ DONE ACHRONIX LIBRARY ---------

-- Component Instantiation
BMACC56_instance_name : BMACC56
  generic map (init_a => X"0000000";
               init_b => X"0000000";
               init_sub => '0';
               init_cin => '0';
               init_mask_adda => '0';
               init_dout => X"00000000000000";
               init_cout => '0';
               rst_value_a => X"0000000";
               rst_value_b => X"0000000";
               rst_value_sub => '0';
               rst_value_cin => '0';
               rst_value_mask_adda => '0';
               rst_value_dout => X"00000000000000";
               rst_value_cout => '0';
               regce_priority_a => "regce";
               regce_priority_b => "regce";
               regce_priority_sub => "regce";
               regce_priority_cin => "regce";
               regce_priority_mask_adda => "regce";
               regce_priority_dout => "regce";
               reg_a => '0';
               reg_b => '0';
```

```
                reg_addb => '0';
                reg_mask_adda => '0';
                reg_dout => '0';
                reg_cout => '0';
                sel_cascade_in => '0';
                sel_cascade_out => '0';
                sel_sub => "00";
                sel_cin => "00";
                mult_bypass => '0';
                clock_edge => '0')
    port map(a => user_a,
             b => user_b,
             sub => user_sub,
             cin => user_cin,
             mask_adda => user_mask_adda,
             ce_a => user_ce_a,
             ce_b => user_ce_b,
             ce_sub => user_ce_sub,
             ce_cin => user_ce_cin,
             ce_mask_adda => user_ce_mask_adda,
             ce_dout => user_ce_dout,
             rst_a => user_rst_a,
             rst_b => user_rst_b,
             rst_sub => user_rst_sub,
             rst_cin => user_rst_cin,
             rst_mask_adda => user_rst_mask_adda,
             rst_dout => user_rst_dout,
             cascade_in => user_cascade_in,
             clk => user_clk,
             dout => user_dout,
             cout => user_cout,
             cascade_out => user_cascade_out);
```

BMULT28X28

28 × 28 Signed Multiplier

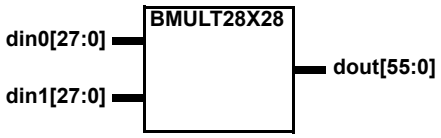


Figure 7-3: *Logic Symbol*

BMULT28X28 implements a signed 28 × 28 multiplier that multiplies two signed (two's complement) 28-bit inputs for produce a 56-bit signed product.

Table 7-6: *Pin Description*

Name	Type	Description
din0[27:0], din1[27:0]	input	**Signed (two's complement) 28-bit multiplier inputs.** Bit 0 is the LSB.
dout[55:0]	output	**Signed (two's complement) 56-bit product.** The value on dout is the product of din0[27:0] and din1[27:0]. dout[0] is the LSB.

Verilog Instantiation Template

```
BMULT28X28 instance_name
  (.dout(user_out[55:0]),.din0(user_in1[27:0]),.din1(user_in0[27:0]));
```

VHDL Instantiation Template

```
------------ ACHRONIX LIBRARY ------------
library speedster22i;
use speedster22i.components.all;
------------ DONE ACHRONIX LIBRARY ---------

-- Component Instantiation
BMULT28X28_instance_name : BMULT28X28
  port map (dout => user_out,
            din0 => user_in0,
            din1 => user_in1);
```

Chapter 8 – Special Functions

ACX_DESERIALIZE (Speedster22iHP Only)

1:N Serial-to-Parallel Converter

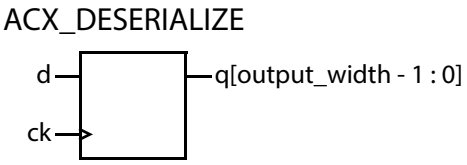


Figure 8-1: *Logic Symbol*

ACX_DESERIALIZE implements an 1:N serial-to-parallel conversion of the data input, where N is specified by the output_width parameter. The parallel output stream, q, is output at N times slower than the frequency of clk. The deserialized data is placed into the parallel output starting with the least significant bit and proceeding to the most significant bit. ACX_DESERIALIZE may be used to reduced the rate at which data is driven off the device if the internal processing rate exceeds the frequency rating of the device I/Os. This block may be used in conjunction with ACX_SERIALIZE to perform the initial serialization process.

Table 8-1: *Pin Description*

Name	Type	Description
d	input	**Data input.**
clk	input	**Clock.**
q[output_width –1 : 0]	output	**Parallel data output.** The value on the q output is filled starting with the LSB and proceeding to the MSB. The output data division rate must be specified by the output_width parameter.

Table 8-2: *Parameters*

Parameter	Defined Values	Default Value
output_width	positive integers	4

Verilog Instantiation Template

```
ACX_DESERIALIZE #(.output_width(4))
        instance_name(.q(user_q[output_width -1 : 0]),
                      .d(user_d),
                      .clk(user_clk));
```

VHDL Instantiation Template

```
------------- ACHRONIX LIBRARY ------------
library speedster22i;
use speedster22i.components.all;
------------- DONE ACHRONIX LIBRARY ---------

-- Component Instantiation
ACX_DESERIALIZE_instance_name : ACX_DESERIALIZE
  generic map (output_width => 4)
  port map (q => user_q,
            d => user_d,
            clk => user_clk);
```

ACX_SERIALIZE (Speedster22iHP Only)

N:1 Parallel-to-Serial Converter

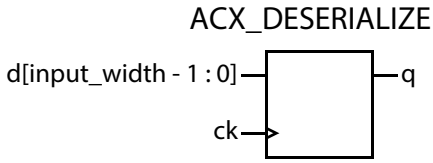


Figure 8-2: *Logic Symbol*

ACX_SERIALIZE implements an N:1 parallel-to-serial conversion of the data input, where N is specified by the input_width parameter. The serial output stream, q, is output at N times faster than the frequency of clk. The serialization is performed starting from the least significant bit and proceeding to the most significant bit. ACX_SERIALIZE may be used to rate multiply input data for higher processing rates inside the chip than is allowed at the device pads. This block may be used in conjunction with ACX_DESERIALIZE to perform a deserialization of data before it is driven off chip at lower rates.

Note: *To make output q N-times faster than the frequency of clk, the user needs to drive also ACX_SERIALIZE block clock port at N-times the frequency of clk. To create an N-times faster clock of the input frequency clk, user can instantiate ACX_VPLL macro.*

Table 8-3: *Pin Description*

Name	Type	Description
d[input_width –1 : 0]	input	**Data inputs.**
clk	input	**Clock.**
q	output	**Data output.** The value on the q output is the LSB to MSB serialization of the parallel input data d. The output data conversion rate must be specified by the input_width parameter.

Table 8-4: *Parameters*

Parameter	Defined Values	Default Value
input_width	positive integers	4

Verilog Instantiation Template

```
ACX_SERIALIZE #(.input_width(4))
        instance_name(.q(user_q),
                      .d(user_d[input_width -1 : 0]),
                      .clk(N-times_faster_of_user_clk));
```

VHDL Instantiation Template

```
------------ ACHRONIX LIBRARY ------------
```

```
library speedster22i;
use speedster22i.components.all;
------------- DONE ACHRONIX LIBRARY ---------

-- Component Instantiation
ACX_SERIALIZE_instance_name : ACX_SERIALIZE
  generic map (input_width => 4)
  port map (q => user_q,
            d => user_d,
            clk => user_clk);
```

Chapter 9 – PLL/DLL Clock Generators

ACX_CLKGEN

Phase-Locked Loop Clock Generator

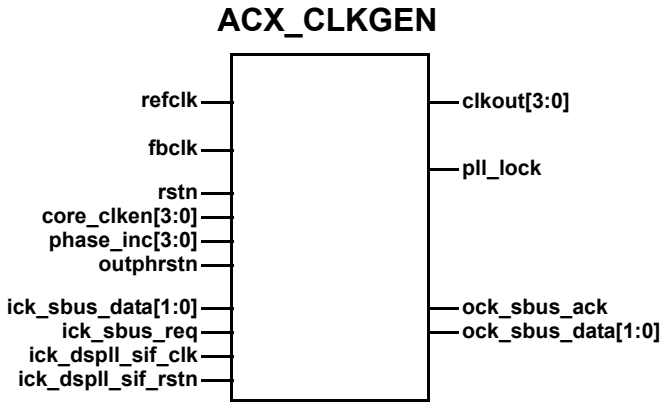


Figure 9-1: *ACX_CLKGEN Logic Symbol*

The reference clock, refclk, is divided by the reference clock divider (6-bit: 1 to 36) before it is sent to the Phase Frequency Detector (PFD). The PLL charge pump and loop filter control the VCO to generate full speed clock. The VCO generates 8 equally separated phases (45 degree separation between adjacent phases). One of the eight phase outputs is sent to the feedback divider through a multiplexer to allow the PLL running in short loop operation without de-skew. All 8-phases are sent to 4 phase rotators such that the outputs of each phase rotator may be independently selectable from any of the 8 phases. Each of the 4 phase rotator outputs can be shifted dynamically by 1/8th of the VCO period in phase at a time using the phase_inc inputs. All the 4 phases can be reset to the original phase when the outphrstn input is asserted. The 4 independently selectable phases each go through an independent output divider (6-bit: 1 to 63) before being sent to the Output Synthesizer block. One of the 4 output clocks, after going through the clock distribution tree, may be sent to the feedback divider for de-skew functionality. The feedback divider has two modes of operations: one is integer mode which offers a range from 2 to 66, while the other is fractional mode which offers a range from 8 to 66. The fractional mode has 16 bits of resolution. **Table 9-2** shows the block diagram of the ACX_CLKGEN block. The ACX_CLKGEN block may be controlled from the pins and parameter settings or optionally from the Control Status Registers (CSR) through the Serial Control Bus.

ACX_CLKGEN Pins

Table 9-1: *Ports*

Name	Type	Description
refclk	input	**Reference Clock.** The reference clock, which is optionally divided by the Reference Divider, is fed into the Phase Frequency Detector. The input to the Phase Frequency Detector must be in the range of 66 MHz to 400 MHz.
fbclk	input	**Feedback Clock.** The user may connect the Feedback Clock to one of the generated clkout[3:0] outputs so that the PLL may compensate for the delay of the clock network.
rstn	input	**PLL Reset.** Setting the rstn input low will reset the PLL. After returning the rstn input to a high level, the PLL should lock within 500 refclk (after the Reference Divider) clock periods if the PLL is used in integer divider mode and 1000 clock periods if the PLL is used in fractional divider mode. The user must hold rstn low for at least one cycle of the clock frequency input into the Phase Frequency Detector (after the Reference Divider).
outphrstn	input	**Output Phase Reset.** Setting the rstn input low will reset the Phase Rotator to its initial startup value.
core_clken[3:0]	input	**Clock Output Enable.** The core_clken[3:0] controls if the clock outputs of the PLL are being driven. Setting the clock enable bit to one enables the clock output of the corresponding bit position.
phase_inc[3:0]	input	**Clock Phase Increment.** A low to high transition of the phase_inc input increments the phase adjustment of the addressed Phase Rotator by 1/8th of the VCO frequency. The high period of the phase_inc signal must be at least at half of the output frequency.
clkout[3:0]	output	**Clock Output.**
pll_lock	output	**PLL Lock Status.** A high level on the pll_lock signal indicates that the PLL is locked.
ick_dspll_sif_clk	input	**Serial Control Bus Clock** (optional).
ick_dspll_sif_rstn	input	**Serial Control Bus Reset** (optional).
ick_sbus_data	input	**Serial Control Bus Data Input** (optional).
ick_sbus_req	input	**Serial Control Bus Request** (optional).
ock_sbus_ack	output	**Serial Control Bus Acknowledge** (optional).
ock_sbus_data	output	**Serial Control Bus Data Output** (optional).

Parameters

Table 9-2: *Parameters*

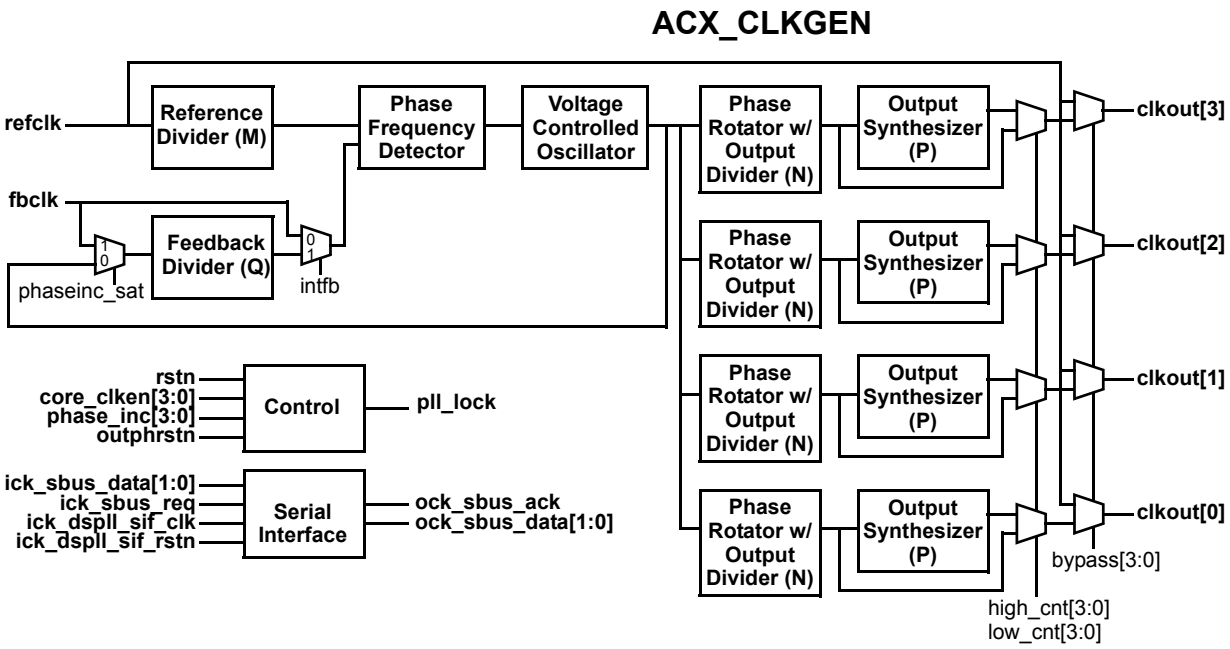
Parameter	Description	Defined Values	Default Value
clkdiv	Reference Divider value.	6'h01 - 6'h24	6'h1
intfb	Internal Feedback Enable. 0: Disable internal feedback. 1: Enables (internal) feedback divisor.	1'b0,1'b1	1'b0
phaseinc_sat	Mixed Feedback Mode. Not supported if intfb=0. 0: Pure internal feedback mode (if intfb = 1), Pure external feedback mode (if intfb = 0) 1: Mixed feedback mode (if intfb = 1).	1'b0,1'b1	1'b0
clkmult	Feedback Divider Divisor Value.	Integer mode:8'h02-8'h42, Fractional mode: 8'h8-8'h42	8'h0
synthmode	Feedback Divider Mode. 0: Integer mode. 1: Fractional mode. Uses frac_div_ctrl	1'b0,1'b1	1'b0
frac_div_ctrl	Feedback Divider Fractional Portion. (requires synth-mode = 1). In fractional mode, frac_div_ctrl is the 16-bit numerator which is divided by 65536. The resultant fraction is then used to scale the Feedback Divider value.	16'h0000-16'hFFFF	16'h0000
rst_short_long	Unused.	1'b0,1'b1	1'b0
clkouten_mode	Clock Enable Mode. 0: Enable clock outputs with the clken_out0 - clken_out3 parameters. 1: Enable clock outputs with the core_clken[3:0] inputs.	1'b0,1'b1	1'b0
pll_user_reset_en	PLL User Reset Enable. 0: Disable PLL rstn input. 1: Enable PLL rstn input.	1'b0,1'b1	1'b0
pll_user_outrst_en	PLL User Output Phase Reset Enable. 0: Disable PLL outphrstn input. 1: Enable PLL outphrstn input.	1'b0,1'b1	1'b0
pll_user_csrrst_en	PLL User Reset Enable for Serial Control Bus block. 0: Disable PLL ick_dspll_sif_rstn input. 1: Enable PLL ick_dspll_sif_rstn input.	1'b0,1'b1	1'b0
bypass0	Clkout[0] Bypass. 0: clkout[0] driven by PLL output. 1: clkout[0] driven by refclk input.	1'b0,1'b1	1'b0
bypass1	Clkout[1] Bypass. 0: clkout[1] driven by PLL output. 1: clkout[1] driven by refclk input.	1'b0,1'b1	1'b0

Parameter	Description	Defined Values	Default Value
bypass2	Clkout[2] Bypass. 0: clkout[2] driven by PLL output. 1: clkout[2] driven by refclk input.	1'b0,1'b1	1'b0
bypass3	Clkout[3] Bypass. 0: clkout[3] driven by PLL output. 1: clkout[3] driven by refclk input.	1'b0,1'b1	1'b0
outdiv0	Clkout[0] Rotator / Output Divider Divisor.	6'h1-6'h3F	6'h4
outdiv1	Clkout[1] Rotator / Output Divider Divisor.	6'h1-6'h3F	6'h4
outdiv2	Clkout[2] Rotator / Output Divider Divisor.	6'h1-6'h3F	6'h4
outdiv3	Clkout[3] Rotator / Output Divider Divisor.	6'h1-6'h3F	6'h4
en_phase0	Clkout[0] Phase Shift Enable. 0: Disable phase_inc[0] input. 1: Enable phase_inc[0] input.	1'b0,1'b1	1'b1
en_phase1	Clkout[1] Phase Shift Enable. 0: Disable phase_inc[1] input. 1: Enable phase_inc[1] input.	1'b0,1'b1	1'b1
en_phase2	Clkout[2] Phase Shift Enable. 0: Disable phase_inc[2] input. 1: Enable phase_inc[2] input.	1'b0,1'b1	1'b1
en_phase3	Clkout[3] Phase Shift Enable. 0: Disable phase_inc[3] input. 1: Enable phase_inc[3] input.	1'b0,1'b1	1'b1
static_phase0	Clkout[0] VCO Static Phase Offset. The offset of the clkout[0] VCO output in 1/8 ths of the clkout[0] VCO period.	3'h0-3'h7	3'h0
static_phase1	Clkout[1] VCO Static Phase Offset. The offset of the clkout[1] VCO output in 1/8 ths of the clkout[1] VCO period.	3'h0-3'h7	3'h0
static_phase2	Clkout[2] VCO Static Phase Offset. The offset of the clkout[2] VCO output in 1/8 ths of the clkout[2] VCO period.	3'h0-3'h7	3'h0
static_phase3	Clkout[3] VCO Static Phase Offset. The offset of the clkout[3] VCO output in 1/8 ths of the clkout[3] VCO period.	3'h0-3'h7	3'h0
dyn_phase0	Clkout[0] Dynamic Phase Shift Select. If en_phase0 is enabled, 0: Clkout[0] phase shift determined by static_phase0 parameter. 1: Clkout[0] phase shift determined from the number of phase_inc[0] rising edges.	1'b0,1'b1	1'b0

Parameter	Description	Defined Values	Default Value
dyn_phase1	Clkout[1] Dynamic Phase Shift Select. If en_phase1 is enabled, 0: Clkout[1] phase shift determined by static_phase1 parameter. 1: Clkout[1] phase shift determined from the number of phase_inc[1] rising edges.	1'b0,1'b1	1'b0
dyn_phase2	Clkout[2] Dynamic Phase Shift Select. If en_phase2 is enabled, 0: Clkout[2] phase shift determined by static_phase2 parameter. 1: Clkout[2] phase shift determined from the number of phase_inc[2] rising edges.	1'b0,1'b1	1'b0
dyn_phase3	Clkout[3] Dynamic Phase Shift Select. If en_phase3 is enabled, 0: Clkout[3] phase shift determined by static_phase3 parameter. 1: Clkout[3] phase shift determined from the number of phase_inc[3] rising edges.	1'b0,1'b1	1'b0
byp_clkdiv0	Clkout[0] Output Synthesizer Bypass. 0: Clkout[0] Output Synthesizer Enabled. 1: Clkout[0] Output Synthesizer Bypassed.	1'b0,1'b1	1'b1
byp_clkdiv1	Clkout[1] Output Synthesizer Bypass. 0: Clkout[1] Output Synthesizer Enabled. 1: Clkout[1] Output Synthesizer Bypassed.	1'b0,1'b1	1'b1
byp_clkdiv2	Clkout[2] Output Synthesizer Bypass. 0: Clkout[2] Output Synthesizer Enabled. 1: Clkout[2] Output Synthesizer Bypassed.	1'b0,1'b1	1'b1
byp_clkdiv3	Clkout[3] Output Synthesizer Bypass. 0: Clkout[3] Output Synthesizer Enabled. 1: Clkout[3] Output Synthesizer Bypassed.	1'b0,1'b1	1'b1
high_cnt0	The output synthesizer divides the PLL output clock by 'high_cnt0 + low_cnt0'. The ratio of high_cnt0 to 'high_cnt0 + low_cnt0' determines the output duty cycle.	10'h000-10'h3FF	10'h0
low_cnt0	The output synthesizer divides the PLL output clock by 'high_cnt0 + low_cnt0'. The ratio of high_cnt0 to 'high_cnt0 + low_cnt0' determines the output duty cycle.	10'h000-10'h3FF	10'h0
high_cnt1	The output synthesizer divides the PLL output clock by 'high_cnt1 + low_cnt1'. The ratio of high_cnt1 to 'high_cnt1 + low_cnt1' determines the output duty cycle.	10'h000-10'h3FF	10'h0
low_cnt1	The output synthesizer divides the PLL output clock by 'high_cnt1 + low_cnt1'. The ratio of high_cnt1 to 'high_cnt1 + low_cnt1' determines the output duty cycle.	10'h000-10'h3FF	10'h0

Parameter	Description	Defined Values	Default Value
high_cnt2	The output synthesizer divides the PLL output clock by 'high_cnt2 + low_cnt2'. The ratio of high_cnt2 to 'high_cnt2 + low_cnt2' determines the output duty cycle.	10'h000-10'h3FF	10'h0
low_cnt2	If set to 0, the output synthesizer is bypassed. Otherwise, the output synthesizer divides the PLL output clock by 'high_cnt2 + low_cnt2'. The ratio of high_cnt2 to 'high_cnt2 + low_cnt2' determines the output duty cycle.	10'h000-10'h3FF	10'h0
high_cnt3	The output synthesizer divides the PLL output clock by 'high_cnt3 + low_cnt3'. The ratio of high_cnt3 to low_cnt3 determines the output duty cycle.	10'h000-10'h3FF	10'h0
low_cnt3	The output synthesizer divides the PLL output clock by 'high_cnt3 + low_cnt3'. The ratio of high_cnt3 to 'high_cnt3 + low_cnt3' determines the output duty cycle.	10'h000-10'h3FF	10'h0
half_cycle0	If set to 1, it extends the high pulse width of clkout[0] by 1/2 the clkout[0] cycle.	1'b0,1'b1	1'b0
half_cycle1	If set to 1, it extends the high pulse width of clkout[1] by 1/2 the clkout[1] cycle.	1'b0,1'b1	1'b0
half_cycle2	If set to 1, it extends the high pulse width of clkout[2] by 1/2 the clkout[2] cycle.	1'b0,1'b1	1'b0
half_cycle3	If set to 1, it extends the high pulse width of clkout[3] by 1/2 the clkout[3] cycle.	1'b0,1'b1	1'b0
clken_out0	Clkout[0] Clock Gating Enable. 0: Disable core_clken[0] input. 1: Enable core_clken[0] input.	1'b0,1'b1	1'b0
clken_out1	Clkout[1] Clock Gating Enable. 0: Disable core_clken[1] input. 1: Enable core_clken[1] input.	1'b0,1'b1	1'b0
clken_out2	Clkout[2] Clock Gating Enable. 0: Disable core_clken[2] input. 1: Enable core_clken[2] input.	1'b0,1'b1	1'b0
clken_out3	Clkout[3] Clock Gating Enable. 0: Disable core_clken[3] input. 1: Enable core_clken[3] input.	1'b0,1'b1	1'b0

Figure 9-2: *ACX_CLKGEN Block Diagram*



ACX_CLKGEN Components

Reference Divider

The input reference clock can be divided by the Reference Divider. The Reference Divider supports values from 1 to 36 and has an output with a 50% duty cycle. The frequency of the clock after it passes through the reference divider must be in the range of 66 MHz to 400 MHz for proper operation of the PLL.

Voltage Controlled Oscillator (VCO)

The ACX_CLKGEN must be configured so that the output of the Voltage Controlled Oscillator falls into the range of 1250 MHz to 2500 MHz. This is accomplished by programming the Reference, Feedback Divider, and Output Divider such that the two clock inputs to the Phase Frequency Detector are the same and fall within the 66 MHz to 400 MHz range required by the Phase Frequency Detector.

Phase Rotator with Output Divider

The Phase Rotator can shift the output clock phase in increments of 1/8th of the internal VCO clock period at a time. The phase selection may be performed dynamically using the phase_inc control input or the phase selection may be set to a static value by using the static_phase parameter. The phase selection of each of the four outputs may be set independently. The outphrstn input allows the four Phase Rotators to be simultaneously reset to zero offset.

The Output Divider provides a 50% duty cycle output and supports division rates from 1 to 63. The Output divider division ratio is set independently for each of the four outputs.

Output Synthesizer

The Output Synthesizer allows the user to divide the output by (high_cnt + low_cnt), where the ratio of high_cnt to (high_cnt + low_cnt) determines the output duty cycle.

However, output cycles other than 50% are not supported at this time. If (high_cnt + low_cnt) is an odd sum, then half_cycle must be 1 to ensure a 50% duty cycle. If enabled, when configured with a 50% duty cycle, the Output Synthesizer will act as a simple divider.

The Output Synthesizer is optional and may be bypassed by setting the byp_clkdiv to one

Phase Frequency Detector (PFD)

The Phase Frequency Detector contains the charge pump and loop filter to control the voltage input of the Voltage Controlled Oscillator. The VCO frequency is adjusted until the phase of the reference clock input (after the Reference Divider) matches the phase of the clock selected as the feedback clock into the Phase Frequency Detector. The two inputs to the PFD must be in the range of 66 MHz to 400 MHz for the PLL to lock.

Feedback Divider

The Feedback Divider supports two modes of operation: integer mode and fractional mode. When the Feedback Divider is set to integer mode, it has a 50% output with the division range of 2 to 66. In fractional mode, the divider supports a division range of 8 to 66 in the integer part. The fractional selection has a 16-bit value which is divided by 65536.

Clock Feedback Selection

The PLL supports three modes of feedback: Internal, External, and Mixed. These modes are selected by the intfb and phaseinc_sat parameters

Table 9-3: *Clock Feedback Selection*

intfb parameter setting	phaseinc_sat parameter setting	Selected Clock Feedback Mode	VCO Frequency	Output Frequency
1	0	Internal Feedback	(Q/M)*Fref	Q/(M*N*P)*Fref
0	0	External Feedback	(N*P/M)*Fref	Fref/M
1	1	Mixed Mode Feedback	(Q*N*P/M)*Fref	(Q/M)*Fref
0	1	Illegal combination		

Internal Feedback Mode

When internal feedback mode is selected, the VCO clock is divided by the Feedback Divider only. In this mode, the PLL can have both integer and fractional divider ratios. The PLL does not perform deskew capability in this mode. The VCO frequency is related to the reference clock through the relationship:

$F_{VCO}=(Q/M)*F_{ref}$ in integer mode and

$F_{VCO}=(Q.F/M)*F_{ref}$ in fractional mode.

External Feedback Mode

When external feedback mode is selected, the VCO clock is divided by the Output Divider and the Output Synthesizer. In this mode only an integer divider ratio is supported (fractional mode disabled). The clkout output of the PLL, after it has been sent through the clock network, is fed back to the PLL for deskewing. In this mode, it is recommended to not feedback a clock output that has been rotated by the Phase Rotator. The operation of the Phase Rotator introduces phase errors to the PLL and can cause the PLL to unlock. The other 3 phase rotators (the ones not in the feedback path) can be used to rotate the clock phase of the other outputs. The VCO frequency is related to the reference clock frequency by the relationship:

$F_{VCO}=(N*P/M)*F_{ref}$

Mixed Feedback Mode

Mixed Feedback mode should only be used in the case that the output divider range is not enough. The VCO is divided by the output divider inside one of the phase rotators. The clkout output of the PLL, after it has been sent through the clock network, is fed back to the PLL for deskewing. The feedback clock is sent to the Feedback Divider before it is sent to the Phase Frequency Detector. Only integer mode of the Feedback Divider should be used in this mode. The VCO frequency is related to the reference clock frequency through the relationship:

$F_{VCO}=(Q*N*P/M)*F_{ref}$

PLL Control

The default control of the ACX_CLKGEN block is by using a combination of the pins and the user-defined parameters. It is recommended that the user configure the ACX_CLKGEN module from within the ACE GUI software. Using the ACE GUI to configure the PLL has the added benefit for cross-checking the parameters for legal combinations and ensuring that the VCO has been configured to operate within the 1250 - 2500 MHz range. Once the user has generated a GUI-based Verilog or VHDL wrapper, he may later modify the settings directly within the wrapper or go back to the GUI and have it regenerate the wrapper. Users may alternatively choose to instantiate the ACX_CLKGEN module directly into their RTL code.

Resetting the PLL

The user is not required to manually reset the PLL. The FPGA Configuration Controller waits for all of the user clocks to stabilize before putting the FPGA into user mode. As a minimal configuration, the user may tie the rstn input high to keep it inactive. The user also has the choice to manually reset the PLL by asserting the rstn input low. The user must assert rstn low for at least one cycle of the clock frequency input into the Phase Frequency Detector (after the Reference Divider). After the rstn is deasserted, the pll_lock will go high within 500 refclk (after the Reference Divider) clock periods if the PLL is used in integer divider mode and 1000 clock periods if the PLL is used in fractional divider mode.

Serial Control Bus (SCB)

The ACX_CLKGEN module also allows the user to control the PLL from the Serial Control Bus (SCB) interface. Upon initial power-up of the FPGA, the PLL operates based on the values of the parameters. The parameters define the initial behavior of the ACX_CLKGEN module. The user may optionally switch control of the ACX_CLKGEN to the Control Status Registers (CSR), which are accessed via the SCB interface. Control is switched to the CSR by setting the csr_enable bit (CSR Address 1C, bit 0) high. After the csr_enable bit is enabled, the user may modify the behavior of the various components within the ACX_CLKGEN module dynamically from the FPGA fabric. **Table 9-4** defines the registers within the CSR.

Chapter 9 – "Serial Control Bus Read Operation" shows the timing for a Serial Control Bus read operation while **Chapter 9 – "Serial Control Bus Write Operation"** shows the timing for the Serial Control Bus write operation. If the user prefers to interface with CSR registers with a parallel interface, he may choose to instantiate an ACX_SBUS_MASTER from the Achronix Macro Library into the design. Note that the internal register file of the ACX_CLKGEN module is implemented as a 32-bit interface with only the bottom 8 bits are used.

Figure 9-3: *Serial Control Bus Read Operation*

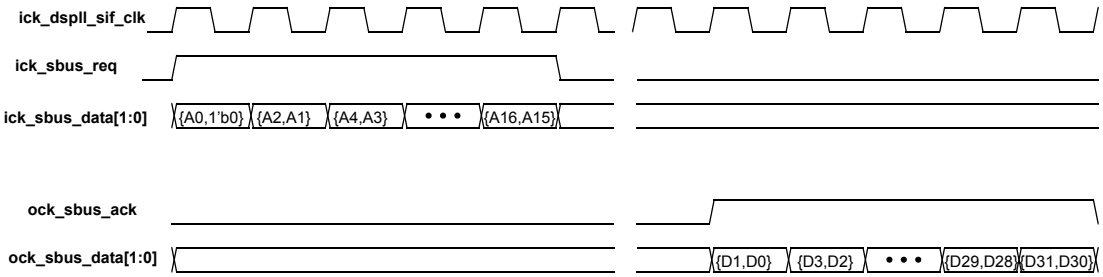
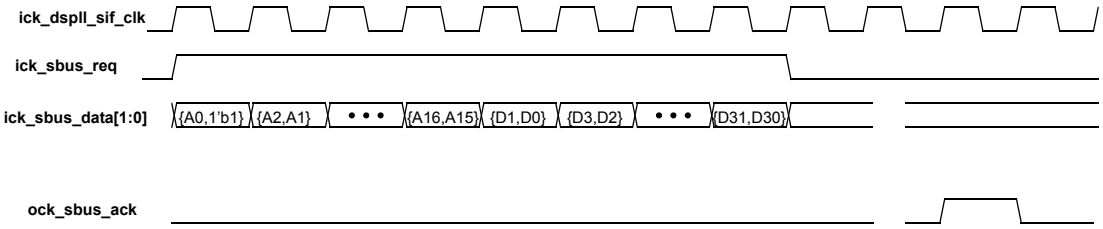


Figure 9-4: *Serial Control Bus Write Operation*



Control Status Registers (CSR) Register Description

Table 9-4: *Control Status Registers (CSR) Description*

CSR NAME	Addr.	Bit	Type	Initial Value	Description
CSR_ADDR_USER_RESERVE_00	8'h00	0	in/out	Reserved	Not Used
		1	in/out	Reserved	Not Used
		2	in/out	Reserved	Not Used
		3	in/out	Reserved	Not Used
		4	in/out	Reserved	Not Used
		5	in/out	Reserved	Not Used
		6	in/out	Reserved	Not Used
		7	in/out	Reserved	Not Used
CSR_ADDR_SYNTHOUT0	8'h01	0	in/out	outdiv0[0]	Clkout[0] Output Divider Divisor bit 0
		1	in/out	outdiv0[1]	Clkout[0] Output Divider Divisor bit 1
		2	in/out	outdiv0[2]	Clkout[0] Output Divider Divisor bit 2
		3	in/out	outdiv0[3]	Clkout[0] Output Divider Divisor bit 3
		4	in/out	outdiv0[4]	Clkout[0] Output Divider Divisor bit 4
		5	in/out	outdiv0[5]	Clkout[0] Output Divider Divisor bit 5
		6	in/out	phase_inc[0]	Increments phase of synthesizer clock output by 1/8th of1 period on rising transition
		7	in/out	clken_out0	Enable for Clkout[0] Output Synthesizer

CSR NAME	Addr.	Bit	Type	Initial Value	Description
CSR_ADDR_SYNTHOUT1	8'h02	0	in/out	outdiv1[0]	Clkout[1] Output Divider Divisor bit 0
		1	in/out	outdiv1[1]	Clkout[1] Output Divider Divisor bit 1
		2	in/out	outdiv1[2]	Clkout[1] Output Divider Divisor bit 2
		3	in/out	outdiv1[3]	Clkout[1] Output Divider Divisor bit 3
		4	in/out	outdiv1[4]	Clkout[1] Output Divider Divisor bit 4
		5	in/out	outdiv1[5]	Clkout[1] Output Divider Divisor bit 5
		6	in/out	phase_inc[1]	Increments phase of synthesizer clock output by 1/8th of1 period on rising transition
		7	in/out	clken_out1	Enable for Clkout[1] Output Synthesizer
CSR_ADDR_SYNTHOUT2	8'h03	0	in/out	outdiv2[0]	Clkout[2] Output Divider Divisor bit 0
		1	in/out	outdiv2[1]	Clkout[2] Output Divider Divisor bit 1
		2	in/out	outdiv2[2]	Clkout[2] Output Divider Divisor bit 2
		3	in/out	outdiv2[3]	Clkout[2] Output Divider Divisor bit 3
		4	in/out	outdiv2[4]	Clkout[2] Output Divider Divisor bit 4
		5	in/out	outdiv2[5]	Clkout[2] Output Divider Divisor bit 5
		6	in/out	phase_inc[2]	Increments phase of synthesizer clock output by 1/8th of1 period on rising transition
		7	in/out	clken_out2	Enable for Clkout[2] Output Synthesizer
CSR_ADDR_SYNTHOUT3	8'h04	0	in/out	outdiv3[0]	Clkout[3] Output Divider Divisor bit 0
		1	in/out	outdiv3[1]	Clkout[3] Output Divider Divisor bit 1
		2	in/out	outdiv3[2]	Clkout[3] Output Divider Divisor bit 2
		3	in/out	outdiv3[3]	Clkout[3] Output Divider Divisor bit 3
		4	in/out	outdiv3[4]	Clkout[3] Output Divider Divisor bit 4
		5	in/out	outdiv3[5]	Clkout[3] Output Divider Divisor bit 5
		6	in/out	phase_inc[3]	Increments phase of synthesizer clock output by 1/8th of1 period on rising transition
		7	in/out	clken_out3	Enable for Clkout[3] Output Synthesizer
CSR_ADDR_USER_RESERVE_05	8'h05	0	in/out	Reserved	Not Used
		1	in/out	Reserved	Not Used
		2	in/out	Reserved	Not Used
		3	in/out	Reserved	Not Used
		4	in/out	Reserved	Not Used
		5	in/out	Reserved	Not Used
		6	in/out	Reserved	Not Used
		7	in/out	Reserved	Not Used
CSR_ADDR_USER_RESERVE_06	8'h06	0	in/out	Reserved	Not Used
		1	in/out	Reserved	Not Used
		2	in/out	Reserved	Not Used
		3	in/out	Reserved	Not Used
		4	in/out	Reserved	Not Used
		5	in/out	Reserved	Not Used
		6	in/out	Reserved	Not Used
		7	in/out	Reserved	Not Used

CSR NAME	Addr.	Bit	Type	Initial Value	Description
CSR_ADDR_SYNTHOUT_BYPASS_RST	8'h07	0	in/out	bypass0	Bypass Clkout[0] synthesizer output with refclk input
		1	in/out	bypass1	Bypass Clkout[1] synthesizer output with refclk input
		2	in/out	bypass2	Bypass Clkout[2] synthesizer output with refclk input
		3	in/out	bypass3	Bypass Clkout[3] synthesizer output with refclk input
		4	in/out	Reserved	Not Used
		5	in/out	Reserved	Not Used
		6	in/out	outphrst	Resets all phase rotators to initial startup value (3'h0). Active-high
		7	in/out	phaseinc_sat	Mixed Feedback Mode
CSR_ADDR_SYNTHMDIV	8'h08	0	in/out	clkmult[0]	Feedback Divider Divisor Value bit 0
		1	in/out	clkmult[1]	Feedback Divider Divisor Value bit 1
		2	in/out	clkmult[2]	Feedback Divider Divisor Value bit 2
		3	in/out	clkmult[3]	Feedback Divider Divisor Value bit 3
		4	in/out	clkmult[4]	Feedback Divider Divisor Value bit 4
		5	in/out	clkmult[5]	Feedback Divider Divisor Value bit 5
		6	in/out	clkmult[6]	Feedback Divider Divisor Value bit 6
		7	in/out	clkmult[7]	Feedback Divider Divisor Value bit 7
CSR_ADDR_SYNTHREFDIV	8'h09	0	in/out	clkdiv[0]	Reference Divider Value bit 0
		1	in/out	clkdiv[1]	Reference Divider Value bit 1
		2	in/out	clkdiv[2]	Reference Divider Value bit 2
		3	in/out	clkdiv[3]	Reference Divider Value bit 3
		4	in/out	clkdiv[4]	Reference Divider Value bit 4
		5	in/out	clkdiv[5]	Reference Divider Value bit 5
		6	in/out	Reserved	Not Used
		7	in/out	Reserved	Not Used
CSR_ADDR_SYNTHPROPGAIN	8'h0A	0	in/out	Reserved	Reserved
		1	in/out	Reserved	Reserved
		2	in/out	Reserved	Reserved
		3	in/out	Reserved	Reserved
		4	in/out	Reserved	Reserved
		5	in/out	Reserved	Reserved
		6	in/out	Reserved	Reserved
		7	in/out	Reserved	Not Used

CSR NAME	Addr.	Bit	Type	Initial Value	Description
CSR_ADDR_SYNTHSSCMODGAIN_LSB	8'h0B	0	in/out	frac_div_ctrl[0]	Feedback Divider Fractional Portion bit 0
		1	in/out	frac_div_ctrl[1]	Feedback Divider Fractional Portion bit 1
		2	in/out	frac_div_ctrl[2]	Feedback Divider Fractional Portion bit 2
		3	in/out	frac_div_ctrl[3]	Feedback Divider Fractional Portion bit 3
		4	in/out	frac_div_ctrl[4]	Feedback Divider Fractional Portion bit 4
		5	in/out	frac_div_ctrl[5]	Feedback Divider Fractional Portion bit 5
		6	in/out	frac_div_ctrl[6]	Feedback Divider Fractional Portion bit 6
		7	in/out	frac_div_ctrl[7]	Feedback Divider Fractional Portion bit 7
CSR_ADDR_SYNTHSSCMODGAIN_MSB	8'h0C	0	in/out	frac_div_ctrl[8]	Feedback Divider Fractional Portion bit 8
		1	in/out	frac_div_ctrl[9]	Feedback Divider Fractional Portion bit 9
		2	in/out	frac_div_ctrl[10]	Feedback Divider Fractional Portion bit 10
		3	in/out	frac_div_ctrl[11]	Feedback Divider Fractional Portion bit 11
		4	in/out	frac_div_ctrl[12]	Feedback Divider Fractional Portion bit 12
		5	in/out	frac_div_ctrl[13]	Feedback Divider Fractional Portion bit 13
		6	in/out	frac_div_ctrl[14]	Feedback Divider Fractional Portion bit 14
		7	in/out	frac_div_ctrl[15]	Feedback Divider Fractional Portion bit 15
CSR_ADDR_SYNTHSSCGENCOUNT_LSB	8'h0D	0	in/out	Reserved	Reserved
		1	in/out	Reserved	Reserved
		2	in/out	Reserved	Reserved
		3	in/out	Reserved	Reserved
		4	in/out	Reserved	Reserved
		5	in/out	Reserved	Reserved
		6	in/out	Reserved	Reserved
		7	in/out	Reserved	Reserved
CSR_ADDR_SYNTHSSCGENCOUNT_MSB	8'h0E	0	in/out	Reserved	Reserved
		1	in/out	Reserved	Reserved
		2	in/out	Reserved	Reserved
		3	in/out	Reserved	Reserved
		4	in/out	Reserved	Reserved
		5	in/out	Reserved	Reserved
		6	in/out	Reserved	Reserved
		7	in/out	Reserved	Reserved
CSR_ADDR_SYNTHINTFB_MODE_TEST_FAST	8'h0F	0	in/out	intfb	Internal Feedback Enable
		1	in/out	synthmode	Feedback Divider Mode
		2	in/out	Reserved	Reserved
		3	in/out	Reserved	Reserved
		4	in/out	Reserved	Not Used.
		5	in/out	Reserved	Not Used.
		6	in/out	Reserved	Not Used.
		7	in/out	Reserved	Not Used.

CSR NAME	Addr.	Bit	Type	Initial Value	Description
CSR_ADDR_DFTADDR	8'h10	0	in/out	Reserved	Reserved
		1	in/out	Reserved	Reserved
		2	in/out	Reserved	Reserved
		3	in/out	Reserved	Reserved
		4	in/out	Reserved	Not Used.
		5	in/out	Reserved	Not Used.
		6	in/out	Reserved	Not Used.
		7	in/out	Reserved	Not Used.
CSR_ADDR_PLL_CTL1	8'h11	0	in/out	Reserved	Reserved
		1	in/out	Reserved	Reserved
		2	in/out	Reserved	Reserved
		3	in/out	Reserved	Reserved
		4	in/out	Reserved	Reserved
		5	in/out	Reserved	Reserved
		6	in/out	Reserved	Reserved
		7	in/out	Reserved	Reserved
CSR_ADDR_PLL_CTL2	8'h12	0	in/out	Reserved	Reserved
		1	in/out	Reserved	Reserved
		2	in/out	Reserved	Reserved
		3	in/out	Reserved	Reserved
		4	in/out	Reserved	Reserved
		5	in/out	Reserved	Reserved
		6	in/out	Reserved	Reserved
		7	in/out	Reserved	Reserved
CSR_ADDR_PLL_CTL3	8'h13	0	in/out	Reserved	Reserved
		1	in/out	Reserved	Reserved
		2	in/out	Reserved	Reserved
		3	in/out	Reserved	Reserved
		4	in/out	Reserved	Reserved
		5	in/out	Reserved	Reserved
		6	in/out	Reserved	Reserved
		7	in/out	Reserved	Reserved
CSR_ADDR_PLL_CTL4	8'h14	0	in/out	Reserved	Reserved
		1	in/out	Reserved	Reserved
		2	in/out	Reserved	Reserved
		3	in/out	Reserved	Reserved
		4	in/out	Reserved	Reserved
		5	in/out	Reserved	Reserved
		6	in/out	Reserved	Reserved
		7	in/out	Reserved	Reserved

CSR NAME	Addr.	Bit	Type	Initial Value	Description
CSR_ADDR_LDO_CTL	8'h15	0	in/out	Reserved	Reserved
		1	in/out	Reserved	Reserved
		2	in/out	Reserved	Reserved
		3	in/out	Reserved	Reserved
		4	in/out	Reserved	Reserved
		5	in/out	Reserved	Reserved
		6	in/out	Reserved	Reserved
		7	in/out	Reserved	Reserved
CSR_ADDR_BGR_CTL1	8'h16	0	in/out	Reserved	Reserved
		1	in/out	Reserved	Reserved
		2	in/out	Reserved	Reserved
		3	in/out	Reserved	Reserved
		4	in/out	Reserved	Reserved
		5	in/out	Reserved	Reserved
		6	in/out	Reserved	Reserved
		7	in/out	Reserved	Reserved
CSR_ADDR_BGR_CTL2	8'h17	0	in/out	Reserved	Reserved
		1	in/out	Reserved	Reserved
		2	in/out	Reserved	Reserved
		3	in/out	Reserved	Reserved
		4	in/out	Reserved	Reserved
		5	in/out	Reserved	Reserved
		6	in/out	Reserved	Reserved
		7	in/out	Reserved	Reserved
CSR_ADDR_ADC_CTL1	8'h18	0	in/out	Reserved	Reserved
		1	in/out	Reserved	Reserved
		2	in/out	Reserved	Reserved
		3	in/out	Reserved	Reserved
		4	in/out	Reserved	Reserved
		5	in/out	Reserved	Reserved
		6	in/out	Reserved	Reserved
		7	in/out	Reserved	Reserved
CSR_ADDR_ADC_CTL2	8'h19	0	in/out	Reserved	Reserved
		1	in/out	Reserved	Reserved
		2	in/out	Reserved	Reserved
		3	in/out	Reserved	Reserved
		4	in/out	Reserved	Reserved
		5	in/out	Reserved	Reserved
		6	in/out	Reserved	Reserved
		7	in/out	Reserved	Reserved

CSR NAME	Addr.	Bit	Type	Initial Value	Description
CSR_ADDR_ADC_DATA1	8'h1A	0	output	Reserved	Reserved
		1	output	Reserved	Reserved
		2	output	Reserved	Reserved
		3	output	Reserved	Reserved
		4	output	Reserved	Reserved
		5	output	Reserved	Reserved
		6	output	Reserved	Reserved
		7	output	Reserved	Reserved
CSR_ADDR_ADC_DATA2	8'h1B	0	output	Reserved	Reserved
		1	output	Reserved	Reserved
		2	output	Reserved	Reserved
		3	output	Reserved	Reserved
		4	output	Reserved	Reserved
		5	output	Reserved	Reserved
		6	output	Reserved	Reserved
		7	output	Reserved	Reserved
CSR_ADDR_USER_CONTROL	8'h1C	0	in/out	csr_enable = 1'b0	CSR Control bit 0: [0]=CSR disabled, PLL controlled by parameters; [1]=: CSR registers control the PLL operation.
		1	in/out	Reserved	Reserved
		2	in/out	Reserved	Reserved
		3	output	1'b0	Not used.
		4	output	1'b0	
		5	output	1'b0	
		6	output	1'b0	
		7	output	1'b0	

Verilog Instantiation Template

```
ACX_CLKGEN #
(
.clkdiv             (6'h1),
.intfb              (1'b0),
.phaseinc_sat       (1'b0),
.clkmult            (8'h0),
.synthmode          (1'b0),
.frac_div_ctrl      (16'h0),
.clkouten_mode      (1'b0),
.pll_user_reset_en  (1'b0),
.pll_user_outrst_en (1'b0),
.pll_user_csrrst_en (1'b0),

.bypass0            (1'b0),
.outdiv0            (6'h4),
.en_phase0          (1'b1),
.static_phase0      (3'h0),
.dyn_phase0         (1'b0),
.byp_clkdiv0        (1'b1),
.high_cnt0          (10'h0),
```

```
.low_cnt0           (10'h0),
.half_cycle0        (1'b0),
.clken_out0         (1'b1),

.bypass1            (1'b0),
.outdiv1            (6'h4),
.en_phase1          (1'b1),
.static_phase1      (3'h0),
.dyn_phase1         (1'b0),
.byp_clkdiv1        (1'b1),
.high_cnt1          (10'h0),
.low_cnt1           (10'h0),
.half_cycle1        (1'b0),
.clken_out1         (1'b0),

.bypass2            (1'b0),
.outdiv2            (6'h4),
.en_phase2          (1'b1),
.static_phase2      (3'h0),
.dyn_phase2         (1'b0),
.byp_clkdiv2        (1'b1),
.high_cnt2          (10'h0),
.low_cnt2           (10'h0),
.half_cycle2        (1'b0),
.clken_out2         (1'b0),

.bypass3            (1'b0),
.outdiv3            (6'h4),
.en_phase3          (1'b1),
.static_phase3      (3'h0),
.dyn_phase3         (1'b0),
.byp_clkdiv3        (1'b1),
.high_cnt3          (10'h0),
.low_cnt3           (10'h0),
.half_cycle3        (1'b0),
.clken_out3         (1'b0)
)
User_ACX_CLKGEN
(
 .refclk             (user_refclk),
 .fbclk              (user_fbclk),
 .rstn               (user_rstn),
 .outphrstn          (user_outphrstn),
 .core_clken         (user_core_clken),
 .phase_inc          (user_phase_inc),
 .clkout             (user_clkout),
 .pll_lock           (user_pll_lock),
 .ick_dspll_sif_clk  (user_ick_dspll_sif_clk),
 .ick_dspll_sif_rstn (user_ick_dspll_sif_rstn),
 .ick_sbus_data      (user_ick_sbus_data),
 .ick_sbus_req       (user_ick_sbus_req),
 .ock_sbus_ack       (user_ock_sbus_ack),
 .ock_sbus_data      (user_ock_sbus_data)
);
```

VHDL Instantiation Template

```
------------ ACHRONIX LIBRARY ------------
library speedster22i;
use speedster22i.components.all;
------------ DONE ACHRONIX LIBRARY ---------

--Component instantiation
ACX_CLKGEN_instance_name
  generic map (
    clkdiv => "000001",
    intfb => "0"  ,
    phaseinc_sat => "0",
    clkmult => X"00",
    synthmode => "0",
    frac_div_ctrl => X"0000",
    clkouten_mode => "0",
    pll_user_reset_en => "0",
    pll_user_outrst_en => "0",
    pll_user_csrrst_en => "0",

    bypass0 => "0",
    outdiv0 => "000100",
    en_phase0 => "1",
    static_phase0 => "000",
    dyn_phase0 => "0",
    byp_clkdiv0 => "1",
    high_cnt0 => "0000000000",
    low_cnt0 => "0000000000",
    half_cycle0 => "0",
    clken_out0 => "1",

    bypass1 => "0",
    outdiv1 => "000100",
    en_phase1 => "1",
    static_phase1 => "000",
    dyn_phase1 => "0",
    byp_clkdiv1 => "1",
    high_cnt1 => "0000000000",
    low_cnt1 => "0000000000",
    half_cycle1 => "0",
    clken_out1 => "0",

    bypass2 => "0",
    outdiv2 => "000100",
    en_phase2 => "1",
    static_phase2 => "000",
    dyn_phase2 => "0",
    byp_clkdiv2 => "1",
    high_cnt2 => "0000000000",
    low_cnt2 => "0000000000",
    half_cycle2 => "0",
    clken_out2 => "0",

    bypass3 => "0",
    outdiv3 => "000100",
    en_phase3 => "1",
    static_phase3 => "000",
```

```
    dyn_phase3 => "0",
    byp_clkdiv3 => "1",
    high_cnt3 => "0000000000",
    low_cnt3 => "0000000000",
    half_cycle3 => "0",
    clken_out3 => "0")

  port map (
   refclk => user_refclk,
   fbclk => user_fbclk,
   rstn => user_rstn,
   outphrstn => user_outphrstn,
   core_clken => user_core_clken,
   phase_inc => user_phase_inc,
   clkout => user_clkout,
   ick_dspll_sif_clk => user_ick_dspll_sif_clk,
   ick_dspll_sif_rstn => user_ick_dspll_sif_rstn,
   ick_sbus_data => user_ick_sbus_data,
   ick_sbus_req => user_ick_sbus_req,
   ock_sbus_ack => user_ock_sbus_ack,
   ock_sbus_data => user_ock_sbus_data,
   pll_lock => user_pll_lock);
```

Revision History

The following table lists the revision history of this document.

Version	Revision
1.0	Initial released version.
1.1	• 2/2/2012 ACE 4.1 Release.
1.2	• 3/31/2012 ACE 4.2 Release
1.3	• 5/4/2012 ACE 4.2 Release
1.4	• 10/25/2012 ACE 5.0 Release
1.5	• 3/29/2013 ACE 5.1 Release
1.6	• 8/4/2014 ACE 5.3 Release
1.7	• 10/2 4/2014 Updated Registers section
1.8	• 1/21/2014 Fixed Typo in BMULT28X28
1.9	• Updated PLL Specs - The minimum reference clock (after "M" divider) is now 66MHz (was 30). The VCO range is now 1250MHz - 2500MHz (was 1000-2000) • Updated Tables 6-3, 6-4, 6-24, 6-25
1.10	• Updated BRAM tables 6-8, 6-10, 6-18, 6-20