LatticeXP2™ Family Handbook

HB1004 Version 01.4, January 2008

LatticeXP2 Family Handbook Table of Contents

January 2008

Section I. LatticeXP2 Family Data Sheet

Section I. LatticeXP2 Family Data Sheet

DS1009 Version 01.2, September 2007

LatticeXP2 Family Data Sheet Introduction

May 2007 Advance Data Sheet DS1009

Features

- **flexiFLASH™ Architecture**
  - Instant-on
  - Infinitely reconfigurable
  - Single chip
  - FlashBAK™ technology
  - Serial TAG memory
  - Design security
- **Live Update Technology**
  - TransFR™ technology
  - Secure updates with 128 bit AES encryption
  - Dual-boot with external SPI
- **sysDSP™ Block**
  - Three to eight blocks for high performance Multiply and Accumulate
  - 12 to 32 18x18 multipliers
  - Each block supports one 36x36 multiplier or four 18x18 or eight 9x9 multipliers
- **Embedded and Distributed Memory**
  - Up to 885 Kbits sysMEM™ EBR
  - Up to 83 Kbits Distributed RAM
- **Flexible I/O Buffer**
  - sysIO™ buffer supports:
    - LVCMOS 33/25/18/15/12; LVTTL
    - SSTL 33/25/18 class I, II
    - HSTL15 class I; HSTL18 class I, II
    - PCI
    - LVDS, Bus-LVDS, MLVDS, LVPECL, RSDS
- **Pre-engineered Source Synchronous Interfaces**
  - DDR / DDR2 interfaces up to 200 MHz
  - 7:1 LVDS interfaces support display applications
  - XGMII
- **sysCLOCK™ PLLs**
  - Up to four analog PLLs per device
  - Clock multiply, divide and phase shifting
- **Density And Package Options**
  - 5k to 40k LUT4s, 86 to 540 I/Os
  - csBGA, TQFP, QFP, ftBGA and fpBGA packages
  - Density migration supported
- **Flexible Device Configuration**
  - SPI (master and slave) Boot Flash Interface
  - Dual Boot Image supported
  - Soft Error Detect (SED) macro embedded
- **System Level Support**
  - IEEE 1149.1 and IEEE 1532 Compliant
  - On-chip oscillator for initialization & general use
  - Devices operate with 1.2V power supply

Table 1-1. LatticeXP2 Family Selection Guide

Device	XP2-5	XP2-8	XP2-17	XP2-30	XP2-40
LUTs (K)	5	8	17	29	40
Distributed RAM (KBits)	10	18	35	56	83
EBR SRAM (KBits)	166	221	276	387	885
EBR SRAM Blocks	9	12	15	21	48
sysDSP Blocks	3	4	5	7	8
18 x 18 Multipliers	12	16	20	28	32
V_{CC} Voltage	1.2	1.2	1.2	1.2	1.2
GPLL	2	2	4	4	4
Max Available I/O	172	201	358	472	540
Packages and I/O Combinations					
132-Ball csBGA (8 x 8 mm)	86	86			
144-Pin TQFP (20 x 20 mm)	100	100			
208-Pin PQFP (28 x 28 mm)	146	146	146		
256-Ball ftBGA (17 x17 mm)	172	201	201	201	
484-Ball fpBGA (23 x 23 mm)			358	363	363
672-Ball fpBGA (27 x 27 mm)				472	540

Note: The information in this Advance Data Sheet is by definition not final and subject to change. Please consult the Lattice website and your local Lattice Sales Manager to ensure you have the latest information regarding the specifications for these products as you make critical design decisions.

Introduction

LatticeXP2 devices combine a Look-up Table (LUT) based FPGA fabric with non-volatile Flash cells in an architecture referred to as flexiFLASH.

The flexiFLASH approach provides benefits including instant-on, infinite reconfigurability, on chip storage with FlashBAK embedded block memory and Serial TAG memory and design security. The parts also support Live Update technology with TransFR, 128-bit AES Encryption and Dual-boot technologies.

The LatticeXP2 FPGA fabric was optimized for the new technology from the outset with high performance and low cost in mind. LatticeXP2 devices include LUT-based logic, distributed and embedded memory, Phase Locked Loops (PLLs), pre-engineered source synchronous I/O support and enhanced sysDSP blocks.

The ispLEVER® design tool from Lattice allows large and complex designs to be efficiently implemented using the LatticeXP2 family of FPGA devices. Synthesis library support for LatticeXP2 is available for popular logic synthesis tools. The ispLEVER tool uses the synthesis tool output along with the constraints from its floor planning tools to place and route the design in the LatticeXP2 device. The ispLEVER tool extracts the timing from the routing and back-annotates it into the design for timing verification.

Lattice provides many pre-designed Intellectual Property (IP) ispLeverCORE™ modules for the LatticeXP2 family. By using these IPs as standardized blocks, designers are free to concentrate on the unique aspects of their design, increasing their productivity.

LatticeXP2 Family Data Sheet Architecture

May 2007 Advance Data Sheet DS1009

Architecture Overview

Each LatticeXP2 device contains an array of logic blocks surrounded by Programmable I/O Cells (PIC). Interspersed between the rows of logic blocks are rows of sysMEM™ Embedded Block RAM (EBR) and a row of sysDSP™ Digital Signal Processing blocks as shown in Figure 2-1.

On the left and right sides of the Programmable Functional Unit (PFU) array, there are Non-volatile Memory Blocks. In configuration mode the nonvolatile memory is programmed via the IEEE 1149.1 TAP port or the sysCONFIG™ peripheral port. On power up, the configuration data is transferred from the Non-volatile Memory Blocks to the configuration SRAM. With this technology, expensive external configuration memory is not required, and designs are secured from unauthorized read-back. This transfer of data from non-volatile memory to configuration SRAM via wide busses happens in microseconds, providing an “instant-on” capability that allows easy interfacing in many applications. LatticeXP2 devices can also transfer data from the sysMEM EBR blocks to the Non-volatile Memory Blocks at user request.

There are two kinds of logic blocks, the PFU and the PFU without RAM (PFF). The PFU contains the building blocks for logic, arithmetic, RAM and ROM functions. The PFF block contains building blocks for logic, arithmetic and ROM functions. Both PFU and PFF blocks are optimized for flexibility allowing complex designs to be implemented quickly and efficiently. Logic Blocks are arranged in a two-dimensional array. Only one type of block is used per row.

LatticeXP2 devices contain one or more rows of sysMEM EBR blocks. sysMEM EBRs are large dedicated 18Kbit memory blocks. Each sysMEM block can be configured in a variety of depths and widths of RAM or ROM. In addition, LatticeXP2 devices contain up to two rows of DSP Blocks. Each DSP block has multipliers and adder/accumulators, which are the building blocks for complex signal processing capabilities.

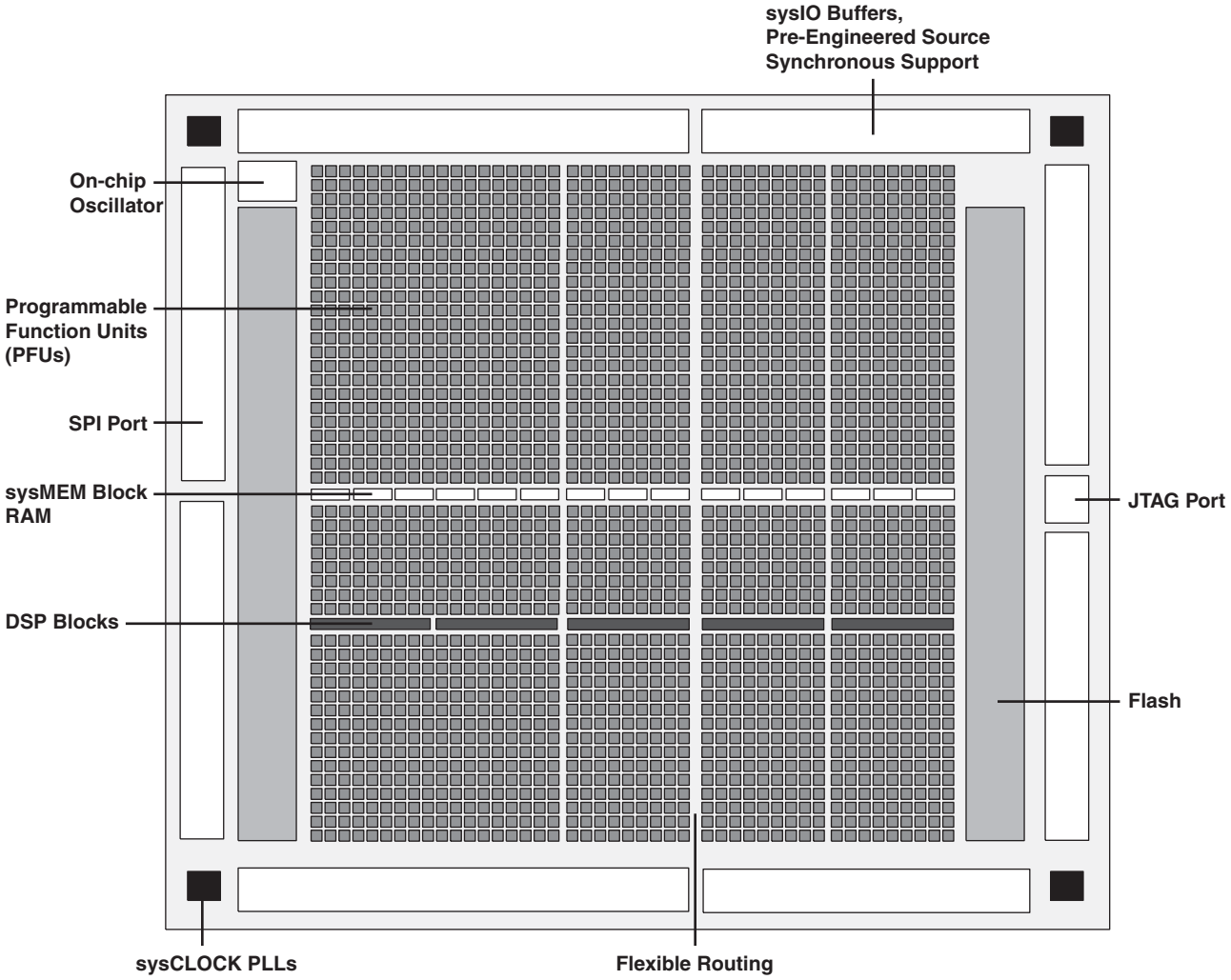
Each PIC block encompasses two PIOs (PIO pairs) with their respective sysIO buffers. The sysIO buffers of the LatticeXP2 devices are arranged into eight banks, allowing the implementation of a wide variety of I/O standards. In addition, a separate I/O bank is provided for programming interfaces. PIO pairs on the left and right edges of the device can be configured as LVDS transmit/receive pairs. The PIC logic also includes pre-engineered support to aid in the implementation of high speed source synchronous standards such as 7:1 LVDS interfaces, found in many display applications, and memory interfaces including DDR and DDR2.

Other blocks provided include PLLs and configuration functions. The LatticeXP2 architecture provides up to four General Purpose PLLs (GPLL) per device. The GPLL blocks are located in the corners of the device.

The configuration block that supports features such as configuration bit-stream de-encryption, transparent updates and dual boot support is located between banks two and three. Every device in the LatticeXP2 family supports a sysCONFIG port, muxed with bank seven I/Os, which supports serial device configuration. A JTAG port is provided between banks two and three.

This family also provides an on-chip oscillator and Soft Error Detect (SED) capability. LatticeXP2 devices use 1.2V as their core voltage.

Figure 2-1. Simplified Block Diagram, LatticeXP2-17 Device (Top Level)

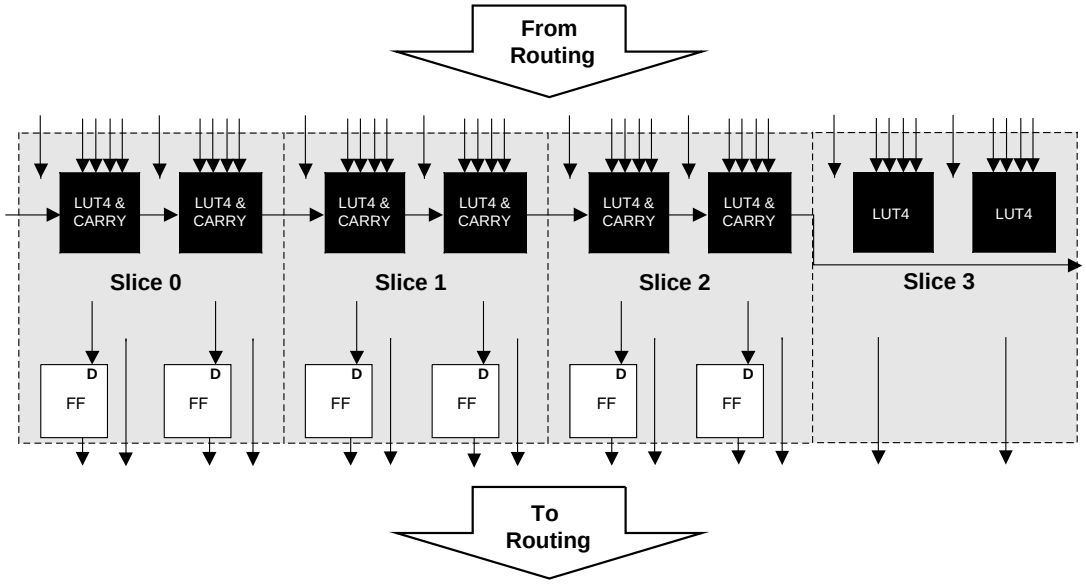


PFU Blocks

The core of the LatticeXP2 device is made up of logic blocks in two forms, PFUs and PFFs. PFUs can be programmed to perform logic, arithmetic, distributed RAM and distributed ROM functions. PFF blocks can be programmed to perform logic, arithmetic and ROM functions. Except where necessary, the remainder of this data sheet will use the term PFU to refer to both PFU and PFF blocks.

Each PFU block consists of four interconnected slices, numbered Slice 0 through Slice 3, as shown in Figure 2-2. All the interconnections to and from PFU blocks are from routing. There are 50 inputs and 23 outputs associated with each PFU block.

Figure 2-2. PFU Diagram



Slice

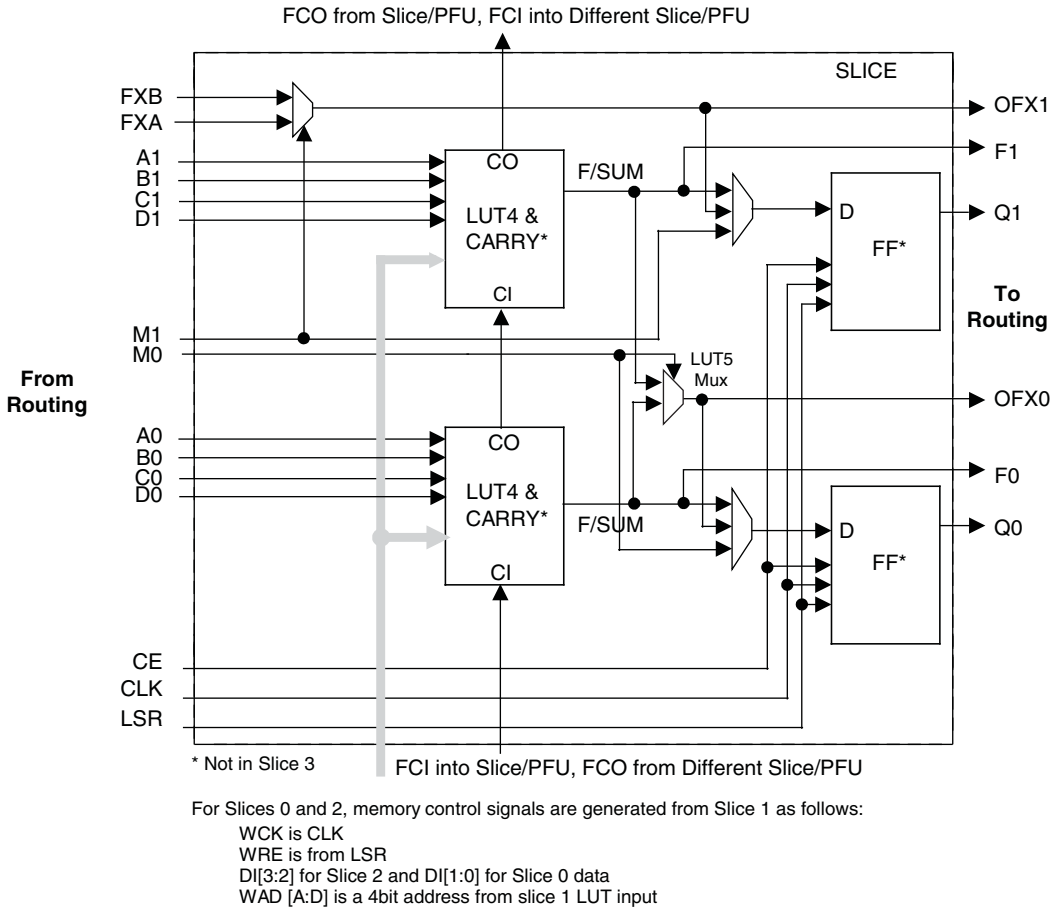
Slice 0 through Slice 2 contain two 4-input combinatorial Look-Up Tables (LUT4), which feed two registers. Slice 3 contains two LUT4s and no registers. For PFUs, Slice 0 and Slice 2 can also be configured as distributed memory, a capability not available in PFF blocks. Table 2-1 shows the capability of the slices in both PFF and PFU blocks along with the operation modes they enable. In addition, each PFU contains logic that allows the LUTs to be combined to perform functions such as LUT5, LUT6, LUT7 and LUT8. There is control logic to perform set/reset functions (programmable as synchronous/asynchronous), clock select, chip-select and wider RAM/ROM functions. Figure 2-3 shows an overview of the internal logic of the slice. The registers in the slice can be configured for positive/negative and edge triggered or level sensitive clocks.

Table 2-1. Resources and Modes Available per Slice

Slice	PFU BLock		PFF Block	
	Resources	Modes	Resources	Modes
Slice 0	2 LUT4s and 2 Registers	Logic, Ripple, RAM, ROM	2 LUT4s and 2 Registers	Logic, Ripple, ROM
Slice 1	2 LUT4s and 2 Registers	Logic, Ripple, ROM	2 LUT4s and 2 Registers	Logic, Ripple, ROM
Slice 2	2 LUT4s and 2 Registers	Logic, Ripple, RAM, ROM	2 LUT4s and 2 Registers	Logic, Ripple, ROM
Slice 3	2 LUT4s	Logic, ROM	2 LUT4s	Logic, ROM

Slice 0 through Slice 2 have 14 input signals: 13 signals from routing and one from the carry-chain (from the adjacent slice or PFU). There are seven outputs: six to routing and one to carry-chain (to the adjacent PFU). Slice 3 has 13 input signals from routing and four signals to routing. Table 2-2 lists the signals associated with Slice 0 to Slice 2.

Figure 2-3. Slice Diagram



For Slices 0 and 2, memory control signals are generated from Slice 1 as follows:

WCK is CLK
WRE is from LSR
DI[3:2] for Slice 2 and DI[1:0] for Slice 0 data
WAD [A:D] is a 4bit address from slice 1 LUT input

Table 2-2. Slice Signal Descriptions

Function	Type	Signal Names	Description
Input	Data signal	A0, B0, C0, D0	Inputs to LUT4
Input	Data signal	A1, B1, C1, D1	Inputs to LUT4
Input	Multi-purpose	M0	Multipurpose Input
Input	Multi-purpose	M1	Multipurpose Input
Input	Control signal	CE	Clock Enable
Input	Control signal	LSR	Local Set/Reset
Input	Control signal	CLK	System Clock
Input	Inter-PFU signal	FCI	Fast Carry-In[1]
Input	Inter-slice signal	FXA	Intermediate signal to generate LUT6 and LUT7
Input	Inter-slice signal	FXB	Intermediate signal to generate LUT6 and LUT7
Output	Data signals	F0, F1	LUT4 output register bypass signals
Output	Data signals	Q0, Q1	Register outputs
Output	Data signals	OFX0	Output of a LUT5 MUX
Output	Data signals	OFX1	Output of a LUT6, LUT7, LUT8[2] MUX depending on the slice
Output	Inter-PFU signal	FCO	Slice 2 of each PFU is the fast carry chain output[1]

1. See Figure 2-3 for connection details.
2. Requires two PFUs.

Modes of Operation

Each slice has up to four potential modes of operation: Logic, Ripple, RAM and ROM.

Logic Mode
In this mode, the LUTs in each slice are configured as LUT4s. A LUT4 has 16 possible input combinations. Four-input logic functions are generated by programming the LUT4. Since there are two LUT4s per slice, a LUT5 can be constructed within one slice. Larger LUTs such as LUT6, LUT7 and LUT8, can be constructed by concatenating two or more slices. Note that a LUT8 requires more than four slices.

Ripple Mode
Ripple mode allows efficient implementation of small arithmetic functions. In ripple mode, the following functions can be implemented by each slice:

- Addition 2-bit
- Subtraction 2-bit
- Add/Subtract 2-bit using dynamic control
- Up counter 2-bit
- Down counter 2-bit
- Up/Down counter with async clear
- Up/Down counter with preload (sync)
- Ripple mode multiplier building block
- Multiplier support
- Comparator functions of A and B inputs
  - A greater-than-or-equal-to B
  - A not-equal-to B
  - A less-than-or-equal-to B

Two carry signals, FCI and FCO, are generated per slice in this mode, allowing fast arithmetic functions to be constructed by concatenating slices.

RAM Mode
In this mode, a 16x4-bit distributed Single Port RAM (SPR) can be constructed using each LUT block in Slice 0 and Slice 2 as a 16x1-bit memory. Slice 1 is used to provide memory address and control signals. A 16x2-bit Pseudo Dual Port RAM (PDPR) memory is created by using one slice as the read-write port and the other companion slice as the read-only port.

The Lattice design tools support the creation of a variety of different size memories. Where appropriate, the software will construct these using distributed memory primitives that represent the capabilities of the PFU. Table 2-3 shows the number of slices required to implement different distributed RAM primitives. For more information on using RAM in LatticeXP2 devices, please see TN1137, *LatticeXP2 Memory Usage Guide.*

Table 2-3. Number of Slices Required For Implementing Distributed RAM

	SPR 16X4	PDPR 16X4
Number of slices	3	3

Note: SPR = Single Port RAM, PDPR = Pseudo Dual Port RAM

ROM Mode
ROM mode uses the LUT logic; hence, Slices 0 through 3 can be used in the ROM mode. Preloading is accomplished through the programming interface during PFU configuration.

Routing

There are many resources provided in the LatticeXP2 devices to route signals individually or as busses with related control signals. The routing resources consist of switching circuitry, buffers and metal interconnect (routing) segments.

The inter-PFU connections are made with x1 (spans two PFU), x2 (spans three PFU) or x6 (spans seven PFU) connections. The x1 and x2 connections provide fast and efficient connections in horizontal and vertical directions. The x2 and x6 resources are buffered to allow both short and long connections routing between PFUs.

The LatticeXP2 family has an enhanced routing architecture to produce a compact design. The ispLEVER design tool takes the output of the synthesis tool and places and routes the design. Generally, the place and route tool is completely automatic, although an interactive routing editor is available to optimize the design.

sysCLOCK Phase Locked Loops (PLL)

The sysCLOCK PLLs provide the ability to synthesize clock frequencies. The LatticeXP2 family supports between two and four full featured General Purpose PLLs (GPLL). The architecture of the GPLL is shown in Figure 2-4.

CLKI, the PLL reference frequency, is provided either from the pin or from routing; it feeds into the Input Clock Divider block. CLKFB, the feedback signal, is generated from CLKOP (the primary clock output) or from a user clock pin/logic. CLKFB feeds into the Feedback Divider and is used to multiply the reference frequency.

Both the input path and feedback signals enter the Voltage Controlled Oscillator (VCO) block. The phase and frequency of the VCO are determined from the input path and feedback signals. A LOCK signal is generated by the VCO to indicate that the VCO is locked with the input clock signal.

The output of the VCO feeds into the CLKOP Divider, a post-scalar divider. The duty cycle of the CLKOP Divider output can be fine tuned using the Duty Trim block, which creates the CLKOP signal. By allowing the VCO to operate at higher frequencies than CLKOP, the frequency range of the GPLL is expanded. The output of the CLKOP Divider is passed through the CLKOK Divider, a secondary clock divider, to generate lower frequencies for the CLKOK output. For applications that require even lower frequencies, the CLKOP signal is passed through a divide-by-three divider to produce the CLKOK2 output. The CLKOK2 output is provided for applications that use source synchronous logic. The Phase/Duty Cycle/Duty Trim block is used to adjust the phase and duty cycle of the CLKOP Divider output to generate the CLKOS signal. The phase/duty cycle setting can be pre-programmed or dynamically adjusted.

The clock outputs from the GPLL; CLKOP, CLKOK, CLKOK2 and CLKOS, are fed to the clock distribution network.

For further information on the GPLL please see TN1126, *LatticeXP2 sysCLOCK PLL Design and Usage Guide.*

Figure 2-4. General Purpose PLL (GPLL) Diagram

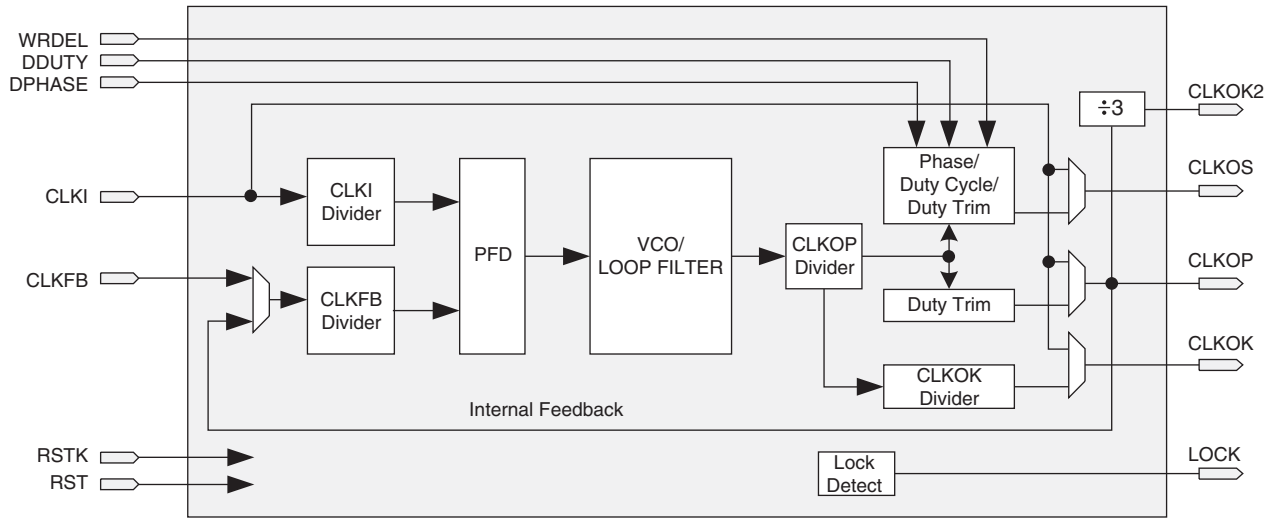


Table 2-4 provides a description of the signals in the GPLL blocks.

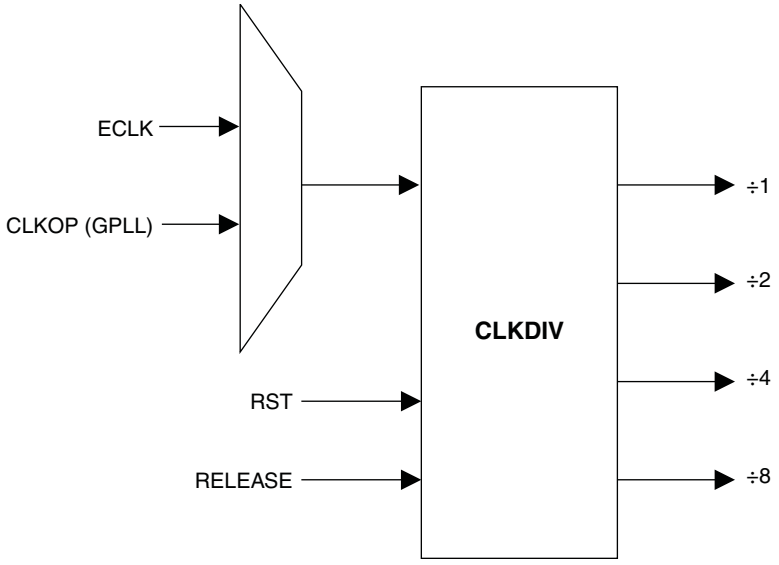
Table 2-4. GPLL Block Signal Descriptions

Signal	I/O	Description
CLKI	I	Clock input from external pin or routing
CLKFB	I	PLL feedback input from CLKOP (PLL internal), from clock net (CLKOP) or from a user clock (PIN or logic)
RST	I	"1" to reset PLL counters, VCO, charge pumps and M-dividers
RSTK	I	"1" to reset K-divider
DPHASE [3:0]	I	DPA Phase Adjust input
DDDUTY [3:0]	I	DPA Duty Cycle Select input
WRDEL	I	DPA Fine Delay Adjust input
CLKOS	O	PLL output clock to clock tree (phase shifted/duty cycle changed)
CLKOP	O	PLL output clock to clock tree (no phase shift)
CLKOK	O	PLL output to clock tree through secondary clock divider
CLKOK2	O	PLL output to clock tree (CLKOP divided by 3)
LOCK	O	"1" indicates PLL LOCK to CLKI

Clock Dividers

LatticeXP2 devices have two clock dividers, one on the left side and one on the right side of the device. These are intended to generate a slower-speed system clock from a high-speed edge clock. The block operates in a ÷2, ÷4 or ÷8 mode and maintains a known phase relationship between the divided down clock and the high-speed clock based on the release of its reset signal. The clock dividers can be fed from the CLKOP output from the GPLLs or from the Edge Clocks (ECLK). The clock divider outputs serve as primary clock sources and feed into the clock distribution network. The Reset (RST) control signal resets the input and forces all outputs to low. The RELEASE signal releases outputs to the input clock. For further information on clock dividers, please see TN1126, *sysCLOCK PLL Design and Usage Guide.* Figure 2-5 shows the clock divider connections.

Figure 2-5. Clock Divider Connections



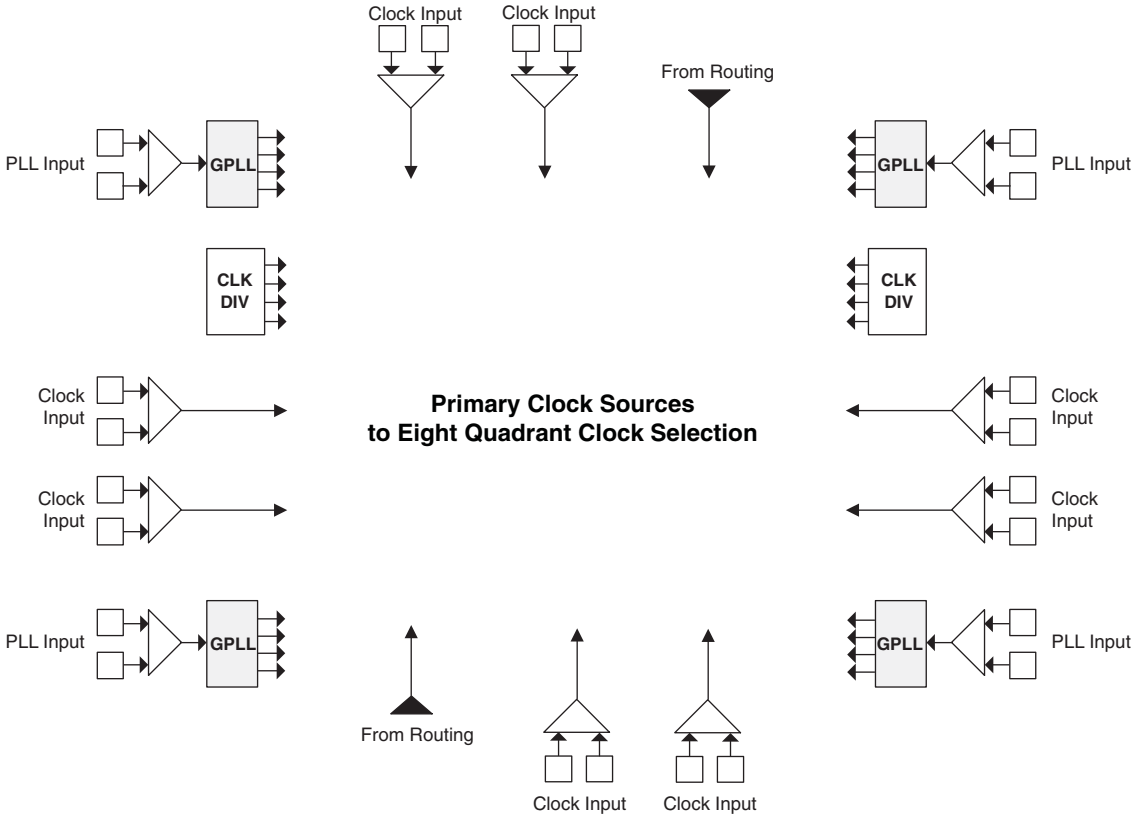
Clock Distribution Network

LatticeXP2 devices have eight quadrant-based primary clocks and between six and eight flexible region-based secondary clocks/control signals. Two high performance edge clocks are available on each edge of the device to support high speed interfaces. The clock inputs are selected from external I/Os, the sysCLOCK PLLs, or routing. Clock inputs are fed throughout the chip via the primary, secondary and edge clock networks.

Primary Clock Sources

LatticeXP2 devices derive primary clocks from four sources: PLL outputs, CLKDIV outputs, dedicated clock inputs and routing. LatticeXP2 devices have two to four sysCLOCK PLLs, located in the four corners of the device. There are eight dedicated clock inputs, two on each side of the device. Figure 2-6 shows the primary clock sources.

Figure 2-6. Primary Clock Sources for XP2-17

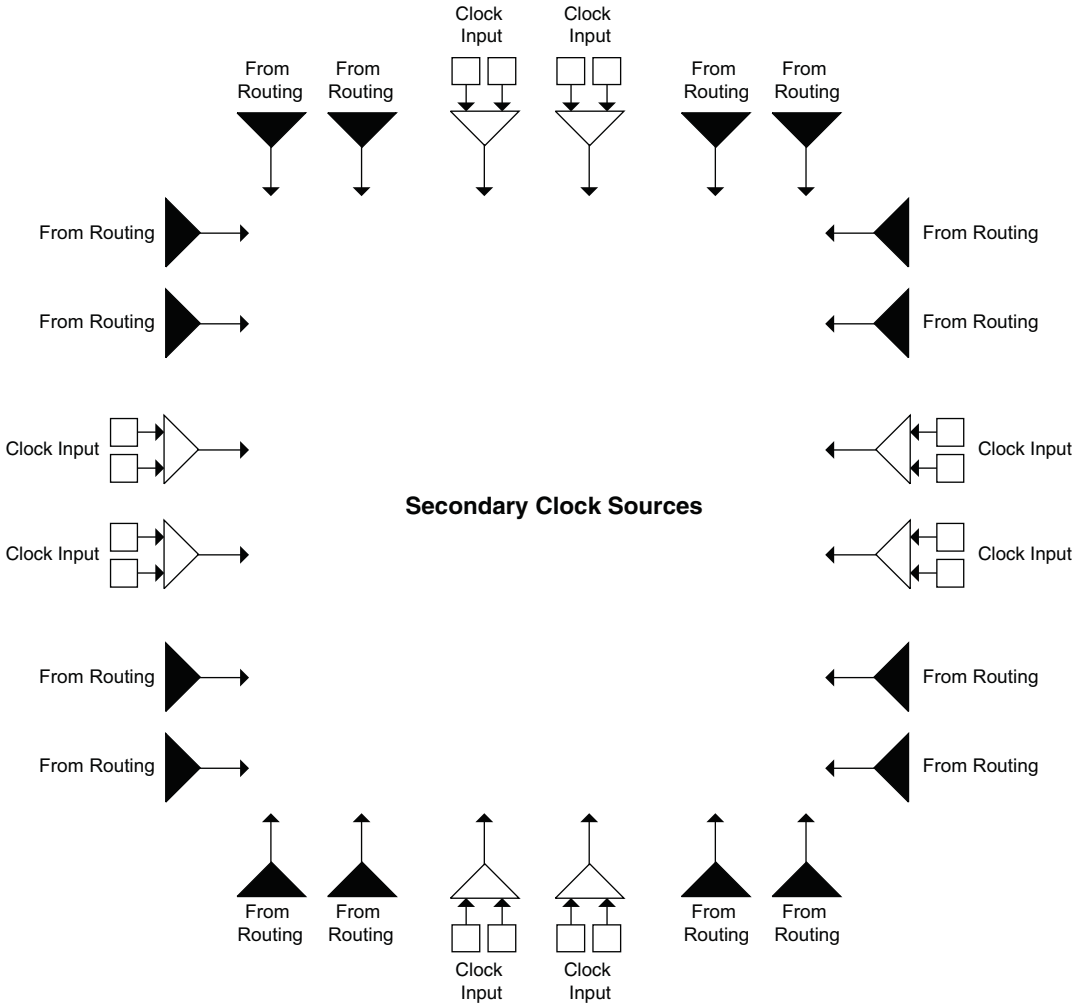


Note: This diagram shows sources for the XP2-17 device. Smaller LatticeXP2 devices have two GPLLs.

Secondary Clock/Control Sources

LatticeXP2 devices derive secondary clocks (SC0 through SC7) from eight dedicated clock input pads and the rest from routing. Figure 2-7 shows the secondary clock sources.

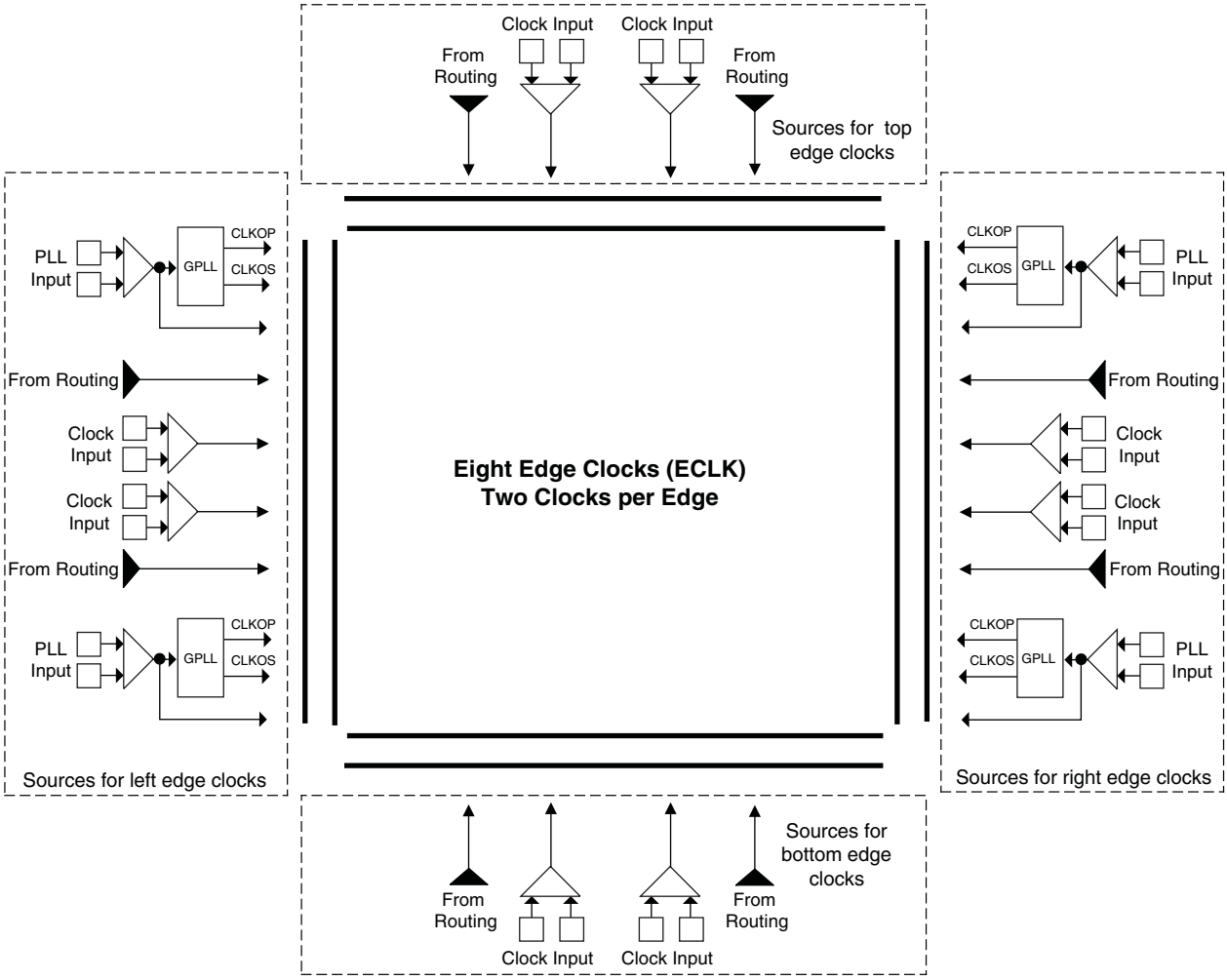
Figure 2-7. Secondary Clock Sources



Edge Clock Sources

Edge clock resources can be driven from a variety of sources at the same edge. Edge clock resources can be driven from adjacent edge clock PIOs, primary clock PIOs, PLLs and clock dividers as shown in Figure 2-8.

Figure 2-8. Edge Clock Sources

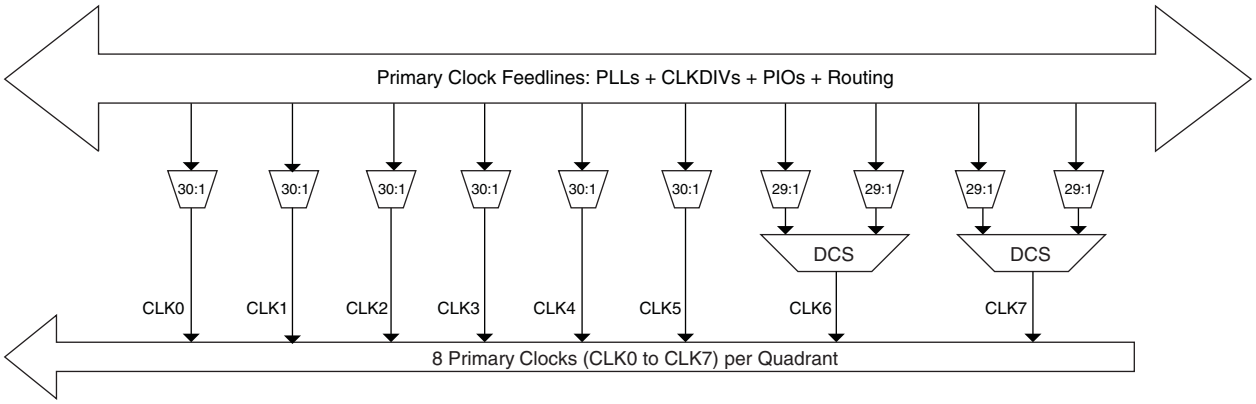


Note: This diagram shows sources for the XP2-17 device. Smaller LatticeXP2 devices have two GPLLs.

Primary Clock Routing

The clock routing structure in LatticeXP2 devices consists of a network of eight primary clock lines (CLK0 through CLK7) per quadrant. The primary clocks of each quadrant are generated from muxes located in the center of the device. All the clock sources are connected to these muxes. Figure 2-9 shows the clock routing for one quadrant. Each quadrant mux is identical. If desired, any clock can be routed globally.

Figure 2-9. Per Quadrant Primary Clock Selection

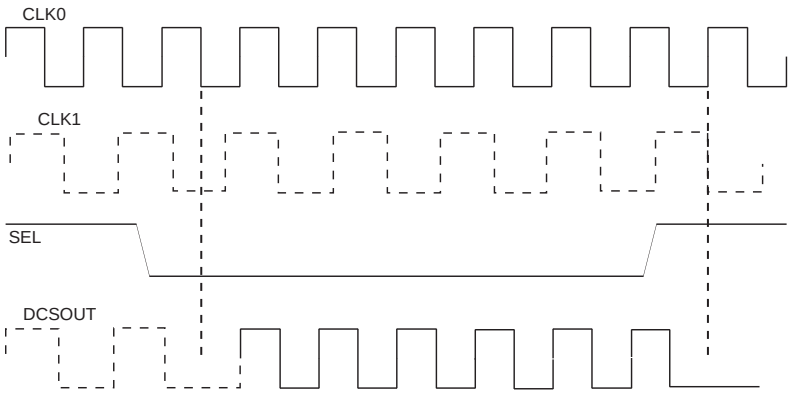


Dynamic Clock Select (DCS)

The DCS is a smart multiplexer function available in the primary clock routing. It switches between two independent input clock sources without any glitches or runt pulses. This is achieved irrespective of when the select signal is toggled. There are two DCS blocks per quadrant; in total, eight DCS blocks per device. The inputs to the DCS block come from the center muxes. The output of the DCS is connected to primary clocks CLK6 and CLK7 (see Figure 2-9).

Figure 2-10 shows the timing waveforms of the default DCS operating mode. The DCS block can be programmed to other modes. For more information on the DCS, please see TN1126, *LatticeXP2 sysCLOCK PLL Design and Usage Guide*.

Figure 2-10. DCS Waveforms



Secondary Clock/Control Routing

Secondary clocks in the LatticeXP2 devices are region-based resources. EBR rows, DSP rows and a special vertical routing channel bound the secondary clock regions. This special vertical routing channel aligns with either the left edge of the center DSP block in the DSP row or the center of the DSP row. Figure 2-11 shows this special vertical routing channel and the eight secondary clock regions for the LatticeXP2-40. LatticeXP2-30 and smaller

devices have six secondary clock regions. All devices in the LatticeXP2 family have eight secondary clock resources per region (SC0 to SC7).

The secondary clock muxes are located in the center of the device. Figure 2-12 shows the mux structure of the secondary clock routing. Secondary clocks SC0 to SC3 are used for high fan-out control and SC4 to SC7 are used for clock signals.

Figure 2-11. Secondary Clock Regions XP2-40

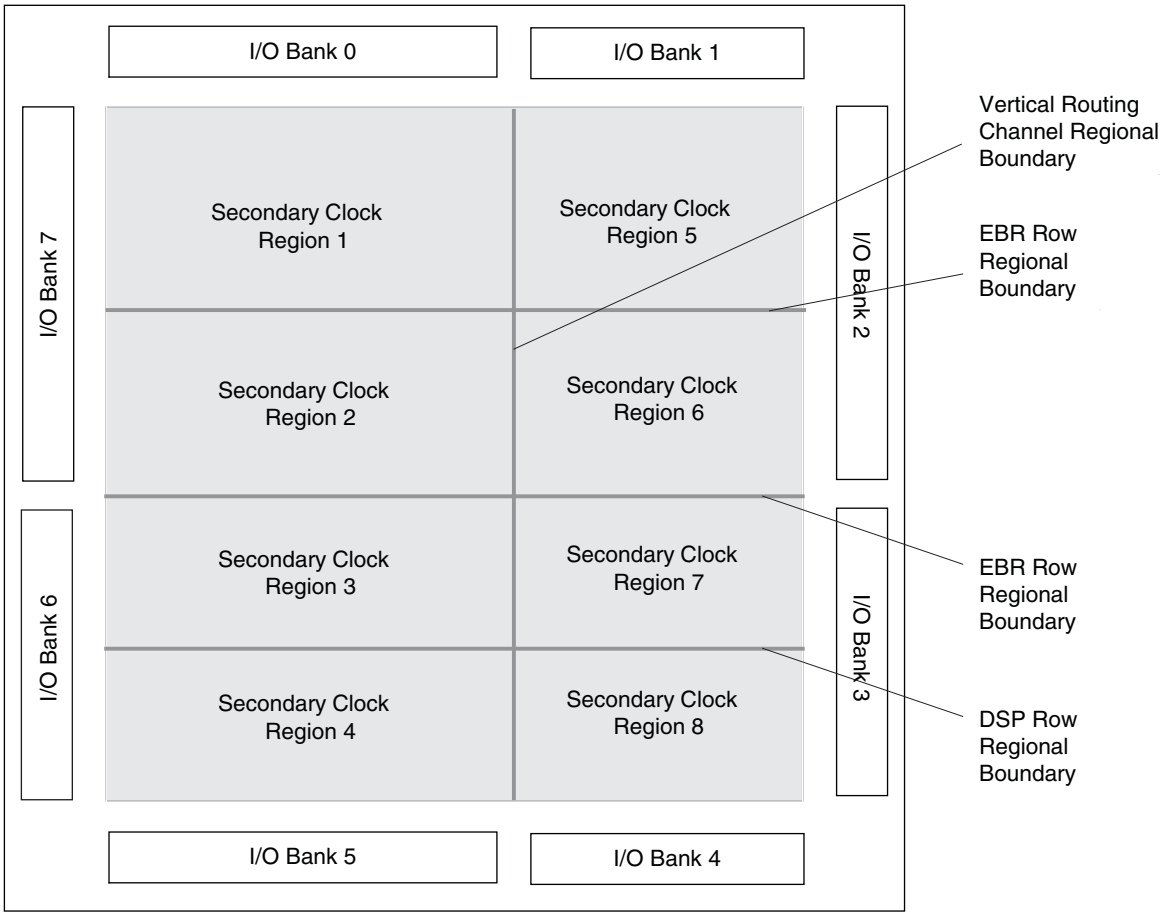
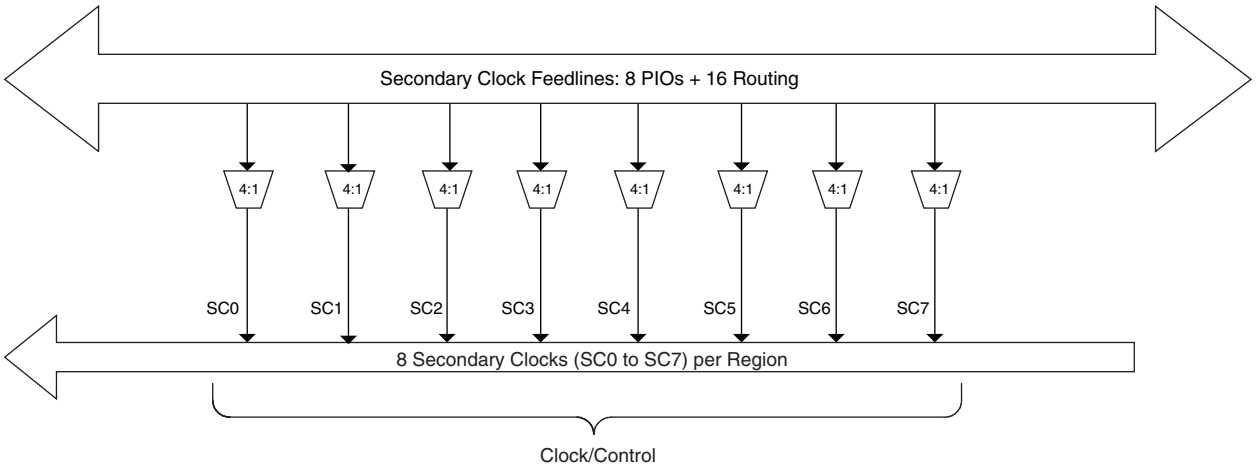


Figure 2-12. Per Region Secondary Clock Selection



Slice Clock Selection

Figure 2-13 shows the clock selections and Figure 2-14 shows the control selections for Slice0 through Slice2. All the primary clocks and the four secondary clocks are routed to this clock selection mux. Other signals, via routing, can be used as clock inputs to the slices. Slice controls are generated from the secondary clocks or other signals connected via routing.

If none of the signals are selected for both clock and control, then the default value of the mux output is 1. Slice 3 does not have any registers; therefore it does not have the clock or control muxes.

Figure 2-13. Slice0 through Slice2 Clock Selection

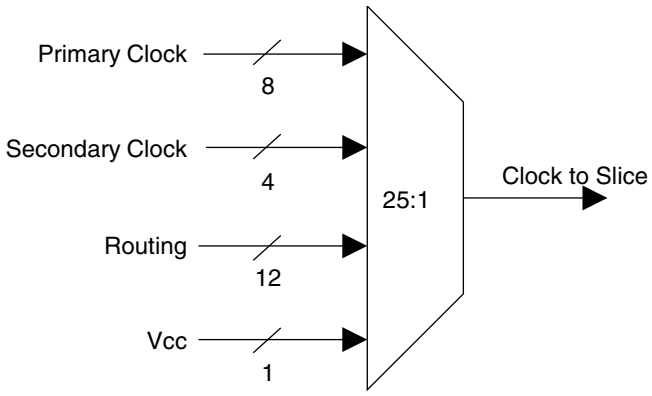
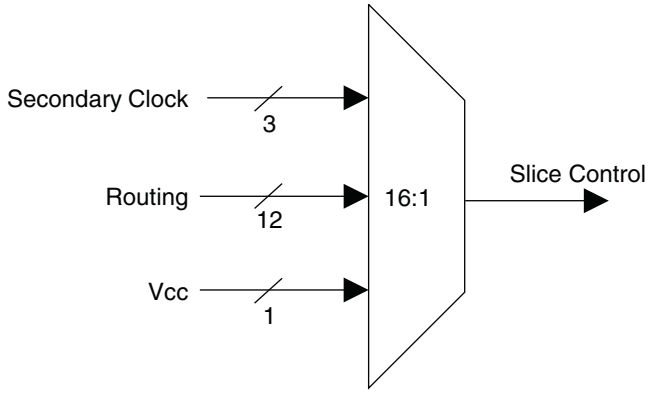


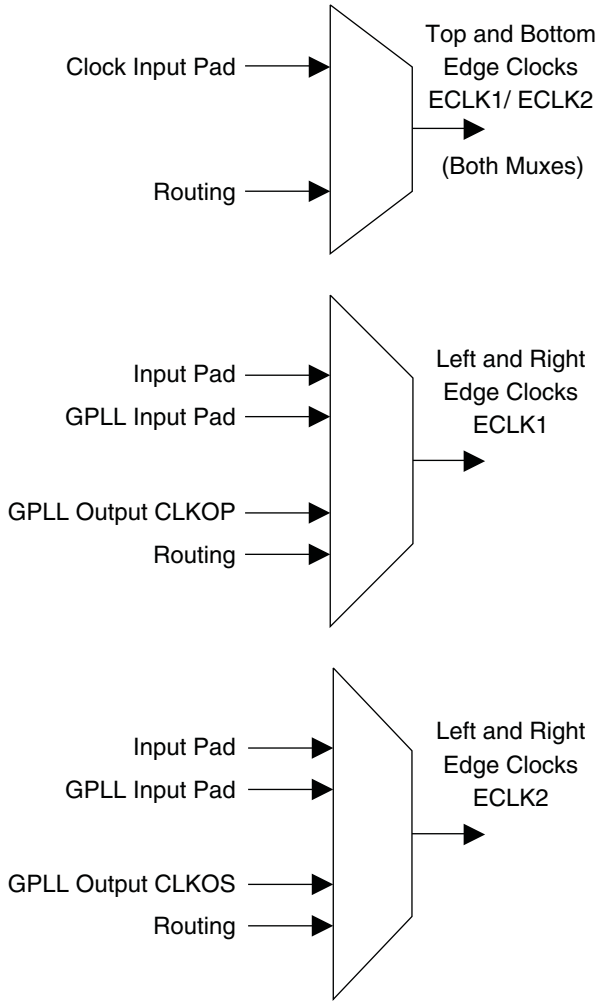
Figure 2-14. Slice0 through Slice2 Control Selection



Edge Clock Routing

LatticeXP2 devices have eight high-speed edge clocks that are intended for use with the PIOs in the implementation of high-speed interfaces. Each device has two edge clocks per edge. Figure 2-15 shows the selection muxes for these clocks.

Figure 2-15. Edge Clock Mux Connections



sysMEM Memory

LatticeXP2 devices contains a number of sysMEM Embedded Block RAM (EBR). The EBR consists of 18 Kbit RAM with dedicated input and output registers.

sysMEM Memory Block

The sysMEM block can implement single port, dual port or pseudo dual port memories. Each block can be used in a variety of depths and widths as shown in Table 2-5. FIFOs can be implemented in sysMEM EBR blocks by using support logic with PFUs. The EBR block supports an optional parity bit for each data byte to facilitate parity checking. EBR blocks provide byte-enable support for configurations with18-bit and 36-bit data widths.

Table 2-5. sysMEM Block Configurations

Memory Mode	Configurations
Single Port	16,384 x 1 8,192 x 2 4,096 x 4 2,048 x 9 1,024 x 18 512 x 36
True Dual Port	16,384 x 1 8,192 x 2 4,096 x 4 2,048 x 9 1,024 x 18
Pseudo Dual Port	16,384 x 1 8,192 x 2 4,096 x 4 2,048 x 9 1,024 x 18 512 x 36

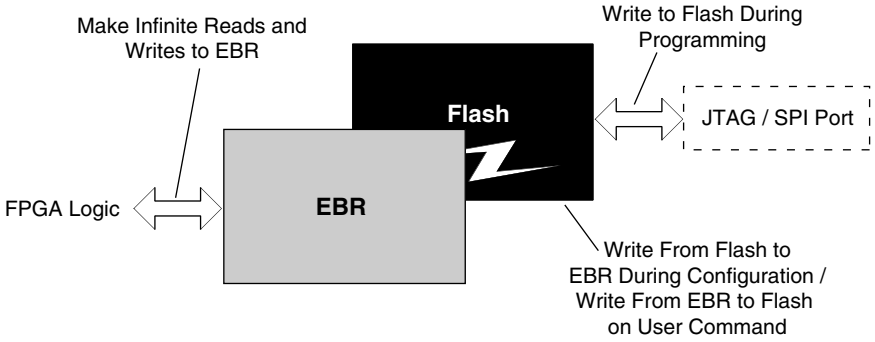
Bus Size Matching

All of the multi-port memory modes support different widths on each of the ports. The RAM bits are mapped LSB word 0 to MSB word 0, LSB word 1 to MSB word 1, and so on. Although the word size and number of words for each port varies, this mapping scheme applies to each port.

FlashBAK EBR Content Storage

All the EBR memory in the LatticeXP2 is shadowed by Flash memory. Optionally, initialization values for the memory blocks can be defined using the Lattice ispLEVER tools. The initialization values are loaded into the Flash memory during device programming and into the SRAM at power up or whenever the device is reconfigured. This feature is ideal for the storage of a variety of information such as look-up tables and microprocessor code. It is also possible to write the current contents of the EBR memory back to Flash memory. This capability is useful for the storage of data such as error codes and calibration information. For additional information on the FlashBAK capability see TN1141, *LatticeXP2 sysCONFIG Usage Guide.*

Figure 2-16. FlashBAK Technology



Memory Cascading

Larger and deeper blocks of RAMs can be created using EBR sysMEM Blocks. Typically, the Lattice design tools cascade memory transparently, based on specific design inputs.

Single, Dual and Pseudo-Dual Port Modes

In all the sysMEM RAM modes the input data and address for the ports are registered at the input of the memory array. The output data of the memory is optionally registered at the output.

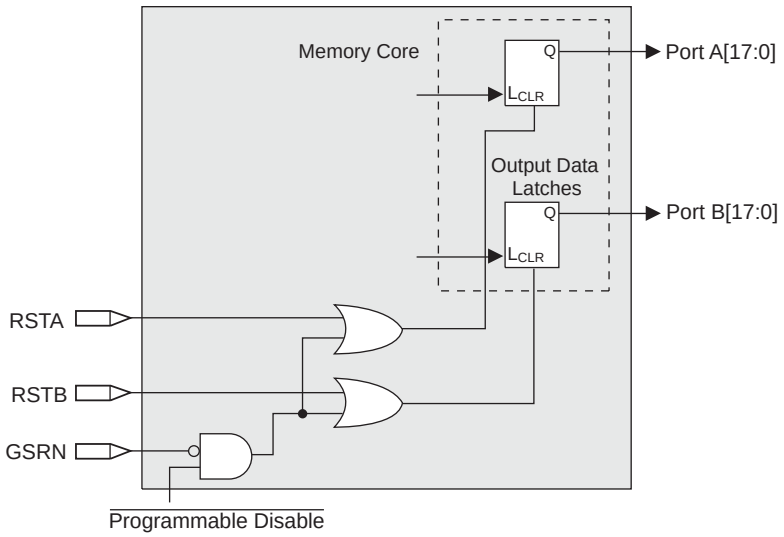
EBR memory supports three forms of write behavior for single port or dual port operation:

1. Normal – Data on the output appears only during a read cycle. During a write cycle, the data (at the current address) does not appear on the output. This mode is supported for all data widths.
2. Write Through – A copy of the input data appears at the output of the same port during a write cycle. This mode is supported for all data widths.
3. Read-Before-Write – When new data is being written, the old content of the address appears at the output. This mode is supported for x9, x18 and x36 data widths.

Memory Core Reset

The memory array in the EBR utilizes latches at the A and B output ports. These latches can be reset asynchronously or synchronously. RSTA and RSTB are local signals, which reset the output latches associated with Port A and Port B respectively. GSRN, the global reset signal, resets both ports. The output data latches and associated resets for both ports are as shown in Figure 2-17.

Figure 2-17. Memory Core Reset

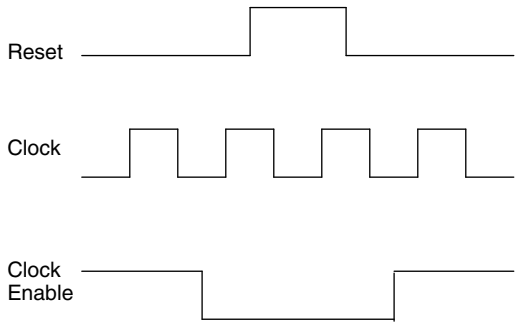


For further information on the sysMEM EBR block, please see TN1137, *LatticeXP2 Memory Usage Guide.*

EBR Asynchronous Reset

EBR asynchronous reset or GSR (if used) can only be applied if all clock enables are low for a clock cycle before the reset is applied and released a clock cycle after the low-to-high transition of the reset signal, as shown in Figure 2-18. The GSR input to the EBR is always asynchronous.

Figure 2-18. EBR Asynchronous Reset (Including GSR) Timing Diagram



If all clock enables remain enabled, the EBR asynchronous reset or GSR may only be applied and released after the EBR read and write clock inputs are in a steady state condition for a minimum of $1/f_{MAX}$ (EBR clock). The reset release must adhere to the EBR synchronous reset setup time before the next active read or write clock edge.

If an EBR is pre-loaded during configuration, the GSR input must be disabled or the release of the GSR during device Wake Up must occur before the release of the device I/Os becoming active.

These instructions apply to all EBR RAM and ROM implementations.

Note that there are no reset restrictions if the EBR synchronous reset is used and the EBR GSR input is disabled.

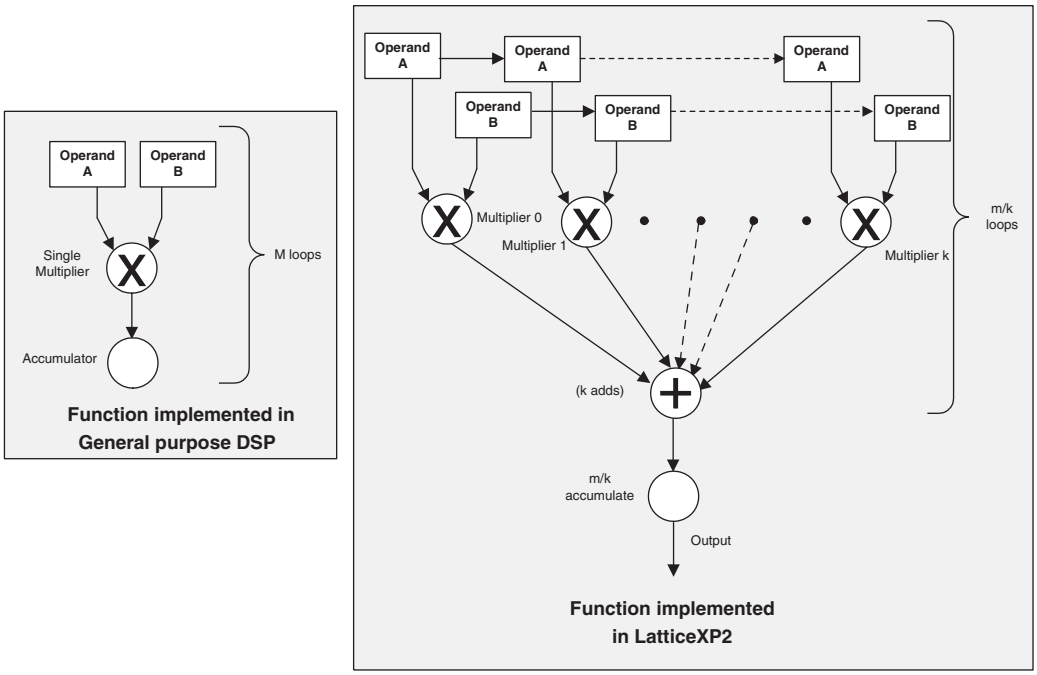
sysDSP™ Block

The LatticeXP2 family provides a sysDSP block making it ideally suited for low cost, high performance Digital Signal Processing (DSP) applications. Typical functions used in these applications include Bit Correlators, Fast Fourier Transform (FFT) functions, Finite Impulse Response (FIR) Filter, Reed-Solomon Encoder/Decoder, Turbo Encoder/Decoder and Convolutional Encoder/Decoder. These complex signal processing functions use similar building blocks such as multiply-adders and multiply-accumulators.

sysDSP Block Approach Compare to General DSP

Conventional general-purpose DSP chips typically contain one to four (Multiply and Accumulate) MAC units with fixed data-width multipliers; this leads to limited parallelism and limited throughput. Their throughput is increased by higher clock speeds. The LatticeXP2 family, on the other hand, has many DSP blocks that support different data-widths. This allows the designer to use highly parallel implementations of DSP functions. The designer can optimize the DSP performance vs. area by choosing appropriate levels of parallelism. Figure 2-19 compares the fully serial and the mixed parallel and serial implementations.

Figure 2-19. Comparison of General DSP and LatticeXP2 Approaches



sysDSP Block Capabilities

The sysDSP block in the LatticeXP2 family supports four functional elements in three 9, 18 and 36 data path widths. The user selects a function element for a DSP block and then selects the width and type (signed/unsigned) of its operands. The operands in the LatticeXP2 family sysDSP Blocks can be either signed or unsigned but not

mixed within a function element. Similarly, the operand widths cannot be mixed within a block. DSP elements can be concatenated.

The resources in each sysDSP block can be configured to support the following four elements:

- MULT (Multiply)
- MAC (Multiply, Accumulate)
- MULTADDSUB (Multiply, Addition/Subtraction)
- MULTADDSUBSUM (Multiply, Addition/Subtraction, Accumulate)

The number of elements available in each block depends on the width selected from the three available options: x9, x18, and x36. A number of these elements are concatenated for highly parallel implementations of DSP functions. Table 2-6 shows the capabilities of the block.

Table 2-6. Maximum Number of Elements in a Block

Width of Multiply	x9	x18	x36
MULT	8	4	1
MAC	2	2	—
MULTADDSUB	4	2	—
MULTADDSUBSUM	2	1	—

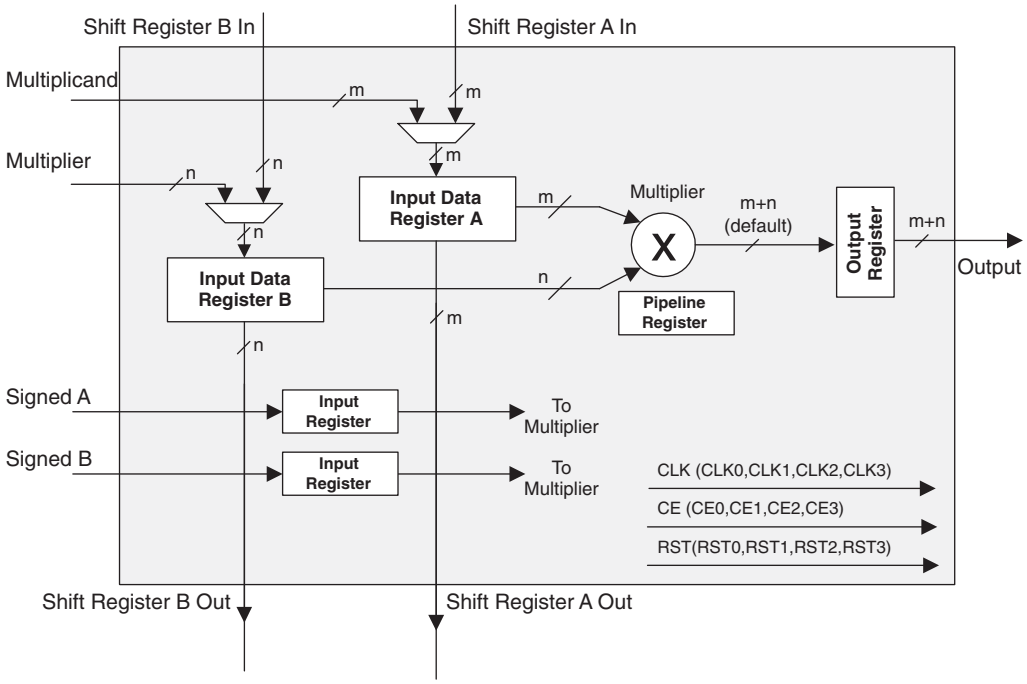
Some options are available in four elements. The input register in all the elements can be directly loaded or can be loaded as shift register from previous operand registers. By selecting ‘dynamic operation’ the following operations are possible:

- In the ‘Signed/Unsigned’ options the operands can be switched between signed and unsigned on every cycle.
- In the ‘Add/Sub’ option the Accumulator can be switched between addition and subtraction on every cycle.
- The loading of operands can switch between parallel and serial operations.

MULT sysDSP Element

This multiplier element implements a multiply with no addition or accumulator nodes. The two operands, A and B, are multiplied and the result is available at the output. The user can enable the input/output and pipeline registers. Figure 2-20 shows the MULT sysDSP element.

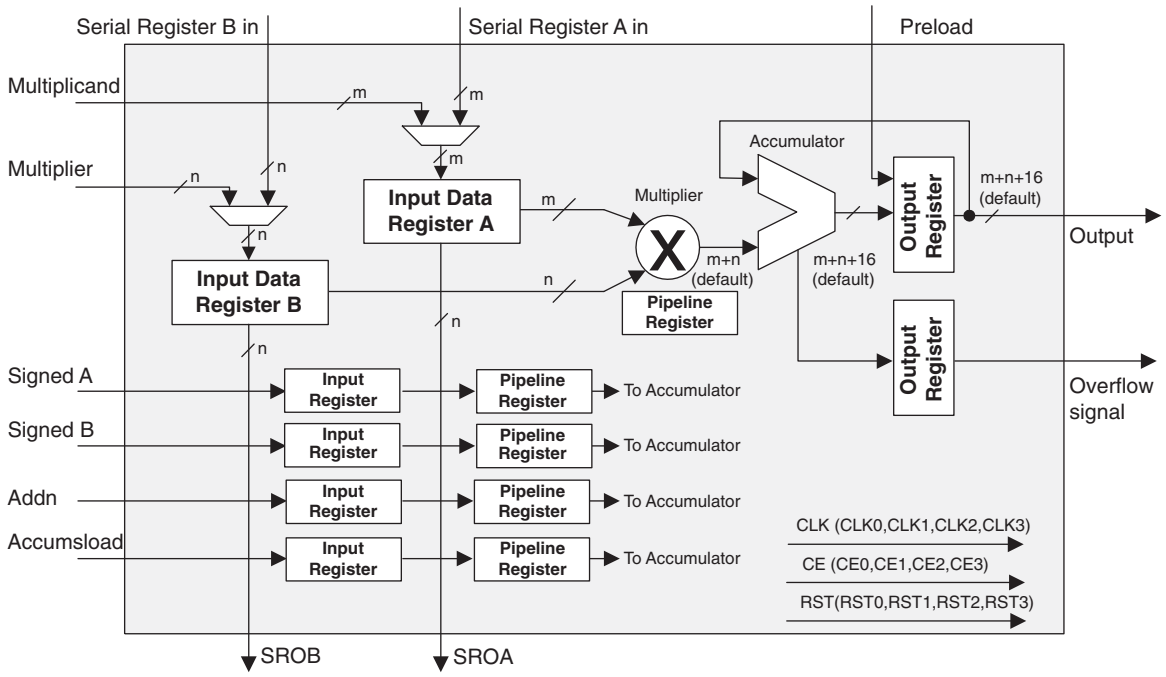
Figure 2-20. MULT sysDSP Element



MAC sysDSP Element

In this case, the two operands, A and B, are multiplied and the result is added with the previous accumulated value. This accumulated value is available at the output. The user can enable the input and pipeline registers but the output register is always enabled. The output register is used to store the accumulated value. The Accumulators in the DSP blocks in LatticeXP2 family can be initialized dynamically. A registered overflow signal is also available. The overflow conditions are provided later in this document. Figure 2-21 shows the MAC sysDSP element.

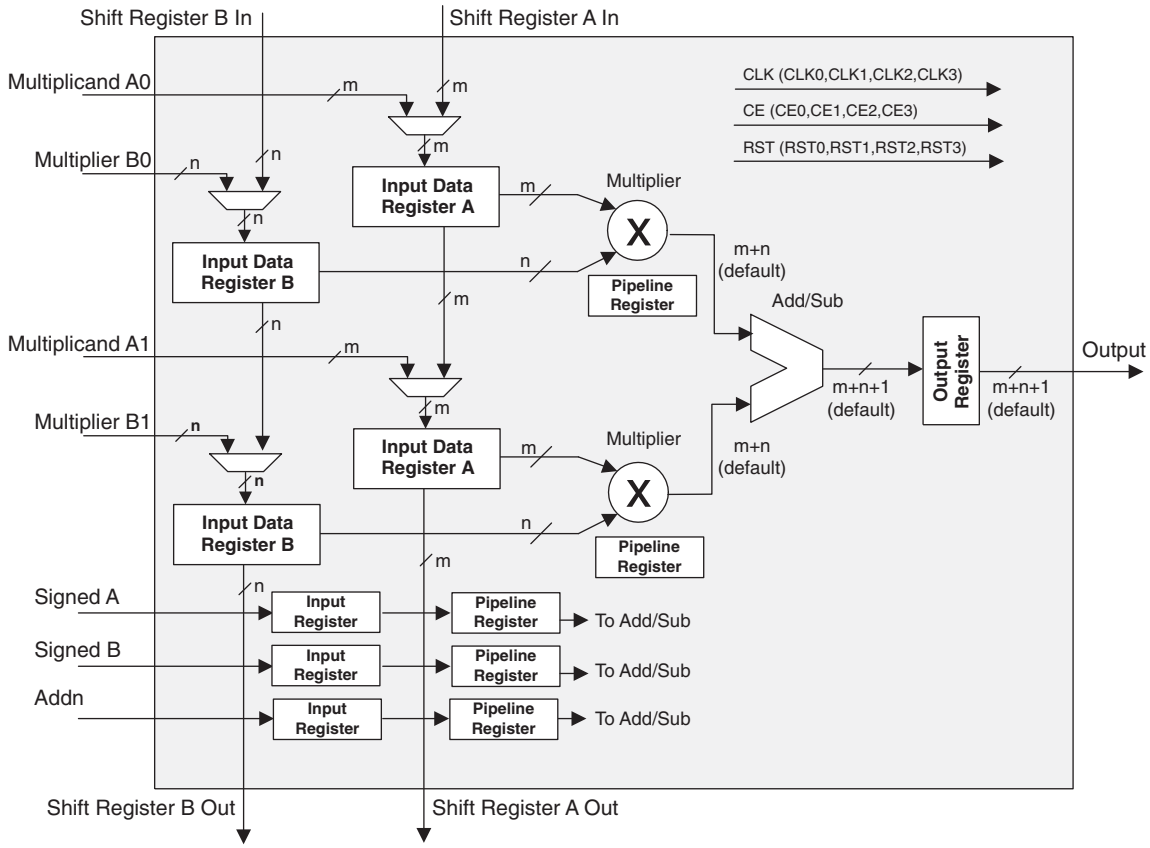
Figure 2-21. MAC sysDSP



MULTADDSUB sysDSP Element

In this case, the operands A0 and B0 are multiplied and the result is added/subtracted with the result of the multiplier operation of operands A1 and B1. The user can enable the input, output and pipeline registers. Figure 2-22 shows the MULTADDSUB sysDSP element.

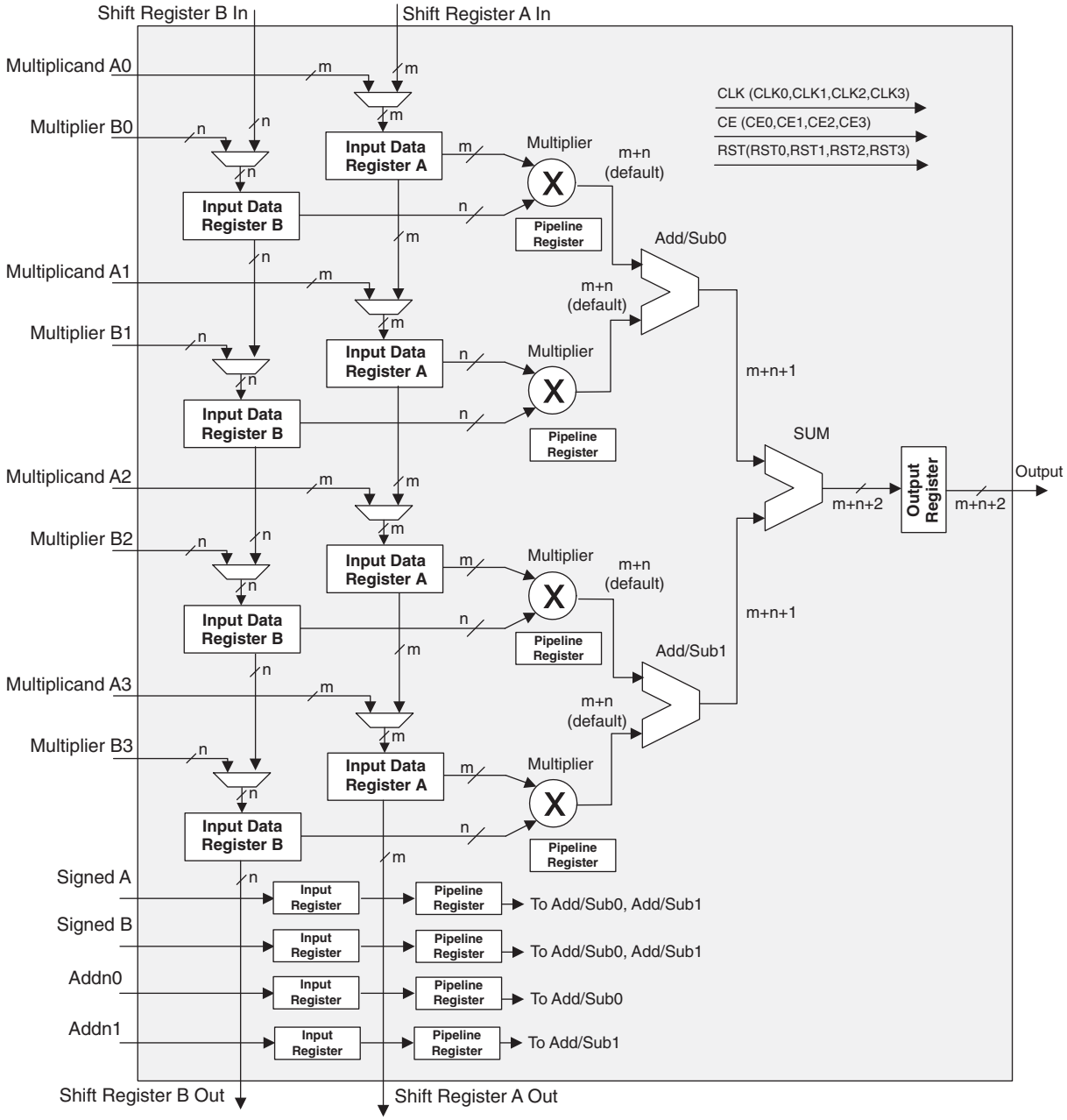
Figure 2-22. MULTADDSUB



MULTADDSUBSUM sysDSP Element

In this case, the operands A0 and B0 are multiplied and the result is added/subtracted with the result of the multiplier operation of operands A1 and B1. Additionally the operands A2 and B2 are multiplied and the result is added/subtracted with the result of the multiplier operation of operands A3 and B3. The result of both addition/subtraction are added in a summation block. The user can enable the input, output and pipeline registers. Figure 2-23 shows the MULTADDSUBSUM sysDSP element.

Figure 2-23. MULTADDSUBSUM



Clock, Clock Enable and Reset Resources

Global Clock, Clock Enable (CE) and Reset (RST) signals from routing are available to every DSP block. From four clock sources (CLK0, CLK1, CLK2, CLK3) one clock is selected for each input register, pipeline register and output

register. Similarly, CE and RST are selected from their four respective sources (CE0, CE1, CE2, CE3 and RST0, RST1, RST2, RST3) at each input register, pipeline register and output register.

Signed and Unsigned with Different Widths

The DSP block supports other widths, in addition to x9, x18 and x36 widths, of signed and unsigned multipliers. For unsigned operands, unused upper data bits should be filled to create a valid x9, x18 or x36 operand. For signed two's complement operands, sign extension of the most significant bit should be performed until x9, x18 or x36 width is reached. Table 2-7 provides an example of this.

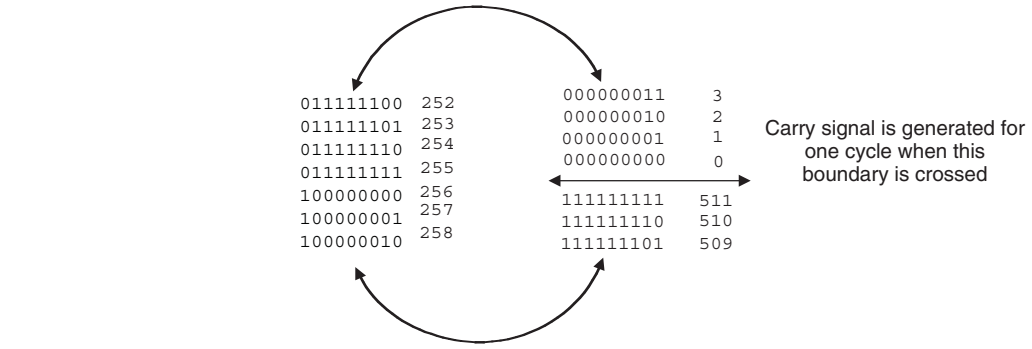
Table 2-7. Sign Extension Example

Number	Unsigned	Unsigned 9-bit	Unsigned 18-bit	Signed	Two's Complement Signed 9 Bits	Two's Complement Signed 18 Bits
+5	0101	000000101	000000000000000101	0101	000000101	000000000000000101
-6	N/A	N/A	N/A	1010	111111010	111111111111111010

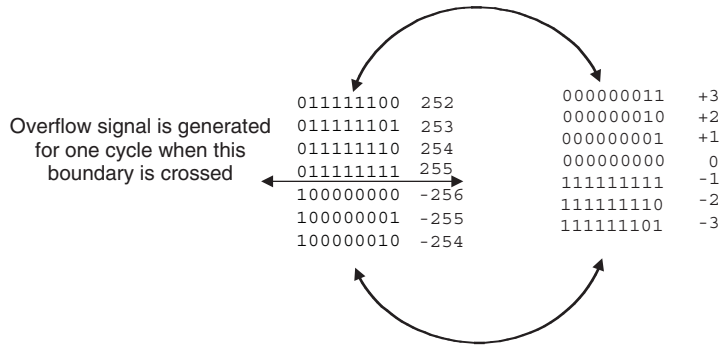
OVERFLOW Flag from MAC

The sysDSP block provides an overflow output to indicate that the accumulator has overflowed. "Roll-over" occurs and an overflow signal is indicated when any of the following is true: two unsigned numbers are added and the result is a smaller number than the accumulator, two positive numbers are added with a negative sum or two negative numbers are added with a positive sum. Note that when overflow occurs the overflow flag is present for only one cycle. By counting these overflow pulses in FPGA logic, larger accumulators can be constructed. The conditions for the overflow signal for signed and unsigned operands are listed in Figure 2-24.

Figure 2-24. Accumulator Overflow/Underflow



Unsigned Operation



Signed Operation

IPexpress™

The user can access the sysDSP block via the ispLEVER IPexpress tool, which provides the option to configure each DSP module (or group of modules), or by direct HDL instantiation. In addition, Lattice has partnered with The MathWorks® to support instantiation in the Simulink® tool, a graphical simulation environment. Simulink works with ispLEVER to dramatically shorten the DSP design cycle in Lattice FPGAs.

Optimized DSP Functions

Lattice provides a library of optimized DSP IP functions. Some of the IP cores planned for the LatticeXP2 DSP include the Bit Correlator, FFT functions, FIR Filter, Reed-Solomon Encoder/Decoder, Turbo Encoder/Decoder and Convolutional Encoder/Decoder. Please contact Lattice to obtain the latest list of available DSP IP cores.

Resources Available in the LatticeXP2 Family

Table 2-8 shows the maximum number of multipliers for each member of the LatticeXP2 family. Table 2-9 shows the maximum available EBR RAM Blocks and Serial TAG Memory bits in each LatticeXP2 device. EBR blocks, together with Distributed RAM can be used to store variables locally for fast DSP operations.

Table 2-8. Maximum Number of DSP Blocks in the LatticeXP2 Family

Device	DSP Block	9x9 Multiplier	18x18 Multiplier	36x36 Multiplier
XP2-5	3	24	12	3
XP2-8	4	32	16	4
XP2-17	5	40	20	5
XP2-30	7	56	28	7
XP2-40	8	64	32	8

Table 2-9. Embedded SRAM/TAG Memory in the LatticeXP2 Family

Device	EBR SRAM Block	Total EBR SRAM (Kbits)	TAG Memory (Bits)
XP2-5	9	166	632
XP2-8	12	221	768
XP2-17	15	276	2184
XP2-30	21	387	2640
XP2-40	48	885	3384

LatticeXP2 DSP Performance

Table 2-10 lists the maximum performance in Millions of MAC (MMAC) operations per second for each member of the LatticeXP2 family.

Table 2-10. DSP Performance

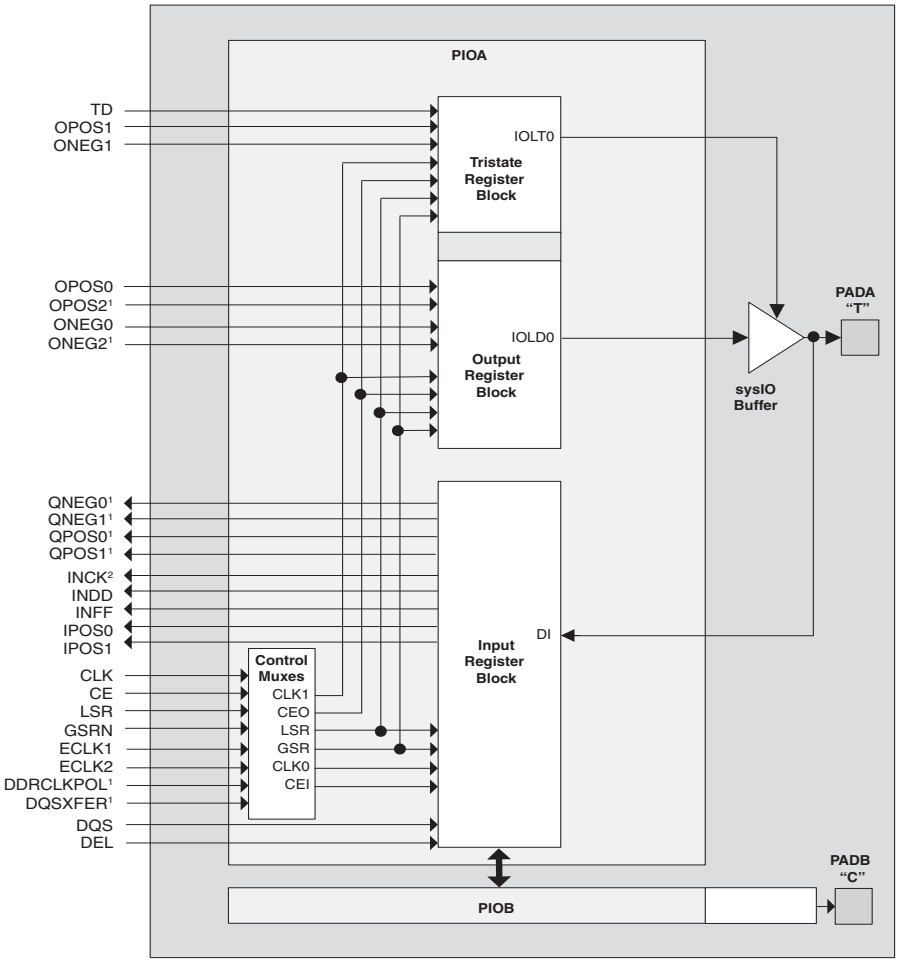
Device	DSP Block	DSP Performance MMAC
XP2-5	3	3,900
XP2-8	4	5,200
XP2-17	5	6,500
XP2-30	7	9,100
XP2-40	8	10,400

For further information on the sysDSP block, please see TN1140, *LatticeXP2 sysDSP Usage Guide.*

Programmable I/O Cells (PIC)

Each PIC contains two PIOs connected to their respective sysIO buffers as shown in Figure 2-25. The PIO Block supplies the output data (DO) and the tri-state control signal (TO) to the sysIO buffer and receives input from the buffer. Table 2-13 provides the PIO signal list.

Figure 2-25. PIC Diagram



1. Signals are available on left/right/bottom edges only.
2. Selected blocks.

Two adjacent PIOs can be joined to provide a differential I/O pair (labeled as “T” and “C”) as shown in Figure 2-25. The PAD Labels “T” and “C” distinguish the two PIOs. Approximately 50% of the PIO pairs on the left and right edges of the device can be configured as true LVDS outputs. All I/O pairs can operate as inputs.

Table 2-11. PIO Signal List

Name	Type	Description
CE	Control from the core	Clock enables for input and output block flip-flops
CLK	Control from the core	System clocks for input and output blocks
ECLK1, ECLK2	Control from the core	Fast edge clocks
LSR	Control from the core	Local Set/Reset
GSRN	Control from routing	Global Set/Reset (active low)
INCK[2]	Input to the core	Input to Primary Clock Network or PLL reference inputs
DQS	Input to PIO	DQS signal from logic (routing) to PIO
INDD	Input to the core	Unregistered data input to core
INFF	Input to the core	Registered input on positive edge of the clock (CLK0)
IPOS0, IPOS1	Input to the core	Double data rate registered inputs to the core
QPOS0[1], QPOS1[1]	Input to the core	Gearbox pipelined inputs to the core
QNEG0[1], QNEG1[1]	Input to the core	Gearbox pipelined inputs to the core
OPOS0, ONEG0, OPOS2, ONEG2	Output data from the core	Output signals from the core for SDR and DDR operation
OPOS1 ONEG1	Tristate control from the core	Signals to Tristate Register block for DDR operation
DEL[3:0]	Control from the core	Dynamic input delay control bits
TD	Tristate control from the core	Tristate signal from the core used in SDR operation
DDRCLKPOL	Control from clock polarity bus	Controls the polarity of the clock (CLK0) that feed the DDR input block
DQSXFER	Control from core	Controls signal to the Output block

1. Signals available on left/right/bottom only.
2. Selected I/O.

PIO

The PIO contains four blocks: an input register block, output register block, tristate register block and a control logic block. These blocks contain registers for operating in a variety of modes along with necessary clock and selection logic.

Input Register Block

The input register blocks for PIOs contain delay elements and registers that can be used to condition high-speed interface signals, such as DDR memory interfaces and source synchronous interfaces, before they are passed to the device core. Figure 2-26 shows the diagram of the input register block.

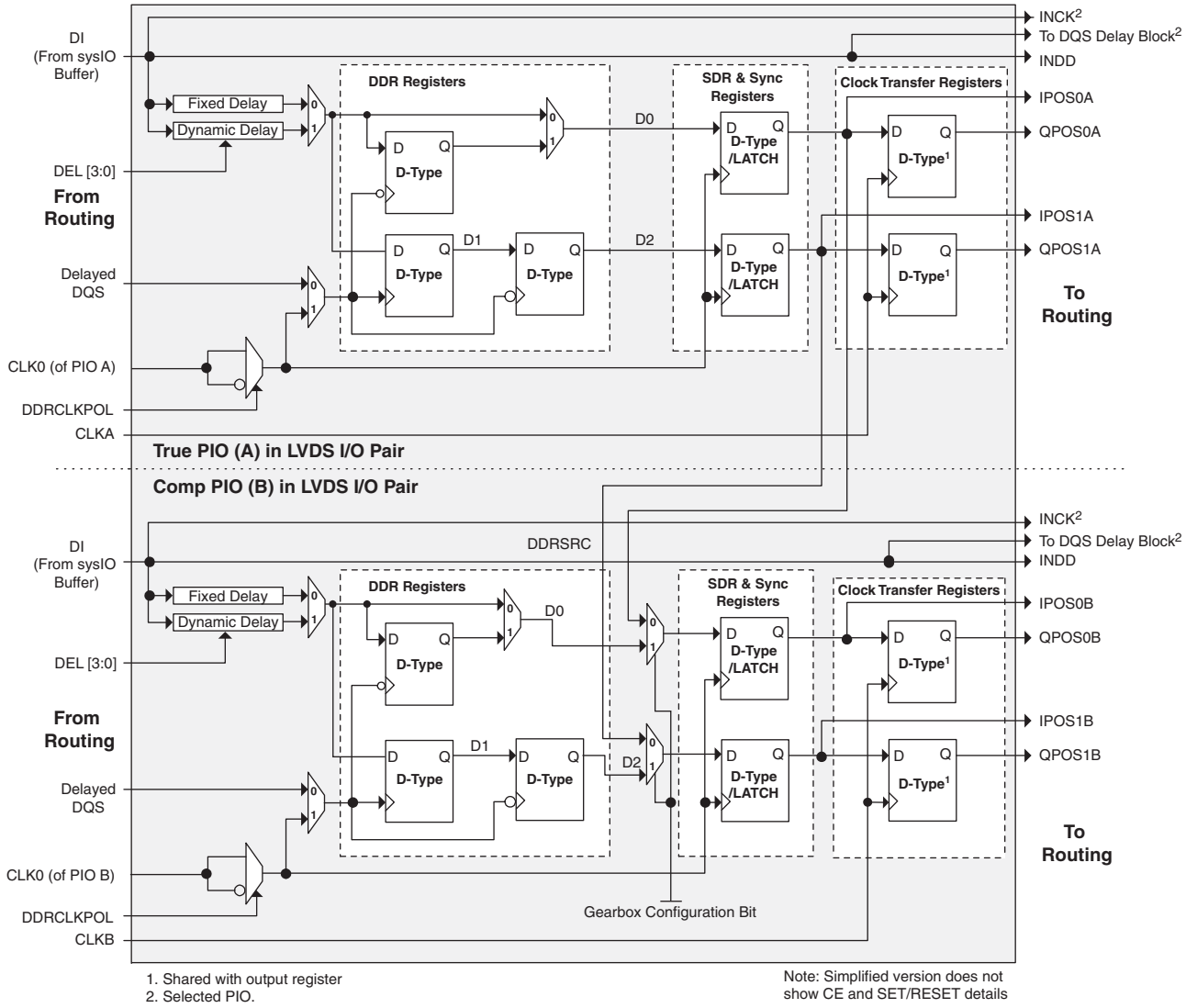
Input signals are fed from the sysIO buffer to the input register block (as signal DI). If desired, the input signal can bypass the register and delay elements and be used directly as a combinatorial signal (INDD), a clock (INCK) and, in selected blocks, the input to the DQS delay block. If an input delay is desired, designers can select either a fixed delay or a dynamic delay DEL[3:0]. The delay, if selected, reduces input register hold time requirements when using a global clock.

The input block allows three modes of operation. In the Single Data Rate (SDR) mode, the data is registered, by one of the registers in the SDR Sync register block, with the system clock. In DDR mode two registers are used to sample the data on the positive and negative edges of the DQS signal which creates two data streams, D0 and D1. D0 and D1 are synchronized with the system clock before entering the core. Further information on this topic can be found in the DDR Memory Support section of this data sheet.

By combining input blocks of the complementary PIOs and sharing registers from output blocks, a gearbox function can be implemented, that takes a double data rate signal applied to PIOA and converts it as four data streams, IPOS0A, IPOS1A, IPOS0B and IPOS1B. Figure 2-26 shows the diagram using this gearbox function. For more information on this topic, please see TN1138, *LatticeXP2 High Speed I/O Interface*.

The signal DDRCLKPOL controls the polarity of the clock used in the synchronization registers. It ensures adequate timing when data is transferred from the DQS to system clock domain. For further discussion on this topic, see the DDR Memory section of this data sheet.

Figure 2-26. Input Register Block



Output Register Block

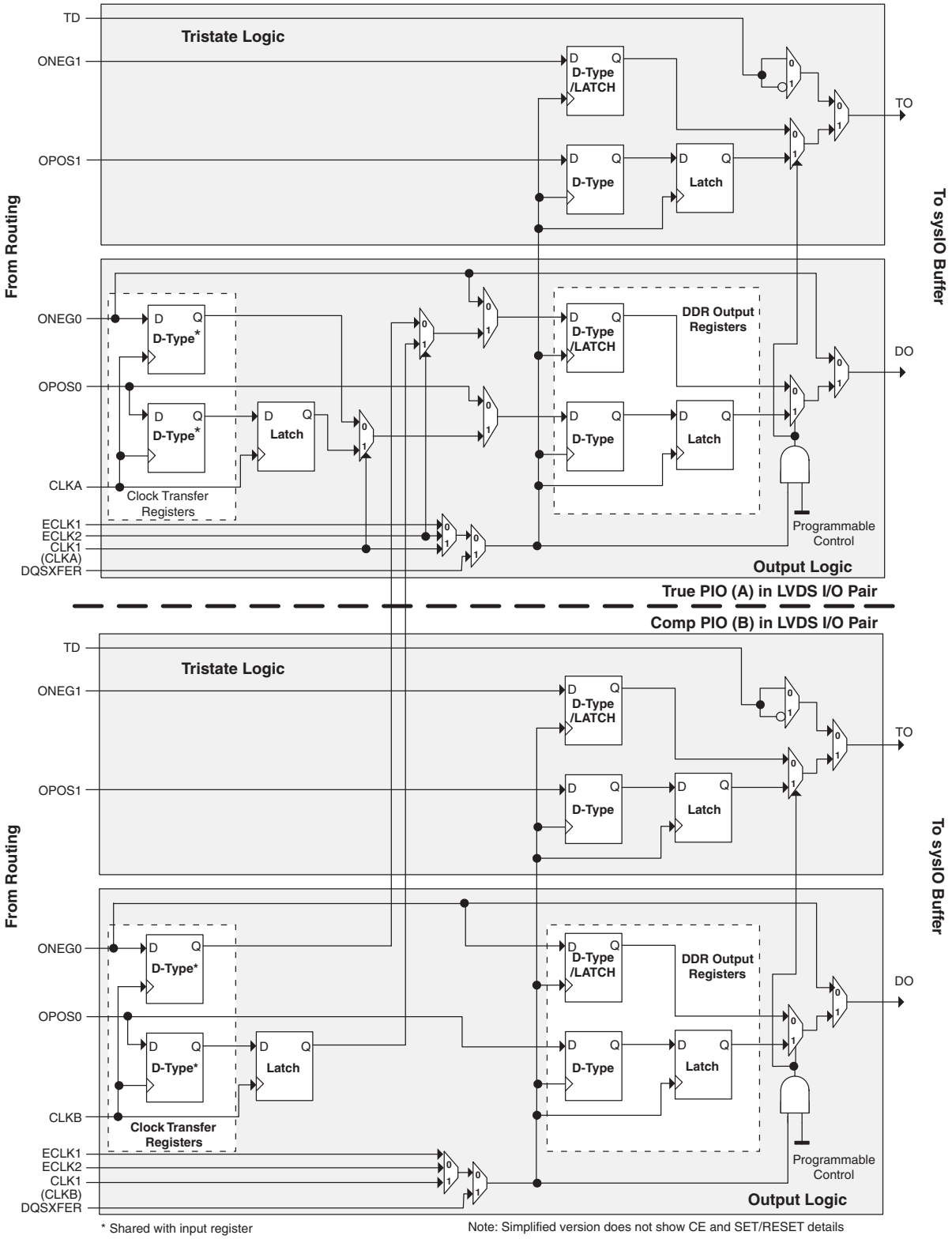
The output register block provides the ability to register signals from the core of the device before they are passed to the sysIO buffers. The blocks on the PIOs on the left, right and bottom contain registers for SDR operation that are combined with an additional latch for DDR operation. Figure 2-27 shows the diagram of the Output Register Block for PIOs.

In SDR mode, ONEG0 feeds one of the flip-flops that then feeds the output. The flip-flop can be configured as a D-type or latch. In DDR mode, ONEG0 and OPOS0 are fed into registers on the positive edge of the clock. At the next clock cycle the registered OPOS0 is latched. A multiplexer running off the same clock cycle selects the correct register to feed the output (D0).

By combining output blocks of the complementary PIOs and sharing some registers from input blocks, a gearbox function can be implemented, to take four data streams ONEG0A, ONEG1A, ONEG1B and ONEG1B. Figure 2-27

shows the diagram using this gearbox function. For more information on this topic, see TN1138, *LatticeXP2 High Speed I/O Interface*.

Figure 2-27. Output and Tristate Block



Tristate Register Block

The tristate register block provides the ability to register tri-state control signals from the core of the device before they are passed to the sysIO buffers. The block contains a register for SDR operation and an additional latch for DDR operation. Figure 2-27 shows the Tristate Register Block with the Output Block

In SDR mode, ONEG1 feeds one of the flip-flops that then feeds the output. The flip-flop can be configured as D-type or latch. In DDR mode, ONEG1 and OPOS1 are fed into registers on the positive edge of the clock. Then in the next clock the registered OPOS1 is latched. A multiplexer running off the same clock cycle selects the correct register for feeding to the output (D0).

Control Logic Block

The control logic block allows the selection and modification of control signals for use in the PIO block. A clock signal is selected from general purpose routing, ECLK1, ECLK2 or a DQS signal (from the programmable DQS pin) and is provided to the input register block. The clock can optionally be inverted.

DDR Memory Support

PICs have additional circuitry to allow implementation of high speed source synchronous and DDR memory interfaces.

PICs have registered elements that support DDR memory interfaces. Interfaces on the left and right edges are designed for DDR memories that support 16 bits of data, whereas interfaces on the top and bottom are designed for memories that support 18 bits of data. One of every 16 PIOs on the left and right and one of every 18 PIOs on the top and bottom contain delay elements to facilitate the generation of DQS signals. The DQS signals feed the DQS buses which span the set of 16 or 18 PIOs. Figure 2-28 and Figure 2-29 show the DQS pin assignments in each set of PIOs.

The exact DQS pins are shown in a dual function in the Logic Signal Connections table in this data sheet. Additional detail is provided in the Signal Descriptions table. The DQS signal from the bus is used to strobe the DDR data from the memory into input register blocks. For additional information on using DDR memory support please see TN1138, *LatticeXP2 High Speed I/O Interface.*

Figure 2-28. DQS Input Routing (Left and Right)

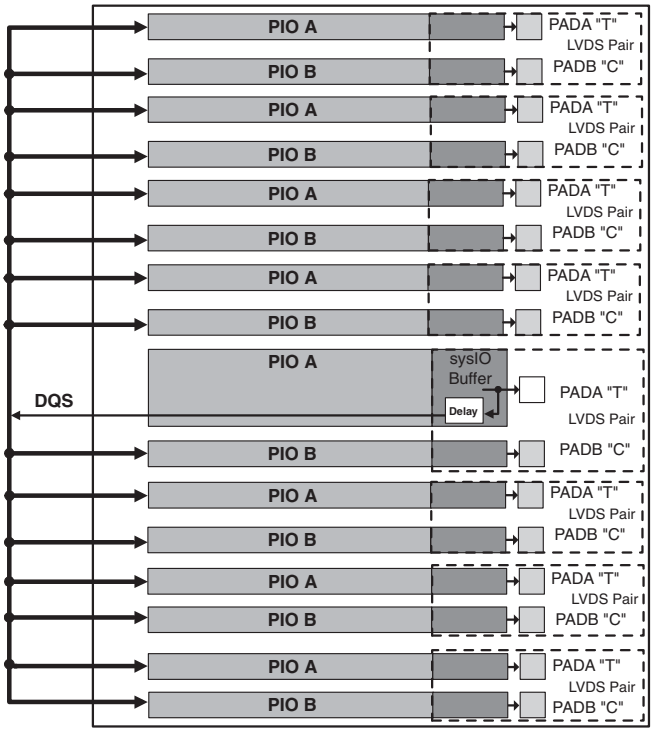
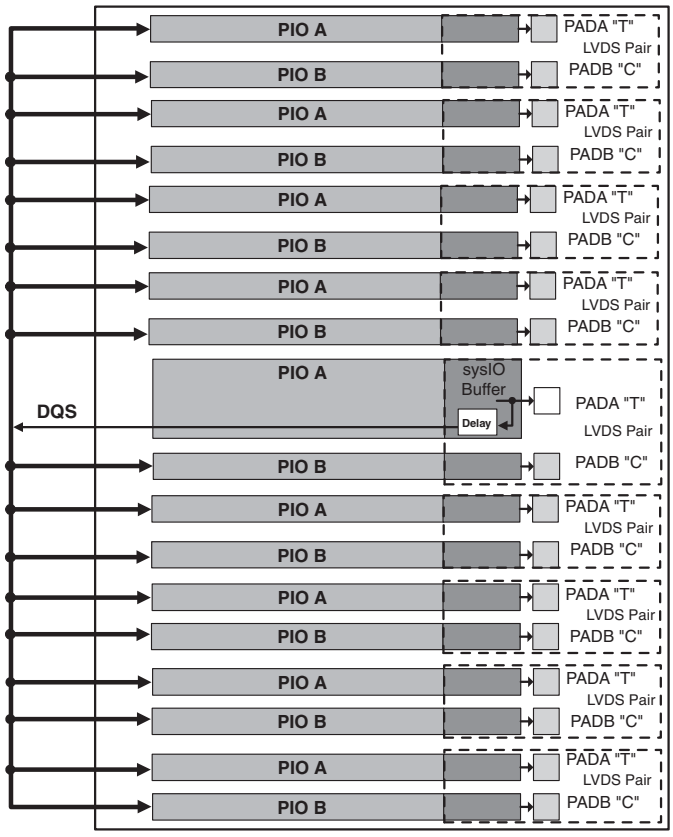


Figure 2-29. DQS Input Routing (Top and Bottom)



DLL Calibrated DQS Delay Block

Source synchronous interfaces generally require the input clock to be adjusted in order to correctly capture data at the input register. For most interfaces a PLL is used for this adjustment. However, in DDR memories the clock, referred to as DQS, is not free-running, and this approach cannot be used. The DQS Delay block provides the required clock alignment for DDR memory interfaces.

The DQS signal (selected PIOs only, as shown in Figure 2-30) feeds from the PAD through a DQS delay element to a dedicated DQS routing resource. The DQS signal also feeds polarity control logic which controls the polarity of the clock to the sync registers in the input register blocks. Figure 2-30 and Figure 2-31 show how the DQS transition signals are routed to the PIOs.

The temperature, voltage and process variations of the DQS delay block are compensated by a set of 6-bit bus calibration signals from two dedicated DLLs (DDR_DLL) on opposite sides of the device. Each DLL compensates DQS delays in its half of the device as shown in Figure 2-30. The DLL loop is compensated for temperature, voltage and process variations by the system clock and feedback loop.

Figure 2-30. Edge Clock, DLL Calibration and DQS Local Bus Distribution

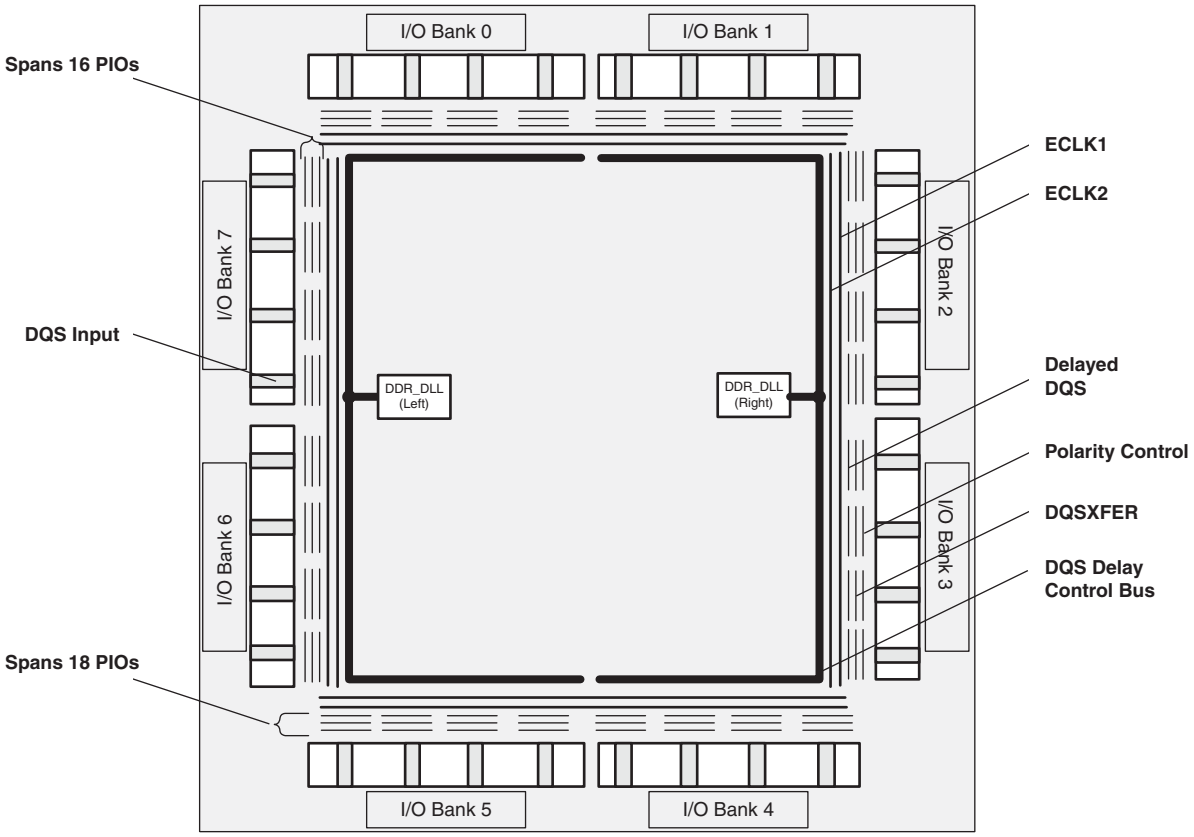
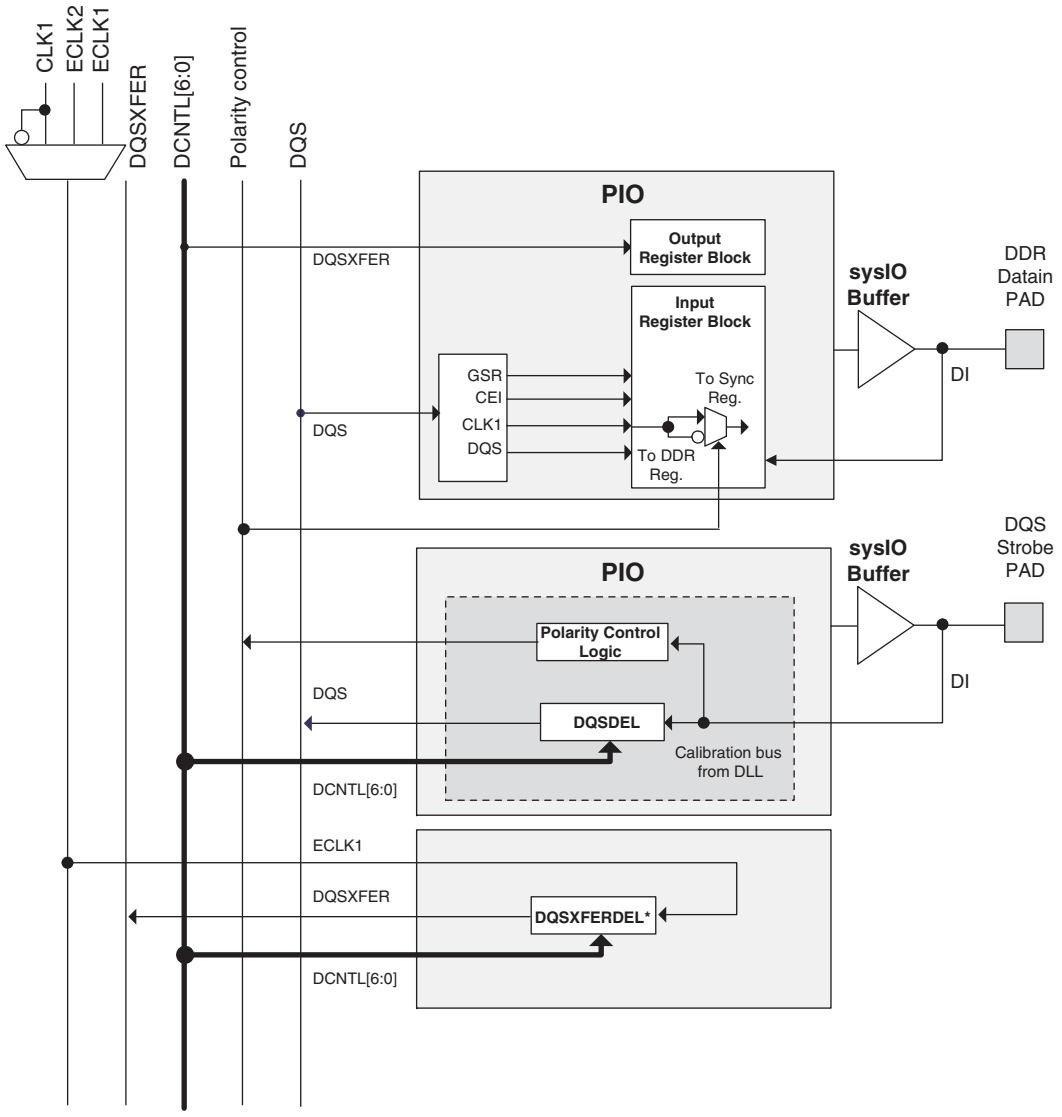


Figure 2-31. DQS Local Bus



*DQSXFERDEL shifts ECLK1 by 90% and is not associated with a particular PIO.

Polarity Control Logic

In a typical DDR memory interface design, the phase relationship between the incoming delayed DQS strobe and the internal system clock (during the READ cycle) is unknown. The LatticeXP2 family contains dedicated circuits to transfer data between these domains. To prevent set-up and hold violations, at the domain transfer between DQS (delayed) and the system clock, a clock polarity selector is used. This changes the edge on which the data is registered in the synchronizing registers in the input register block and requires evaluation at the start of each READ cycle for the correct clock polarity.

Prior to the READ operation in DDR memories, DQS is in tristate (pulled by termination). The DDR memory device drives DQS low at the start of the preamble state. A dedicated circuit detects this transition. This signal is used to control the polarity of the clock to the synchronizing registers.

DQSXFER

LatticeXP2 devices provide a DQSXFER signal to the output buffer to assist it in data transfer to DDR memories that require DQS strobe be shifted 90º. This shifted DQS strobe is generated by the DQSDEL block. The DQSXFER signal runs the span of the data bus.

sysIO Buffer

Each I/O is associated with a flexible buffer referred to as a sysIO buffer. These buffers are arranged around the periphery of the device in groups referred to as banks. The sysIO buffers allow users to implement the wide variety of standards that are found in today's systems including LVCMOS, SSTL, HSTL, LVDS and LVPECL.

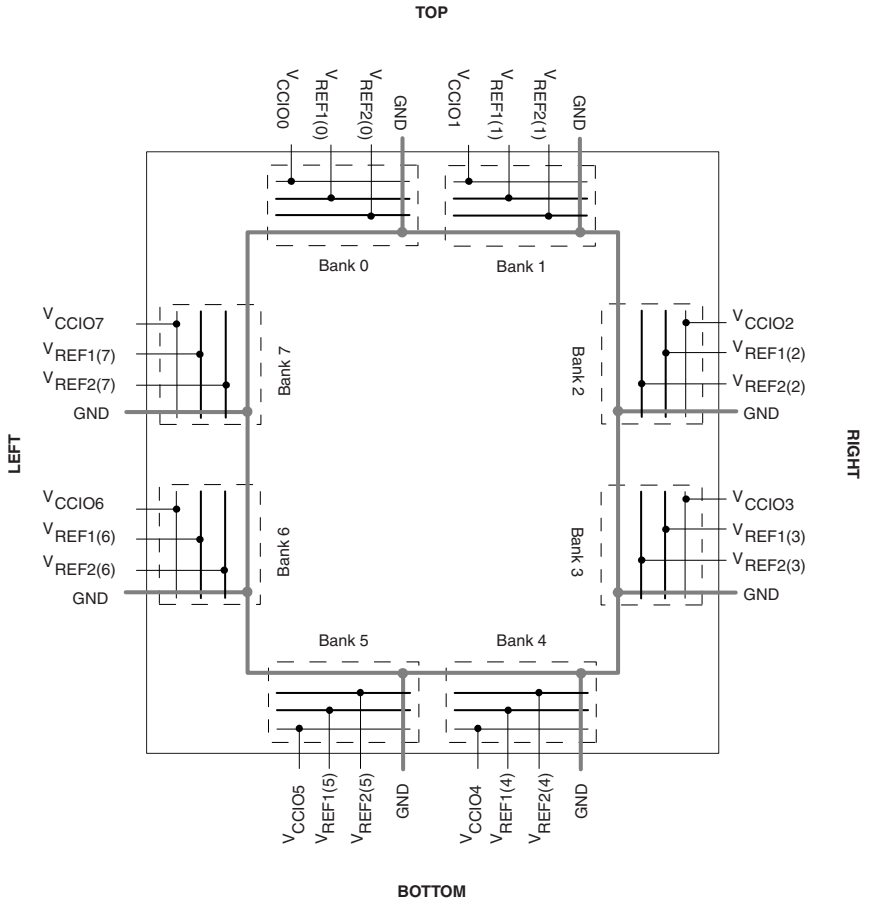
sysIO Buffer Banks

LatticeXP2 devices have eight sysIO buffer banks for user I/Os arranged two per side. Each bank is capable of supporting multiple I/O standards. Each sysIO bank has its own I/O supply voltage (V_{CCIO}). In addition, each bank has voltage references, V_{REF1} and V_{REF2}, that allow it to be completely independent from the others. Figure 2-32 shows the eight banks and their associated supplies.

In LatticeXP2 devices, single-ended output buffers and ratioed input buffers (LVTTL, LVCMOS and PCI) are powered using V_{CCIO}. LVTTL, LVCMOS33, LVCMOS25 and LVCMOS12 can also be set as fixed threshold inputs independent of V_{CCIO}.

Each bank can support up to two separate V_{REF} voltages, V_{REF1} and V_{REF2}, that set the threshold for the referenced input buffers. Some dedicated I/O pins in a bank can be configured to be a reference voltage supply pin. Each I/O is individually configurable based on the bank's supply and reference voltages.

Figure 2-32. LatticeXP2 Banks



LatticeXP2 devices contain two types of sysIO buffer pairs.

1. **Top and Bottom (Banks 0, 1, 4 and 5) sysIO Buffer Pairs (Single-Ended Outputs Only)**
   The sysIO buffer pairs in the top banks of the device consist of two single-ended output drivers and two sets of single-ended input buffers (both ratioed and referenced). One of the referenced input buffers can also be configured as a differential input.

   The two pads in the pair are described as “true” and “comp”, where the true pad is associated with the positive side of the differential input buffer and the comp (complementary) pad is associated with the negative side of the differential input buffer.

   Only the I/Os on the top and bottom banks have programmable PCI clamps.

2. **Left and Right (Banks 2, 3, 6 and 7) sysIO Buffer Pairs (50% Differential and 100% Single-Ended Outputs)**
   The sysIO buffer pairs in the left and right banks of the device consist of two single-ended output drivers, two sets of single-ended input buffers (both ratioed and referenced) and one differential output driver. One of the referenced input buffers can also be configured as a differential input.

   The two pads in the pair are described as “true” and “comp”, where the true pad is associated with the positive side of the differential I/O, and the comp pad is associated with the negative side of the differential I/O.

   LVDS differential output drivers are available on 50% of the buffer pairs on the left and right banks.

Typical sysIO I/O Behavior During Power-up

The internal power-on-reset (POR) signal is deactivated when V_{CC} and V_{CCAUX} have reached satisfactory levels. After the POR signal is deactivated, the FPGA core logic becomes active. It is the user’s responsibility to ensure that all other V_{CCIO} banks are active with valid input logic levels to properly control the output logic states of all the I/O banks that are critical to the application. For more information on controlling the output logic state with valid input logic levels during power-up in LatticeXP2 devices, please see TN1136, *LatticeXP2 sysIO Usage Guide.*

The V_{CC} and V_{CCAUX} supply the power to the FPGA core fabric, whereas the V_{CCIO} supplies power to the I/O buffers. In order to simplify system design while providing consistent and predictable I/O behavior, it is recommended that the I/O buffers be powered-up prior to the FPGA core fabric. V_{CCIO} supplies should be powered-up before or together with the V_{CC} and V_{CCAUX} supplies.

Supported sysIO Standards

The LatticeXP2 sysIO buffer supports both single-ended and differential standards. Single-ended standards can be further subdivided into LVCMOS, LVTTL and other standards. The buffers support the LVTTL, LVCMOS 1.2V, 1.5V, 1.8V, 2.5V and 3.3V standards. In the LVCMOS and LVTTL modes, the buffer has individual configuration options for drive strength, bus maintenance (weak pull-up, weak pull-down, or a bus-keeper latch) and open drain. Other single-ended standards supported include SSTL and HSTL. Differential standards supported include LVDS, MLVDS, BLVDS, LVPECL, RSDS, differential SSTL and differential HSTL. Tables 2-13 and 2-14 show the I/O standards (together with their supply and reference voltages) supported by LatticeXP2 devices. For further information on utilizing the sysIO buffer to support a variety of standards please see TN1136, *LatticeXP2 sysIO Usage Guide.*

Table 2-12. Supported Input Standards

Input Standard	V_{REF} (Nom.)	V_{CCIO}[1] (Nom.)
Single Ended Interfaces		
LVTTL	—	—
LVCMOS33	—	—
LVCMOS25	—	—
LVCMOS18	—	1.8
LVCMOS15	—	1.5
LVCMOS12	—	—
PCI33	—	—
HSTL18 Class I, II	0.9	—
HSTL15 Class I	0.75	—
SSTL33 Class I, II	1.5	—
SSTL25 Class I, II	1.25	—
SSTL18 Class I, II	0.9	—
Differential Interfaces		
Differential SSTL18 Class I, II	—	—
Differential SSTL25 Class I, II	—	—
Differential SSTL33 Class I, II	—	—
Differential HSTL15 Class I	—	—
Differential HSTL18 Class I, II	—	—
LVDS, MLVDS, LVPECL, BLVDS, RSDS	—	—

1. When not specified, V_{CCIO} can be set anywhere in the valid operating range (page 3-1).

Table 2-13. Supported Output Standards

Output Standard	Drive	V_{CCIO} (Nom.)
Single-ended Interfaces		
LVTTL	4mA, 8mA, 12mA, 16mA, 20mA	3.3
LVCMOS33	4mA, 8mA, 12mA 16mA, 20mA	3.3
LVCMOS25	4mA, 8mA, 12mA, 16mA, 20mA	2.5
LVCMOS18	4mA, 8mA, 12mA, 16mA	1.8
LVCMOS15	4mA, 8mA	1.5
LVCMOS12	2mA, 6mA	1.2
LVCMOS33, Open Drain	4mA, 8mA, 12mA 16mA, 20mA	—
LVCMOS25, Open Drain	4mA, 8mA, 12mA 16mA, 20mA	—
LVCMOS18, Open Drain	4mA, 8mA, 12mA 16mA	—
LVCMOS15, Open Drain	4mA, 8mA	—
LVCMOS12, Open Drain	2mA, 6mA	—
PCI33	N/A	3.3
HSTL18 Class I, II	N/A	1.8
HSTL15 Class I	N/A	1.5
SSTL33 Class I, II	N/A	3.3
SSTL25 Class I, II	N/A	2.5
SSTL18 Class I, II	N/A	1.8
Differential Interfaces		
Differential SSTL33, Class I, II	N/A	3.3
Differential SSTL25, Class I, II	N/A	2.5
Differential SSTL18, Class I, II	N/A	1.8
Differential HSTL18, Class I, II	N/A	1.8
Differential HSTL15, Class I	N/A	1.5
LVDS[1, 2]	N/A	2.5
MLVDS[1]	N/A	2.5
BLVDS[1]	N/A	2.5
LVPECL[1]	N/A	3.3
RSDS[1]	N/A	2.5

1. Emulated with external resistors. For more detail, please see TN1138, *LatticeXP2 High Speed I/O Interface.*
2. On the left and right edges, LVDS outputs are supported with a dedicated differential output driver on 50% of the I/Os. This solution does not require external resistors at the driver.

Hot Socketing

LatticeXP2 devices have been carefully designed to ensure predictable behavior during power-up and power-down. Power supplies can be sequenced in any order. During power-up and power-down sequences, the I/Os remain in tri-state until the power supply voltage is high enough to ensure reliable operation. In addition, leakage into I/O pins is controlled to within specified limits. This allows for easy integration with the rest of the system. These capabilities make the LatticeXP2 ideal for many multiple power supply and hot-swap applications.

IEEE 1149.1-Compliant Boundary Scan Testability

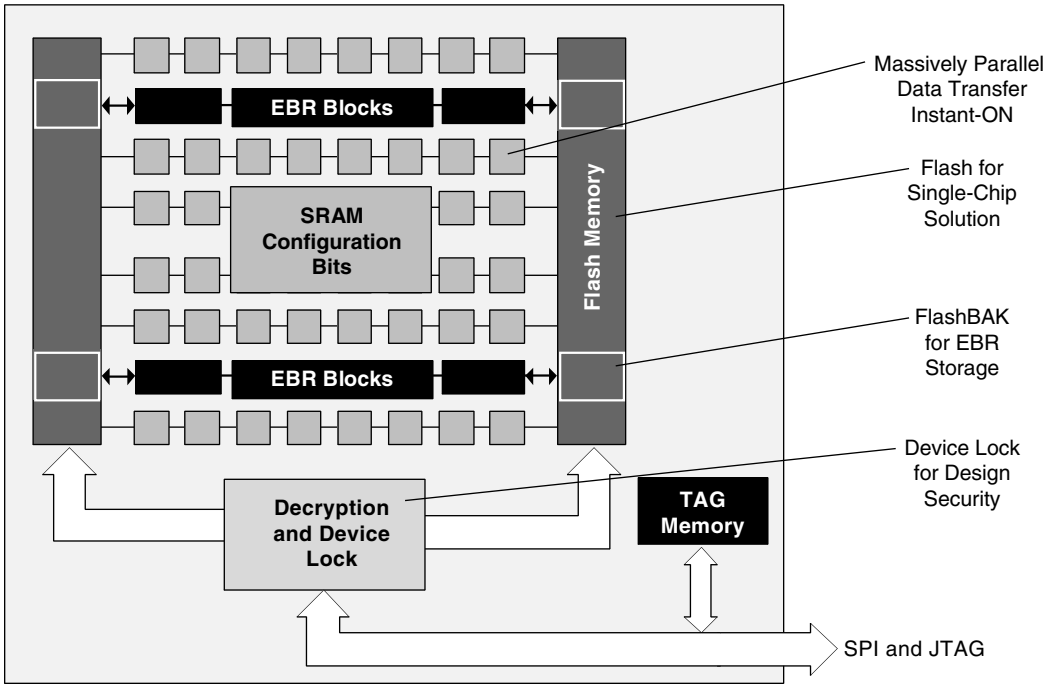
All LatticeXP2 devices have boundary scan cells that are accessed through an IEEE 1149.1 compliant Test Access Port (TAP). This allows functional testing of the circuit board, on which the device is mounted, through a serial scan path that can access all critical logic nodes. Internal registers are linked internally, allowing test data to be shifted in and loaded directly onto test nodes, or test data to be captured and shifted out for verification. The test access port

consists of dedicated I/Os: TDI, TDO, TCK and TMS. The test access port has its own supply voltage V_{CCJ} and can operate with LVCMOS3.3, 2.5, 1.8, 1.5 and 1.2 standards. For more information, please see TN1141, *LatticeXP2 sysCONFIG Usage Guide.*

flexiFLASH Device Configuration

The LatticeXP2 devices combine Flash and SRAM on a single chip to provide users with flexibility in device programming and configuration. Figure 2-33 provides an overview of the arrangement of Flash and SRAM configuration cells within the device. The remainder of this section provides an overview of these capabilities. See TN1141, *LatticeXP2 sysCONFIG Usage Guide,* for a more detailed description.

Figure 2-33. Overview of Flash and SRAM Configuration Cells Within LatticeXP2 Devices



At power-up, or on user command, data is transferred from the on-chip Flash memory to the SRAM configuration cells that control the operation of the device. This is done with massively parallel buses enabling the parts to operate within microseconds of the power supplies reaching valid levels; this capability is referred to as Instant-On.

The on-chip Flash enables a single-chip solution eliminating the need for external boot memory. This Flash can be programmed through either the JTAG or SPI ports of the device. The SRAM configuration space can also be infinitely reconfigured through the JTAG and SPI ports. The JTAG port is IEEE 1149.1 and IEEE 1532 compliant.

As described in the EBR section of the data sheet, the FlashBAK capability of the parts enables the contents of the EBR blocks to be written back into the Flash storage area without erasing or reprogramming other aspects of the device configuration. Serial TAG memory is also available to allow the storage of small amounts of data such as calibration coefficients and error codes.

For applications where security is important, the lack of an external bitstream provides a solution that is inherently more secure than SRAM only FPGAs. This is further enhanced by device locking. The device can be in one of three modes:

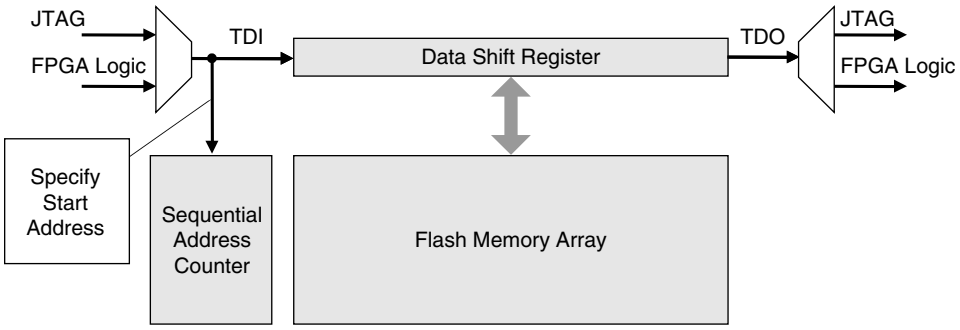
1. Unlocked
2. Key Locked – Presenting the key through the programming interface allows the device to be unlocked.
3. Permanently Locked – The device is permanently locked.

To further complement the security of the device a One Time Programmable (OTP) mode is available. Once the device is set in this mode it is not possible to erase or re-program the Flash portion of the device.

Serial TAG Memory

LatticeXP2 devices offer 0.6 to 3.3kbits of Flash memory in the form of Serial TAG memory. The TAG memory is an area of the on-chip Flash that can be used for non-volatile storage including electronic ID codes, version codes, date stamps, asset IDs and calibration settings. A block diagram of the TAG memory is shown in Figure 2-34. The TAG memory is accessed in the same way as external SPI Flash and it can be read or programmed either through JTAG or directly from FPGA logic. To read the TAG memory, a start address is specified and the entire TAG memory contents are read sequentially in a first-in-first-out manner. The TAG memory is independent of the Flash used for device configuration and given its use for general-purpose storage functions is always accessible regardless of the device security settings. For more information, see TN1137, *LatticeXP2 Memory Usage Guide*, and TN1141, *LatticeXP2 sysCONFIG Usage Guide*.

Figure 2-34. Serial TAG Memory Diagram



Live Update Technology

Many applications require field updates of the FPGA. LatticeXP2 devices provide three features that enable this configuration to be done in a secure and failsafe manner while minimizing impact on system operation.

1. **Decryption Support**
   LatticeXP2 devices provide on-chip, non-volatile key storage to support decryption of a 128-bit AES encrypted bitstream, securing designs and deterring design piracy.

2. **TransFR (Transparent Field Reconfiguration)**
   TransFR I/O (TFR) is a unique Lattice technology that allows users to update their logic in the field without interrupting system operation using a single ispVM command. TransFR I/O allows I/O states to be frozen during device configuration. This allows the device to be field updated with a minimum of system disruption and downtime. For more information please see TN1143, *LatticeXP2 TransFR I/O.*

3. **Dual Boot Image Support**
   Dual boot images are supported for applications requiring reliable remote updates of configuration data for the system FPGA. After the system is running with a basic configuration, a new boot image can be downloaded remotely and stored in a separate location in the configuration storage device. Any time after the update the LatticeXP2 can be re-booted from this new configuration file. If there is a problem such as corrupt data during download or incorrect version number with this new boot image, the LatticeXP2 device can revert back to the original backup configuration and try again. This all can be done without power cycling the system. For more information please see TN1144, *LatticeXP2 Dual Boot Usage Guide.*

For more information on device configuration, please see TN1141, *LatticeXP2 sysCONFIG Usage Guide*.

Soft Error Detect (SED) Support

LatticeXP2 devices have dedicated logic to perform Cyclic Redundancy Code (CRC) checks. During configuration, the configuration data bitstream can be checked with the CRC logic block. In addition, LatticeXP2 devices can be programmed for checking soft errors in SRAM. The SED operation can run in the background during user mode (normal operation). In the event a soft error occurs, the device can be programmed to either reload from a known good boot image (from internal Flash or external SPI memory) or generate an error signal.

For further information on SED support, please see TN1130, *LatticeXP2 Soft Error Detection (SED) Usage Guide*.

On-Chip Oscillator

Every LatticeXP2 device has an internal CMOS oscillator that is used to derive a Master Clock (CCLK) for configuration. The oscillator and CCLK run continuously and are available to user logic after configuration is complete. The available CCLK frequencies are listed in Table 2-14. When a different CCLK frequency is selected during the design process, the following sequence takes place:

1. Device powers up with the default CCLK frequency.
2. During configuration, users select a different CCLK frequency.
3. CCLK frequency changes to the selected frequency after clock configuration bits are received.

This internal CMOS oscillator is available to the user by routing it as an input clock to the clock tree. For further information on the use of this oscillator for configuration or user mode, please see TN1141, *LatticeXP2 sysCONFIG Usage Guide.*

Table 2-14. Selectable CCLKs and Oscillator Frequencies During Configuration and User Mode

CCLK/Oscillator (MHz)
2.5[1]
3.1[2]
4.3
5.4
6.9
8.1
9.2
10
13
15
20
26
32
40
54
80[3]
163[3]

1. Software default oscillator frequency.
2. Software default CCLK frequency.
3. Frequency not valid for CCLK.

Density Shifting

The LatticeXP2 family is designed to ensure that different density devices in the same family and in the same package have the same pinout. Furthermore, the architecture ensures a high success rate when performing design migration from lower density devices to higher density devices. In many cases, it is also possible to shift a lower utilization design targeted for a high-density device to a lower density device. However, the exact details of the final resource utilization will impact the likely success in each case.

LatticeXP2 Family Data Sheet DC and Switching Characteristics

September 2007 **Advance Data Sheet DS1009**

Absolute Maximum Ratings[1, 2, 3]

Supply Voltage V_{CC} . . . . . -0.5 to 1.32V

Supply Voltage V_{CCAUX} . . . . . -0.5 to 3.75V

Supply Voltage V_{CCJ} . . . . . -0.5 to 3.75V

Supply Voltage V_{CCPLL}[4] . . . . . -0.5 to 3.75V

Output Supply Voltage V_{CCIO} . . . . . -0.5 to 3.75V

Input or I/O Tristate Voltage Applied[5] . . . . . -0.5 to 3.75V

Storage Temperature (Ambient) . . . . . -65 to 150°C

Junction Temperature Under Bias (Tj) . . . . . +125°C

1. Stress above those listed under the "Absolute Maximum Ratings" may cause permanent damage to the device. Functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied.
2. Compliance with the Lattice *Thermal Management* document is required.
3. All voltages referenced to GND.
4. V_{CCPLL} only available on PQFP and TQFP packages.
5. Overshoot and undershoot of -2V to (V_{IHMAX} + 2) volts is permitted for a duration of <20ns.

Recommended Operating Conditions

Symbol	Parameter	Min.	Max.	Units
V_{CC}	Core Supply Voltage	1.14	1.26	V
V_{CCAUX}	Auxiliary Supply Voltage	3.135	3.465	V
V_{CCPLL}[1]	PLL Supply Voltage	3.135	3.465	V
V_{CCIO}[2, 3]	I/O Driver Supply Voltage	1.14	3.465	V
V_{CCJ}[2]	Supply Voltage for IEEE 1149.1 Test Access Port	1.14	3.465	V
t_{JCOM}	Junction Temperature, Commercial Operation	0	85	°C
t_{JIND}	Junction Temperature, Industrial Operation	-40	100	°C

1. V_{CCPLL} only available on PQFP and TQFP packages.
2. If V_{CCIO} or V_{CCJ} is set to 1.2V, they must be connected to the same power supply as V_{CC}. If V_{CCIO} or V_{CCJ} is set to 3.3V, they must be connected to the same power supply as V_{CCAUX}.
3. See recommended voltages by I/O standard in subsequent table.

On-Chip Flash Memory Specifications

Symbol	Parameter	Max.	Units
$N_{PROGCYC}$	Flash Programming Cycles		Cycles
$t_{RETENTION}$	Data Retention	20	Years

Hot Socketing Specifications[1, 2, 3, 4, 5]

Symbol	Parameter	Condition	Min.	Typ.	Max.	Units
I_{DK}	Input or I/O Leakage Current	$0 \le V_{IN} \le V_{IH}$ (MAX.)	—	—	+/-1	mA

1. Insensitive to sequence of V_{CC}, V_{CCAUX} and V_{CCIO}. However, assumes monotonic rise/fall rates for V_{CC}, V_{CCAUX} and V_{CCIO}.
2. $0 \le V_{CC} \le V_{CC}$ (MAX), $0 \le V_{CCIO} \le V_{CCIO}$ (MAX) or $0 \le V_{CCAUX} \le V_{CCAUX}$ (MAX).
3. I_{DK} is additive to I_{PU}, I_{PW} or I_{BH}.
4. LVCMOS and LVTTL only.
5. Note this table represents DC conditions. For the first 20ns after hot insertion, current specification is 8mA.

DC Electrical Characteristics

Over Recommended Operating Conditions

Symbol	Parameter	Condition	Min.	Typ.	Max.	Units
I_{IL}, I_{IH}[1]	Input or I/O Low Leakage	$0 \le V_{IN} \le V_{CCIO}$	—	—	10	µA
		$V_{CCIO} \le V_{IN} \le V_{IH}$ (MAX)	—	—	100	µA
I_{PU}	I/O Active Pull-up Current	$0 \le V_{IN} \le 0.7\ V_{CCIO}$	-30	—	-150	µA
I_{PD}	I/O Active Pull-down Current	V_{IL} (MAX) $\le V_{IN} \le V_{CCIO}$	30	—	210	µA
I_{BHLS}	Bus Hold Low Sustaining Current	$V_{IN} = V_{IL}$ (MAX)	30	—	—	µA
I_{BHHS}	Bus Hold High Sustaining Current	$V_{IN} = 0.7\ V_{CCIO}$	-30	—	—	µA
I_{BHLO}	Bus Hold Low Overdrive Current	$0 \le V_{IN} \le V_{CCIO}$	—	—	210	µA
I_{BHHO}	Bus Hold High Overdrive Current	$0 \le V_{IN} \le V_{CCIO}$	—	—	-150	µA
V_{BHT}	Bus Hold Trip Points		V_{IL} (MAX)	—	V_{IH} (MIN)	V
C1	I/O Capacitance[2]	V_{CCIO} = 3.3V, 2.5V, 1.8V, 1.5V, 1.2V, V_{CC} = 1.2V, V_{IO} = 0 to V_{IH} (MAX)	—	8	—	pf
C2	Dedicated Input Capacitance[2]	V_{CCIO} = 3.3V, 2.5V, 1.8V, 1.5V, 1.2V, V_{CC} = 1.2V, V_{IO} = 0 to V_{IH} (MAX)	—	6	—	pf

1. Input or I/O leakage current is measured with the pin configured as an input or as an I/O with the output driver tri-stated. It is not measured with the output driver active. Bus maintenance circuits are disabled.
2. T_A 25°C, f = 1.0MHz.

Supply Current (Standby)[1, 2, 3, 4]

Over Recommended Operating Conditions

Symbol	Parameter	Device	Typical[5]	Max.[6]	Units
I_{CC}	Core Power Supply Current	XP2-5			mA
		XP2-8			mA
		XP2-17	37		mA
		XP2-30			mA
		XP2-40			mA
I_{CCAUX}	Auxiliary Power Supply Current[7]	XP2-5			mA
		XP2-8			mA
		XP2-17			mA
		XP2-30			mA
		XP2-40			mA
I_{CCPLL}	PLL Power Supply Current (per PLL)				mA
I_{CCIO}	Bank Power Supply Current (per bank)		1.3		mA
I_{CCJ}	V_{CCJ} Power Supply Current		0.1		mA

1. For further information on supply current, please see TN1139, *Power Estimation and Management for LatticeXP2 Devices.*
2. Assumes all outputs are tristated, all inputs are configured as LVCMOS and held at the V_{CCIO} or GND.
3. Frequency 0MHz.
4. Pattern represents a "blank" configuration data file.
5. T_J = 25°C, power supplies at nominal voltage.
6. Commercial conditions, T_J = 85°C, power supplies at their respective maximum values. For maximum current in Industrial conditions, contact Lattice Semiconductor.
7. In fpBGA and ftBGA packages the PLLs are connected to and powered from the auxiliary power supply. For these packages, the actual auxiliary supply current is the sum of I_{CCAUX} and I_{CCPLL}. For csBGA, PQFP and TQFP packages the PLLs are powered independent of the auxiliary power supply.

Initialization Supply Current[1, 2, 3, 4, 5]

Over Recommended Operating Conditions

Symbol	Parameter	Device	Typical[6]	Max.[7]	Units
I_{CC}	Core Power Supply Current	XP2-5			mA
		XP2-8			mA
		XP2-17	69		mA
		XP2-30			mA
		XP2-40			mA
I_{CCAUX}	Auxiliary Power Supply Current[8]	XP2-5			mA
		XP2-8			mA
		XP2-17			mA
		XP2-30			mA
		XP2-40			mA
I_{CCPLL}	PLL Power Supply Current (per PLL)				mA
I_{CCIO}	Bank Power Supply Current (per Bank)		6.4		mA
I_{CCJ}	VCCJ Power Supply Current				mA

1. For further information on supply current, please see TN1139, *Power Estimation and Management for LatticeXP2 Devices.*
2. Assumes all outputs are tristated, all inputs are configured as LVCMOS and held at the V_{CCIO} or GND.
3. Frequency 0MHz.
4. Bypass or decoupling capacitor across the supply.
5. A specific configuration pattern is used that scales with the size of the device; consists of 75% PFU utilization, 50% EBR, and 25% I/O configuration.
6. T_J = 25°C, power supplies at nominal voltage.
7. Commercial conditions, T_J = 85°C, power supplies at their respective maximum values. For maximum current in Industrial conditions, contact Lattice Semiconductor.
8. In fpBGA and ftBGA packages the PLLs are connected to and powered from the auxiliary power supply. For these packages, the actual auxiliary supply current is the sum of I_{CCAUX} and $I_{CCPLL.}$ For csBGA, PQFP and TQFP packages the PLLs are powered independent of the auxiliary power supply.

Programming and Erase Flash Supply Current[1, 2, 3, 4, 5]

Over Recommended Operating Conditions

Symbol	Parameter	Device	Typical[6]	Max.[7]	Units
I_{CC}	Core Power Supply Current	XP2-5			mA
		XP2-8			mA
		XP2-17	46		mA
		XP2-30			mA
		XP2-40			mA
I_{CCAUX}	Auxiliary Power Supply Current[8]	XP2-5			mA
		XP2-8			mA
		XP2-17			mA
		XP2-30			mA
		XP2-40			mA
I_{CCPLL}	PLL Power Supply Current (per PLL)				mA
I_{CCIO}	Bank Power Supply Current (per Bank)		5		mA
I_{CCJ}	V_{CCJ} Power Supply Current[9]				mA

1. For further information on supply current, please see TN1139, *Power Estimation and Management for LatticeXP2 Devices.*
2. Assumes all outputs are tristated, all inputs are configured as LVCMOS and held at the V_{CCIO} or GND.
3. Frequency 0MHz.
4. Pattern represents a “blank” configuration data file.
5. Bypass or decoupling capacitor across the supply.
6. T_J = 25°C, power supplies at nominal voltage.
7. Commercial conditions, T_J = 85°C, power supplies at their respective maximum values. For maximum current in Industrial conditions, contact Lattice Semiconductor.
8. In fpBGA and ftBGA packages the PLLs are connected to and powered from the auxiliary power supply. For these packages, the actual auxiliary supply current is the sum of I_{CCAUX} and I_{CCPLL}. For csBGA, PQFP and TQFP packages the PLLs are powered independent of the auxiliary power supply.
9. When programming via JTAG.

sysIO Recommended Operating Conditions

Over Recommended Operating Conditions

Standard	V_{CCIO}			V_{REF} (V)		
	Min.	Typ.	Max.	Min.	Typ.	Max.
LVCMOS33[2]	3.135	3.3	3.465	—	—	—
LVCMOS25[2]	2.375	2.5	2.625	—	—	—
LVCMOS18	1.71	1.8	1.89	—	—	—
LVCMOS15	1.425	1.5	1.575	—	—	—
LVCMOS12[2]	1.14	1.2	1.26	—	—	—
LVTTL33[2]	3.135	3.3	3.465	—	—	—
PCI33	3.135	3.3	3.465	—	—	—
SSTL18_I[2], SSTL18_II[2]	1.71	1.8	1.89	0.833	0.9	0.969
SSTL25_I[2], SSTL25_II[2]	2.375	2.5	2.625	1.15	1.25	1.35
SSTL33_I[2], SSTL33_II[2]	3.135	3.3	3.465	1.3	1.5	1.7
HSTL15_I[2]	1.425	1.5	1.575	0.68	0.75	0.9
HSTL18_I[2], HSTL18_II[2]	1.71	1.8	1.89	0.816	0.9	1.08
LVDS25[2]	2.375	2.5	2.625	—	—	—
MLVDS25[1]	2.375	2.5	2.625	—	—	—
LVPECL33[1, 2]	3.135	3.3	3.465	—	—	—
BLVDS25[1, 2]	2.375	2.5	2.625	—	—	—
RSDS[1, 2]	2.375	2.5	2.625	—	—	—
SSTL18D_I[2], SSTL18D_II[2]	1.71	1.8	1.89	—	—	—
SSTL25D_ I[2], SSTL25D_II[2]	2.375	2.5	2.625	—	—	—
SSTL33D_ I[2], SSTL33D_ II[2]	3.135	3.3	3.465	—	—	—
HSTL15D_ I[2]	1.425	1.5	1.575	—	—	—
HSTL18D_ I[2], HSTL18D_ II[2]	1.71	1.8	1.89	—	—	—

1. Inputs on chip. Outputs are implemented with the addition of external resistors.
2. Input on this standard does not depend on the value of V_{CCIO}.

sysIO Single-Ended DC Electrical Characteristics

Over Recommended Operating Conditions

Input/Output Standard	V_{IL}		V_{IH}		V_{OL}	V_{OH}		
	Min. (V)	Max. (V)	Min. (V)	Max. (V)	Max. (V)	Min. (V)	I_{OL}[1] (mA)	I_{OH}[1] (mA)
LVCMOS33	-0.3	0.8	2.0	3.6	0.4	V_{CCIO} - 0.4	20, 16, 12, 8, 4	-20, -16, -12, -8, -4
					0.2	V_{CCIO} - 0.2	0.1	-0.1
LVTTL33	-0.3	0.8	2.0	3.6	0.4	V_{CCIO} - 0.4	20, 16, 12, 8, 4	-20, -16, -12, -8, -4
					0.2	V_{CCIO} - 0.2	0.1	-0.1
LVCMOS25	-0.3	0.7	1.7	3.6	0.4	V_{CCIO} - 0.4	20, 16, 12, 8, 4	-20, -16, -12, -8, -4
					0.2	V_{CCIO} - 0.2	0.1	-0.1
LVCMOS18	-0.3	0.35 V_{CCIO}	0.65 V_{CCIO}	3.6	0.4	V_{CCIO} - 0.4	16, 12, 8, 4	-16, -12, -8, -4
					0.2	V_{CCIO} - 0.2	0.1	-0.1
LVCMOS15	-0.3	0.35 V_{CCIO}	0.65 V_{CCIO}	3.6	0.4	V_{CCIO} - 0.4	8, 4	-8, -4
					0.2	V_{CCIO} - 0.2	0.1	-0.1
LVCMOS12	-0.3	0.35 V_{CC}	0.65 V_{CC}	3.6	0.4	V_{CCIO} - 0.4	6, 2	-6, -2
					0.2	V_{CCIO} - 0.2	0.1	-0.1
PCI33	-0.3	0.3 V_{CCIO}	0.5 V_{CCIO}	3.6	0.1 V_{CCIO}	0.9 V_{CCIO}	1.5	-0.5
SSTL33_I	-0.3	V_{REF} - 0.2	V_{REF} + 0.2	3.6	0.7	V_{CCIO} - 1.1	8	-8
SSTL33_II	-0.3	V_{REF} - 0.2	V_{REF} + 0.2	3.6	0.5	V_{CCIO} - 0.9	16	-16
SSTL25_I	-0.3	V_{REF} - 0.18	V_{REF} + 0.18	3.6	0.54	V_{CCIO} - 0.62	7.6	-7.6
							12	-12
SSTL25_II	-0.3	V_{REF} - 0.18	V_{REF} + 0.18	3.6	0.35	V_{CCIO} - 0.43	15.2	-15.2
							20	-20
SSTL18_I	-0.3	V_{REF} - 0.125	V_{REF} + 0.125	3.6	0.4	V_{CCIO} - 0.4	6.7	-6.7
SSTL18_II	-0.3	V_{REF} - 0.125	V_{REF} + 0.125	3.6	0.28	V_{CCIO} - 0.28	8	-8
							11	-11
HSTL15_I	-0.3	V_{REF} - 0.1	V_{REF} + 0.1	3.6	0.4	V_{CCIO} - 0.4	4	-4
							8	-8
HSTL18_I	-0.3	V_{REF} - 0.1	V_{REF} + 0.1	3.6	0.4	V_{CCIO} - 0.4	8	-8
							12	-12
HSTL18_II	-0.3	V_{REF} - 0.1	V_{REF} + 0.1	3.6	0.4	V_{CCIO} - 0.4	16	-16

1. The average DC current drawn by I/Os between GND connections, or between the last GND in an I/O bank and the end of an I/O bank, as shown in the logic signal connections table shall not exceed n * 8mA, where n is the number of I/Os between bank GND connections or between the last GND in a bank and the end of a bank.

sysIO Differential Electrical Characteristics

LVDS

Over Recommended Operating Conditions

Parameter	Description	Test Conditions	Min.	Typ.	Max.	Units
V_{INP}, V_{INM}	Input Voltage		0	—	2.4	V
V_{CM}	Input Common Mode Voltage	Half the Sum of the Two Inputs	0.05	—	2.35	V
V_{THD}	Differential Input Threshold	Difference Between the Two Inputs	+/-100	—	—	mV
I_{IN}	Input Current	Power On or Power Off	—	—	+/-10	µA
V_{OH}	Output High Voltage for V_{OP} or V_{OM}	R_T = 100 Ohm	—	1.38	1.60	V
V_{OL}	Output Low Voltage for V_{OP} or V_{OM}	R_T = 100 Ohm	0.9V	1.03	—	V
V_{OD}	Output Voltage Differential	(V_{OP} - V_{OM}), R_T = 100 Ohm	250	350	450	mV
ΔV_{OD}	Change in V_{OD} Between High and Low		—	—	50	mV
V_{OS}	Output Voltage Offset	(V_{OP} + V_{OM})/2, R_T = 100 Ohm	1.125	1.20	1.375	V
ΔV_{OS}	Change in V_{OS} Between H and L		—	—	50	mV
I_{SA},I_{SA}	Output Short Circuit Current	V_{OD} = 0V Driver Outputs Shorted to Ground	—	—	24	mA
I_{SAB}	Output Short Circuit Current	V_{OD} = 0V Driver Outputs Shorted to Each Other	—	—	12	mA

Differential HSTL and SSTL

Differential HSTL and SSTL outputs are implemented as a pair of complementary single-ended outputs. All allowable single-ended output classes (class I and class II) are supported in this mode.

For further information on LVPECL, RSDS, MLVDS, BLVDS and other differential interfaces please see details of additional technical information at the end of this data sheet.

LVDS25E

The top and bottom sides of LatticeXP2 devices support LVDS outputs via emulated complementary LVCMOS outputs in conjunction with a parallel resistor across the driver outputs. The scheme shown in Figure 3-1 is one possible solution for point-to-point signals.

Figure 3-1. LVDS25E Output Termination Example

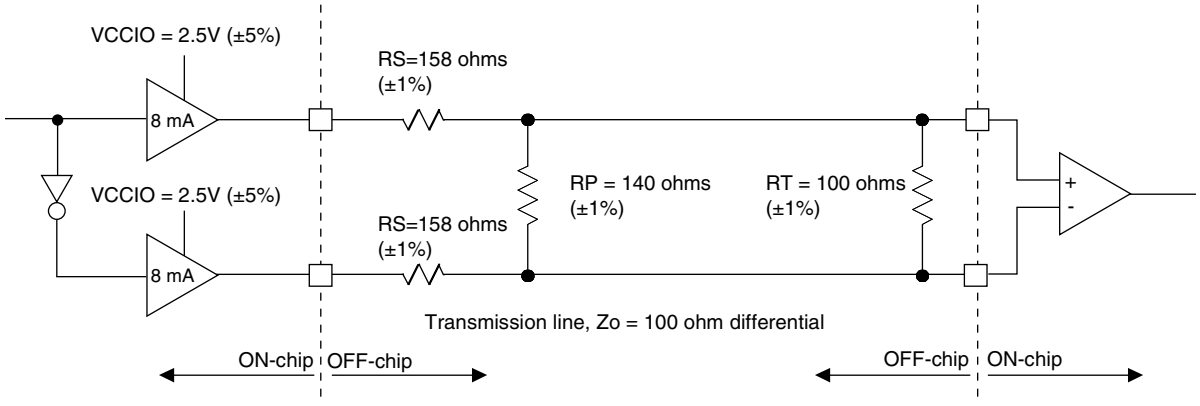


Table 3-1. LVDS25E DC Conditions

Parameter	Description	Typical	Units
V_{CCIO}	Output Driver Supply (+/-5%)	2.50	V
Z_{OUT}	Driver Impedance	20	Ω
R_S	Driver Series Resistor (+/-1%)	158	Ω
R_P	Driver Parallel Resistor (+/-1%)	140	Ω
R_T	Receiver Termination (+/-1%)	100	Ω
V_{OH}	Output High Voltage (after R_1)	1.43	V
V_{OL}	Output Low Voltage (after R_1)	1.07	V
V_{OD}	Output Differential Voltage (After R_1)	0.35	V
V_{CM}	Output Common Mode Voltage	1.25	V
Z_{BACK}	Back Impedance	100.5	Ω
I_{DC}	DC Output Current	6.03	mA

BLVDS

The LatticeXP2 devices support the BLVDS standard. This standard is emulated using complementary LVCMOS outputs in conjunction with a parallel external resistor across the driver outputs. BLVDS is intended for use when multi-drop and bi-directional multi-point differential signaling is required. The scheme shown in Figure 3-2 is one possible solution for bi-directional multi-point differential signals.

Figure 3-2. BLVDS Multi-point Output Example

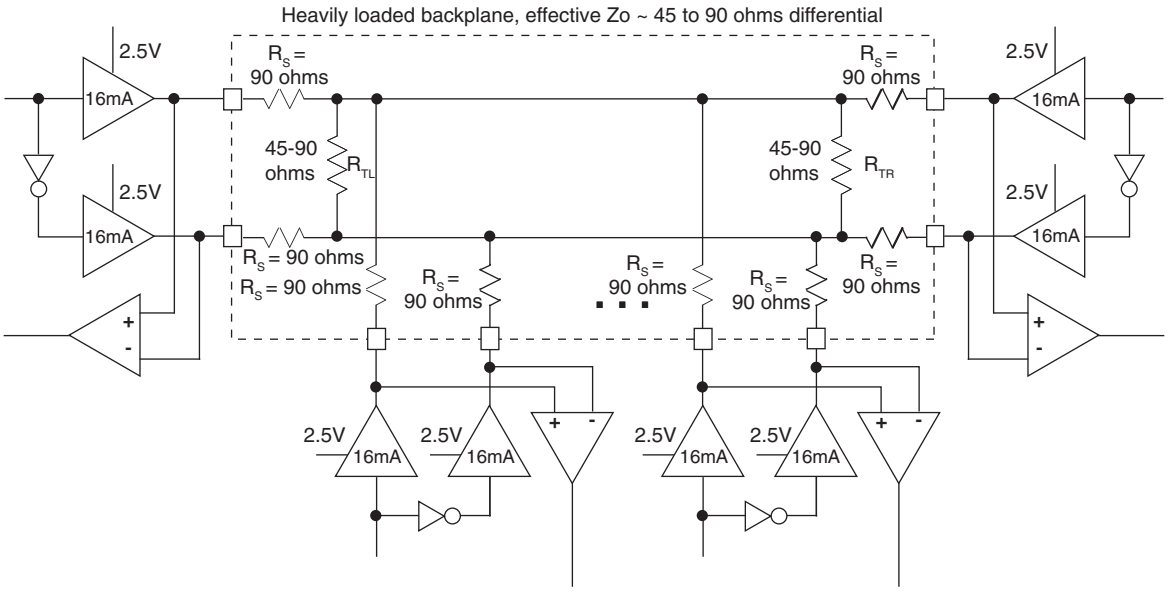


Table 3-2. BLVDS DC Conditions[1]

Over Recommended Operating Conditions

Parameter	Description	Typical		Units
		Zo = 45Ω	Zo = 45Ω	
V_{CCIO}	Output Driver Supply (+/- 5%)	2.50	2.50	V
Z_{OUT}	Driver Impedance	10.00	10.00	Ω
R_S	Driver Series Resistor (+/- 1%)	90.00	90.00	Ω
R_{TLEFT}	Driver Parallel Resistor (+/- 1%)	45.00	90.00	Ω
R_{TRIGHT}	Receiver Termination (+/- 1%)	45.00	90.00	Ω
V_{OH}	Output High Voltage (After R1)	1.38	1.48	V
V_{OL}	Output Low Voltage (After R1)	1.12	1.02	V
V_{OD}	Output Differential Voltage (After R1)	0.25	0.46	V
V_{CM}	Output Common Mode Voltage	1.25	1.25	V
I_{DC}	DC Output Current	11.24	10.20	mA

1. For input buffer, see LVDS table.

LVPECL

The LatticeXP2 devices support the differential LVPECL standard. This standard is emulated using complementary LVCMOS outputs in conjunction with a parallel resistor across the driver outputs. The LVPECL input standard is supported by the LVDS differential input buffer. The scheme shown in Figure 3-3 is one possible solution for point-to-point signals.

Figure 3-3. Differential LVPECL

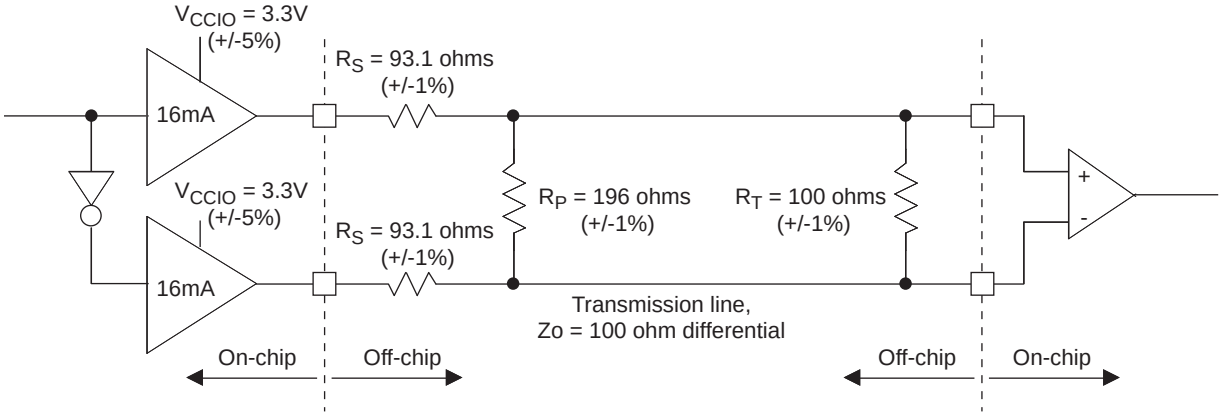


Table 3-3. LVPECL DC Conditions[1]

Over Recommended Operating Conditions

Parameter	Description	Typical	Units
V_{CCIO}	Output Driver Supply (+/-5%)	3.30	V
Z_{OUT}	Driver Impedance	10	Ω
R_S	Driver Series Resistor (+/-1%)	93	Ω
R_P	Driver Parallel Resistor (+/-1%)	196	Ω
R_T	Receiver Termination (+/-1%)	100	Ω
V_{OH}	Output High Voltage (After R_1)	2.05	V
V_{OL}	Output Low Voltage (After R_1)	1.25	V
V_{OD}	Output Differential Voltage (After R_1)	0.80	V
V_{CM}	Output Common Mode Voltage	1.65	V
Z_{BACK}	Back Impedance	100.5	Ω
I_{DC}	DC Output Current	12.11	mA

1. For input buffer, see LVDS table.

RSDS

The LatticeXP2 devices support differential RSDS standard. This standard is emulated using complementary LVCMOS outputs in conjunction with a parallel resistor across the driver outputs. The RSDS input standard is supported by the LVDS differential input buffer. The scheme shown in Figure 3-4 is one possible solution for RSDS standard implementation. Resistor values in Figure 3-4 are industry standard values for 1% resistors.

Figure 3-4. RSDS (Reduced Swing Differential Standard)

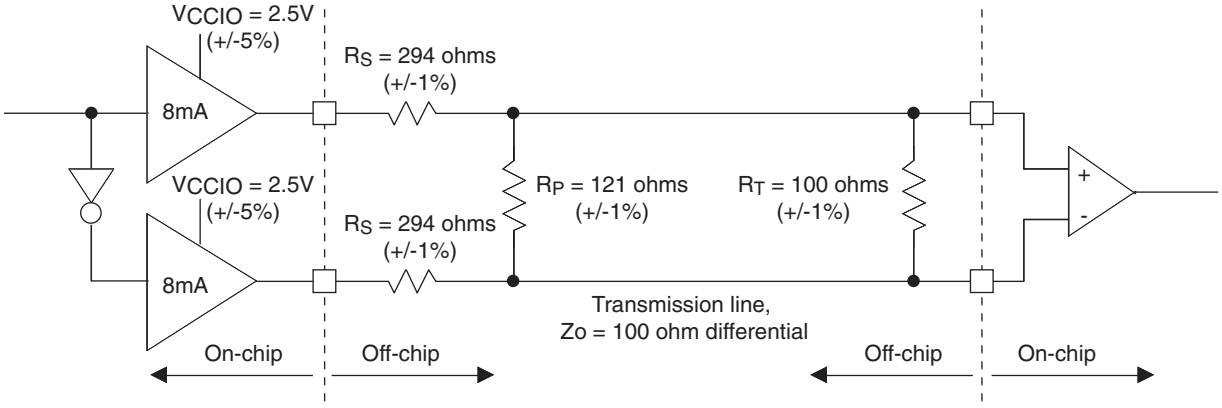


Table 3-4. RSDS DC Conditions[1]

Over Recommended Operating Conditions

Parameter	Description	Typical	Units
V_{CCIO}	Output Driver Supply (+/-5%)	2.50	V
Z_{OUT}	Driver Impedance	20	Ω
R_S	Driver Series Resistor (+/-1%)	294	Ω
R_P	Driver Parallel Resistor (+/-1%)	121	Ω
R_T	Receiver Termination (+/-1%)	100	Ω
V_{OH}	Output High Voltage (After R_1)	1.35	V
V_{OL}	Output Low Voltage (After R_1)	1.15	V
V_{OD}	Output Differential Voltage (After R_1)	0.20	V
V_{CM}	Output Common Mode Voltage	1.25	V
Z_{BACK}	Back Impedance	101.5	Ω
I_{DC}	DC Output Current	3.66	mA

1. For input buffer, see LVDS table.

MLVDS

The LatticeXP2 devices support the differential MLVDS standard. This standard is emulated using complementary LVCMOS outputs in conjunction with a parallel resistor across the driver outputs. The MLVDS input standard is supported by the LVDS differential input buffer. The scheme shown in Figure 3-5 is one possible solution for MLVDS standard implementation. Resistor values in Figure 3-5 are industry standard values for 1% resistors.

Figure 3-5. MLVDS (Reduced Swing Differential Standard)

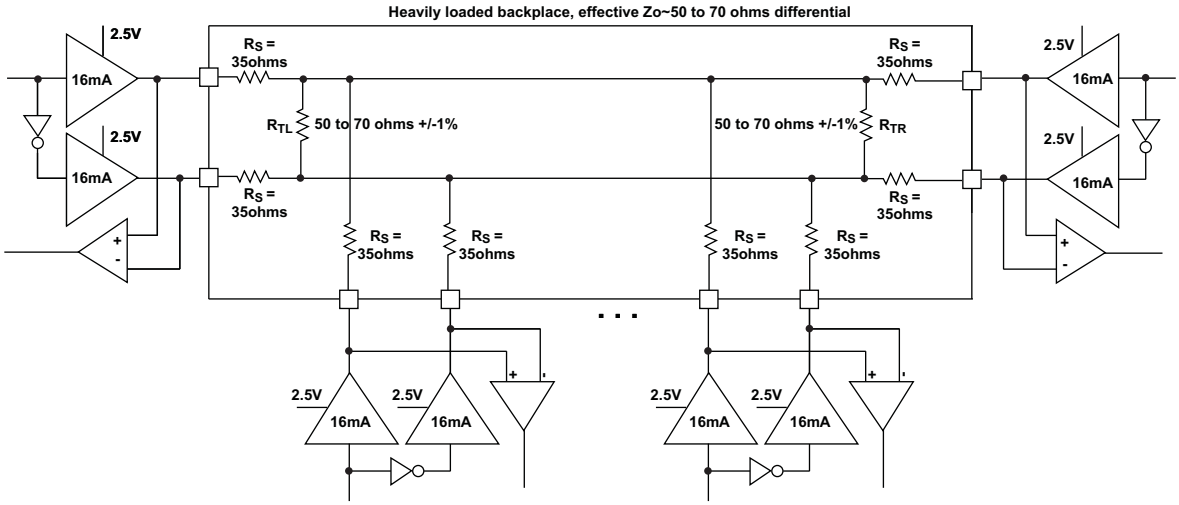


Table 3-5. MLVDS DC Conditions[1]

Parameter	Description	Typical		Units
		Zo=50Ω	Zo=70Ω	
V_{CCIO}	Output Driver Supply (+/-5%)	2.50	2.50	V
Z_{OUT}	Driver Impedance	10.00	10.00	Ω
R_S	Driver Series Resistor (+/-1%)	35.00	35.00	Ω
R_{TLEFT}	Driver Parallel Resistor (+/-1%)	50.00	70.00	Ω
R_{TRIGHT}	Receiver Termination (+/-1%)	50.00	70.00	Ω
V_{OH}	Output High Voltage (After R_1)	1.52	1.60	V
V_{OL}	Output Low Voltage (After R_1)	0.98	0.90	V
V_{OD}	Output Differential Voltage (After R_1)	0.54	0.70	V
V_{CM}	Output Common Mode Voltage	1.25	1.25	V
I_{DC}	DC Output Current	21.74	20.00	mA

1. For input buffer, see LVDS table.

For further information on LVPECL, RSDS, MLVDS, BLVDS and other differential interfaces please see details of additional technical information at the end of this data sheet.

Typical Building Block Function Performance[1]

Pin-to-Pin Performance (LVCMOS25 12mA Drive)

Function	-7 Timing	Units
Basic Functions		
16-bit Decoder	4.4	ns
32-bit Decoder	5.3	ns
64-bit Decoder	5.8	ns
4:1 MUX	3.8	ns
8:1 MUX	4.1	ns
16:1 MUX	4.4	ns
32:1 MUX	4.6	ns

Register-to-Register Performance

Function	-7 Timing	Units
Basic Functions		
16-bit Decoder	437	MHz
32-bit Decoder	418	MHz
64-bit Decoder	345	MHz
4:1 MUX	646	MHz
8:1 MUX	596	MHz
16:1 MUX	550	MHz
32:1 MUX	477	MHz
8-bit Adder	437	MHz
16-bit Adder	402	MHz
64-bit Adder	189	MHz
16-bit Counter	412	MHz
32-bit Counter	336	MHz
64-bit Counter	245	MHz
64-bit Accumulator	205	MHz
Embedded Memory Functions		
512x36 Single Port RAM, EBR Output Registers	315	MHz
1024x18 True-Dual Port RAM (Write Through or Normal, EBR Output Registers)	315	MHz
1024x18 True-Dual Port RAM (Read-Before-Write, EBR Output Registers)	225	MHz
1024x18 True-Dual Port RAM (Write Through or Normal, PLC Output Registers)	220	MHz
Distributed Memory Functions		
16x4 Pseudo-Dual Port RAM (One PFU)	690	MHz
32x2 Pseudo-Dual Port RAM	419	MHz
64x1 Pseudo-Dual Port RAM	309	MHz
DSP Functions		
18x18 Multiplier (All Registers)	365	MHz
9x9 Multiplier (All Registers)	365	MHz
36x36 Multiply (All Registers)	304	MHz
18x18 Multiply/Accumulate (Input and Output Registers)	281	MHz
18x18 Multiply-Add/Sub-Sum (All Registers)	274	MHz

Register-to-Register Performance (Continued)

Function	-7 Timing	Units
DSP IP Functions		
16-Tap Fully-Parallel FIR Filter		MHz
1024-pt FFT	210	MHz
8X8 Matrix Multiplication	200	MHz

1. These timing numbers were generated using the ispLEVER design tool. Exact performance may vary with device, design and tool version. The tool uses internal parameters that have been characterized but are not tested on every device.

Timing v. A 0.07

Derating Timing Tables

Logic timing provided in the following sections of this data sheet and the ispLEVER design tools are worst case numbers in the operating range. Actual delays at nominal temperature and voltage for best case process, can be much better than the values given in the tables. The ispLEVER design tool can provide logic timing numbers at a particular temperature and voltage.

LatticeXP2 External Switching Characteristics

Over Recommended Operating Conditions

Parameter	Description	Device	-7		-6		-5		Units
			Min.	Max.	Min.	Max.	Min.	Max.	
General I/O Pin Parameters (using Primary Clock without PLL)[1]									
t_{CO}	Clock to Output - PIO Output Register	XP2-17	—	3.60	—	4.00	—	4.40	ns
t_{SU}	Clock to Data Setup - PIO Input Register	XP2-17	-0.50	—	-0.70	—	-0.90	—	ns
t_{H}	Clock to Data Hold - PIO Input Register	XP2-17	1.35	—	1.50	—	1.65	—	ns
t_{SU_DEL}	Clock to Data Setup - PIO Input Register with Data Input Delay	XP2-17	1.10	—	1.50	—	1.90	—	ns
t_{H_DEL}	Clock to Data Hold - PIO Input Register with Input Data Delay	XP2-17	0.00	—	0.00	—	0.00	—	ns
f_{MAX_IO}	Clock Frequency of I/O and PFU Register	XP2-17	—	420	—	420	—	420	MHz
General I/O Pin Parameters (using Edge Clock without PLL)[1]									
t_{COE}	Clock to Output - PIO Output Register	XP2-17	—	3.00	—	3.33	—	3.66	ns
t_{SUE}	Clock to Data Setup - PIO Input Register	XP2-17	0.00	—	0.00	—	0.00	—	ns
t_{HE}	Clock to Data Hold - PIO Input Register	XP2-17	0.80	—	0.90	—	1.00	—	ns
t_{SU_DELE}	Clock to Data Setup - PIO Input Register with Data Input Delay	XP2-17	1.00	—	1.10	—	1.20	—	ns
t_{H_DELE}	Clock to Data Hold - PIO Input Register with Input Data Delay	XP2-17	0.00	—	0.00	—	0.00	—	ns
f_{MAX_IOE}	Clock Frequency of I/O and PFU Register	XP2-17	—	420	—	420	—	420	MHz
General I/O Pin Parameters (using Primary Clock with PLL)[1]									
t_{COPLL}	Clock to Output - PIO Output Register	XP2-17	—		—		—		ns
t_{SUPLL}	Clock to Data Setup - PIO Input Register	XP2-17		—		—		—	ns
t_{HPLL}	Clock to Data Hold - PIO Input Register	XP2-17		—		—		—	ns
t_{SU_DELPLL}	Clock to Data Setup - PIO Input Register with Data Input Delay	XP2-17		—		—		—	ns
t_{H_DELPLL}	Clock to Data Hold - PIO Input Register with Input Data Delay	XP2-17		—		—		—	ns
DDR[2] and DDR2[3] I/O Pin Parameters									
t_{DVADQ}	Data Valid After DQS (DDR Read)	XP2-17	—	0.23	—	0.23	—	0.23	UI
t_{DVEDQ}	Data Hold After DQS (DDR Read)	XP2-17	0.64	—	0.64	—	0.64	—	UI
t_{DQVBS}	Data Valid Before DQS	XP2-17	0.25	—	0.25	—	0.25	—	UI
t_{DQVAS}	Data Valid After DQS	XP2-17	0.25	—	0.25	—	0.25	—	UI
f_{MAX_DDR}	DDR Clock Frequency	XP2-17	95	200	95	166	95	166	MHz
f_{MAX_DDR2}	DDR Clock Frequency	XP2-17	133	200	133	200	133	166	MHz

LatticeXP2 External Switching Characteristics (Continued)

Over Recommended Operating Conditions

Parameter	Description	Device	-7		-6		-5		Units
			Min.	Max.	Min.	Max.	Min.	Max.	
Source Synchronous I/O Pin Parameters for 7:1 LVDS Display Interfaces									
$t_{DVACLKDISP}$	Receive Data Valid After RDTCLK *3.5[4]	XP2-17	—		—		—		ps
$t_{DVECLKDISP}$	Receive Data Valid Hold Time After RDTCLK * 3.5[4]	XP2-17		—		—		—	ps
$t_{DIADISP}$	Data Invalid After RDTCLK * 3.5[4]	XP2-17	—		—		—		ps
$t_{DIBDISP}$	Data Invalid Before RDTCLK * 3.5[4]	XP2-17	—		—		—		ps
XGMII I/O Pin Parameters[5]									
$t_{SUXGMII}$	Data Setup Before Read Clock	XP2-17		—		—		—	ps
t_{HXGMII}	Data Hold After Read Clock	XP2-17		—		—		—	ps
$t_{DQVBCLKXGMII}$	Data Valid Before Clock	XP2-17		—		—		—	ps
$t_{DQVACLKXGMII}$	Data Valid After Clock	XP2-17		—		—		—	ps
Primary									
f_{MAX_PRI}	Frequency for Primary Clock Tree	XP2-17	—	420	—	420	—	420	MHz
t_{W_PRI}	Clock Pulse Width for Primary Clock	XP2-17	1	—	1	—	1	—	ns
t_{SKEW_PRI}	Primary Clock Skew Within a Bank	XP2-17	—	160	—	160	—	160	ps
Edge Clock (ECLK1 and ECLK2)									
f_{MAX_ECLK}	Frequency for Edge Clock	XP2-17	—	420	—	420	—	420	MHz
t_{W_ECLK}	Clock Pulse Width for Edge Clock	XP2-17	1	—	1	—	1	—	ns
t_{SKEW_ECLK}	Edge Clock Skew Within an Edge of the Device	XP2-17	—	130	—	130	—	130	ps

1. General timing numbers based on LVCMOS 2.5, 12mA, 0pf load.
2. DDR timing numbers based on SSTL25.
3. DDR2 timing numbers based on SSTL18.
4. See Figure 3-7 for definition of RDTCLK * 3.5.
5. XGMII timing numbers based on HSTL class I.

Timing v. A 0.07

Figure 3-6. DDR and DDR2 Parameters

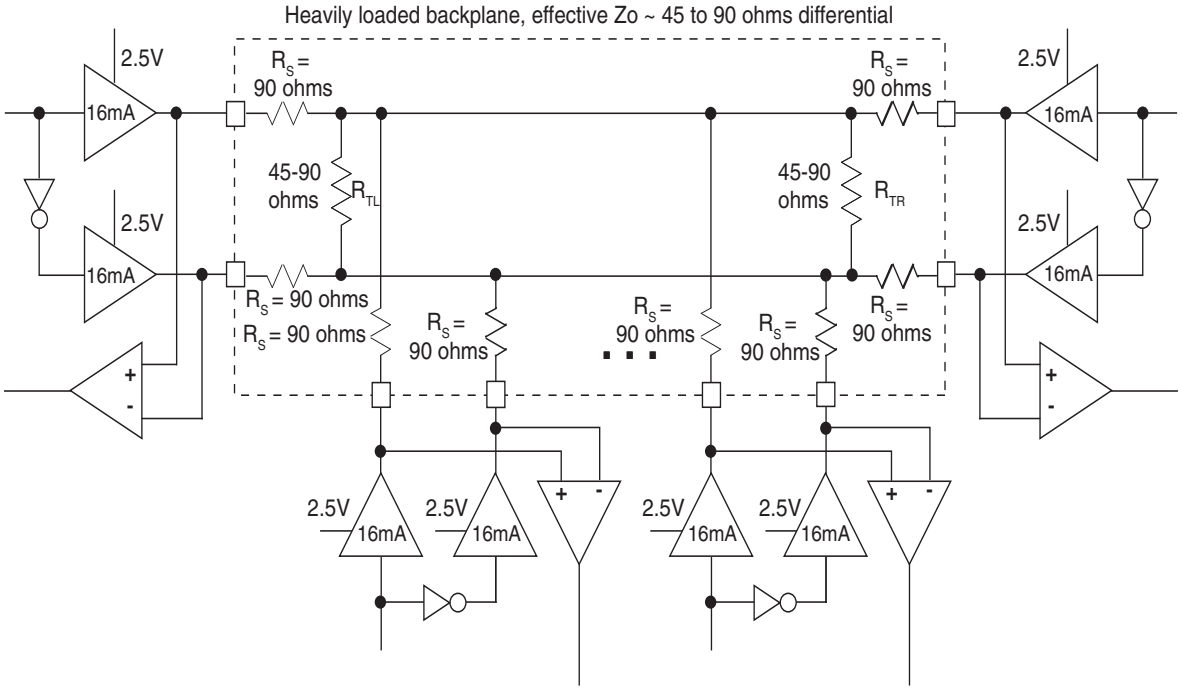
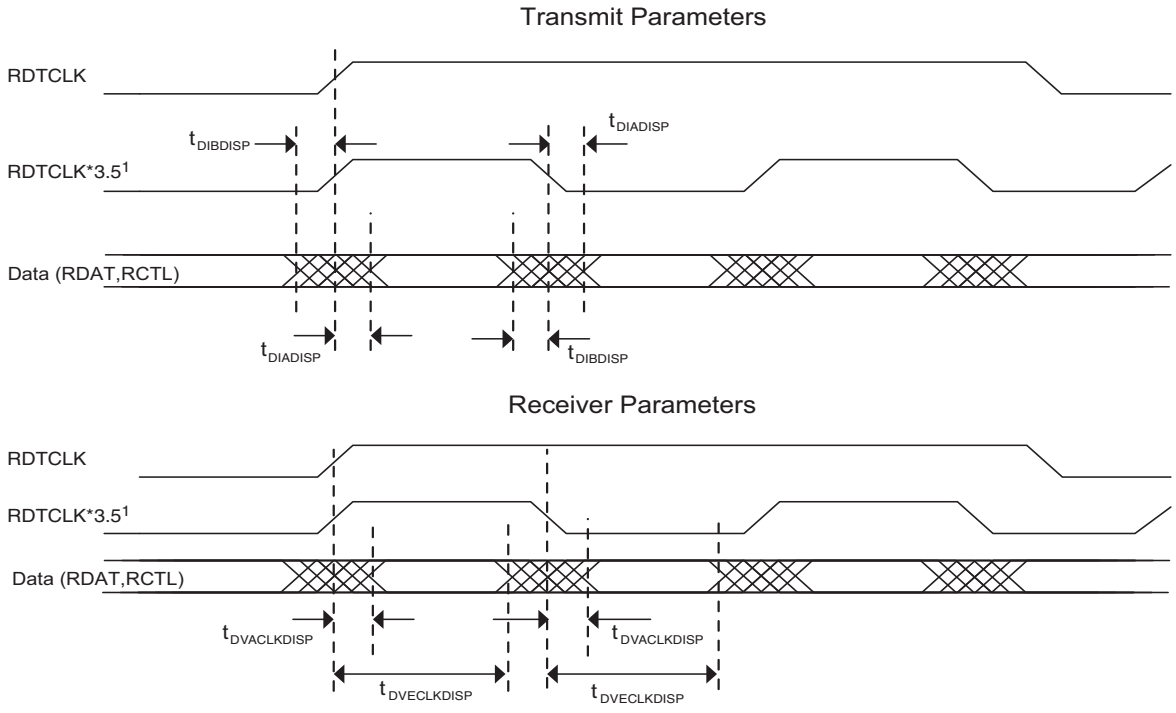
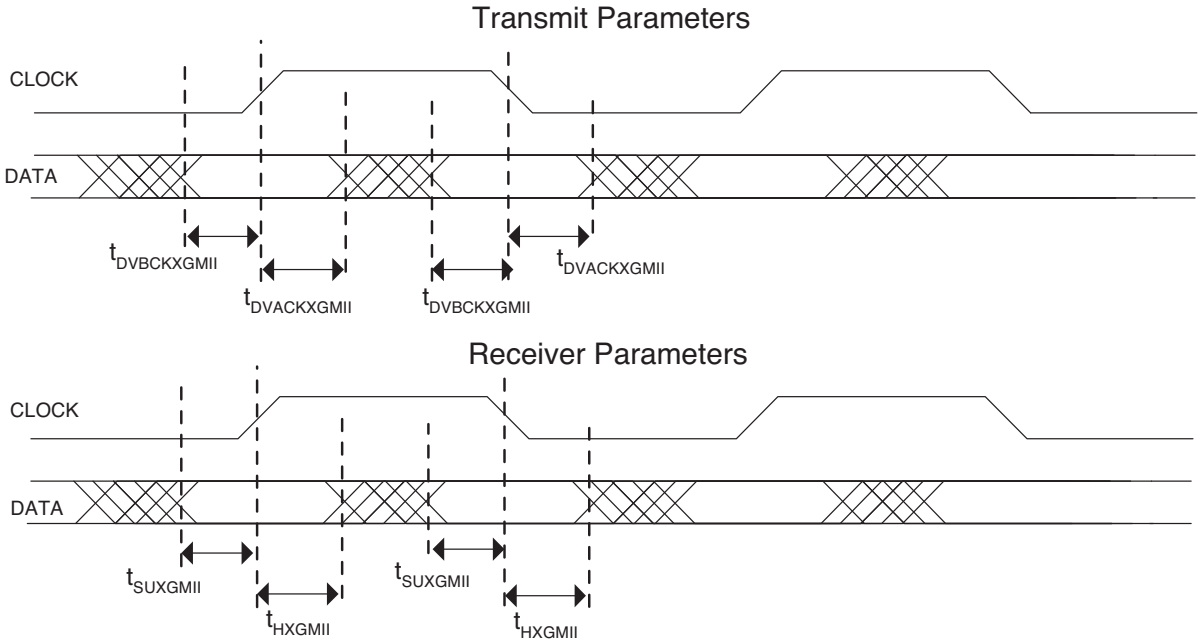


Figure 3-7. Source Synchronous Parameters for 7:1 LVDS Display Interfaces



1. For convenience, timing is specified relative to RDTCLK*3.5, an idealized reference clock that runs 3.5 times the speed of the 7:1 interface clock (RDTCLK) and has a 50% duty cycle.

Figure 3-8. XGMII Parameters



LatticeXP2 Internal Switching Characteristics[1]

Over Recommended Operating Conditions

Parameter	Description	-7		-6		-5		Units
		Min.	Max.	Min.	Max.	Min.	Max.	
PFU/PFF Logic Mode Timing								
t_{LUT4_PFU}	LUT4 delay (A to D inputs to F output)	—	0.218	—	0.239	—	0.260	ns
t_{LUT6_PFU}	LUT6 delay (A to D inputs to OFX output)	—	0.420	—	0.457	—	0.494	ns
t_{LSR_PFU}	Set/Reset to output of PFU (Asynchronous)	—	0.690	—	0.754	—	0.818	ns
t_{SUM_PFU}	Clock to Mux (M0,M1) Input Setup Time	0.147	—	0.147	—	0.148	—	ns
t_{HM_PFU}	Clock to Mux (M0,M1) Input Hold Time	-0.059	—	-0.056	—	-0.053	—	ns
t_{SUD_PFU}	Clock to D input setup time	0.070	—	0.082	—	0.093	—	ns
t_{HD_PFU}	Clock to D input hold time	0.002	—	0.003	—	0.003	—	ns
t_{CK2Q_PFU}	Clock to Q delay, (D-type Register Configuration)	—	0.328	—	0.356	—	0.383	ns
t_{RSTREC_PFU}	Asynchronous reset recovery time for PFU Logic	—	0.498	—	0.623	—	0.748	ns
t_{RST_PFU}	Asynchronous reset time for PFU Logic	—	0.690	—	0.754	—	0.818	ns
PFU Dual Port Memory Mode Timing								
t_{CORAM_PFU}	Clock to Output (F Port)	—	1.037	—	1.244	—	1.452	ns
t_{SUDATA_PFU}	Data Setup Time	-0.198	—	-0.236	—	-0.274	—	ns
t_{HDATA_PFU}	Data Hold Time	0.229	—	0.271	—	0.312	—	ns
t_{SUADDR_PFU}	Address Setup Time	-0.282	—	-0.327	—	-0.371	—	ns
t_{HADDR_PFU}	Address Hold Time	0.282	—	0.327	—	0.371	—	ns
t_{SUWREN_PFU}	Write/Read Enable Setup Time	-0.140	—	-0.167	—	-0.193	—	ns
t_{HWREN_PFU}	Write/Read Enable Hold Time	0.152	—	0.179	—	0.207	—	ns
PIO Input/Output Buffer Timing								
t_{IN_PIO}	Input Buffer Delay (LVCMOS25)	—	0.609	—	0.641	—	0.674	ns
t_{OUT_PIO}	Output Buffer Delay (LVCMOS25)	—	1.029	—	1.029	—	1.246	ns
IOLOGIC Input/Output Timing								
t_{SUI_PIO}	Input Register Setup Time (Data Before Clock)	0.596	—	0.645	—	0.694	—	ns
t_{HI_PIO}	Input Register Hold Time (Data after Clock)	-0.570	—	-0.614	—	-0.658	—	ns
t_{COO_PIO}	Output Register Clock to Output Delay	—	0.61	—	0.66	—	0.72	ns
t_{SUCE_PIO}	Input Register Clock Enable Setup Time	0.032	—	0.037	—	0.041	—	ns
t_{HCE_PIO}	Input Register Clock Enable Hold Time	-0.022	—	-0.025	—	-0.028	—	ns
t_{SULSR_PIO}	Set/Reset Setup Time	0.184	—	0.201	—	0.217	—	ns
t_{HLSR_PIO}	Set/Reset Hold Time	-0.080	—	-0.086	—	-0.093	—	ns
t_{RSTREC_PIO}	Asynchronous reset recovery time for IO Logic	0.228	—	0.247	—	0.266	—	ns

LatticeXP2 Internal Switching Characteristics[1] (Continued)

Over Recommended Operating Conditions

Parameter	Description	-7		-6		-5		Units
		Min.	Max.	Min.	Max.	Min.	Max.	
t_{RST_PIO}	Asynchronous reset time for PFU Logic	—	0.386	—	0.419	—	0.452	ns
t_{DEL}	Dynamic Delay Step Size	0.035	0.035	0.035	0.035	0.035	0.035	ns
EBR Timing								
t_{CO_EBR}	Clock (Read) to Output from Address or Data	—	2.77	—	3.14	—	3.51	ns
t_{COO_EBR}	Clock (Write) to Output from EBR Output Register	—	0.36	—	0.41	—	0.46	ns
t_{SUDATA_EBR}	Setup Data to EBR Memory (Write Clk)	-0.167	—	-0.198	—	-0.229	—	ns
t_{HDATA_EBR}	Hold Data to EBR Memory (Write Clk)	0.194	—	0.231	—	0.267	—	ns
t_{SUADDR_EBR}	Setup Address to EBR Memory (Write Clk)	-0.117	—	-0.137	—	-0.157	—	ns
t_{HADDR_EBR}	Hold Address to EBR Memory (Write Clk)	0.157	—	0.182	—	0.207	—	ns
t_{SUWREN_EBR}	Setup Write/Read Enable to EBR Memory (Write/Read Clk)	-0.135	—	-0.159	—	-0.182	—	ns
t_{HWREN_EBR}	Hold Write/Read Enable to EBR Memory (Write/Read Clk)	0.158	—	0.186	—	0.214	—	ns
t_{SUCE_EBR}	Clock Enable Setup Time to EBR Output Register (Read Clk)	0.144	—	0.160	—	0.176	—	ns
t_{HCE_EBR}	Clock Enable Hold Time to EBR Output Register (Read Clk)	-0.097	—	-0.113	—	-0.129	—	ns
t_{RSTO_EBR}	Reset To Output Delay Time from EBR Output Register (Asynchronous)	—	1.16	—	1.34	—	1.53	ns
t_{SUBE_EBR}	Byte Enable Set-Up Time to EBR Output Register	-0.117	—	-0.137	—	-0.157	—	ns
t_{HBE_EBR}	Byte Enable Hold Time to EBR Output Register Dynamic Delay on Each PIO	0.157	—	0.182	—	0.207	—	ns
t_{RSTREC_EBR}	Asynchronous reset recovery time for EBR	0.203	—	0.253	—	0.302	—	ns
t_{RST_EBR}	Asynchronous reset time for EBR	—	1.156	—	1.341	—	1.526	ns
PLL Parameters								
$t_{RSTKREC\_PLL}$	After RSTK De-assert, Recovery Time Before Next Clock Edge Can Toggle K-divider Counter	1.0	—	1.0	—	1.0	—	ns
t_{RSTREC_PLL}	After RST De-assert, Recovery Time Before Next Clock Edge Can Toggle M-divider Counter (Applies to M-Divider Portion of RST Only[2])	1.0	—	1.0	—	1.0	—	ns
DSP Block Timing								
t_{SUI_DSP}	Input Register Setup Time	0.14	—	0.15	—	0.17	—	ns
t_{HI_DSP}	Input Register Hold Time	0.02	—	-0.01	—	-0.03	—	ns
t_{SUP_DSP}	Pipeline Register Setup Time	2.51	—	2.78	—	3.06	—	ns
t_{HP_DSP}	Pipeline Register Hold Time	-0.79	—	-0.89	—	-0.99	—	ns

LatticeXP2 Internal Switching Characteristics[1] (Continued)

Over Recommended Operating Conditions

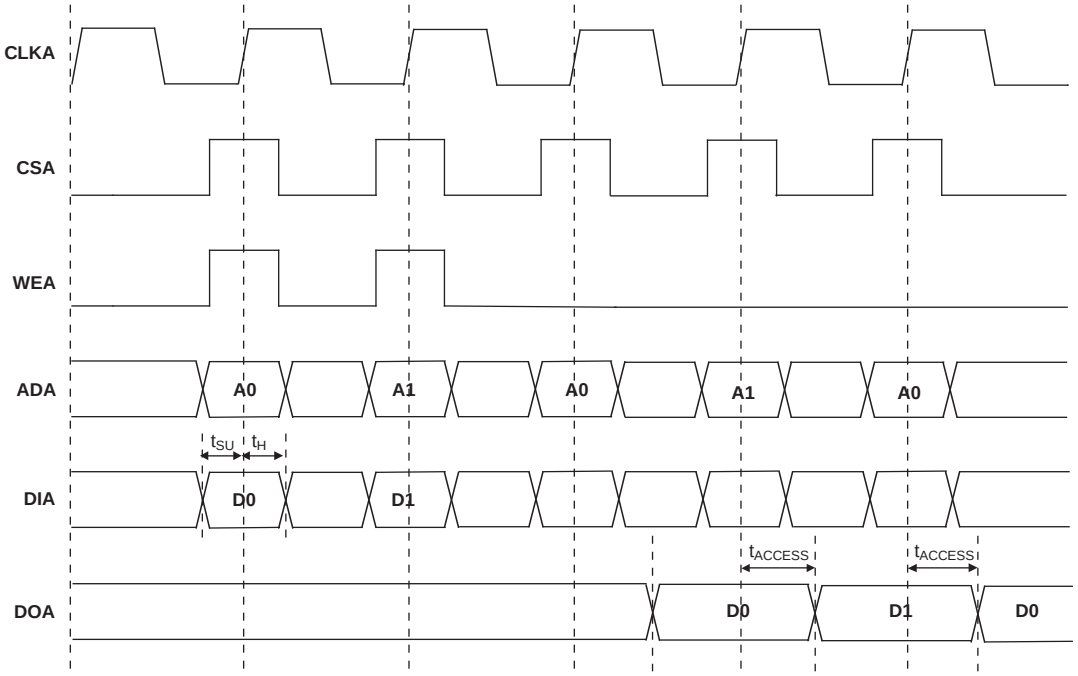
Parameter	Description	-7		-6		-5		Units
		Min.	Max.	Min.	Max.	Min.	Max.	
t_{SUO_DSP}	Output Register Setup Time	4.90	—	5.41	—	5.93	—	ns
t_{HO_DSP}	Output Register Hold Time	-1.44	—	-1.60	—	-1.77	—	ns
t_{COI_DSP}[3]	Input Register Clock to Output Time	—	4.51	—	4.95	—	5.38	ns
t_{COP_DSP}[3]	Pipeline Register Clock to Output Time	—	2.15	—	2.27	—	2.39	ns
t_{COO_DSP}[3]	Output Register Clock to Output Time	—	0.57	—	0.60	—	0.63	ns
$t_{SUADSUB}$	AdSub Input Register Setup Time	-0.27	—	-0.30	—	-0.33	—	ns
t_{HADSUB}	AdSub Input Register Hold Time	0.31	—	0.34	—	0.37	—	ns

1. Internal parameters are characterized, but not tested on every device.
2. RST resets VCO and all counters in PLL.
3. These parameters include the Adder Subtractor block in the path.

Timing v. A 0.07

Timing Diagrams

Figure 3-9. Read/Write Mode (Normal)



Note: Input data and address are registered at the positive edge of the clock and output data appears after the positive edge of the clock.

Figure 3-10. Read/Write Mode with Input and Output Registers

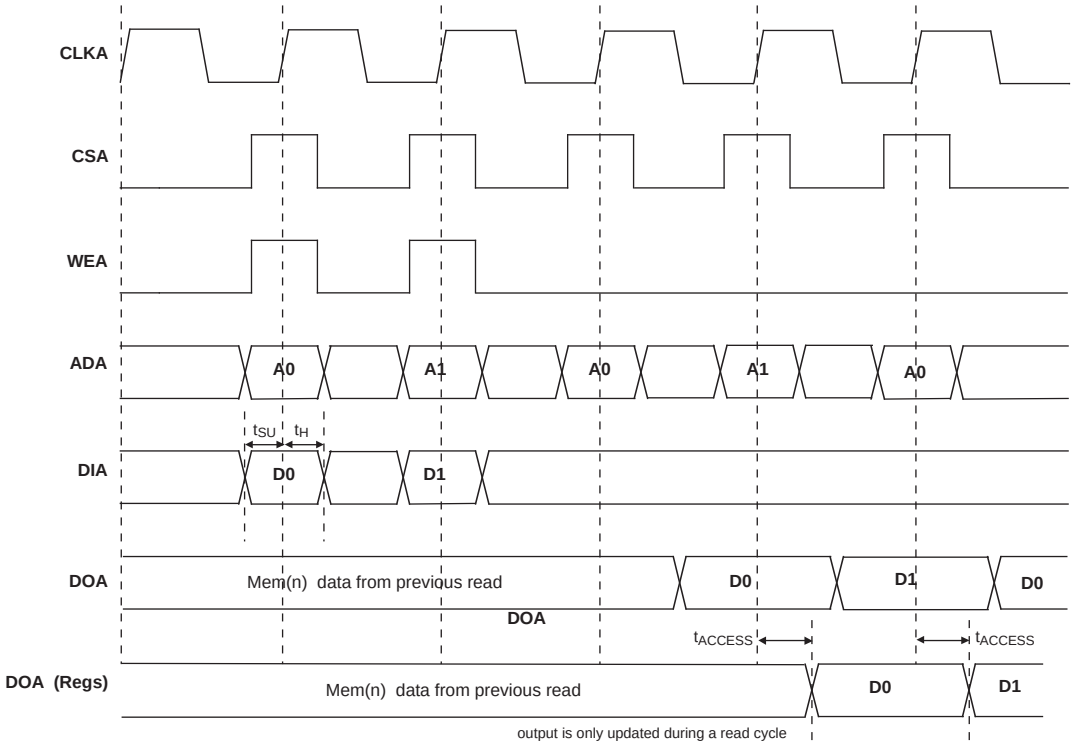
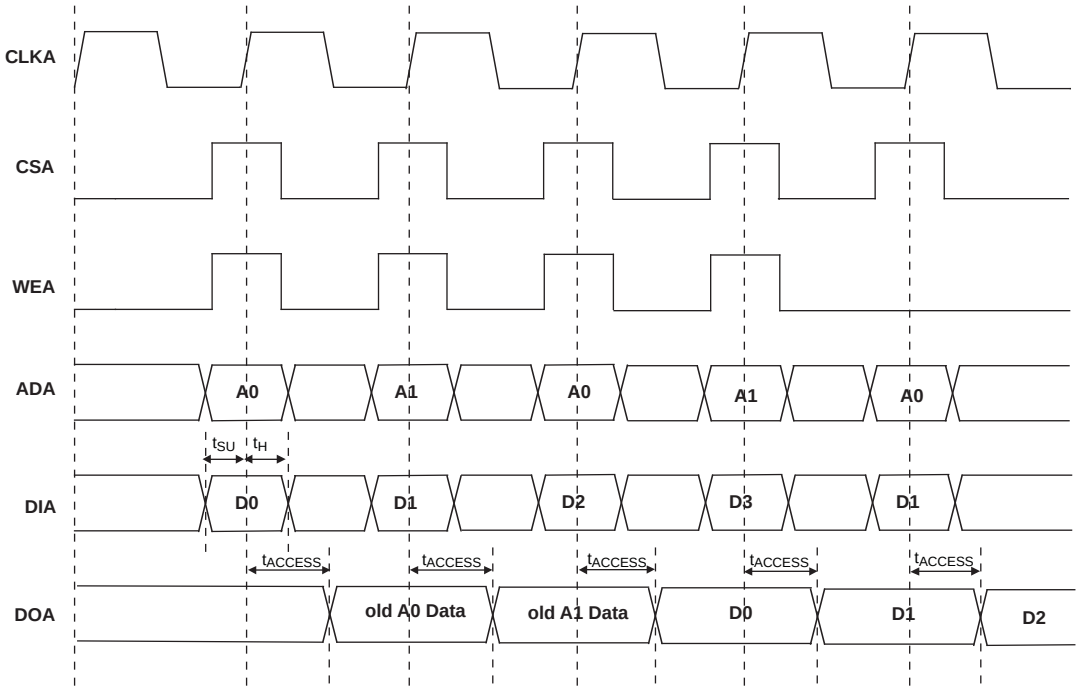
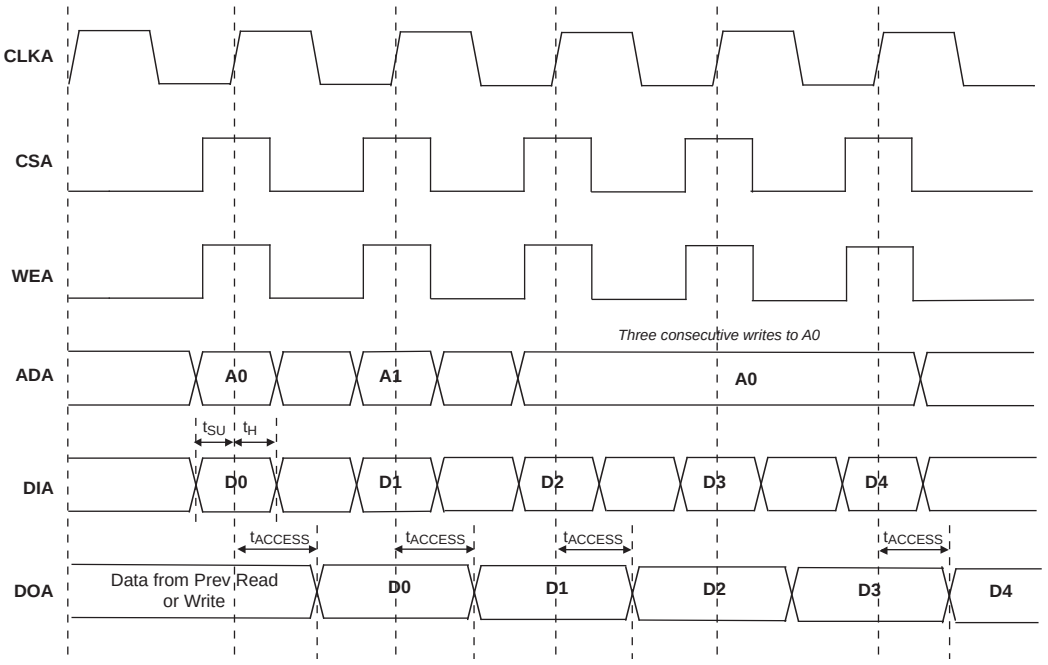


Figure 3-11. Read Before Write (SP Read/Write on Port A, Input Registers Only)



Note: Input data and address are registered at the positive edge of the clock and output data appears after the positive edge of the clock.

Figure 3-12. Write Through (SP Read/Write on Port A, Input Registers Only)



Note: Input data and address are registered at the positive edge of the clock and output data appears after the positive edge of the clock.

LatticeXP2 Family Timing Adders[1, 2, 3]

Over Recommended Operating Conditions

Buffer Type	Description	-7	-6	-5	Units
Input Adjusters					
LVDS25	LVDS	-0.01	0.02	0.04	ns
BLVDS25	BLVDS	-0.01	0.02	0.04	ns
MLVDS	LVDS	-0.01	0.02	0.04	ns
RSDS	RSDS	-0.01	0.02	0.04	ns
LVPECL33	LVPECL	-0.01	0.02	0.04	ns
HSTL18_I	HSTL_18 class I	0.02	0.05	0.07	ns
HSTL18_II	HSTL_18 class II	0.02	0.05	0.07	ns
HSTL18D_I	Differential HSTL 18 class I	-0.03	-0.01	0.02	ns
HSTL18D_II	Differential HSTL 18 class II	-0.03	-0.01	0.02	ns
HSTL15_I	HSTL_15 class I	0.02	0.04	0.06	ns
HSTL15D_I	Differential HSTL 15 class I	-0.03	-0.01	0.01	ns
SSTL33_I	SSTL_3 class I	0.05	0.09	0.12	ns
SSTL33_II	SSTL_3 class II	0.05	0.09	0.12	ns
SSTL33D_I	Differential SSTL_3 class I	-0.02	0.01	0.04	ns
SSTL33D_II	Differential SSTL_3 class II	-0.02	0.01	0.04	ns
SSTL25_I	SSTL_2 class I	0.04	0.07	0.10	ns
SSTL25_II	SSTL_2 class II	0.04	0.07	0.10	ns
SSTL25D_I	Differential SSTL_2 class I	-0.02	0.00	0.03	ns
SSTL25D_II	Differential SSTL_2 class II	-0.02	0.00	0.03	ns
SSTL18_I	SSTL_18 class I	0.02	0.05	0.07	ns
SSTL18_II	SSTL_18 class II	0.02	0.05	0.07	ns
SSTL18D_I	Differential SSTL_18 class I	-0.03	-0.01	0.02	ns
SSTL18D_II	Differential SSTL_18 class II	-0.03	-0.01	0.02	ns
LVTTL33	LVTTL	0.16	0.17	0.18	ns
LVCMOS33	LVCMOS 3.3	0.16	0.17	0.18	ns
LVCMOS25	LVCMOS 2.5	0.00	0.00	0.00	ns
LVCMOS18	LVCMOS 1.8	0.02	0.06	0.09	ns
LVCMOS15	LVCMOS 1.5	0.05	0.11	0.16	ns
LVCMOS12	LVCMOS 1.2	-0.10	-0.07	-0.04	ns
PCI33	3.3V PCI	0.16	0.17	0.18	ns
Output Adjusters					
LVDS25E	LVDS 2.5 E[4]	0.28	0.29	0.30	ns
LVDS25	LVDS 2.5	0.28	0.29	0.30	ns
BLVDS25	BLVDS 2.5	0.26	0.27	0.28	ns
MLVDS	MLVDS 2.5[4]	0.26	0.27	0.28	ns
RSDS	RSDS 2.5[4]	0.28	0.29	0.30	ns
LVPECL33	LVPECL 3.3[4]	0.16	0.17	0.18	ns
HSTL18_I	HSTL_18 class I 8mA drive	0.36	0.39	0.43	ns
HSTL18_II	HSTL_18 class II	0.25	0.27	0.29	ns
HSTL18D_I	Differential HSTL 18 class I 8mA drive	0.36	0.39	0.43	ns
HSTL18D_II	Differential HSTL 18 class II	0.25	0.27	0.29	ns

LatticeXP2 Family Timing Adders[1, 2, 3] (Continued)

Over Recommended Operating Conditions

Buffer Type	Description	-7	-6	-5	Units
HSTL15_I	HSTL_15 class I 4mA drive	0.85	0.96	1.06	ns
HSTL15D_I	Differential HSTL 15 class I 4mA drive	0.85	0.96	1.06	ns
SSTL33_I	SSTL_3 class I	0.28	0.32	0.35	ns
SSTL33_II	SSTL_3 class II	0.22	0.25	0.27	ns
SSTL33D_I	Differential SSTL_3 class I	0.28	0.32	0.35	ns
SSTL33D_II	Differential SSTL_3 class II	0.22	0.25	0.27	ns
SSTL25_I	SSTL_2 class I 8mA drive	0.28	0.29	0.30	ns
SSTL25_II	SSTL_2 class II 16mA drive	0.26	0.27	0.28	ns
SSTL25D_I	Differential SSTL_2 class I 8mA drive	0.28	0.29	0.30	ns
SSTL25D_II	Differential SSTL_2 class II 16mA drive	0.26	0.27	0.28	ns
SSTL18_I	SSTL_1.8 class I	0.36	0.39	0.43	ns
SSTL18_II	SSTL_1.8 class II 8mA drive	0.36	0.39	0.42	ns
SSTL18D_I	Differential SSTL_1.8 class I	0.36	0.39	0.43	ns
SSTL18D_II	Differential SSTL_1.8 class II 8mA drive	0.36	0.39	0.42	ns
LVTTL33_4mA	LVTTL 4mA drive	0.16	0.21	0.26	ns
LVTTL33_8mA	LVTTL 8mA drive	0.08	0.09	0.10	ns
LVTTL33_12mA	LVTTL 12mA drive	0.02	0.03	0.04	ns
LVTTL33_16mA	LVTTL 16mA drive	0.11	0.12	0.14	ns
LVTTL33_20mA	LVTTL 20mA drive	0.07	0.08	0.09	ns
LVCMOS33_4mA	LVCMOS 3.3 4mA drive, fast slew rate	0.16	0.21	0.26	ns
LVCMOS33_8mA	LVCMOS 3.3 8mA drive, fast slew rate	0.08	0.09	0.10	ns
LVCMOS33_12mA	LVCMOS 3.3 12mA drive, fast slew rate	0.02	0.03	0.04	ns
LVCMOS33_16mA	LVCMOS 3.3 16mA drive, fast slew rate	0.11	0.12	0.14	ns
LVCMOS33_20mA	LVCMOS 3.3 20mA drive, fast slew rate	0.07	0.08	0.09	ns
LVCMOS25_4mA	LVCMOS 2.5 4mA drive, fast slew rate	0.11	0.12	0.13	ns
LVCMOS25_8mA	LVCMOS 2.5 8mA drive, fast slew rate	0.05	0.05	0.05	ns
LVCMOS25_12mA	LVCMOS 2.5 12mA drive, fast slew rate	0.00	0.00	0.00	ns
LVCMOS25_16mA	LVCMOS 2.5 16mA drive, fast slew rate	0.08	0.08	0.08	ns
LVCMOS25_20mA	LVCMOS 2.5 20mA drive, fast slew rate	0.05	0.05	0.04	ns
LVCMOS18_4mA	LVCMOS 1.8 4mA drive, fast slew rate	0.07	0.08	0.10	ns
LVCMOS18_8mA	LVCMOS 1.8 8mA drive, fast slew rate	0.02	0.02	0.02	ns
LVCMOS18_12mA	LVCMOS 1.8 12mA drive, fast slew rate	-0.03	-0.03	-0.03	ns
LVCMOS18_16mA	LVCMOS 1.8 16mA drive, fast slew rate	0.03	0.03	0.03	ns
LVCMOS15_4mA	LVCMOS 1.5 4mA drive, fast slew rate	0.08	0.09	0.11	ns
LVCMOS15_8mA	LVCMOS 1.5 8mA drive, fast slew rate	0.01	0.01	0.00	ns
LVCMOS12_2mA	LVCMOS 1.2 2mA drive, fast slew rate	0.07	0.08	0.08	ns
LVCMOS12_6mA	LVCMOS 1.2 6mA drive, fast slew rate	-0.02	-0.02	-0.02	ns
LVCMOS33_4mA	LVCMOS 3.3 4mA drive, slow slew rate	1.51	1.68	1.84	ns
LVCMOS33_8mA	LVCMOS 3.3 8mA drive, slow slew rate	1.27	1.42	1.58	ns
LVCMOS33_12mA	LVCMOS 3.3 12mA drive, slow slew rate	1.09	1.24	1.38	ns
LVCMOS33_16mA	LVCMOS 3.3 16mA drive, slow slew rate	1.30	1.46	1.61	ns
LVCMOS33_20mA	LVCMOS 3.3 20mA drive, slow slew rate	1.11	1.25	1.40	ns

LatticeXP2 Family Timing Adders[1, 2, 3] (Continued)

Over Recommended Operating Conditions

Buffer Type	Description	-7	-6	-5	Units
LVCMOS25_4mA	LVCMOS 2.5 4mA drive, slow slew rate	1.58	1.70	1.81	ns
LVCMOS25_8mA	LVCMOS 2.5 8mA drive, slow slew rate	1.31	1.42	1.52	ns
LVCMOS25_12mA	LVCMOS 2.5 12mA drive, slow slew rate	1.12	1.23	1.33	ns
LVCMOS25_16mA	LVCMOS 2.5 16mA drive, slow slew rate	1.34	1.45	1.55	ns
LVCMOS25_20mA	LVCMOS 2.5 20mA drive, slow slew rate	1.14	1.24	1.35	ns
LVCMOS18_4mA	LVCMOS 1.8 4mA drive, slow slew rate	1.54	1.65	1.75	ns
LVCMOS18_8mA	LVCMOS 1.8 8mA drive, slow slew rate	1.25	1.35	1.45	ns
LVCMOS18_12mA	LVCMOS 1.8 12mA drive, slow slew rate	1.06	1.16	1.26	ns
LVCMOS18_16mA	LVCMOS 1.8 16mA drive, slow slew rate	1.28	1.38	1.48	ns
LVCMOS15_4mA	LVCMOS 1.5 4mA drive, slow slew rate	1.49	1.60	1.71	ns
LVCMOS15_8mA	LVCMOS 1.5 8mA drive, slow slew rate	0.01	0.01	0.00	ns
LVCMOS12_2mA	LVCMOS 1.2 2mA drive, slow slew rate	1.43	1.54	1.65	ns
LVCMOS12_6mA	LVCMOS 1.2 6mA drive, slow slew rate	-0.02	-0.02	-0.02	ns
PCI33	3.3V PCI	0.25	0.25	0.26	ns

1. Timing Adders are characterized but not tested on every device.
2. LVCMOS timing measured with the load specified in Switching Test Condition table.
3. All other standards tested according to the appropriate specifications.
4. These timing adders are measured with the recommended resistor values.

Timing v. A 0.07

sysCLOCK PLL Timing

Over Recommended Operating Conditions

Parameter	Description	Conditions	Min.	Typ.	Max.	Units
f_{IN}	Input Clock Frequency (CLKI, CLKFB)		10	—	435	MHz
f_{OUT}	Output Clock Frequency (CLKOP, CLKOS)		10	—	435	MHz
f_{OUT2}	K-Divider Output Frequency	CLKOK		—	217.5	MHz
		CLKOK2		—	145	MHz
f_{VCO}	PLL VCO Frequency		435	—	870	MHz
f_{PFD}	Phase Detector Input Frequency		10	—	435	MHz
AC Characteristics						
t_{DT}	Output Clock Duty Cycle	Default duty cycle selected [3]	45	50	55	%
t_{CPA}	Coarse Phase Adjust		-5	0	5	%
t_{PH} [4]	Output Phase Accuracy		—	—		
t_{OPJIT} [1]	Output Clock Period Jitter	f_{OUT} > 400 MHz	—	—	±100	ps
		100 MHz > f_{OUT} > 400 MHz	—	—	±100	ps
		f_{OUT} < 100 MHz	—	—	0.02	UIPP
t_{SK}	Input Clock to Output Clock Skew	N/M = integer	—	—	±200	ps
t_{OPW}	Output Clock Pulse Width	At 90% or 10%	1	—	—	ns
t_{LOCK} [2]	PLL Lock-in Time	25 to 420MHz			50	µs
		10 to 25MHz	—	—	100	µs
t_{IPJIT}	Input Clock Period Jitter		—	—	±200	ps
t_{FBKDLY}	External Feedback Delay		—	—	10	ns
t_{HI}	Input Clock High Time	90% to 90%	0.5	—	—	ns
t_{LO}	Input Clock Low Time	10% to 10%	0.5	—	—	ns
t_{RSTKW}	Reset Signal Pulse Width (RSTK)			—	—	ns
t_{RSTW}	Reset Signal Pulse Width (RST)			—	—	ns

1. Jitter sample is taken over 10,000 samples of the primary PLL output with clean reference clock.
2. Output clock is valid after t_{LOCK} for PLL reset and dynamic delay adjustment.
3. Using LVDS output buffers.
4. Relative to CLKOP.

Timing v. A 0.07

LatticeXP2 sysCONFIG Port Timing Specifications

Over Recommended Operating Conditions

Parameter	Description	Min	Max	Units
sysCONFIG POR, Initialization and Wake Up				
t_{ICFG}	Minimum Vcc to INITN High	—	50	ms
t_{VMC}	Time from tICFG to valid Master CCLK	—	2	µs
t_{PRGMRJ}	PROGRAMN Pin Pulse Rejection	—	12	ns
t_{PRGM}	PROGRAMN Low Time to Start Configuration	50	—	ns
t_{DINIT}	PROGRAMN High to INITN High Delay	—	1	ms
$t_{DPPINIT}$	Delay Time from PROGRAMN Low to INITN Low	—	50	ns
$t_{DPPDONE}$	Delay Time from PROGRAMN Low to DONE Low	—	50	ns
t_{IODISS}	User I/O Disable from PROGRAMN Low	—	35	ns
t_{IOENSS}	User I/O Enabled Time from CCLK Edge During Wake-up Sequence	—	25	ns
t_{MWC}	Additional Wake Master Clock Signals after DONE Pin High	0	—	cycles
sysCONFIG SPI Port (Master)				
t_{CFGX}	INITN High to CCLK Low	—	1	µs
t_{CSSPI}	INITN High to CSSPIN Low	—	2	µs
t_{CSCCLK}	CCLK Low before CSSPIN Low	0	—	ns
t_{SOCDO}	CCLK Low to Output Valid	—	15	ns
t_{SOE}	CSSPIN[0:1] Active Setup Time	300	—	ns
t_{CSPID}	CSSPIN[0:1] Low to First CCLK Edge Setup Time	2cyc	600+6cyc	ns
f_{MAXSPI}	Max CCLK Frequency	—	20	MHz
t_{SUSPI}	SOSPI Data Setup Time Before CCLK	7	—	ns
t_{HSPI}	SOSPI Data Hold Time After CCLK	10	—	ns
	Master Clock Frequency	Selected value -30%	Selected value +30%	MHz
	Duty Cycle	40	60	%
sysCONFIG SPI Port (Slave)				
$f_{MAXSPIS}$	Slave CCLK Frequency	—	25	MHz
t_{RF}	Rise and Fall Time	50		mV/ns
t_{STCO}	Falling Edge of CCLK to SOSPI Active		20	ns
t_{STOZ}	Falling Edge of CCLK to SOSPI Disable		20	ns
t_{STSU}	Data Setup Time (SISPI)	8		ns
t_{STH}	Data Hold Time (SISPI)	10		ns
t_{STCKH}	CCLK Clock Pulse Width, High	0.02	200	µs
t_{STCKL}	CCLK Clock Pulse Width, Low	0.02	200	µs
t_{STVO}	Falling Edge of CCLK to Valid SOSPI Output		20	ns
t_{SCS}	CSSPIN High Time			ns
t_{SCSS}	CSSPIN Setup Time			ns
t_{SCSH}	CSSPIN Hold Time			ns

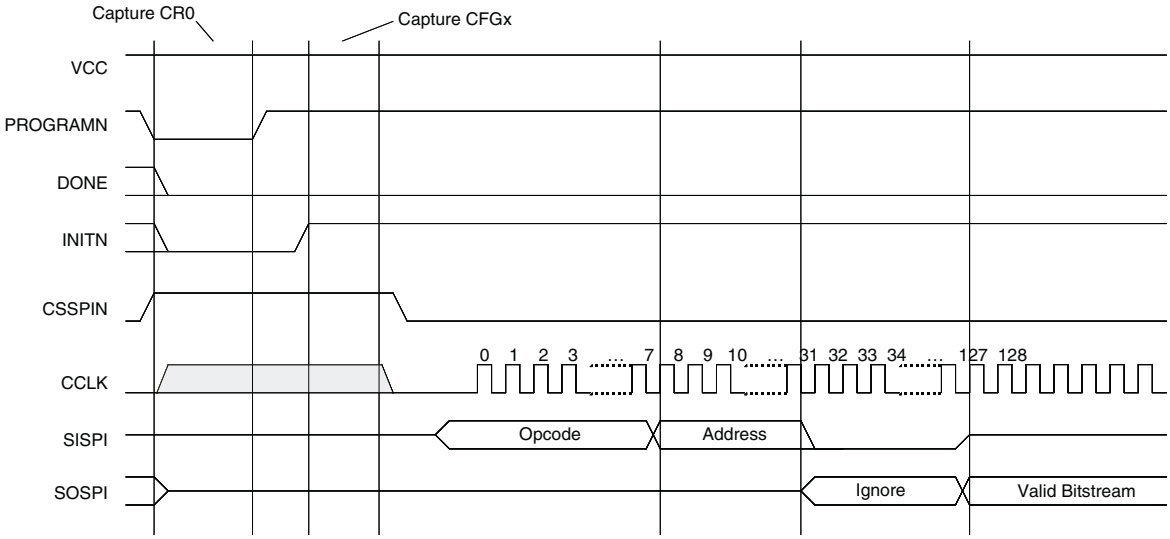
On-Chip Oscillator Characteristics

Over Recommended Operating Conditions

Parameter	Min.	Max.	Units
Master Clock Frequency	Selected value -30%	Selected value +30%	MHz
Master Clock Period Jitter			UIPP
Duty Cycle	40	60	%

Timing v. A 0.07

Figure 3-13. Master SPI Configuration Waveforms



Flash Download Time (from On-Chip Flash to SRAM)

Over Recommended Operating Conditions

Symbol	Parameter		Min.	Typ.	Max.	Units
$t_{REFRESH}$	PROGRAMN Low-to-High. Transition to Done High.	XP2-5	—			ms
		XP2-8	—			ms
		XP2-17	—	1.65		ms
		XP2-30	—			ms
		XP2-40	—			ms
	PROGRAMN $V_{CC} = V_{CC}$ Min.	XP2-5	—			ms
		XP2-8	—			ms
		XP2-17	—	1.65		ms
		XP2-30	—			ms
		XP2-40	—			ms

Flash Program Time

Over Recommended Operating Conditions

Device	Flash Density		Program Time		Units
			Typ.	Max.	
XP2-5	1.2M	TAG			Seconds
		Main Array			Seconds
XP2-8	2.0M	TAG			Seconds
		Main Array			Seconds
XP2-17	3.6M	TAG	0.02		Seconds
		Main Array	5.97		Seconds
XP2-30	6.0M	TAG			Seconds
		Main Array			Seconds
XP2-40	8.0M	TAG			Seconds
		Main Array			Seconds

Flash Erase Time

Over Recommended Operating Conditions

Device	Flash Density		Erase Time		Units
			Typ.	Max.	
XP2-5	1.2M	TAG			Seconds
		Main Array			Seconds
XP2-8	2.0M	TAG			Seconds
		Main Array			Seconds
XP2-17	3.6M	TAG	0.33		Seconds
		Main Array	4.20		Seconds
XP2-30	6.0M	TAG			Seconds
		Main Array			Seconds
XP2-40	8.0M	TAG			Seconds
		Main Array			Seconds

FlashBAK Program Time (from EBR to Flash)

Over Recommended Operating Conditions

Device	Flash Density	Program Time		Units
		Typ.	Max.	
XP2-5	1.2M			Seconds
XP2-8	2.0M			Seconds
XP2-17	3.6M	1.3		Seconds
XP2-30	6.0M			Seconds
XP2-40	8.0M			Seconds

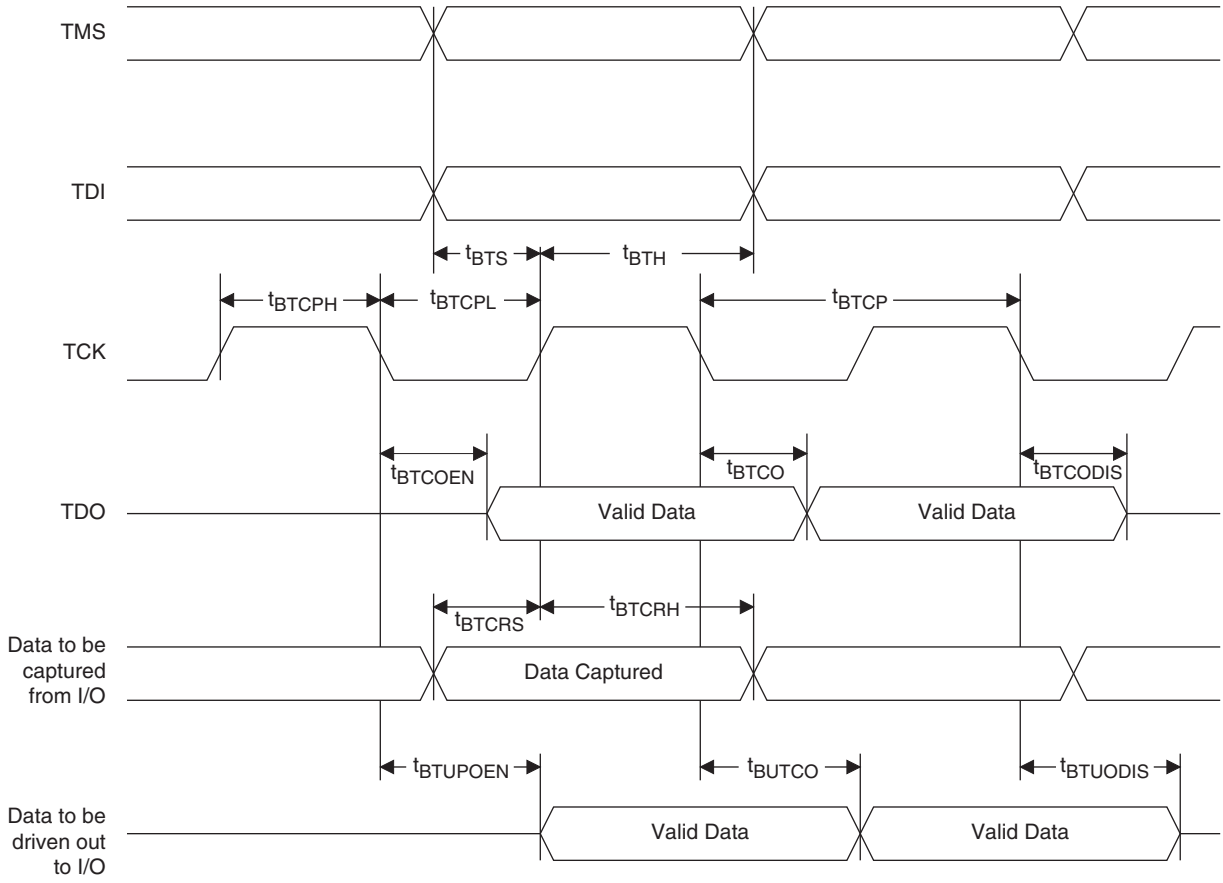
JTAG Port Timing Specifications

Over Recommended Operating Conditions

Symbol	Parameter	Min.	Max.	Units
f_{MAX}	TCK Clock Frequency	—	25	MHz
t_{BTCP}	TCK [BSCAN] clock pulse width	40	—	ns
t_{BTCPH}	TCK [BSCAN] clock pulse width high	20	—	ns
t_{BTCPL}	TCK [BSCAN] clock pulse width low	20	—	ns
t_{BTS}	TCK [BSCAN] setup time	8	—	ns
t_{BTH}	TCK [BSCAN] hold time	10	—	ns
t_{BTRF}	TCK [BSCAN] rise/fall time	50	—	mV/ns
t_{BTCO}	TAP controller falling edge of clock to valid output	—	10	ns
$t_{BTCODIS}$	TAP controller falling edge of clock to valid disable	—	10	ns
t_{BTCOEN}	TAP controller falling edge of clock to valid enable	—	10	ns
t_{BTCRS}	BSCAN test capture register setup time	8	—	ns
t_{BTCRH}	BSCAN test capture register hold time	25	—	ns
t_{BUTCO}	BSCAN test update register, falling edge of clock to valid output	—	25	ns
$t_{BTUODIS}$	BSCAN test update register, falling edge of clock to valid disable	—	25	ns
$t_{BTUPOEN}$	BSCAN test update register, falling edge of clock to valid enable	—	25	ns

Timing v. A 0.07

Figure 3-14. JTAG Port Timing Waveforms



Switching Test Conditions

LatticeXP2 Family Data Sheet Pinout Information

September 2007 Advance Data Sheet DS1009

Signal Descriptions

Signal Name	I/O	Description
General Purpose		
P[Edge] [Row/Column Number*]_[A/B]	I/O	[Edge] indicates the edge of the device on which the pad is located. Valid edge designations are L (Left), B (Bottom), R (Right), T (Top). [Row/Column Number] indicates the PFU row or the column of the device on which the PIC exists. When Edge is T (Top) or B (Bottom), only need to specify Row Number. When Edge is L (Left) or R (Right), only need to specify Column Number. [A/B] indicates the PIO within the PIC to which the pad is connected. Some of these user-programmable pins are shared with special function pins. These pins, when not used as special purpose pins, can be programmed as I/Os for user logic. During configuration the user-programmable I/Os are tri-stated with an internal pull-up resistor enabled. If any pin is not used (or not bonded to a package pin), it is also tri-stated with an internal pull-up resistor enabled after configuration.
GSRN	I	Global RESET signal (active low). Any I/O pin can be GSRN.
NC	—	No connect.
GND	—	Ground. Dedicated pins.
V_{CC}	—	Power supply pins for core logic. Dedicated pins.
V_{CCAUX}	—	Auxiliary power supply pin. This dedicated pin powers all the differential and referenced input buffers.
V_{CCPLL}	—	PLL supply pins. PQFP and TQFP packages only.
V_{CCIOx}	—	Dedicated power supply pins for I/O bank x.
V_{REF1_x}, V_{REF2_x}	—	Reference supply pins for I/O bank x. Pre-determined pins in each bank are assigned as V_{REF} inputs. When not used, they may be used as I/O pins.
PLL and Clock Functions (Used as user programmable I/O pins when not in use for PLL or clock pins)		
[LOC][num]_V_{CCPLL}	—	Power supply pin for PLL: LLC, LRC, URC, ULC, num = row from center.
[LOC][num]_GPLL[T, C]_IN_A	I	General Purpose PLL (GPLL) input pads: LLC, LRC, URC, ULC, num = row from center, T = true and C = complement, index A,B,C...at each side.
[LOC][num]_GPLL[T, C]_FB_A	I	Optional feedback GPLL input pads: LLC, LRC, URC, ULC, num = row from center, T = true and C = complement, index A,B,C...at each side.
PCLK[T, C]_[n:0]_[3:0]	I	Primary Clock pads, T = true and C = complement, n per side, indexed by bank and 0,1,2,3 within bank.
[LOC]DQS[num]	I	DQS input pads: T (Top), R (Right), B (Bottom), L (Left), DQS, num = ball function number. Any pad can be configured to be output.
Test and Programming (Dedicated Pins)		
TMS	I	Test Mode Select input, used to control the 1149.1 state machine. Pull-up is enabled during configuration.
TCK	I	Test Clock input pin, used to clock the 1149.1 state machine. No pull-up enabled.
TDI	I	Test Data in pin. Used to load data into device using 1149.1 state machine. After power-up, this TAP port can be activated for configuration by sending appropriate command. (Note: once a configuration port is selected it is locked. Another configuration port cannot be selected until the power-up sequence). Pull-up is enabled during configuration.

Signal Descriptions (Cont.)

Signal Name	I/O	Description
TDO	O	Output pin. Test Data Out pin used to shift data out of a device using 1149.1.
VCCJ	—	Power supply pin for JTAG Test Access Port.
Configuration Pads (Used during sysCONFIG)		
CFG[1:0]	I	Mode pins used to specify configuration mode values latched on rising edge of INITN. During configuration, an internal pull-up is enabled.
INITN	I/O	Open Drain pin. Indicates the FPGA is ready to be configured. During configuration, a pull-up is enabled.[1]
PROGRAMN	I	Initiates configuration sequence when asserted low. This pin always has an active pull-up.
DONE	I/O	Open Drain pin. Indicates that the configuration sequence is complete, and the startup sequence is in progress.
CCLK	I/O	Configuration Clock for configuring an FPGA in sysCONFIG mode.
SISPI	I/O	Input data pin in slave SPI mode and Output data pin in Master SPI mode.
SOSPI	I/O	Output data pin in slave SPI mode and Input data pin in Master SPI mode.
CSSPIN	O	Chip select for external SPI Flash memory in Master SPI mode. This pin has a weak internal pull-up.
CSSPISN	I	Chip select in Slave SPI mode. This pin has a weak internal pull-up.
TOE	I	Test Output Enable tristates all I/O pins when driven low. This pin has a weak internal pull-up, but when not used an external pull-up to V_{CC} is recommended.

1. If not actively drive, the internal pull-up may not be sufficient. An external pull-up resistor of 4.7k to 10k ohms is recommended.

PICs and DDR Data (DQ) Pins Associated with the DDR Strobe (DQS) Pin

PICs Associated with DQS Strobe	PIO Within PIC	DDR Strobe (DQS) and Data (DQ) Pins
For Left and Right Edges of the Device		
P[Edge] [n-4]	A	DQ
	B	DQ
P[Edge] [n-3]	A	DQ
	B	DQ
P[Edge] [n-2]	A	DQ
	B	DQ
P[Edge] [n-1]	A	DQ
	B	DQ
P[Edge] [n]	A	[Edge]DQSn
	B	DQ
P[Edge] [n+1]	A	DQ
	B	DQ
P[Edge] [n+2]	A	DQ
	B	DQ
P[Edge] [n+3]	A	DQ
	B	DQ
For Top and Bottom Edges of the Device		
P[Edge] [n-4]	A	DQ
	B	DQ
P[Edge] [n-3]	A	DQ
	B	DQ
P[Edge] [n-2]	A	DQ
	B	DQ
P[Edge] [n-1]	A	DQ
	B	DQ
P[Edge] [n]	A	[Edge]DQSn
	B	DQ
P[Edge] [n+1]	A	DQ
	B	DQ
P[Edge] [n+2]	A	DQ
	B	DQ
P[Edge] [n+3]	A	DQ
	B	DQ
P[Edge] [n+4]	A	DQ
	B	DQ

Notes:
1. “n” is a row PIC number.
2. The DDR interface is designed for memories that support one DQS strobe up to 16 bits of data for the left and right edges and up to 18 bits of data for the top and bottom edges. In some packages, all the potential DDR data (DQ) pins may not be available. PIC numbering definitions are provided in the “Signal Names” column of the Signal Descriptions table.

Pin Information Summary

Pin Type		XP2-17		
		208 PQFP	256 ftBGA	484 fpBGA
Single Ended User I/O		146	201	358
Differential Pair User I/O	Normal	57	77	135
	Highspeed	16	23	44
Configuration	TAP	5	5	5
	Muxed	9	9	9
	Dedicated	1	1	1
Non Configuration	Muxed	11	11	21
	Dedicated	1	1	1
Vcc		9	6	16
Vccaux		4	4	8
VCCPLL		4	-	-
VCCIO	Bank0	2	2	4
	Bank1	2	2	4
	Bank2	2	2	4
	Bank3	2	2	4
	Bank4	2	2	4
	Bank5	2	2	4
	Bank6	2	2	4
	Bank7	2	2	4
GND, GND0-GND7		22	20	56
NC		—	2	7
Single Ended/ Differential I/O per Bank	Bank0	20/10	28/14	52/26
	Bank1	18/9	22/11	36/18
	Bank2	18/9	26/13	46/23
	Bank3	16/8	24/12	44/22
	Bank4	18/9	26/13	36/18
	Bank5	20/10	24/12	52/26
	Bank6	18/9	27/13	46/23
	Bank7	18/9	24/12	46/23

Available Devices Resources per Packaged Device

Resource	Device	208 PQFP	256 ftBGA	484 fpBGA
PLL	XP2-17	4	4	4

PCI and DDR Capabilities of the Device-Package Combinations

Functionality	Device	208 PQFP	256 ftBGA	484 fpBGA
PCI (B edge)	XP2-17			
SPI (L/R edge)	XP2-17			
DDR (L/R/B edges)	XP2-17			
PCI + DDR	XP2-17			
PCI + SPI	XP2-17			
DDR + SPI	XP2-17			
PCI + DDR + SPI	XP2-17			

Power Supply and No Connect Connections

Signals	208 PQFP	256 ftBGA	484 fpBGA
VCC	4, 26, 27, 49, 79, 108, 131, 153, 182	G7, G9, H7, J10, K10, K8	N9, P10, J10, J11, J12, P11, P12, J13, K14, P13, K9, L14, L9, M14, M9, N14
VCCIO0	191, 200	C5, E7	B5, D10, E7, H9
VCCIO1	166, 172	C12, E10	B18, D13, E16, H14
VCCIO2	141, 152	E14, G12	E21, G18, J15, K19
VCCIO3	107, 117	K12, M14	N19, P15, T18, V21
VCCIO4	89, 95	M10, P12	AA18, R14, V16, W13
VCCIO5	61, 70	M7, P5	AA5, R9, V7, W10
VCCIO6	38, 50	K5, M3	N4, P8, T5, V2
VCCIO7	5, 16	E3, G5	E2, G5, J8, K4
VCCJ	130	K7	H15
VCCAUX	25, 80, 132, 181	G8, H10, J7, K9	H11, H12, L15, L8, M15, M8, R11, R12
VCCPLL	57, 99, 162, 204	—	—
GND, GND0-GND7	3, 14, 24, 37, 48, 58, 62, 71, 90, 96, 100, 109, 119, 133, 143, 154, 161, 165, 171, 190, 199, 203	A1, A16, B12, B5, C8, E15, E2, H14, H8, H9, J3, J8, J9, M15, M2, P9, R12, R5, T1, T16	A1, A22, AA19, AA4, AB1, AB22, B19, B4, C10, C13, D16, D2, D21, D7, G19, G4, H10, H13, J14, J9, K10, K11, K12, K13, K15, K20, K3, K8, L10, L11, L12, L13, M10, M11, M12, M13, N10, N11, N12, N13, N15, N20, N3, N8, P14, P9, R10, R13, T19, T4, W16, W2, W21, W7, Y10, Y13
NC	—	T15, G10	AB21, H8, R15, R8, U19, U17, U18

XP2-17 Logic Signal Connections: 208 PQFP

Ball Number	Ball Function	Bank	Dual Function	Differential
1	PL2A	7	VREF1_7	T
2	PL2B	7	VREF2_7	C
3	GNDIO7	7		
4	VCC	-		
5	VCCIO7	7		
6	PL17A	7		T (LVDS)*
7	PL17B	7		C (LVDS)*
8	PL18A	7	PROGRAMN	T
9	PL19A	7	CFG1	T (LVDS)*
10	PL18B	7	DONE	C
11	PL20A	7	CSSPISN	T
12	PL19B	7		C (LVDS)*
13	PL20B	7	CSSPIN	C
14	GNDIO7	7		
15	PL21A	7	LDQS21	T (LVDS)*
16	VCCIO7	7		
17	PL22A	7	CCLK	T
18	PL21B	7		C (LVDS)*
19	PL22B	7	SOSPI	C
20	PL23A	7	SISPI	T (LVDS)*
21	PL23B	7	INITN	C (LVDS)*
22	PL24A	7	PCLKT7_0	T
23	PL24B	7	PCLKC7_0	C
24	GND	-		
25	VCCAUX	-		
26	VCC	-		
27	VCC	-		
28	TOE	-		
29	CFG0	-		
30	PL26A	6	PCLKT6_0	T (LVDS)*
31	PL27A	6		T
32	PL26B	6	PCLKC6_0	C (LVDS)*
33	PL27B	6		C
34	PL28A	6		T (LVDS)*
35	PL28B	6		C (LVDS)*
36	PL29A	6		T
37	GNDIO6	6		
38	VCCIO6	6		
39	PL29B	6		C
40	PL31A	6		T
41	PL30A	6	LDQS30	T (LVDS)*
42	PL31B	6		C
43	PL30B	6		C (LVDS)*

XP2-17 Logic Signal Connections: 208 PQFP (Cont.)

Ball Number	Ball Function	Bank	Dual Function	Differential
44	PL32B	6		C (LVDS)*
45	PL32A	6		T (LVDS)*
46	PL33A	6		T
47	PL33B	6		C
48	GNDIO6	6		
49	VCC	-		
50	VCCIO6	6		
51	PL49A	6	VREF1_6	T
52	PL49B	6	VREF2_6	C
53	PB4A	5	LLC_GPLLT_IN_A	T
54	PB3A	5	VREF1_5	T
55	PB4B	5	LLC_GPLLC_IN_A	C
56	PB3B	5	VREF2_5	C
57	LLC_VCCPLL	-		
58	LLC_GNDPLL	-		
59	PB5A	5	LLC_GPLLT_FB_A	T
60	PB5B	5	LLC_GPLLC_FB_A	C
61	VCCIO5	5		
62	GNDIO5	5		
63	PB22B	5		C
64	PB22A	5		T
65	PB24A	5	BDQS24	T
66	PB23A	5		T
67	PB24B	5		C
68	PB23B	5		C
69	PB25A	5		T
70	VCCIO5	5		
71	GNDIO5	5		
72	PB25B	5		C
73	PB26A	5		T
74	PB26B	5		C
75	PB27A	5		T
76	PB28A	5	PCLKT5_0	T
77	PB27B	5		C
78	PB28B	5	PCLKC5_0	C
79	VCC	-		
80	VCCAUX	-		
81	PB29A	4	PCLKT4_0	T
82	PB29B	4	PCLKC4_0	C
83	PB30A	4		T
84	PB30B	4		C
85	PB31A	4		T
86	PB32A	4		T
87	PB31B	4		C

XP2-17 Logic Signal Connections: 208 PQFP (Cont.)

Ball Number	Ball Function	Bank	Dual Function	Differential
88	PB32B	4		C
89	VCCIO4	4		
90	GNDIO4	4		
91	PB33A	4	BDQS33	T
92	PB34A	4		T
93	PB33B	4		C
94	PB34B	4		C
95	VCCIO4	4		
96	GNDIO4	4		
97	PB44A	4	LRC_GPLLT_IN_A	T
98	PB44B	4	LRC_GPLLC_IN_A	C
99	LRC_VCCPLL	-		
100	LRC_GNDPLL	-		
101	PB45A	4	LRC_GPLLT_FB_A	T
102	PB46A	4	VREF1_4	T
103	PB45B	4	LRC_GPLLC_FB_A	C
104	PB46B	4	VREF2_4	C
105	PR48A	3	VREF1_3	T
106	PR48B	3	VREF2_3	C
107	VCCIO3	3		
108	VCC	-		
109	GNDIO3	3		
110	PR32A	3		T (LVDS)*
111	PR32B	3		C (LVDS)*
112	PR30B	3		C (LVDS)*
113	PR31B	3		C
114	PR30A	3	RDQS30	T (LVDS)*
115	PR31A	3		T
116	PR29B	3		C
117	VCCIO3	3		
118	PR29A	3		T
119	GNDIO3	3		
120	PR28B	3		C (LVDS)*
121	PR28A	3		T (LVDS)*
122	PR27B	3		C
123	PR26B	3	PCLKC3_0	C (LVDS)*
124	PR27A	3		T
125	PR26A	3	PCLKT3_0	T (LVDS)*
126	TMS	-		
127	TCK	-		
128	TDI	-		
129	TDO	-		
130	VCCJ	-		
131	VCC	-		

XP2-17 Logic Signal Connections: 208 PQFP (Cont.)

Ball Number	Ball Function	Bank	Dual Function	Differential
132	VCCAUX	-		
133	GND	-		
134	PR24B	2	PCLKC2_0	C
135	PR24A	2	PCLKT2_0	T
136	PR23B	2		C (LVDS)*
137	PR23A	2		T (LVDS)*
138	PR22B	2		C
139	PR21B	2		C (LVDS)*
140	PR22A	2		T
141	VCCIO2	2		
142	PR21A	2	RDQS21	T (LVDS)*
143	GNDIO2	2		
144	PR20B	2		C
145	PR19B	2		C (LVDS)*
146	PR20A	2		T
147	PR18B	2		C
148	PR19A	2		T (LVDS)*
149	PR18A	2		T
150	PR17B	2		C (LVDS)*
151	PR17A	2		T (LVDS)*
152	VCCIO2	2		
153	VCC	-		
154	GNDIO2	2		
155	PR2B	2	VREF2_2	C
156	PR2A	2	VREF1_2	T
157	PT46B	1	VREF2_1	C
158	PT45B	1	URC_GPLLC_FB_A	C
159	PT46A	1	VREF1_1	T
160	PT45A	1	URC_GPLLT_FB_A	T
161	URC_GNDPLL	-		
162	URC_VCCPLL	-		
163	PT44B	1	URC_GPLLC_IN_A	C
164	PT44A	1	URC_GPLLT_IN_A	T
165	GNDIO1	1		
166	VCCIO1	1		
167	PT34B	1		C
168	PT33B	1		C
169	PT34A	1		T
170	PT33A	1	TDQS33	T
171	GNDIO1	1		
172	VCCIO1	1		
173	PT32B	1		C
174	PT31B	1		C
175	PT32A	1		T

XP2-17 Logic Signal Connections: 208 PQFP (Cont.)

Ball Number	Ball Function	Bank	Dual Function	Differential
176	PT31A	1		T
177	PT30B	1		C
178	PT30A	1		T
179	PT29B	1	PCLKC1_0	C
180	PT29A	1	PCLKT1_0	T
181	VCCAUX	-		
182	VCC	-		
183	PT28B	0	PCLKC0_0	C
184	PT27B	0		C
185	PT28A	0	PCLKT0_0	T
186	PT27A	0		T
187	PT26B	0		C
188	PT26A	0		T
189	PT25B	0		C
190	GNDIO0	0		
191	VCCIO0	0		
192	PT25A	0		T
193	PT23B	0		C
194	PT24B	0		C
195	PT23A	0		T
196	PT24A	0	TDQS24	T
197	PT22A	0		T
198	PT22B	0		C
199	GNDIO0	0		
200	VCCIO0	0		
201	PT5B	0	ULC_GPLLC_FB_A	C
202	PT5A	0	ULC_GPLLT_FB_A	T
203	ULC_GNDPLL	-		
204	ULC_VCCPLL	-		
205	PT3B	0	VREF2_0	C
206	PT4B	0	ULC_GPLLC_IN_A	C
207	PT3A	0	VREF1_0	T
208	PT4A	0	ULC_GPLLT_IN_A	T

* Supports true LVDS. Other differential signals must be emulated with external resistors.

XP2-17 Logic Signal Connections: 256 ftBGA

Ball Number	Ball Function	Bank	Dual Function	Differential
VCCIO7	VCCIO7	7		
D3	PL2A	7	VREF1_7	T
E4	PL2B	7	VREF2_7	C
GND	GNDIO7	7		
VCCIO7	VCCIO7	7		
C3	PL14A	7		T
C2	PL14B	7		C
C1	PL15A	7		T (LVDS)*
D1	PL15B	7		C (LVDS)*
D2	PL16A	7		T
GND	GNDIO7	7		
E1	PL16B	7		C
VCCIO7	VCCIO7	7		
F2	PL17A	7		T (LVDS)*
G2	PL18A	7	PROGRAMN	T
F1	PL17B	7		C (LVDS)*
G4	PL18B	7	DONE	C
G1	PL19A	7	CFG1	T (LVDS)*
H1	PL19B	7		C (LVDS)*
F3	PL20A	7	CSSPISN	T
GND	GNDIO7	7		
G3	PL20B	7	CSSPIN	C
VCCIO7	VCCIO7	7		
H2	PL21A	7	LDQS21	T (LVDS)*
H6	PL22A	7	CCLK	T
H3	PL21B	7		C (LVDS)*
J6	PL22B	7	SOSPI	C
H5	PL23A	7	SISPI	T (LVDS)*
K6	PL23B	7	INITN	C (LVDS)*
H4	PL24A	7	PCLKT7_0	T
GND	GNDIO7	7		
J5	PL24B	7	PCLKC7_0	C
L6	TOE	-		
L7	CFG0	-		
VCCIO6	VCCIO6	6		
J2	PL26A	6	PCLKT6_0	T (LVDS)*
J4	PL27A	6		T
J1	PL26B	6	PCLKC6_0	C (LVDS)*
K4	PL27B	6		C
K2	PL28A	6		T (LVDS)*
K1	PL28B	6		C (LVDS)*
L5	PL29A	6		T
GND	GNDIO6	6		

XP2-17 Logic Signal Connections: 256 ftBGA (Cont.)

Ball Number	Ball Function	Bank	Dual Function	Differential
K3	PL29B	6		C
VCCIO6	VCCIO6	6		
L1	PL30A	6	LDQS30	T (LVDS)*
L4	PL31A	6		T
L2	PL30B	6		C (LVDS)*
L3	PL31B	6		C
M1	PL32A	6		T (LVDS)*
N1	PL32B	6		C (LVDS)*
M4	PL33A	6		T
GND	GNDIO6	6		
M5	PL33B	6		C
VCCIO6	VCCIO6	6		
P1	PL35A	6		T (LVDS)*
M6	PL36A	6		T
N2	PL35B	6		C (LVDS)*
N4	PL36B	6		C
R1	PL37A	6		T (LVDS)*
P2	PL37B	6		C (LVDS)*
R2	PL38A	6		T
GND	GNDIO6	6		
T2	PL38B	6		C
VCCIO6	VCCIO6	6		
GND	GNDIO6	6		
N3	PL48A	6		T
R3	PL49A	6	VREF1_6	T
P3	PL49B	6	VREF2_6	C
GND	GNDIO6	6		
GND	GNDIO5	5		
P4	PB3A	5	VREF1_5	T
T3	PB4A	5	LLC_GPLLT_IN_A	T
N5	PB3B	5	VREF2_5	C
T4	PB4B	5	LLC_GPLLC_IN_A	C
VCCIO5	VCCIO5	5		
L9	PB5A	5	LLC_GPLLT_FB_A	T
GND	GNDIO5	5		
L8	PB5B	5	LLC_GPLLC_FB_A	C
VCCIO5	VCCIO5	5		
GND	GNDIO5	5		
P6	PB20A	5		T
N6	PB20B	5		C
VCCIO5	VCCIO5	5		
R4	PB21A	5		T
GND	GNDIO5	5		
T5	PB21B	5		C

XP2-17 Logic Signal Connections: 256 ftBGA (Cont.)

Ball Number	Ball Function	Bank	Dual Function	Differential
N7	PB22A	5		T
P7	PB22B	5		C
R6	PB23A	5		T
M8	PB24A	5	BDQS24	T
T6	PB23B	5		C
N8	PB24B	5		C
VCCIO5	VCCIO5	5		
R7	PB25A	5		T
GND	GNDIO5	5		
T7	PB25B	5		C
P8	PB26A	5		T
N9	PB26B	5		C
R8	PB27A	5		T
T8	PB28A	5	PCLKT5_0	T
T9	PB27B	5		C
R9	PB28B	5	PCLKC5_0	C
VCCIO5	VCCIO5	5		
T10	PB29A	4	PCLKT4_0	T
GND	GNDIO4	4		
R10	PB29B	4	PCLKC4_0	C
M9	PB30A	4		T
L10	PB30B	4		C
T11	PB31A	4		T
P10	PB32A	4		T
R11	PB31B	4		C
N10	PB32B	4		C
VCCIO4	VCCIO4	4		
T12	PB33A	4	BDQS33	T
GND	GNDIO4	4		
P11	PB33B	4		C
L11	PB34A	4		T
M12	PB34B	4		C
T13	PB35A	4		T
M11	PB36A	4		T
R13	PB35B	4		C
N11	PB36B	4		C
VCCIO4	VCCIO4	4		
T14	PB37A	4		T
GND	GNDIO4	4		
R14	PB37B	4		C
R15	PB38A	4		T
P15	PB38B	4		C
VCCIO4	VCCIO4	4		
GND	GNDIO4	4		

XP2-17 Logic Signal Connections: 256 ftBGA (Cont.)

Ball Number	Ball Function	Bank	Dual Function	Differential
R16	PB44A	4	LRC_GPLLT_IN_A	T
P16	PB44B	4	LRC_GPLLC_IN_A	C
VCCIO4	VCCIO4	4		
N12	PB45A	4	LRC_GPLLT_FB_A	T
GND	GNDIO4	4		
P14	PB45B	4	LRC_GPLLC_FB_A	C
P13	PB46A	4	VREF1_4	T
M13	PB46B	4	VREF2_4	C
VCCIO4	VCCIO4	4		
GND	GNDIO3	3		
N14	PR48B	3	VREF2_3	C
N13	PR48A	3	VREF1_3	T
VCCIO3	VCCIO3	3		
GND	GNDIO3	3		
N16	PR37B	3		C (LVDS)*
N15	PR37A	3		T (LVDS)*
M16	PR36B	3		C
L12	PR35B	3		C (LVDS)*
L16	PR36A	3		T
L13	PR35A	3		T (LVDS)*
VCCIO3	VCCIO3	3		
L14	PR33B	3		C
GND	GNDIO3	3		
L15	PR33A	3		T
K13	PR32B	3		C (LVDS)*
K14	PR32A	3		T (LVDS)*
K16	PR31B	3		C
K11	PR30B	3		C (LVDS)*
K15	PR31A	3		T
J11	PR30A	3	RDQS30	T (LVDS)*
VCCIO3	VCCIO3	3		
J12	PR29B	3		C
GND	GNDIO3	3		
J13	PR29A	3		T
J16	PR28B	3		C (LVDS)*
J15	PR28A	3		T (LVDS)*
H13	PR27B	3		C
H16	PR26B	3	PCLKC3_0	C (LVDS)*
J14	PR27A	3		T
H15	PR26A	3	PCLKT3_0	T (LVDS)*
VCCIO3	VCCIO3	3		
G11	TMS	-		
H11	TCK	-		
G13	TDI	-		

XP2-17 Logic Signal Connections: 256 ftBGA (Cont.)

Ball Number	Ball Function	Bank	Dual Function	Differential
H12	TDO	-		
K7	VCCJ	-		
G14	PR24B	2	PCLKC2_0	C
GND	GNDIO2	2		
F12	PR24A	2	PCLKT2_0	T
G16	PR23B	2		C (LVDS)*
G15	PR23A	2		T (LVDS)*
F11	PR22B	2		C
F16	PR21B	2		C (LVDS)*
F13	PR22A	2		T
F15	PR21A	2	RDQS21	T (LVDS)*
VCCIO2	VCCIO2	2		
F14	PR20B	2		C
GND	GNDIO2	2		
E16	PR20A	2		T
E13	PR19B	2		C (LVDS)*
D13	PR19A	2		T (LVDS)*
D16	PR18B	2		C
C16	PR17B	2		C (LVDS)*
D15	PR18A	2		T
C15	PR17A	2		T (LVDS)*
VCCIO2	VCCIO2	2		
F10	PR16B	2		C
GND	GNDIO2	2		
E11	PR16A	2		T
B16	PR15B	2		C (LVDS)*
B15	PR15A	2		T (LVDS)*
D14	PR14B	2		C
C14	PR14A	2		T
VCCIO2	VCCIO2	2		
GND	GNDIO2	2		
A15	PR3B	2		C (LVDS)*
A14	PR3A	2		T (LVDS)*
B14	PR2B	2	VREF2_2	C
C13	PR2A	2	VREF1_2	T
VCCIO2	VCCIO2	2		
VCCIO1	VCCIO1	1		
A13	PT46B	1	VREF2_1	C
B13	PT46A	1	VREF1_1	T
D12	PT45B	1	URC_GPLLC_FB_A	C
GND	GNDIO1	1		
E12	PT45A	1	URC_GPLLT_FB_A	T
VCCIO1	VCCIO1	1		
A12	PT44B	1	URC_GPLLC_IN_A	C

XP2-17 Logic Signal Connections: 256 ftBGA (Cont.)

Ball Number	Ball Function	Bank	Dual Function	Differential
C11	PT44A	1	URC_GPLLT_IN_A	T
GND	GNDIO1	1		
VCCIO1	VCCIO1	1		
D11	PT36B	1		C
B11	PT35B	1		C
C10	PT36A	1		T
A11	PT35A	1		T
D10	PT34B	1		C
D9	PT34A	1		T
A10	PT33B	1		C
GND	GNDIO1	1		
B10	PT33A	1	TDQS33	T
VCCIO1	VCCIO1	1		
F9	PT32B	1		C
E9	PT31B	1		C
C9	PT32A	1		T
D8	PT31A	1		T
A9	PT30B	1		C
B9	PT30A	1		T
A8	PT29B	1	PCLKC1_0	C
GND	GNDIO1	1		
B8	PT29A	1	PCLKT1_0	T
VCCIO0	VCCIO0	0		
F8	PT28B	0	PCLKC0_0	C
A7	PT27B	0		C
E8	PT28A	0	PCLKT0_0	T
B7	PT27A	0		T
F7	PT26B	0		C
D7	PT26A	0		T
A6	PT25B	0		C
GND	GNDIO0	0		
B6	PT25A	0		T
VCCIO0	VCCIO0	0		
C7	PT24B	0		C
A5	PT23B	0		C
D6	PT24A	0	TDQS24	T
C6	PT23A	0		T
F6	PT22B	0		C
E6	PT22A	0		T
A4	PT21B	0		C
GND	GNDIO0	0		
B4	PT21A	0		T
VCCIO0	VCCIO0	0		
G6	PT20B	0		C

XP2-17 Logic Signal Connections: 256 ftBGA (Cont.)

Ball Number	Ball Function	Bank	Dual Function	Differential
A3	PT19B	0		C
F5	PT20A	0		T
C4	PT19A	0		T
D5	PT18B	0		C
E5	PT18A	0		T
GND	GNDIO0	0		
VCCIO0	VCCIO0	0		
D4	PT5B	0	ULC_GPLLC_FB_A	C
GND	GNDIO0	0		
F4	PT5A	0	ULC_GPLLT_FB_A	T
VCCIO0	VCCIO0	0		
A2	PT4B	0	ULC_GPLLC_IN_A	C
B3	PT3B	0	VREF2_0	C
B1	PT4A	0	ULC_GPLLT_IN_A	T
B2	PT3A	0	VREF1_0	T
GND	GNDIO0	0		
A1	GND	-		
A16	GND	-		
B12	GND	-		
B5	GND	-		
C8	GND	-		
E15	GND	-		
E2	GND	-		
H14	GND	-		
H8	GND	-		
H9	GND	-		
J3	GND	-		
J8	GND	-		
J9	GND	-		
M15	GND	-		
M2	GND	-		
P9	GND	-		
R12	GND	-		
R5	GND	-		
T1	GND	-		
T16	GND	-		
G7	VCC	-		
G9	VCC	-		
H7	VCC	-		
J10	VCC	-		
K10	VCC	-		
K8	VCC	-		
G8	VCCAUX	-		
H10	VCCAUX	-		

XP2-17 Logic Signal Connections: 256 ftBGA (Cont.)

Ball Number	Ball Function	Bank	Dual Function	Differential
J7	VCCAUX	-		
K9	VCCAUX	-		
C5	VCCIO0	0		
E7	VCCIO0	0		
C12	VCCIO1	1		
E10	VCCIO1	1		
E14	VCCIO2	2		
G12	VCCIO2	2		
K12	VCCIO3	3		
M14	VCCIO3	3		
M10	VCCIO4	4		
P12	VCCIO4	4		
M7	VCCIO5	5		
P5	VCCIO5	5		
K5	VCCIO6	6		
M3	VCCIO6	6		
E3	VCCIO7	7		
G5	VCCIO7	7		
T15	NC	-		
G10	NC	-		

* Supports true LVDS. Other differential signals must be emulated with external resistors.

XP2-17 Logic Signal Connections: 484 fpBGA

Ball Number	Ball Function	Bank	Dual Function	Differential
VCCIO7	VCCIO7	7		
C3	PL2A	7	VREF1_7	T
B2	PL2B	7	VREF2_7	C
B1	PL3A	7		T (LVDS)*
C2	PL3B	7		C (LVDS)*
C1	PL4A	7		T
GND	GNDIO7	7		
D1	PL4B	7		C
VCCIO7	VCCIO7	7		
E3	PL5A	7		T (LVDS)*
D4	PL6A	7		T
E1	PL5B	7		C (LVDS)*
D3	PL6B	7		C
E4	PL7A	7		T (LVDS)*
E5	PL7B	7		C (LVDS)*
F3	PL8A	7		T
GND	GNDIO7	7		
F2	PL8B	7		C
VCCIO7	VCCIO7	7		
F1	PL9A	7		T (LVDS)*
F5	PL10A	7		T
G1	PL9B	7		C (LVDS)*
F4	PL10B	7		C
H3	PL11A	7		T (LVDS)*
H4	PL11B	7		C (LVDS)*
G3	PL12A	7		T
GND	GNDIO7	7		
G2	PL12B	7		C
VCCIO7	VCCIO7	7		
H1	PL13A	7	LDQS13	T (LVDS)*
G6	PL14A	7		T
H2	PL13B	7		C (LVDS)*
H6	PL14B	7		C
H5	PL15A	7		T (LVDS)*
J5	PL15B	7		C (LVDS)*
J1	PL16A	7		T
GND	GNDIO7	7		
J2	PL16B	7		C
VCCIO7	VCCIO7	7		
K1	PL17A	7		T (LVDS)*
J6	PL18A	7	PROGRAMN	T
K2	PL17B	7		C (LVDS)*
K5	PL18B	7	DONE	C

XP2-17 Logic Signal Connections: 484 fpBGA (Cont.)

Ball Number	Ball Function	Bank	Dual Function	Differential
J3	PL19A	7	CFG1	T (LVDS)*
J4	PL19B	7		C (LVDS)*
L1	PL20A	7	CSSPISN	T
GND	GNDIO7	7		
L2	PL20B	7	CSSPIN	C
VCCIO7	VCCIO7	7		
M1	PL21A	7	LDQS21	T (LVDS)*
K6	PL22A	7	CCLK	T
M2	PL21B	7		C (LVDS)*
K7	PL22B	7	SOSPI	C
L5	PL23A	7	SISPI	T (LVDS)*
L6	PL23B	7	INITN	C (LVDS)*
L3	PL24A	7	PCLKT7_0	T
GND	GNDIO7	7		
L4	PL24B	7	PCLKC7_0	C
L7	TOE	-		
N1	CFG0	-		
VCCIO6	VCCIO6	6		
P1	PL26A	6	PCLKT6_0	T (LVDS)*
M6	PL27A	6		T
R1	PL26B	6	PCLKC6_0	C (LVDS)*
M7	PL27B	6		C
M4	PL28A	6		T (LVDS)*
M5	PL28B	6		C (LVDS)*
M3	PL29A	6		T
GND	GNDIO6	6		
N2	PL29B	6		C
VCCIO6	VCCIO6	6		
T1	PL30A	6	LDQS30	T (LVDS)*
N5	PL31A	6		T
U1	PL30B	6		C (LVDS)*
N6	PL31B	6		C
P2	PL32A	6		T (LVDS)*
P3	PL32B	6		C (LVDS)*
V1	PL33A	6		T
GND	GNDIO6	6		
W1	PL33B	6		C
VCCIO6	VCCIO6	6		
Y1	PL35A	6		T (LVDS)*
N7	PL36A	6		T
AA1	PL35B	6		C (LVDS)*
P7	PL36B	6		C
P4	PL37A	6		T (LVDS)*
P5	PL37B	6		C (LVDS)*

XP2-17 Logic Signal Connections: 484 fpBGA (Cont.)

Ball Number	Ball Function	Bank	Dual Function	Differential
AA2	PL38A	6		T
GND	GNDIO6	6		
Y2	PL38B	6		C
VCCIO6	VCCIO6	6		
R2	PL39A	6	LDQS39	T (LVDS)*
R5	PL40A	6		T
T2	PL39B	6		C (LVDS)*
P6	PL40B	6		C
R3	PL41A	6		T (LVDS)*
R4	PL41B	6		C (LVDS)*
Y3	PL42A	6		T
GND	GNDIO6	6		
W3	PL42B	6		C
VCCIO6	VCCIO6	6		
U2	PL43A	6		T (LVDS)*
R7	PL44A	6		T
V3	PL43B	6		C (LVDS)*
R6	PL44B	6		C
U4	PL45A	6		T (LVDS)*
U5	PL45B	6		C (LVDS)*
T3	PL46A	6		T
GND	GNDIO6	6		
U3	PL46B	6		C
VCCIO6	VCCIO6	6		
AA3	PL47A	6		T (LVDS)*
T6	PL48A	6		T
Y4	PL47B	6		C (LVDS)*
T7	PL48B	6		C
V4	PL49A	6	VREF1_6	T
V5	PL49B	6	VREF2_6	C
GND	GNDIO6	6		
GND	GNDIO5	5		
U7	PB3A	5	VREF1_5	T
W4	PB4A	5	LLC_GPLLT_IN_A	T
U8	PB3B	5	VREF2_5	C
Y5	PB4B	5	LLC_GPLLC_IN_A	C
VCCIO5	VCCIO5	5		
V6	PB5A	5	LLC_GPLLT_FB_A	T
GND	GNDIO5	5		
U6	PB5B	5	LLC_GPLLC_FB_A	C
V8	PB6A	5	BDQS6	T
W8	PB6B	5		C
T8	PB7A	5		T
W5	PB8A	5		T

XP2-17 Logic Signal Connections: 484 fpBGA (Cont.)

Ball Number	Ball Function	Bank	Dual Function	Differential
T9	PB7B	5		C
W6	PB8B	5		C
VCCIO5	VCCIO5	5		
Y6	PB9A	5		T
GND	GNDIO5	5		
Y7	PB9B	5		C
Y9	PB10A	5		T
Y8	PB10B	5		C
U9	PB11A	5		T
V9	PB12A	5		T
U10	PB11B	5		C
W9	PB12B	5		C
VCCIO5	VCCIO5	5		
AA6	PB13A	5		T
GND	GNDIO5	5		
AA7	PB13B	5		C
T10	PB14A	5		T
V10	PB14B	5		C
V11	PB15A	5	BDQS15	T
AA8	PB16A	5		T
W11	PB15B	5		C
AA9	PB16B	5		C
VCCIO5	VCCIO5	5		
AB2	PB17A	5		T
GND	GNDIO5	5		
AB3	PB17B	5		C
AA11	PB18A	5		T
AA10	PB18B	5		C
T11	PB19A	5		T
AB4	PB20A	5		T
U11	PB19B	5		C
AB5	PB20B	5		C
VCCIO5	VCCIO5	5		
AB6	PB21A	5		T
GND	GNDIO5	5		
AB7	PB21B	5		C
AA12	PB22A	5		T
Y11	PB22B	5		C
T12	PB23A	5		T
AB8	PB24A	5	BDQS24	T
T13	PB23B	5		C
AB9	PB24B	5		C
VCCIO5	VCCIO5	5		
AB11	PB25A	5		T

XP2-17 Logic Signal Connections: 484 fpBGA (Cont.)

Ball Number	Ball Function	Bank	Dual Function	Differential
GND	GNDIO5	5		
AB10	PB25B	5		C
AB13	PB26A	5		T
AB12	PB26B	5		C
Y12	PB27A	5		T
AB14	PB28A	5	PCLKT5_0	T
W12	PB27B	5		C
AB15	PB28B	5	PCLKC5_0	C
VCCIO5	VCCIO5	5		
AB16	PB29A	4	PCLKT4_0	T
GND	GNDIO4	4		
AB17	PB29B	4	PCLKC4_0	C
Y14	PB30A	4		T
AA13	PB30B	4		C
V12	PB31A	4		T
AB18	PB32A	4		T
U12	PB31B	4		C
AB19	PB32B	4		C
VCCIO4	VCCIO4	4		
AA14	PB33A	4	BDQS33	T
GND	GNDIO4	4		
AA15	PB33B	4		C
T14	PB34A	4		T
T15	PB34B	4		C
V13	PB35A	4		T
AA16	PB36A	4		T
U13	PB35B	4		C
AA17	PB36B	4		C
VCCIO4	VCCIO4	4		
AB20	PB37A	4		T
GND	GNDIO4	4		
AA20	PB37B	4		C
W14	PB38A	4		T
V14	PB38B	4		C
U14	PB39A	4		T
AA22	PB40A	4		T
U15	PB39B	4		C
AA21	PB40B	4		C
VCCIO4	VCCIO4	4		
Y22	PB41A	4		T
GND	GNDIO4	4		
Y21	PB41B	4		C
W15	PB42A	4	BDQS42	T
V15	PB42B	4		C

XP2-17 Logic Signal Connections: 484 fpBGA (Cont.)

Ball Number	Ball Function	Bank	Dual Function	Differential
T16	PB43A	4		T
Y16	PB44A	4	LRC_GPLLT_IN_A	T
U16	PB43B	4		C
Y15	PB44B	4	LRC_GPLLC_IN_A	C
VCCIO4	VCCIO4	4		
Y17	PB45A	4	LRC_GPLLT_FB_A	T
GND	GNDIO4	4		
Y18	PB45B	4	LRC_GPLLC_FB_A	C
W17	PB46A	4	VREF1_4	T
W18	PB46B	4	VREF2_4	C
VCCIO4	VCCIO4	4		
GND	GNDIO3	3		
V18	PR48B	3	VREF2_3	C
V17	PR48A	3	VREF1_3	T
V19	PR47B	3		C (LVDS)*
W19	PR47A	3		T (LVDS)*
VCCIO3	VCCIO3	3		
W20	PR46B	3		C
GND	GNDIO3	3		
W22	PR46A	3		T
Y19	PR45B	3		C (LVDS)*
Y20	PR45A	3		T (LVDS)*
T17	PR44B	3		C
V20	PR43B	3		C (LVDS)*
R16	PR44A	3		T
U20	PR43A	3		T (LVDS)*
VCCIO3	VCCIO3	3		
V22	PR42B	3		C
GND	GNDIO3	3		
U21	PR42A	3		T
R17	PR41B	3		C (LVDS)*
R18	PR41A	3		T (LVDS)*
P17	PR40B	3		C
U22	PR39B	3		C (LVDS)*
P16	PR40A	3		T
T22	PR39A	3	RDQS39	T (LVDS)*
VCCIO3	VCCIO3	3		
T20	PR38B	3		C
GND	GNDIO3	3		
T21	PR38A	3		T
R19	PR37B	3		C (LVDS)*
P19	PR37A	3		T (LVDS)*
N16	PR36B	3		C
R20	PR35B	3		C (LVDS)*

XP2-17 Logic Signal Connections: 484 fpBGA (Cont.)

Ball Number	Ball Function	Bank	Dual Function	Differential
N17	PR36A	3		T
R21	PR35A	3		T (LVDS)*
VCCIO3	VCCIO3	3		
R22	PR33B	3		C
GND	GNDIO3	3		
P22	PR33A	3		T
P20	PR32B	3		C (LVDS)*
P21	PR32A	3		T (LVDS)*
N18	PR31B	3		C
N22	PR30B	3		C (LVDS)*
P18	PR31A	3		T
M22	PR30A	3	RDQS30	T (LVDS)*
VCCIO3	VCCIO3	3		
N21	PR29B	3		C
GND	GNDIO3	3		
M21	PR29A	3		T
M16	PR28B	3		C (LVDS)*
M17	PR28A	3		T (LVDS)*
M20	PR27B	3		C
L21	PR26B	3	PCLKC3_0	C (LVDS)*
M19	PR27A	3		T
K21	PR26A	3	PCLKT3_0	T (LVDS)*
VCCIO3	VCCIO3	3		
L22	TMS	-		
L16	TCK	-		
L20	TDI	-		
M18	TDO	-		
H15	VCCJ	-		
K22	PR24B	2	PCLKC2_0	C
GND	GNDIO2	2		
J22	PR24A	2	PCLKT2_0	T
L18	PR23B	2		C (LVDS)*
L19	PR23A	2		T (LVDS)*
K16	PR22B	2		C
J21	PR21B	2		C (LVDS)*
J16	PR22A	2		T
H21	PR21A	2	RDQS21	T (LVDS)*
VCCIO2	VCCIO2	2		
H22	PR20B	2		C
GND	GNDIO2	2		
G22	PR20A	2		T
L17	PR19B	2		C (LVDS)*
K18	PR19A	2		T (LVDS)*
K17	PR18B	2		C

XP2-17 Logic Signal Connections: 484 fpBGA (Cont.)

Ball Number	Ball Function	Bank	Dual Function	Differential
F22	PR17B	2		C (LVDS)*
J17	PR18A	2		T
G21	PR17A	2		T (LVDS)*
VCCIO2	VCCIO2	2		
F21	PR16B	2		C
GND	GNDIO2	2		
F20	PR16A	2		T
J18	PR15B	2		C (LVDS)*
J19	PR15A	2		T (LVDS)*
H17	PR14B	2		C
E22	PR13B	2		C (LVDS)*
H16	PR14A	2		T
D22	PR13A	2	RDQS13	T (LVDS)*
VCCIO2	VCCIO2	2		
C22	PR12B	2		C
GND	GNDIO2	2		
B22	PR12A	2		T
H19	PR11B	2		C (LVDS)*
J20	PR11A	2		T (LVDS)*
F19	PR10B	2		C
G20	PR9B	2		C (LVDS)*
E19	PR10A	2		T
H20	PR9A	2		T (LVDS)*
VCCIO2	VCCIO2	2		
C21	PR8B	2		C
GND	GNDIO2	2		
B21	PR8A	2		T
H18	PR7B	2		C (LVDS)*
G17	PR7A	2		T (LVDS)*
G16	PR6B	2		C
D20	PR5B	2		C (LVDS)*
G15	PR6A	2		T
E20	PR5A	2		T (LVDS)*
VCCIO2	VCCIO2	2		
C20	PR4B	2		C
GND	GNDIO2	2		
B20	PR4A	2		T
F17	PR3B	2		C (LVDS)*
F18	PR3A	2		T (LVDS)*
F16	PR2B	2	VREF2_2	C
F15	PR2A	2	VREF1_2	T
VCCIO2	VCCIO2	2		
VCCIO1	VCCIO1	1		
E18	PT46B	1	VREF2_1	C

XP2-17 Logic Signal Connections: 484 fpBGA (Cont.)

Ball Number	Ball Function	Bank	Dual Function	Differential
E17	PT46A	1	VREF1_1	T
C19	PT45B	1	URC_GPLLC_FB_A	C
GND	GNDIO1	1		
C18	PT45A	1	URC_GPLLT_FB_A	T
VCCIO1	VCCIO1	1		
D19	PT44B	1	URC_GPLLC_IN_A	C
D18	PT43B	1		C
A20	PT44A	1	URC_GPLLT_IN_A	T
D17	PT43A	1		T
A19	PT42B	1		C
A21	PT42A	1	TDQS42	T
D15	PT41B	1		C
GND	GNDIO1	1		
C15	PT41A	1		T
VCCIO1	VCCIO1	1		
E15	PT40B	1		C
G14	PT39B	1		C
E14	PT40A	1		T
G13	PT39A	1		T
D14	PT38B	1		C
F14	PT38A	1		T
C17	PT37B	1		C
GND	GNDIO1	1		
C16	PT37A	1		T
VCCIO1	VCCIO1	1		
A18	PT36B	1		C
G12	PT35B	1		C
B17	PT36A	1		T
F13	PT35A	1		T
C14	PT34B	1		C
E13	PT34A	1		T
A17	PT33B	1		C
GND	GNDIO1	1		
B16	PT33A	1	TDQS33	T
VCCIO1	VCCIO1	1		
A16	PT32B	1		C
G11	PT31B	1		C
B15	PT32A	1		T
G10	PT31A	1		T
D12	PT30B	1		C
E12	PT30A	1		T
A15	PT29B	1	PCLKC1_0	C
GND	GNDIO1	1		
B14	PT29A	1	PCLKT1_0	T

XP2-17 Logic Signal Connections: 484 fpBGA (Cont.)

Ball Number	Ball Function	Bank	Dual Function	Differential
VCCIO0	VCCIO0	0		
A14	PT28B	0	PCLKC0_0	C
F12	PT27B	0		C
B13	PT28A	0	PCLKT0_0	T
F11	PT27A	0		T
C12	PT26B	0		C
E11	PT26A	0		T
A13	PT25B	0		C
GND	GNDIO0	0		
B12	PT25A	0		T
VCCIO0	VCCIO0	0		
A12	PT24B	0		C
G9	PT23B	0		C
B11	PT24A	0	TDQS24	T
G8	PT23A	0		T
C11	PT22B	0		C
D11	PT22A	0		T
A11	PT21B	0		C
GND	GNDIO0	0		
A10	PT21A	0		T
VCCIO0	VCCIO0	0		
B10	PT20B	0		C
E10	PT19B	0		C
B9	PT20A	0		T
F10	PT19A	0		T
C9	PT18B	0		C
D9	PT18A	0		T
A9	PT17B	0		C
GND	GNDIO0	0		
A8	PT17A	0		T
VCCIO0	VCCIO0	0		
B8	PT16B	0		C
F9	PT15B	0		C
A7	PT16A	0		T
E9	PT15A	0	TDQS15	T
C8	PT14B	0		C
D8	PT14A	0		T
B7	PT13B	0		C
GND	GNDIO0	0		
B6	PT13A	0		T
VCCIO0	VCCIO0	0		
A6	PT12B	0		C
J7	PT11B	0		C
A5	PT12A	0		T

XP2-17 Logic Signal Connections: 484 fpBGA (Cont.)

Ball Number	Ball Function	Bank	Dual Function	Differential
H7	PT11A	0		T
C7	PT10B	0		C
C6	PT10A	0		T
A4	PT9B	0		C
GND	GNDIO0	0		
A3	PT9A	0		T
VCCIO0	VCCIO0	0		
C4	PT8B	0		C
E8	PT7B	0		C
C5	PT8A	0		T
F8	PT7A	0		T
D5	PT6B	0		C
D6	PT6A	0	TDQS6	T
G7	PT5B	0	ULC_GPLLC_FB_A	C
GND	GNDIO0	0		
F7	PT5A	0	ULC_GPLLT_FB_A	T
VCCIO0	VCCIO0	0		
B3	PT4B	0	ULC_GPLLC_IN_A	C
E6	PT3B	0	VREF2_0	C
A2	PT4A	0	ULC_GPLLT_IN_A	T
F6	PT3A	0	VREF1_0	T
GND	GNDIO0	0		
A1	GND	-		
A22	GND	-		
AA19	GND	-		
AA4	GND	-		
AB1	GND	-		
AB22	GND	-		
B19	GND	-		
B4	GND	-		
C10	GND	-		
C13	GND	-		
D16	GND	-		
D2	GND	-		
D21	GND	-		
D7	GND	-		
G19	GND	-		
G4	GND	-		
H10	GND	-		
H13	GND	-		
J14	GND	-		
J9	GND	-		
K10	GND	-		
K11	GND	-		

XP2-17 Logic Signal Connections: 484 fpBGA (Cont.)

Ball Number	Ball Function	Bank	Dual Function	Differential
K12	GND	-		
K13	GND	-		
K15	GND	-		
K20	GND	-		
K3	GND	-		
K8	GND	-		
L10	GND	-		
L11	GND	-		
L12	GND	-		
L13	GND	-		
M10	GND	-		
M11	GND	-		
M12	GND	-		
M13	GND	-		
N10	GND	-		
N11	GND	-		
N12	GND	-		
N13	GND	-		
N15	GND	-		
N20	GND	-		
N3	GND	-		
N8	GND	-		
P14	GND	-		
P9	GND	-		
R10	GND	-		
R13	GND	-		
T19	GND	-		
T4	GND	-		
W16	GND	-		
W2	GND	-		
W21	GND	-		
W7	GND	-		
Y10	GND	-		
Y13	GND	-		
N9	VCC	-		
P10	VCC	-		
J10	VCC	-		
J11	VCC	-		
J12	VCC	-		
P11	VCC	-		
P12	VCC	-		
J13	VCC	-		
K14	VCC	-		
P13	VCC	-		

XP2-17 Logic Signal Connections: 484 fpBGA (Cont.)

Ball Number	Ball Function	Bank	Dual Function	Differential
K9	VCC	-		
L14	VCC	-		
L9	VCC	-		
M14	VCC	-		
M9	VCC	-		
N14	VCC	-		
H11	VCCAUX	-		
H12	VCCAUX	-		
L15	VCCAUX	-		
L8	VCCAUX	-		
M15	VCCAUX	-		
M8	VCCAUX	-		
R11	VCCAUX	-		
R12	VCCAUX	-		
B5	VCCIO0	0		
D10	VCCIO0	0		
E7	VCCIO0	0		
H9	VCCIO0	0		
B18	VCCIO1	1		
D13	VCCIO1	1		
E16	VCCIO1	1		
H14	VCCIO1	1		
E21	VCCIO2	2		
G18	VCCIO2	2		
J15	VCCIO2	2		
K19	VCCIO2	2		
N19	VCCIO3	3		
P15	VCCIO3	3		
T18	VCCIO3	3		
V21	VCCIO3	3		
AA18	VCCIO4	4		
R14	VCCIO4	4		
V16	VCCIO4	4		
W13	VCCIO4	4		
AA5	VCCIO5	5		
R9	VCCIO5	5		
V7	VCCIO5	5		
W10	VCCIO5	5		
N4	VCCIO6	6		
P8	VCCIO6	6		
T5	VCCIO6	6		
V2	VCCIO6	6		
E2	VCCIO7	7		
G5	VCCIO7	7		

XP2-17 Logic Signal Connections: 484 fpBGA (Cont.)

Ball Number	Ball Function	Bank	Dual Function	Differential
J8	VCCIO7	7		
K4	VCCIO7	7		
AB21	NC	-		
H8	NC	-		
R15	NC	-		
R8	NC	-		
U19	NC	-		
U17	NC	-		
U18	NC	-		

* Supports true LVDS. Other differential signals must be emulated with external resistors.

Thermal Management

Thermal management is recommended as part of any sound FPGA design methodology. To assess the thermal characteristics of a system, Lattice specifies a maximum allowable junction temperature in all device data sheets. Designers must complete a thermal analysis of their specific design to ensure that the device and package do not exceed the junction temperature limits. Refer to the Thermal Management document to find the device/package specific thermal values.

For Further Information

For further information regarding Thermal Management, refer to the following located on the Lattice website at www.latticesemi.com.

- Thermal Management document
- Technical Note TN1139 - Power Estimation and Management for LatticeXP2 Devices
- Power Calculator tool included with Lattice's ispLEVER design tool, or as a standalone download from www.latticesemi.com/software

LatticeXP2 Family Data Sheet Ordering Information

May 2007 Advance Data Sheet DS1009

Part Number Description

LFXP2 – XX E – X XXXXX X

Device Family
XP2

Logic Capacity
5 = 5K LUTs
8 = 8K LUTs
17 = 17K LUTs
30 = 30K LUTs
40 = 40K LUTs

Supply Voltage
E = 1.2V

Speed
5 = Slowest
6
7 = Fastest

Grade
C = Commercial
I = Industrial

Package
M132 = 132-ball csBGA
T144 = 144-pin TQFP
Q208 = 208-pin PQFP
FT256 = 256-ball ftBGA
F484 = 484-ball fpBGA
F672 = 672-ball fpBGA

MN132 = 132-ball Lead-Free csBGA
TN144 = 144-pin Lead-Free TQFP
QN208 = 208-pin Lead-Free PQFP
FTN256 = 256-ball Lead-Free ftBGA
FN484 = 484-ball Lead-Free fpBGA
FN672 = 672-ball Lead-Free fpBGA

Ordering Information

Note: LatticeXP2 devices are dual marked. For example, the commercial speed grade LFXP2-17E-7256C is also marked with industrial grade -6I (LFXP2-17E-6256I). The commercial grade is one speed grade faster than the associated dual mark industrial grade. The slowest commercial speed grade does not have industrial markings. The markings appear as follows:

Conventional Packaging

Commercial

Part Number	Voltage	Grade	Package	Pins	Temp.	LUTs (k)
LFXP2-17E-5Q208C	1.2V	-5	PQFP	208	COM	17
LFXP2-17E-6Q208C	1.2V	-6	PQFP	208	COM	17
LFXP2-17E-7Q208C	1.2V	-7	PQFP	208	COM	17
LFXP2-17E-5FT256C	1.2V	-5	ftBGA	256	COM	17
LFXP2-17E-6FT256C	1.2V	-6	ftBGA	256	COM	17
LFXP2-17E-7FT256C	1.2V	-7	ftBGA	256	COM	17
LFXP2-17E-5F484C	1.2V	-5	fpBGA	484	COM	17
LFXP2-17E-6F484C	1.2V	-6	fpBGA	484	COM	17
LFXP2-17E-7F484C	1.2V	-7	fpBGA	484	COM	17

Industrial

Part Number	Voltage	Grade	Package	Pins	Temp.	LUTs (k)
LFXP2-17E-5Q208I	1.2V	-5	PQFP	208	IND	17
LFXP2-17E-6Q208I	1.2V	-6	PQFP	208	IND	17
LFXP2-17E-5FT256I	1.2V	-5	ftBGA	256	IND	17
LFXP2-17E-6FT256I	1.2V	-6	ftBGA	256	IND	17
LFXP2-17E-5F484I	1.2V	-5	fpBGA	484	IND	17
LFXP2-17E-6F484I	1.2V	-6	fpBGA	484	IND	17

Lead-Free Packaging

Commercial

Part Number	Voltage	Grade	Package	Pins	Temp.	LUTs (k)
LFXP2-17E-5QN208C	1.2V	-5	Lead-Free PQFP	208	COM	17
LFXP2-17E-6QN208C	1.2V	-6	Lead-Free PQFP	208	COM	17
LFXP2-17E-7QN208C	1.2V	-7	Lead-Free PQFP	208	COM	17
LFXP2-17E-5FTN256C	1.2V	-5	Lead-Free ftBGA	256	COM	17
LFXP2-17E-6FTN256C	1.2V	-6	Lead-Free ftBGA	256	COM	17
LFXP2-17E-7FTN256C	1.2V	-7	Lead-Free ftBGA	256	COM	17
LFXP2-17E-5FN484C	1.2V	-5	Lead-Free fpBGA	484	COM	17
LFXP2-17E-6FN484C	1.2V	-6	Lead-Free fpBGA	484	COM	17
LFXP2-17E-7FN484C	1.2V	-7	Lead-Free fpBGA	484	COM	17

Industrial

Part Number	Voltage	Grade	Package	Pins	Temp.	LUTs (k)
LFXP2-17E-5QN208I	1.2V	-5	Lead-Free PQFP	208	IND	17
LFXP2-17E-6QN208I	1.2V	-6	Lead-Free PQFP	208	IND	17
LFXP2-17E-5FTN256I	1.2V	-5	Lead-Free ftBGA	256	IND	17
LFXP2-17E-6FTN256I	1.2V	-6	Lead-Free ftBGA	256	IND	17
LFXP2-17E-5FN484I	1.2V	-5	Lead-Free fpBGA	484	IND	17
LFXP2-17E-6FN484I	1.2V	-6	Lead-Free fpBGA	484	IND	17

LatticeXP2 Family Data Sheet Supplemental Information

May 2007 Advance Data Sheet DS1009

For Further Information

A variety of technical notes for the LatticeXP2 family are available on the Lattice web site at www.latticesemi.com.

- LatticeXP2 sysIO Usage Guide (TN1136)
- LatticeXP2 Memory Usage Guide (TN1137)
- LatticeXP2 High Speed I/O Interface (TN1138)
- LatticeXP2 sysCLOCK PLL Design and Usage Guide (TN1126)
- Power Estimation and Management for LatticeXP2 Devices (TN1139)
- LatticeXP2 sysDSP Usage Guide (TN1140)
- LatticeXP2 sysCONFIG Usage Guide (TN1141)
- LatticeXP2 Configuration Encryption and Security Usage Guide (TN1142)
- Minimizing System Interruption During Configuration Using TransFR Technology (TN1087)
- LatticeXP2 Dual Boot Usage Guide (TN1144)
- LatticeXP2 Soft Error Detection (SED) Usage Guide (TN1130)

For further information on interface standards refer to the following web sites:

- JEDEC Standards (LVTTL, LVCMOS, SSTL, HSTL): www.jedec.org
- PCI: www.pcisig.com

LatticeXP2 Family Data Sheet Revision History

Revision History

Date	Version	Section	Change Summary
May 2007	01.1	—	Initial release.
September 2007	01.2	DC and Switching Characteristics	Added JTAG Port Timing Waveforms diagram.
			Updated sysCLOCK PLL Timing table.
		Pinout Information	Added Thermal Management text section.

Section II. LatticeXP2 Family Technical Notes

LatticeXP2 sysIO Usage Guide

April 2007 Technical Note TN1136

Introduction

The LatticeXP2™ sysIO™ buffers give the designer the ability to easily interface with other devices using advanced system I/O standards. This technical note describes the sysIO standards available and how they can be implemented using Lattice's ispLEVER® design software.

sysIO Buffer Overview

The LatticeXP2 sysIO interface contains multiple Programmable I/O Cells (PIC) blocks. Each PIC contains two Programmable I/Os (PIO), PIOA and PIOB, connected to their respective sysIO Buffers. Two adjacent PIOs can be joined to provide a differential I/O pair (labeled as "T" and "C").

Each Programmable I/O (PIO) includes a sysIO Buffer and I/O Logic (IOLOGIC). The LatticeXP2 sysIO buffers supports a variety of single-ended and differential signaling standards. The sysIO buffer also supports the DQS strobe signal that is required for interfacing with the DDR memory. One of every 16/18 PIOs in the LatticeXP2 contains a delay element to facilitate the generation of DQS signals. The DQS signal from the bus is used to strobe the DDR data from the memory into input register blocks. For more information on the architecture of the sysIO buffer please refer to the LatticeXP2 Family Data Sheet.

The IOLOGIC includes input, output and tristate registers that implement both single data rate (SDR) and double data rate (DDR) applications along with the necessary clock and data selection logic. Programmable delay lines and dedicated logic within the IOLOGIC are used to provide the required shift to incoming clock and data signals and the delay required by DQS inputs in DDR memory. The DDR implementation in the IOLOGIC and the DDR memory interface support are discussed in more detail in Lattice technical note TN1138, *LatticeXP2 High Speed I/O Interface.*

Supported sysIO Standards

The LatticeXP2 sysIO buffer supports both single-ended and differential standards. Single-ended standards can be further subdivided into internally ratioed standard such as LVCMOS, LVTTL and PCI; and externally referenced standards such as HSTL and SSTL. The buffers support the LVTTL, LVCMOS 1.2, 1.5, 1.8, 2.5 and 3.3V standards. In the LVCMOS and LVTTL modes, the buffer has individually configurable options for drive strength, bus maintenance (weak pull-up, weak pull-down, or a bus-keeper latch). Other single-ended standards supported include SSTL and HSTL. Differential standards supported include MLVDS, LVDS, RSDS, BLVDS, LVPECL, differential SSTL and differential HSTL. Tables 1 and 2 list the sysIO standards supported in LatticeXP2 devices.

Table 8-1. Supported Input Standards

Input Standard	V_{REF} (Nom.)	V_{CCIO}[1] (Nom.)
Single Ended Interfaces		
LVTTL	—	—
LVCMOS33	—	—
LVCMOS25	—	—
LVCMOS18	—	1.8
LVCMOS15	—	1.5
LVCMOS12	—	—
PCI 33	—	3.3
HSTL18 Class I, II	0.9	—
HSTL15 Class I	0.75	—
SSTL3 Class I, II	1.5	—

Table 8-1. Supported Input Standards (Continued)

Input Standard	V_{REF} (Nom.)	V_{CCIO}[1] (Nom.)
SSTL2 Class I, II	1.25	—
SSTL18 Class I, II	0.9	—
Differential Interfaces		
Differential SSTL18 Class I, II	—	—
Differential SSTL2 Class I, II	—	—
Differential SSTL3 Class I, II	—	—
Differential HSTL15 Class I	—	—
Differential HSTL18 Class I, II	—	—
LVDS, MLVDS, LVPECL, BLVDS, RSDS	—	—

1 When not specified, V_{CCIO} can be set anywhere in the valid operating range.

Table 8-2. Supported Output Standards

Output Standard	Drive	V_{CCIO} (Nom.)
Single-ended Interfaces		
LVTTL	4mA, 8mA, 12mA, 16mA, 20mA	3.3
LVTTL, Open Drain	4mA, 8mA, 12mA, 16mA, 20mA	—
LVCMOS33	4mA, 8mA, 12mA 16mA, 20mA	3.3
LVCMOS25	4mA, 8mA, 12mA, 16mA, 20mA	2.5
LVCMOS18	4mA, 8mA, 12mA, 16mA	1.8
LVCMOS15	4mA, 8mA	1.5
LVCMOS12	2mA, 6mA	1.2
LVCMOS33, Open Drain	4mA, 8mA, 12mA 16mA, 20mA	—
LVCMOS25, Open Drain	4mA, 8mA, 12mA 16mA, 20mA	—
LVCMOS18, Open Drain	4mA, 8mA, 12mA 16mA	—
LVCMOS15, Open Drain	4mA, 8mA	—
LVCMOS12, Open Drain	2mA, 6mA	—
PCI33[2]	N/A	3.3
HSTL18 Class I	8mA, 12mA	1.8
HSTL18 Class II	N/A	1.8
HSTL15 Class I	4mA, 8mA	1.5
SSTL3 Class I, II	N/A	3.3
SSTL2 Class I	8mA, 12mA	2.5
SSTL2 Class II	16mA, 20mA	2.5
SSTL18 Class I	N/A	1.8
SSTL18 Class II	8mA, 12mA	1.8
Differential Interfaces		
Differential SSTL3, Class I, II	N/A	3.3
Differential SSTL2, Class I	8mA, 12mA	2.5
Differential SSTL2, Class II	16mA, 20mA	2.5
Differential SSTL18, Class I	N/A	1.8
Differential SSTL18, Class II	8mA, 12mA	1.8
Differential HSTL18, Class I	8mA, 12mA	1.8
Differential HSTL18, Class II	N/A	1.8
Differential HSTL15, Class I	4mA, 8mA	1.5

Table 8-2. Supported Output Standards (Continued)

Output Standard	Drive	V_{CCIO} (Nom.)
LVDS	N/A	2.5
MLVDS[1]	N/A	2.5
BLVDS[1]	N/A	2.5
LVPECL[1]	N/A	3.3
RSDS[1]	N/A	2.5

1. Emulated with external resistors.
2. PCI33 is PCIX compatible.

sysIO Banking Scheme

LatticeXP2 devices have eight general purpose programmable sysIO banks. Each of the eight general purpose sysIO banks has a V_{CCIO} supply voltage, and two reference voltages, V_{REF1} and V_{REF2}. Figure 8-1 shows the eight general purpose banks.

On the top and bottom banks, the sysIO buffer pair consists of two single-ended output drivers and two sets of single-ended input buffers (both ratioed and referenced). The left and right sysIO buffer pair consists of two single-ended output drivers and two sets of single-ended input buffers (both ratioed and referenced). The referenced input buffer can also be configured as a differential input. True LVDS support is available on only 50% of the left and right I/Os (starting with the topmost pairs). There are no LVDS on the top and bottom I/Os. In 50% of the pairs there is also one differential output driver. The two pads in the pair are described as “true” and “comp”, where the true pad is associated with the positive side of the differential input buffer and the comp (complementary) pad is associated with the negative side of the differential input buffer.

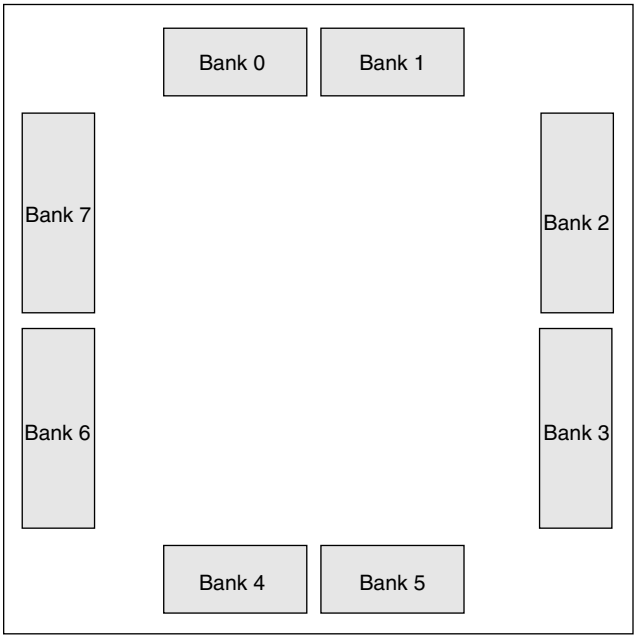
SPI Flash Interface

The SPI pins (master/slave) are multiplexed with the I/Os in Bank 7. The two dedicated pins CFG[0] and TOE are powered by V_{CC} and reside between Banks 6 and 7.

JTAG Interface

The JTAG pins are located between Banks 2 and 3 and are powered by V_{CCJ}.

Figure 8-1. LatticeXP2 sysIO Banking



V_{CCIO} (1.2V/1.5V/1.8V/2.5V/3.3V)

There are a total of eight V_{CCIO} supplies, V_{CCIO0} - V_{CCIO7}. Each bank has a separate V_{CCIO} supply that powers the single-ended output drivers and the ratioed input buffers such as LVTTL, LVCMOS, and PCI. LVTTL, LVCMOS3.3, LVCMOS2.5 and LVCMOS1.2 also have fixed threshold options allowing them to be placed in any bank. The V_{CCIO} voltage applied to the bank determines the ratioed input standards that can be supported in that bank. It is also used to power the differential output drivers.

V_{CCAUX} (3.3V)

In addition to the bank V_{CCIO} supplies, devices have a V_{CC} core logic power supply and a V_{CCAUX} auxiliary supply that powers the differential and referenced input buffers. V_{CCAUX} is used to supply I/O reference voltage requiring 3.3V to satisfy the common-mode range of the drivers and input buffers.

V_{CCJ} (1.2V/1.5V/1.8V/2.5V/3.3V)

The JTAG pins have a separate V_{CCJ} power supply that is independent of the bank V_{CCIO} supplies. V_{CCJ} determines the electrical characteristics of the LVCMOS JTAG pins, both the output high level and the input threshold.

Table 8-3 shows a summary of all the required power supplies.

Table 8-3. Power Supplies

Power Supply	Description	Value[1]
V_{CC}	Core Power Supply	1.2V
V_{CCIO}[2]	Power Supply for the I/O Banks	1.2V/1.5V/1.8V/2.5V/3.3V
V_{CCAUX}	Auxiliary Power Supply	3.3V
V_{CCJ}[2]	Power Supply for JTAG Pins	1.2V/1.5V/1.8V/2.5V/3.3V

1. Refer to LatticeXP2 Family Data Sheet for recommended min. and max. values.
2. If V_{CCIO} or V_{CCJ} is set to 3.3V, they MUST be connected to the same power supply as VCCAUX.

Input Reference Voltage (V_{REF1}, V_{REF2})

Each bank can support up to two separate V_{REF} input voltages, V_{REF1} and V_{REF2}, that are used to set the threshold for the referenced input buffers. The locations of these V_{REF} pins are pre-determined within the bank. These pins can be used as regular I/Os if the bank does not require a V_{REF} voltage.

V_{REF1} for DDR Memory Interface

When interfacing to DDR memory, the V_{REF1} input must be used as the reference voltage for the DQS and DQ input from the memory. A voltage divider between V_{REF1} and GND is used to generate an on-chip reference voltage that is used by the DQS transition detector circuit. This voltage divider is only present on V_{REF1} it is not available on V_{REF2}. For more information on the DQS transition detect logic and its implementation, please refer to Lattice technical note number TN1138, *LatticeXP2 High Speed I/O Interface*. For DDR1 memory interfaces, the V_{REF1} should be connected to 1.25V. Therefore, only SSTL25_II signaling is allowed. For DDR2 memory interfaces this should be connected to 0.9V, and only SSTL18_II signaling is allowed.

Mixed Voltage Support in a Bank

The LatticeXP2 sysIO buffer is connected to three parallel ratioed input buffers. These three parallel buffers are connected to V_{CCIO}, V_{CCAUX} and V_{CC}, giving support for thresholds that track with V_{CCIO} as well as fixed thresholds for 3.3V (V_{CCAUX}) and 1.2V (V_{CC}) inputs. This allows the input threshold for ratioed buffers to be assigned on a pin-by-pin basis rather than tracking with V_{CCIO}. This option is available for all 1.2V, 2.5V and 3.3V ratioed inputs and is independent of the bank V_{CCIO} voltage. For example, if the bank V_{CCIO} is 1.8V, it is possible to have 1.2V and 3.3V ratioed input buffers with fixed thresholds, as well as 2.5V ratioed inputs with tracking thresholds.

Prior to device configuration, the ratioed input thresholds always tracks the bank V_{CCIO}. This option only takes effect after configuration. Output standards within a bank are always set by V_{CCIO}. Table 8-4 shows the sysIO standards that can be mixed in the same bank.

Table 8-4. Mixed Voltage Support

	Input sysIO Standards					Output sysIO Standards				
V_{CCIO}	1.2V	1.5V	1.8V	2.5V	3.3V	1.2V	1.5V	1.8V	2.5V	3.3V
1.2V	Yes			Yes	Yes	Yes				
1.5V	Yes	Yes		Yes	Yes		Yes			
1.8V	Yes		Yes	Yes	Yes			Yes		
2.5V	Yes			Yes	Yes				Yes	
3.3V	Yes			Yes	Yes					Yes

sysIO Standards Supported by Bank

Table 8-5. I/O Standards Supported by Bank

Description	Top Side Banks 0-1	Right Side Banks 2-3	Bottom Side Banks 4-5	Left Side Banks 6-7
I/O Buffers	Single-ended	Single-ended and Differential	Single-ended	Single-ended and Differential
Output Standards Supported	LVTTL LVCMOS33 LVCMOS25 LVCMOS18 LVCMOS15 LVCMOS12 SSTL18 Class I, II SSTL25 Class I, II SSTL33 Class I, II HSTL15 Class I HSTL18_I, II SSTL18D Class I, II SSTL25D Class I, II SSTL33D Class I, II HSTL15D Class I HSTL18D Class I, II MLVDS LVDS25E[1] LVPECL[1] BLVDS[1] RSDS[1]	LVTTL LVCMOS33 LVCMOS25 LVCMOS18 LVCMOS15 LVCMOS12 SSTL18 Class I, II SSTL25 Class I, II SSTL33 Class I, II HSTL15 Class I HSTL18 Class I, II SSTL18D Class I, II SSTL25D Class I, II SSTL33D Class I, II HSTL15D Class I, II HSTL18D Class I, II MLVDS LVDS LVDS25E[1] LVPECL[1] BLVDS[1] RSDS[1]	LVTTL LVCMOS33 LVCMOS25 LVCMOS18 LVCMOS15 LVCMOS12 SSTL18 Class I SSTL2 Class I, II SSTL3 Class I, II HSTL15 Class I HSTL18 Class I, II SSTL18D Class I, II SSTL25D Class I, II, SSTL33D Class I, II HSTL15D Class I HSTL18D Class I, II MLVDS LVDS25E[1] LVPECL[1] BLVDS[1] RSDS[1]	LVTTL LVCMOS33 LVCMOS25 LVCMOS18 LVCMOS15 LVCMOS12 SSTL18 Class I SSTL2 Class I, II SSTL3 Class I, II HSTL15 Class I, III HSTL18 Class I, II, III SSTL18D Class I, SSTL25D Class I, II, SSTL33D_I, II HSTL15D Class I HSTL18D Class I, II MLVDS LVDS LVDS25E[1] LVPECL[1] BLVDS[1] RSDS[1]
Inputs	All Single-ended, Differential	All Single-ended, Differential	All Single-ended, Differential	All Single-ended, Differential
Clock Inputs	All Single-ended, Differential	All Single-ended, Differential	All Single-ended, Differential	All Single-ended, Differential
PCI Support	PCI33 with clamp	PCI33 without clamp	PCI33 with clamp	PCI33 without clamp
LVDS Output Buffers		LVDS (3.5mA) Buffers[2]		LVDS (3.5mA) Buffers[2]

1. These differential standards are implemented by using a complementary LVCMOS driver with external resistor pack.
2. Available only on 50% of the I/Os in the bank.

LVCMOS Buffer Configurations

All LVCMOS buffer have programmable pull, programmable drive and programmable slew configurations that can be set in the software.

Bus Maintenance Circuit

Each pad has a weak pull-up, weak pull-down and weak buskeeper capability. The pull-up and pull-down settings offer a fixed characteristic, which is useful in creating wired logic such as wired ORs. However, current can be slightly higher than other options, depending on the signal state. The bus-keeper option latches the signal in the last driven state, holding it at a valid level with minimal power dissipation. Users can also choose to turn off the bus maintenance circuitry, minimizing power dissipation and input leakage. Note that in this case, it is important to ensure that inputs are driven to a known state to avoid unnecessary power dissipation in the input buffer. The weak bus keeper is not available when V_{CCIO} of the bank is assigned to 3.3V.

Programmable Drive

Each LVCMOS or LVTTL, as well as some of the referenced (SSTL and HSTL) output buffers, has a programmable drive strength option. This option can be set for each I/O independently. The drive strength settings available are 2mA, 4mA, 6mA, 8mA, 12mA, 16mA and 20mA. Actual options available vary by the I/O voltage. The user must consider the maximum allowable current per bank and the package thermal limit current when selecting the drive strength. Table 8-6 shows the available drive settings for each of the output standards.

Table 8-6. Programmable Drive Values for Single-ended Buffers

Single Ended I/O Standard	Programmable Drive (mA)
HSTL15_I	4, 8
HSTL18_I	8, 12
SSTL25_I	8, 12
SSTL25_II	16, 20
SSTL18_II	8, 12
LVCMOS12	2, 6
LVCMOS15	4, 8
LVCMOS18	4, 8, 12, 16
LVCMOS25	4, 8, 12, 16, 20
LVCMOS33	4, 8, 12, 16, 20
LVTTL	4, 8, 12, 16, 20

Programmable Slew Rate

Each LVCMOS or LVTTL output buffer pin also has a programmable output slew rate control that can be configured for either low noise or high-speed performance. Each I/O pin has an individual slew rate control. This allows designers to specify slew rate control on a pin-by-pin basis. This slew rate control affects both the rising and falling edges.

Open-Drain Control

All LVCMOS and LVTTL output buffers can be configured to function as open drain outputs. The user can implement an open drain output by turning on the OPENDRAIN attribute in the software.

Differential SSTL and HSTL support

The single-ended driver associated with the complementary ‘C’ pad can optionally be driven by the complement of the data that drives the single-ended driver associated with the true pad. This allows a pair of single-ended drivers to be used to drive complementary outputs with the lowest possible skew between the signals. This is used for driving complementary SSTL and HSTL signals (as required by the differential SSTL and HSTL clock inputs on syn-

chronous DRAM and synchronous SRAM devices respectively). This capability is also used in conjunction with off-chip resistors to emulate LVPECL, and BLVDS output drivers.

Table 8-7. Programmable Drive Values for Differential Buffers

Differential I/O Standard	Programmable Drive (mA)
HSTL15D_I	4, 8
HSTL18D_I	8, 12
SSTL25D_I	8, 12
SSTL25D_II	16, 20
SSTL18D_II	8, 12

PCI Support with Programmable PCICLAMP

Each sysIO buffer can be configured to support PCI33. The buffers on the top and bottom sides of the device have an optional PCI clamp diode that may optionally be specified in the ispLEVER design tools.

Programmable PCICLAMP can be turned ON or OFF. This option is available on each I/O independently on the top and bottom side banks.

Programmable Input Delay

Each input can optionally be delayed before it is passed to the core logic or input registers. The primary use for the input delay is to achieve zero hold time for the input registers when using a direct drive primary clock. To arrive at zero hold time, the input delay will delay the data by at least as much as the primary clock injection delay. This option can be turned ON or OFF for each I/O independently in the software using the FIXEDDELAY attribute. This attribute is described in more detail in the software sysIO attribute section. Appendix A shows how this feature can be enabled in the software using HDL attributes.

Software sysIO Attributes

sysIO attributes can be specified in the HDL, using the Spreadsheet view of the Design Planner or in the ASCII preference file (.lpf) file directly. Appendices A, B and C list examples of how these can be assigned using each of these methods. This section describes each of these attributes in detail.

IO_TYPE

This is used to set the sysIO standard for an I/O. The V_{CCIO} required to set these I/O standards are embedded in the attribute names itself. There is no separate attribute to set the V_{CCIO} requirements. Table 8-8 lists the available I/O types.

Table 8-8. IO_TYPE Attribute Values

sysIO Signaling Standard	IO_TYPE
DEFAULT	LVCMOS25
LVDS 2.5V	LVDS25
RSDS	RSDS[1]
Emulated LVDS 2.5V	LVDS25E[1]
Bus LVDS 2.5V	BLVDS25[1]
LVPECL 3.3V	LVPECL33[1]
HSTL18 Class I and II	HSTL18_I, HTSL18_II
Differential HSTL 18 Class I and II	HSTL18D_I, HSTL18D_II
HSTL 15 Class I	HSTL15_I
Differential HSTL 15 Class I	HSTL15D_I
SSTL 33 Class I and II	SSTL33_I, SSTL33_II
Differential SSTL 33 Class I and II	SSTL33D_I, SSTL3D_II
SSTL 25 Class I and II	SSTL25_I,SSTL25_II
Differential SSTL 25 Class I and II	SSTL25D_I, SSTL25D_II
SSTL 18 Class I and II	SSTL18_I, SSTL18_II
Differential SSTL 18 Class I and II	SSTL18D_I,SSTL18D_II
LVTTL	LVTTL33
3.3V LVCMOS	LVCMOS33
2.5V LVCMOS	LVCMOS25
1.8V LVCMOS	LVCMOS18
1.5V LVCMOS	LVCMOS15
1.2V LVCMOS	LVCMOS12
3.3V PCI	PCI33
MLVDS	MLVDS[1]

1. These differential standards are implemented by using a complementary LVCMOS driver with external resistor pack.

OPENDRAIN

LVCMOS and LVTTL I/O standards can be set to open drain configuration by using the OPENDRAIN attribute.

Values: ON, OFF
Default: OFF

DRIVE

The DRIVE attribute will set the programmable drive strength for the output standards that have programmable drive capability

Table 8-9. DRIVE Settings

Output Standard	DRIVE (mA)	Default (mA)
HSTL15_I/ HSTL15D_I	4, 8	8
HSTL18_I/ HSTL18D_I	8, 12	12
SSTL25_I/ SSTL25D_I	8, 12	8
SSTL25_II/ SSTL25D_II	16, 20	16
SSTL18_II/SSTL18D_II	8, 12	12
LVCMOS12	2, 6	6
LVCMOS15	4, 8	8
LVCMOS18	4, 8, 12, 16	12
LVCMOS25	4, 8, 12, 16, 20	12
LVCMOS33	4, 8, 12, 16, 20	12
LVTTL	4, 8, 12, 16, 20	12

PULLMODE

The PULLMODE attribute is available for all the LVTTL, LVCMOS and PCI inputs and outputs. This attribute can be enabled for each I/O independently.

Values: UP, DOWN, NONE, KEEPER
Default: UP

Table 8-10. PULLMODE Values

PULL Options	PULLMODE Value
Pull-up (Default)	UP
Pull-down	DOWN
Bus Keeper	KEEPER
Pull Off	NONE

PCICLAMP

PCI33 inputs on the top and bottom of the device have an optional PCI clamp that is enabled via the PCICLAMP attribute. The PCICLAMP is also available for all LVCMOS33 and LVTTL inputs.

Values: ON, OFF

Table 8-11. PCICLAMP Values

Input Type	PCICLAMP Value
PCI33	ON (default), OFF
LVCMOS33	OFF (default), ON
LVTTL	OFF (default), ON

SLEWRATE

The SLEWRATE attribute is available for all LVTTL and LVCMOS output drivers. Each I/O pin has an individual slew rate control. This allows a designer to specify slew rate control on a pin-by-pin basis.

Values: FAST, SLOW
Default: FAST

FIXEDDELAY

The FIXEDDELAY attribute is available to each input pin. This attribute, when enabled, is used to achieve zero hold time for the input registers when using global clock. This attribute can only be assigned in the HDL source.

Values: TRUE, FALSE
Default: FALSE

INBUF

By default, all the unused input buffers are disabled. The INBUF attribute is used to enable the unused input buffers when performing a boundary scan test. This is a global attribute and can be globally set to ON or OFF.

Values: ON, OFF
Default: ON

DIN/DOUT

This attribute can be used to assign I/O registers. Using DIN will assert an input register and using the DOUT attribute will assert an output register. By default, the software will try to assign the I/O registers, if applicable. The user can turn this OFF by using the synthesis attribute or by using the Spreadsheet view of the Design Planner. These attributes can only be applied to registers.

LOC

This attribute can be used to make pin assignments to the I/O ports in the design. This attributes is only used when the pin assignments are made in HDL source. Designers can also assign pins directly using the Spreadsheet view of the Design Planner in the ispLEVER software. The appendices explain this in further detail.

Design Considerations and Usage

This section discusses some of the design rules and considerations that must be taken into account when designing with the LatticeXP2 sysIO buffer

Banking Rules

- If V_{CCIO} or V_{CCJ} for any bank is set to 3.3 V, it is recommended that it be connected to the same power supply as $V_{CCAUX,}$ thus minimizing leakage.
- If V_{CCIO} or V_{CCJ} for any bank is set to 1.2V, it is recommended that it be connected to the same power supply as V_{CC}, thus minimizing leakage.
- When implementing DDR memory interfaces, the V_{REF1} of the bank is used to provide reference to the interface pins and cannot be used to power any other referenced inputs.
- Only the top and bottom banks (banks 0, 1, 4 and 5)) will support PCI clamps.
- All legal input buffers should be independent of bank $V_{CCIO,}$ except for 1.8V and 1.5V buffers, which require a bank V_{CCIO} of 1.8V and 1.5V.

Differential I/O Rules

- All banks can support LVDS input buffers. Only the banks on the right and left sides (Banks 2, 3, 6 and 7) will support True Differential output buffers. The banks on all sides will support the LVDS input buffers. The user can use emulated LVDS output buffers on these banks.
- All banks support emulated differential buffers using external resistor pack and complementary LVCMOS drivers.
- Only 50% of the I/Os on the left and right sides can provide LVDS output buffer capability. LVDS can only be assigned to the TRUE pad. The ispLEVER design tool will automatically assign the other I/Os of the differential pair to the complementary pad. Refer to the device data sheet to see the pin listings for all LVDS pairs.

Differential I/O Implementation

The LatticeXP2 devices support a variety of differential standards as detailed in the following sections.

LVDS

True LVDS (LVDS25) drivers are available on 50% of the I/Os on the left and right side of the devices. LVDS input support is provided on all sides of the device. All four sides of the device support LVDS using complementary LVCMOS drivers with external resistors (LVDS25E). Refer to the LatticeXP2 Family Data Sheet for a detailed explanation of these LVDS implementations.

BLVDS

All single-ended sysIO buffers pairs support the Bus-LVDS standard using complementary LVCMOS drivers with external resistors. Please refer to the LatticeXP2 Family Data Sheet for a detailed explanation of BLVDS implementation.

RSDS

All single-ended sysIO buffers pairs support RSDS standard using complementary LVCMOS drivers with external resistors. Please refer to the LatticeXP2 Family Data Sheet for a detailed explanation of RSDS implementation.

LVPECL

All the sysIO buffers will support LVPECL inputs. LVPECL outputs are supported using complementary LVCMOS driver with external resistors. Please refer to the LatticeXP2 Family Data Sheet for a detailed explanation of LVPECL implementation.

Differential SSTL and HSTL

All single-ended sysIO buffers pairs support differential SSTL and HSTL. Please refer to the LatticeXP2 Family Data Sheet for a detailed explanation of Differential HSTL and SSTL implementation.

MLVDS

All single-ended sysIO buffers pairs support MLVDS standard using complementary LVCMOS drivers with external resistors. Please refer to the LatticeXP2 Family Data Sheet for a detailed explanation of MLVDS implementation.

Technical Support Assistance

Hotline: 1-800-LATTICE (North America)
+1-503-268-8001 (Outside North America)
e-mail: techsupport@latticesemi.com
Internet: www.latticesemi.com

Revision History

Date	Version	Change Summary
February 2007	01.0	Initial release.
April 2007	01.1	Updated Supported Output Standards table.

Appendix A. HDL Attributes for Synplicity® and Precision® RTL Synthesis

Using these HDL attributes, designers can assign the sysIO attributes directly in their source. The attribute definition and syntax for the appropriate synthesis vendor must be used. Below are a list of all the sysIO attributes, syntax and examples for Precision RTL Synthesis and Synplicity. This section only lists the sysIO buffer attributes for these devices. These attributes are available through the ispLEVER software Help system.

VHDL Synplicity/Precision RTL Synthesis

This section lists syntax and examples for all the sysIO Attributes in VHDL when using the Precision RTL Synthesis or Synplicity synthesis tools.

Syntax

Table 8-12. VHDL Attribute Syntax for Synplicity and Precision RTL Synthesis

Attribute	Syntax
IO_TYPE	attribute IO_TYPE: string; attribute IO_TYPE of *Pinname*: signal is *"IO_TYPE Value";*
OPENDRAIN	attribute OPENDRAIN: string; attribute OPENDRAIN of *Pinname:* signal is *"OpenDrain Value";*
DRIVE	attribute DRIVE: string; attribute DRIVE of *Pinname:* signal is *"Drive Value";*
PULLMODE	attribute PULLMODE: string; attribute PULLMODE of *Pinname:* signal is *"Pullmode Value";*
PCICLAMP	attribute PCICLAMP: string; attribute PCICLAMP of *Pinname:* signal is *"PCIClamp Value";*
SLEWRATE	attribute PULLMODE: string; attribute PULLMODE of *Pinname:* signal is *"Slewrate Value";*
FIXEDDELAY	attribute FIXEDDELAY: string; attribute FIXEDDELAY of *Pinname:* signal is "*Fixeddelay Value*";
DIN	attribute DIN: string; attribute DIN of *Pinname*: signal is " ";
DOUT	attribute DOUT: string; attribute DOUT of *Pinname:* signal is " ";
LOC	attribute LOC: string; attribute LOC of *Pinname:* signal is "pin_locations";

Examples

IO_TYPE

```
--***Attribute Declaration***
ATTRIBUTE IO_TYPE: string;
--***IO_TYPE assignment for I/O Pin***
ATTRIBUTE IO_TYPE OF portA:    SIGNAL IS "PCI33";
ATTRIBUTE IO_TYPE OF portB:    SIGNAL IS "LVCMOS33";
ATTRIBUTE IO_TYPE OF portC:    SIGNAL IS "LVDS25";
```

OPENDRAIN

```
--***Attribute Declaration***
ATTRIBUTE OPENDRAIN: string;
--***DRIVE assignment for I/O Pin***
ATTRIBUTE OPENDRAIN OF portB: SIGNAL IS "ON";
```

DRIVE

```
--***Attribute Declaration***
ATTRIBUTE DRIVE: string;
--***DRIVE assignment for I/O Pin***
ATTRIBUTE DRIVE OF portB: SIGNAL IS “20”;
```

PULLMODE

```
--***Attribute Declaration***
ATTRIBUTE PULLMODE : string;
--***PULLMODE assignment for I/O Pin***
ATTRIBUTE PULLMODE OF portA:    SIGNAL IS “DOWN”;
ATTRIBUTE PULLMODE OF portB:    SIGNAL IS “UP”;
```

PCICLAMP

```
--***Attribute Declaration***
ATTRIBUTE PCICLAMP: string;
--***PULLMODE assignment for I/O Pin***
ATTRIBUTE PCICLAMP OF portA:    SIGNAL IS “ON”;
```

SLEWRATE

```
--***Attribute Declaration***
ATTRIBUTE SLEWRATE : string;
--*** SLEWRATE assignment for I/O Pin***
ATTRIBUTE SLEWRATE OF portB: SIGNAL IS “FAST”;
```

FIXEDDELAY

```
--***Attribute Declaration***
ATTRIBUTE FIXEDDELAY: string;
--*** SLEWRATE assignment for I/O Pin***
ATTRIBUTE FIXEDDELAY OF portB: SIGNAL IS “TRUE”;
```

DIN/DOUT

```
--***Attribute Declaration***
ATTRIBUTE din : string;
ATTRIBUTE dout : string;
--*** din/dout assignment for I/O Pin***
ATTRIBUTE din OF input_vector: SIGNAL IS “ “;
ATTRIBUTE dout OF output_vector: SIGNAL IS “ “;
```

LOC

```
--***Attribute Declaration***
ATTRIBUTE LOC : string;
--*** LOC assignment for I/O Pin***
ATTRIBUTE LOC OF input_vector: SIGNAL IS “E3,B3,C3 “;
```

Verilog Synplicity

This section lists syntax and examples for all the sysIO Attributes in Verilog using the Synplicity synthesis tool.

Syntax

Table 8-13. Verilog Synplicity Attribute Syntax

Attribute	Syntax
IO_TYPE	*PinType PinName* /* synthesis IO_TYPE=”IO_Type Value”*/;
OPENDRAIN	*PinType PinName* /* synthesis OPENDRAIN =”OpenDrain Value”*/;
DRIVE	*PinType PinName* /* synthesis DRIVE=”Drive Value”*/;
PULLMODE	*PinType PinName* /* synthesis PULLMODE=”Pullmode Value”*/;
PCICLAMP	*PinType PinName* /* synthesis PCICLAMP =” PCIClamp Value”*/;
SLEWRATE	*PinType PinName* /* synthesis SLEWRATE=”Slewrate Value”*/;
FIXEDDELAY	*PinType PinName* /* synthesis FIXEDDELAY=”Fixeddelay Value”*/;
DIN	*PinType PinName* /* synthesis DIN=” “*/;
DOUT	*PinType PinName* /* synthesis DOUT=” “*/;
LOC	*PinType PinName* /* synthesis LOC=”pin_locations “*/;

Examples

```
//IO_TYPE, PULLMODE, SLEWRATE and DRIVE assignment
output portB /*synthesis IO_TYPE=”LVCMOS33” PULLMODE =”UP” SLEWRATE =”FAST”
DRIVE =”20”*/;
output portC /*synthesis IO_TYPE=”LVDS25” */;

//OPENDRAIN
output portA /*synthesis OPENDRAIN =”ON”*/;

//PCICLAMP
output portA /*synthesis IO_TYPE=”PCI33” PULLMODE =”PCICLAMP”*/;

// Fixeddelay
input load /* synthesis FIXEDDELAY=”TRUE” */;

// Place the flip-flops near the load input
input load /* synthesis din=”” */;

// Place the flip-flops near the outload output
output outload /* synthesis dout=”” */;

//I/O pin location
input [3:0] DATA0 /* synthesis loc=”E3,B1,F3”*/;

//Register pin location
reg data_in_ch1_buf_reg3 /* synthesis loc=”R40C47” */;

//Vectored internal bus
reg [3:0] data_in_ch1_reg /*synthesis loc =”R40C47,R40C46,R40C45,R40C44” */;
```

Verilog Precision

This section lists syntax and examples for all the sysIO Attributes in Verilog using the Precision RTL Synthesis tool.

Syntax

Table 8-14. Verilog Precision Attribute Syntax

Attribute	Syntax
IO_TYPE	//pragma attribute *PinName* IO_TYPE IO_TYPE Value
OPENDRAIN	// pragma attribute *PinName* OPENDRAIN OpenDrain Value
DRIVE	// pragma attribute *PinName* DRIVE Drive Value
PULLMODE	// pragma attribute *PinName* IO_TYPE Pullmode Value
PCICLAMP	// pragma attribute *PinName* PCICLAMP PCIClamp Value
SLEWRATE	// pragma attribute *PinName* IO_TYPE Slewrate Value
FIXEDDELAY	// pragma attribute *PinName* IO_TYPE Fixeddelay Value
LOC	// pragma attribute *PinName* LOC pin_location

Examples

```
//****IO_TYPE ***
//pragma  attribute portA IO_TYPE PCI33
//pragma  attribute portB IO_TYPE LVCMOS33
//pragma  attribute portC IO_TYPE SSTL25_II

//*** Opendrain ***
//pragma  attribute portB OPENDRAIN ON
//pragma  attribute portD OPENDRAIN OFF

//*** Drive ***
//pragma  attribute portB DRIVE 20
//pragma  attribute portD DRIVE 8

//*** Pullmode***
//pragma  attribute portB PULLMODE UP

//*** PCIClamp***
//pragma  attribute portB PCICLAMP ON

//*** Slewrate ***
//pragma  attribute portB SLEWRATE FAST
//pragma  attribute portD SLEWRATE SLOW

// ***Fixeddelay***
// pragma  attribute load FIXEDDELAY TRUE

//***LOC***
//pragma  attribute portB loc E3
```

Appendix B. sysIO Attributes Using the Design Planner User Interface

Designers can assign sysIO buffer attributes using the Design Planner GUI available in the ispLEVER design tool. The Pin Attribute Sheet list all the ports in a design and all the available sysIO attributes as preferences. By clicking on each of these cells, a list of all the valid I/O preference for that port is displayed. Each column takes precedence over the next. Therefore, when a particular IO_TYPE is chosen, the DRIVE, PULLMODE and SLEWRATE columns will only list the valid combinations for that IO_TYPE. The pin locations can be locked using the pin location column of the Pin Attribute sheet. Right-clicking on a cell will list the available pin locations. The Spreadsheet View will also conduct a DRC check to search for any incorrect pin assignments.

Designers can enter DIN/DOUT preferences using the Cell Attributes sheet of the Preference Editor. All the preferences assigned using the Preference Editor are written into the preference file (.prf).

Figures 8-2 and 8-3 show the Pin Attribute sheet and the Cell Attribute sheet views of the preference editor. For further information on how to use the Preference Editor, refer to the ispLEVER Help documentation in the Help menu option of the software.

Figure 8-2. Port Attributes Tab

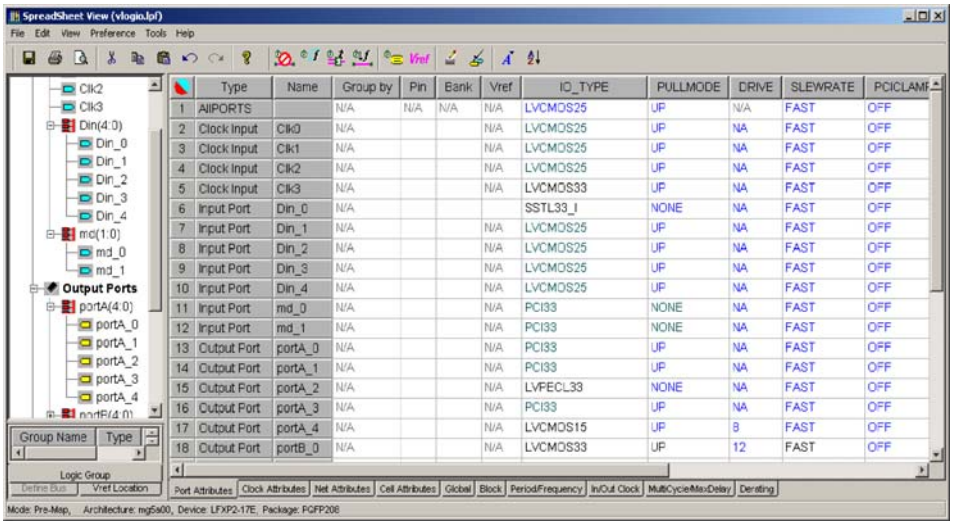
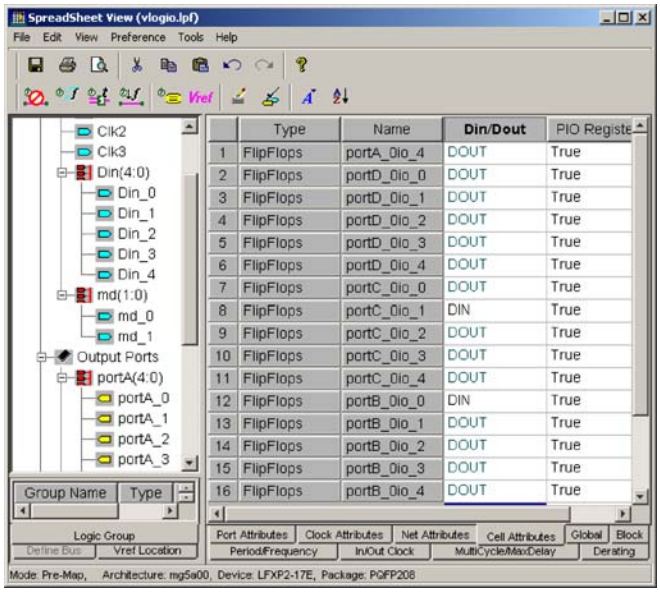


Figure 8-3. Cell Attributes Tab



Appendix C. sysIO Attributes Using Preference File (ASCII File)

Designers can enter sysIO attributes directly in the preference (.lpf) file as sysIO buffer preferences. The LPF file is a post-synthesis FPGA constraint file that stores logical preferences that have been created or modified in the Design Planner or Text Editor. It also contains logical preferences originating in the HDL source that have been modified.

. Below are a list of sysIO buffer preference syntax and examples.

IOBUF

This preference is used to assign the attribute IO_TYPE, PULLMODE, SLEWRATE,PCICLAMP and DRIVE.

Syntax

```
IOBUF [ALLPORTS | PORT <port_name> | GROUP <group_name>] (keyword=<value>)+;
```

where:

<port_name> = These are not the actual top-level port names, but should be the signal name attached to the port. PIOs in the physical design (.ncd) file are named using this convention. Any multiple listings or wildcarding should be done using GROUPs

Keyword = IO_TYPE, OPENDRAIN, DRIVE, PULLMODE, PCICLAMP, SLEWRATE.

Example

```
IOBUF PORT “port1” IO_TYPE=LVTTL33 OPENDRAIN=ON DRIVE=8 PULLMODE=UP
PCICLAMP =OFF SLEWRATE=FAST;
DEFINE PORT GROUP “bank1” “in*” “out_[0-31]”;
IOBUF GROUP “bank1” IO_TYPE=SSTL18_II;
```

LOCATE

When this preference is applied to a specified component, it places the component at a specified site and locks the component to the site. If applied to a specified macro instance, it places the macro’s reference component at a specified site, places all of the macro’s pre-placed components (that is, all components that were placed in the macro’s library file) in sites relative to the reference component, and locks all of these placed components at their sites. Below are some of the LOCATE syntax and examples. For further information, refer to the ispLEVER Help documentation in the Help menu option of the software.

Syntax

```
LOCATE [COMP <comp_name> | MACRO <macro_name>] SITE <site_name>;
LOCATE VREF <vref_name> SITE <site_name>;
```

Note: If the comp_name, macro_name, or site_name begins with anything other than an alpha character (for example, “11C7”), you must enclose the name in quotes. Wildcard expressions are allowed in <comp_name>.

Examples

This command places the port Clk0 on the site A4:

```
LOCATE COMP “Clk0” SITE “A4”;
```

This command places the component PFU1 on the site named R1C7:

```
LOCATE COMP “PFU1” SITE “R1C7”;
LOCATE VREF “ref1” SITE PR29C;
```

USE DIN CELL

This preference specifies the given register to be used as an input flip-flop.

Syntax

```
USE DIN CELL <cell_name>;
```

where:

```
<cell_name> := string
```

Example

```
USE DIN CELL “din0”;
```

USE DOUT CELL

Specifies the given register to be used as an output flip-flop.

Syntax

```
USE DOUT CELL <cell_name>;
```

where:

```
<cell_name> := string
```

Example

```
USE DOUT CELL “dout1”;
```

GROUP VREF

This preference is used to group all the components that need to be associated to one V_{REF} pin within a bank.

Syntax

```
LOCATE VREF <vref_name> SITE <site_name>;
```

Example

```
IOBUF GROUP <group_name> BANK=<bank_name> VREF=<Vref_name>
LOCATE VREF “ref1” SITE PR29C;
LOCATE VREF “ref2” SITE PR48B;
IOBUF GROUP "group1" IO_TYPE=SSTL18_II BANK=0 VREF=vref1 ;
```

LatticeXP2 sysCLOCK PLL Design and Usage Guide

February 2007 Technical Note TN1126

Introduction

This user's guide describes the clock resources available in the LatticeXP2™ device architecture. Details are provided for primary clocks, secondary clocks and edge clocks as well as clock elements such as PLLs, clock dividers and more.

The number of PLLs and DDR-DLLs for each device is listed in Table 9-1.

Table 9-1. Number of PLLs and DDR-DLLs

Parameter	Description	XP2-5	XP2-8	XP2-17	XP2-30	XP2-40
Number of GPLLs	General purpose PLL	2	2	4	4	4
Number of DDR-DLLs	DDL for DDR applications	2	2	2	2	2

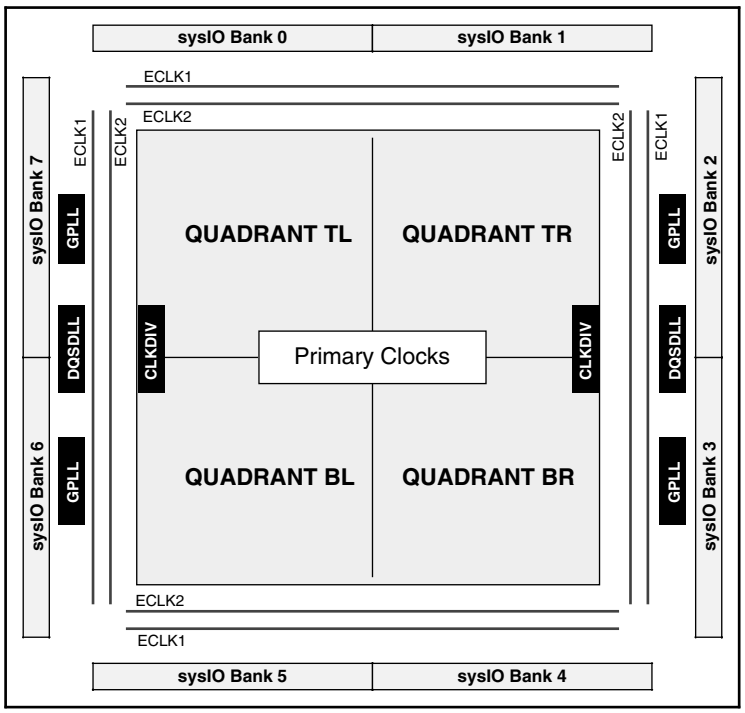
Clock/Control Distribution Network

LatticeXP2 devices provide global clock distribution in the form of eight quadrant-based primary clocks and flexible secondary clocks. Two edge clocks are also provided on every edge of the device. Other clock sources include clock input pins, internal nodes, PLLs, and clock dividers.

LatticeXP2 Top Level View

Figure 9-1 provides a view of the primary clocking structure of the LatticeXP2-40 device.

Figure 9-1. LatticeXP2 Clocking Structure (LFXP2-40)



Primary Clocks

Each quadrant receives up to eight primary clocks. Two of these clocks provide the Dynamic Clock Selection (DCS) feature. The six primary clocks without DCS can be specified in the Pre-map Preference Editor as 'Primary Pure' and the two DCS clocks as 'Primary-DCS'.

The sources of primary clocks are:

- PLL outputs
- CLKDIV outputs
- Dedicated clock pins
- Internal nodes

Secondary Clocks

The LatticeXP2 secondary clocks are a flexible region-based clocking resource. Each region can have four independent clock inputs. Since the secondary clock is a regional resource, it can cross the primary clock quadrant boundaries.

There are eight secondary clock muxes per quadrant. Each mux has inputs from four different sources. Three of these are from internal nodes. The fourth input comes from a primary clock pin. The input sources are not necessarily located in the same quadrant as the mux. This structure enables global use of secondary clocks.

The sources of secondary clocks are:

- Dedicated clock pins
- Clock Divider (CLKDIV) outputs
- Internal nodes

Table 9-2 lists the number of secondary clock regions in LatticeXP2 devices.

Table 9-2. Number of Secondary Clock Regions

Parameter	XP2-5	XP2-8	XP2-17	XP2-30	XP2-40
Number of regions	6	6	6	6	8

Edge Clocks

The LatticeXP2 device has two Edge Clocks (ECLK) per side. These clocks, which have low injection time and skew, are used to clock I/O registers. Edge clock resources are designed for high speed I/O interfaces with high fanout capability. Refer to Appendix B for detailed information on ECLK locations and connectivity.

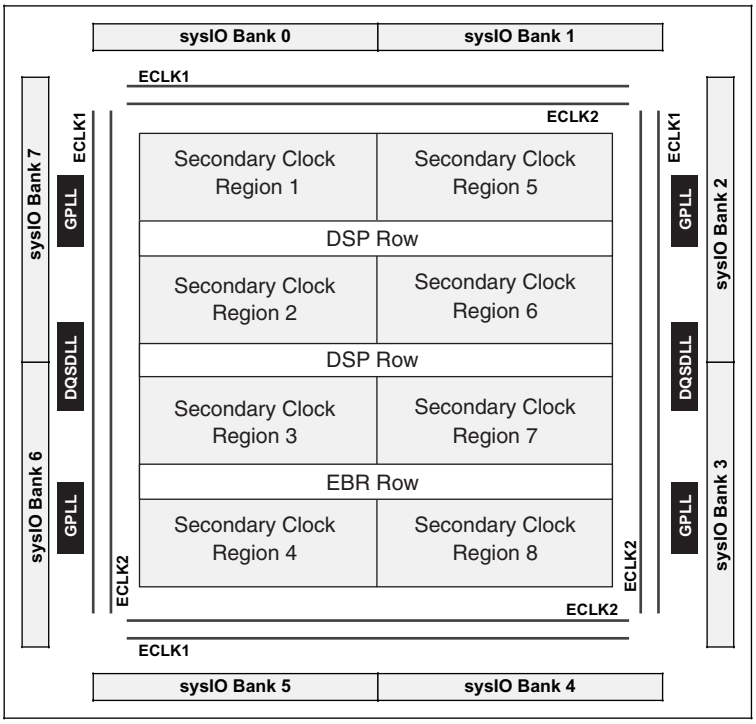
The sources of edge clocks are:

- Left and Right Edge Clocks
  - Dedicated clock pins
  - PLL outputs
  - PLL input pins
  - Internal nodes
- Top and Bottom Edge Clocks
  - Dedicated clock pins
  - Internal nodes

Edge clocks can directly drive the secondary clock resources and general routing resources. Refer to Figure 9-21 for detailed information on edge clock routing.

Figure 9-2 describes the structure of the secondary clocks and edge clocks.

Figure 9-2. LatticeXP2 Secondary Clocks and Edge Clocks (LFXP2-40)



Primary Clock Note

The CLKOP must be used as the feedback source to optimize PLL performance.

Most designers use PLLs for clock tree injection removal mode and the CLKOP should be assigned to a primary clock. This is done automatically by the software unless otherwise specified by the user.

CLKOP can route only to CLK0 to CLK5, while CLKOS/CLKOK can route to all primary clocks (CLK0 TO CLK7).

When CLK6 or CLK7 is used as a primary clock and there is only one clock input to the DCS, the DCS is assigned as a buffer mode by the software. Please see the DCS section of this document for detailed information.

Specifying Clocks in the Design Tools

If desired, designers can specify the clock resources, primary, secondary or edge to be used to distribute a given clock source. Figure 9-3 illustrates how this can be done in the Pre-map Preference editor. Alternatively, the preference file can be used, as discussed in Appendix C.

Primary-Pure and Primary-DCS

Primary Clock Net can be assigned to either Primary-Pure (CLK0 to CLK5) or Primary-DCS (CLK6 and CLK7).

Global Primary Clock and Quadrant Primary Clock

Global Primary Clock

If a primary clock is not assigned as a quadrant clock, the software assumes it is a global clock.

There are six Global Primary/Pure clocks and two Global Primary/DCS clocks available.

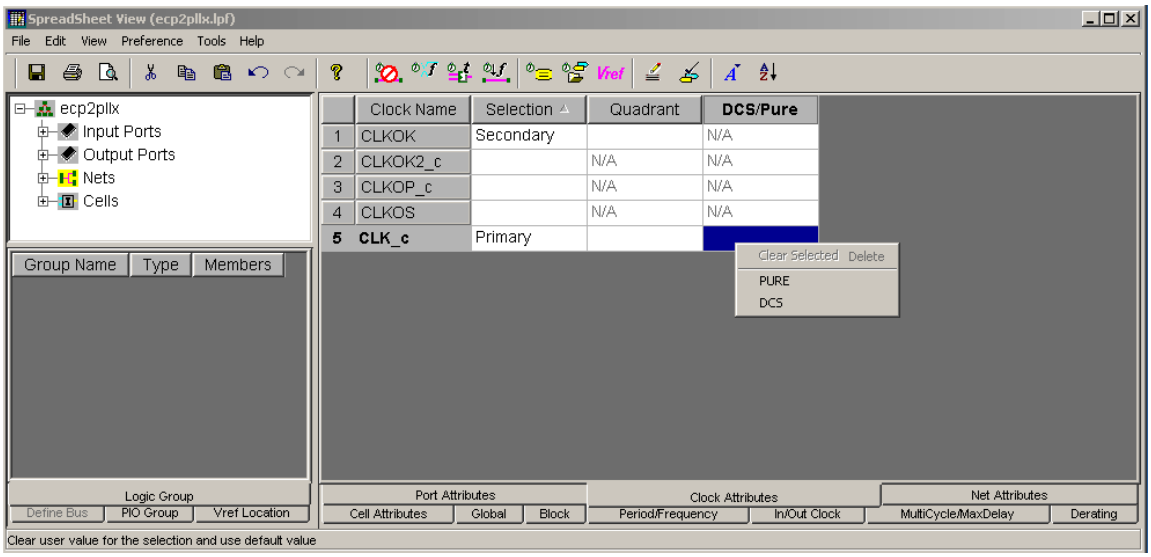
Quadrant Primary Clock

Any primary clock may be assigned to a quadrant clock. The clock may be assigned to a single quadrant or to two adjacent quadrants (not diagonally adjacent).

When a quadrant clock net is used, the user must ensure that the registers each clock drives can be assigned in that quadrant without any routing issues.

In the Quadrant Primary Clocking scheme, the maximum number of primary clocks is 32, as long as all the primary clock sources are available.

Figure 9-3. Clock Attributes in the Pre-map Preference Editor

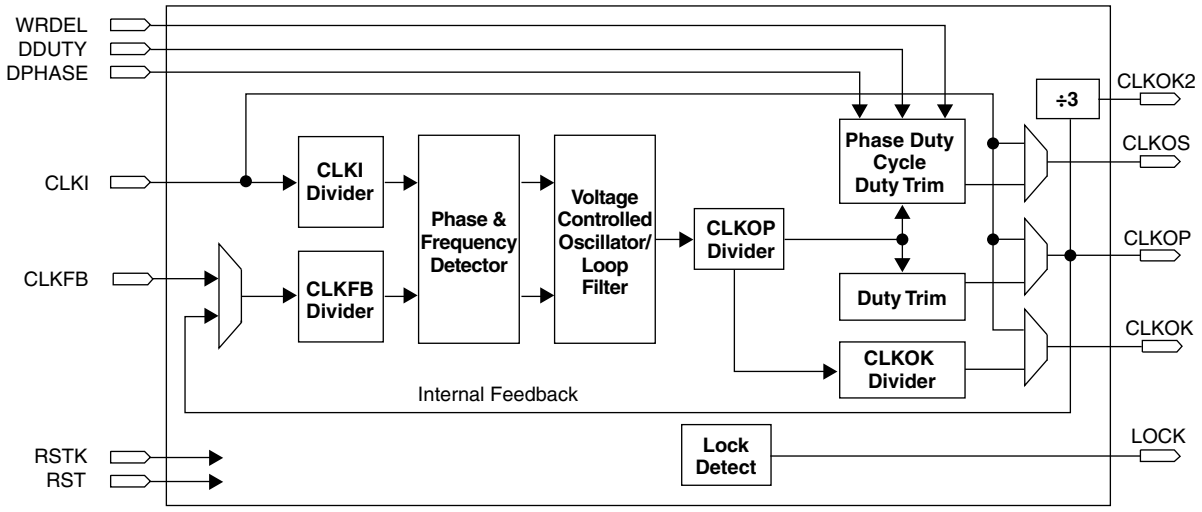


Refer to Appendix A for detailed clock network diagrams.

sysCLOCK™ PLL

The LatticeXP2 PLL provides features such as clock injection delay removal, frequency synthesis, phase/duty cycle adjustment, and dynamic delay adjustment. Figure 9-4 shows the block diagram of the LatticeXP2 PLL.

Figure 9-4. LatticeXP2 PLL Block Diagram



Functional Description

PLL Divider and Delay Blocks

Input Clock (CLKI) Divider
The CLKI divider is used to control the input clock frequency into the PLL block. The divider setting directly corresponds to the divisor of the output clock. The input and output of the input divider must be within the input and output frequency ranges specified in the device data sheet.

Feedback Loop (CLKFB) Divider
The CLKFB divider is used to divide the feedback signal. Effectively, this multiplies the output clock, because the divided feedback must speed up to match the input frequency into the PLL block. The PLL block increases the output frequency until the divided feedback frequency equals the input frequency. The input and output of the feedback divider must be within the input and output frequency ranges specified in the device data sheet.

Output Clock (CLKOP) Divider
The CLKOP divider serves the dual purposes of squaring the duty cycle of the VCO output and scaling up the VCO frequency into the 435MHz to 870MHz range to minimize jitter. The CLKOP divider values are the same whether or not the CLKOS is used.

CLKOK Divider
The CLKOK divider acts as a source for the global clock nets. It divides the CLKOP signal of the PLL by the value of the divider to produce lower frequency clock.

CLKOK2 Divider
The CLKOK2 is CLKOP divided by 3 for generating 140 MHz from 420 MHz to support SPI4.2.

Phase Adjustment and Duty Cycle Select (Static Mode)
Users can program CLKOS with Phase and Duty Cycle options. Phase adjustment can be done in 22.5° steps. The duty cycle resolution is 1/16th of a period except 1/16th and 15/16th duty cycle options are not supported to avoid minimum pulse violation.

Dynamic Phase Adjustment (DPHASE) and Dynamic Duty Cycle (DDUTY) Select
The Phase Adjustment and Duty Cycle Select can be controlled in dynamic mode. When this mode is selected, both the Phase Adjustment and Duty Cycle Select must be in dynamic mode. If only one of the features is to be used in dynamic mode, users can set the other control inputs with the fixed logic levels of their choice.

Duty Trim Adjustment
With the LatticeXP2 device family, the duty cycle can be fine-tuned with the Duty Trim Adjustment.

Fine Delay Adjust
This optional feature is controlled by the input port, WRDEL. See information on the WRDEL input in the next section of this document.

PLL Inputs and Outputs

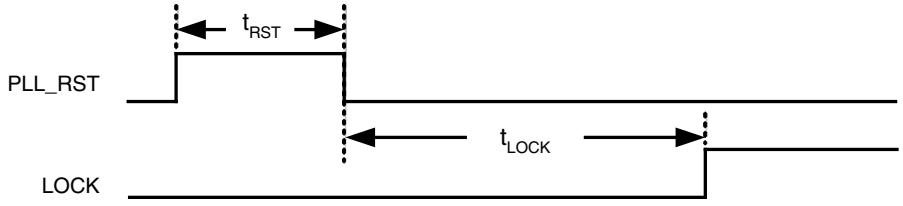
CLKI Input

The CLKI signal is the reference clock for the PLL. It must conform to the specifications in the data sheet in order for the PLL to operate correctly. The CLKI can be derived from a dedicated dual-purpose pin or from routing.

RST Input

The PLL reset occurs under two conditions. At power-up an internal power-up reset signal from the configuration block resets the PLL. The user-controlled PLL reset signal RST is provided as part of the PLL module that can be driven by an internally generated reset function or a pin. This RST signal resets all internal PLL counters, flip-flops (including the M-Divider) and the charge pump. The M-Divider reset synchronizes the M-Divider output to the input clock. When RST goes inactive, the PLL will start the lock-in process, and will take the t_{LOCK} time to complete the PLL lock. Figure 9-5 shows the timing diagram of the RST input. RST is active high. The RST signal is optional.

Figure 9-5. RST Input Timing Diagram



RSTK Input

RSTK is the reset input for the K-Divider. This K-Divider reset is used to synchronize the K-Divider output clock to the input clock. LatticeXP2 has an optional gearbox in the I/O cell for both outputs and inputs. The K-Divider reset is useful for the gearbox implementation. RSTK is active high.

CLKFB Input

The feedback signal to the PLL, which is fed through the feedback divider, can be derived from the Primary Clock net (CLKOP), a preferred pin, directly from the CLKOP divider or from general routing. External feedback allows the designer to compensate for board-level clock alignment.

CLKOP Output

The sysCLOCK PLL main clock output, CLKOP, is a signal available for selection as a primary clock. This clock signal is available at the CLK_OUT pin.

CLKOS Output with Phase and Duty Cycle Select

The sysCLOCK PLL auxiliary clock output, CLKOS, is a signal available for selection as a primary clock. The CLKOS is used when phase shift and/or duty cycle adjustment is desired. The programmable phase shift allows for different phase in increments of 22.5°. The duty select feature provides duty select in 1/16th of the clock period. This feature is also supported in Dynamic Control Mode.

CLKOK Output with Lower Frequency

The CLKOK is used when a lower frequency is desired. It is a signal available for selection as a primary clock.

CLKOK2 Output

This extra clock is provided for SPI4.2 application. The 420 MHz CLKOP clock is divided by 3, producing 140 MHz. The clock can also be used for any applications where CLKOP-divided-by-3 is required.

LOCK Output

The LOCK output provides information about the status of the PLL. After the device is powered up and the input clock is valid, the PLL will achieve lock within the specified lock time. Once lock is achieved, the PLL lock signal will be asserted. If, during operation, the input clock or feedback signals to the PLL become invalid, the PLL will lose lock. It is recommended to assert PLL RST to re-synchronize the PLL to the reference clock. The LOCK signal is available to the FPGA routing to implement generation of RST.

Dynamic Phase and Dynamic Duty Cycle Adjustment

The DPHASE[3:0] port is used with the Dynamic Phase Adjustment feature to allow the user to connect a control signal to the PLL. The DDUTY[3:0] port is used with the Dynamic Duty Adjustment feature to allow the user to connect a control signal to the PLL. The DPHASE and DDUTY ports are listed in Table 9-3.

The Dynamic Phase and Dynamic Duty Cycle Adjustment features will be discussed in more detail in later sections of this document.

Table 9-3. Dynamic Phase and Duty Cycle Adjust Ports

Port Name	I/O	Description
DPHASE[3:0]	I	Dynamic Phase Adjust inputs
DDUTY[3:0]	I	Dynamic Duty Cycle Adjust inputs

WRDEL (Write Delay)

The fine delay option supports SPI4.2. The PLL has a coarse phase adjust feature in which a cycle is divided into 16 equal steps (22.5°). For SPI4.2 running at 840 Mbps, the clock frequency is 420 MHz. At this frequency, the period is roughly 150 ps. This is slightly too coarse for dynamic phase adjust requirements. It may be more effective to use increments half as large. Combined with coarse phase adjust, a 70 ps (nominal) step provides effectively 32 steps of phase adjustment. This fine delay only applies to CLKOS (just like the coarse phase adjust). This is convenient since it allows one GPLL to be used for both read and write (where read uses CLKOS and write uses CLKOP).

PLL Attributes

The PLL utilizes several attributes that allow the configuration of the PLL through source constraints and a preference file. The following section details these attributes and their usage.

FIN

The input frequency can be any value within the specified frequency range based on the divider settings.

CLKI_DIV, CLKFB_DIV, CLKOP_DIV, CLKOK_DIV

These dividers determine the output frequencies of each output clock. The user is not allowed to input an invalid combination. Valid combinations are determined by the input frequency, the dividers, and the PLL specifications.

Note: Unlike PLLs in the LatticeECP™, LatticeEC™, LatticeXP™ and MachXO™ devices, the CLKOP divider values are the same whether or not CLKOS is used.

The CLKOP_DIV value is calculated to maximize the f_{VCO} within the specified range based on FIN and CLKOP_FREQ in conjunction with the CLKI_DIV and CLKFB_DIV values. These value settings are designed such that the output clock duty cycle is as close to 50% as possible. Table 9-4 shows the possible divider ranges.

Table 9-4. Divider Ranges

Attribute	Name	Value	Default
CLKI Divider Setting	CLKI_DIV	1 to 43	1
CLKFB Divider Setting	CLKFB_DIV	1 to 43	1
CLKOP Divider Setting	CLKOP_DIV	2, 4, 8, 16, 32, 48, 64, 80	8
CLKOK Divider Setting	CLKOK_DIV	2, 4, 6,.., 126, 128	2

FREQUENCY_PIN_CLKI, FREQUENCY_PIN_CLKOP, FREQUENCY_PIN_CLKOK

These input and output clock frequencies determine the divider values.

CLKOP Frequency Tolerance

When the desired output frequency is not achievable, users may enter the frequency tolerance of the clock output.

PHASEADJ (Phase Shift Adjust)
The PHASEADJ attribute is used to select Phase Shift for CLKOS output. The phase adjustment is programmable in 22.5° increments.

DUTY (Duty Cycle)
The DUTY attribute is used to select the Duty Cycle for CLKOS output. The Duty Cycle is programmable at 1/16th of the period increment. Steps 2 to 14 are supported. 1/16th and 15/16th duty cycles are not supported to avoid the minimum pulse width violation.

FB_MODE
There are three sources of feedback signals that can drive the CLKFB Divider: Internal, CLKOP (Clock Tree) and user clock. CLKOP (Clock Tree) feedback is used by default. Internal feedback takes the CLKOP output at CLKOP Divider output before the Clock Tree to minimize the feedback path delay. The user clock feedback is driven from the dedicated pin, clock pin or user-specified internal logic.

DUTY_TRIM Adjustment (Dynamic mode only)
Users can fine tune the duty cycle of CLKOP and/or CLKOS with the DUTY_TRIM feature when Dynamic PHASE/DUTY Adjustment is selected.

- TRIM Polarity Select: Users can select either rising edge or falling edge of clock to trim.
- TRIM Delay of CLKOP can be set to 0 to 7 steps of unit trim delay.
- TRIM Delay of CLKOS can be set to 0 to 3 steps of unit trim delay.

CLKOS/CLKOK/CLKOK2 Select
Users select these output clocks only when they are used in the design.

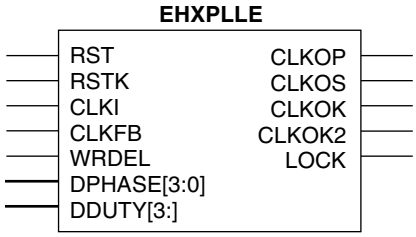
CLKOP/CLKOS/CLKOK BYPASS
These bypasses are enabled if set. CLKI is directly routed to its corresponding output clock.

RST/RSTK Select
Users may select these reset signals only when they are used in the design.

LatticeXP2 PLL Primitive Definition

One PLL primitive is used for LatticeXP2 PLL implementation. Figure 9-6 shows the LatticeXP2 PLL primitive library symbols.

Figure 9-6. LatticeXP2 PLL Primitive Symbol



EPLLD Design Migration from LatticeECP2 to LatticeXP2

The EPLLD generated for LatticeECP2 can be used with minor changes. If the configuration does not include Dynamic Phase and Duty Options, the migration is fully supported. If Dynamic Phase and Duty Options are included, the user must tie the DPAMODE port to ground.

Dynamic Phase/Duty Mode

This mode sets both Dynamic Phase Adjustment and Dynamic Duty Select at the same time. There are two modes, “Dynamic Phase and Dynamic Duty” and “Dynamic Phase and 50% Duty”.

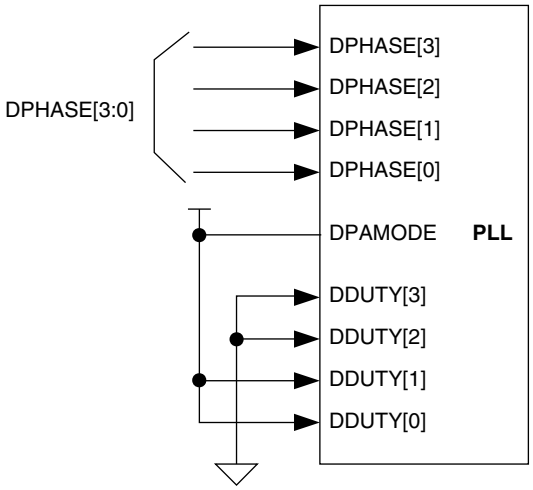
Dynamic Phase and 50% Duty
This mode allows users to set up Dynamic Phase inputs only. The 50% Duty Cycle is handled internally by the ispLEVER® software. The DDUTY[3:0] ports are user-transparent in this mode.

Dynamic Phase and Dynamic Duty
This mode allows designers to use both DDPHASE[3:0] and DDUTY[3:0] ports to input dynamic values.

To use Dynamic Phase Adjustment with a fixed duty cycle other than a 50%, simply set the DDUTY[3:0] inputs to the desired duty cycle value. Figure 9-7 illustrates an example circuit.

Example: Assume a design uses dynamic phase adjustment and a fixed duty cycle select and the desired duty cycle in 3/16th of a period. The setup should be as shown in Figure 9-7.

Figure 9-7. Example of Dynamic Phase Adjustment with a Fixed Duty Cycle of 3/16th of a Period



Dynamic Phase Adjustment/Duty Cycle Select

Phase Adjustment settings are described in Table 9-5.

Table 9-5. Phase Adjustment Settings

DPHASE[3:0]	Phase (°)
0000	0
0001	22.5
0010	45
0011	67.5
0100	90
0101	112.5
0110	135
0111	157.5
1000	180
1001	202.5
1010	225
1011	247.5
1100	270
1101	292.5
1110	315
1111	337.5

Duty Cycle Select settings are described in Table 9-6.

Table 9-6. Duty Cycle Select Settings

DDUTY[3:0]	Duty Cycle (1/16th of a Period)	Comment
0000	0	Not Supported
0001	1	Not Supported
0010	2	
0011	3	
0100	4	
0101	5	
0110	6	
0111	7	
1000	8	
1001	9	
1010	10	
1011	11	
1100	12	
1101	13	
1110	14	
1111	15	Not Supported

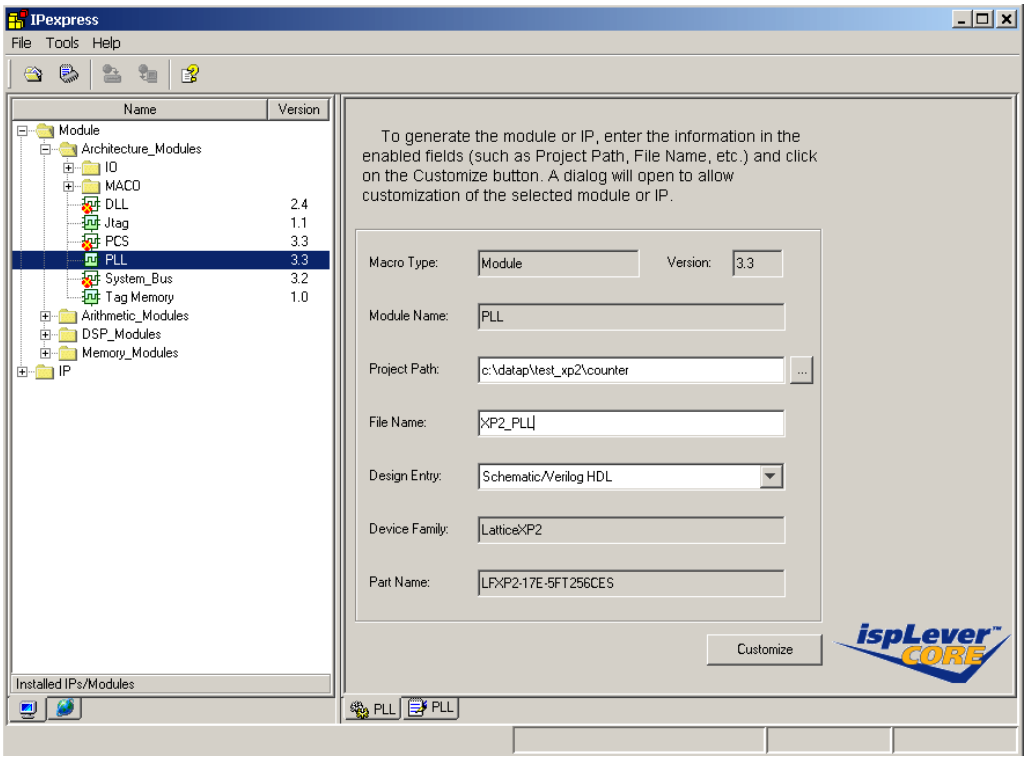
Note: PHASE/DUTY_CTNL is selected in the GUI 'PLL Phase & Duty Options' box and if it is set to 'Dynamic Mode', then both DPHASE[3:0] and DDUTY[3:0] inputs must be provided. If one of these inputs is a fixed value, the inputs must be tied to the desired fixed logic levels.

PLL Usage in IPexpress™

IPexpress is used to create and configure a PLL. The graphical user interface is used to select parameters for the PLL. The result is an HDL model to be used in the simulation and synthesis flow.

Figure 9-8 shows the main window when PLL is selected. The only entry required in this window is the module name. Other entries are set to the project settings. Users may change these entries, if desired. After entering the module name of choice, clicking on **Customize** will open the Configuration Tab window as shown in Figure 9.

Figure 9-8. IPexpress Main Window



Configuration Tab

The Configuration Tab lists all user accessible attributes with default values set. Upon completion, clicking **Generate** will generate source and constraint files. Users may choose to use the .lpc file to load parameters.

Configuration Modes

There are two modes that can be used to configure the PLL in the Configuration Tab, Frequency Mode and Divider Mode.

Frequency Mode: In this mode, the user enters input and output clock frequencies and the software calculates the divider settings. If the output frequency entered is not achievable the nearest frequency will be displayed in the 'Actual' text box. After input and output frequencies are entered, clicking the **Calculate** button will display the divider values.

Divider Mode: In this mode, the user sets the divider settings with input frequency. Users must choose the CLKOP Divider value to maximize the f_{VCO} and achieve optimum PLL performance. After input frequency and divider settings are set, clicking the **Calculate** button will display the frequencies. Figure 9-9 shows the Configuration Tab.

Figure 9-9. LatticeXP2 PLL Configuration Tab

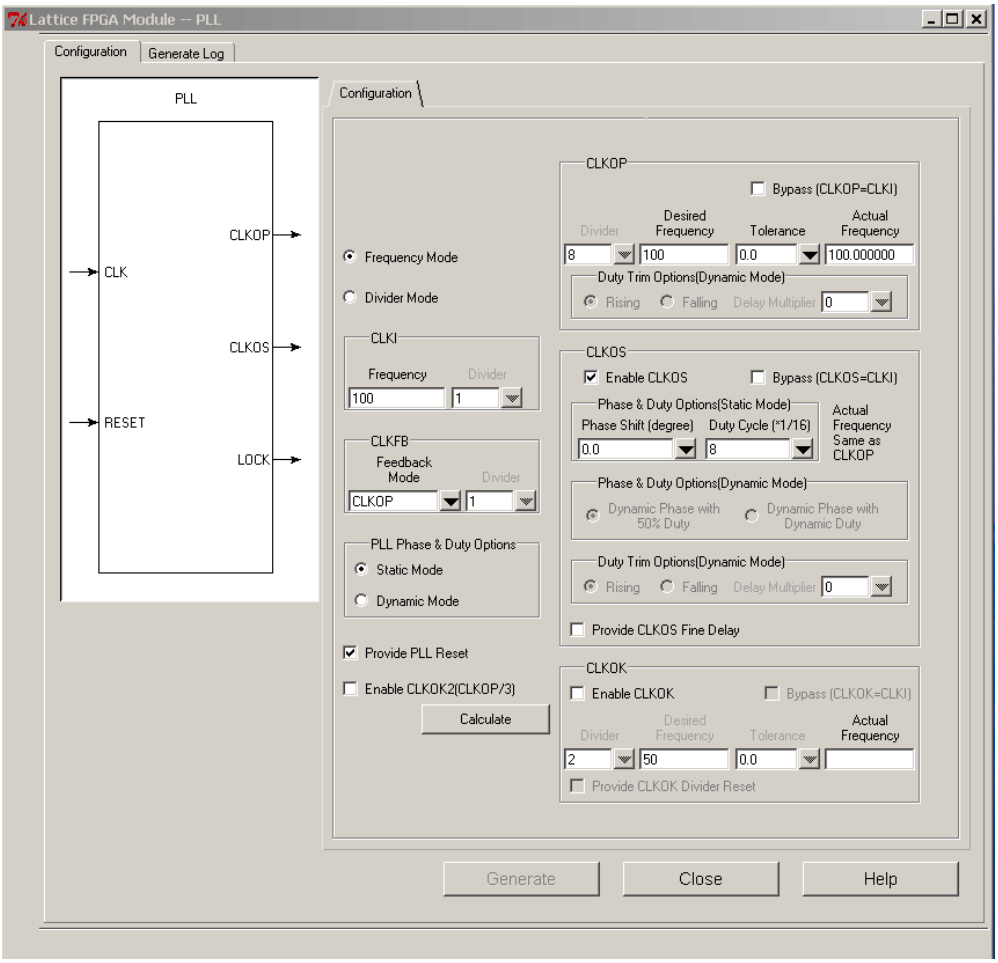


Table 9-7 describes the user parameters in the IPexpress GUI and their usage.

Table 9-7. User Parameters in the IPexpress GUI

User Parameters		Description	Range	Default
Frequency Mode		User desired CLKI and CLKOP frequency	ON/OFF	ON
Divider Mode		User desired CLKI frequency and dividers settings	ON/OFF	OFF
CLKI	Frequency	Input Clock frequency	10 MHz to 435 MHz	100 MHz
	Divider	Input Clock Divider Setting (Divider Mode)	1 to 43	1
CLKFB	Feedback Mode	Feedback Mode	Internal, CLKOP, User Clock	CLKOP
	Divider	Feedback Clock Divider Setting (Divider Mode)	1 to 43	1
PLL Phase & Duty Options	None	No Phase & Duty Options	ON/OFF	ON
	Static Mode	CLKOS Phase/Duty in Static Mode	ON/OFF	OFF
	Dynamic Mode	CLKOS Dynamic Mode Phase/Duty Setting	ON/OFF	OFF
		CLKOS Duty Trimming	ON/OFF	OFF
		CLKOP Duty Trimming	ON/OFF	OFF

Table 9-7. User Parameters in the IPexpress GUI (Continued)

User Parameters		Description	Range	Default
CLKOP	Bypass	Bypass PLL: CLKOP = CLKI	ON/OFF	OFF
	Desired Frequency	User enters desired CLKOP frequency	10 MHz to 435 MHz	100 MHz
	Divider	CLKOP Divider Setting (Divider Mode)	2, 4, 8, 16, 32, 48, 64, 80	8
	Tolerance	CLKOP tolerance users can tolerate	0.0, 0.1, 0.2, 0.5, 0.1, 0.2, 0.5, 1.0	0.0
	Actual Frequency	Actual frequency achievable. Read only	—	—
	Rising	Rising Edge Trim	ON/OFF	OFF
	Falling	Falling Edge Trim	ON/OFF	OFF
	Delay Multiplier	Number of delay steps	0 to 7	0
CLKOS	Enable	Enable CLKOS output clock	ON/OFF	OFF
	Bypass	Bypass PLL: CLKOS = CLKI	ON/OFF	OFF
	Phase Shift	CLKOS Static Phase Shift	0°, 22.5°, 45°..337.5°	0°
	Rising	Rising Edge Trim	ON/OFF	OFF
	Delay Multiplier	Number of Delay steps	0 to 7	0
CLKOK	Enable	Enable CLKOS output clock	ON/OFF	OFF
	Bypass	Bypass PLL: CLKOK = CLKI	ON/OFF	OFF
	Frequency	User enters desired CLKOK frequency	78.125 kHz to 217.5 MHz	50 MHz
	Divider	CLKOK Divider Setting	2 to 128	2
	Tolerance	CLKOK tolerance users can tolerate	0.0, 0.1, 0.2, 0.5, 0.1, 0.2, 0.5, 1.0	0.00
	Actual Frequency	Actual frequency achievable. Read only	—	—
CLKOK2	Enable	Enable CLKOK2 output clock	ON/OFF	OFF
Provide PLL Reset		Provide PLL Reset Port (RESET)	ON/OFF	OFF
Provide CLKOK Divide Reset		Provide CLKOK Reset Port (RSTK)	ON/OFF	OFF
Provide CLKOS Fine Delay Port		Provide CLKOS Fine Delay Port (WRDEL)	ON/OFF	OFF
Import LPC to ispLEVER Project		Import .lpc file to ispLEVER project	ON/OFF	OFF

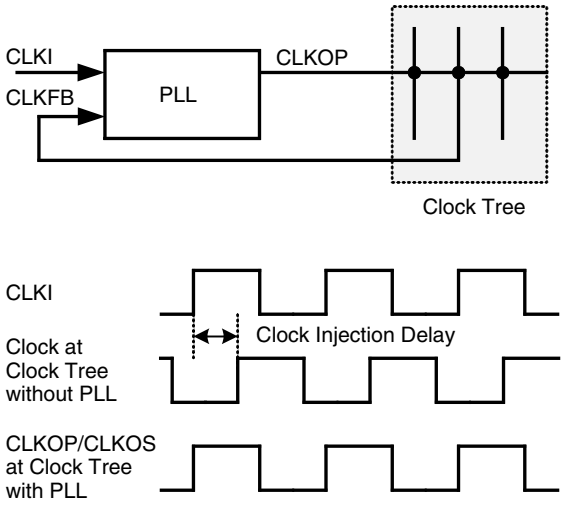
PLL Modes of Operation

PLLs have many uses within a logic design. The two most popular are Clock Injection Removal and Clock Phase Adjustment. These two modes of operation are described below.

PLL Clock Injection Removal

In this mode the PLL is used to reduce clock injection delay. Clock injection delay is the delay from the input pin of the device to a destination element such as a flip-flop. The phase detector of the PLL aligns the CLKI with CLKFB. If the CLKFB signal comes from the clock tree (CLKOP), then the PLL delay and the clock tree delay is removed. Figure 9-10 Illustrates an example block diagram and waveform.

Figure 9-10. Clock Injection Delay Removal Application

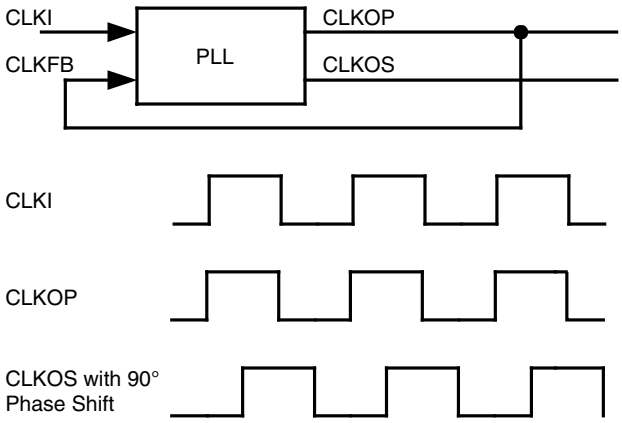


PLL Clock Phase Adjustment

In this mode the PLL is used to create fixed phase relationships in 22.5° increments. Creating fixed phase relationships is useful for forward clock interfaces where a specific relationship between clock and data is required.

The fixed phase relationship can be used between CLKI and CLKOS or between CLKOP and CLKOS.

Figure 9-11. CLKOS Phase Adjustment from CLKOP



IPexpress Output

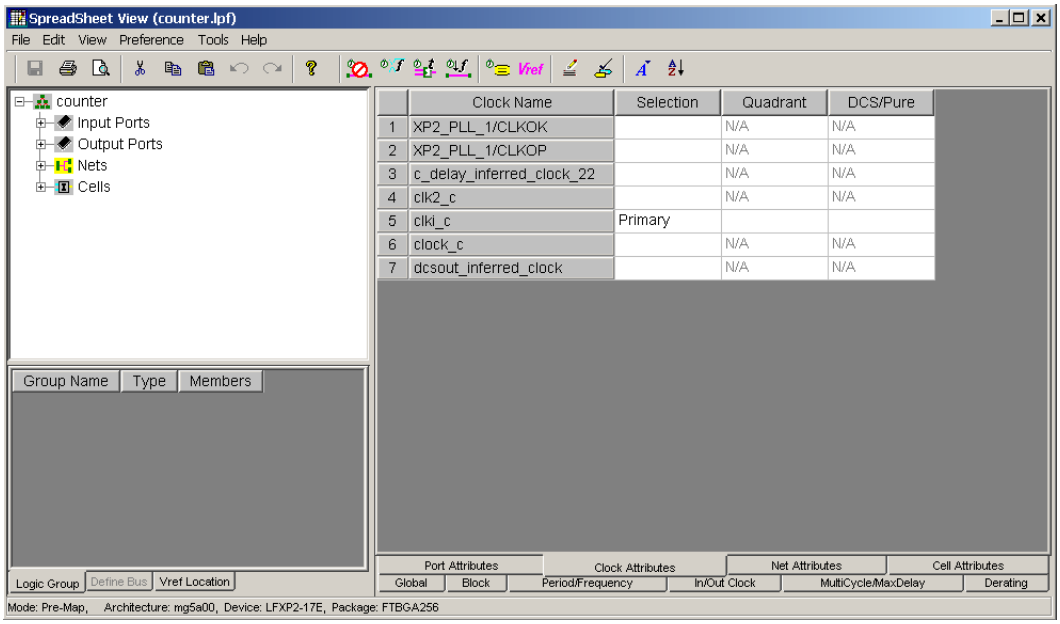
There are two IPexpress outputs that are important for use in the design. The first is the <module_name>.[v|vhd] file. This is the user-named module that was generated by the tool to be used in both synthesis and simulation flows. The second is a template file, <module_name>_tmpl.[v|vhd]. This file contains a sample instantiation of the module. This file is provided for the user to copy/paste the instance and is not intended to be used in the synthesis or simulation flows directly.

For the PLL, IPexpress sets attributes in the HDL module that are specific to the data rate selected. Although these attributes can be easily changed, they should only be modified by re-running the GUI so that the performance of the PLL is maintained. After the map stage in the design flow, FREQUENCY preferences will be included in the preference file to automatically constrain the clocks produced from the PLL.

Use of the Pre-Map Preference Editor

Clock preferences can be set in the Pre-Map Preference Editor. Figure 9-12 shows an example screen shot. The Pre-Map Preference Editor is a part of the ispLEVER Design Planner.

Figure 9-12. Pre-Map Preference Editor Example



Clock Dividers (CLKDIV)

The clock divider divides the high-speed clock by 1, 2, 4 or 8. All the outputs have matched input to output delay. CLKDIV can take as its input the edge clocks and the CLKOP of the PLL. The divided outputs drive the primary clock and are also available for general routing or secondary clocks. The clock dividers are used for providing the low speed FPGA clocks for shift registers (x2, x4, x8) and DDR/SPI4 I/O logic interfaces.

CLKDIV Primitive Definition

Users can instantiate CLKDIV in the source code as defined in this section. Figure 9-13 and Tables 9-8 and 9-9 describe the CLKDIVB definitions.

Figure 9-13. CLKDIV Primitive Symbol

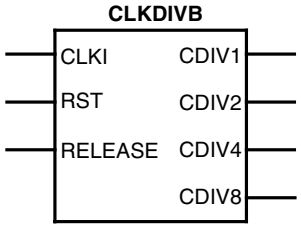


Table 9-8. CLKDIVB Port Definition

Name	Description
CLKI	Clock Input
RST	Reset Input, asynchronously forces all outputs low.
RELEASE	Releases outputs synchronously to input clock.
CDIV1	Divided BY 1 Output
CDIV2	Divided BY 2 Output
CDIV4	Divided BY 4 Output
CDIV8	Divided BY 8 Output

Table 9-9. CLKDIVB Attribute Definition

Name	Description	Value	Default
GSR	GSR Enable	ENABLED/DISABLED	DISABLED

CLKDIV Declaration in VHDL Source Code

```
COMPONENT CLKDIVB
-- synthesis translate_off
      GENERIC (
         GSR : in String);
-- synthesis translate_on
      PORT (
         CLKI,RST, RELEASE:IN    std_logic;
         CDIV1, CDIV2, CDIV4, CDIV8:OUT   std_logic);
END COMPONENT;

      attribute GSR : string;
      attribute GSR of CLKDIVinst0 : label is “DISABLED”;

begin

CLKDIVinst0:         CLKDIVB
-- synthesis translate_off
     GENERIC MAP(
         GSR         => “disabled”
         );
-- synthesis translate_on
      PORT MAP(
         CLKI        => CLKIsig,
         RST         => RSTsig,
         RELEASE     => RELEASEsig,
         CDIV1       => CDIV1sig,
         CDIV2       => CDIV2sig,
         CDIV4       => CDIV4sig,
         CDIV8       => CDIV8sig
         );
```

CLKDIV Usage with Verilog - Example

```
module clkdiv_top(RST,CLKI,RELEASE,CDIV1,CDIV2,CDIV4,CDIV8);

input CLKI,RST,RELEASE;
output CDIV1,CDIV2,CDIV4,CDIV8;

CLKDIVB CLKDIBinst0 (.RST(RST),.CLKI(CLKI),.RELEASE(RELEASE),
     .CDIV1(CDIV1),.CDIV2(CDIV2),.CDIV4(CDIV4),.CDIV8(CDIV8));

defparam CLKDIBint0.GXR = “DISABLED”;

endmodule
```

CLKDIV Example Circuits

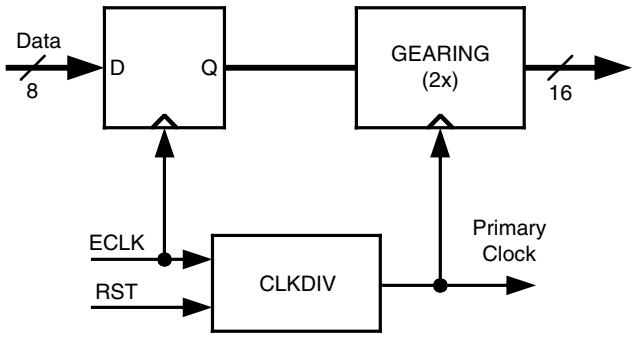
The clock divider (CLKDIV) can divide a clock by 2 or 4 and drives a primary clock network. Clock dividers are useful for providing the low speed FPGA clocks for I/O shift registers (x2, x4) and DDR (x2, x4) I/O logic interfaces. Divide by 8 is provided for slow speed/low power operation.

To guarantee a synchronous transfer in the I/O logic the CLKDIV input clock must come from an edge clock and the output drive from a primary clock. In this case, they are phase matched.

It is especially useful to synchronously reset the I/O logic when Mux/DeMux gearing is used in order to synchronize the entire data bus as shown in Figure 9-14. Using the low skew characteristics of the edge clock routing a reset can be provided to all bits of the data bus to synchronize the Mux/DeMux gearing.

The second circuit shows that a DLL can replace CLKDIV for x2 and x4 applications.

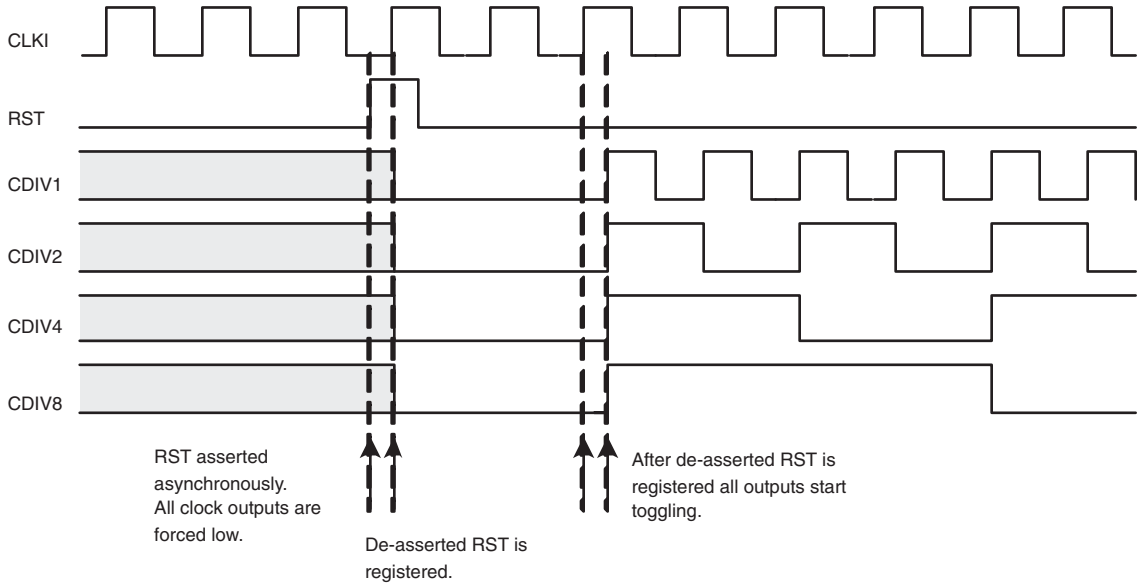
Figure 9-14. CLKDIV Application Example



Reset Behavior

Figure 9-15 illustrates the asynchronous RST behavior.

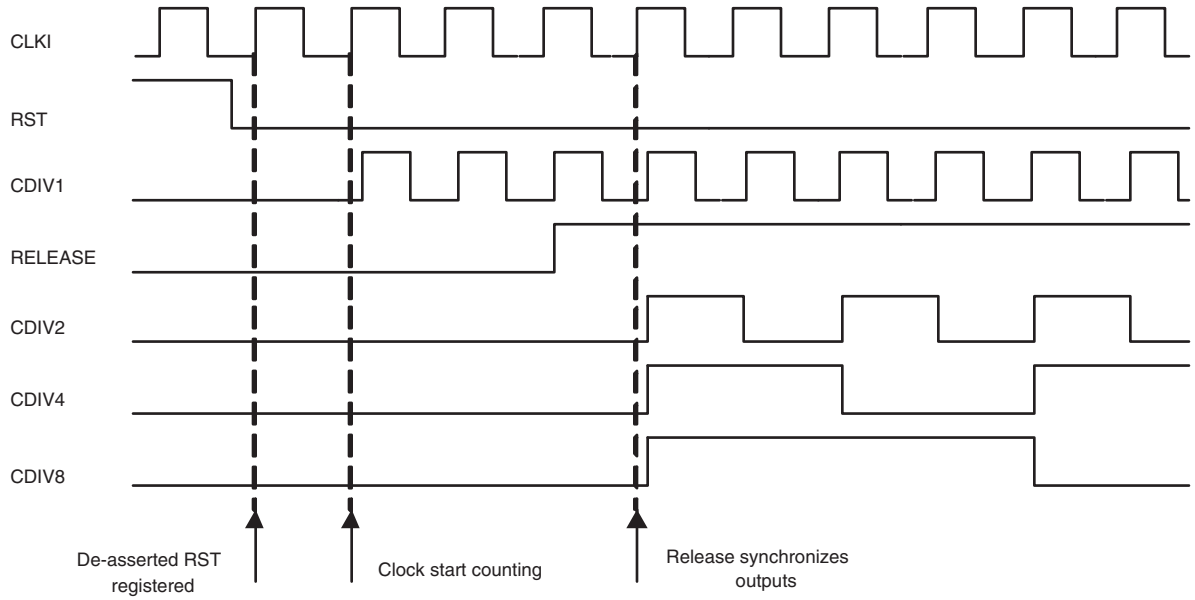
Figure 9-15. CLKDIV Reset Behavior



Release Behavior

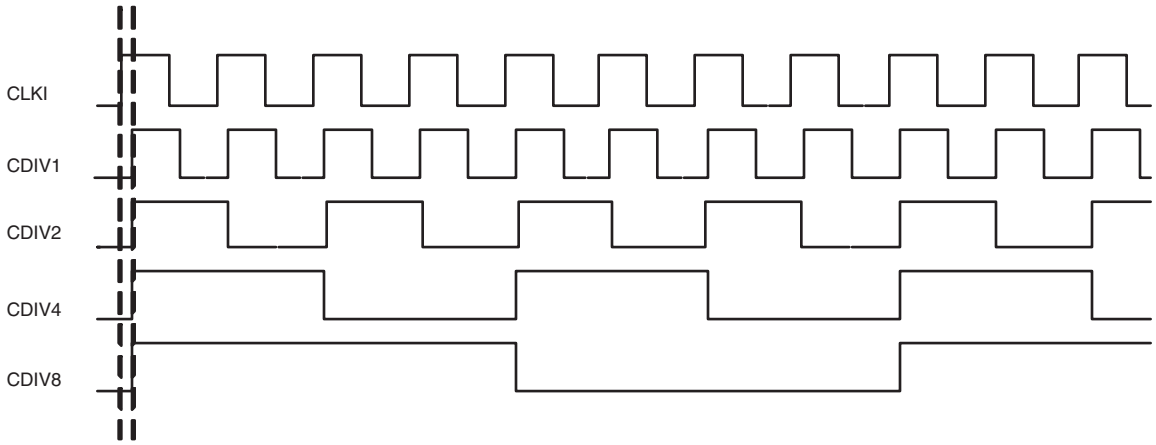
The port, “Release” is used to synchronize the all outputs after RST is de-asserted. Figure 9-16 Illustrates the release behavior.

Figure 9-16. CLKDIV Release Behavior



CLKDIV Inputs-to-Outputs Delay Matching

Figure 9-17. CLKDIV Inputs-to-Outputs Delay Matching

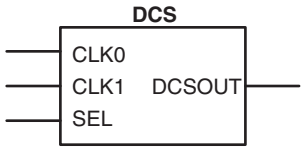


DCS (Dynamic Clock Select)

DCS is a global clock buffer incorporating a smart multiplexer function that takes two independent input clock sources and avoids glitches or runt pulses on the output clock, regardless of where the enable signal is toggled. There are two DCSs for each quadrant.

The outputs of the DCS then reach primary clock distribution via the feedlines. Figure 9-18 shows the block diagram of the DCS.

Figure 9-18. DCS Primitive Symbol



DCS Primitive Definition

Table 9-10 defines the I/O ports of the DCS block. There are eight modes to select from. Table 9-11 describes how each mode is configured.

Table 9-10. DCS I/O Definition

I/O	Name	Description
Input	SEL	Input Clock Select
	CLK0	Clock input 0
	CLK1	Clock Input 1
Output	DCSOUT	Clock Output

Table 9-11. DCS Modes of Operation

Attribute Name	Description	Output		Value
		SEL=0	SEL=1	
DCS MODE	Rising edge triggered, latched state is high	CLK0	CLK1	POS
	Falling edge triggered, latched state is low	CLK0	CLK1	NEG
	Sel is active high, Disabled output is low	0	CLK1	HIGH_LOW
	Sel is active high, Disabled output is high	1	CLK1	HIGH_HIGH
	Sel is active low, Disabled output is low	CLK0	0	LOW_LOW
	Sel is active low, Disabled output is high	CLK0	1	LOW_HIGH
	Buffer for CLK0	CLK0	CLK0	CLK0
	Buffer for CLK1	CLK1	CLK1	CLK1

DCS Timing Diagrams

Each mode performs a unique operation. The clock output timing is determined by input clocks and the edge of the SEL signal. Figure 9-19 describes the timing of each mode.

Figure 9-19. Timing Diagrams by DCS MODE

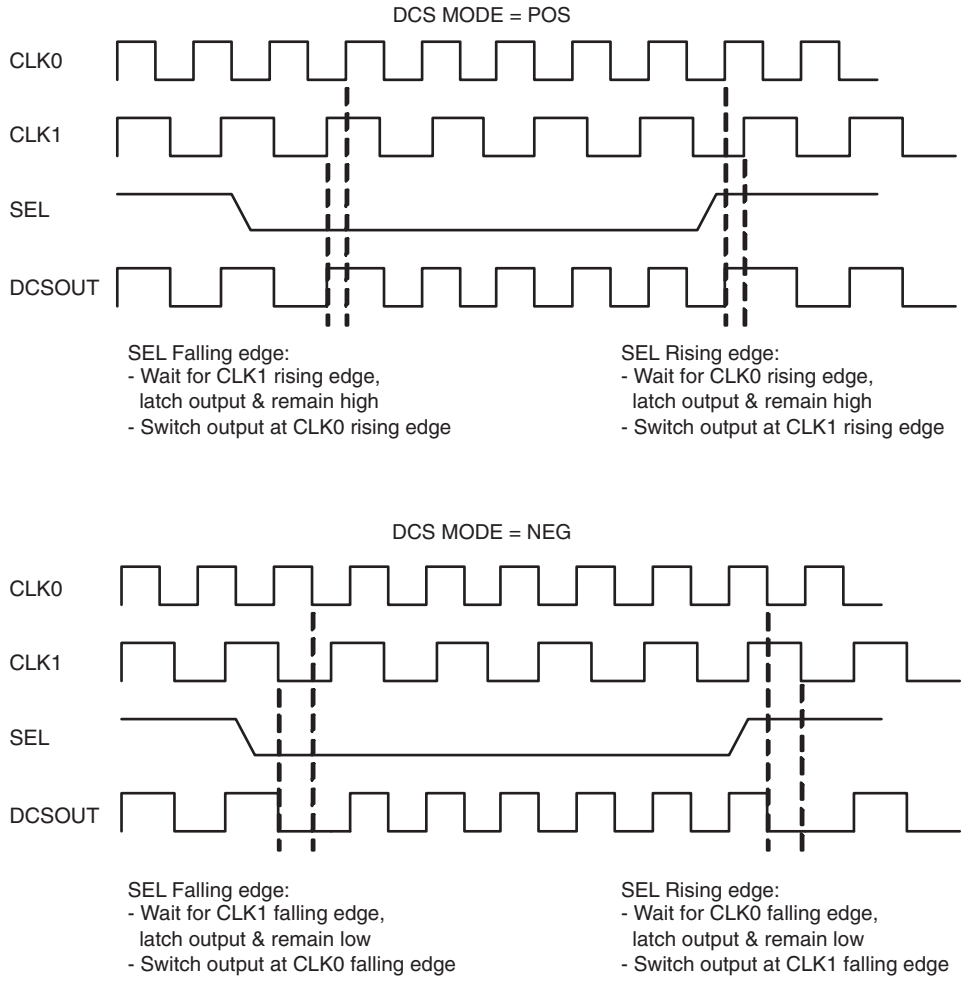
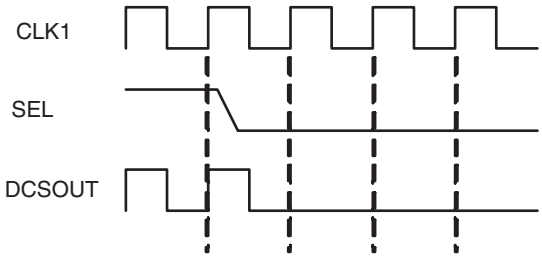


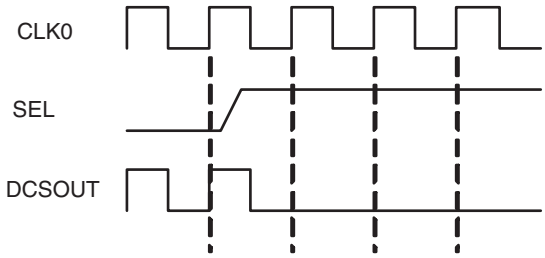
Figure 9-20. Timing Diagrams by DCS MODE (Cont.)

DCS MODE = HIGH_LOW



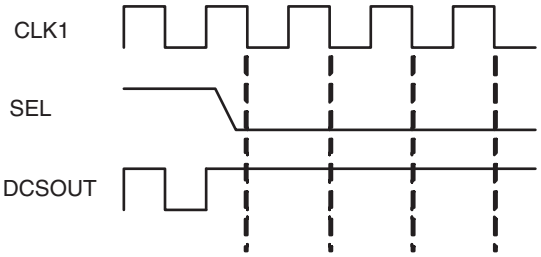
- Switch low @CLK1 falling edge.
- If SEL is low, output stays low at on CLK1 rising edge. SEL must not change during setup prior to rising clock.

DCS MODE = LOW_LOW



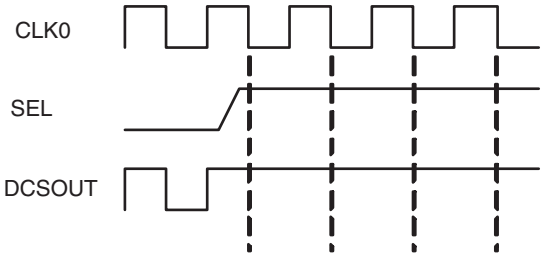
- Switch low @CLK0 falling edge.
- If SEL is high, output stays low at on CLK0 rising edge.

DCS MODE = HIGH_HIGH



- Switch high @CLK1 rising edge.
- If SEL is low, output stays low high on CLK1 falling edge.

DCS MODE = LOW_HIGH



- Switch high @ CLK0 rising edge.
- If SEL is high, output stays high on CLK0 falling edge.

DCS Usage with VHDL - Example

```
COMPONENT DCS
-- synthesis translate_off
        GENERIC   (
                  DCSMODE : string :=  “POS”
                  ;
-- synthesis translate_on

        PORT      (

END COMPONENT;

    attribute DCSMODE : string;
    attribute DCSMODE of DCSinst0 : label is “POS”;

begin

DCSInst0: DCS
-- synthesis translate_off

        GENERIC MAP (
```

```
        DCSMODE  => “POS”
)
-- synthesis translate_on

        PORT MAP (
                SEL          =>  clksel,
                CLK0         =>  dcsclk0,
                CLK1         =>  sysclk1,
                DCSOUT       =>  dcsclk
                );
```

DCS Usage with Verilog - Example

```
module dcs(clk0,clk1,sel,dcsout);

input clk0, clk1, sel;
output dcsout;

DCS DCSInst0 (.SEL(sel),.CLK0(clk0),.CLK1(clk1),.DCSOUT(dcsout));
defparam DCSInst0.DCSMODE = “CLK0”;

endmodule
```

Oscillator (OSCE)

There is a dedicated oscillator in the LatticeXP2 device whose output is made available for users.

The oscillator frequency output is routed through a divider which is used as an input clock to the clock tree. The available outputs of the divider are shown in Table 9-13. The oscillator frequency output can be further divided by internal logic (user logic) for lower frequencies, if desired. The oscillator is powered down when not in use.

The output of this oscillator is not a precision clock. It is intended as an extra clock that does not require accurate clocking.

Primitive Name: OSCE

Table 9-12. OSCE Port Definition

I/O	Name	Description
Output	OSC	Oscillator Clock Output

Table 9-13. OSCE Attribute Definition

User Attribute	Attribute Name	Value (MHz)	Default Value
Nominal Frequency	NORM_FREQ	2.5, 3.14, 4.3, 5.4, 6.9, 8.1, 9.2, 10, 13, 15, 20, 26, 32, 40, 54, 80, 163	2.5

OSC Primitive Symbol (OSCE)

Figure 9-21. OSC Symbol

OSC Usage with VHDL - Example

```
COMPONENT OSCE

      PORT (OSC:OUT   std_logic);

END COMPONENT;
begin
OSCInst0: OSCE
      PORT MAP ( OSC=>  osc_int);
```

OSC Usage with Verilog - Example

```
module OSC_TOP(OSC_CLK);

output OSC_CLK;

OSCE OSCinst0 (.OSC(OSC_CLK));

endmodule
```

Setting Clock Preferences

Designers can use clock preferences to implement clocks to the desired performance. Preferences can be set in the Pre-Map Preference Editor (Design Planner) or in preference files. Frequently used preferences are described in Appendix C.

Power Supplies

Each PLL has its own power supply pin, VCCPLL. Since VCC and VCCPLL are normally the same 3.3V, it is recommended that they are driven from the same power supply on the circuit board, thus minimizing leakage. In addition, each of these supplies should be independently isolated from the main 3.3V supply on the board using proper board filtering techniques to minimize the noise coupling between them.

Technical Support Assistance

Hotline: 1-800-LATTICE (North America)
+1-503-268-8001 (Outside North America)
e-mail: techsupport@latticesemi.com
Internet: www.latticesemi.com

Revision History

Date	Version	Change Summary
February 2007	01.0	Initial release.

Appendix A. Primary Clock Sources and Distribution

Figure 9-22. LatticeXP2 Primary Clock Sources and Distribution

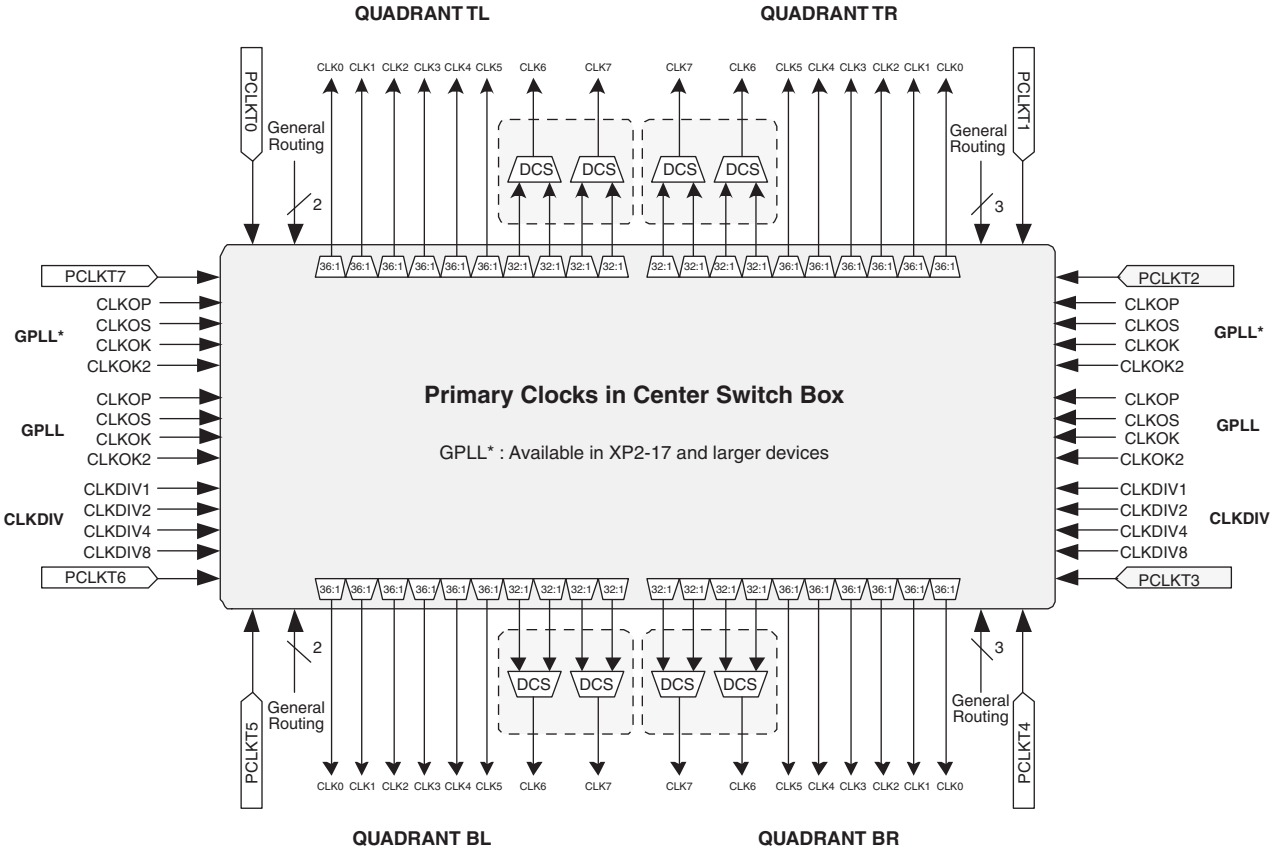
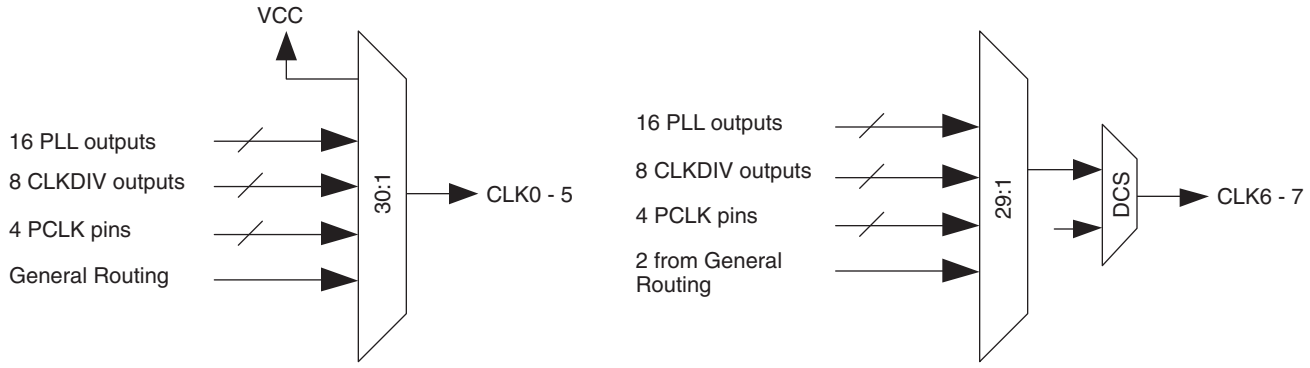


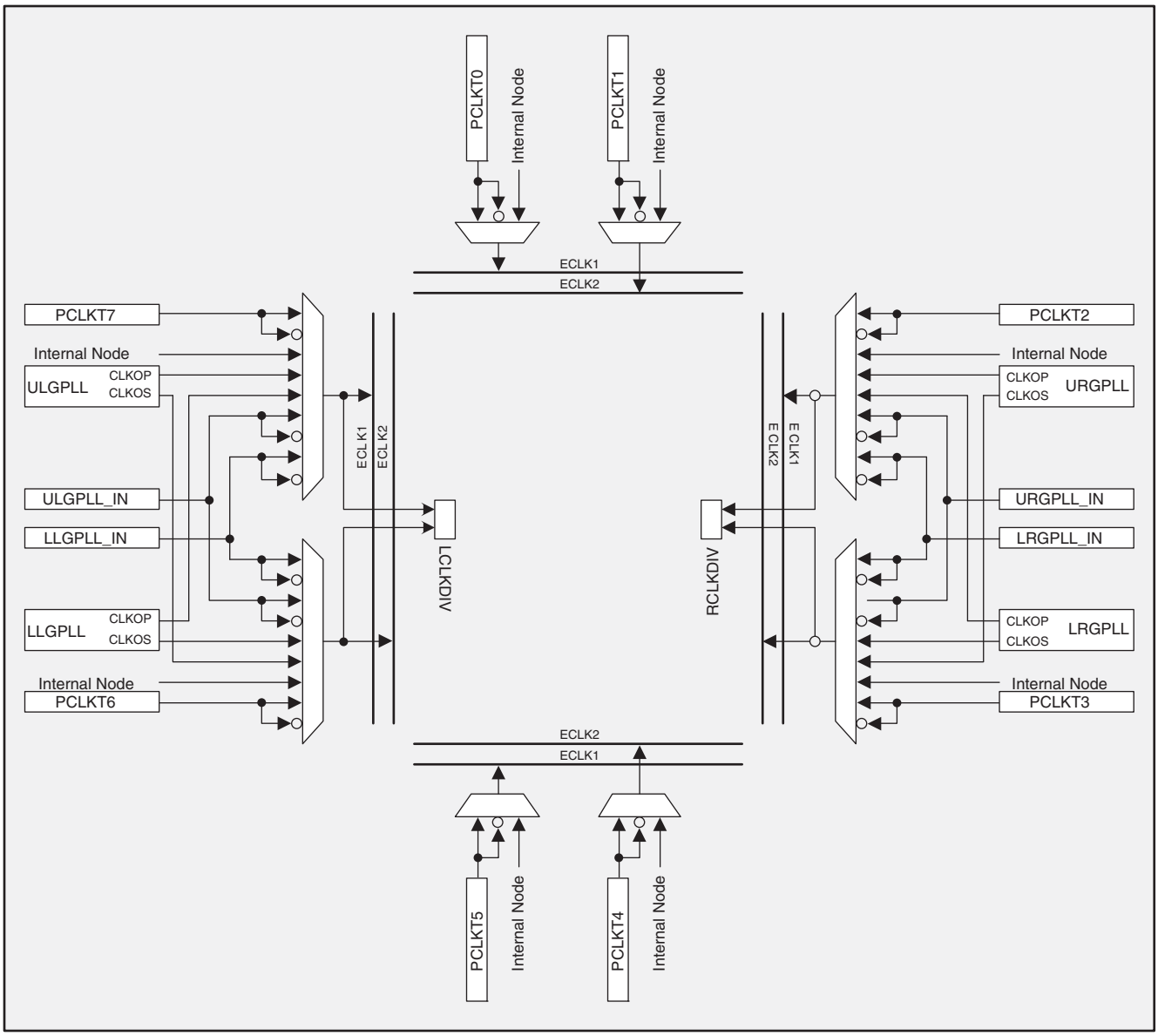
Figure 9-23. LatticeXP2 Primary Clock Muxes



Appendix B. PLL, CLKIDV and ECLK Locations and Connectivity

Figure 9-24 shows the locations, site names and connectivity of the PLLs, CLKDIVs and ECLKs

Figure 9-24. PLL, CLKIDV and ECLK Locations and Connectivity



Appendix C. Clock Preferences

A few key clock preferences are introduced below. Refer to the 'Help' file for other preferences and detailed information.

ASIC

The following preference command assigns a phase of 90 degrees to the CIMDLLA CLKOP.

```
ASIC "my_dll" TYPE "CIMDLLA" CLKOP_PHASE=90;
```

FREQUENCY

The following physical preference command assigns a frequency of 100 MHz to a net named clk1:

```
FREQUENCY NET "clk1" 100 MHz;
```

The following preference specifies a hold margin value for each clock domain:

```
FREQUENCY NET "RX_CLKA_CMOS_c" 100.000 MHz HOLD_MARGIN 1 ns;
```

MAXSKEW

The following command assigns a maximum skew of 5 nanoseconds to a net named NetB:

```
MAXSKEW NET "NetB" 5 NS;
```

MULTICYCLE

The following command will relax the period to 50 nanoseconds for the path starting at COMPA to COMPB (NET1):

```
MULTICYCLE "PATH1" START COMP "COMPA" END COMP "COMPB" NET "NET1" 50 NS ;
```

PERIOD

The following command assigns a clock period of 30 nanoseconds to the port named Clk1:

```
PERIOD PORT "Clk1" 30 NS;
```

PROHIBIT

This command prohibits the use of a primary clock to route a clock net named bf_clk:

```
PROHIBIT PRIMARY NET "bf_clk";
```

USE PRIMARY

Use a primary clock resource to route the specified net:

```
USE PRIMARY NET clk_fast;

USE PRIMARY DCS NET "bf_clk";

USE PRIMARY PURE NET "bf_clk" QUADRANT_TL;
```

USE SECONDARY

Use a secondary clock resource to route the specified net:

```
USE SECONDARY NET "clk_lessfast" QUADRANT_TL;
```

USE EDGE

Use a edge clock resource to route the specified net:

```
USE EDGE NET "clk_fast";
```

CLOCK_TO_OUT

Specifies a maximum allowable output delay relative to a clock.

Here are two preferences using both the CLKPORT and CLKNET keywords showing the corresponding scope of TRACE reporting.

The CLKNET will stop tracing the path before the PLL, so you will not get PLL compensation timing numbers.

```
CLOCK_TO_OUT PORT "RxAddr_0" 6.000000 ns CLKNET "pll_rxclk" ;
```

The above preference will yield the following clock path:

```
Clock path pll_inst/pll_utp_0_0 to PFU_33:

Name       Fanout    Delay (ns)  Site           Resource
ROUTE      49        2.892       ULPPLL.MCLK to  R3C14.CLK0 pll_rxclk
          --------
           2.892   (0.0% logic, 100.0% route), 0 logic levels.
```

If CLKPORT is used, the trace is complete back to the clock port resource and provides PLL compensation timing numbers.

```
CLOCK_TO_OUT PORT "RxAddr_0" 6.000000 ns CLKPORT "RxClk" ;
```

The above preference will yield the following clock path:

```
Clock path RxClk to PFU_33:

Name       Fanout    Delay (ns)  Site           Resource
IN_DEL     ---       1.431       D5.PAD to      D5.INCK RxClk
ROUTE      1         0.843       D5.INCK to     ULPPLL.CLKIN RxClk_c
MCLK_DEL   ---       3.605       ULPPLL.CLKIN to ULPPLL.MCLK pll_inst/pll_utp_0_0
ROUTE      49        2.892       ULPPLL.MCLK to  R3C14.CLK0 pll_rxclk
        --------
          8.771   (57.4% logic, 42.6% route), 2 logic levels.
```

INPUT_SETUP

Specifies an setup time requirement for input ports relative to a clock net.

```
INPUT_SETUP PORT "datain" 2.000000 ns HOLD 1.000000 ns CLKPORT "clk"
PLL_PHASE_BACK ;
```

PLL_PHASE_BACK

This preference is used with INPUT_SETUP when a user needs a trace calculation based on the previous clock edge.

This preference is useful when setting the PLL output phase adjustment. Since there is no negative phase adjustment provided, the PLL_PHASE_BACK preference works as if negative phase adjustment is available.

For example:

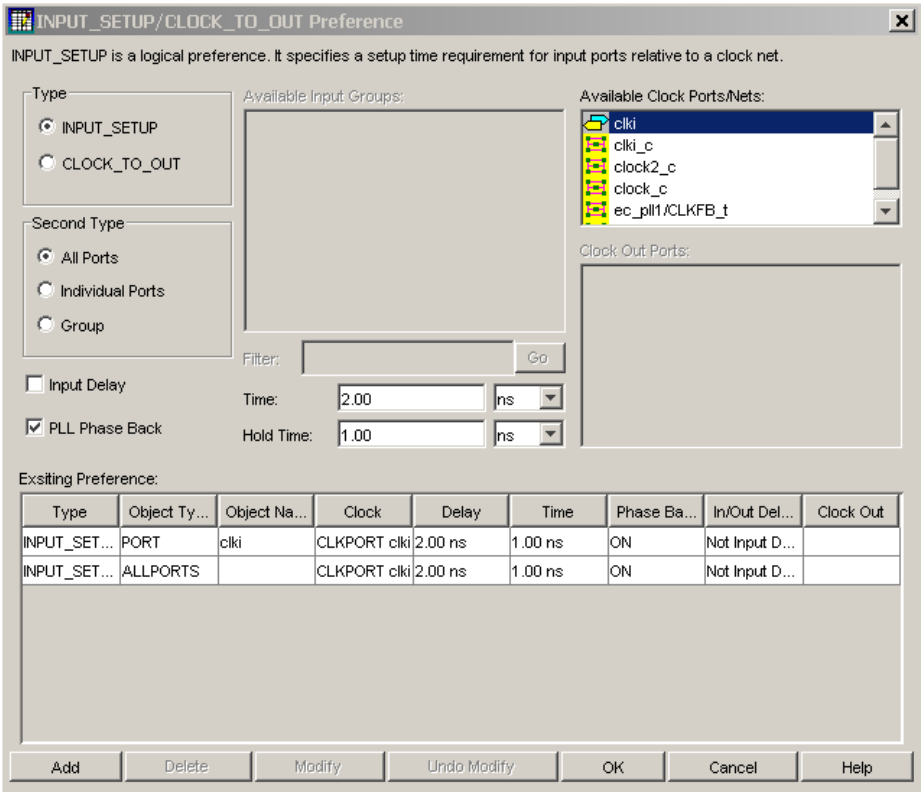
If phase adjustment of -90° of CLKOS is desired, a user can set the Phase to 270° and set the INPUT_SETUP preference with PLL_PHASE_BACK.

PLL_PHASE_BACK Usage in Pre-Map Preference Editor
The Pre-Map Preference Editor can be used to set the PLL_PHASE_BACK attribute.

1. Open the Design Planner (Pre-Map).
2. In the Design Planner control window, select **View -> Spreadsheet View**.
3. In the Spreadsheet View window, select **Input_setup/Clock_to_out...**

The INPUT_SETUP/CLOCK_TO_OUT Preference window is shown in Figure 9-25.

Figure 9-25. INPUT_SETUP/CLOCK_TO_OUT Preference Window



LatticeXP2 Memory Usage Guide

January 2008 Technical Note TN1137

Introduction

This technical note discusses memory usage for the LatticeXP2™ device family. It is intended to be used by design engineers as a guide for integrating the User TAG, EBR- (Embedded Block RAM) and PFU-based memories in this device family using the ispLEVER® design tool.

The architecture of these devices provides resources for FPGA on-chip memory applications. The sysMEM™ EBR complements the distributed PFU-based memory. Single-Port RAM, Dual-Port RAM, Pseudo Dual-Port RAM, FIFO and ROM memories can be constructed using the EBR. LUTs and PFU can implement Distributed Single-Port RAM, Dual-Port RAM and ROM. User TAG memories in varying sizes, depending on the specific chip, are also on the device.

The capabilities of the User TAG memory, EBR RAM and PFU RAM are referred to as primitives and are described later in this document. Designers can utilize the memory primitives in two ways via the IPexpress™ tool in the ispLEVER software. The IPexpress GUI allows users to specify the memory type and size required. IPexpress takes this specification and constructs a netlist to implement the desired memory by using one or more of the memory primitives.

The remainder of this document discusses the use of IPexpress, memory modules and memory primitives.

Memories in LatticeXP2 Devices

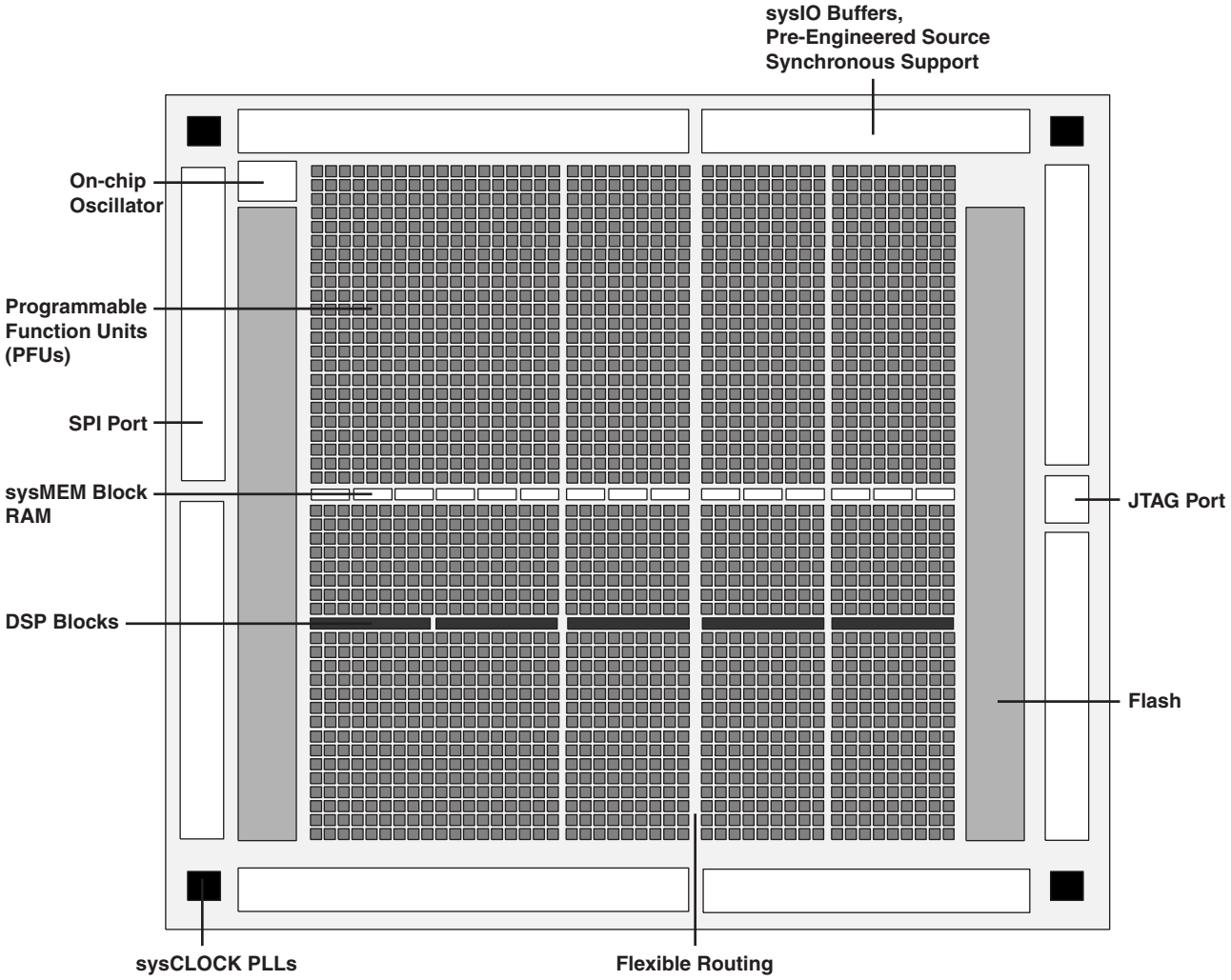
There are two kinds of logic blocks, the Programmable Functional Unit (PFU) and Programmable Functional Unit without RAM (PFF). The PFU contains the building blocks for logic, arithmetic, RAM, ROM and register functions. The PFF block contains building blocks for logic, arithmetic and ROM functions. Both PFU and PFF blocks are optimized for flexibility allowing complex designs to be implemented quickly and efficiently. Logic Blocks are arranged in a two-dimensional array. Only one type of block is used per row.

The LatticeXP2 family of devices contains up to two rows of sysMEM EBR blocks. sysMEM EBRs are large, dedicated 18K fast memory blocks. Each sysMEM block can be configured in a variety of depths and widths of RAM or ROM. Each LatticeXP2 device also contains one dedicated row of User TAG memory with up to 451 bytes of space.

Table 10-1. LatticeXP2 LUT and Memory Densities

Parameter	XP2-5	XP2-8	XP2-17	XP2-30	XP2-40
EBR Rows	1	1	1	1	2
EBR Blocks	9	12	15	21	48
EBR Bits	165888	221184	276480	387072	884736
Distributed RAM Bits	10368	18432	34560	64512	82944
Total Memory Bits	176256	239616	311040	451584	967680

Figure 10-1. Simplified Block Diagram, LatticeXP2 Device (Top Level)



Utilizing IPexpress

Designers can utilize IPexpress to easily specify a variety of memories in their designs. These modules are constructed using one or more memory primitives along with general purpose routing and LUTs, as required. The available primitives are:

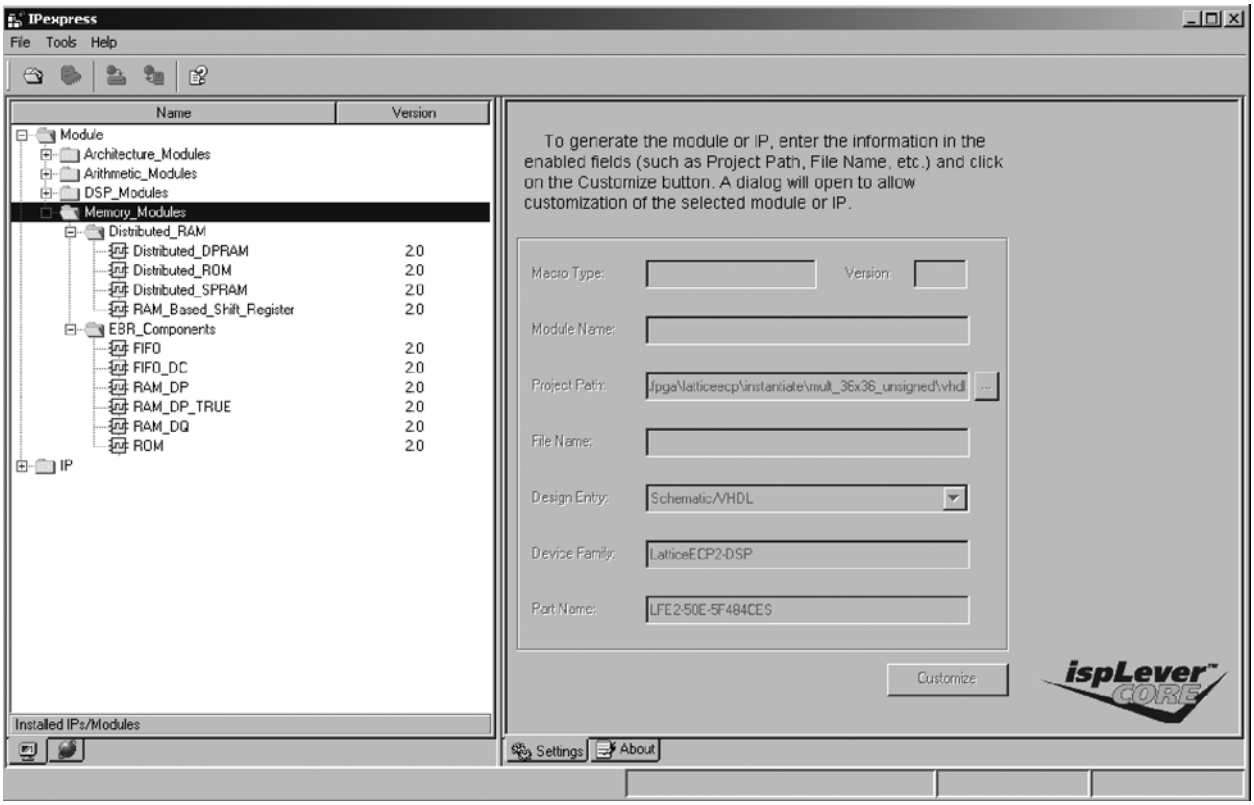
- Single Port RAM (RAM_DQ) – EBR-based
- Dual PORT RAM (RAM_DP_TRUE) – EBR-based
- Pseudo Dual Port RAM (RAM_DP) – EBR-based
- Read Only Memory (ROM) – EBR-Based
- First In First Out Memory (Dual Clock) (FIFO_DC) – EBR-based
- Distributed Single Port RAM (Distributed_SPRAM) – PFU-based
- Distributed Dual Port RAM (Distributed_DPRAM) – PFU-based
- Distributed ROM (Distributed_ROM) – PFU/PFF-based
- User TAG memory (SSPIA) – TAG-based

IPexpress Flow

For generating any of these memories, create (or open) a project for the LatticeXP2 devices.

From the Project Navigator, select **Tools > IPexpress** or click on the button in the toolbar when LatticeXP2 devices are targeted in the project. This opens the IPexpress main window as shown in Figure 10-2.

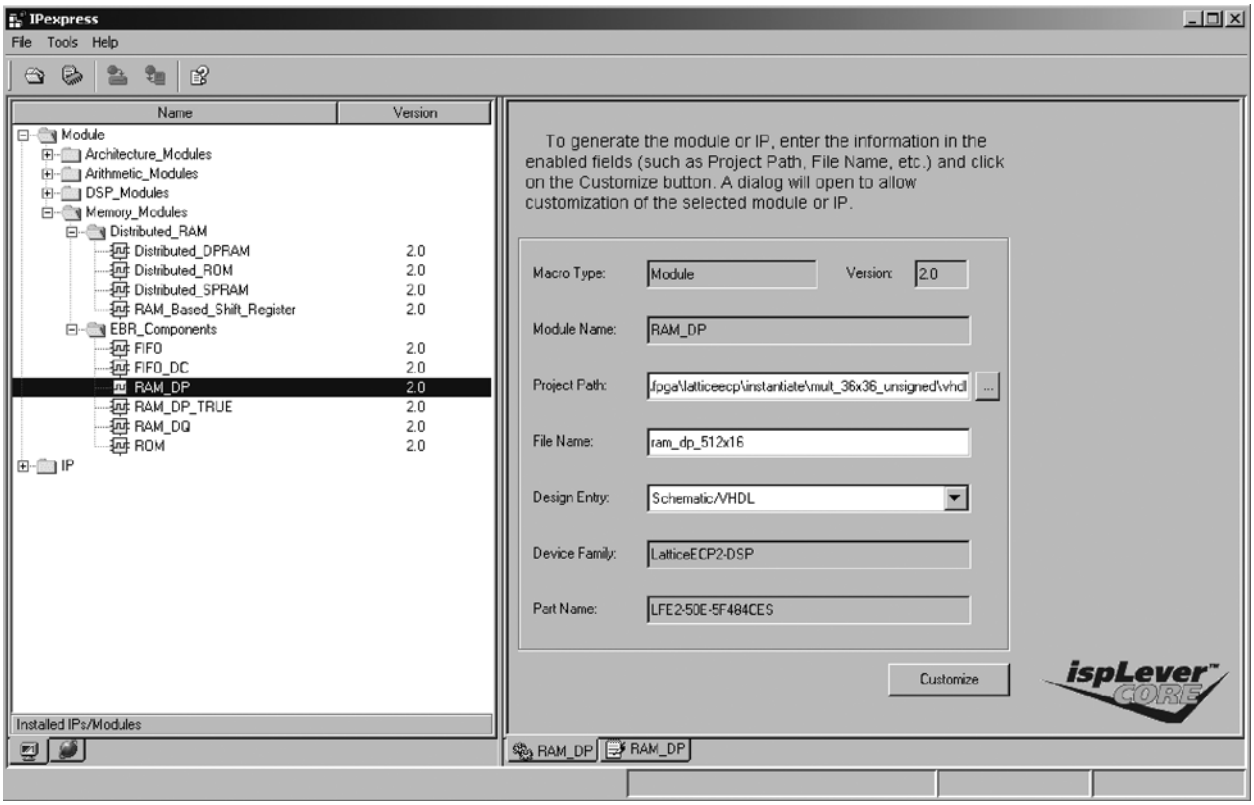
Figure 10-2. IPExpress - Main Window



The left pane of this window includes the Module Tree. The EBR-based Memory Modules are under the **EBR_Components** and the PFU-based Distributed Memory Modules are under **Storage_Components**, as shown in Figure 10-2.

As an example, let us consider generating an EBR-based Pseudo Dual Port RAM of size 512x16. Select **RAM_DP** under **EBR_Components**. The right pane changes as shown in Figure 10-3.

Figure 10-3. Example Generating Pseudo Dual Port RAM (RAM_DP) Using IPexpress



In the right pane, options like the **Device Family**, **Macro Type**, **Category**, and **Module_Name** are device and selected module dependent. These cannot be changed in IPexpress.

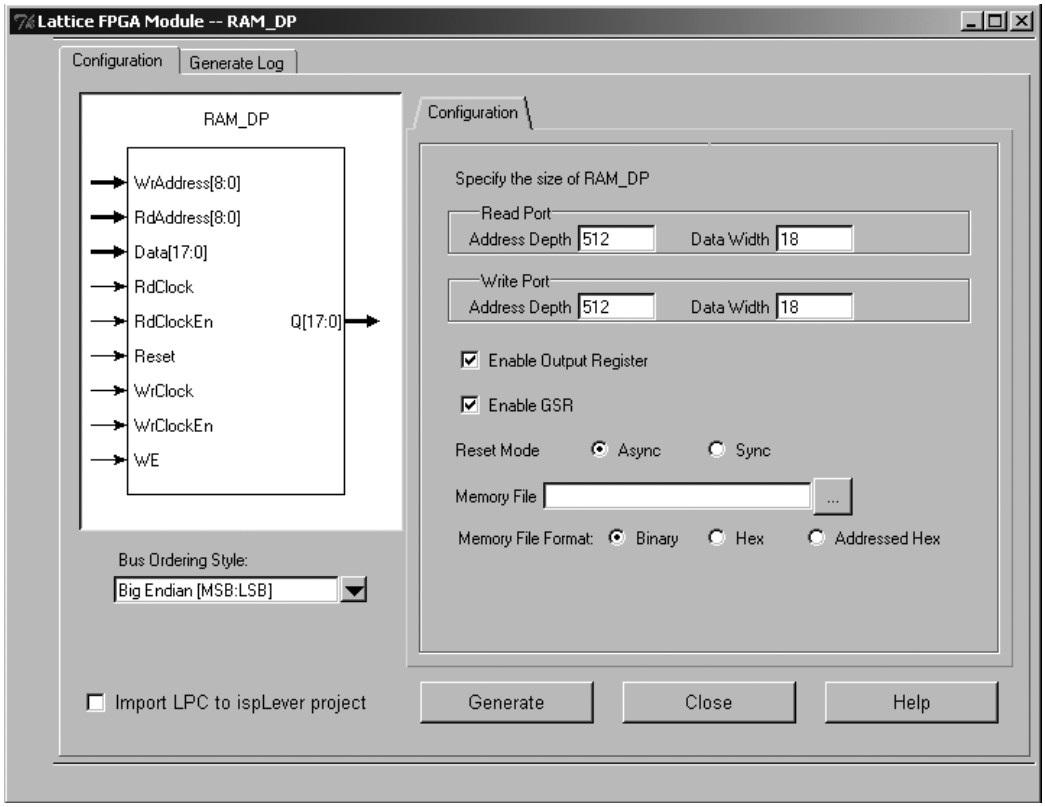
Users can change the directory where the generated module files will be placed by clicking the **Browse** button in the **Project Path**.

The **Module Name** text box allows users to specify an entity name for the module they are about to generate. Users must provide this entity name.

Design entry, Verilog or VHDL, by default, is the same as the project type. If the project is a VHDL project, the selected design entry option will be “Schematic/ VHDL”, and “Schematic/ Verilog-HDL” if the project type is Verilog-HDL.

The **Device** pull-down menu allows users to select different devices within the same family, LatticeXP2 in this example. By clicking the **Customize** button, another window opens where users can customize the RAM (Figure 10-4).

Figure 10-4. Example Generating Pseudo Dual Port RAM (RAM_DP) Module Customization



The left side of this window shows the block diagram of the module. The right side includes the **Configuration** tab where users can choose options to customize the RAM_DP (e.g. specify the address port sizes and data widths).

Users can specify the address depth and data width for the **Read Port** and the **Write Port** in the text boxes provided. In this example, we are generating a Pseudo Dual Port RAM of size 512 x 16. Users can also create RAMs of different port widths for Pseudo Dual Port and True Dual Port RAMs.

The Input Data and the Address Control are always registered, as the hardware only supports the clocked write operation for the EBR based RAMs. The check box **Enable Output Registers** inserts the output registers in the Read Data Port. Output registers are optional for EBR-based RAMs.

Users have the option to set the **Reset Mode** as Asynchronous Reset or Synchronous Reset. **Enable GSR** can be checked to enable the Global Set Reset.

Users can also pre-initialize their memory with the contents specified in the **Memory File**. It is optional to provide this file in the RAM; however for ROM, the Memory File is required. These files can be of Binary, Hex or Addresses Hex format. The details of these formats are discussed in the Initialization File section of this document.

At this point, users can click the **Generate** button to generate the module they have customized. A VHDL or Verilog netlist is then generated and placed in the specified location. Users can incorporate this netlist in their designs.

Another important button is the **Load Parameters** button. IPexpress stores the parameters specified in a <module_name>.lpc file. This file is generated along with the module. Users can click on the Load Parameters button to load the parameters of a previously generated module to re-visit or make changes to them.

Once the module is generated, users can either instantiate the *.lpc or the Verilog-HDL/ VHDL file in top-level module of their design.

The various memory modules, both EBR and distributed, are discussed in detail in this document.

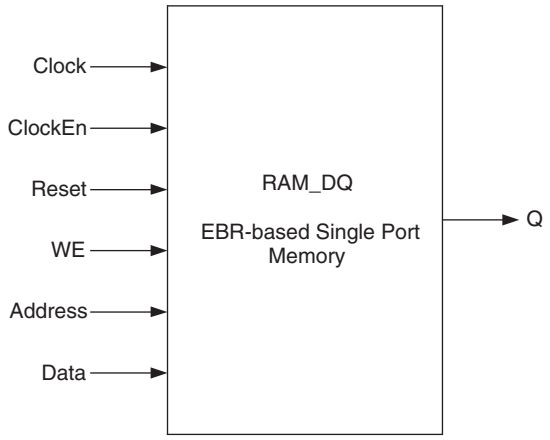
Memory Modules

Single Port RAM (RAM_DQ) – EBR Based

The EBR blocks in LatticeXP2 devices can be configured as Single Port RAM or RAM_DQ. IPexpress allows users to generate the Verilog-HDL or VHDL along EDIF netlist for the memory size as per design requirements.

IPexpress generates the memory module as shown in Figure 10-5.

Figure 10-5. Single Port Memory Module Generated by IPexpress



Since the device has a number of EBR blocks, the generated module makes use of these EBR blocks, or primitives, and cascades them to create the memory sizes specified by the user in the IPexpress GUI. For memory sizes smaller than an EBR block, the module will be created in one EBR block. For memory sizes larger than one EBR block, multiple EBR blocks can be cascaded in depth or width (as required to create these sizes).

In Single Port RAM mode, the input data and address for the ports are registered at the input of the memory array. The output data of the memory is optionally registered at the output.

The various ports and their definitions for the Single Port Memory are listed in Table 10-2. The table lists the corresponding ports for the module generated by IPexpress and for the EBR RAM_DQ primitive.

Table 10-2. EBR-based Single Port Memory Port Definitions

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

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

Chip Select (CS) is a useful port in the EBR primitive when multiple cascaded EBR blocks are required by the memory. The CS signal forms the MSB for the address when multiple EBR blocks are cascaded. CS is a 3-bit bus,

so it can cascade eight memories easily. If the memory size specified by the user requires more than eight EBR blocks, the ispLEVER software automatically generates the additional address decoding logic, which is implemented in the PFU (external to the EBR blocks).

Each EBR block consists of 18,432 bits of RAM. The values for x (address) and y (data) for each EBR block for the devices are listed in Table 10-3.

Table 10-3. Single Port Memory Sizes for 16K Memories for LatticeXP2

Single Port Memory Size	Input Data	Output Data	Address [MSB:LSB]
16K x 1	DI	DO	AD[13:0]
8K x 2	DI[1:0]	DO[1:0]	AD[12:0]
4K x 4	DI[3:0]	DO[3:0]	AD[11:0]
2K x 9	DI[8:0]	DO[8:0]	AD[10:0]
1K x 18	DI[17:0]	DO[17:0]	AD[9:0]
512 x 36	DI[35:0]	DO[35:0]	AD[8:0]

Table 10-4 shows the various attributes available for the Single Port Memory (RAM_DQ). Some of these attributes are user-selectable through the IPexpress GUI. For detailed attribute definitions, refer to Appendix A.

Table 10-4. Single Port RAM Attributes for LatticeXP2

Attribute	Description	Values	Default Value	User Selectable Through IPexpress
Address depth	Address Depth Read Port	16K, 8K, 4K, 2K, 1K, 512		YES
Data Width	Data Word Width Read Port	1, 2, 4, 9, 18, 36	1	YES
Enable Output Registers	Register Mode (Pipelining) for Write Port	NOREG, OUTREG	NOREG	YES
Enable GSR	Enables Global Set Reset	ENABLE, DISABLE	ENABLE	YES
Reset Mode	Selects the Reset type	ASYNC, SYNC	ASYNC	YES
Memory File Format		BINARY, HEX, ADDRESSED HEX		YES
Write Mode	Read / Write Mode for Write Port	NORMAL, WRITETHROUGH, READ-BEFOREWRITE	NORMAL	YES
Chip Select Decode	Chip Select Decode for Read Port	0b000, 0b001, 0b010, 0b011, 0b100, 0b101, 0b110, 0b111	0b000	NO
Init Value	Initialization value	0x00000000000000000000 00000000000000000000 00000000000000000000 0000000000000.....0xFFFF FFFFFFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFFFFFF FFFFFFFFFFFFF	0x000000000 00000000000 00000000000 00000000000 00000000000 00000000000 00000000000 00000	NO

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

Additionally, users can select to enable the output registers for RAM_DQ. Figures 10-6-10-11 show the internal timing waveforms for the Single Port RAM (RAM_DQ) with these options.

Figure 10-6. Single Port RAM Timing Waveform - NORMAL Mode, without Output Registers

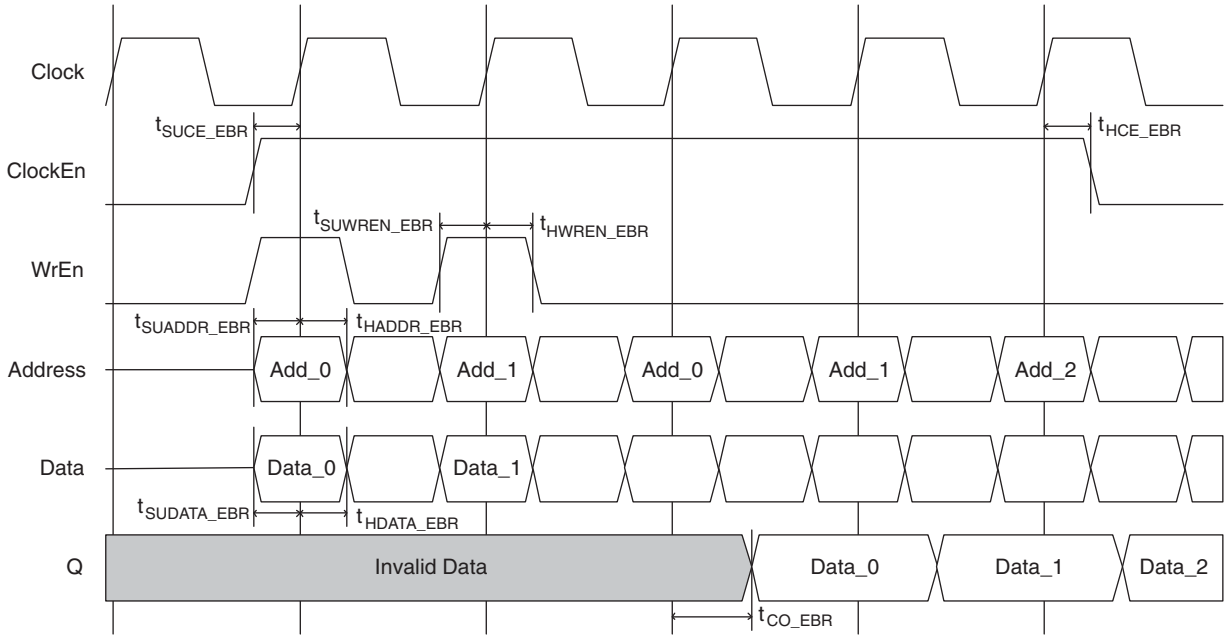


Figure 10-7. Single Port RAM Timing Waveform - NORMAL Mode, with Output Registers

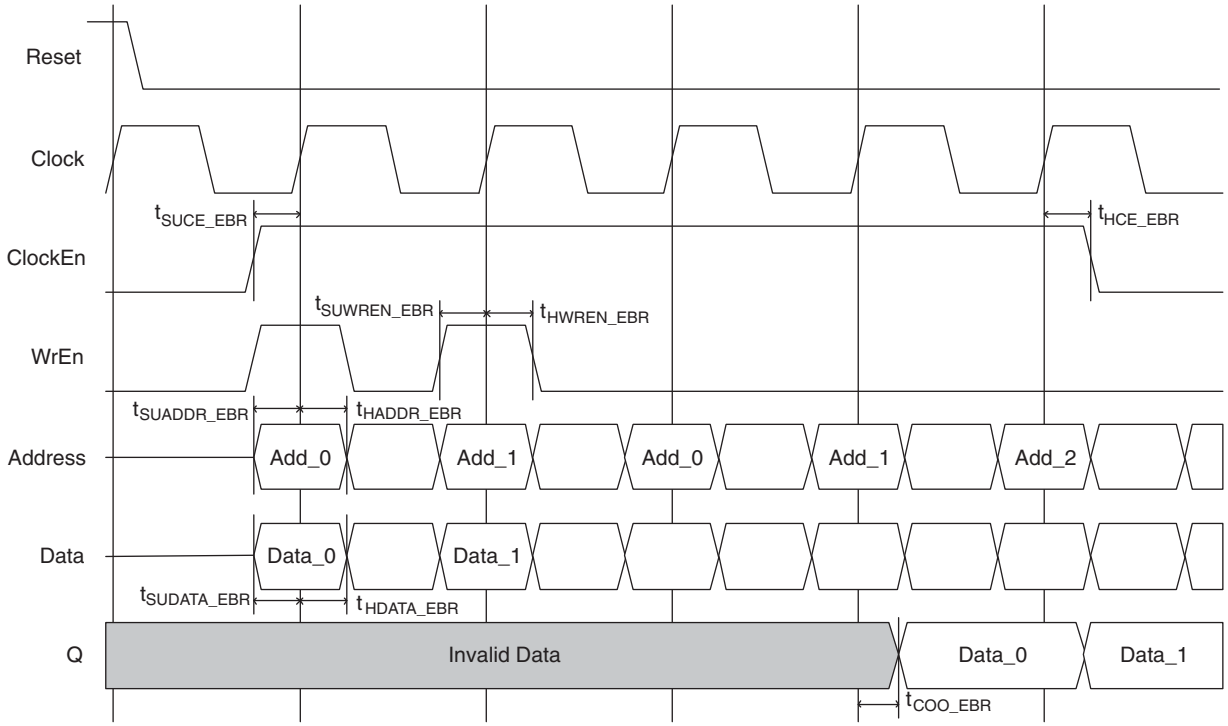


Figure 10-8. Single Port RAM Timing Waveform - READ BEFORE WRITE Mode, without Output Registers

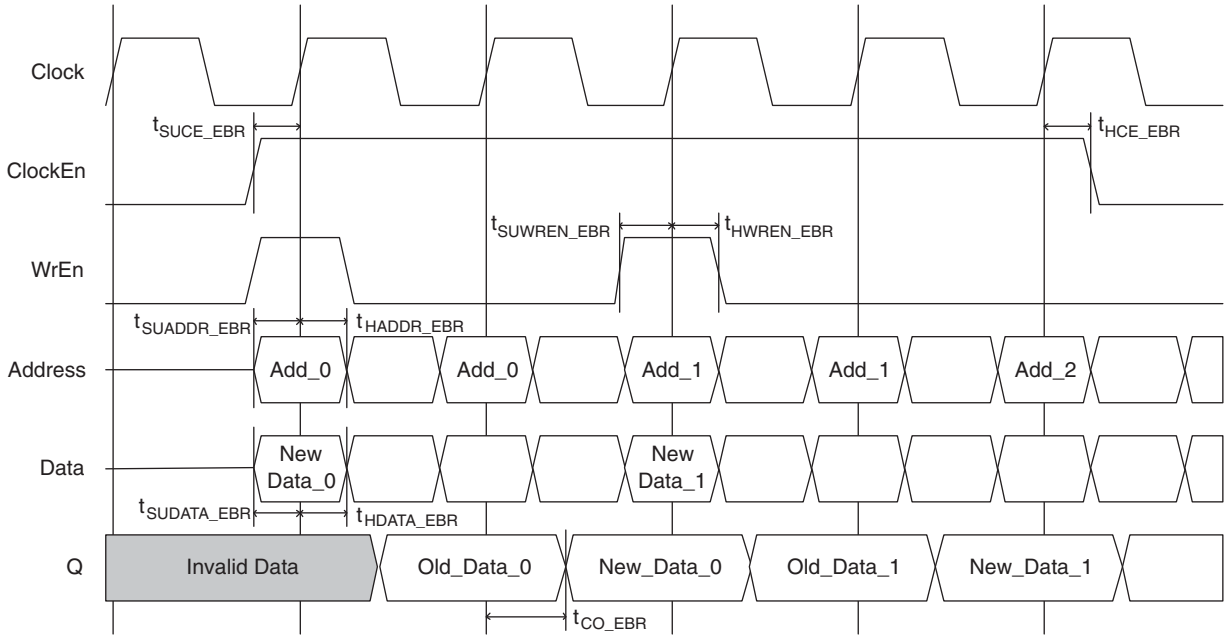


Figure 10-9. Single Port RAM Timing Waveform - READ BEFORE WRITE Mode, with Output Registers

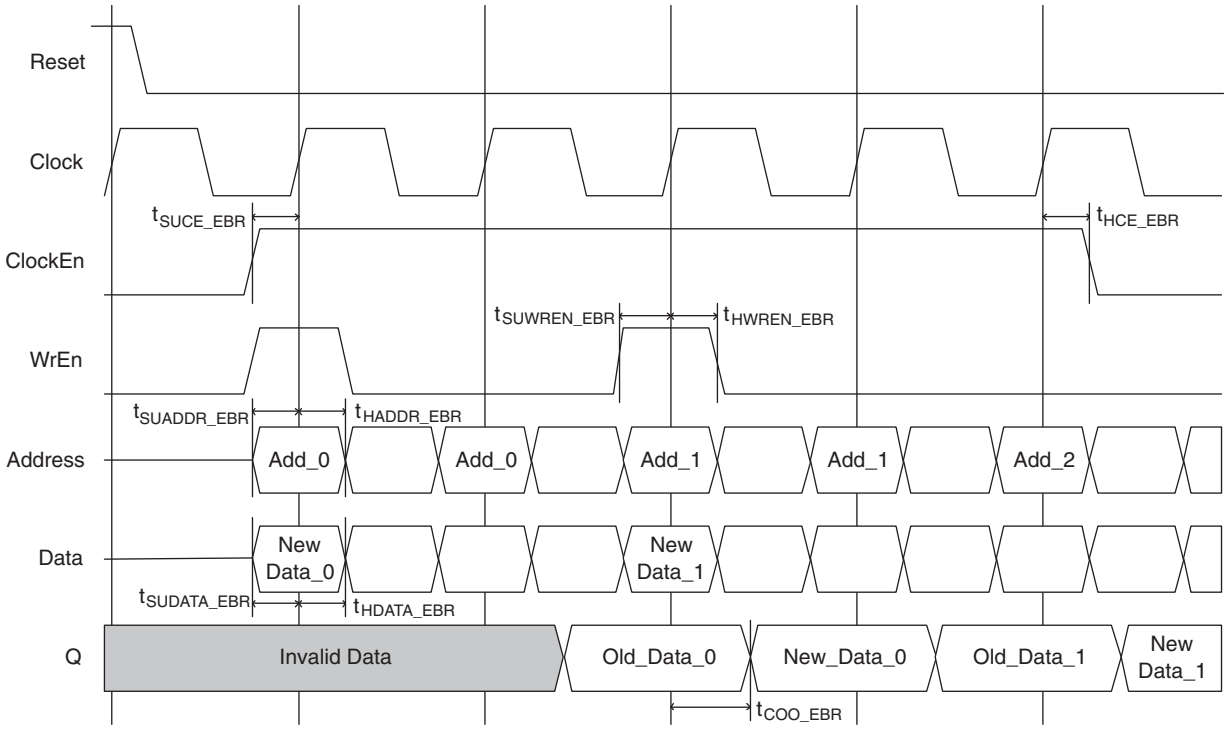


Figure 10-10. Single Port RAM Timing Waveform - WRITE THROUGH Mode, without Output Registers

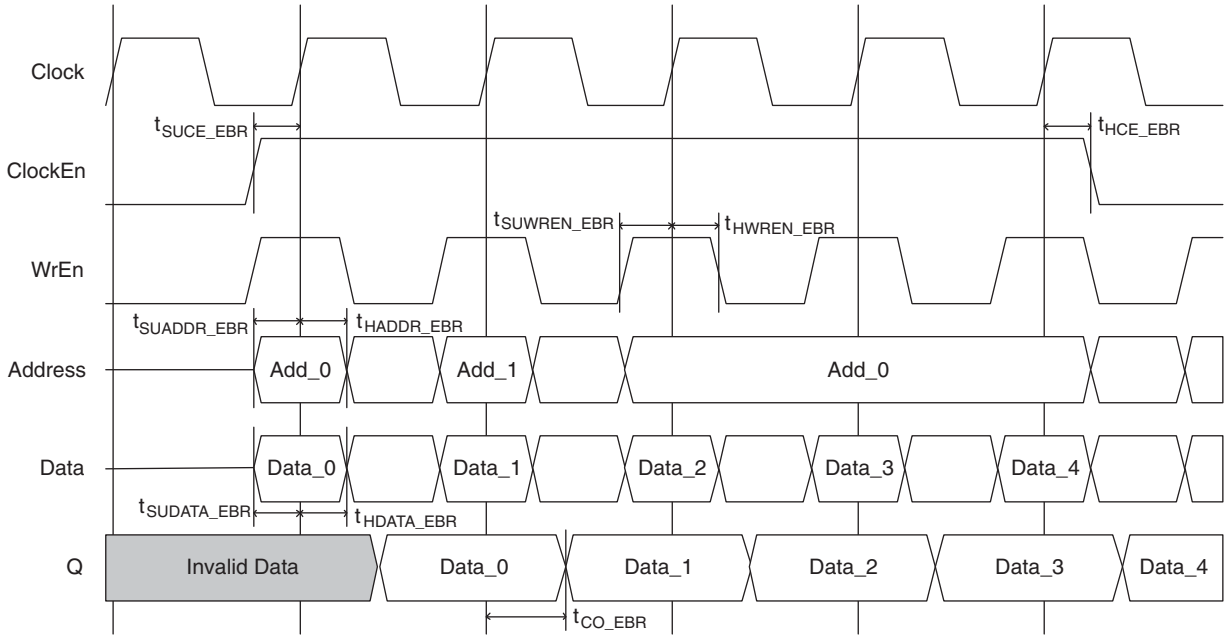
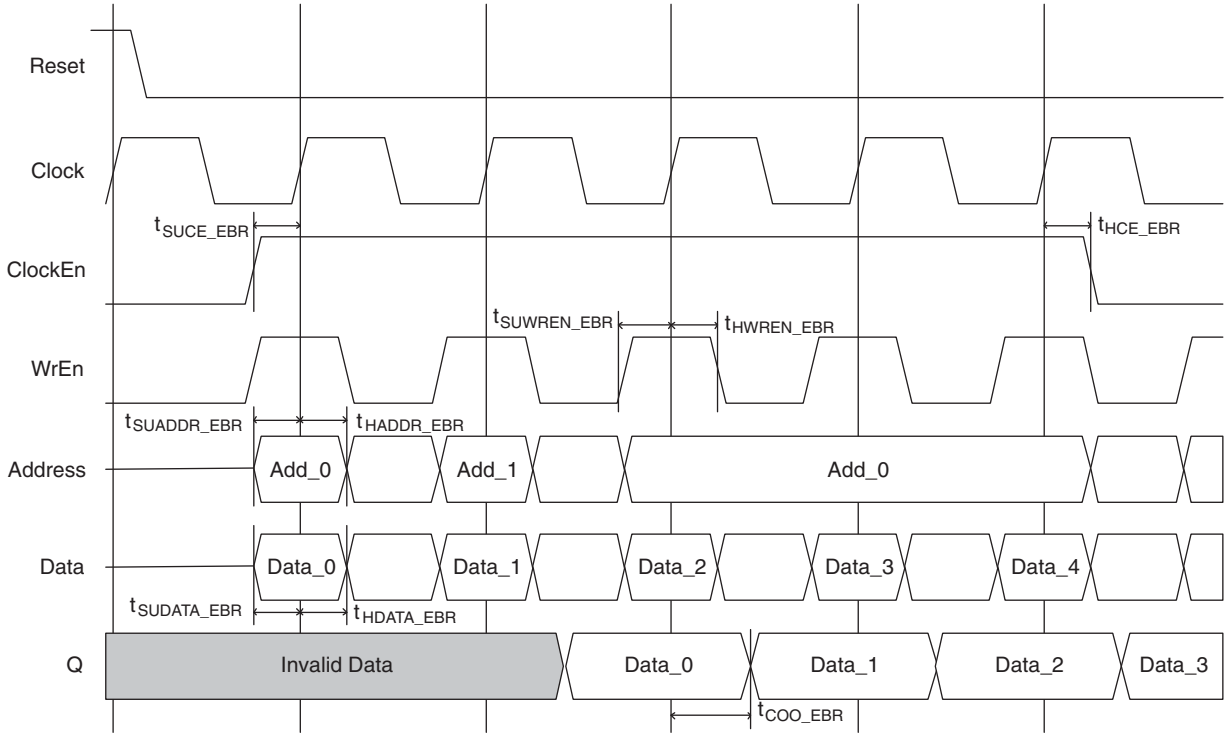


Figure 10-11. Single Port RAM Timing Waveform - WRITE THROUGH Mode, with Output Registers

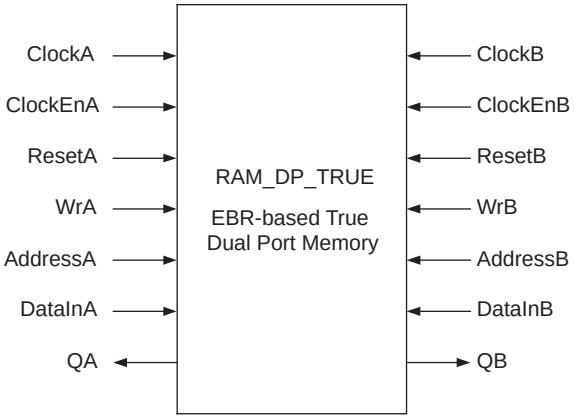


True Dual Port RAM (RAM_DP_TRUE) – EBR Based

The EBR blocks in the LatticeXP2 devices can be configured as True-Dual Port RAM or RAM_DP_TRUE. IPexpress allows users to generate the Verilog-HDL, VHDL or EDIF netlists for the memory size as per design requirements.

IPexpress generates the memory module as shown in Figure 10-12.

Figure 10-12. True Dual Port Memory Module Generated by IPexpress



The generated module makes use of these EBR blocks or primitives. For memory sizes smaller than an EBR block, the module will be created in one EBR block. When the specified memory is larger than one EBR block, multiple EBR blocks can be cascaded in depth or width (as required to create these sizes).

In True Dual Port RAM mode, the input data and address for the ports are registered at the input of the memory array. The output data of the memory is optionally registered at the output.

The various ports and their definitions for Single Port Memory are listed in Table 10-5. The table lists the corresponding ports for the module generated by IPexpress and for the EBR RAM_DP_TRUE primitive.

Table 10-5. EBR-based True Dual Port Memory Port Definitions

Port Name in Generated Module	Port Name in the EBR Block Primitive	Description	Active State
ClockA, ClockB	CLKA, CLKB	Clock for PortA and PortB	Rising Clock Edge
ClockEnA, ClockEnB	CEA, CEB	Clock Enables for Port CLKA and CLKB	Active High
AddressA, AddressB	ADA[x1:0], ADB[x2:0]	Address Bus port A and port B	—
DataA, DataB	DIA[y1:0], DIB[y2:0]	Input Data port A and port B	—
QA, QB	DOA[y1:0], DOB[y2:0]	Output Data port A and port B	—
WrA, WrB	WEA, WEB	Write enable port A and port B	Active High
ResetA, ResetB	RSTA, RSTB	Reset for PortA and PortB	Active High
—	CSA[2:0], CSB[2:0]	Chip Selects for each port	—

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

Chip Select (CS) is a useful port in the EBR primitive when multiple cascaded EBR blocks are required by the memory. The CS signal forms the MSB for the address when multiple EBR blocks are cascaded. Since CS is a 3-bit bus, it can cascade eight memories easily. However, if the memory size specified by the user requires more than eight EBR blocks, the ispLEVER software automatically generates the additional address decoding logic, which is implemented in the PFU external to the EBR blocks.

Each EBR block consists of 18,432 bits of RAM. The values for x's (for address) and y's (data) for each EBR block for the devices are listed in Table 10-6.

Table 10-6. True Dual Port Memory Sizes for 16K Memory for LatticeXP2

Dual Port Memory Size	Input Data Port A	Input Data Port B	Output Data Port A	Output Data Port B	Address Port A [MSB:LSB]	Address Port B [MSB:LSB]
16K x 1	DIA	DIB	DOA	DOB	ADA[13:0]	ADB[13:0]
8K x 2	DIA[1:0]	DIB[1:0]	DOA[1:0]	DOB[1:0]	ADA[12:0]	ADB[12:0]
4K x 4	DIA[3:0]	DIB[3:0]	DOA[3:0]	DOB[3:0]	ADA[11:0]	ADB[11:0]
2K x 9	DIA[8:0]	DIB[8:0]	DOA[8:0]	DOB[8:0]	ADA[10:0]	ADB[10:0]
1K x 18	DIA[17:0]	DIB[17:0]	DOA[17:0]	DOB[17:0]	ADA[9:0]	ADB[9:0]

Table 10-7 shows the various attributes available for the Single Port Memory (RAM_DQ). Some of these attributes are user-selectable through the IPexpress GUI. For detailed attribute definitions, refer to the Appendix A.

Table 10-7. True Dual Port RAM Attributes for LatticeXP2

Attribute	Description	Values	Default Value	User Selectable Through IPexpress
Port A Address depth	Address Depth Port A	16K, 8K, 4K, 2K, 1K		YES
Port A Data Width	Data Word Width Port A	1, 2, 4, 9, 18	1	YES
Port B Address depth	Address Depth Port B	16K, 8K, 4K, 2K, 1K		YES
Port B Data Width	Data Word Width Port B	1, 2, 4, 9, 18	1	YES
Port A Enable Output Registers	Register Mode (Pipelining) for Port A	NOREG, OUTREG	NOREG	YES
Port B Enable Output Registers	Register Mode (Pipelining) for Port B	NOREG, OUTREG	NOREG	YES
Enable GSR	Enables Global Set Reset	ENABLE, DISABLE	ENABLE	YES
Reset Mode	Selects the Reset type	ASYNC, SYNC	ASYNC	YES
Memory File Format		BINARY, HEX, ADDRESSED HEX		YES
Port A Write Mode	Read / Write Mode for Port A	NORMAL, WRITETHROUGH, READ-BEFOREWRITE	NORMAL	YES
Port B Write Mode	Read / Write Mode for Port B	NORMAL, WRITETHROUGH, READ-BEFOREWRITE	NORMAL	YES
Chip Select Decode for Port A	Chip Select Decode for Port A	0b000, 0b001, 0b010, 0b011, 0b100, 0b101, 0b110, 0b111	0b000	NO
Chip Select Decode for Port B	Chip Select Decode for Port B	0b000, 0b001, 0b010, 0b011, 0b100, 0b101, 0b110, 0b111	0b000	NO
Init Value	Initialization value	0x0000000000000000000000000000000000000000000000000000000000000000000000000000000.......0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF	0x00000000000000000000000000000000000000000000000000000000000000000000000000000000	NO

The True Dual Port RAM (RAM_DP_TRUE) can be configured as NORMAL, READ BEFORE WRITE or WRITE THROUGH modes. Each of these modes affects what data comes out of the port Q of the memory during the write operation followed by the read operation at the same memory location. The detailed discussions of the WRITE modes and the constraints of the True Dual Port can be found in Appendix A.

Additionally, users can select to enable the output registers for RAM_DP_TRUE. Figures 10-13 through 10-18 show the internal timing waveforms for the True Dual Port RAM (RAM_DP_TRUE) with these options.

Figure 10-13. True Dual Port RAM Timing Waveform - NORMAL Mode, without Output Registers

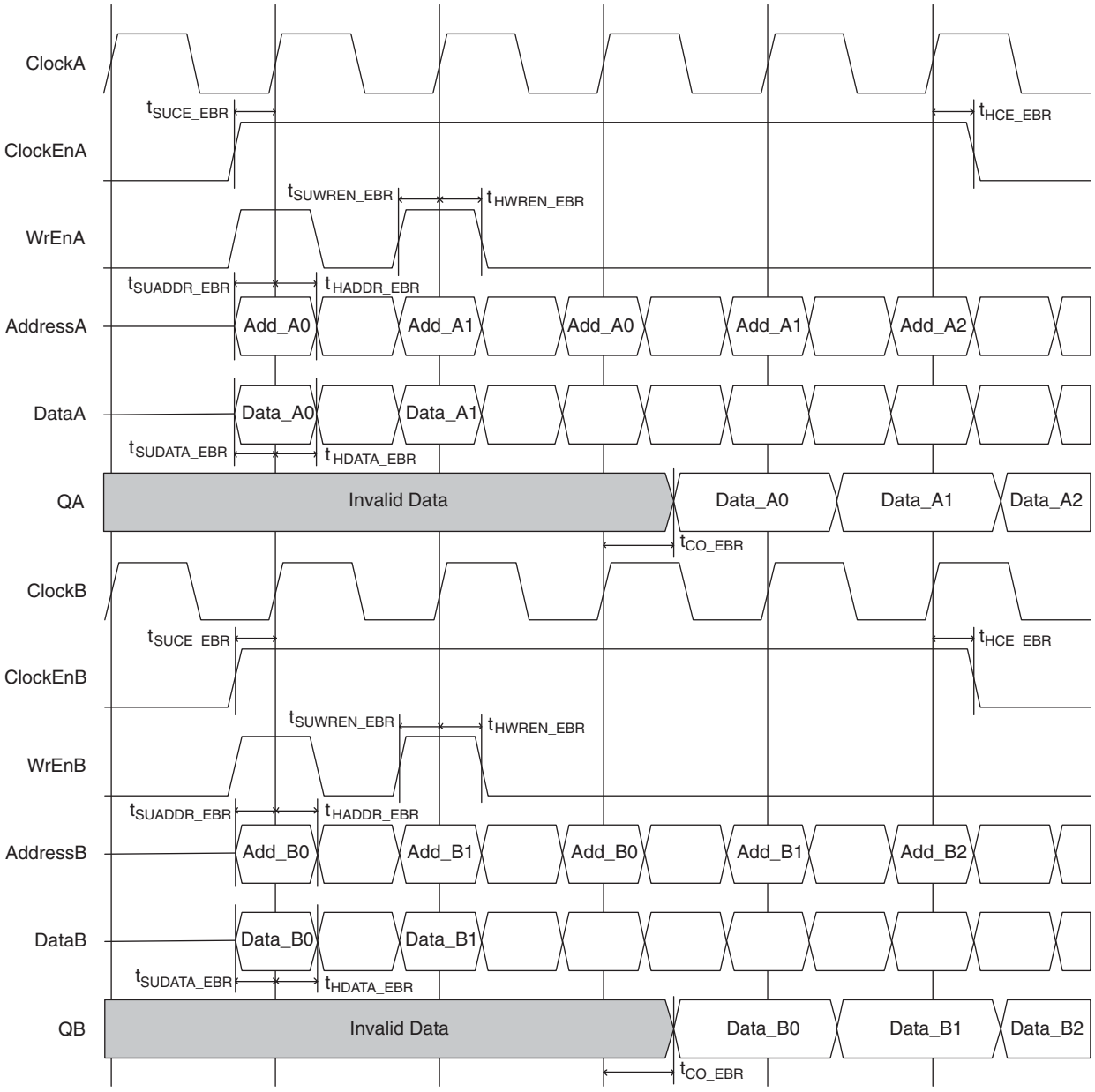


Figure 10-14. True Dual Port RAM Timing Waveform - NORMAL Mode with Output Registers

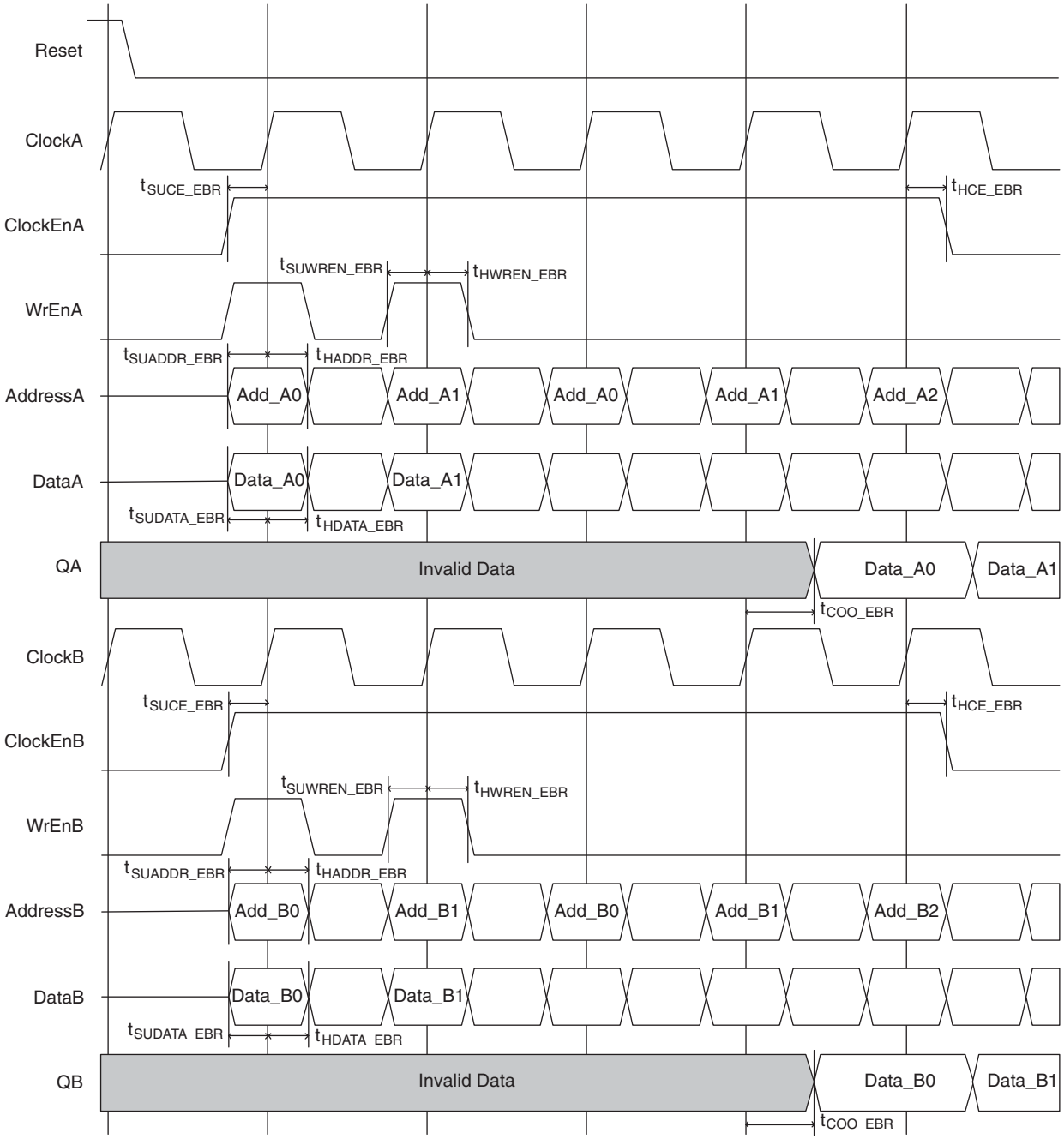


Figure 10-15. True Dual Port RAM Timing Waveform - READ BEFORE WRITE Mode, without Output Registers

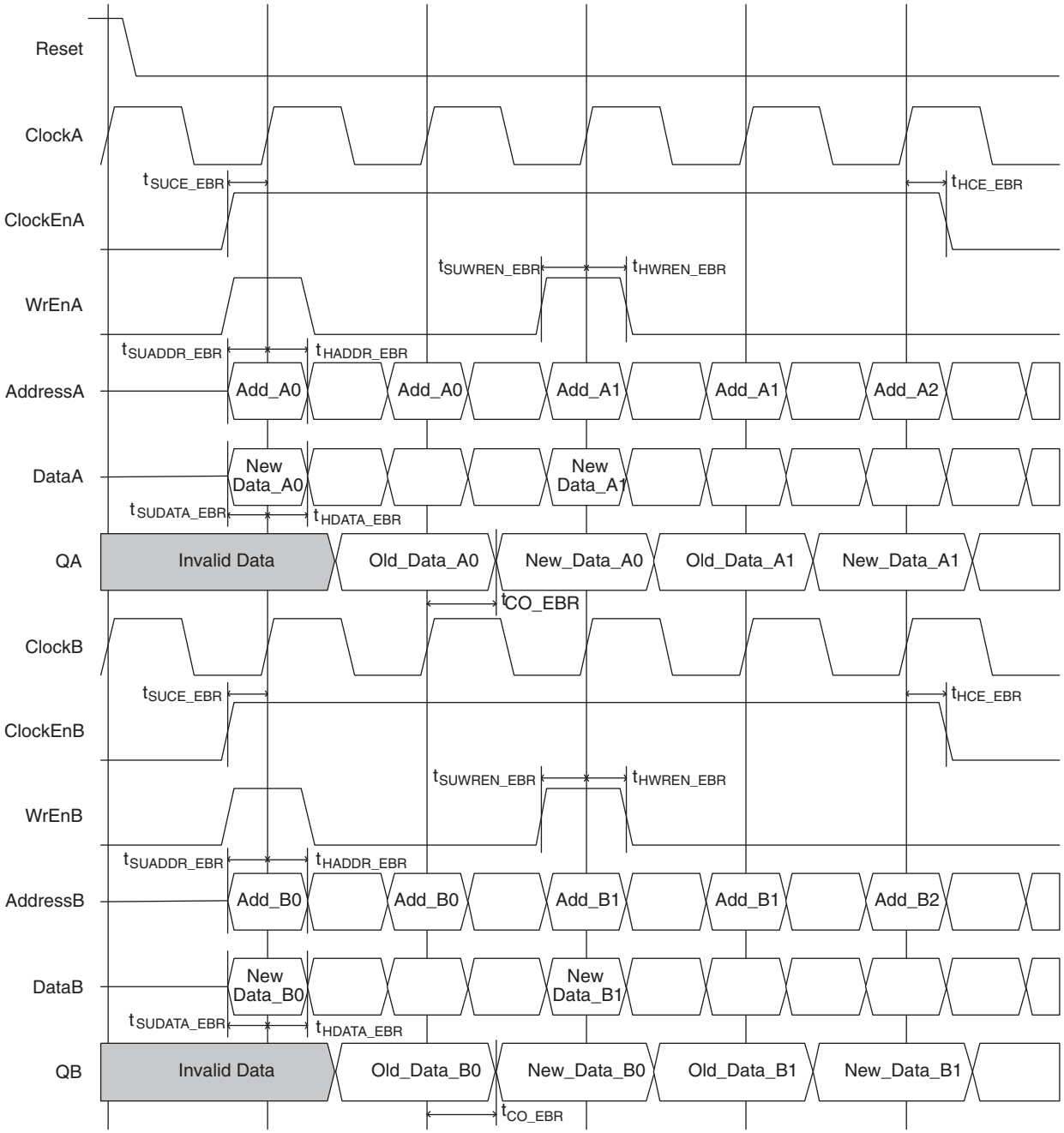


Figure 10-16. True Dual Port RAM Timing Waveform - READ BEFORE WRITE Mode, with Output Registers

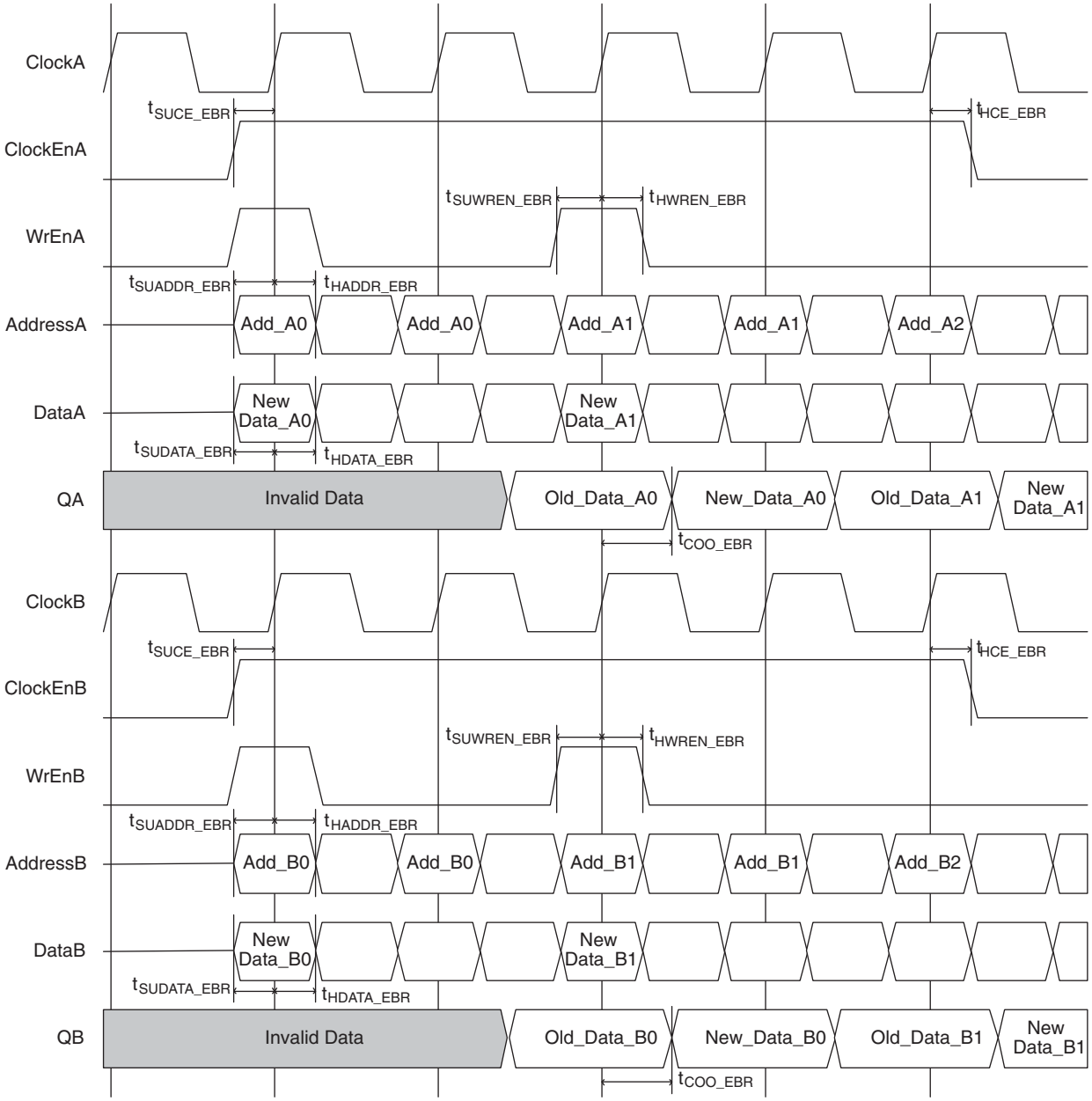


Figure 10-17. True Dual Port RAM Timing Waveform - WRITE THROUGH Mode, without Output Registers

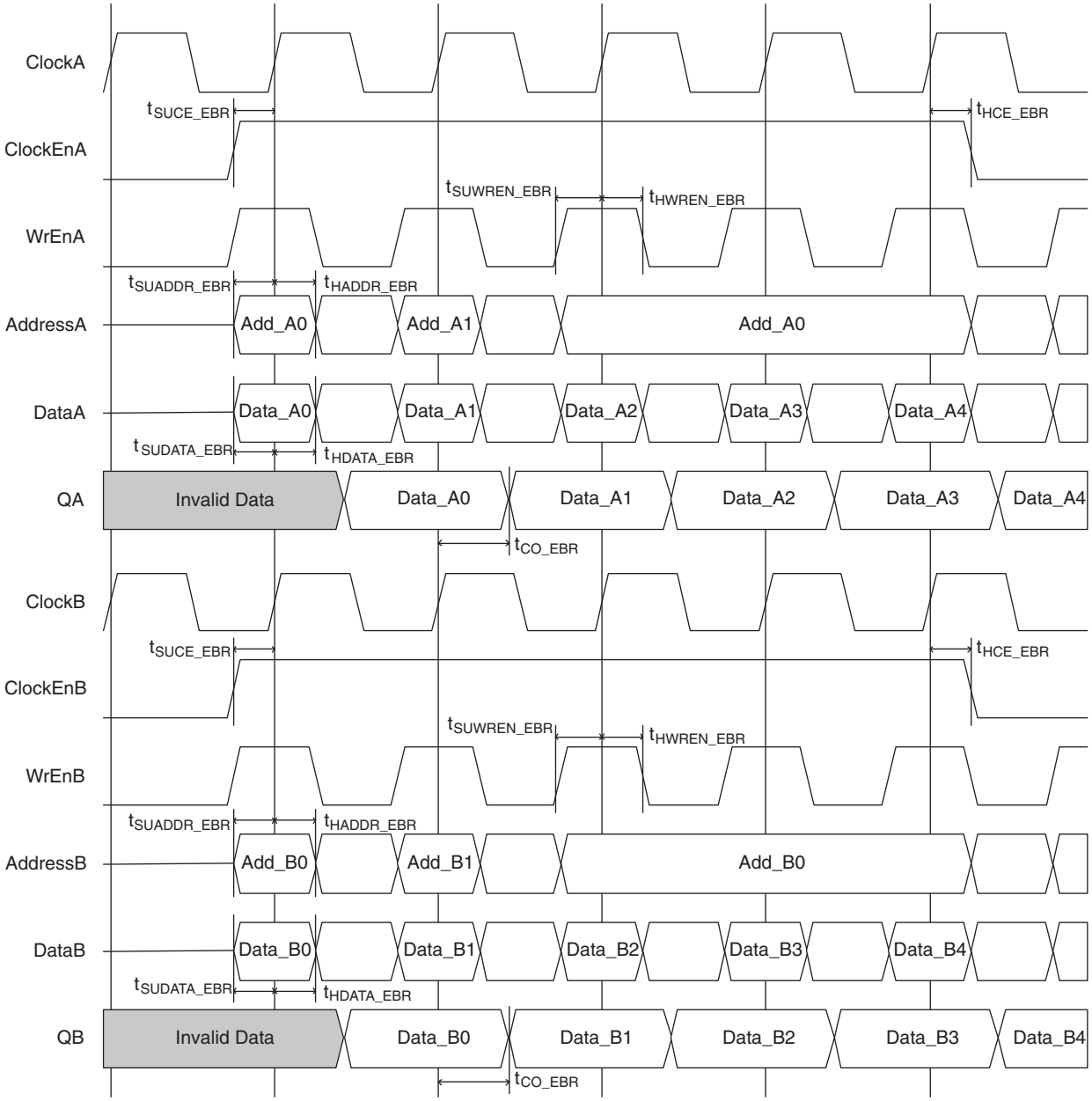
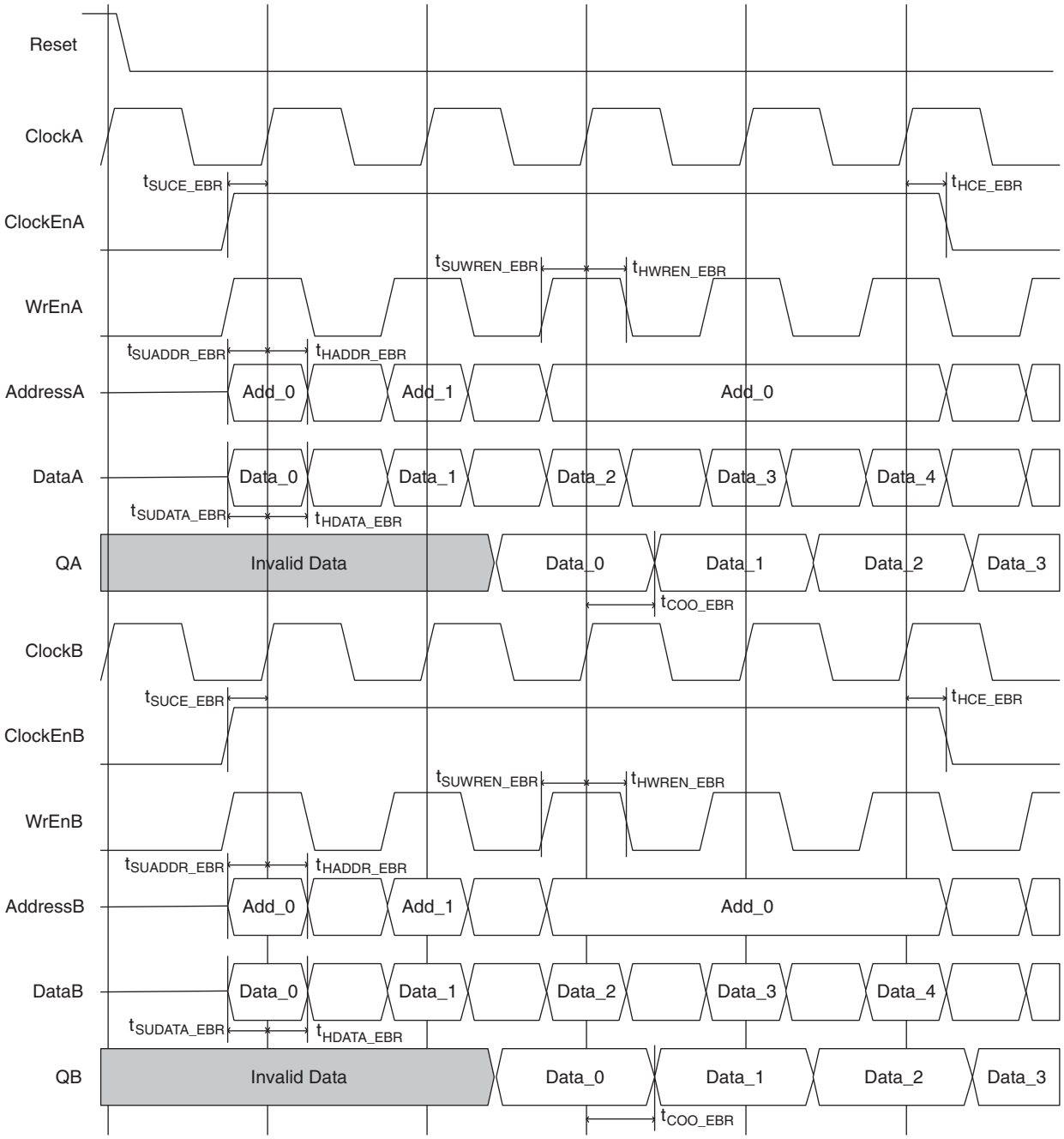


Figure 10-18. True Dual Port RAM Timing Waveform - WRITE THROUGH Mode, with Output Registers

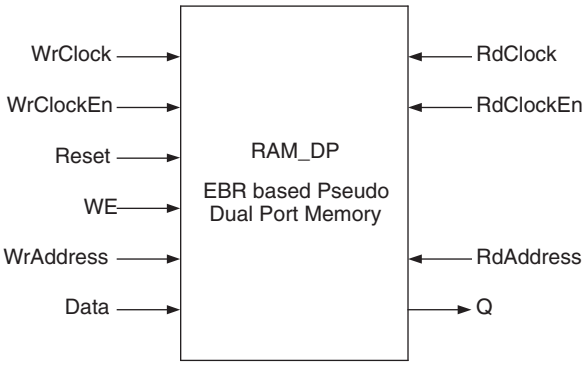


Pseudo Dual Port RAM (RAM_DP) – EBR Based

The EBR blocks in LatticeXP2 devices can be configured as Pseudo-Dual Port RAM or RAM_DP. IPexpress allows users to generate the Verilog-HDL or VHDL along with EDIF netlists for the memory size as per design requirements.

IPexpress generates the memory module as shown in Figure 10-19.

Figure 10-19. Pseudo Dual Port Memory Module Generated by IPexpress



The generated module makes use of these EBR blocks or primitives. For memory sizes smaller than an EBR block, the module will be created in one EBR block. If the specified memory is larger than one EBR block, multiple EBR blocks can be cascaded in depth or width (as required to create these sizes).

In Pseudo Dual Port RAM mode, the input data and address for the ports are registered at the input of the memory array. The output data of the memory is optionally registered at the output.

The various ports and their definitions for the Single Port Memory are listed in Table 10-8. The table lists the corresponding ports for the module generated by IPexpress and for the EBR RAM_DP primitive.

Table 10-8. EBR-based Pseudo-Dual Port Memory Port Definitions

Port Name in Generated Module	Port Name in the EBR Block Primitive	Description	Active State
RdAddress	ADR[x1:0]	Read Address	—
WrAddress	ADW[x2:0]	Write Address	—
RdClock	CLKR	Read Clock	Rising Clock Edge
WrClock	CLKW	Write Clock	Rising Clock Edge
RdClockEn	CER	Read Clock Enable	Active High
WrClockEn	CEW	Write Clock Enable	Active High
Q	DO[y1:0]	Read Data	—
Data	DI[y2:0]	Write Data	—
WE	WE	Write Enable	Active High
Reset	RST	Reset	Active High
—	CS[2:0]	Chip Select	—

Reset (RST) resets only the input and output registers of the RAM. It does not reset the contents of the memory.

Chip Select (CS) is a useful port when multiple cascaded EBR blocks are required by the memory. The CS signal forms the MSB for the address when multiple EBR blocks are cascaded. Since CS is a 3-bit bus, it can cascade eight memories easily. However, if the memory size specified by the user requires more than eight EBR blocks, the ispLEVER software automatically generates the additional address decoding logic, which is implemented in the PFU external to the EBR blocks.

Each EBR block consists of 18,432 bits of RAM. The values for x's (for address) and y's (data) for each EBR block for the devices are as in Table 10-9.

Table 10-9. Pseudo-Dual Port Memory Sizes for 16K Memory for LatticeXP2

Pseudo-Dual Port Memory Size	Input Data Port A	Input Data Port B	Output Data Port A	Output Data Port B	Read Address Port A [MSB:LSB]	Write Address Port B [MSB:LSB]
16K x 1	DIA	DIB	DOA	DOB	RAD[13:0]	WAD[13:0]
8K x 2	DIA[1:0]	DIB[1:0]	DOA[1:0]	DOB[1:0]	RAD[12:0]	WAD[12:0]
4K x 4	DIA[3:0]	DIB[3:0]	DOA[3:0]	DOB[3:0]	RAD[11:0]	WAD[11:0]
2K x 9	DIA[8:0]	DIB[8:0]	DOA[8:0]	DOB[8:0]	RAD[10:0]	WAD[10:0]
1K x 18	DIA[17:0]	DIB[17:0]	DOA[17:0]	DOB[17:0]	RAD[9:0]	WAD[9:0]
512 x 36	DIA[35:0]	DIB[35:0]	DOA[35:0]	DOB[35:0]	RAD[8:0]	WAD[8:0]

Table 10-10 shows the various attributes available for the Pseudo-Dual Port Memory (RAM_DP). Some of these attributes are user-selectable through the IPexpress GUI. For detailed attribute definitions, refer to Appendix A.

Table 10-10. Pseudo-Dual Port RAM Attributes for LatticeXP2

Attribute	Description	Values	Default Value	User Selectable Through IPexpress
Read Port Address Depth	Address Depth Read Port	16K, 8K, 4K, 2K, 1K, 512		YES
Read Port Data Width	Data Word Width Read Port	1, 2, 4, 9, 18, 36	1	YES
Write Port Address Depth	Address Depth Write Port	16K, 8K, 4K, 2K, 1K		YES
Write Port Data Width	Data Word Width Write Port	1, 2, 4, 9, 18, 36	1	YES
Write Port Enable Output Registers	Register Mode (Pipelining) for Write Port	NOREG, OUTREG	NOREG	YES
Enable GSR	Enables Global Set Reset	ENABLE, DISABLE	ENABLE	YES
Reset Mode	Selects the Reset type	ASYNC, SYNC	ASYNC	YES
Memory File Format		BINARY, HEX, ADDRESSED HEX		YES
Read Port Write Mode	Read / Write Mode for Read Port	NORMAL	NORMAL	YES
Write Port Write Mode	Read / Write Mode for Write Port	NORMAL	NORMAL	YES
Chip Select Decode for Read Port	Chip Select Decode for Read Port	0b000, 0b001, 0b010, 0b011, 0b100, 0b101, 0b110, 0b111	0b000	NO
Chip Select Decode for Write Port	Chip Select Decode for Write Port	0b000, 0b001, 0b010, 0b011, 0b100, 0b101, 0b110, 0b111	0b000	NO
Init Value	Initialization value	0x0000000000000000000000000 000000000000000000000000000 000000000000000000000000000 0......0xFFFFFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFFFFFFFFFF FFFFFFFFFFFFF	0x000000000000 00000000000000 00000000000000 00000000000000 00000000000000 000000000000	NO

Users have the option to enable the output registers for Pseudo-Dual Port RAM (RAM_DP). Figures 10-20 and 10-21 show the internal timing waveforms for Pseudo-Dual Port RAM (RAM_DP) with these options.

Figure 10-20. PSEUDO DUAL PORT RAM Timing Diagram - without Output Registers

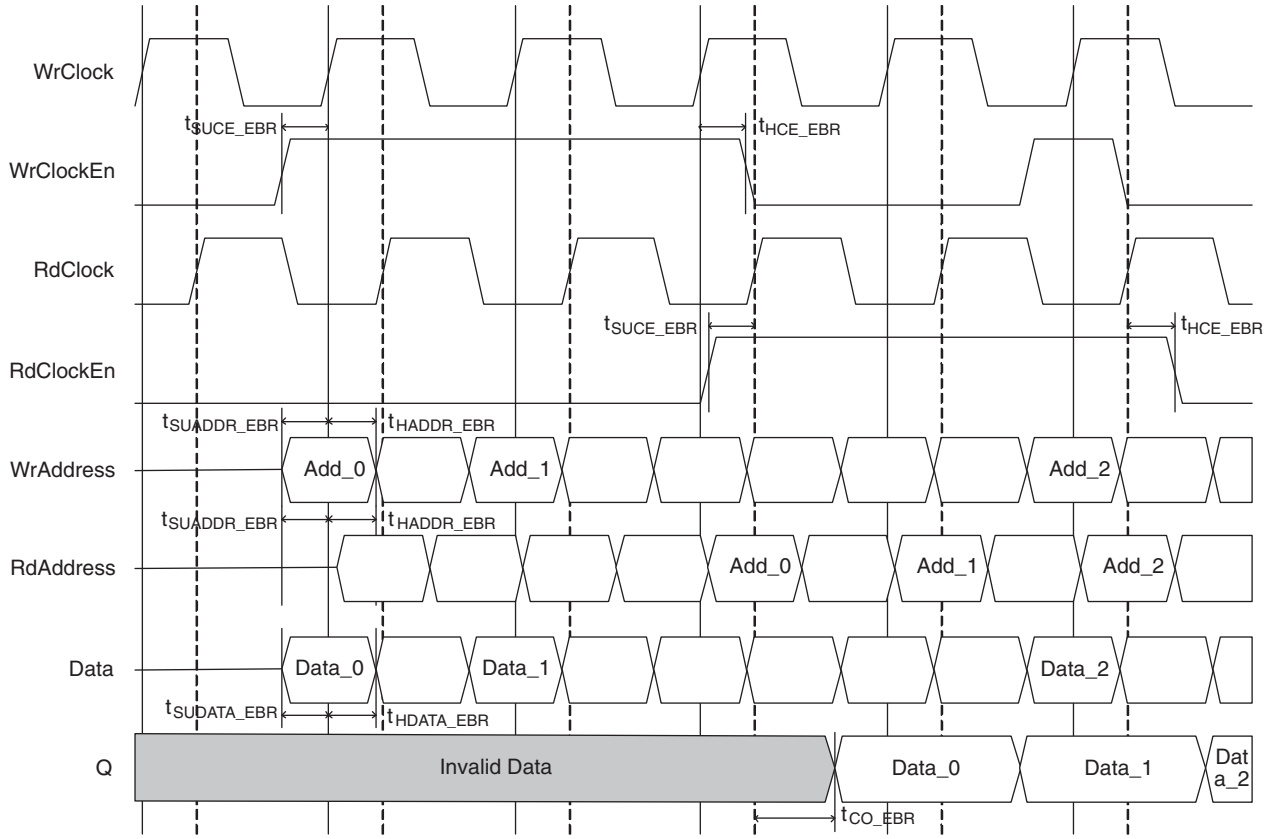
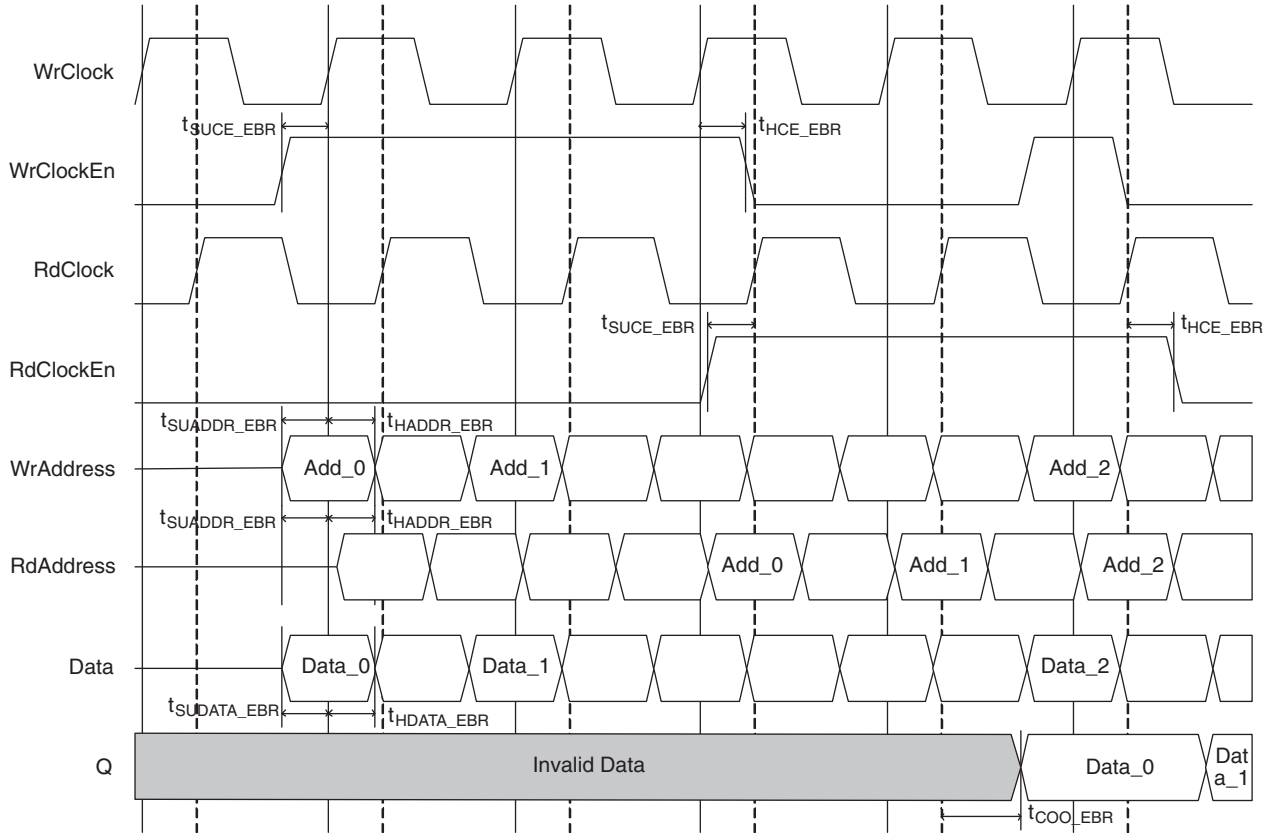


Figure 10-21. PSEUDO DUAL PORT RAM Timing Diagram - with Output Registers

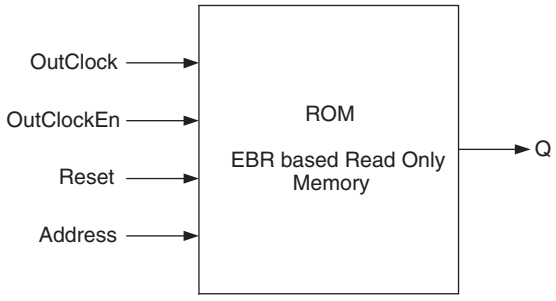


Read Only Memory (ROM) - EBR Based

The EBR blocks in the LatticeXP2 devices can be configured as Read Only Memory or ROM. IPexpress allows users to generate the Verilog-HDL or VHDL and the EDIF netlist for the memory size, as per design requirements. Users are required to provide the ROM memory content in the form of an initialization file.

IPexpress generates the memory module as shown in Figure 10-22.

Figure 10-22. Read-Only Memory Module Generated by IPexpress



The generated module makes use of these EBR blocks or primitives. For memory sizes smaller than an EBR block, the module will be created in one EBR block. If the specified memory is larger than one EBR block, multiple EBR blocks can be cascaded, in depth or width (as required to create these sizes).

In ROM mode, the address for the port is registered at the input of the memory array. The output data of the memory is optionally registered at the output.

The various ports and their definitions for the ROM are listed in Table 10-11. The table lists the corresponding ports for the module generated by IPexpress and for the ROM primitive.

Table 10-11. EBR-based ROM Port Definitions

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

Reset (RST) resets only the input and output registers of the RAM. It does not reset the contents of the memory.

Chip Select (CS) is a useful port when multiple cascaded EBR blocks are required by the memory. The CS signal forms the MSB for the address when multiple EBR blocks are cascaded. Since CS is a 3-bit bus, it can cascade eight memories easily. However, if the memory size specified by the user requires more than eight EBR blocks, the ispLEVER software automatically generates the additional address decoding logic, which is implemented in the PFU external to the EBR blocks.

While generating the ROM using IPexpress, the user must provide the initialization file to pre-initialize the contents of the ROM. These files are the *.mem files and they can be of Binary, Hex or the Addressed Hex formats. The initialization files are discussed in detail in the Initializing Memory section of this document.

Users have the option of enabling the output registers for Read Only Memory (ROM). Figures 10-23 and 10-24 show the internal timing waveforms for the Read Only Memory (ROM) with these options.

Each EBR block consists of 18,432 bits of RAM. The values for x's (for address) and y's (data) for each EBR block for the devices are as per Table 10-12.

Table 10-12. ROM Memory Sizes for 16K Memory for LatticeXP2

ROM	Output Data	Address Port [MSB:LSB]
16K x 1	DOA	WAD[13:0]
8K x 2	DOA[1:0]	WAD[12:0]
4K x 4	DOA[3:0]	WAD[11:0]
2K x 9	DOA[8:0]	WAD[10:0]
1K x 18	DOA[17:0]	WAD[9:0]
512 x 36	DOA[35:0]	WAD[8:0]

Table 10-13 shows the various attributes available for the Read Only Memory (ROM). Some of these attributes are user-selectable through the IPexpress GUI. For detailed attribute definitions, refer to Appendix A.

Table 10-13. Pseudo-Dual Port RAM Attributes for LatticeXP2

Attribute	Description	Values	Default Value	User Selectable Through IPexpress
Address depth	Address Depth Read Port	16K, 8K, 4K, 2K, 1K, 512		YES
Data Width	Data Word Width Read Port	1, 2, 4, 9, 18, 36	1	YES
Enable Output Registers	Register Mode (Pipelining) for Write Port	NOREG, OUTREG	NOREG	YES
Enable GSR	Enables Global Set Reset	ENABLE, DISABLE	ENABLE	YES
Reset Mode	Selects the Reset type	ASYNC, SYNC	ASYNC	YES
Memory File Format		BINARY, HEX, ADDRESSED HEX		YES
Chip Select Decode	Chip Select Decode for Read Port	0b000, 0b001, 0b010, 0b011, 0b100, 0b101, 0b110, 0b111	0b000	NO

Figure 10-23. ROM Timing Waveform - without Output Registers

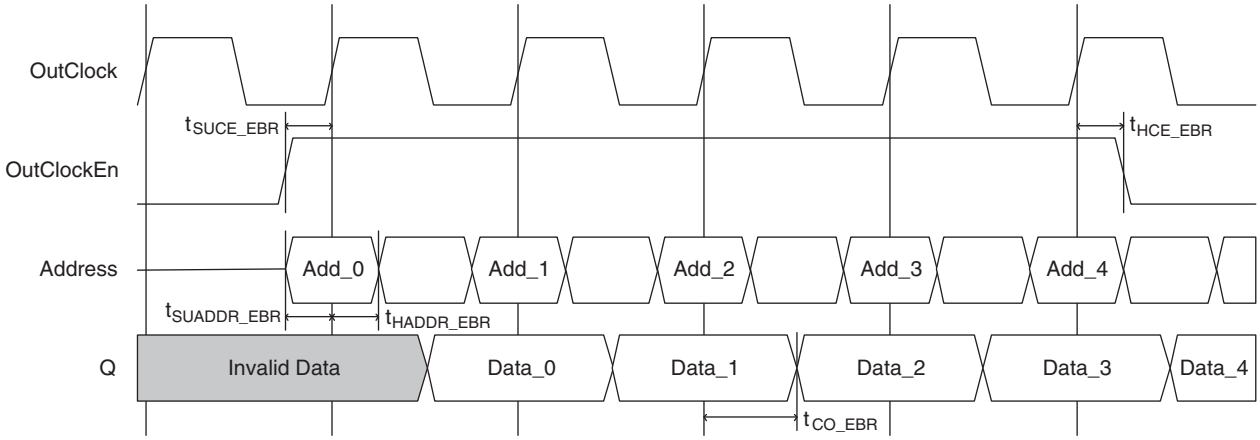
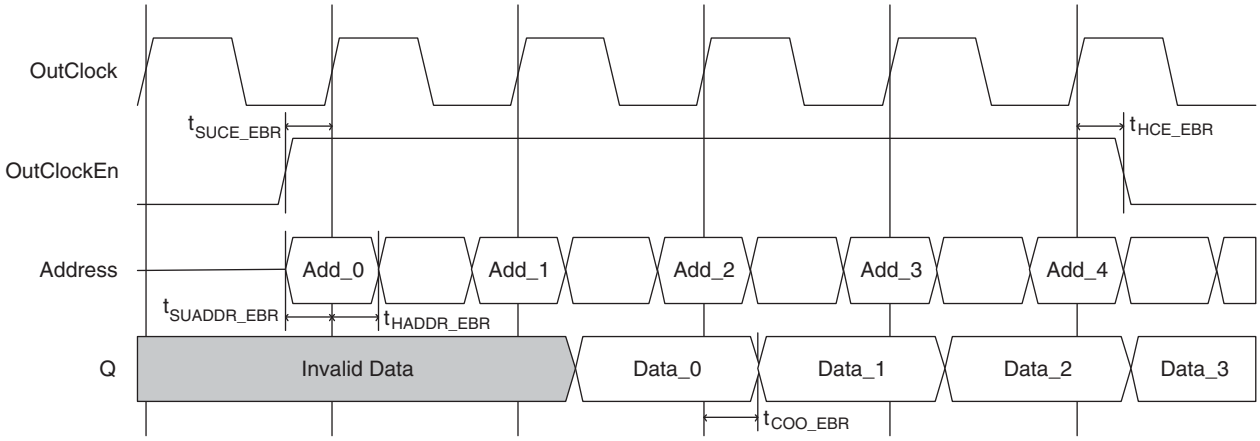


Figure 10-24. ROM Timing Waveform - with Output Registers

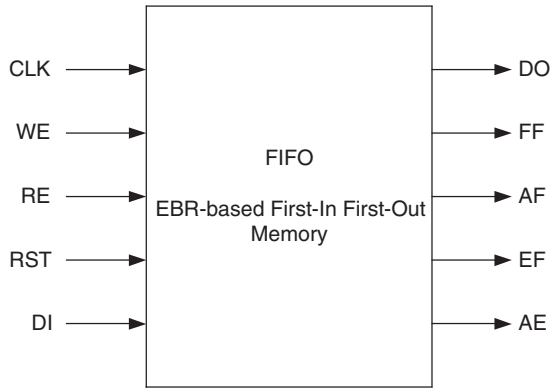


First In First Out (FIFO, FIFO_DC) – EBR Based

The EBR blocks in LatticeXP2 devices can be configured as Dual Clock First In First Out Memories, FIFO_DC. IPexpress allows users to generate the Verilog-HDL or VHDL along EDIF netlist for the memory size, according to design requirements.

IPexpress generates the FIFO_DC memory module as shown in Figure 10-25.

Figure 10-25. FIFO Module Generated by IPexpress



The generated module makes use of these EBR blocks or primitives. For memory sizes smaller than an EBR block, the module will be created in one EBR block. If the specified memory is larger than one EBR block, multiple EBR blocks can be cascaded, in depth or width (as required to create these sizes).

A clock is always required, as only synchronous write is supported. The various ports and their definitions for the FIFO_DC are listed in Figure 10-14.

Table 10-14. EBR based FIFO_DC Memory Port Definitions

Port Name in Generated Module	Port Name in Primitive	Description	Active State
CLKR		Read Port Clock	Rising Clock Edge
CLKW		Write Port Clock	Rising Clock Edge
WE		Write Enable	Active High
RE		Read Enable	Active High
RST		Reset	Active High
DI		Data Input	—
DO		Data Output	—
FF		Full Flag	Active High
AF		Almost Full Flag	Active High
EF		Empty Flag	Active High
AE		Almost Empty	Active High

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

The various supported sizes for the FIFO_DC for LatticeXP2 are as per Table 10-15.

Table 10-15. FIFO_DC Data Widths Sizes for LatticeXP2

FIFO Size	Input Data	Output Data
16K x 1	DI	DO
8K x 2	DI[1:0]	DO[1:0]
4K x 4	DI[3:0]	DO[3:0]
2K x 9	DI[8:0]	DO[8:0]
1K x 18	DI[17:0]	DO[17:0]
512 x 36	DI[35:0]	DO[35:0]

Table 10-16 shows the various attributes available for the FIFO_DC. Some of these attributes are user-selectable through the IPexpress GUI. For detailed attribute definitions, refer to the Appendix A.

Table 10-16. FIFO_DC Attributes for LatticeXP2

Attribute	Description	Values	Default Value	User Selectable Through IPexpress
Write Port Depth	Number of Address locations	16K, 8K, 4K, 2K, 1K, 512		YES
Write Port Width	Data Port Width	1, 2, 4, 9, 18, 36	1	YES
Read Port Depth	Number of Address locations	16K, 8K, 4K, 2K, 1K, 512		YES
Read Port Width	Data Port Width	1, 2, 4, 9, 18, 36	1	YES
Enable Output Registers	Register Mode (Pipelining) for Write Port	NOREG, OUTREG	NOREG	YES
Reset Mode	Selects the Reset type	ASYNC, SYNC	ASYNC	YES
Almost Full	Almost Full Flag value			YES
Almost Empty	Almost Empty Flag value			YES

FIFO_DC Flags

The FIFO_DC have four flags available: Empty, Almost Empty, Almost Full and Full. Almost Empty and Almost Full flags have a programmable range.

The program ranges for the four FIFO_DC flags are specified in Table 10-17.

Table 10-17. FIFO Flag Settings

FIFO Attribute Name	Description	Programming Range	Program Bits
FF	Full flag setting	2^N - 1	15
AFF	Almost full setting	1 to (FF-1)	15
AEF	Almost empty setting	1 to (FF-1)	15
EF	Empty setting	0	5

The only restriction is that the values must be in an order (Empty=0, Almost Empty next, followed by Almost Full and Full, respectively). The value of Empty is not equal to the value of Almost Empty (or Full is equal to Almost Full). In case, a warning is generated and the value of Empty (or Full) is used in place of Almost Empty (or Almost Full). When coming out of reset, the active high flags empty and almost empty are set to high, since they are true.

The user should specify the absolute value of the address at which the Almost Empty and Almost Full flags will go true. For example, if the Almost Full flag is required to go true at the address location 500 for a FIFO of depth 512, the user should specify a value of 500 in IPexpress.

The Empty and Almost Empty flags are always registered to the read clock and the Full and Almost Full flags are always registered to the write clock.

At reset, both the write and read counters are pointing to address zero. After reset is de-asserted data can be written into the FIFO_DC to the address pointed to by the write counter at the positive edge of the write clock when the write enable is asserted.

Similarly, data can be read from the FIFO_DC from the address pointed to by the read counter at the positive edge of the read clock when read enable is asserted.

Read Pointer Reset (RPReset) is used to indicate a retransmit, and is more commonly used in “packetized” communications. In this application, the user must keep careful track of when a packet is written into or read from the FIFO_DC.

The data output of the FIFO_DC can be registered or non-registered through a selection in IPexpress. The output registers are enabled by read enable. A reset will clear the contents of the FIFO_DC by resetting the read and write pointers and will put the flags in the initial reset state.

FIFO_DC Operation

If the output registers are not enabled it will take two clock cycles to read the first word out. The register for the flag logic causes this extra clock latency. In the architecture of the emulated FIFO_DC, the internal read enables for reading the data out is controlled not only by the read enable provided by the user but also the empty flag. When the data is written into the FIFO, an internal empty flag is registered using write clock that is enabled by write enable (WrEn). Another clock latency is added due to the clock domain transfer from write clock to read clock using another register which is clocked by read clock that is enabled by read enable.

Internally, the output of this register is inverted and then ANDed with the user-provided read enable that becomes the internal read enable to the RAM_DP which is at the core of the FIFO_DC.

Thus, the first read data takes two clock cycles to propagate through. During the first data out, read enable goes high for one clock cycle, empty flag is de-asserted and is not propagated through the second register enabled by the read enable. The first clock cycle brings the Empty Low and the second clock cycle brings the internal read enable high (RdEn and !EF) and then the data is read out by the second clock cycle. Similarly, the first write data after the full flag has a similar latency.

If the user has enabled the output registers, the output registers will cause an extra clock delay during the first data out as they are clocked by the read clock and enabled by the read enable.

1. First RdEn and Clock Cycle to propagate the EF internally.
2. Second RdEn and Clock Cycle to generate internal Read Enable into the DPRAM.
3. Third RdEn and Clock Cycle to get the data out of the output registers.

Figure 10-26. FIFO_DC without Output Registers (Non-Pipelined)

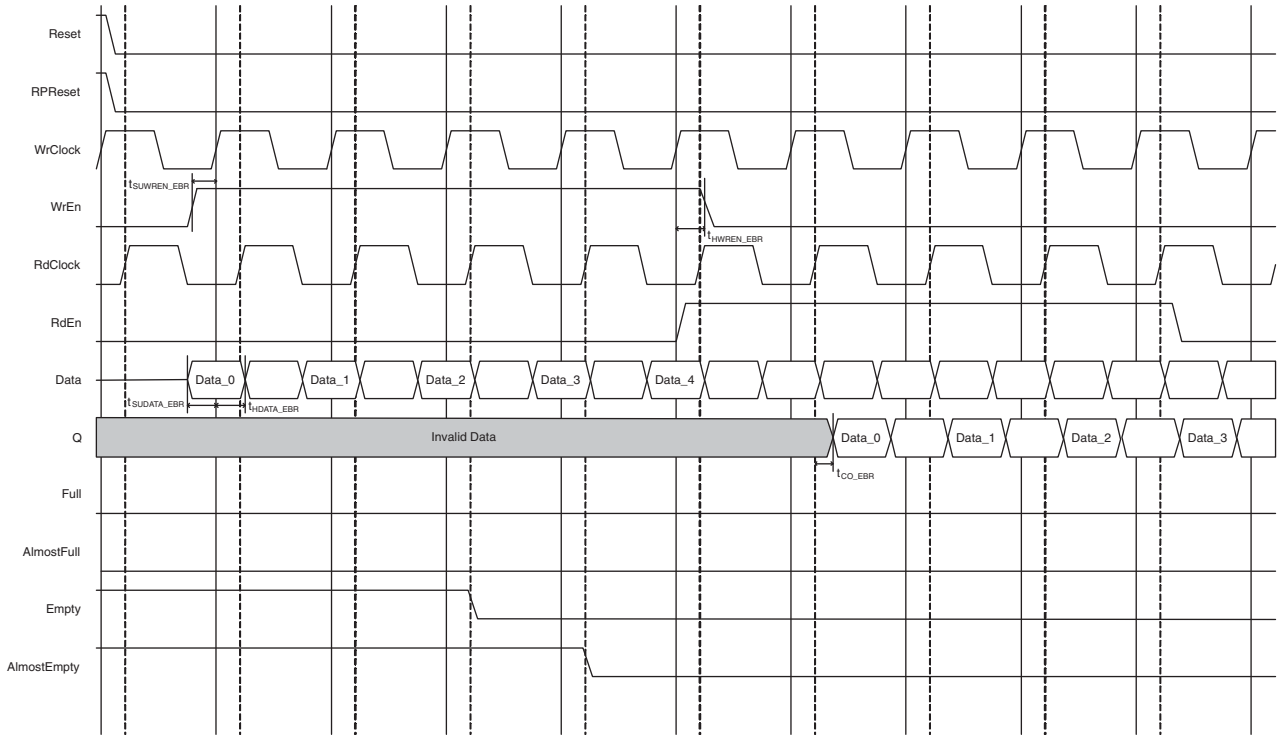
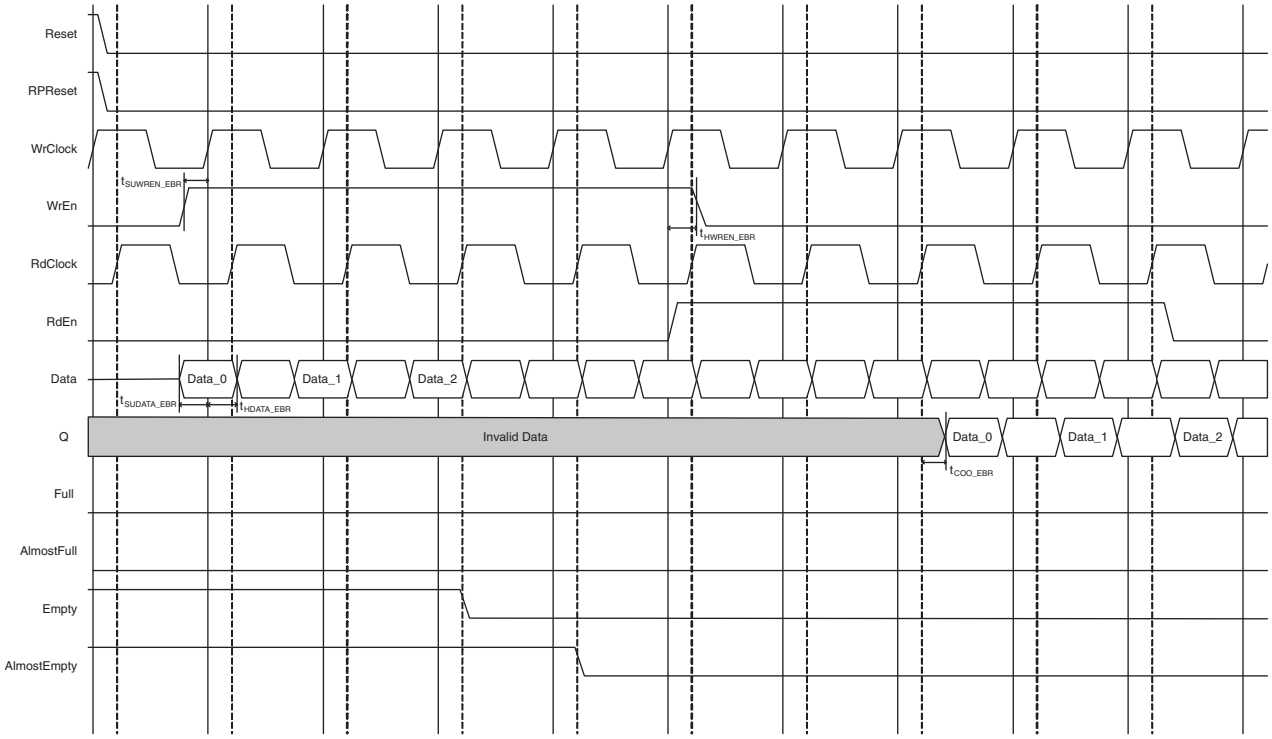


Figure 10-27. FIFO_DC with Output Registers (Pipelined)

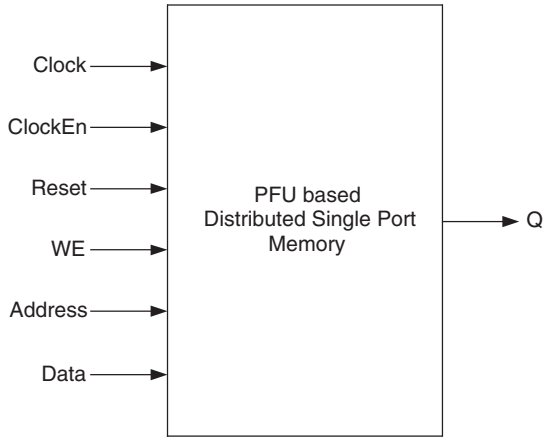


Distributed Single Port RAM (Distributed_SPRAM) – PFU Based

PFU-based Distributed Single Port RAM is created using the 4-input LUT (Look-Up Table) available in the PFU. These LUTs can be cascaded to create larger Distributed Memory sizes.

Figure 10-28 shows the Distributed Single Port RAM module as generated by IPexpress.

Figure 10-28. Distributed Single Port RAM Module Generated by IPexpress



The generated module makes use 4-input LUT available in the PFU. Additional logic like Clock, Reset is generated by utilizing the resources available in the PFU.

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

The various ports and their definitions for the memory are as per Table 10-18. The table lists the corresponding ports for the module generated by IPexpress and for the primitive.

Table 10-18. PFU-based Distributed Single Port RAM Port Definitions

Port Name in Generated Module	Port Name in the PFU Primitive	Description	Active State
Clock	CK	Clock	Rising Clock Edge
ClockEn	—	Clock Enable	Active High
Reset	—	Reset	Active High
WE	WRE	Write Enable	Active High
Address	AD[3:0]	Address	—
Data	DI[1:0]	Data In	—
Q	DO[1:0]	Data Out	—

Ports such as Clock Enable (ClockEn) are not available in the hardware primitive. These are generated by IPexpress when the user wishes to enable the output registers in the IPexpress configuration.

Users have the option of enabling the output registers for Distributed Single Port RAM (Distributed_SPRAM). Figures 10-29 and 10-30 show the internal timing waveforms for the Distributed Single Port RAM (Distributed_SPRAM) with these options.

Figure 10-29. PFU Based Distributed Single Port RAM Timing Waveform - without Output Registers

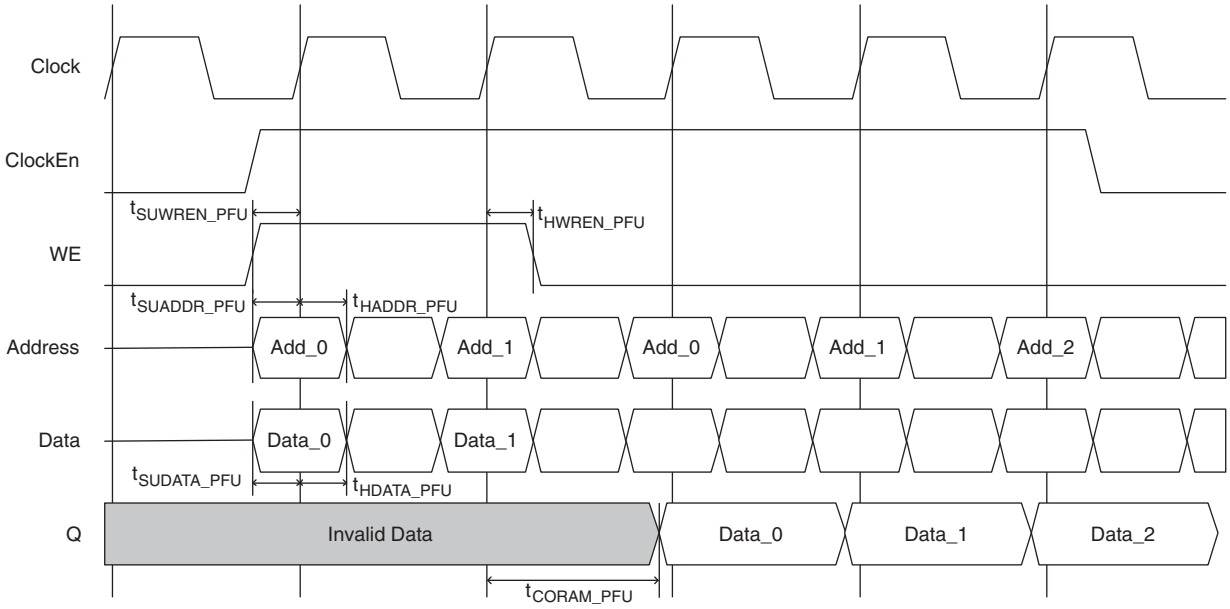
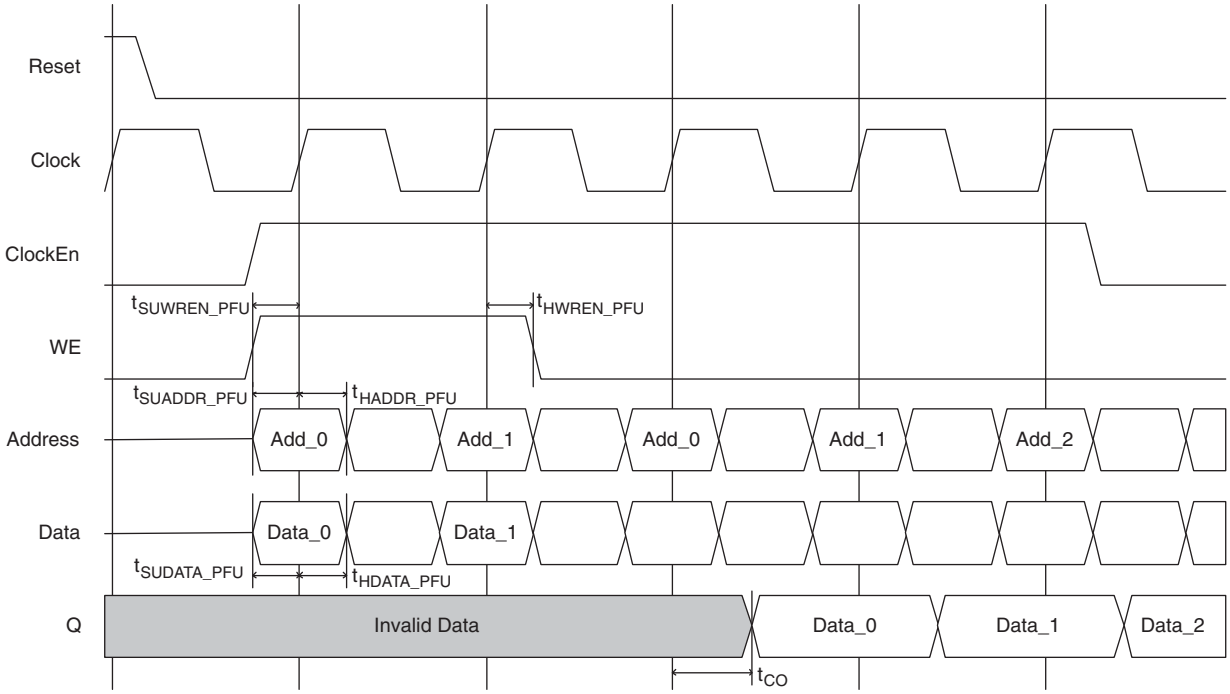


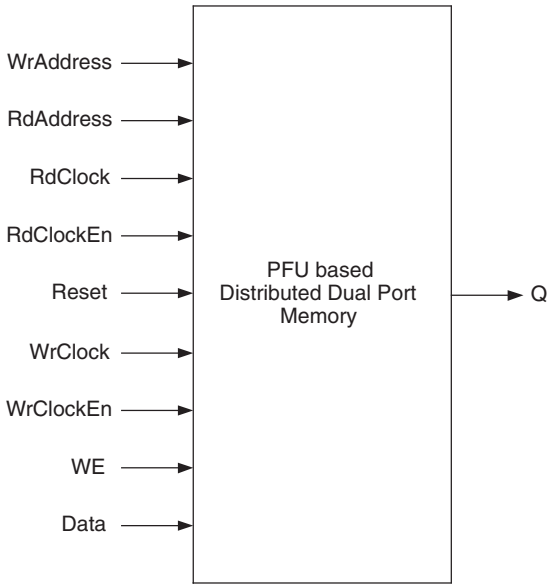
Figure 10-30. PFU Based Distributed Single Port RAM Timing Waveform - with Output Registers



Distributed Dual Port RAM (Distributed_DPRAM) – PFU Based

PFU-based Distributed Dual Port RAM is also created using the 4-input LUT (Look-Up Table) available in the PFU. These LUTs can be cascaded to create a larger Distributed Memory sizes.

Figure 10-31. Distributed Dual Port RAM Module Generated by IPexpress



The generated module makes use of the 4-input LUT available in the PFU. Additional logic like Clock and Reset is generated by utilizing the resources available in the PFU.

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

The various ports and their definitions for memory are as per Table 10-19. The table lists the corresponding ports for the module generated by IPexpress and for the primitive.

Table 10-19. PFU-based Distributed Dual-Port RAM Port Definitions

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

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

Users have the option of enabling the output registers for Distributed Dual Port RAM (Distributed_DPRAM). Figures 10-32 and 10-33 show the internal timing waveforms for the Distributed Dual Port RAM (Distributed_DPRAM) with these options.

Figure 10-32. PFU Based Distributed Dual Port RAM Timing Waveform - without Output Registers

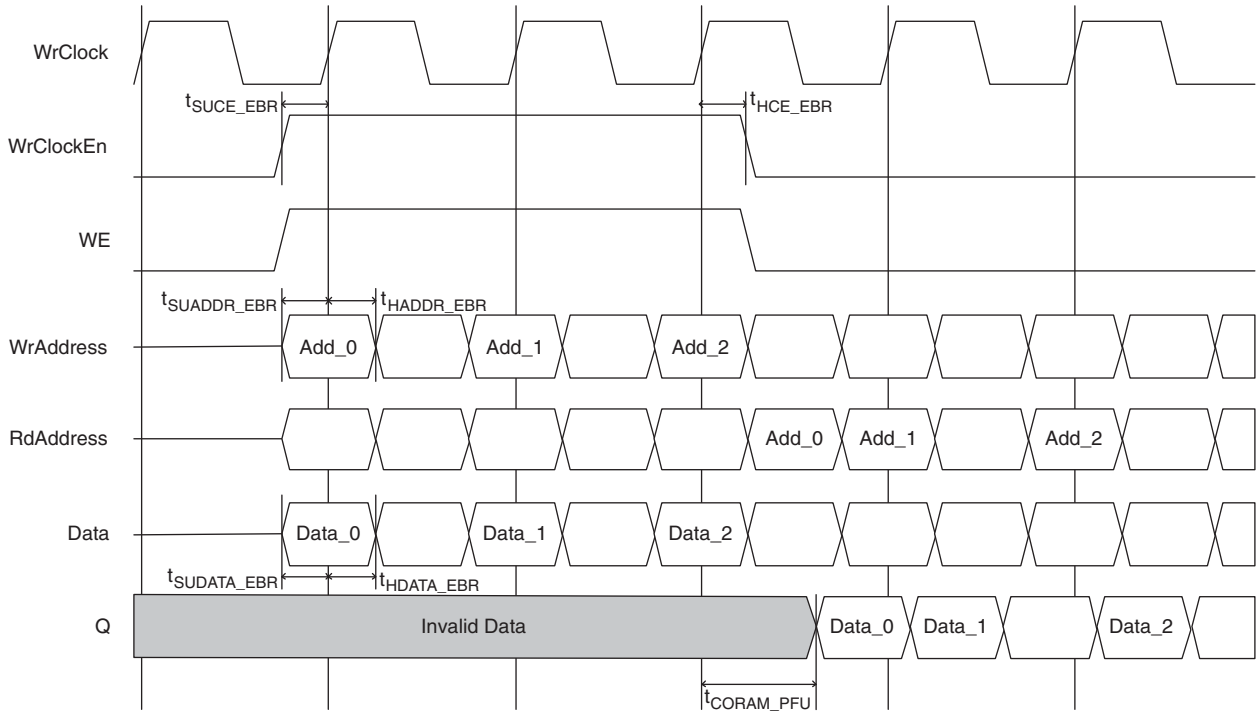
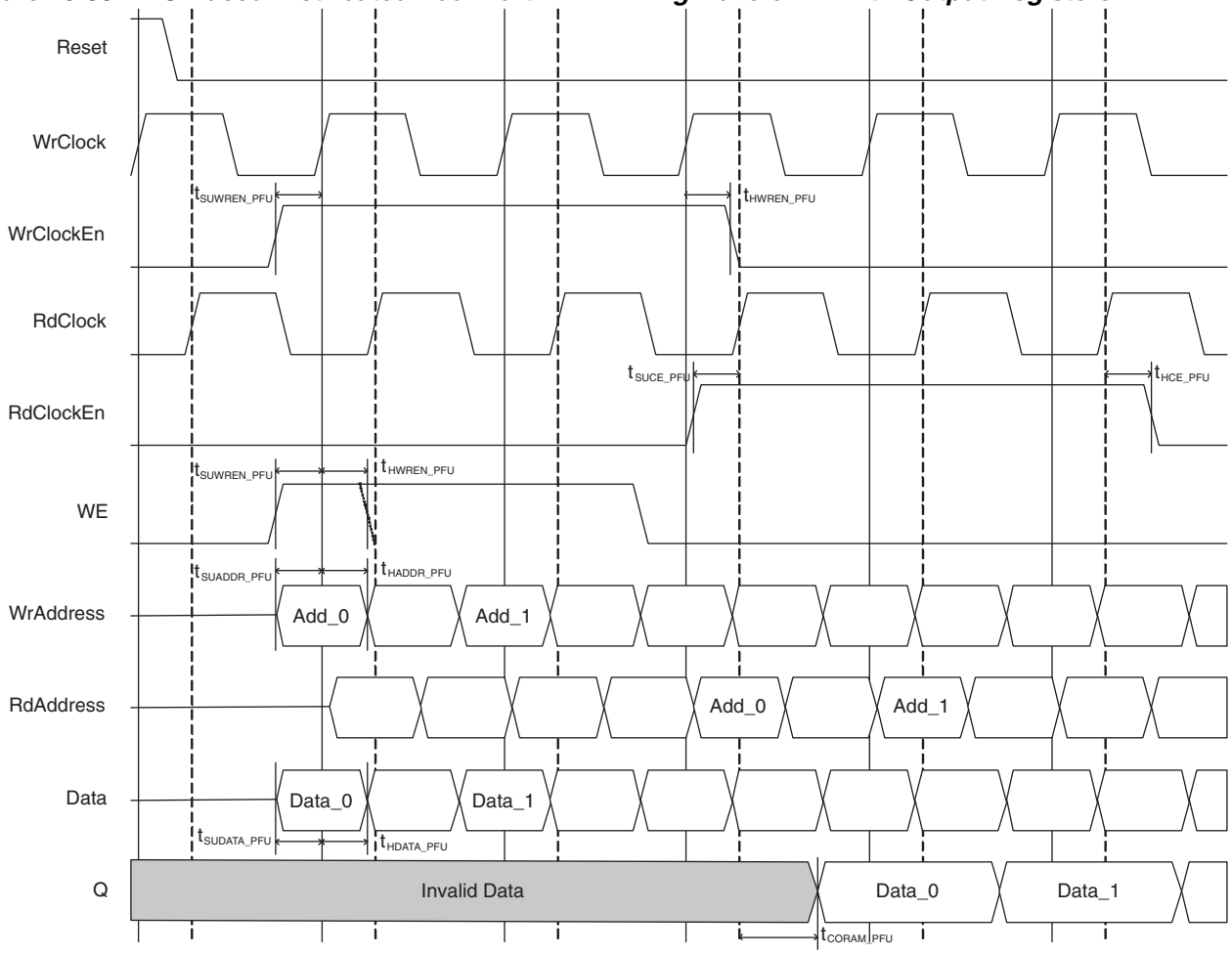


Figure 10-33. PFU Based Distributed Dual Port RAM Timing Waveform - with Output Registers

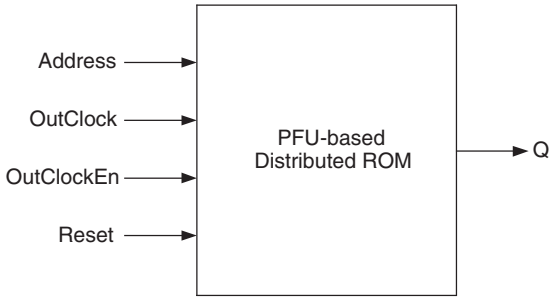


Distributed ROM (Distributed_ROM) – PFU Based

PFU-based Distributed ROM is also created using the 4-input LUT (Look-Up Table) available in the PFU. These LUTs can be cascaded to create larger Distributed Memory sizes.

Figure 10-34 shows the Distributed ROM module as generated by IPexpress.

Figure 10-34. Distributed ROM Generated by IPexpress



The generated module makes use of the 4-input LUT available in the PFU. Additional logic like Clock and Reset is generated by utilizing the resources available in the PFU.

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

The various ports and their definitions for memory are as per Table 10-20. The table lists the corresponding ports for the module generated by IPexpress and for the primitive.

Table 10-20. PFU-based Distributed ROM Port Definitions

Port Name in Generated Module	Port Name in the PFU Block Primitive	Description	Active State
Address	AD[3:0]	Address	—
OutClock	—	Out Clock	Rising Clock Edge
OutClockEn	—	Out Clock Enable	Active High
Reset	—	Reset	Active High
Q	DO	Data Out	—

Users have the option to enable the output registers for Distributed ROM (Distributed_ROM). Figures 10-35 and 10-36 show the internal timing waveforms for the Distributed ROM with these options.

Figure 10-35. PFU Based ROM Timing Waveform – without Output Registers

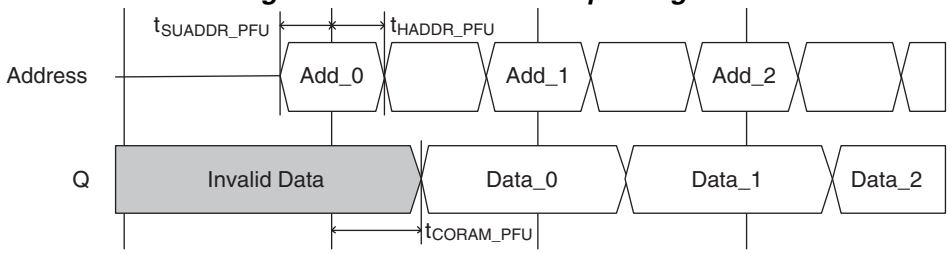
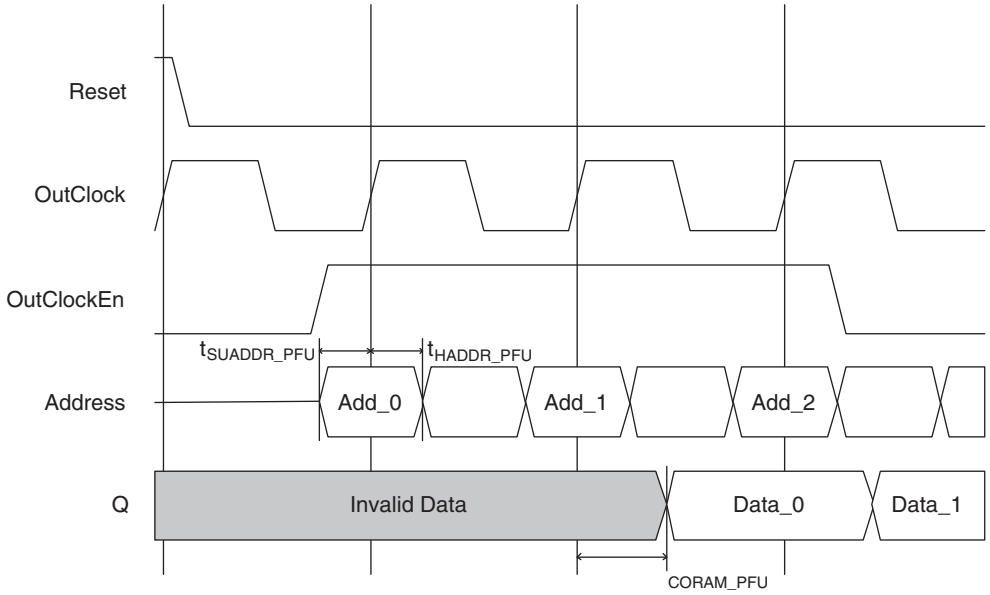


Figure 10-36. PFU Based ROM Timing Waveform – with Output Registers



User TAG Memory

The TAG memory is an area on the on-chip Flash which can be used for non-volatile storage. It is accessed in your design as if it were an external SPI Flash. Both SPI bus operation modes 0 (0,0) and 3 (1,1) are supported.

Table 10-21. TAG Memory Size

Device	TAG Memory (Bits)	TAG Memory (Bytes)
XP2-5	632	79
XP2-8	768	96
XP2-17	2184	273
XP2-30	2640	330
XP2-40	3384	423

Figure 10-37. SSPIA Primitive

Table 10-22. User TAG Memory Signal Description

Primitive Port Name	Description
SI	Data input
SO	Data output
CLK	Clock
CS	Chip select

Serial Data Input (SI)
The SPI Serial Data Input pin provides a means for commands and data to be serially written to (shifted into) the device. Data is latched on the rising edge of serial clock (CLK) input pin.

Serial Data Output (SO)
The SPI Serial Data Output pin provides a means for status and data to be serially read out (shifted out of) the device. Data is shifted out on the falling edge of serial clock (CLK) input pin.

Serial Clock (CLK)
The SPI Serial Clock Input pin provides the timing for serial input and output operations.

Chip Select (CS)
The SPI Chip Select pin enables and disables SPI interface operations. When the Chip Select is high the SPI interface is deselected and the Serial Data Output (SO) pin is at high impedance. When it is brought low, the SPI interface is selected, commands can be written into and data read from the device. After power up, CS must transit from high to low before a new command can be accepted.

Programming via the JTAG Interface

.VME files can be generated for ispVM which only programs the TAG memory. These .VME files are handled according to the standard ispVME flow.

Programming via the SPI Interface

Since the SSPIA module is an internal module, I/Os can be treated as I/Os of any other soft module. Therefore, they can be routed to other internal modules, or they can go out to I/O pads. The recommended routing is to the sysCONFIG port pins.

Table 10-23. Usage Of Commands

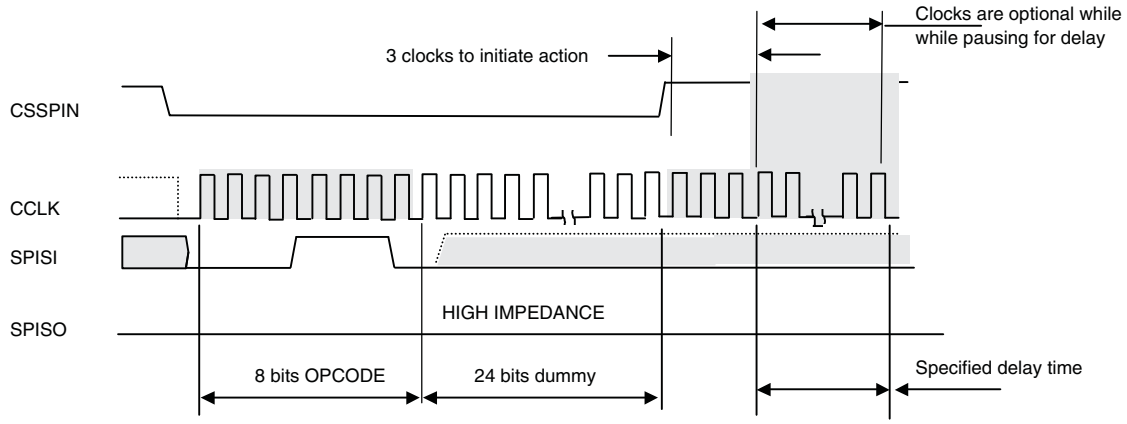
Command Name	OPCODE	Bytes 1-3[1, 2]	Data	Delay Time	Description
READ_TAG	0x4E	Dummy	Out	5µs min.	Read TAG memory
PROGRAM_TAG	0x8E	Dummy	In	1ms min., 25ms max.	Program TAG memory
ERASE_TAG	0x0E	Dummy		100ms min., 1000ms max.	Erase TAG memory

1. Data bytes are shifted with most significant bit first.
2. Byte 1-3 are dummy clocks to provide extra timing for the device to execute the command. The data presented at the SI pin during these dummy clocks can be any value and do not have to be 0x00 as shown.

Erase_Tag Commands Waveform

The Erase_Tag command executes an erase of the TAG memory. Use the PROGRAM_STATUS command to make sure the action has been completed to eliminate the possibility of more than one action in progress causing action clashes.

Figure 10-38. Erase_Tag Commands Waveform

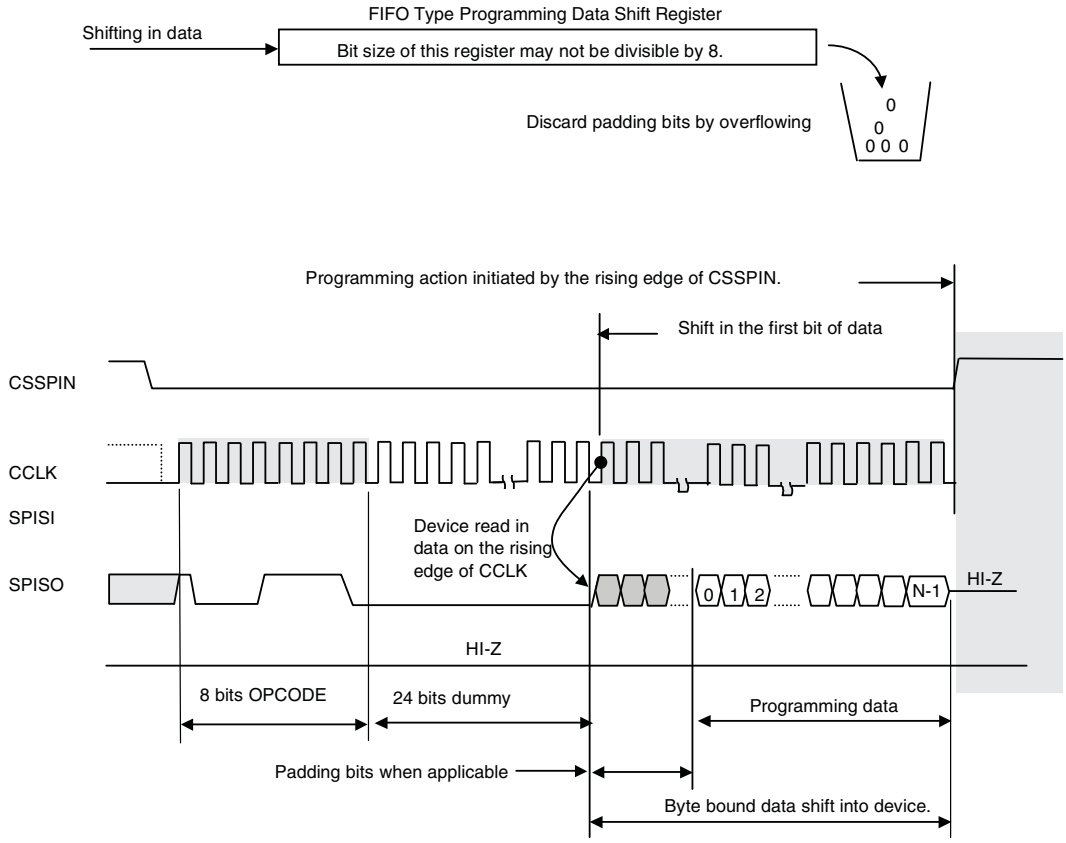


Program_Tag Commands Waveform

All the programming data registers are constructed as a FIFO. The exact amount of programming data (in number of bits) must be provided to the selected programming data register. If over-shift or under-shift occurs, the resulting data programmed into the device will be invalid.

The configuration programming data may not be byte-bounded. Leading padding bits of 0 can be used to pad the configuration data into the byte bound hex file format. When configuration programming data is shifted into the device, the input shift register's FIFO will overflow, causing the leading padding bits to be discarded.

Figure 10-39. Program_Tag Commands Waveform



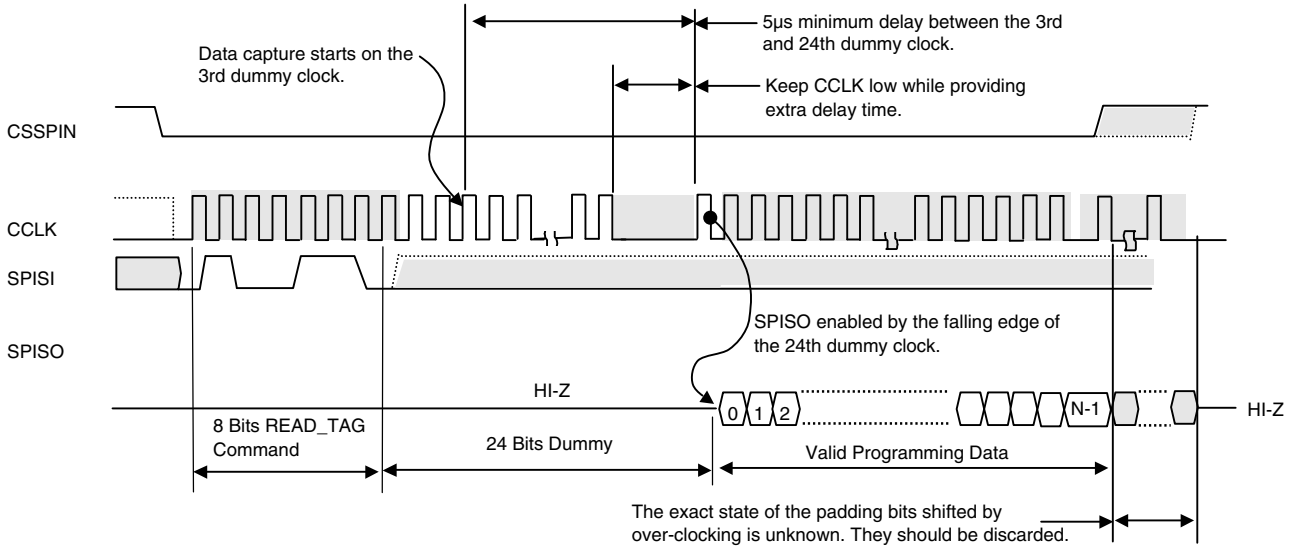
Read_Tag Commands Waveform

When reading TAG Memory data, there is a ~3µs verify delay time required to provide the device the time it needs to physically transfer the data stored in Flash cells into the FIFO data shift register. The device considers the 24 clocks for shifting in the dummy bits as a substitution for the delay. If the 24 clocks are applied faster than 1µs, then apply the extra delay by holding the CCLK low before sampling SPISO.

This command must be qualified by first shifting in the X_PROGRAM_EN command, which is equivalent to the WRITE_EN command in standard SPI Flash devices. If this does not occur, the transfer action will not take place, resulting in unknown data shifting out.

If the programming data size is not divisible by 8, trailing padding clocks can be used to shift out byte-bound data and then discard the residual bits clocked out by the trailing padding clocks.

Figure 10-40. Read_Tag Commands Waveform



Initializing Memory

In the EBR based ROM or RAM memory modes and the PFU based ROM memory mode, it is possible to specify the power-on state of each bit in the memory array. Each bit in the memory array can have one of two values: 0 or 1.

Initialization File Format

The initialization file is an ASCII file, which users can create or edit using any ASCII editor. IPexpress supports three types of memory file formats:

- Binary file
- Hex File
- Addressed Hex

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

The Initialization File is primarily used for configuring the ROMs. The EBR in RAM mode can optionally use this Initialization File also to preload the memory contents.

The TAG memory uses hex or binary non-addressed files. Since it is a SPI, it cannot use the addressed hex file.

Binary File

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

```
Memory Size 20x32
00100000010000000010000001000000
00000001000000010000000100000001
00000010000000100000001000000010
00000011000000110000001100000011
00000100000001000000010000000100
00000101000001010000010100000101
00000110000001100000011000000110
00000111000001110000011100000111
00001000010010000000100001001000
00001001010010010000100101001001
00001010010010100000101001001010
00001011010010110000101101001011
00001100000011000000110000001100
00001101001011010000110100101101
00001110001111100000111000111110
00001111001111110000111100111111
00010000000100000001000000010000
00010001000100010001000100010001
00010010000100100001001000010010
00010011000100110001001100010011
```

Hex File

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

```
Memory Size 8x16
A001
0B03
1004
CE06
0007
040A
0017
02A4
```

Addressed Hex

Addressed Hex consists of lines of address and data. Each line starts with an address, followed by a colon, and any number of data. The format of memfile is address: data data data data ... where address and data are hexadecimal numbers.

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

The first line puts 03 at address A0, F3 at address A1, 3E at address A2,and 4F at address A3. The second line puts 3B at address B2 and 9F at address B3.

There is no limitation on the values of address and data. The value range is automatically checked based on the values of addr_width and data_width. If there is an error in an address or data value, an error message is printed.

Users need not specify data at all address locations. If data is not specified at certain address, the data at that location is initialized to 0. IPexpress makes memory initialization possible both through the synthesis and simulation flows.

FlashBak™ Capability

The LatticeXP2 FPGA family offers FlashBak capability, which is a way to store the data in the EBRs to the Flash memory upon user command. This protects the user's data from being lost when the system is powered off. The FlashBak module (STFA primitive) has a single-command-two-operation process (see Figure 10-41). When the FlashBak operation is initiated, an erase-UFM-Flash signal is enabled to erase the Flash, followed by the transfer-to-flash operation. Once the transfer is done, the Flash controller sends a transfer-done signal back to the user logic. During the FlashBak operation, the EBRs are not accessible. There is no difference between the regular EBR RAM configuration and the shadow Flash (UFM) EBR RAM configuration in the ispLEVER GUI. The presence of the STFA (FlashBak) primitive in a design determines the EBR RAM configuration. FlashBak cannot be used if the soft-error detect (SED) is operating in an Always mode. Since there is no addressing but just a 'dump' of all EBR to Flash, only one STFA module is necessary. Multiple modules are not necessary or allowed.

Figure 10-41. FlashBak Primitive

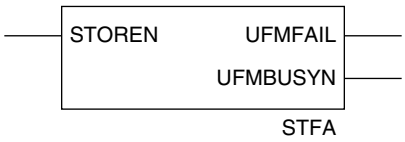


Table 10-24. STFA Port Descriptions

Port Name	Corresponding Hardware Port Name	I/O	Description
STOREN	storecmdn	I	Initiates to store the EBR content to Flash
UFMFAIL	ufm_fail	O	Store to Flash operation failed
UFMBUSYN	fl_busyn	O	Tells the user whether the Flash is in the busy state or not

Technical Support Assistance

Hotline: 1-800-LATTICE (North America)
+1-503-268-8001 (Outside North America)
e-mail: techsupport@latticesemi.com
Internet: www.latticesemi.com

Revision History

Date	Version	Change Summary
February 2007	01.0	Initial release.
July 2007	01.1	Added FlashBak Capability section.
November 2007	01.2	TAG memory added.
January 2008	01.3	Updated Read_Tag Commands Waveform diagram. Changed minimum delay between the 3rd and 24th dummy clock from 3µs to 5µs.

Appendix A. Attribute Definitions

DATA_WIDTH

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

REGMODE

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

RESETMODE

The RESETMODE attribute allows users to select the mode of reset in the memory. This attribute is associated with the block RAM elements. RESETMODE takes two parameters: SYNC and ASYNC. SYNC means that the memory reset is synchronized with the clock. ASYNC means that the memory reset is asynchronous to clock.

CSDECODE

CSDECODE or the Chip Select Decode attributes are associated to block RAM elements. CS, or Chip Select, is the port available in the EBR primitive that is useful when memory requires multiple EBR blocks cascaded. The CS signal forms the MSB for the address when multiple EBR blocks are cascaded. CS is a 3-bit bus, so it can cascade eight memories easily. CSDECODE takes the following parameters: “000”, “001”, “010”, “011”, “100”, “101”, “110”, and “111”. CSDECODE values determine the decoding value of CS[2:0]. CSDECODE_W is chip select decode for write and CSDECODE_R is chip select decode for read for Pseudo Dual Port RAM. CSDECODE_A and CSDECODE_B are used for true dual port RAM elements and refer to the A and B ports.

WRITEMODE

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

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

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

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

WRITEMODE_A and WRITEMODE_B are used for dual port RAM elements and refer to the A and B ports in case of a True Dual Port RAM.

For all modes (of the True Dual Port module), simultaneous read access from one port and write access from the other port to the same memory address is not recommended. The read data may be unknown in this situation. Also, simultaneous write access to the same address from both ports is not recommended. (When this occurs, the data stored in the address becomes undetermined when one port tries to write a ‘H’ and the other tries to write a ‘L’).

It is recommended that the designer implements control logic to identify this situation if it occurs and either:

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

GSR

GSR or the Global Set/ Reset attribute is used to enable or disable the global set/reset for RAM element.

LatticeXP2 High-Speed I/O Interface

May 2007 Technical Note TN1138

Introduction

LatticeXP2™ devices support Double Data Rate (DDR) and Single Data Rate (SDR) interfaces using the logic built into the Programmable I/O (PIO). SDR applications capture data on one edge of a clock while the DDR interfaces capture data on both the rising and falling edges of the clock, thus doubling performance. The LatticeXP2 I/Os also have dedicated circuitry to support DDR and DDR2 SDRAM memory interfaces. This technical note details the use of LatticeXP2 devices to implement both a high-speed generic DDR interface and DDR and DDR2 memory interfaces.

DDR and DDR2 SDRAM Interfaces Overview

A DDR SDRAM interface will transfer data at both the rising and falling edges of the clock. The DDR2 is the second generation of the DDR SRDRAM memory.

The DDR and DDR2 SDRAM interfaces rely on the use of a data strobe signal, called DQS, for high-speed operation. The DDR SDRAM interface uses a single-ended DQS strobe signal, whereas the DDR2 interface uses a differential DQS strobe. Figures 11-1 and 11-2 show typical DDR and DDR2 SDRAM interface signals. SDRAM interfaces are typically implemented with eight DQ data bits per DQS. An x16 interface will use two DQS signals and each DQS is associated with eight DQ bits. Both the DQ and DQS are bi-directional ports used to both read and write to the memory.

When reading data from the external memory device, data coming into the device is edge-aligned with respect to the DQS signal. This DQS strobe signal needs to be phase-shifted 90 degrees before FPGA logic can sample the read data. When writing to a DDR/DDR2 SDRAM, the memory controller (FPGA) must shift the DQS by 90 degrees to center-align with the data signals (DQ). A clock signal is also provided to the memory. This clock is provided as a differential clock (CLKP and CLKN) to minimize duty cycle variations. The memory also uses these clock signals to generate the DQS signal during a read via a DLL inside the memory. Figures 11-3 and 11-4 show DQ and DQS timing relationships for read and write cycles. For other detailed timing requirements, please refer to the DDR SDRAM JEDEC specification (JESD79C).

During read, the DQS signal is LOW for some duration after it comes out of tristate. This state is called Preamble. The state when the DQS is LOW before it goes into Tristate is the Postamble state. This is the state after the last valid data transition.

DDR SDRAM also requires a Data Mask (DM) signal to mask data bits during write cycles. Note that the ratio of DQS to data bits is independent of the overall width of the memory. An 8-bit interface will have one strobe signal.

DDR SDRAM interfaces use the SSTL25 Class I/II I/O standards whereas the DDR2 SDRAM interface uses the SSTL18 Class I/II I/O standards. The DDR2 SDRAM interface also supports differential DQS (DQS and DQS#).

Figure 11-1. Typical DDR SDRAM Interface

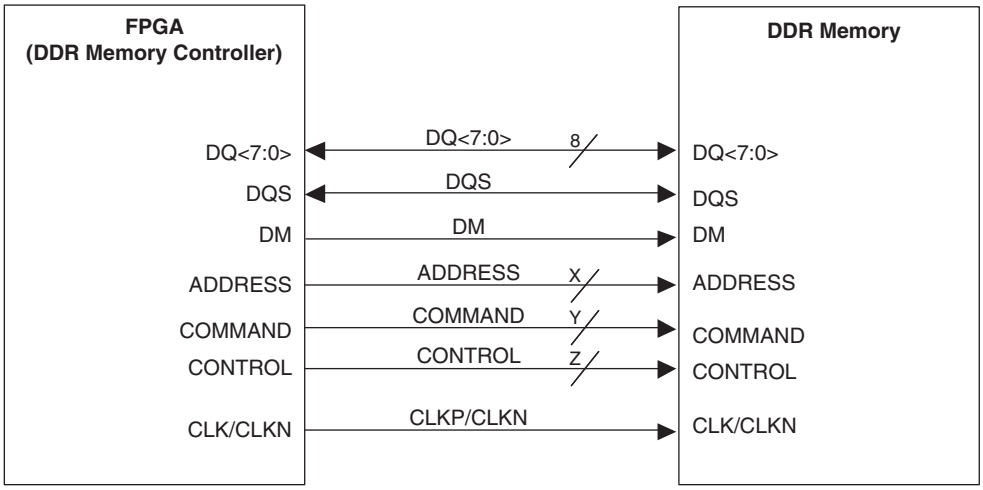
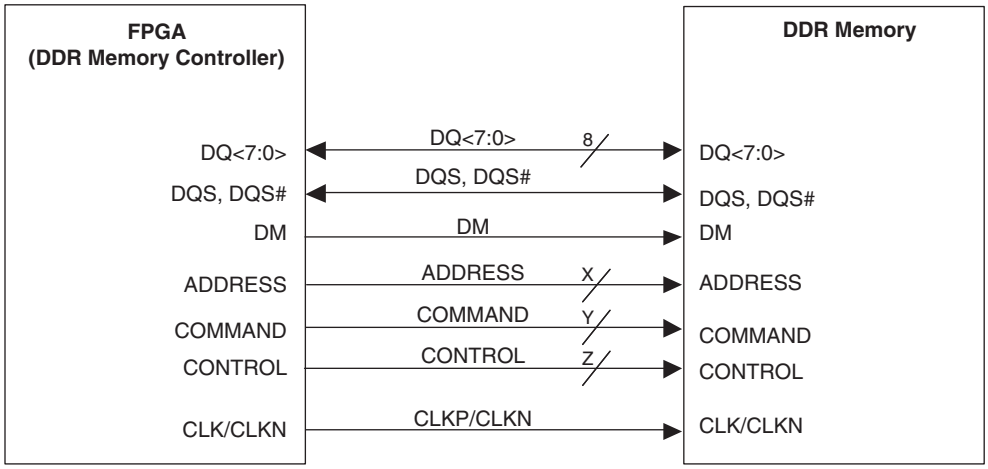


Figure 11-2. Typical DDR2 SDRAM Interface



The following two figures show the DQ and DQS relationship for memory read and write interfaces.

Figure 11-3. DQ-DQS During READ

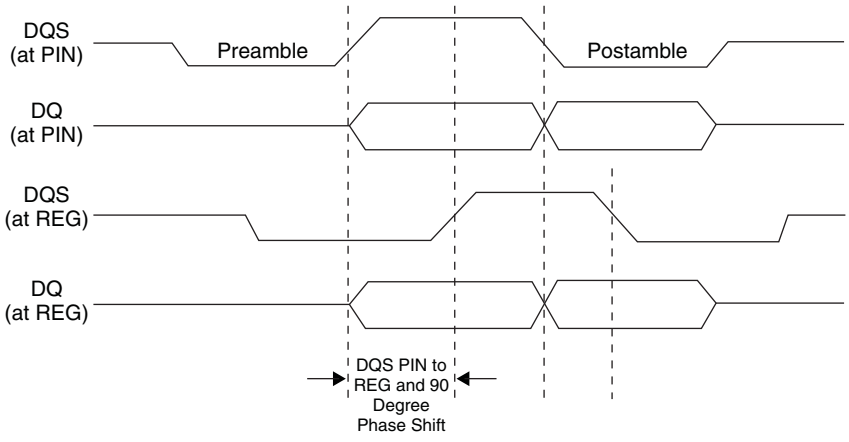
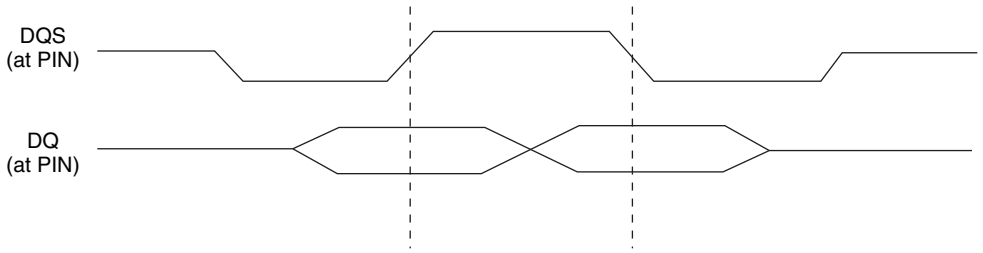


Figure 11-4. DQ-DQS During WRITE



Implementing DDR Memory Interfaces with LatticeXP2 Devices

As described in the DDRSDRAM overview section, the DDR SDRAM interfaces rely primarily on the use of a data strobe signal called DQS for high-speed operation. When reading data from the external memory device, data coming into the LatticeXP2 device is edge-aligned with respect to the DQS signal. Therefore, the LatticeXP2 device needs to shift the DQS (a 90-degree phase shift) before using it to sample the read data. When writing to a DDR SDRAM, the memory controller from the LatticeXP2 device must generate a DQS signal that is center-aligned with the DQ, the data signals. This is accomplished by ensuring the DQS strobe is 90 degrees ahead relative to DQ data.

LatticeXP2 devices have dedicated DQS support circuitry for generating the appropriate phase shifting for DQS. The DQS phase shift circuit uses a frequency reference DLL to generate delay control signals associated with each of the dedicated DQS pins and is designed to compensate for process, voltage and temperature (PVT) variations. The frequency reference is provided through one of the global clock pins.

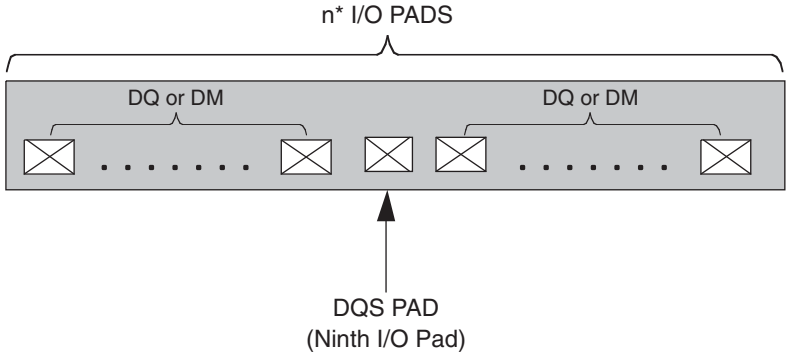
The dedicated DDR support circuit is also designed to provide comfortable and consistent margins for data sampling window.

This section describes how to implement the read and write sections of a DDR memory interface. It also provides details of the DQ and DQS grouping rules associated with the LatticeXP2 devices.

DQS Grouping

Each DQS group generally consists of at least 10 I/Os (one DQS, eight DQ and one DM) to implement a complete 8-bit DDR memory interface. LatticeXP2 devices support DQS signals on all sides of the device. Each DQS signal on the top and bottom halves of the device will span across 18 I/Os and on the left and right sides of the device will span across 16 I/Os. Any 10 of these I/Os spanned by the DQS can be used to implement an 8-bit DDR memory interface. In addition to the DQS grouping, the user must also assign one reference voltage VREF1 for a given I/O bank.

Figure 11-5. DQ-DQS Grouping



*n=18 on bottom banks and n=16 on the left and right side banks.

Figure 11-5 shows a typical DQ-DQS group for LatticeXP2 devices. The ninth I/O of this group of 16 or 18 I/Os is the dedicated DQS pin. The 8 pads before of the DQS and 6/9 (6 for left and right side and 9 for top and bottom side) pads after the DQS are covered by the DQS bus span. Users can assign any eight of these I/O pads to be DQ data pins. Hence, to implement a 32-bit wide memory interface you would need to use four such DQ-DQS groups.

When not interfacing with the memory, the dedicated DQS pin can be used as a general purpose I/O. Each of the dedicated DQS pin is internally connected to the DQS phase shift circuitry. The pinout information contained in the LatticeXP2 Family Data Sheet shows pin locations for the DQS pads.

DDR Software Primitives

This section describes the software primitives that can be used to implement DDR interfaces. These primitives include:

- **DQSDLL** – The DQS delay calibration DLL
- **DQSBUFC** – The DQS delay function and the clock polarity selection logic
- **IDDRMX1A** – The DDR input and DQS to system clock transfer registers with half clock cycle transfer
- **IDDRMFX1A** – The DDR input and DQS to system clock transfer registers with full clock cycle transfer
- **ODDRMXA** – The DDR output registers

HDL usage examples for each of these primitives are listed in Appendices A and B.

DQSDLL

The DQSDLL generates a 90-degree phase shift required for the DQS signal. This primitive implements the on-chip DQSDLL. Only one DQSDLL should be instantiated for all the DDR implementations on one half of the device. The clock input to this DLL should be at the same frequency as the DDR interface. The DLL generates the delay based on this clock frequency and the update control input to this block. The DLL updates the dynamic delay control to the DQS delay block when this update control (UDDCNTL) input is asserted. Figure 11-6 shows the primitive symbol. The active low signal on UDDCNTL updates the DQS phase alignment and should be initiated at the beginning of READ cycles.

Figure 11-6. DQSDLL Symbol

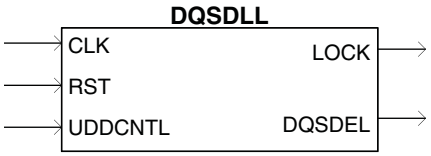


Table 11-1 provides a description of the ports.

Table 11-1. DQSDLL Ports

Port Name	I/O	Description
CLK	I	System CLK should be at the frequency of the DDR interface from the FPGA core.
RST	I	Resets the DQSDLL
UDDCNTL	I	Provides update signal to the DLL that will update the dynamic delay.
LOCK	O	Indicates when the DLL is in phase.
DQSDEL	O	The digital delay generated by the DLL, should be connected to the DQSBUF primitive.

DQSDLL Update Control: The DQS Delay can be updated for PVT variation using the UDDCNTL input. The DQSDEL is updated when the when the UDDCNTL is held LOW. The DQSDEL can be updated when variations are expected. It can be updated for every READ cycle, when READ is not enabled. This signal should be asserted at least one clock cycle before READ is asserted.

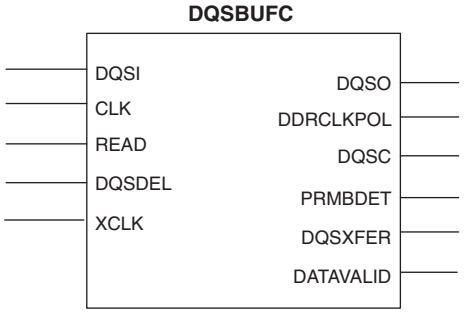
DQSDLL Configuration: By default, this DLL will generate a 90-degree phase shift for the DQS strobe based on the frequency of the input reference clock to the DLL. The user can control the sensitivity to jitter by using the LOCK_SENSITIVITY attribute. This configuration bit can be programmed to be either “HIGH” or “LOW”.

The DLL Lock Detect circuit has two modes of operation controlled by the LOCK_SENSITIVITY bit, which selects more or less sensitivity to jitter. If this DLL is operated at or above 150 MHz, it is recommended that the LOCK_SENSITIVITY bit be programmed “HIGH” (more sensitive). When running at or below 100 MHz, it is recommended that the bit be programmed “LOW” (more tolerant). For 133 MHz, the LOCK_SENSITIVITY bit can go either way.

DQSBUFC

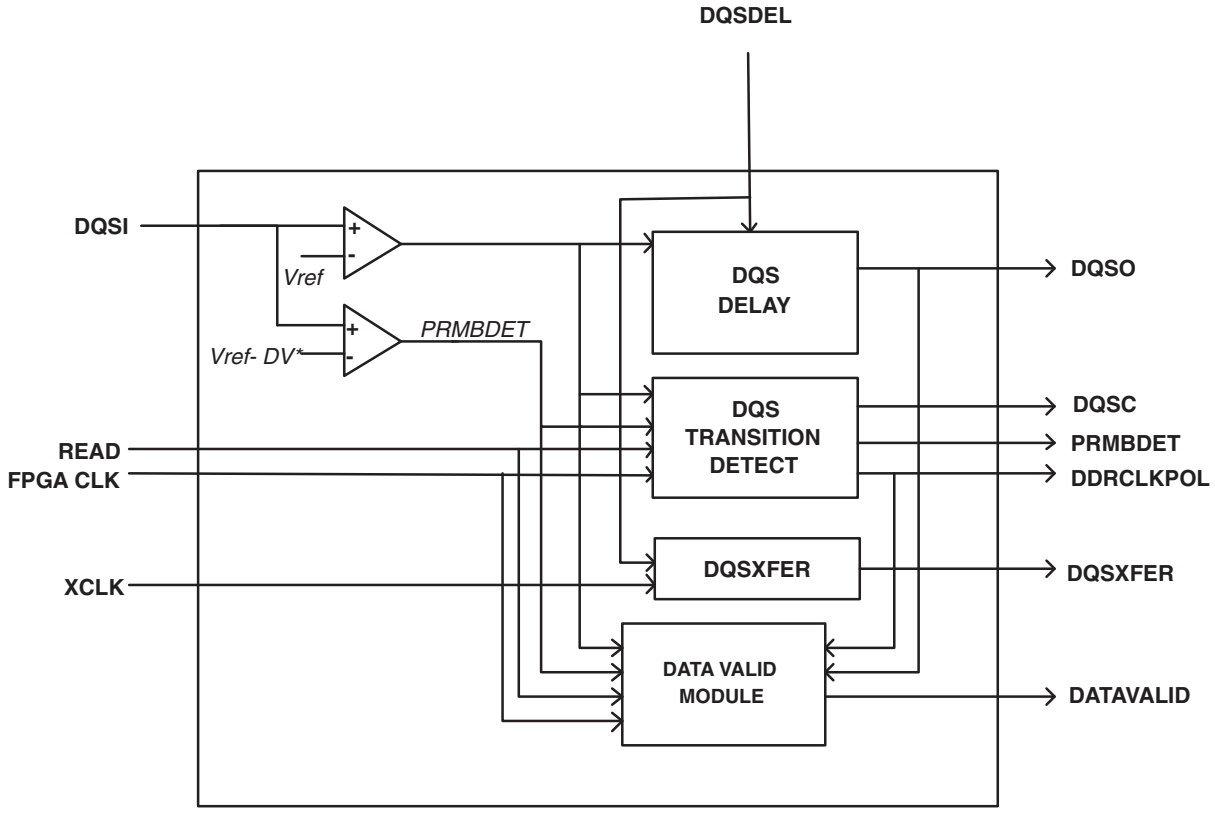
This primitive implements the DQS delay and the DQS transition detector logic. Figure 11-7 shows the primitive symbol.

Figure 11-7. DQSBUFC Symbol



The DQSBUFC is composed of the DQS Delay, the DQS Transition Detect and the DQSXFER block as shown in Figure 11-8. This block inputs the DQS and delays it by 90 degrees. It also generates the DDR Clock Polarity and the DQSXFER signal. The preamble detect (PRMBDET) signal is generated from the DQSI input using a voltage divider circuit.

Figure 11-8. DQSBUFC Function



**DV ~ 170mV for DDR1 (SSTL25 signaling)*
**DV ~ 120mV for DDR2 (SSTL18 signaling)*

DQS Delay Block: The DQS Delay block receives the digital control delay line (DQSDEL) coming from one of the two DQSDLL blocks. These control signals are used to delay the DQSI by 90 degrees. DQSO is the delayed DQS and is connected to the clock input of the first set of DDR registers.

DQS Transition Detect: The DQS Transition Detect block generates the DDR Clock Polarity signal based on the phase of the FPGA clock at the first DQS transition. The DDR READ control signal and FPGA CLK inputs to this coming and should be coming from the FPGA core.

DQSXFER: This block generates the 90-degree phase shifted clock to for the DDR Write interface. The input to this block is the XCLK. The user can choose to connect this either to the edge clock or the FPGA clocks. The DQSXFER is routed using the DQSXFER tree to all the I/Os spanned by that DQS.

Data Valid Module: The data valid module generates a DATAVALID signal. This signal indicates to the FPGA that valid data is transmitted out of the input DDR registers to the FPGA core.

Table 11-2 provides a description of the I/O ports associated with the DQSBUFC primitive.

Table 11-2. DQSBUFC Ports

Port Name	I/O	Description
DQSI	I	DQS Strobe signal from memory
CLK	I	System CLK
READ	I	Read generated from the FPGA core
DQSDEL	I	DQS Delay from the DQSDLL primitive
XCLK	I	Edge Clock or System CLK
DQSO	O	Delayed DQS Strobe signal, to the input capture register block
DQSC	O	DQS Strobe signal before delay, going to the FPGA core logic
DDRCLKPOL	O	DDR Clock Polarity signal
PRMBDET	O	Preamble detect signal, going to the FPGA core logic
DQSXFER	O	90 degree shifted clock going to the Output DDR register Block
DATAVALID	O	Signal indicating transmission of Valid data to the FPGA core

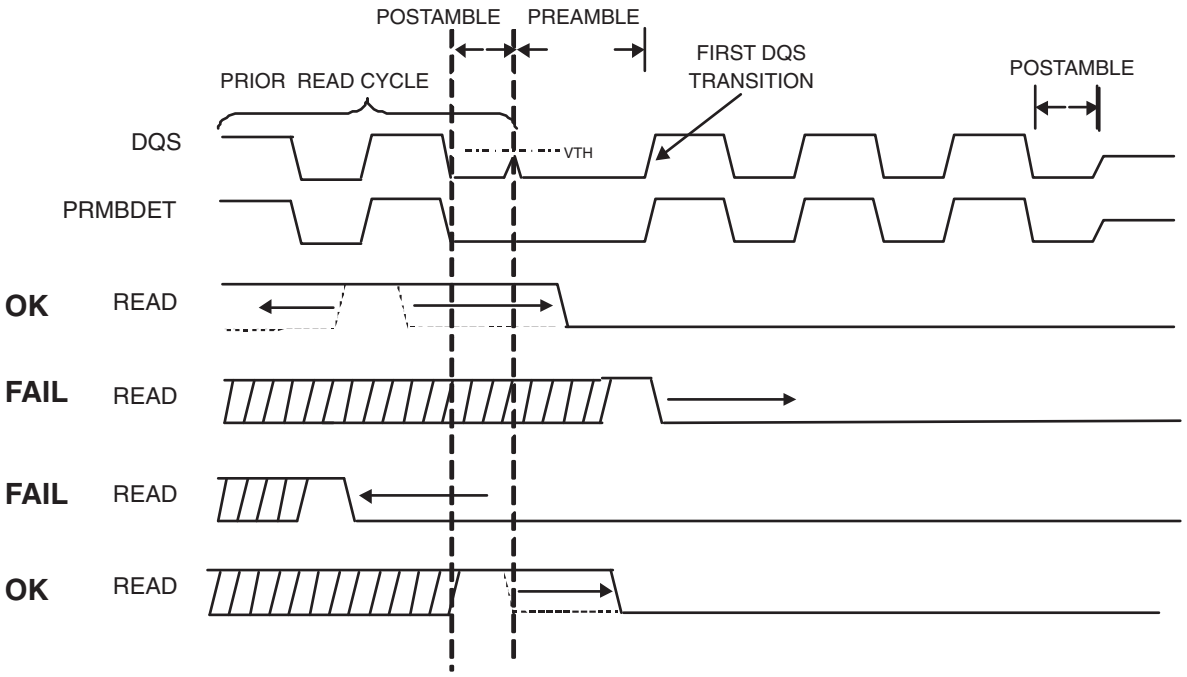
READ Pulse Generation

The READ signal to the DQSBUFC block is internally generated in the FPGA core. The READ signal goes high when the READ command to control the DDR-SDRAM is initially asserted. This precedes the DQS preamble by one cycle, yet may overlap the trailing bits of a prior read cycle. The DQS Detect circuitry of the LatticeXP2 device requires the falling edge of the READ signal to be placed within the preamble stage.

The preamble state of the DQS can be detected using the CAS latency and the round trip delay for the signals between the FPGA and the memory device. Note that the internal FPGA core generates the READ pulse. The rise of the READ pulse should coincide with the initial READ command of the Read Burst and need to go low before the Preamble goes high.

Figure 11-9 shows a READ Pulse timing example with respect to the PRMBDET signal.

Figure 11-9. READ Pulse Generation



IDDRMX1A

This primitive will implement the input register block in memory mode. The DDR registers are designed to use edge clock routing on the I/O side and the primary clock on the FPGA side. The ECLK input is used to connect to the DQS strobe coming from the DQS delay block (DQSBUFC primitive). The SCLK input is connected to the system (FPGA) clock. DDRCLKPOL is an input from the DQS Clock Polarity tree. This signal is generated by the DQS Transition detect circuit in the hardware. The DDRCLKPOL signal is used to choose the polarity of the SCLK to the synchronization registers.

Figure 11-10. IDDRMX1A Symbol

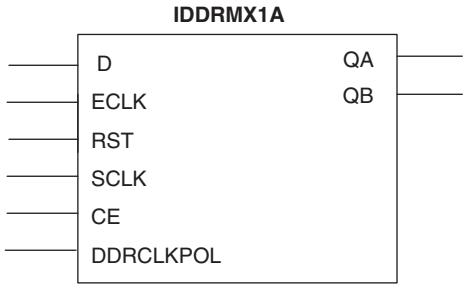


Table 11-3 provides a description of all I/O ports associated with the IDDRMX1A primitive.

Table 11-3. IDDRMX1A Ports

Port Name	I/O	Definition
D	I	DDR Data
ECLK	I	The phase shifted DQS should be connected to this input
RST	I	Reset
SCLK	I	System CLK
CE	I	Clock enable
DDRCLKPOL	I	DDR clock polarity signal
QA	O	Data at Positive edge of the CLK
QB	O	Data at the negative edge of the CLK

Note: The DDRCLKPOL input to IDDRMX1A should be connected to the DDRCLKPOL output of DQSBUFC.

Figure 11-11 shows the Input Register Block configured in the IDDRMX1A mode.

Figure 11-11. Input Register Block in IDDRMX1A Mode

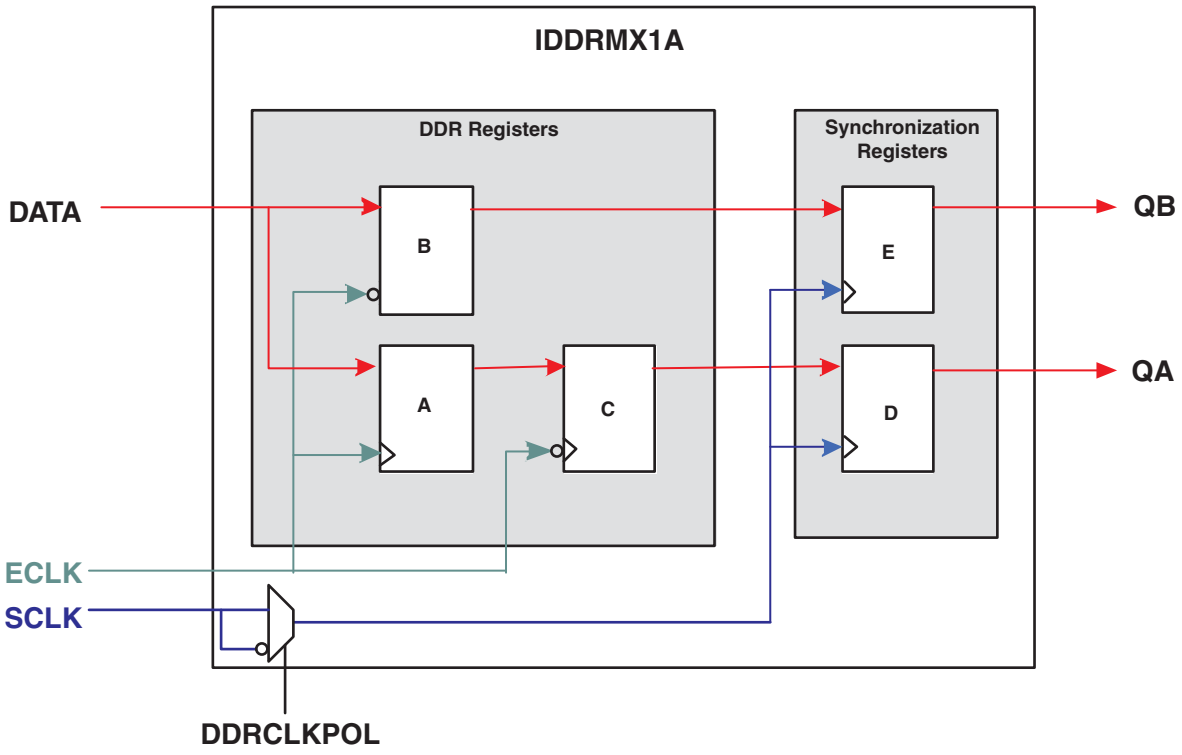
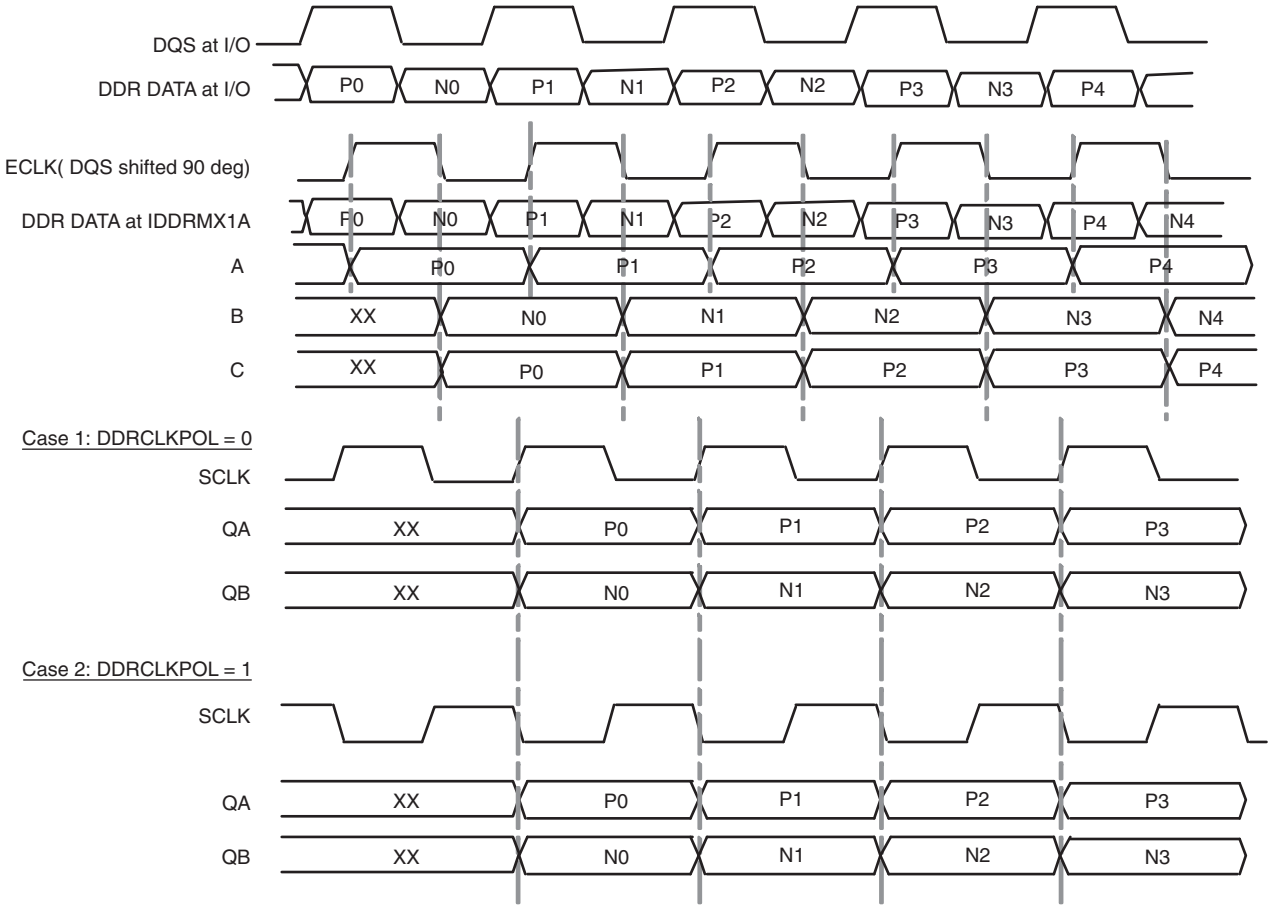


Figure 11-12 shows the IDDRMX1A timing waveform.

Figure 11-12. IDDRMX1A Waveform



IDDRMFX1A

With the IDDRMX1A, the data can enter the FPGA at either the positive or negative edge of the SCLK depending on the state of the DDRCLKPOL signal. The IDDRMFX1A module includes an additional clock transfer stage that ensures that the data is transferred at a known edge of the system clock.

Figure 11-13. IDDRMFX1A Symbol

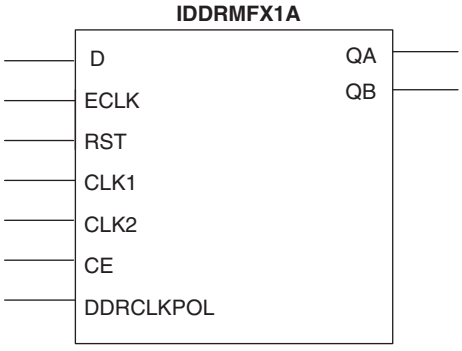


Table 11-4 provides a description of all I/O ports associated with the IDDRMFX1A primitive.

Table 11-4. IDDRMFX1A Ports

Port Name	I/O	Description
D	I	DDR Data
ECLK	I	The phase shifted DQS should be connected to this input
RST	I	Reset
CLK1	I	Slow FPGA CLK
CLK2	I	Slow FPGA CLK
CE	I	Clock enable
DDRCLKPOL	I	DDR clock polarity signal
QA	O	Data at the positive edge of the CLK
QB	O	Data at the negative edge of the CLK

Note: The DDRCLKPOL input to IDDRMFX1A should be connected to the DDRCLKPOL output of DQSBUFC.

Figure 11-14 shows the LatticeXP2 Input Register Block configured to function in the IDDRXMFX1A mode.

The DDR registers are designed to use Edge clock routing on the I/O side and the primary clock on the FPGA side. The ECLK input is used to connect to the DQS strobe coming from the DQS delay block (DQSBUFC primitive). The CLK1 and CLK2 inputs should be connected to the slow system (FPGA) clock. DDRCLKPOL is an input from the DQS Clock Polarity tree. This signal is generated by the DQS Transition detect circuit in the hardware. The additional clock transfer registers are shared with the output register block.

Figure 11-14. Input Register Block in IDDRMFX1A Mode

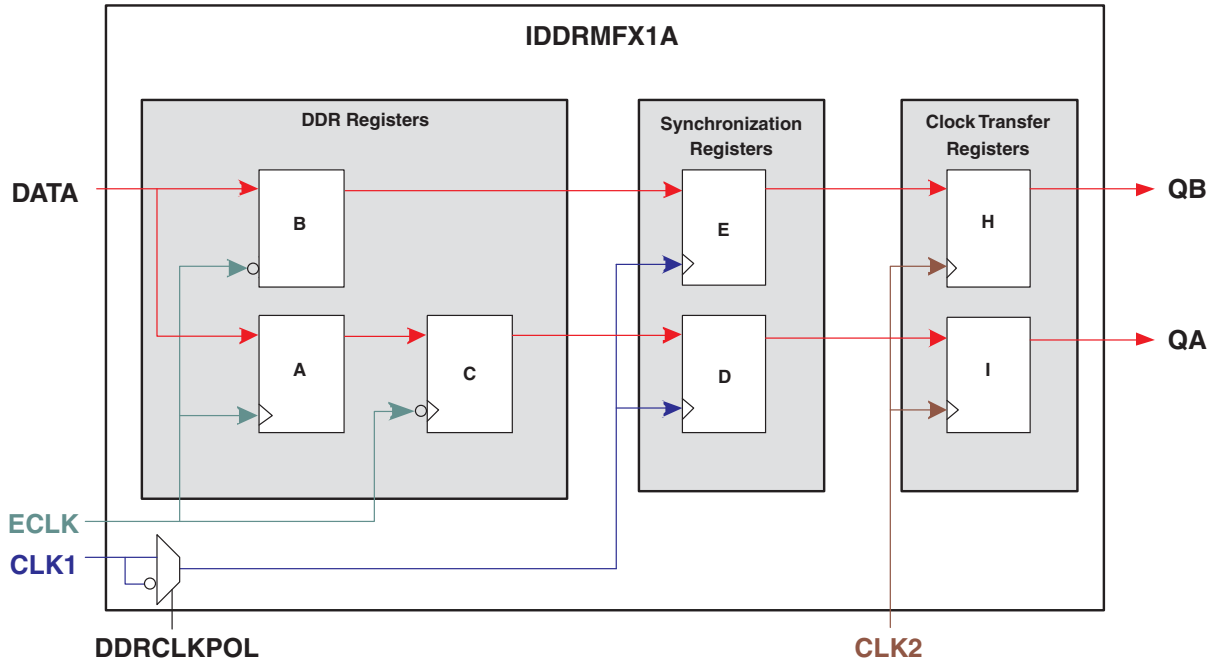
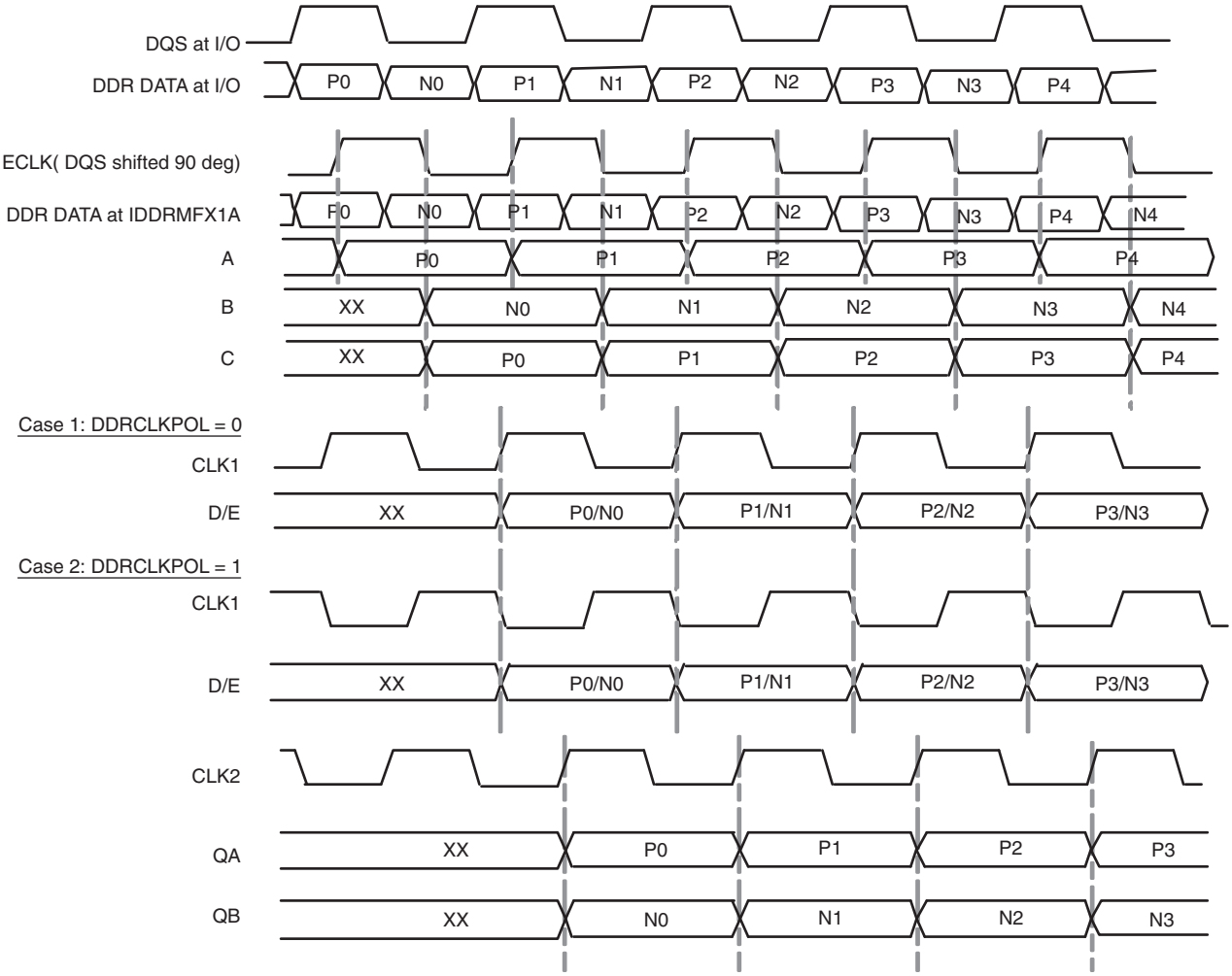


Figure 11-15 shows the IDDRMFX1A timing waveform.

Figure 11-15. IDDRMFX1A Waveform



ODDRMXA

The ODDRMXA primitive implements the output register for both the write and the tristate functions. This primitive is used to output DDR data and the DQS strobe to the memory. All the DDR output tristate functions are also implemented using this primitive.

Figure 11-16 shows the ODDRMXA primitive symbol and its I/O ports.

Figure 11-16. ODDRMXA Symbol

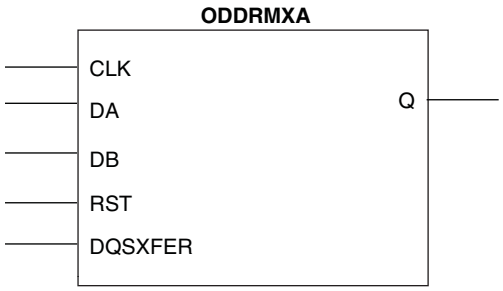


Table 11-5 provides a description of all I/O ports associated with the ODDRMXA primitive.

Table 11-5. ODDRMXA Ports

Port Name	I/O	Description
CLK	I	System CLK or ECLK
DA	I	Data at the negative edge of the clock
DB	I	Data at the positive edge of the clock
RST	I	Reset
DQSXFER	I	90-degree phase shifted clock coming from the DQSBUFC block
Q	I	DDR data to the memory

Notes:
1. RST should be held low during DDR Write operation.
2. DDR output and tristate registers do not have CE support. RST is available for the tristate DDRX mode (while reading). The LSR will default to set when used in the tristate mode.
3. When asserting reset during DDR writes, it is important to keep in mind that this only resets the flip-flops and not the latches.

Figure 11-17 shows the LatticeXP2 Output Register Block configured in the ODDRXMA mode.

Figure 11-17. Output Register Block in ODDRXMA Mode

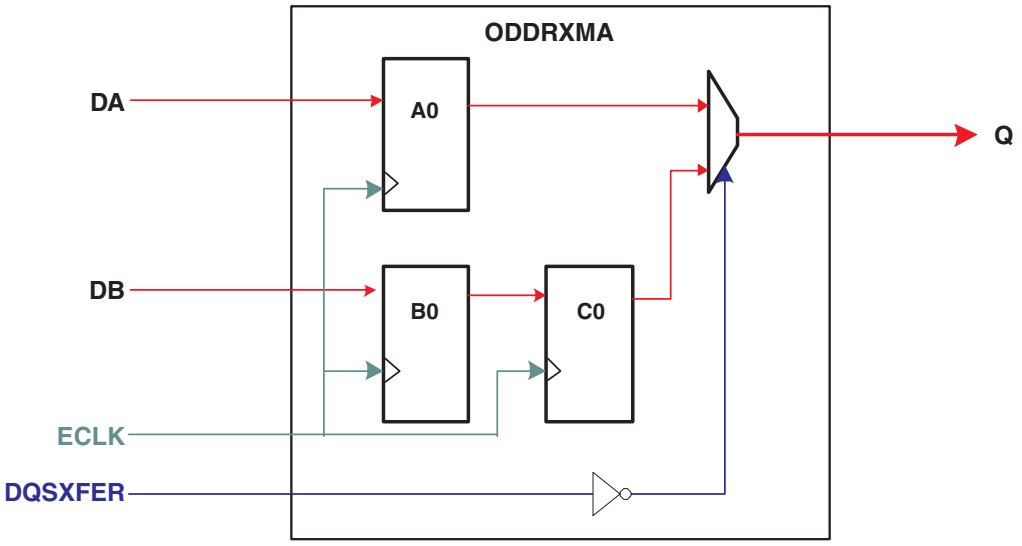
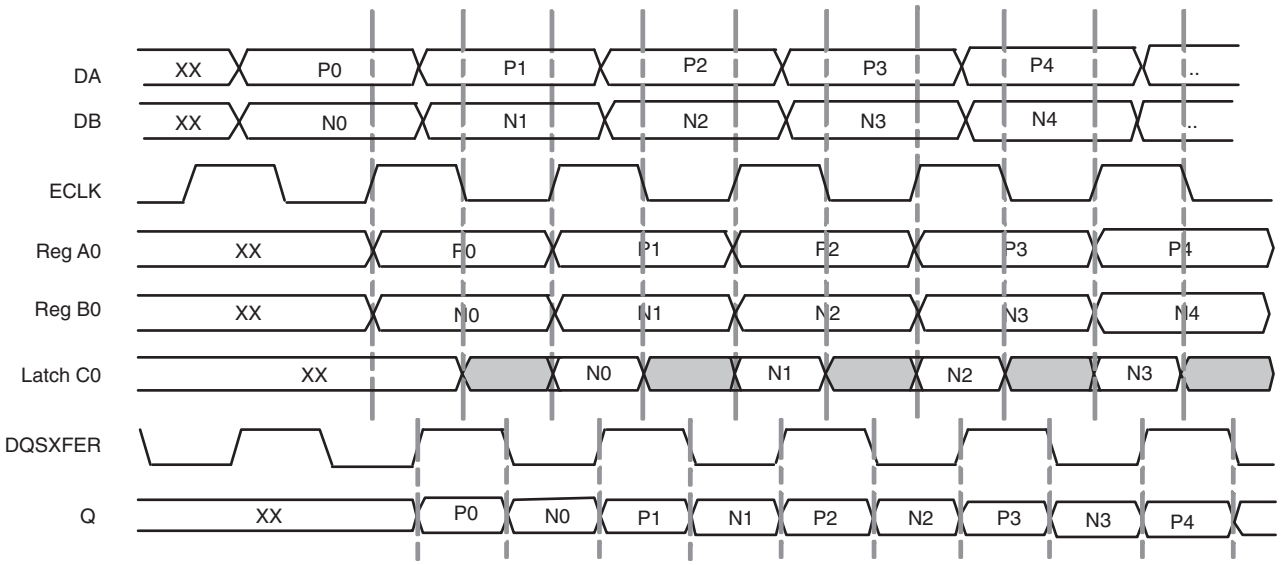


Figure 11-18 shows the ODDRMXA timing waveform.

Figure 11-18. ODDRMXA Waveform



Note that the DQSXFER is inverted inside the ODDRXMA. This will cause the data coming out of the ODDRXMA to be -90° in phase with the output of the ODDRXC module.

Memory Read Implementation

LatticeXP2 devices contain a variety of features to simplify implementation of the read portion of a DDR interface:

- DLL compensated DQS delay elements
- DDR input registers
- Automatic DQS to system clock domain transfer circuitry
- Data Valid Module

DLL Compensated DQS Delay Elements

The DQS from the memory is connected to the DQS Delay element. The DQS Delay block receives a 6-bit delay control from the on-chip DQSDLL. The LatticeXP2 devices support two DQSDLL, one on the left and one on the right side of the device. The DQSDEL generated by the DQSDLL on the left side is routed to all the DQS blocks on the left and bottom half of the device. The delay generated by the DQSDLL on the right side is distributed to all the DQS Delay blocks on the right side and the other bottom half of the device. There are no DQS pins on the top banks of the device. These digital delay control signals are used to delay the DQS from the memory by 90 degrees.

The DQS received from the memory is delayed in each of the DQS Delay blocks and this delay DQS is used to clock the first set stage DDR input registers.

DQS Transition Detect or Automatic Clock Polarity Select

In a typical DDR memory interface design, the phase relation between the incoming delayed DQS strobe and the internal system clock (during the READ cycle) is unknown. Prior to the READ operation in DDR memories, DQS is in tristate (pulled by termination). Coming out of tristate, the DDR memory device drives DQS low in the Preamble State. The DQS Transition Detect block detects the first DQS rising edge after the Preamble transition and generates a signal indicating the required polarity for the FPGA system clock (DDRCLKPOL). This signal is used to control the polarity of the clock to the synchronizing registers.

Data Valid Module

The data valid module generates a DATAVALID signal. This signal indicates to the FPGA that valid data is transmitted out the input DDR registers to the FPGA core.

DDR I/O Register Implementation

The first set of DDR registers is used to de-mux the DDR data at the positive and negative edge of the phase shifted DQS signal. The register that captures the positive-edge data is followed by a negative-edge triggered register. This register transfers the positive edge data from the first register to the negative edge of DQS so that both the positive and negative portions of the data are now aligned to the negative edge of DQS.

The second stage of registers is clocked by the FPGA clock, the polarity of this clock is selected by the DDR Clock Polarity signal generated by the DQS Transition Detect Block.

The I/O Logic registers can be implemented in two modes:

- Half Clock Transfer Mode
- Full Clock Transfer Mode

In Half Clock Transfer mode the data is transferred to the FPGA core after the second stage of the register. In Full Clock Transfer mode, an additional stage of I/O registers clocked by the FPGA clock is used to transfer the data to the FPGA core.

The LatticeXP2 Family Data Sheet explains each of these circuit elements in more detail.

Memory Read Implementation in Software

Three primitives in the ispLEVER® design tools represent the capability of these three elements. The DQSDLL represents the DLL used for calibration. The IDDRMX1A/IDDRMFX1A primitive represents the DDR input registers and clock domain transfer registers with or without full clock transfer. Finally, the DQSBUFC represents the DQS delay block, the clock polarity control logic and the Data Valid module. Figures 11-19 and 11-20 show the READ interface block generated using the IPexpress™ tool in the ispLEVER software.

Figure 11-19. Software Primitive Implementation for Memory READ (Half Clock Transfer)

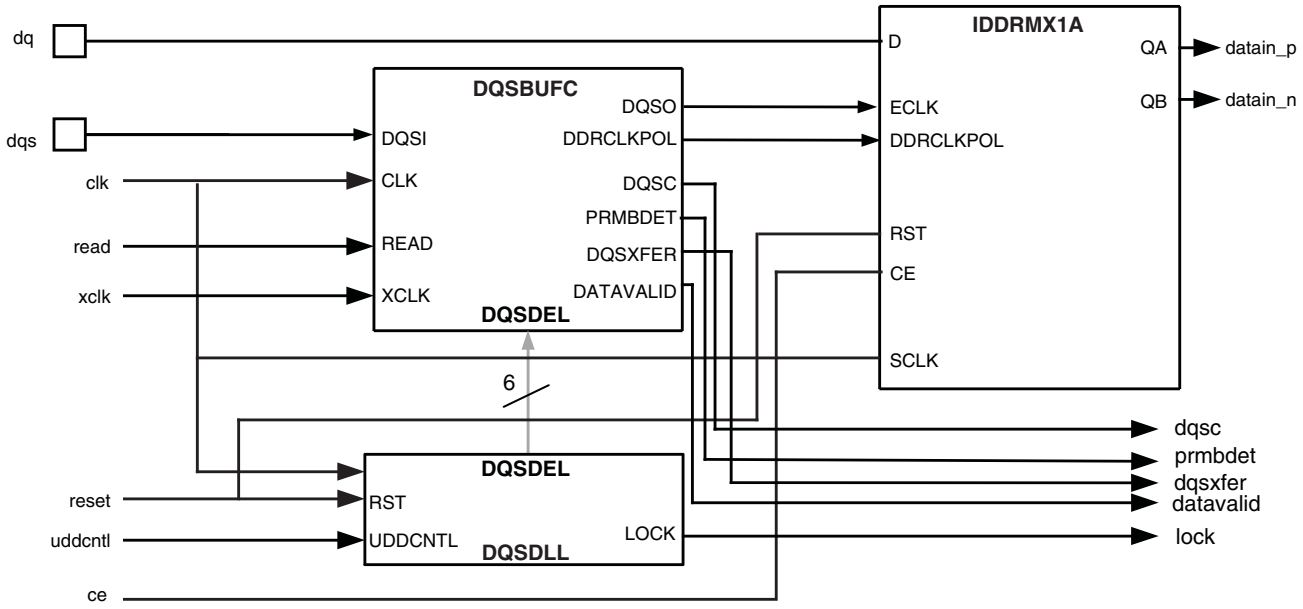
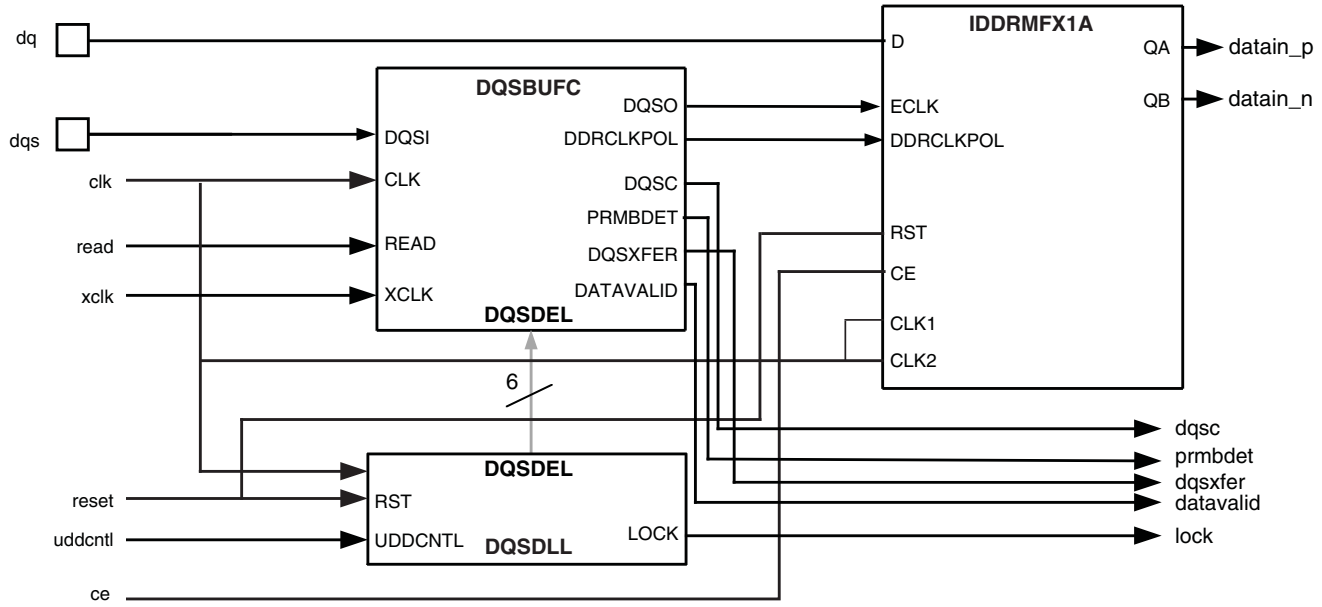


Figure 11-20. Software Primitive Implementation for Memory READ (Full Clock Transfer)



Read Timing Waveforms

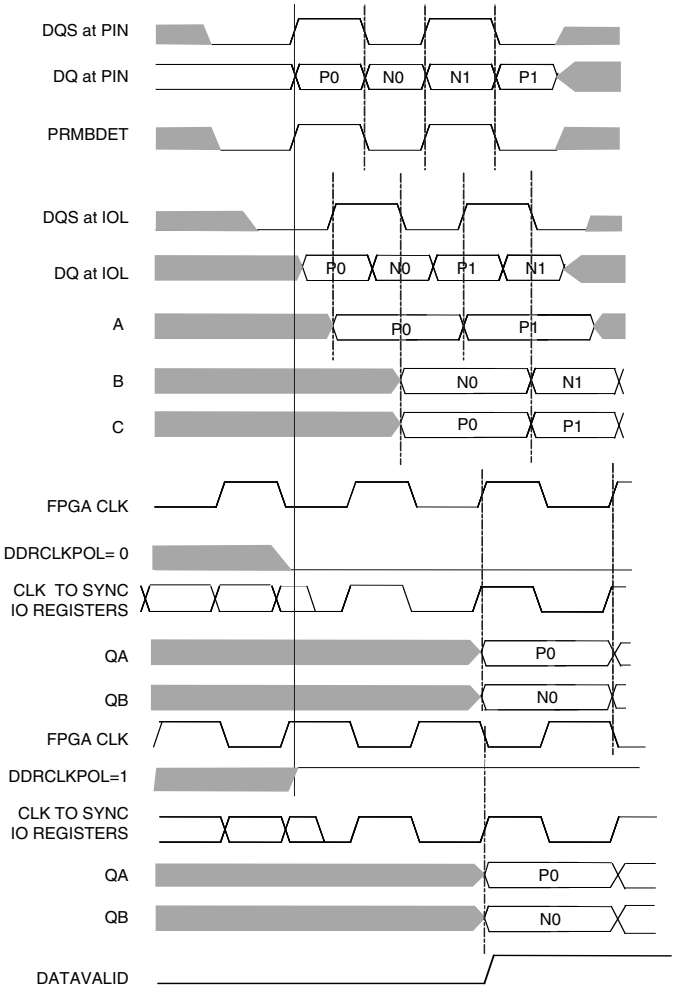
Figures 11-21 and 11-22 show READ data transfer for half and full clock cycle data transfer based on the results of the DQS Transition detector logic. This circuitry decides whether or not to invert the phase of FPGA system CLK to the synchronization registers based on the relative phases of PRMBDET and CLK.

- **Case 1:** If the FPGA clock is low on the first PRMBDET transition, then DDRCLKPOL is low and no inversion is required.
- **Case 2:** If the FPGA clock is high on the first PRMBDET, then DDRCLKPOL is high and the FPGA clock (CLK) needs to be inverted before it is used for synchronization.

Figure 11-21 illustrates the DDR data timing using half clock transfer mode at different stages of the IDDRMX1A registers. The first stage of the register captures data on the positive edge as shown by signal A and the negative edge as shown by signal B. Data stream A goes through an additional half clock cycle transfer shown by signal C. Phase-aligned data streams B and C are presented to the next stage registers clocked by the FPGA clock.

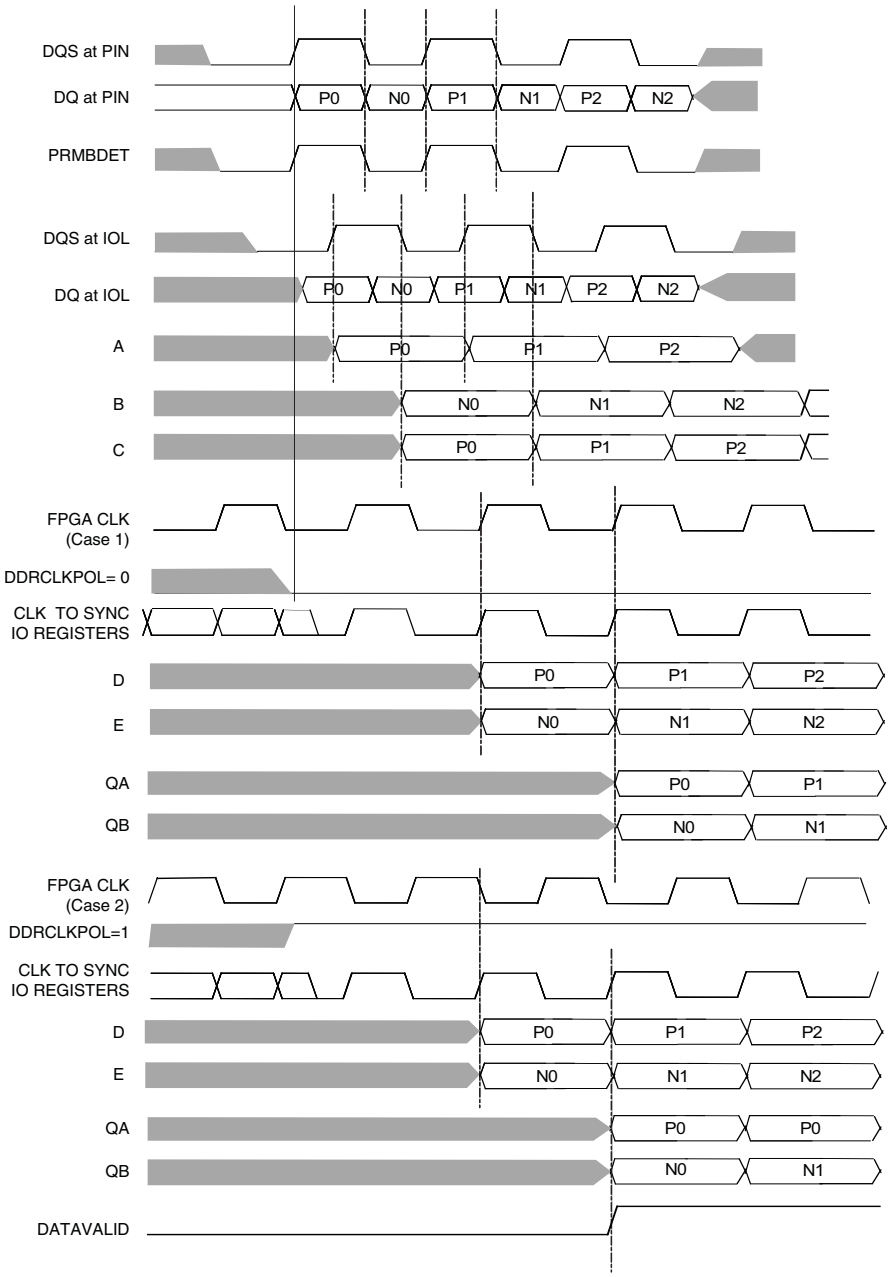
Figure 11-22 illustrates the DDR data timing using full clock transfer mode at different stages of IDDRMFX1A registers. In addition to the first two register stages in the half clock mode, the full clock transfer mode has an additional stage register clocked by the FPGA clock. In this case, D and E are the data streams after the second register stage presented to the final stage of registers clocked by the FPGA clock.

Figure 11-21. READ Data Transfer When Using IDDRMX1A



Notes:
1. DDR memory sends DQ aligned to DQS Strobe.
2. The DQS Strobe is delayed by 90 degrees using the dedicated DQS logic.
3. DQ is now center aligned to DQS Strobe.
4. PRMBDET is the Preamble detect signal generated using the DQSBUFB primitive. This is used to generate the DDRCLKPOL signal.
5. The first set of I/O registers, A and B, capture data on the positive and negative edges of DQS.
6. I/O Register C transfers data so that both data are now aligned to negative edge of DQS.
7. DDCLKPOL signal generated will determine if the FPGA CLK going into the synchronization registers need to be inverted. The DDRCLK-POL=0 when the FPGA CLK is LOW at the first rising edge of PRMBDET. The clock to the synchronization registers is not inverted. The DDRCLKPOL=1 when the FPGA CLK is HIGH at the first rising edge of PRMBDET. In this case the clock to the synchronization register is inverted.
8. The I/O synchronization registers capture data on either the rising or falling edge of the FPGA clock.
9. The DATAVALID signal goes HIGH when valid data enters the FPGA core. Once DATA VALID is asserted, it stays high until the next READ pulse.

Figure 11-22. Read Data Transfer When Using IDDRMFX1A



Notes:
1. DDR memory sends DQ aligned to DQS Strobe.
2. The DQS Strobe is delayed by 90 degrees using the dedicated DQS logic.
3. DQ is now center-aligned to DQS Strobe.
4. PRMBDET is the Preamble detect signal generated using the DQSBUFB primitive. This is used to generate the DDRCLKPOL signal.
5. The first set of I/O registers, A and B, capture data on the positive and negative edges of DQS.
6. I/O register C transfers data so that both data are now aligned to the negative edge of DQS.
7. DDCLKPOL signal generated will determine if the FPGA clock going into the synchronization registers need to be inverted. The DDRCLK-POL=0 when the FPGA CLK is LOW at the first rising edge of PRMBDET. So the clock to the synchronization registers is not inverted. The DDRCLKPOL=1 when the FPGA CLK is HIGH at the first rising edge of PRMBDET. In this case the clock to the synchronization register is inverted.
8. Registers D and E capture data at the FPGA clock.
9. The data is then again registers at the FPGA clock to ensure a Full Clock Cycle transfer.
10. DATAVALID signal goes HIGH when valid data enters the FPGA core. Once DATA VALID is asserted, it stays high until the next READ pulse.

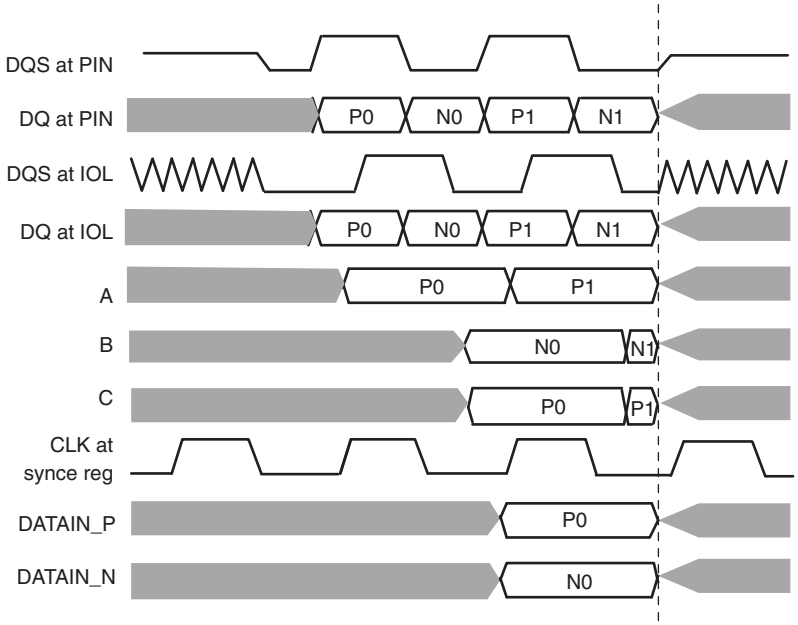
Data Read Critical Path

Data in the second stage DDR registers can be registered either on the positive edge or on the falling edge of FPGA clock depending on the DDRCLKPOL signal. In order to ensure that the data transferred to the FPGA core registers is aligned to the rising edge of the system clock, this path should be constrained with a half clock transfer. This half clock transfer can be forced in the software by assigning a multi-cycle constraint (multi-cycle of 0.5 X) on all the data paths to first PFU register.

DQS Postamble

At the end of a READ cycle, the DDR SDRAM device executes the READ cycle postamble and then immediately tristates both the DQ and DQS output drivers. Since neither the memory controller (FPGA) nor the DDR SDRAM device are driving DQ or DQS at that time, these signals float to a level determined by the off-chip termination resistors. While these signals are floating, noise on the DQS strobe may be interpreted as a valid strobe signal by the FPGA input buffer. This can cause the last READ data captured in the IOL input DDR registers to be overwritten before the data has been transferred to the free running resynchronization registers inside the FPGA.

Figure 11-23. Postamble Effect on READ



LatticeXP2 devices have extra dedicated logic in the in the DQS Delay Block that prevents this postamble problem. The DQS postamble logic is automatically implemented when the user instantiates the DQS Delay logic (DQSBUFC software primitive) in the design.

Memory Write Implementation

To implement the write portion of a DDR memory interface, two streams of single data rate data must be multiplexed together with data transitioning on both edges of the clock. In addition, during a write cycle, DQS must arrive at the memory pins center-aligned with the data, DQ. Along with the DQS strobe and data this portion of the interface must also provide the CLKP, CLKN Address/Command and Data Mask (DM) signals to the memory.

It is the responsibility of the FPGA output control to edge-align the DDR output signals (ADDR,CMD, DQS, but not DQ, DM) to the rising edge of the outgoing differential clock (CLKP/CLKN).

Challenges encountered by the during Memory WRITE:

1. DQS needs to be center-aligned with the outgoing DDR Data, DQ.
2. Differential CLK signals (CLKP and CLKN) need to be generated.

3. The controller must meet the DDR interface specification for t_{DSS} and t_{DSH} parameters, defined as DQS falling to CLKP rising setup and hold times.
4. The DDR output data must be muxed from two SDR streams into a single outgoing DDR data stream.

All DDR output signals (“ADDR, CMD”, DQS, DQ, DM) are initially aligned to the rising edge of the FPGA clock inside the FPGA core. The relative phase of the signals may be adjusted in the IOL logic before departing the FPGA. These adjustments are shown in Figure 11-24.

LatticeXP2 devices contain DDR output and tri-state registers along with the DQSXFER signal generated by the DQSBUFC that allows easy implementation of the write portion of the DDR memory interfaces. The DDR output registers can be accessed in the design tools via the ODDRMXA and the ODDRXC primitives.

The DQS signal and the DDR clock outputs are generated using the ODDRXC primitive. As shown in the figure, the CLKP and DQS signals are generated so that they are 180 degrees in phase with the clock. This is done by connecting “1” to the DA input and “0” to the DB inputs of the ODDRXC primitive. Refer to the DDR Generic Software Primitive section of this document to see the ODDRXC timing waveforms.

The DDR clock output is then fed into a SSTL differential output buffer to generate CLKP and CLKN differential clocks. Generating the CLKN in this manner prevents any skew between the two signals. When interfacing to DDR1, SDRAM memory CLKP should be connected to the SSTL25D I/O standard. When interfacing to DDR2 memory, it should be connected to the SSTL18D I/O standard.

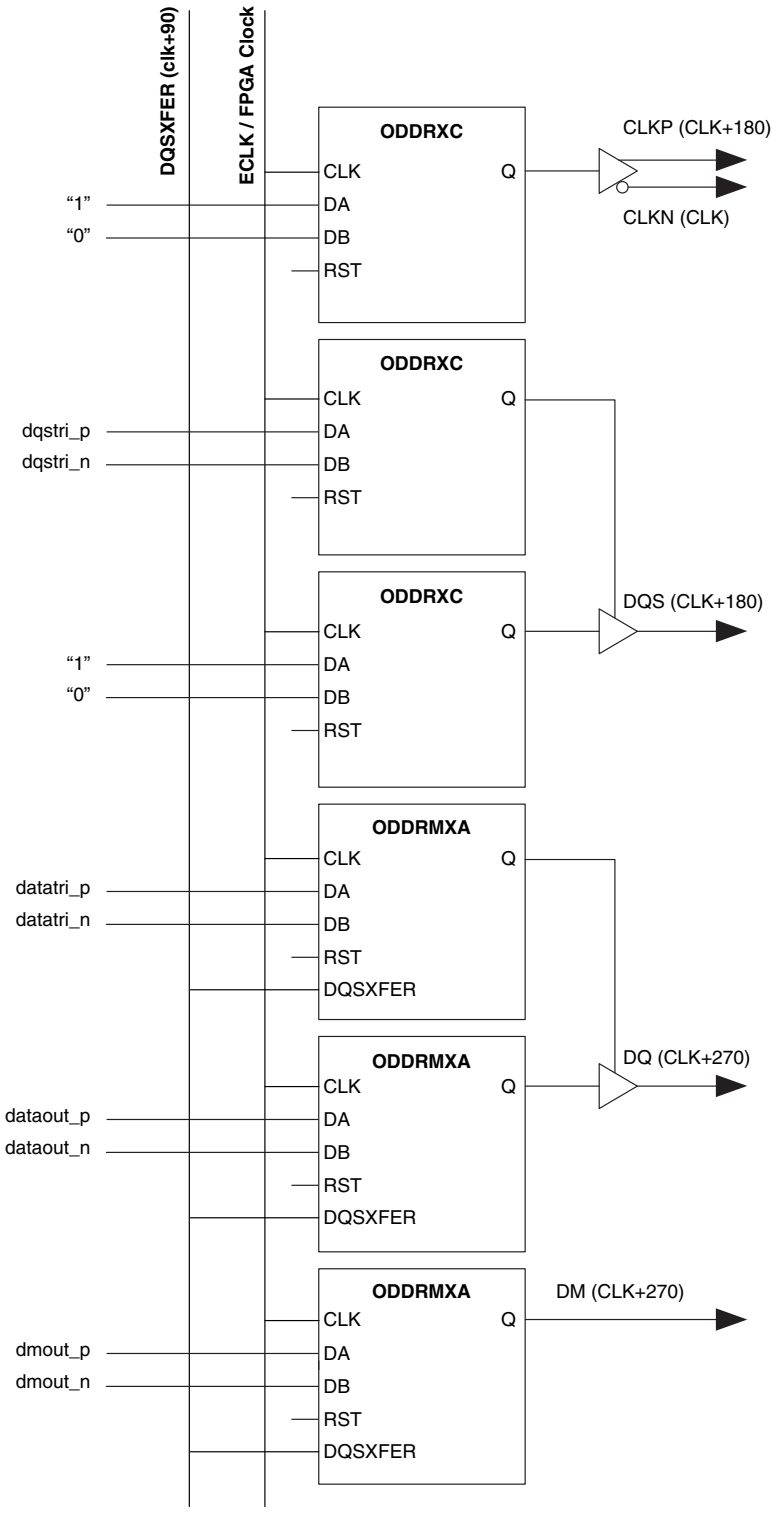
The DQSXFER output from the DQSBUFC block is the 90-degree phase shifted clock. This 90-degree phase shifted clock is used as an input to the ODDRMXA block. The ODDRMXA is used to generate the DQ and DM data outputs going to the memory. In the ODDRMXA module, the data is first registered using the ECLK or FPGA clock input and then shifted out using the DQSXFER signal. To ensure that the data going to the memory is center-aligned to the DQS, the DQSXFER is inverted inside the ODDRXMA primitive. This will generate data that is center-aligned to the DQS. Refer to the Software Primitives section of this document for the ODDRXMA timing waveforms.

The DDR interface specification for t_{DSS} and t_{DSH} parameters defined as DQS falling to CLKP rising setup and hold times must be met. This is accomplished by ensuring that the CLKP and DQS signals are identical in phase.

The tristate control for the DQS and DQ outputs can also be implemented using the ODDRXC primitive.

Figure 11-24 shows the DDR Write implementation using the DDR primitives.

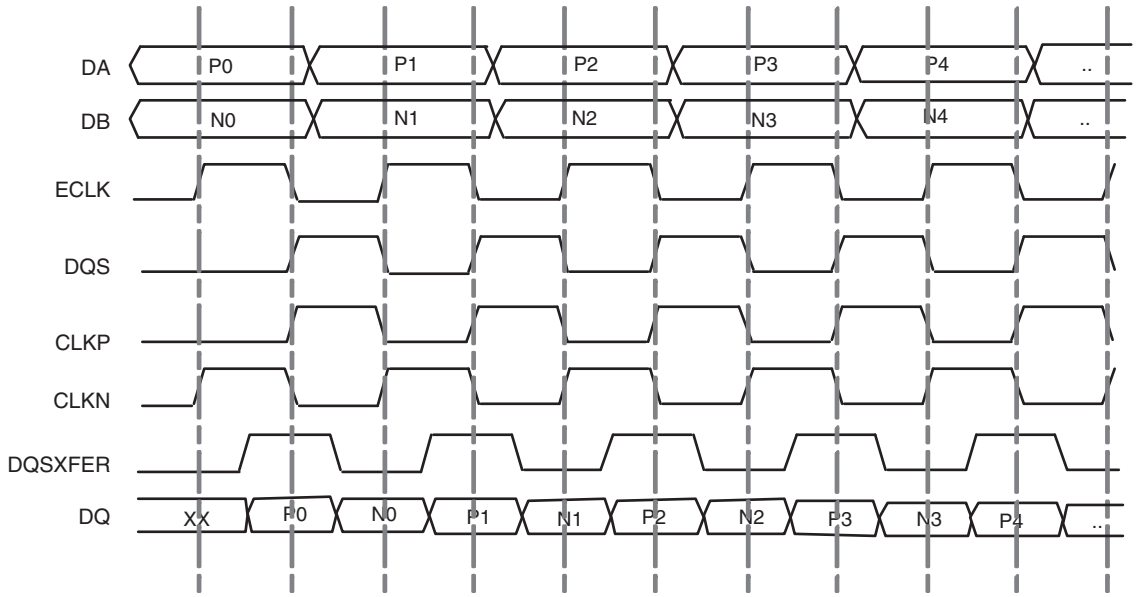
Figure 11-24. Software Implementation for Memory Write



Write Timing Waveforms

Figure 11-25 shows the DDR write side data transfer timing for the DQ Data pad and the DQS Strobe Pad. When writing to the DDR memory device, the DM (Data Mask) and the ADDR/ CMD (Address and Command) signals are also sent to the memory device along with the data and strobe signals.

Figure 11-25. DDR Write Data Transfer for DQ Data



Design Rules/Guidelines

Listed below are some rules and guidelines to keep in mind when implementing DDR memory interfaces in the LatticeXP2 devices.

- The LatticeXP2 devices have dedicated DQ-DQS banks. Please refer to the logical signal connections of the groups in the LatticeXP2 Family Data Sheet before locking these pins.
- There are two DQSDLL on the device, one for the left half and one for the right half of the device. Therefore, only one DQSDLL primitive should be instantiated for each half of the device. Since there is only one DQSDLL on each half of the device, all the DDR memory interfaces on that half of the device should run at the same frequency. Each of the DQSDLL will generate 90-degree digital delay bits for all the DQS delay blocks on that half of the device based on the reference clock input to the DLL.
- When implementing a DDR SDRAM interface, all interface signals should be connected to the SSTL25 I/O standard. In the case of the DDR2 SDRAM interface, the interface signal should be connected to SSTL18 I/O standard.
- For DDR2, the differential DQS signals need to be connected to SSTL18 the Differential I/O standard.
- When implementing the DDR interface, the VREF1 of the bank is used to provide the reference voltage for the interface pins.

Generic High Speed DDR Implementation

In addition to the DDR memory interface, the I/O logic DDR registers can be used to implement high speed DDR interfaces. The Input DDR registers can operate in full clock transfer and half clock transfer modes. The DDR input and output register also support x1 and x2 gearing ratios. A gearing capability is provided to Mux/DeMux the I/O data rate (ECLK) to the FPGA clock rate (SCLK). For DDR interfaces, this ratio is slightly different than the SDR ratio. A basic 2x DDR element provides four FPGA side bits for two I/O side bits at half the clock rate on the FPGA side.

The data going to the DDR registers can be optionally delayed before going to the DDR register block.

Generic DDR Software Primitives

The IPexpress tool in the ispLEVER software can be used to generate the DDR modules. The various DDR modes described below can be configured in the IPexpress tool. The various modes are implemented using the following software primitives.

- IDDRXC – DDR Generic Input
- IDDRFXA – DDR Generic Input with full clock transfer (x1 gearbox)
- IDDRX2B – DDR Generic Input with 2x gearing ratio. DDRX2 inputs a double data rate signal as four data streams. Two stages of DDR registers are used to convert serial DDR data at input pad into four SDR data streams entering FPGA core logic.
- ODDRXC – DDR Generic Output
- ODDRX2B – DDR Generic Output with 2x gearing ratio. The DDRX2 inputs four separate data streams and outputs a single data stream to the I/O buffer.
- DELAYB – The DDR input can be optionally delayed before it is input to the DDR registers. The user can choose to implement a fixed delay value or use a dynamic delay.

IDDRXC

This primitive inputs DDR data at both edges of the CLK and generates two streams of data. The CLK to this module can be connected to either the edge clock or the primary FPGA clock.

Figure 11-26 shows the primitive symbol for IDDRXC mode.

Figure 11-26. IDDRXC Symbol

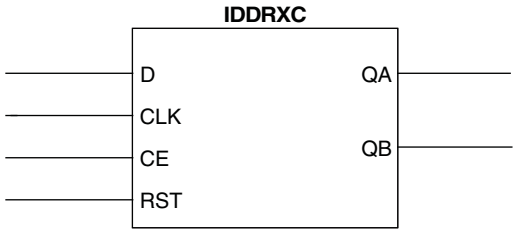


Table 11-6 lists the port names and descriptions for the IDDRXC primitive.

Table 11-6. IDDRXC Port Names

Port Name	I/O	Definition
D	I	DDR data
CLK	I	This clock can be connected to the ECLK or the FPGA clock
CE	I	Clock enable signal
RST	I	Reset to the DDR register
QA	O	Data at the positive edge of the clock
QB	O	Data at the negative edge of the clock

Figure 11-27 shows the LatticeXP2 Input Register Block configured in the IDDRXFC mode.

Figure 11-27. Input Register Block Configured as IDDRXC

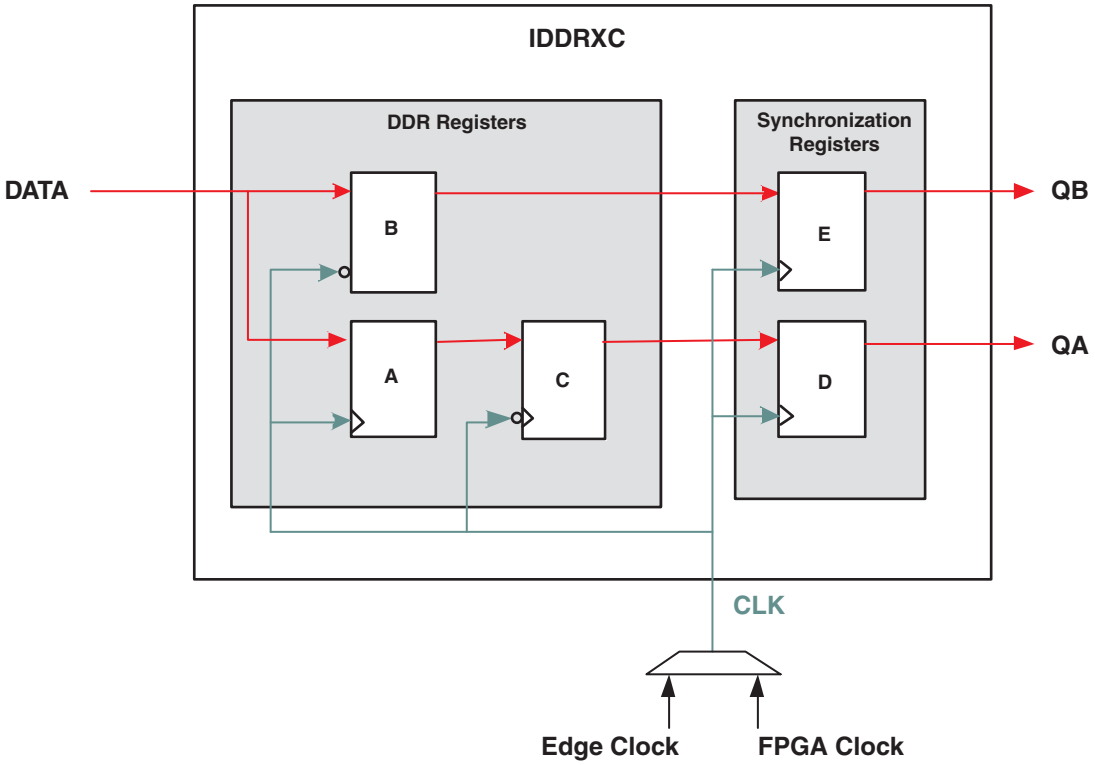
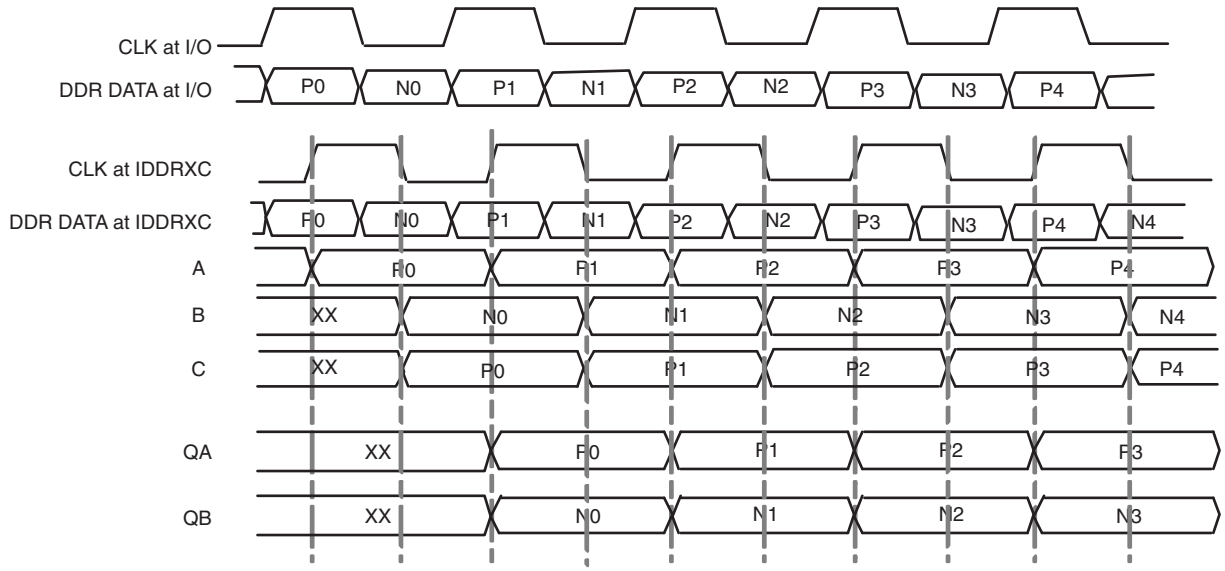


Figure 11-28 shows the timing waveform when using the IDDRXC module.

Figure 11-28. IDDRXC Waveform



IDDRFXA

This primitive inputs DDR data at both edges of clock CLK1 and generates two streams of data aligned to clock CLK2. CLK1 can be connected either to the edge clock or the internal FPGA clock. If the Edge clock input is used for CLK1 then CLK2 should be generated from the same clock going to CLK1.

Figure 11-29. IDDRFXA Symbol

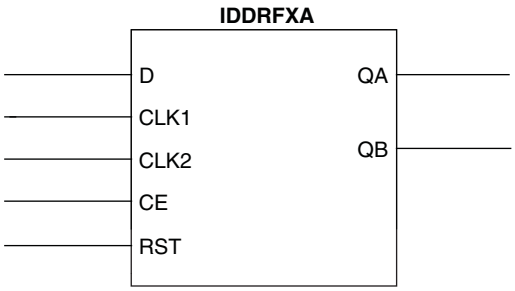


Table 11-7 lists the port names and descriptions for the IDDRFXA primitive.

Table 11-7. IDDRFXA Port Names

Port Name	I/O	Description
D	I	DDR data
CLK1	I	This clock can be connected to the ECLK or the FPGA clock
CLK2	I	This clock should be connected to the FPGA clock
CE	I	Clock Enable signal
RST	I	Reset to the DDR register
QA	O	Data at the positive edge of the clock
QB	O	Data at the negative edge of the clock

Figure 11-31 shows the LatticeXP2 Input Register Block configured in the IDDRXFXA mode. CLK1 used to register the DDR registers and the first set of synchronization registers. CLK2 is used by the third stage of registers and should be clocked by the FPGA clock. These clock transfer registers are shared with the output register block.

Figure 11-30. Input Register Block configured as IDDRFXA

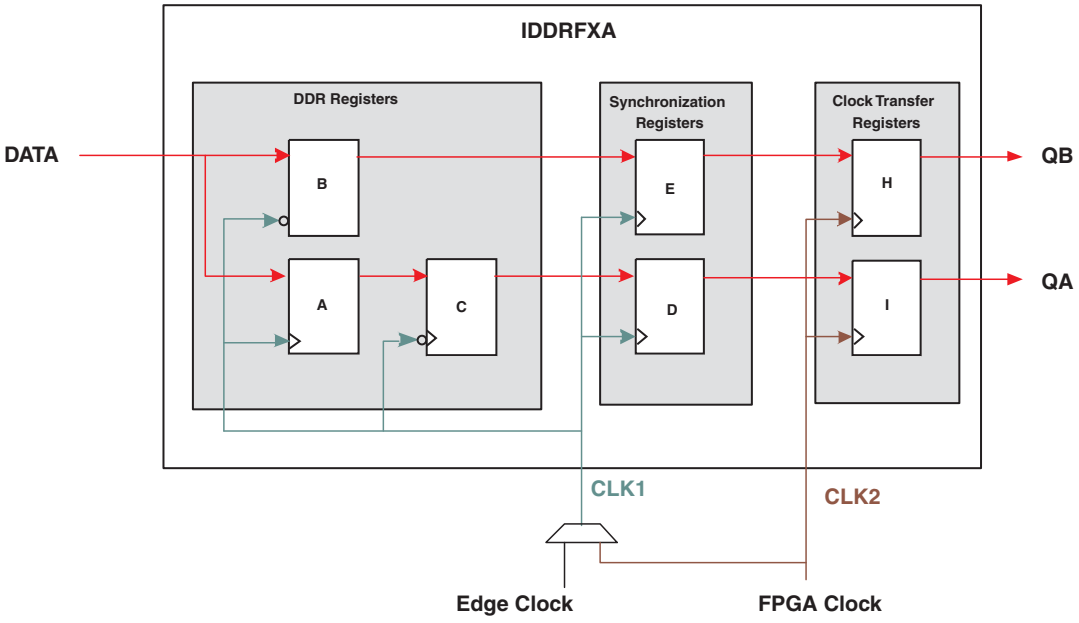
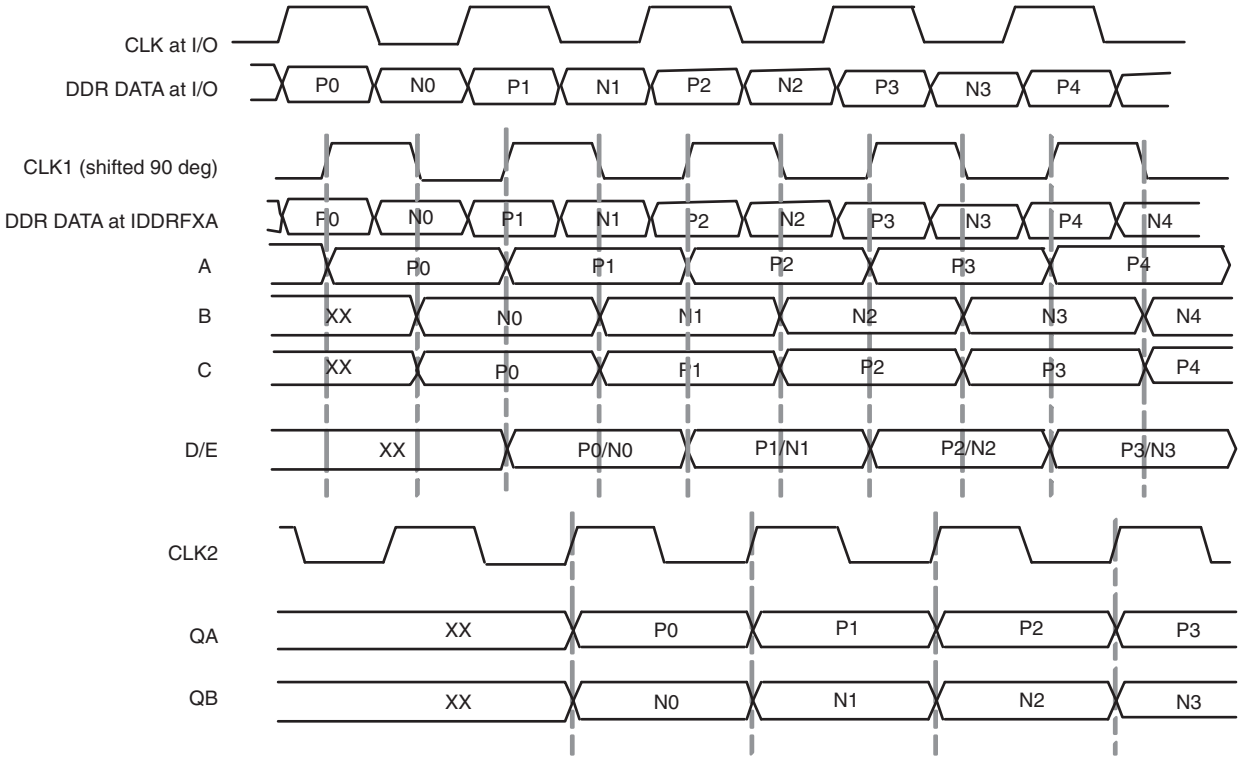


Figure 11-31 shows the timing waveform when using the IDDRFXA module.

Figure 11-31. IDDRFXA Waveform



IDDRX2B

This module is used when a gearing function is required. This primitive inputs the DDR data at both edges of the edge clock and generates four streams of data aligned to SCLK. SCLK is always half the frequency of ECLK. It is recommended that the CLKDIV module or PLL be used to generate the SCLK from the ECLK.

Figure 11-32 shows the primitive symbol for the IDDRX2B mode.

Figure 11-32. IDDRX2B Symbol

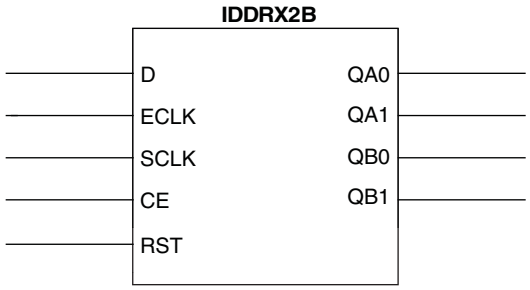


Table 11-8 lists the port names and descriptions for the IDDRX2B primitive.

Table 11-8. IDDRX2B Port Names

Port Name	I/O	Description
D	I	DDR data
ECLK	I	This clock can be connected to the fast edge clock
SCLK	I	This clock should be connected to the FPGA clock
CE	I	Clock enable signal
RST	I	Reset to the DDR register
QA0, QA1	O	Data at the positive edge of the clock
QB0, QB1	O	Data at the negative edge of the clock

Figure 11-33 shows the LatticeXP2 Input Register Block configured in the IDDRX2B mode. The DDR registers and the first set of synchronization registers are clocked by the ECLK input. The SCLK is used to clock the third stage of register. This primitive will output four streams of data. The 2x gearing function is implemented by using the synchronization registers of the complementary PIO. The clock transfer registers are shared with the output register block.

Figure 11-33. Input Register Block Configured as IDDRX2B

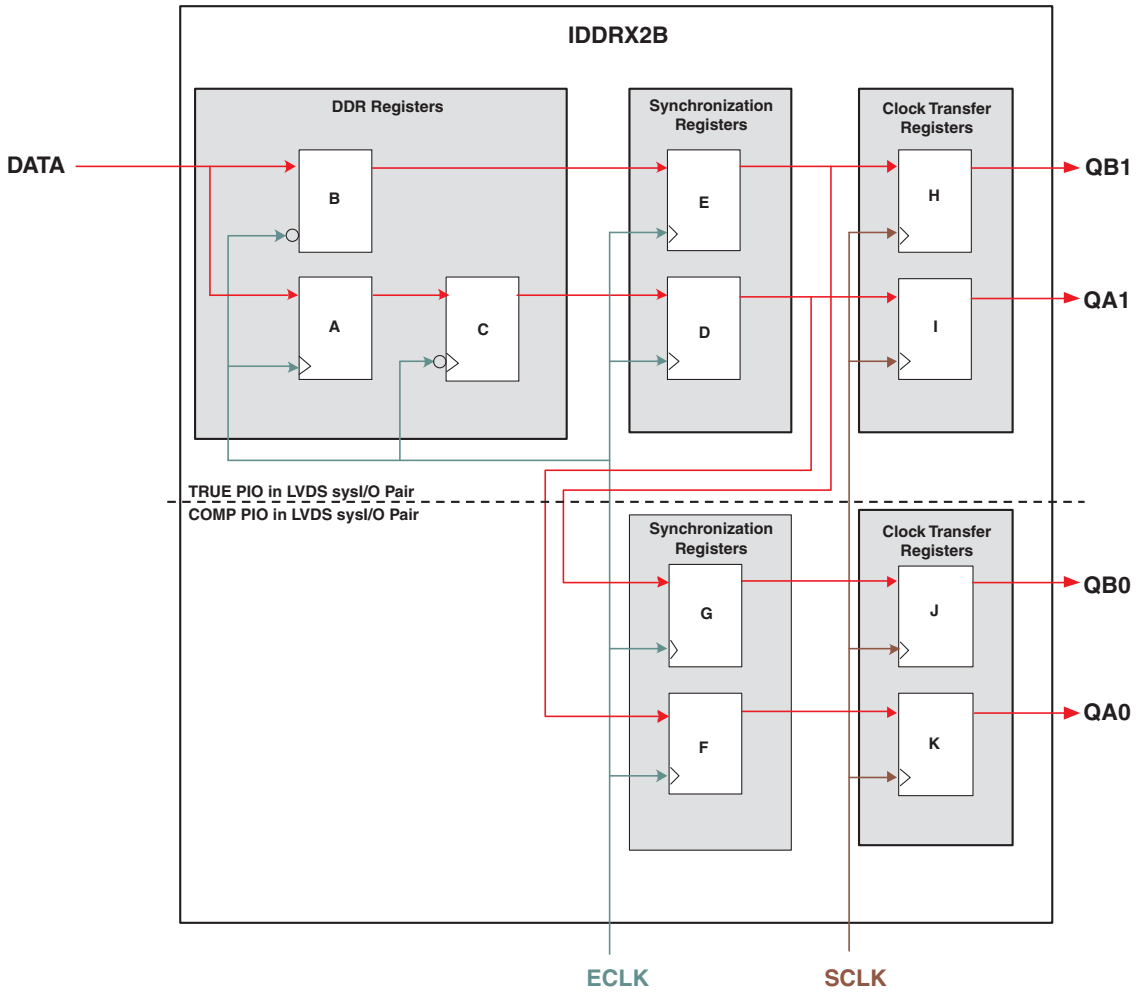
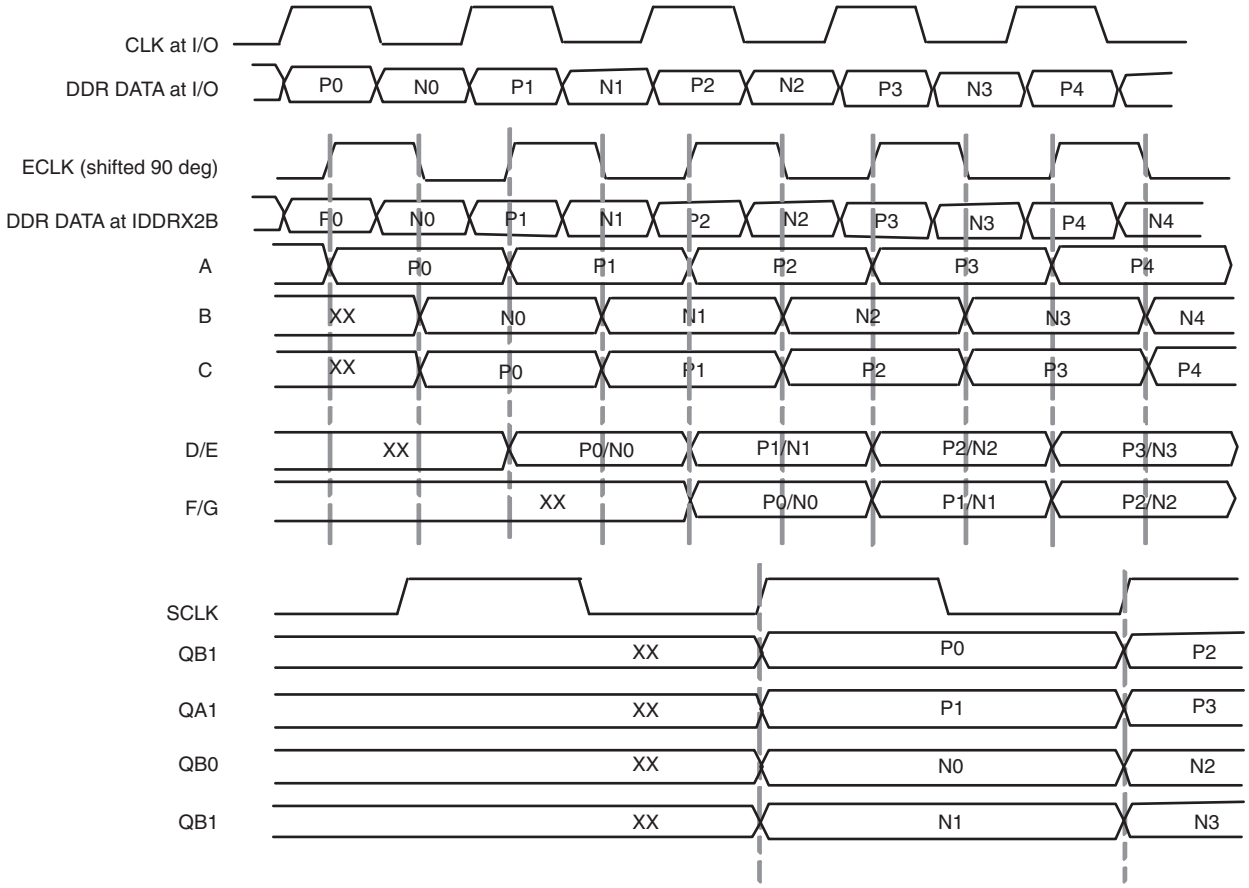


Figure 11-34 shows the timing waveform using the IDDRX2B module.

Figure 11-34. IDDRX2B Waveform



ODDRXC

This is the DDR output module. This primitive will input two data streams and mux them together to generate a single stream of data going to the sysIO™ buffer. The CLK to this module can be connected to the edge clock or to the FPGA clock. This primitive is also used for when DDR function is required for the tristate signal.

Figure 11-35 shows the primitive symbol for the ODDRXC mode.

Figure 11-35. ODDRXC Symbol

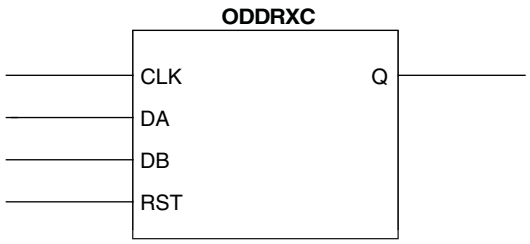


Table 11-9 lists the port names and descriptions for the ODDRXC primitive.

Table 11-9. ODDRXC Port Names

Port Name	I/O	Definition
DA	I	Data at the negative edge of the clock
DB	I	Data at the positive edge of the clock
CLK	I	This clock can be connected to the edge clock or to the FPGA clock
RST	I	Reset signal
Q	O	DDR data output

Figure 11-36 shows the Output Register Block of the LatticeXP2 device configured in ODDRXC mode.

Figure 11-36. Output Register Block in ODDRC Mode

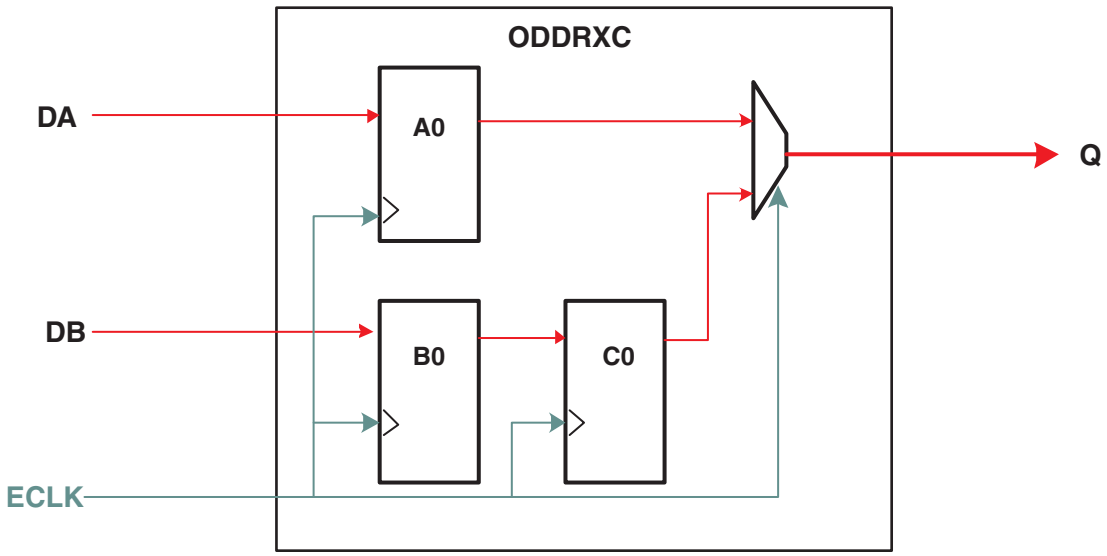
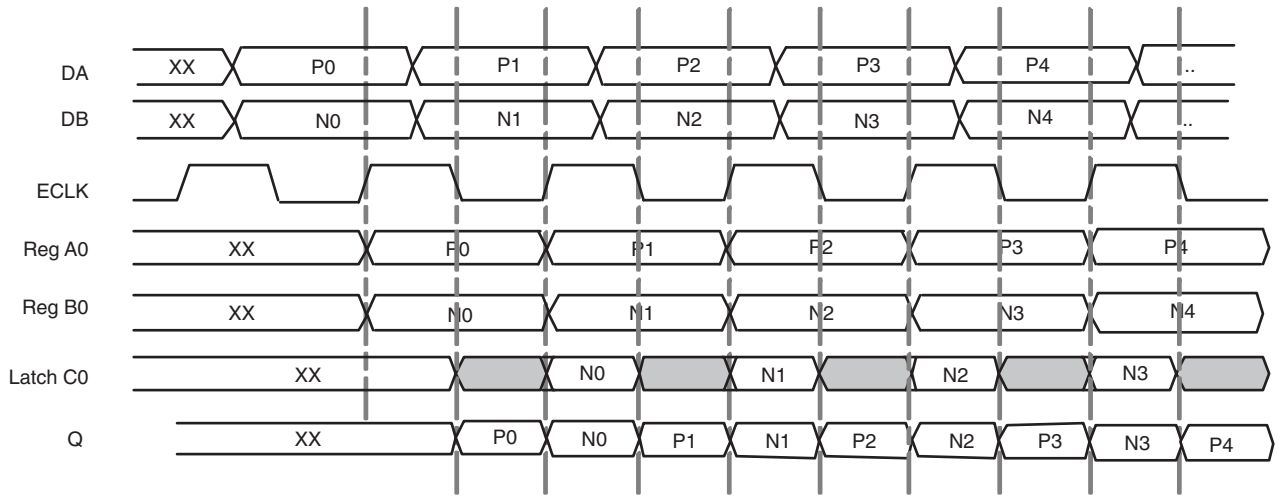


Figure 11-37 shows the timing waveform when using the ODDRXC module.

Figure 11-37. ODDRXC Waveform



ODDRX2B

This DDR output module can be used when a gearbox function is required. This primitive inputs four data streams and muxes them together to generate a single stream of data going to the sysIO buffer.

DDR registers of the complementary PIO are used in this mode. The complementary PIO register can no longer be used to perform the DDR function. There are two clocks going to this primitive. The ECLK is connected to the faster edge clock and the SCLK is connected to the slower FPGA clock. The DDR data output of this primitive is aligned to the faster edge clock.

Figure 11-38 shows the primitive symbol for the ODDRX2B mode.

Figure 11-38. ODDRX2B Symbol

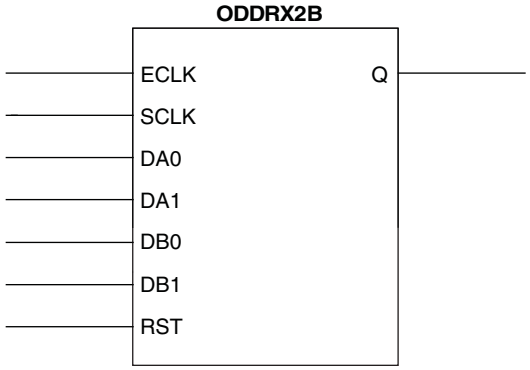


Table 11-10 lists the port names and descriptions for the ODDRX2B primitive.

Table 11-10. ODDRX2B Port Names

Port Name	I/O	Description
DA0, DB0	I	Data at the negative edge of the clock
DA1, DB1	I	Data at the positive edge of the clock
ECLK	I	This clock should be connected to the faster edge clock
SCLK	I	This clock should be connected to the slower FPGA clock
RST	I	Reset signal
Q	O	DDR data output

Figure 11-39 shows the LatticeXP2 Output Register Block in the ODDRX2B mode.

Figure 11-39. Output Register Block Configured in ODDRX2B Mode

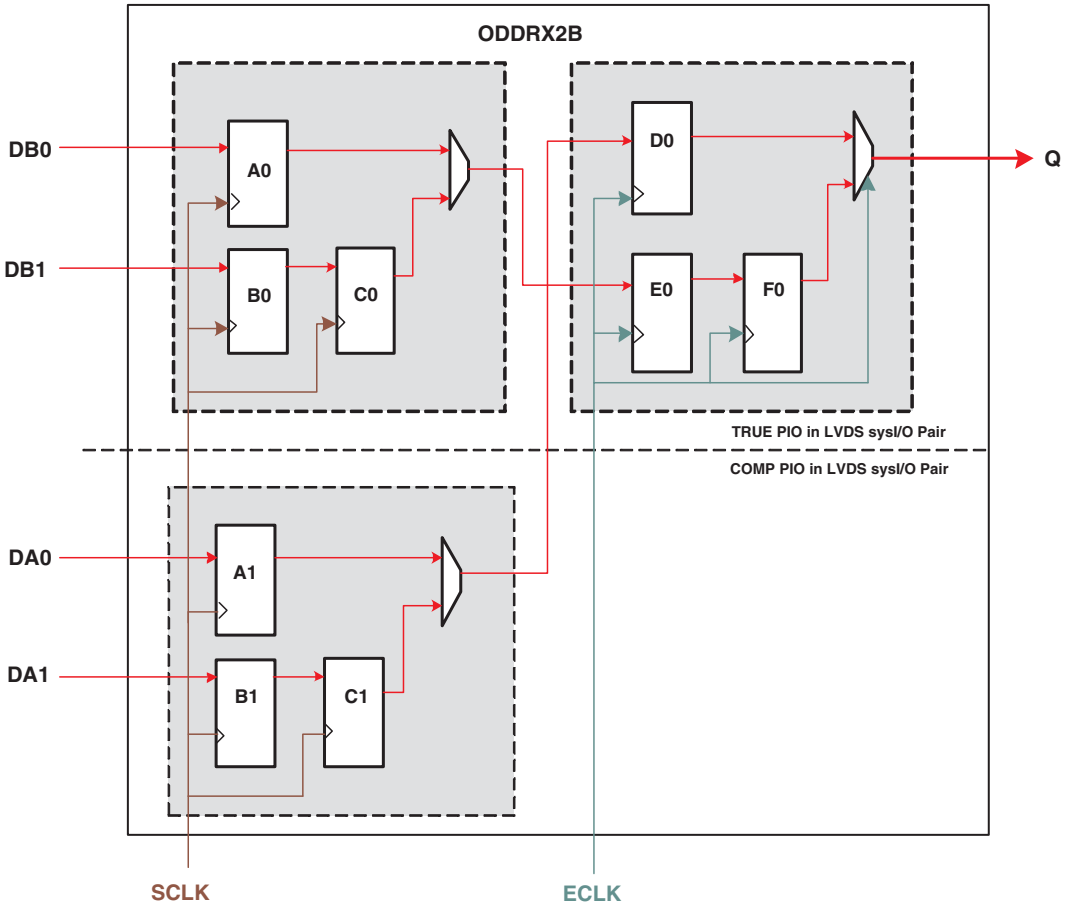
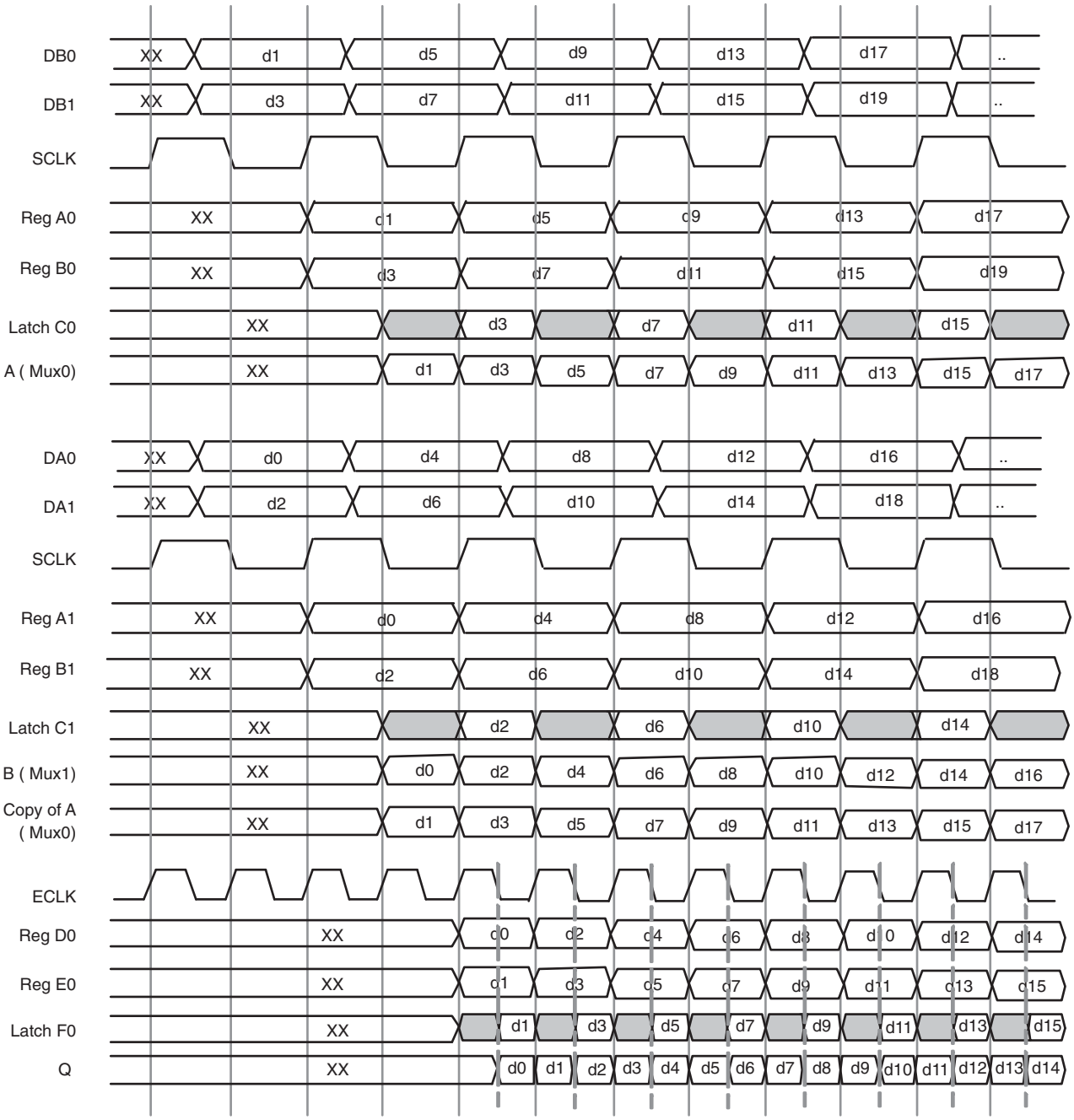


Figure 11-40 shows the timing waveform when using the ODDRXC module.

Figure 11-40. ODDRX2B Waveform



DELAYB

Data going to the DDR registers can be optionally delayed using the Delay block. The Delay block receives 4-bit delay control. The 4-bit delay can be set using fixed multiplier values or it can be controlled by the user. The DELAYB block is available for use with the input DDR registers.

The DELAYB block can be configured when generating the DDR input modules in the IPexpress tool of the software. The delay can be adjusted in 35ps steps. Users can choose from three types of delay values:

1. Dynamic – The delay value is controlled by the user logic using the DEL[3:0] input of the DELAYB block.

2. Fixed – When choosing the fixed value, the user will also need to choose from one of the 16 multiplier values. This will tie the inputs DEL[3:0] of the DELAYB block to a fixed value depending on the multiplier value chosen.
3. FIXED_XGMII – The DEL [3:0] will be configured with the delay value required when implementing a XGMII interface.

Figure 11-41 shows the primitive symbol for the DELAYB mode.

Figure 11-41. DELAYB Symbol

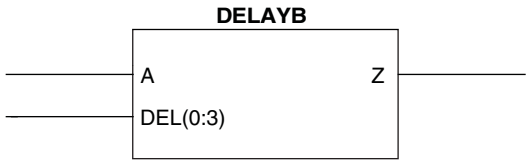


Table 11-11 lists the port names and descriptions for the DELAYB primitive.

Table 11-11. DELAYB Port Names

Port Name	I/O	Definition
A	I	DDR input from the sysIO buffer
DEL (0:3)	I	Delay inputs
Z	O	Delay DDR data

Design Rules/Guidelines

Listed below are some rules and guidelines for implementing generic DDR interfaces in LatticeXP2 devices.

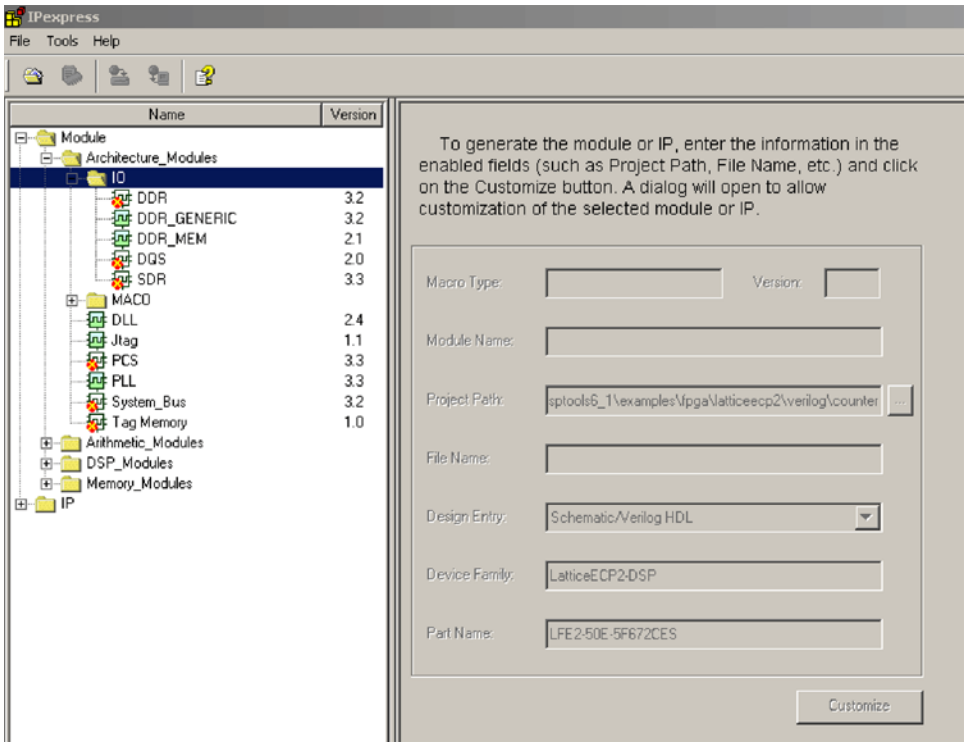
- When implementing a 2x gearing mode, the complement PIO registers are used. This complementary PIO register can no longer be used and should not be connected.

DDR Usage In IPexpress

IPexpress can be used to configure and generate the DDR Memory Interface and Generic DDR Module. The tool will generate an HDL module that will contain the DDR primitives. This module can be using in the top level design.

Figure 11-42 shows the main window of IPexpress. The DDR_Generic and DDR_MEM options under **Architecture->IO** are used to configure the DDR modules.

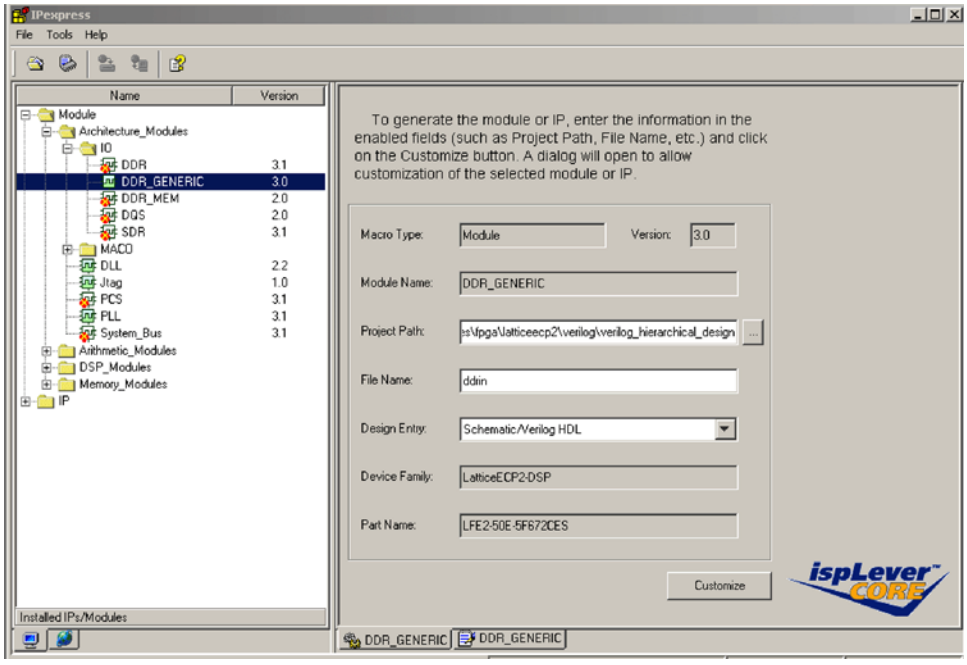
Figure 11-42. IPexpress Main Window



DDR Generic

Figure 11-43 shows the main window when DDR_Generic is selected. The only entry required in this window is the module name. Other entries are set to the project settings. The user may change these entries if desired. After entering the module name, click on **Customize** to open the **Configuration Tab** window as shown in Figure 11-44.

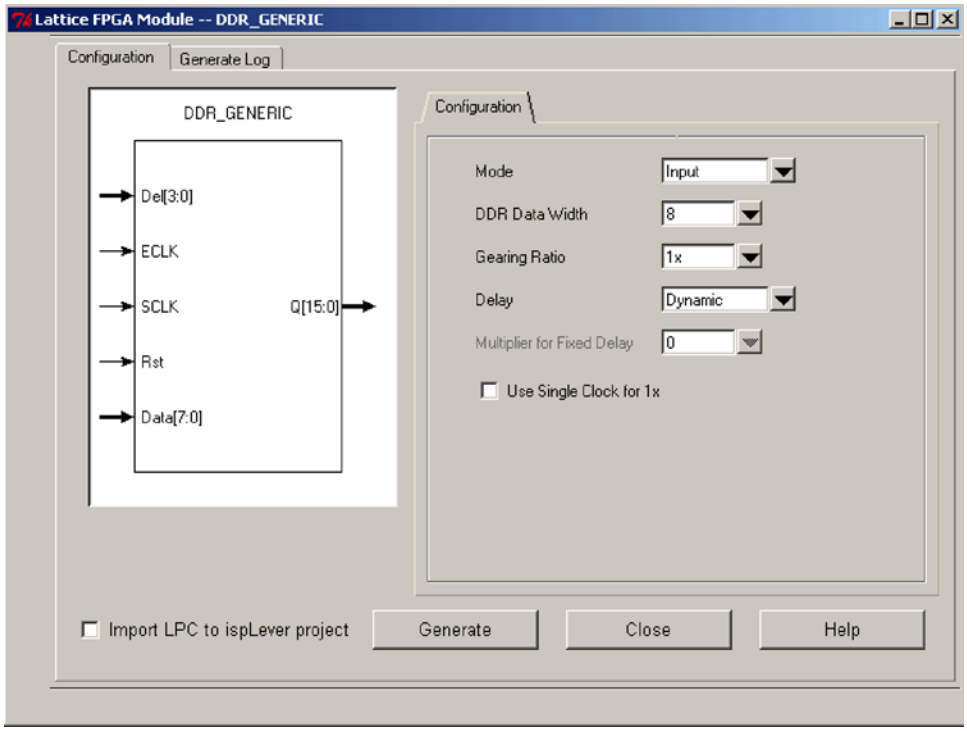
Figure 11-43. IPexpress Main Window for DDR_Generic



Configuration Tab

The Configuration Tab lists all user-accessible attributes with default values set. Upon completion, click **Generate** to generate source and constraint files. The user may choose to use the .lpc file to load parameters.

Figure 11-44. Configuration Tab for DDR_Generic



The user can change the Mode parameter to choose either Input, Output, Bidirection or Tristate DDR module. The other configuration parameters will change according to the mode selected. The Delay parameter is only available for Input and Bidirectional modes. Similarly the Multiplier for Fixed Delay parameter is only available when the Delay parameter is configured to Fixed.

Table 11-12. User Parameters in the IPexpress GUI

User Parameters	Description	Values/Range	Default
Mode	Mode selection for the DDR block.	Input, Output, Bidirectional, Tristate	Input
Data Width	Width of the data bus.	1-64	8
Gearing Ratio	Gearing ratio selection.	1x, 2x[1]	1x
Delay	Input delay configuration	Dynamic, Fixed, Fixed XGMII	Dynamic
Multiplier for Fixed Delay	Fixed delay setting. Available only when delay is configured as fixed.	0-15	0
Use Single Clk for 1x	Allows for the selection of a single clock for the gearing logic.	On/Off	Off

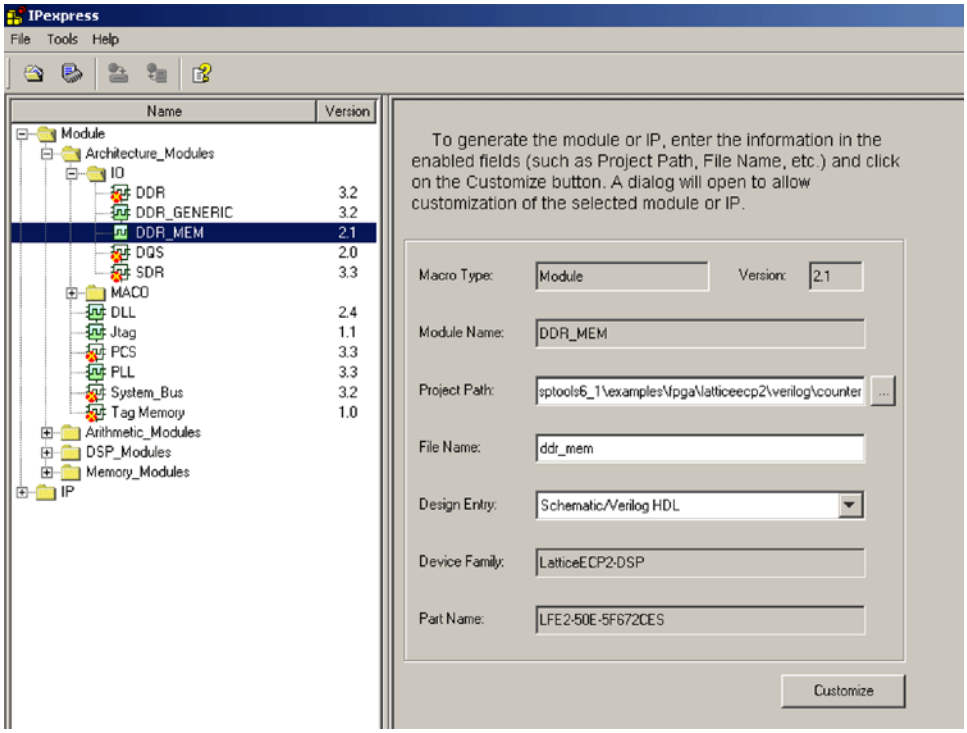
1. Only 1x available when Mode is Bidirection or Tristate.

DDR_MEM

Figure 11-45 shows the main window when DDR_MEM is selected. Similar to the DDR_Generic, the only entry required here is the module name. Other entries are set to the project settings. The user may change these entries

if desired. After entering the module name, click on **Customize** to open the **Configuration Tab** window as shown in Figure 11-46.

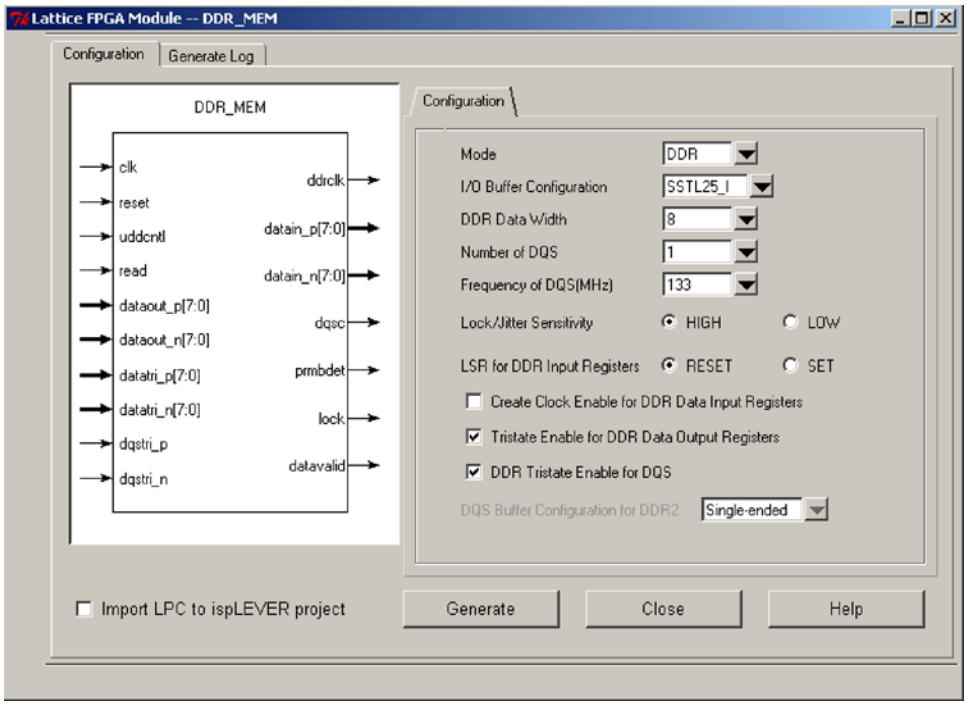
Figure 11-45. IPexpress Main Window for DDR_MEM



Configuration Tab

The Configuration Tab lists all user-accessible attributes with default values set. Upon completion, click **Generate** to generate source and constraint files. The user may choose to use the .lpc file to load parameters.

Figure 11-46. Configuration Tab for DDR_MEM



The user can change the Mode parameter to choose either the DDR or DDR2 interface. The other configuration parameters will change according to the Mode selected. The Number of DQS parameter determines the number of DDR interfaces. The software will assume there are eight data bits for every DQS. The user can also choose the frequency of operation and the DDR DLL will be configured to this frequency.

The user has an option to enable the clock enable and tristate enables for the DDR registers. It is recommend that the Lock/Jitter be enabled if the DDR interface is running at 150MHz or higher.

The parameters available depend on the mode selected. Tables 11-13 and 11-14 describe all user parameters in the IPexpress GUI and their usage for modes DDR and DDR2.

Table 11-13. User Parameters in the IPexpress GUI when in DDR Mode

User Parameters	Description	Values/Range	Default
I/O Buffer Configuration	I/O Standard used for the Interface. This will also depend on the Mode selected.	SSTL25_I, SSTL25_II	SSTL25_I
Data Width	Width of the Data bus	8-64	8
Number of DQS	Number of DQS will determine the number of DQS Groups	1, 2, 4, 8	1
Frequency of DQS	DDR Interface Frequency. This is also input to the DDR DLL. The values will depend on the mode selected.	100MHz, 133MHz, 166MHz, 200MHz	200MHz
Lock/Jitter Sensitivity	DLL Sensitivity to Jitter	High, Low	High
LSR for DDR Input Register	LSR Control	RESET, SET	RESET
Create Clock Enable for DDR Input Register	Create Clock enable inputs to the block	On/Off	Off
Tri-state Enable for DDR Output Registers	Creates Tri-state control for the DDR data output registers.	On/Off	On
DDR Tristate enable for the DQS output	Creates Tristate control for DQS output	On/Off	On

Table 11-14. User Parameters in the IPexpress GUI when in DDR2 Mode

User Parameters	Description	Values/Range	Default
I/O Buffer Configuration	I/O Standard used for the Interface. This will also depend on the Mode selected.	SSTL18_I, SSTL18_II	SSTL18_I
Data Width	Width of the Data bus	8-64	8
Number of DQS	Number of DQS will determine the number of DQS Groups	1, 2, 4, 8	1
Frequency of DQS	DDR Interface Frequency. This is also input to the DDR DLL. The values will depend on the mode selected.	166MHz, 200MHz, 266MHz	200MHz
Lock/Jitter Sensitivity	DLL Sensitivity to Jitter	High, Low	High
LSR for DDR Input Register	LSR Control	RESET, SET	RESET
Create Clock Enable for DDR Input Register	Create Clock enable inputs to the block	On/Off	Off
Tri-state Enable for DDR Output Registers	Creates Tri-state control for the DDR data output registers.	On/Off	On
DDR Tristate enable for the DQS output	Creates Tristate control for DQS output	On/Off	On
DQS Buffer Configuration for DDR2	DQS Buffer can be configured as Differential	On/Off	Off

FCRAM ("Fast Cycle Random Access Memory") Interface

FCRAM is a DDR-type DRAM, which performs data output at both the rising and falling edges of the clock. FCRAM devices operate at a core voltage of 2.5V with SSTL Class II I/O. It has enhanced both the core and peripheral logic of the SDRAM. In FCRAM the address and command signals are synchronized with the clock input, and the data pins are synchronized with the DQS signal. Data output takes place at both the rising and falling edges of the DQS. DQS is in phase with the clock input of the device. The DDR SDRAM and DDR FCRAM controller will have different pinouts.

LatticeXP2 devices can implement the FCRAM interface using dedicated DQS logic, input DDR registers and output DDR registers, as described in the Implementing Memory Interfaces section of this document. Generation of address and control signals for FCRAM are different than in DDR SDRAM devices. Please refer to the FCRAM data sheets to see detailed specifications. Toshiba, Inc. and Fujitsu, Inc. offer FCRAM devices in 256Mb densities. They are available in x8 or x16 configurations.

Board Design Guidelines

The most common challenge associated with implementing DDR memory interfaces is the board design and layout. It is required that users strictly follow the guidelines recommended by memory device vendors.

Some of the common recommendations include matching trace lengths of interface signals to avoid skew, proper DQ-DQS signal grouping, proper termination of the SSTL2 or SSTL18 I/O Standard, proper VREF and VTT generation decoupling and proper PCB routing.

The following documents include board layout guidelines:

- www.idt.com, IDT, *PCB Design for Double Data Rate Memory*
- www.motorola.com, AN2582, *Hardware and Layout Design Considerations for DDR Interfaces*

References

- www.jedec.org, JEDEC Standard 79, Double Data Rate (DDR) SDRAM Specification
- www.micron.com, DDR SDRAM Data Sheets

- www.infinion.com, DDR SDRAM Data Sheets
- www.samsung.com, DDR SDRAM Data Sheets
- www.latticesemi.com, RD1019, *QDR Memory Controller Reference Design for Lattice ECP/EC Devices*
- www.toshiba.com, DDR FCRAM Data Sheet
- www.fujitsu.com, DDR FCRAM Data Sheet
- www.latticesemi.com, *LatticeEC Advanced Evaluation Board User's Guide*
- www.latticesemi.com, *DDR SDRAM Controller (Pipelined Version for Lattice ECP/EC and LatticeXP™ Devices) User's Guide*

Technical Support Assistance

Hotline: 1-800-LATTICE (North America)
+1-503-268-8001 (Outside North America)
e-mail: techsupport@latticesemi.com
Internet: www.latticesemi.com

Revision History

Date	Version	Change Summary
February 2007	01.0	Initial release.
May 2007	01.1	Updated the port names on the input DDR block diagrams.
		Updated text in DQS Transition Detect section under Memory Read Implementation.

Power Estimation and Management for LatticeXP2 Devices

February 2007 Technical Note TN1139

Introduction

One requirement for design engineers using programmable devices is the ability to calculate the power dissipation for a particular device used on a board. Lattice's ispLEVER® design tools include the Power Calculator tool, which allows designers to calculate the power dissipation for a given device. This technical note describes how to use the Power Calculator tool to calculate the power consumption of the LatticeXP2™ family of devices. General guidelines to reduce power consumption are also included.

Power Supply Sequencing and Hot Socketing

LatticeXP2 devices have been designed to ensure predictable behavior during power-up and power-down. Power supplies can be sequenced in any order. During power-up and power-down sequences, the I/Os remain in tri-state until the power supply voltage is high enough to ensure reliable operation. In addition, leakage into I/O pins is controlled to within the limits specified in the LatticeXP2 Family Data Sheet, allowing for easy integration with the rest of the system. These capabilities make the LatticeXP2 ideal for many multiple power supply and hot-swap applications.

Recommended Power-up Sequence

As described above, the supplies can be sequenced in any order. However, once internal power good is achieved (determined by VCC, VCCAUX, VCCIO Bank x) the device releases the I/Os from tri-state and the management of the I/Os becomes the designer's responsibility. To simplify a system design, it is recommended that supplies be sequenced in the following order: VCCIO, VCC, VCCAUX. If VCCIO is tied to VCC or VCCAUX, then it is recommended that VCCIO and the associated power supply are powered up before the remaining supply.

Please refer to the Hot Socketing section of the LatticeXP2 Family Data Sheet for more information.

Power Calculator Hardware Assumptions

Power consumption for a device can be coarsely broken down into the DC portion and the AC portion.

The Power Calculator reports the power dissipation in terms of:

1. DC portion of the power consumption.
2. AC portion of the power consumption.

The DC Power (or the Static power consumption) is the total power consumption of the used and the unused resources. These power components are fixed for each resource used and depends upon the number of resource units utilized. The DC component also includes the static power dissipation for the unused resources of the device.

The AC portion of power consumption is associated with the used resources. This is the dynamic part of the power consumption. Its power dissipation is directly proportional to the frequency at which the resource is running and the number of resource units used.

Power Calculation Equations

The following are the power equations used in the Power Calculator:

Total DC Power (Resource)

= Total DC Power of Used Portion + Total DC Power of Unused Portion
= [DC Leakage per Resource when Used * $N_{RESOURCE}$]
+ [DC Leakage per Resource when Unused * ($N_{TOTAL\ RESOURCE}$ - $N_{RESOURCE}$)]

Where:

$N_{TOTAL\ RESOURCE}$	is the total number of resources in a device
$N_{RESOURCE}$	is the number of resources used in the design

The total DC power consumption for all the resources as per the design data is the Quiescent Power in the Power Calculator.

The AC Power is governed by the following equation:

Total AC Power (Resource)
= $K_{RESOURCE}$ * f_{MAX} * $AF_{RESOURCE}$ * $N_{RESOURCE}$

Where:

$N_{TOTALRESOURCE}$	is the number of resources in a device
$N_{RESOURCE}$	is the number of resources used in the design
$K_{RESOURCE}$	is the power constant for the resource in mW/MHz
f_{MAX}	is the max frequency at which the resource is running. Frequency is measured in MHz.
$AF_{RESOURCE}$	is the activity factor for the resource group.The Activity Factor is a percentage of the switching frequency.

For example, the power consumption of the Slice portion is calculated in the following equation,

Total DC Power (Slice)
= Total DC Power of Used Portion + Total DC Power of Unused Portion
= [DC Leakage per Slice when Used * N_{SLICE}]
= +[DC Leakage per Slice when Unused * ($N_{TOTALSLICE}$ - N_{SLICE})]

Total AC Power (Slice)
= K_{SLICE} * f_{MAX} * AF_{SLICE} * N_{SLICE}

Power Calculations

The Power Calculator allows users to estimate power consumption at three different levels:

1. Estimate of the utilized resources before completing place and route
2. Post place and route design
3. Post place and route, post trace, and post simulation

For the first level of estimation, the user provides estimates of device usage in the Power Calculator Wizard and the tool provides a rough estimate of the power consumption.

The second level is a more accurate approach. The user imports the actual device utilization by importing the post Place and Route netlist (NCD) file.

The third level brings even more accuracy to the calculation by importing the Trace (TWR) file to populate the maximum frequencies (f_{MAX}) into the tool. Users also have the option of importing the information from the post simulation (VCD) file and calculating the Activity Factor and Toggle Rates of various components.

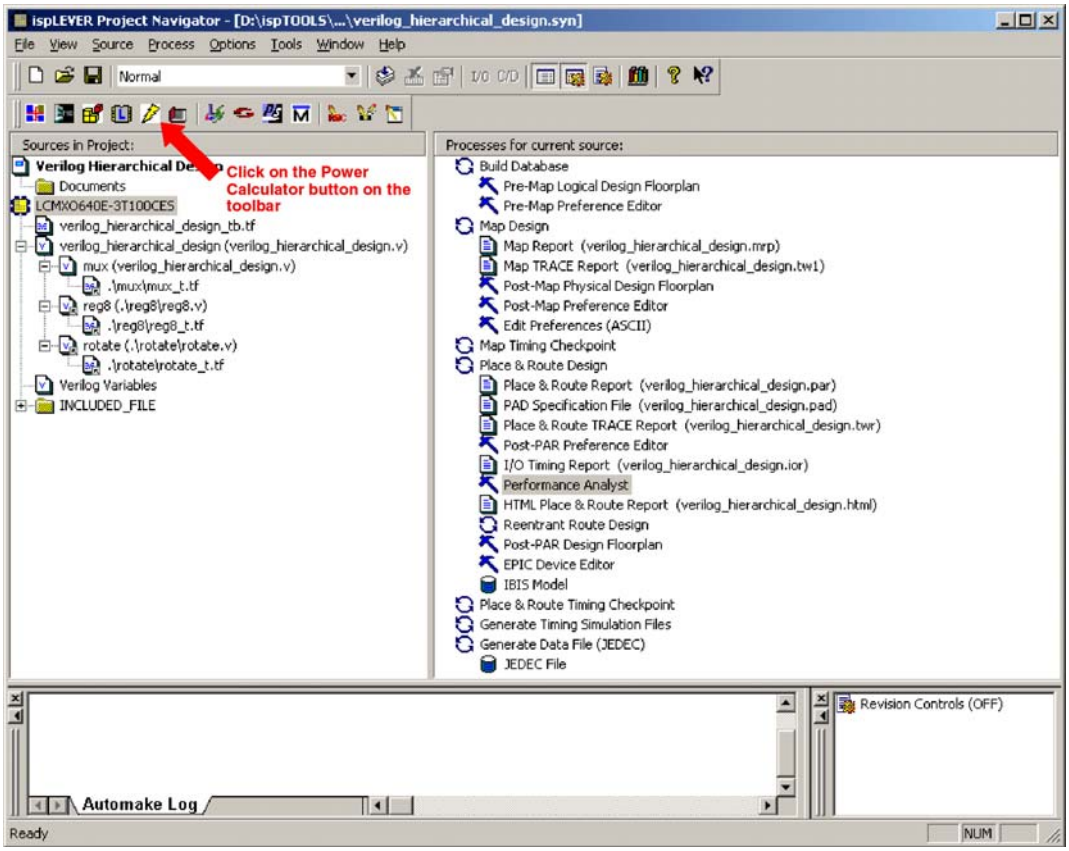
These three stages of power calculation are discussed in detail in the following sections of this document.

Using the Power Calculator

Starting the Power Calculator

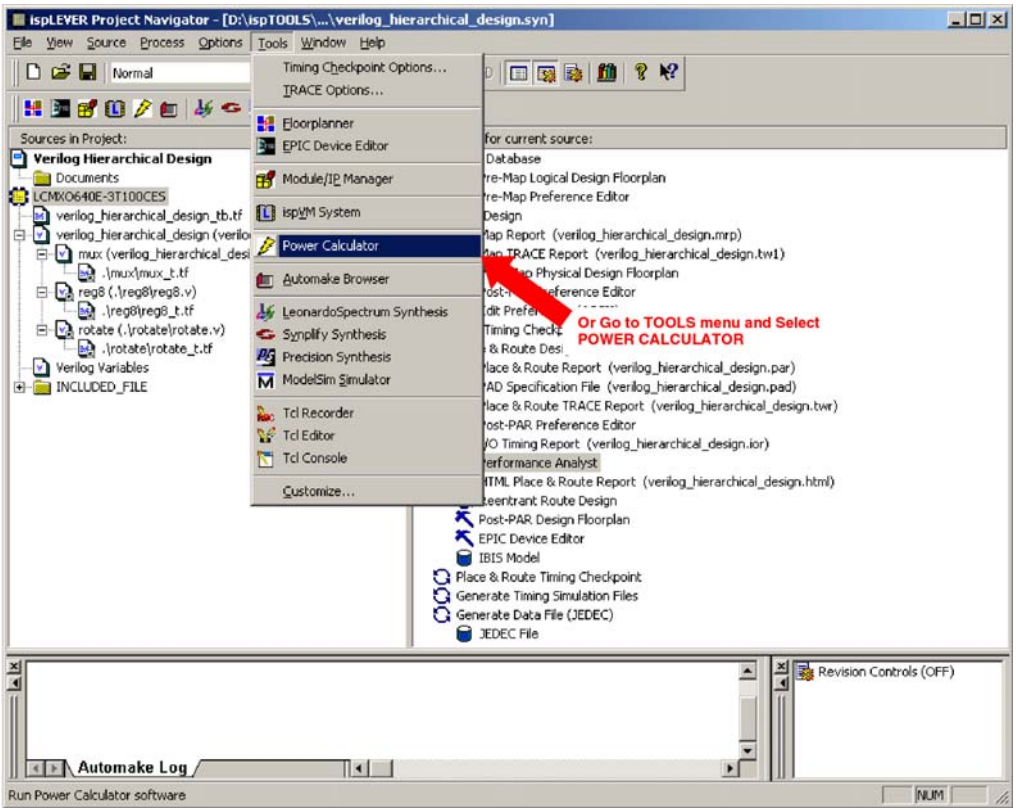
The Power Calculator can be launched by one of two methods. The first method is to click the **Power Calculator button** in the toolbar as shown in Figure 12-1.

Figure 12-1. Starting Power Calculator from the Toolbar



The Power Calculator can also be launched by going to the **Tools** menu and selecting the option **Power Calculator** as shown in Figure 12-2.

Figure 12-2. Starting Power Calculator from Tools Menu

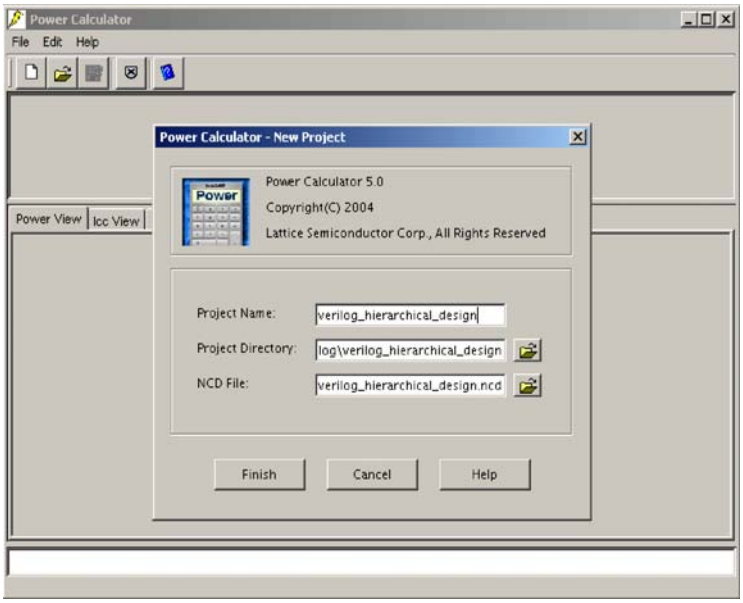


Note: The Power Calculator does not support some older Lattice devices. The toolbar button and menu item are only present when supported devices are selected.

Creating a Power Calculator Project

After starting the Power Calculator, the Power Calculator window is displayed. Click on the **File** menu and select **New** to get to the **Start Project** window as shown in Figure 12-3.

Figure 12-3. Power Calculator Start Project Window (Create New Project)



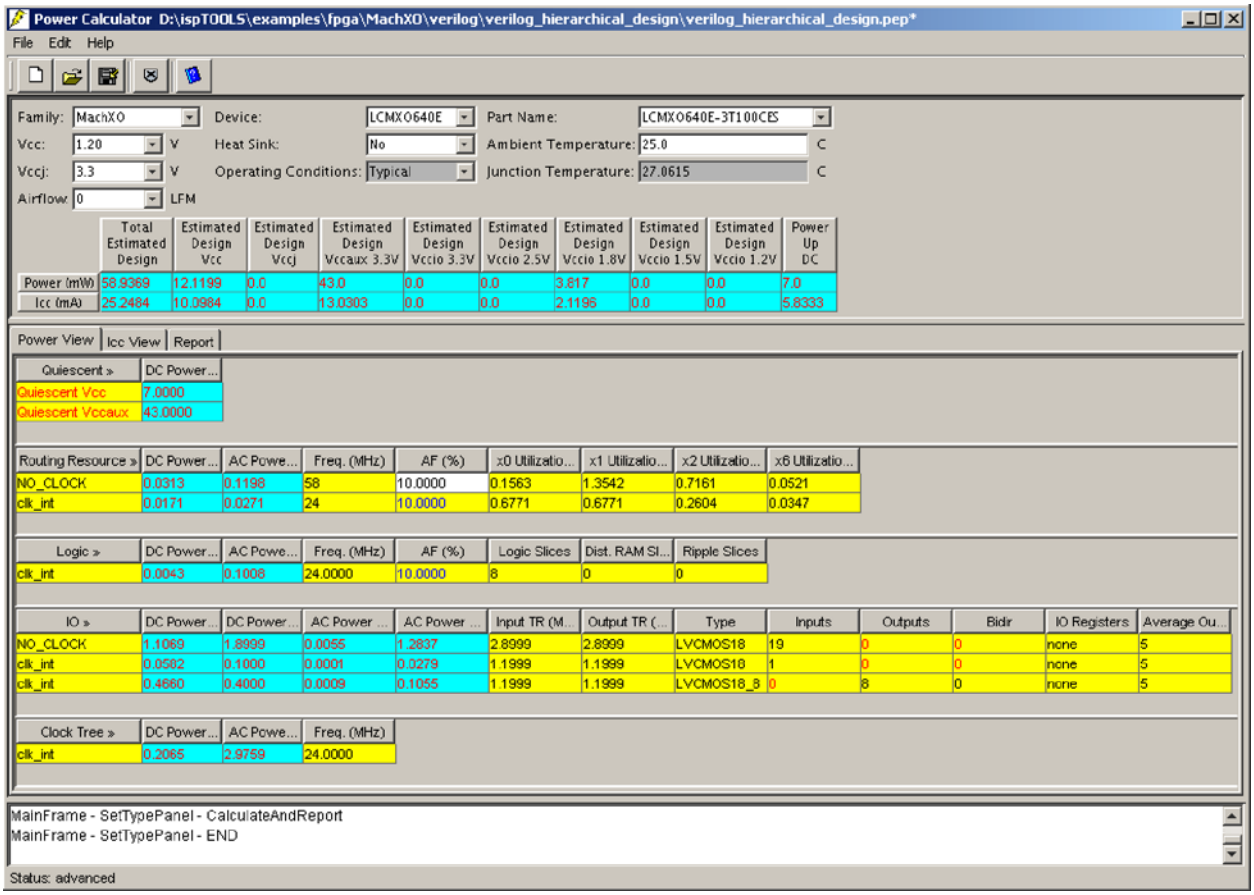
The Start Project window is used to create a new Power Calculator Project (*.pep project). Three pieces of data are entered into the Start Project window.

1. The Power Calculator project name by default is same as the Project Navigator project name. It can be changed, if desired.
2. Project Directory is where the Power Calculator project (*.pep) file will be stored. By default it is stored in the main project folder.
3. The NCD file is automatically selected, if available, or users can browse to the NCD file in a different location.

Power Calculator Main Window

The Power Calculator main window is shown in Figure 4.

Figure 12-4. Power Calculator Main Window (Type View)



The top pane of the window shows information about the device family, device and the part number as it appears in the Project Navigator. The VCC used for the power calculation is also listed. Users have the choice of selecting the core voltage VCC with +5% of the nominal value (or values). The option of selecting VCCJ is also available. Users can provide the ambient temperature, and the junction temperature is calculated based on that.

Users can also enter values for airflow in Linear Feet per Minute (LFM) along with heat sink to obtain the junction temperature.

The table near the top of the Power Calculator main window summarizes the currents and power consumption associated with each type of power supply for the device. This also takes into consideration the I/O power supplies.

In the middle pane of the window, there are three tabs:

1. **Power View**

   The first tab is the Power view. This tab provides an interactive spreadsheet type interface with all values in terms of power consumption mW.

   The first column breaks down the design in terms of clock domains. The second and third columns, which are shaded blue, provide the DC (static) and AC (dynamic) power consumption, respectively.

   Next are four columns shaded blue. These provide information on the I/O DC and AC power, including core voltage, VCC and the I/O voltage supply, VCCIO.

   The first three rows show the Quiescent Power for VCC, VCCAUX and VCCJ. These are DC power numbers for a blank device or device with no resource utilization.

   Some of the cells are shaded yellow. These cells are editable cells and users can type in values such as frequency, activity factors and resource utilization. The second tab (the Report tab) is the summary of the Power View. This report is in text format and provides details of the power consumption.

2. **ICC View**

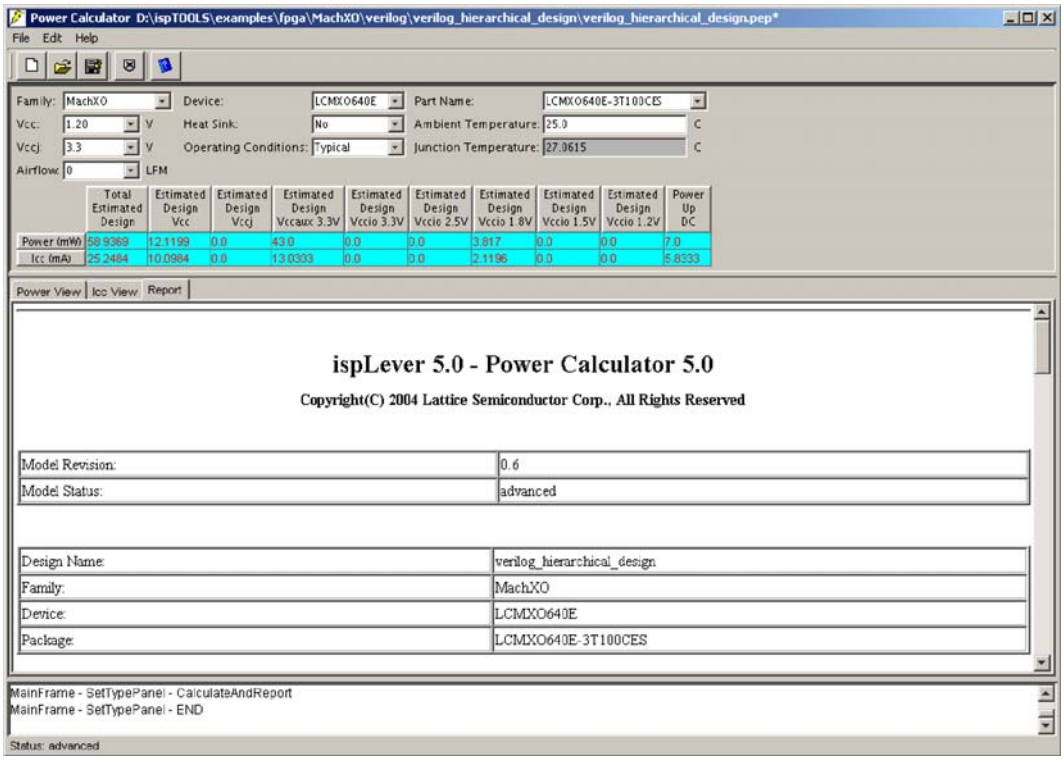
   The second tab is the ICC view. This tab is exactly same as the Power View tab, except that all values are expressed in terms of current or mA.

3. **Report View**

   The third and final tab is the Report view. This is an HTML type of Power Calculator report. It summarizes the contents of the Power and ICC views.

The final pane, the lower pane of the window, is the log pane where users can see a log of the various operations in the Power Calculator.

Figure 12-5. Power Calculator Main Window (Power Report View)

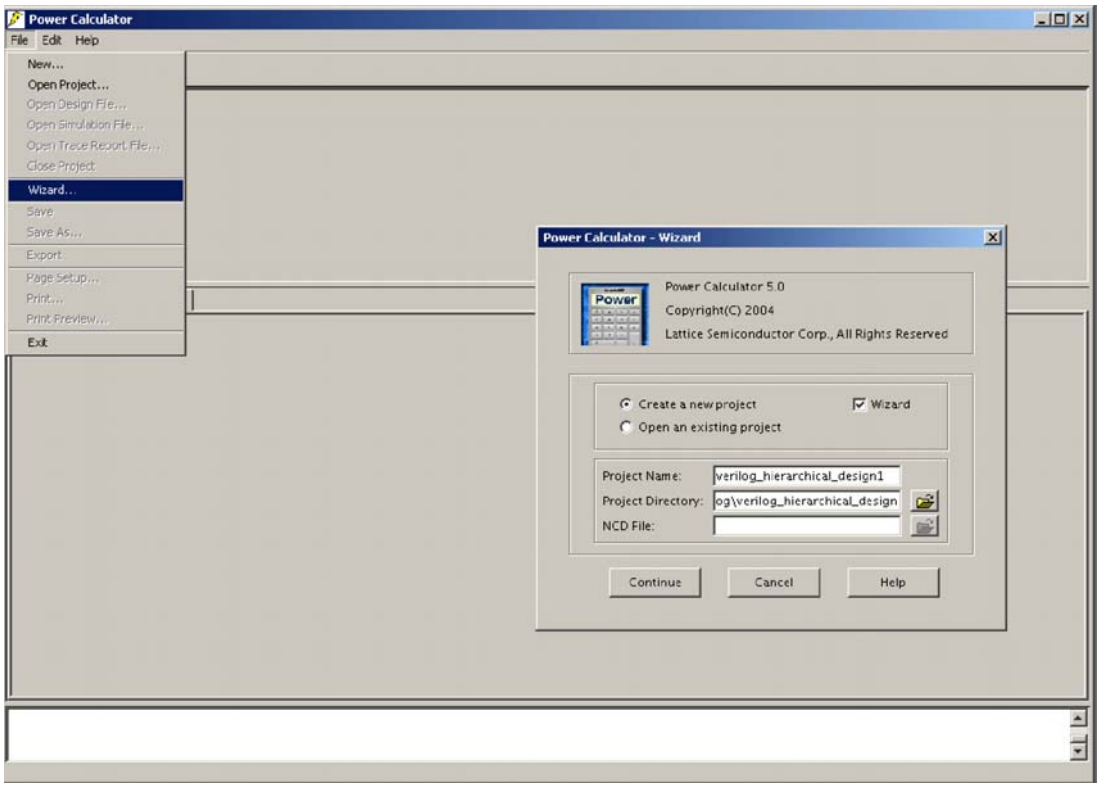


Power Calculator Wizard

The Power Calculator Wizard allows users to estimate the power consumption of a design. This estimation is done before the design is created. It is important for the user to understand the logic requirements of the design. The wizard allows the user to provide the required parameters and then estimates the power consumption of the device.

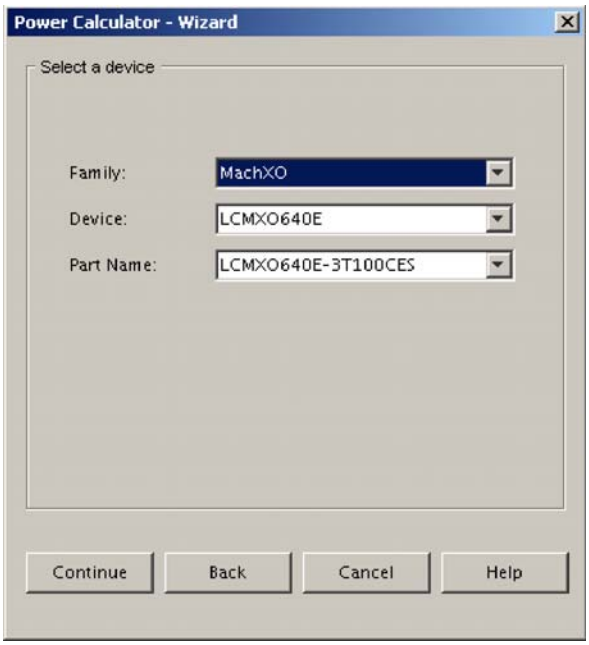
To start the Power Calculator in the wizard mode, go to the **File** menu and select **Wizard**. Alternatively, you can click on the **Wizard button** to see the Power Calculator - Wizard window as shown in Figure 12-6. Select the option **Create a new Project** and check the **Wizard check box** in the Power Calculator Start Project window. Provide the project name and the project folder and click **Continue**. Since this is power estimation before the actual design, no NCD file is required.

Figure 12-6. Power Calculator Start Project Window (Using the New Project Window Wizard)



In the next screen, the device family, device, and part number are chosen. After making the proper selections, click **Continue**. This is shown in Figure 7.

Figure 12-7. Power Calculator Wizard Mode Window - Device Selection

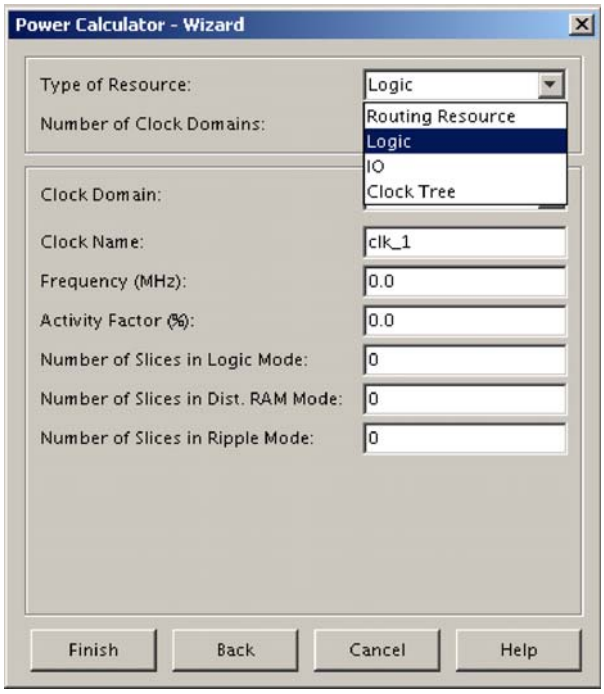


In the following screens (Figures 8-12) users can select additional resources, such as I/O types, clock name, frequency at which the clock in running and other parameters by selecting the appropriate resource using the pull-down **Type** menu.

1. Routing Resources
2. Logic
3. EBR
4. I/O
5. PLL
6. Clock Tree

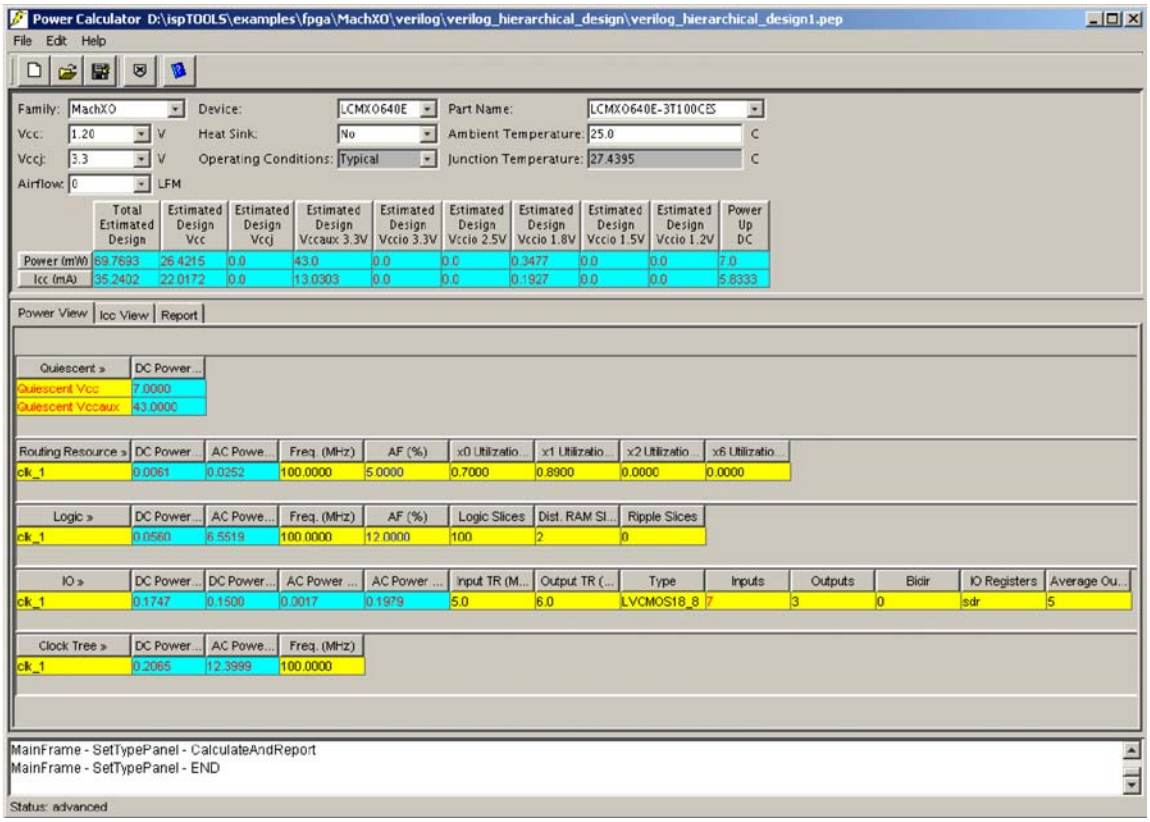
The number in these windows refers to the number of clocks and the index corresponds to each of the clocks. By default, the clock names are clk_1, clk_2, and so on. Clock names can be changed by typing in the Clock Name text box. For each clock domain and resource, parameters such as Frequency and Activity Factor can be specified. In the final window, click the **Create** button for each clock driven resource to include the parameters specified for it.

Figure 12-8. Power Calculator Wizard Mode Window - Resource Specification



These parameters are then used in the Power Type View window, which can be seen upon clicking on **Finish** (see Figure 9).

Figure 12-9. Power Calculator Wizard Mode - Main Window



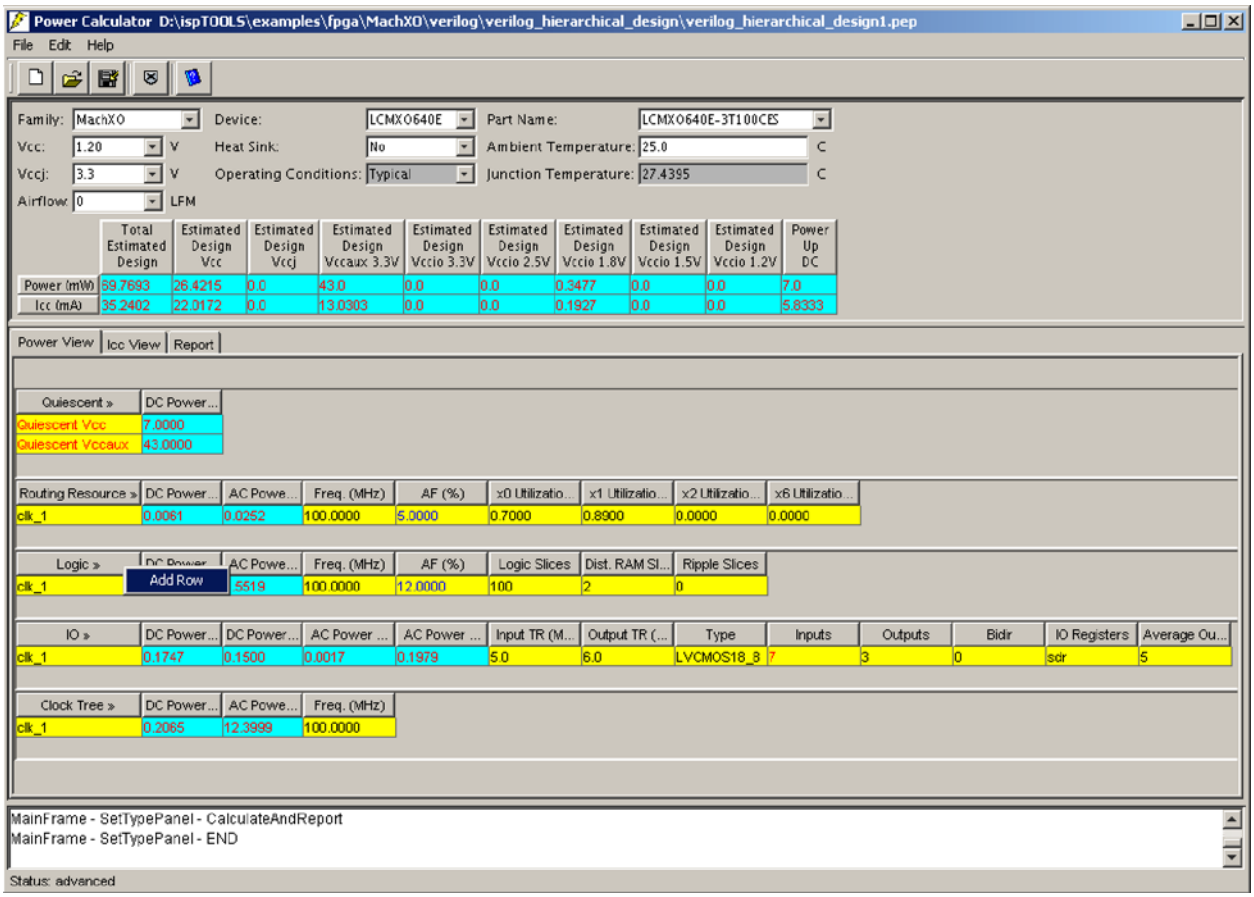
Creating a New Project Without the NCD File

A new project can be started without the NCD file, either by using the wizard (as discussed above) or by selecting the **Create a New Project** option in the Power Calculator - Start Project. The project name and project directory must be provided. After clicking **Continue** the Power Calculator main window will be displayed.

However, in this case there are no resources added. The power estimation row for the routing resources is always available in the Power Calculator. Additional information such as the Slice, EBR, I/O, PLL, and clock tree utilization must be provided to calculate the power consumption.

For example, if the user wants to add the logic resources shown in Figure 10, right-click on **Logic >>** and then select **Add Row** in the menu that pops up.

Figure 12-10. Power Calculator Main Window - Adding Resources



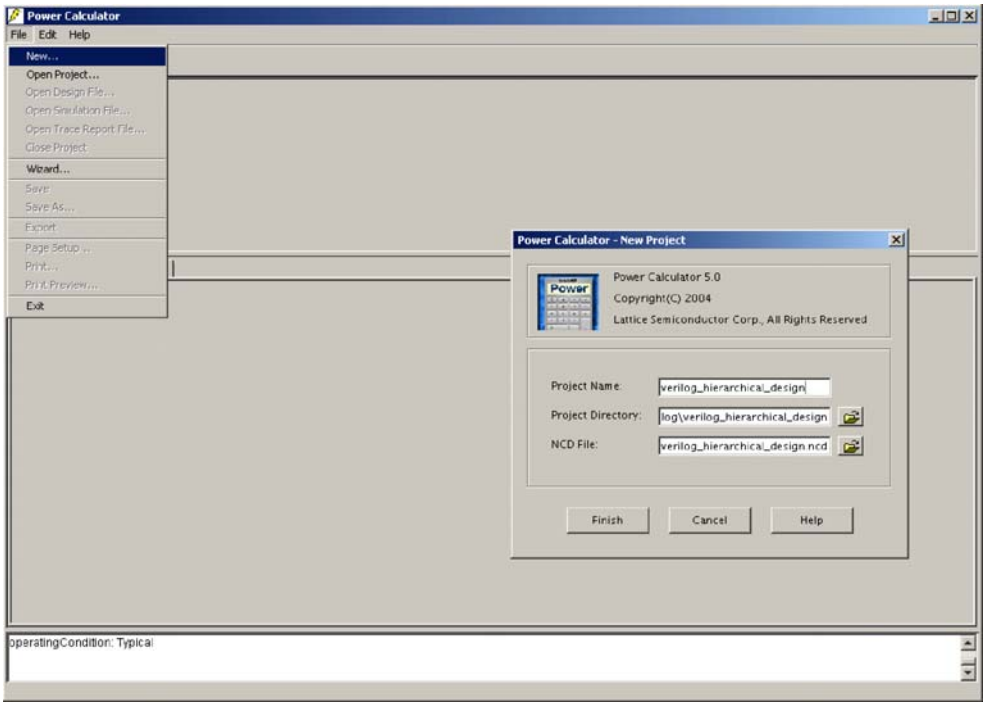
This adds a new row for the logic resource utilization with clock domain named clk_1.

Similarly, other resources like EBR, I/Os, PLLs and routing can be added. Each of these resources is used for AC power estimation and categorized by clock domains.

Creating a New Project with the NCD File

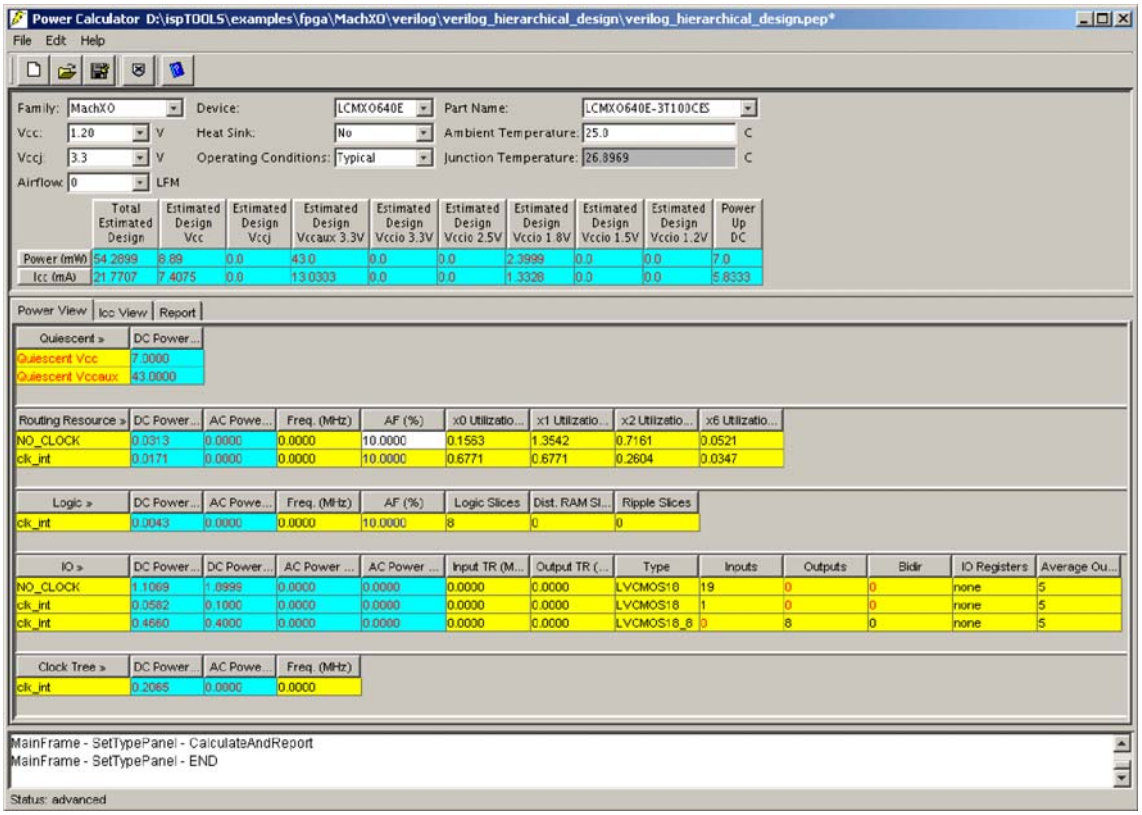
If the post place and routed NCD file is available, the Power Calculator can use it to import accurate information about the design data and resource utilization for power calculation. After starting the Power Calculator, the NCD file is automatically placed in the NCD File option, if it is available in the project directory. Otherwise, browse to the NCD file using the Power Calculator.

Figure 12-11. Power Calculator Start Project Window with Post PAR NCD File



The information from the NCD file is automatically inserted into the correct rows and Power Calculator uses the Clock names from the design as shown in Figure 12-12.

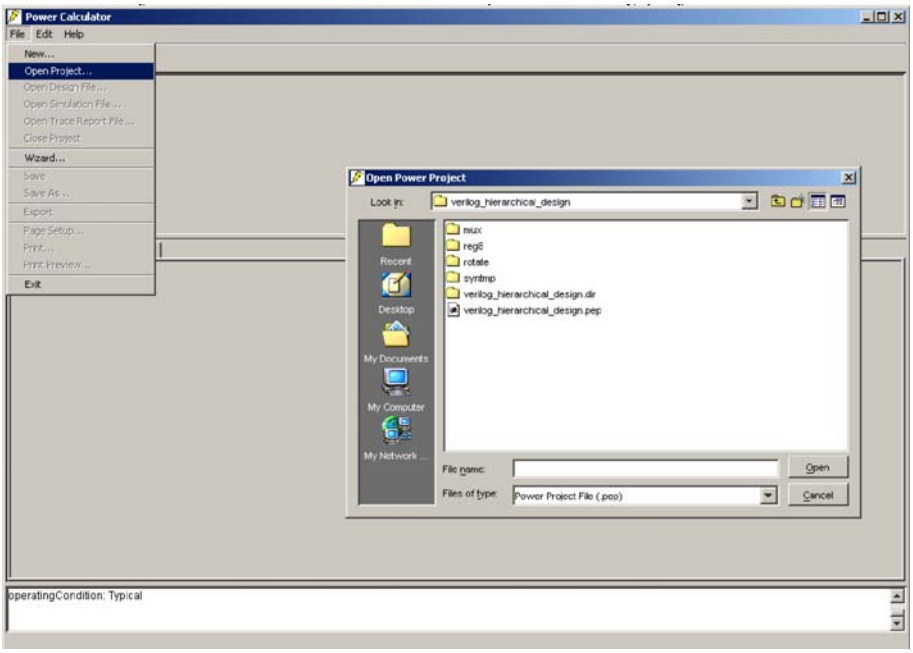
Figure 12-12. Power Calculator Main Window - Resource Utilization Picked Up from the NCD File



Opening an Existing Project

The Power Calculator - Start Project window also allows users to open an existing project. Select **Open Existing Project** and browse to the ***.pep project file** and click **Continue**. This opens the existing project in windows similar to those discussed above (see Figure 12-13).

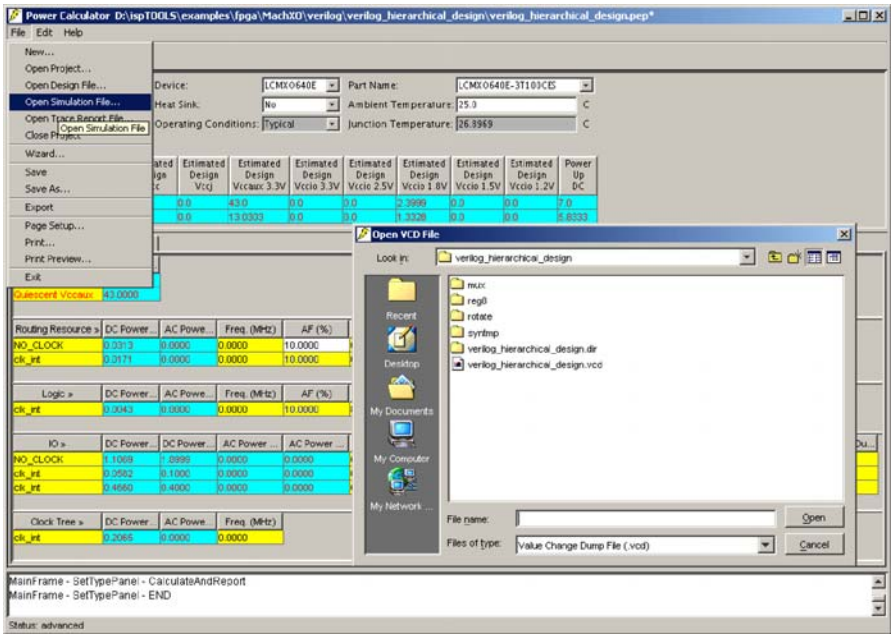
Figure 12-13. Opening an Existing Project in Power Calculator



Importing a Simulation File (VCD)

A post simulation VCD file can be imported into the Power Calculator project to estimate a design's activity factors. Under the **File** menu, select **Open Simulation File** and browse to the VCD file location and select **Open**. All AF and Toggle Rate fields are then populated with the information from this file, as shown in Figure 12-14.

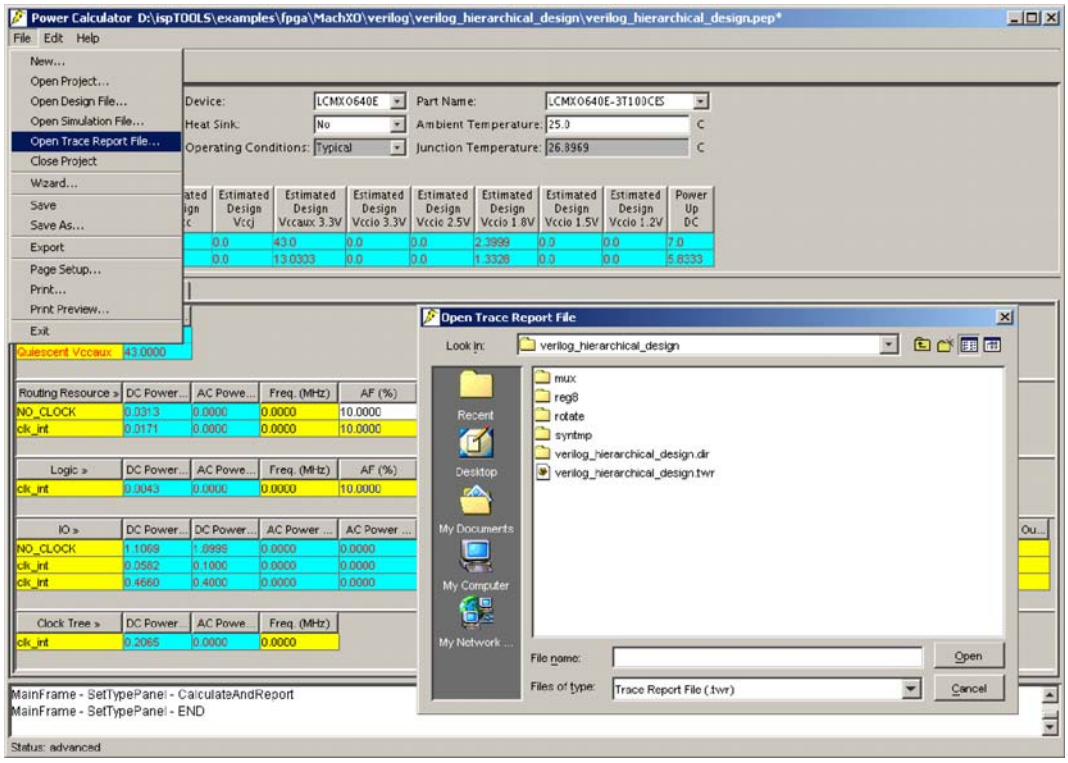
Figure 12-14. Importing a Simulation File in an Existing Power Calculator Project



Importing a Trace Report File (TWR)

Post Trace TWR file can be imported into the Power Calculator project to estimate a design's activity factors. Under the **File** menu, select **Open Trace Report File**, browse to the TWR file location and select **Open**. All Freq. (MHz) fields are then populated with the information from this file, as shown in Figure 12-15.

Figure 12-15. Importing a Trace Report File to a Power Calculator Project



Activity Factor and Toggle Rate

Activity Factor% (or AF%) is defined as the percentage of frequency (or time) that a signal is active or toggling of the output. Most of the resources associated with a clock domain are running or toggling at some percentage of the frequency at which the clock is running. Users must provide this value as a percentage under the AF% column in the Power Calculator tool.

Another term used is the I/O Toggle Rate or the I/O Toggle Frequency. The AF% is applicable to the PFU, routing and memory read write ports, etc. The activity of the I/Os is determined by the signals provided by the user (for inputs) or as an output of the design (for outputs). Therefore, the rates at which I/Os toggle define their activity. The Toggle Rate (or TR) in MHz of the output is defined by the following equation:

$$\text{Toggle Rate (MHz)} = 1/2 * f_{MAX} * \text{AF\%}$$

Users must provide the TR (MHz) value for the I/O instead of providing the Frequency and AF% in case of other resources. The AF can be calculated for each routing resource, output or PFU. However, it involves long calculations. The general recommendation for a design occupying roughly 30% to 70% of the device is an AF% of 15% to 25%. This is an average value that can be seen most of the design. An accurate value for an activity factor depends upon clock frequency, stimulus to the design and the final output.

Ambient and Junction Temperatures and Airflow

A common method of characterizing a packaged device's thermal performance is with thermal resistance, or Θ. For a semiconductor device, thermal resistance indicates the steady state temperature rise of the die junction above a given reference for each watt of power (heat) dissipated at the die surface. Its units are °C/W.

The most common examples are Θ_{JA}, Thermal Resistance Junction-to-Ambient (in °C/W) and Θ_{JC}, Thermal Resistance Junction-to-Case (also in °C/W). Another factor is Θ_{JB}, Thermal Resistance Junction-to-Board (in °C/W).

Knowing the reference (i.e. ambient, case, or board) temperature, the power, and the relevant Θ value, the junction temp can be calculated by the following equations.

$$T_J = T_A + \Theta_{JA} * P \quad (1)$$

$$T_J = T_C + \Theta_{JC} * P \quad (2)$$

$$T_J = T_B + \Theta_{JB} * P \quad (3)$$

While T_J, T_A,T_C and T_B are the Junction, Ambient, Case (or Package) and Board temperatures (in °C) respectively, P is the total power dissipation of the device.

Θ_{JA} is commonly used with natural and forced convection air-cooled systems. Θ_{JC} is useful when the package has a high conductivity case mounted directly to a PCB or heatsink. And Θ_{JB} applies when the board temp adjacent to the package is known.

The Power Calculator utilizes the Ambient Temperature (°C) to calculate the Junction Temperature (°C) based on the Θ_{JA} for the targeted device, per Equation 1 above. Users can also provide the Airflow values (in LFM) to get a more accurate value of the Junction temperature.

Managing Power Consumption

One of the most critical design factors today is the reduction of system power consumption, especially for modern handheld devices and electronics. There are several design techniques that can significantly reduce overall system power consumption. Some of these include:

1. Reducing operating voltage.
2. Operating within the specified package temperature limitations.
3. Using the optimum clock frequency to reduce power consumption, since dynamic power is directly proportional to the frequency of operation. Designers must determine if a portion of their design can be clocked at a lower rate that will reduce power.
4. Reducing the span of the design across
5. Reducing the voltage swing of the I/Os where possible.
6. Using optimum encoding where possible. For example, a 16-bit binary counter has, on average, only 12% Activity Factor and a 7-bit binary counter has an average of 28% Activity Factor. On the other hand, a 7-bit Linear Feedback Shift Register could toggle as much as 50% Activity Factor, which causes higher power consumption. A gray code counter, where only one bit changes at each clock edge, will use the least amount of power, as the Activity Factor would be less than 10%.
7. Minimize the operating temperature, by the following methods:
    a. Use packages that can better dissipate heat. For example, packages with lower thermal impedance.
    b. Place heat sinks and thermal planes around the device on the PCB.
    c. Better airflow techniques using mechanical airflow guides and fans (both system fans and device mounted fans).

Power Calculator Assumptions

The following are the assumptions made by the Power Calculator:

1. The Power Calculator tool is based on equations with constants based on a room temperature of 25°C.
2. The user can define the Ambient Temperature (T_A) for device Junction Temperature (T_J) calculation based on the power estimation. T_J is calculated from user-entered T_A and power calculation of typical room temperature.
3. I/O power consumption is based on output loading of 5pF. Users can change this capacitive loading.
4. The Power Calculator provides an estimate of the power dissipation and current for each of the following types of power supplies: VCC, VCCIO, VCCJ and VCCAUX.
5. The nominal VCC is used by default to calculate power consumption. A lower or higher VCC can be chosen from a list of available values.
6. Airflow in Linear Feet per Minute (LFM) can be entered by the user along with the Heat Sink option to calculate the Junction Temperature.
7. The default value of the I/O types is LVCMOS12, 6mA.
8. The Activity Factor (AF) is defined as the toggle rate of the registered output. For example, assuming that the input of a flip-flop is changing at every clock cycle, 100% AF of a flip-flop running at 100MHz is 50MHz.

Technical Support Assistance

Hotline: 1-800-LATTICE (North America)
+1-503-268-8001 (Outside North America)
e-mail: techsupport@latticesemi.com
Internet: www.latticesemi.com

Revision History

Date	Version	Change Summary
February 2007	01.0	Initial release.

LatticeXP2 sysDSP Usage Guide

February 2007 Technical Note TN1140

Introduction

This technical note discusses how to access the features of the LatticeXP2™ sysDSP™ (Digital Signal Processing) Block described in the LatticeXP2 Family Data Sheet. Designs targeting the sysDSP Block can offer significant improvement over traditional LUT-based implementations. Table 13-1 provides an example of the performance and area benefits of this approach:

Table 13-1. sysDSP Block vs. LUT-based Multipliers

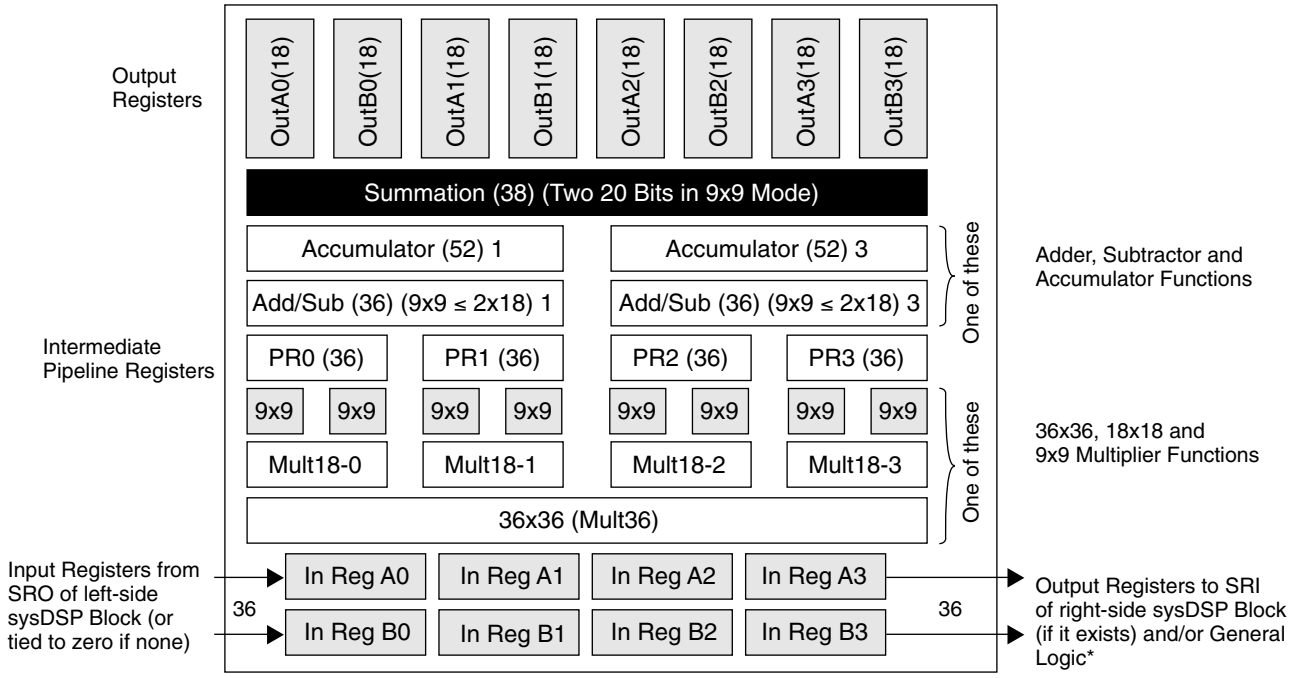
Multiplier Width	Register Pipelining	XP2-17-7 Uses One sysDSP Block		XP2-17-7 Uses LUTs	
		f_{MAX} (MHz)[1]	LUTs	f_{MAX} (MHz)[1]	LUTs
9x9	Input, Multiplier, Output	365	0	103	192
18x18	Input, Multiplier, Output	365	0	76	698
36x36	Input, Multiplier, Output	323	0	50	2732

1. These timing numbers were generated using the ispLEVER® design tool. Exact performance may vary with design and tool version.

sysDSP Block Hardware

The LatticeXP2 sysDSP Blocks are located in rows throughout device. Below is a block diagram of one of the sysDSP Blocks:

Figure 13-1. LatticeXP2 sysDSP Block



*Can only be routed to general logic routing when configured with less than three MULT18X18.

Note: Each sysDSP Block spans nine columns of PFUs.

The sysDSP Block can be configured as:

- One 36x36 Multiplier
  - Basic multiplier, no add/sub/accum/sum blocks
- Four 18x18 Multipliers
  - Two add/sub/accum blocks
  - One summation block for adding four multipliers
- Eight 9x9 Multipliers
  - Four add/sub blocks
  - Two summation blocks

Note that a sysDSP block can only be configured in one mode at a time.

sysDSP Block Software

Overview

The sysDSP Block of the LatticeXP2 device can be targeted in a number of ways.

- The IPexpress™ tool in the ispLEVER software allows the rapid creation of modules implementing sysDSP elements. These modules can then be used in HDL designs as appropriate.
- The coding of certain functions into a design's HDL and allowing the synthesis tools to Inference the use of a sysDSP block.
- The implementation of designs in The MathWorks® Simulink® tool using a Lattice block set. The ispLEVER sysDSP portion of the ispLEVER design tools will then convert these blocks into HDL as appropriate.
- Instantiation of sysDSP primitives directly in the source code

Targeting sysDSP Block Using IPexpress

IPexpress allows you to graphically specify sysDSP elements. Once the element is specified, a HDL file is generated, which can be instantiated in a design. IPexpress allows users to configure all ports and set all available parameters. The following modules target the sysDSP Block. For design examples using IPexpress, refer to EXAMPLES in the ispLEVER Software (from the Project Navigator pull down-menu, go to **File** and open **Example**). The following four types of elements can be specified via IPexpress:

- MULT (Multiplier)
- MAC (Multiplier Accumulate)
- MULTADDSUB (Multiplier Add/Subtract)
- MULTADDSUBSUM (Multiply Add/Subtract and SUM)

MULT Module

The MULT Module configures elements to be packed into the sysDSP primitives. The Basic mode screen illustrated in Figure 13-2 consists of an optional one clock, one clock enable and one reset tied to all registers. Multiple sysDSP Blocks can be spanned to accommodate large multiplications. Additional LUTs may be required if multiple sysDSP blocks are needed. Select **Area/Speed** to determine the LUT implementation. The input data format can be selected as Parallel, Shift or Dynamic. The Shift format can only be enabled if inputs are less than 18 bits. The Shift format enables a sample/shift register, which is useful in applications such as the FIR filter. The Advanced mode screen, illustrated in Figure 13-3, allows finer control over the register. In the Advanced mode, users can control each register with independent clocks, clock enables and resets. MULT inputs can be from 2 to 72 bits.

Figure 13-2. MULT Mode Basic Set-up

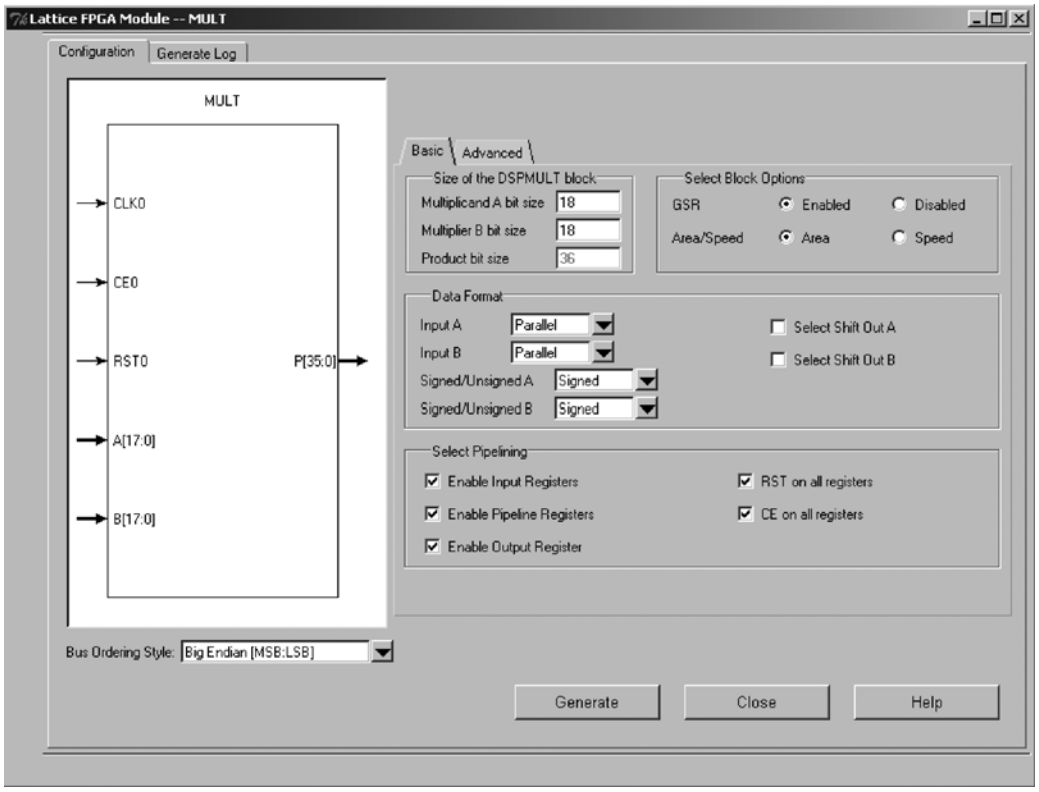
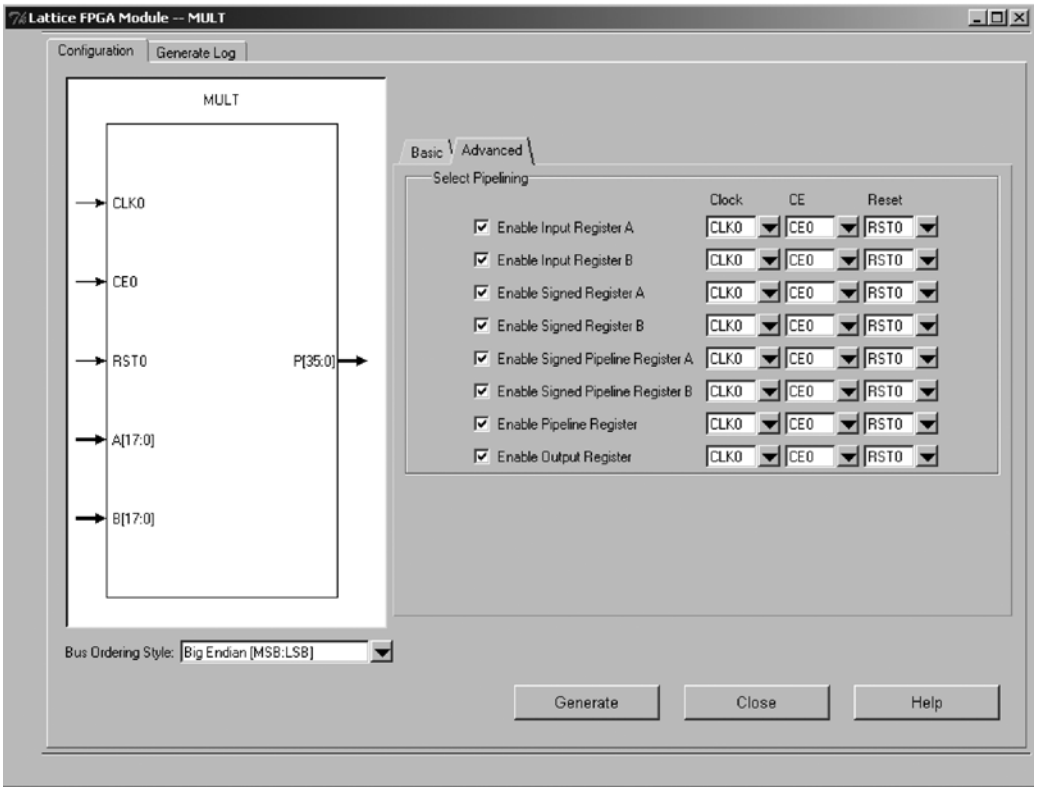


Figure 13-3. MULT Mode Advanced Set-up



MAC Module

The MAC Module configures multiply accumulate elements to be packed into the primitive MULT18X18MACB. The Basic mode, shown in Figure 13-4, consists of an optional one clock, one clock enable and one reset tied to all registers. Because of the accumulator, the output register is automatically enabled. Multiple sysDSP Blocks can be spanned to accommodate large multiplications. The accumulator of the sysDSP block is 52 bits deep and additional LUTs can be used if a larger accumulation is required. If sysDSP blocks are spanned, additional LUT logic may be required. Select **Area/Speed** to determine the LUT implementation. The input data format can be selected as Parallel, Shift or Dynamic. The Shift format can only be enabled if inputs are less than 18 bits. The Shift format enables a sample/shift register. The Accumsload loads the accumulator with the value from the LD port. This is required to initialize and load the first value of the accumulation. The Advanced mode, shown in Figure 13-5, allows finer control over the registers. In the advanced mode, users can control each register with independent clocks, clock enables and resets. MAC inputs can be from 2 to 72 bits.

Figure 13-4. MAC Mode Basic Set-up

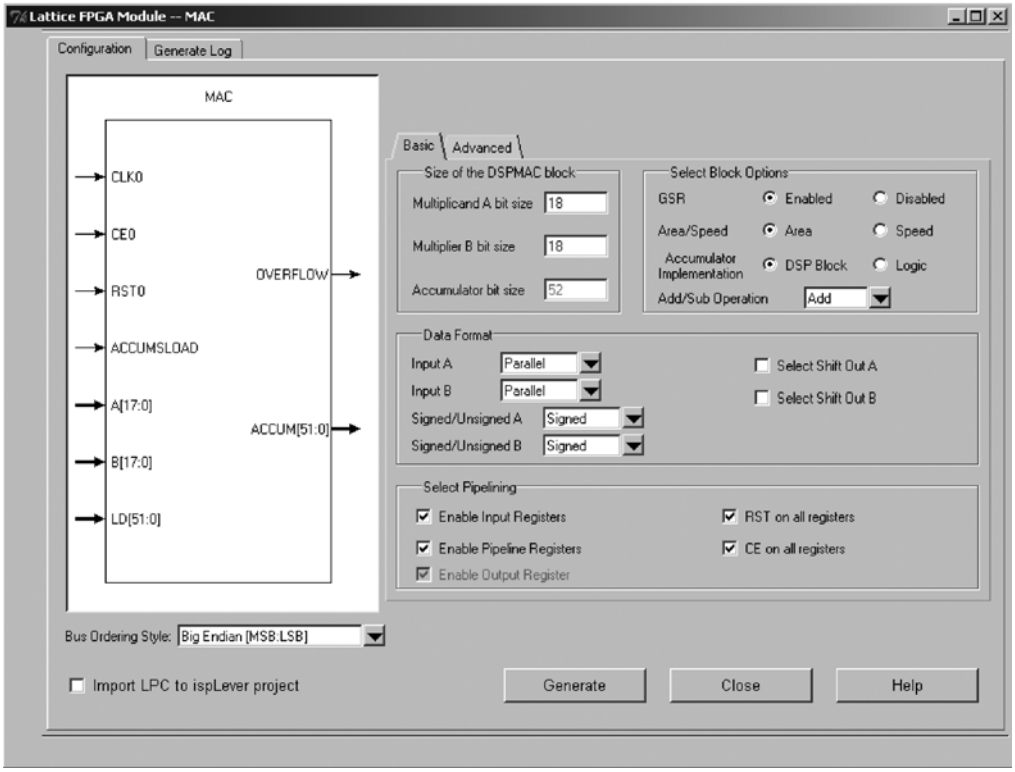
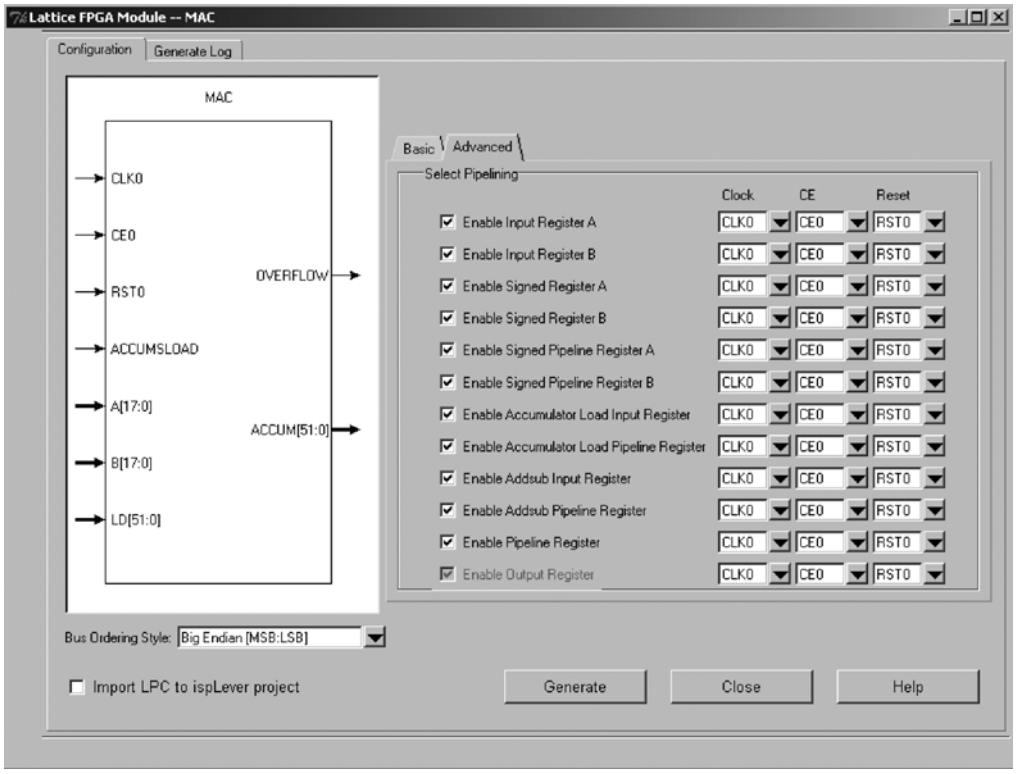


Figure 13-5. MAC Mode Advanced Set-up



MULTADDSUB Module

The MULTADDSUB GUI configures multiplier addition/subtraction elements to be packed into the primitives MULT18X18ADDSUBB or MULT9X9ADDSUBB. The Basic mode, shown in Figure 13-6, consists of an optional one clock, one clock enable and one reset tied to all registers. Multiple sysDSP Blocks can be spanned to accommodate large multiplications. If sysDSP blocks are spanned, additional LUT logic may be required. Select **Area/Speed** to determine the LUT implementation. The input data format can be selected as Parallel, Shift or Dynamic. The Shift format can only be enabled if inputs are less than 18 bits. The Shift format enables a sample/shift register, which is useful in applications such as the FIR filter. The Advanced mode, shown in Figure 13-7, provides finer control over the registers. In the advanced mode, users can control each register with independent clocks, clock enables and resets. MULTADDSUB inputs can be from 2 to 72 bits.

Figure 13-6. MULTADDSUB Mode Basic Set-up

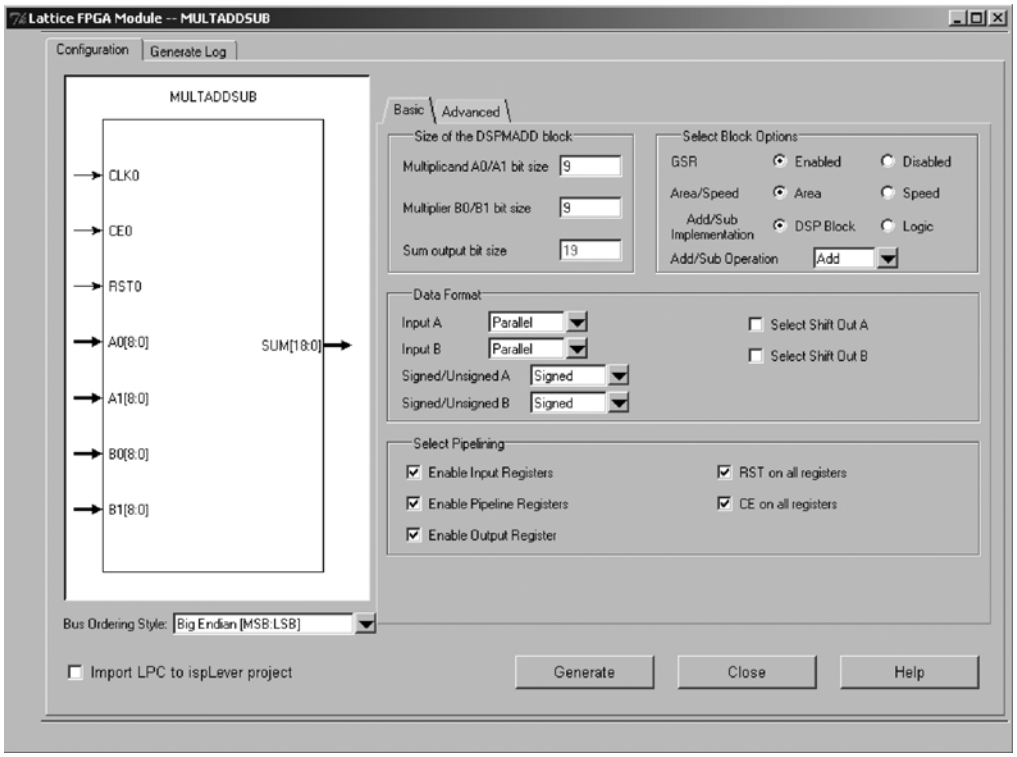
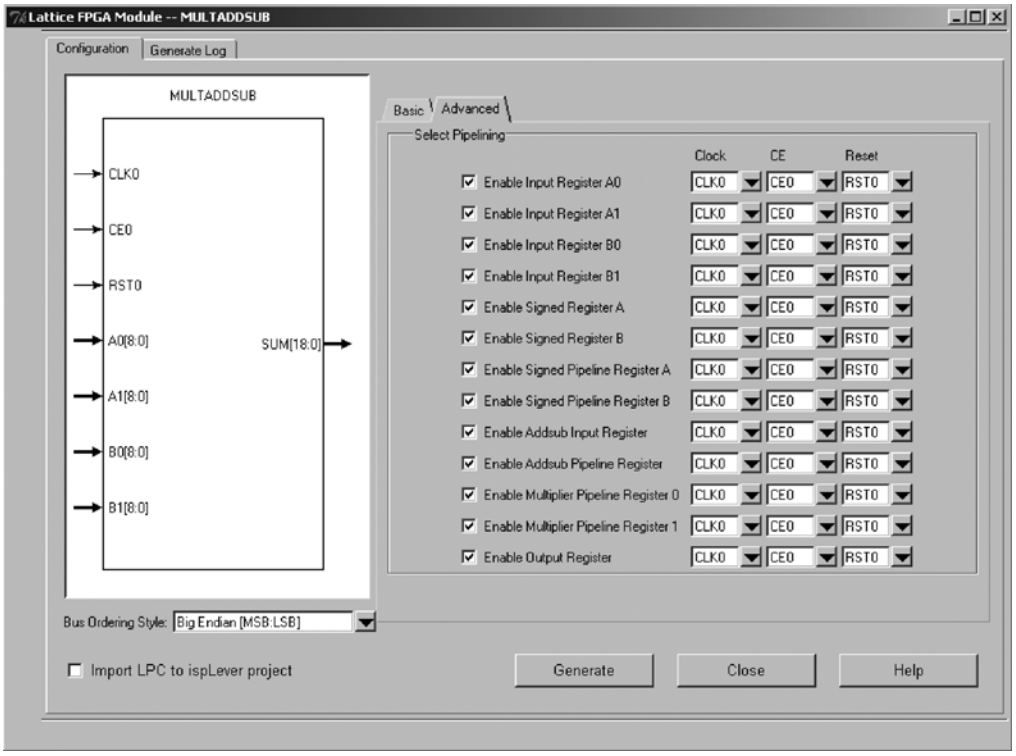


Figure 13-7. MULTADDSUB Mode Advanced Set-up



MULTADDSUBSUM Module

The MULTADDSUBSUM GUI configures Multiplier Addition/Subtraction Addition elements to be packed into the primitives MULT18X18ADDSUBSUMB or MULT9X9ADDSUBSUMB. The Basic mode, shown in Figure 13-8, consists of an optional one clock, one clock enable and one reset tied to all registers. Multiple sysDSP Blocks can be spanned to accommodate large multiplications. If sysDSP blocks are spanned, additional LUT logic may be required. Select **Area/Speed** to determine the LUT implementation. The input data format can be selected as Parallel, Shift or Dynamic. The Shift format is can only be enabled if inputs are less than 18 bits. The Shift format enables a sample/shift register, which is useful in applications such as the FIR filter. The Advanced mode, shown in Figure 13-9, provides finer control over the registers. In the advanced mode, users can control each register with independent clocks, clock enables and resets. MULTADDSUBSUM inputs can be from 2 to 72 bits.

Figure 13-8. MULTADDSUBSUM Mode Basic Set-up

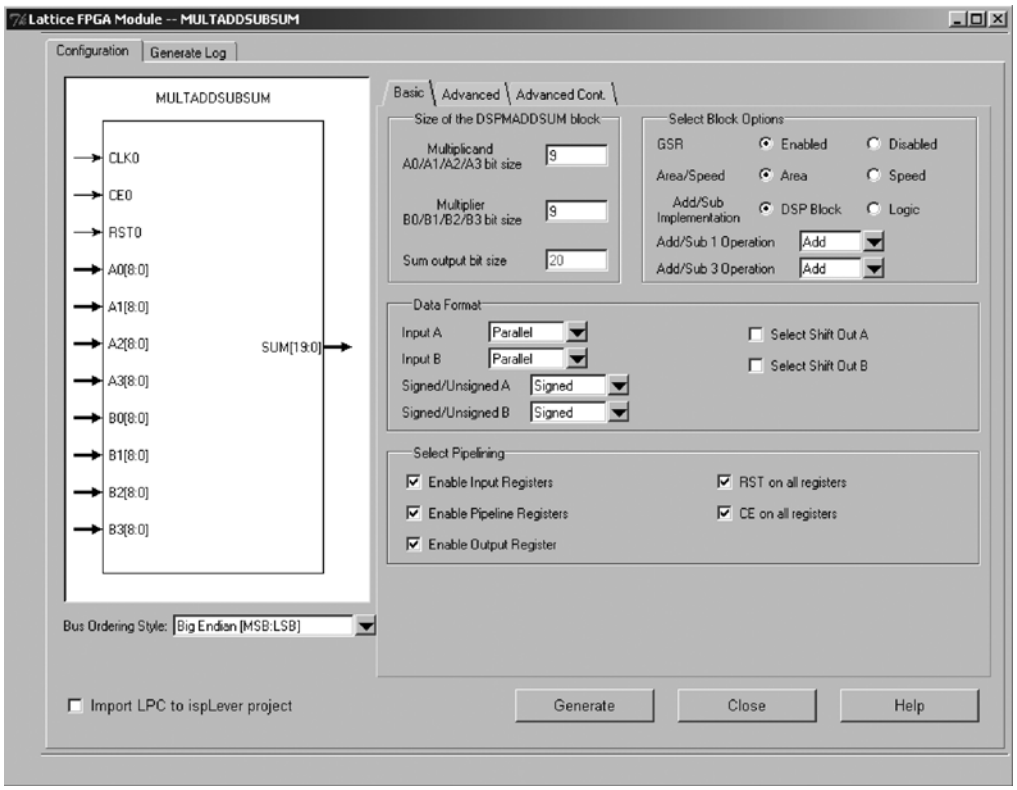


Figure 13-9. MULTADDSUBSUM Mode Advance

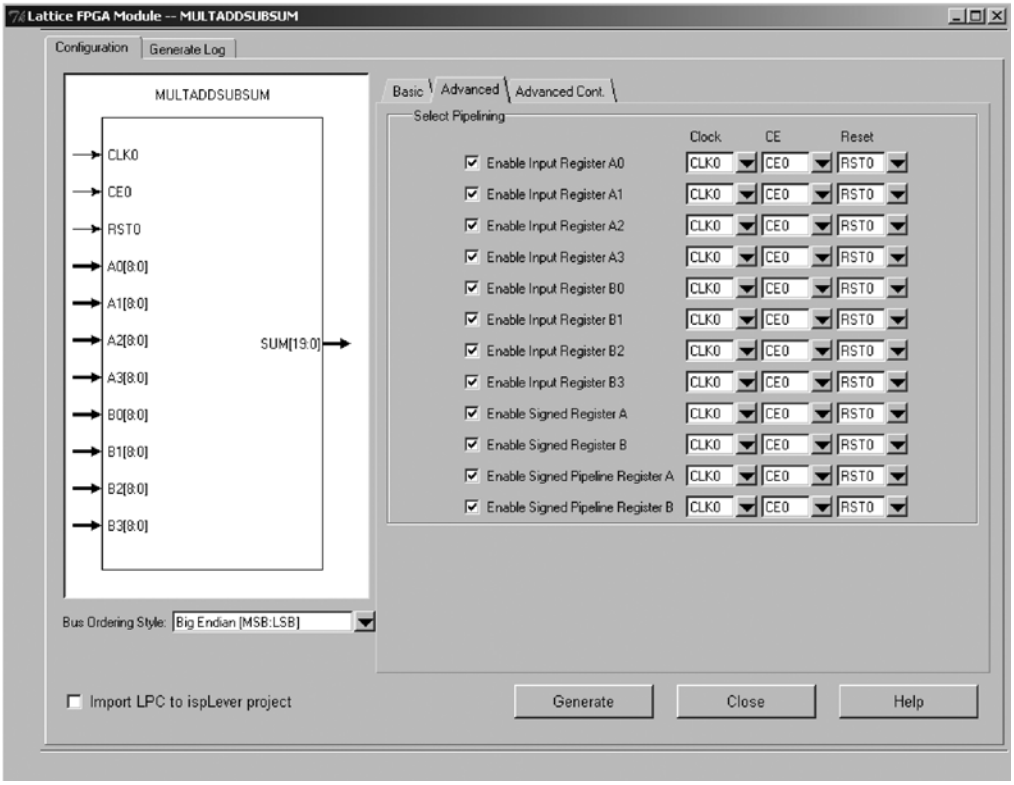
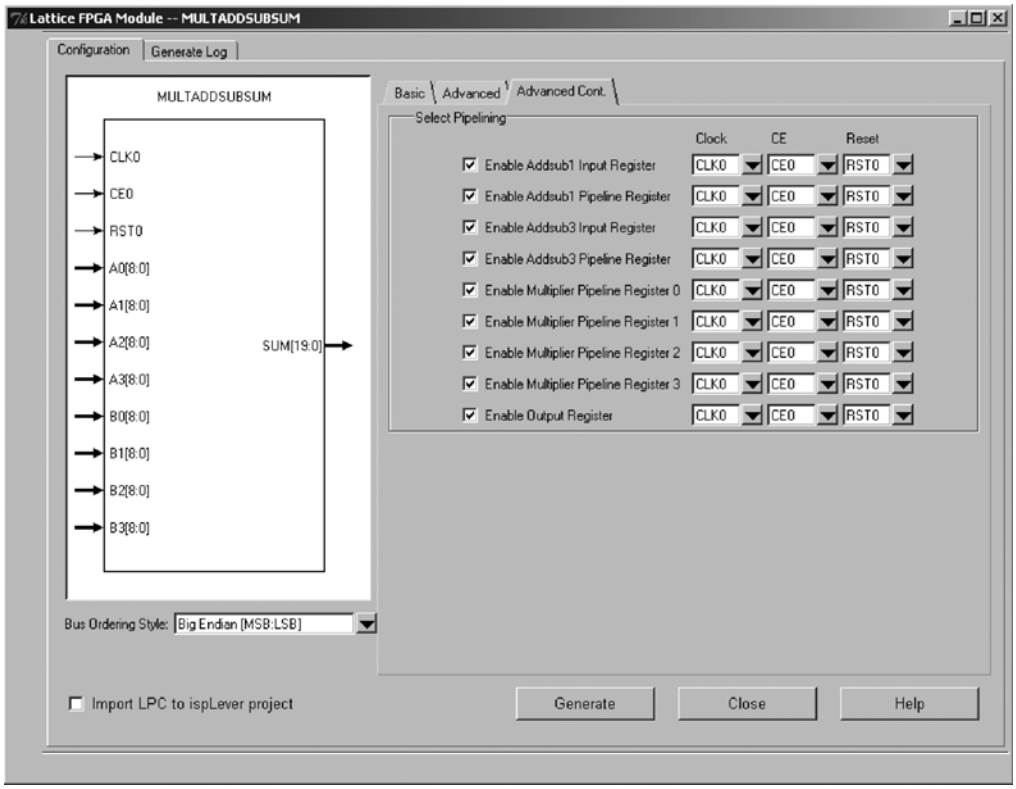


Figure 13-10. MULTADDSUBSUM Mode Advance



Targeting the sysDSP Block by Inference

The Inferencing flow enables the design tools to infer sysDSP Blocks from a HDL design. It is important to note that when using the Inferencing flow, unless the code style matches the sysDSP Block, results will not be optimal. Consider the following Verilog and VHDL examples:

```
// This Verilog example will be mapped into single MULT18X18MACB with the output register enabled
module mult_acc (dataout, dataax, dataay, clk);
   output [16:0] dataout;
   input [7:0] dataax, dataay;
   input clk;
   reg  [16:0] dataout;

   wire [15:0] multa = dataax * dataay; // 9x9 Multiplier
   wire [16:0] adder_out;
   assign adder_out = multa + dataout; // Accumulator
   always @(posedge clk)
   begin
     dataout <= adder_out; // Output Register of the Accumulator
   end
endmodule

-- This VHDL example will be mapped into single MULT18X18MACB with all the registers enabled
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity mac is
port (clk, reset : in std_logic;
      dataax, dataay : in std_logic_vector(8 downto 0);
      dataout : out std_logic_vector(17 downto 0));
end;

architecture arch of mac is
signal dataax_reg, dataay_reg : std_logic_vector(8 downto 0);
signal multout, multout_reg : std_logic_vector(17 downto 0);
signal addout : std_logic_vector(17 downto 0);
signal dataout_reg : std_logic_vector(17 downto 0);
begin

dataout <= dataout_reg;

process (clk, reset)
begin
if (reset = '1') then
  dataax_reg <= (others => '0');
  dataay_reg <= (others => '0');
elsif (clk'event and clk='1') then
  dataax_reg <= dataax;
  dataay_reg <= dataay;
end if;
end process;

multout <= dataax_reg * dataay_reg;

process (clk, reset)
begin
if (reset = '1') then
  multout_reg <= (others => '0');
elsif (clk'event and clk='1') then
  multout_reg <= multout;
end if;
```

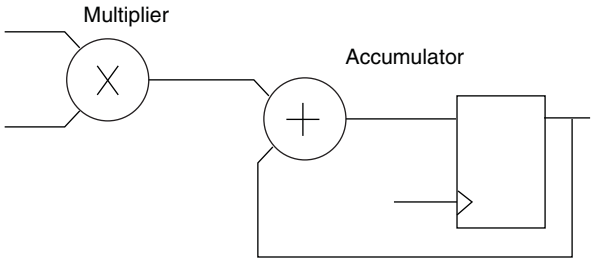
```
end process;

addout <= multout_reg + dataout_reg;

process (clk, reset)
begin
if (reset = '1') then
  dataout_reg <= (others => '0');
elsif (clk'event and clk='1') then
  dataout_reg <= addout;
end if;
end process;
end arch;
```

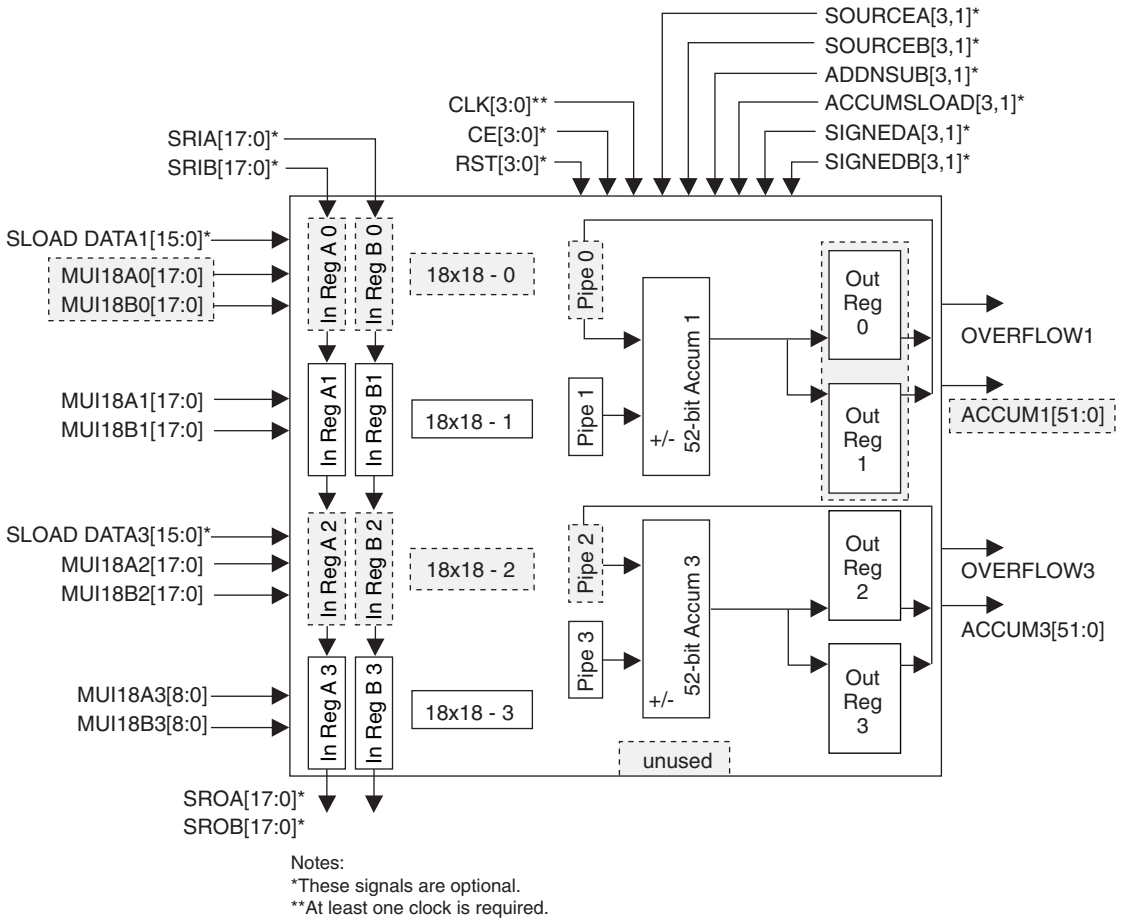
The above RTL will infer the following block diagram:

Figure 13-11. MULT18X18MACB Block Diagram



This block diagram can be mapped directly into the sysDSP primitives. Note that if a test point were added between the multiplier and the accumulator, or two output registers, etc. the code could not be mapped into a MULT18X18MACB of a sysDSP Block. Therefore, options that could be included in a design are input registers, pipeline registers, etc. For more Inferring design examples refer to EXAMPLES in the ispLEVER software.

Figure 13-12. MAC18X18MACB Packed into a sysDSP Block



Notes:
*These signals are optional.
**At least one clock is required.

sysDSP Blocks in the Report File

To check the configuration of the sysDSP Blocks in your design you can look at the MAP and Post PAR report files. The MAP report file shows the mapped sysDSP components/primitives in your design. The Post PAR report file shows the number of components in each sysDSP Block. The report files that follow show how the inferred MAC was used.

MAP Report File

```
. MULT18X18MACB  addout_17_0:

Multiplier
        Operation               Unsigned
        Operation Registers     CLK     CE      RST
        ------------------------------------------
                Input
                Pipeline
        Operation Registers     CLK     CE      RST
        ------------------------------------------
                Input
                Pipeline
```

```
AddSub
        Operation               Add
        Operation Registers     CLK     CE      RST
        ------------------------------------------
                Input
                Pipeline
Data
        Input Registers         CLK     CE      RST
        ------------------------------------------
                A               CLK0    CE0     RST0
                B               CLK0    CE0     RST0
        Pipeline Registers      CLK     CE      RST
        ------------------------------------------
                Pipe            CLK0    CE0     RST0
        Output Register         CLK     CE      RST
        ------------------------------------------
                Output          CLK0    CE0     RST0
Other
        GSR     ENABLED
   Number Of Mapped DSP Components:
   --------------------------------
   MULT36X36B          0
   MULT18X18B          0
   MULT18X18MACB       1
   MULT18X18ADDSUBB    0
   MULT18X18ADDSUBSUMB 0
   MULT9X9B            0
   MULT9X9ADDSUBB      0
   MULT9X9ADDSUBSUMB   0
   --------------------------------
```

Post PAR Report File

```
DSP Utilization Summary:
DSP Block #: 1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16   17   18
# of MULT36X36B
# of MULT18X18B
# of MULT18X18MACB                                        1
# of MULT18X18ADDSUBB
# of MULT18X18ADDSUBSUMB
# of MULT9X9B
# of MULT9X9ADDSUBB
# of MULT9X9ADDSUBSUMB
DSP Block  1    Component_Type         Instance_Name
DSP Block  2    Component_Type         Instance_Name
DSP Block  3    Component_Type         Instance_Name
DSP Block  4    Component_Type         Instance_Name
DSP Block  5    Component_Type         Instance_Name
DSP Block  6    Component_Type         Instance_Name
DSP Block  7    Component_Type         Instance_Name
DSP Block  8    Component_Type         Instance_Name
DSP Block  9    Component_Type         Instance_Name
  R45C81        MULT18X18MACB           addout_17_0
DSP Block 10    Component_Type         Instance_Name
DSP Block 11    Component_Type         Instance_Name
DSP Block 12    Component_Type         Instance_Name
DSP Block 13    Component_Type         Instance_Name
DSP Block 14    Component_Type         Instance_Name
DSP Block 15    Component_Type         Instance_Name
DSP Block 16    Component_Type         Instance_Name
DSP Block 17    Component_Type         Instance_Name
DSP Block 18    Component_Type         Instance_Name
```

Targeting the sysDSP Block Using Simulink

Simulink Overview

Simulink is a graphical add-on (similar to schematic entry) for Matlab®, which is produced by The MathWorks. For more information, refer to the Simulink web page at www.mathworks.com/products/simulink/.

Why is Simulink used?

- It allows users to create algorithms using floating point numbers.
- It helps users convert floating point algorithms into fixed point algorithms.

How does Simulink fit into the normal ispLEVER design flow?

- Once you have converted have your algorithm working in fixed point. You can use the Lattice ispDSP Block to create HDL files, which can be instantiated in your HDL design. Currently there is only support for VHDL.

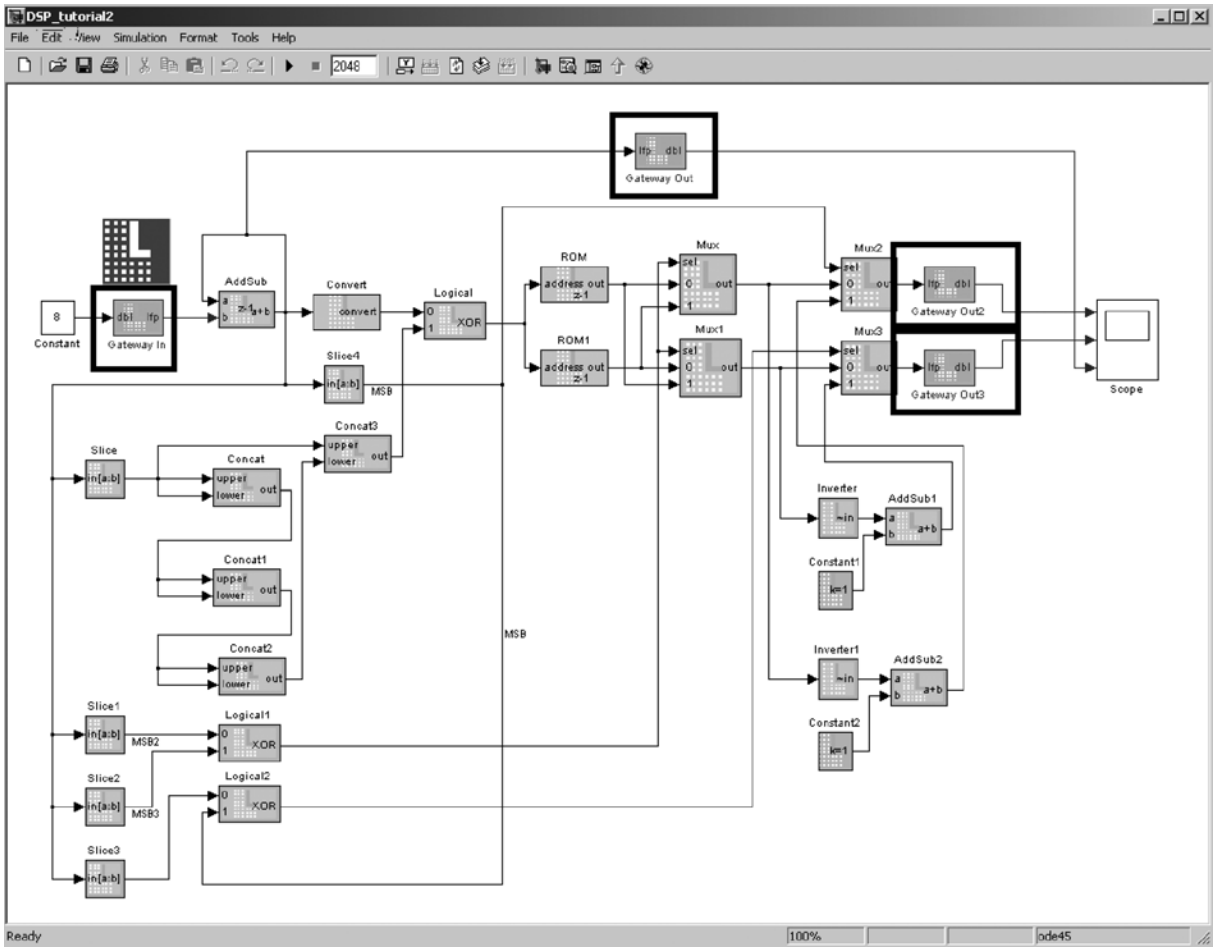
What does Lattice provide?

- Lattice provides a library of blocks for the Simulink tool, which include Multipliers, Adders, Registers, and other standard building blocks. Besides the basic building blocks there are a couple of unique Lattice blocks:

  Gateways In and Out
  Everything between Gateways In and Out represents the HDL code. Everything before a Gateway In is the stimulus your test bench. Everything after the Gateway Out are the signals you will be monitoring in the test bench. Below is an example. The box on the left contains Gateways In's and the three boxes on the right contain Gateway Outs in Figure 13-13.

  Generate
  The Generate block is used to convert Fixed point Simulink design into HDL files which can be instantiated in a HDL design. The Generate Block is identified by the Lattice logo and can be seen in Figure 13-13.

Figure 13-13. Simulink Design



Targeting the sysDSP Block by Instantiating Primitives

The sysDSP Block can be targeted by instantiating the sysDSP Block primitives into the design. The advantage of instantiating primitives is that it provides access to all ports and sets all available parameters. The disadvantage of this flow is that all this customization requires extra coding by the user. Appendix A details the syntax for the sysDSP Block primitives.

sysDSP Block Control Signal and Data Signal Descriptions

RST	Asynchronous reset of selected registers
SIGNEDA	Dynamic signal: 0 = unsigned, 1 = signed
SIGNEDB	Dynamic signal: 0 = unsigned, 1 = signed
ACCUMSLOAD	Dynamic signal: 0 = accumulate, 1 = load
ADDNSUB	Dynamic signal: 0 = subtract, 1 = add
SOURCEA	Dynamic signal: 0 = parallel input, 1 = shift input
SOURCEB	Dynamic signal: 0 = parallel input, 1 = shift input

Technical Support Assistance

Hotline: 1-800-LATTICE (North America)
+1-503-268-8001 (Outside North America)
e-mail: techsupport@latticesemi.com
Internet: www.latticesemi.com

Revision History

Date	Version	Change Summary
February 2007	01.0	Initial release.

Appendix A. DSP Block Primitives

MULT18X18B

```
input A17,A16,A15,A14,A13,A12,A11,A10,A9,A8,A7,A6,A5,A4,A3,A2,A1,A0;
input B17,B16,B15,B14,B13,B12,B11,B10,B9,B8,B7,B6,B5,B4,B3,B2,B1,B0;
input SIGNEDA, SIGNEDB, SOURCEA, SOURCEB;
input CE0,CE1,CE2,CE3,CLK0,CLK1,CLK2,CLK3,RST0,RST1,RST2,RST3;
input SRIA17,SRIA16,SRIA15,SRIA14,SRIA13,SRIA12,SRIA11,SRIA10,SRIA9;
input SRIA8,SRIA7,SRIA6,SRIA5,SRIA4,SRIA3,SRIA2,SRIA1,SRIA0;
input SRIB17,SRIB16,SRIB15,SRIB14,SRIB13,SRIB12,SRIB11,SRIB10,SRIB9;
input SRIB8,SRIB7,SRIB6,SRIB5,SRIB4,SRIB3,SRIB2,SRIB1,SRIB0;
output SROA17,SROA16,SROA15,SROA14,SROA13,SROA12,SROA11,SROA10,SROA9;
output SROA8,SROA7,SROA6,SROA5,SROA4,SROA3,SROA2,SROA1,SROA0;
output SROB17,SROB16,SROB15,SROB14,SROB13,SROB12,SROB11,SROB10,SROB9;
output SROB8,SROB7,SROB6,SROB5,SROB4,SROB3,SROB2,SROB1,SROB0;
output P35,P34,P33,P32,P31,P30,P29,P28,P27,P26,P25,P24,P23,P22,P21,P20,P19,P18;
output P17,P16,P15,P14,P13,P12,P11,P10,P9,P8,P7,P6,P5,P4,P3,P2,P1,P0;
parameter REG_INPUTA_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_INPUTA_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_INPUTA_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_INPUTB_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_INPUTB_CE = = " CE0, CE1, CE2, CE3 ";
parameter REG_INPUTB_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_PIPELINE_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_PIPELINE_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_PIPELINE_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_OUTPUT_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_OUTPUT_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_OUTPUT_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_SIGNEDA_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_SIGNEDA_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_SIGNEDA_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_SIGNEDB_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_SIGNEDB_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_SIGNEDB_RST = " RST0, RST1, RST2, RST3 ";
parameter GSR = " Enabled, Disabled ";
```

MULT18X18ADDSUBB

```
input A017,A016,A015,A014,A013,A012,A011,A010,A09;
input A08,A07,A06,A05,A04,A03,A02,A01,A00;
input A117,A116,A115,A114,A113,A112,A111,A110,A19;
input A18,A17,A16,A15,A14,A13,A12,A11,A10;
input B017,B016,B015,B014,B013,B012,B011,B010,B09;
input B08,B07,B06,B05,B04,B03,B02,B01,B00;
input B117,B116,B115,B114,B113,B112,B111,B110,B19;
input B18,B17,B16,B15,B14,B13,B12,B11,B10;
input SIGNEDA, SIGNEDB, SOURCEA0, SOURCEA1, SOURCEB0, SOURCEB1, ADDNSUB;
input CE0,CE1,CE2,CE3,CLK0,CLK1,CLK2,CLK3,RST0,RST1,RST2,RST3;
input SRIA17,SRIA16,SRIA15,SRIA14,SRIA13,SRIA12,SRIA11,SRIA10,SRIA9;
input SRIA8,SRIA7,SRIA6,SRIA5,SRIA4,SRIA3,SRIA2,SRIA1,SRIA0;
input SRIB17,SRIB16,SRIB15,SRIB14,SRIB13,SRIB12,SRIB11,SRIB10,SRIB9;
input SRIB8,SRIB7,SRIB6,SRIB5,SRIB4,SRIB3,SRIB2,SRIB1,SRIB0;
output SROA17,SROA16,SROA15,SROA14,SROA13,SROA12,SROA11,SROA10,SROA9;
output SROA8,SROA7,SROA6,SROA5,SROA4,SROA3,SROA2,SROA1,SROA0;
output SROB17,SROB16,SROB15,SROB14,SROB13,SROB12,SROB11,SROB10,SROB9;
output SROB8,SROB7,SROB6,SROB5,SROB4,SROB3,SROB2,SROB1,SROB0;
output
SUM36,SUM35,SUM34,SUM33,SUM32,SUM31,SUM30,SUM29,SUM28,SUM27,SUM26,SUM25,SUM24,SUM23,SUM22,SUM21,SU
M20,SUM19,SUM18,SUM17,SUM16,SUM15,SUM14,SUM13,SUM12,SUM11,SUM10,SUM9,SUM8,SUM7,SUM6,SUM5,SUM4,SUM3
,SUM2,SUM1,SUM0;
```

```
parameter REG_INPUTA0_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_INPUTA0_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_INPUTA0_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_INPUTA1_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_INPUTA1_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_INPUTA1_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_INPUTB0_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_INPUTB0_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_INPUTB0_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_INPUTB1_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_INPUTB1_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_INPUTB1_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_PIPELINE0_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_PIPELINE0_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_PIPELINE0_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_PIPELINE1_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_PIPELINE1_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_PIPELINE1_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_OUTPUT_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_OUTPUT_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_OUTPUT_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_SIGNEDA_0_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_SIGNEDA_0_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_SIGNEDA_0_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_SIGNEDA_1_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_SIGNEDA_1_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_SIGNEDA_1_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_SIGNEDB_0_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_SIGNEDB_0_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_SIGNEDB_0_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_SIGNEDB_1_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_SIGNEDB_1_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_SIGNEDB_1_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_ADDNSUB_0_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_ADDNSUB_0_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_ADDNSUB_0_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_ADDNSUB_1_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_ADDNSUB_1_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_ADDNSUB_1_RST = " RST0, RST1, RST2, RST3 ";
parameter GSR = " Enabled, Disabled ";
```

MULT18X18ADDSUBSUMB

```
input A017,A016,A015,A014,A013,A012,A011,A010,A09;
input A08,A07,A06,A05,A04,A03,A02,A01,A00;
input A117,A116,A115,A114,A113,A112,A111,A110,A19;
input A18,A17,A16,A15,A14,A13,A12,A11,A10;
input A217,A216,A215,A214,A213,A212,A211,A210,A29;
input A28,A27,A26,A25,A24,A23,A22,A21,A20;
input A317,A316,A315,A314,A313,A312,A311,A310,A39;
input A38,A37,A36,A35,A34,A33,A32,A31,A30;
input B017,B016,B015,B014,B013,B012,B011,B010,B09;
input B08,B07,B06,B05,B04,B03,B02,B01,B00;
input B117,B116,B115,B114,B113,B112,B111,B110,B19;
input B18,B17,B16,B15,B14,B13,B12,B11,B10;
input B217,B216,B215,B214,B213,B212,B211,B210,B29;
input B28,B27,B26,B25,B24,B23,B22,B21,B20;
input B317,B316,B315,B314,B313,B312,B311,B310,B39;
input B38,B37,B36,B35,B34,B33,B32,B31,B30;
input SIGNEDA, SIGNEDB,ADDNSUB1,ADDNSUB3;
input SOURCEA0, SOURCEA1, SOURCEA2, SOURCEA3;
input SOURCEB0, SOURCEB1, SOURCEB2, SOURCEB3;
```

```
input CE0,CE1,CE2,CE3,CLK0,CLK1,CLK2,CLK3,RST0,RST1,RST2,RST3;
input SRIA17,SRIA16,SRIA15,SRIA14,SRIA13,SRIA12,SRIA11,SRIA10,SRIA9;
input SRIA8,SRIA7,SRIA6,SRIA5,SRIA4,SRIA3,SRIA2,SRIA1,SRIA0;
input SRIB17,SRIB16,SRIB15,SRIB14,SRIB13,SRIB12,SRIB11,SRIB10,SRIB9;
input SRIB8,SRIB7,SRIB6,SRIB5,SRIB4,SRIB3,SRIB2,SRIB1,SRIB0;
output SROA17,SROA16,SROA15,SROA14,SROA13,SROA12,SROA11,SROA10,SROA9;
output SROA8,SROA7,SROA6,SROA5,SROA4,SROA3,SROA2,SROA1,SROA0;
output SROB17,SROB16,SROB15,SROB14,SROB13,SROB12,SROB11,SROB10,SROB9;
output SROB8,SROB7,SROB6,SROB5,SROB4,SROB3,SROB2,SROB1,SROB0;
output
SUM37,SUM36,SUM35,SUM34,SUM33,SUM32,SUM31,SUM30,SUM29,SUM28,SUM27,SUM26,SUM25,SUM24,SUM23,SUM22,SU
M21,SUM20,SUM19,SUM18,SUM17,SUM16,SUM15,SUM14,SUM13,SUM12,SUM11,SUM10,SUM9,SUM8,SUM7,SUM6,SUM5,SUM
4,SUM3,SUM2,SUM1,SUM0;
parameter REG_INPUTA0_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_INPUTA0_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_INPUTA0_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_INPUTA1_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_INPUTA1_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_INPUTA1_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_INPUTA2_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_INPUTA2_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_INPUTA2_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_INPUTA3_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_INPUTA3_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_INPUTA3_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_INPUTB0_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_INPUTB0_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_INPUTB0_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_INPUTB1_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_INPUTB1_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_INPUTB1_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_INPUTB2_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_INPUTB2_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_INPUTB2_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_INPUTB3_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_INPUTB3_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_INPUTB3_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_PIPELINE0_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_PIPELINE0_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_PIPELINE0_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_PIPELINE1_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_PIPELINE1_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_PIPELINE1_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_PIPELINE2_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_PIPELINE2_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_PIPELINE2_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_PIPELINE3_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_PIPELINE3_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_PIPELINE3_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_OUTPUT_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_OUTPUT_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_OUTPUT_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_SIGNEDA_0_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_SIGNEDA_0_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_SIGNEDA_0_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_SIGNEDA_1_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_SIGNEDA_1_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_SIGNEDA_1_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_SIGNEDB_0_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_SIGNEDB_0_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_SIGNEDB_0_RST = " RST0, RST1, RST2, RST3 ";
```

```
parameter REG_SIGNEDB_1_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_SIGNEDB_1_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_SIGNEDB_1_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_ADDNSUB1_0_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_ADDNSUB1_0_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_ADDNSUB1_0_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_ADDNSUB1_1_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_ADDNSUB1_1_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_ADDNSUB1_1_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_ADDNSUB3_0_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_ADDNSUB3_0_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_ADDNSUB3_0_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_ADDNSUB3_1_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_ADDNSUB3_1_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_ADDNSUB3_1_RST = " RST0, RST1, RST2, RST3 ";
parameter GSR = " Enabled, Disabled ";
```

MULT18X18MACB

```
input A17,A16,A15,A14,A13,A12,A11,A10,A9;
input A8,A7,A6,A5,A4,A3,A2,A1,A0;
input B17,B16,B15,B14,B13,B12,B11,B10,B9;
input B8,B7,B6,B5,B4,B3,B2,B1,B0;
input ADDNSUB, SIGNEDA, SIGNEDB,ACCUMSLOAD;
input SOURCEA, SOURCEB;
input CE0,CE1,CE2,CE3,CLK0,CLK1,CLK2,CLK3,RST0,RST1,RST2,RST3;
input LD51, LD50, LD49, LD48, LD47, LD46, LD45, LD44, LD43, LD42, LD41, LD40
input LD39, LD38, LD37, LD36, LD35, LD34, LD33, LD32, LD31, LD30
input LD29, LD28, LD27, LD26, LD25, LD24, LD23, LD22, LD21, LD20
input LD19, LD18, LD17, LD16, LD15, LD14, LD13, LD12, LD11, LD10
input LD9, LD8, LD7, LD6, LD5, LD4, LD3, LD2, LD1, LD0;
input SRIA17,SRIA16,SRIA15,SRIA14,SRIA13,SRIA12,SRIA11,SRIA10,SRIA9;
input SRIA8,SRIA7,SRIA6,SRIA5,SRIA4,SRIA3,SRIA2,SRIA1,SRIA0;
input SRIB17,SRIB16,SRIB15,SRIB14,SRIB13,SRIB12,SRIB11,SRIB10,SRIB9;
input SRIB8,SRIB7,SRIB6,SRIB5,SRIB4,SRIB3,SRIB2,SRIB1,SRIB0;
output SROA17,SROA16,SROA15,SROA14,SROA13,SROA12,SROA11,SROA10,SROA9;
output SROA8,SROA7,SROA6,SROA5,SROA4,SROA3,SROA2,SROA1,SROA0;
output SROB17,SROB16,SROB15,SROB14,SROB13,SROB12,SROB11,SROB10,SROB9;
output SROB8,SROB7,SROB6,SROB5,SROB4,SROB3,SROB2,SROB1,SROB0;
output
ACCUM51,ACCUM50,ACCUM49,ACCUM48,ACCUM47,ACCUM46,ACCUM45,ACCUM44,ACCUM43,ACCUM42,ACCUM41,ACCUM40,AC
CUM39,ACCUM38,ACCUM37,ACCUM36,ACCUM35,ACCUM34,ACCUM33,ACCUM32,ACCUM31,ACCUM30,ACCUM29,ACCUM28,ACCU
M27,ACCUM26,ACCUM25,ACCUM24,ACCUM23,ACCUM22,ACCUM21,ACCUM20,ACCUM19,ACCUM18,ACCUM17,ACCUM16,ACCUM1
5,ACCUM14,ACCUM13,ACCUM12,ACCUM11,ACCUM10,ACCUM9,ACCUM8,ACCUM7,ACCUM6,ACCUM5,ACCUM4,ACCUM3,ACCUM2,
ACCUM1,ACCUM0,OVERFLOW;
parameter REG_INPUTA_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_INPUTA_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_INPUTA_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_INPUTB_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_INPUTB_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_INPUTB_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_PIPELINE_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_PIPELINE_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_PIPELINE_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_OUTPUT_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_OUTPUT_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_OUTPUT_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_SIGNEDA_0_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_SIGNEDA_0_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_SIGNEDA_0_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_SIGNEDA_1_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_SIGNEDA_1_CE = " CE0, CE1, CE2, CE3 ";
```

```
parameter REG_SIGNEDA_1_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_SIGNEDB_0_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_SIGNEDB_0_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_SIGNEDB_0_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_SIGNEDB_1_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_SIGNEDB_1_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_SIGNEDB_1_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_ACCUMSLOAD_0_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_ACCUMSLOAD_0_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_ACCUMSLOAD_0_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_ACCUMSLOAD_1_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_ACCUMSLOAD_1_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_ACCUMSLOAD_1_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_ADDNSUB_0_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_ADDNSUB_0_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_ADDNSUB_0_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_ADDNSUB_1_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_ADDNSUB_1_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_ADDNSUB_1_RST = " RST0, RST1, RST2, RST3 ";
parameter GSR = " Enabled, Disabled ";
```

MULT36X36B

```
input A35,A34,A33,A32,A31,A30,A29,A28,A27,A26,A25,A24,A23,A22,A21,A20,A19,A18;
input A17,A16,A15,A14,A13,A12,A11,A10,A9,A8,A7,A6,A5,A4,A3,A2,A1,A0;
input B35,B34,B33,B32,B31,B30,B29,B28,B27,B26,B25,B24,B23,B22,B21,B20,B19,B18;
input B17,B16,B15,B14,B13,B12,B11,B10,B9,B8,B7,B6,B5,B4,B3,B2,B1,B0;
input SIGNEDA, SIGNEDB;
input CE0,CE1,CE2,CE3,CLK0,CLK1,CLK2,CLK3,RST0,RST1,RST2,RST3;
output P71,P70,P69,P68,P67,P66,P65,P64,P63,P62,P61,P60,P59,P58,P57,P56,P55,P54;
output P53,P52,P51,P50,P49,P48,P47,P46,P45,P44,P43,P42,P41,P40,P39,P38,P37,P36;
output P35,P34,P33,P32,P31,P30,P29,P28,P27,P26,P25,P24,P23,P22,P21,P20,P19,P18;
output P17,P16,P15,P14,P13,P12,P11,P10,P9,P8,P7,P6,P5,P4,P3,P2,P1,P0;
parameter REG_INPUTA_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_INPUTA_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_INPUTA_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_INPUTB_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_INPUTB_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_INPUTB_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_PIPELINE_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_PIPELINE_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_PIPELINE_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_OUTPUT_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_OUTPUT_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_OUTPUT_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_SIGNEDA_0_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_SIGNEDA_0_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_SIGNEDA_0_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_SIGNEDA_1_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_SIGNEDA_1_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_SIGNEDA_1_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_SIGNEDB_0_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_SIGNEDB_0_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_SIGNEDB_0_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_SIGNEDB_1_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_SIGNEDB_1_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_SIGNEDB_1_RST = " RST0, RST1, RST2, RST3 ";
parameter GSR = " Enabled, Disabled ";
```

MULT9X9B

```
input A8,A7,A6,A5,A4,A3,A2,A1,A0;
input B8,B7,B6,B5,B4,B3,B2,B1,B0;
input SIGNEDA, SIGNEDB, SOURCEA, SOURCEB;
input CE0,CE1,CE2,CE3,CLK0,CLK1,CLK2,CLK3,RST0,RST1,RST2,RST3;
input SRIA8,SRIA7,SRIA6,SRIA5,SRIA4,SRIA3,SRIA2,SRIA1,SRIA0;
input SRIB8,SRIB7,SRIB6,SRIB5,SRIB4,SRIB3,SRIB2,SRIB1,SRIB0;
output SROA8,SROA7,SROA6,SROA5,SROA4,SROA3,SROA2,SROA1,SROA0;
output SROB8,SROB7,SROB6,SROB5,SROB4,SROB3,SROB2,SROB1,SROB0;
output P17,P16,P15,P14,P13,P12,P11,P10,P9,P8,P7,P6,P5,P4,P3,P2,P1,P0;
parameter REG_INPUTA_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_INPUTA_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_INPUTA_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_INPUTB_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_INPUTB_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_INPUTB_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_PIPELINE_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_PIPELINE_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_PIPELINE_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_OUTPUT_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_OUTPUT_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_OUTPUT_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_SIGNEDA_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_SIGNEDA_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_SIGNEDA_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_SIGNEDB_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_SIGNEDB_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_SIGNEDB_RST = " RST0, RST1, RST2, RST3 ";
parameter GSR = " Enabled, Disabled ";
```

MULT9X9ADDSUBB

```
input A08,A07,A06,A05,A04,A03,A02,A01,A00;
input A18,A17,A16,A15,A14,A13,A12,A11,A10;
input B08,B07,B06,B05,B04,B03,B02,B01,B00;
input B18,B17,B16,B15,B14,B13,B12,B11,B10;
input SIGNEDA, SIGNEDB,ADDNSUB;
input SOURCEA0, SOURCEA1, SOURCEB0, SOURCEB1;
input CE0,CE1,CE2,CE3,CLK0,CLK1,CLK2,CLK3,RST0,RST1,RST2,RST3;
input SRIA8,SRIA7,SRIA6,SRIA5,SRIA4,SRIA3,SRIA2,SRIA1,SRIA0;
input SRIB8,SRIB7,SRIB6,SRIB5,SRIB4,SRIB3,SRIB2,SRIB1,SRIB0;
output SROA8,SROA7,SROA6,SROA5,SROA4,SROA3,SROA2,SROA1,SROA0;
output SROB8,SROB7,SROB6,SROB5,SROB4,SROB3,SROB2,SROB1,SROB0;
output
SUM18,SUM17,SUM16,SUM15,SUM14,SUM13,SUM12,SUM11,SUM10,SUM9,SUM8,SUM7,SUM6,SUM5,SUM4,SUM3,SUM2,SUM1
,SUM0;
parameter REG_INPUTA0_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_INPUTA0_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_INPUTA0_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_INPUTA1_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_INPUTA1_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_INPUTA1_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_INPUTB0_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_INPUTB0_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_INPUTB0_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_INPUTB1_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_INPUTB1_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_INPUTB1_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_PIPELINE0_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_PIPELINE0_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_PIPELINE0_RST = " RST0, RST1, RST2, RST3 ";
```

```
parameter REG_PIPELINE1_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_PIPELINE1_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_PIPELINE1_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_OUTPUT_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_OUTPUT_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_OUTPUT_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_SIGNEDA_0_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_SIGNEDA_0_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_SIGNEDA_0_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_SIGNEDA_1_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_SIGNEDA_1_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_SIGNEDA_1_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_SIGNEDB_0_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_SIGNEDB_0_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_SIGNEDB_0_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_SIGNEDB_1_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_SIGNEDB_1_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_SIGNEDB_1_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_ADDNSUB_0_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_ADDNSUB_0_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_ADDNSUB_0_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_ADDNSUB_1_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_ADDNSUB_1_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_ADDNSUB_1_RST = " RST0, RST1, RST2, RST3 ";
parameter GSR = " Enabled, Disabled ";
```

MULT9X9ADDSUBSUMB

```
input A08,A07,A06,A05,A04,A03,A02,A01,A00;
input A18,A17,A16,A15,A14,A13,A12,A11,A10;
input A28,A27,A26,A25,A24,A23,A22,A21,A20;
input A38,A37,A36,A35,A34,A33,A32,A31,A30;
input B08,B07,B06,B05,B04,B03,B02,B01,B00;
input B18,B17,B16,B15,B14,B13,B12,B11,B10;
input B28,B27,B26,B25,B24,B23,B22,B21,B20;
input B38,B37,B36,B35,B34,B33,B32,B31,B30;
input SIGNEDA, SIGNEDB,ADDNSUB1,ADDNSUB3;
input SOURCEA0, SOURCEA1, SOURCEA2, SOURCEA3;
input SOURCEB0, SOURCEB1, SOURCEB2, SOURCEB3;
input CE0,CE1,CE2,CE3,CLK0,CLK1,CLK2,CLK3,RST0,RST1,RST2,RST3;
input SRIA8,SRIA7,SRIA6,SRIA5,SRIA4,SRIA3,SRIA2,SRIA1,SRIA0;
input SRIB8,SRIB7,SRIB6,SRIB5,SRIB4,SRIB3,SRIB2,SRIB1,SRIB0;
output SROA8,SROA7,SROA6,SROA5,SROA4,SROA3,SROA2,SROA1,SROA0;
output SROB8,SROB7,SROB6,SROB5,SROB4,SROB3,SROB2,SROB1,SROB0;
output
SUM19,SUM18,SUM17,SUM16,SUM15,SUM14,SUM13,SUM12,SUM11,SUM10,SUM9,SUM8,SUM7,SUM6,SUM5,SUM4,SUM3,SUM
2,SUM1,SUM0;
parameter REG_INPUTA0_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_INPUTA0_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_INPUTA0_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_INPUTA1_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_INPUTA1_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_INPUTA1_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_INPUTA2_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_INPUTA2_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_INPUTA2_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_INPUTA3_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_INPUTA3_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_INPUTA3_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_INPUTB0_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_INPUTB0_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_INPUTB0_RST = " RST0, RST1, RST2, RST3 ";
```

```
parameter REG_INPUTB1_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_INPUTB1_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_INPUTB1_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_INPUTB2_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_INPUTB2_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_INPUTB2_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_INPUTB3_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_INPUTB3_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_INPUTB3_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_PIPELINE0_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_PIPELINE0_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_PIPELINE0_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_PIPELINE1_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_PIPELINE1_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_PIPELINE1_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_PIPELINE2_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_PIPELINE2_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_PIPELINE2_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_PIPELINE3_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_PIPELINE3_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_PIPELINE3_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_OUTPUT_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_OUTPUT_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_OUTPUT_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_SIGNEDA_0_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_SIGNEDA_0_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_SIGNEDA_0_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_SIGNEDA_1_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_SIGNEDA_1_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_SIGNEDA_1_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_SIGNEDB_0_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_SIGNEDB_0_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_SIGNEDB_0_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_SIGNEDB_1_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_SIGNEDB_1_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_SIGNEDB_1_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_ADDNSUB1_0_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_ADDNSUB1_0_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_ADDNSUB1_0_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_ADDNSUB1_1_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_ADDNSUB1_1_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_ADDNSUB1_1_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_ADDNSUB3_0_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_ADDNSUB3_0_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_ADDNSUB3_0_RST = " RST0, RST1, RST2, RST3 ";
parameter REG_ADDNSUB3_1_CLK = " NONE, CLK0, CLK1, CLK2, CLK3 ";
parameter REG_ADDNSUB3_1_CE = " CE0, CE1, CE2, CE3 ";
parameter REG_ADDNSUB3_1_RST = " RST0, RST1, RST2, RST3 ";
parameter GSR = " Enabled, Disabled ";
```

LatticeXP2 sysCONFIG Usage Guide

January 2008 Technical Note TN1141

Introduction

The memory in the LatticeXP2™ FPGAs is built using Flash cells, along with SRAM cells, so that configuration memory can be loaded automatically at power-up, or at any time the user wishes to update the device. In addition to "instant-on" capability, on-chip Flash memory greatly increases design security by getting rid of the external configuration bitstream; while maintaining the ease of use and reprogrammability of an SRAM-based FPGA.

The LatticeXP2 supports the use of an encryption key to protect the contents of the Flash memory for additional security. The LatticeXP2 also supports the use of a One-Time-Programmable (OTP) feature to protect the Flash memory from being erased or re-programmed.

While an external device is not required, the LatticeXP2 does support several external configuration modes. The available external configuration modes are:

- Slave SPI
- Master SPI
- ispJTAG™ (1149.1 interface)

This guide will cover all the configuration options available for the LatticeXP2.

Programming Overview

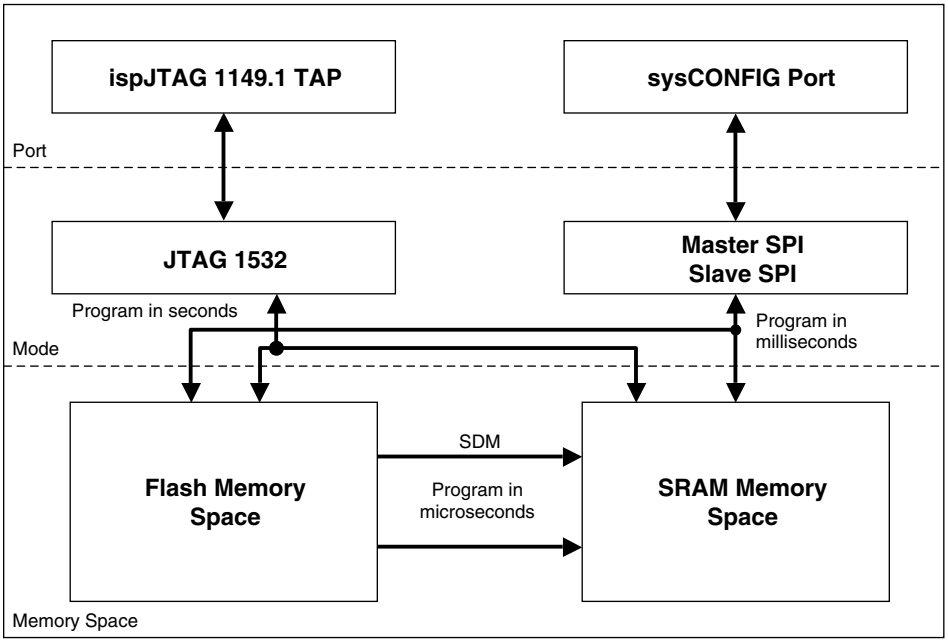
The LatticeXP2 contains two types of memory, SRAM and Flash (refer to Figure 14-1). SRAM contains the FPGA configuration, essentially the "fuses" that define the circuit connections; Flash provides an internal storage space for the configuration data.

The LatticeXP2 also contains additional Flash memory area and that is designated for Tag memory and User Flash memory. The Tag memory is a scratch pad memory that is available to the user for storage of mission critical data, board serialization, revision information, programmed pattern identification, or other information. The User Flash memory is available to provide a back up the contents of the EBR RAM modules if the user desires. These functions will be discussed in more detail in later sections of this document.

The SRAM can be configured using JTAG, the external Master SPI port, or by using the data stored in on-chip Flash. The configuration process consists of SRAM initialization (clear the RAM and the address pointers), loading the SRAM with the configuration data, and setting the FPGA into user mode (waking up the FPGA).

On-chip Flash can be programmed by using JTAG or by using the external Slave SPI port. JTAG Flash programming can be performed any time the device is powered up. The Slave SPI port uses the sysCONFIG™ pins and can program the Flash directly or in the background. Direct programming takes place during config mode, background programming during user mode. The FPGA enters config mode at power up, when the PROGRAMN pin is pulled low, or when a refresh command is issued via JTAG; it enters user mode when it wakes up, i.e. when the device begins running user code. These two programming modes, direct and background, will be referred to in this document as Flash Direct and Flash Background.

Figure 14-1. Programming Block Diagram



Configuration Pins

The LatticeXP2 has one dedicated and nine dual-purpose sysCONFIG pins. The dual-purpose pins are available as extra I/O pins if they are not used for configuration.

The configuration mode pins, along with a programmable option, controls how the LatticeXP2 will be configured. The configuration mode pins (CFG[1:0]) are generally hard wired on the PCB and determine which configuration mode will be used; the programmable option is accessed via preferences in Lattice ispLEVER® design software, or as HDL source file attributes, and allows the user to protect the configuration pins from accidental use by the user or the place-and-route software. The LatticeXP2 devices also support ispJTAG for configuration, including transparent readback, and for JTAG testing. The following sections describe the functionality of the sysCONFIG and JTAG pins. Note that JTAG and ispJTAG will be used interchangeably in this document. Table 14-1 is provided for reference.

Table 14-1. Configuration Pins for the LatticeXP2 Device [1]

Pin Name	I/O Type	Pin Type	Mode Used
CFG0	Input, weak pull-up	Dedicated	All
CFG1	Input, weak pull-up	Dual-Purpose[2]	MSPI, SSPI
PROGRAMN	Input, weak pull-up	Dual-Purpose[2]	MSPI, SSPI
INITN	Bi-Directional Open Drain, weak pull-up	Dual-Purpose[2]	MSPI, SSPI
DONE	Bi-Directional Open Drain with weak pull-up or Active Drive	Dual-Purpose[2]	MSPI, SSPI
CCLK	Input or Output	Dual-Purpose	MSPI, SSPI
SISPI	Input or Output	Dual-Purpose	MSPI, SSPI
SOSPI	Input or Output	Dual-Purpose	MSPI, SSPI
CSSPISN	Input, weak pull-up	Dual-Purpose	Slave SPI
CSSPIN	Output, tri-state, weak pull-up	Dual-Purpose	Master SPI
TDI	Input, weak pull-up	JTAG	
TDO	Output	JTAG	
TCK	Input with Hysteresis	JTAG	

Table 14-1. Configuration Pins for the LatticeXP2 Device (Continued)[1]

Pin Name	I/O Type	Pin Type	Mode Used
TMS	Input, weak pull-up	JTAG	

1. Weak pull-ups consist of a current source of 30uA to 150uA. The pull-up for CFG tracks V_{CC} (core); the pull-ups for TDI and TMS track V_{CCJ}; all other pull-ups track the V_{CCIO} for that pin.
2. This pin becomes a dedicated programming pin when the CFG0 pin is low.

sysCONFIG Pins

Following is a description of the sysCONFIG pins for the LatticeXP2 device. These pins are used to control or monitor the configuration process. These pins are used for non-JTAG programming sequences only. The JTAG pins will be explained later in the ispJTAG Pins section of this document.

CFG[1:0]

The Configuration Mode pin CFG0 is a dedicated input with a weak pull-up. The CFG1 pin is a dual-purpose input pin with a weak pull-up. The CFG pins are used to select the configuration mode for the LatticeXP2, i.e. what type of device the LatticeXP2 will configure from. At Power-On-Reset (POR), or when the PROGRAMN pin is driven low, the device will enter the configuration mode selected by the CFG[1:0] pins.

Table 14-2. LatticeXP2 Configuration Modes

Configuration Mode	CFG[1]	CFG[0]
SPI Flash Boot	0	0
Embedded Flash Boot	1	0
Self Download Mode (SDM)	X	1

When the CFG0 pin is high, the device will configure itself by reading the data stored in on-chip Flash; this is referred to as SDM, or Self Download Mode. See the Self-Download section of this document for more information regarding SDM. If the CFG0 pin is low then the device will read the CFG1 pin to determine which mode to enter. When CFG1 is low the device will first attempt to configure the SRAM using Master SPI mode with the external SPI Flash port. If this fails then the device will configure itself from the on-chip Flash if a configuration file is stored there. When CFG1 is high the device will first attempt to configure the SRAM using on-chip Flash. If a configuration file is not stored there then the device will configure itself using Master SPI mode with the external SPI Flash port.

Dual-Purpose sysCONFIG Pins

The following is a list of the dual-purpose sysCONFIG pins. These pins are available as general purpose I/O after configuration. If a dual-purpose pin is to be used both for configuration and as a general purpose I/O the user must adhere to the following:

- The general purpose I/O (GPIO) must maintain the same direction as it has during configuration. In other words, if the pin is an input during configuration it must remain an input as a GPIO. If it's an output during configuration it must remain an output as a GPIO. If it's a bi-directional it must remain a bi-directional as a GPIO.
- The I/O type must remain the same. In other words, if the pin is a 3.3V CMOS pin (LVCMOS33) during configuration it must remain a 3.3V CMOS pin as a GPIO.
- The Persistent option must be set to OFF. The Persistent option will be set to OFF by the software unless the user sets the SLAVE_SPI_PORT to ENABLE using the Design Planner in ispLEVER. This option is shown in the Global tab of the Design Planner Speadsheet view.
- The user is responsible for insuring that no internal or external logic will interfere with device configuration.

After configuration these pins, if not used as GPIO, are tri-stated and weakly pulled up.

CFG1
The CFG1 pin is a dual-purpose input with a weak pull-up. Its function is described in the section above. When the CFG0 pin is high, the CFG1 pin is not used for configuration and becomes a general purpose I/O pin available to the user.

PROGRAMN
The PROGRAMN pin is a dual-purpose input with a weak pull-up. This pin is used to initiate a non-JTAG SRAM configuration sequence. A high to low signal applied to PROGRAMN sets the device into configuration mode. The PROGRAMN pin can be used to trigger configuration at any time. If the device is using JTAG then PROGRAMN will be ignored until the device is released from JTAG mode.

The PROGRAMN pin is only available if the CFG0 pin is set to 0 (not in SDM mode). When the CFG0 pin is set to 1 then PROGRAMN becomes a general purpose I/O pin available to the user.

When the CFG0 pin is set to 0, the PROGRAMN pin becomes a dedicated programming pin.

INITN
The INITN pin is a dual-purpose bi-directional open drain pin with a weak pull-up. INITN is capable of driving a low pulse out as well as detecting a low pulse driven in.

During SRAM configuration from an external device INITN going low indicates that the SRAM is being initialized; INITN going high indicates that the FPGA is ready to accept configuration data. To delay configuration the INITN pin can be held low externally. The device will not enter configuration mode as long as the INITN pin is held low. After configuration has started INITN is used to indicate a bitstream error. The INITN pin will be driven low if the calculated CRC and the configuration data CRC do not match; DONE will then remain low and the LatticeXP2 will not wake up.

During SRAM configuration from on-chip Flash INITN is not used or monitored.

When programming on-chip Flash the INITN pin is not used. During Flash Direct programming an error will prevent the FPGA from configuring from the Flash, during Flash Background programming an error will not affect the configuration already running in SRAM.

The INITN pin is only available if the CFG0 pin is set to 0 (not in SDM mode). When the CFG0 pin is set to 1 then INITN becomes a general purpose I/O pin available to the user.

When the CFG0 pin is set to 0, the INITN pin becomes a dedicated programming pin.

DONE
The DONE pin is a dual-purpose bi-directional open drain with a weak pull-up (default), or an actively driven pin. DONE will be driven low when the device is in configuration mode and the internal DONE bit is not programmed. When the INITN and PROGRAMN pins go high, and the internal DONE bit is programmed, the DONE pin will be released (or driven high, if it is an actively driven pin). The DONE pin can be held low externally and, depending on the wake-up sequence selected, the device will not become functional until the DONE pin is externally brought high.

Reading the DONE bit is a good way for an external device to tell if the FPGA is configured.

When using JTAG to configure SRAM the DONE pin is driven by the boundary scan cell, so the state of the DONE pin has no meaning until configuration is completed.

The DONE pin is only available if the CFG0 pin is set to 0 (not in SDM mode). When the CFG0 pin is set to 1 then DONE becomes a general purpose I/O pin available to the user.

When the CFG0 pin is set to 0, the DONE pin becomes a dedicated programming pin.

CCLK
CCLK is a dual-purpose bi-directional pin; direction depends on whether a Master or Slave mode is selected. If a Master mode is selected, the CCLK pin will become an output pin; otherwise CCLK is an input pin.

If the CCLK pin becomes an output, the internal programmable oscillator is connected to the CCLK and is driven out to slave devices. CCLK will stop 100 to 500 clock cycles after the DONE pin is brought high and the device wake-up sequence completed. The extra clock cycles ensure that enough clocks are provided to wake-up other devices in the chain. When stopped, CCLK becomes an input (tri-stated output). CCLK will restart (become an output) on the next configuration initialization sequence.

The MCCLK_FREQ parameter (one of the global settings in the Design Planner of ispLEVER) controls the CCLK master frequency (see the LatticeXP2 Family Data Sheet On-Chip Oscillator section for the frequency selections available). The software default setting for the configuration CCLK is 2.5 MHz. One of the first things loaded during configuration is the MCCLK_FREQ parameter; once this parameter is loaded the frequency changes to the selected value. Care should be exercised not to exceed the frequency specification of the slave devices or the signal integrity capabilities of the PCB layout.

CSSPIN
The CSSPIN pin is a dual-purpose output pin with a weak pull-up. The CSSPIN is an active low chip select to an external SPI flash when used with the Master SPI mode. The CSSPIN pin becomes a dedicated pin if the CFG0 pin is set to 0 (not in SDM mode). When the CFG0 pin is set to 1 then CSSPIN becomes a general purpose I/O pin available to the user.

If the CFG0 is set to 0 then this pin should be driven high unless the Master SPI mode is selected to avoid contention between the Master and Slave SPI modes.

CSSPISN
The CSSPISN pin is a dual-purpose input pin with a weak pull-up. The CSSPISN is an active low chip select to the internal SPI interface and is used with the Slave SPI mode.

If the CSSPISN is driven low while in the middle of Master SPI port activity the Master SPI shall be disabled and the Slave SPI interface activated.

The PERSISTENT preference must be set to ON in order to preserve this pin as CSSPISN and allow access to the Slave SPI interface. The PERSISTENT preference will be set by the software automatically when the user sets the SLAVE_SPI_PORT option in the Design Planner.

SISPI
The SISPI pin is a dual-purpose bi-directional pin; direction depends upon whether a Master or Slave mode is active. The SISPI is the Input data pin when using the Slave SPI mode and is the Output data pin when using the Master SPI mode.

The PERSISTENT preference must be set to ON in order to preserve this pin as SISPI and allow access to the Slave SPI interface. The PERSISTENT preference will be set by the software automatically when the user sets the SLAVE_SPI_PORT option in the Design Planner.

SOSPI
The SOSPI pin is a dual-purpose bi-directional pin; direction depends upon whether a Master or Slave mode is active. The SOSPI is the Input data pin when using the Master SPI mode and is the Output data pin when using the Slave SPI mode.

The PERSISTENT preference must be set to ON in order to preserve this pin as SOSPI and allow access to the Slave SPI interface. The PERSISTENT preference will be set by the software automatically when the user sets the SLAVE_SPI_PORT option in the Design Planner.

Table 14-3. Flash Programming Mode Pin Usage

Flash Programming Mode	Direct	Background	Direct	Background
Port[6]	**Slave SPI**		**ispJTAG[1]**	
Pins	**CCLK, CSSPISN, SISPI, SOSPI**		**TAP**	
User I/O States	**Tristate**	**User**	**BSCAN**	**User**
PROGRAMN	↓	Keep at High	Keep At High[2]	
INITN	Pass/Fail	Pass/Fail	Not Used[3]	
DONE	Done	Not Used	Keep at High[4]	
SLAVE_SPI_PORT preference	Don't Care[t5]	ENABLE[5]	Don't Care	

1. ispJTAG can be used to program the Flash regardless of the state of the CFG pins.
2. The state of the PROGRAMN pin is ignored by the device during JTAG Flash programming but the pin should be held high as a low will cause a configuration failure. When the device is in the SDM mode, the PROGRAMN pin is a dedicated I/O pin so it does not affect configuration.
3. The state of the INITN pin is ignored by the device during JTAG Flash programming but the pin should be allowed to float high using the internal pull-up. When the device is in the SDM mode, the INITN pin is a dedicated I/O pin so it does not affect configuration.
4. The state of the DONE pin is ignored by the device during JTAG Flash programming but the pin should be allowed to float high using the internal pull-up as a low can keep the device from waking up. When the device is in the SDM mode, the DONE pin is a dedicated I/O pin so it does not affect configuration.
5. The SLAVE_SPI_PORT preference must be set to ENABLE to use the Slave SPI port after the device has been configured. The Slave SPI port is also available when the device is not configured.
6. The Master SPI port can only be used to configure the SRAM in direct mode from an external SPI Flash memory. The CFG pins must be set per Table 14-2 to enable this mode.

Table 14-4. Memory Access Modes

Mode	Flash		SRAM	
	Read	**Write**	**Read**	**Write**
Slave SPI	Yes[2]	Yes[1]	Yes	No
Master SPI	No	No	No	Yes[3]

1. Slave SPI mode can only write to on-board Flash memory in background mode unless the Flash memory is erased.
2. Slave SPI mode can read from on-board Flash memory in background mode only.
3. Master SPI mode can write to SRAM in direct mode only.

External SPI Flash

When the Master SPI mode is used for configuration an external SPI Flash device is required to hold the configuration data. The size of the bitstream and the required external SPI Flash is shown in Table 14-5.

Table 14-5. Maximum Configuration Bits

Density	Bitstream Size (Mb)	SPI Flash (Mb)
XP2-5	1.27	2
XP2-8	1.99	2
XP2-17	3.54	4
XP2-30	5.79	8
XP2-40	8.03	16

Programming Sequence

There are two types of programming, SRAM, and Flash Background. This section goes through the process for each showing how the dedicated pins are used.

SRAM

When not using SDM (Self Download Mode, on-chip Flash) to program the SRAM, the sequence begins when the internal power-on reset (POR) is released or the PROGRAMN pin is driven low (see Figure 14-2). The LatticeXP2 then drives INITN low, tri-states the I/Os, and initializes the internal SRAM and control logic. When this is complete, if PROGRAMN is high, INITN will be released. If INITN is held low externally the LatticeXP2 will wait until it goes high. When INITN goes high the LatticeXP2 begins looking for the configuration data using the internal Flash memory or the Master SPI port, as determined by the CFG pins.

If the CFG1 pin is high and the Flash Done bit is set (indicating that the on-board Flash memory is programmed) then the LatticeXP2 will boot from the on-board Flash memory. If the Flash Done bit is not set then the LatticeXP2 will boot from the external SPI Flash memory using the Master SPI mode.

If the CFG1 pin is low then the LatticeXP2 will boot from the external SPI Flash memory using the Master SPI mode. In the event of an error the LatticeXP2 will boot from the on-board Flash memory if the Flash Done bit is set.

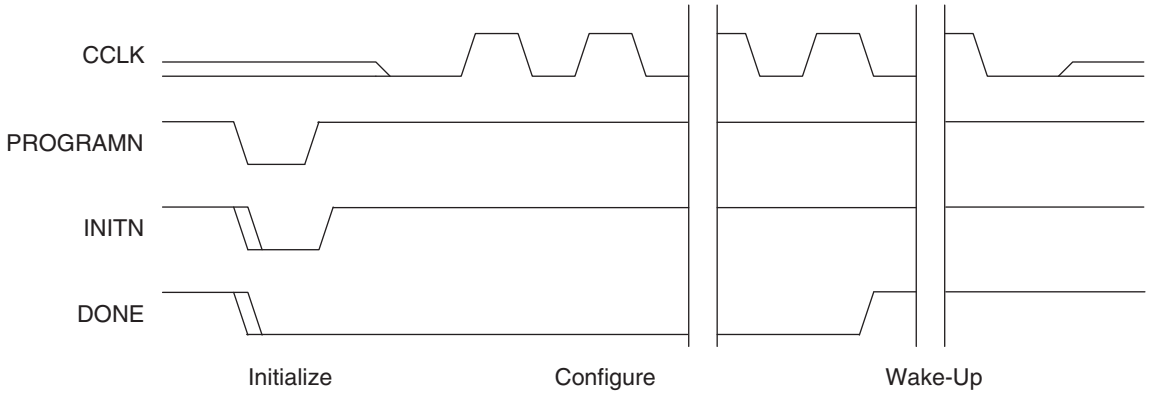
Once configuration is complete the internal DONE bit is set, the DONE pin goes high, and the FPGA wakes up (enters user mode). If a CRC error is detected when reading the bitstream INITN will go low, the internal DONE bit will not be set, the DONE pin will stay low, and the LatticeXP2 will not wake up.

When using SDM to program SRAM the sequence is similar but INITN is not used or monitored. The sequence begins when the internal power-on reset (POR) is released. The LatticeXP2 then tri-states the I/Os and initializes the internal SRAM and control logic. When initialization is complete the LatticeXP2 begins loading configuration data from on-chip Flash.

When using SDM, if the Flash has been programmed, then the configuration sequence will proceed using the data in on-chip Flash. If the Flash has not been programmed, the configuration sequence will stop. Once the Flash has been programmed, a POR or JTAG Refresh instruction must occur to restart the configuration sequence.

As with non-SDM, once configuration is complete the internal DONE bit is set, the DONE pin goes high, and the FPGA wakes up (enters user mode).

Figure 14-2. SRAM Configuration Timing Diagram



Flash Background

Flash Background programming is possible using the Slave SPI port when it is enabled. The Slave SPI port can be enabled in the SDM mode as well as the SPI mode. Flash Background will not disturb the FPGA's present configuration in SRAM.

Flash Background programming may be used in both config mode and user mode (Done bit = 0 or 1). To support Flash Background programming in user mode the SLAVE_SPI_PORT preference must be set to ENABLE.

When the CSSPISN pin goes low, the FPGA will wait for the preamble and then look for the proper commands. A low on INITN indicates an error during a Flash erase or program. Data is written and read on the SISPI and SOSPI pins.

After programming the Flash the user may toggle the PROGRAMN pin to transfer the Flash data to SRAM if the SPI mode is being used. If the SDM mode is being used then the SRAM will be updated on the next power up sequence of when a refresh instruction is issued.

If the CSSPISN pin is driven low, the CSSPIN should be driven high and CCLK maintained as an input to prevent activating the Master SPI interface. This will avoid contention between the Master and Slave SPI interfaces.

ispJTAG Pins

The ispJTAG pins are standard IEEE 1149.1 TAP (Test Access Port) pins. The ispJTAG pins are dedicated pins and are always accessible when the LatticeXP2 device is powered up. When programming the SRAM via ispJTAG the dedicated programming pins, such as DONE, cannot be used to determine programming progress. This is because the state of the boundary scan cell will drive the pin, per JTAG 1149.1, rather than normal internal logic.

TDO

The Test Data Output pin is used to shift out serial test instructions and data. When TDO is not being driven by the internal circuitry, the pin will be in a high impedance state.

TDI

The Test Data Input pin is used to shift in serial test instructions and data. An internal pull-up resistor on the TDI pin is provided. The internal resistor is pulled up to $V_{CCJ.}$

TMS

The Test Mode Select pin controls test operations on the TAP controller. On the falling edge of TCK, depending on the state of TMS, a transition will be made in the TAP controller state machine. An internal pull-up resistor on the TMS pin is provided. The internal resistor is pulled up to $V_{CCJ.}$

TCK

The test clock pin, TCK, provides the clock to run the TAP controller, which loads and unloads the data and instruction registers. TCK can be stopped in either the high or low state and can be clocked at frequencies up to the frequency indicated in the device data sheet. The TCK pin supports the value is shown in the DC parameter table of the data sheet. The TCK pin does not have a pull-up. A pull-down on the PCB of 4.7 K is recommended to avoid inadvertent clocking of the TAP controller as V_{CC} ramps up.

VCCJ

JTAG V_{CC} (V_{CCJ}) supplies independent power to the JTAG port to allow chaining with other JTAG devices at a common voltage. V_{CCJ} must be connected even if JTAG is not used. This voltage may also power the JTAG download cable. Valid voltage levels are 3.3V, 2.5V, 1.8V, 1.5V, and 1.2V.

Please see *In-System Programming Design Guidelines for ispJTAG Devices,* available on the Lattice web site at www.latticesemi.com, for further JTAG chain information.

Configuration and JTAG Voltage Levels

All of the control pins and programming pins default to LVCMOS. The CFG0 pin is linked to V_{CC} (core); TCK, TDI, TDO, and TMS track V_{CCJ}; all other pins track the V_{CCIO} for that pin.

Configuration Modes and Options

The LatticeXP2 device supports two configuration modes, utilizing the SPI port or self-configuration. On power up, or upon driving the PROGRAMN pin low depending upon the current mode, the CFG[1:0] pins are sampled to determine the mode that will be used to configure the LatticeXP2 device. The CFG pins are generally hard wired on

the PCB and determine how the device will retrieve its configuration data. The SLAVE_SPI_PORT preference is a programmable option which can be set using the Design Planner in Lattice ispLEVER design software, or as HDL source file attributes, and allow the user to protect the configuration pins from accidental use by the user or the place-and-route software.

Configuration Options

Several configuration options are available for each configuration mode.

- When using a master clock, the master clock frequency can be set.
- A security bit is provided to prevent SRAM or Flash readback.

By setting the proper parameters in the Lattice ispLEVER design software the selected configuration options are set in the generated bitstream. As the bitstream is loaded into the device the selected configuration options take effect. These options are described in the following sections.

Master Clock
If the LatticeXP2 is a Master device the CCLK pin will become an output with the frequency set by the user. The default Master Clock Frequency is 2.5 MHz.

The user can determine the Master Clock frequency by setting the MCCLK_FREQ preference in the Lattice ispLEVER design software. One of the first things loaded during configuration is the MCCLK_FREQ parameter; once this parameter is loaded the frequency changes to the selected value using a glitchless switch. Care should be exercised not to exceed the frequency specification of the slave devices or the signal integrity capabilities of the PCB layout.

Security Bit
Setting the security bit prevents readback of the SRAM and Flash from JTAG or the sysCONFIG pins. When the security bit is set the only operations available are erase and write. The security bit is updated as the last operation of SRAM configuration or Flash programming. By using on-chip Flash, and setting the security bit, the user can create a very secure device.

The security bit is accessed via the Design Planner in ispLEVER design software.

More information on device security can be found in the document *FPGA Design Security Issues: Using the ispXPGA Family of FPGAs to Achieve High Design Security,* available on the Lattice Semiconductor web site at www.latticesemi.com.

Slave SPI Mode

In the Slave SPI mode the CCLK pin becomes an input and commands will be read into the LatticeXP2 on the SISPI pin at the rising edge of CCLK. Data will be written out of the LatticeXP2 on the SOSPI pin at the falling edge of CCLK.

Care must be exercised during read back of EBR or PFU memory. It is up to the user to ensure that reading these RAMs will not cause data corruption, i.e. these RAMs may not be read while being accessed by user code.

The CSSPISN enables and disables the SPI interface operation. When CSSPISN is high the SPI interface is deselected and the SOSPI pin is at high impedance. When CSSPISN is brought low the SPI interface is selected, commands can be written into and data read from the LatticeXP2. After power up the CSSPISN must transition from high to low before a new command can be accepted.

The Slave SPI mode can also be used to access on-chip Flash. The CSSPISN pin must be held low to write to on-chip Flash; data is input from SISPI. The Slave SPI mode can also be used for readback of both Flash and SRAM. By driving the CSSPISN low, the device will input the readback instructions on the SISPI pin and the data will be written out on the SOSPI pin; a bit in the read command will determine if the read is directed to Flash or SRAM. In order to support readback while the device is in user mode (the DONE pin is high), the SLAVE_SPI_PORT preference must be set to ENABLE using the Design Planner.

Master SPI Mode

In Master SPI mode the LatticeXP2 will drive CCLK out to the Slave SPI Flash device that will provide the bitstream. The Master device will write commands out on SISPI at the rising edge of CCLK and will accept data on SOSPI at the falling edge of CCLK. The Master Serial device starts driving CCLK at the beginning of the configuration and continues to drive CCLK until the external DONE pin is driven high and an additional 100 to 500 clock cycles have been generated. The CCLK frequency on power up defaults to 2.5 MHz. The master clock frequency default remains unless a new clock frequency is loaded from the bitstream.

Self Download Mode

Self Download Mode (SDM) allows the FPGA to configure itself without using any external devices, and because the bitstream is not exposed this is also a very secure configuration mode. The user may access on-chip Flash using ispJTAG or the slave SPI port pins.

JTAG may access the on-chip Flash any time the device is powered up, without disturbing device operation. JTAG may also read and write the configuration SRAM. If access to the on-chip Flash and SRAM is limited to JTAG then SLAVE_SPI_PORT can be set to DISABLE, freeing the dual-purpose pins for use as general purpose I/O.

ispJTAG Mode

The LatticeXP2 device can be configured through the ispJTAG port. The ispJTAG port is always on and available, regardless of the configuration mode selected. The SLAVE_SPI_PORT can be set to ENABLE in the Lattice ispLEVER design software to tell the place and route tools that the JTAG port will be used exclusively, i.e. the SPI port will not be used. Setting the SLAVE_SPI_PORT to ENABLE allows software to use all of the dual-purpose pins as general purpose I/Os.

ISC 1532

Configuration through the ispJTAG port conforms to the IEEE 1532 Standard. The Boundary Scan cells take control of the I/Os during any 1532 mode instruction. The Boundary Scan cells can be set to a pre-determined value whenever using the JTAG 1532 mode. Because of this the dedicated pins, such as DONE, cannot be relied upon for valid configuration status.

Transparent Readback

The ispJTAG Transparent Readback mode allows the user to read the content of the device SRAM or Flash while the device remains in a functional state. Care must be exercised when reading EBR and distributed RAM, as it is possible to cause conflicts with accesses from the user design (causing possible data corruption).

The I/O and non-JTAG configuration pins remain active during a Transparent Readback. The device enters the Transparent Readback mode through a JTAG instruction.

Boundary Scan and BSDL Files

BSDL files for this device can be found on the Lattice web site at www.latticesemi.com. The boundary scan ring covers all of the I/O pins, as well as the dedicated and dual-purpose sysCONFIG pins.

Wake Up Options

When configuration is complete (the SRAM has been loaded), the device should wake up in a predictable fashion. The following selections determine how the device will wake up. Two synchronous wake up processes are available. One automatically wakes the device up when the internal Done bit is set regardless of whether the DONE pin is held low externally or not, the other waits for the DONE pin to be driven high before starting the wake up process. The DONE_EX preference determines whether the external DONE pin will control the synchronous wake up. When the device is in the SDM mode the DONE pin is not used and therefore the DONE_EX preference has no effect.

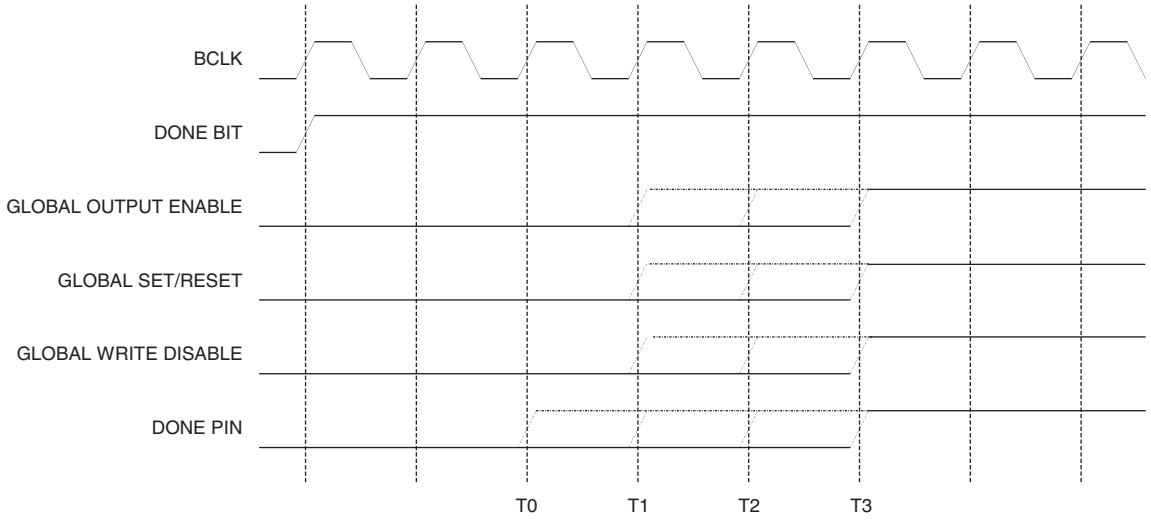
Wake Up Sequence

Table 14-6 provides a list of the wake up sequences supported by the LatticeXP2.

Table 14-6. Wake Up Sequences Supported by LatticeXP2

Sequence	Phase T0	Phase T1	Phase T2	Phase T3
1	DONE	GOE, GWDIS, GSR		
2	DONE		GOE, GWDIS, GSR	
3	DONE			GOE, GWDIS, GSR
4	DONE	GOE	GWDIS, GSR	
5	DONE	GOE		GWDIS, GSR
6	DONE	GOE	GWDIS	GSR
7	DONE	GOE	GSR	GWDIS
8		DONE	GOE, GWDIS, GSR	
9		DONE		GOE, GWDIS, GSR
10		DONE	GWDIS, GSR	GOE
11		DONE	GOE	GWDIS, GSR
12			DONE	GOE, GWDIS, GSR
13		GOE, GWDIS, GSR	DONE	
14		GOE	DONE	GWDIS, GSR
15		GOE, GWDIS	DONE	GSR
16		GWDIS	DONE	GOE, GSR
17		GWDIS, GSR	DONE	GOE
18		GOE, GSR	DONE	GWDIS
19			GOE, GWDIS, GSR	DONE
20		GOE, GWDIS, GSR		DONE
21 (Default)		GOE	GWDIS, GSR	DONE
22		GOE, GWDIS	GSR	DONE
23		GWDIS	GOE, GSR	DONE
24		GWDIS, GSR	GOE	DONE
25		GOE, GSR	GWDIS	DONE

Figure 14-3. Wake Up Sequence to Internal Clock



Synchronous to Internal Done Bit
If the LatticeXP2 device is the only device in the chain, or the last device in a chain, the wake up process should be initiated by the completion of the configuration. Once the configuration is complete, the internal Done bit will be set and then the wake up process will begin.

Synchronous to External DONE Signal
The DONE pin can be selected to delay wake up. If DONE_EX is true then the wake up sequence will be delayed until the DONE pin is high. The device will then follow the WAKE_UP sequence selected. When the device is in the SDM mode the DONE pin is not used and therefore the DONE_EX preference has no effect.

Software Selectable Options

In order to control the configuration of the LatticeXP2 device beyond the default settings, software preferences are used. Table 14-7 is a list of the preferences with their default settings.

Table 14-7. Software Preference List for the LatticeXP2

Preference Name	Default Setting [List of All Settings]
SLAVE_SPI_PORT	DISABLE [disable, enable]
MASTER_SPI_PORT	DISABLE [disable, enable]
DONE_OD	ON [off, on]
DONE_EX	OFF [off, on]
MCCLK_FREQ	Lowest Frequency
CONFIG_SECURE	OFF [off, on]
WAKE_UP	21 (DONE_EX = off) 4 (DONE_EX = on)
WAKE_ON_LOCK	OFF [off, on]
INBUF	ON [off, on]

Slave SPI Port

In order to use the Slave SPI port while in user mode to read SRAM or Flash memory, the SLAVE_SPI_PORT preference must be set to ENABLE. Setting this preference preserves the Slave SPI port pins so the FPGA can be accessed by an external device while in user mode. This also lets the software know that the Slave SPI port pins are reserved and NOT available for use by the fitter or the user.

Master SPI Port

In order to use the Master SPI Port for configuration, the MASTER_SPI_PORT preference should be set to ENABLE.

Configuration Mode

The device knows which physical sysCONFIG port will be used by reading the state of the CFG[1:0] pins, but the fitter software also needs to know which port will be used. The fitter will determine the configuration mode based upon the setting of the SLAVE_SPI_PORT and the MASTER_SPI_PORT preferences. The user may set these preferences, and the ones listed below, using the Design Planner tool.

There are several additional configuration options, such as overflow, that are set by software. These options are selected by clicking Properties under Generate Bitstream Data in ispLEVER. If either overflow option is selected, then the DONE_EX and WAKE_UP selections will be set to correspond (see Table 14-8). Refer to the Configuration Modes and Options section of this document for more details.

Table 14-8. Overflow Option Defaults

Overflow Option	DONE_EX Preference	WAKE_UP Preference
Off	Off (Default)	Default 21 (user selectable 1 through 25)
Off	On	Default 21 (user selectable 1 through 25)
On (either)	On (automatically set by software)	Default 4 (User selectable 1 through 7)

DONE Open Drain

The “DONE_OD” preference allows the user to configure the DONE pin as an open drain pin. The “DONE_OD” preference is only used for the DONE pin. When the DONE pin is driven low, internally or externally, this indicates that configuration is not complete and the device is not ready for the wake up sequence. Once configuration is complete, with no errors, and the device is ready for wake up, the DONE pin must be driven high. For other devices to be able to control the wake up process an open drain configuration is needed to avoid contention on the DONE pin. The “DONE_OD” preference for the DONE pin defaults to ON. The DONE_OD preference is automatically set to ON if the DONE_EX preference is set to ON. See Table 14-9 for more information on the relationship between DONE_OD and DONE_EX. When the device is in the SDM mode the DONE pin is not used and therefore the DONE_OD and DONE_EX preferences have no effect.

DONE External

The LatticeXP2 device can wake up on its own after the Done bit is set or wait for the DONE pin to be driven high externally. Set DONE_EX = ON to delay wake up until the DONE pin is driven high by an external signal synchronous to the clock; select OFF to synchronously wake up when the internal Done bit is set and ignore any external driving of the DONE pin. The default is DONE_EX = OFF. If DONE_EX is set to ON, DONE_OD will be set to ON. If an external signal is driving the DONE pin it should be open drain as well (an external pull-up resistor may need to be added). See Table 14-9 for more information on the relationship between DONE_OD and DONE_EX.

Table 14-9. Summary of DONE Pin Preferences

DONE_EX[1]	Wake Up Process	DONE_OD[1]
OFF	External DONE ignored	User selected
ON	External DONE low delays	Set to Default (ON)

1. When the device is in the SDM mode the DONE pin is not used and therefore the DONE_OD and DONE_EX preferences have no effect.

Master Clock Selection

When the user has determined that the LatticeXP2 will be a master configuration device (by properly setting the CFG[1:0] pins), and therefore provide the source clocking for configuration, the CCLK pin becomes an output with the frequency set by the value in MCCLK_FREQ. At the start of configuration the device operates at the default Master Clock Frequency of 2.5 MHz. Some of the first bits in the configuration bitstream are MCCLK_FREQ, once these are read the clock immediately starts operating at the user-defined frequency. The clock frequency is changed using a glitchless switch.

Security

When CONFIG_SECURE is set to ON, NO read back operation will be supported through the sysCONFIG or ispJTAG port of the general contents. The ispJTAG DeviceID area is readable and not considered securable. Default is OFF.

Wake Up Sequence

The WAKE_UP sequence controls three internal signals and the DONE pin. The DONE pin will be driven after configuration and prior to user mode. See the Wake Up Sequence section of this document for an example of the phase controls and information on the wake up selections. The default setting for the WAKE_UP preference is determined by the DONE_EX setting.

Wake Up with DONE_EX = Off (Default Setting)
The WAKE_UP preference for DONE_EX = OFF (default) supports the user selectable options 1 through 25, as shown in Table 14-6. If the user does not select a wake-up sequence, the default, for DONE_EX = OFF, will be wake-up sequence 21.

Wake Up with DONE_EX = On
The WAKE_UP preference for DONE_EX = ON supports the user selectable options 1 through 7, as shown in Table 14-6. If the user does not select a wake-up sequence, the default will be wake-up sequence 4.

Wake On Lock Selection

The Wake On Lock preference determines whether the device will wait for the PLL to lock before beginning the wake-up process.

ON – The device will not wake up until the PLL lock signal for the given PLL is active.

OFF (default) – The device will wake up regardless of the state of the PLL lock signal.

Power Save

An I/O Power Save mode option, called INBUF, is used for the LatticeXP2 device and will deactivate unused input buffers to save power. This is only affects comparator type inputs pins (pins that use VREF), like HSTL, SSTL, etc.

The Power Save mode limits some of the functionality of Boundary Scan. For Boundary Scan testing it is recommended that the INBUF global preference be turned ON to activate all unused input buffers.

One Time Programmable Fuse

The LatticeXP2 has a One Time Programmable (OTP) fuse that can be used to prevent the on chip Flash configuration memory from being erased or programmed. This does not prevent the Flash Tag Memory or Flash User memory from being programmed, so these features are still available. The OTP fuse can be set using the Global Configuration options in the ispLEVER Design Planner or it can be set directly using the ispVM® System software at the time of download.

User GOE

The LatticeXP2 has a User GOE (Global Output Enable) feature. This allows the I/Os to be held in Boundary scan control after the standard wake-up sequence has completed. User logic determines when the outputs get turned over from Boundary Scan to User Logic control. This user logic input will be through a CIB and is valid for JTAG “wake up” instructions only.

This feature is instantiated by the user as a macro called IOWAKEUP. This macro only has one signal and can only be controlled immediately after the wakeup sequence (not anytime after).

Tag Memory

The TAG Memory is a block of Flash memory which is always available for read or write (programming in Flash terms) through the Slave SPI port. The LatticeXP2 can be in user mode or an un-programmed state (blank device), independent of the CFG[1:0] pin setting. The only exceptions would be when the LatticeXP2 is in BSCAN test or in direct programming mode. During these modes the SPI interface is unavailable because the I/O is under BSCAN control.

The TAG memory is also available through the JTAG port.

Table 14-10 shows the amount of Tag memory available in each LatticeXP2 device. Each LatticeXP2 device has one dedicated row of TAG memory.

Table 14-10. LatticeXP2 Family TAG Memory

Device Density	Tag Memory (Bits)	Tag Memory (Bytes)
XP2-5	632	79
XP2-8	768	96
XP2-17	2184	273
XP2-30	2640	330
XP2-40	3384	423

Note: The Initial Power on Value (INITVAL) for all Flash cells is all 1's.

The TAG memory has the following features and limitations.

- Each row of TAG memory is limited to sequential access only. Once the read command is specified, the entire TAG memory contents are read sequentially in a first-in-first-out manner.
- Data access speed is limited by the speed of the TAG memory which is Flash based.
- The TAG memory comes from the factory erased. It will retain the user assigned value after programming even during power off periods.
- The TAG memory can be read or written using the hardwired JTAG and SPI interface pins.
- The TAG memory is ready to use upon power-up of the LatticeXP2. It does not need any software IP or design loaded into the device to access the TAG memory via the hardwired interface.
- The TAG memory can also be read and modified from the FPGA core logic using the slave-SPI CIB interface pins to emulate an I^2C port.
- The TAG memory is always accessible regardless of the security setting of the device.

The TAG memory is designed for storing typically “static” data - data that is not likely to change. This TAG memory can take the place of on-board EEPROM or simple Flash memory which might be used for the following system management and manufacturing control information (listed as examples only):

- Saving Electronic ID codes
- Version management
- Date stamping
- Manufacturing version control
- Asset management and tracking
- System calibration settings
- Device serialization and/or inventory control.

Slave SPI Mode Operation

The Slave SPI Mode interface to the TAG memory supports both SPI Bus Mode 0 and Mode 3 operations. In SPI Bus Mode 0 the CLK pin is normally low when the SPI master is in standby and data is not being transmitted. In SPI Bus Mode 3 the CLK pin is normally high during this condition. In both cases the data at the SISPI pin is sampled on the rising edge of CCLK and the data output on the SOSPI pin is clocked on the falling edge of CCLK.

For more information on using the TAG memory please see Lattice technical note TN1137, *LatticeXP2 Memory Usage Guide.*

User Flash

The User Flash is designed as a non-volatile memory location to back up the data stored in the EBR Ram blocks. This gives the user a reliable method to save the contents of the Ram memory for later use.

The amount of User Flash for LatticeXP2 devices is directly tied to the number of EBR blocks in the device and it scales with density. The User Flash is organized physically as either 1- or 2- separate User Flash Modules. However, all User Flash Modules are logically treated as one unified block.

Table 14-11. User Flash Organization

Block Type	XP2-5K	XP2-8K	XP2-17K	XP2-30K	XP2-40K
Physical UFM blocks	1	1	2	2	2
Logical UFM blocks	1	1	1	1	1

The EBR blocks act as the primary interface for the User Flash. Users do not have direct access to the User Flash.

The contents of the EBR can be saved into the User Flash via a store-to-flash control signal. The EBR contents must be saved into the User Flash when required. Two signals, UFMFAIL and UFMBUSYN are provided for keeping track of the status of the store-to-flash command. If the UFMFAIL signal is low and the UFMBUSYN signal is high then the EBR contents were successfully stored in the Flash memory.

The User Flash memory has the following constraints upon its usage.

- The Store-to-Flash operation has impact only on the EBR RAM (Single-Port, True Dual-Port, Pseudo Dual-Port) configurations.
- During the Store-to-Flash operation, the EBR blocks are unavailable for user operation and the Flash is unavailable for configuration operation.
- No selective EBR storing is supported. A Store-to-Flash operation will store the contents of all EBR blocks.
- Due to silicon limitations the user cannot use Store-to-Flash operation if the SED is operating in an Always mode.
- UFM mode cannot concurrently be used with Transparent/Background mode (Flash or SRAM). The SSPI configuration and verify operations (which are essentially Transparent Mode operations) are initiated by the user and the user needs to ensure that the UFM operation is not requested at the same time.

Table 14-12. Differences Between User Flash and Shadow Flash (EBR) Behavior

Parameter	User Flash	Shadow Flash (EBR)
Read/write access speed	Slow	Fast
Access nature	Sequential	Random
Data access	Limited (refer to Sec13.3)	Infinite or unlimited
Data organization	Sequential, one bit at a time	Flexible, variable data width
Data granularity	Whole UFM block	One EBR block

For more information, please see Lattice technical note TN1137, *LatticeXP2 Memory Usage Guide.*

Technical Support Assistance

Hotline: 1-800-LATTICE (North America)
+1-503-268-8001 (Outside North America)
e-mail: techsupport@latticesemi.com
Internet: www.latticesemi.com

Revision History

Date	Version	Change Summary
February 2007	01.0	Initial release.
May 2007	01.1	Updated sysCONFIG Pin descriptions
		Added footnote to Summary of DONE Pin Preferences table.
January 2008	01.2	Updated Slave SPI Port information and PROGRAMN pin usage.
		Removed PERSISTENT and CONFIG_MODE preferences.
		Added SLAVE_SPI_PORT and MASTER_SPI_PORT preferences.

LatticeXP2 Configuration Encryption and Security Usage Guide

February 2007 Technical Note TN1142

Introduction

Unlike a volatile FPGA, which requires an external boot-prom to store configuration data, the LatticeXP2™ devices are non-volatile and have on-chip configuration Flash. Once programmed (either by JTAG or SPI port), this data is a part of the FPGA device and can be used to self-download the SRAM portion without requiring any additional external boot prom. Hence it is inherently more secure than volatile FPGAs. Like the LatticeECP2/M, the LatticeXP2 family also offers the 128-bit Advanced Encryption Standard (AES) to protect the externally stored programming file. The user has total control over the 128-bit key and no special voltages are required to maintain the key within the FPGA. Additional security enhancement for the LatticeXP2 includes:

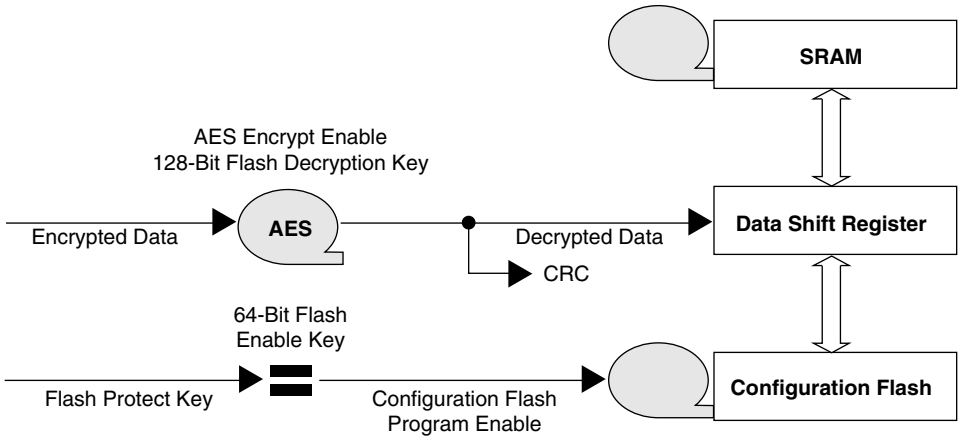
- A security bit for the Configuration and User Flash
- One-Time-Programmable (OTP) or Permanent Lock capability
- Flash Protect

This document explains the encryption and security features and how to take advantage of them.

Encryption/Decryption Flow

The LatticeXP2 supports both encrypted and non-encrypted JEDEC files. Since the non-encrypted flow is covered in Lattice technical note TN1141, *LatticeXP2 sysCONFIG™ Usage Guide*, this document will concentrate on the additional steps needed for the encrypted flow. The encrypted flow adds only two steps to the normal FPGA design flow, encryption of the configuration JEDEC file and programming the encryption key into the LatticeXP2. Figure 15-1 is a block diagram describing the LatticeXP2 encryption data paths that will be used throughout this document.

Figure 15-1. Encryption Block Diagram along with Flash Protect



Encrypting the JEDEC File

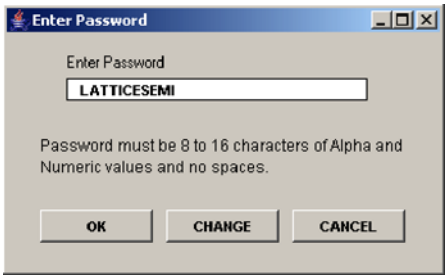
As with any other Lattice FPGA design flow, the design engineer must first create the design using the ispLEVER® design tool suite. The design is synthesized, mapped, placed and routed, and verified. Once the user is satisfied with the design, the final JEDEC file is ready for FPGA programming. This final JEDEC file is used to secure the design.

The JEDEC file can be encrypted using ispLEVER by going to the **Tools -> Security Settings** pull-down menu or by using the Universal File Writer (ispUFW), which is part of the Lattice ispVM® System tool suite.

ispLEVER Flow

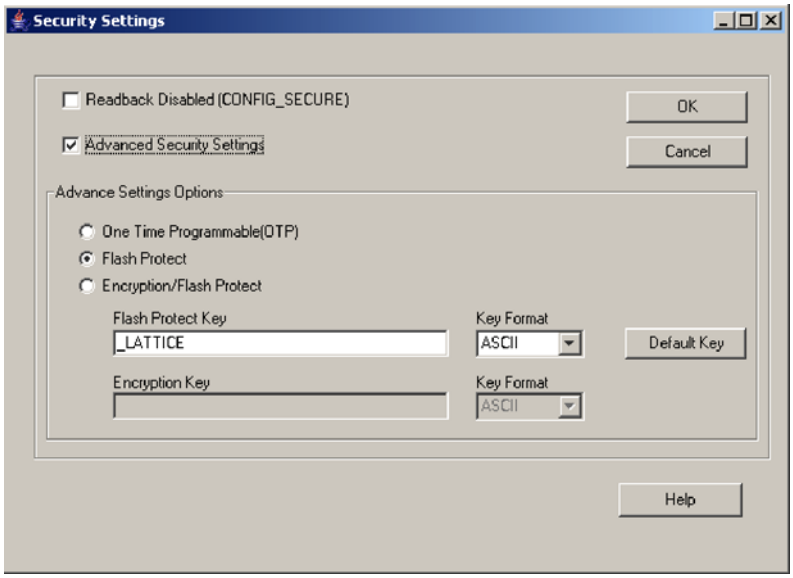
1. As mentioned above, to access the LatticeXP2 security setting GUI, go to **Project Navigator -> Tools -> Security Settings**. A password is required before entering the security features GUI section of the LatticeXP2.

Figure 15-2. Password Prompt with Default Password



2. This Password GUI prompt will automatically show the default password “LATTICESEMI”. The default password is in place for users who do not want to remember any administrative password, and especially for those who want to use the CONFIG_SECURE setting only. Users have the option of changing the password. It is the user’s responsibility to track all the keys and passwords since they will not be stored in the design files.

Figure 15-3. Security Settings



3. Once the user has selected security features, encrypted files will then be generated.

ispVM Flow

1. Start **ispUFW**. You can start ispUFW from the **Start -> Programs -> Lattice Semiconductor** menu in Windows. You will see a window that looks similar to Figure 15-4. You can also launch the ispUFW from the ispVM GUI by clicking on the UFW button on the toolbar (shown in Figure 15-5). Select **JEDEC** as the output file format, as shown in Figure 15-4.

Figure 15-4. Universal File Writer

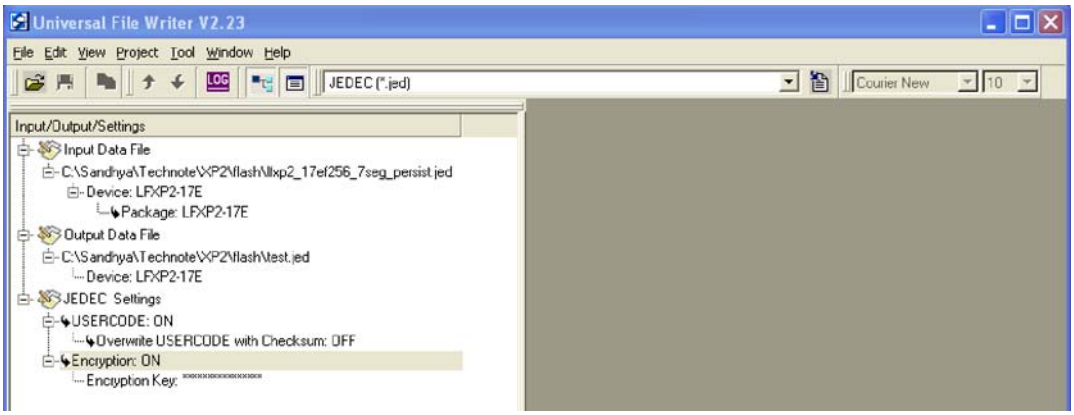
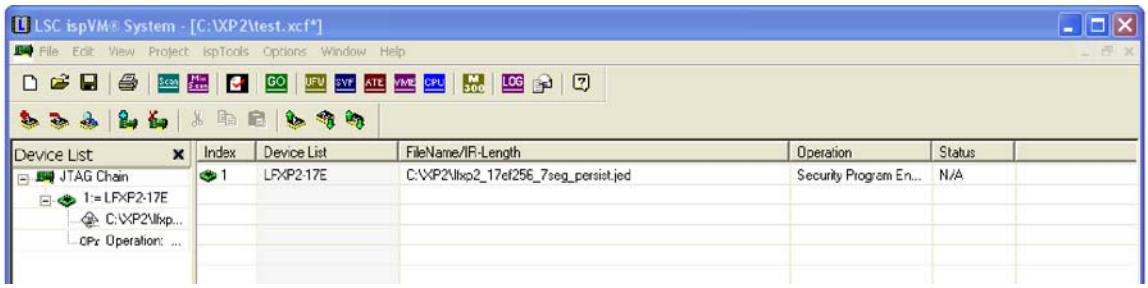
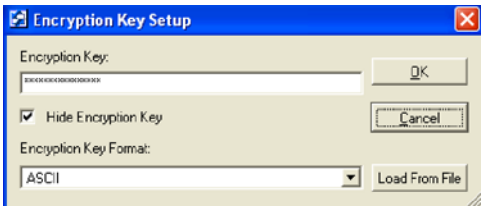


Figure 15-5. ispVM Main Window



2. Double-click on **Input Data File** and browse to the non-encrypted JEDEC file created using ispLEVER. Double-click on **Output Data File** and select an output file name. Right-click on **Encryption** and select **ON**. Right-click on **Encryption Key** and select **Edit Encryption Key**. You will see a window that looks similar to Figure 15-6.

Figure 15-6. Encryption Dialog Window



3. Enter the desired 128-bit encryption key. The key can be entered in Hexadecimal or ASCII. Hex supports 0 through F and is not case sensitive. ASCII supports all printable (ASCII codes 30 through 126) characters. You may also import the 128-bit encryption key from the <Project_Name>.bek generated by ispLEVER. To do so, click on the **Load From File** button shown in Figure 15-6. Click On **OK** to go back to the main ispUFW window.

   Note: Be sure to remember this key, as Lattice cannot recover lost keys.

4. From the ispUFW menu bar click on **Project -> Generate** or the  icon to create the encrypted JEDEC file.

5. Before the LatticeXP2 can configure with the encrypted JEDEC file, the 128-bit encryption key used to encrypt the JEDEC file must be programmed into the LatticeXP2 device.

Programming the Key into the Device

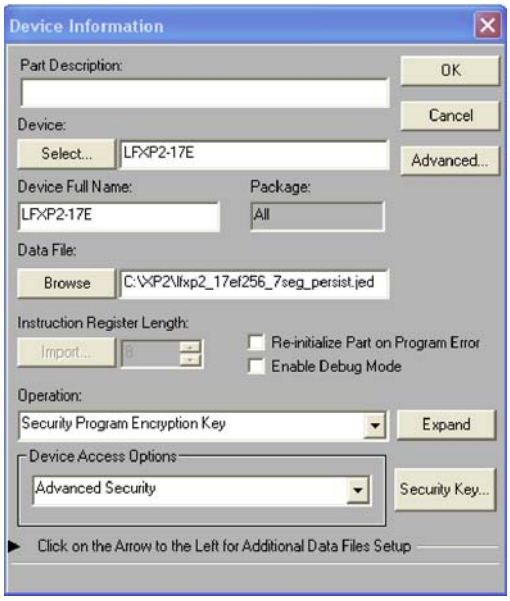
The next step is to program the 128-bit encryption key into the LatticeXP2. Note that this step is separated from JEDEC file encryption to allow flexibility in the manufacturing flow. This flow adds to design security and it allows the user to control over-building of a design. Over-building occurs when a third party builds more boards than are authorized and sells them to grey market customers. If the key is programmed at the factory, then the factory controls the number of working boards that enter the market. The LatticeXP2 will only configure from a file that has been encrypted with the same 128-bit encryption key that is programmed into the LatticeXP2.

To program the 128-bit encryption key into the LatticeXP2, proceed as follows.

1. Start ispVM. You can start ispVM from the **Start -> Programs -> Lattice Semiconductor** menu in Windows or from the ispLEVER GUI. You will see a window that looks similar to Figure 15-5.
2. Attach a Lattice ispDOWNLOAD® cable from a PC to the JTAG connector wired to the LatticeXP2.

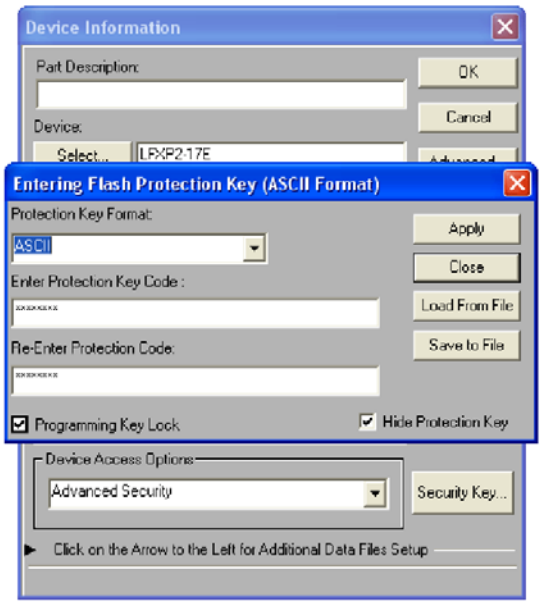
   Note: The 128-bit encryption key can only be programmed into the LatticeXP2 using the JTAG port.
3. Apply power to the board.
4. If the window does not show the board's JTAG chain, then proceed as follows. Otherwise, proceed to step number 5.
   a. Click the SCAN button in the toolbar to find all Lattice devices in the JTAG chain. The chain shown in Figure 15-5 has only one device, the LatticeXP2.

Figure 15-7. ispVM Device information GUI



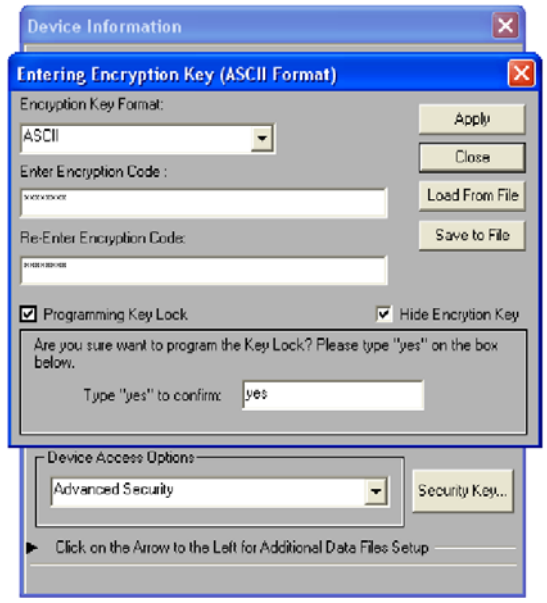
5. Double-click on the line in the chain containing the LatticeXP2. This will open the **Device Information** window (see Figure 15-7). From the **Device Access Options** drop-down box, select **Advanced Security Mode**, then click on the **Security Key** button to the right. The window will look similar to Figure 15-8.

Figure 15-8. ispVM Flash Protection Key



6. Enter the 64-bit Flash Protect key or load it from the file (<Project_Name>.key). You can save this to a file by clicking **Save to File** button. Once you click **Apply**, you will be asked to enter the 128-bit encryption as shown in Figure 15-9.

Figure 15-9. ispVM Encryption Key GUI



7. Now enter the desired 128-bit encryption key. The key can be entered in Hexadecimal or ASCII. Hex supports 0 through F and is not case sensitive. ASCII supports all printable (ASCII codes 30 through 126) characters. This key must be the same as the key used to encrypt the JEDEC file. The LatticeXP2 will only configure from an encrypted JEDEC file whose 128-bit encryption key matches the one loaded into the LatticeXP2.

Note: Be sure to remember this key. Once the Key Lock is programmed, Lattice Semiconductor cannot read back the128-bit encryption key.

a. The 128-bit encryption key can be saved to a file using the **Save to File** button. The 128-bit encryption key will be encrypted using an 8-character file password that the user selects. The name of the file will be **<Project_Name>.bek**. In the future, instead of entering the 128-bit key, simply click on **Load from File** and provide the file password.

8. Programming the Key Lock secures the 128-bit encryption key. When satisfied, type **Yes** to confirm, and then click **Apply**. Once the Key Lock is programmed and the device is power cycled, the 128-bit encryption key cannot be read out of the device.

9. From the main ispVM window (Figure 15-5) click on the green **GO** button on the toolbar to program the 128-bit encryption key into the LatticeXP2. When complete, the LatticeXP2 will only configure from a JEDEC file encrypted with a key that exactly matches the one just programmed.

Security Bit for the Configuration and User Flash (CONFIG_SECURE)

The CONFIG_SECURE setting is located in the GUI setting (Figure 15-3) mentioned in the previous section. After security for the device is selected, NO readback operation is supported through the sysCONFIG port or ispJTAG™ port of the general contents. This is considered the lowest level of security.

Advanced Security Settings

Selecting Advanced Security Settings will enable more security features. One-Time Programmable (OTP), Flash Protect and Encryption as shown in Figure 15-3. These settings are mutually exclusive. Selecting one or the other may nullify other fields that are not required for each particular security settings.

One-Time Programmable (OTP) or Permanent Lock

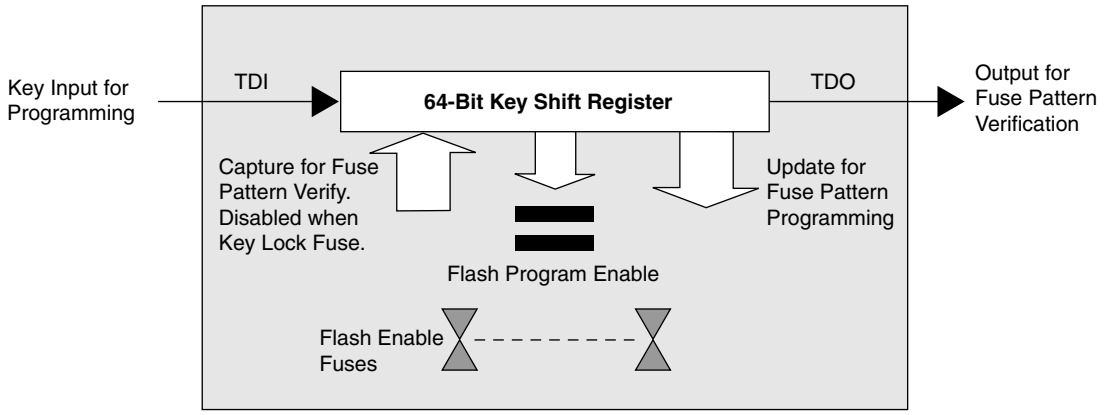
One-Time Programmable (OTP) or permanent lock is another feature that provides the highest level of security. This form of security is currently available only for LatticeXP2 devices. If OTP fuses are programmed it permanently prevents write access to the devices contents. Users must be aware before using this feature that once the OTP is programmed it is not possible to erase or reprogram the device or its security settings. **As the name implies, it is a one-time event only.** Table 15-1 specifies the behavior of the chip when security bit and the OTP bit are programmed.

Table 15-1. Security and OTP Bit Settings

Security CONFIG_SECURE	OTP Bit	Action	Re-Program	Read Back	Erase
0	0	Do nothing	Yes	Yes	Yes
0	1	Inhibit Erase or Programming	No	Yes	No
1	0	Inhibit Readback	Yes	No	Yes
1	1	Inhibit Erase, Programming or Readback	No	No	No

Flash Protect

Figure 15-10. Flash Protect



The next highest level of security for the LatticeXP2 is the 64-bit Flash Protect feature. The 64-bit Flash protect key is used to protect the embedded configuration flash from accidental or unauthorized erasure or reprogramming. This feature does not prevent the device from read back. Therefore, user is given the option to turn ON the CONFIG_SECURE feature.

The default 64-bit Flash protect key is "_LATTICE". Users can also enter their own 64-bit Flash protect key. If there is an existing 64-bit Flash protect key in the Flash Protect key file, the 64-bit Flash protect key can be imported for the Flash Protect key file (<Project_Name>.key). The ispVM GUI will display the Flash Protect key in the same format selected when the 64-bit Flash Protect key was created. Users have the option to change the 64-bit Flash protect key using the default by clicking on the **Default Key** button.

The ispVM software will automatically check the LatticeXP2 device to see if the Flash Protect feature is enabled. If it is, ispVM software will prompt the user to enter the 64-bit Flash Protect key before performing an erase or programming operation. If the 64-bit Flash Protect programmed in the device matches the 64-bit Flash Protect key entered in the ispVM GUI, the device can be erased and reprogrammed. If the keys are lost, the programmed device will be an OTP device. The re-programming of the device requires the user to enter the 64-bit Flash Protect key programmed into the device first. If it matches the 64-bit key (stored in the device), the device will enter the programming mode for erasure and re-programming of the Flash as well as the SRAM fuses.

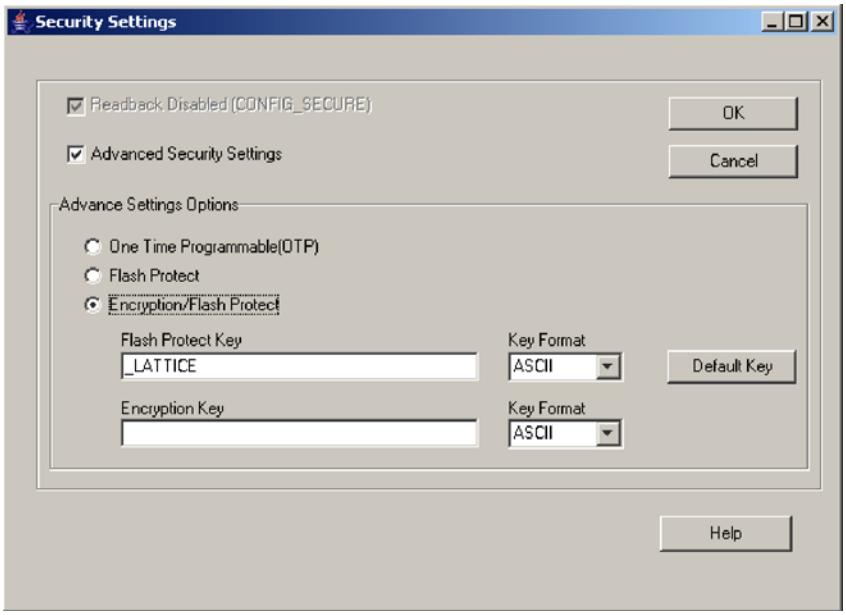
Note: The Flash Protect key cannot be the same as the Encryption Key. The Flash Protect key is 64 bits and the encryption key is 128 bits.

Changing Flash Protect

If the user decides to change the key, it can be done in the key field. The newly created Flash Protect key file (<Project_Name>.key) contains the new 64-bit Flash Protect key and is now ready to be programmed into a device.

The user can also revert back to using the default password by clicking on the **Default** button next to the Flash Protect Key. The default password is **"_LATTICE"**.

Figure 15-11. Encryption/Flash Protect Advanced Feature



Encryption

The LatticeXP2 family of devices uses the 128-bit Advanced Encryption Standard (AES) security mechanism and has a built-in AES decryption engine hardwired in the core and embedded in the device. The JEDEC file must be encrypted with the same 128-bit AES encryption key programmed into the device in order to configure it. The file is shifted into the device's JTAG port using ispVM System software. The device decrypts the JEDEC file using the 128-bit encryption key programmed into the device. The device can only be programmed if the 128-bit encryption key programmed into the device matches the 128-bit encryption key used to encrypt the JEDEC file.

Usercode in Encrypted Files

Note: The usercode is stored as a comment and will be programmed into the device's usercode.

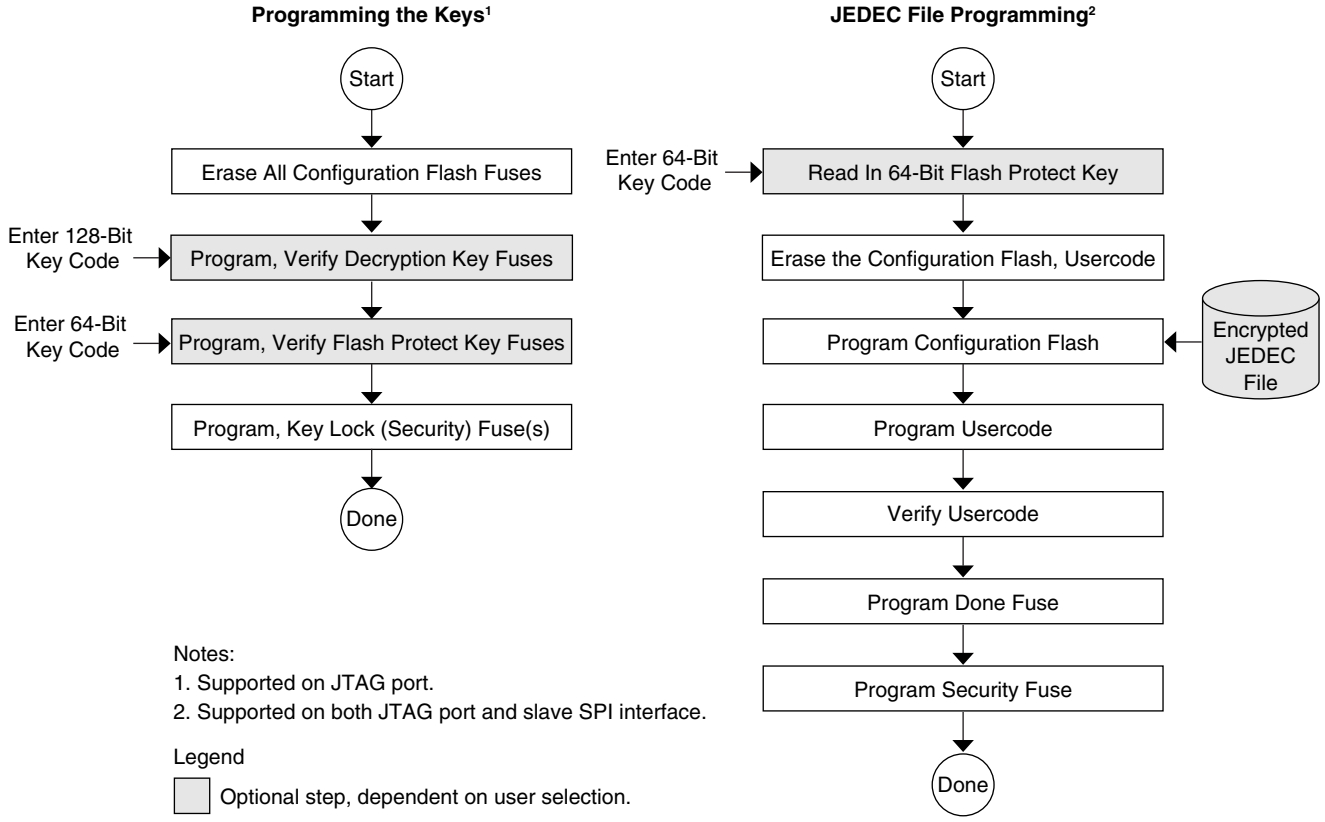
If the USERCODE is used as a custom device ID (MY_ASSP), the U field will still be used in the JEDEC file for the CRC and the value of the Custom Device ID set by the user will be part of the comment field.

The USERCODE will be available for readback regardless of the encryption setting (the USERCODE is always available for readback). Readback of the decrypted JEDEC file from a device through the JTAG port is not permitted because the security fuse is programmed when Encryption/Flash Protect is selected. Encryption will not affect the functionality of the SED

Decryption Flow

The decryption flow is a much simpler process. Start by programming the device with the 128-bit Encryption Key and the 64-bit Flash Protect Key fuses. Once the keys are programmed, the device can then be programmed with the encrypted JEDEC file.

Figure 15-12. Decryption Flow



Verifying a Configuration

If the Flash is programmed directly, the data is first decrypted and then the FPGA performs a CRC on the data. If all CRCs pass, configuration was successful. If a CRC does not pass, the Done fuse is not programmed.

References

- • Lattice Technical Note TN1141, *LatticeXP2 sysCONFIG Usage Guide*
- • Federal Information Processing Standard Publication 197, Nov. 26, 2001. Advanced Encryption Standard (AES)

Technical Support Assistance

Hotline: 1-800-LATTICE (North America)
+1-503-268-8001 (Outside North America)
e-mail: techsupport@latticesemi.com
Internet: www.latticesemi.com

Revision History

Date	Version	Change Summary
February 2007	01.0	Initial release.

LatticeXP2 Soft Error Detection (SED) Usage Guide

January 2008 Technical Note TN1130

Introduction

Soft errors occur when high-energy charged particles alter the stored charge in a memory cell in an electronic circuit. The phenomenon first became an issue in DRAM, requiring error detection and correction for large memory systems in high-reliability applications. As device geometries have continued to shrink, the probability of soft errors in SRAM has become significant for some systems. Designers are using a variety of approaches to minimize the effects of soft errors on system behavior.

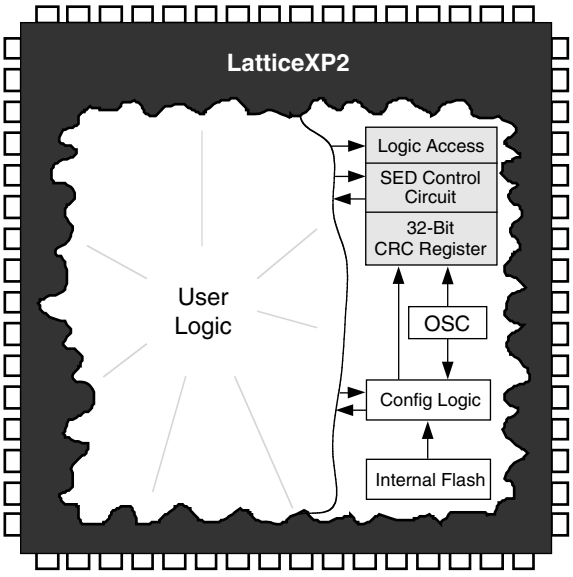
SRAM-based FPGAs store logic configuration data in SRAM cells. As the number and density of SRAM cells in an FPGA increase, the probability that a soft error will alter the programmed logical behavior of the system increases. A number of approaches have been taken to address this issue, but most involve Intellectual Property (IP) cores that the user instantiates into the logic of their design, using valuable resources and possibly affecting design performance. The LatticeXP2 devices have a hardware implemented soft error detector which does not affect performance or heat dissipation of the devices.

This document describes the hardware based soft error detect (SED) approach taken by Lattice Semiconductor for LatticeXP2™ FPGAs.

SED Overview

The SED hardware in the LatticeXP2 devices consists of an access point to FPGA configuration memory, a controller circuit, and a 32-bit register to store the CRC for a given bitstream (see Figure 16-1). The SED hardware reads serial data from the FPGA's configuration memory and calculates a CRC. The data that is read, and the CRC that is calculated, does not include EBR memory or PFUs used as RAM. The calculated CRC is then compared with the expected CRC that was stored in the 32-bit register. If the CRC values match it indicates that there has been no configuration memory corruption, but if the values differ an error signal is generated. SED checking does not impact the performance or operation of the user logic.

Figure 16-1. System Block Diagram[1]



Note that the calculated CRC is based on the particular arrangement of configuration memory for a particular design. Consequently, the expected CRC results cannot be specified until after the design is placed and routed. The ispLEVER® bitstream generation software analyzes the configuration of a placed and routed design and updates the 32-bit SED CRC register contents during bitstream generation.

The following sections describe the LatticeXP2 SED implementation and flow, along with some sample code to get started with.

Basic SED and One-shot SED Modes

Basic SED

Basic SED checks the CRC for all bits. For Basic SED (SEDBA), the inputs are SEDCLKIN, SEDENABLE, SEDSTART, and SEDFRCERR. The output signals are SEDCLKOUT, SEDDONE, SEDINPROG, and SEDERR. The user has no dynamic control over the SED module.

Once an error is detected the SEDERR signal will stay high. SED supports the following Soft Error Corrections (SEC): “Do Nothing”, Auto Reconfigure or on-demand user reconfiguration by pulling the PROGAMN pin low.

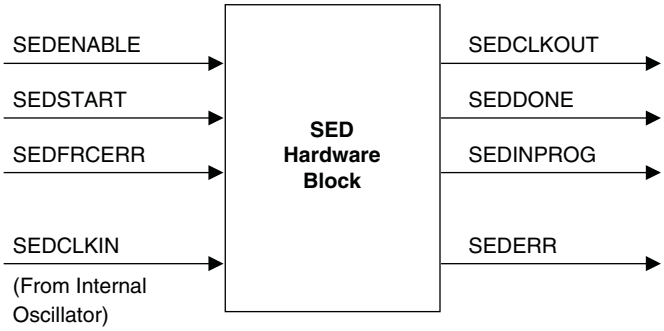
One-Shot SED

The One-Shot SED setting is based on the One-Shot Fuse. The module (SEDBB) has no input ports. The output signals are SEDDONE, SEDINPROG, and SEDERR. At a minimum, the user must connect SEDERR to an I/O pin in order to detect an error.

Hardware Description

As shown in Figure 16-2, the LatticeXP2 SED hardware has several inputs and outputs that allow the user to control, and monitor, SED behavior.

Figure 16-2. Signal Block Diagram



Signal Descriptions

Table 16-1. SED Signal Descriptions

Signal Name	Direction	Active	Description
SEDCLKIN	Input	N/A	Clock
SEDENABLE	Input	High	SED enable
SEDCLKOUT	Output	N/A	Output clock
SEDSTART	Input	High	Start SED cycle
SEDINPROG	Output	High	SED cycle is in progress
SEDDONE	Output	High	SED cycle is complete
SEDFRCERR	Input	High	Force an SED error flag
SEDERR	Output	High	SED error flag

SEDCLKIN

Clock input to the SED hardware.

This clock is derived from the LatticeXP2's on-chip oscillator. The on-chip oscillator's output goes through a divider to create MCCLK. MCCLK goes through another divider to create SEDCLKIN.

The software default for MCCLK is 2.5 MHz, but this can be modified using the MCCLK_FREQ global preference in ispLEVER's pre-map Design Planner (see Lattice technical note TN1141, *LatticeXP2 sysCONFIG Usage Guide,* for supported values of MCCLK). It has a range of 2.5 MHz to 66 MHz.

The divider for SEDCLKIN can be set to 1, 2, 4, 8, 16, 32, 64, 128, or 256. The default is 1, so the default SEDCLKIN frequency is 2.5 MHz. The divider value can be set using a parameter, see the example code at the end of this document.

Note that SEDCLKIN is an internally generated signal, so it should not be included as an input in the user design. See the examples at the end of this document. Also note that while inputs to the SED block are clocked using SEDCLKIN, no attempt has been made to synchronize between clock domains. If this is a concern for a particular design then the designer will need to provide synchronization.

OSC_DIV

Options: 1, 2, 4, 8, 16, 32, 64, 128, or 256. The CLK that drives the SED module will be set by CCLK/OSC_DIV.

SEDENABLE

Active high input to the SED hardware, sampled on the rising edge of SEDCLKIN.

Table 16-2. SEDENABLE

State	Description
1	Enables output of SEDCLKOUT, arms SED hardware.
0	Aborts SED and forces all SED hardware outputs low.

SEDCLKOUT

Gated version of SEDCLKIN, SEDCLKOUT is gated by SEDENABLE.

SEDSTART

Active high input to the SED hardware, sampled on the rising edge of SEDCLKIN.

Table 16-3. SEDSTART

State	Description
1	Start error detection. Must be high a minimum of one SEDCLKIN period.
0	No action.

SEDFRCERR

Active high input to the SED hardware, sampled on the rising edge of SEDCLKIN.

Table 16-4. SEDFRCERR

State	Description
1	Forces SEDERR high, simulating an SED error.
0	No action.

SEDINPROG

Active high output from the SED hardware, clocked out on the rising edge of SEDCLKOUT.

Table 16-5. SEDINPROG

State	Description
1	SED checking is in progress, goes high on the clock following SED-START high.
0	SED checking is not active.

SEDDONE

Active high output from the SED hardware, clocked out on the rising edge of SEDCLKOUT.

Table 16-6. SEDDONE

State	Description
1	SED checking is complete. Reset by a high on SEDSTART or a low on SEDENABLE.
0	SED checking is not complete.

SEDERR

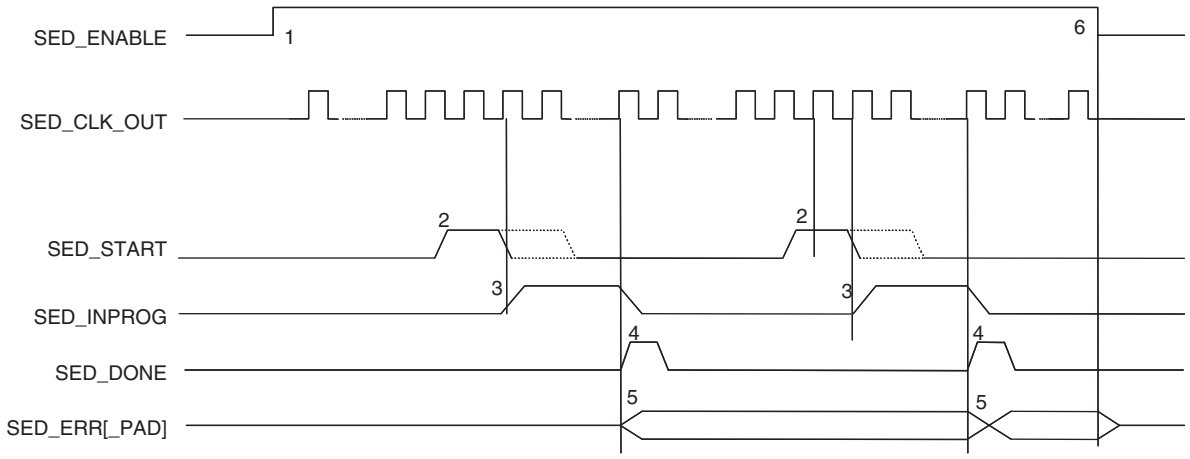
Active high output from the SED hardware, clocked out on the rising edge of SEDCLKOUT.

Table 16-7. SEDERR

State	Description
1	SED has detected an error. Reset by SEDENABLE going low.
0	SED has not detected an error.

SED Flow

Figure 16-3. Timing Diagram

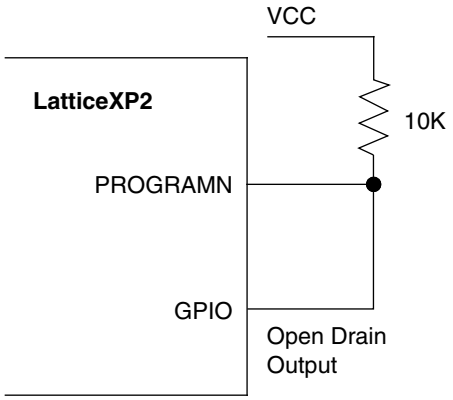


The general SED flow is as follows.

1. User logic sets SEDENABLE high. This signal may be tied high if desired.
2. User logic sets SEDSTART high. SEDINPROG goes high. If SEDDONE is already high it is driven low. SEDSTART may be tied high to enable continuous SED checking.
3. SED starts reading back data from the configuration SRAM.
4. SED finishes checking. SEDERR is updated, SEDINPROG goes low, and SEDDONE goes high.
5. If SEDERR is driven high there are only two ways to reset it, drive SEDENABLE low or reconfigure the FPGA.
6. SEDENABLE goes low when/if the user specifies, and SED is no longer in use.

The user has two choices when an error is detected, ignore the error, and possibly log it, or reconfigure the FPGA. Reconfiguration can be accomplished by driving the PROGRAMN pin low; this can be done with external logic or by wiring one of the FPGA's general purpose I/Os to the PROGRAMN pin and toggling the pin with user logic, perhaps something as simple as inverting SEDERR. If a general purpose I/O is tied to PROGRAMN it is recommended that the I/O Type be set to open drain and an external pull-up resistor be connected to the pin.

Figure 16-4. Example Schematic



SED Run Time

The amount of time needed to perform an SED check depends on the density of the device and the frequency of SEDCLKIN. There will also be some overhead time for calculation, but it is fairly short in comparison. An approximation of the time required can be found by using the following formula:

Maxbits / SEDCLKIN = Time

Maxbits is in mega-bits and depends on the density of the FPGA (see Table 16-8). SEDCLKIN is frequency in MHz. Time is in seconds

For example, for a design using a LatticeXP2 with 5K look-up tables and the SEDCLKIN is the software default of 2.5 MHz:

1.236 Mbits / 2.5 MHz = 494.44 ms

In this example, SED checking will take approximately 496.44 ms. Remember that this happens in the background and does not affect user logic performance.

Note that the internal oscillator used to generate SEDCLKIN can vary by ±30%.

Table 16-8. SED Run Time

Device	XP2-5K	XP2-8K	XP2-17K	XP2-30K	XP2-40K
Density	1.236M	1.954M	3.636M	5.964M	8.304M
66MHz	18.7ms	29.6ms	55.1ms	90.4ms	126.2ms
50MHz	24.7ms	39.1ms	72.7ms	119.3ms	166.1ms
2.5MHz	495ms	782ms	1.455s	2.395s	3.325s

Sample Code

The following simple example code shows how to instantiate the SED. In the example the SED is always on and always running, and the outputs of the SED hardware have been routed to FPGA output pins.

Note that the SEDBA primitive is part of ispLEVER 6.1 or later.

Basic SED VHDL Example

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

entity example is
   port (
         sed_done       : out std_logic;
         sed_in_prog    : out std_logic;
         sed_clk_out    : out std_logic;
         sed_out        : out std_logic);
end;

architecture behavioral of example is

   component  SEDBA -- SED component
      generic (OSC_DIV : string := "1"); -- set SEDCLKIN divider
      port (
          SEDENABLE     : in std_logic;
          SEDSTART      : in std_logic;
          SEDFRCERRN    : in std_logic;
          SEDERR        : out std_logic;
          SEDDONE       : out std_logic;
          SEDINPROG     : out std_logic;
          SEDCLKOUT     : out std_logic) ;
   end component;

   begin

      isnt1: SEDBA
       generic map (OSC_DIV=> "1")
       port map (
          SEDENABLE     => '1',   -- tied high
          SEDSTART      => '1',   -- tied high
          SEDFRCERRN    => '1',  -- tied high
          SEDERR        => sed_out,  -- wired to an output
          SEDDONE       => sed_done,  -- wired to an output
          SEDINPROG     => sed_in_prog,  -- wired to an output
          SEDCLKOUT     => sed_clk_out ) ;    -- wired to an output

end behavioral ;
```

One Shot SED in VHDL

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

entity example is
   port (
        sed_done : out std_logic;
        sed_in_prog : out std_logic;
        sed_out : out std_logic);
end;

architecture behavioral of example is
   component SEDBB -- This is for One Shot SED
     generic (OSC_DIV : string := "1"); -- set SEDCLKIN divider
     port (
         SEDDONE : out std_logic;
         SEDINPROG : out std_logic;
         SEDERR : out std_logic
         );
end component;

begin

     isnt1: SEDBB
     generic map (OSC_DIV=> "1")
     port map (
         SEDERR => sed_out, -- wired to an output
         SEDDONE => sed_done, -- wired to an output
         SEDINPROG => sed_in_prog); -- wired to an output
end behavioral ;
```

Basic SED Verilog Example

```
module example (
   sed_done,
   sed_in_prog,
   sed_clk_out,
   sed_out) ;

output sed_done;
output sed_in_prog;
output sed_clk_out;
output sed_out;

assign V_hi = 1'b1;
assign V_lo = 1'b0;

parameter OSC_DIV = “1”; // set SEDCLKIN divider

   SEDBA sed_ip (
      .SEDENABLE(V_hi), // always high
      .SEDSTART(V_hi),    // always high
      .SEDFRCERRN(V_hi), // always high
      .SEDERR(sed_out),   // wired to an output
      .SEDDONE(sed_done), // wired to an output
      .SEDINPROG(sed_in_prog), // wired to an output
      .SEDCLKOUT(sed_clk_out)); // wired to an output

endmodule
```

One-Shot SED in Verilog

```
module example (
     sed_done,
     sed_in_prog,
     sed_clk_out,
     sed_out) ;

output sed_done;
output sed_in_Prog;
output sed_clk_out;
output sed_out;

assign V_hi = 1'b1;
assign V_lo = 1'b0;

parameter OSC_DIV = "1"; // set SEDCLKIN divider

     SEDBB sed_ip // This is for One Shot SED
          (
          .SEDDONE(sed_done),
          .SEDINPROG(sed_inprog),
          .SEDERR(sed_out)
          );
endmodule

module SEDBB (SEDERR, SEDDONE, SEDINPROG);
    output SEDERR, SEDDONE, SEDINPROG ;
endmodule
```

Technical Support Assistance

Hotline: 1-800-LATTICE (North America)
+1-503-268-8001 (Outside North America)
e-mail: techsupport@latticesemi.com
Internet: www.latticesemi.com

Revision History

Date	Version	Change Summary
February 2007	01.0	Initial release.
December 2007	01.1	Updated code in the following sections: Basic SED VHDL Example, One Shot SED in VHDL, Basic SED Verilog Example, One Shot SED in Verilog.

LatticeXP2 Dual Boot Usage Guide

May 2007 Technical Note TN1144

Introduction

Complementing its internal Flash configuration memory, the LatticeXP2™ also provides support for inexpensive SPI Flash devices. This provides the ability to use an alternate or backup bitstream, referred to as the “golden” image. The device always attempts to load the primary image from the selected source. Should any unexpected interrupts occur during configuration of the primary image, the LatticeXP2 device will automatically switch sources and configure from the golden image location.

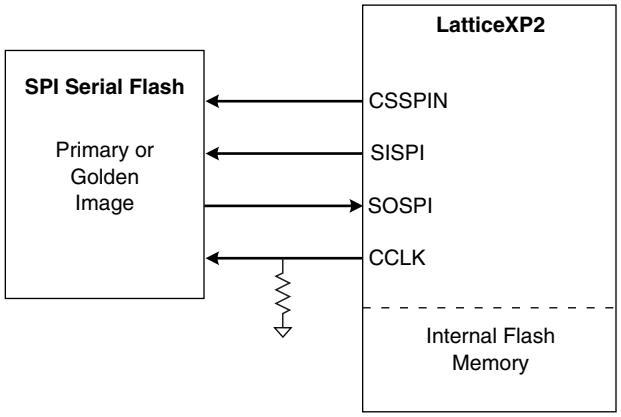
Dual Boot Mode

The LatticeXP2 Dual Boot sysCONFIG™ mode is selected using CFG pin settings. Table 17-1 lists the sysCONFIG modes supported by the LatticeXP2 device family. Figure 17-1 illustrates the SPI Flash hardware connections.

Table 17-1. LatticeXP2 sysCONFIG Modes

CFG1	CFG0	Configuration Mode	Primary Boot Source	Secondary Boot Source
0	0	Dual Boot	External SPI Flash	Internal Flash
1	0		Internal Flash	External SPI Flash
X	1	Self Download Mode (SDM)	Internal Flash	None

Figure 17-1. LatticeXP2 Hardware Connections to SPI Flash



Internal logic is used to detect a configuration failure from the primary source and provides the ability to reattempt configuration from the secondary source. This sequence is used when the LatticeXP2 is set to dual boot mode and configuration is initiated.

Configuration initiates in dual boot mode when any of the following events occur:

- The device is powered-up with all supplies reaching their required minimum values
- The PROGRAMN pin is toggled
- The REFRESH command is issued via the ispJTAG™ port

Should configuration from both primary and golden images in dual boot mode fail, the INITN pin will be driven low and the configuration process will halt.

Dual Boot Flash Programming

In order to use dual configuration images, the data must be programmed for storage within the corresponding Flash locations separately. ispVM® System software provides the ability to program both LatticeXP2 internal Flash and supported external SPI Flash memory devices.

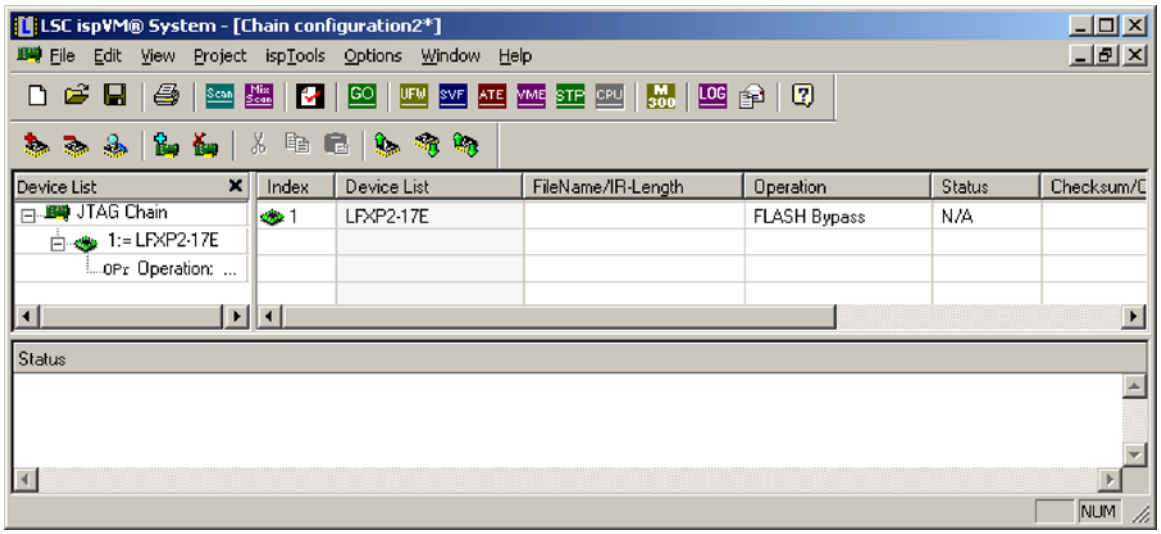
Note: To allow programming of the external SPI Flash device, the LatticeXP2 CFG0 pin should be low.

Procedure

Note: This procedure assumes basic familiarity with ispVM System. For more information on using ispVM System, refer to the tutorials and contents within the help system.

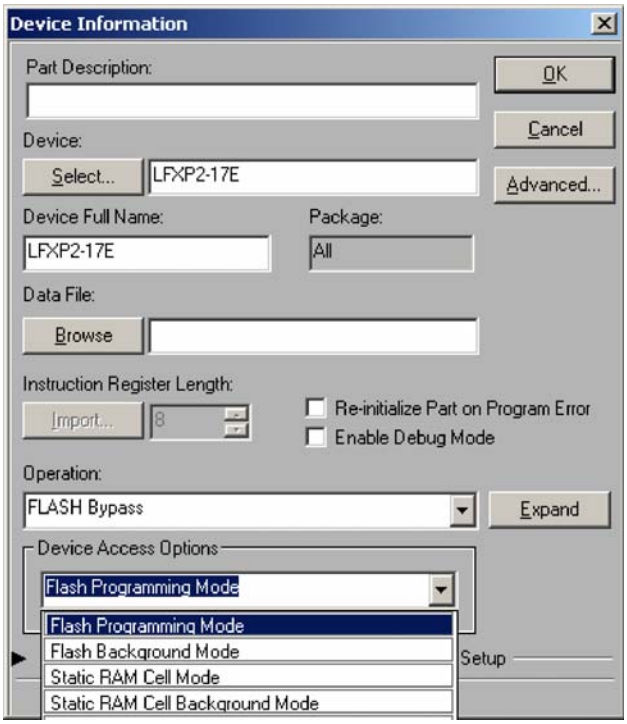
1. Using ispLEVER®, create JEDEC files for the LatticeXP2 device to be used as primary and golden images. To preserve the sysCONFIG port, the PERSISTENT option in the ispLEVER Design Planner must be set to ON.
2. Open ispVM System.
3. Scan the chain or manually insert the devices representing the JTAG chain. An example chain is shown in Figure

Figure 17-2. ispVM System



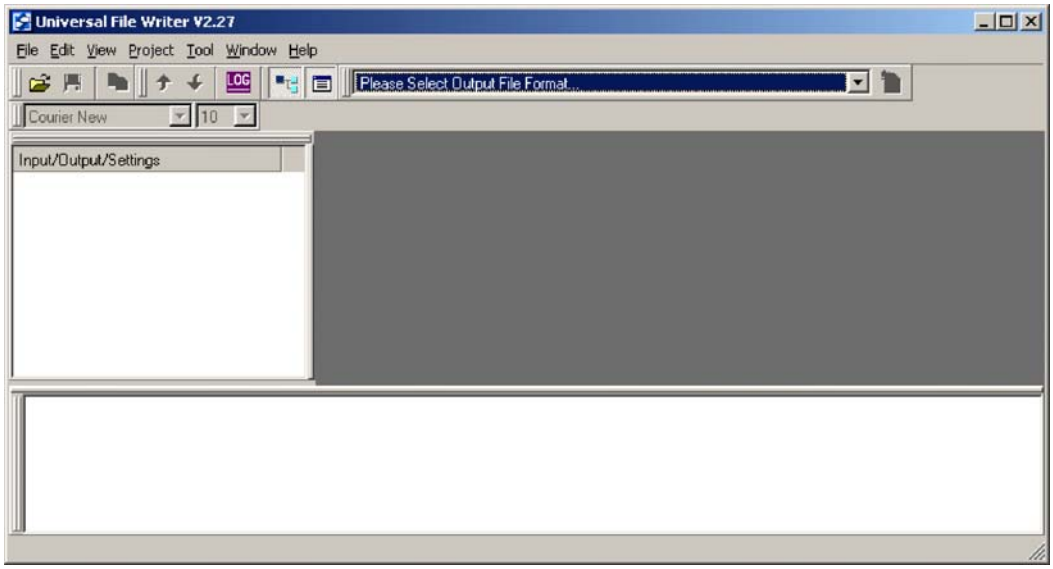
4. Double-click the LatticeXP2 device to open the Device Information Window, as shown in Figure 17-3, and select **Flash Programming Mode** from **Device Access Options**.

Figure 17-3. Device Information Window



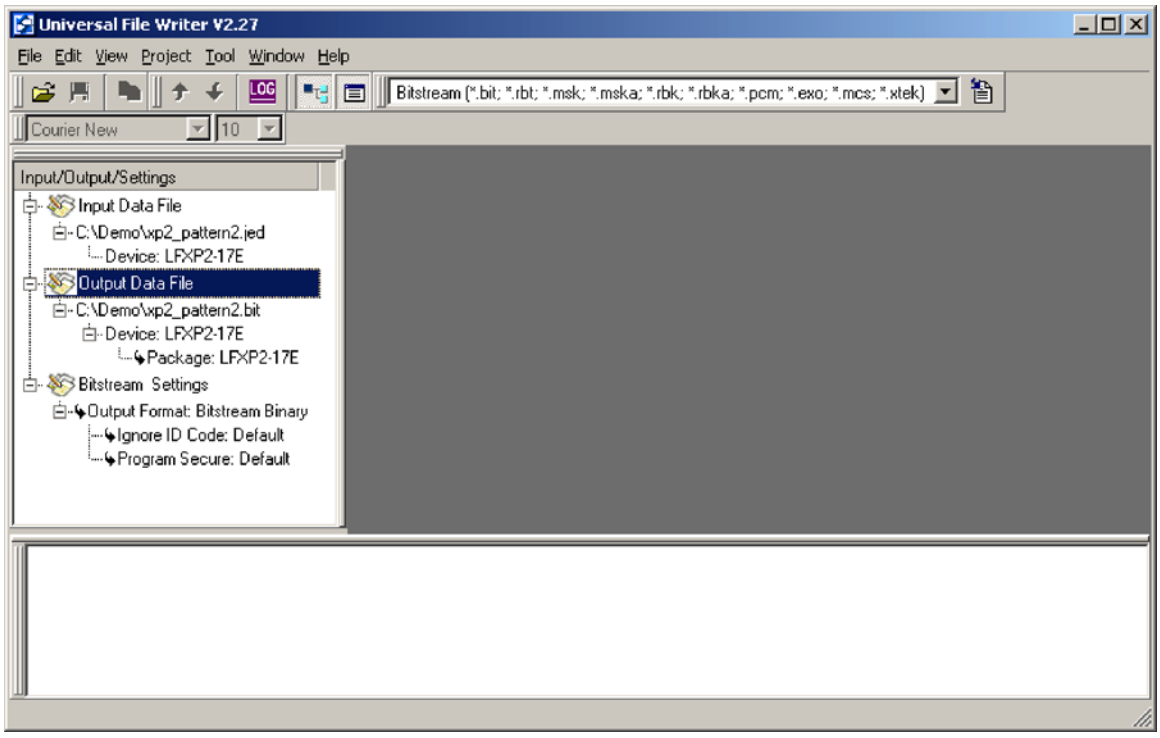
5. Under **Data File**, browse to the appropriate JEDEC file to be programmed into the internal Flash memory.
6. Set the **Operation** field to **Flash Erase, Program, Verify**.
7. Press **OK** to return to the main ispVM System window.
8. Press the green **GO** button to download the data file into the LatticeXP2 internal Flash memory.
9. To prepare a data file to program the SPI Flash, open the Universal File Writer tool by clicking the **UFW** button from the ispVM System toolbar. A window will appear, as shown in Figure 17-4.

Figure 17-4. Universal File Writer



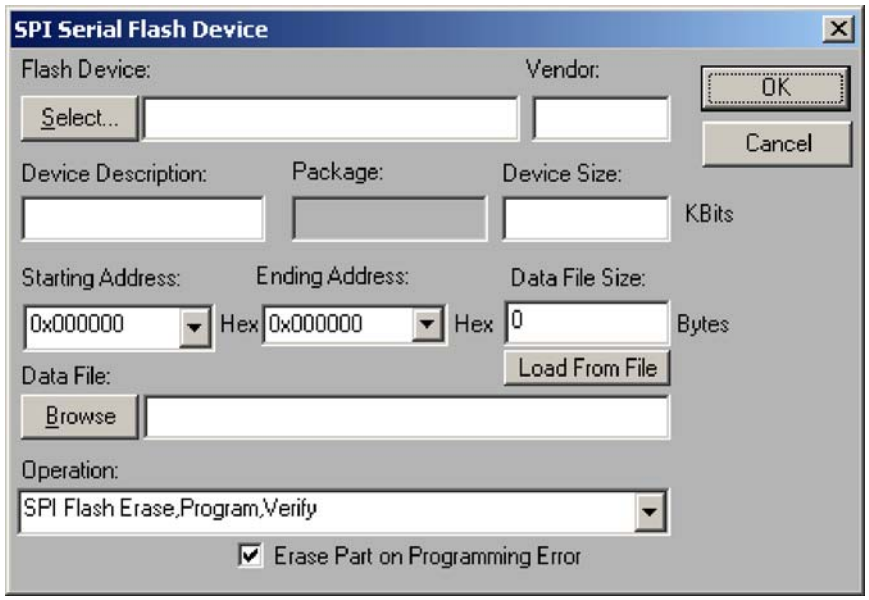
10. Specify a bitstream file output by pulling down the selector from the toolbar.
11. Double-click **Input Data File** in the left-hand windowpane. Browse to the appropriate ispLEVER-created JEDEC file.
12. Double-click **Output Data File** in the left-hand windowpane. Select the desired path and name of the file to be created. The UFW window should appear similar to Figure 17-5.

Figure 17-5. UFW Settings for JEDEC to Bitstream Conversion



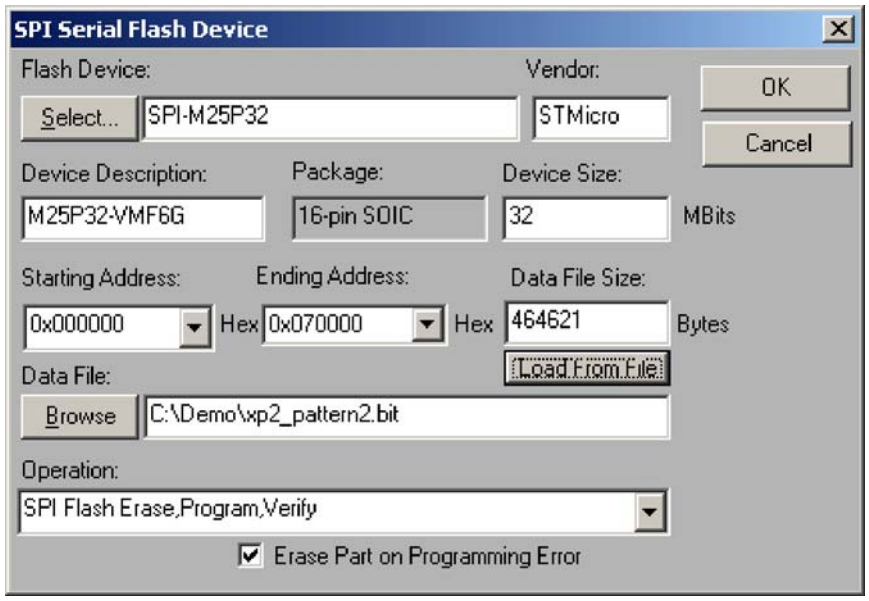
13. Press the **Generate** button or choose **Project->Generate**. The bitstream file will then be created.
14. Close the UFW tool and return to ispVM System.
15. Double-click the device in ispVM System to bring up the device settings dialog.
16. Change the **Device Access Options** to **SPI Flash Programming**. This will bring up another dialog, as shown in Figure 17-6.

Figure 17-6. SPI Serial Flash Device Selection



17. Press **Select** to choose from a list of supported SPI Flash devices.
18. Under **Data File**, browse to the appropriate bitstream file to be programmed into the external SPI Flash memory. The results should be similar to Figure 17-7.

Figure 17-7. Completed SPI Serial Flash Device Settings



19. Press **OK** to exit the SPI Serial Flash Device setup window.
20. At the main ispVM System window, the LatticeXP2 device is now ready to program the SPI Flash device. Click the green **GO** button on the toolbar to execute this process.

Provided the LatticeXP2 CFG pins are set to one of the dual boot modes, the device is now configured to use the corresponding dual boot capability. Upon toggling the PROGRAMN pin or power cycling, the device will load the primary image.

Technical Support Assistance

Hotline: 1-800-LATTICE (North America)
+1-503-268-8001 (Outside North America)
e-mail: techsupport@latticesemi.com
Internet: www.latticesemi.com

Revision History

Date	Version	Change Summary
February 2007	01.0	Initial release.
May 2007	01.1	Added note to Dual Boot Flash Programming section.
		Updated step 1 in the Procedure section.

Section III. LatticeXP2 Family Handbook Revision History

LatticeXP2 Family Handbook Revision History

January 2008 **Handbook HB1004**

Revision History

Date	Handbook Revison Number	Change Summary
May 2007	01.1	Initial release.
July 2007	01.2	Technical note TN1137 updated to version 01.1.
September 2007	01.3	LatticeXP2 Family Data Sheet updated to version 01.2.
January 2008	01.4	Technical note TN1130 updated to version 01.1.
		Technical note TN1137 updated to version 01.3.
		Technical note TN1141 updated to version 01.2.

Note: For detailed revision changes, please refer to the revision history for each document.