LatticeECP2 Family Handbook

Version 01.0, February 2006

LatticeECP2 Family Handbook Table of Contents

February 2006

Section I. LatticeECP2 Family Data Sheet

LatticeECP2 sysCONFIG Usage Guide

Section I. LatticeECP2 Family Data Sheet

Version 01.0, February 2006

LatticeECP2 Family Data Sheet Introduction

February 2006 **Advance Data Sheet**

Features

- **High Logic Density for System Integration**
  - 6K to 68K LUTs
  - 192 to 628 I/Os
- **sysDSP™ Block**
  - 3 to 22 blocks for high performance multiply and accumulate
  - 12 to 88 18x18 multipliers
  - Each block supports
    - One 36x36 multiplier or four 18X18 or eight 9X9 multipliers
- **Flexible Memory Resources**
  - 55Kbits to 1032Kbits sysMEM™ Embedded Block RAM (EBR) 18-Kbit block
    - Single, pseudo dual and true dual port
  - 12K to 136Kbits distributed RAM
    - Single port and pseudo dual port
- **sysCLOCK Analog PLLs And DLLs**
  - Two GPLLs and up to four SPLLs per device
    - Clock multiply, divide, phase adjust and delay adjust
  - Two general purpose DLLs per device
- **Pre-Engineered Source Synchronous I/O**
  - Dedicated DQS support
  - DDR registers in I/O cells
  - Dedicated gearing logic
  - Source synchronous standards support
    - SPI4.2, SFI4, XGMII
    - High Speed ADC/DAC devices
  - Dedicated DDR and DDR2 memory support
    - DDR1 400 (200MHz)
    - DDR2 400 (200MHz)
- **Programmable sysIO™ Buffer Supports Wide Range Of Interfaces**
  - LVTTL and LVCMOS 33/25/18/15/12
  - SSTL 3/2/18 I, II
  - HSTL15 I and HSTL18 I, II
  - PCI and Differential HSTL, SSTL
  - LVDS, RSDS, Bus-LVDS, MLVDS, LVPECL
- **Flexible Device Configuration**
  - 1149.1 Boundary Scan compliant
  - Dedicated bank for configuration for I/Os
  - SPI boot flash interface
  - Dual boot images supported
  - Encrypted bit stream support
  - TransFR™ I/O for simple field updates
  - Optional Soft Error Detect macro embedded
- **Optional Bitstream Encryption**
- **System Level Support**
  - ispTRACY™ internal logic analyzer capability
  - Onboard oscillator for initialization and general use
  - 1.2V power supply

Table 1-1. LatticeECP2 Family Selection Guide

Device	ECP2-6	ECP2-12	ECP2-20	ECP2-35	ECP2-50	ECP2-70
LUTs (K)	6	12	21	32	48	68
Distributed RAM (Kbits)	12	24	42	64	96	136
EBR SRAM (Kbits)	55	221	276	332	387	1032
EBR SRAM Blocks	3	12	15	18	21	56
sysDSP Blocks	3	6	7	8	18	22
18x18 Multipliers	12	24	28	32	72	88
GPLL + SPLL + GDLL	2+0+2	2+0+2	2+0+2	2+0+2	2+2+2	2+4+2
Maximum Available I/O	192	297	363	452	500	628
Packages and I/O Combinations						
144-pin TQFP (20 x 20 mm)	95	95				
208-pin PQFP (28 x 28 mm)		127	127			
256-ball fpBGA (17 x 17 mm)	192	192	192			
484-ball fpBGA (23 x 23 mm)		297	332	332	339	
672-ball fpBGA (27 x 27 mm)			363	452	500	500
900-ball fpBGA (31 x 31 mm)						628

Introduction

The LatticeECP2 family of FPGA devices has been optimized to deliver high performance features such as advanced DSP blocks and high speed source synchronous interfaces in an economical FPGA fabric. This combination was achieved through advances in device architecture and the use of 90nm technology.

The LatticeECP2 FPGA fabric was optimized for the new technology from the outset with high performance low cost in mind. The LatticeECP2 devices include LUT-based logic, distributed and embedded memory, Phase Locked Loops (PLLs), Delay Locked Loops (DLLs), pre-engineered source synchronous I/O support, enhanced sysDSP blocks and advanced configuration support, including encryption and dual-boot capabilities.

The ispLEVER® design tool from Lattice allows large complex designs to be efficiently implemented using the LatticeECP2 family of FPGA devices. Synthesis library support for LatticeECP2 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 LatticeECP2 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 IP (Intellectual Property) ispLeverCORE™ modules for the LatticeECP2 family. By using these IPs as standardized blocks, designers are free to concentrate on the unique aspects of their design, increasing their productivity.

LatticeECP2 Family Data Sheet Architecture

February 2006 Advance Data Sheet

Architecture Overview

Each LatticeECP2 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 rows of sysDSP™ Digital Signal Processing blocks as shown in Figure 2-1.

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 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.

The LatticeECP2 family of devices contain up to two rows of sysMEM EBR blocks. sysMEM EBRs are large dedicated 18K fast memory blocks. Each sysMEM block can be configured in variety of depths and widths of RAM or ROM. In addition, LatticeECP2 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

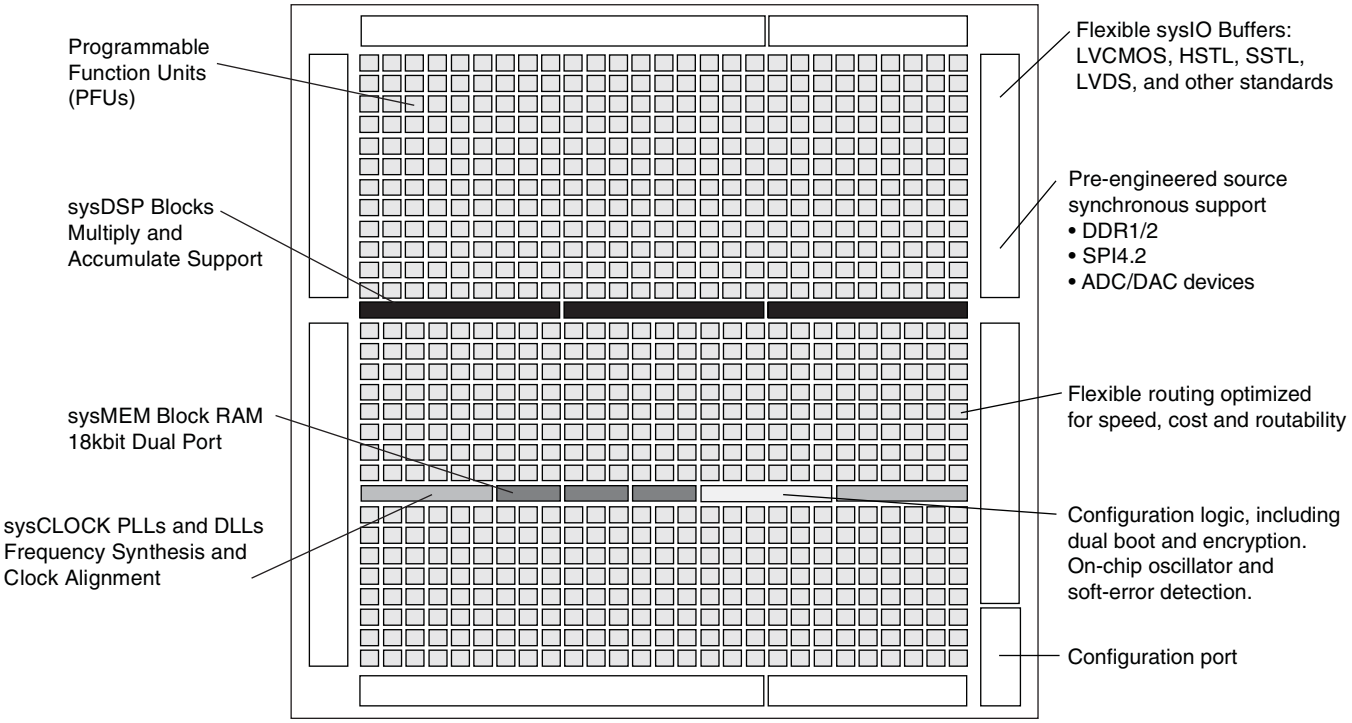
Each PIC block encompasses two PIOs (PIO pairs) with their respective sysIO buffers. The sysIO buffers of the LatticeECP2 devices are arranged into eight banks, allowing the implementation of a wide variety of I/O standards. In addition, a separate I/O bank is provided for the programming interfaces. 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 the high speed source synchronous standards such as SPI4.2 along with memory interfaces including DDR2.

Other blocks provided include PLLs, DLLs and configuration functions. The LatticeECP2 architecture provides two General PLLs (GPLL) and up to four Standard PLLs (SPLL) per device. In addition, each LatticeECP2 family member provides two DLLs per device. The GPLLs and DLLs blocks are located in pairs at the end of the bottom-most EBR row; the DLL block located towards the edge of the device. The SPLL blocks are located at the end of the other EBR/DSP rows.

The configuration block that supports features such as configuration bit-stream de-encryption, transparent updates and dual boot support is located toward the center of this EBR row. Every device in the LatticeECP2 family supports a sysCONFIG™ port located in the corner between banks four and five, which allows for serial or parallel device configuration.

In addition, every device in the family has a JTAG port. This family also provides an on-chip oscillator and soft error detect capability. The LatticeECP2 devices use 1.2V as their core voltage.

Figure 2-1. Simplified Block Diagram, ECP2-6 Device (Top Level)

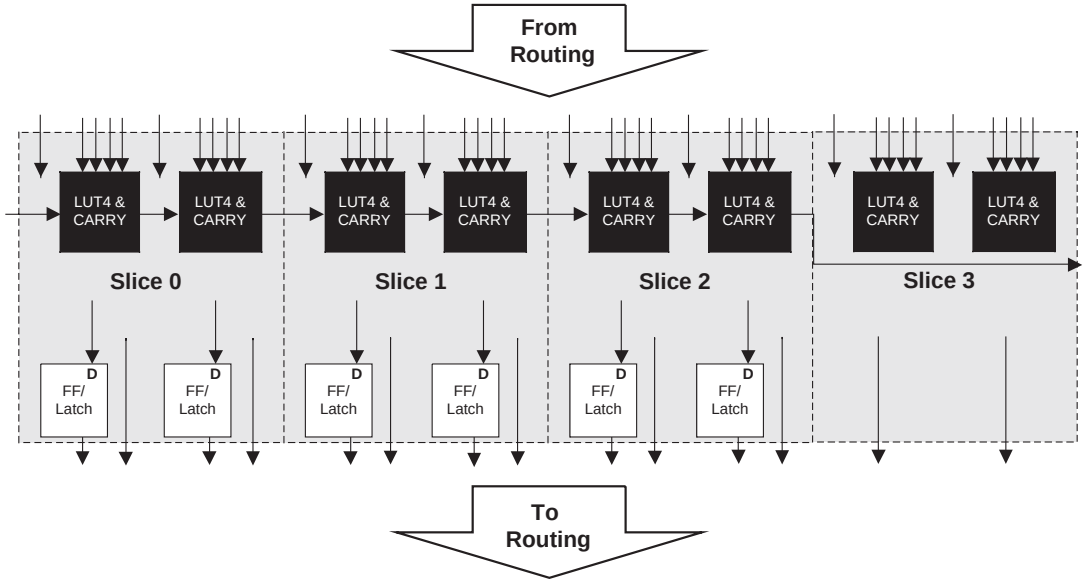


PFU Blocks

The core of the LatticeECP2 device consists of PFU blocks which are provided in two forms, the PFU and PFF. The 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 0-3 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

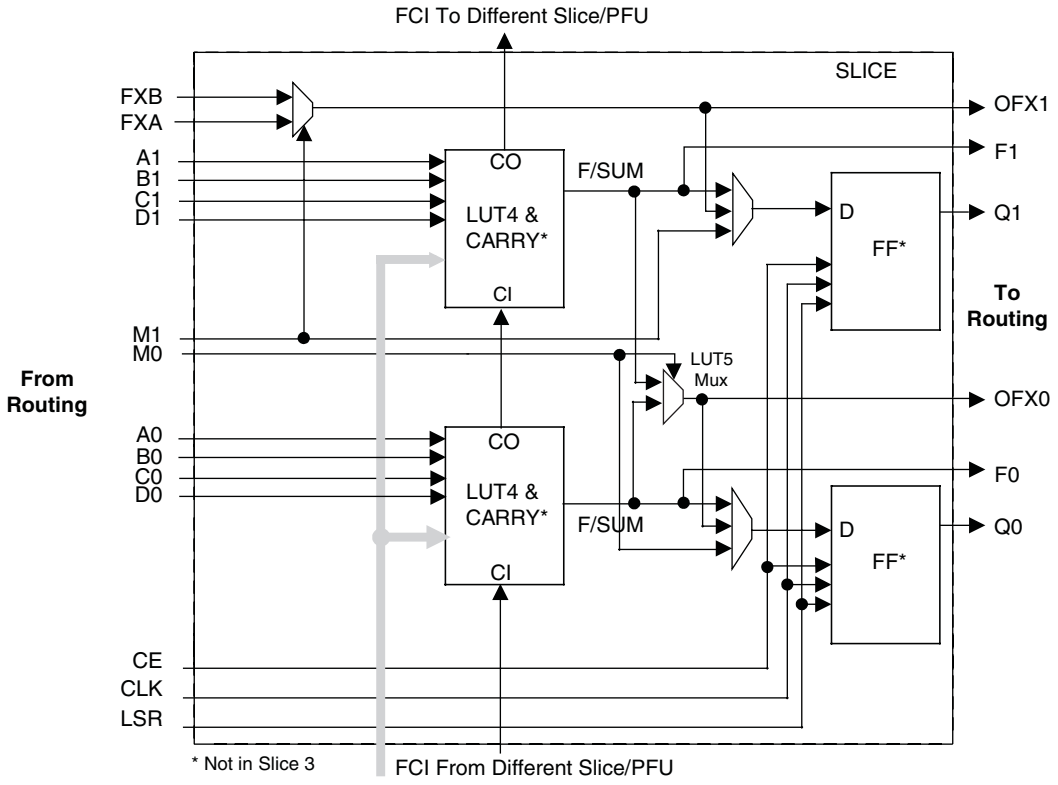
Slice 0 through Slice 2 contain two LUT4s feeding two registers, whereas Slice 3 contains two LUT4s only. For PFUs, Slice 0 and Slice 2 can also be configured as distributed memory, a capability not available in the PFF. Table 2-1 shows the capability of the slices in both PFF and PFU blocks along with the operation modes they enable. In addition, each PFU contains some 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

Slices 0, 1 and 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	FCIN	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 4-input combinatorial lookup tables. A LUT4 can have 16 possible input combinations. Any four input logic functions can be generated by programming this lookup table. Since there are two LUT4s per slice, a LUT5 can be constructed within one slice. Larger look-up tables such as LUT6, LUT7 and LUT8 can be constructed by concatenating other slices. Note LUT8 requires more than four slices.

Ripple Mode

Ripple mode allows the 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 additional signals: Carry Generate and Carry Propagate 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 LatticeECP2 devices, please see details of additional technical documentation at the end of this data sheet.

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 LatticeECP2 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) and x6 (spans seven PFU). The x1 and x2 connections provide fast and efficient connections in horizontal and vertical directions. The x2 and x6 resources are buffered allowing both short and long connections routing between PFUs.

The LatticeECP2 family has an enhanced routing architecture that produces 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 (GPLL/SPLL)

The sysCLOCK PLLs provide the ability to synthesize clock frequencies. All the devices in the LatticeECP2 family support two General Purpose PLLs (GPLLs) which are full-featured PLLs. In addition, some of the larger devices have two to four Standard PLLs (SPLLs) that have a subset of GPLL functionality.

General Purpose PLL (GPLL)

The architecture of the GPLL is shown in Figure 2-4. A description of the GPLL functionality follows.

CLKI is the reference frequency (generated either from the pin or from routing) for the PLL. CLKI feeds into the Input Clock Divider block. The CLKFB is the feedback signal (generated from CLKOP or from a user clock PIN/logic). This signal feeds into the Feedback Divider. The Feedback Divider is used to multiply the reference frequency.

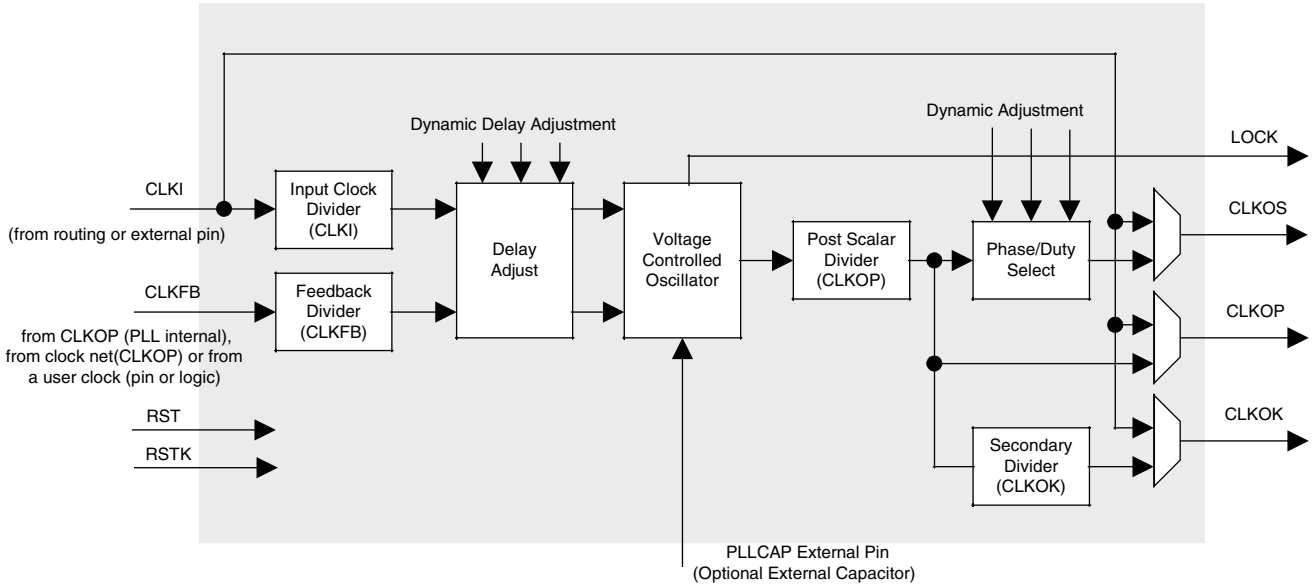
The Delay Adjust Block adjusts either the delays of the reference or feedback signals. The Delay Adjust Block can either be programmed during configuration or can be adjusted dynamically. The setup, hold or clock-to-out times of the device can be improved by programming a delay in the feedback or input path of the PLL which will advance or delay the output clock with reference to the input clock.

Following the Delay Adjust Block, both the input path and feedback signals enter the Voltage Controlled Oscillator (VCO) block. In this block the difference between the input path and feedback signals is used to control the frequency and phase of the oscillator. A LOCK signal is generated by the VCO to indicate that VCO has locked onto the input clock signal. In dynamic mode, the PLL may lose lock after a dynamic delay adjustment and not relock until the t_{LOCK} parameter has been satisfied. LatticeECP2 devices have two dedicated pins on the left and right edges of the device for connecting optional external capacitors to the VCO. This allows the PLLs to operate at a lower frequency. This is a shared resource which can only be used by one PLL (GPLL or SPLL) per side.

The output of the VCO then enters the post-scalar divider. The post-scalar divider allows the VCO to operate at higher frequencies than the clock output (CLKOP), thereby increasing the frequency range. A secondary divider takes the CLKOP signal and uses it to derive lower frequency outputs (CLKOK). The Phase/Duty Select block adjusts the phase and duty cycle of the CLKOP signal and generates the CLKOS signal. The phase/duty cycle setting can be pre-programmed or dynamically adjusted.

The primary output from the post scalar divider CLKOP along with the outputs from the secondary divider (CLKOK) and Phase/Duty select (CLKOS) are fed to the clock distribution network.

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



Standard PLL (SPLL)

Some of the larger devices have two to four Standard PLLs (SPLLs). SPLLs have the same features as GPLLs but without delay adjustment capability. SPLLs also provide different parametric specifications. For more information, please see details of additional technical documentation at the end of this data sheet.

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

Table 2-4. GPLL and SPLL Blocks 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
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
LOCK	O	"1" indicates PLL LOCK to CLKI
DDAMODE[1]	I	Dynamic Delay Enable. "1": Pin control (dynamic), "0": Fuse Control (static)
DDAIZR[1]	I	Dynamic Delay Zero. "1": delay = 0, "0": delay = on
DDAILAG[1]	I	Dynamic Delay Lag/Lead. "1": Lead, "0": Lag
DDAIDEL[2:0][1]	I	Dynamic Delay Input
DPA MODES	I	DPA (Dynamic Phase Adjust/Duty Cycle Select) mode
DPHASE [3:0]	I	DPA Phase Adjust inputs
DDDUTY [3:0]	—	DPA Duty Cycle Select inputs

1. These signals are not available in SPLL.

Delay Locked Loops (DLL)

In addition to PLLs, the LatticeECP2 family of devices has two DLLs per device.

CLKI is the input frequency (generated either from the pin or routing) for the DLL. CLKI feeds into the output muxes block to bypass the DLL, directly to the DELAY CHAIN block and (directly or through divider circuit) to the reference input of the Phase Frequency Detector (PFD) input mux. The reference signal for the PFD can also be generated from the Delay Chain and CLKFB signals. The feedback input to the PFD is generated from the CLKFB pin, CLKI or from tapped signal from the Delay chain.

The PFD produces a binary number proportional to the phase and frequency difference between the reference and feedback signals. This binary output of the PFD is feed into a Arithmetic Logic Unit (ALU). Based on these inputs, the ALU determines the correct digital control codes to send to the delay chain in order to better match the reference and feedback signals. This digital code from the ALU is also transmitted via the Digital Control bus (DCNTL) bus to its associated DLL_DEL delay block. The ALUHOLD input allows the user to suspend the ALU output at its current value. The UDDCNTL signal allows the user to latch the current value on the DCNTL bus.

The DLL has two independent clock outputs, CLKOP and CLKOS. These outputs can individually select one of the outputs from the tapped delay line. The CLKOS has optional fine phase shift and divider blocks to allow this output to be further modified, if required. The fine phase shift block allows the CLKOS output to phase shifted a further 45, 22.5 or 11.25 degrees relative to its normal position. Both the CLKOS and CLKOP outputs are available with optional duty cycle correction. Divide by two and divide by four frequencies are available at CLKOS. The LOCK output signal is asserted when the DLL is locked. Figure 2-5 shows the DLL block diagram and Table 2-5 provides a description of the DLL inputs and outputs.

The sysCLOCK DLL is configured at power-up and, if desired, can be reconfigured dynamically through the Serial Memory Interface (SMI) bus. Users can drive the SMI interface from routing using Lattice software tools.

The user can configure the DLL for many common functions such as time reference delay mode and clock injection removal mode. Lattice provides primitives in its design tools for these functions. For more information on the DLL, please see details of additional technical documentation at the end of this data sheet.

Figure 2-5. Delay Locked Loop Diagram (DLL)

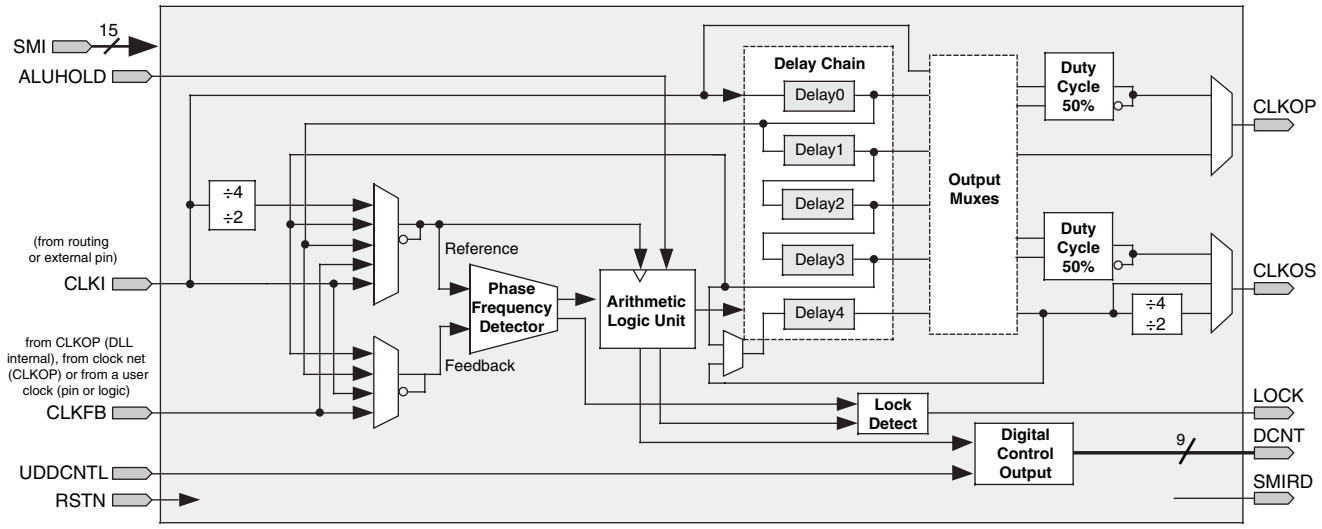


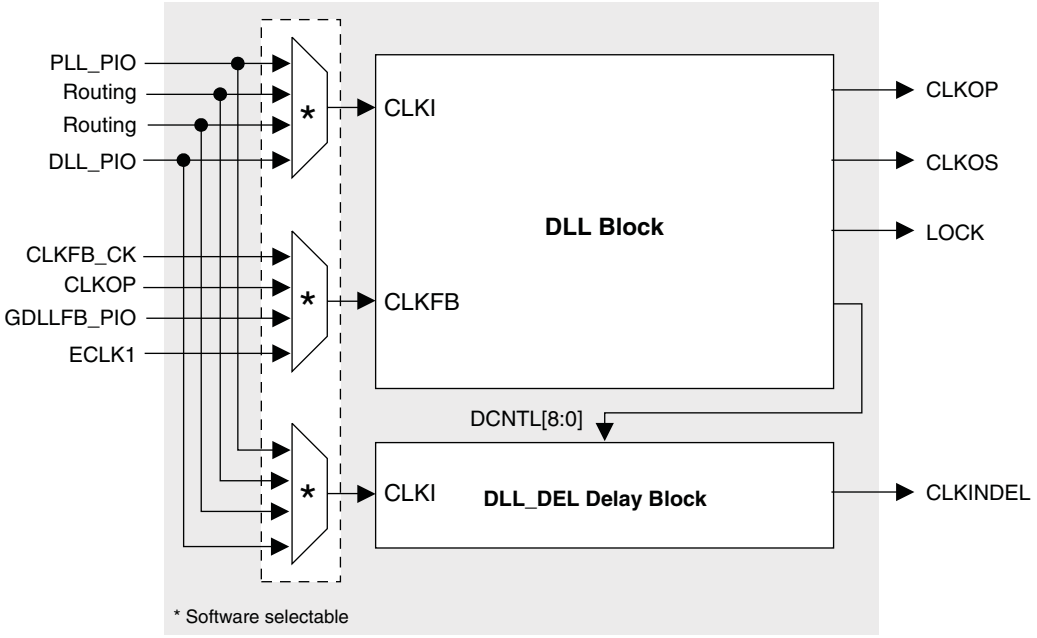
Table 2-5. DLL Signals

Signal	I/O	Description
CLKI	I	Clock input from external pin or routing
CLKFB	I	DLL feed input from DLL output, clock net, routing or external pin
RSTN	I	Active low synchronous reset
ALUHOLD	I	Active high freezes the ALU
UDDCNTL	I	Synchronous enable signal (hold high for two cycles) from routing
DCNTL[8:0]	O	Encoded digital control signals for PIC INDEL and slave delay calibration
CLKOP	O	The primary clock output
CLKOS	O	The secondary clock output with fine phase shift and/or division by 2 or by 4
LOCK	O	Active high phase lock indicator
SMIADDR[9:0]	I	SMI Address
SMICLK	I	SMI Clock
SMIRSTN	I	SMI Reset (Active low)
SMIRD	I	SMI Read
SMIWDATA	I	SMI Write Data
SMIWR	I	SMI Write
SMIRDATA	O	SMI Read Data

DLL_DEL Delay Block

Closely associated with each DLL is a DLL_DEL block. This is a delay block consisting of a delay line with taps and a selection scheme that selects one of the taps. The DCNTL[8:0] bus controls the delay of the CLKINDEL signal. Typically this is the delay setting that the DLL uses to achieve phase alignment. This results in the delay providing a calibrated 90° phase shift that is useful in centering a clock in the middle of a data cycle for source synchronous data. Note that it is possible to make small adjustments to the delay by programming registers available via the SMI bus. The CLKINDEL signal feeds the edge clock network. Figure 2-6 shows the connections between the DLL block and the DLL_DEL delay block. For more information, please see details of additional technical documentation at the end of this data sheet.

Figure 2-6. DLL_DEL Delay Block



PLL/DLL Cascading

LatticeECP2 devices have been designed to allow certain combinations of PLL (GPLL and SPLL) and DLL cascading. The allowable combinations are as follows:

- PLL to PLL supported
- PLL to DLL supported

The DLLs in the LatticeECP2 are used to shift the clock in relation to the data for source synchronous inputs. PLLs are used for frequency synthesis and clock generation for source synchronous interfaces. Cascading PLL and DLL blocks allows applications to utilize the unique benefits of both DLLs and PLLs.

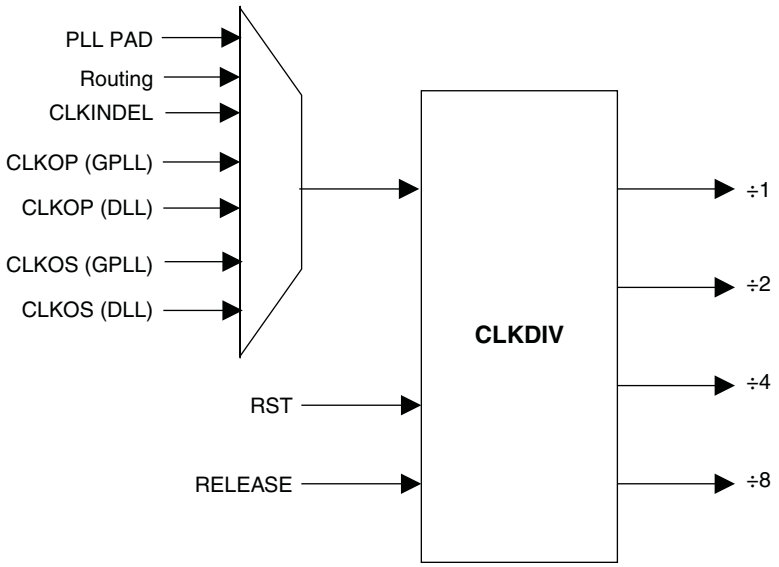
For further information on the DLL, please see details of additional technical documentation at the end of this data sheet.

Clock Dividers

LatticeECP2 devices have two clock dividers on the left and right sides 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 selected PLL/DLL outputs, DLL_DEL delay blocks, routing or from an external clock input. The clock divider outputs serve as primary clock sources and feed into the

clock distribution network. The Reset (RST) control signal resets input and synchronously forces all outputs to low. The RELEASE signal releases outputs synchronously to the input clock. For further information on clock dividers, please see details of additional technical documentation at the end of this data sheet. Figure 2-7 shows the clock divider connections.

Figure 2-7. Clock Divider Connections



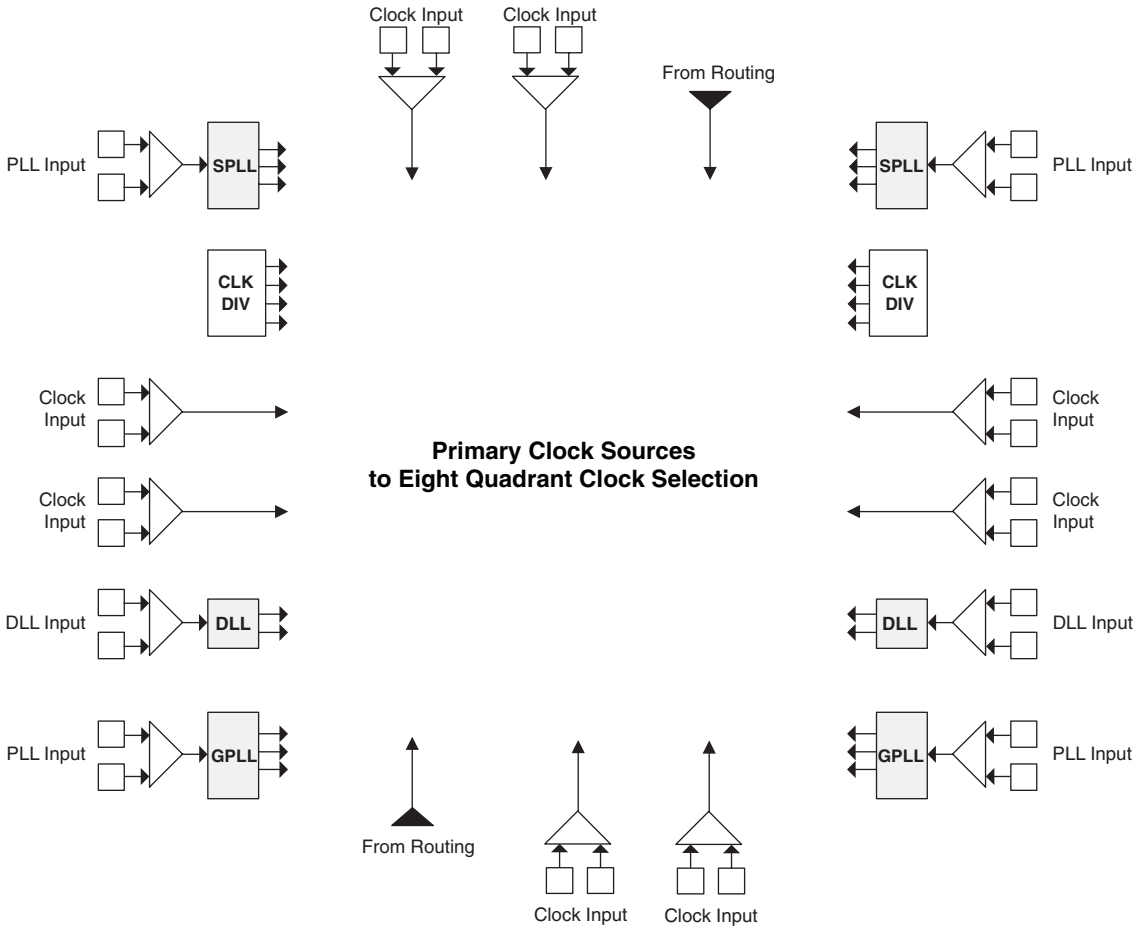
Clock Distribution Network

LatticeECP2 devices have eight quadrant-based primary clocks 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. These clock inputs are selected from external I/Os, the sysCLOCK PLLs, DLLs or routing. These clock inputs are fed throughout the chip via a clock distribution system.

Primary Clock Sources

LatticeECP2 devices derive clocks from five primary sources: PLL (GPLL and SPLL) outputs, DLL outputs, CLKDIV outputs, dedicated clock inputs and routing. LatticeECP2 devices have two to six sysCLOCK PLLs and two DLLs, located on the left and right sides of the device. There are eight dedicated clock inputs, two on each side of the device. Figure 2-8 shows the primary clock sources.

Figure 2-8. Primary Clock Sources for ECP2-50

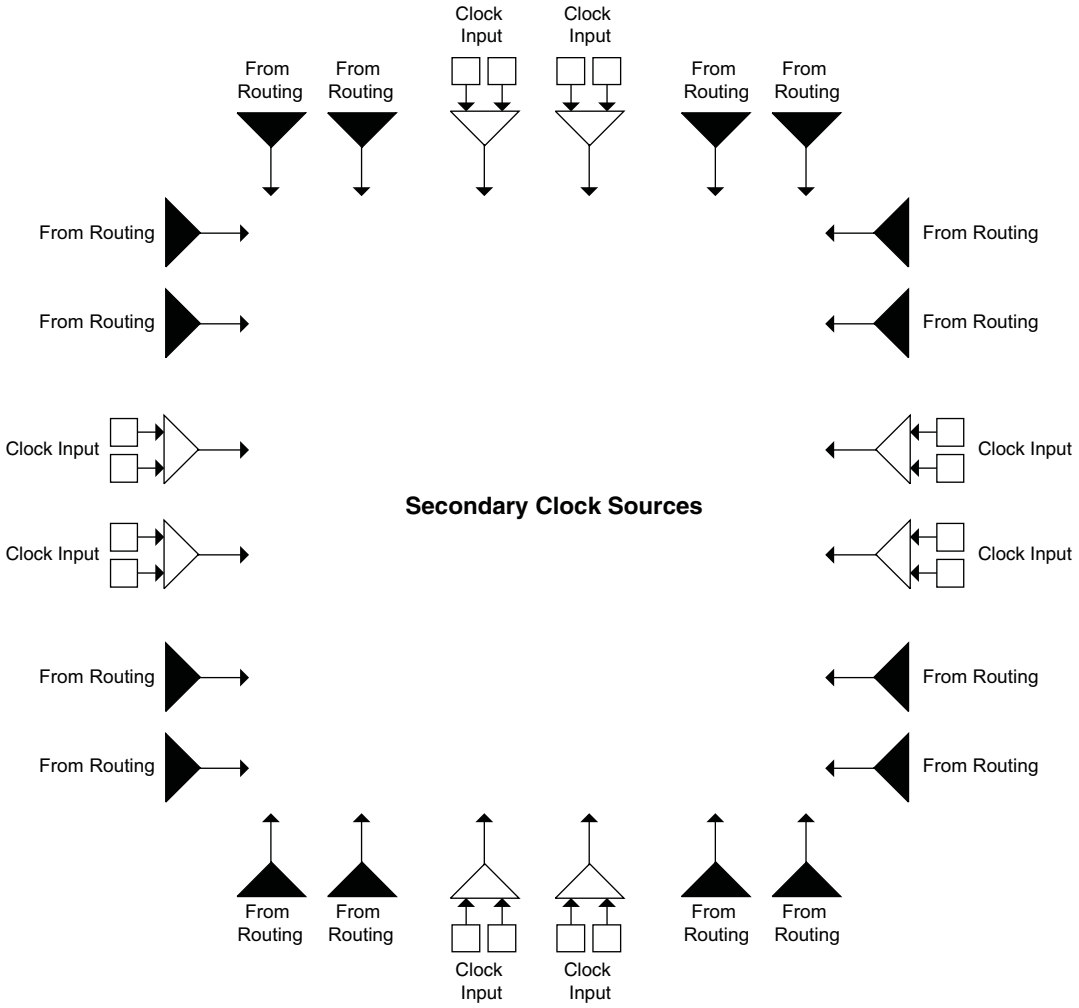


Note: This diagram shows sources for the ECP2-50 device. Smaller devices have fewer SPLLs.

Secondary Clock/Control Sources

LatticeECP2 devices derive secondary clocks (SC0 through EC7) from eight dedicated clock input pads and the rest from routing. Figure 2-9 shows the secondary clock sources.

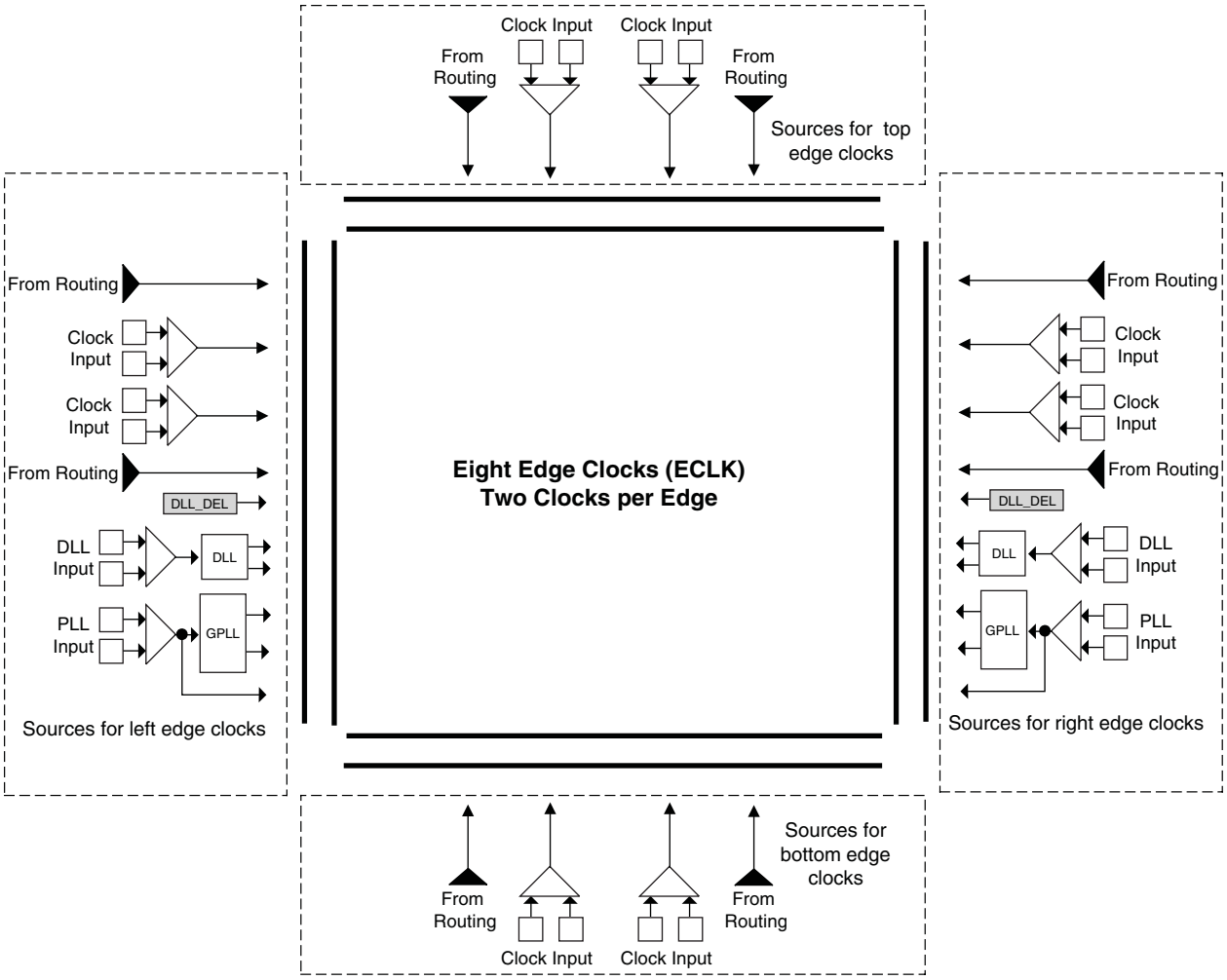
Figure 2-9. Secondary Clock Sources



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/DLLs and clock dividers as shown in Figure 2-10.

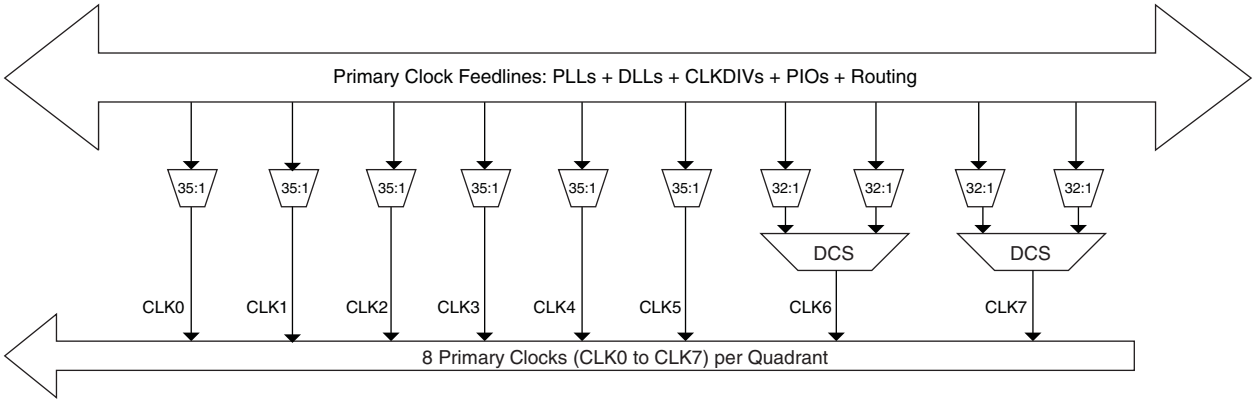
Figure 2-10. Edge Clock Sources



Primary Clock Routing

The clock routing structure in LatticeECP2 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 2-11 shows the clock routing for one quadrant. Each quadrant mux is identical. If desired, any clock can be routed globally

Figure 2-11. Per Quadrant Primary Clock Selection

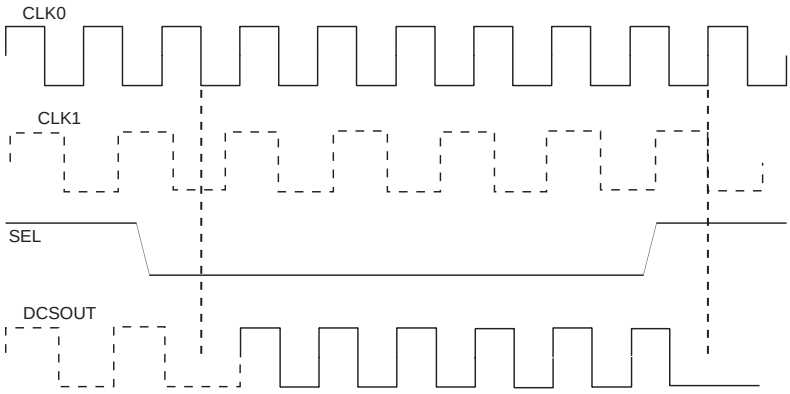


Dynamic Clock Select (DCS)

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 2-11).

Figure 2-12 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 details of additional technical documentation at the end of this data sheet.

Figure 2-12. DCS Waveforms



Secondary Clock/Control Routing

Secondary clocks in the LatticeECP2 devices are region-based resources. EBR/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 2-13 shows this special vertical routing channel and the eight secondary clock regions for the ECP2-50. LatticeECP2 devices have eight secondary clock resources per region (SC0 to SC7).

The secondary clock muxes are located in the center of the device. Figure 2-13 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 2-13. Secondary Clock Regions ECP2-50

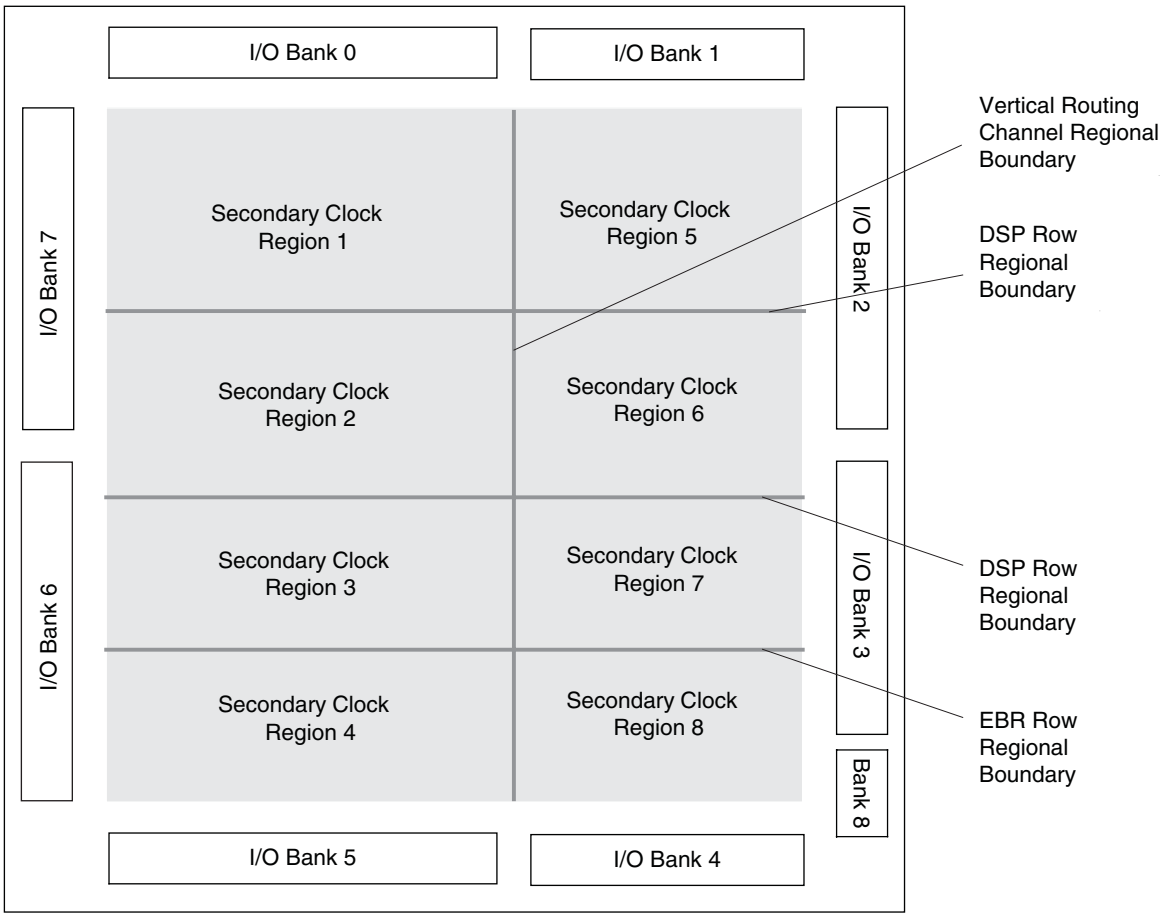
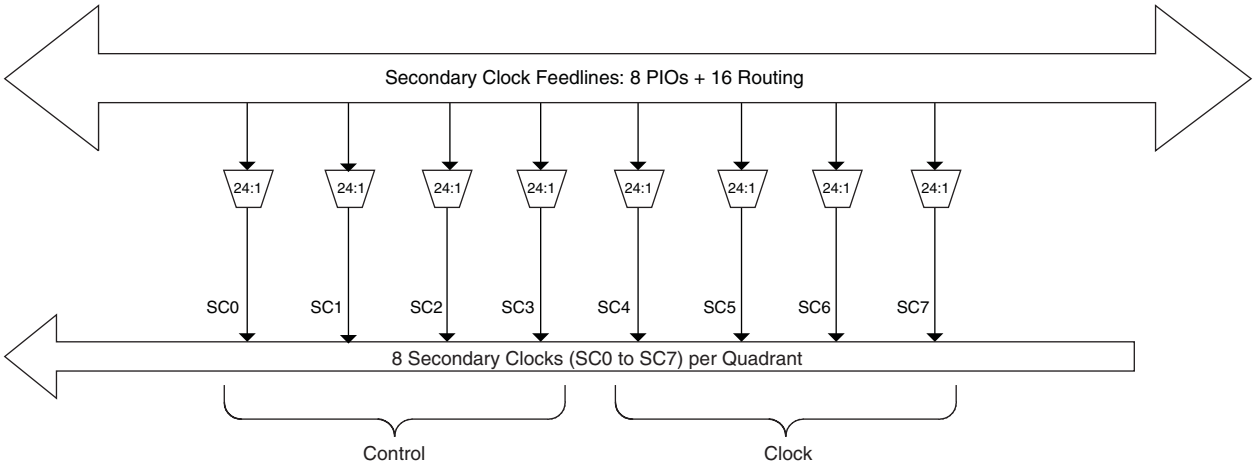


Figure 2-14. Per Region Secondary Clock Selection



Slice Clock Selection

Figure 2-15 shows the clock selections and Figure 2-16 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 a clock input 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 2-15. Slice0 through Slice2 Clock Selection

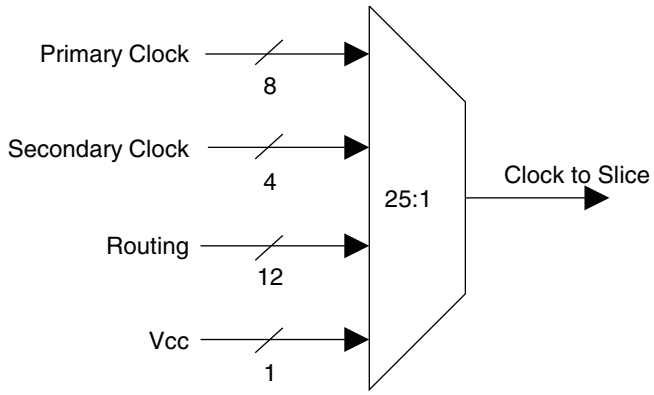
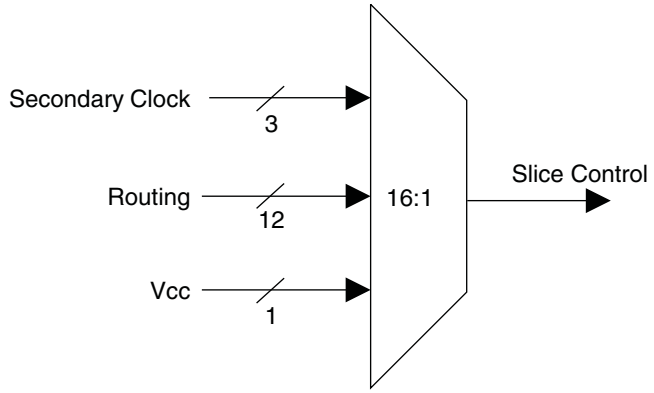


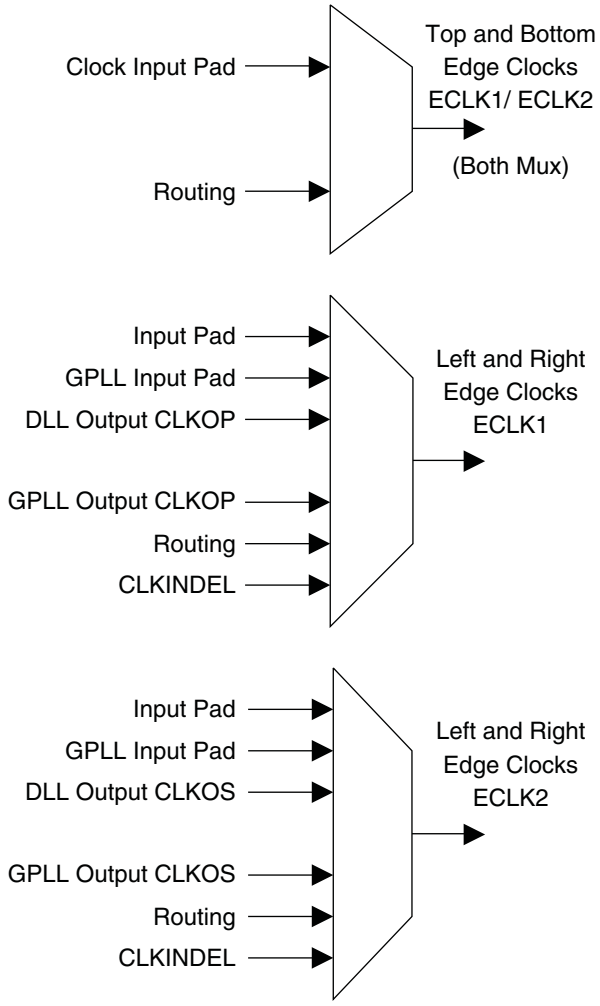
Figure 2-16. Slice0 through Slice2 Control Selection



Edge Clock Routing

LatticeECP2 devices have a number of high-speed edge clocks that are intended for use with the PIOs in the implementation of high-speed interfaces. There are eight edge clocks per device: two edge clocks per edge. Different PLL and DLL outputs are routed to the two muxes on the left and right sides of the device. In addition, the CLKINDEL signal (generated from the DLL_DEL block) is routed to all the edge clock muxes on the left and right sides of the device. Figure 2-17 shows the selection muxes for these clocks.

Figure 2-17. Edge Clock Mux Connections



sysMEM Memory

LatticeECP2 devices contains a number of sysMEM Embedded Block RAM (EBR). The EBR consists of an 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 2-6. FIFOs can be implemented in sysMEM EBR blocks by implementing support logic with PFUs. The EBR block facilitates parity checking by supporting an optional parity bit for each data byte. EBR blocks provide byte-enable support for configurations with18-bit and 36-bit data widths.

Table 2-6. 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.

RAM Initialization and ROM Operation

If desired, the contents of the RAM can be pre-loaded during device configuration. By preloading the RAM block during the chip configuration cycle and disabling the write controls, the sysMEM block can also be utilized as a ROM.

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.

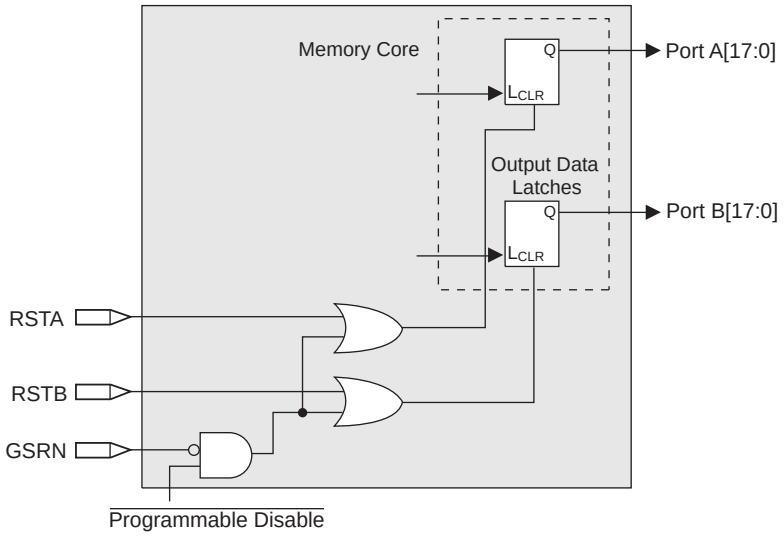
2. Write Through – A copy of the input data appears at the output of the same port during a write cycle. This mode is supported for all data widths.

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. The Global Reset (GSRN) signal resets both ports. The output data latches and associated resets for both ports are as shown in Figure 2-18.

Figure 2-18. Memory Core Reset



For further information on the sysMEM EBR block, please see the details of additional technical documentation at the end of this data sheet.

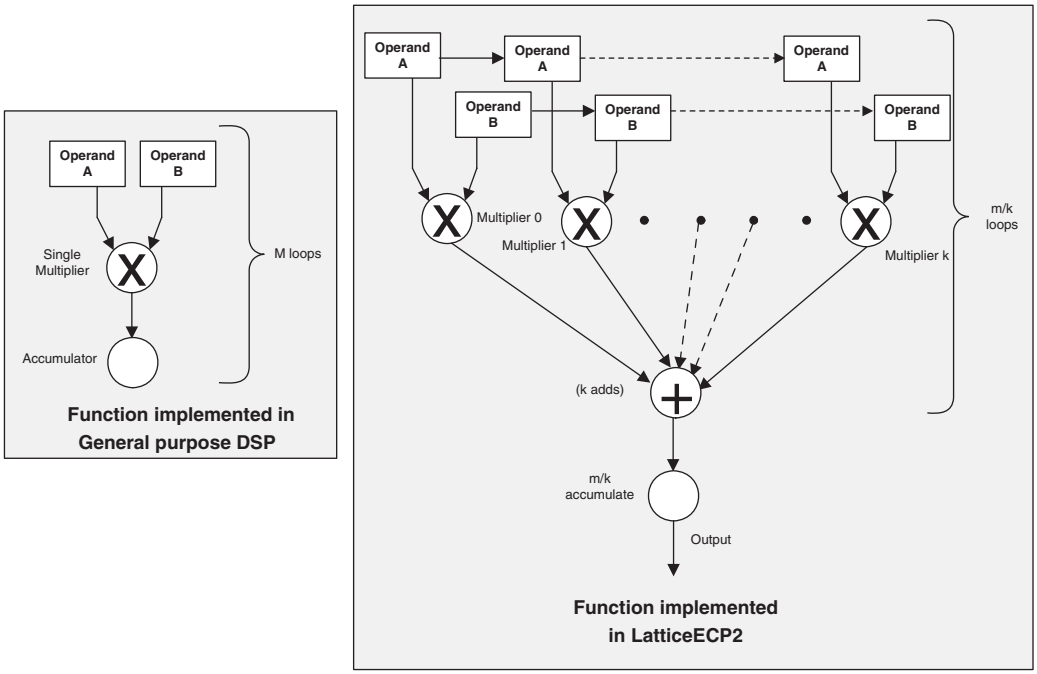
sysDSP™ Block

The LatticeECP2 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 are Finite Impulse Response (FIR) filters, Fast Fourier Transforms (FFT) functions, Correlators, Reed-Solomon/Turbo/Convolution encoders and decoders. 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 LatticeECP2, 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 level of parallelism. Figure 2-19 compares the fully serial and the mixed parallel and serial implementations.

Figure 2-19. Comparison of General DSP and LatticeECP2 Approaches



sysDSP Block Capabilities

The sysDSP block in the LatticeECP2 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 LatticeECP2 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. In LatticeECP2 family of devices the 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)
- MULTADD (Multiply, Addition/Subtraction)
- MULTADDSUM (Multiply, Addition/Subtraction, Accumulate)

The number of elements available in each block depends in 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 2-7 shows the capabilities of the block.

Table 2-7. Maximum Number of Elements in a Block

Width of Multiply	x9	x18	x36
MULT	8	4	1
MAC	2	2	—
MULTADD	4	2	—
MULTADDSUM	2	1	—

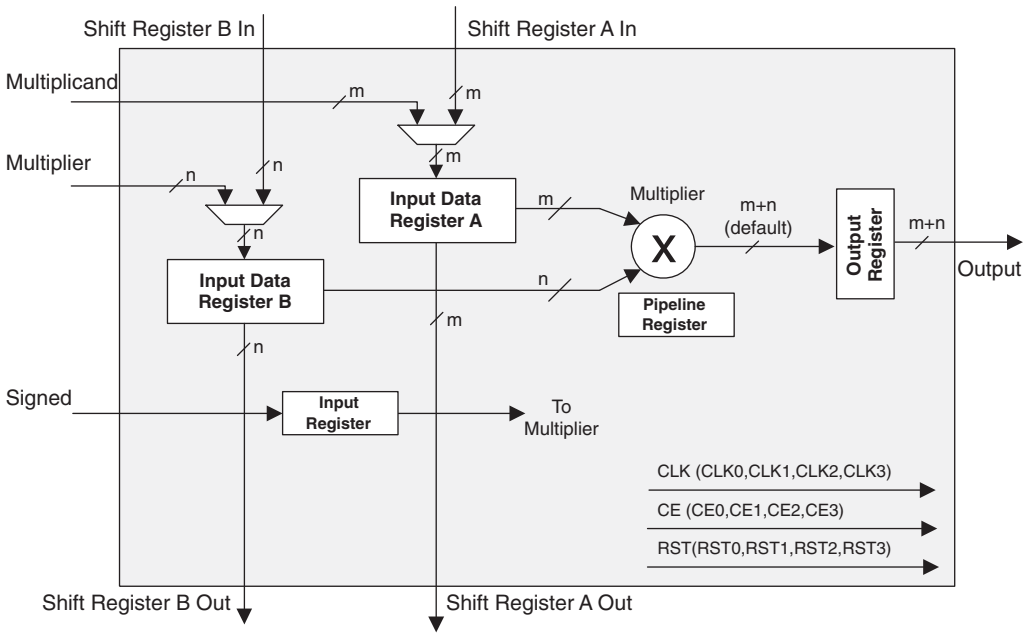
Some options are available in four elements. The input register in all the elements can be directly loaded or can be loaded as shift register from previous operand registers. By selecting ‘dynamic operation’ the following operations are possible:

- 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.

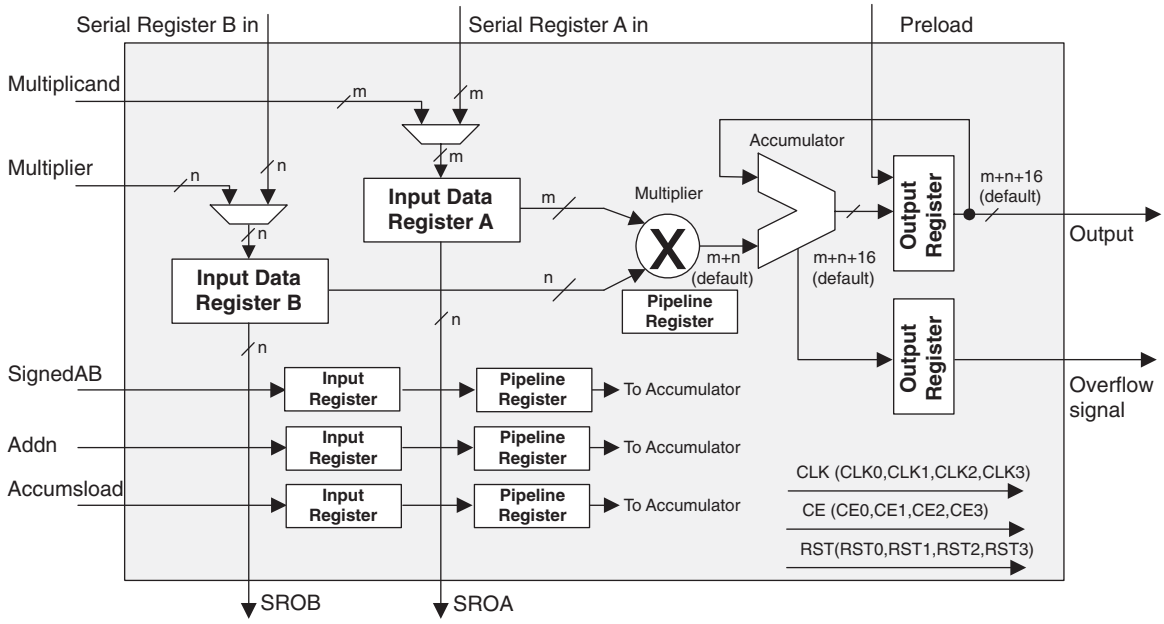
Figure 2-20. MULT sysDSP Element



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 LatticeECP2 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.

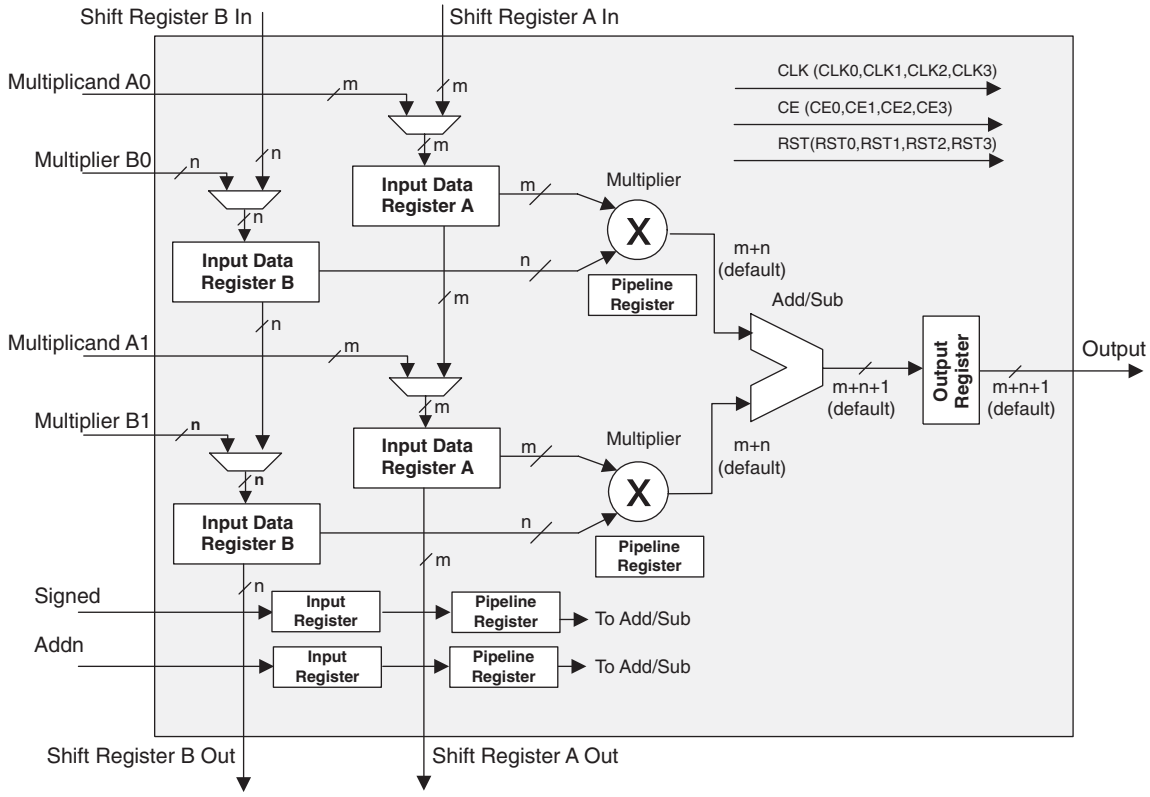
Figure 2-21. MAC sysDSP



MULTADD 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 A2. The user can enable the input, output and pipeline registers. Figure 2-20 shows the MULTADD sysDSP element.

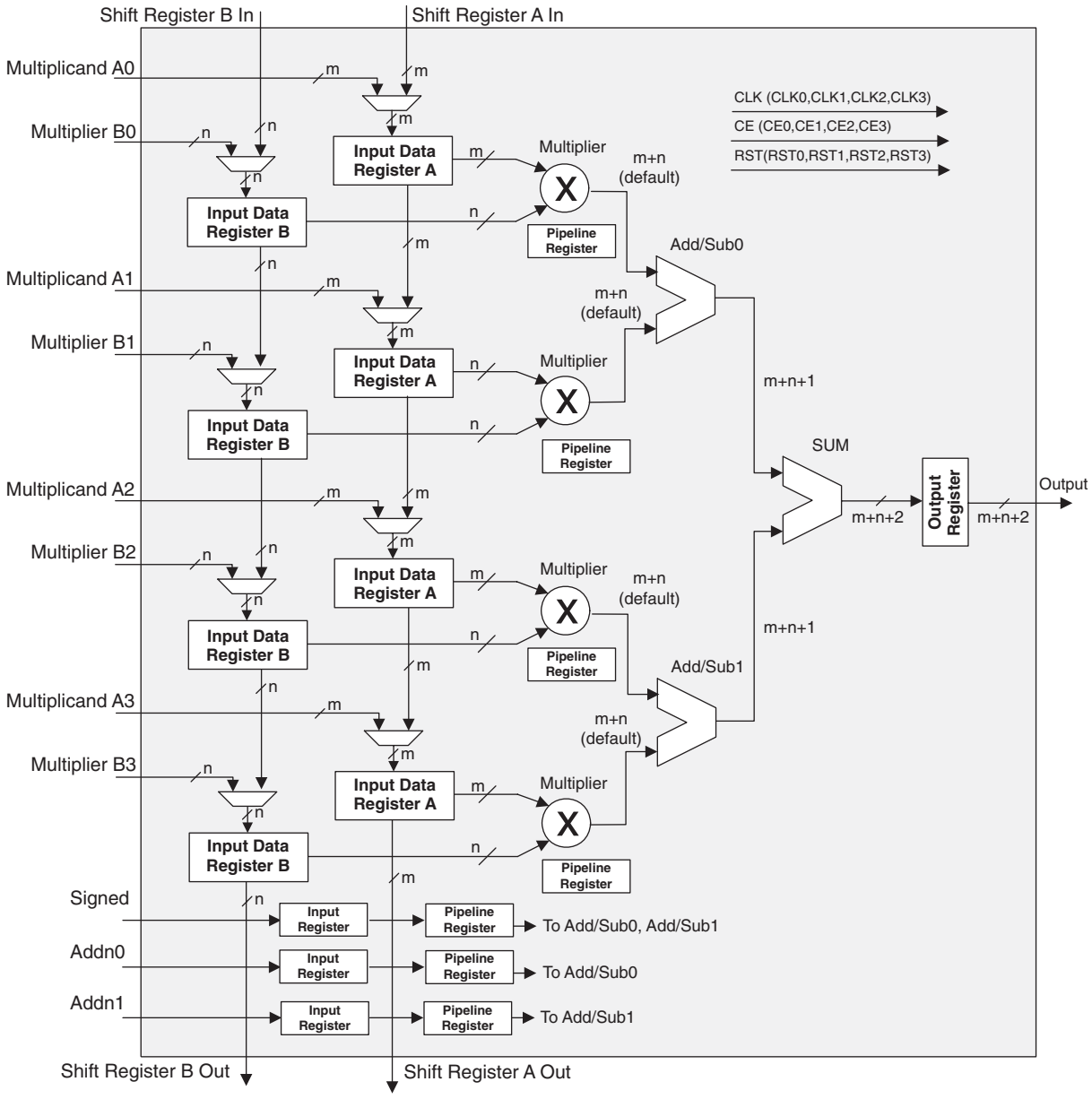
Figure 2-22. MULTADD



MULTADDSUM 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 MULTADDSUM sysDSP element.

Figure 2-23. MULTADDSUM



Clock, Clock Enable and Reset Resources

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

Reset (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 different widths of signed and unsigned multipliers besides x9, x18 and x36 widths. 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 2-8 provides an example of this.

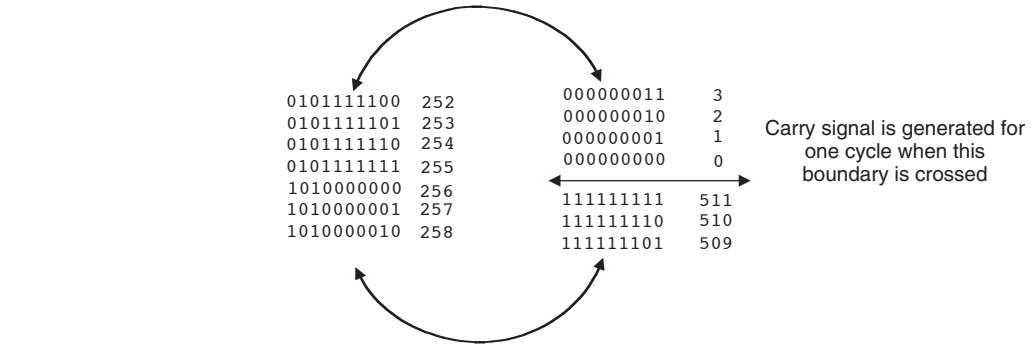
Table 2-8. Sign Extension Example

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

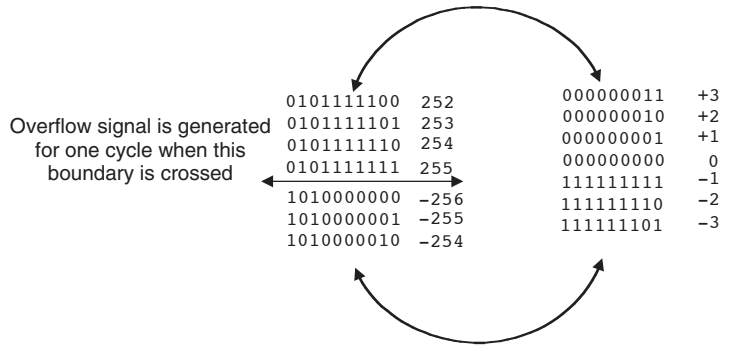
OVERFLOW Flag from MAC

The sysDSP block provides an overflow output to indicate that the accumulator has overflowed. When two unsigned numbers are added and the result is a smaller number than the accumulator, "roll-over" is said to have occurred and an overflow signal is indicated. When two positive numbers are added with a negative sum and when two negative numbers are added with a positive sum, then the accumulator "roll-over" is said to have occurred and an overflow signal is indicated. 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 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 LatticeECP2 DSP include the Bit Correlator, Fast Fourier Transform, Finite Impulse Response (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 LatticeECP2 Family

Table 2-9 shows the maximum number of multipliers for each member of the LatticeECP2 family. Table 2-10 shows the maximum available EBR RAM Blocks in each LatticeECP2 device. EBR blocks, together with Distributed RAM can be used to store variables locally for fast DSP operations.

Table 2-9. Maximum Number of DSP Blocks in the LatticeECP2 Family

Device	DSP Block	9x9 Multiplier	18x18 Multiplier	36x36 Multiplier
ECP2-6	3	24	12	3
ECP2-12	6	48	24	6
ECP2-20	7	56	28	7
ECP2-35	8	64	32	8
ECP2-50	18	144	72	18
ECP2-70	22	176	88	22

Table 2-10. Embedded SRAM in the LatticeECP2 Family

Device	EBR SRAM Block	Total EBR SRAM (Kbits)
ECP2-6	3	55
ECP2-12	12	221
ECP2-20	15	277
ECP2-35	18	332
ECP2-50	21	387
ECP2-70	56	1032

LatticeECP2 DSP Performance

Table 2-11 lists the maximum performance in millions of MAC operations per second (MMAC) for each member of the LatticeECP2 family.

Table 2-11. DSP Performance

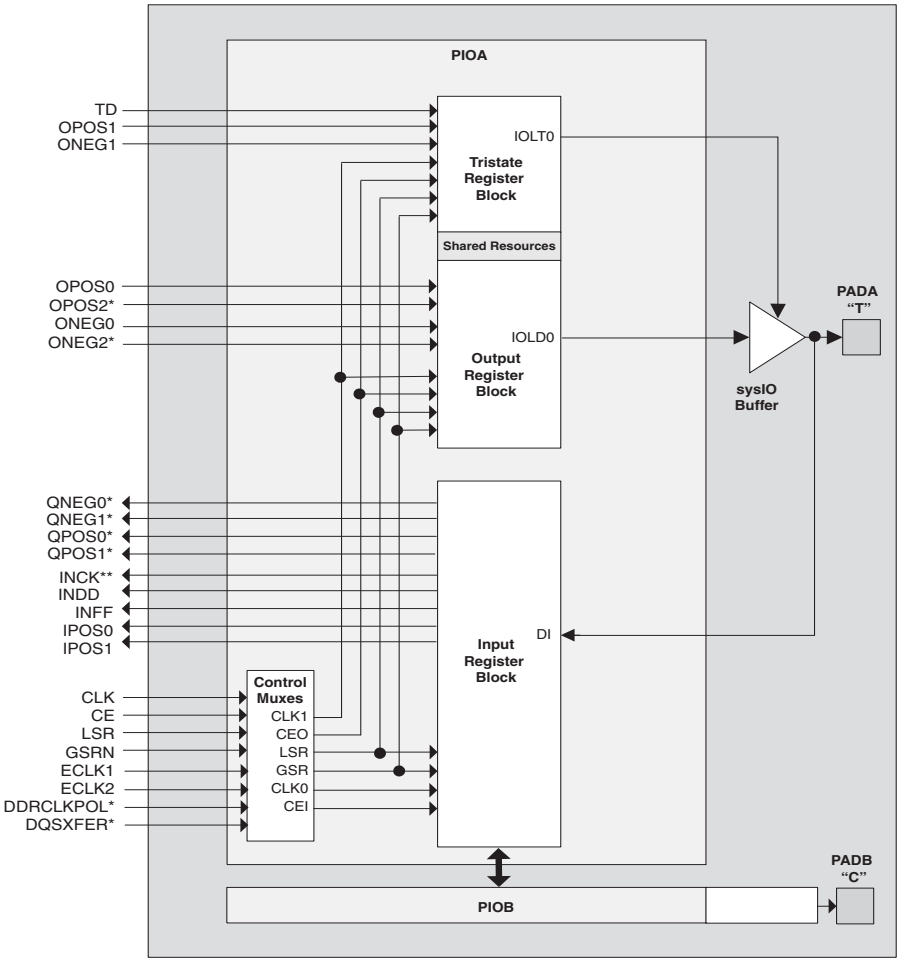
Device	DSP Block	DSP Performance MMAC
ECP2-6	3	TBA
ECP2-12	6	TBA
ECP2-20	7	TBA
ECP2-35	8	TBA
ECP2-50	18	TBA
ECP2-70	22	TBA

For further information on the sysDSP block, please see details of additional technical information at the end of this data sheet.

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. Figure 2-12 provides the PIO signal list.

Figure 2-25. PIC Diagram



*Signals are available on left/right/bottom edges only.
** 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 2-12. PIO Signal List

Name	Type	Description
CE0, CE1	Control from the core	Clock enables for input and output block flip-flops
CLK0, CLK1	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	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	Control 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 the necessary clock and selection logic.

Input Register Block

The input register blocks for PIOs in left, right and bottom edges 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 for left, right and bottom edges. The input register block for the top edge contains one memory element to register the input signal as shown in Figure 2-27. The following description applies to the input register block for PIOs in left, right and bottom edges of the device.

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) the data is registered, by one of the registers in the single data rate 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, creating two data streams, D0 and D1.

These two data streams are synchronized with the system clock before entering the core. Further discussion on this topic is in the DDR Memory section of this data sheet.

By combining input blocks of the complementary PIOs and sharing some 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 information regarding additional documentation at the end of this data sheet.

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 for Left, Right and Bottom Edges

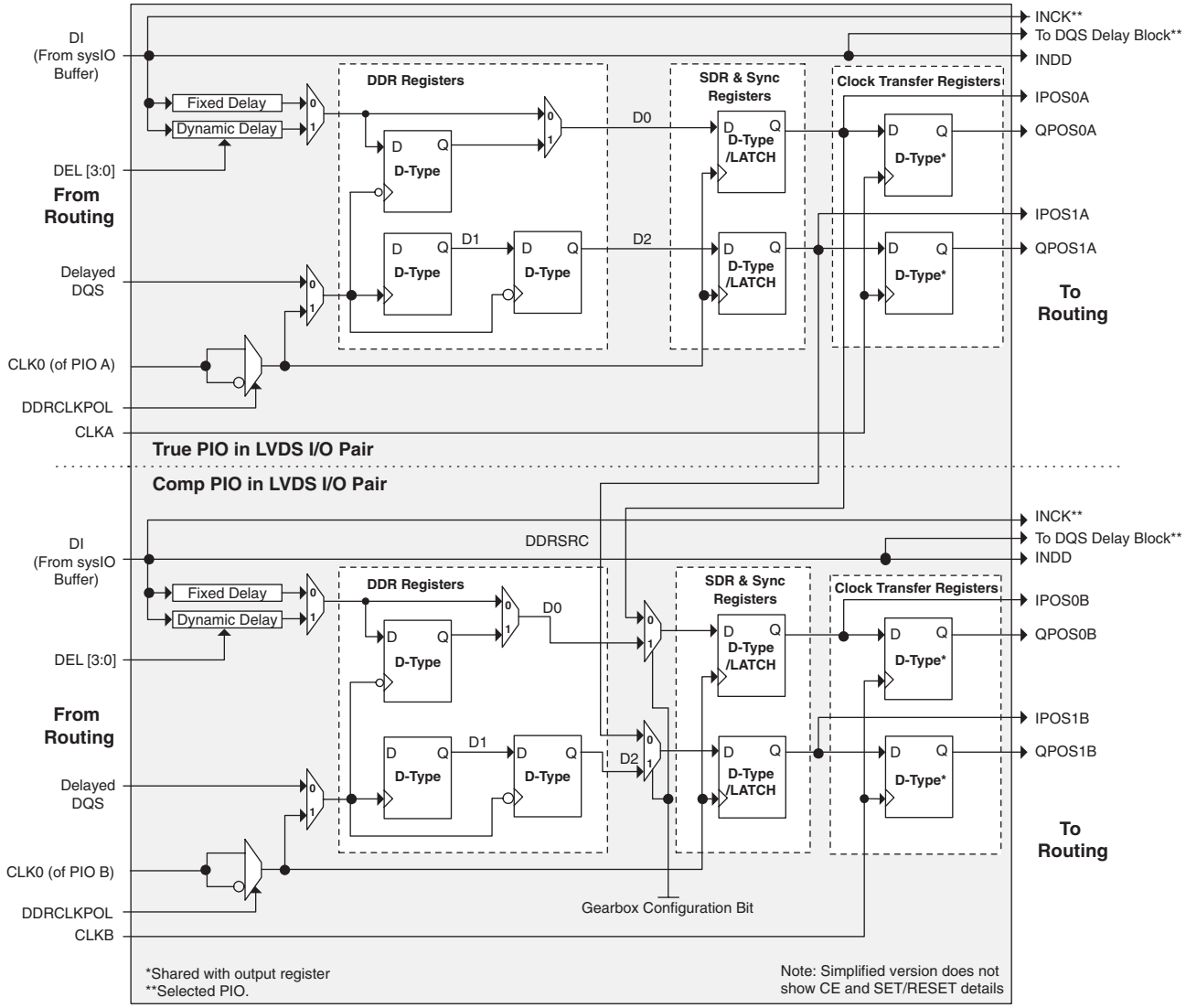
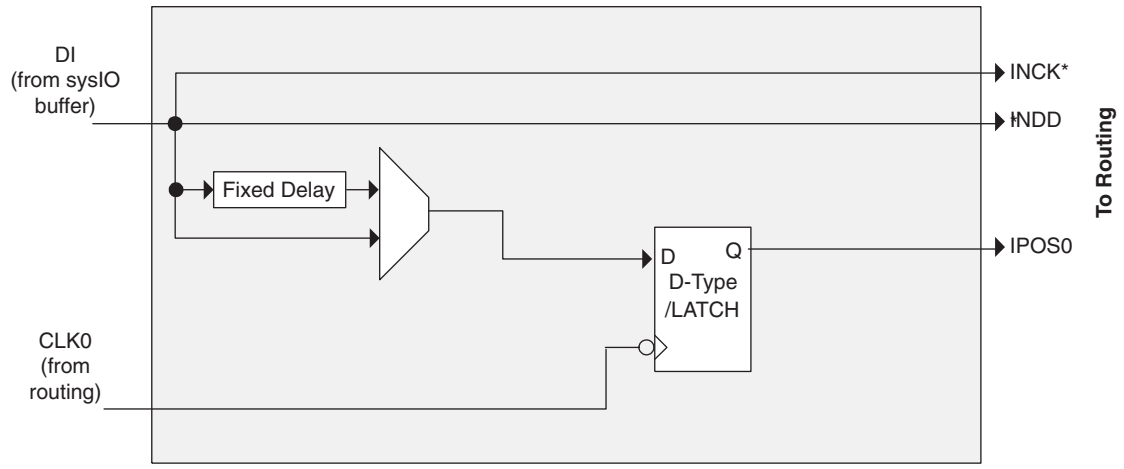


Figure 2-27. Input Register Block Top Edge



Note: Simplified version does not show CE and SET/RESET details.
*On selected blocks.

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 contains a register for SDR operation that is combined with an additional latch for DDR operation. Figure 2-28 shows the diagram of the Output Register Block for PIOs on the left, right and the bottom edges. Figure 2-29 shows the diagram of the Output Register Block for PIOs on the top edge of the device.

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 is fed into registers on the positive edge of the clock. Then at the next clock cycle this registered OPOS0 is latched. A multiplexer running off the same clock selects the correct register for feeding to the output (D0).

By combining output blocks of the complementary PIOs and sharing some registers from input blocks, a gearbox function can be implemented, that takes four data streams ONEG0A, ONEG1A, ONEG1B and ONEG1B. Figure 2-29 shows the diagram using this gearbox function. For more information on this topic, please see information regarding additional documentation at the end of this data sheet.

Figure 2-28. Output and Tristate Block for Left, Right and Bottom Edges

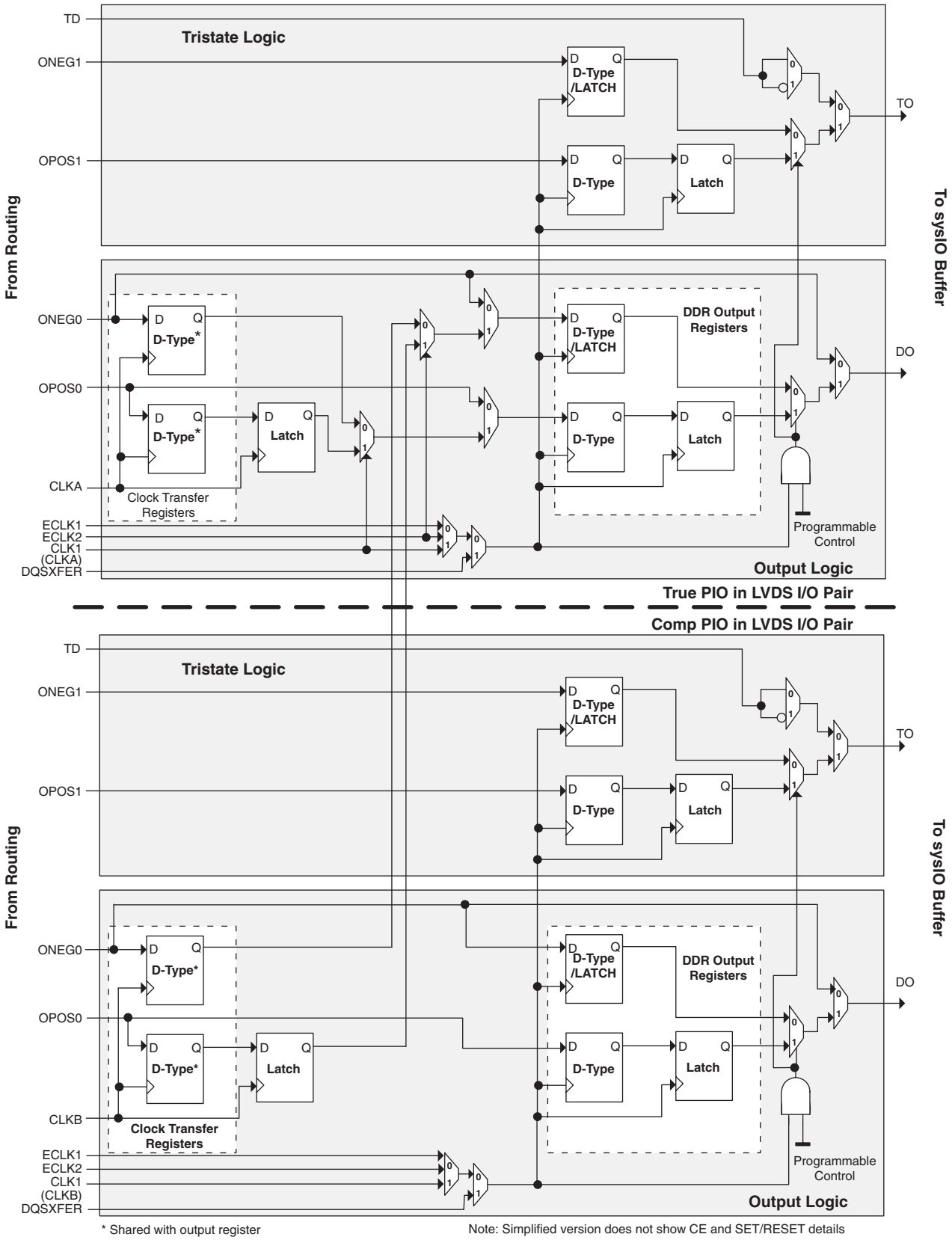
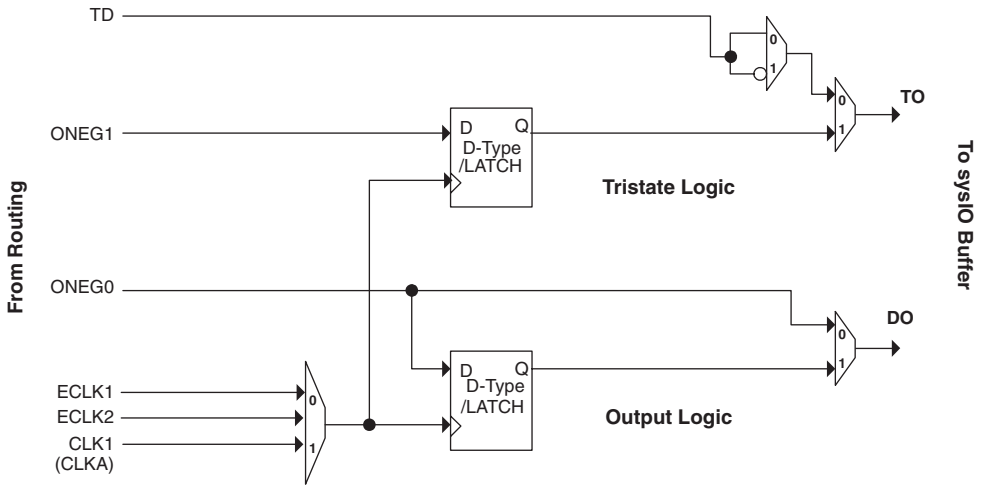


Figure 2-29. Output and Tristate Block, Top Edge



Note: Simplified version does not show CE and SET/RESET details.

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 2-28 shows the diagram of the Tristate Register Block with the Output Block for the left, right and bottom edges and Figure 2-29 shows the diagram of the Tristate Register Block with the Output Block for the top edge.

In SDR mode, ONEG1 feeds one of the flip-flops that then feeds the output. The flip-flop can be configured a 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 is selected from one of the clock signals provided from the general purpose routing, one of the edge clocks (ECLK1/ ECLK2) and a DQS signal provided from the programmable DQS pin and provided to the input register block. The clock can optionally be inverted.

DDR Memory Support

Certain PICs have additional circuitry to allow the implementation of high speed source synchronous and DDR memory interfaces. The support varies by edge of the device as detailed below.

Left and Right Edges

PICs on these edges have registered elements that support DDR memory interfaces. One of every 16 PIOs contains a delay element to facilitate the generation of DQS signals. The DQS signal feeds the DQS bus which spans the set of 16 PIOs. Figure 2-30 shows the assignment of DQS pins in each set of 16 PIOs.

Bottom Edge

PICs on these edges have registered elements that support DDR memory interfaces. One of every 18 PIOs contains a delay element to facilitate the generation of DQS signals. The DQS signal feeds the DQS bus that spans the set of 18 PIOs. Figure 2-31 shows the assignment of DQS pins in each set of 18 PIOs.

Top Edge

The PICs on the top edge are different from PIOs on the left, right and bottom edges. PIOs on this edge do not have registers or DQS signals.

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. Interfaces on the left and right edges are designed for DDR memories that support 16 bits of data, whereas interfaces on the bottom are designed for memories that support 18 bits of data.

Figure 2-30. DQS Routing for the Left and Right Edges of the Device

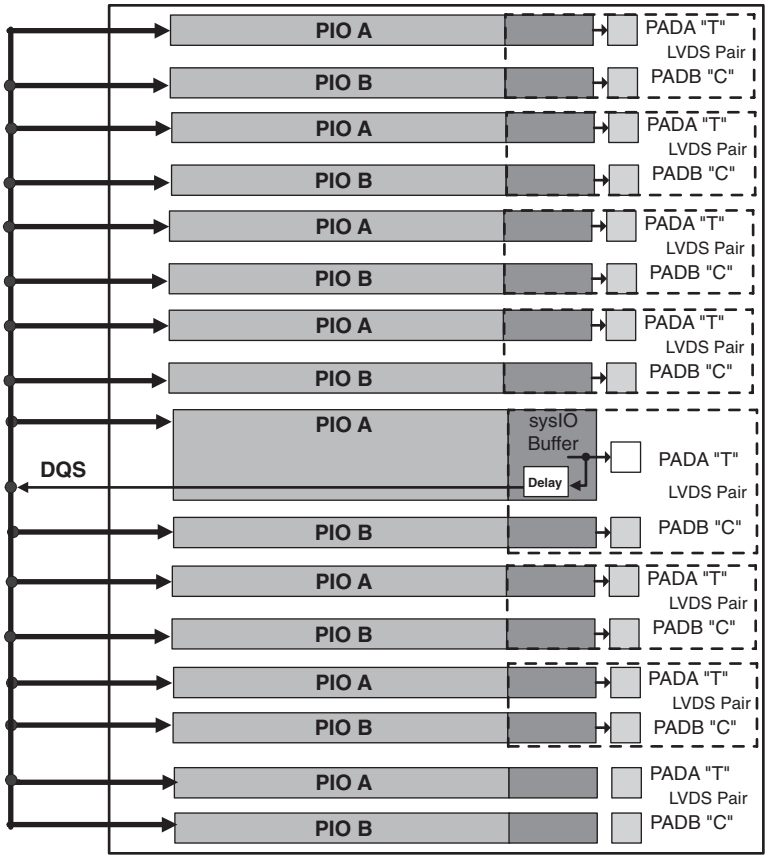
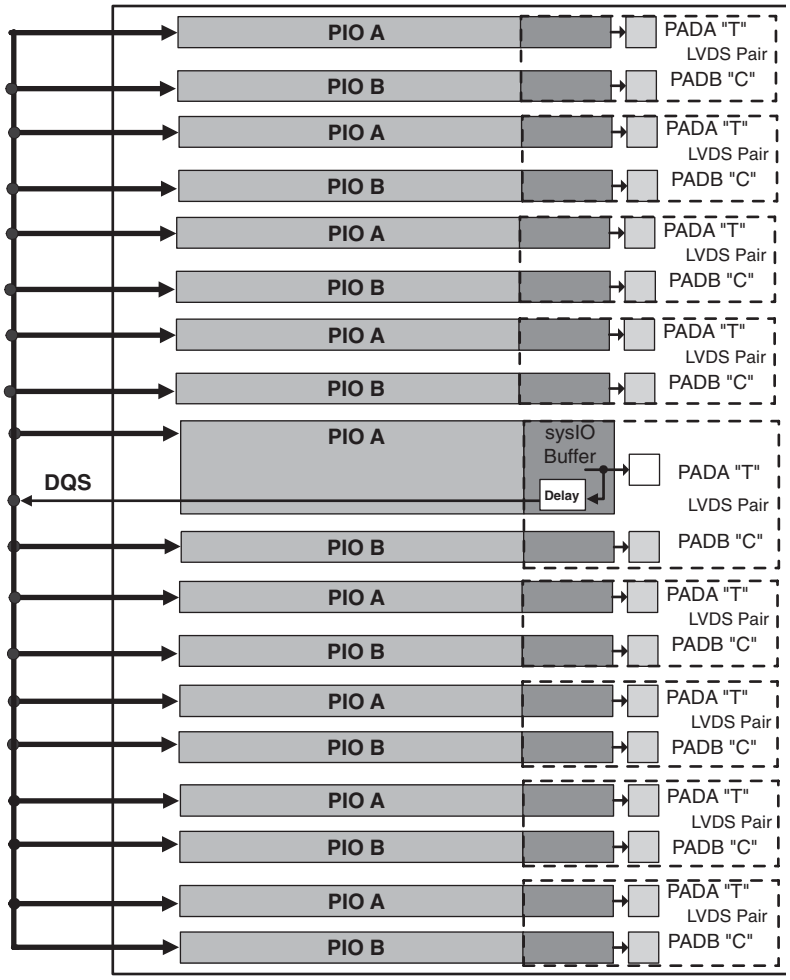


Figure 2-31. DQS Routing for the Bottom Edge of the Device



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 so 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 2-32) 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 2-32 and Figure 2-33 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 calibration (6-bit bus) 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 2-32. The DLL loop is compensated for temperature, voltage and process variations by the system clock and feedback loop.

Figure 2-32. DLL Calibration Bus and DQS/DQS Transition Distribution

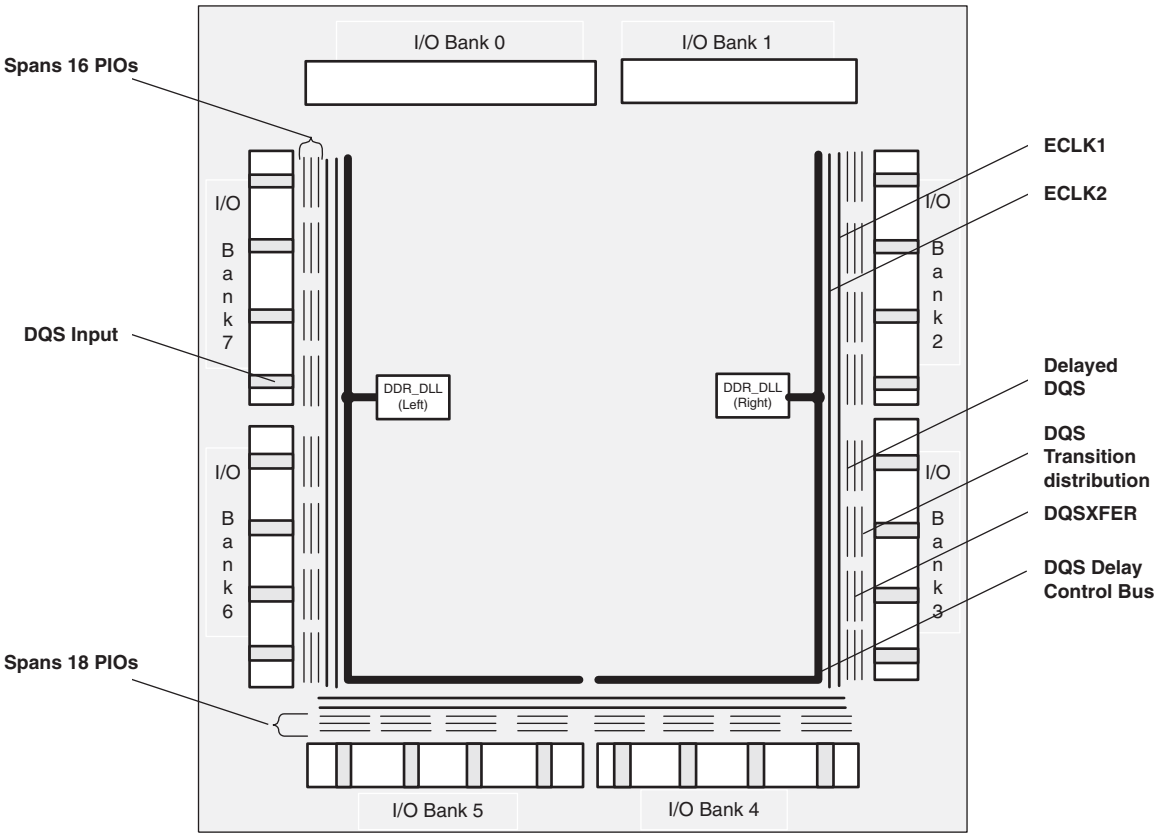
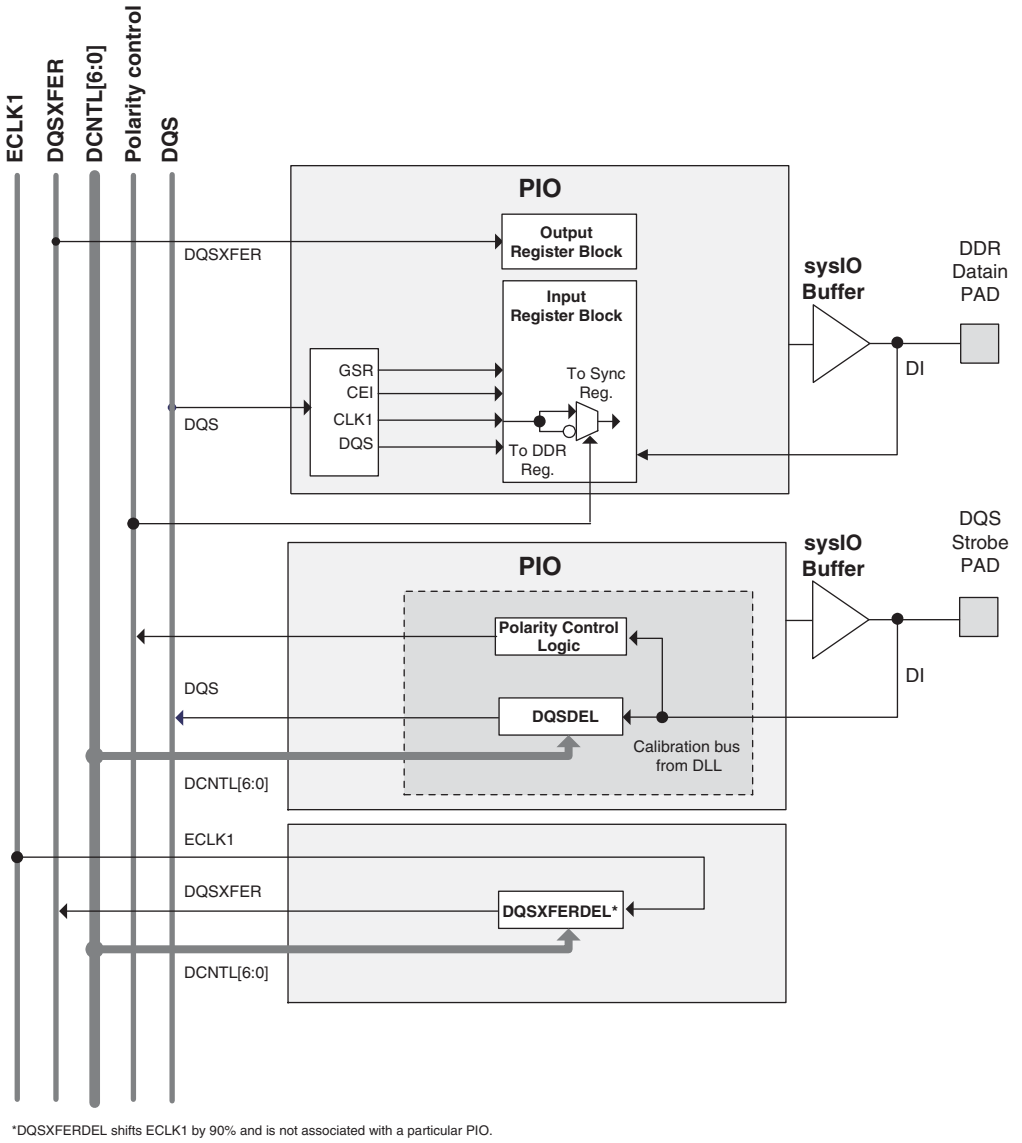


Figure 2-33. DQS Local Bus



*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 LatticeECP2 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. This 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

LatticeECP2 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.

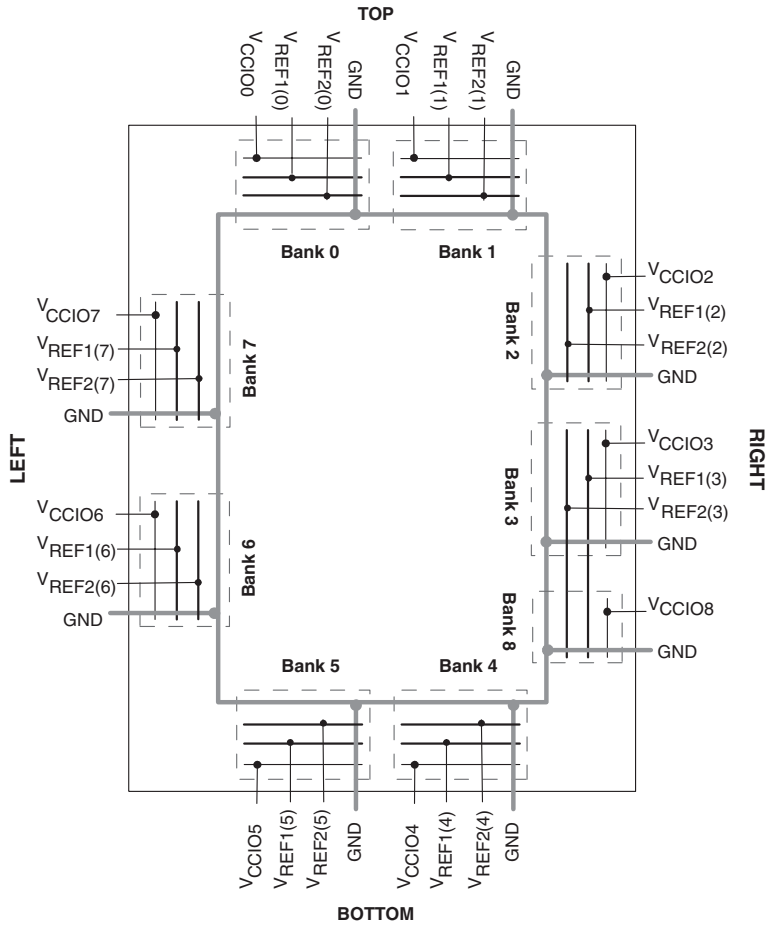
sysIO Buffer Banks

LatticeECP2 devices have nine sysIO buffer banks: eight banks for user I/Os arranged two per side. The ninth sysIO buffer bank (Bank 8) is located adjacent to Bank 3 and has dedicated/shared I/Os for configuration. When a shared pin is not used for configuration it is available as a user I/O. 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, except Bank 8, has voltage references, V_{REF1} and V_{REF2}, that allow it to be completely independent from the others. Bank 8 shares two voltage references, V_{REF1} and V_{REF2}, with Bank 3. Figure 2-34 shows the nine banks and their associated supplies.

In LatticeECP2 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 2-34. LatticeECP2 Banks



LatticeECP2 devices contain two types of sysIO buffer pairs.

1. **Top (Bank 0 and Bank 1) 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.

2. **Bottom (Bank 4 and Bank 5) sysIO Buffer Pairs (Single-Ended Outputs Only)**
   The sysIO buffer pairs in the bottom 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 bottom banks have programmable PCI clamps.

3. **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. In these banks 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 (complementary) 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.

4. **Bank 8 sysIO Buffer Pairs (Single-Ended Outputs, Only on Shared Pins When Not Used by Configuration)**
   The sysIO buffers in Bank 8 consist of single-ended output drivers and single-ended input buffers (both ratioed and referenced). The referenced input buffer 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.

Typical I/O Behavior During Power-up

The internal power-on-reset (POR) signal is deactivated when V_{CC}, V_{CCIO8} 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 LatticeECP2 devices, see details of additional technical documentation at the end of this data sheet.

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 Standards

The LatticeECP2 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 LatticeECP2 devices. For further information on utilizing the sysIO buffer to support a variety of standards please see the details of additional technical information at the end of this data sheet.

Table 2-13. 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	—
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 2-14. 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/PCIX†	N/A	3.3
HSTL18 Class I, II	N/A	1.8
HSTL15 Class I	N/A	1.5
SSTL3 Class I, II	N/A	3.3
SSTL2 Class I, II	N/A	2.5
SSTL18 Class I, II	N/A	1.8
Differential Interfaces		
Differential SSTL3, Class I, II	N/A	3.3
Differential SSTL2, 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	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 information regarding additional technical documentation at the end of this data sheet.

Hot Socketing

LatticeECP2 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 LatticeECP2 ideal for many multiple power supply and hot-swap applications.

Configuration and Testing

This section describes the configuration and testing features of the LatticeECP2 family of devices.

IEEE 1149.1-Compliant Boundary Scan Testability

All LatticeECP2 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 details on boundary scan test, please see information regarding additional technical documentation at the end of this data sheet.

Device Configuration

All LatticeECP2 devices contain two ports that can be used for device configuration. The Test Access Port (TAP), which supports bit-wide configuration, and the sysCONFIG port, support both byte-wide and serial configuration. The TAP supports both the IEEE Standard 1149.1 Boundary Scan specification and the IEEE Standard 1532 In-System Configuration specification. The sysCONFIG port is a 20-pin interface with six I/Os used as dedicated pins with the remainder used as dual-use pins. See Lattice technical note number TN1108, *LatticeECP2 sysCONFIG Usage Guide* for more information on using the dual-use pins as general purpose I/Os.

There are five ways to configure a LatticeECP2 device:

1. Industry standard SPI serial memories
2. Industry standard byte wide flash with an ispMACH™ 4000, providing control and addressing
3. System microprocessor to drive a sysCONFIG port or JTAG TAP
4. Industry standard FPGA boot PROM memory
5. JTAG

On power-up, the FPGA SRAM is ready to be configured using the selected sysCONFIG port. Once a configuration port is selected, it will remain active throughout that configuration cycle. The IEEE 1149.1 port can be activated any time after power-up by sending the appropriate command through the TAP port.

Enhanced Configuration Option

LatticeECP2 devices have enhanced configuration features such as: decryption support, TransFR™ I/O and dual boot image support.

1. **Decryption Support**
   LatticeECP2 devices provide on-chip, non-volatile key storage to support decryption of a 128-bit AES encrypted bitstream, securing designs and deterring design piracy. The decryption block supports nearly all the programming modes.

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. See Lattice technical note number TN1087, *Minimizing System Interruption During Configuration Using TransFR Technology,* for details.

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 LatticeECP2 can be re-booted from this new configuration file. If there is a problem such as corrupt data during down or incorrect version number with this new boot image, the LatticeECP2 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 on device configuration, please see details of additional technical documentation at the end of this data sheet.

Software Error Detect (SED) Support

LatticeECP2 devices have dedicated logic to perform CRC checking of the bitstream and can be programmed such that, if an error occurs, the device will reload from a known good boot image or generate an error signal and stop configuring.

For more information on Software Error Detect support, please see details of additional technical documentation at the end of this data sheet.

External Resistor

LatticeECP2 devices require a single external, 10K ohm +/- 1% value between the XRES pin and ground. Device configuration will not be completed if this resistor is missing. There is no boundary scan register on the external resistor pad.

On-Chip Oscillator

Every LatticeECP2 device has an internal CMOS oscillator which is used to derive a Master Clock for configuration. The oscillator and the Master Clock run continuously and are available to user logic after configuration is completed. The default value of the Master Clock is 2.5MHz. Table 2-15 lists all the available Master Clock frequencies. When a different Master Clock is selected during the design process, the following sequence takes place:

1. User selects a different Master Clock frequency
2. During configuration the device starts with the default (2.5MHz) Master Clock frequency
3. The clock configuration settings are contained in the early configuration bitstream
4. The Master Clock frequency changes to the selected frequency once the 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 details of additional technical documentation at the end of this data sheet.

Table 2-15. Selectable Master Clock (CCLK) Frequencies During Configuration

CCLK (MHz)	CCLK (MHz)	CCLK (MHz)
2.5[1]	13	45
4.3	15	51
5.4	20	55
6.9	26	60
8.1	30	130
9.2	34	—
10.0	41	—

1. Default frequency.

Density Shifting

The LatticeECP2 family is designed to ensure that different density devices in the same package have the same pin-out. 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.

LatticeECP2 Family Data Sheet DC and Switching Characteristics

February 2006 **Advance Data Sheet**

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

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

Input or I/O Tristate Voltage Applied[4] . . . . . -0.5 to 3.75V

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

Junction Temperature (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. 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_{CCIO}[1, 2]	I/O Driver Supply Voltage	1.14	3.465	V
V_{CCJ}[1]	Supply Voltage for IEEE 1149.1 Test Access Port	1.14	3.465	V
t_{JCOM}	Junction Commercial Operation	0	85	°C
t_{JIND}	Junction Industrial Operation	-40	100	°C

1. 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}.
2. See recommended voltages by I/O standard in subsequent table.

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

Symbol	Parameter	Condition	Min.	Typ.	Max.	Units
I_{DK}	Input or I/O Leakage Current	$0 \le V_{IN} \le V_{IH}$ (MAX.)	—	—	+/-1000	µA

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	$0 \le V_{IN} \le V_{IH}$ (MAX)	—	—	10	µA
I_{PU}	I/O Active Pull-up Current	$0 \le V_{IN} \le 0.7\ V_{CCIO}$	-30	—	-210	µA
I_{PD}	I/O Active Pull-down Current	V_{IL} (MAX) $\le V_{IN} \le V_{IH}$ (MAX)	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	$0 \le V_{IN} \le V_{IH}$ (MAX)	—	—	210	µA
I_{BHHO}	Bus Hold High Overdrive Current	$0 \le V_{IN} \le V_{IH}$ (MAX)	—	—	-210	µA
V_{BHT}	Bus Hold Trip Points	$0 \le V_{IN} \le V_{IH}$ (MAX)	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]	Units
I_{CC}	Core Power Supply Current	ECP2-6		mA
		ECP2-12		mA
		ECP2-20		mA
		ECP2-35		mA
		ECP2-50		mA
		ECP2-70		mA
I_{CCAUX}	Auxiliary Power Supply Current	ECP2-6		mA
		ECP2-12		mA
		ECP2-20		mA
		ECP2-35		mA
		ECP2-50		mA
		ECP2-70		mA
I_{CCGPLL}	GPLL Power Supply Current (per GPLL)			mA
I_{CCSPLL}	SPLL Power Supply Current (per SPLL)			mA
I_{CCIO}	Bank Power Supply Current[6]			mA
I_{CCJ}	V_{CCJ} Power Supply Current			mA

1. For further information on supply current, please see details of additional technical documentation at the end of this data sheet.
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 Per bank.

Initialization Supply Current[1, 2, 3, 4, 5, 6]

Over Recommended Operating Conditions

Symbol	Parameter	Device	Typical[5]	Units
I_{CC}	Core Power Supply Current	ECP2-6		mA
		ECP2-12		mA
		ECP2-20		mA
		ECP2-35		mA
		ECP2-50		mA
		ECP2-70		mA
I_{CCAUX}	Auxiliary Power Supply Current	ECP2-6		mA
		ECP2-12		mA
		ECP2-20		mA
		ECP2-35		mA
		ECP2-50		mA
		ECP2-70		mA
I_{CCGPLL}	GPLL Power Supply Current (per GPLL)			mA
I_{CCSPLL}	SPLL Power Supply Current (per SPLL)			mA
I_{CCIO}	Bank Power Supply Current[7]			mA
I_{CCJ}	VCCJ Power Supply Current			mA

1. Until DONE signal is active.
2. For further information on supply current, please see details of additional technical documentation at the end of this data sheet.
3. Assumes all outputs are tristated, all inputs are configured as LVCMOS and held at the V_{CCIO} or GND.
4. Frequency 0MHz.
5. T_J = 25°C, power supplies at nominal voltage.
6. 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.
7. Per bank.

sysIO Recommended Operating Conditions

Standard	V_{CCIO}			V_{REF} (V)		
	Min.	Typ.	Max.	Min.	Typ.	Max.
LVCMOS 3.3[2]	3.135	3.3	3.465	—	—	—
LVCMOS 2.5[2]	2.375	2.5	2.625	—	—	—
LVCMOS 1.8	1.71	1.8	1.89	—	—	—
LVCMOS 1.5	1.425	1.5	1.575	—	—	—
LVCMOS 1.2[2]	1.14	1.2	1.26	—	—	—
LVTTL[2]	3.135	3.3	3.465	—	—	—
PCI	3.135	3.3	3.465	—	—	—
SSTL18[2] Class I, II	1.71	1.8	1.89	0.833	0.9	0.969
SSTL2[2] Class I, II	2.375	2.5	2.625	1.15	1.25	1.35
SSTL3[2] Class I, II	3.135	3.3	3.465	1.3	1.5	1.7
HSTL[2] Class I	1.425	1.5	1.575	0.68	0.75	0.9
HSTL[2] 18 Class I, II	1.71	1.8	1.89	0.816	0.9	1.08
LVDS[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], II[2]	1.71	1.8	1.89	—	—	—
SSTL25D_ I[2], II[2]	2.375	2.5	2.625	—	—	—
SSTL33D_ I[2], II[2]	3.135	3.3	3.465	—	—	—
HSTL15D_ I[2], II[2]	1.425	1.5	1.575	—	—	—
HSTL18D_ I[2], 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

Input/Output Standard	V_{IL}		V_{IH}		V_{OL}		I_{OL}[1] (mA)	I_{OH}[1] (mA)
	Min. (V)	Max. (V)	Min. (V)	Max. (V)	Max. (V)	Min. (V)		
LVCMOS 3.3	-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
LVTTL	-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
LVCMOS 2.5	-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
LVCMOS 1.8	-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
LVCMOS 1.5	-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
LVCMOS 1.2	-0.3	0.35 V_{CCIO}	0.65 V_{CCIO}	3.6	0.4	V_{CCIO} - 0.4	6, 2	-6, -2
					0.2	V_{CCIO} - 0.2	0.1	-0.1
PCI	-0.3	0.3 V_{CCIO}	0.5 V_{CCIO}	3.6	0.1 V_{CCIO}	0.9 V_{CCIO}	1.5	-0.5
SSTL3 Class I	-0.3	V_{REF} - 0.2	V_{REF} + 0.2	3.6	0.7	V_{CCIO} - 1.1	8	-8
SSTL3 Class II	-0.3	V_{REF} - 0.2	V_{REF} + 0.2	3.6	0.5	V_{CCIO} - 0.9	16	-16
SSTL2 Class 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
SSTL2 Class 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 Class I	-0.3	V_{REF} - 0.125	V_{REF} + 0.125	3.6	0.4	V_{CCIO} - 0.4	6.7	-6.7
SSTL18 Class II	-0.3	V_{REF} - 0.125	V_{REF} + 0.125	3.6	0.28	V_{CCIO} - 0.28	8	-8
							11	-11
HSTL Class I	-0.3	V_{REF} - 0.1	V_{REF} + 0.1	3.6	0.4	V_{CCIO} - 0.4	4	-4
							8	-8
HSTL18 Class I	-0.3	V_{REF} - 0.1	V_{REF} + 0.1	3.6	0.4	V_{CCIO} - 0.4	8	-8
							12	-12
HSTL18 Class 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 LatticeECP2 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

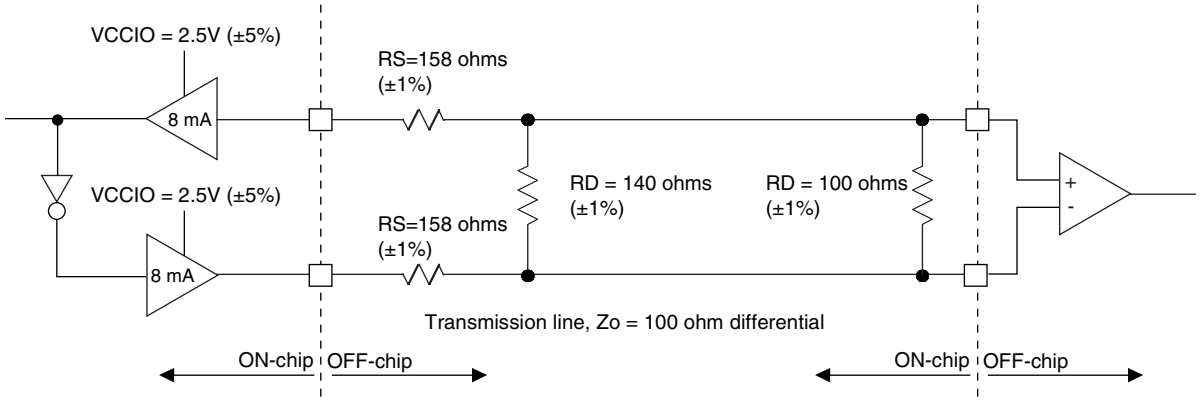


Table 3-1. LVDS25E DC Conditions

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 LatticeECP2 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

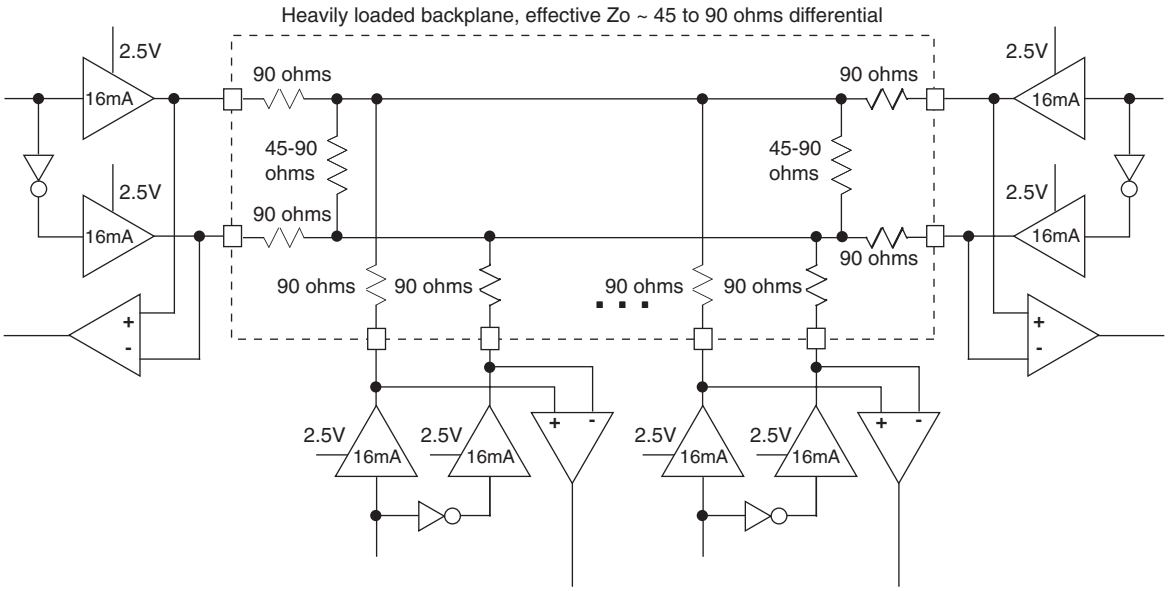


Table 3-2. BLVDS DC Conditions[1]

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 LatticeECP2 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

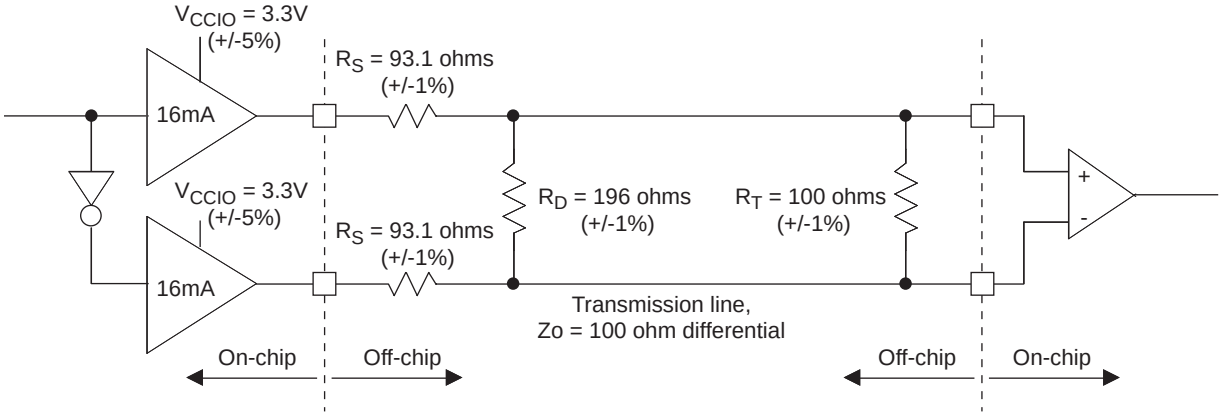


Table 3-3. LVPECL DC Conditions[1]

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 LatticeECP2 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)

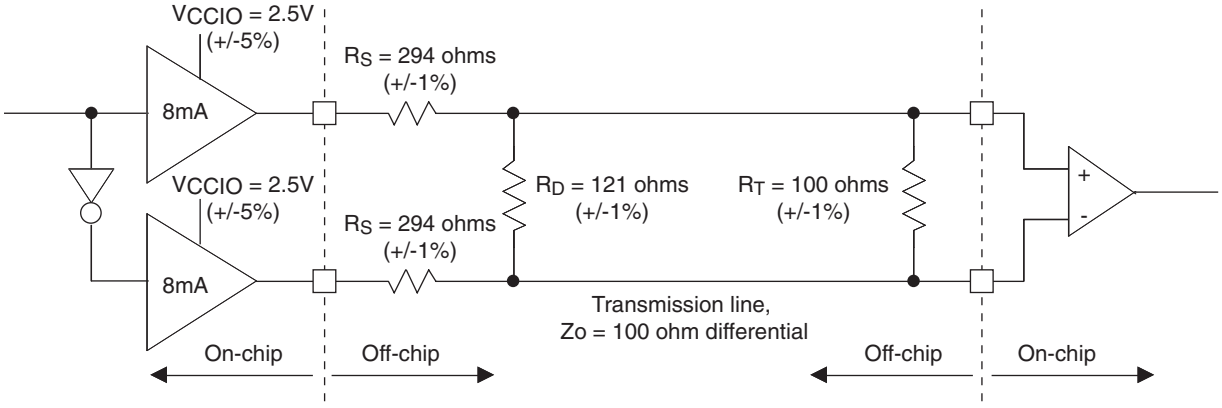


Table 3-4. RSDS DC Conditions[1]

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 LatticeECP2 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)

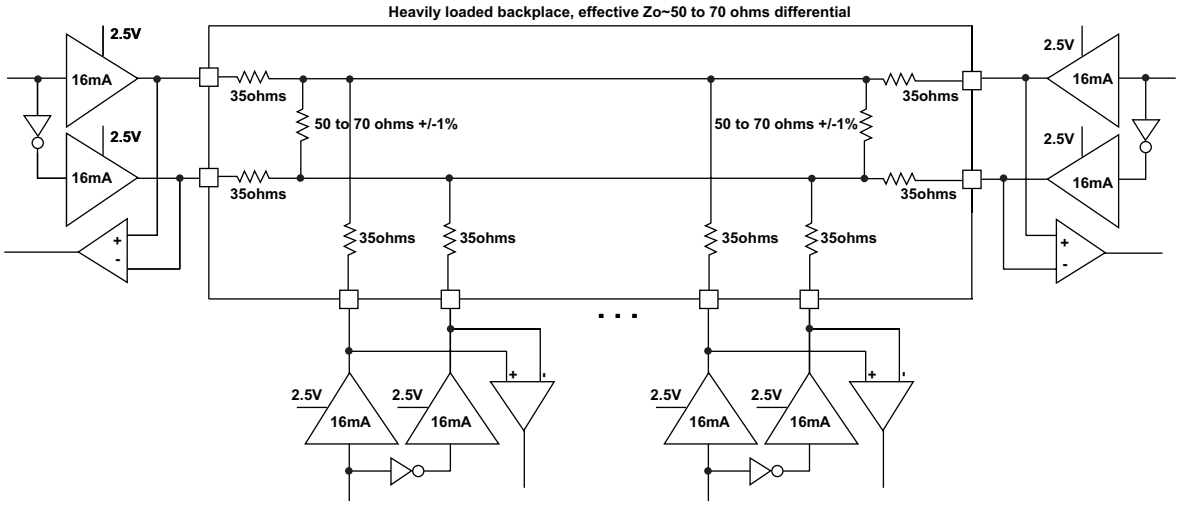


Table 3-5. MLVDS DC Conditions[1]

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	6.4	ns
32-bit Decoder	10.0	ns
64-bit Decoder	18.4	ns
4:1 MUX	3.6	ns
8:1 MUX	3.6	ns
16:1 MUX	4.1	ns
32:1 MUX	4.6	ns

Register-to-Register Performance

Function	-7 Timing	Units
Basic Functions		
16-bit Decoder	518	MHz
32-bit Decoder	335	MHz
64-bit Decoder	316	MHz
4:1 MUX	785	MHz
8:1 MUX	618	MHz
16:1 MUX	575	MHz
32:1 MUX	247	MHz
8-bit Adder	458	MHz
16-bit Adder	371	MHz
64-bit Adder	247	MHz
16-bit Counter	469	MHz
32-bit Counter	362	MHz
64-bit Counter	249	MHz
64-bit Accumulator	242	MHz
Embedded Memory Functions		
512x36 Single Port RAM, EBR Output Registers	353	MHz
1024x18 True-Dual Port RAM (Write Through or Normal, EBR Output Registers)	353	MHz
1024x18 True-Dual Port RAM (Read-Before-Write, EBR Output Registers)	280	MHz
1024x18 True-Dual Port RAM (Write Through or Normal, PLC Output Registers)	260	MHz
Distributed Memory Functions		
16x4 Pseudo-Dual Port RAM (One PFU)	864	MHz
32x2 Pseudo-Dual Port RAM	461	MHz
64x1 Pseudo-Dual Port RAM	349	MHz
DSP Functions		
18x18 Multiplier (All Registers)	479	MHz
9x9 Multiplier (All Registers)	479	MHz

Register-to-Register Performance (Continued)

Function	-7 Timing	Units
36x36 Multiply (All Registers)	422	MHz
18x18 Multiply/Accumulate (Input and Output Registers)	TBA	MHz
18x18 Multiply-Add/Sub-Sum (All Registers)	479	MHz
DSP IP Functions		
16-Tap Fully-Parallel FIR Filter	TBA	MHz
1024-pt, Radix 4, Decimation in Frequency FFT	TBA	MHz
8X8 Matrix Multiplication	TBA	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. A0.01

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.

LatticeECP2 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	LFEC2-50	—	3.49	—	3.79	—	4.10	ns
t_{SU}	Clock to Data Setup - PIO Input Register	LFEC2-50	0.24	—	0.15	—	0.09	—	ns
t_{H}	Clock to Data Hold - PIO Input Register	LFEC2-50	0.02	—	0.11	—	0.23	—	ns
t_{SU_DEL}	Clock to Data Setup - PIO Input Register with Data Input Delay	LFEC2-50		—		—		—	ns
t_{H_DEL}	Clock to Data Hold - PIO Input Register with Input Data Delay	LFEC2-50		—		—		—	ns
f_{MAX_IO}	Clock Frequency of I/O and PFU Register	LFEC2-50	—		—		—		MHz
General I/O Pin Parameters (Using Primary Clock with PLL)[1]									
t_{COPLL}	Clock to Output - PIO Output Register	LFEC2-50	—		—		—		ns
t_{SUPLL}	Clock to Data Setup - PIO Input Register	LFEC2-50		—		—		—	ns
t_{HPLL}	Clock to Data Hold - PIO Input Register	LFEC2-50		—		—		—	ns
t_{SU_DELPLL}	Clock to Data Setup - PIO Input Register with Data Input Delay	LFEC2-50		—		—		—	ns
t_{H_DELPLL}	Clock to Data Hold - PIO Input Register with Input Data Delay	LFEC2-50		—		—		—	ns
DDR[2] and DDR2[3] I/O Pin Parameters									
t_{DVADQ}	Data Valid After DQS (DDR Read)	LFEC2-50	—		—		—		UI
t_{DVEDQ}	Data Hold After DQS (DDR Read)	LFEC2-50		—		—		—	UI
t_{DQVBS}	Data Valid Before DQS	LFEC2-50		—		—		—	UI
t_{DQVAS}	Data Valid After DQS	LFEC2-50		—		—		—	UI
f_{MAX_DDR}	DDR Clock Frequency[6]	LFEC2-50	—		—		—		MHz
f_{MAX_DDR2}	DDR Clock Frequency	LFEC2-50	—		—		—		MHz
SPI4.2 I/O Pin Parameters[4]									
$t_{DVACLKSPI}$	Data Valid After CLK	LFEC2-50	—		—		—		ps
$t_{DVECLKSPI}$	Data Hold After CLK	LFEC2-50		—		—		—	ps
t_{DIASPI}	Data Invalid After Clock	LFEC2-50	—		—		—		ps
t_{DIBSPI}	Data Invalid Before Clock	LFEC2-50	—		—		—		ps
SFI4 I/O Pin Parameters[4]									
$t_{DVACLKSFI}$	Data Valid After DQS (DDR Read)	LFEC2-50	—		—		—		ps
$t_{DVECLKSFI}$	Data Hold After DQS (DDR Read)	LFEC2-50		—		—		—	ps
t_{DIASFI}	Data Invalid After Clock	LFEC2-50	—		—		—		ps
t_{DIBSFI}	Data Invalid Before Clock	LFEC2-50	—		—		—		ps
XGMII I/O Pin Parameters[5]									
$t_{SUXGMII}$	Data Setup Before Read Clock	LFEC2-50		—		—		—	ps
t_{HXGMII}	Data Hold After Read Clock	LFEC2-50		—		—		—	ps

LatticeECP2 External Switching Characteristics (Continued)

Over Recommended Operating Conditions

Parameter	Description	Device	-7		-6		-5		Units
			Min.	Max.	Min.	Max.	Min.	Max.	
$t_{DQVBCLKXGMII}$	Data Invalid Before Strobe/Clock	LFEC2-50		—		—		—	ps
$t_{DQVACLKXGMII}$	Data Invalid After Strobe/Clock	LFEC2-50		—		—		—	ps
Primary									
f_{MAX_PRI}	Frequency for Primary Clock Tree	LFEC2-50	—		—		—		MHz
t_{W_PRI}	Clock Pulse Width for Primary Clock	LFEC2-50		—		—		—	ns
t_{SKEW_PRI}	Primary Clock Skew Within a Device	LFEC2-50	—		—		—		ps
Secondary Clocks									
f_{MAX_SEC}	Frequency for Secondary Clock Tree	LFEC2-50	—		—		—		MHz
t_{W_SEC}	Clock Pulse Width for Secondary Clock	LFEC2-50		—		—		—	ns
Edge Clocks (ECK1 and ECK2)									
f_{MAX_ECK}	Frequency for Edge Clock	LFEC2-50	—		—		—		MHz
t_{W_ECK}	Clock Pulse Width for ECK	LFEC2-50		—		—		—	ns
t_{SKEW_ECK}	ECK Skew Within an Edge of the Device	LFEC2-50	—		—		—		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. SPI4.2 and SFI4 timing numbers based on LVDS25.
5. XGMII timing numbers based on HSTL class I.
6. Device supports DDR and DDR2 memory data rates down to 95MHz.

Timing v. A0.01

Figure 3-6. SPI4.2 and SFI Transmit Parameters

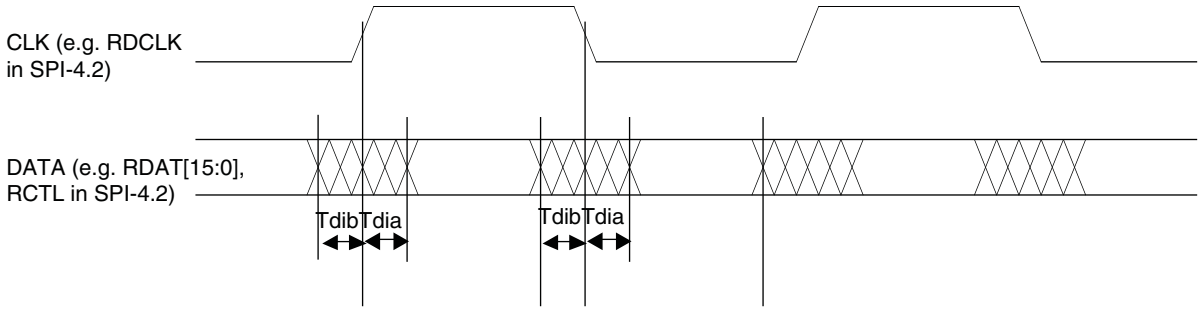


Figure 3-7. DDR1,DDR2, XGMII Transmit Parameters

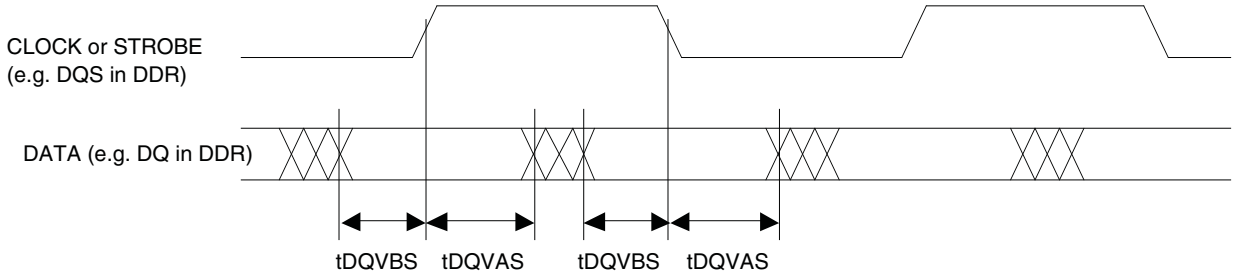


Figure 3-8. XGMII Receiver Parameters

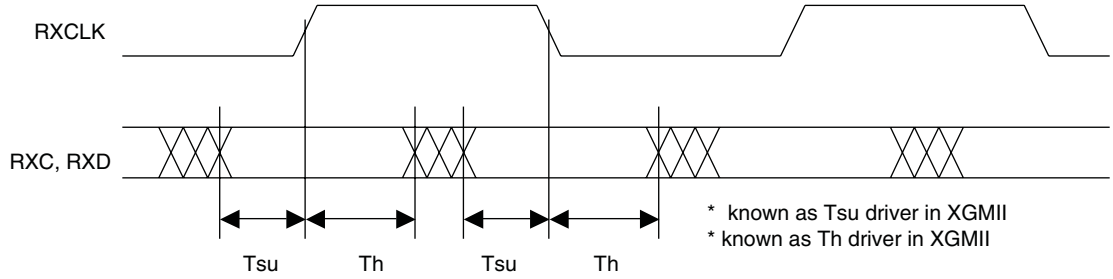
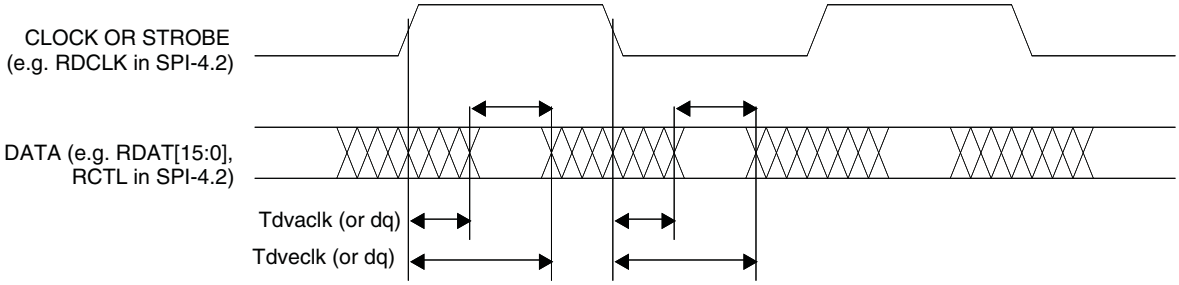


Figure 3-9. DDR1, DDR2, SPI4.2, SFI4 Receiver Parameters



LatticeECP2 Internal Switching Characteristics[1]

Over Recommended Operating Conditions

Parameter	Description	-7		-6		-5		Units
		Min.	Max.	Min.	Max.	Min.	Max.	
PFU Logic Mode Timing								
t_{LUT4_PFU}	LUT4 Delay (A to D Inputs to F Output)	—		—		—		ns
t_{LUT6_PFU}	LUT6 Delay (A to D Inputs to OFX Output)	—		—		—		ns
t_{LSR_PFU}	Set/Reset to Output (Asynchronous)	—		—		—		ns
t_{SUM_PFU}	Clock to Mux (M0, M1) Input Setup Time		—		—		—	ns
t_{HM_PFU}	Clock to Mux (M0, M1) Input Hold Time		—		—		—	ns
t_{SUD_PFU}	Clock to D Input Setup Time		—		—		—	ns
t_{HD_PFU}	Clock to D Input Hold Time		—		—		—	ns
t_{CK2Q_PFU}	Clock to Q Delay, (D-type Register Configuration)	—		—		—		ns
PFU Memory Mode Timing								
t_{CORAM_PFU}	Clock to Output Write (F Port)	—		—		—		ns
t_{SUDATA_PFU}	Data Setup Time		—		—		—	ns
t_{HDATA_PFU}	Data Hold Time		—		—		—	ns
t_{SUADDR_PFU}	Address Setup Time		—		—		—	ns
t_{HADDR_PFU}	Address Hold Time		—		—		—	ns
t_{SUWREN_PFU}	Write/Read Enable Setup Time		—		—		—	ns
t_{HWREN_PFU}	Write/Read Enable Hold Time		—		—		—	ns
PIO Input/Output Buffer Timing								
t_{IN_PIO}	Input Buffer Delay (LVCMOS25)	—		—		—		ns
t_{OUT_PIO}	Output Buffer Delay (LVCMOS25)	—		—		—		ns
t_{SUI_PIO}	Input Register Setup Time (Data Before Clock)		—		—		—	ns
t_{HI_PIO}	Input Register Hold Time (Data After Clock)		—		—		—	ns
t_{COO_PIO}	Output Register Clock to Output Delay	—		—		—		ns
t_{SUCE_PIO}	Input Register Clock Enable Setup Time		—		—		—	ns
t_{HCE_PIO}	Input Register Clock Enable Hold Time		—		—		—	ns
t_{SULSR_PIO}	Set/Reset Setup Time		—		—		—	ns
t_{HLSR_PIO}	Set/Reset Hold Time		—		—		—	ns
EBR Timing								
t_{CO_EBR}	Clock (Read) to Output from Address or Data	—		—		—		ns
t_{COO_EBR}	Clock (Write) to Output from EBR Output Register	—		—		—		ns
t_{SUDATA_EBR}	Setup Data to EBR Memory (Write Clk)		—		—		—	ns
t_{HDATA_EBR}	Hold Data to EBR Memory (Write Clk)		—		—		—	ns
t_{SUADDR_EBR}	Setup Address to EBR Memory (Write Clk)		—		—		—	ns
t_{HADDR_EBR}	Hold Address to EBR Memory (Write Clk)		—		—		—	ns
t_{SUWREN_EBR}	Setup Write/Read Enable to EBR Memory (Write/Read Clk)		—		—		—	ns
t_{HWREN_EBR}	Hold Write/Read Enable to EBR Memory (Write/Read Clk)		—		—		—	ns

LatticeECP2 Internal Switching Characteristics[1] (Continued)

Over Recommended Operating Conditions

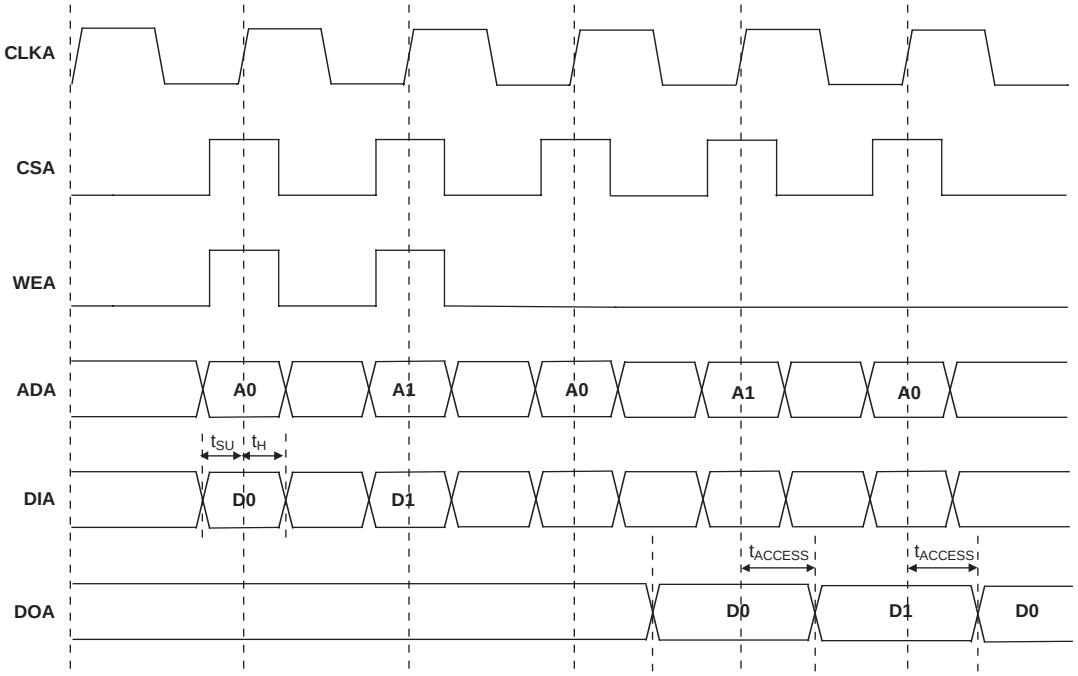
Parameter	Description	-7		-6		-5		Units
		Min.	Max.	Min.	Max.	Min.	Max.	
t_{SUCE_EBR}	Clock Enable Setup Time to EBR Output Register (Read Clk)		—		—		—	ns
t_{HCE_EBR}	Clock Enable Hold Time to EBR Output Register (Read Clk)		—		—		—	ns
t_{RSTO_EBR}	Reset To Output Delay Time from EBR Output Register (Asynchronous)	—		—		—		ns
t_{SUBE_EBR}	Byte Enable Set-Up Time to EBR Output Register		—		—		—	ns
t_{HBE_EBR}	Byte Enable Hold Time to EBR Output Register		—		—		—	ns
Dynamic Delay on Each PIO								
$t_{DEL\_ADJ\_RNG}$	Delay Adjustment Range (in Nine Delay Steps)							ps
GPLL Parameters								
t_{RSTREC_GPLL}		—		—		—		ns
t_{RSTSU_GPLL}			—		—		—	ns
SPLL Parameters								
t_{RSTREC_SPLL}		—		—		—		ns
t_{RSTSU_SPLL}			—		—		—	ns
DSP Block Timing								
t_{SUI_DSP}	Input Register Setup Time		—		—		—	ns
t_{HI_DSP}	Input Register Hold Time		—		—		—	ns
t_{SUP_DSP}	Pipeline Register Setup Time		—		—		—	ns
t_{HP_DSP}	Pipeline Register Hold Time		—		—		—	ns
t_{SUO_DSP}	Output Register Setup Time		—		—		—	ns
t_{HO_DSP}	Output Register Hold Time		—		—		—	ns
t_{COI_DSP}[2]	Input Register Clock to Output Time	—		—		—		ns
t_{COP_DSP}[2]	Pipeline Register Clock to Output Time	—		—		—		ns
t_{COO_DSP}[2]	Output Register Clock to Output Time	—		—		—		ns
$t_{SUADSUB}$	AdSub Input Register Setup Time		—		—		—	ns
t_{HADSUB}	AdSub Input Register Hold Time		—		—		—	ns

1. Internal parameters are characterized but not tested on every device.
2. These parameters include the Adder Subtractor block in the path.

Timing v.

Timing Diagrams

Figure 3-10. 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 3-11. Read/Write Mode with Input and Output Registers

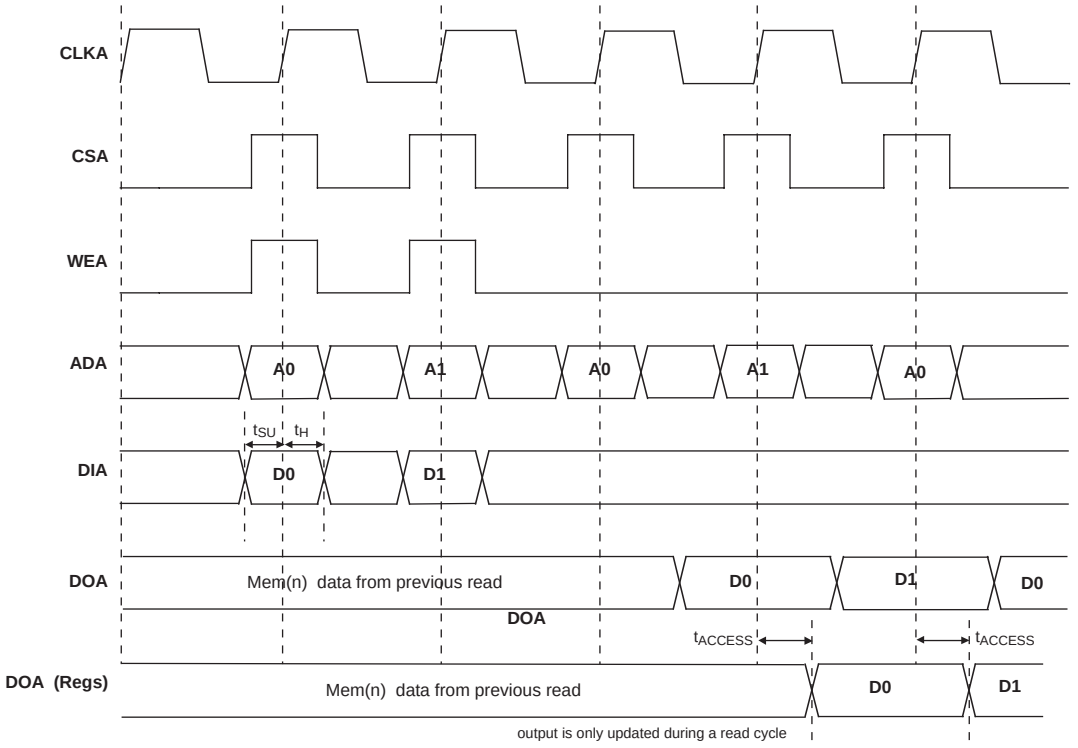
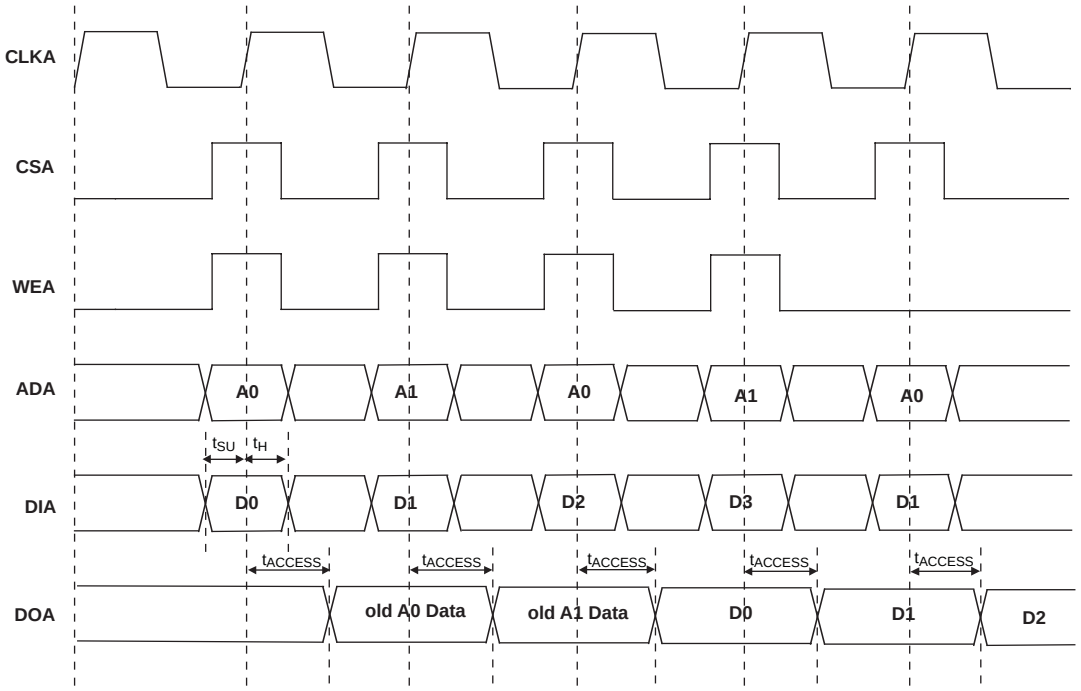
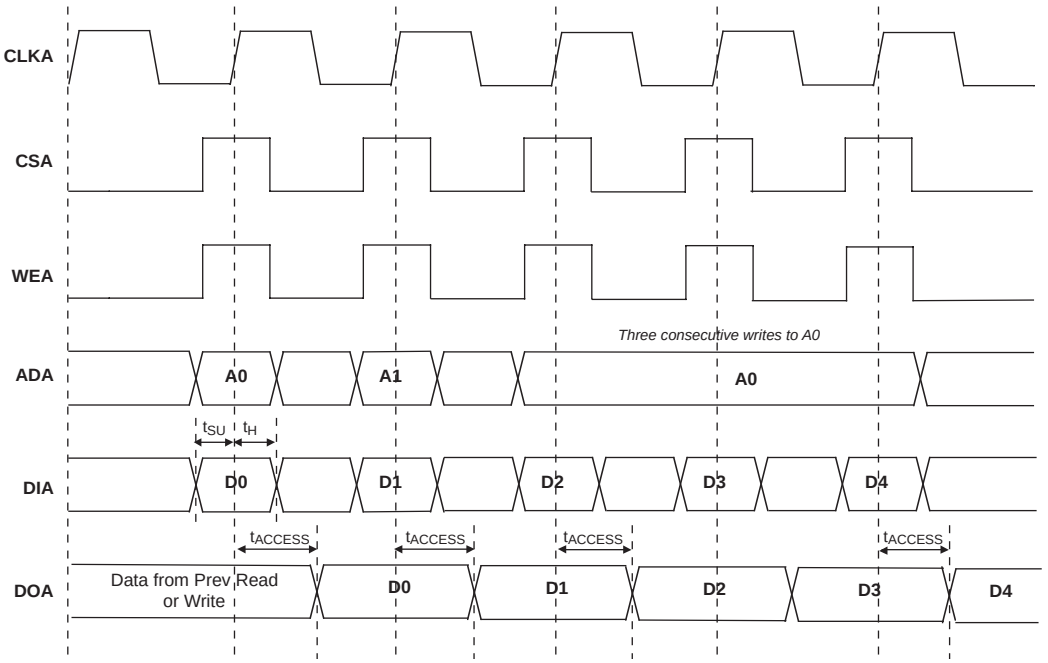


Figure 3-12. Read Before Write (SP Read/Write on Port A, Input Registers Only)



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 3-13. 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.

LatticeECP2 Family Timing Adders[1, 2, 3]

Over Recommended Operating Conditions

Buffer Type	Description	-7	-6	-5	Units
Input Adjusters					
LVDS25	LVDS				ns
MLVDS	MLVDS				ns
RSDS	RSDS				ns
BLVDS25	BLVDS				ns
LVPECL33	LVPECL				ns
HSTL18_I	HSTL_18 class I				ns
HSTL18_II	HSTL_18 class II				ns
HSTL18D_I	Differential HSTL 18 class I				ns
HSTL18D_II	Differential HSTL 18 class II				ns
HSTL15_I	HSTL_15 class I				ns
HSTL15D_I	Differential HSTL 15 class I				ns
SSTL33_I	SSTL_3 class I				ns
SSTL33_II	SSTL_3 class II				ns
SSTL33D_I	Differential SSTL_3 class I				ns
SSTL33D_II	Differential SSTL_3 class II				ns
SSTL25_I	SSTL_2 class I				ns
SSTL25_II	SSTL_2 class II				ns
SSTL25D_I	Differential SSTL_2 class I				ns
SSTL25D_II	Differential SSTL_2 class II				ns
SSTL18_I	SSTL_18 class I				ns
SSTL18_II	SSTL_18 class II				ns
SSTL18D_I	Differential SSTL_18 class I				ns
SSTL18D_II	Differential SSTL_18 class II				ns
LVTTL33	LVTTL				ns
LVCMOS33	LVCMOS 3.3				ns
LVCMOS25	LVCMOS 2.5				ns
LVCMOS18	LVCMOS 1.8				ns
LVCMOS15	LVCMOS 1.5				ns
LVCMOS12	LVCMOS 1.2				ns
PCI33	3.3V PCI				ns
Output Adjusters					
LVDS25E[4]	LVDS 2.5 E				ns
LVDS25	LVDS 2.5				ns
RSDS[4]	RSDS				ns
MLVDS[4]	MLVDS				ns
BLVDS25[4]	BLVDS 2.5				ns
LVPECL33[4]	LVPECL 3.3				ns
HSTL18_I	HSTL_18 class I				ns
HSTL18_II	HSTL_18 class II				ns
HSTL18D_I	Differential HSTL 18 class I				ns
HSTL18D_II	Differential HSTL 18 class II				ns

LatticeECP2 Family Timing Adders[1, 2, 3] (Continued)

Over Recommended Operating Conditions

Buffer Type	Description	-7	-6	-5	Units
HSTL15_I	HSTL_15 class I				ns
HSTL15D_I	Differential HSTL 15 class I				ns
SSTL33_I	SSTL_3 class I				ns
SSTL33_II	SSTL_3 class II				ns
SSTL33D_I	Differential SSTL_3 class I				ns
SSTL33D_II	Differential SSTL_3 class II				ns
SSTL25_I	SSTL_2 class I				ns
SSTL25_II	SSTL_2 class II				ns
SSTL25D_I	Differential SSTL_2 class I				ns
SSTL25D_II	Differential SSTL_2 class II				ns
SSTL18_I	SSTL_1.8 class I				ns
SSTL18_II	SSTL_1.8 class II				ns
SSTL18D_I	Differential SSTL_1.8 class I				ns
SSTL18D_II	Differential SSTL_1.8 class II				ns
LVTTL33_4mA	LVTTL 4mA drive				ns
LVTTL33_8mA	LVTTL 8mA drive				ns
LVTTL33_12mA	LVTTL 12mA drive				ns
LVTTL33_16mA	LVTTL 16mA drive				ns
LVTTL33_20mA	LVTTL 20mA drive				ns
LVCMOS33_4mA	LVCMOS 3.3 4mA drive, fast slew rate				ns
LVCMOS33_8mA	LVCMOS 3.3 8mA drive, fast slew rate				ns
LVCMOS33_12mA	LVCMOS 3.3 12mA drive, fast slew rate				ns
LVCMOS33_16mA	LVCMOS 3.3 16mA drive, fast slew rate				ns
LVCMOS33_20mA	LVCMOS 3.3 20mA drive, fast slew rate				ns
LVCMOS25_4mA	LVCMOS 2.5 4mA drive, fast slew rate				ns
LVCMOS25_8mA	LVCMOS 2.5 8mA drive, fast slew rate				ns
LVCMOS25_12mA	LVCMOS 2.5 12mA drive, fast slew rate				ns
LVCMOS25_16mA	LVCMOS 2.5 16mA drive, fast slew rate				ns
LVCMOS25_20mA	LVCMOS 2.5 20mA drive, fast slew rate				ns
LVCMOS18_4mA	LVCMOS 1.8 4mA drive, fast slew rate				ns
LVCMOS18_8mA	LVCMOS 1.8 8mA drive, fast slew rate				ns
LVCMOS18_12mA	LVCMOS 1.8 12mA drive, fast slew rate				ns
LVCMOS18_16mA	LVCMOS 1.8 16mA drive, fast slew rate				ns
LVCMOS15_4mA	LVCMOS 1.5 4mA drive, fast slew rate				ns
LVCMOS15_8mA	LVCMOS 1.5 8mA drive, fast slew rate				ns
LVCMOS12_2mA	LVCMOS 1.2 2mA drive, fast slew rate				ns
LVCMOS12_6mA	LVCMOS 1.2 6mA drive, fast slew rate				ns
LVCMOS33_4mA	LVCMOS 3.3 4mA drive, slow slew rate				ns
LVCMOS33_8mA	LVCMOS 3.3 8mA drive, slow slew rate				ns
LVCMOS33_12mA	LVCMOS 3.3 12mA drive, slow slew rate				ns
LVCMOS33_16mA	LVCMOS 3.3 16mA drive, slow slew rate				ns
LVCMOS33_20mA	LVCMOS 3.3 20mA drive, slow slew rate				ns
LVCMOS25_4mA	LVCMOS 2.5 4mA drive, slow slew rate				ns

LatticeECP2 Family Timing Adders[1, 2, 3] (Continued)

Over Recommended Operating Conditions

Buffer Type	Description	-7	-6	-5	Units
LVCMOS25_8mA	LVCMOS 2.5 8mA drive, slow slew rate				ns
LVCMOS25_12mA	LVCMOS 2.5 12mA drive, slow slew rate				ns
LVCMOS25_16mA	LVCMOS 2.5 16mA drive, slow slew rate				ns
LVCMOS25_20mA	LVCMOS 2.5 20mA drive, slow slew rate				ns
LVCMOS18_4mA	LVCMOS 1.8 4mA drive, slow slew rate				ns
LVCMOS18_8mA	LVCMOS 1.8 8mA drive, slow slew rate				ns
LVCMOS18_12mA	LVCMOS 1.8 12mA drive, slow slew rate				ns
LVCMOS18_16mA	LVCMOS 1.8 16mA drive, slow slew rate				ns
LVCMOS15_4mA	LVCMOS 1.5 4mA drive, slow slew rate				ns
LVCMOS15_8mA	LVCMOS 1.5 8mA drive, slow slew rate				ns
LVCMOS12_2mA	LVCMOS 1.2 2mA drive, slow slew rate				ns
LVCMOS12_6mA	LVCMOS 1.2 6mA drive, slow slew rate				ns
PCI33	3.3V PCI				ns

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

Timing v.

sysCLOCK GPLL Timing

Over Recommended Operating Conditions

Parameter	Description	Conditions	Min.	Typ.	Max.	Units
f_{IN}	Input Clock Frequency (CLKI, CLKFB)	Without external capacitor		—	420	MHz
		With external capacitor		—		MHz
f_{OUT}	Output Clock Frequency (CLKOP, CLKOS)	Without external capacitor		—	420	MHz
		With external capacitor		—		MHz
f_{OUT2}	K-Divider Output Frequency	Without external capacitor		—	840	MHz
		With external capacitor		—		MHz
f_{VCO}	PLL VCO Frequency			—		MHz
f_{PFD}	Phase Detector Input Frequency			—		MHz
AC Characteristics						
t_{DT}	Output Clock Duty Cycle	Default duty cycle selected [3]				%
t_{PH} [4]	Output Phase Accuracy		—	—		
t_{OPJIT} [1]	Output Clock Period Jitter	f_{OUT} > 100 MHz	—	—		ps
		f_{OUT} < 100 MHz	—	—		UIPP
t_{SK}	Input Clock to Output Clock Skew	N/M = integer	—	—		ps
t_W	Output Clock Pulse Width	At 90% or 10%		—	—	ns
t_{LOCK} [2]	PLL Lock-in Time	Without external capacitor				µs
		With external capacitor	—	—		µs
t_{PA}	Programmable Delay Unit					ps
t_{IPJIT}	Input Clock Period Jitter		—	—		ps
t_{FBKDLY}	External Feedback Delay		—	—		ns
t_{HI}	Input Clock High Time	90% to 90%		—	—	ns
t_{LO}	Input Clock Low Time	10% to 10%		—	—	ns
t_{RST}	Reset Signal Pulse Width (RESETM/ RESETK)					ns
	Reset Signal Pulse Width (CNTRST)	Without capacitor				ns
	Reset Signal Pulse Width (CNTRST)	With capacitor				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.

sysCLOCK SPLL Timing

Over Recommended Operating Conditions

Parameter	Description	Conditions	Min.	Typ.	Max.	Units
f_{IN}	Input Clock Frequency (CLKI, CLKFB)	Without external capacitor		—	420	MHz
		With external capacitor		—		MHz
f_{OUT}	Output Clock Frequency (CLKOP, CLKOS)	Without external capacitor		—	420	MHz
		With external capacitor		—		MHz
f_{OUT2}	K-Divider Output Frequency (CLKOK)	Without external capacitor		—	840	MHz
		With external capacitor		—		MHz
f_{VCO}	PLL VCO Frequency			—		MHz
f_{PFD}	Phase Detector Input Frequency			—		MHz
AC Characteristics						
t_{DT}	Output Clock Duty Cycle	Default Duty Cycle Selected[3]				%
t_{PH}[4]	Output Phase Accuracy		—	—		
t_{OPJIT}[1]	Output Clock Period Jitter	f_{OUT} > 100 MHz	—	—		ps
		f_{OUT} < 100 MHz	—	—		UIPP
t_{SK}	Input Clock to Output Clock Skew	N/M = Integer	—	—		ps
t_W	Output Clock Pulse Width	At 90% or 10%		—		ns
t_{LOCK}[2]	PLL Lock-in Time	Without external capacitor	—	—		µs
		With external capacitor	—	—		µs
t_{PA}	Programmable Delay Unit		—	—		ps
t_{IPJIT}	Input Clock Period Jitter		—	—		ps
t_{FBKDLY}	External Feedback Delay		—	—		ns
t_{HI}	Input Clock High Time	90% to 90%		—	—	ns
t_{LO}	Input Clock Low Time	10% to 10%		—	—	ns
t_{RST}	Reset Signal Pulse Width (RESETM/ RESETK)					ns
	Reset Signal Pulse Width (CNTRST)	Without external capacitor				ns
	Reset Signal Pulse Width (CNTRST)	With external capacitor				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.

DLL Timing

Over Recommended Operating Conditions

Parameter	Description	Min.	Typ.	Max.	Units
f_{REF}	Input reference clock frequency (on-chip or off-chip)		—	420	MHz
f_{FB}	Feedback clock frequency (on-chip or off-chip)		—		MHz
f_{CLKOP}[1]	Output clock frequency, CLKOP		—	420	MHz
f_{CLKOS}[2]	Output clock frequency, CLKOS		—		MHz
t_{PJIT}	Output clock period jitter (clean input)		—	420	ps p-p
t_{CYJIT}	Output clock cycle to cycle jitter (clean input)		—		ps p-p
t_{DUTY}	Output clock duty cycle (at 50% levels, 50% duty cycle input clock, 50% duty cycle circuit turned off, time reference delay mode)				%
$t_{DUTYTRD}$	Output clock duty cycle (at 50% levels, arbitrary duty cycle input clock, 50% duty cycle circuit enabled, time reference delay mode)				%
$t_{DUTYCIR}$	Output clock duty cycle (at 50% levels, arbitrary duty cycle input clock, 50% duty cycle circuit enabled, clock injection removal mode)				%
t_{SKEW}[3]	Output clock to clock skew between two outputs with the same phase setting	—	—		ps
t_{PHASE}	Phase error measured at device pads using off-chip reference clock and feedback clocks	—	—		ps
t_{PWH}	Input clock minimum pulse width high (at 80% level)		—	—	ps
t_{PWL}	Input clock minimum pulse width low (at 20% level)		—	—	ps
t_R, t_F	Input clock rise and fall time (20% to 80% levels)	—	—		ps
t_{INSTB}	Input clock period jitter		—	—	ps
t_{LOCK}	DLL lock time (input stable until assertion of LOCK)		—	—	cycles
t_{RSWD}	Digital reset minimum pulse width (at 80% level)		—	—	ns
t_{PA}	Delay step size				ps
t_{RANGE1}	Max. delay setting for single delay block (144 taps)				ns
t_{RANGE4}	Max. delay setting for four chained delay blocks				ns

1. CLKOP runs at the same frequency as the input clock.
2. CLKOS minimum frequency is obtained with divide by 4.
3. This is intended to be a “path-matching” design guideline and is not a measurable specification.

LatticeECP2 sysCONFIG Port Timing Specifications

Over Recommended Operating Conditions

Parameter	Description	Min.	Max.	Units
sysCONFIG Byte Data Flow				
t_{SUCBDI}	Byte D[0:7] Setup Time to CCLK		—	ns
t_{HCBDI}	Byte D[0:7] Hold Time to CCLK		—	ns
t_{CODO}	CCLK to DOUT in Flowthrough Mode	—		ns
t_{SUCS}	CSN[0:1] Setup Time to CCLK		—	ns
t_{HCS}	CSN[0:1] Hold Time to CCLK		—	ns
t_{SUWD}	Write Signal Setup Time to CCLK		—	ns
t_{HWD}	Write Signal Hold Time to CCLK		—	ns
t_{DCB}	CCLK to BUSY Delay Time	—		ns
t_{CORD}	CCLK to Out for Read Data	—		ns
sysCONFIG Byte Slave Clocking				
t_{BSCH}	Byte Slave CCLK Minimum High Pulse		—	ns
t_{BSCL}	Byte Slave CCLK Minimum Low Pulse		—	ns
t_{BSCYC}	Byte Slave CCLK Cycle Time		—	ns
sysCONFIG Serial (Bit) Data Flow				
t_{SUSCDI}	DI Setup Time to CCLK Slave Mode		—	ns
t_{HSCDI}	DI Hold Time to CCLK Slave Mode		—	ns
t_{CODO}	CCLK to DOUT in Flowthrough Mode	—		ns
t_{SUMCDI}	DI Setup Time to CCLK Master Mode		—	ns
t_{HMCDI}	DI Hold Time to CCLK Master Mode		—	ns
sysCONFIG Serial Slave Clocking				
t_{SSCH}	Serial Slave CCLK Minimum High Pulse		—	ns
t_{SSCL}	Serial Slave CCLK Minimum Low Pulse		—	ns
sysCONFIG POR, Initialization and Wake-up				
t_{ICFG}	Minimum Vcc to INITN High	—		ms
t_{VMC}	Time from t_{ICFG} to Valid Master CCLK	—		us
t_{PRGMRJ}	PROGRAMN Pin Pulse Rejection	—		ns
t_{PRGM}	PROGRAMN Low Time to Start Configuration		—	ns
t_{DINIT}	PROGRAMN High to INITN High Delay	—		ms
$t_{DPPINIT}$	Delay Time from PROGRAMN Low to INITN Low	—		ns
$t_{DPPDONE}$	Delay Time from PROGRAMN Low to DONE Low	—		ns
t_{IODISS}	User I/O Disable from PROGRAMN Low	—		ns
t_{IOENSS}	User I/O Enabled Time from CCLK Edge During Wake-up Sequence	—		ns
t_{MWC}	Additional Wake Master Clock Signals after DONE Pin High		—	cycles
sysCONFIG SPI Port				
t_{CFGX}	INITN High to CCLK Low	—		µs
t_{CSSPI}	INITN High to CSSPIN Low	—		us
t_{CSCCLK}	CCLK Low before CSSPIN Low		—	ns
t_{SOCDO}	CCLK Low to Output Valid	—		ns
t_{SOE}	CSSPIN[0:1] Active Setup Time		—	ns
t_{CSPID}	CSSPIN[0:1] Low to First CCLK Edge Setup Time			ns

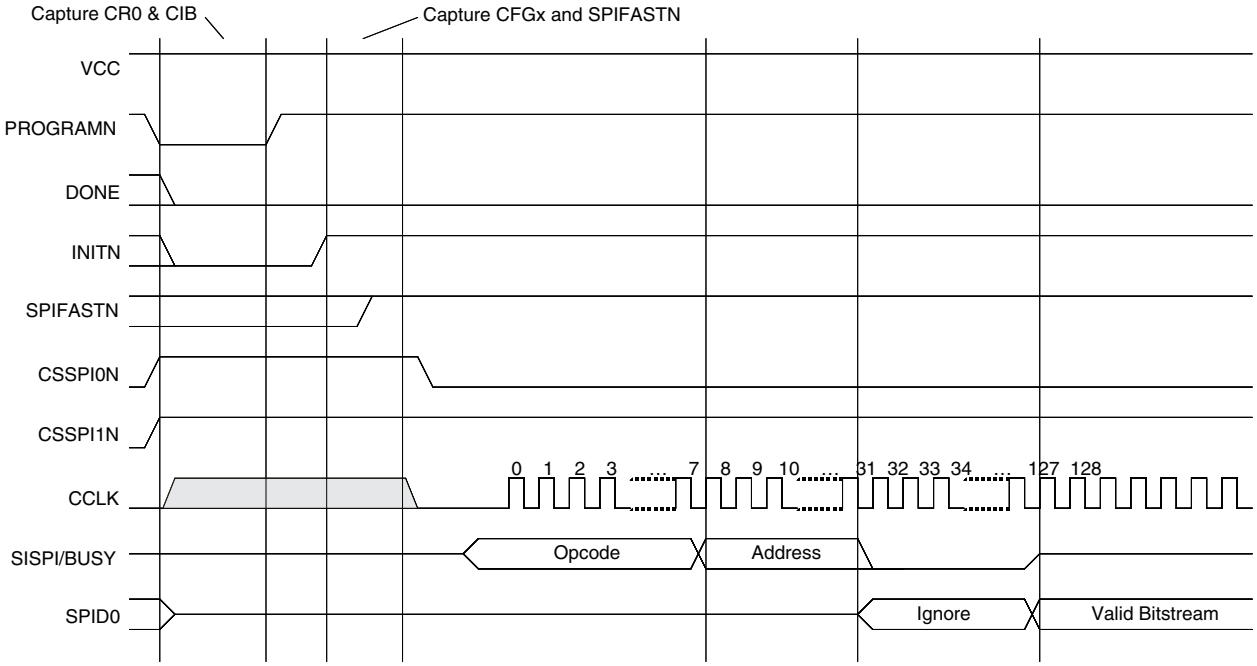
LatticeECP2 sysCONFIG Port Timing Specifications (Continued)

Over Recommended Operating Conditions

Parameter	Description	Min.	Max.	Units
f_{MAXSPI}	Max. CCLK Frequency - SPI Flash Read Opcode (0x03) (SPIFASTN = 1)	—		MHz
	Max. CCLK Frequency - SPI Flash Fast Read Opcode (0x0B) (SPIFASTN = 0)	—		MHz
t_{SUSPI}	SOSPI Data Setup Time Before CCLK		—	ns
t_{HSPI}	SOSPI Data Hold Time After CCLK		—	ns
Master Clock Frequency		Selected value -30%	Selected value +30%	MHz
Duty Cycle				%

Timing v.

Figure 3-14. SPI/SPIm Configuration Waveforms



JTAG Port Timing Specifications

Over Recommended Operating Conditions

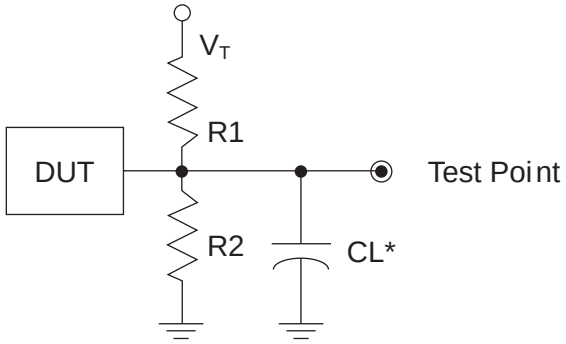
Symbol	Parameter	Min	Max	Units
f_{MAX}	TCK clock frequency	—		MHz
t_{BTCP}	TCK [BSCAN] clock pulse width		—	ns
t_{BTCPH}	TCK [BSCAN] clock pulse width high		—	ns
t_{BTCPL}	TCK [BSCAN] clock pulse width low		—	ns
t_{BTS}	TCK [BSCAN] setup time		—	ns
t_{BTH}	TCK [BSCAN] hold time		—	ns
t_{BTRF}	TCK [BSCAN]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

Timing v.

Switching Test Conditions

Figure 3-15 shows the output test load that is used for AC testing. The specific values for resistance, capacitance, voltage, and other test conditions are shown in Table 3-6.

Figure 3-15. Output Test Load, LVTTL and LVCMOS Standards



*CL Includes Test Fixture and Probe Capacitance

Table 3-6. Test Fixture Required Components, Non-Terminated Interfaces

Test Condition	R_1	R_2	C_L	Timing Ref.	V_T
LVTTL and other LVCMOS settings (L -> H, H -> L)	∞	∞	0pF	LVCMOS 3.3 = 1.5V	—
				LVCMOS 2.5 = V_{CCIO}/2	—
				LVCMOS 1.8 = V_{CCIO}/2	—
				LVCMOS 1.5 = V_{CCIO}/2	—
				LVCMOS 1.2 = V_{CCIO}/2	—
LVCMOS 2.5 I/O (Z -> H)	∞	1MΩ		V_{CCIO}/2	—
LVCMOS 2.5 I/O (Z -> L)	1MΩ	∞		V_{CCIO}/2	V_{CCIO}
LVCMOS 2.5 I/O (H -> Z)	∞	100		V_{OH} - 0.10	—
LVCMOS 2.5 I/O (L -> Z)	100	∞		V_{OL} + 0.10	V_{CCIO}

Note: Output test conditions for all other interfaces are determined by the respective standards.

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_{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.
XRES	—	10K ohm +/-1% resistor must be connected between this pad and ground.
PLL, DLL and Clock Functions (Used as user programmable I/O pins when not in use for PLL or clock pins)		
[LOC][num]_GPLL[T, C]_IN_A	I	General Purpose PLL (GPLL) input pads: ULM, LLM, URM, LRM, 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: ULM, LLM, URM, LRM, num = row from center, T = true and C = complement, index A,B,C...at each side.
[LOC][num]_SPLL[T, C]_IN_A	I	Secondary PLL (SPLL) input pads: ULM, LLM, URM, LRM, num = row from center, T = true and C = complement, index A,B,C...at each side.
[LOC][num]_SPLL[T, C]_FB_A	I	Optional feedback (SPLL) input pads: ULM, LLM, URM, LRM, num = row from center, T = true and C = complement, index A,B,C...at each side.
[LOC][num]_DLL[T, C]_IN_A	I	DLL input pads: ULM, LLM, URM, LRM, num = row from center, T = true and C = complement, index A,B,C...at each side.
[LOC][num]_DLL[T, C]_FB_A	I	Optional feedback (DLL) input pads: ULM, LLM, URM, LRM, 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.

Signal Descriptions (Cont.)

Signal Name	I/O	Description
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.
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[2:0]	I	Mode pins used to specify configuration mode values latched on rising edge of INITN. During configuration, a pull-up is enabled. These are dedicated pins.
INITN	I/O	Open Drain pin. Indicates the FPGA is ready to be configured. During configuration, a pull-up is enabled. It is a dedicated pin.
PROGRAMN	I	Initiates configuration sequence when asserted low. This pin always has an active pull-up. This is a dedicated pin.
DONE	I/O	Open Drain pin. Indicates that the configuration sequence is complete, and the startup sequence is in progress. This is a dedicated pin.
CCLK	I/O	Configuration Clock for configuring an FPGA in sysCONFIG mode.
BUSY/SISPI	I/O	Read control command in SPI3 or SPIX mode.
CSN	I	sysCONFIG chip select (active low). During configuration, a pull-up is enabled.
CS1N	I	sysCONFIG chip select (active low). During configuration, a pull-up is enabled.
WRITEN	I	Write Data on Parallel port (active low).
D[7:0]/SPID[0:7]	I/O	sysCONFIG Port Data I/O.
DOUT/CSON	O	Output for serial configuration data (rising edge of CCLK) when using sysCONFIG port.
DI/CSSPIN	I/O	Input for serial configuration data (clocked with CCLK) when using sysCONFIG port. During configuration, a pull-up is enabled. Output when used in SPI/SPIX modes.

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 Bottom Edge 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 15 bits of data for the left and right edges and up to 17 bits of data for the bottom edge. 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		Package	
		484 fpBGA	672 fpBGA
Single Ended User I/O			—
Differential Pair User I/O			—
Configuration	Dedicated		—
	Muxed		—
TAP			—
Dedicated (total without supplies)			—
V_{CC}			20
V_{CCAUX}			16
V_{CCIO}	Bank0		5
	Bank1		5
	Bank2		5
	Bank3		5
	Bank4		5
	Bank5		5
	Bank6		5
	Bank7		5
	Bank8		2
GND			72
NC			3
Single Ended/ Differential I/O per Bank	Bank0		—
	Bank1		—
	Bank2		—
	Bank3		—
	Bank4		—
	Bank5		—
	Bank6		—
	Bank7		—
V_{CCJ}			—

Power Supply and NC Connections

Signal	484 fpBGA	672 fpBGA
VCC		L12, L13, L14, L15, M11, M12, M15, M16, N11, N16, P11, P16, R11, R12, R15, R16, T12, T13, T14, T15
VCCIO0		D11, D6, G9, J12, K12
VCCIO1		D16, D21, G18, J15, K15
VCCIO2		F23, J20, L23, M17, M18
VCCIO3		AA23, R17, R18, T23, V20
VCCIO4		AC16, AC21, U15, V15, Y18
VCCIO5		AC11, AC6, U12, V12, Y9
VCCIO6		AA4, R10, R9, T4, V7
VCCIO7		F4, J7, L4, M10, M9
VCCIO8		AE25, V18
VCCJ		AB5
VCCAUX		J10, J11, J16, J17, K18, L18, T18, U18, V16, V17, V10, V11, T9, U9, K9, L9
GND		A2, A25, AA18, AA24, AA3, AA9, AD11, AD16, AD21, AD6, AE1, AE26, AF2, AF25, B1, B26, C11, C16, C21, C6, F18, F24, F3, F9, J13, J14, J21, J6, K10, K11, K13, K14, K16, K17, L10, L11, L16, L17, L24, L3, M13, M14, N10, N12, N13, N14, N15, N17, P10, P12, P13, P14, P15, P17, R13, R14, T10, T11, T16, T17, T24, T3, U10, U11, U13, U14, U16, U17, V13, V14, V21, V6
NC		N6, P24, M3

ECP2-50 Logic Signal Connections: 672 fpBGA

Ball Number	Ball Function	Bank	Dual Function	Differential
D2	PL2A	7	VREF2_7	T*
D1	PL2B	7	VREF1_7	C*
GND	GNDIO	7		
F6	PL5A	7		T
F5	PL5B	7		C
VCCIO	VCCIO	7		
E4	PL6A	7		T*
E3	PL6B	7		C*
VCC	VCC	7		
E2	PL7A	7		T
E1	PL7B	7		C
GND	GNDIO	7		
GND	GND	7		
H6	PL8A	7	LDQS8	T*
H5	PL8B	7		C*
F2	PL9A	7		T
VCCIO	VCCIO	7		
F1	PL9B	7		C
H8	PL10A	7		T*
J9	PL10B	7		C*
G4	PL11A	7		T
GND	GNDIO	7		
G3	PL11B	7		C
H7	PL12A	7		T*
VCCAUX	VCCAUX	7		
J8	PL12B	7		C*
G2	PL13A	7		T
G1	PL13B	7		C
H3	PL14A	7		T*
VCCIO	VCCIO	7		
H4	PL14B	7		C*
J5	PL15A	7		T
VCC	VCC	7		
J4	PL15B	7		C
J3	PL16A	7	LDQS16	T*
GND	GNDIO	7		
GND	GND	7		
K4	PL16B	7		C*
H1	PL17A	7		T
H2	PL17B	7		C
VCCIO	VCCIO	7		
K6	PL18A	7		T*
K7	PL18B	7		C*

ECP2-50 Logic Signal Connections: 672 fpBGA (Cont.)

Ball Number	Ball Function	Bank	Dual Function	Differential
J1	PL19A	7		T
J2	PL19B	7		C
GND	GNDIO	7		
VCCIO	VCCIO	7		
VCC	VCC	7		
K3	PL23A	7		T
K2	PL23B	7		C
GND	GNDIO	7		
GND	GND	7		
K1	PL24A	7	LDQS24	T*
L2	PL24B	7		C*
L1	PL25A	7	LUM0_SPLLT_IN_A	T
VCCIO	VCCIO	7		
M2	PL25B	7	LUM0_SPLLC_IN_A	C
M1	PL26A	7	LUM0_SPLLT_FB_A	T
N2	PL26B	7	LUM0_SPLLC_FB_A	C
GND	GNDIO	7		
M8	LUM0_VCCPLL	7		
GND	GNDAUX	7		
VCCAUX	VCCAUX	7		
VCCIO	VCCIO	7		
VCC	VCC	7		
GND	GNDIO	7		
GND	GND	7		
VCCIO	VCCIO	7		
GND	GNDIO	7		
N1	PL37A	7		*
L8	PL38A	7		T
K8	PL38B	7		C
VCCIO	VCCIO	7		
L6	PL39A	7		T*
K5	PL39B	7		C*
VCC	VCC	7		
L7	PL40A	7		T
L5	PL40B	7		C
GND	GNDIO	7		
GND	GND	7		
P1	PL41A	7	LDQS41	T*
P2	PL41B	7		C*
M6	PL42A	7		T
VCCIO	VCCIO	7		
N8	PL42B	7		C
R1	PL43A	7		T*
R2	PL43B	7		C*

ECP2-50 Logic Signal Connections: 672 fpBGA (Cont.)

Ball Number	Ball Function	Bank	Dual Function	Differential
M7	PL44A	7	PCLKT7_0	T
GND	GNDIO	7		
N9	PL44B	7	PCLKC7_0	C
GND	GNDAUX	7		
M4	PL46A	6	PCLKT6_0	T*
VCCAUX	VCCAUX	6		
M5	PL46B	6	PCLKC6_0	C*
N7	PL47A	6	VREF2_6	T
P9	PL47B	6	VREF1_6	C
N3	PL48A	6		T*
VCCIO	VCCIO	6		
N4	PL48B	6		C*
N5	PL49A	6		T
VCC	VCC	6		
P7	PL49B	6		C
T1	PL50A	6	LDQS50	T*
GND	GNDIO	6		
GND	GND	6		
T2	PL50B	6		C*
P8	PL51A	6		T
P6	PL51B	6		C
VCCIO	VCCIO	6		
P5	PL52A	6		T*
P4	PL52B	6		C*
U1	PL53A	6		T
V1	PL53B	6		C
GND	GNDIO	6		
P3	PL54A	6		T*
R3	PL54B	6		C*
R4	PL55A	6		T
U2	PL55B	6		C
VCCIO	VCCIO	6		
V2	PL56A	6		T*
W2	PL56B	6		C*
VCC	VCC	6		
T6	PL57A	6		T
R5	PL57B	6		C
GND	GNDIO	6		
GND	GND	6		
R6	PL58A	6	LDQS58	T*
R7	PL58B	6		C*
W1	PL59A	6		T
VCCIO	VCCIO	6		
Y2	PL59B	6		C

ECP2-50 Logic Signal Connections: 672 fpBGA (Cont.)

Ball Number	Ball Function	Bank	Dual Function	Differential
Y1	PL60A	6	LLM0_GDLLT_IN_A	T*
AA2	PL60B	6	LLM0_GDLLC_IN_A	C*
T5	PL61A	6	LLM0_GDLLT_FB_A	T
GND	GNDIO	6		
T7	PL61B	6	LLM0_GDLLC_FB_D	C
GND	GNDAUX	6		
R8	LLM0_VCCPLL	6		
T8	LLM0_PLLCAP	6		
U3	PL63A	6	LLM0_GPLLT_IN_A	T*
VCCAUX	VCCAUX	6		
U4	PL63B	6	LLM0_GPLLC_IN_A	C*
V3	PL64A	6	LLM0_GPLLT_FB_A	T
U5	PL64B	6	LLM0_GPLLC_FB_A	C
V4	PL65A	6		T*
VCCIO	VCCIO	6		
V5	PL65B	6		C*
Y3	PL66A	6		T
VCC	VCC	6		
Y4	PL66B	6		C
W3	PL67A	6	LDQS67	T*
GND	GNDIO	6		
GND	GND	6		
W4	PL67B	6		C*
AA1	PL68A	6		T
AB1	PL68B	6		C
VCCIO	VCCIO	6		
U8	PL69A	6		T*
U7	PL69B	6		C*
V8	PL70A	6		T
U6	PL70B	6		C
GND	GNDIO	6		
W6	PL71A	6		T*
W5	PL71B	6		C*
AC1	PL72A	6		T
AD1	PL72B	6		C
VCCIO	VCCIO	6		
Y6	PL73A	6		T*
Y5	PL73B	6		C*
VCC	VCC	6		
AE2	PL74A	6		T
AD2	PL74B	6		C
GND	GNDIO	6		
GND	GND	6		
AB3	PL75A	6	LDQS75	T*

ECP2-50 Logic Signal Connections: 672 fpBGA (Cont.)

Ball Number	Ball Function	Bank	Dual Function	Differential
AB2	PL75B	6		C*
W7	PL76A	6		T
VCCIO	VCCIO	6		
W8	PL76B	6		C
Y7	PL77A	6		T*
Y8	PL77B	6		C*
AC2	PL78A	6		T
GND	GNDIO	6		
AD3	PL78B	6		C
AC3	TCK	9		
AA8	TDI	9		
AB4	TMS	9		
GND	GNDAUX	6		
AA5	TDO	9		
AB5	VCCJ	9		
AE3	PB2A	5	VREF2_5	T
AF3	PB2B	5	VREF1_5	C
VCCAUX	VCCAUX	5		
AC4	PB3A	5		T
AD4	PB3B	5		C
AE4	PB4A	5		T
AF4	PB4B	5		C
VCCIO	VCCIO	5		
V9	PB5A	5		T
W9	PB5B	5		C
GND	GNDIO	5		
AA6	PB6A	5	BDQS6	T
AB6	PB6B	5		C
VCC	VCC	5		
AC5	PB7A	5		T
AD5	PB7B	5		C
AA7	PB8A	5		T
AB7	PB8B	5		C
VCCIO	VCCIO	5		
AE5	PB9A	5		T
AF5	PB9B	5		C
AC7	PB10A	5		T
AD7	PB10B	5		C
GND	GNDIO	5		
GND	GNDAUX	5		
GND	GND	5		
VCCIO	VCCIO	5		
VCC	VCC	5		
GND	GNDIO	5		

ECP2-50 Logic Signal Connections: 672 fpBGA (Cont.)

Ball Number	Ball Function	Bank	Dual Function	Differential
VCCIO	VCCIO	5		
GND	GNDIO	5		
GND	GND	5		
W10	PB20A	5		T
Y10	PB20B	5		C
VCCAUX	VCCAUX	5		
W11	PB21A	5		T
AA10	PB21B	5		C
AC8	PB22A	5		T
AD8	PB22B	5		C
VCCIO	VCCIO	5		
AB8	PB23A	5		T
AB10	PB23B	5		C
GND	GNDIO	5		
AE6	PB24A	5	BDQS24	T
AF6	PB24B	5		C
VCC	VCC	5		
AA11	PB25A	5		T
AC9	PB25B	5		C
AB9	PB26A	5		T
AD9	PB26B	5		C
VCCIO	VCCIO	5		
Y11	PB27A	5		T
AB11	PB27B	5		C
AE7	PB28A	5		T
AF7	PB28B	5		C
GND	GNDIO	5		
GND	GNDAUX	5		
AC10	PB29A	5		T
GND	GND	5		
AD10	PB29B	5		C
AA12	PB30A	5		T
W12	PB30B	5		C
AB12	PB31A	5		T
VCCIO	VCCIO	5		
Y12	PB31B	5		C
AD12	PB32A	5		T
VCC	VCC	5		
AC12	PB32B	5		C
AC13	PB33A	5	BDQS33	T
GND	GNDIO	5		
AA13	PB33B	5		C
AD13	PB34A	5		T
AC14	PB34B	5		C

ECP2-50 Logic Signal Connections: 672 fpBGA (Cont.)

Ball Number	Ball Function	Bank	Dual Function	Differential
AE8	PB35A	5		T
VCCIO	VCCIO	5		
AF8	PB35B	5		C
AB15	PB36A	5		T
Y13	PB36B	5		C
AE9	PB37A	5		T
GND	GNDIO	5		
GND	GND	5		
AF9	PB37B	5		C
W13	PB38A	5		T
AA14	PB38B	5		C
VCCAUX	VCCAUX	5		
AE10	PB39A	5		T
AF10	PB39B	5		C
W14	PB40A	5		T
AB13	PB40B	5		C
VCCIO	VCCIO	5		
Y14	PB41A	5		T
AB14	PB41B	5		C
GND	GNDIO	5		
AE11	PB42A	5	BDQS42	T
AF11	PB42B	5		C
VCC	VCC	5		
AD14	PB43A	5		T
AA15	PB43B	5		C
AE12	PB44A	5	PCLKT5_0	T
AF12	PB44B	5	PCLKC5_0	C
VCCIO	VCCIO	5		
GND	GNDIO	5		
GND	GNDAUX	5		
GND	GND	4		
AD15	PB49A	4	PCLKT4_0	T
VCCIO	VCCIO	4		
AC15	PB49B	4	PCLKC4_0	C
AE13	PB50A	4		T
VCC	VCC	4		
AF13	PB50B	4		C
AB17	PB51A	4	BDQS51	T
GND	GNDIO	4		
Y15	PB51B	4		C
AE14	PB52A	4		T
AF14	PB52B	4		C
AA16	PB53A	4		T
VCCIO	VCCIO	4		

ECP2-50 Logic Signal Connections: 672 fpBGA (Cont.)

Ball Number	Ball Function	Bank	Dual Function	Differential
W15	PB53B	4		C
AC17	PB54A	4		T
AB16	PB54B	4		C
AE15	PB55A	4		T
GND	GNDIO	4		
GND	GND	4		
AF15	PB55B	4		C
AE16	PB56A	4		T
AF16	PB56B	4		C
VCCAUX	VCCAUX	4		
Y16	PB57A	4		T
AB18	PB57B	4		C
AD17	PB58A	4		T
AD18	PB58B	4		C
VCCIO	VCCIO	4		
AC18	PB59A	4		T
AD19	PB59B	4		C
GND	GNDIO	4		
AC19	PB60A	4	BDQS60	T
AE17	PB60B	4		C
VCC	VCC	4		
AB19	PB61A	4		T
AE19	PB61B	4		C
AF17	PB62A	4		T
AE18	PB62B	4		C
VCCIO	VCCIO	4		
W16	PB63A	4		T
AA17	PB63B	4		C
AF18	PB64A	4		T
AF19	PB64B	4		C
GND	GNDIO	4		
GND	GNDAUX	4		
AA19	PB65A	4		T
GND	GND	4		
W17	PB65B	4		C
Y19	PB66A	4		T
Y17	PB66B	4		C
AF20	PB67A	4		T
VCCIO	VCCIO	4		
AE20	PB67B	4		C
AA20	PB68A	4		T
VCC	VCC	4		
W18	PB68B	4		C
AD20	PB69A	4	BDQS69	T

ECP2-50 Logic Signal Connections: 672 fpBGA (Cont.)

Ball Number	Ball Function	Bank	Dual Function	Differential
GND	GNDIO	4		
AE21	PB69B	4		C
AF21	PB70A	4		T
AF22	PB70B	4		C
VCCIO	VCCIO	4		
GND	GNDIO	4		
GND	GND	4		
AE22	PB74A	4		T
AD22	PB74B	4		C
VCCAUX	VCCAUX	4		
AF23	PB75A	4		T
AE23	PB75B	4		C
AD23	PB76A	4		T
AC23	PB76B	4		C
VCCIO	VCCIO	4		
AB20	PB77A	4		T
AC20	PB77B	4		C
GND	GNDIO	4		
AB21	PB78A	4	BDQS78	T
AC22	PB78B	4		C
VCC	VCC	4		
W19	PB79A	4		T
AA21	PB79B	4		C
AF24	PB80A	4		T
AE24	PB80B	4		C
VCCIO	VCCIO	4		
Y20	PB81A	4		T
AB22	PB81B	4		C
Y21	PB82A	4	VREF2_4	T
AB23	PB82B	4	VREF1_4	C
GND	GNDIO	4		
GND	GNDAUX	4		
AD24	CFG2	8		
W20	CFG1	8		
AC24	CFG0	8		
GND	GNDAUX	8		
V19	PROGRAMN	8		
AA22	CCLK	8		
AB24	INITN	8		
AD25	DONE	8		
GND	GNDIO	8		
W21	PR77B	8	WRITEN	C
Y22	PR77A	8	CS1N	T
AC25	PR76B	8	CSN	C

ECP2-50 Logic Signal Connections: 672 fpBGA (Cont.)

Ball Number	Ball Function	Bank	Dual Function	Differential
AB25	PR76A	8	D0	T
VCCIO	VCCIO	8		
AD26	PR75B	8	D1	C
AC26	PR75A	8	D2	T
GND	GND	8		
Y23	PR74B	8	D3	C
GND	GNDIO	8		
W22	PR74A	8	D4	T
AA25	PR73B	8	D5	C
VCC	VCC	8		
AB26	PR73A	8	D6	T
W23	PR72B	8	D7	C
VCCIO	VCCIO	8		
V22	PR72A	8	DI	T
Y24	PR71B	8	DOUT,CSON	C
Y25	PR71A	8	BUSY	T
W24	PR70B	3		C
GND	GNDIO	3		
V23	PR70A	3		T
AA26	PR69B	3		C*
Y26	PR69A	3		T*
U21	PR68B	3		C
VCCIO	VCCIO	3		
U19	PR68A	3		T
W25	PR67B	3		C*
GND	GND	3		
W26	PR67A	3	RDQS67	T*
GND	GNDIO	3		
V24	PR66B	3		C
V25	PR66A	3		T
VCC	VCC	3		
V26	PR65B	3		C*
U26	PR65A	3		T*
VCCIO	VCCIO	3		
U22	PR64B	3	RLM0_GPLLC_FB_A	C
U23	PR64A	3	RLM0_GPLLT_FB_A	T
U24	PR63B	3	RLM0_GPLLC_IN_A	C*
U25	PR63A	3	RLM0_GPLLT_IN_A	T*
VCCAUX	VCCAUX	3		
R20	RLM0_PLLCAP	3		
P18	RLM0_VCCPLL	3		
GND	GNDAUX	3		
T19	PR61B	3	RLM0_GDLLC_FB_A	C
U20	PR61A	3	RLM0_GDLLT_FB_A	T

ECP2-50 Logic Signal Connections: 672 fpBGA (Cont.)

Ball Number	Ball Function	Bank	Dual Function	Differential
GND	GNDIO	3		
T25	PR60B	3	RLM0_GDLLC_IN_A	C*
T26	PR60A	3	RLM0_GDLLT_IN_A	T*
T20	PR59B	3		C
T22	PR59A	3		T
VCCIO	VCCIO	3		
R26	PR58B	3		C*
R25	PR58A	3	RDQS58	T*
GND	GND	3		
R22	PR57B	3		C
GND	GNDIO	3		
T21	PR57A	3		T
P26	PR56B	3		C*
VCC	VCC	3		
P25	PR56A	3		T*
R24	PR55B	3		C
VCCIO	VCCIO	3		
R23	PR55A	3		T
P20	PR54B	3		C*
R19	PR54A	3		T*
P21	PR53B	3		C
GND	GNDIO	3		
P19	PR53A	3		T
P23	PR52B	3		C*
P22	PR52A	3		T*
N22	PR51B	3		C
VCCIO	VCCIO	3		
R21	PR51A	3		T
N26	PR50B	3		C*
GND	GND	3		
N25	PR50A	3	RDQS50	T*
GND	GNDIO	3		
N19	PR49B	3		C
N20	PR49A	3		T
VCC	VCC	3		
M26	PR48B	3		C*
M25	PR48A	3		T*
VCCIO	VCCIO	3		
N18	PR47B	3	VREF2_3	C
N21	PR47A	3	VREF1_3	T
L26	PR46B	3	PCLKC3_0	C*
L25	PR46A	3	PCLKT3_0	T*
VCCAUX	VCCAUX	3		
GND	GNDAUX	2		

ECP2-50 Logic Signal Connections: 672 fpBGA (Cont.)

Ball Number	Ball Function	Bank	Dual Function	Differential
N24	PR44B	2	PCLKC2_0	C
M23	PR44A	2	PCLKT2_0	T
GND	GNDIO	2		
L21	PR43B	2		C*
K22	PR43A	2		T*
M24	PR42B	2		C
N23	PR42A	2		T
VCCIO	VCCIO	2		
K26	PR41B	2		C*
K25	PR41A	2	RDQS41	T*
GND	GND	2		
M20	PR40B	2		C
GND	GNDIO	2		
M19	PR40A	2		T
L22	PR39B	2		C*
VCC	VCC	2		
M22	PR39A	2		T*
K21	PR38B	2		C
VCCIO	VCCIO	2		
M21	PR38A	2		T
K24	PR37B	2		C*
J24	PR37A	2		T*
GND	GNDIO	2		
VCCIO	VCCIO	2		
GND	GND	2		
GND	GNDIO	2		
VCC	VCC	2		
VCCIO	VCCIO	2		
VCCAUX	VCCAUX	2		
GND	GNDAUX	2		
L20	RUM0_VCCPLL	2		
GND	GNDIO	2		
J26	PR26B	2	RUM0_SPLLC_FB_A	C
J25	PR26A	2	RUM0_SPLLT_FB_A	T
J23	PR25B	2	RUM0_SPLLC_IN_A	C
K23	PR25A	2	RUM0_SPLLT_IN_A	T
VCCIO	VCCIO	2		
H26	PR24B	2		C*
H25	PR24A	2	RDQS24	T*
GND	GND	2		
H24	PR23B	2		C
GND	GNDIO	2		
H23	PR23A	2		T
VCC	VCC	2		

ECP2-50 Logic Signal Connections: 672 fpBGA (Cont.)

Ball Number	Ball Function	Bank	Dual Function	Differential
VCCIO	VCCIO	2		
G26	PR19B	2		C
GND	GNDIO	2		
G25	PR19A	2		T
F26	PR18B	2		C*
F25	PR18A	2		T*
K20	PR17B	2		C
VCCIO	VCCIO	2		
L19	PR17A	2		T
E26	PR16B	2		C*
GND	GND	2		
E25	PR16A	2	RDQS16	T*
GND	GNDIO	2		
J22	PR15B	2		C
H22	PR15A	2		T
VCC	VCC	2		
G24	PR14B	2		C*
G23	PR14A	2		T*
VCCIO	VCCIO	2		
K19	PR13B	2		C
J19	PR13A	2		T
D26	PR12B	2		C*
C26	PR12A	2		T*
VCCAUX	VCCAUX	2		
F22	PR11B	2		C
E24	PR11A	2		T
GND	GNDIO	2		
D25	PR10B	2		C*
C25	PR10A	2		T*
D24	PR9B	2		C
B25	PR9A	2		T
VCCIO	VCCIO	2		
H21	PR8B	2		C*
G22	PR8A	2	RDQS8	T*
GND	GND	2		
B24	PR7B	2		C
GND	GNDIO	2		
C24	PR7A	2		T
D23	PR6B	2		C*
VCC	VCC	2		
C23	PR6A	2		T*
G21	PR5B	2		C
VCCIO	VCCIO	2		
H20	PR5A	2		T

ECP2-50 Logic Signal Connections: 672 fpBGA (Cont.)

Ball Number	Ball Function	Bank	Dual Function	Differential
GND	GNDIO	2		
E22	PR2B	2	VREF2_2	C*
F21	PR2A	2	VREF1_2	T*
GND	GNDAUX	1		
E23	PT82B	1	VREF2_1	C
GND	GNDIO	1		
D22	PT82A	1	VREF1_1	T
G20	PT81B	1		C
J18	PT81A	1		T
F20	PT80B	1		C
VCCIO	VCCIO	1		
H19	PT80A	1		T
A24	PT79B	1		C
A23	PT79A	1		T
E21	PT78B	1		C
VCC	VCC	1		
F19	PT78A	1		T
C22	PT77B	1		C
GND	GNDIO	1		
E20	PT77A	1		T
B22	PT76B	1		C
VCCIO	VCCIO	1		
B23	PT76A	1		T
C20	PT75B	1		C
D20	PT75A	1		T
A22	PT74B	1		C
VCCAUX	VCCAUX	1		
A21	PT74A	1		T
GND	GND	1		
GND	GNDIO	1		
E19	PT71B	1		C
C19	PT71A	1		T
VCCIO	VCCIO	1		
B21	PT70B	1		C
B20	PT70A	1		T
D19	PT69B	1		C
B19	PT69A	1		T
GND	GNDIO	1		
G17	PT68B	1		C
E18	PT68A	1		T
VCC	VCC	1		
G19	PT67B	1		C
F17	PT67A	1		T
VCCIO	VCCIO	1		

ECP2-50 Logic Signal Connections: 672 fpBGA (Cont.)

Ball Number	Ball Function	Bank	Dual Function	Differential
A20	PT66B	1		C
A19	PT66A	1		T
E17	PT65B	1		C
D18	PT65A	1		T
GND	GND	1		
GND	GNDAUX	1		
B18	PT64B	1		C
GND	GNDIO	1		
A18	PT64A	1		T
E16	PT63B	1		C
G16	PT63A	1		T
F16	PT62B	1		C
VCCIO	VCCIO	1		
H18	PT62A	1		T
A17	PT61B	1		C
B17	PT61A	1		T
C18	PT60B	1		C
VCC	VCC	1		
B16	PT60A	1		T
C17	PT59B	1		C
GND	GNDIO	1		
D17	PT59A	1		T
E15	PT58B	1		C
VCCIO	VCCIO	1		
G15	PT58A	1		T
A16	PT57B	1		C
B15	PT57A	1		T
D15	PT56B	1		C
VCCAUX	VCCAUX	1		
F15	PT56A	1		T
A14	PT55B	1		C
GND	GND	1		
B14	PT55A	1		T
GND	GNDIO	1		
C15	PT54B	1		C
A15	PT54A	1		T
A13	PT53B	1		C
B13	PT53A	1		T
VCCIO	VCCIO	1		
H17	PT52B	1		C
H15	PT52A	1		T
D13	PT51B	1		C
C14	PT51A	1		T
GND	GNDIO	1		

ECP2-50 Logic Signal Connections: 672 fpBGA (Cont.)

Ball Number	Ball Function	Bank	Dual Function	Differential
G14	PT50B	1		C
E14	PT50A	1		T
VCC	VCC	1		
A12	PT49B	1		C
B12	PT49A	1		T
VCCIO	VCCIO	1		
F14	PT48B	1	PCLKC1_0	C
D14	PT48A	1	PCLKT1_0	T
GND	GND	1		
H16	XRES	1		
GND	GNDAUX	0		
H14	PT46B	0	PCLKC0_0	C
GND	GNDIO	0		
H13	PT46A	0	PCLKT0_0	T
A11	PT45B	0		C
B11	PT45A	0		T
C13	PT44B	0		C
VCCIO	VCCIO	0		
E13	PT44A	0		T
D12	PT43B	0		C
F13	PT43A	0		T
A10	PT42B	0		C
VCC	VCC	0		
B10	PT42A	0		T
C12	PT41B	0		C
GND	GNDIO	0		
C10	PT41A	0		T
G13	PT40B	0		C
VCCIO	VCCIO	0		
H12	PT40A	0		T
A9	PT39B	0		C
B9	PT39A	0		T
E12	PT38B	0		C
VCCAUX	VCCAUX	0		
G12	PT38A	0		T
A8	PT37B	0		C
GND	GND	0		
B8	PT37A	0		T
GND	GNDIO	0		
E11	PT36B	0		C
C9	PT36A	0		T
A7	PT35B	0		C
B7	PT35A	0		T
VCCIO	VCCIO	0		

ECP2-50 Logic Signal Connections: 672 fpBGA (Cont.)

Ball Number	Ball Function	Bank	Dual Function	Differential
F12	PT34B	0		C
D10	PT34A	0		T
H11	PT33B	0		C
G11	PT33A	0		T
GND	GNDIO	0		
A6	PT32B	0		C
B6	PT32A	0		T
VCC	VCC	0		
D8	PT31B	0		C
C8	PT31A	0		T
VCCIO	VCCIO	0		
F11	PT30B	0		C
E10	PT30A	0		T
E9	PT29B	0		C
D9	PT29A	0		T
GND	GND	0		
GND	GNDAUX	0		
G10	PT28B	0		C
GND	GNDIO	0		
H10	PT28A	0		T
A5	PT27B	0		C
B5	PT27A	0		T
C7	PT26B	0		C
VCCIO	VCCIO	0		
D7	PT26A	0		T
E8	PT25B	0		C
F10	PT25A	0		T
F8	PT24B	0		C
VCC	VCC	0		
H9	PT24A	0		T
C5	PT23B	0		C
GND	GNDIO	0		
D5	PT23A	0		T
B4	PT22B	0		C
VCCIO	VCCIO	0		
VCCAUX	VCCAUX	0		
GND	GND	0		
GND	GNDIO	0		
VCCIO	VCCIO	0		
GND	GNDIO	0		
VCC	VCC	0		
VCCIO	VCCIO	0		
GND	GND	0		
GND	GNDAUX	0		

ECP2-50 Logic Signal Connections: 672 fpBGA (Cont.)

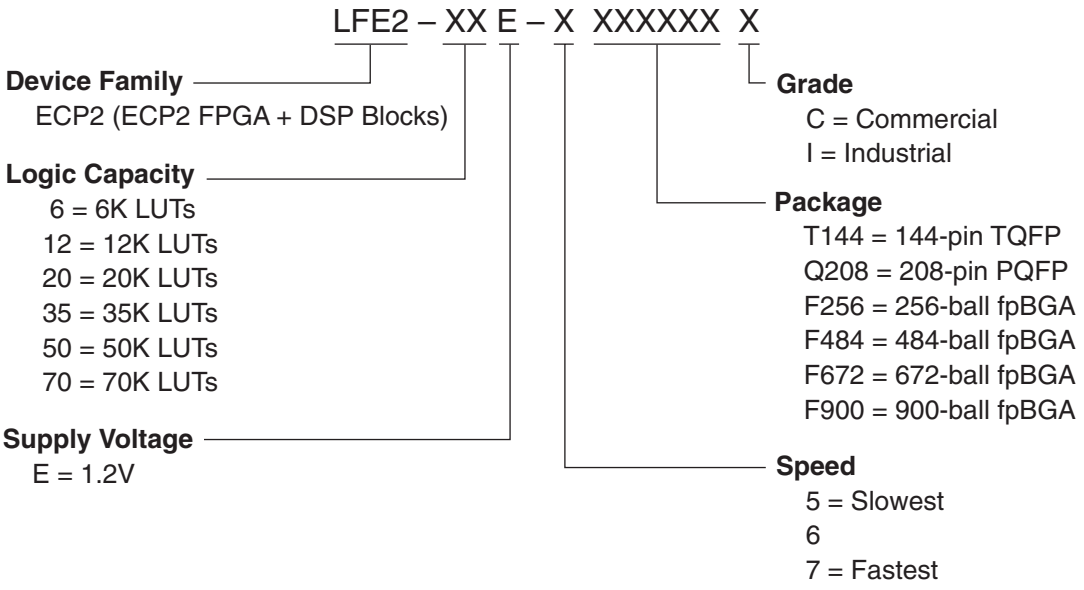
Ball Number	Ball Function	Bank	Dual Function	Differential
C4	PT10B	0		C
GND	GNDIO	0		
C3	PT10A	0		T
A4	PT9B	0		C
A3	PT9A	0		T
B3	PT8B	0		C
VCCIO	VCCIO	0		
B2	PT8A	0		T
D4	PT7B	0		C
D3	PT7A	0		T
C2	PT6B	0		C
VCC	VCC	0		
C1	PT6A	0		T
G8	PT5B	0		C
GND	GNDIO	0		
G7	PT5A	0		T
E7	PT4B	0		C
VCCIO	VCCIO	0		
F7	PT4A	0		T
E6	PT3B	0		C
E5	PT3A	0		T
G6	PT2B	0	VREF2_0	C
VCCAUX	VCCAUX	0		
G5	PT2A	0	VREF1_0	T

*Supports dedicated LVDS outputs.

LatticeECP2 Family Data Sheet Ordering Information

February 2006 **Advance Data Sheet**

Part Number Description



Ordering Information

Note: LatticeECP2 devices are dual marked. For example, the commercial speed grade LFE2-50E-7F672C is also marked with industrial grade -6I (LFE2-50E-6F672I). 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:

Commercial

Part Number	I/Os	Voltage	Grade	Package	Pins	Temp.	LUTs (K)
LFE2-50E-5F484C	339	1.2V	-5	fpBGA	484	COM	50
LFE2-50E-6F484C	339	1.2V	-6	fpBGA	484	COM	50
LFE2-50E-7F484C	339	1.2V	-7	fpBGA	484	COM	50
LFE2-50E-5F672C	500	1.2V	-5	fpBGA	672	COM	50
LFE2-50E-6F672C	500	1.2V	-6	fpBGA	672	COM	50
LFE2-50E-7F672C	500	1.2V	-7	fpBGA	672	COM	50

Industrial

Part Number	I/Os	Voltage	Grade	Package	Pins	Temp.	LUTs (K)
LFE2-50E-5F484I	339	1.2V	-5	fpBGA	484	IND	50
LFE2-50E-6F484I	339	1.2V	-6	fpBGA	484	IND	50
LFE2-50E-5F672I	500	1.2V	-5	fpBGA	672	IND	50
LFE2-50E-6F672I	500	1.2V	-6	fpBGA	672	IND	50

LatticeECP2 Family Data Sheet Supplemental Information

February 2006 **Advance Data Sheet**

For Further Information

A variety of technical notes for the LatticeECP2 family are available on the Lattice web site at www.latticesemi.com.

- LatticeECP2 sysIO Usage Guide (TN1102)
- LatticeECP2 sysCLOCK PLL Design and Usage Guide (TN1103)
- LatticeECP2 Memory Usage Guide (TN1104)
- LatticeECP2 High-Speed I/O Interface (TN1105)
- Power Estimation and Management for LatticeECP2 Devices (TN1106)
- LatticeECP2 sysDSP Usage Guide (TN1107)
- LatticeECP2 sysCONFIG Usage Guide (TN1110)

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

Section II. LatticeECP2 Family Technical Notes

LatticeECP2 sysIO Usage Guide

February 2006 Technical Note TN1102

Introduction

The LatticeECP2™ 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

LatticeECP2 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 LatticeECP2 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 LatticeECP2 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 device 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 details in Lattice technical note number TN1105, *LatticeECP2 DDR Usage Guide.*

Supported sysIO Standards

The LatticeECP2 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 LVDS, RSDS, BLVDS, LVPECL, differential SSTL and differential HSTL. Tables 1 and 2 list the sysIO standards supported in LatticeECP2 devices.

Table 7-1. 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	—

Table 7-1. Supported Input Standards (Continued)

Input Standard	V_{REF} (Nom.)	V_{CCIO}[1] (Nom.)
SSTL3 Class I, II	1.5	—
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 7-2. 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/PCIX	N/A	3.3
HSTL18 Class I, II	N/A	1.8
HSTL15 Class I	N/A	1.5
SSTL3 Class I, II	N/A	3.3
SSTL2 Class I, II	N/A	2.5
SSTL18 Class I, II	N/A	1.8
Differential Interfaces		
Differential SSTL3, Class I, II	N/A	3.3
Differential SSTL2, 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	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

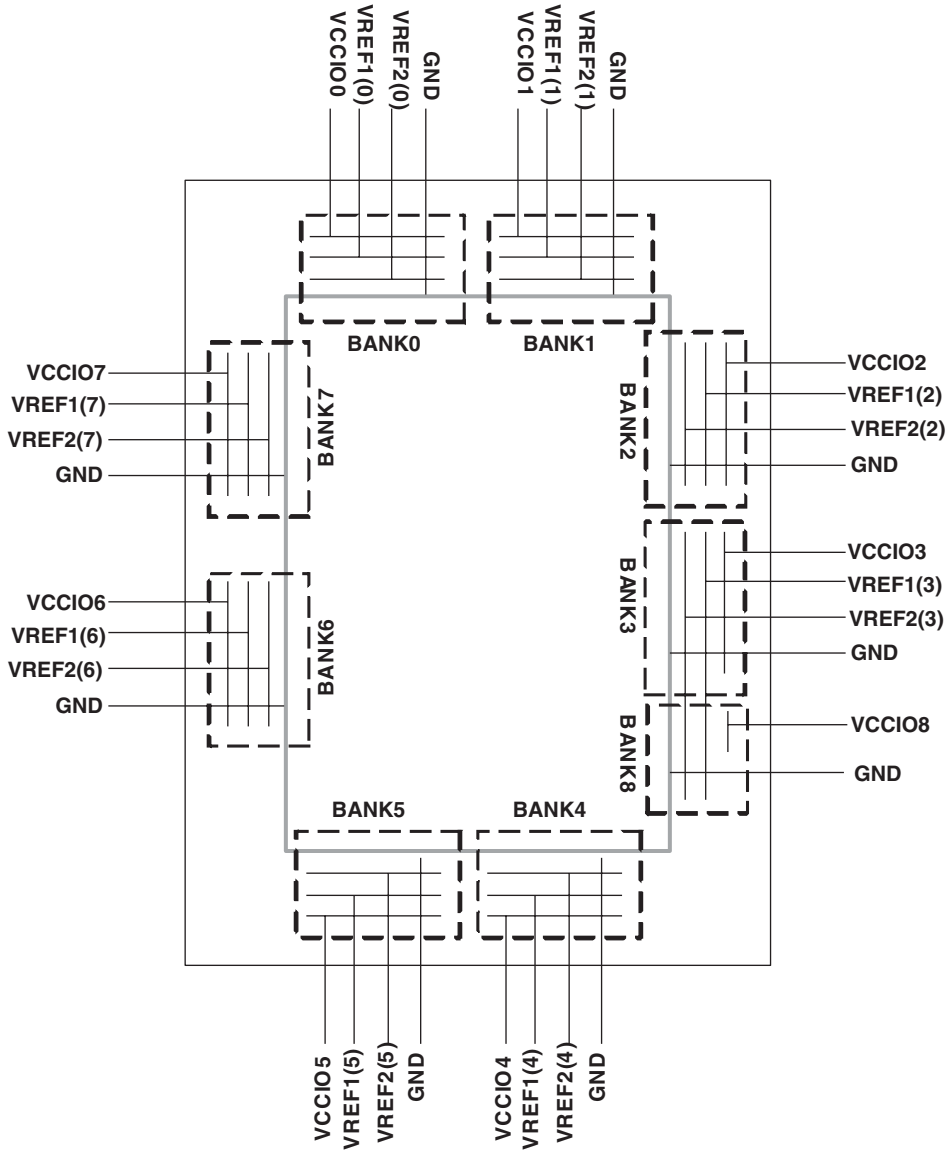
1. Emulated with external resistors. For more detail, please see information regarding additional technical documentation at the end of this data sheet.

sysIO Banking Scheme

LatticeECP2 devices have eight general purpose programmable sysIO banks and a ninth configuration bank. Each of the eight general purpose sysIO banks has a V_{CCIO} supply voltage, and two reference voltages, V_{REF1} and V_{REF2}. Figure 7-1 shows the eight general purpose banks and the configuration bank with associated supplies. Bank 8 is a bank dedicated to configuration logic and has seven dedicated configuration I/Os and 14 multiplexed configuration I/Os. Bank 8 does have the power supply pads (V_{CCIO} and V_{CCAUX}) but does not have any independent V_{REF} pads. The I/Os in Bank 8 are connected to V_{REF} from Bank 3.

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. 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.

Figure 7-1. 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. In addition, V_{CCIO8} is used to supply power to the sysCONFIG™ signals.

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 7-3 shows a summary of all the required power supplies.

Table 7-3. Power Supplies

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

1. Refer to device data sheet for recommended min. and max. values.

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 TN1105, *LatticeECP2 DDR Usage Guide*. 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 LatticeECP2 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 7-4 shows the sysIO standards that can be mixed in the same bank.

Table 7-4. 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 7-5. 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 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 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 PCI33 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 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 without 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 7-6 shows the available drive settings for each out the output standards.

Table 7-6. Programmable Drive Values for Single-ended Buffers

Single Ended I/O Standards	Programmable Drive (mA)
HSTL15_I/ HSTL15D_I	4, 8
HSTL18_I/ HSTL18D_I	8, 12
SSTL25_I/ SSTL25D_I	8, 12
SSTL25_II/ SSTL25D_II	16, 20
SSTL18_II/SSTL18D_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.

PCI Support with Programmable PCICLAMP

Each sysIO buffer can be configured to support PCI33. The buffers on the bottom 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 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 Preference Editor GUI or in the ASCII preference file (.prf) 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 7-7 lists the available I/O types.

Table 7-7. IO_TYPE Attribute Values

sysIO Signaling Standard	IO_TYPE
DEFAULT	LVCMOS25
LVDS 2.5V	LVDS25
RSDS	RSDS
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	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

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 7-8. 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 LVTLL and LVCMOS inputs and outputs. This attribute can be enabled for each I/O independently.

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

Table 7-9. PULLMODE Values

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

PCICLAMP

PCI33 inputs on the 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
Default: OFF

Table 7-10. PCICLAMP Values

Input Type	PCICLAMP Value
PCI33	ON
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: OFF

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 Preference Editor of the ispLEVER software. 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 GUI in the Preference Editor of 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 LatticeECP2 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 bottom banks (Banks 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 the top and bottom will support the LVDS input buffers but will not support True LVDS outputs. 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.

Assigning V_{REF1}/ V_{REF2} Groups for Referenced Inputs

Each bank has two dedicated V_{REF} input pins, V_{REF1} and V_{REF2}. Designers can group buffers to a particular V_{REF} rail, V_{REF1} or V_{REF2}. This grouping is done by assigning a PGROUP VREF preference along with the LOCATE PGROUP preference.

Preference Syntax

```
PGROUP <pgrp_name> [(VREF <vref_name>)+] (COMP <comp_name>)+;
LOCATE PGROUP <pgrp_name> BANK <bank_num>;
LOCATE VREF <vref_name> SITE <site_name>;
```

Example Showing VREF Groups

```
PGROUP “vref_pg1” VREF “ref1” COMP “ah(0)” COMP “ah(1)” COMP “ah(2)” COMP “ah(3)”
COMP “ah(4)” COMP “ah(5)” COMP “ah(6)” COMP “ah(7)”;

PGROUP “vref_pg2” VREF “ref2” COMP “al(0)” COMP “al(1)” COMP “al(2)” COMP “al(3)”
COMP “al(4)” COMP “al(5)” COMP “al(6)” COMP “al(7)”;

LOCATE VREF “ref1” SITE PR29C;
LOCATE VREF “ref2” SITE PR48B;
```

or

```
LOCATE PGROUP “ vref_pg1” BANK 2;
LOCATE PGROUP “ vref_pg2” BANK 2;
```

The example shows two V_{REF} groups, “vref_pg1” assigned to VREF “ref1” and “vref_pg2” assigned to “ref2”. The user must lock these V_{REF} to either V_{REF1} or V_{REF2} using the LOCATE preference. Alternatively, users can designate to which bank the V_{REF} group should be located. The software will then assign these to either the V_{REF1} or V_{REF2} of the bank.

If the PGROUP VREF is not used, the software will automatically group all pins that need the same V_{REF} reference voltage. This preference is most useful when there is more than one bus that uses the same reference voltage and the user wishes to associate each of these busses to different V_{REF} resources.

Differential I/O Implementation

The LatticeECP2 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 LatticeECP2 Device 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 LatticeECP2 device 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 LatticeECP2 device 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 LatticeECP2 device 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 LatticeECP2 device data sheet for a detailed explanation of Differential HSTL and SSTL 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

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. You can refer to the Precision RTL Synthesis and Synplicity user manuals for a complete list of synthesis attributes. These manuals 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 7-11. 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 7-12. 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 7-13. 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
```

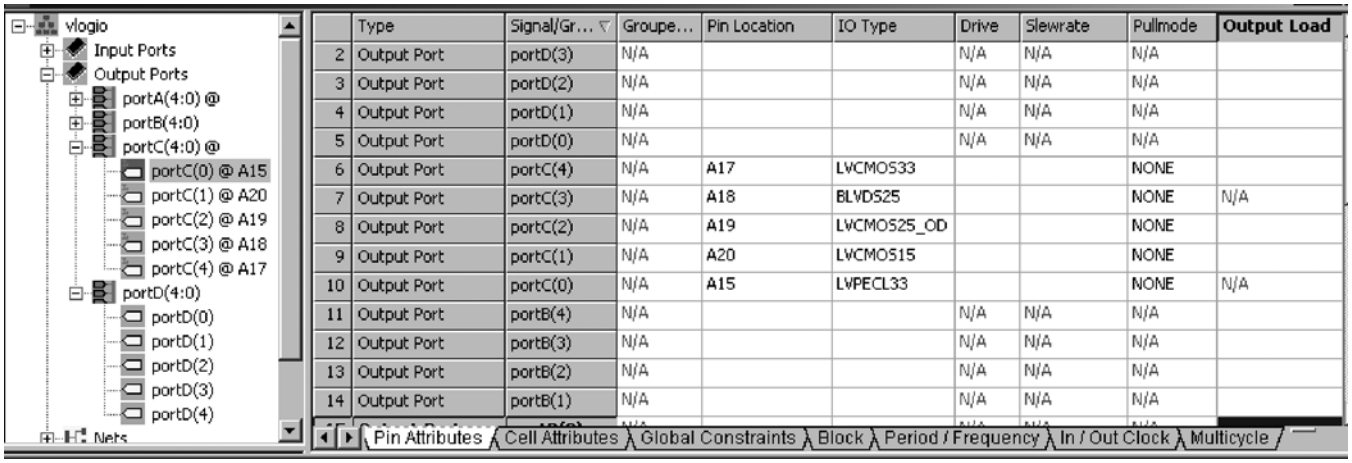
Appendix B. sysIO Attributes Using the Preference Editor User Interface

Designers can assign sysIO buffer attributes using the Pre-Map Preference Editor 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 Preference Editor 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).

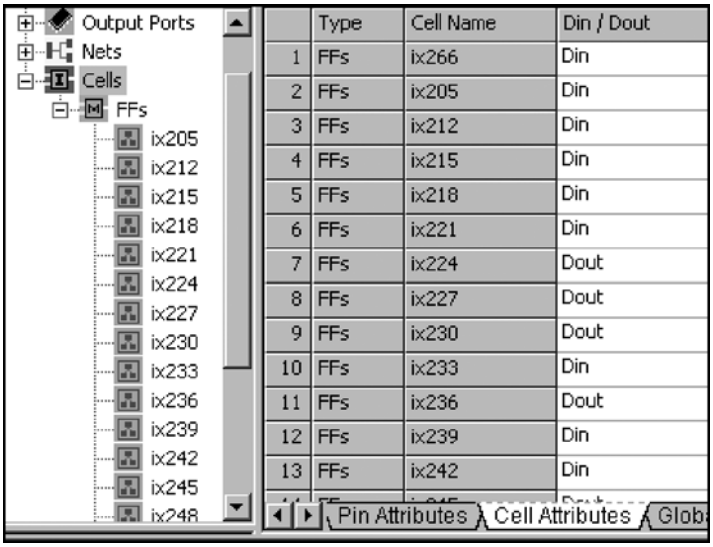
Figures 2 and 3 show the Pin Attribute sheet and the Cell Attribute sheet views of the preference editor. For further information on how to use the Preference Editor, refer to the ispLEVER Help documentation in the Help menu option of the software.

Figure 7-2. Pin Attributes Tab



	Type	Signal/Gr...	Groupe...	Pin Location	IO Type	Drive	Slewrate	Pullmode	Output Load
2	Output Port	portD(3)	N/A			N/A	N/A	N/A	
3	Output Port	portD(2)	N/A			N/A	N/A	N/A	
4	Output Port	portD(1)	N/A			N/A	N/A	N/A	
5	Output Port	portD(0)	N/A			N/A	N/A	N/A	
6	Output Port	portC(4)	N/A	A17	LVCMOS33			NONE	
7	Output Port	portC(3)	N/A	A18	BLVDS25			NONE	N/A
8	Output Port	portC(2)	N/A	A19	LVCMOS25_OD			NONE	
9	Output Port	portC(1)	N/A	A20	LVCMOS15			NONE	
10	Output Port	portC(0)	N/A	A15	LVPECL33			NONE	N/A
11	Output Port	portB(4)	N/A			N/A	N/A	N/A	
12	Output Port	portB(3)	N/A			N/A	N/A	N/A	
13	Output Port	portB(2)	N/A			N/A	N/A	N/A	
14	Output Port	portB(1)	N/A			N/A	N/A	N/A	

Figure 7-3. Cell Attributes Tab



	Type	Cell Name	Din / Dout
1	FFs	ix266	Din
2	FFs	ix205	Din
3	FFs	ix212	Din
4	FFs	ix215	Din
5	FFs	ix218	Din
6	FFs	ix221	Din
7	FFs	ix224	Dout
8	FFs	ix227	Dout
9	FFs	ix230	Dout
10	FFs	ix233	Din
11	FFs	ix236	Dout
12	FFs	ix239	Din
13	FFs	ix242	Din

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

Designers can enter sysIO attributes directly in the preference (.prf) file as sysIO buffer preferences. The PRF file is an ASCII file containing two separate sections: a schematic section for those preferences created by the mapper or translator, and a user section for preferences entered by the user. User preferences can be written directly into this file. The synthesis attributes appear between the schematic start and schematic end of the file. The sysIO buffer preferences can be entered after the schematic end line using the preference file syntax. Below are a list of sysIO buffer preference syntax and examples.

IOBUF

This preference is used to assign the attribute IO_TYPE, PULLMODE, SLEWRATE 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 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. This can also be applied to a specified PGROUP.

Syntax

```
LOCATE [COMP <comp_name> | MACRO <macro_name>] SITE <site_name>;
LOCATE PGROUP <pgroup_name> [SITE <site_name>; | REGION <region_name>;]
LOCATE PGROUP <pgroup_name> RANGE <site_1> [<site_2> | <count>] [<direction>] |
RANGE <chip_side> [<direction>];
LOCATE BUS < bus_name> ROW|COL <number>;
<bus_name> := string
<number>  := integer
```

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”;
```

This command places bus1 on ROW 3 and bus2 on COL4

```
LOCATE BUS “bus1” ROW 3;
LOCATE BUS “bus2” COL 4;
```

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”;
```

PGROUP VREF

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

Syntax

```
PGROUP <pgrp_name> [(VREF <vref_name>)+] (COMP <comp_name>)+;
LOCATE PGROUP <pgrp_name> BANK <bank_num>;
LOCATE VREF <vref_name> SITE <site_name>;
```

Example

```
PGROUP “vref_pg1” VREF “ref1” COMP “ah(0)” COMP “ah(1)” COMP “ah(2)” COMP “ah(3)”
COMP “ah(4)” COMP “ah(5)” COMP “ah(6)” COMP “ah(7)”;
PGROUP “vref_pg2” VREF “ref2” COMP “al(0)” COMP “al(1)” COMP “al(2)” COMP “al(3)”
COMP “al(4)” COMP “al(5)” COMP “al(6)” COMP “al(7)”;
LOCATE VREF “ref1” SITE PR29C;
LOCATE VREF “ref2” SITE PR48B;
```

or:

```
LOCATE PGROUP " vref_pg1" BANK 2;
LOCATE PGROUP " vref_pg2" BANK 2;
```

LatticeECP2 sysCLOCK PLL/DLL Design and Usage Guide

February 2006 Technical Note TN1103

Introduction

This user's guide describes the clock resources available in the LatticeECP2™ device architecture. Details are provided for primary clocks, secondary clocks and edge clocks, as well as clock elements such as PLLs, DLLs, Clock Dividers, and several other elements.

The number of PLLs and DLLs for each device package is described in Table 8-1.

Table 8-1. LatticeECP2 Family and Number of PLLs

Device	ECP2-3	ECP2-6	ECP2-12	ECP2-21	ECP2-35	ECP2-50	ECP2-70	ECP2-100
Number of SPLLs	0	0	0	0	0	2	4	4
Number of GPLLs	2	2	2	2	2	2	2	2
Number of DLLs	2	2	2	2	2	2	2	2
Number of DQSDLLs	2	2	2	2	2	2	2	2

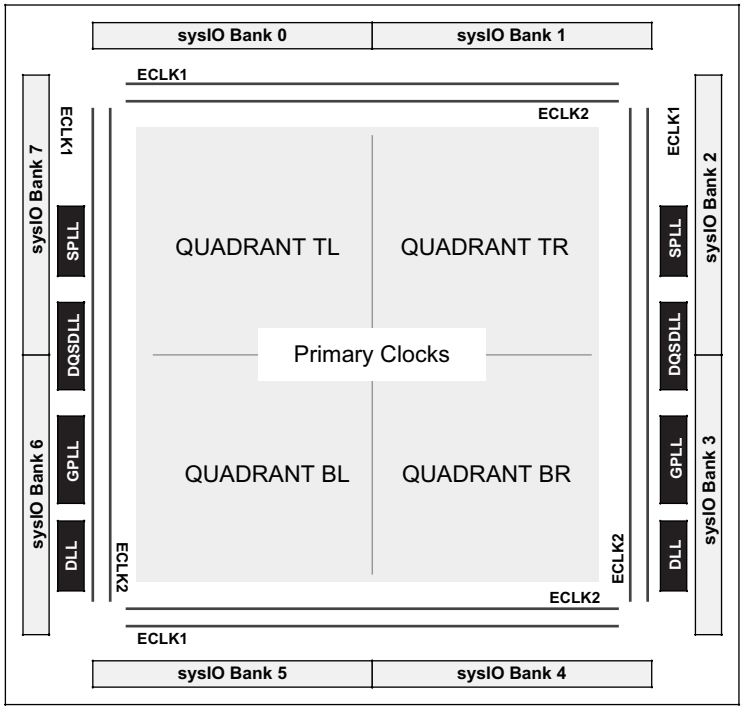
Clock/Control Distribution Network

The LatticeECP2 family provides global clocks, eight quadrant-based primary clocks and flexible secondary clocks. The devices also provide two edge clocks on every edge of the device. Other clock sources include PLLs, DLLs, Slave Delay Lines and Clock Dividers.

LatticeECP2 Top Level View

Figure 8-1 provides a view of the primary clocking structure of the LatticeECP2-50 architecture.

Figure 8-1. LatticeECP2-50 Clocking Structure



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 the primary clocks are:

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

Secondary Clocks

The LatticeECP2 secondary clocks are a flexible region-based clocking resource. Each region can have four independent clock inputs. As a regional resource, it can cross the primary clock quadrant boundaries.

The sources of secondary clocks are:

- Dedicated Clock pins
- Internal nodes

Edge Clocks

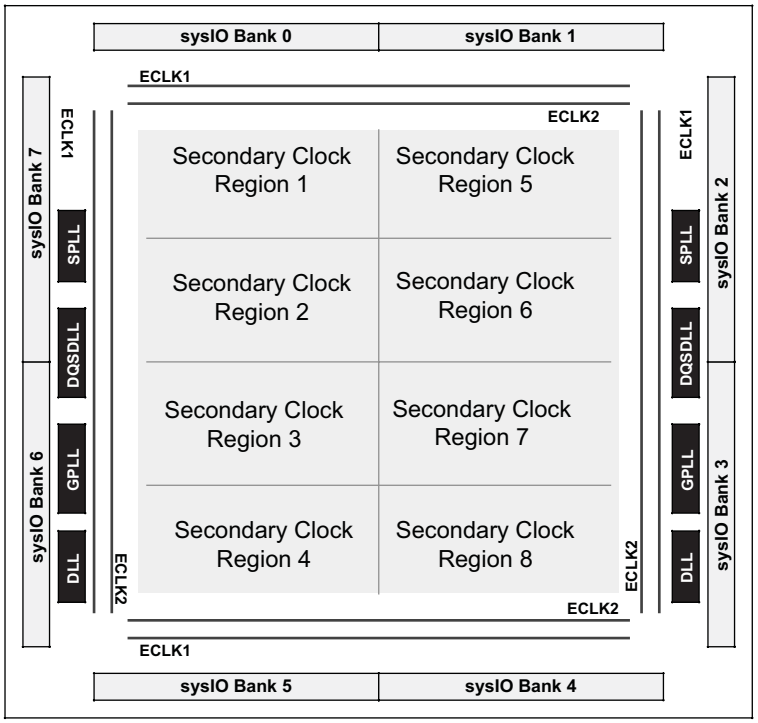
The LatticeECP2 has two edge clocks on every edge. These clocks have low injection time and skew and are used to clock I/O registers. The edge clock (ECLK) resources are designed for high speed I/O interfaces with high fanout capability.

The sources of edge clocks are:

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

Figure 8-2 describes the secondary clocks and edge clocks structure.

Figure 8-2. LatticeECP2-50 Secondary Clocks and Edge Clocks



Clocking Preferences

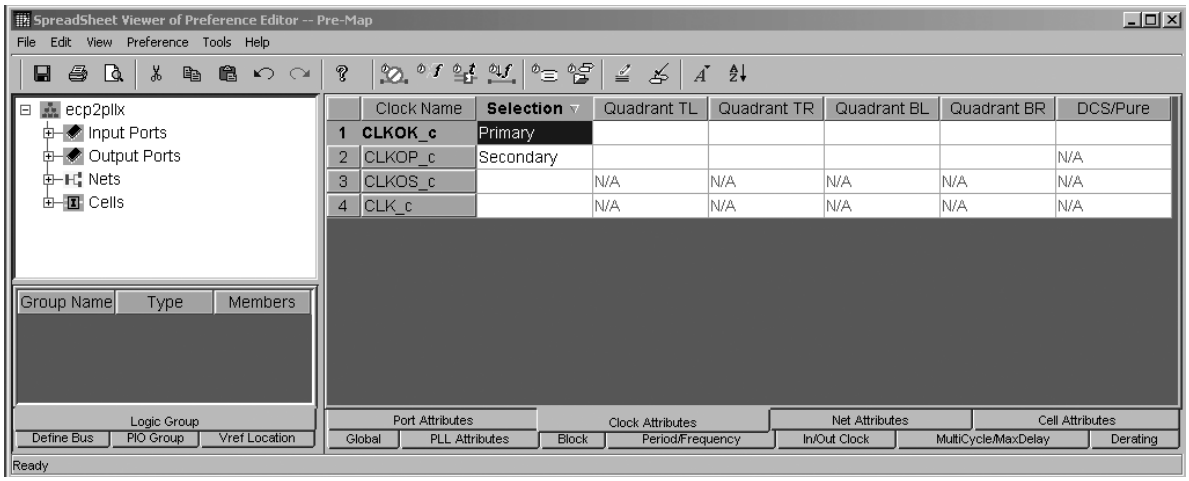
As illustrated in Figure 8-3, users can set each clock to the desired clock net.

The selection attributes are, Primary, Secondary and Edge Clock.

Under DCS/Pure, users can specify the primary clocks to “Primary/Pure’ or “Primary/DCS’.

Primary/Pure routes the primary clocks directly to each quadrant without going through DCS. Primary/DCS sets the primary clock with the Dynamic Clock Select feature.

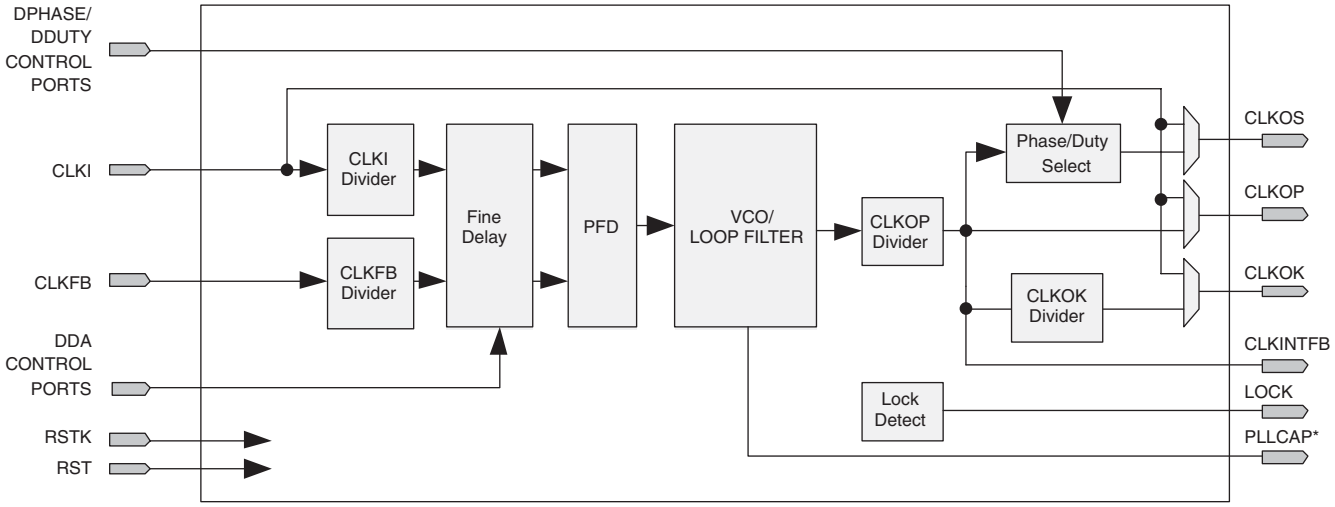
Figure 8-3. Clock Attributes in the Pre-map Preference Editor



sysCLOCK PLL

The LatticeECP2 PLL provides features such as clock injection delay removal, frequency synthesis, phase/duty cycle adjustment, and dynamic delay adjustment. Figure 8-4 shows the block diagram of the PLL.

Figure 8-4. 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.

Delay Adjustment

The delay adjust circuit provides programmable clock delay. The programmable clock delay allows for step delays in increments of 130ps (nominal) for a total of 1.04ns lagging or leading. The time delay setting has a tolerance. See device data sheet for details. Under this mode, CLKOP, CLKOS and CLKOK are identically affected. The delay adjustment has two modes of operation:

- Static Delay Adjustment
  - In this mode, the user-selected delay is configured at power-up.
- Dynamic Delay Adjustment (DDA)
  - In this mode, a simple bus is used to configure the delay. The bus signals are available to the general purpose FPGA.

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 420MHz to 840MHz range to minimize jitter. The CLKOP Divider values are the same whether or not CLKOS is used.

CLKOK Divider
The CLKOK divider feeds the global clock net. It divides the CLKOP signal of the PLL by the value of the divider to produce lower frequency clock.

Phase Adjustment and Duty Cycle Select
User can program CLKOS with Phase and Duty Cycle options. Phase adjustment can be done in 22.5° step. The Duty Cycle resolution is 1/16th of a period, however 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
With LatticeECP2 device families, users can control the Phase Adjustment and Duty Cycle Select 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 fixed logic levels of choice.

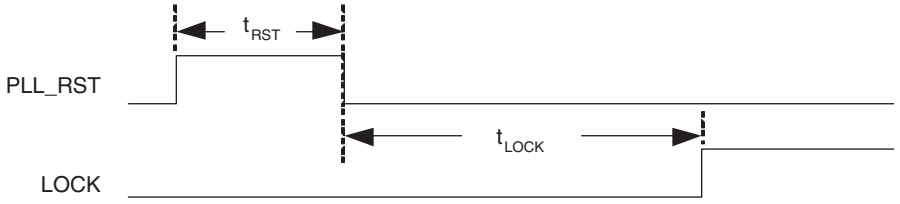
External Capacitor
An optional external capacitor can be used with PLLs to accommodate the low frequency input clock. See the Optional External Capacitor section of this document for further information.

PLL Inputs and Outputs

CLKI Input
The CLKI signal is the reference clock for the PLL. It must conform to the specifications in the LatticeECP2 data sheet 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 M-Dividers), 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 8-5 shows the timing diagram of the RST Input. RST is active high.

Figure 8-5. 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. LatticeECP2 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 dedicated dual-purpose pin, directly from the CLKOP divider (CLKINTFB) 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 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.

Dynamic Delay Control/Dynamic Phase Adjustment/Dynamic Duty Cycle
Detailed information about these features are described later in this document.

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.

CLKINTFB
CLKOP Divider output before CLOCK TREE is routed to CLKFB input in Internal Feedback mode. This signal is user-transparent.

Dynamic Delay Adjust and Dynamic Phase and Duty Cycle Adjust Ports

Parameter	I/O	Description
DDAMODE	I	DDA (Dynamic Delay Adjust) Mode. “1” Pin control (dynamic), “0”: Fuse Control (static)
DDAIZR	I	DDA Delay Zero. “1”: delay = 0, “0”: delay = on,
DDAILAG	I	DDA Lag/Lead. “1”: Lead, “0”: Lag
DDAIDEL[2:0}	I	DDA Delay Step value.
DPAMODE	I	DPA (Dynamic Phase Adjust/Duty Cycle Select) mode
DPHASE[3:0]	I	DPA Phase Adjust inputs
DDUTY[3:0]	I	DPA Duty Cycle Select inputs

PLLCAP
This port is not included in the software module, instead it is hard-wired to the PLLCAP pin of the device. See the Optional External Capacitor section of this document for further information.

PLL Attributes

The PLL utilizes several attributes that allow the configuration of the PLL through source constraints and preference files. 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; determined by the input frequency, the dividers, and the PLL specifications.

Note: Unlike PLLs in the LatticeECP™, LatticeEC™, LatticeXP™ and MacoXO™ 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 CLKI_DIV and CLKFB_DIV values. These value settings are designed so that the output clock duty cycle is as close to 50% as possible.

Table 8-2. Divider Ranges

Attribute	Name	Value	Default
CLKI Divider Setting	CLKI_DIV	1 to 64	1
CLKFB Divider Setting	CLKFB_DIV	1 to 64	1
CLKOP Divider Setting	CLKOP_DIV	2, 4, 8, 16, 32, 48, 64, 80, 96, 112, 128	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 desired output frequency is not achievable, the user may enter the frequency tolerance of the clock output.

FDEL (Fine Delay Adjust: EHXPLLD only)
The FDEL attribute is used to pass the Delay Adjustment step associated with the Output Clock of the PLL. This allows the user to advance or retard the Output Clock by the step value passed multiplied by 130ps (nominal). The step ranges from -8 to +8 resulting the total delay range to +/-1.04ns.

Note: FDEL entry is not available in the IPexpress™ GUI. There are four ways for user to enter the desired FDEL value.

1. Enter the FDEL attribute value in source code generated by IPexpress.
2. Use the Pre-map Preference Editor.
3. Use the preference file. The preference syntax is described in the ispLEVER® on-line HELP files.
4. Use EPIC Editor.

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 (CLKINTFB) 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.

DELAY_CNTL
This attribute is designed to select the Delay Adjustment mode. If the attribute is set to “DYNAMIC” the delay control switches between Dynamic and Static depending upon the input logic of the DDAMODE pin. If the attribute is set to “STATIC”, Dynamic Delay inputs are ignored in this mode.

PHASE/DUTY_CNTL
This attribute is designed to select the Phase Adjustment/Duty Cycle Select mode. If the attribute is set to “DYNAMIC” the Phase Adjustment/Duty Cycle Select control switches between Dynamic and Static depending upon the input logic of DPAMODE pin. If the attribute is set to “STATIC”, Dynamic Phase Adjustment/Duty Cycle Select inputs are ignored in this 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.

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

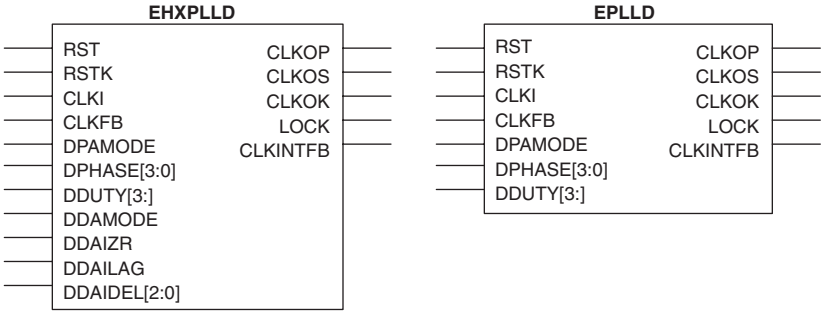
CLKOP/CLKOS/CLKOK BYPASS
These bypass are enabled if set. The CLKI is routed to corresponding output clock directly.

LatticeECP2 PLL Primitive Definitions

All the devices in the LatticeECP2 family support two General Purpose PLLs (GPLLs) which are full-featured PLLs. In addition some of the larger devices have two to four Standard PLLs (SPLLs) that have a subset of GPLL funtionalities.

Two PLL primitives are defined for LatticeECP2 PLL implementation. Figure 8-6 shows the LatticeECP2 PLL primitive library symbols. The GPLL may be configured as either EPLLD or EHXPLLD. The SPLL can be configured as EPLLD only.

Figure 8-6. LatticeECP2 PLL Primitive Symbols



Dynamic Delay Adjustment (EHXPLLD Only)

The Dynamic Delay Adjustment is controlled by the DDAMODE input. When the DDAMODE input is set to “1”, the delay control is done through the inputs, DDAIZR, DDAILAG and DDAIDEL(2:0). For this mode, the attribute “DELAY_CNTL” must be set to “DYNAMIC”. Table 8-3 shows the delay adjustment values based on the attribute/input settings.

In this mode, the PLL may come out of lock due to the abrupt change of phase. RST must be asserted to re-lock the PLL. Upon de-assertion of RST, the PLL will start the lock-in process and will take the t_{LOCK} time to complete the PLL lock.

Table 8-3. Delay Adjustment

DDAMODE = 1: Dynamic Delay Adjustment			Delay 1 Tdly = 130 ps (nominal)	DDAMODE = 0
DDAIZR	DDAILAG	DDAIDEL[2:0]		Equivalent FDEL Value
0	1	111	Lead 8 Tdly	-8
0	1	110	Lead 7 Tdly	-7
0	1	101	Lead 6 Tdly	-6
0	1	100	Lead 5 Tdly	-5
0	1	011	Lead 4 Tdly	-4
0	1	010	Lead 3 Tdly	-3
0	1	001	Lead 2 Tdly	-2
0	1	000	Lead 1 Tdly	-1
1	Don't Care	Don't Care	no delay	0
0	0	000	Lag 1 Tdly	1
0	0	001	Lag 2 Tdly	2
0	0	010	Lag 3 Tdly	3
0	0	011	Lag 4 Tdly	4
0	0	100	Lag 5 Tdly	5
0	0	101	Lag 6 Tdly	6
0	0	110	Lag 7 Tdly	7
0	0	111	Lag 8 Tdly	8

Dynamic Phase Adjustment/Duty Cycle Select

Phase Adjustment Settings are described in Table 8-4.

Table 8-4. Phase Adjustment Settings

DPHASE[3:0]	Phase (O)
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 8-5.

Table 8-5. Duty Cycle Select Settings

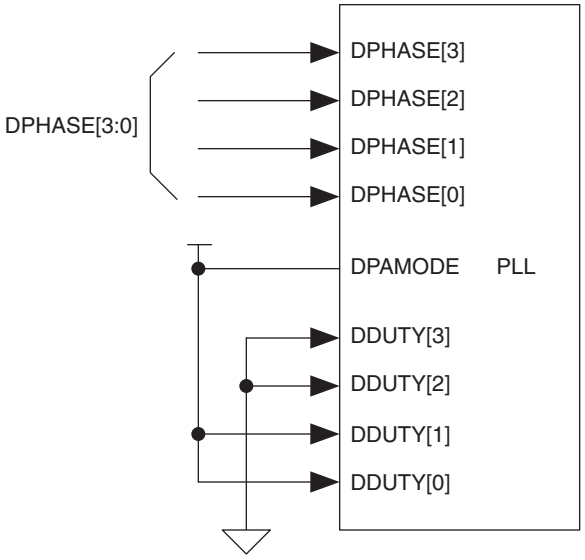
DDUTY[3:0]	Duty Cycle (1/16th of a period)	Comment
0000	0	Not Supported
0001	1	Not Supported
0010	2	
0011	3	
0100	4	
0101	5	
0110	6	
0111	7	
1000	8	
1001	9	
1010	10	
1011	11	
1100	12	
1101	13	
1110	14	
1111	15	Not Supported

Note: When PHASE/DUTY_CTNL is selected in the GUI 'Dynamic Control' box and the DPAMODE is set to '1', then both DPHASE[3:0] and DDUTY[3:0] inputs must be provided. If one of these inputs is a fixed value, the user must tie the inputs to the desired fixed logic levels.

Example:

If a design uses dynamic phase adjustment and a fixed duty cycle select and the desired duty cycle 3/16th of a period, then the set up should be as shown in Figure 8-7.

Figure 8-7. Example Hardware Setup - Dynamic Phase Adjustment /Duty Cycle Select



Optional External Capacitor

An optional external capacitor can be used with both the EXHPLLD and the EPLLD to change the frequency response of the on-chip loop filter. When an external capacitor is used, the frequency at the phase detector inputs (Fpd) can be as low as 1MHz, allowing the PLLs to extend the low-end of their operating ranges. The external capacitor has no effect on the high-end of their operating ranges. IPexpress checks the phase detector frequency to determine if an external capacitor is required or not.

The allowable ranges for the PLL parameters, with and without the external capacitor, are described in the LatticeECP2 Family Data Sheet.

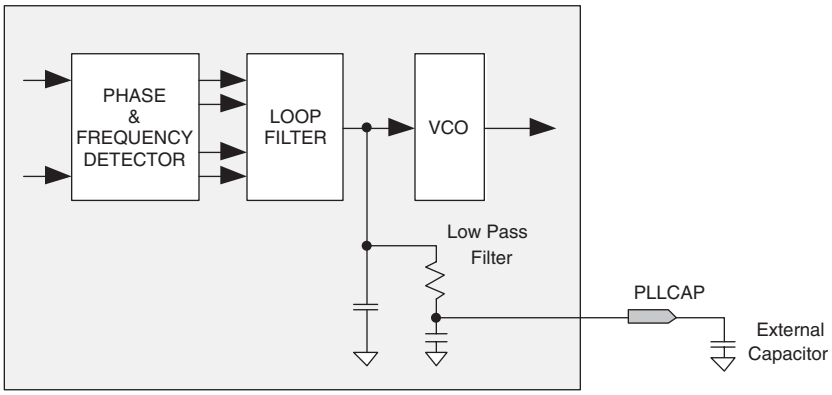
Recommended Optional External Capacitor Specifications

Value: 5.6 nF, +/- 20%
Type: Ceramic chip capacitor, NPO dielectric
Package: 1206 or smaller

Each device has two external capacitor pins, one for the left side PLLs and one for the right side PLLs. These pins are in fixed locations. They are dedicated function pins that are NOT shared with user I/Os.

When an external capacitor pin is used by a PLL on one side of the device, it cannot be used by any other PLLs on that same side of the device. This means that a maximum of two PLLs per device, one on the left side and one on the right side, can have external capacitors attached.

Figure 8-8. External Capacitor Usage

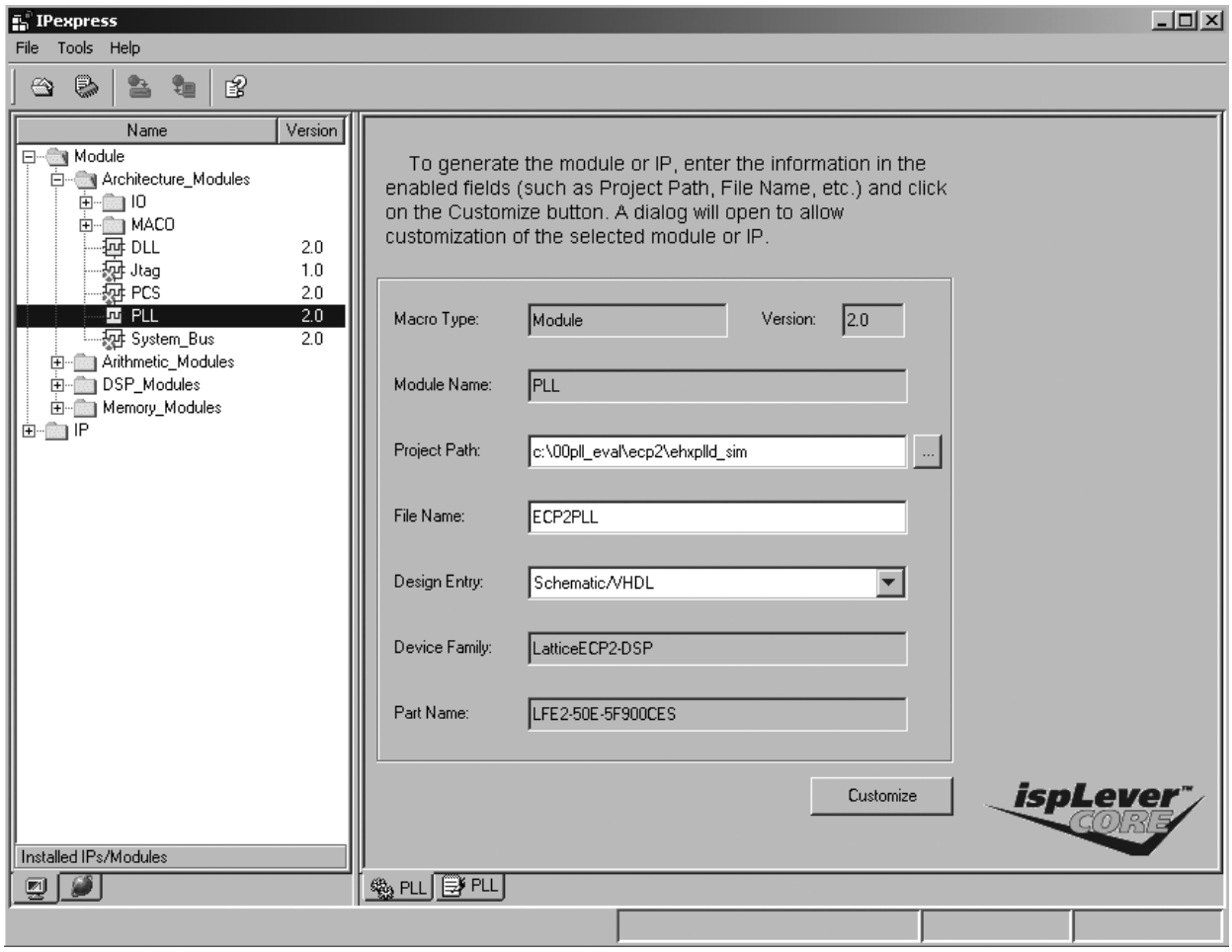


PLL Usage in IPexpress

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

Figure 8-9 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. 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 8-10.

Figure 8-9. IPexpress Main Window



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.

Modes

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

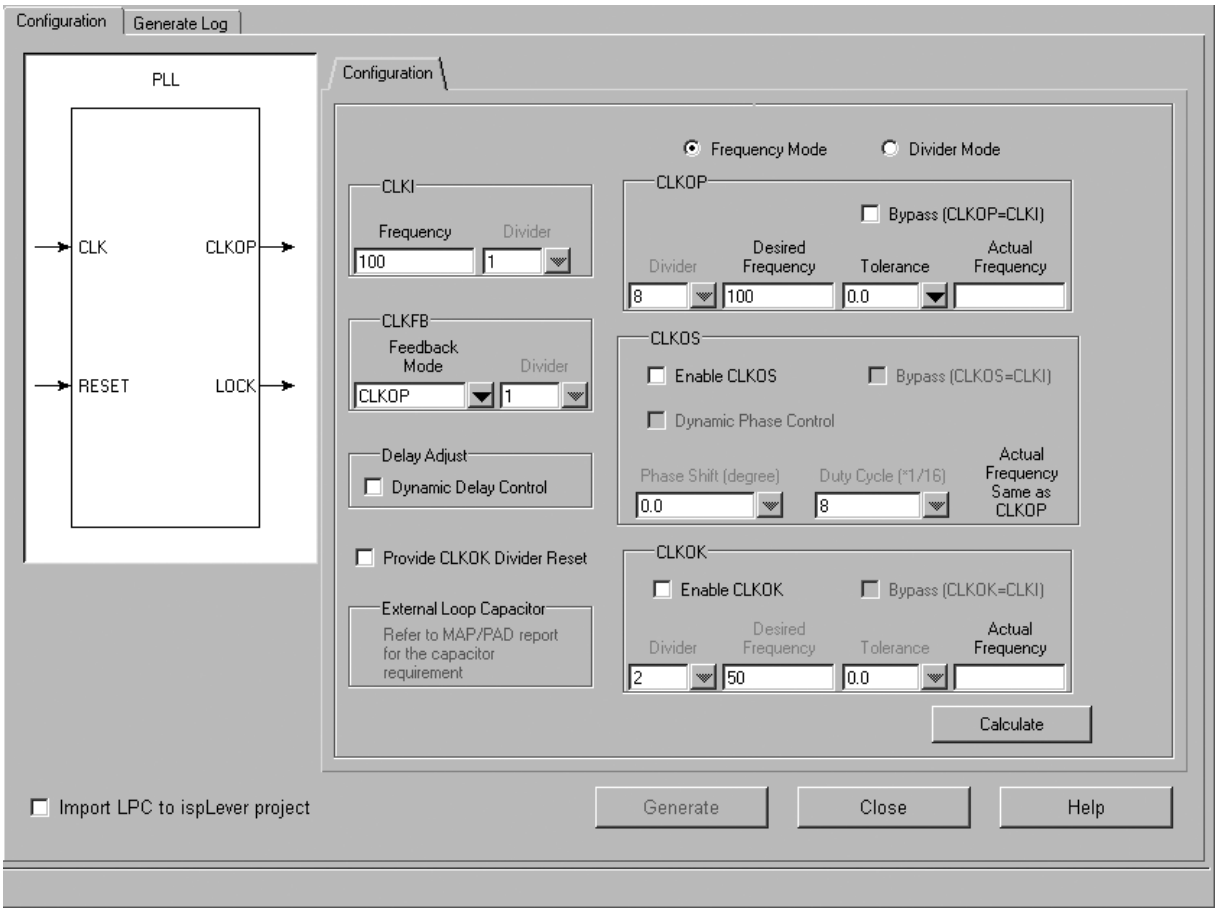
Frequency Mode

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 the input frequency. The user must choose the CLKOP Divider value in order to maximize the f_{VCO} and achieve optimum PLL performance. After setting the input frequency and divider settings, click **Calculate** to display the frequencies. Figure 8-10 shows the Configuration Tab.

Figure 8-10. LatticeECP2 PLL Configuration Tab



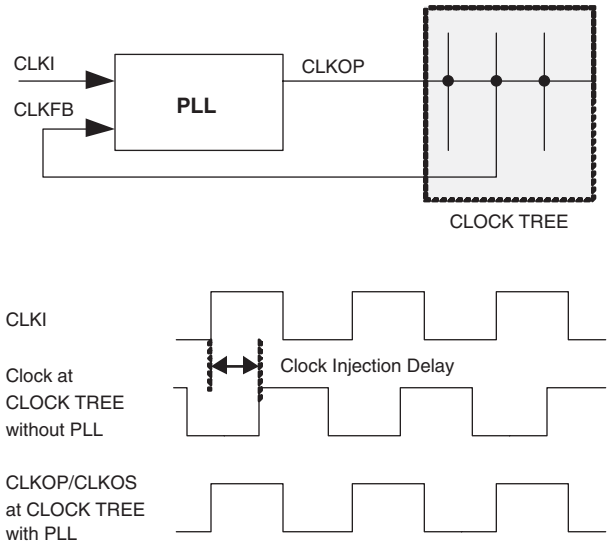
Note: In the External Loop Capacitor, the grayed out text turns on if the CLKI frequency is less than 25MHz to alert the user about the required external loop capacitor. Other grayed-out sections turn on as their sections are enabled.

PLL Modes of Operation

PLL Clock Injection Removal

PLLs are 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 8-11 Illustrates an example block diagram and waveform.

Figure 8-11. Clock Injection Delay Removal Application

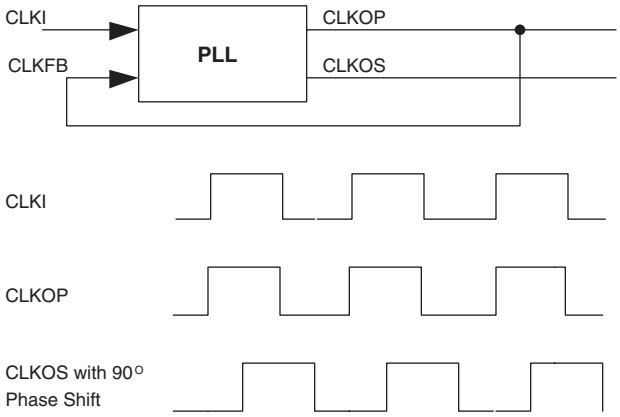


PLL Clock Phase Adjustment

The PLL is used to create fixed phase relationships in 22.5º increments. Creating fixed phase relationships are useful for forward clock interfaces where a specific relationship between the clock and data is required.

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

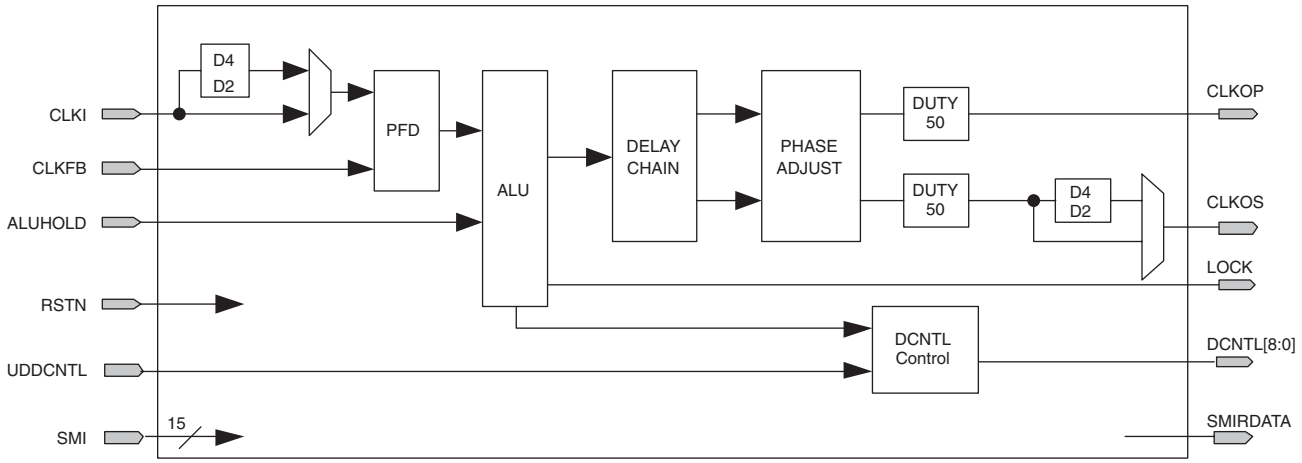
Figure 8-12. CLKOS Phase Adjustment from CLKOP



sysCLOCK DLL

The LatticeECP2 DLL provides features such as clock injection delay removal, delay match, time reference delay (90º phase delay), and output phase adjustment. The DLL performs clock manipulation by adding delay to the CLKI input signal to create specific phase relationships. There are two types of outputs of the DLL. The first are clock signals similar to the PLL CLKOP and CLKOS. The other type of output is a delay control vector (DCNTL[8:0]). The delay control vector is connected to a Slave Delay Line (DLLDEL) element located in the I/O logic which matches the delay cells in the DLL. This delay vector allows the DLL to dynamically delay an input signal by a specific amount. Figure 8-13 provides a block diagram of the LatticeECP2 DLL.

Figure 8-13. LatticeECP2 DLL Block Diagram



Both clock injection delay removal and output phase adjustment use only the clock outputs of the DLL. Time reference delay and delay match modes use the delay control vector output. Specific examples of these features will be discussed later in this document.

DLL Overview

The LatticeECP2 DLL is created and configured by IPexpress. The following is a list of port names and descriptions for the DLL. There are three library elements used to implement the DLL: CIDDLLA (Clock Injection Delay), CIMDLLA (Clock Injection delay Match), and TRDDLLA (Time Reference Delay). IPexpress will wrap one of these library elements to create a customized DLL module based on user selections.

DLL Inputs and Outputs

CLKI Input

The CLKI signal is the reference clock for the DLL. The CLKI input can be sourced from any type of FPGA routing and pin. The DLL CLKI input has a preferred pin per DLL which provides the lowest latency and best case performance.

CLKFB Input

The CLKFB input is available only if the user chooses to use a user clock signal for the feedback or in clock delay match mode. If internal feedback or CLKOS/CLKOP is used for the feedback, this connection will be made inside the module. In Clock Injection Delay Removal mode the DLL will align the input clock phase with the feedback clock phase by delaying the input clock.

In Clock Injection Delay Match Mode the DLL will calculate the delta between the CLKI and CLKFB signals. This delay value is then output on the DCNTL vector. The DLL CLKFB input has a preferred pin per DLL which is discussed later in this document. The preferred pin provides the lowest latency and best case performance.

CLKOP Output

An output of the DLL based on the CLKI rate. The CLKOP output can drive primary, edge, and secondary clock routing.

CLKOS Output

An output of the PLL based on the CLKI rate which can be divided and/or phase shifted. The CLKOS output can drive the primary, edge, and secondary clock routing.

DCNTL[8:0] Output

This output of the DLL is used to delay a signal by a specific amount. The DCNTL[8:0] vector connects to a Slave Delay Line element. The DLL can then control multiple input delays from a single DLL.

UDDCNTL Input
This input is used to enable or disable updating of the DCNTL[8:0]. To ensure that the signal is captured by the synchronizer in the DLL block, it must be driven high for a time equal to at least two clock cycles when an update is required. If the signal is driven high and held in that state, the DCNTL[8:0] outputs are continuously updated.

ALUHOLD Input
This active high input stops the DLL from adding and subtracting delays to the CLKI signal. The DCNTL[8:0], CLKOP, and CLKOS outputs will still be valid, but will not change from the current delay setting.

LOCK Output
Active high lock indicator output. The LOCK output will be high when the CLKI and CLKFB signal are in phase. If the CLKI input stops the LOCK output will remain asserted. The clock is stopped so there is no clock to de-assert the LOCK output. Note that this is different from the operation of the PLL where the VCO continues to run when the input clock stops.

RSTN
Active low reset input to reset the DLL. The DLL can optionally be reset by the GSRN as well. It is recommended that if the DLL requires a reset, the reset should not be the same as the FPGA logic reset. Typically, logic requires that a clock is running during a reset condition. If the data path reset also resets the DLL, the source of the logic clock will stop and this may cause problems in the logic.

SMI (Serial Memory Interface)
The DLL supports a run time control interface for modifying the behavior of the DLL in the system. Such parameters as output dividers and phase offset can be changed while in the system. This control interface is known as the Serial Memory Interface (SMI). More information on using the SMI will be found in Appendix A.

DLL Attributes

The LatticeECP2 DLL utilizes several attributes that allow the configuration of the DLL through source constraints, IPexpress and preference files. The following section details these attributes and their usage.

DLL Lock on Divide by 2 or Divide by 4 CLKOS Output
Usually, the DLL is a 'times one' device, allowing neither frequency multiplication or division. But the LatticeECP2 DLL allows 'divide by 2' or 'divide by 4' CLKOS outputs. Two optional 'divide by 2' and 'divide by 4' blocks are placed at the CLKI input as well as the CLKOS and this enables the use of divided CLKOS in the DLL feedback path. This allows the DLL to perform clock injection removal on a 'divide by 2' or 'divide by 4' clock, which is useful for DDRX2 and DDRX4 modes of I/O buffer operation.

When this optional clock divider is used only in the CLKOS output path, it allows the DLL to output two time-aligned clocks at different frequencies. When the divider is set to divide by 2 or divide by 4, a 'dummy' delay is inserted in the CLKOP output path to match the clock to Q delay of the CLKOS divider.

Optional Clock Fine Phase Shift
The optional fine phase shift in the CLKOS output path is built from a delay block that matches the other four blocks in the main delay chain. This delay block allows the CLKOS output to phase shifted a further 45, 22.5 or 11.25 degrees relative to its normal position.

Duty Cycle Selection
TRDLLA has an optional 50% Duty Cycle Selection feature. The inverted output of the DUTY50 block provides another 180-degree phase shift from the source.

CIDDLLA also has optional 50% Duty Cycle Selection feature.

GSR

The PLL and DLL can be reset by the GSR, if enabled. The GSR keyword can be set to ENABLED/DISABLED. This option is provided in the IPexpress GUI. Below is an example of the use of this preference.

```
ASIC "dll/dll_0_0" TYPE "EHXPLLA" GSR=DISABLED;
```

DLL Lock Time Control

The DLL will lock when the CLKI and CLKFB phases are aligned. In a simulation environment, the lock time is fixed to 100µs (default). This value can be changed through an HDL parameter or preference (for the back annotation simulation). The DLL contains a parameter named LOCK_DELAY which accepts an integer value for the total time in µs until the lock output goes high. Below is an example of how to set this value for front-end simulation. This option is also available in the IPexpress GUI.

Verilog:

```
defparam mydll.mypll_0_0.LOCK_DELAY=500;
mydll dll_inst(.CLKI(clkin), .CLKOP(clk1), .CLKOS(clk2),
```

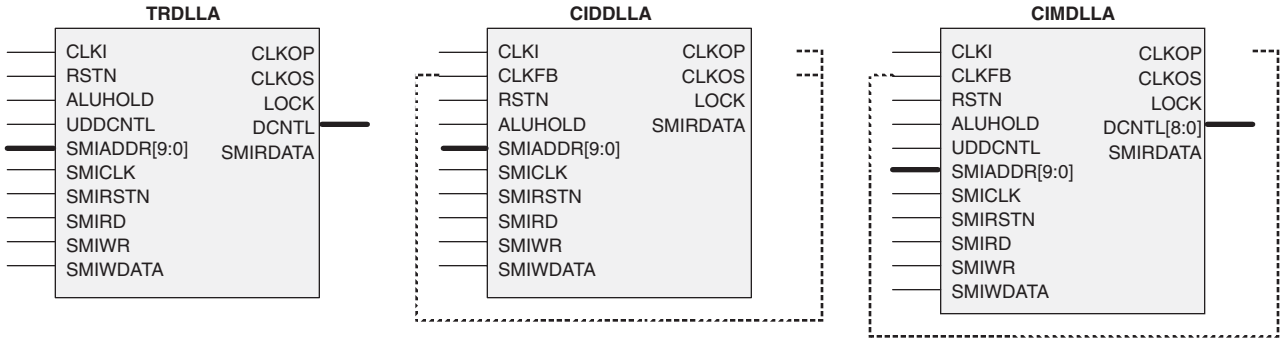
VHDL:

Not supported. For back annotation simulation LOCK_DELAY needs to be set in the preference file. Below is an example for the PLL.

```
ASIC "pll/pll_0_0" TYPE "EHXPLLA" LOCK_DELAY=200;
```

DLL Primitive Symbols

Figure 8-14. DLL Primitive Symbols



DLL Primitives Definition

The Lattice library contains primitives to allow designers to utilize the DLL. These primitives use the DLL attributes defined in the "DLL Attributes" section.

The three modes of operation will be presented as primitives (or library elements) as listed below.

Table 8-6. DLL Primitives

Primitive Name	Mode of Operation	Description
TRDLLA	Time Reference Delay DLL	This mode generates four phases of the clock, 0^O, 90^O, 180^O, 270^O, along with the control setting used to generate these phases.
CIDDLLA	Clock Injection Delay DLL (Four Delay Cell Mode)	This mode removes the clock tree delay, aligning the external feedback clock to the reference clock. It has a single output coming from the fourth delay block.
	Clock Injection Delay DLL (Single Delay Cell Mode)	This mode takes a single delay cell, corrects for clock injection and enables the 9-bit ALU output
CIMDLLA	Clock Injection Match DLL	This mode matches the delay between the input clock and the feedback clock to one delay cell. This allows other inputs to take the registered ALU outputs and adjust their delay to match the clock-to-clock delay

DLL Primitive I/Os

Table 8-7. DLL Primitive IO Description

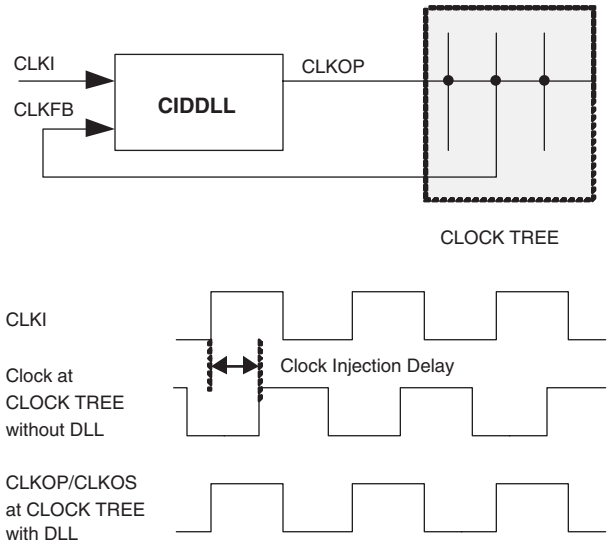
Signal	I/O	Description	Freq. (MHz)
CLKI	I	Clock input pin from (1) dedicated clock input pin (2) other I/O or logic block.	100 - 700
CLKFB	I	Clock feedback input pin from (1) dedicated feedback input pin, (2) internal feedback or (3) other I/O or logic block. This signal is not user selectable.	100 - 700
RSTN	I	Active low synchronous reset. From dedicated pin or internal node.	
ALUHOLD	I	“1” freezes the ALU. For TRDLLA, CIDDLLA and CIMDLLA.	
UDDCNTL	I	Active high synchronous enable signal from CIB for updating digital control to PIC delay. It must be driven high at least two clock cycles.	
DCNTL[8:0]	O	Digital delay control code signals.	
CLKOP	O	The primary clock output for all possible modes.	1 - 700
CLKOS	O	The secondary clock output with finer phase shift and/or division by 2 or by 4.	1 - 700
LOCK	O	Active high phase lock indicator. Lock means the reference and feedback clocks are in phase.	
SMIADDR[9:0]	I	SMI Address	
SMICLK	I	SMI Clock	
SMIRSTN	I	SMI Reset (Active low)	
SMIRD	I	SMI Read	
SMIWDATA	I	SMI Write Data	
SMIWR	I	SMI Write	
SMIRDATA	O	SMI Read Data	

DLL Modes of Operation

Clock Injection Removal Mode (CIDDLLA)

The DLL can be used to reduce clock injection delay (CIDDLLA). Clock injection delay is the delay from the input pin of the device to a destination element such as a flip-flop. The DLL will add delay to the CLKI input to align CLKI to CLKFB. If the CLKFB signal comes from the clock tree (CLKOP, CLKOS) then the delay of the DLL and the clock tree will be removed from the overall clock path. Figure 8-15 shows a circuit example and waveform.

Figure 8-15. Clock Injection Delay Removal via DLL

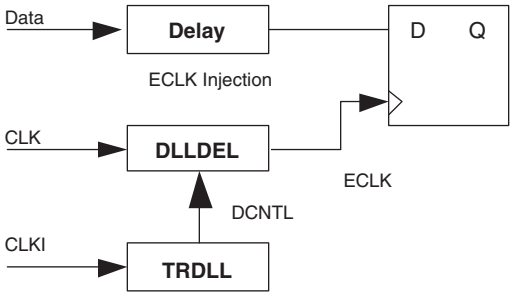


Clock injection removal mode can also provide a DCNTL port. In this mode the delay added to the CLKI signal is output on the DCNTL port so that other input signals can be delayed by the same amount. This is very useful if several clocks are used in the same circuit to minimize the number of DLLs required. When using the DCNTL, the DLL delay will be limited to the range of the DCNTL vector. Therefore, IPexpress will restrict the CLKI rate from 300MHz to 700MHz.

Time Reference Delay Mode (TRDLLA: 90-Degree Phase Delay)
The Time Reference Delay (TRDDLLA) mode of the DLL is used to calculate 90 degrees of delay to be placed on the DCNTL vector. This is a useful mode in delaying a clock 90 degrees for use in clocking a DDR type interface.

Figure 8-16 provides a circuit example of this mode.

Figure 8-16. Time Reference Delay Circuit Example

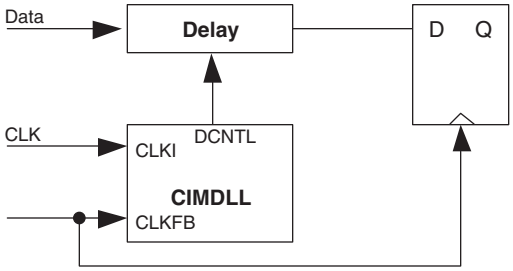


In this mode, CLKI accepts a clock input. The DLL produces a DCNTL vector that will delay an input signal by 90 degrees of a full period of the CLKI signal. This DCNTL vector can then be connected to a Slave Delay Line (DLLDELA) to delay the signal by 90 degrees of the full period of CLKI.

Clock Delay Match Mode (CIMDLLA)
The Clock Delay Match (CIMDLLA) mode of the DLL allows two synchronous clock inputs to be used to create a DCNTL vector that is the delta of the delay between the two clocks. In this circuit the DLL will compare the phase difference between the CLKI and CLKFB inputs to create a delay vector. This is a useful mode when data is transmitted off one clock (CLKI) and captured on a different clock (CLKFB) that are synchronous to each other. By

delaying the data inside the I/O logic before being latched by CLKFB, the data signal will be delayed the same amount as the difference in the clock delay. Figure 8-17 provides a circuit example of this mode.

Figure 8-17. Clock Delay Match Circuit Example



In Clock Delay Match mode the maximum delay is limited to 3.9ns.

DLL Usage in IPexpress

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

Configuration Tab

- **Usage Mode** – Select the mode of the DLL (Time Reference Delay [TRDLLA], Clock Injection Delay Removal [CIDDLLA], or Clock Delay Match [CIMDLLA]). This selection will enable or disable further options in the GUI.
- **CLKI Frequency** – The rate of the CLKI input in MHz.
- **CLKOS Divider** – Set the divider for the CLKOS output to be either no divide, divide by 2, or divide by 4.
- **CLKOS Phase Shift** – Set the phase offset of the CLKOS to the CLKOP output. CLKOP will lead CLKOS by the amount of phase shift selected. The phase increment is 11.25 degrees. The pull-down list numbers are abbreviated to a decimal point.
- **CLKFB Feedback Mode** – Sets the feedback mode of the DLL to either CLKOP, CLKOS or User Clock. If CLKOP/CLKOS is selected, the clock tree injection delay for the specific output clock will be removed. If User Clock is selected, the user will be provided with the CLKFB port on the DLL.
- **CLKFB Frequency** – This is used only with User Clock Feedback Mode.
- **Provide RSTN Port** – The RSTN port allows the user to reset the DLL through a user signal.
- **Enable GSR to reset DLL** – If selected, the DLL will be reset via the GSR. No user signal is required. RSTN port can still be used.
- **Provide SMI Ports** – The SMI ports allow the user to change DLL behavior for output dividers and phase offset in-system.
- **Provide DCNTL Port** – In Clock Injection Delay Mode it is possible to obtain the delay added to the CLKI port on the DCNTL port. This can then be used to delay other clock inputs by the same amount by connecting the DCNTL vector to DELAY elements.

PLL/DLL Cascading

It is possible to connect several arrangements of PLLs and DLLs. There are three possible cascading schemes:

- PLL to PLL
- PLL to DLL
- DLL to DLL

It is not possible to connect the DLL to a PLL. The DLL produces abrupt changes on its output clocks when changing delay settings. The PLL sees this as radical phase changes that prevent the PLL from locking correctly.

IPexpress Output

There are two outputs of IPexpress 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 file is a template file <module_name>_tmpl.[v|vhd]. This file contains a sample instantiation of the module. This file is only 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/DLL, IPexpress sets attributes in the HDL module created that are specific to the data rate selected. Although these attributes can easily be changed, they should only be modified by re-running the GUI so that the performance of the PLL/DLL 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/DLL.

Clock Dividers (CLKDIV)

The clock divider can divide the high-speed edge clock by 2, 4 or 8. The divided output can then be used as a primary clock or secondary clock input. The phase can be changed by 180 degrees for divided by 2 or 90/180/270 degrees when divided by 4. The phase change is used to facilitate clock domain transfer after the de-mux operation in PIC. 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 18, Table 8-6 and Table 8-7 describe the definition of CLKDIVB.

Figure 8-18. CLKDIV Primitive Symbol

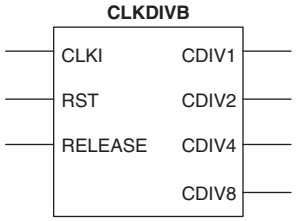


Table 8-8. CLKDIVB Port Definitions

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 8-9. 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

CLKDIVint0: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
         );
```

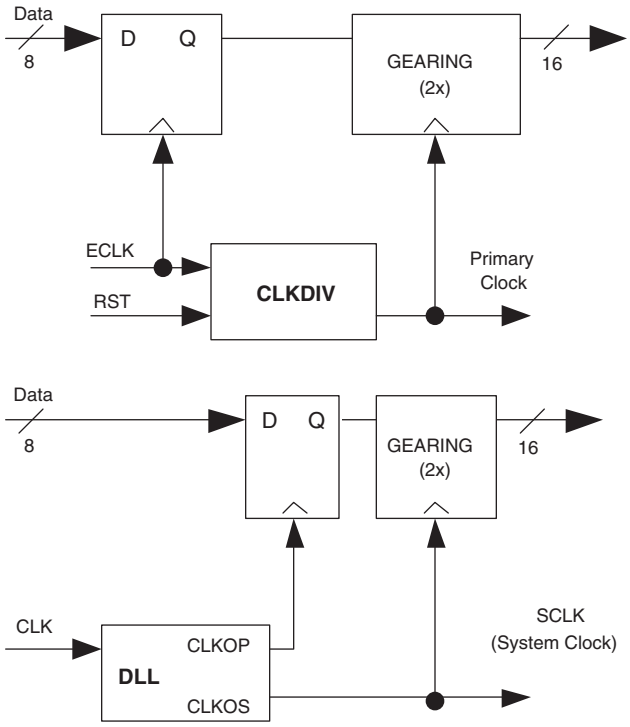
CLKDIV Example Circuits

The clock divider (CLKDIV) can divide a clock by 2 or 4 and drives a primary clock network. The 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 drives a primary clock. In this case, they are phase matched. This is especially useful for synchronously resetting the I/O logic when Mux/DeMux gearing is used in order to synchronize the entire data bus as shown in Figure 8-19. 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.

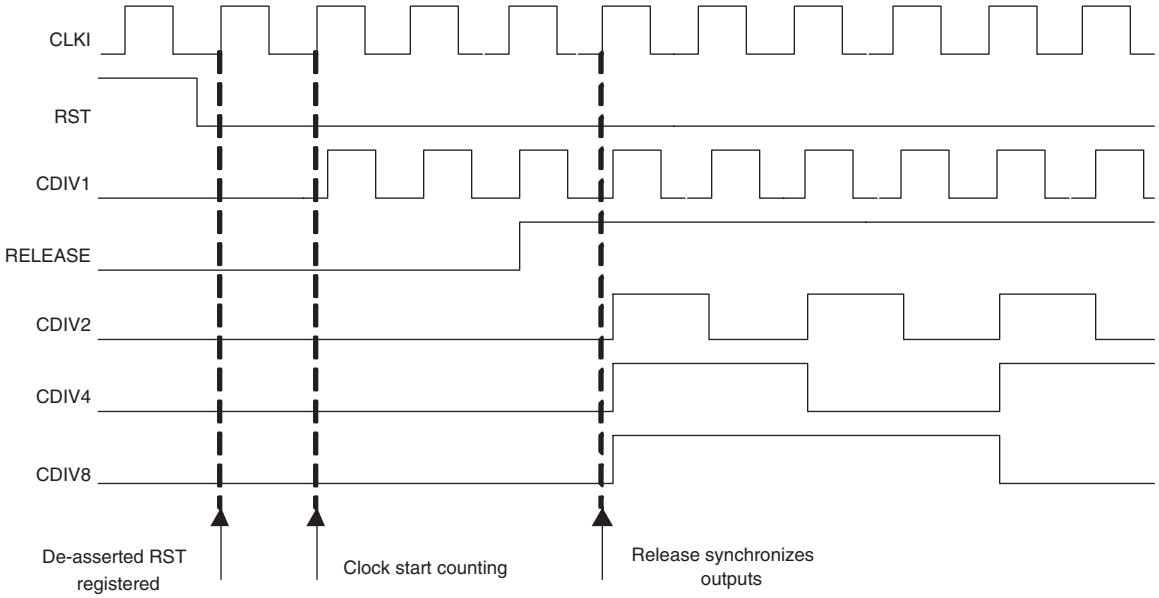
Figure 8-19. CLKDIV Application Examples



Release Behavior

The port “Release” is used to synchronize the all outputs after RST is de-asserted. Figure 8-20 Illustrates the Release behavior.

Figure 8-20. CLKDIV Release Behavior



DLLDEL (Slave Delay Line)

The Slave Delay line is designed to generate desired delay in DDR/SPI4 applications. The delay control inputs (DCNTL[8:0]) are fed from the general purpose DLL outputs. The primitive definitions are described in Figure 8-21 and Figure 8-10.

Figure 8-21. DLLDELA Primitive Symbol

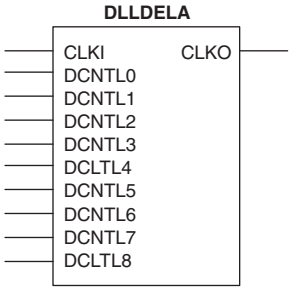


Table 8-10. DLLDELA I/O

Name	I/O	Description
CLKI	I	Clock Input
DCNTL[8:0]	I	Delay Control Bits
CLKO	O	Clock Output

DLLDELA Declaration in VHDL Source Code

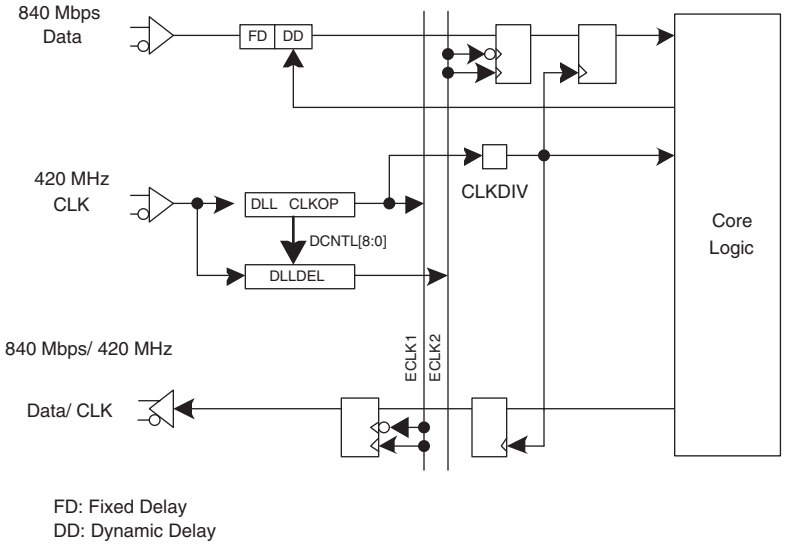
```
COMPONENT DLLDELA
      PORT     (
               CLKI :IN std_logic;
               DCNTL0 :IN std_logic;
               DCNTL1 :IN std_logic;
               DCNTL2 :IN std_logic;
               DCNTL3 :IN std_logic;
               DCNTL4 :IN std_logic;
               DCNTL5 :IN std_logic;
               DCNTL6 :IN std_logic;
               DCNTL7 :IN std_logic;
               DCNTL8 :IN std_logic;
               CLKO :OUT std_logic
               );
END COMPONENT;

begin
DLLDELAinst0: DLLDELA1
      PORT MAP (
               CLKI => clkisig,
               DCNTL0 => dcntl0sig,
               DCNTL1 => dcntl1sig,
               DCNTL2 => dcntl2sig,
               DCNTL3 => dcntl3sig,
               DCNTL4 => dcntl4sig,
               DCNTL5 => dcntl5sig,
               DCNTL6 => dcntl6sig,
               DCNTL7 => dcntl7sig,
```

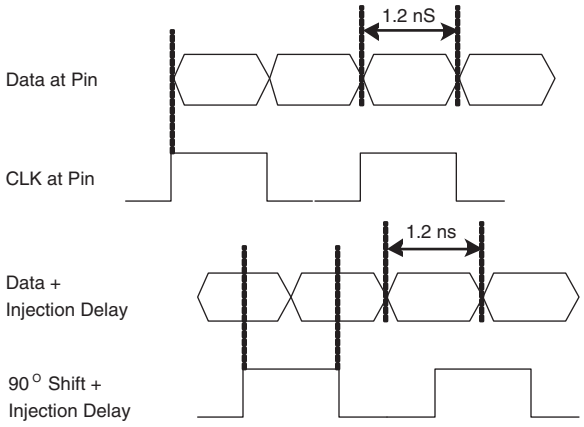
```
DCNTL8 => dcntl8sig,
CLKO => clkosig
);
```

DLLDELA Application Example

Figure 8-22 describes an example of DDLDEL application. As shown in the timing diagram, DDLDEL shifts the clock by 90 degrees to center both edges in the middle of data window.

Figure 8-22. SPI4.2 and DDR Registers Interface Application





DQSDLL and DQSDEL

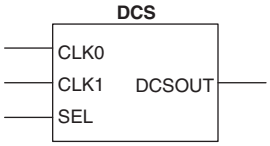
There is another combination of DLL and Slave Delay Line, DQSDLL and DQSDEL, in the LatticeECP2 device family. This pair is similar in design and function to DLL and DLLDEL, but usage is limited to DDR implementation. For additional information, see Lattice Technical Note, TN1102, *LatticeECP2 sysIO Usage Guide.*

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 8-23 shows the block diagram of the DCS.

Figure 8-23. DCS Primitive Symbol



DCS Primitive Definition

Table 8-11 defines the I/O ports of the DCS block. There are eight modes from which the user can select. Table 8-12 describes how each mode is configured.

Table 8-11. 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 8-12. DCS Modes of Operations

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 its unique operation. The clock output timing is determined by input clocks and the edge of SEL signal. Figure 8-24 describes the timing of each mode.

Figure 8-24. Timing Diagrams by DCS MODE

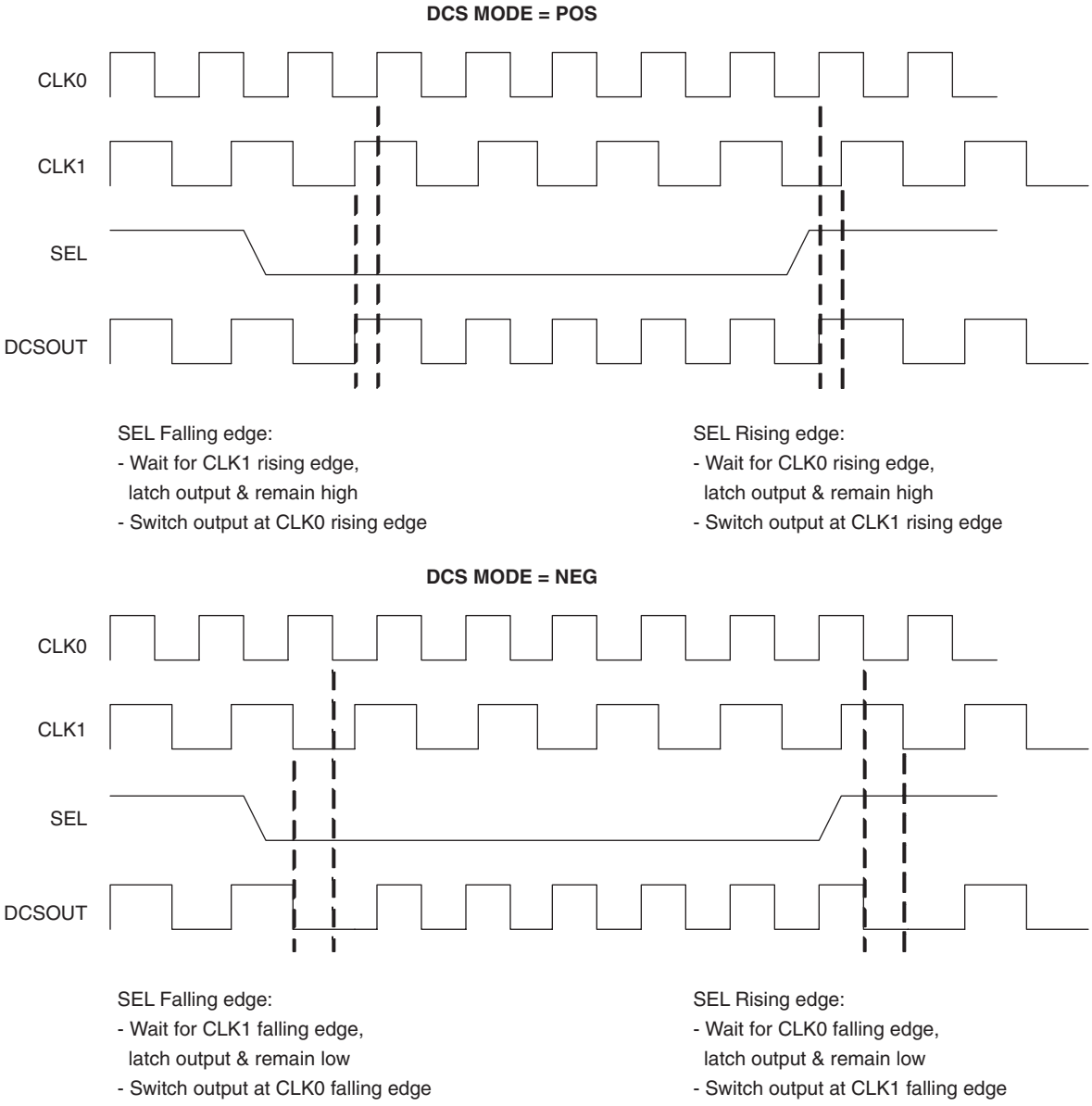
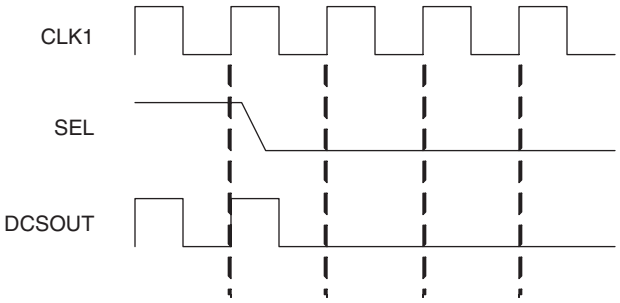


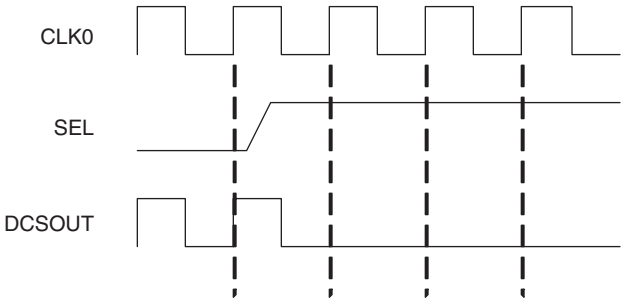
Figure 8-24. Timing Diagrams by DCS MODE (Cont.)

DCS MODE = HIGH_LOW



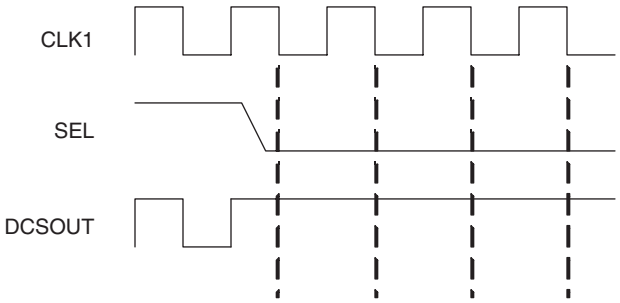
- Switch low @CLK1 falling edge.
- If SEL is low, output stays low at on CLK1 rising edge. SEL must not change during setup prior to rising clock.

DCS MODE = LOW_LOW



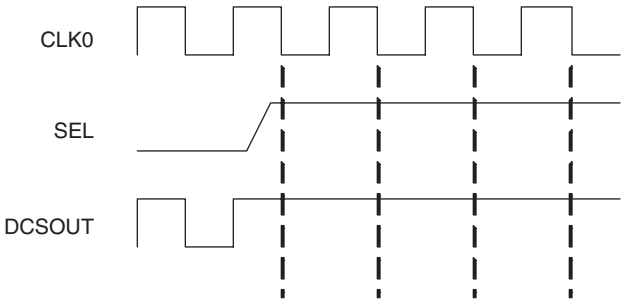
- Switch low @CLK0 falling edge.
- If SEL is high, output stays low at on CLK0 rising edge.

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 (
         CLK0         :IN    std_logic;
         CLK1         :IN    std_logic;
         SEL:IN    std_logic;
         DCSOUT     :OUT   std_logic);
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
         );
```

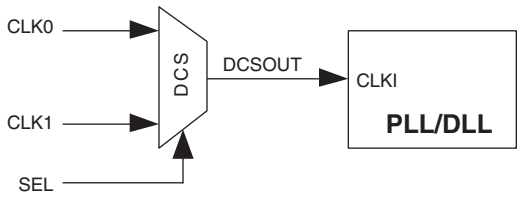
Dynamic Clock Switching at PLL/DLL Inputs

Input Reference Clock Switchover

A dynamically controlled 2:1 mux is included in the reference clock path to allow for dynamic switching of the reference clock. The intent of this feature is to allow the PLL/DLL to switch between two reference input clocks. This can be used in systems with clock redundancy and in dual clock domain systems where one of the clocks may stop running for some reason.

A user signal from the FPGA core is used to switch the reference clock from CLK0 to CLK1. A circuit example of the PLL reference clock switchover is shown in Figure 8-25. The two clocks from pads are inputs to a 2:1 mux that is controlled by the clock switchover circuit. The output of the mux drives into the reference clock input of the PLL/DLL block.

Figure 8-25. PLL Reference Clock Switchover Circuit Example



Oscillator (OSCD)

There is a dedicated oscillator in the LatticeECP2 devices whose output is made available for users. The oscillator frequency is programmable with a range is 2.5 to 130MHz. The output of the oscillator can also be routed as an input clock to the clock tree. 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 for use as an extra clock that does not require accurate clocking.

Primitive Name: OSCD

Table 8-13. OSCD Port Definition

I/O	Name	Description
Output	OSC	Oscillator Clock Output

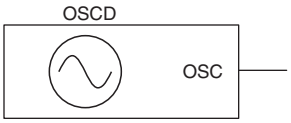
Table 8-14. OSCD Attribute Definition

User Attribute	Attribute Name	Value (MHz)	Default Value
Nominal Frequency	NOM_FREQ	2.5, 4.3, 5.4, 6.9, 8.1, 9.2, 10.0, 13, 15, 20, 26, 30, 34, 41, 45, 51, 55, 60, 130	2.5

Please refer to the LatticeECP2 Data Sheet for detailed specifications.

OSC Primitive Symbol (OSCD)

Figure 8-26. OCS Symbol



OSC Usage with VHDL - Example

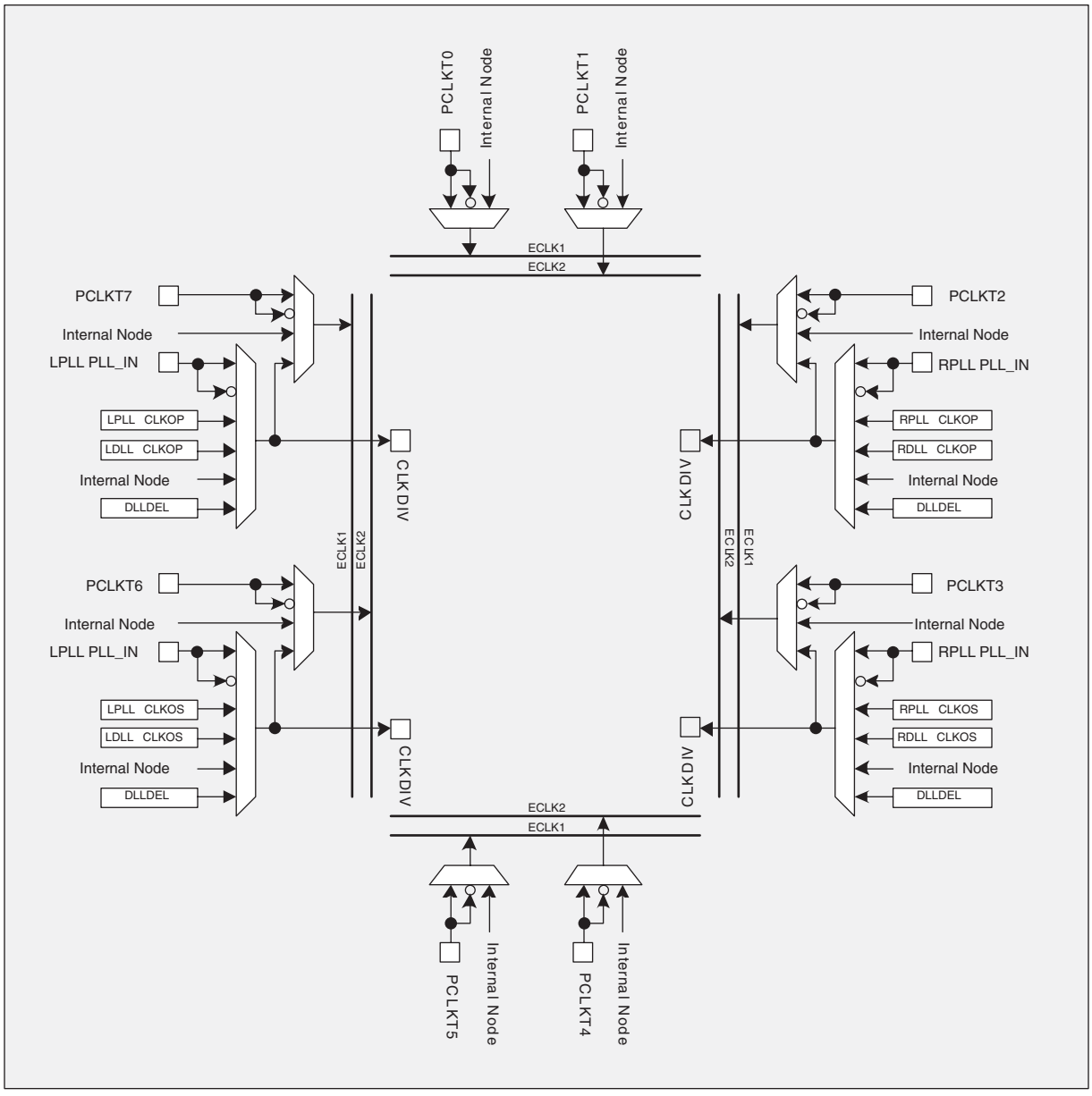
```
COMPONENT OSCD
-- synthesis translate_off
      GENERIC(NOM_FREQ: string := 2.5);
-- synthesis translate_on
PORT (OSC:OUT   std_logic);

END COMPONENT;

      attribute NOM_FREQ : string;
      attribute NOM_FREQ of OSCins0 : signal is “2.5”;
begin

OSCInst0: OSCD
-- synthesis translate_off
```

```
      GENERIC MAP (NOM_FREQ => "2.5")
-- synthesis translate_on
      PORT MAP ( OSC=>  osc_int);
```

PLL, DLL, CLKIDV and ECLK Locations and Connectivity

Figure 8-27 shows the locations, site names, and connectivity of the PLLs, DLLs, CLKDIVs and ECLKs

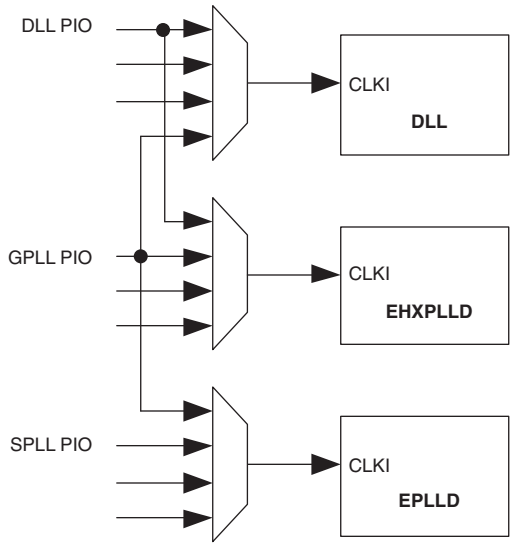
Figure 8-27. PLL, DLL, CLKIDV and ECLK Locations and Connectivity



Input Clock Sharing

The reference clock from the pads can be shared in LatticeECP2 PLLS and DLLS as shown in Figure 28. This feature is useful when only one clock source is available for multiple PLLs/DLLs.

Figure 8-28. Input Clock Sharing



Setting Clock Preferences

Clock preferences allow designers to implement clocks to the desired performance. Preferences can be set in the Pre-Map Preference Editor in ispNavigator or in preference files. For additional information see the ispLEVER online Help system.

Power Supplies

Each PLL has its own power supply pin, VCCPLL or GNDPLL. Since VCC and VCCPLL are normally the same 1.2V, 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 1.2V supply on the board using proper board filtering techniques to minimize the noise coupling between them.

The DLL is powered from the FPGA core power supply.

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

Appendix A. Serial Memory Interface (SMI)

The DLL can connect to the system bus via the Serial Memory Interface (SMI). The SMI allows a subset of the DLL features to be modified during run time. The DLL must be created in Time Reference Delay mode to use this capability. The SMI interface is a broadcast protocol that is driven from the system bus or from FPGA logic. The SMI bus is connected via FPGA routing to all of the SMI targets that are instantiated in the FPGA design. Each target is given an offset address. When a user accesses the system bus at this specific offset address the SMI bus will transact with the given target. It is also possible to given several SMI targets the same offset address to broadcast data to all of them.

The DLL each have a memory map that describes registers accessible from the SMI. The DLL are first configured via the FPGA bitstream. Once the configuration is loaded into the FPGA the registers available on the SMI bus are loaded with the configuration set in the bitstream. A subset of these settings can now be changed at run time. For example, the phase offset can be changed as well as the output dividers.

The SMI capabilities of the DLL are not supported in simulation. SMI transactions can be simulated, but the DLL will not respond to any changes during simulation.

The user must set the SMI offset address for the DLL. Below are examples for setting the DLL target address to 0x410 in the preference file.

```
ASIC “dll/dll_0_0” TYPE “TRDLLA” SMI_OFFSET=0x410;
```

When making modifications to the DLL a reset must be issued following the write transaction. The reset must be performed via the RSTN port or GSR. Be sure to include a RSTN port on the DLL when creating the module if the SMI interface will be utilized.

Table 8-15 provides the memory map for the DLL. -This memory map begins at the base address specified by the SMI_OFFSET value.

Table 8-15. SMI Map

Register	Bits	Description
0x0	[7:5]	CLKOP Phase Shift 000: 90 degree 001: 180 degree 010: 270 degree 011: 360 degree 100: 0 degree Other combinations reserved
	[4:0]	Reserved
0x1	[7:5]	CLKOP Phase Shift 000: 90 degree 001: 180 degree 010: 270 degree 011: 360 degree 100: 0 degree Other combinations reserved
	[4:0]	Reserved
0x2	[7:6]	CLKOS Divider 00: no division 01: divide by 2 10: divide by 4 11: 50% duty cycle correction
	[5:4]	Reserved
	[3:2]	CLKOS Fine Phase Shift if 0x3 bit 5 is set 00: 0 degree 01: 45 degree 10: 22.5 degree 11: 11.25 degree
	[1:0]	Reserved
0x3	[7]	Hold
	[6]	Reserved
	[5]	CLKOS Fine Phase Shift Enable 0: Disabled 1: Enabled (0x2[3:2])
	[4:0]	Reserved

LatticeECP2 Memory Usage Guide

February 2006 Technical Note TN1104

Introduction

This technical note discusses memory usage for the LatticeECP2™ device family. It is intended to be used by design engineers as a guide for integrating the 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.

The capabilities of the 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.

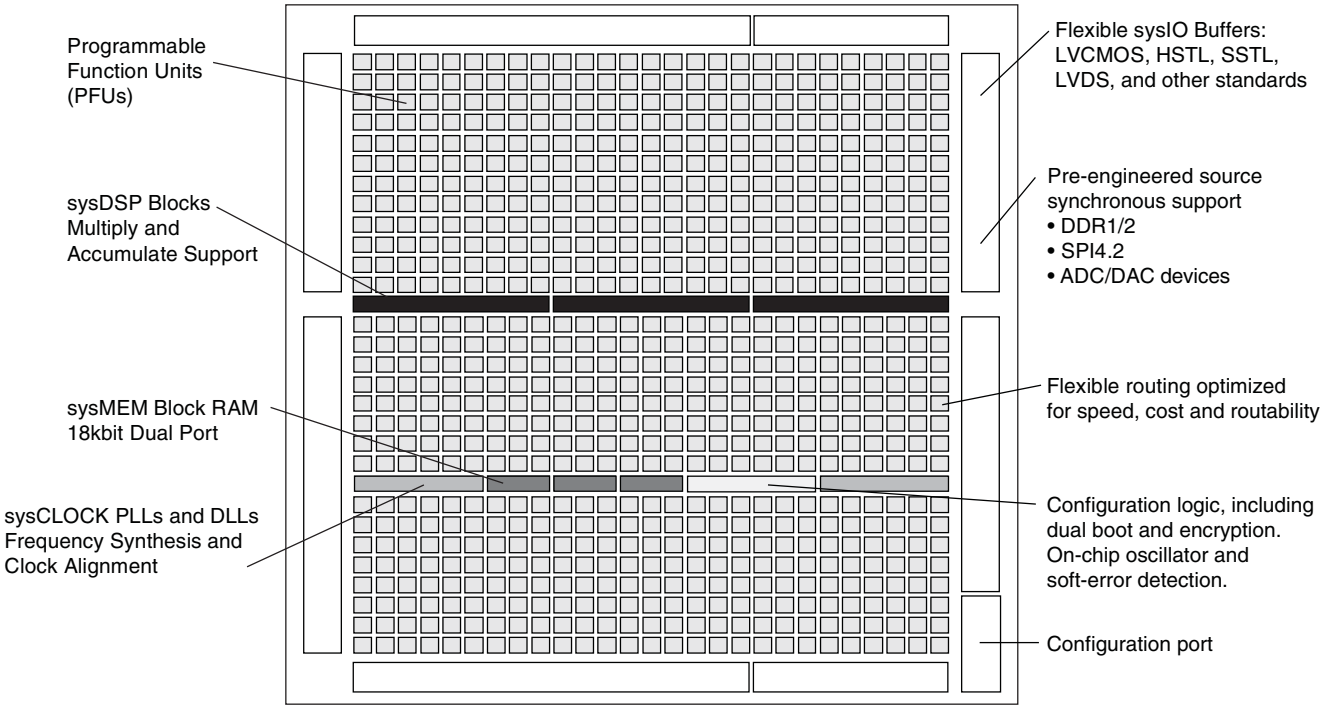
The remainder of this document discusses the use of IPexpress, memory modules and memory primitives.

Memories in LatticeECP2 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 LatticeECP2 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. In addition, LatticeECP2 devices contain up to two rows of sysDSP™ blocks. Each sysDSP block has multipliers and accumulators, which are the building blocks for complex signal processing capabilities

Figure 9-1. Simplified Block Diagram, LatticeECP2-6 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:

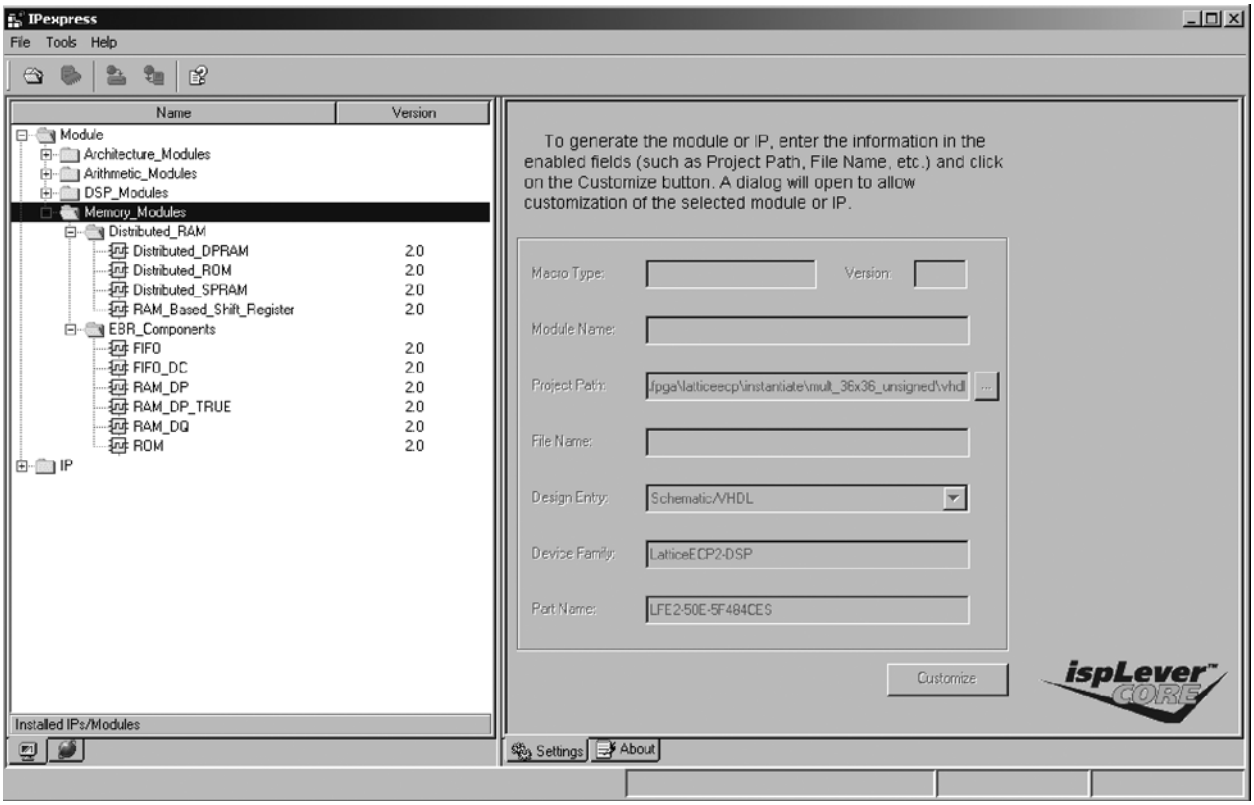
- Single Port RAM (RAM_DQ) – EBR-based
- Dual PORT RAM (RAM_DP_TRUE) – EBR-based
- Pseudo Dual Port RAM (RAM_DP) – EBR-based
- Read Only Memory (ROM) – EBR-Based
- First In First Out Memory (Dual Clock) (FIFO_DC) – EBR-based
- Distributed Single Port RAM (Distributed_SPRAM) – PFU-based
- Distributed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

IPexpress Flow

For generating any of these memories, create (or open) a project for the LatticeECP2 devices.

From the Project Navigator, select **Tools > IPexpress** or click on the button in the toolbar when LatticeECP2 devices are targeted in the project. This opens the IPexpress main window as shown in Figure 9-2.

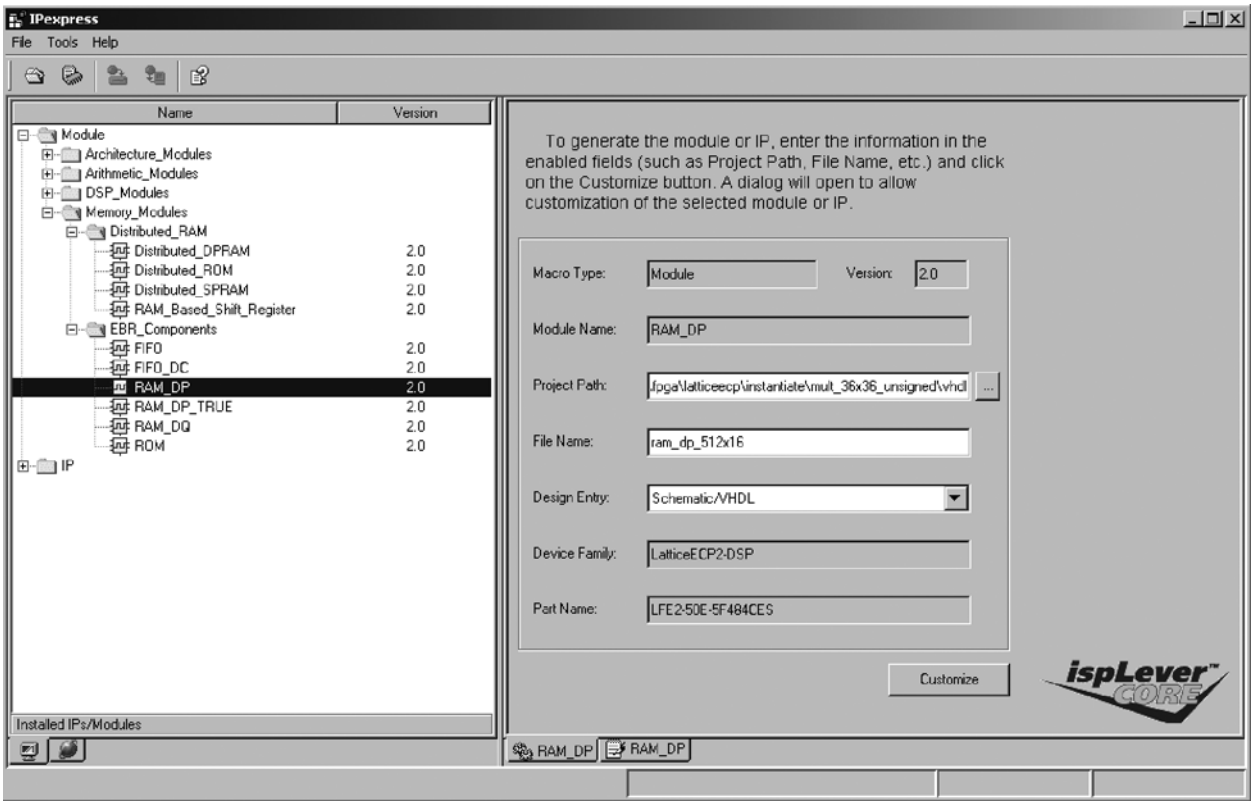
Figure 9-2. IPExpress - Main Window



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 9-2.

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 9-3.

Figure 9-3. 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.

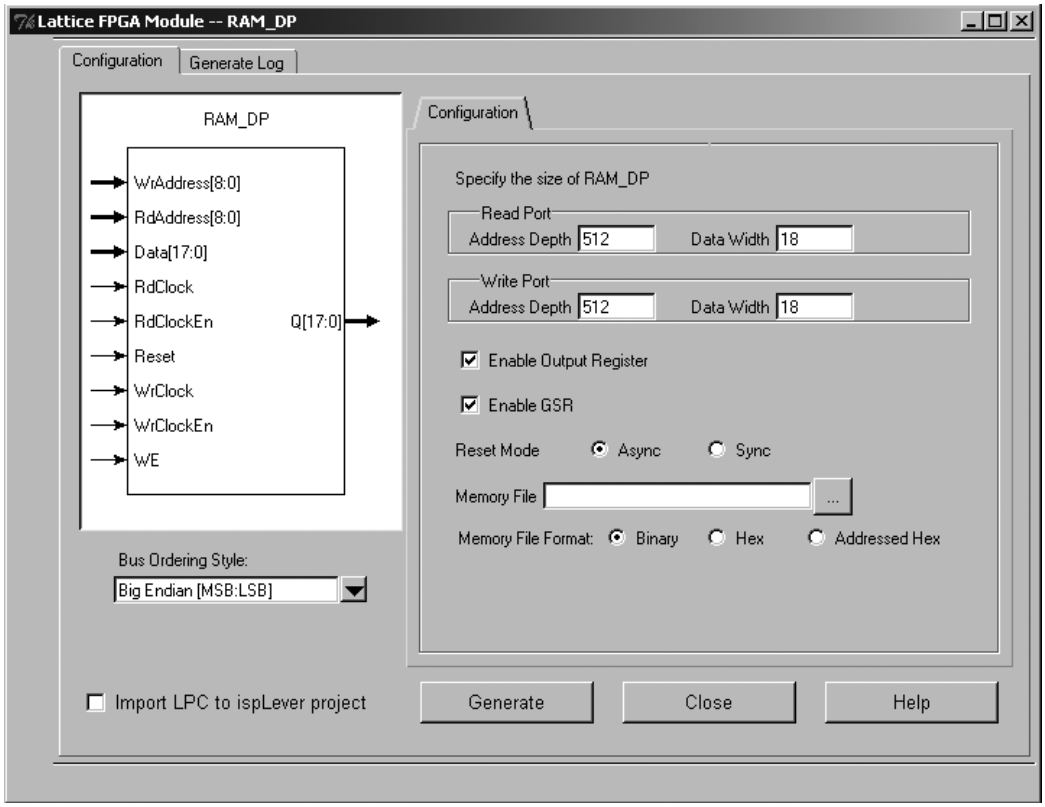
Users can change the directory where the generated module files will be placed by clicking the **Browse** button in the **Project Path**.

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, LatticeECP2 in this example. By clicking the **Customize** button, another window opens where users can customize the RAM (Figure 9-4).

Figure 9-4. 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.

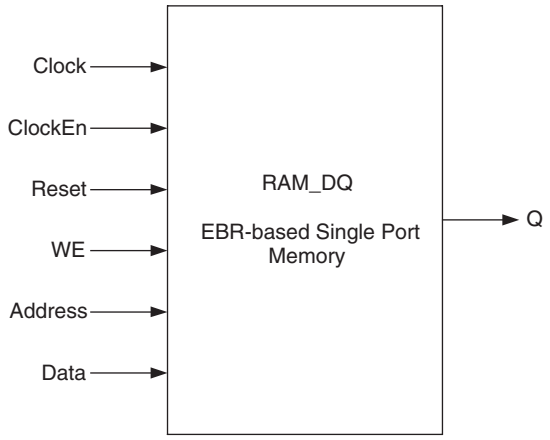
Memory Modules

Single Port RAM (RAM_DQ) – EBR Based

The EBR blocks in LatticeECP2 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 9-5.

Figure 9-5. 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 9-1. The table lists the corresponding ports for the module generated by IPexpress and for the EBR RAM_DQ primitive.

Table 9-1. 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 16,384 bits of RAM. The values for x (address) and y (data) for each EBR block for the devices are listed in Table 9-2.

Table 9-2. Single Port Memory Sizes for 16K Memories for LatticeECP2

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 9-3 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 9-3. Single Port RAM Attributes for LatticeECP2

Attribute	Description	Values	Default Value	User Selectable Through IPexpress
Address depth	Address Depth Read Port	16K, 8K, 4K, 2K, 1K		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 9-6-9-11 show the internal timing waveforms for the Single Port RAM (RAM_DQ) with these options.

Figure 9-6. Single Port RAM Timing Waveform - NORMAL Mode, without Output Registers

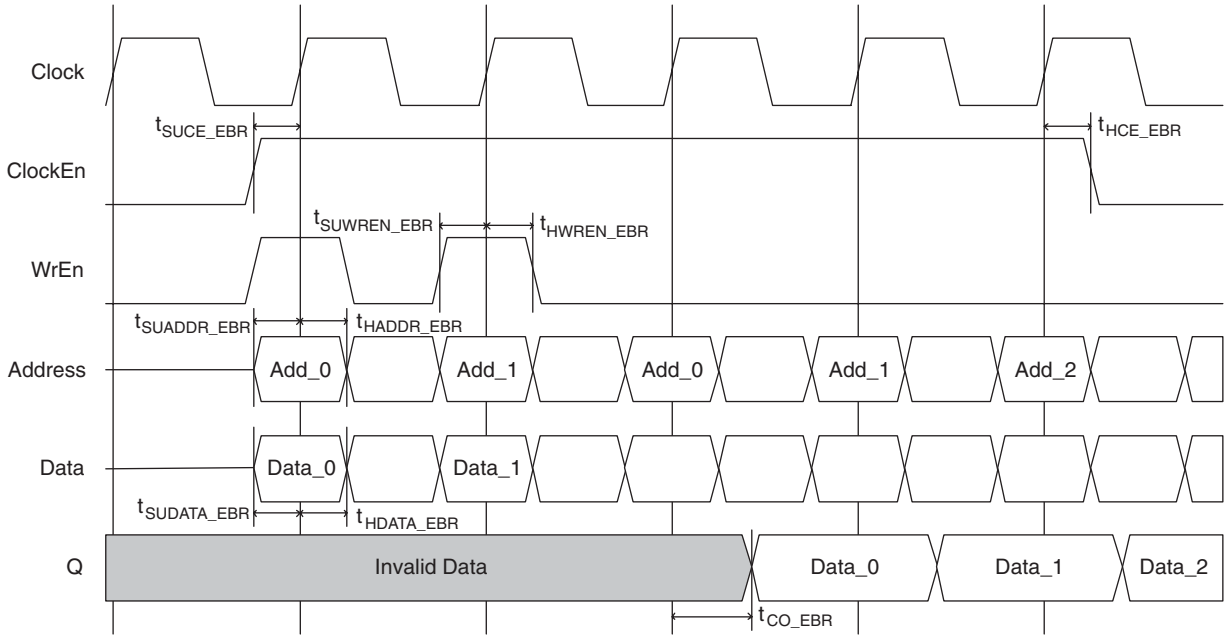


Figure 9-7. Single Port RAM Timing Waveform - NORMAL Mode, with Output Registers

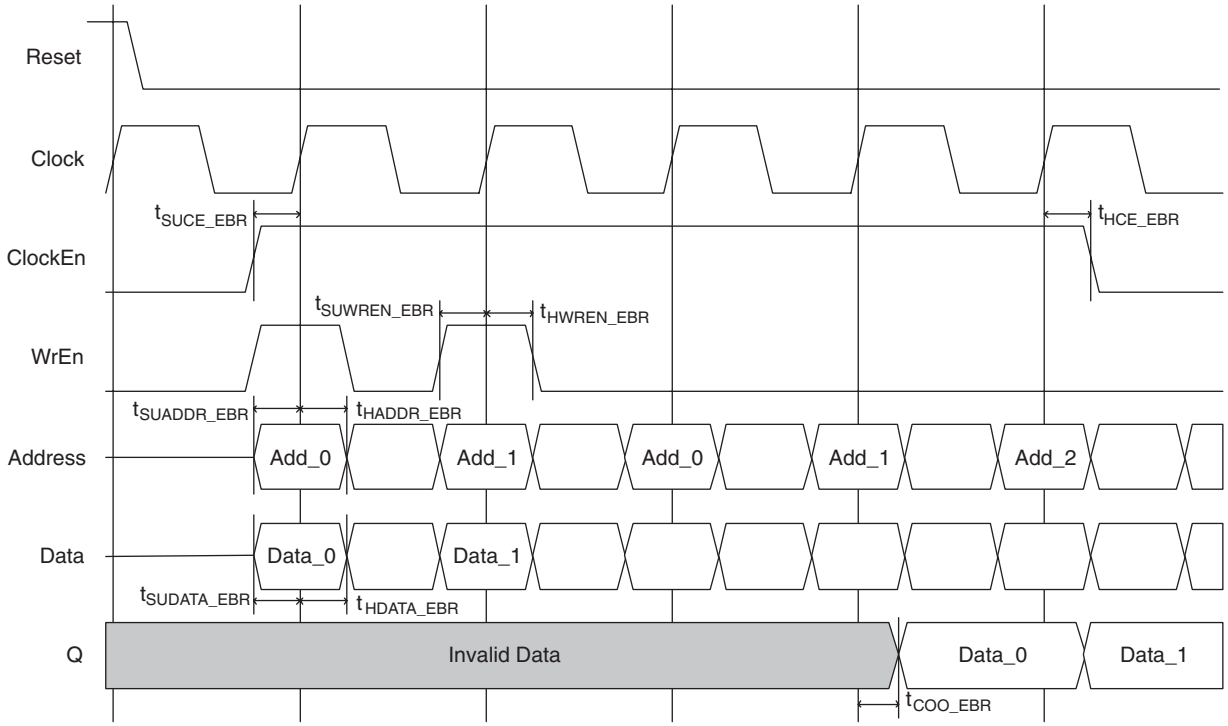


Figure 9-8. Single Port RAM Timing Waveform - READ BEFORE WRITE Mode, without Output Registers

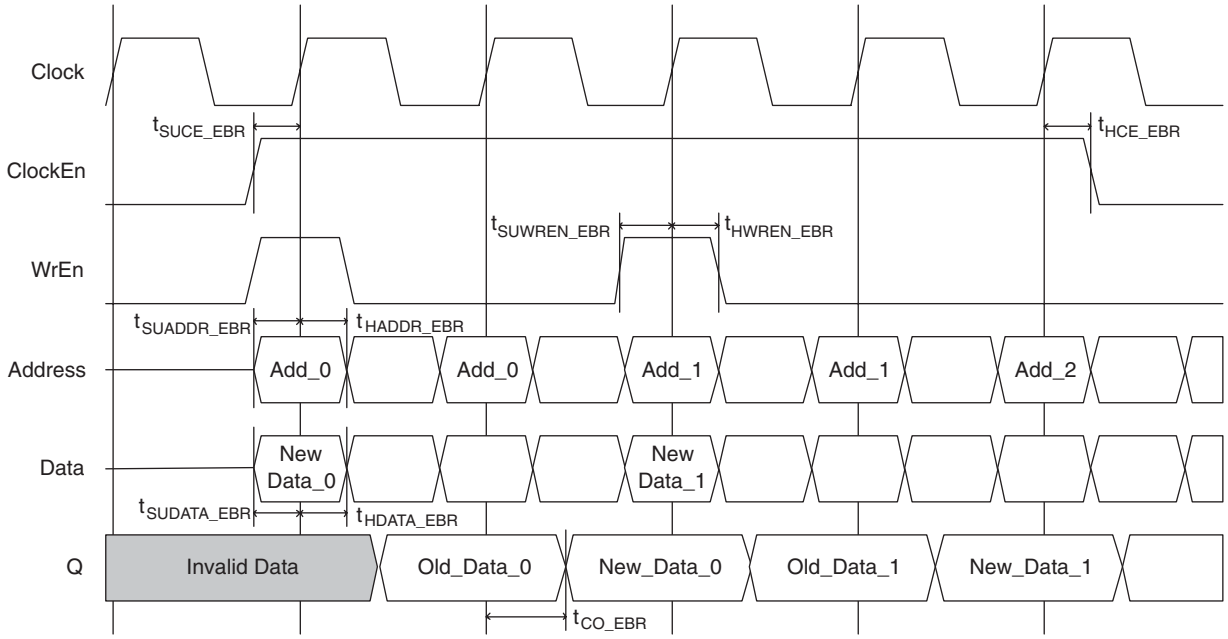


Figure 9-9. Single Port RAM Timing Waveform - READ BEFORE WRITE Mode, with Output Registers

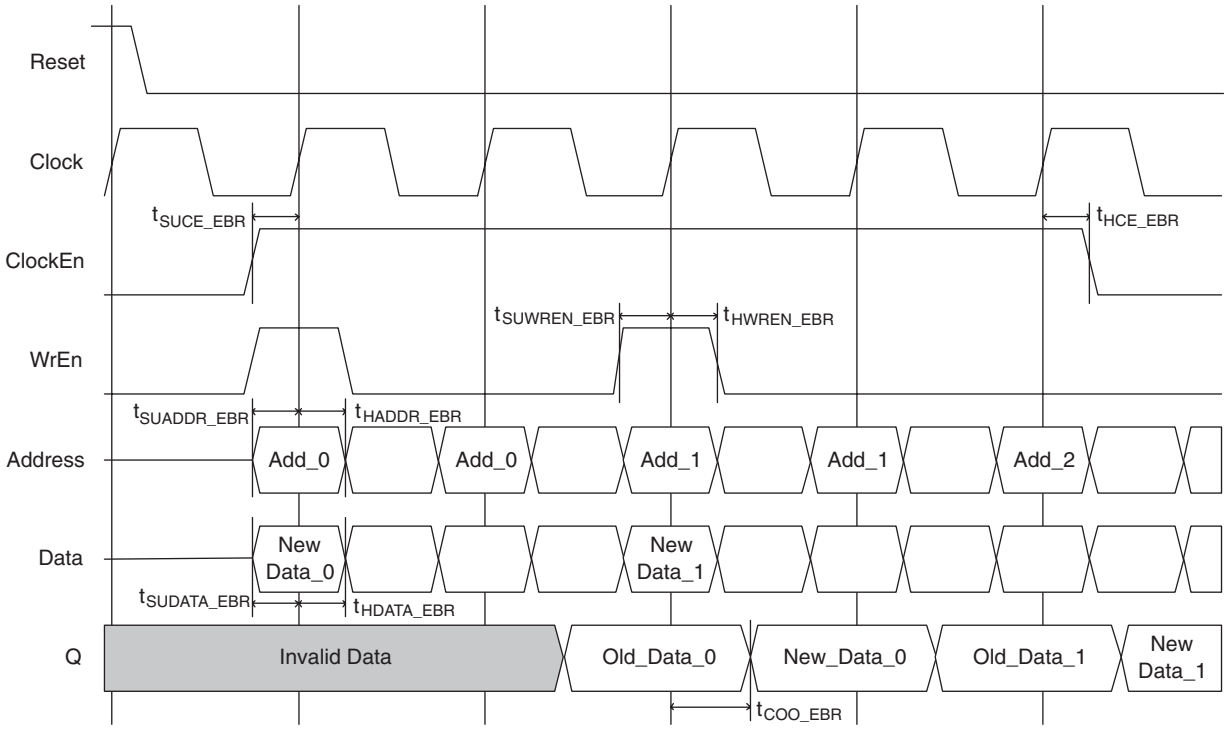


Figure 9-10. Single Port RAM Timing Waveform - WRITE THROUGH Mode, without Output Registers

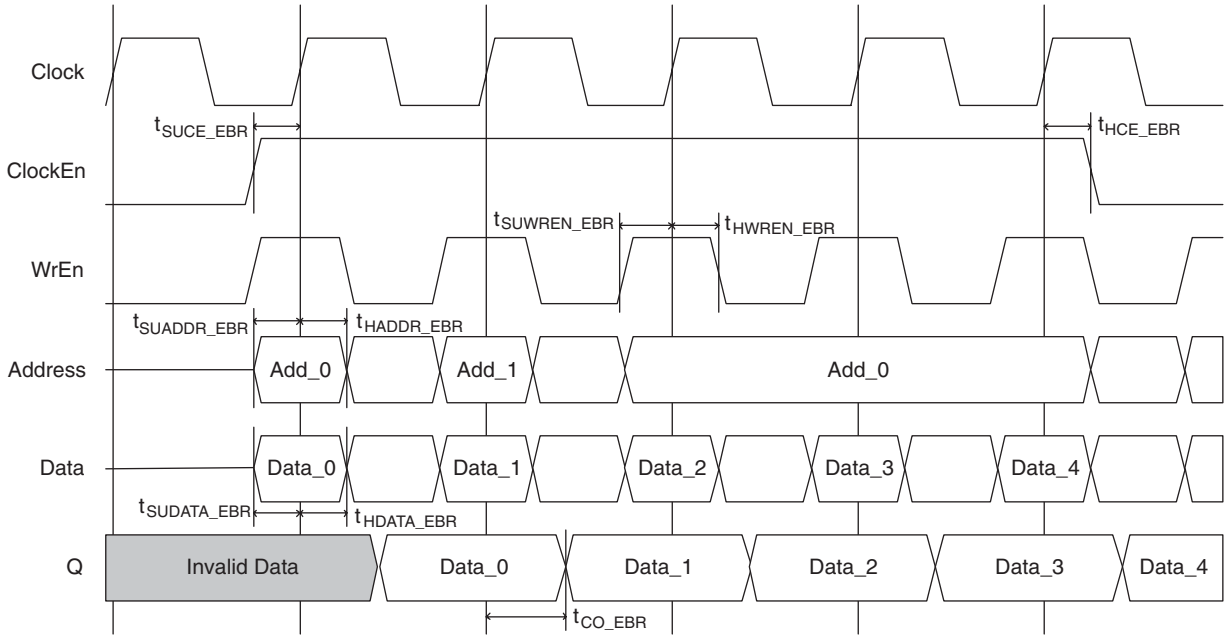
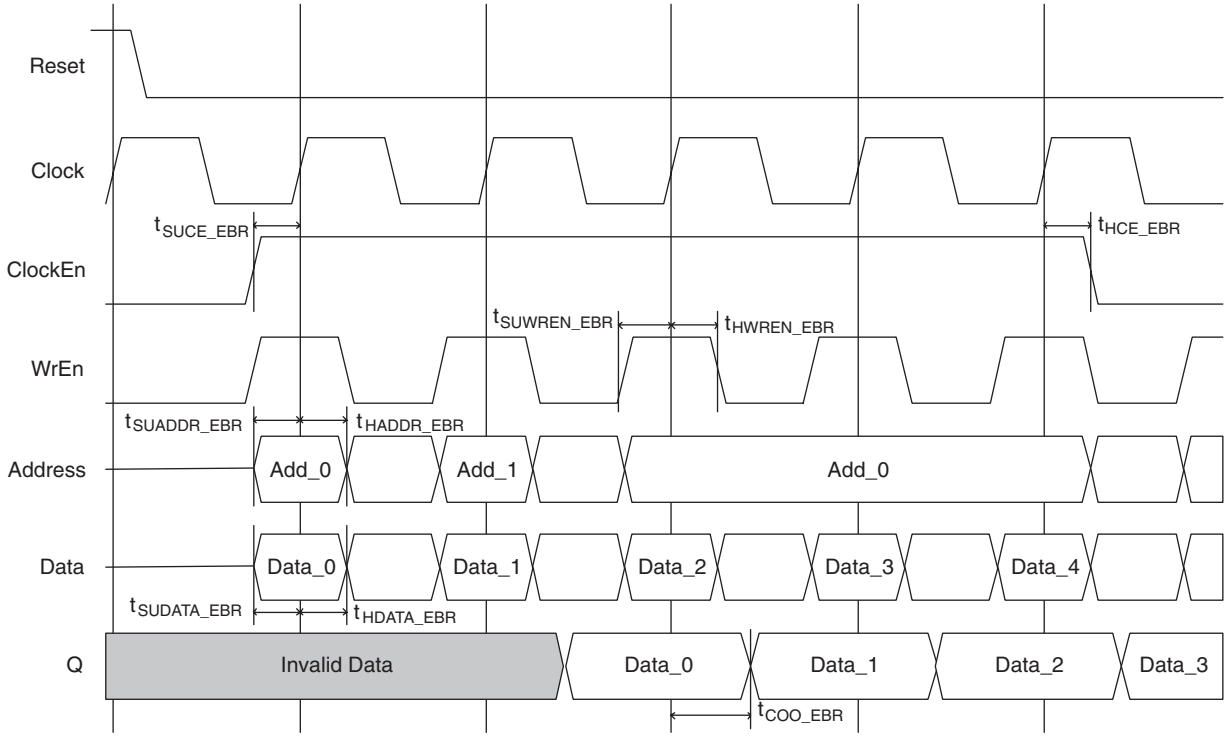


Figure 9-11. Single Port RAM Timing Waveform - WRITE THROUGH Mode, with Output Registers

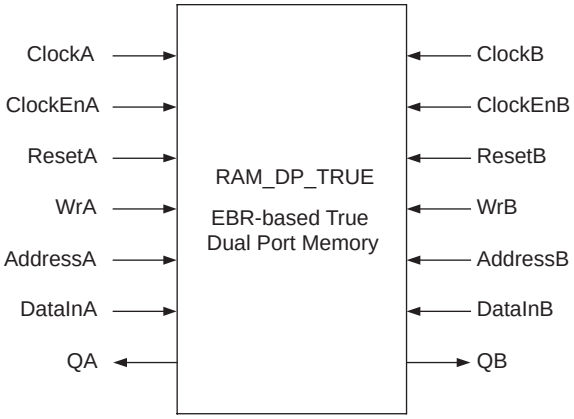


True Dual Port RAM (RAM_DP_TRUE) – EBR Based

The EBR blocks in the LatticeECP2 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 9-12.

Figure 9-12. 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 9-4. The table lists the corresponding ports for the module generated by IPexpress and for the EBR RAM_DP_TRUE primitive.

Table 9-4. 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 16,384 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 9-5.

Table 9-5. True Dual Port Memory Sizes for 16K Memory for LatticeECP2

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 9-6 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 9-6. True Dual Port RAM Attributes for LatticeECP2

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	0x0000000000000000000 000000000000000000000 000000000000000000000 0000000000000000......0xF FFFFFFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFFFFF	0x000000000 00000000000 00000000000 00000000000 00000000000 00000000000 00000000000 00000	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.

Additionally, users can select to enable the output registers for RAM_DP_TRUE. Figures 9-13 through 9-18 show the internal timing waveforms for the True Dual Port RAM (RAM_DP_TRUE) with these options.

Figure 9-13. True Dual Port RAM Timing Waveform - NORMAL Mode, without Output Registers

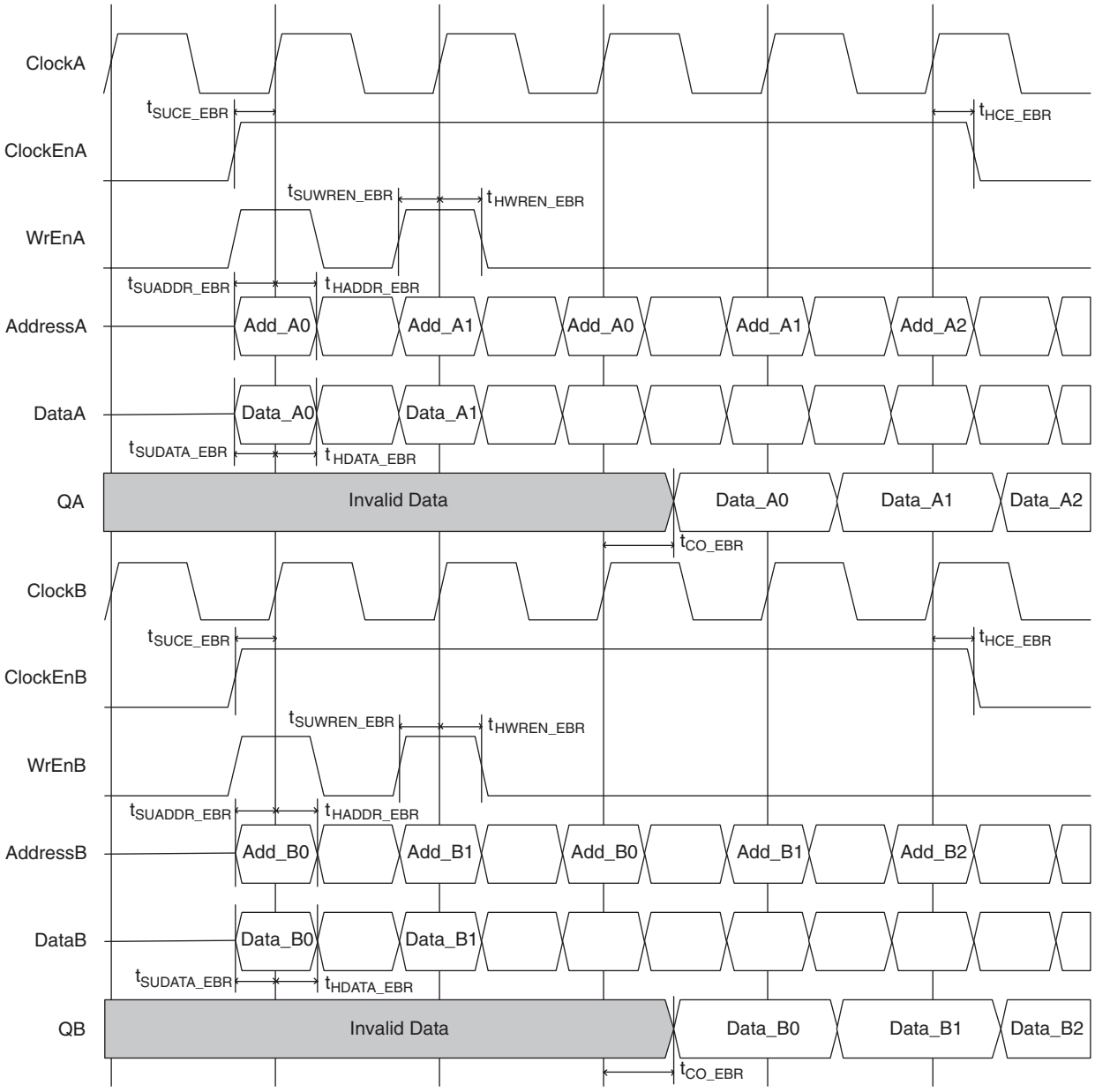


Figure 9-14. True Dual Port RAM Timing Waveform - NORMAL Mode with Output Registers

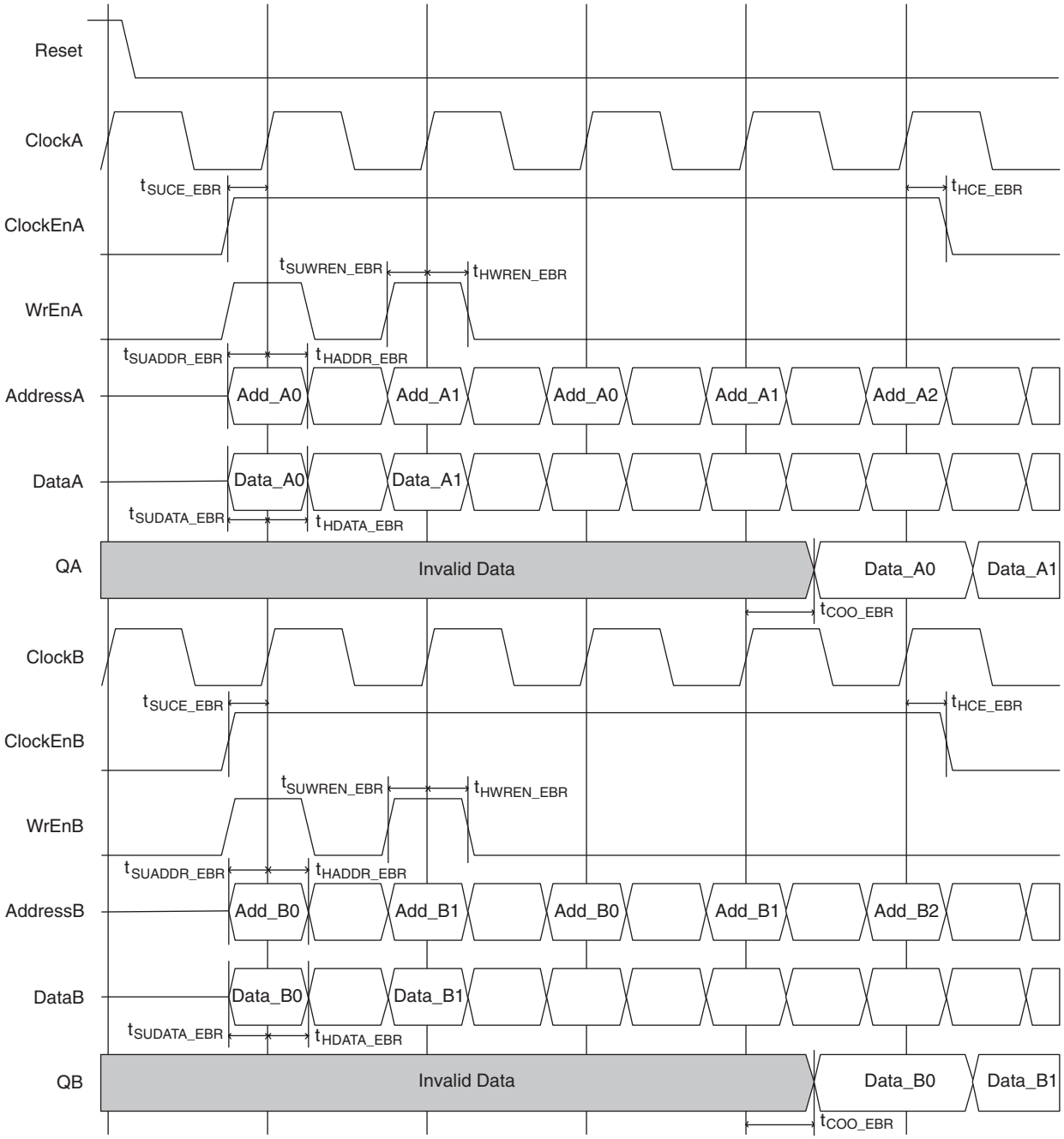


Figure 9-15. True Dual Port RAM Timing Waveform - READ BEFORE WRITE Mode, without Output Registers

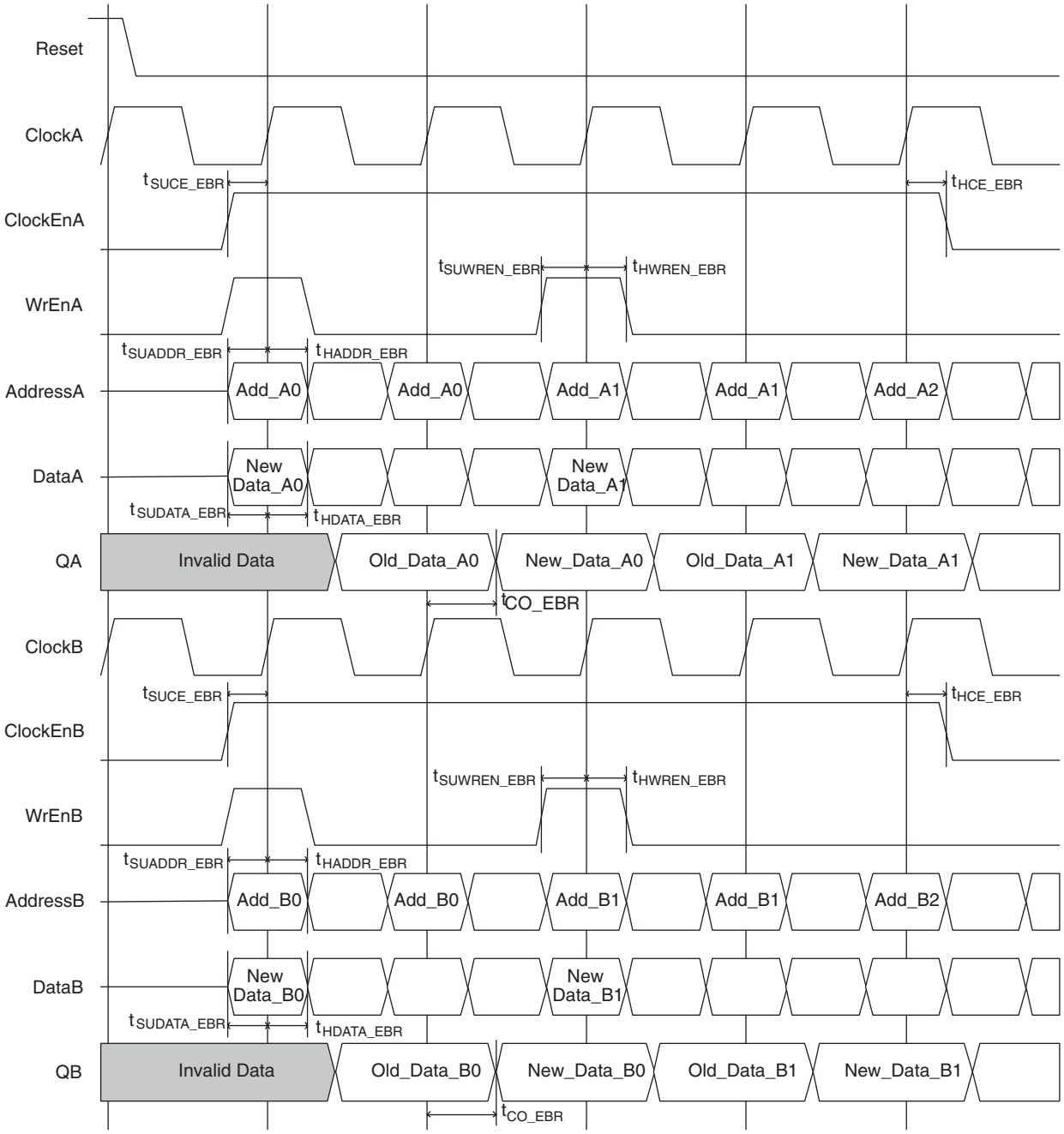


Figure 9-16. True Dual Port RAM Timing Waveform - READ BEFORE WRITE Mode, with Output Registers

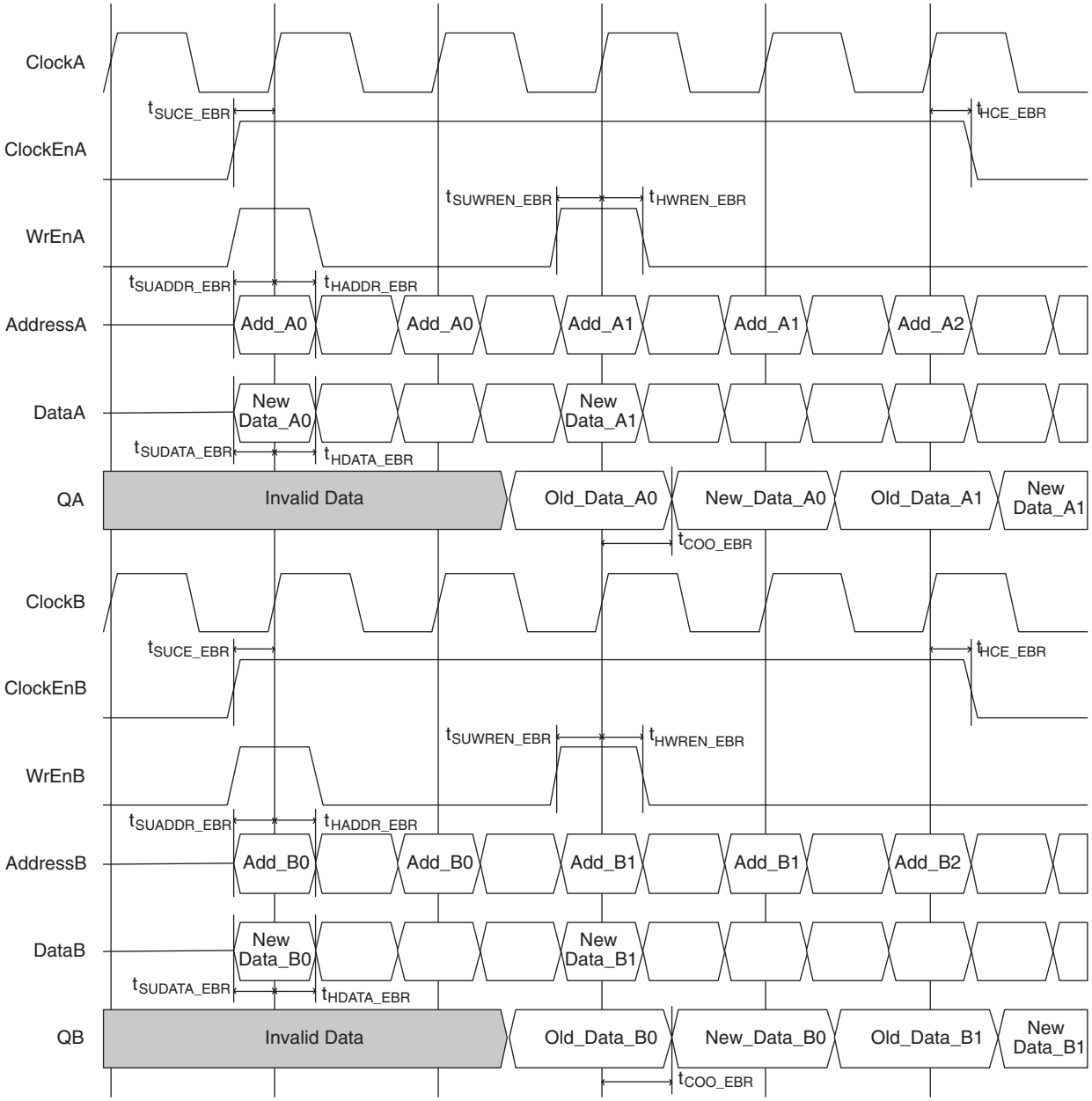


Figure 9-17. True Dual Port RAM Timing Waveform - WRITE THROUGH Mode, without Output Registers

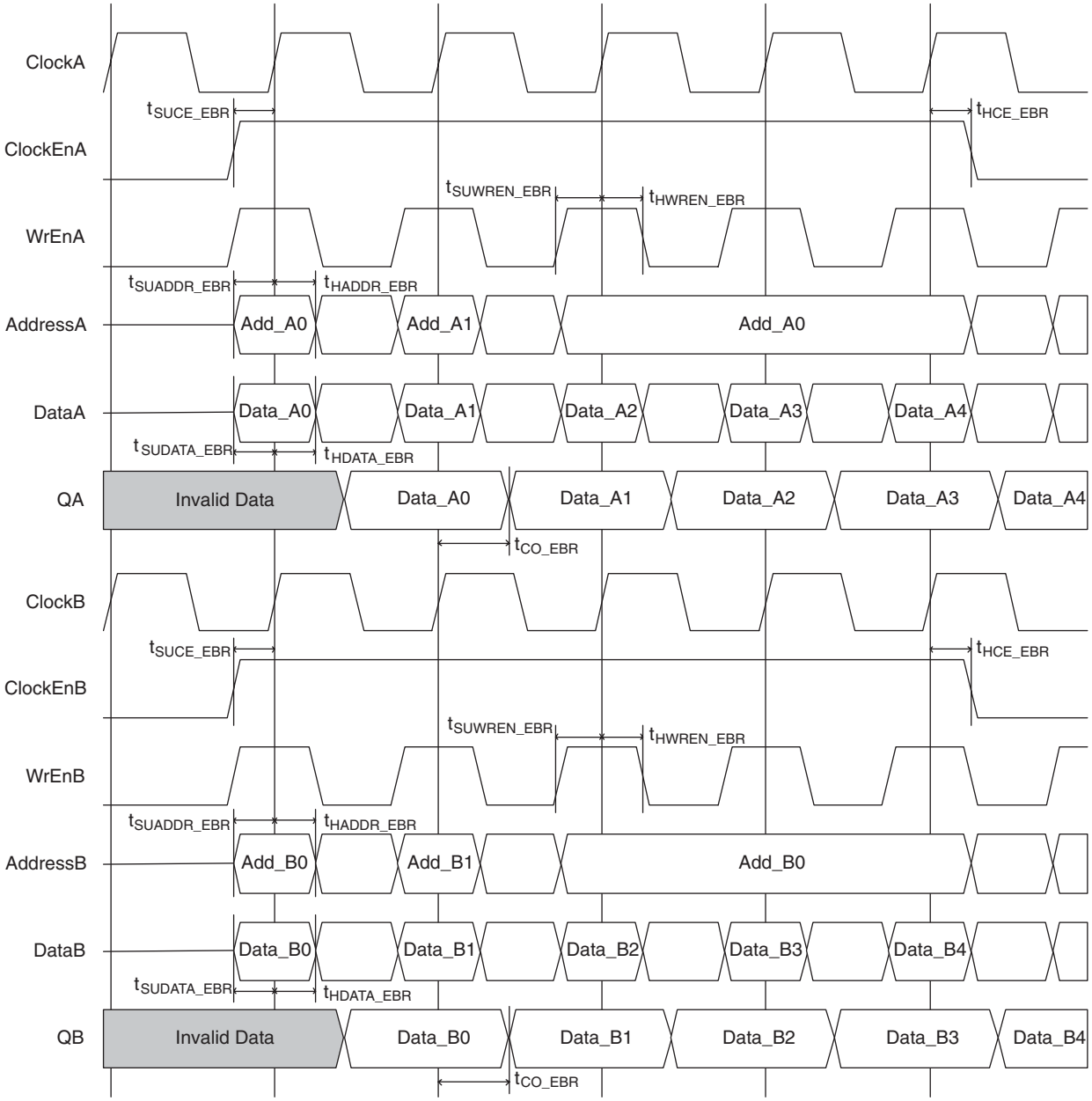
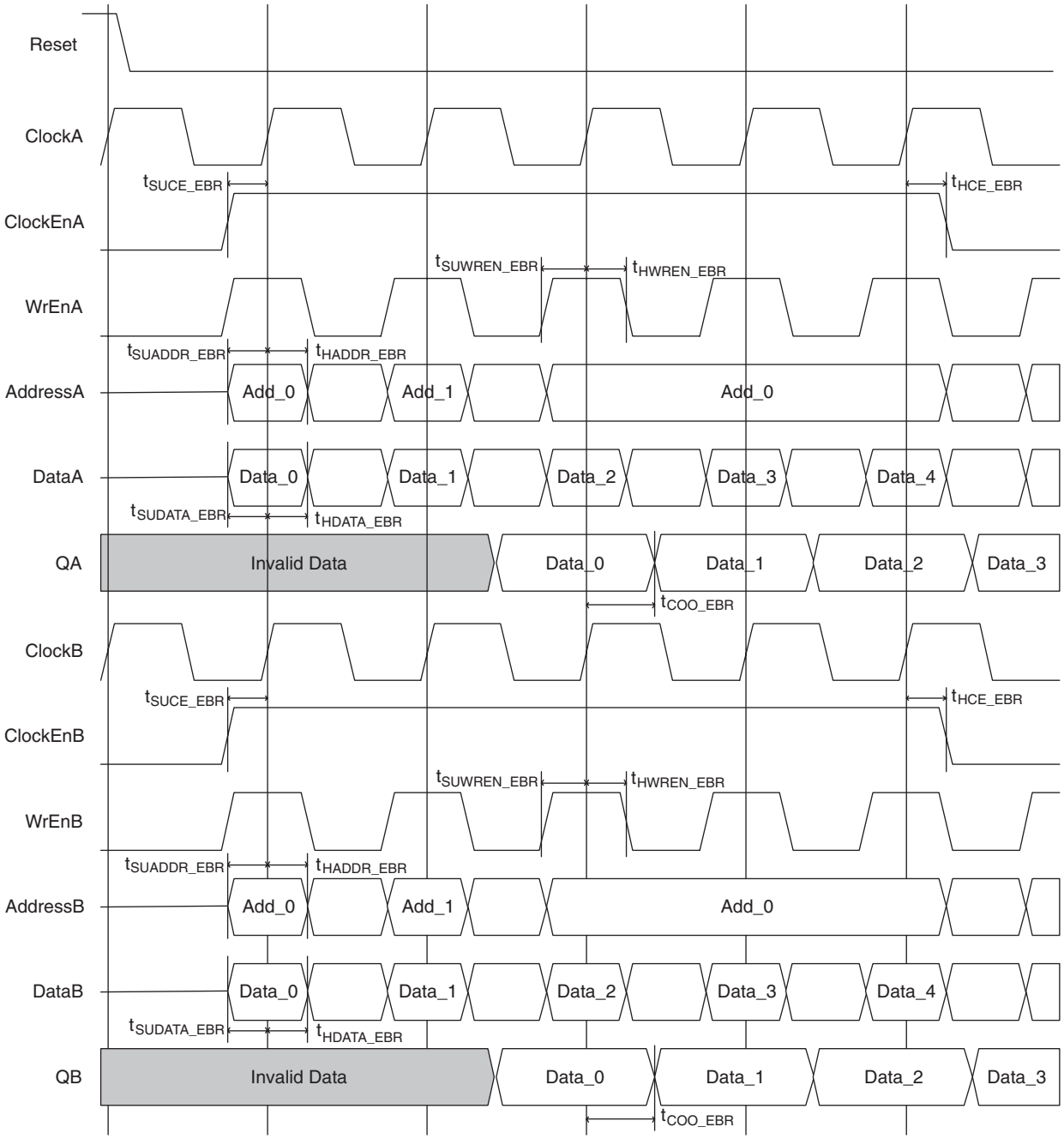


Figure 9-18. True Dual Port RAM Timing Waveform - WRITE THROUGH Mode, with Output Registers

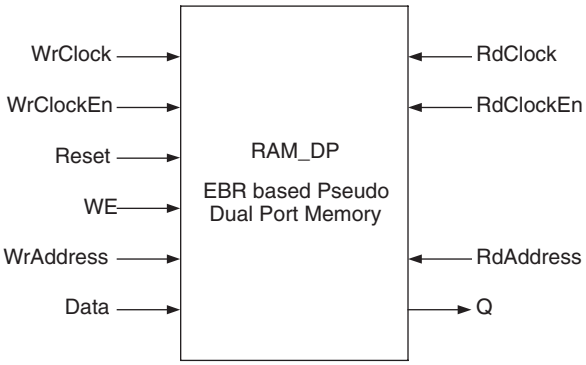


Pseudo Dual Port RAM (RAM_DP) – EBR Based

The EBR blocks in LatticeECP2 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 9-19.

Figure 9-19. 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 9-7. The table lists the corresponding ports for the module generated by IPexpress and for the EBR RAM_DP primitive.

Table 9-7. 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 16,384 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 9-8.

Table 9-8. Pseudo-Dual Port Memory Sizes for 16K Memory for LatticeECP2

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 9-9 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 9-9. Pseudo-Dual Port RAM Attributes for LatticeECP2

Attribute	Description	Values	Default Value	User Selectable Through IPexpress
Read Port Address Depth	Address Depth Read Port	16K, 8K, 4K, 2K, 1K		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, WRITETHROUGH, READBEFOREWRITE	NORMAL	YES
Write Port Write Mode	Read / Write Mode for Write Port	NORMAL, WRITETHROUGH, READBEFOREWRITE	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 00000000000000000000000000 00000000000000000000000000 0......0xFFFFFFFFFFFFFFFFFF 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 9-20 and 9-21 show the internal timing waveforms for Pseudo-Dual Port RAM (RAM_DP) with these options.

Figure 9-20. PSEUDO DUAL PORT RAM Timing Diagram - without Output Registers

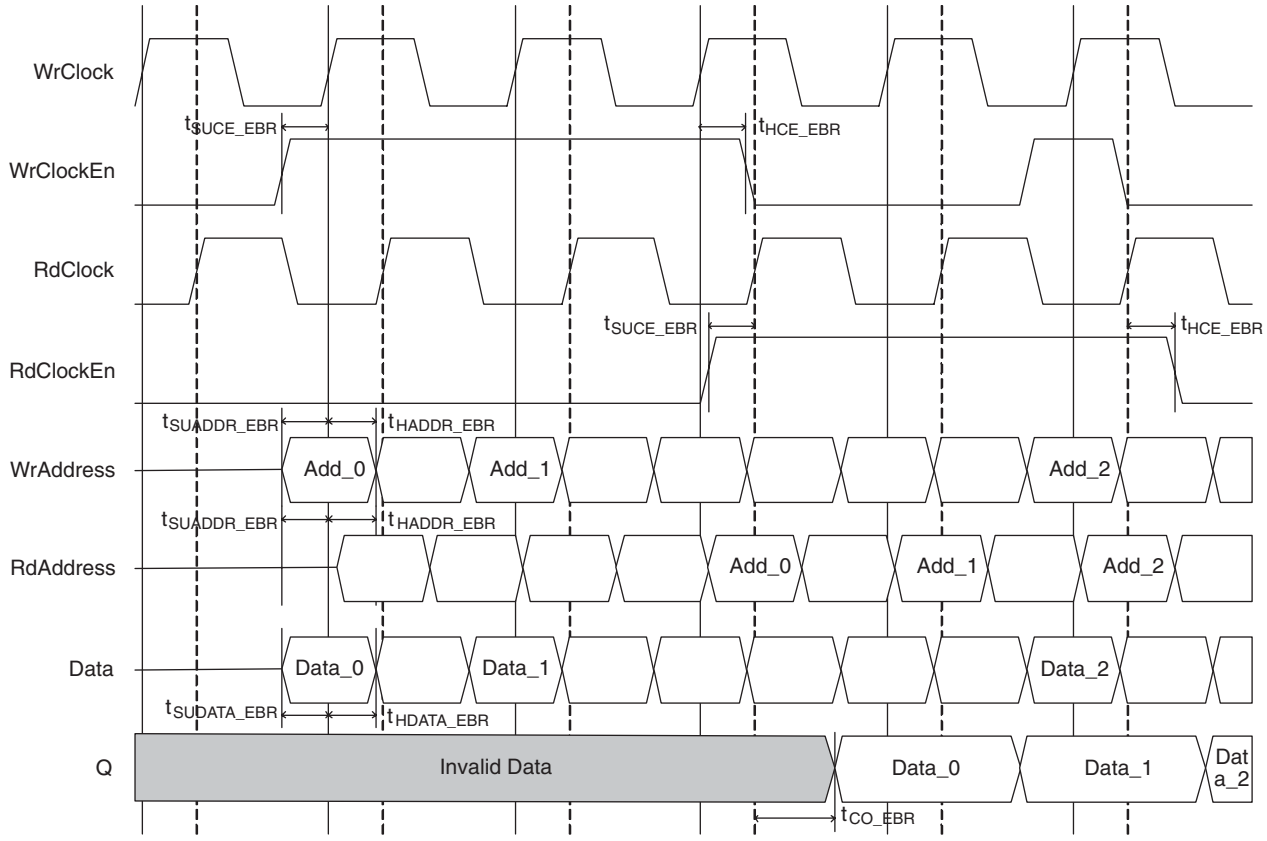
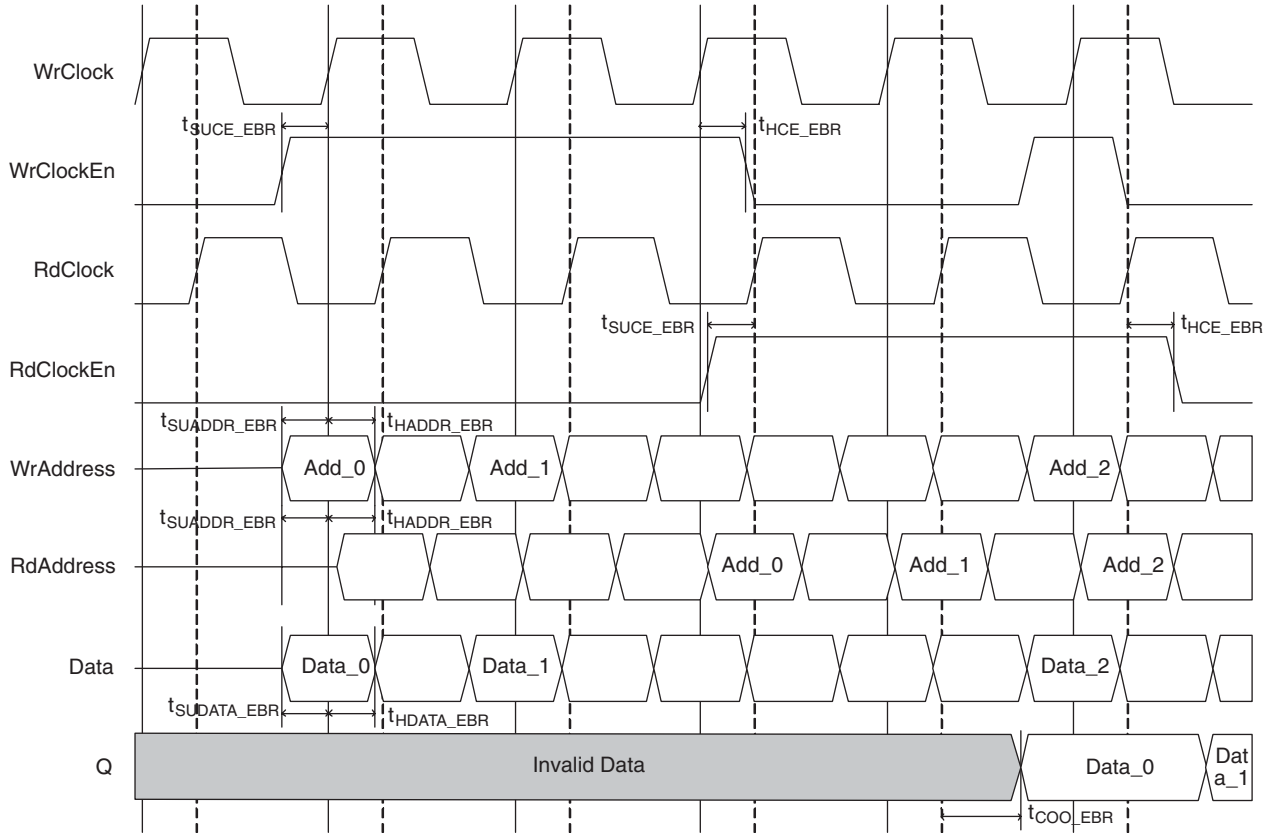


Figure 9-21. PSEUDO DUAL PORT RAM Timing Diagram - with Output Registers

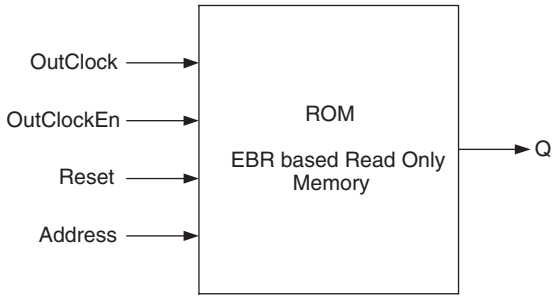


Read Only Memory (ROM) - EBR Based

The EBR blocks in the LatticeECP2 devices can be configured as Read Only Memory or ROM. IPexpress allows users to generate the Verilog-HDL or VHDL along 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 9-22.

Figure 9-22. 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 9-10. The table lists the corresponding ports for the module generated by IPexpress and for the ROM primitive.

Table 9-10. 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 9-23 and 9-24 show the internal timing waveforms for the Read Only Memory (ROM) with these options.

Each EBR block consists of 16,384 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 9-11.

Table 9-11. ROM Memory Sizes for 16K Memory for LatticeECP2

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 9-12 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 9-12. Pseudo-Dual Port RAM Attributes for LatticeECP2

Attribute	Description	Values	Default Value	User Selectable Through IPexpress
Address depth	Address Depth Read Port	16K, 8K, 4K, 2K, 1K		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 9-23. ROM Timing Waveform - without Output Registers

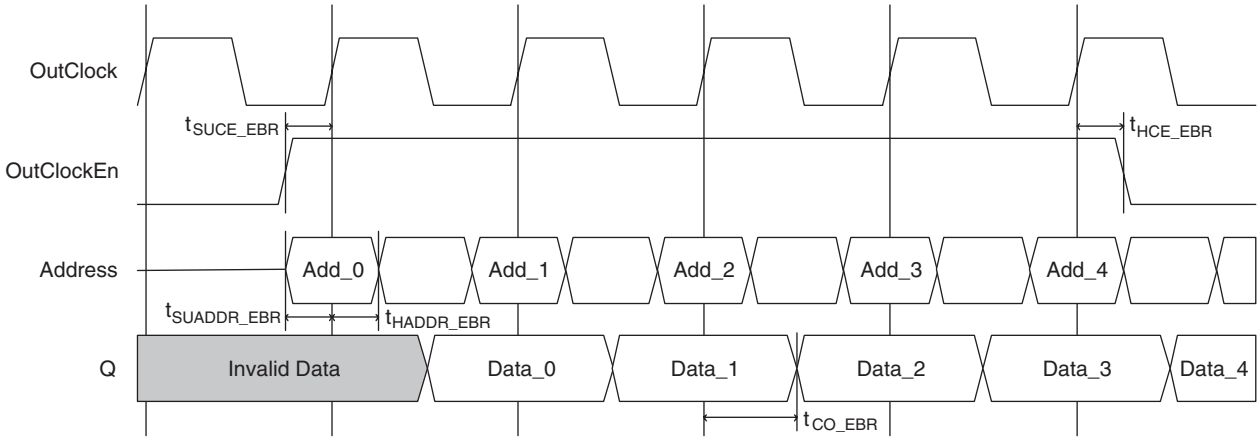
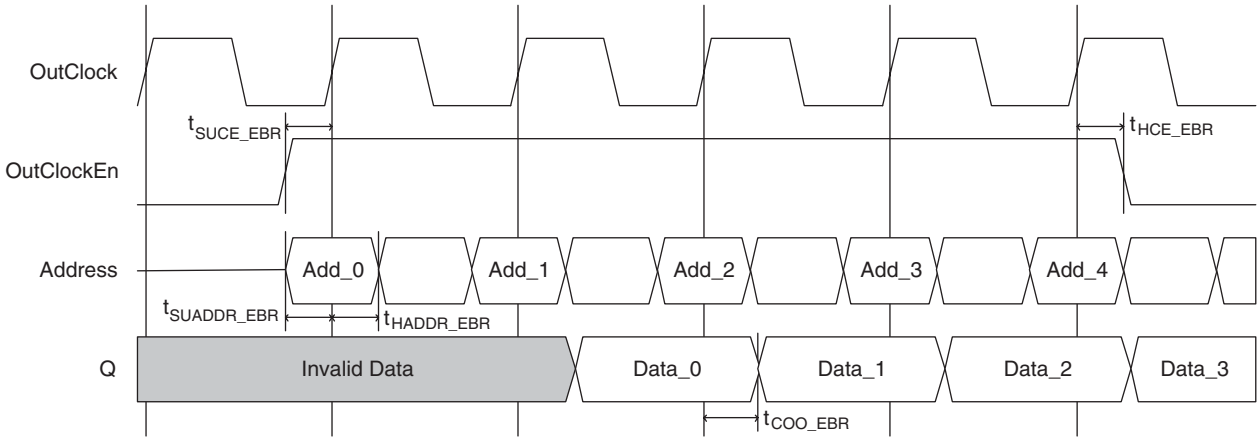


Figure 9-24. ROM Timing Waveform - with Output Registers

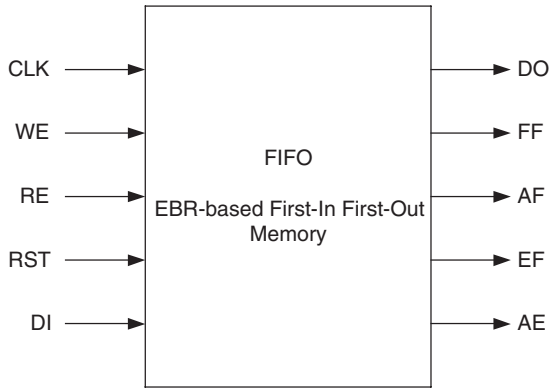


First In First Out (FIFO, FIFO_DC) – EBR Based

The EBR blocks in LatticeECP2 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 9-25.

Figure 9-25. 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 9-13.

Table 9-13. 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 LatticeECP2 are as per Table 9-14.

Table 9-14. FIFO_DC Data Widths Sizes for LatticeECP2

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 9-15 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 9-15. FIFO_DC Attributes for LatticeECP2

Attribute	Description	Values	Default Value	User Selectable Through IPexpress
Write Port Depth	Number of Address locations	16K, 8K, 4K, 2K, 1K		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		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 9-16.

Table 9-16. 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 9-26. FIFO_DC without Output Registers (Non-Pipelined)

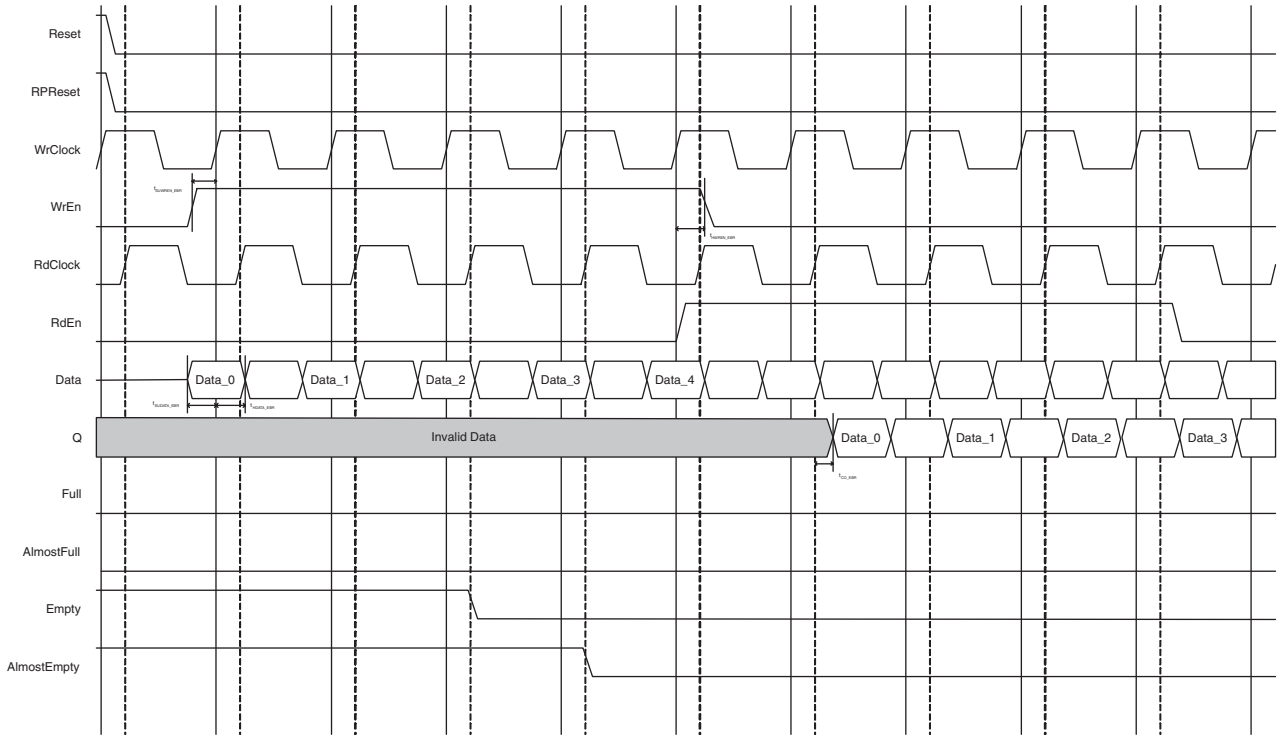
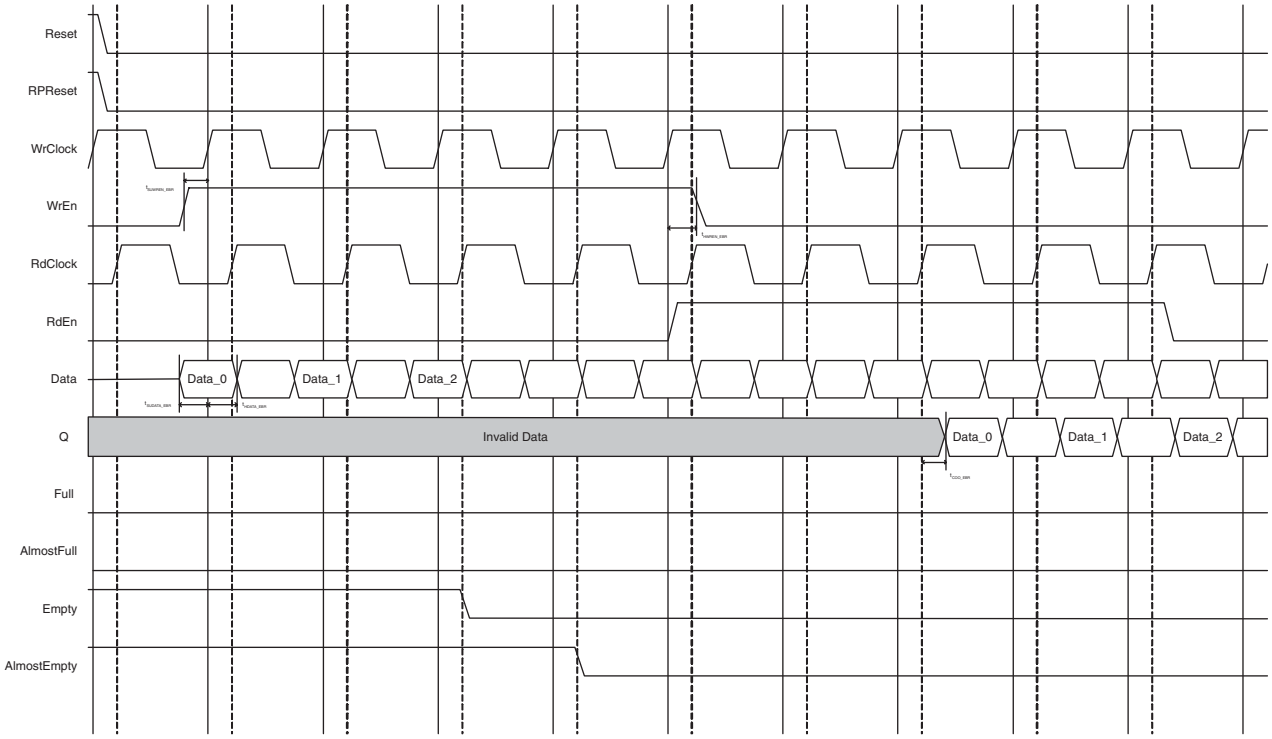


Figure 9-27. FIFO_CD with Output Registers (Pipelined)

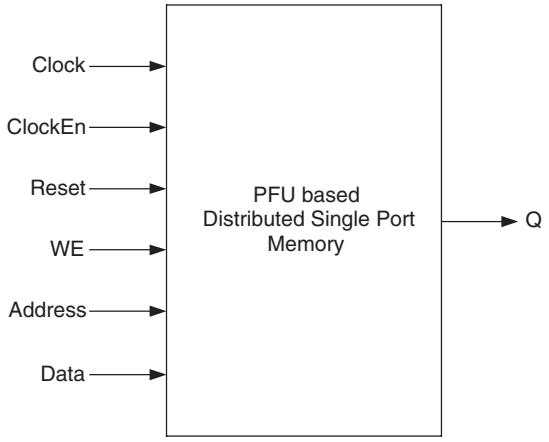


Distributed Single Port RAM (Distributed_SPRAM) – PFU Based

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 9-28 shows the Distributed Single Port RAM module as generated by IPexpress.

Figure 9-28. 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 9-17. The table lists the corresponding ports for the module generated by IPexpress and for the primitive.

Table 9-17. 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 9-29 and 9-30 show the internal timing waveforms for the Distributed Single Port RAM (Distributed_SPRAM) with these options.

Figure 9-29. PFU Based Distributed Single Port RAM Timing Waveform - without Output Registers

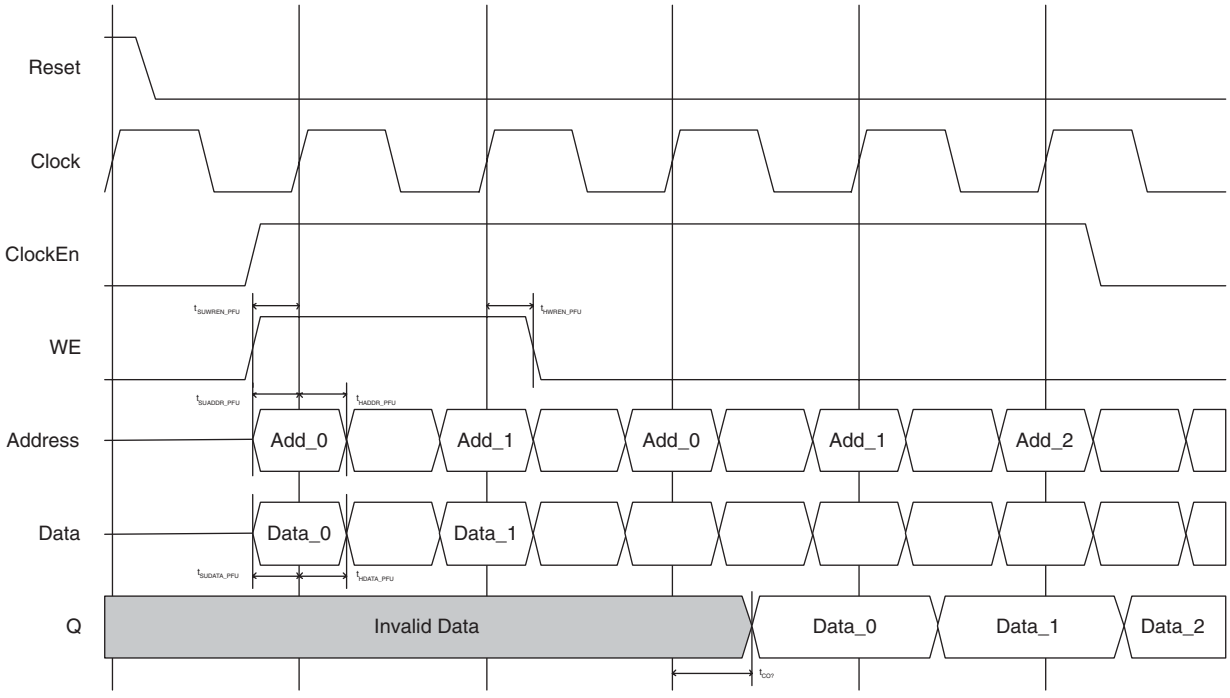
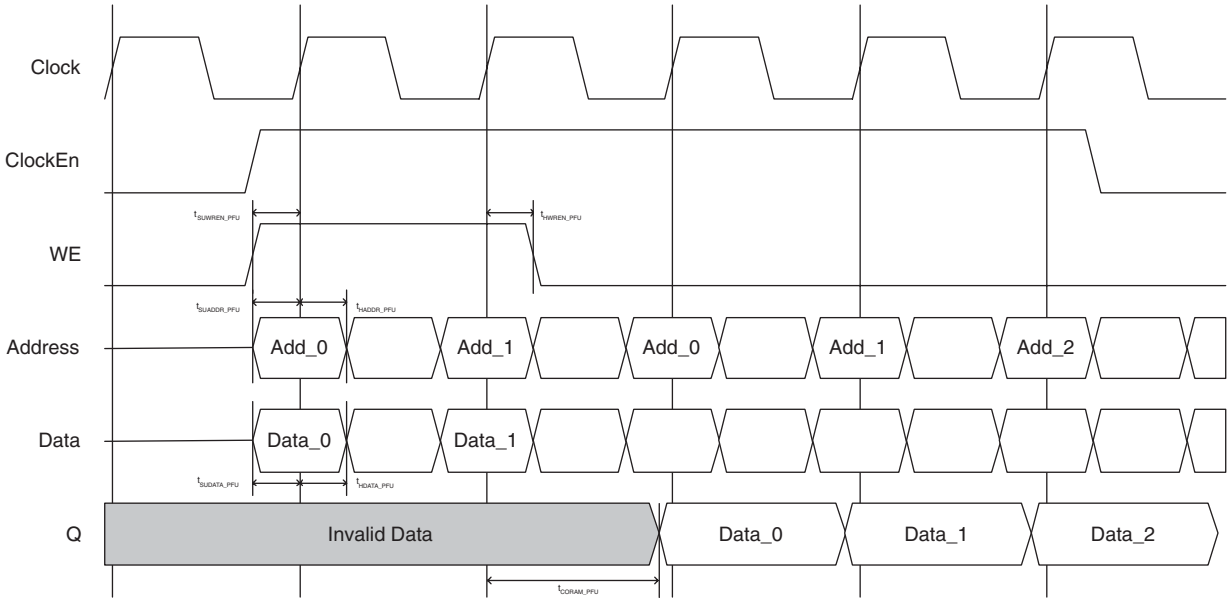


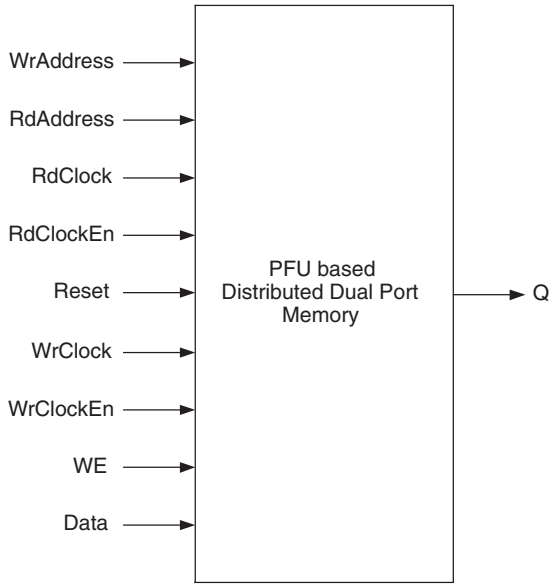
Figure 9-30. PFU Based Distributed Single Port RAM Timing Waveform - with Output Registers



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 9-31. 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 9-18. The table lists the corresponding ports for the module generated by IPexpress and for the primitive.

Table 9-18. 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 9-32 and 9-33 show the internal timing waveforms for the Distributed Dual Port RAM (Distributed_DPRAM) with these options.

Figure 9-32. PFU Based Distributed Dual Port RAM Timing Waveform - without Output Registers

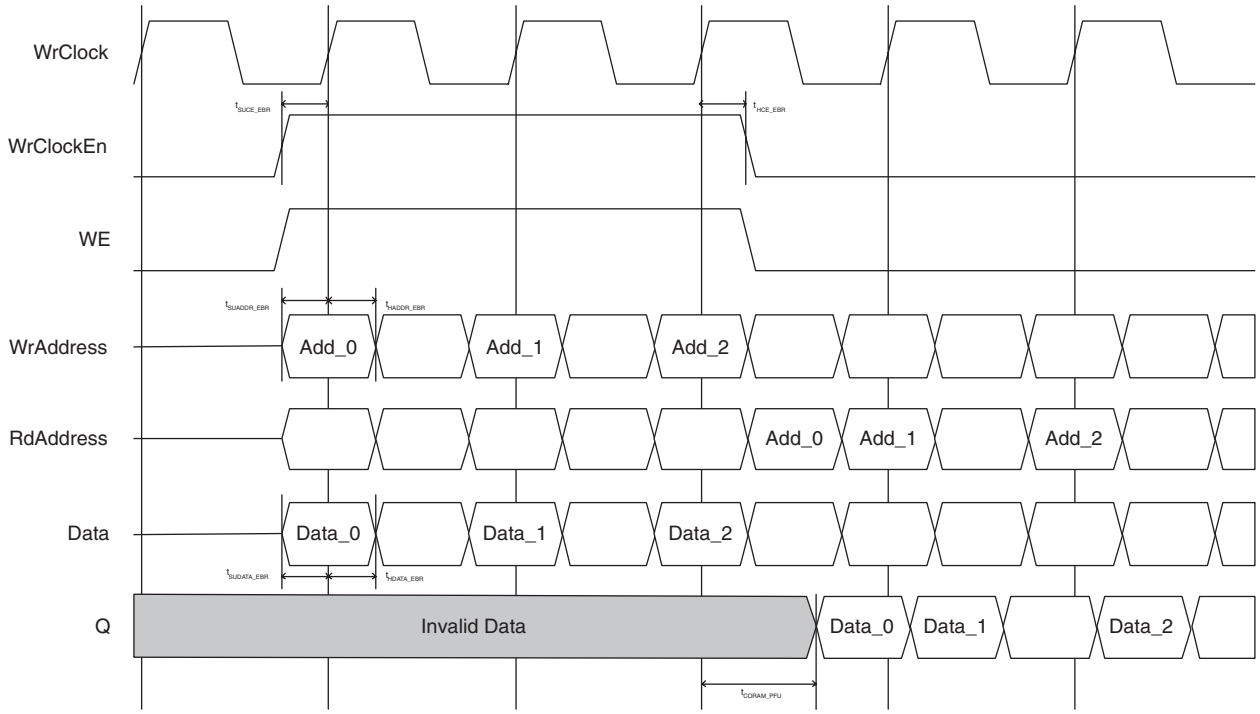
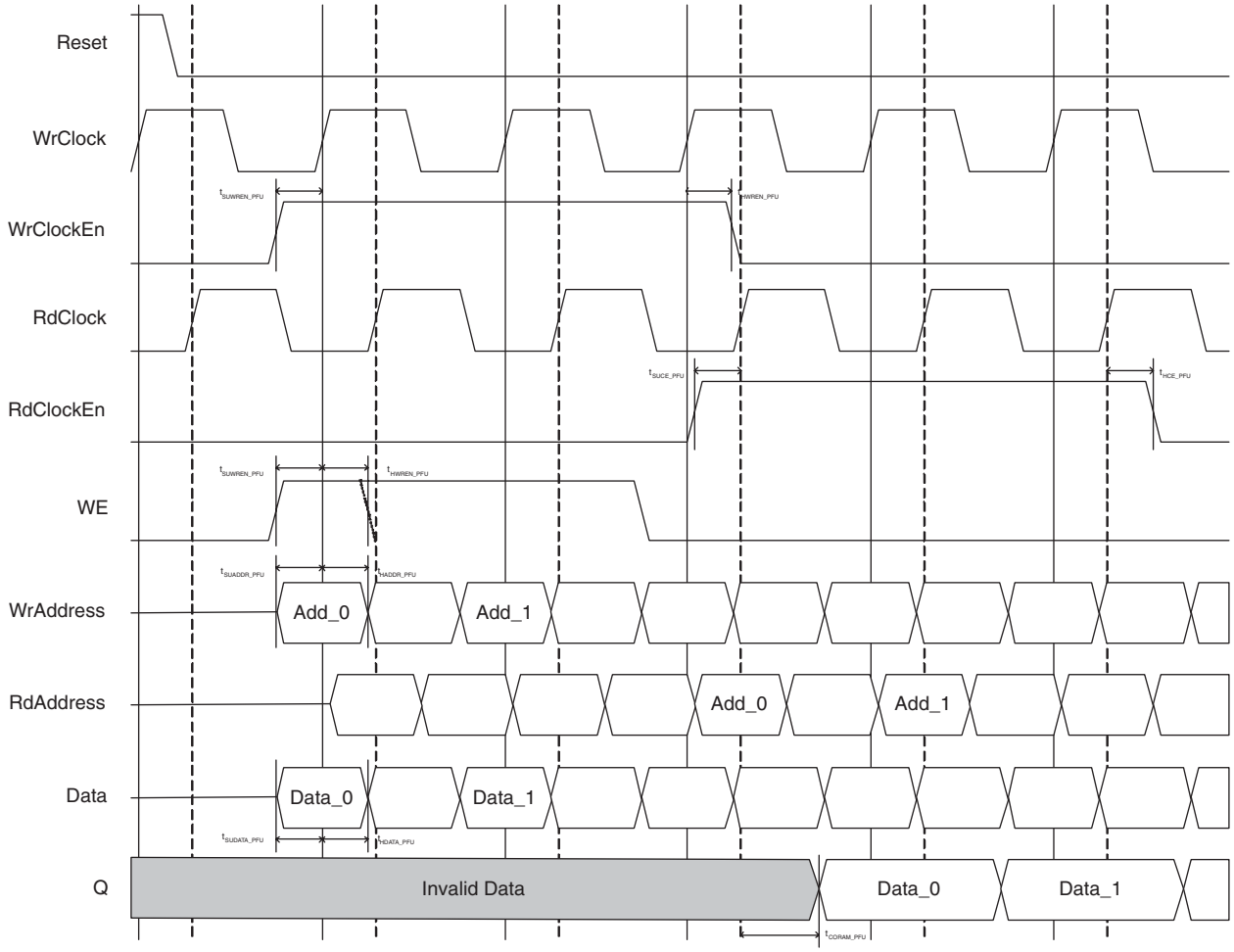


Figure 9-33. PFU Based Distributed Dual Port RAM Timing Waveform - with Output Registers

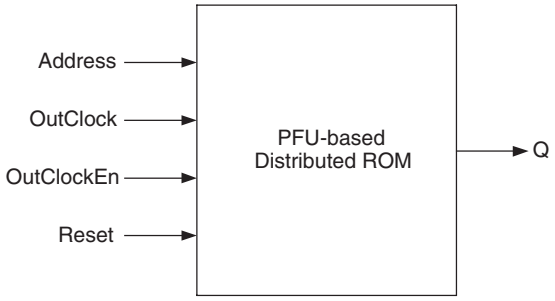


Distributed ROM (Distributed_ROM) – PFU Based

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 9-34 shows the Distributed ROM module as generated by IPexpress.

Figure 9-34. 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 9-19. The table lists the corresponding ports for the module generated by IPexpress and for the primitive.

Table 9-19. 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 9-35 and 9-36 show the internal timing waveforms for the Distributed ROM with these options.

Figure 9-35. PFU Based ROM Timing Waveform – without Output Registers

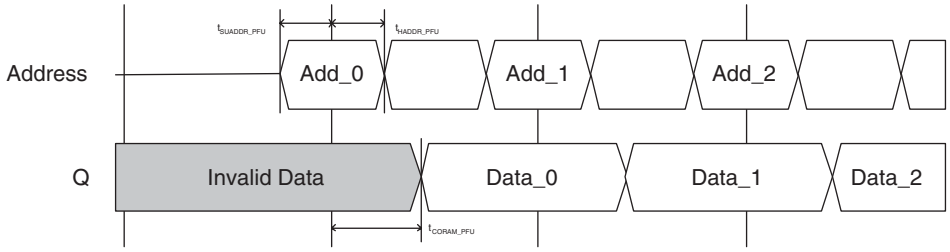
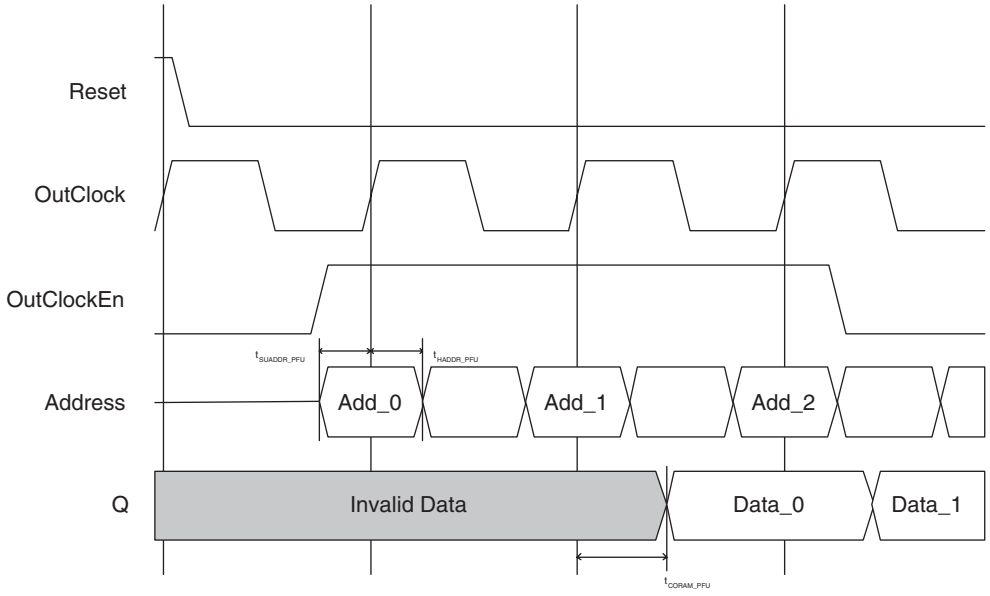


Figure 9-36. PFU Based ROM Timing Waveform – with Output Registers



Initializing Memory

In each of the memory modes it is possible to specify the power-on state of each bit in the memory array. This allows the memory to be used as ROM, if desired. 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. RAMs can optionally use this Initialization File also to preload the memory contents.

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.

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

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.

GSR

GSR or the Global Set/ Reset attribute is used to enable or disable the global set/reset for RAM element.

LatticeECP2 High-Speed I/O Interface

February 2006 Technical Note TN1105

Introduction

LatticeECP2™ 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 LatticeECP2 I/Os also have dedicated circuitry to support DDR and DDR2 SDRAM memory interfaces. This technical note details the use of LatticeECP2 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 1 and 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 3 and 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 require a Data Mask (DM) signals 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 10-1. Typical DDR SDRAM Interface

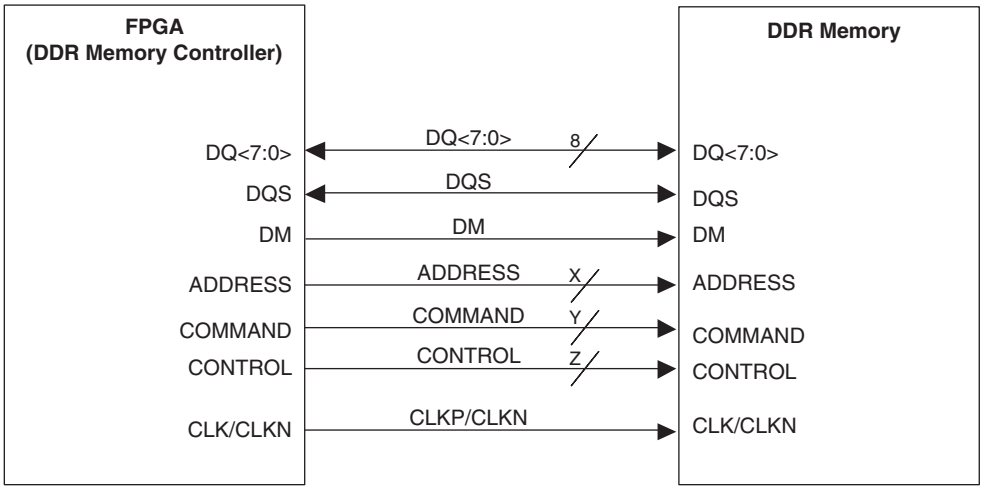
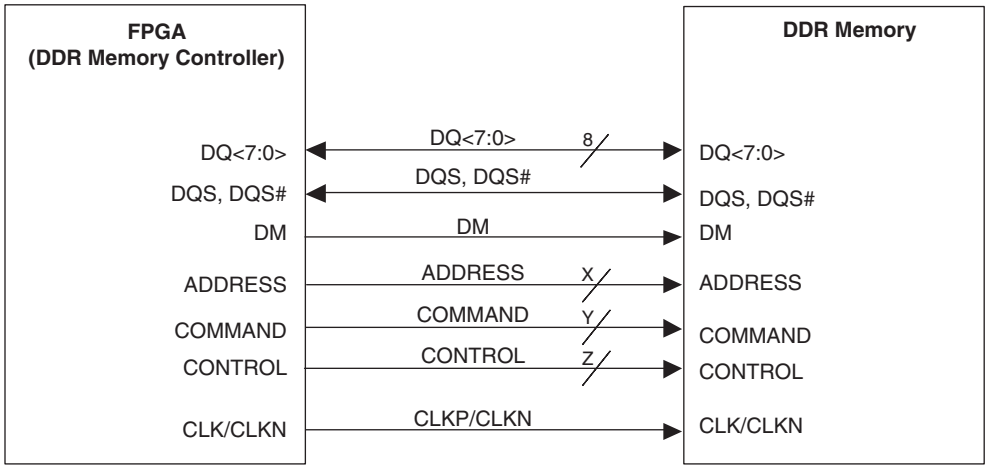


Figure 10-2. Typical DDR2 SDRAM Interface



The following two figures show the DQ and DQS relationship for memory read and write interfaces.

Figure 10-3. DQ-DQS During READ

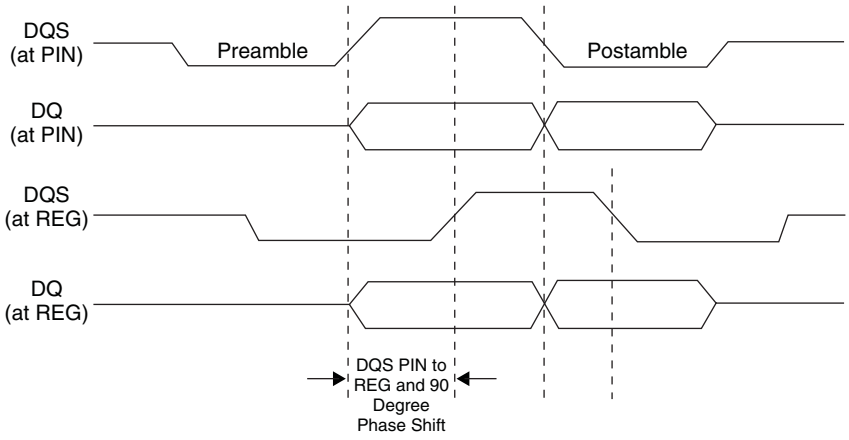
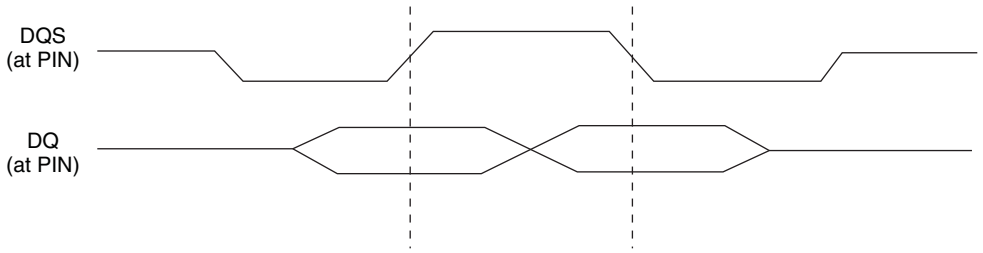


Figure 10-4. DQ-DQS During WRITE



Implementing DDR Memory Interfaces with LatticeECP2 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 LatticeECP2 device is edge-aligned with respect to the DQS signal. Therefore, the LatticeECP2 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 LatticeECP2 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.

LatticeECP2 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.

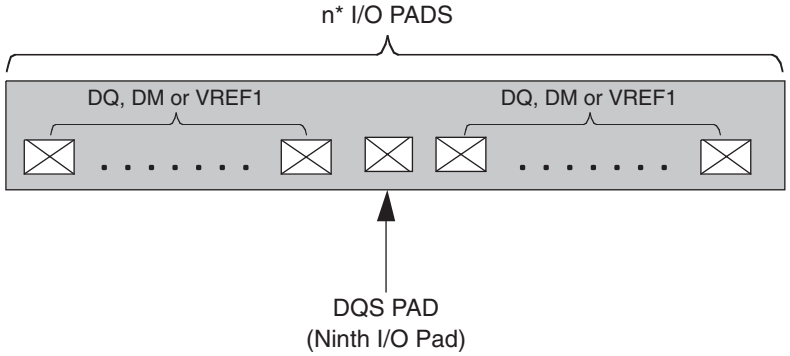
The dedicated DDR support circuit is also designed to provide comfortable and consistent margins for data sampling window.

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 LatticeECP2 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. LatticeECP2 device supports DQS signals on the bottom, left and right sides of the device. Each DQS signal on the bottom half 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 10-5. DQ-DQS Grouping



*n=18 on bottom banks and n=16 on the left and right side banks.

Figure 10-5 shows a typical DQ-DQS group for a LatticeECP2 device. The ninth I/O of this group of 16 I/Os (on the left and right side banks) and 14 I/Os (on the bottom bank) is the dedicated DQS pin. All eight pads before of the DQS and 7 (on the left and right side) and 9 (on the bottom bank) pads after the DQS are covered by this DQS bus span. The user can assign any eight of these I/O pads to be DQ data pins. Therefore, 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 pins is internally connected to the DQS phase shift circuitry. The pinout information contained in the LatticeECP2 Family Data Sheet shows pin locations for the DQS pads. Table 10-1 shows an extract from the data sheet.

In this case, the DQS is marked as LDQS8 (L = left side, 8 = associated PFU row/column). Since DQS is always the fifth True Pad in the DQ-DQS group, counting from low to high PFU Row/Column number, LDQS6 will cover PL2A to PL11B. Following this convention, there are eight pads before and seven pads after DQS for DQ available following counter-clockwise for left and bottom side of the device and following clockwise for top and right side of the device. The user can assign any eight of these pads to be DQ data signals.

Table 10-1. ECP2-50 672 fpBGA Pinout from LatticeECP2 Family Data Sheet

Ball Number	Ball Function	Bank	Dual Function	Differential
D2	PL2A	7	VREF2_7	T*
D1	PL2B	7	VREF1_7	C*
GND	GNDIO	7		
F6	PL5A	7		T
F5	PL5B	7		C
VCCIO	VCCIO	7		
E4	PL6A	7		T*
E3	PL6B	7		C*
VCC	VCC	7		
E2	PL7A	7		T
E1	PL7B	7		C
GND	GNDIO	7		
GND	GND	7		
H6	PL8A	7	LDQS8	T*
H5	PL8B	7		C*
F2	PL9A	7		T
VCCIO	VCCIO	7		
F1	PL9B	7		C
H8	PL10A	7		T*
J9	PL10B	7		C*
G4	PL11A	7		T
GND	GNDIO	7		
G3	PL11B	7		C
H7	PL12A	7		T*
VCCAUX	VCCAUX	7		

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
- **IDDRXMX1A** – The DDR input and DQS to system clock transfer registers with half clock cycle transfer
- **IDDRXMFX1A** – 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 10-6 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 10-6. DQSDLL Symbol

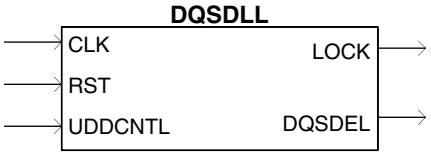


Table 10-2 provides a description of the ports.

Table 10-2. 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 an update signal to the DLL that will update the dynamic delay. When held low, this signal will update the DQSDEL.
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.

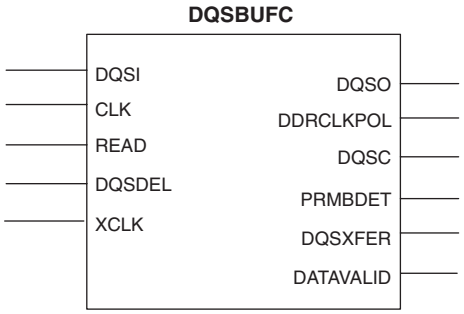
DQSDLL Configuration: By default, this DLL will generate a 90-degree phase shift for the DQS strobe based on the frequency of the input reference clock to the DLL. The user can control the sensitivity to jitter by using the LOCK_SENSITIVITY attribute. This configuration bit can be programmed to be either “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

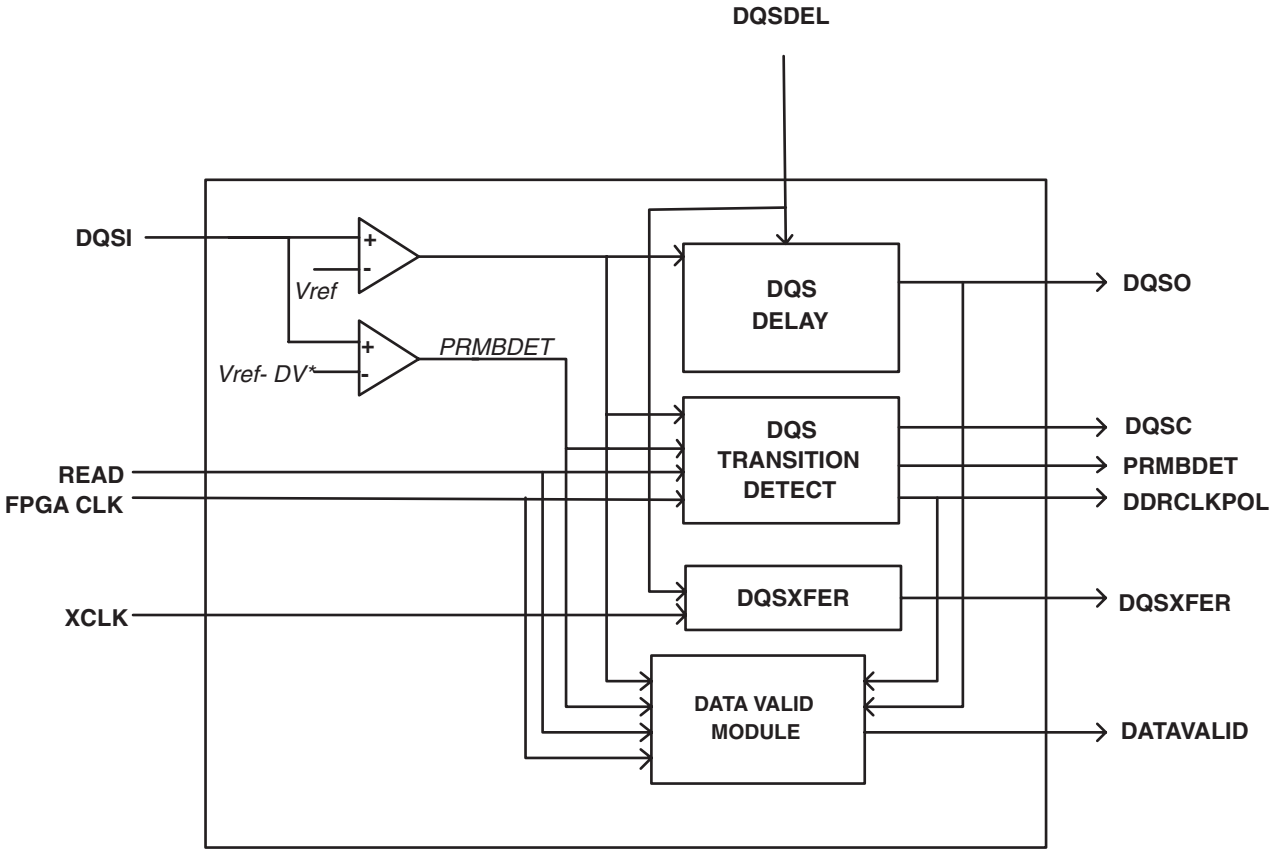
This primitive implements the DQS delay and the DQS transition detector logic. Figure 10-7 shows the primitive symbol.

Figure 10-7. DQSBUFC Symbol



The DQSBUFC is composed of the DQS Delay, the DQS Transition Detect and the DQSXFER block as shown in Figure 10-8. 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 10-8. 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 10-3 provides a description of the I/O ports associated with the DQSBUFC primitive.

Table 10-3. 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

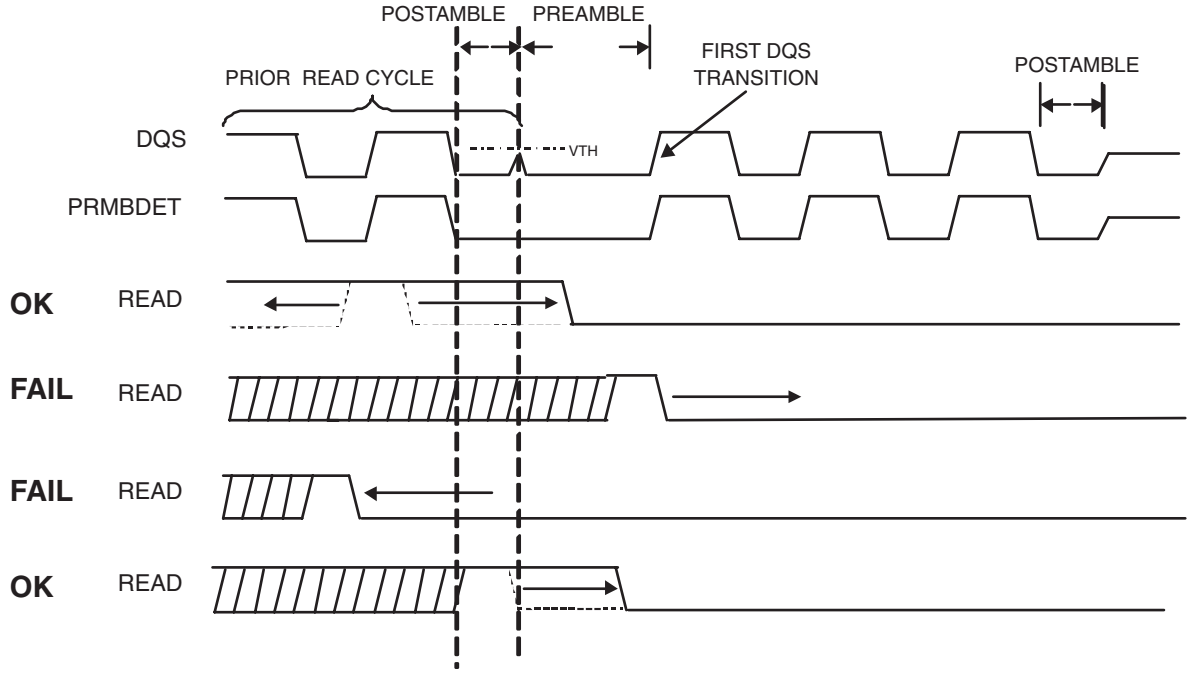
READ Pulse Generation

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 LatticeECP2 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 10-9 shows a READ Pulse timing example with respect to the PRMBDET signal.

Figure 10-9. READ Pulse Generation



IDDRMX1A

This primitive will implement the input register block. 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 should be 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 10-10 shows the primitive symbol and all the I/O ports.

Figure 10-10. IDDRMX1A Symbol

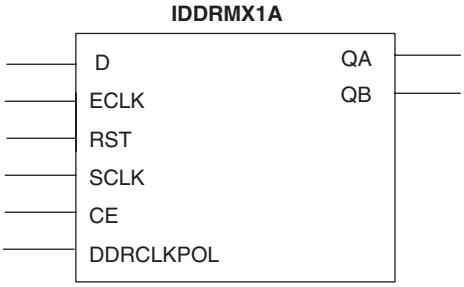


Table 10-4 provides a description of all I/O ports associated with the IDDRXB primitive.

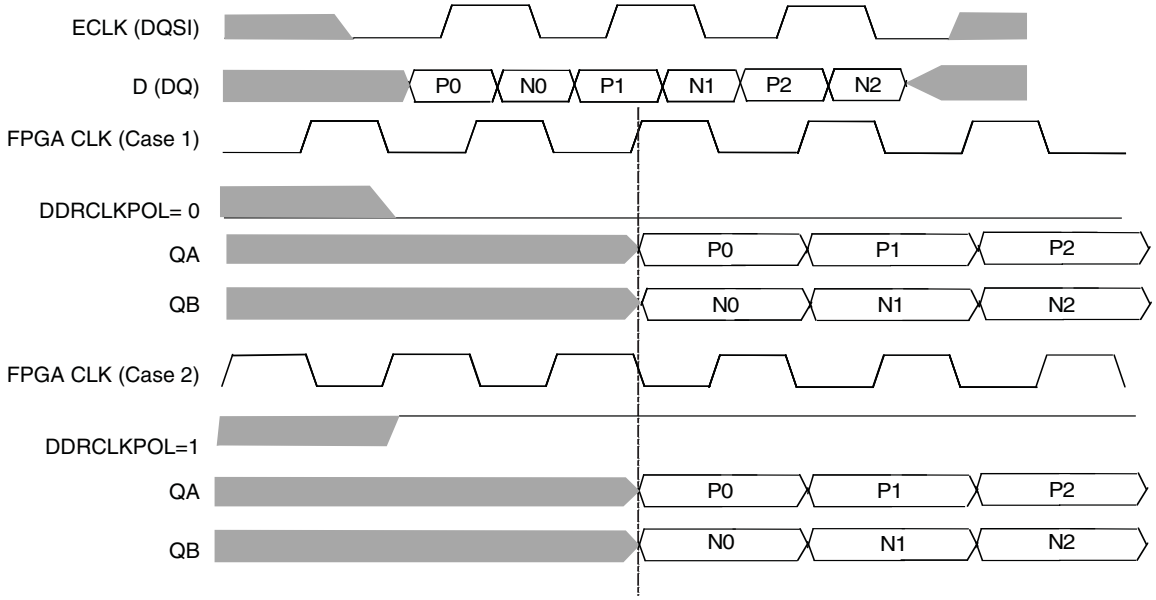
Table 10-4. IDDRMX1A Ports

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 10-11 shows the IDDRMX1A timing waveform.

Figure 10-11. IDDRMX1A Waveform



IDDRMFX1A

This primitive implements a full clock cycle transfer as compared the to the IDDRMX1A primitive that will only implement an half clock cycle transfer. 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. Figure 10-12 shows the primitive symbol and all the I/O ports.

Figure 10-12. IDDRMFX1A Symbol

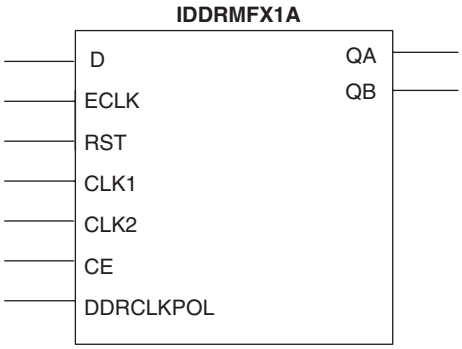


Table 10-5 provides a description of all I/O ports associated with the IDDRMFX1A primitive.

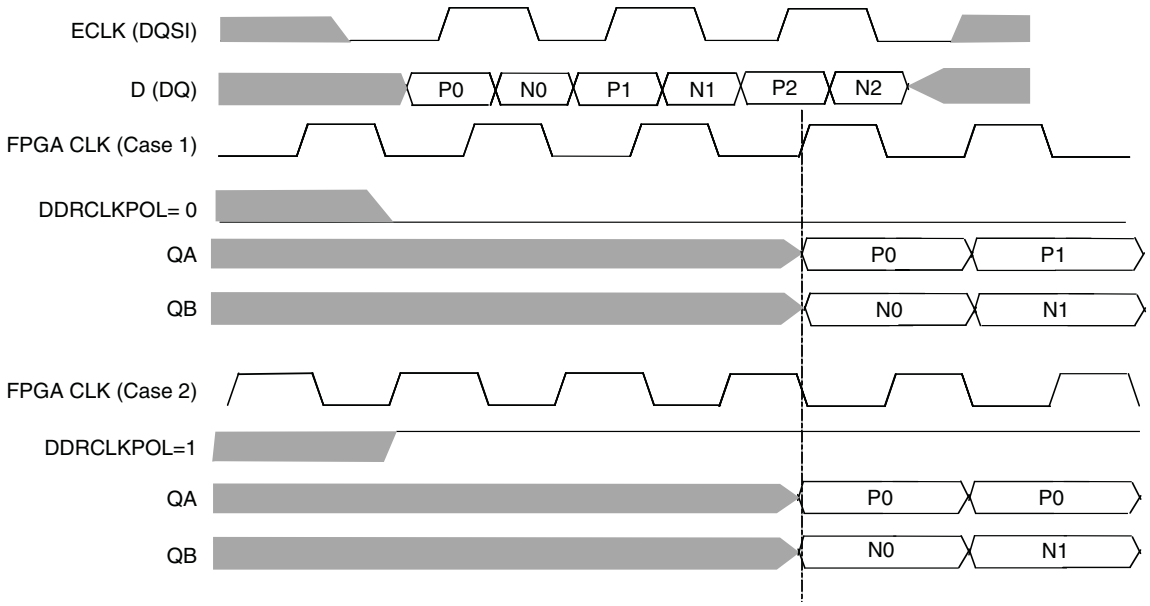
Table 10-5. IDDRMFX1A Ports

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 10-13 shows the IDDRMFX1A timing waveform.

Figure 10-13. IDDRMFX1A Waveform



ODDRMXA

The ODDRMXA primitive implements the output register for both the write and 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 10-14 shows the ODDRMXA primitive symbol and its I/O ports.

Figure 10-14. ODDRMXA Symbol

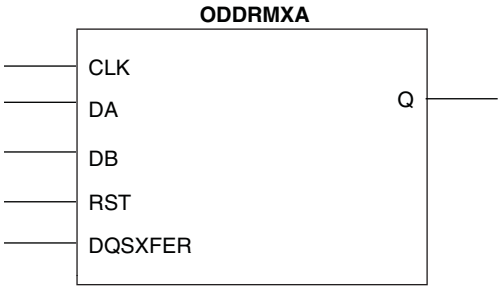


Table 10-6 provides a description of all I/O ports associated with the ODDRXB primitive.

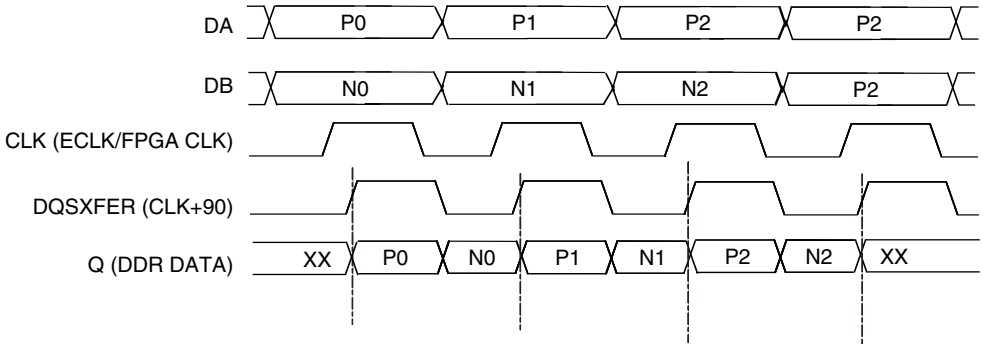
Table 10-6. ODDRMXA Ports

Port Name	I/O	Description
CLK	I	System CLK or ECLK
DA	I	Data at the positive edge of the clock
DB	I	Data at the negative 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. By default the software will implemented CE High and RST low.
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 10-15 shows the ODDRMXA timing waveform.

Figure 10-15. ODDRMXA Waveform



Memory Read Implementation

LatticeECP2 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 5-bit delay control from the on-chip DQSDLL. LatticeECP2 device supports two DQSDLL, one on the left and the other on the right side of the device. The DQSDEL generated by the DQSDLL on the left side of the device 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 of the device 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 this 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 mode the data is transferred to the FPGA core after the second stage of the register. In full clock transfer, an additional stage of I/O registers clocked by the FPGA clock is used to transfer the data to the FPGA core.

The LatticeECP2 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 10-16 and 10-17 show the READ interface block generated using the IPexpress™ tool in the ispLEVER software.

Figure 10-16. Software Primitive Implementation for Memory READ (Half Clock Transfer)

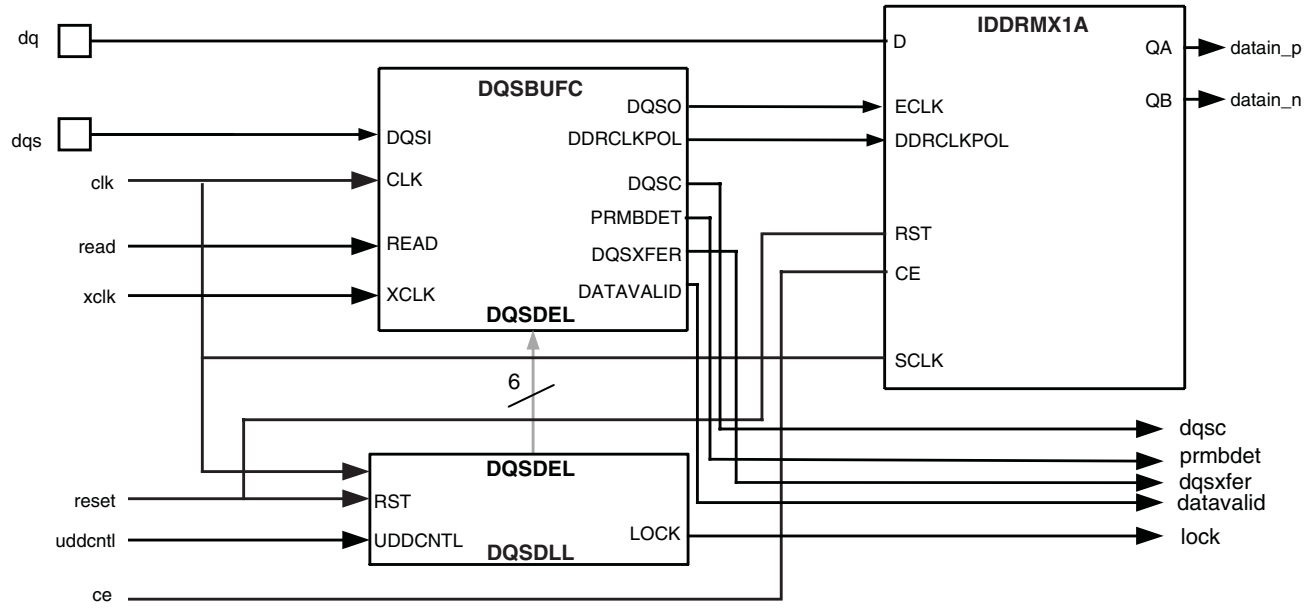
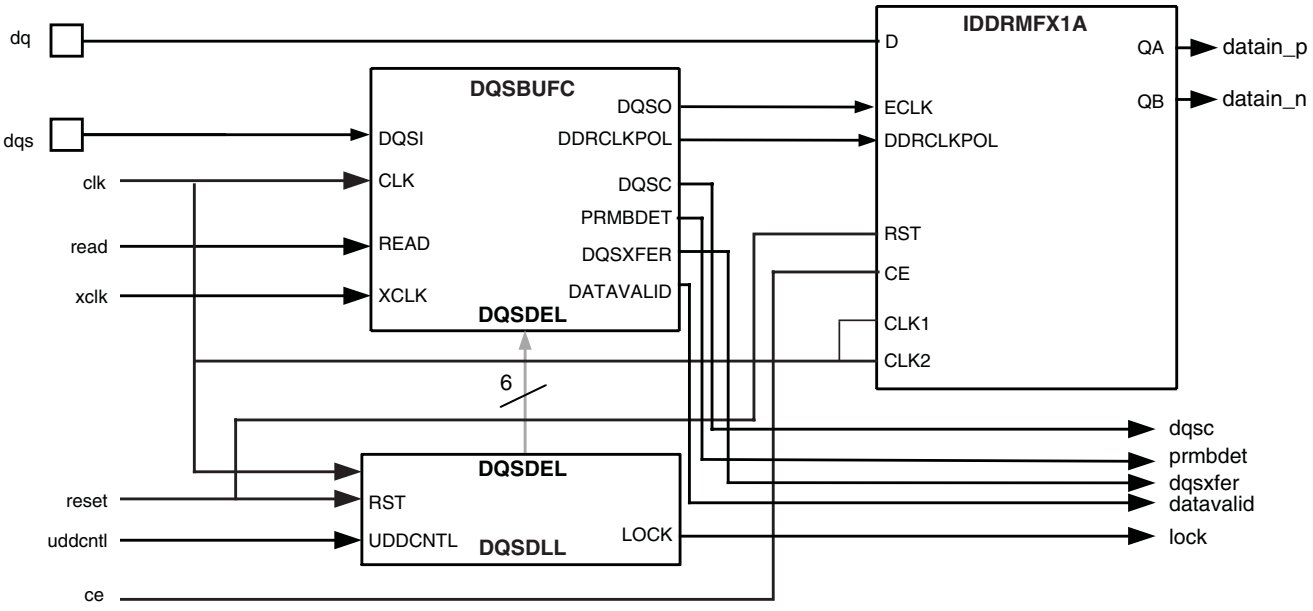


Figure 10-17. Software Primitive Implementation for Memory READ (Full Clock Transfer)



Read Timing Waveforms

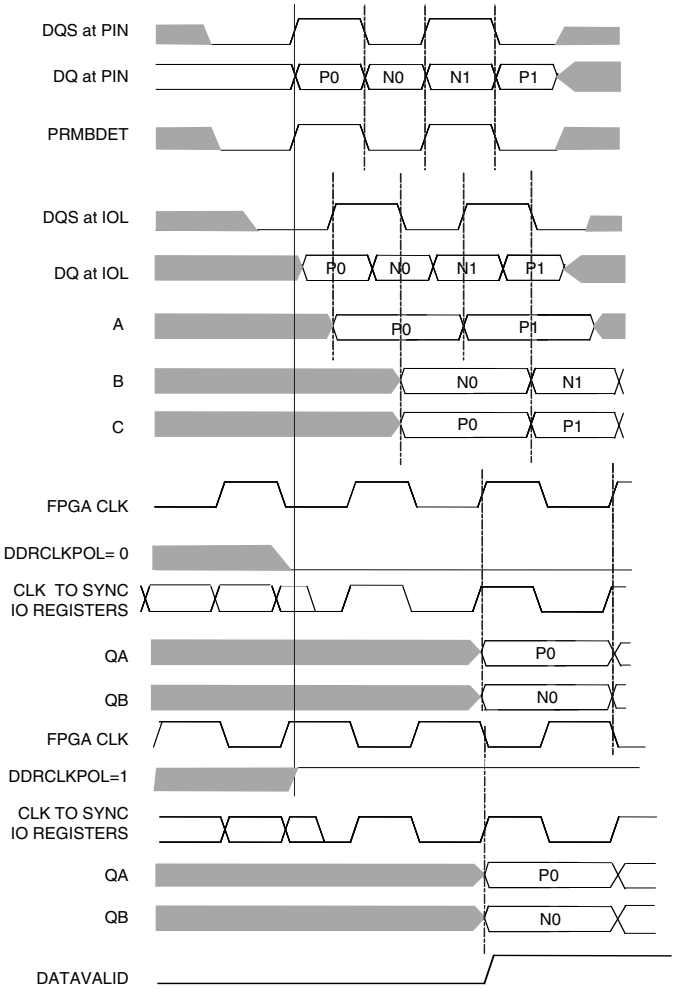
Figures 10-16 and 10-17 show READ data transfer for half and full clock cycle data transfer based on the results of the DQS Transition detector logic. This circuitry decides whether or not to invert the phase of FPGA system CLK to the synchronization registers based on the relative phases of PRMBDET 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 10-18 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.

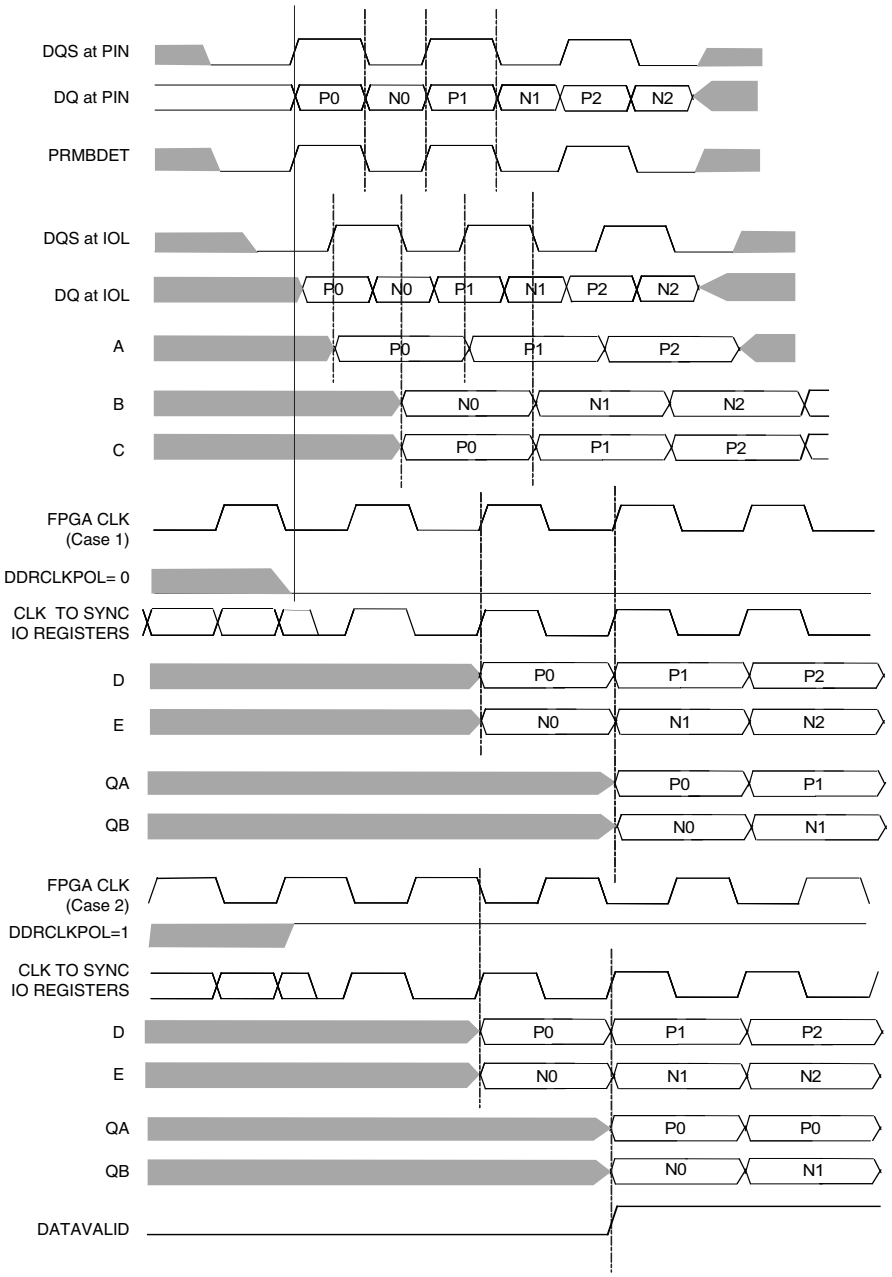
Figure 10-19 illustrates the DDR data timing using full clock transfer mode at different stages of IDDRMFX1A registers. In addition to the first two register stages in the half clock mode, the full clock transfer mode has an additional stage register clocked by the FPGA clock. In this case, D and E are the data streams after the second register stage presented to the final stage of registers clocked by the FPGA clock.

Figure 10-18. 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.DATAVALID signal goes HIGH when valid data enters the FPGA core.

Figure 10-19. 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.

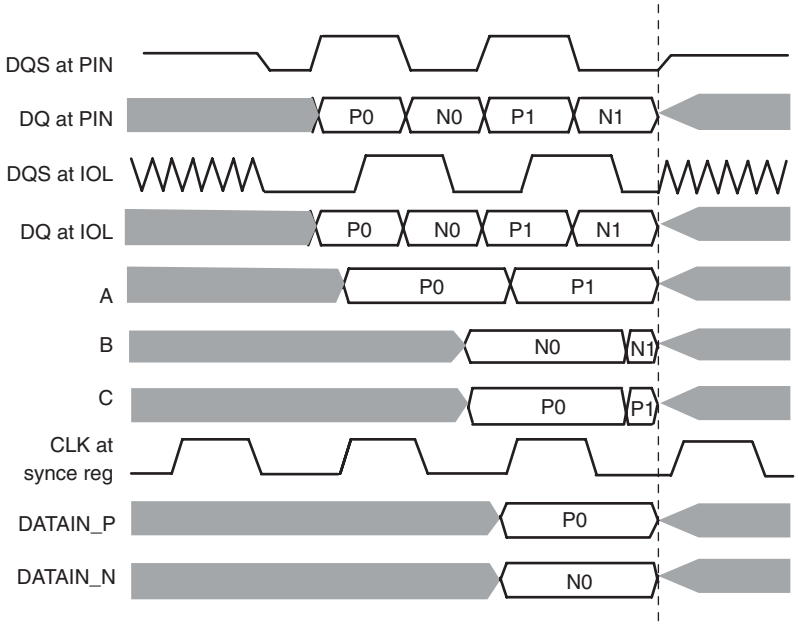
Data Read Critical Path

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 10-20. Postamble Effect on READ



LatticeECP2 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 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 10-21.

LatticeECP2 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.

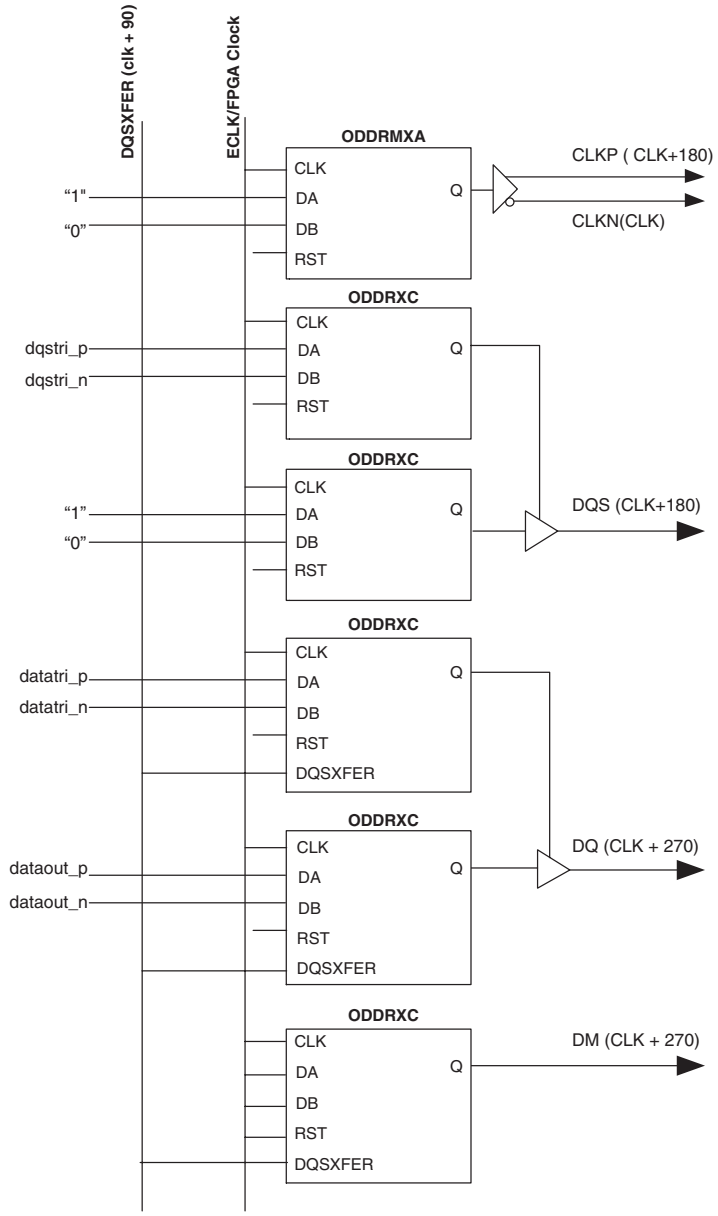
The DQSXFER output from the DQSBUFC block is the 90-degree phase shifted clock. This 90-degree phase shifted clock is used as an input to the ODDRMXA block. The ODDRMXA is used to generate the DQ and DM data outputs going to the memory. In the ODDRMXA module, the data is first registered using the ECLK 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 now 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 10-21 shows the DDR Write implementation using the DDR primitives.

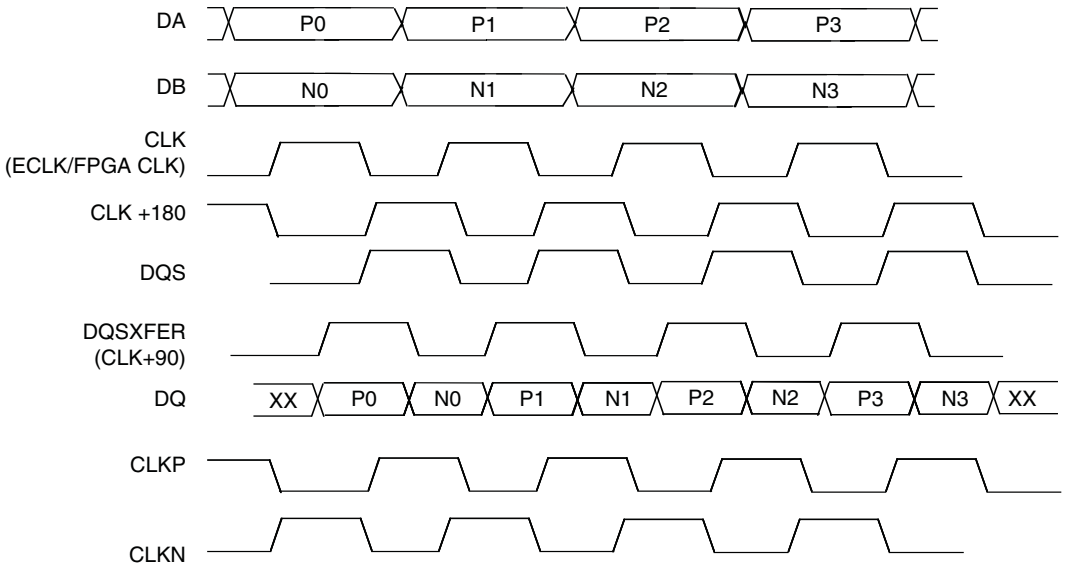
Figure 10-21. Software Implementation for Memory Write



Write Timing Waveforms

Figure 10-22 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 10-22. 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 LatticeECP2 devices.

- The LatticeECP2 devices have dedicated DQ- DQS banks. Please refer to the logical signal connections of the groups in the LatticeECP2 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 compared to the SDR ratio. A basic 2x DDR element provides four FPGA side bits for two I/O side bit at the 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. This delay 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 the DDR data at both edges of the edge 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 10-23 shows the primitive symbol for IDDRXC mode.

Figure 10-23. IDDRXC Symbol

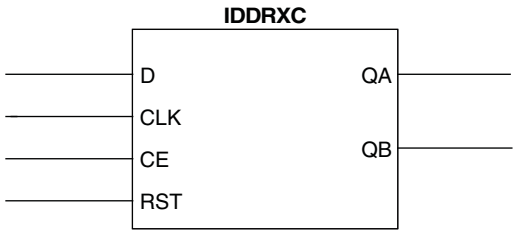


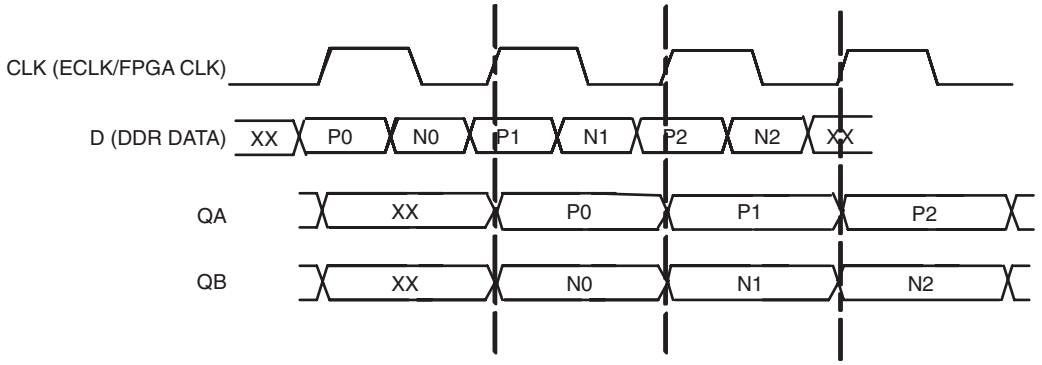
Table 10-7 lists the port names and descriptions for the IDDRXC primitive.

Table 10-7. 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 10-24 shows the timing waveform when using the IDDRXC module.

Figure 10-24. IDDRXC Waveform



IDDRFXA

This primitive is inputs the DDR data at both edges of the edge and generates two streams of data. The data is now registered by another stage of pipeline I/O registers clocked by the slow FPGA clock. There are two clocks going to this module. CLK1 registers the DDR and the first set of synchronization registers. This clock can be connected to the Edge clock or the FGPA clock. CLK2 is used by the third stage of registers and should be clocked by the FPGA clock.

Figure 10-25 shows the primitive symbol for the IDDRFXA mode.

Figure 10-25. IDDRFXA Symbol

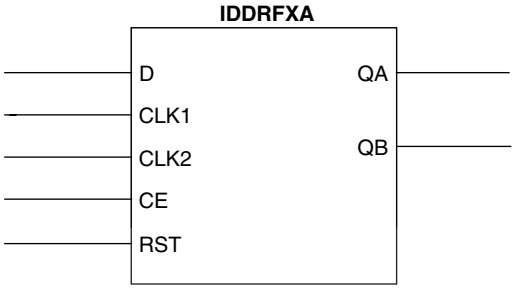


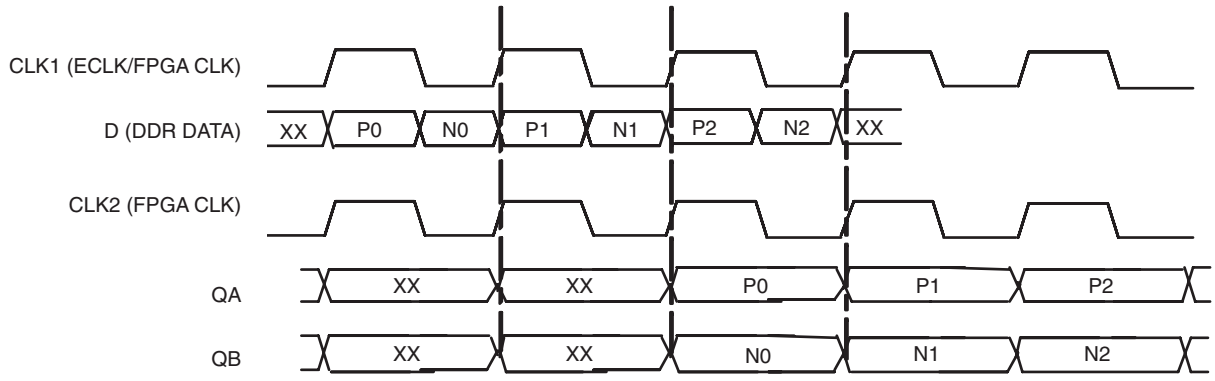
Table 10-8 lists the port names and descriptions for the IDDRFXA primitive.

Table 10-8. 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 10-26 shows the timing waveform when using the IDDRFXA module.

Figure 10-26. IDDRFXA Waveform



IDDRX2B

This primitive is used when a gearing function is required. This primitive inputs the DDR data at both edges of the edge and generates four streams of data. The DDR registers and the first set of synchronization registers are clock by the ECLK input of the primitive, which should be connected to the fast ECLK. The SCLK is used to clock the third stage of registers and should be connected to the FPGA clock.

This primitive outputs four streams of data. Two of these data streams are generated using the complementary PIO registers.

Figure 10-27 shows the primitive symbol for the IDDRX2B mode.

Figure 10-27. IDDRX2B Symbol

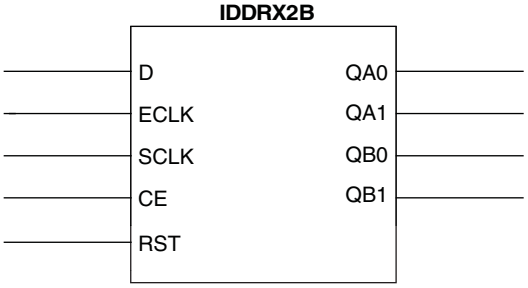


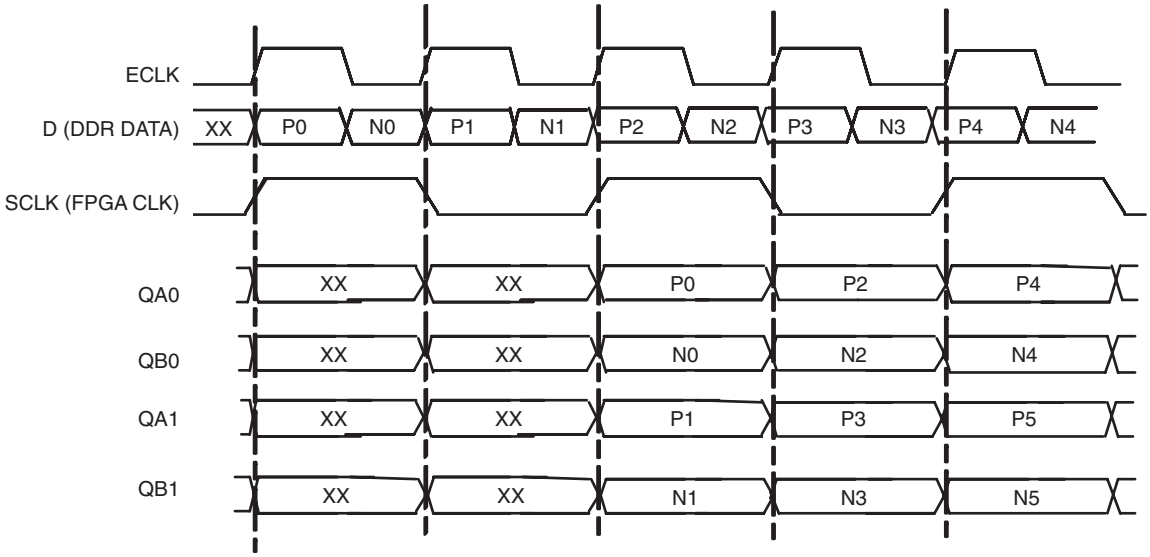
Table 10-9 lists the port names and descriptions for the IDDRX2B primitive.

Table 10-9. 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 10-28 shows the timing waveform when using the IDDRX2B module.

Figure 10-28. 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 10-29 shows the primitive symbol for the ODDRXC mode.

Figure 10-29. ODDRXC Symbol

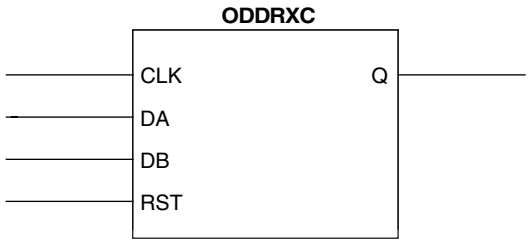


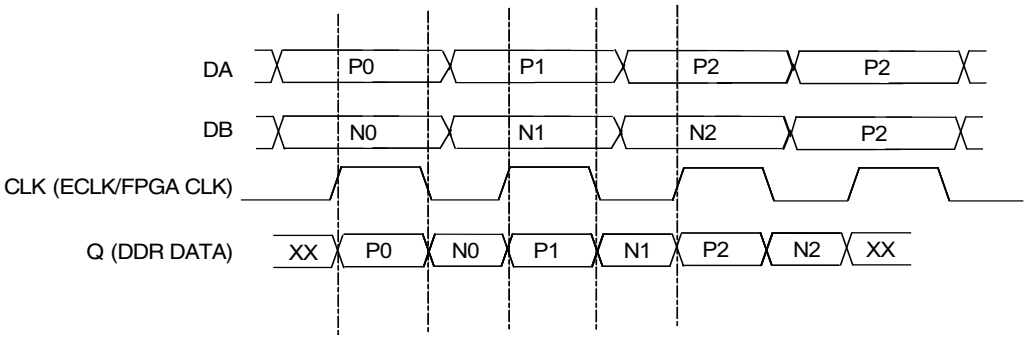
Table 10-10 lists the port names and descriptions for the IDDRX2B primitive.

Table 10-10. ODDRXC Port Names

PORT NAME	I/O	Definition
DA	I	Data at the Positive Edge of the clock
DB	I	Data at the Negative 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 10-30 shows the timing waveform when using the ODDRXC module.

Figure 10-30. 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 is used when using 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 10-31 shows the primitive symbol for the ODDRX2B mode.

Figure 10-31. ODDRX2B Symbol

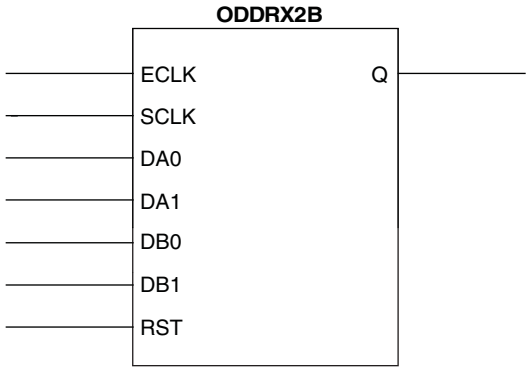


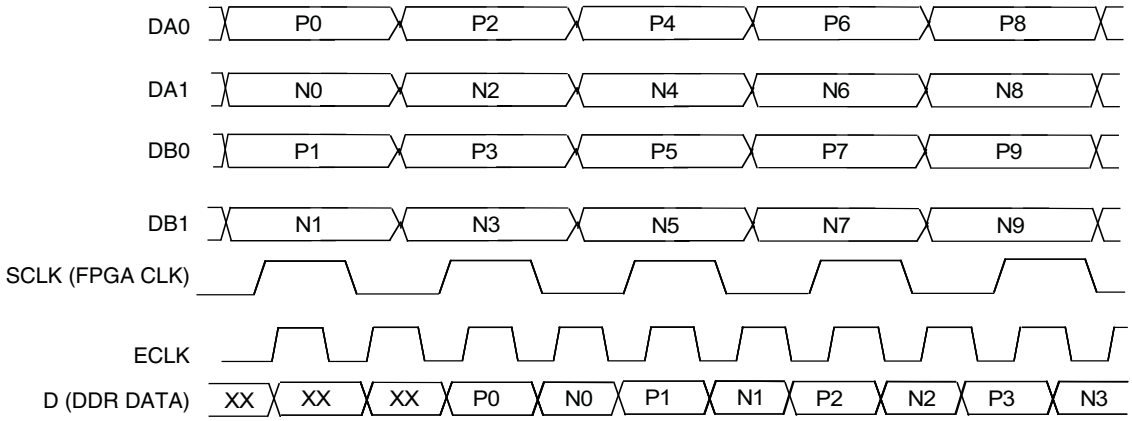
Table 10-11 lists the port names and descriptions for the ODDRX2B primitive.

Table 10-11. ODDRX2B Port Names

Port Name	I/O	Description
DA0, DA1	I	Data at the positive edge of the clock
DB0, DB1	I	Data at the negative 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 10-32 shows the timing waveform when using the ODDRXC module.

Figure 10-32. ODDRX2B Waveform



DELAYB

Data going to the DDR registers can be optionally delayed using the delay block. The DELAYB block receives a 4-bit delay value from the DLL. The user can also choose to implement a fixed delay value instead of using delay generated by the DLL. The various fixed delay choices can be made in the IPexpress software tool.

The DELAYB block can also be used to delay non-DDR inputs that use the input PIO register.

Figure 10-33 shows the primitive symbol for the DELAYB mode.

Figure 10-33. DELAYB Symbol

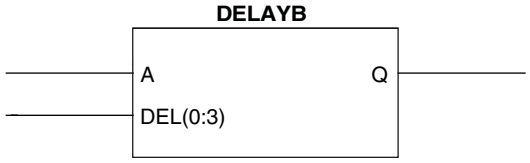


Table 10-12 lists the port names and descriptions for the DELAYB primitive.

Table 10-12. 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 to keep in mind when implementing generic DDR interfaces in LatticeECP2 devices.

- When implementing a 2x gearing mode, the complement PIO registers are used. This complementary PIO register can no longer be used to other perform DDR functions and should not be connected.

FCRAM ("Fast Cycle Random Access Memory") Interface

FCRAM is a DDR-type DRAM, which performs data output at both the rising and the 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 pin outs.

LatticeECP2 devices can implement FCRAM interface using the 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 compared to the 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

Power Estimation and Management for LatticeECP2 Devices

February 2006 Technical Note TN1106

Introduction

Power considerations in FPGA design are critical for determining the maximum system power requirements and sequencing requirements of the FPGA for the board. This technical note provides users with detailed power considerations such as sequencing. Also included are instructions for calculating power consumption in LatticeECP2™ devices using the Power Calculator available in the Lattice ispLEVER® design tool. General guidelines for reducing power consumption are also discussed.

Power Supply Sequencing

Power-Up Sequencing

There are three main power supplies that are required to power-up the LatticeECP2 device for proper operation: V_{CC}, V_{CCAUX} and V_{CCIO8}. Bank 8, or V_{CCIO8}, powers the sysCONFIG™ port and configuration circuitry and thus, it is required during power-up.

The nominal voltages for these power supplies are 1.2V for V_{CC}, 3.3V for V_{CCAUX} and 1.2 V for V_{CCIO8}. The nominal trip points for these power supplies are 0.6V to 0.8 V for V_{CC} and V_{CCIO8}, and 2.2V to 2.5 V for V_{CCAUX}. These power supplies are recommended to transition from the trip point to the corresponding minimum operating voltage (refer to device data sheet for the minimum voltage supply values) no faster than 2ms. This transition recommendation is most critical for the last power supply that reaches the trip point.

Also, each power supply must follow a monotonically clean ramp between the trip points and the minimum required supply voltage. Slow power supply ramps in the 10's of milliseconds to 100's of milliseconds are critical to ensure that the transitions around trip points are monotonic. Multiple transitions through the trip point may cause multiple internal power-on reset sequencing.

The rest of the bank voltages can come up in any sequence.

After initialization is complete, if any V_{CC}, V_{CCAUX} or V_{CCIO8} drops below its power-down trip point, the device will reset. Any $V_{CCIO[7:0]}$ can be removed without resetting the device after initialization is complete.

Power-Down Sequencing

During power-down, power should be removed from one of the supplies' V_{CC}, V_{CCAUX} or V_{CCIO8} first to ensure that no high currents are seen on the input pins as the other V_{CCIO} supplies are removed. This only applies when input signals are still being driven, such as in hot-socketing applications.

For non-hot-socketing applications, the input signals are likely to be powered from the same supply as V_{CCIO}. Therefore, they will usually be less than or equal to V_{CCIO} during power down.

Power Sequencing Recommendations

The recommended power supply sequencing for LatticeECP2 devices is V_{CC} or V_{CCIO8} in any sequence followed by V_{CCAUX}. It is suggested that the V_{CC} and V_{CCIO8} should reach their minimum voltage value before V_{CCAUX} reaches its minimum value. It should also follow the rule of a minimum of 2ms during transition from the trip point to the minimum voltage.

Power Calculation Hardware Assumptions

The power consumption for the device can be coarsely broken down into the DC portion and the AC portion.

The DC Power (or the static power consumption) is the total power consumption of the used and the unused resources. These components are fixed for each resource used and depend 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 the power consumption, associated with the used resources, is the dynamic part of the power consumption. The AC 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 * NRESOURCE]
+ [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, on the other hand, is governed by the following equation:

Total AC Power (Resource)
= $K_{RESOURCE} * f_{MAX} * AF_{RESOURCE} * N_{RESOURCE}$

Where:

$N_{RESOURCE}$	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 LUT is calculated as per the following equation,

Total AC Power (LUT)
= $K_{LUT} * f_{MAX} * AF_{LUT} * N_{LUT}$

N_{LUT}	is the number of LUTs used in the design.
K_{LUT}	is the Power constant for the LUT blocks in mW/MHz.
f_{MAX}	is the max frequency of the LUT clock measured in MHz
AF_{LUT}	is the activity factor for the LUT. The Activity Factor is a percentage of the switching frequency.

Another example is the power consumption of the EBR block, which is calculated as follows:

Total AC Power (EBR)
= $K_{EBR} * f_{MAX} * AF_{EBR} * N_{EBR}$

NEBR	is the number of EBR blocks used in the design.
KEBR	is the Power constant for the EBR blocks in mW/MHz.
FMAX	is the max frequency of the EBR clock measured in MHz.
AFEBR	is the activity factor for the Read and Write ports of the EBR. The Activity Factor is a percentage of the switching frequency.

Also note that the LUT can be configured in Logic, Ripple or Distributed RAM modes. Each of these modes has a different power constant/power coefficient. However, the equations stay the same.

The AC power of some of the dedicated blocks can be calculated using the following equation:

Total AC Power (Dedicated Resource)
= $K_{RESOURCE}$ * f_{MAX} * $N_{RESOURCE}$

Where:

$N_{RESOURCE}$	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 measured in MHz.

Power Calculator

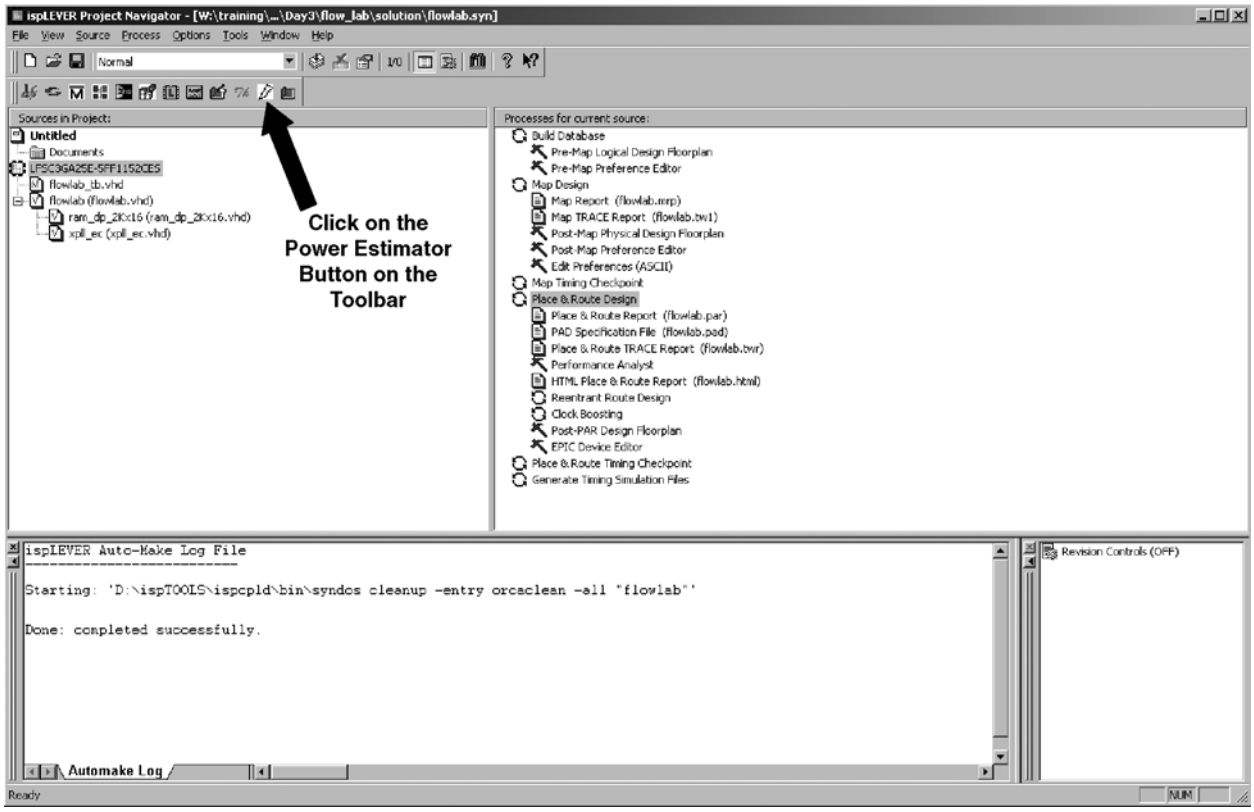
Power Calculator is a powerful tool which allows users to estimate power consumption at three different levels as described below. The LatticeECP2 device family Power Calculator is supported in ispLEVER 6.0 or later.

1. **Estimate of the Utilized Resources**
   This is the first level of estimation. In this stage, the user provides estimates of device usage in the Power Calculator Wizard and the tool provides a rough estimate of the power consumption.
2. **Post Place and Route Design**
   This is the second level of estimation. In this stage, the user's design is placed and routed in the ispLEVER design tool. This is a more accurate approach since the exact device utilization information is imported from the placed and routed netlist (NCD) file. During this stage, the user must provide frequencies (f_{MAX}), activity factors (AF%), and the toggle rate (TR in MHz).
3. **Post Place and Route and Post Simulation**
   The third level of power calculation takes the power calculations in the second stage and allows users to also import the post-simulation file (VCD file, output from ModelSim®) and the trace report file (TWR file) or the preference file (PRF file) as generated by ispLEVER. These files provide the data for the frequency (f_{MAX}), activity factors (AF%), and the toggle rate (TR in MHz) cells in the power calculator making.

Starting the Power Calculator

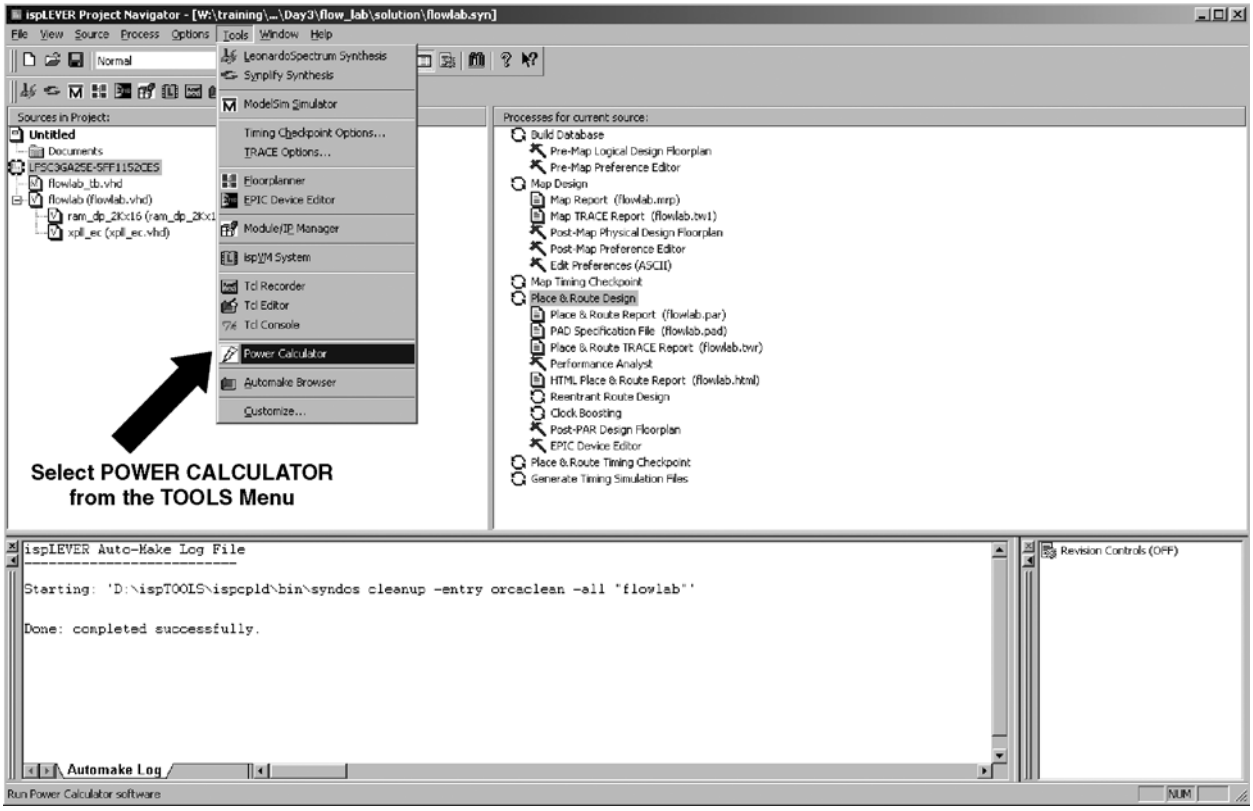
There are two methods for launching the Power Calculator. The first is by clicking the **Power Calculator** button in the toolbar as shown in Figure 11-1.

Figure 11-1. Starting Power Calculator from the Toolbar



Alternatively, users can launch Power Calculator by going to the **Tools** menu and selecting the **Power Calculator** option as shown in Figure 11-2.

Figure 11-2. Starting Power Calculator from the Tools Menu

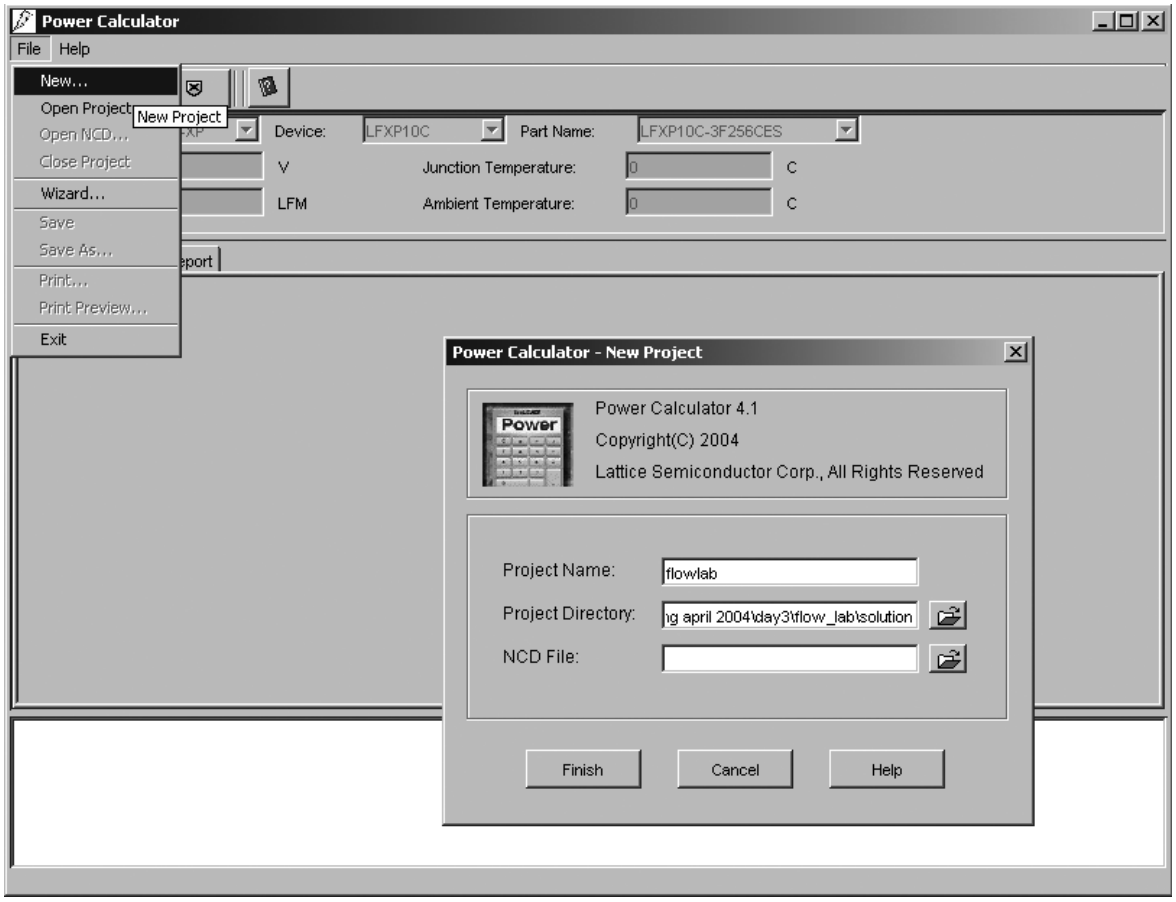


Power Calculator does not support some of Lattice's older devices. The toolbar button and menu item are only present when supported devices are selected.

Creating a Power Calculator Project

Once the Power Calculator is started, users will see the Power Calculator window. Click on **File Menu** and select **New** to get to the Start Project window, as shown in Figure 11-3.

Figure 11-3. Power Calculator Start Project Window (Create New Project)



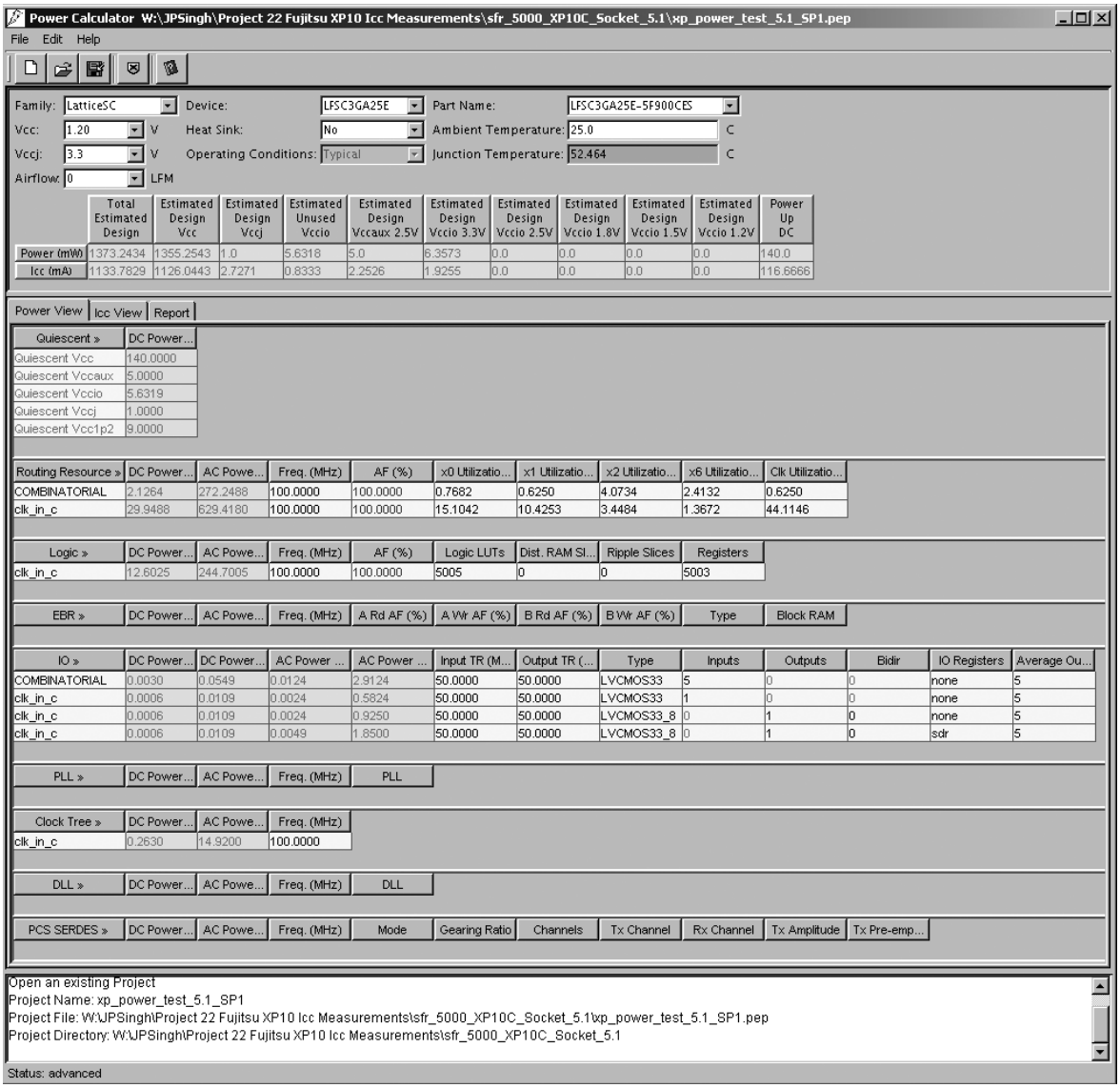
The Start Project Window is used to create a new Power Calculator Project (*.pep project). Three pieces of data are input into the Start Project Window.

1. The Power Calculator Project Name by default is same as the Project Navigator project name. Users can change the name if desired.
2. The Power Calculator project (*.pep) file is stored in the Project Directory. By default, it is stored in the main project folder.
3. Input an NCD file (if available) or browse to the NCD file in a different location.

Power Calculator Main Window

The Power Calculator main window is shown in Figure 11-4.

Figure 11-4. Power Calculator Main Window (Type View)



The top pane of the window contains information about the device family, device and the part number as it appears in the Project Navigator. The V_{CC} used for the power calculation is also listed. Users have the choice to select the core voltage, V_{CC} with +5% of the nominal value (or values). The option to select V_{CCJ} is also available. Users provide the ambient temperature and the junction temperature is then calculated.

Values for airflow in Linear Feet per Minute (LFM) can also be entered along with heat sink to get the junction temperature.

A table in the upper section of the Power Calculator summarizes the currents and power consumption associated with each type of power supply for the device. This table 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, an interactive spreadsheet type interface with all values in terms of power consumption in mW.

The first column breaks down the design into clock domains. The second and third columns, which are shaded blue in the tool, provide the DC (or static) and AC (or dynamic) power consumption, respectively.

For I/Os, there are four columns that are shaded blue. These provide the DC and AC power for I/Os for the core voltage, V_{CC} and the I/O voltage supply, V_{CCIO}. The first three rows show the Quiescent Power for V_{CC}, V_{CCAUX} and V_{CCJ}.

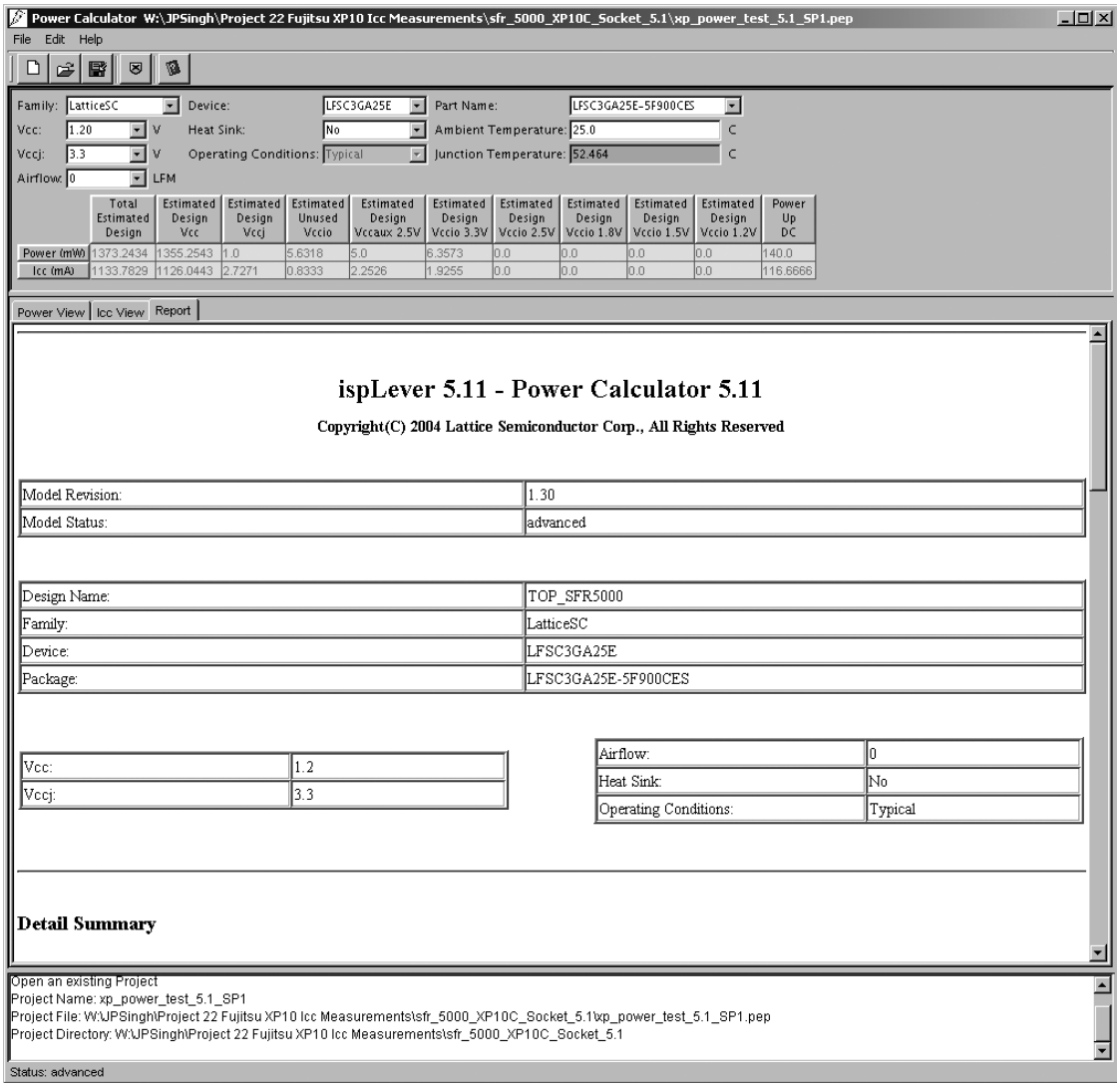
These are DC power numbers for a blank device or a device with no resource utilization.

Some of the cells are shaded yellow in the tool. 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. **I_{CC} View**
   The second tab is the I_{CC} view. This tab is exactly same as the Power View tab, except all values are in terms of current measured in mA.

3. **Report**
   The third and final tab is the Report View, an HTML report. It summarizes the contents of the Power and I_{CC} views.

Figure 11-5. Power Calculator Main Window (Report View)

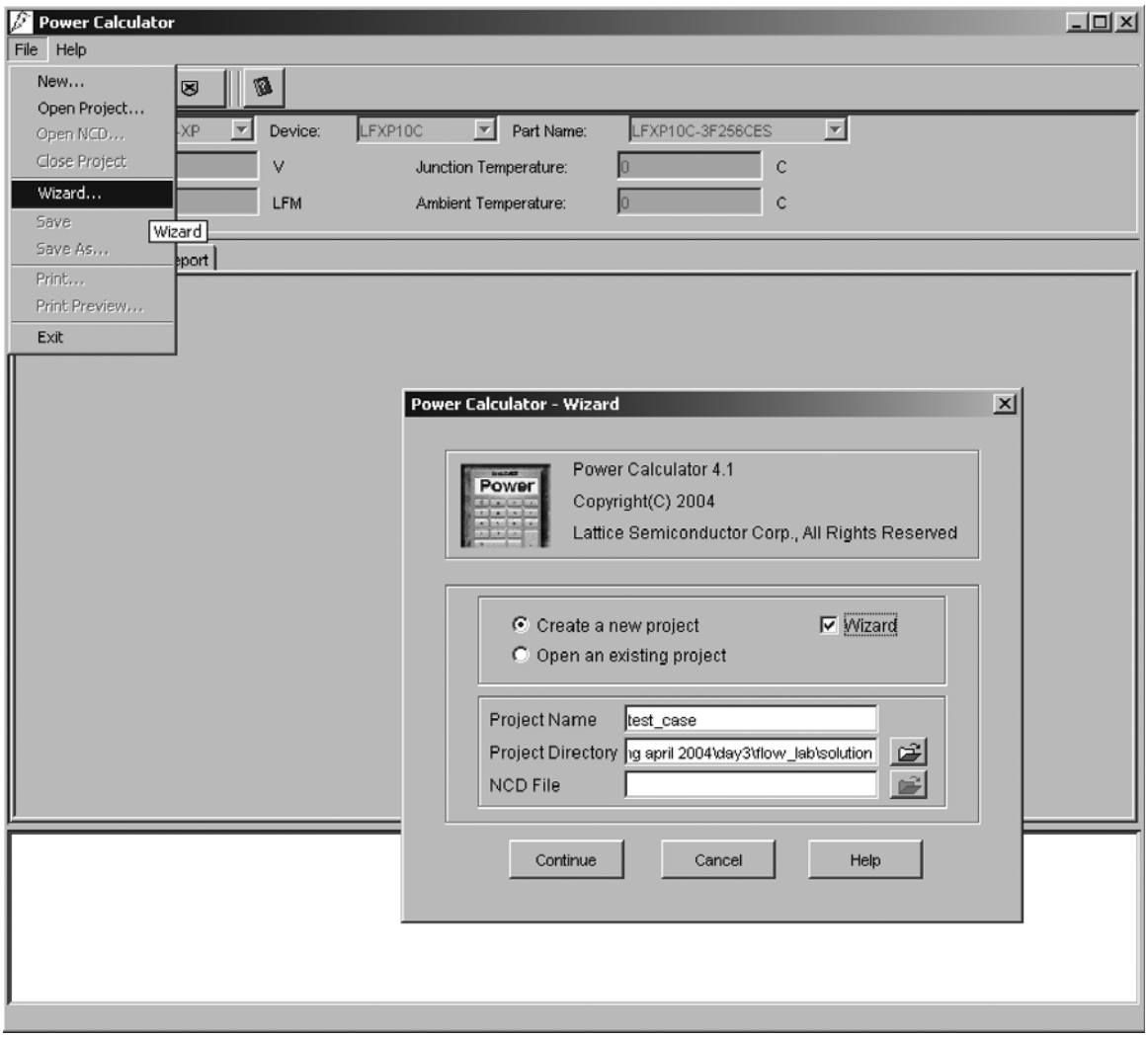


Power Calculator Wizard

The Power Calculator Wizard allows users to estimate the power consumption of a design. Since this estimation is done before the design is created, it is important for users to understand the logic requirements of their design. Through the wizard, users can provide these parameters for Power Calculator to estimate the power consumption of the device.

To start the Power Calculator in the wizard mode, go to the **File** menu and select **Wizard**. Alternatively, click on the Wizard button and get the Power Calculator - Wizard window as shown in Figure 11-6. Select **Create a New Project** and check the **Wizard** check box in the Power Calculator Start Project window. Users are required to 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 11-6. Power Calculator Start Project Window (Using the New Project Window Wizard)



In the next screen, as shown in Figure 11-7, users select the device family, device and appropriate part number. After making these selections, click **Continue**, as shown in Figure 11-8.

Figure 11-7. Power Calculator Wizard Mode Window - Device Selection

In the following screens (Figures 11-8-11-13), users can select additional resources such as I/O types, a clock name, the frequency at which the clock in running, and other parameters by selecting the appropriate resource using the pull-down menu.

1. Routing Resources
2. Logic
3. EBR
4. I/O
5. PLL
6. Clock Tree
7. DLL

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, users can specify parameters such as frequency, activity factor, etc. Click the **Create** button for each clock-driven resource to include the parameters specified for it in the final window.

These parameters are then used in the **Power Type View** window after clicking **Finish** (see Figure 11-14).

Figure 11-8. Power Calculator Wizard Mode Window - Resource Specification - Logic

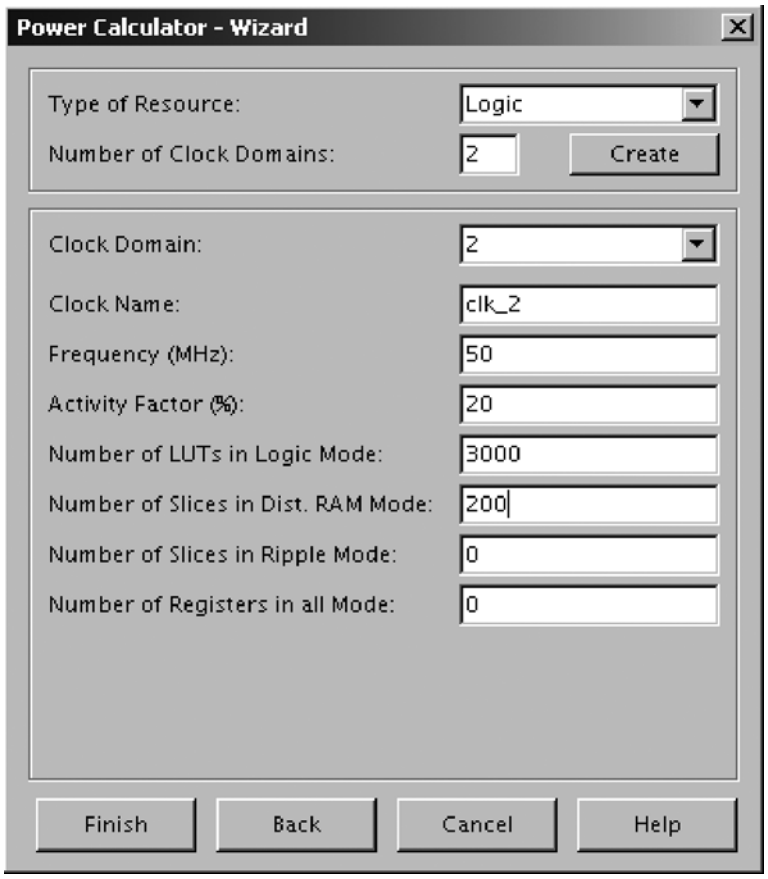


Figure 11-9. Power Calculator Wizard Mode Window - Resource Specification - EBR

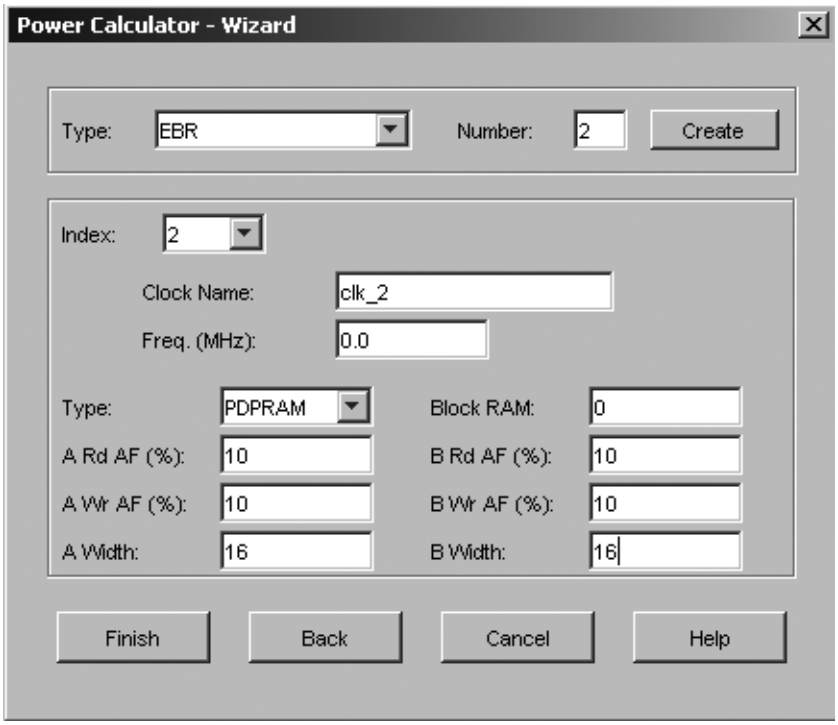


Figure 11-10. Power Calculator Wizard Mode Window - Resource Specification - PLL

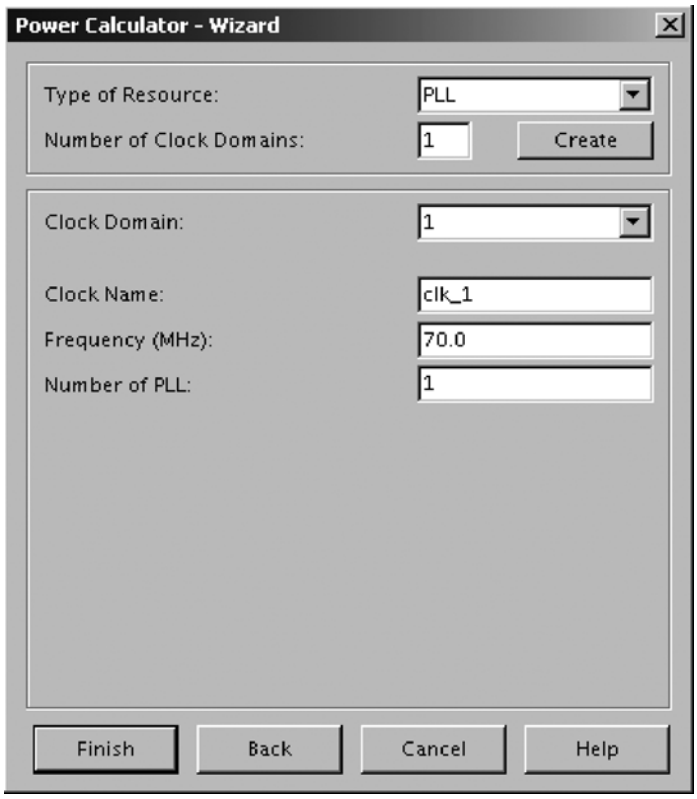


Figure 11-11. Power Calculator Wizard Mode Window - Resource Specification - Routing Resources

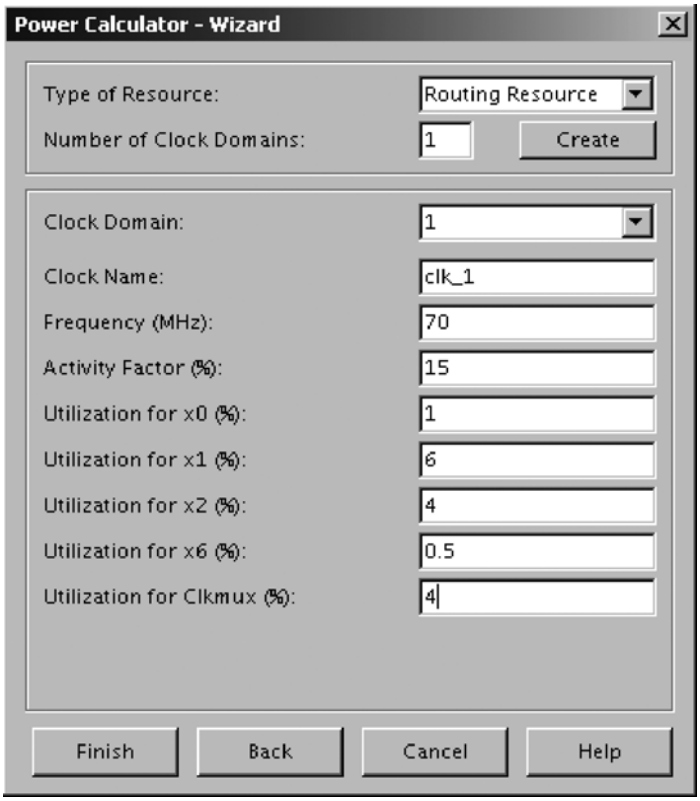


Figure 11-12. Power Calculator Wizard Mode Window - Resource Specification - I/Os

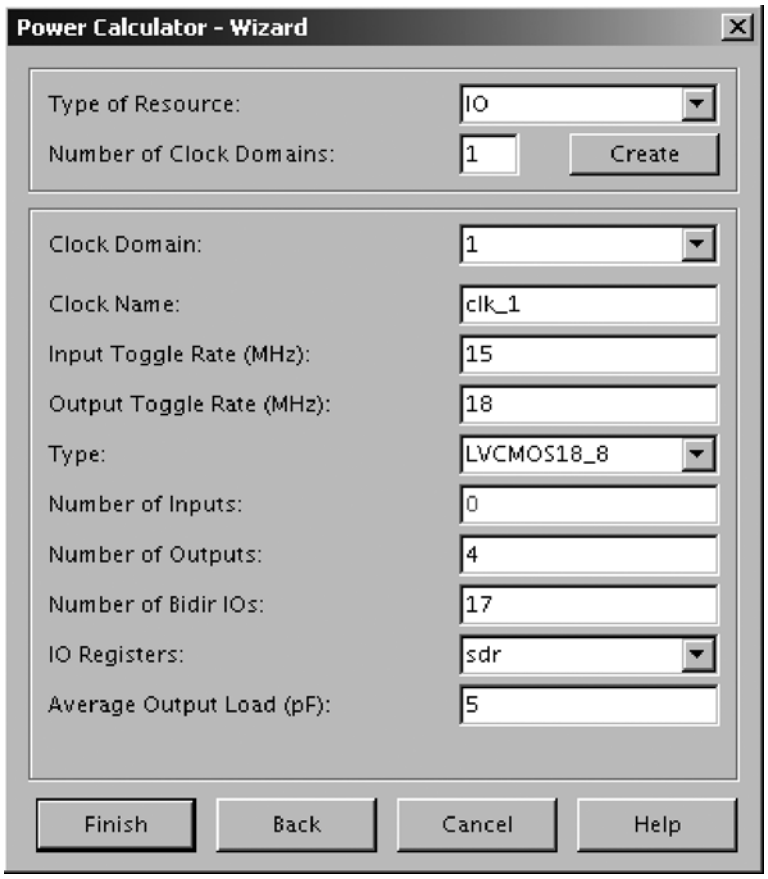
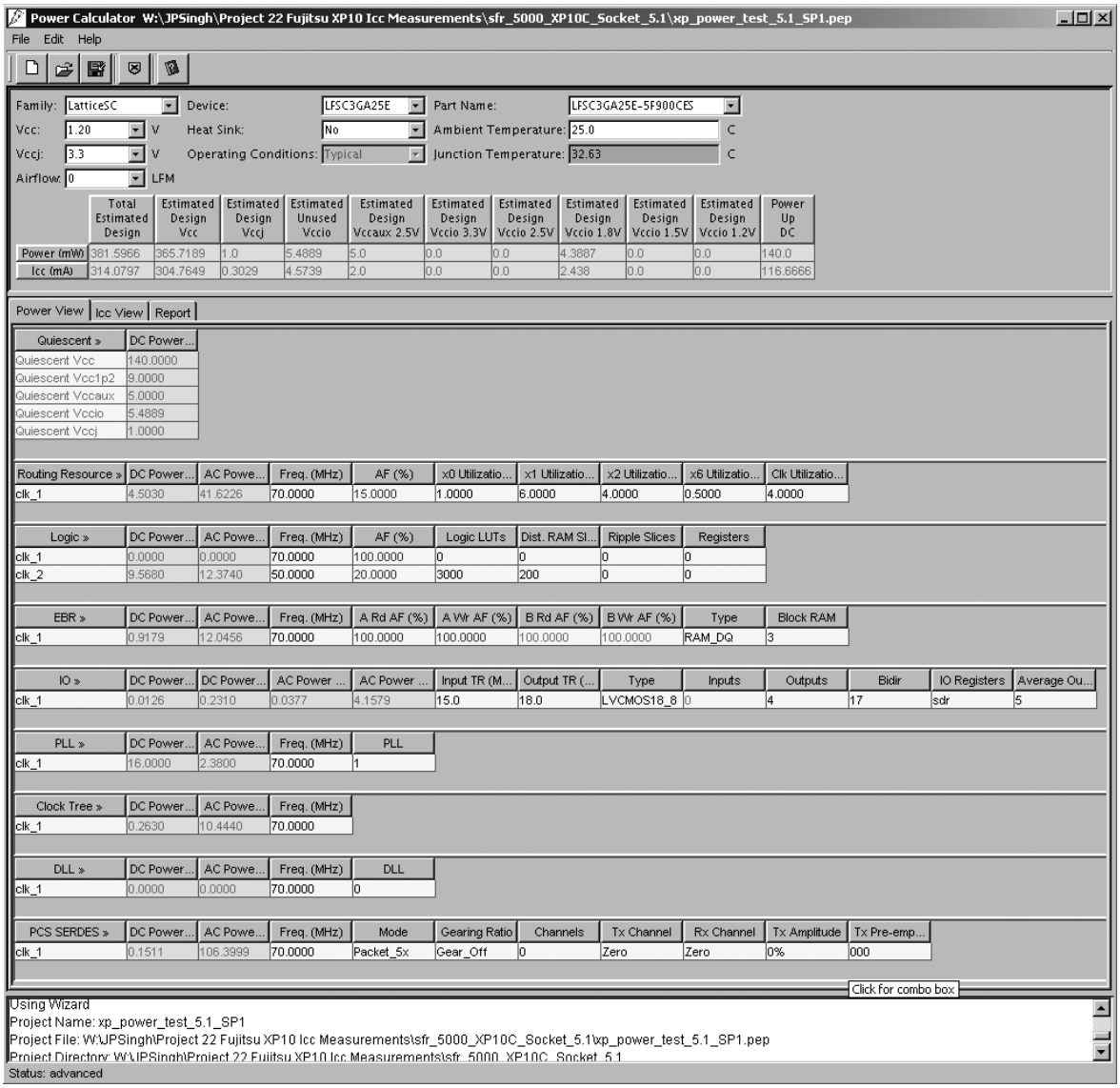


Figure 11-13. Power Calculator Wizard Mode - Main Window



Creating a New Project Without the NCD File

A new project can be created 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**. Users must provide the project name and project directory. 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 routing resources is always available in Power Calculator. Additional information such as LUT, EBR, I/O, PLL and Clock tree utilization must be added to calculate power consumption. For example, to add the logic resources, right-click on **Logic** and then select **Add** in the pop-up menu (see Figure 11-14).

Figure 11-14. Power Calculator Main Window - Adding Resources

This adds a new row for the logic resource utilization with a clock domain named clk_1.

Similarly, other resources like EBR, I/Os, PLLs and routing can be added. Each of these resources is for AC power estimation and is categorized by clock domain.

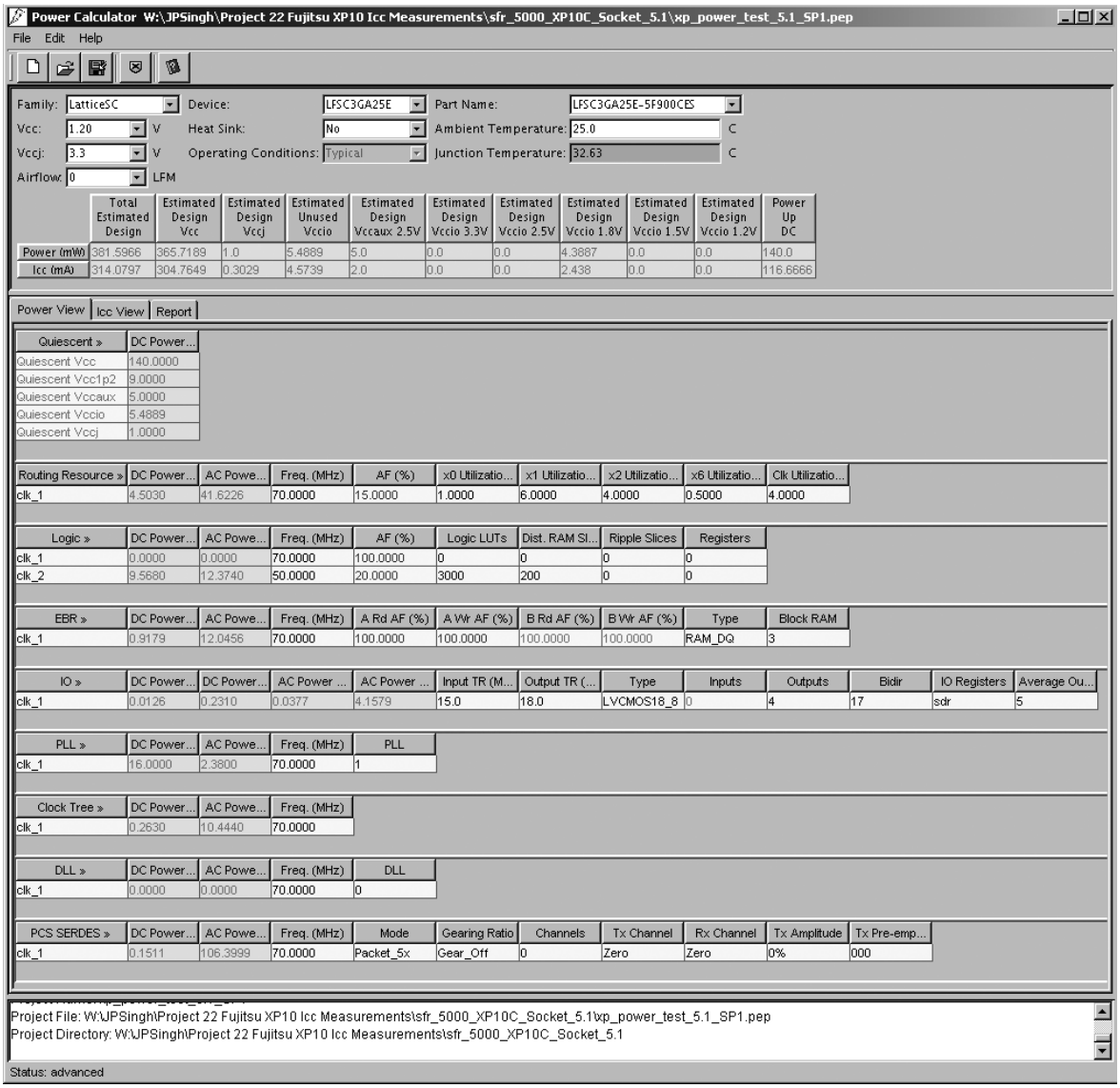
Creating a New Project with the NCD File

If the post placed and routed NCD file is available, Power Calculator can use this file to import accurate information about the design data and resource utilization and calculate the power. When Power Calculator is started, the NCD file is automatically placed in the **NCD File** option if it is available in the project directory. Otherwise, users can browse to the NCD file in the Power Calculator.

Figure 11-15. Power Calculator Start Project Window - with Post Place and Route 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 11-16.

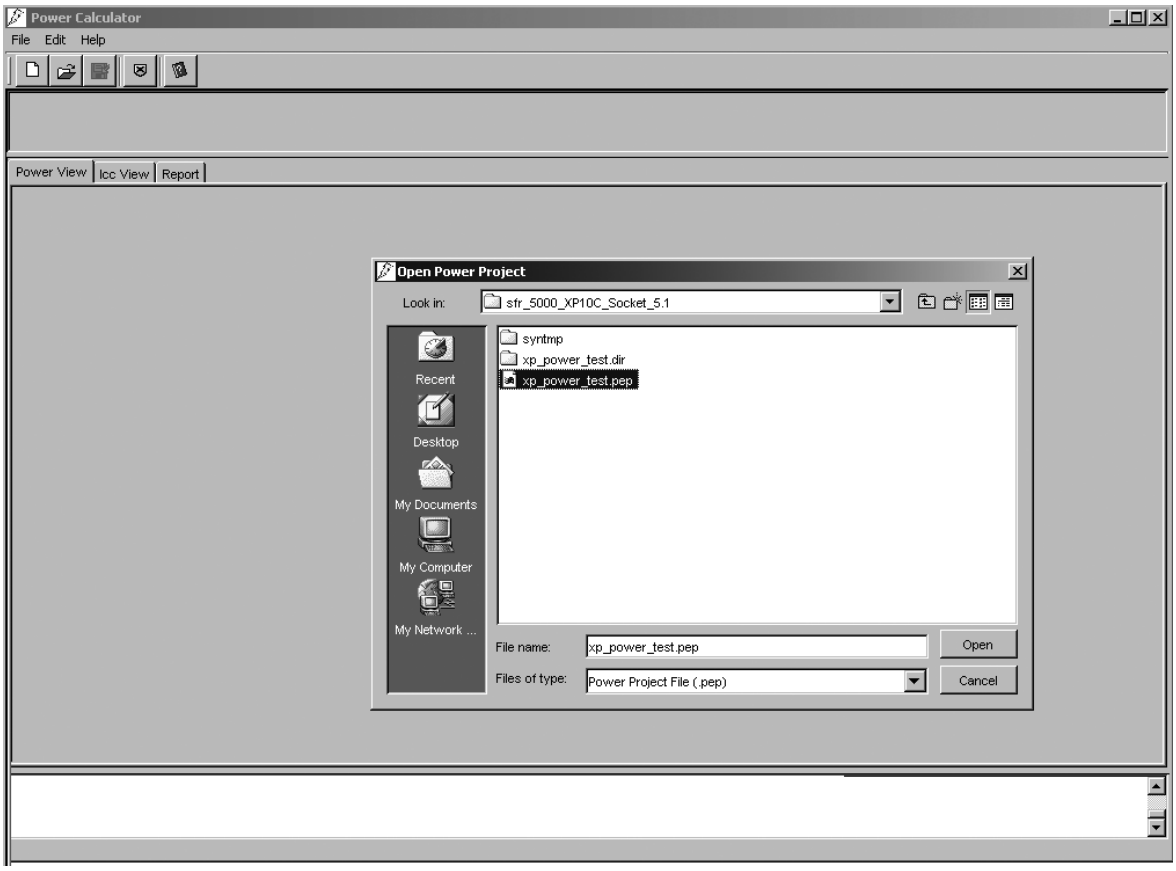
Figure 11-16. Power Calculator Main Window - Resource Utilization Picked Up from the NCD File



Open 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. This is shown in Figure 11-17.

Figure 11-17. Opening Existing Project in Power Calculator



Activity Factor Calculation

Activity Factor % (or AF %) is defined as the percentage of frequency (or time) that a signal is active or toggling the output.

Most 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 for I/Os is the I/O Toggle Rate. The AF% is applicable to the PFU, Routing, and Memory Read Write Ports, etc. The activity of I/Os is determined by the signals provided by the user (in the case of inputs) or as an output of the design (in the case of outputs). The rates at which I/Os toggle define their activity. The I/O Toggle Rate or the I/O Toggle Frequency is a better measure of their activity.

The Toggle Rate (or TR) in MHz of the Output is defined as the following equation:

Toggle Rate (MHz) = 1/2 * f_{MAX} * AF%

The users are required to provide the TR (MHz) value for the I/O instead of providing the frequency and AF% for other resources.

AF can be calculated for each routing resource, output or PFU. However, this involves long calculations. The general recommendation for a design occupying roughly 30% to 70% of the device is that the AF% used can be from 15% to 25%. This is an average value. The accurate value of an AF 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, 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 temperature can be calculated as follows.

$$T_J = T_A + \theta_{JA} * P \quad (1)$$
$$T_J = T_C + \theta_{JC} * P \quad (2)$$
$$T_J = T_B + \theta_{JB} * P \quad (3)$$

Where 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 heat sink. θ_{JB} applies when the board temperature adjacent to the package is known.

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 factors in design today is reducing the system power consumption. Low power consumption is especially important for hand-held devices and other modern electronic products. There are several design techniques that can significantly reduce overall system power consumption. These include:

1. Reducing operating voltage.
2. Operating within the specified package temperature limitations.
3. Using optimum clock frequency reduces power consumption, as the 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, which will reduce power.
4. Reducing the span of the design across the device. A more closely placed design utilizes fewer routing resources for less power consumption.
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 can 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 is less than 10%.
7. Minimizing 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 uses 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 the user-entered T_A and the power calculation of typical room temperature.
3. I/O power consumption is based on an output loading of 5pF. Users have the ability to change this capacitive loading.
4. Users can estimate power dissipation and current for each type of power supplies that are V_{CC}, V_{CCIO}, V_{CCJ}, and V_{CCAUX}.
5. The nominal V_{CC} is used by default to calculate the power consumption. A lower or higher V_{CC} can be chosen from a list of available values.
6. Users can enter an Airflow in Linear Feet per Minute (LFM) along with a Heat Sink option to calculate the Junction Temperature.
7. The default value of the I/O types for the LatticeECP2 devices 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

LatticeECP2 sysDSP Usage Guide

February 2006 Technical Note TN1107

Introduction

This technical note discusses how to access the features of the LatticeECP2™ sysDSP™ (Digital Signal Processing) Block described in the LatticeECP2 Family Data Sheet. Designs targeting the sysDSP Block can offer significant improvement over traditional LUT-based implementations. Table 12-1 provides an example of the performance and area benefits of this approach:

Table 12-1. sysDSP Block vs. LUT-based Multipliers

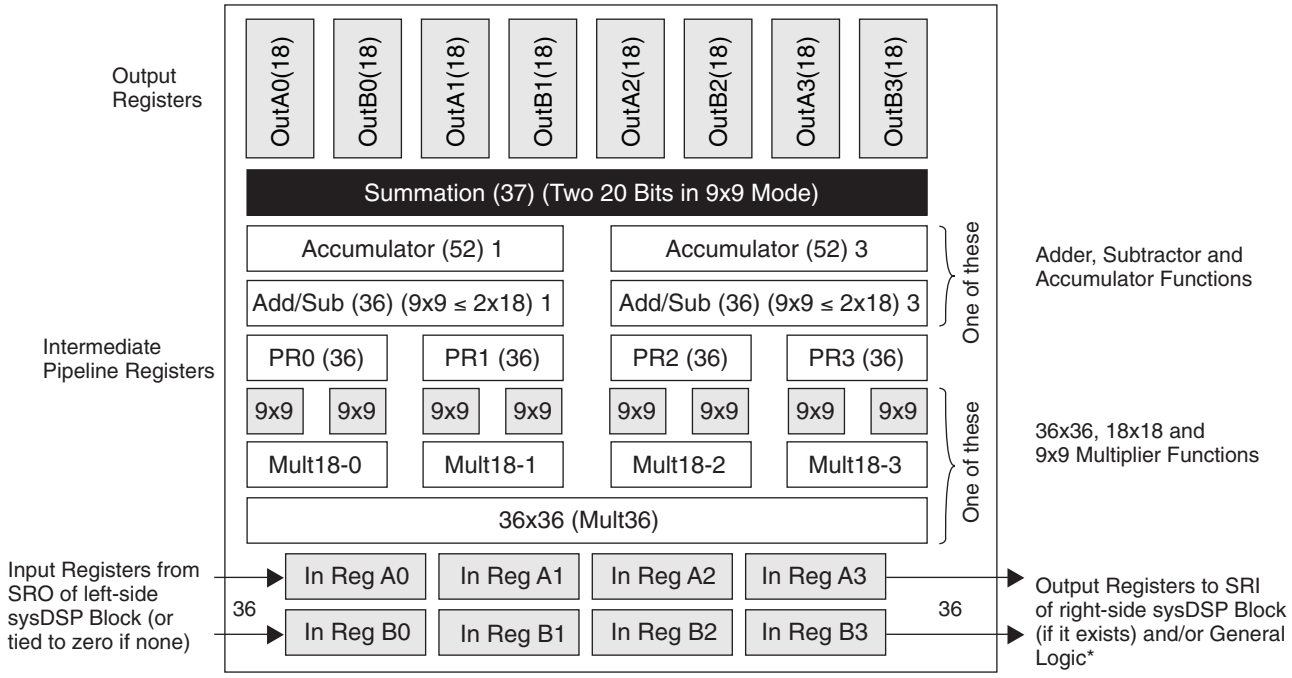
Multiplier Width	Register Pipelining	ECP2-50-7 Uses One sysDSP Block		ECP2-50-7 Uses LUTs	
		f_{MAX} (MHz)[1]	LUTs	f_{MAX} (MHz)[1]	LUTs
9x9	Input, Multiplier, Output	479	0	128	192
18x18	Input, Multiplier, Output	479	0	92	698
36x36	Input, Multiplier, Output	422	0	59	2732

1. These timing numbers were generated using the ispLEVER® design tool. Exact performance may vary with design and tool version.

sysDSP Block Hardware

The LatticeECP2 sysDSP Blocks are located in rows throughout device. Below is a block diagram of one of the sysDSP Blocks:

Figure 12-1. LatticeECP2 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 LatticeECP2 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 12-2 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 12-3, allows finer control over the register. In the Advanced mode, users can control each register with independent clocks, clock enables and resets.

Figure 12-2. MULT Mode Basic Set-up

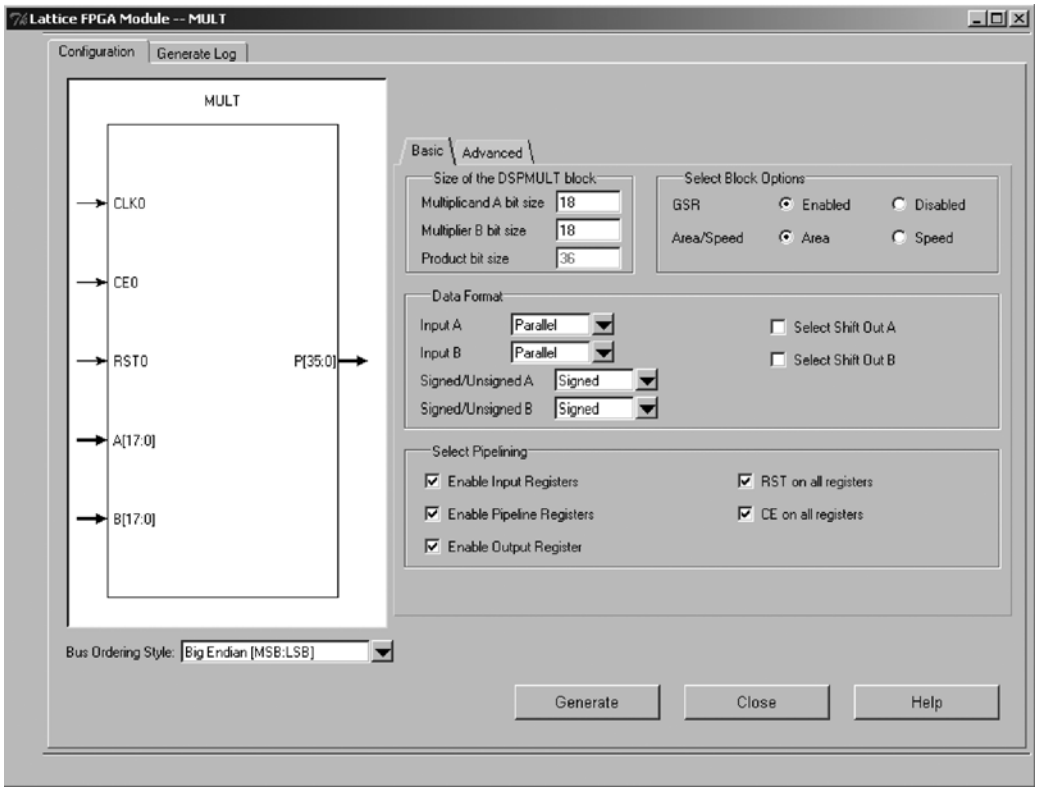
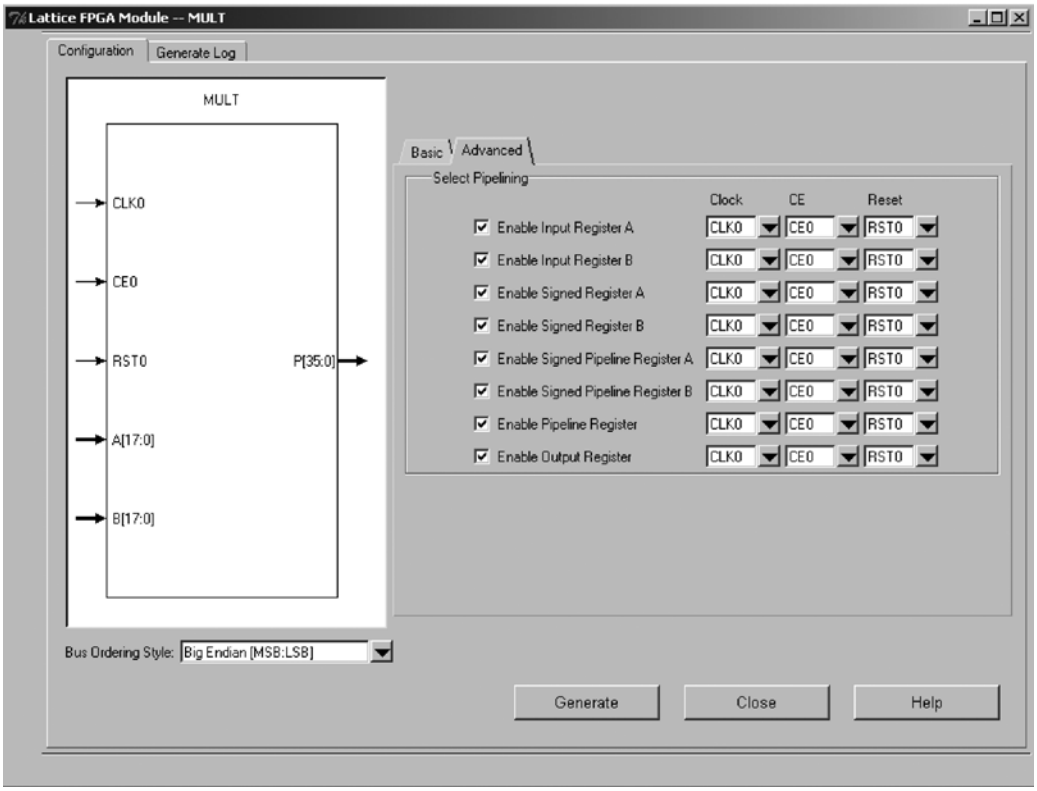


Figure 12-3. MULT Mode Advanced Set-up



MAC Module

The MAC Module configures multiply accumulate elements to be packed into the primitive MULT18X18MACB. The Basic mode, shown in Figure 12-4, 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 12-5, allows finer control over the registers. In the advanced mode, users can control each register with independent clocks, clock enables and resets.

Figure 12-4. MAC Mode Basic Set-up

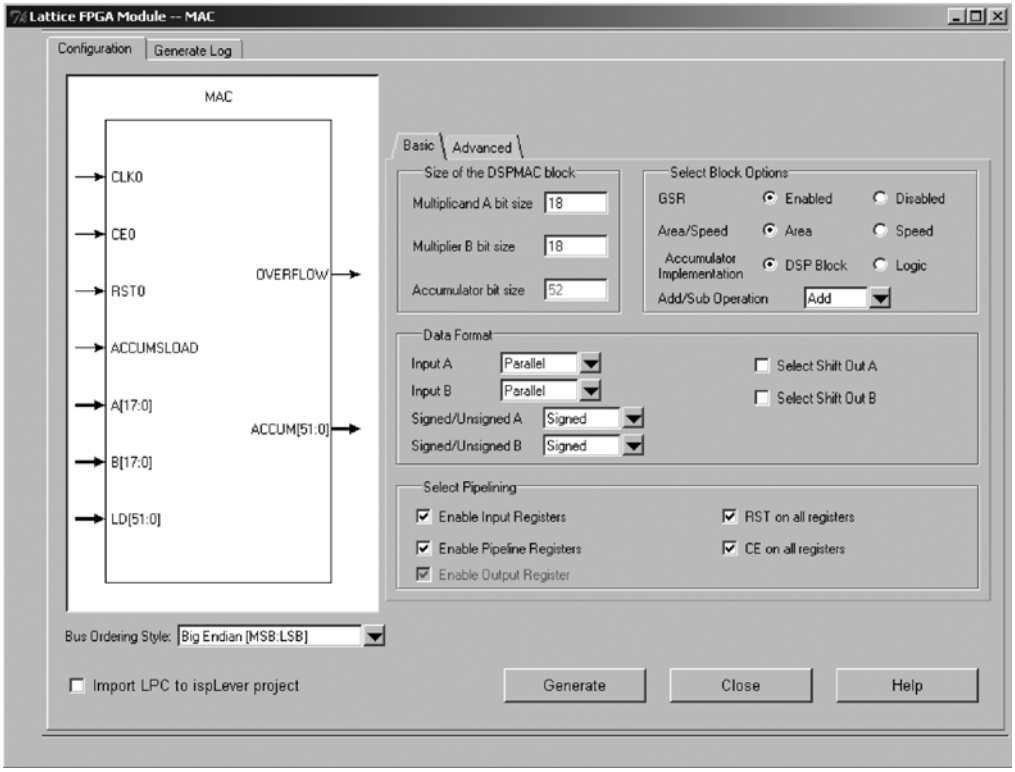
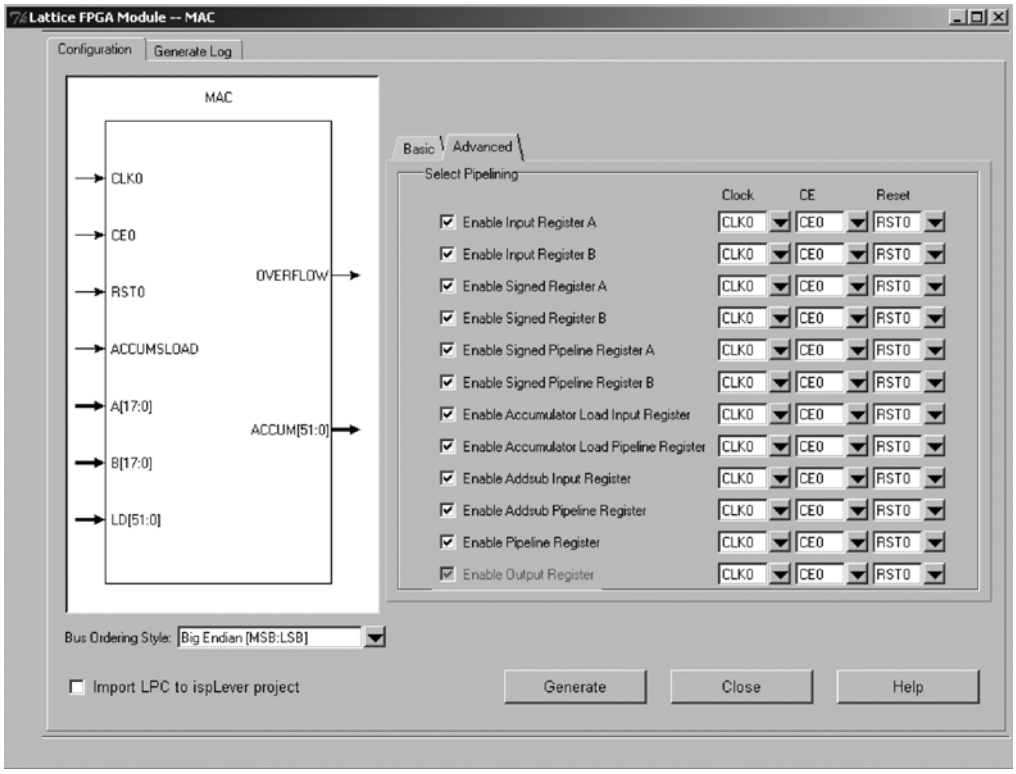


Figure 12-5. MAC Mode Advanced Set-up



MULTADDSUB Module

The MULTADDSUB GUI configures Multiplier Addition/Subraction elements to be packed into the primitives MULT18X18ADDSUBB or MULT9X9ADDSUBB. The Basic mode, shown in Figure 12-6, 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 12-7, provides finer control over the registers. In the advanced mode, users can control each register with independent clocks, clock enables and resets.

Figure 12-6. MULTADDSUB Mode Basic Set-up

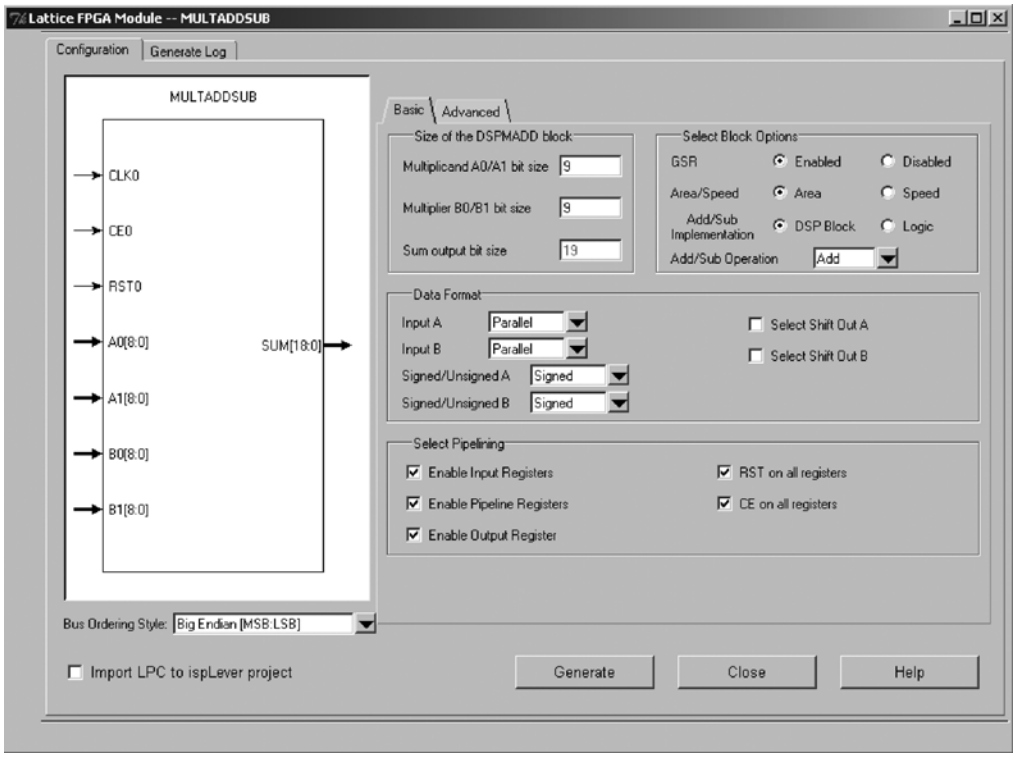
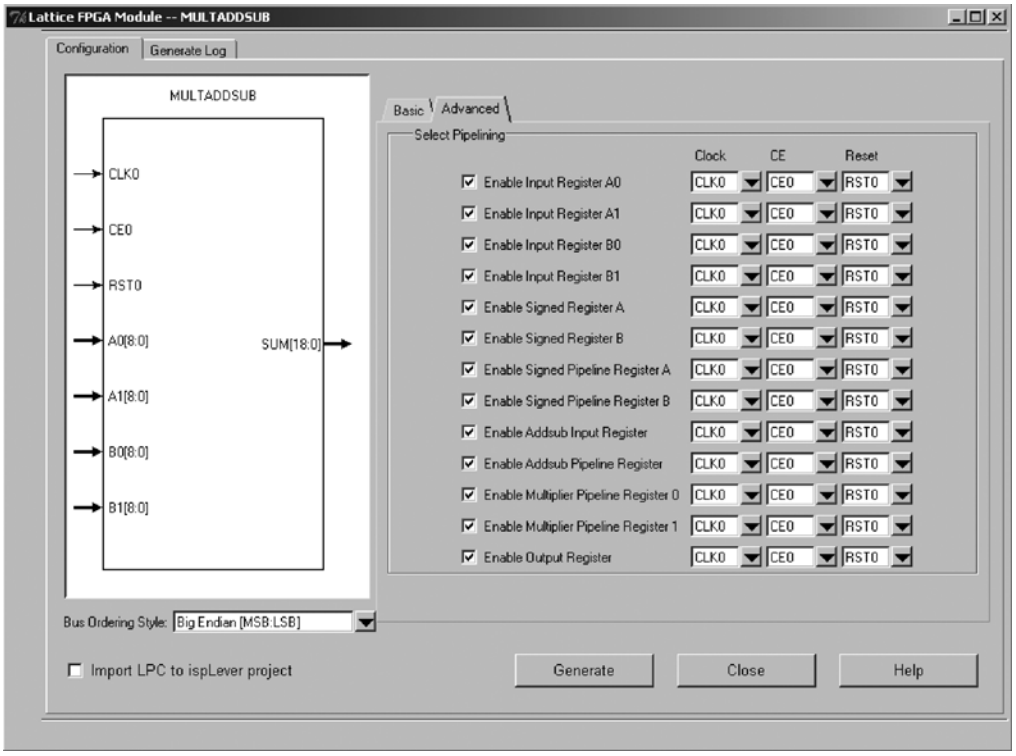


Figure 12-7. MULTADDSUB Mode Advanced Set-up



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 12-8, 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 12-9, provides finer control over the registers. In the advanced mode, users can control each register with independent clocks, clock enables and resets.

Figure 12-8. MULTADDSUBSUM Mode Basic Set-up

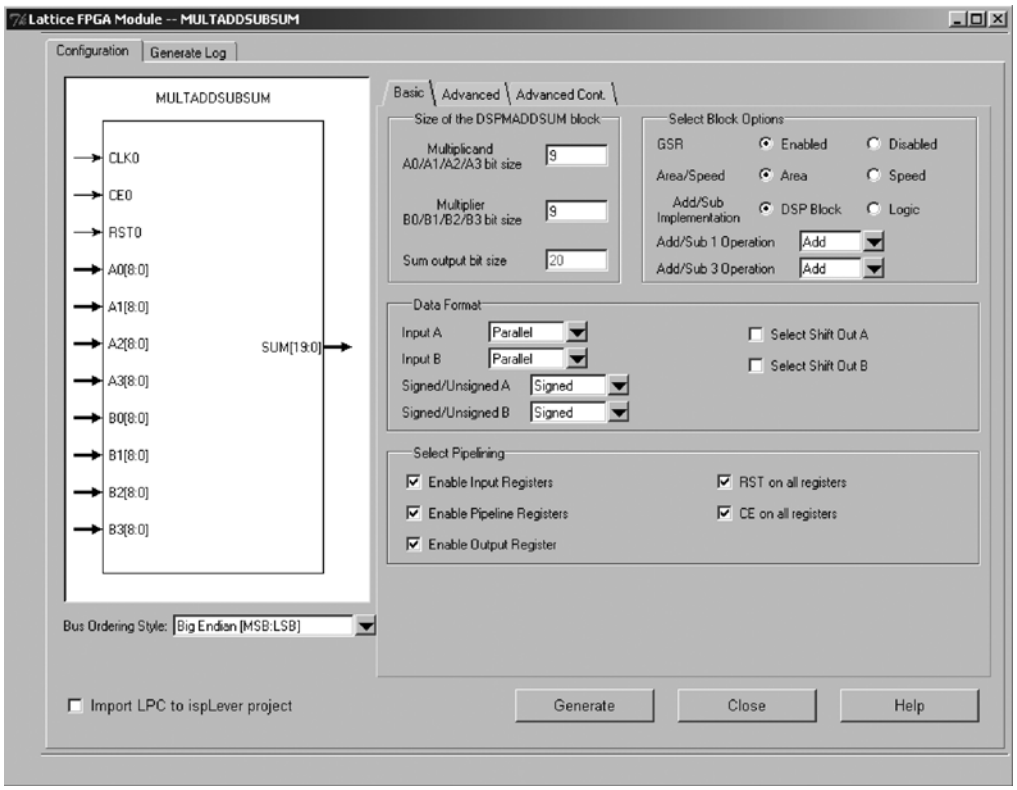


Figure 12-9. MULTADDSUBSUM Mode Advance

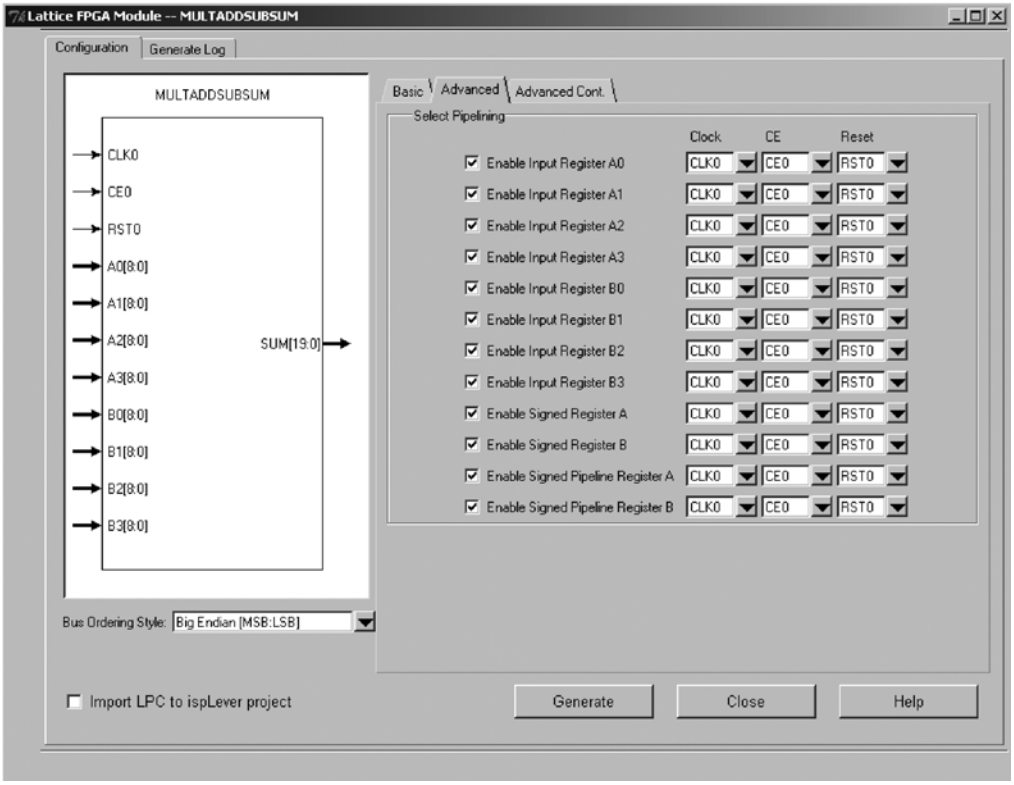
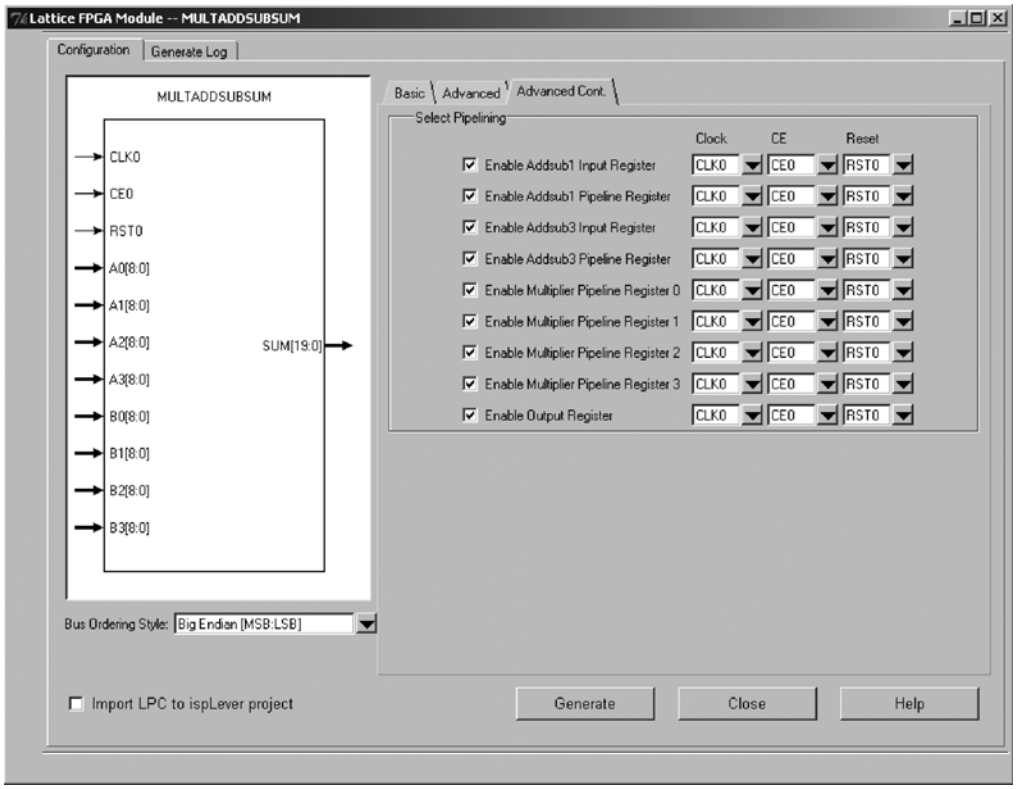


Figure 12-10. MULTADDSUBSUM Mode Advance



Targeting the sysDSP Block by Inference

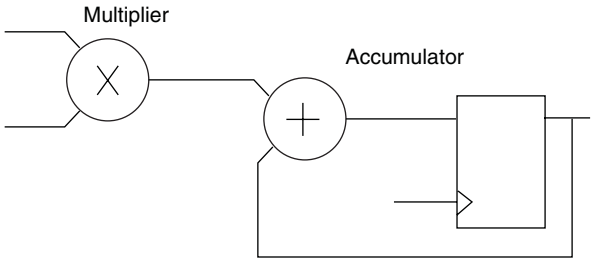
The Inferencing flow enables the design tools to infer sysDSP Blocks from a HDL design. It is important to note that when using the Inferencing flow, unless the code style matches the sysDSP Block, results will not be optimal. Consider the following Verilog and VHDL examples:

```
// 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 12-11. 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.

sysDSP Blocks in the Report File

To check configuration of the sysDSP Blocks in a design, the user can look at the MAP and Post PAR report files. The MAP Report File shows the Mapped sysDSP Components/Primitives in the 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
```

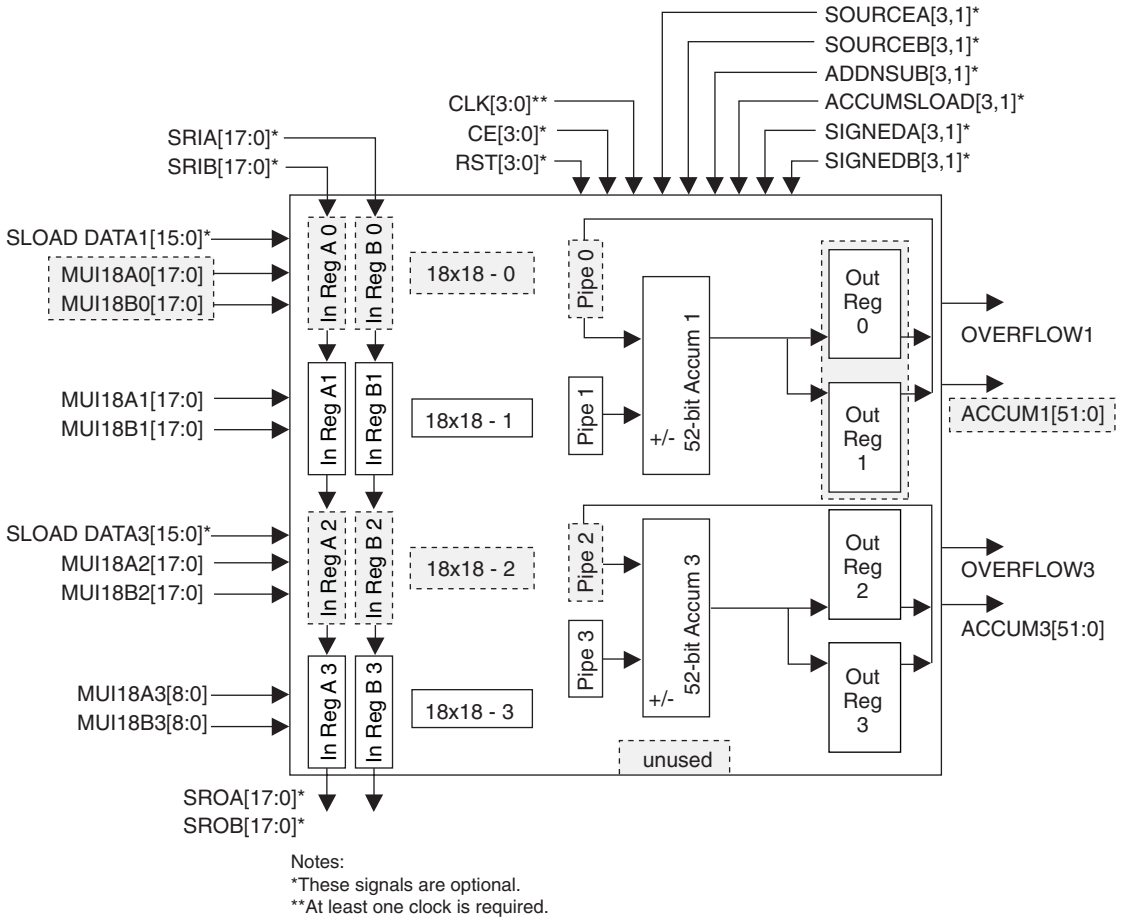
```
Data
        Input Registers         CLK     CE      RST
        ------------------------------------------
                A               CLK0    CE0     RST0
                B               CLK0    CE0     RST0

        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
```

Figure 12-12. MAC18X18MACB Packed into a sysDSP Block



Notes:
*These signals are optional.
**At least one clock is required.

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.

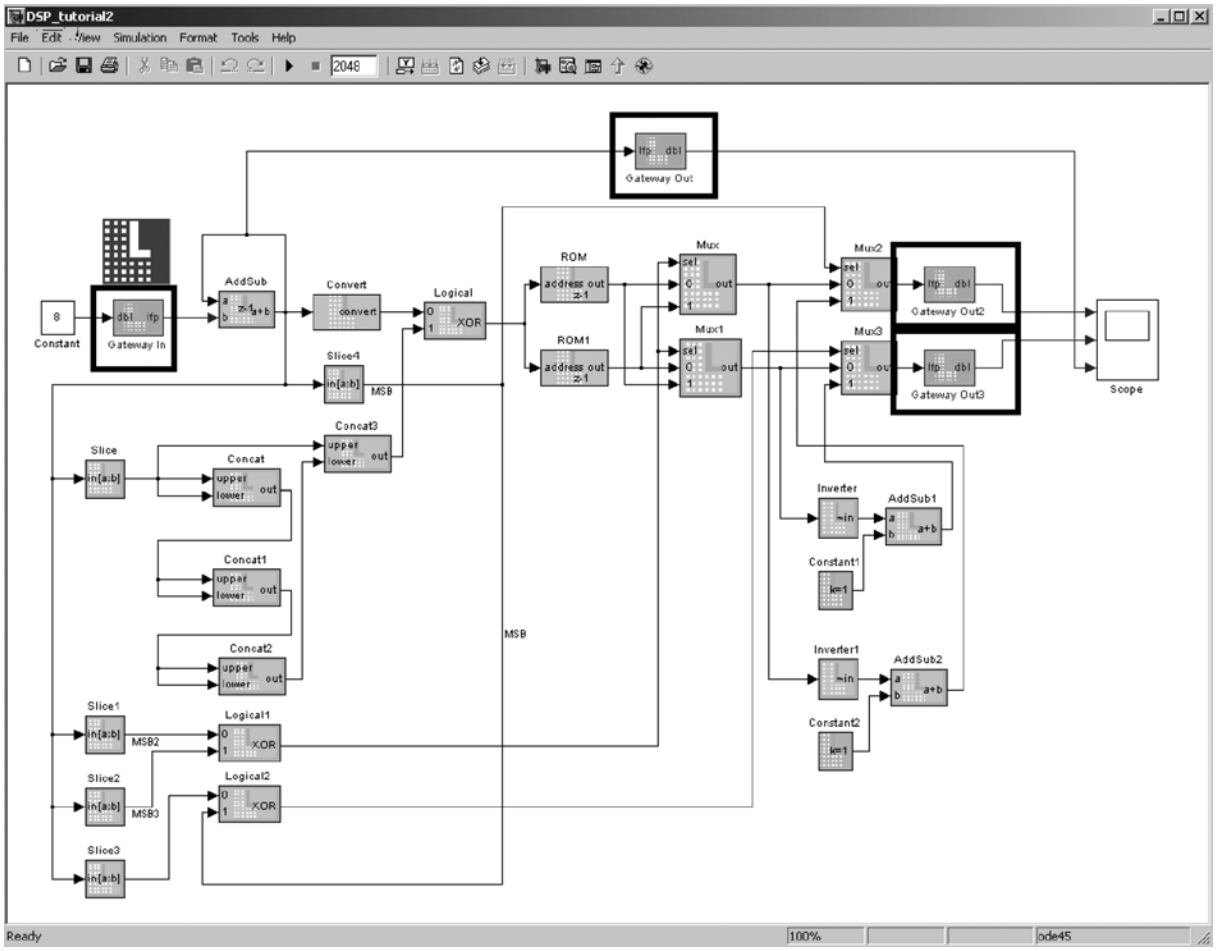
What does Lattice provide?

- 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 12-13.

  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 12-13.

Figure 12-13. 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

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 ";
```

LatticeECP2 sysCONFIG Usage Guide

February 2006 Technical Note TN1108

Introduction

The configuration memory in the LatticeECP2™ FPGAs is built using volatile SRAM; therefore, an external non-volatile configuration memory is required to maintain the configuration data when the power is removed. This non-volatile memory supplies the configuration data to the LatticeECP2 when it powers-up, or any other time the device needs to be updated.

To support multiple configuration options the LatticeECP2 supports the Lattice sysCONFIG™ interface, as well as the dedicated ispJTAG™ port. The available configuration options, or ports, are listed in Table 13-1.

Table 13-1. Supported Configuration Ports

Interface	Port
sysCONFIG	SPI
	SPIm
	Slave Serial
	Slave Parallel
ispJTAG	JTAG (IEEE 1149.1 and IEEE 1532 compliant)

This technical note covers all of the configuration options available for LatticeECP2.

General Configuration Flow

The LatticeECP2 will enter configuration mode when one of three things happens, power is applied to the chip, the PROGRAMN pin is driven low, or when a JTAG Refresh command is issued. Upon entering configuration mode the INITN pin and the DONE pin are driven low to indicate that the device is initializing, i.e. getting ready to receive configuration data.

Once the LatticeECP2 has finished initializing, the INITN pin will be driven high. The low to high transition of the INITN pin causes the CFG pins to be sampled, telling the LatticeECP2 which port it is going to configure from. The LatticeECP2 then begins reading data from the selected port and starts looking for the preamble, BDB3 (hex). All data after the preamble is valid configuration data.

When the LatticeECP2 has finished reading all of the configuration data, assuming there have been no errors, the DONE pin goes high and the LatticeECP2 enters user mode, in other words the device begins to function according to the user's design.

Note that the LatticeECP2 may also be programmed via JTAG. When programming via JTAG, the PROGRAMN, INITN, and DONE signals have no meaning, because JTAG, per the IEEE standard, takes complete control of the chip and it's I/Os.

The following sections define each configuration pin, each configuration mode, and all of the configuration options for the LatticeECP2.

Configuration Pins

The LatticeECP2 supports two types of configuration pins, dedicated and dual-purpose. The dedicated pins are used exclusively for configuration; the dual-purpose pins, when not being used for configuration, are available as extra I/O pins. 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 I/O 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 an output during configuration it must remain an output as a GPIO, if 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 can be accessed using the Preference Editor in ispLEVER®.
- The user is responsible for insuring that no internal or external logic will interfere with device configuration.

Programmable options control the direction and type of each dual-purpose configuration pin. These options are controlled via pin preferences in Lattice's ispLEVER software, or as HDL source file attributes.

The LatticeECP2 also supports ispJTAG for configuration, transparent read back, and JTAG testing. The following sections describe the function of the various sysCONFIG and JTAG pins. Table 13-2 is provided for reference.

Table 13-2. Configuration Pins for the LatticeECP2

Pin Name	I/O Type	Pin Type	Mode Used
CFG[2:0]	Input, weak pull-up	Dedicated	All
PROGRAMN	Input, weak pull-up	Dedicated	All
INITN	Bi-Directional Open Drain, weak pull-up	Dedicated	All
DONE	Bi-Directional Open Drain with weak pull-up, or Active Drive	Dedicated	All
CCLK	Input or Output	Dedicated	All
DIN	Input, weak pull up	Dual-Purpose	Serial
DOUT/CSON	Output	Dual-Purpose	Serial, Parallel, SPI, SPIm
CSN	Input, weak pull-up	Dual-Purpose	Parallel
CS1N	Input, weak pull-up	Dual-Purpose	Parallel
WRITEN	Input, weak pull-up	Dual-Purpose	Parallel
BUSY	Output, tri-state, weak pull-up	Dual-Purpose	Parallel
D[0]/SPIFASTN	Input or Output	Dual-Purpose	Parallel, SPI, SPIm
D[1:6]			Parallel
D[7]/SPID0			Parallel, SPI, SPIm
TDI	Input, weak pull-up	Dedicated	JTAG
TDO	Output, weak pull-up	Dedicated	JTAG
TCK	Input with Hysterisis	Dedicated	JTAG
TMS	Input, weak pull-up	Dedicated	JTAG

Note: Weak pull-ups consist of a current source of 30µA to 150µA. The pull-ups for sysCONFIG dedicated and dual-purpose pins track V_{CCIO8}; the pull-ups for TDI, TDO, and TMS track V_{CCJ}.

Dedicated Control Pins

The following sub-sections describe the LatticeECP2 dedicated sysCONFIG pins. These pins are powered by V_{CCIO8}.

While the device is under IEEE 1149.1 or 1532 JTAG control the dedicated programming pins have no meaning. This is because a boundary scan cell will control each pin, per JTAG 1149.1, rather than normal internal logic.

CFG[2:0]

The Configuration Mode pins, CFG[2:0], are dedicated inputs with weak pull-ups. The CFG pins are sampled on the rising edge of INITN and are used to select the configuration mode, i.e. what type of device the LatticeECP2

will configure from. As a consequence the CFG pins determine which groups of dual-purpose pins will be used for device configuration (see the right most column in Table 13-2). See Table 13-3 for a list of Configuration Modes.

Table 13-3. Configuration Modes

Configuration Mode		CFG[2]	CFG[1]	CFG[0]	D[0]/SPIFASTN
SPI	Normal (0x03)	0	0	0	Pull-Up
	Fast (0x0B)	0	0	0	Pull-Down
Reserved		0	0	1	X
SPIm	Normal (0x03)	0	1	0	Pull-Up
	Fast (0x0B)	0	1	0	Pull-Down
Reserved		0	1	1	X
Reserved		1	0	0	X
Slave Serial		1	0	1	X
Reserved		1	1	0	X
Slave Parallel		1	1	1	D0

Notes:
JTAG is always available for IEEE 1149.1 and 1532 support.
JTAG can be used to program SPI Serial Flash only if CFG[2:0] are set to SPI or SPIm.

PROGRAMN
The PROGRAMN pin is a dedicated 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 takes the device out of user mode and sets it into configuration mode. The PROGRAMN pin can be used to trigger configuration at any time. If the device is being accessed by JTAG then PROGRAMN will be ignored until the device is released from JTAG mode.

INITN
The INITN pin is a bidirectional open drain control pin. INITN is capable of driving a low pulse out as well as detecting a low pulse driven in.

When the PROGRAMN pin is driven low, a JTAG Reset command is received, or after the internal Power-On-Reset signal is released during power-up, the INITN pin will be driven low to reset the internal configuration circuitry. Once configuration initialization is complete, and the PROGRAMN pin is high, the INITN pin will go high. 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. A low to high transition on INITN causes the CFG pins to be sampled, telling the LatticeECP2 which port to use, and starts configuration.

During configuration the INITN pin becomes an error detection pin. If a Verify ID or CRC error is detected during configuration INITN will be driven low. The error will be cleared at the beginning of the next configuration.

DONE
The DONE pin is a dedicated bi-directional open drain with a weak pull-up (default), or it is an actively driven pin.

DONE goes low when INITN goes low, when INITN and PROGRAMN go high, and the internal Done bit is programmed at the end of configuration, 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. Externally delaying the wake-up sequence using the DONE pin is a good way to synchronize the wake-up of multiple FPGAs; it is also required when configuring multiple FPGAs from a single configuration device.

Sampling the DONE pin is a good way for an external device to tell if the FPGA has finished configuration. However, when using IEEE 1532 JTAG to configure SRAM the DONE pin is driven by a boundary scan cell, so the state of the DONE pin has no meaning during IEEE 1532 JTAG configuration.

CCLK
CCLK is a dedicated bi-directional pin; direction depends on whether a Master or Slave mode is selected. If a Master mode (SPI or SPIm) is selected, via the CFG pins, the CCLK pin becomes an output; otherwise CCLK is an input.

If the CCLK pin becomes an output, the internal programmable oscillator is connected to CCLK and is driven out to slave devices. CCLK will stop 100 to 500 clock cycles after the DONE pin is brought high. 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 again) on the next configuration initialization sequence.

The MCCLK_FREQ parameter (one of the global settings in the Preference Editor of ispLEVER) controls the CCLK master frequency (see Table 13-4 below). Unless changed during configuration CCLK will be 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 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.

Table 13-4. Master Clock Frequency Options

CCLK (MHz)	CCLK (MHz)	CCLK (MHz)
2.5	13	45
4.3	15	51
5.4	20	55
6.9	26	60
8.1	30	130
9.2	34	—
10.0	41	—

Note: Default is the lowest frequency, 2.5 MHz.

Table 13-5. Maximum Configuration Bits

Density	Bitstream Size (Mb)	Required Boot Memory (Mb)
ECP2-6	1.5	2
ECP2-12	2.9	4
ECP2-20	4.5	8
ECP2-35	6.3	8
ECP2-50	9	16
ECP2-70	13.3	16

Dual-Purpose sysCONFIG Pins

The following is a list of the dual-purpose sysCONFIG pins. If any of these pins are used for configuration and for user I/O, the user must adhere to the requirements listed above in the section entitled Configuration Pins.

These pins are powered by V_{CCIO8}.

DI/CSSPIN
The DI/CSSPIN dual-purpose pin is designated as DI (Data Input) for Serial configurations. DI has an internal weak pull-up. DI captures data on the rising edge of CCLK.

In SPI or SPIm mode the DI/CSSPIN becomes a low true Chip Select output that drives the SPI Serial Flash chip select.

DOUT/CSON
The DOUT/CSON pin is an output pin and has two purposes.

For serial and parallel configuration modes, when BYPASS mode is selected, this pin becomes DOUT (see Figure 13-6). When the device is fully configured a Bypass instruction in the bitstream is executed and the data on DI, or D[0:7] in the case of a parallel configuration mode, will then be routed to the DOUT pin. This allows data to be passed, serially, to the next device. In a parallel configuration mode D0 will be shifted out first followed by D1, D2, and so on.

For parallel configuration mode there is a FLOW_THROUGH option as well. When FLOW_THROUGH mode is selected this pin becomes Chip Select Out (CSON). When the device is fully configured, and the Flow-Through instruction in the bitstream is executed, the CSON pin is driven low to enable the next device. The data pins, D[0:7], are wired in parallel to each device in the chain (see Figure 13-7).

The DOUT/CSON drives out a high on power-up and will continue to do so until the execution of the Bypass/Flow Through instruction within the bitstream, or until the I/O Type is changed by the user code.

CSN and CS1N
Both CSN and CS1N are active low input pins with weak pull-ups and are used in parallel mode only. These inputs are OR'ed and used to enable the D[0:7] data pins to receive or output a byte of data.

When CSN or CS1N is high, the D[0:7], INITN, and BUSY pins are tri-stated. CSN and CS1N are interchangeable when controlling the D[0:7], INITN, and BUSY pins. Driving both CSN and CS1N high causes the LatticeECP2 to exit Bypass or Flow-Through mode and resets the Bypass register. If Bypass or Flow-Through mode will not be used then CSN or CS1N may be tied low, i.e. in this case it is only required that one of these pins be driven.

If SRAM (configuration memory) needs to be accessed using the parallel pins while the part is in user mode (the DONE pin is high) then the PERSISTENT preference must be set to ON to preserve these pins as CSN and CS1N. Note that SRAM may only be read using JTAG or Slave Parallel mode.

WRITEN
The WRITEN pin is an active low input with a weak pull-up and used for parallel mode only.

The WRITEN pin is used to determine the direction of the data pins D[0:7]. The WRITEN pin must be driven low in order to clock a byte of data into the device and driven high to clock data out of the device.

If SRAM (configuration memory) needs to be accessed using the parallel pins while the part is in user mode (the DONE pin is high) then the PERSISTENT preference must be set to ON to preserve this pin as WRITEN. Note that SRAM may only be read using JTAG or Slave Parallel mode.

BUSY/SISPI
The BUSY/SISPI pin has two functions.

In parallel configuration mode, the BUSY pin is a tri-stated output. The BUSY pin will be driven low by the device only when it is ready to receive a byte of data on D[0:7] or a byte of data is ready for reading. The BUSY pin allows the LatticeECP2 to pause transfers on the parallel port.

If SRAM (configuration memory) needs to be accessed using the parallel pins while the part is in user mode (the DONE pin is high) then the PERSISTENT preference must be set to ON to preserve this pin as BUSY. Note that SRAM may only be read using JTAG or Slave Parallel mode.

In SPI or SPIm configuration modes, the BUSY/SISPI pin becomes an output that drives control and data to the SPI Serial Flash. Control and data are output on the falling edge of CCLK.

D[0]/SPIFASTN
The D[0]/SPIFASTN pin has two functions.

In parallel mode this pin is D[0] and operates in the same way as D[1:6] below. Taken together D[0:7] form the parallel data bus, D[0] is the most significant bit in the byte. As with D[1:6], if SRAM (configuration memory) needs to be accessed using the parallel pins while the part is in user mode (the DONE pin is high) then the PERSISTENT preference must be set to ON to preserve this pin as D[0]. Note that SRAM may only be read using JTAG or Slave Parallel mode.

In SPI or SPIm mode the D[0]/SPIFASTN pin becomes an input. SPIFASTN is sampled on the rising edge of INITN. If SPIFASTN is high the LatticeECP2 will use SPI Serial Flash read op-code 03 (hex). Read op-code 03 (hex) is the standard read command used by all "25" series SPI Serial Flash. If SPIFASTN is low the LatticeECP2 will use SPI Serial Flash op-code 0B (hex). Read op-code 0B (hex) accommodates higher frequency read clocks, exact clock speeds can be found in the SPI Serial Flash manufacturer's data sheet.

Not all SPI Serial Flash support the 0B (hex) op-code, consult the manufacturer's data sheet. Care must also be taken not to exceed the signal integrity capabilities of the PCB layout.

D[1:6]
The D[1:6] pins support parallel mode only. The D[1:6] pins are tri-statable bi-directional I/O pins used for data write and read. When the WRITEN signal is low, and the CSN and CS1N pins are low, the D[1:6] pins become data inputs. When the WRITEN signal is driven high, and the CSN and CS1N pins are low, the D[1:6] pins become data outputs. If either CSN or CS1N is high D[1:6] will be tri-state.

If SRAM (configuration memory) needs to be accessed using the parallel pins while the part is in user mode (the DONE pin is high) then the PERSISTENT preference must be set to ON to preserve this pin as D[1:6]. Note that SRAM may only be read using JTAG or Slave Parallel mode.

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; corruption may be caused when these RAMs are read while being accessed by user code.

D[7]/SPID0
The D[7]/SPID0 pin has two functions.

In parallel mode this pin is D[7] and operates in the same way as D[1:6] above. Taken together D[0:7] form the parallel data bus, D[7] is the least significant bit in the byte. As with D[1:6], if SRAM (configuration memory) needs to be accessed using the parallel pins while the part is in user mode (the DONE pin is high) then the PERSISTENT preference must be set to ON to preserve this pin as D[7]. Note that SRAM may only be read using JTAG or Slave Parallel mode.

In SPI or SPIm mode the D[7]/SPID0 pin becomes an input and should be wired to the output data pin of the SPI Serial Flash. The data on SPID0 is clocked in on the rising edge of CCLK.

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 LatticeECP2 device is powered up. While the device is under 1149.1 or 1532 JTAG control the dedicated programming pins INITN, DONE, and CCLK have no meaning. This is because a boundary scan cell will control each pin, per the IEEE standard, rather than normal internal logic. While the LatticeECP2 is under JTAG control the PROGRAMN pin will be ignored.

These pins are powered by V_{CCJ}.

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. This pin should be wired to TDO of the JTAG connector, or to TDI of a downstream device in a JTAG chain. An internal pull-up resistor on the TDO pin is provided. The internal resistor is pulled up to V_{CCJ}.

TDI
The Test Data Input pin is used to shift in serial test instructions and data. This pin should be wired to TDI of the JTAG connector, or to TDO of an upstream device in a JTAG chain. 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 state machine, which loads and unloads the JTAG data and instruction registers. TCK can be stopped in either the high or low state and can be clocked at frequencies up to that indicated in the device data sheet. The TCK pin supports hysterisis; 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 resistor between TCK and ground on the PCB of 4.7 K is recommended to avoid inadvertent clocking of the TAP controller as V_{CC} ramps up.

Optional TRST
Test Reset, TRST, in not supported on the LatticeECP2.

V_{CCJ}
Having a separate JTAG V_{CC} (V_{CCJ}) pin lets the user apply a voltage level to the JTAG port that is independent from the rest of the device. Valid voltage levels are 3.3V, 2.5V, 1.8V, 1.5V, and 1.2V, but the voltage used must match the other voltages in the JTAG chain. V_{CCJ} must be connected even if JTAG is not used.

Please see *In-System Programming Design Guidelines for ispJTAG Devices* for further JTAG chain information.

Configuration and JTAG Pin Physical Description
All of the sysCONFIG dedicated and dual-purpose pins are part of Bank 8. Bank 8 V_{CCIO} determines the output voltage level of these pins, input thresholds are determined by the I/O Type selected in the ispLEVER Preference Editor (default is 3.3V LVCMOS).

JTAG voltage levels and thresholds are determined by the V_{CCJ} pin, allowing the LatticeECP2 to accommodate JTAG chain voltages from 1.2V to 3.3V.

Configuration Modes

The LatticeECP2 devices support many different configuration modes, utilizing either serial or parallel data paths. On power-up, when a JTAG Refresh command is issued, or when the PROGRAMN pin is toggled, the CFG[2:0] pins are sampled to determine the configuration mode. See Table 13-3 above for a list of available configuration modes.

The following sub-sections break down each configuration mode. For more information on each mode's options, see the section below entitled Configuration Options.

SPI Mode

The LatticeECP2 offers a direct connection to memories that support the SPI Serial Flash standard (see Table 13-6). By setting the configuration pins, CFG[2:0], to all zeros the LatticeECP2 will configure from the SPI interface. The SPI interface supports two configuration topologies:

- One FPGA configured from one SPI Serial Flash
- Multiple FPGAs configured from one SPI Serial Flash

Table 13-6. SPI Serial Flash Vendor List

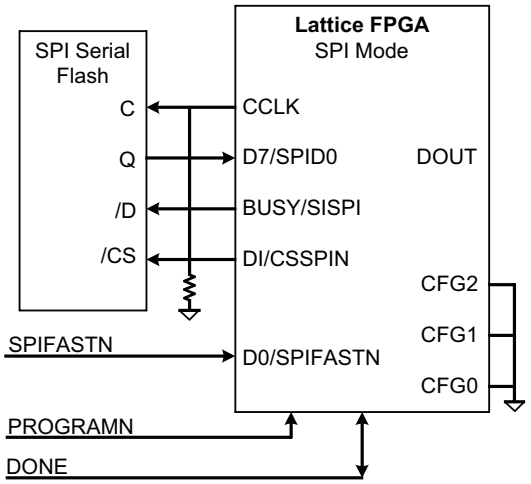
Vendor	Part Number
ST Microelectronics	M25Pxx
Winbond	W25Pxx
Silicon Storage Technology	SST25VFxx
Spansion	S25FLxx
PMC pFLASH	Pm25LVxx
Atmel	AT25Fxx

Note: This is not meant to be an exhaustive list and may be updated from time to time.

One FPGA, One SPI Flash

The simplest SPI configuration consists of one SPI Serial Flash connected to one LatticeECP2, as shown in Figure 13-1.

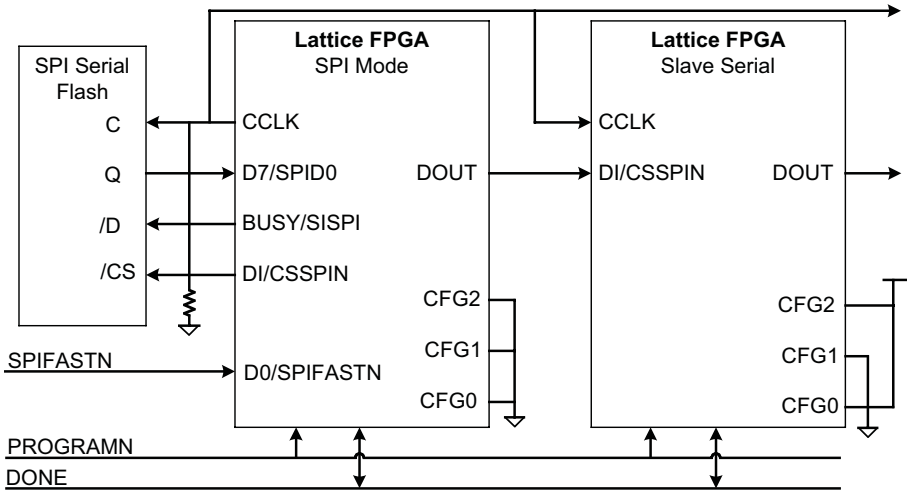
Figure 13-1. One FPGA, One SPI Serial Flash



Multiple FPGA, One SPI Flash

With a sufficiently large SPI Flash multiple FPGAs can be configured as shown in Figure 13-2. The first FPGA is configured in SPI mode; the following FPGAs are configured in Slave Serial mode.

Figure 13-2. Multiple FPGAs, One SPI Serial Flash



SPIm Mode

Externally, except for the CFG pins, SPIm mode looks just like SPI mode, they support the same configuration topologies (see Figures 13-1 and 13-2), use the same SPI Serial Flash devices, and are wired to the FPGA in the same way. Internally the two modes are treated differently.

SPI mode treats the SPI Serial Flash as a single block of storage starting at address zero. Even if the SPI Serial Flash is storing configuration data for multiple FPGAs, as in Figure 13-2, the data is still treated as a single block (separated by Bypass or Flow-Through instructions). SPIm supports dual configuration (or boot) images, here referred to as a "golden" image and a primary image.

A golden image is used when there is the possibility that corrupt data could be inadvertently loaded into the SPI Serial Flash, such as during remote updates. For instance, if the Flash erase or program procedure is interrupted, perhaps due to a power failure, then the Flash will contain corrupt data and the system could be rendered inoperable. Ideally the FPGA should detect that the data is corrupt and boot from a known good, or golden, boot image. This is exactly what SPIm does.

The golden image is stored at the beginning of the Flash address space; the updatable, or primary, image is above the golden image. During configuration, if the FPGA detects data corruption in the primary image, it will automatically reboot from the golden image. Each time the FPGA powers up, the PROGRAMN pin is toggled, or a JTAG Refresh command is issued, it will try to configure from the primary image first. Note that if the LatticeECP2 detects that the data in the golden image is corrupt as well INITN will be driven low and the part will stop trying to configure.

Dual Boot Image Setup

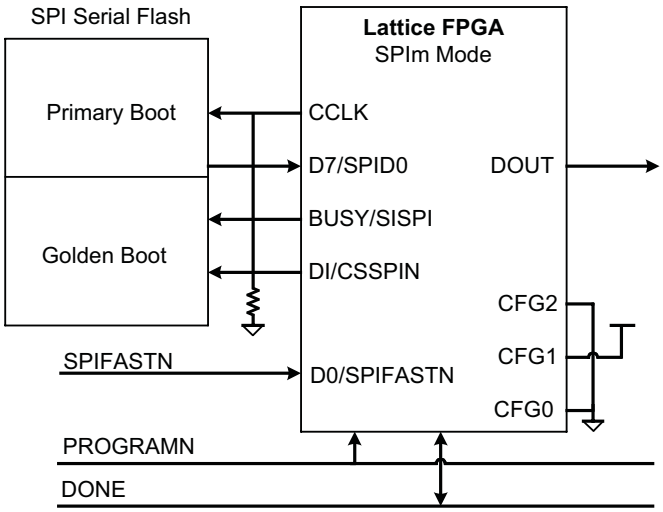
In order to use dual configuration files the files must first be properly stacked and referenced within the SPI Serial Flash. Lattice's ispVM® software makes this easy.

First, create the desired files using ispLEVER. Depending on the application, these files might be the same design or different designs. There is nothing special about these files, in other words they contain all of the information needed to fully configure the FPGA.

Next open Universal File Writer (UFW) in ispVM. In the drop down list on the tool bar select "PROM File". In the left pane, double click on "Input Data File" and select one of the files created with ispLEVER. Double click on "Input Data File" again and select the other file (the order of the files doesn't really matter). Under "PROM File Setting" right click on "Output Format" and select the desired file type. Go up and double click on "Output Data File", select the output file name. Go down and right click on "Dual Boot", select "Yes". Right click on "Golden File:" and select the desired file. Right click on "Primary File:" and select the desired file. Click on the Generate button on the toolbar to create the downloadable file. Go back to ispVM and program the SPI Serial Flash as usual.

At power-up, when the PROGRAMN pin is toggled, or when a JTAG Refresh command is issued the LatticeECP2 reads the primary file from SPI Serial Flash. If an error is found then the LatticeECP2 will re-boot automatically, reading the data from the golden file instead of the primary file. Note that if an error is found in the golden file the LatticeECP2 will drive the INITN pin low and stop trying to configure.

Figure 13-3. SPIm, Dual Configuration Images



Programming SPI Serial Flash

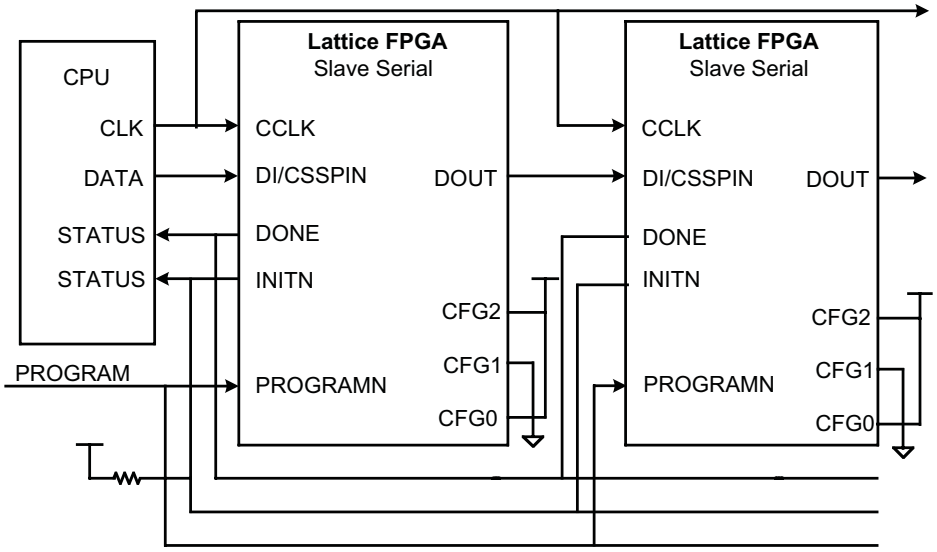
The LatticeECP2 contains dedicated hardware that allows JTAG to access the SPI port, allowing ispVM, embedded hardware, or ATE equipment to program the Flash while it is on the board. In order to program SPI Serial Flash using JTAG, the CFG pins must be set to SPI or SPIm (see Table 13-3). Please refer to ispVM's help facility for more information.

Slave Serial Mode

In Slave Serial mode the CCLK pin becomes an input, receiving the clock from an external device. The LatticeECP2 accepts data on the DI pin on the rising edge of CCLK. Slave Serial only supports writes to the FPGA, it does not support reading from the FPGA.

After the device is fully configured, if the Bypass option has been set, any additional data clocked into DI will be presented to the next device via the DOUT pin, as shown in Figure 13-4.

Figure 13-4. Serial Mode Daisy Chain

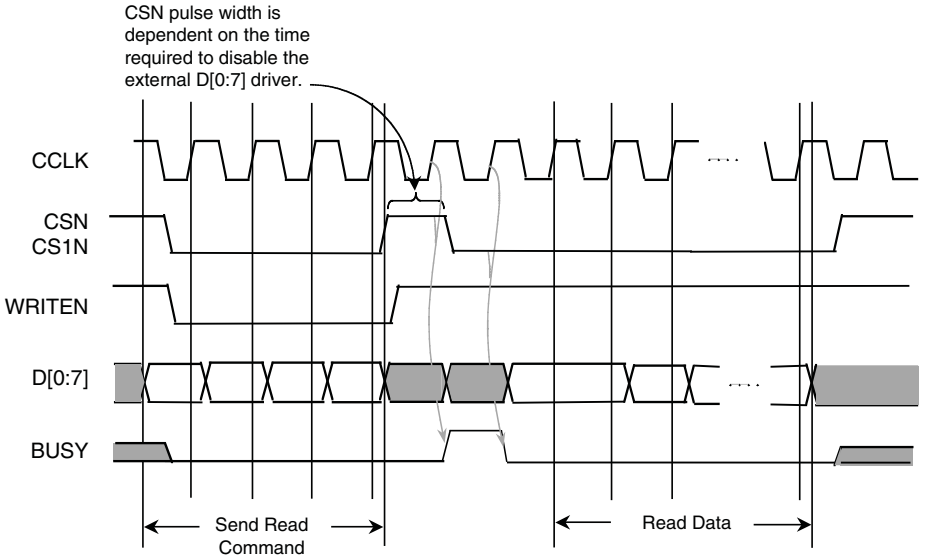


Slave Parallel Mode

In Slave Parallel mode a host system sends the configuration data in a byte-wide stream to the LatticeECP2. The CCLK, CSN, CS1N, and WRITEN pins are driven by the host system. WRITEN, CSN, and CS1N must be held low to write to the device; data is input from D[0:7]. D0 is the MSb and D7 is the LSb.

Slave Parallel mode can also be used for readback of the internal configuration. By driving the WRITEN pin low, and CSN and CS1N low, the device will input the readback instructions on the D[0:7] pins; WRITEN is then driven high and read data is output on D[0:7] (see Figure 13-5). In order to support readback the PERSISTENT bit in ispLEVER's Preference Editor must be set to ON.

Figure 13-5. Parallel Port Read Timing Diagram



The host sends the preamble, BDB3 (hex), and then sends the read command. The LatticeECP2 sends read data on D[0:7], driving BUSY high as needed to pause data flow. For an example bitstream see Table 13-7. Table 13-8 lists the various read commands. Note that the sample bitstream and the list of read commands are for reference

only; to help the user better understand the flow. The actual bitstream, containing the read commands, is created by ispLEVER and ispVM, the host toggles the control signals and sends the bitstream.

Table 13-7. Parallel Port Read Bitstream Example

Frame	Contents	Description
Header	1111...1111	2 Dummy Bytes
	10111101 10110011	2-byte Preamble (BDB3)
Verify ID		8 bytes of command and data
Reset Address		4 bytes of command and data
Read Increment		4 bytes of command and data

Table 13-8. Parallel Port Read Commands

Command	32-bit Opcode	Function
Reset Address	62xxxxxx	Reset address register to point to the first data frame
Read Increment	01vvvvvv	Read back the configuration memory frame selected by the address register and post increment the address
Read Usercode	03xxxxxx	Read the content of the USERCODE register
Read Ctrl Reg 0	04xxxxxx	Read the content of Control Register 0
Read CRC	06xxxxxx	Read CRC register content
Read ID Code	07xxxxxx	Read ID code
NO OP	FFxxxxxx	No operation

Note: x = don't care, v = variable.

Slave Parallel mode can support two types of overflow, Bypass and Flow-Through. After the first device has received all of it's configuration data, and the Bypass command is detected in the bitstream, the data presented to the D[0:7] pins will be serialized and bypassed to the DOUT pin (see Figure 13-6). If the Flow-Through command is detected in the bitstream, instead of the bypass command, the CSON signal will drive the following parallel mode device's chip select low as shown in Figure 13-7. If either type of overflow is active, driving both the CSN and CS1N pins high will reset overflow, i.e. take the device out of overflow.

Figure 13-6. Slave Parallel with Bypass Option

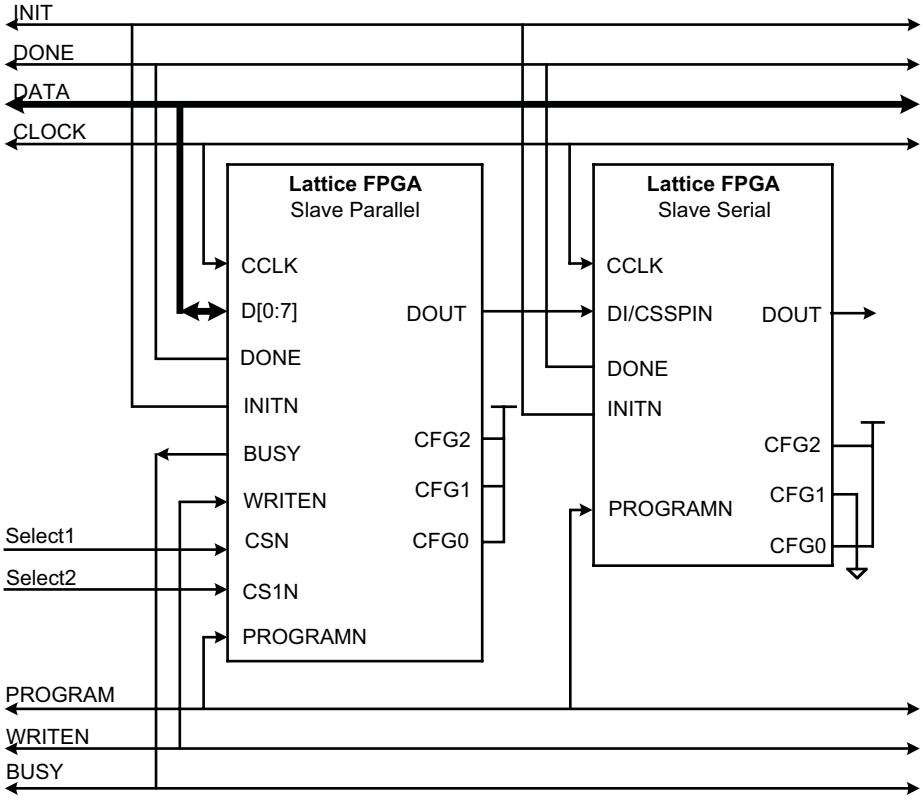
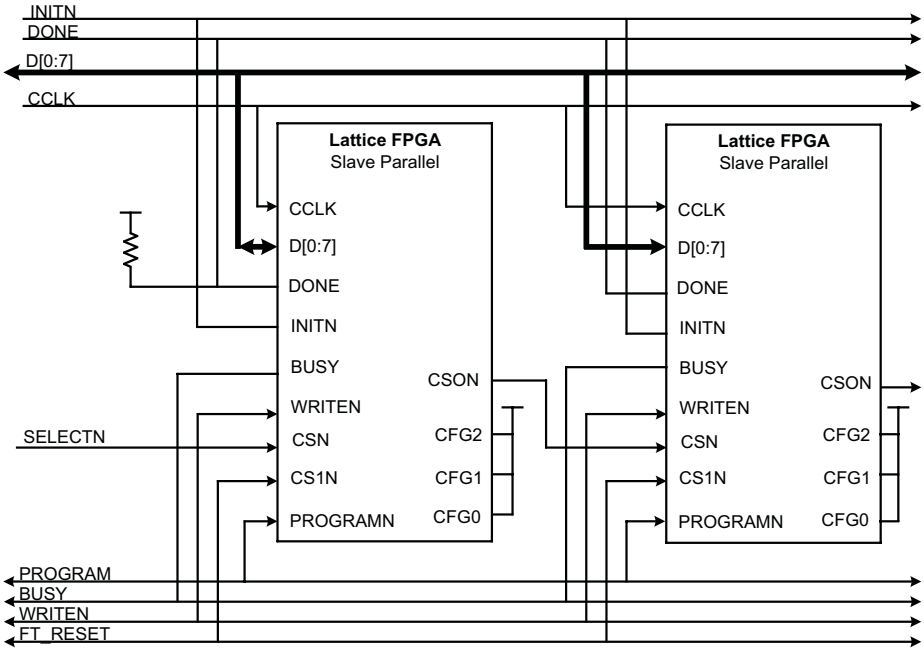


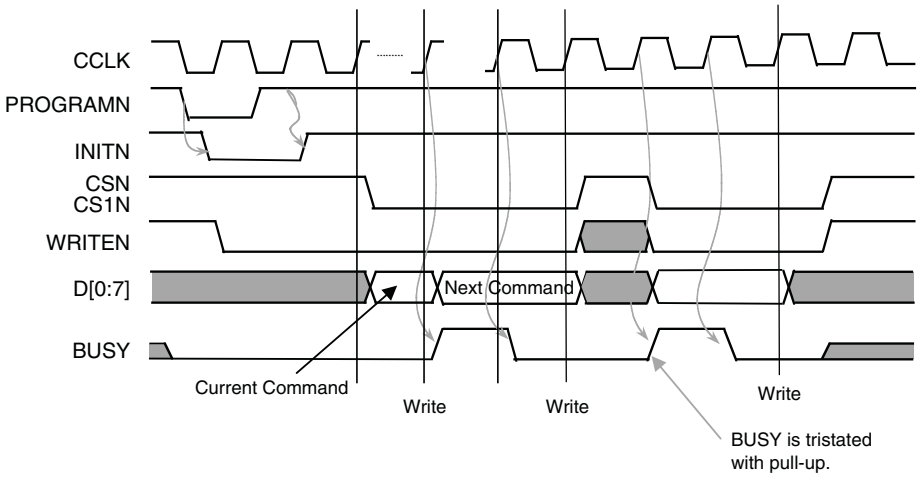
Figure 13-7. Slave Parallel with Flow-Through



To support asynchronous configuration, where the host may provide data faster than the FPGA can accept it, Slave Parallel mode can use the BUSY signal. By driving the BUSY signal high the Slave Parallel device tells the host to

pause sending data. Please note that all data and control are still synchronous with CCLK, asynchronous refers to the ability to throttle the data transfer using BUSY. See Figure 13-8.

Figure 13-8. Parallel Port Write Timing Diagram



The CSN or CS1N signal can be used to temporarily stop the write process by setting either signal to a high state. The LatticeECP2 will resume configuration when both CSN and CS1N are set low again (see Figure 13-8).

ispJTAG Mode

The LatticeECP2 device can be configured through the ispJTAG port using either Bitstream-Burst or IEEE 1532 mode. The JTAG port is always on and available, regardless of the configuration mode selected.

Bitstream-Burst

Bitstream-Burst can be thought of as serial configuration using the JTAG port. The data file used for Bitstream-Burst is the same as a file used for a sysCONFIG Serial mode configuration, in other words there is a header, a preamble, and configuration data.

Note that PROGRAMN, INITN, and DONE have no meaning while using JTAG, therefore these pins should not be used to indicate configuration status when using Bitstream-Burst.

IEEE 1532

Besides Bitstream-Burst the LatticeECP2 can also be configured through JTAG using the IEEE 1532 Standard. IEEE 1532 configuration files contain JTAG commands, as well as the configuration data. IEEE 1532 files, including ISC, SVF, and VME, can be created using ispVM's Universal File Writer (UFW). These files can be used by ispVM or by third party ATE equipment. These files can also be used in embedded situations, where an on-board processor provides the data while controlling the JTAG signals (this is called ispVM Embedded, more information can be found in ispVM's help facility).

During 1532 configuration the Boundary Scan cells take control of the LatticeECP2 I/Os. The Boundary Scan cells will usually drive the I/Os to a tri-state level but this can be controlled and even customized using ispVM. This customization can be saved in the IEEE 1532 configuration data file.

Note that PROGRAMN, INITN, and DONE have no meaning while in 1532 mode, therefore these pins should not be used to indicate configuration status when using IEEE 1532 JTAG.

Transparent Read Back

The ispJTAG transparent read back mode allows the user to read the content of the device while the device remains fully functional. All I/O, as well at the non-JTAG configuration pins, remain under internal logic control during a Transparent Read Back. The device enters the Transparent Read Back mode through a JTAG instruction. The

user must ensure that Transparent Read Back does not access EBR or distributed RAM at the same time internal logic is accessing these resources or corruption of the RAM may occur.

Boundary Scan and BSDL Files
The LatticeECP2 BSDL files can be found on the Lattice Semiconductor web site. The boundary scan ring covers all of the I/O pins, as well as the dedicated and dual-purpose sysCONFIG pins. Note that PROGRAMN, CCLK, and the CFG pins are observe only (BC4) boundary scan cells.

Configuration Options

Several configuration options are available for each configuration mode.

- When daisy chaining multiple FPGA devices an overflow option is provided for serial and parallel configuration modes
- When using SPI or SPIm mode, the master clock frequency can be set
- A security bit can be set to prevent SRAM readback
- The bitstream can be compressed
- The Persistent option can be set
- Configuration pins can be protected
- DONE pin options can be selected

By setting the proper parameter in the Lattice 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.

Bypass Option
The Bypass option can be set by using ispLEVER's Bitgen properties, or a chain of bitstreams can be assembled and Bypass set using ispVM. The Bypass option can be used in parallel and serial mode daisy chains.

When the first device completes configuration, and a Bypass command is input from the bitstream, any additional data coming into the FPGA configuration port will overflow serially on DOUT. This data is applied to the DI pin of the next device (downstream devices must be set to slave serial mode).

In serial configuration mode the Bypass option connects DI to DOUT via a bypass register. The bypass register is initialized with a '1' at the beginning of configuration and will stay at that value until the Bypass command is executed. In parallel configuration mode the Bypass option causes the excess data coming in on D[0:7] to be serially shifted to DOUT. The serialized data is shifted to DOUT through a bypass register. D0 will be shifted out first followed by D1, D2, and so on. Once the Bypass option starts the device will remain in Bypass until the Wake-up sequence completes. In parallel mode, if Bypass needs to be aborted, drive both CSN and CS1N high, this acts as a Bypass reset signal.

Flow-Though Option
As with Bypass, Flow-Through can be found in the software's Bitgen properties. The Flow-Through option can be used with parallel daisy chains only.

When the first device completes configuration, and a Flow-Through command is input from the bitstream, the CSON pin is driven low. In addition to driving CSON low, Flow-Through also tri-states the device's D[0:7] and BUSY pins in order to avoid contention with the other daisy chained devices. Once the Flow-Through option starts the device will remain in Flow-Through until the Wake-up sequence completes. If Flow-Through needs to be aborted drive both CSN and CS1N high, this acts as a Flow-Through reset signal.

Master Clock
If the CFG pins indicate an SPI or SPIm mode 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 change the Master Clock frequency by setting the MCCLK_FREQ global preference in the Lattice ispLEVER Preference Editor. 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. See Table 13-4 above for available options.

Configuration time is computed by dividing the maximum number of configuration bits, as given in Table 13-5 above, by the Master Clock frequency.

Security Bit
Setting the CONFIG_SECURE option to ON prevents readback of the SRAM from JTAG or the sysCONFIG pins. When CONFIG_SECURE is set to ON the only operations available are erase and write. The security fuse is updated as the last operation of SRAM configuration. If a secured device is read it will output all zeros.

The CONFIG_SECURE option is accessed via the Preference Editor in ispLEVER, the default is OFF.

Compress Bitstream
Setting the global COMPRESS_CONFIG option to ON in ispLEVER's Preference Editor will cause the software to generate a compressed bitstream. The LatticeECP2 will automatically decompress the bitstream as it comes into the device. The actual amount of compression varies according to the data pattern in the uncompressed bitstream. Though unlikely, it is theoretically possible for the compressed bitstream to be larger than the uncompressed bitstream.

Compressing the bitstream can result in faster configuration. The default setting is OFF.

Persistent Option
The PERSISTENT Option is set using ispLEVER's Preference Editor (default is OFF). PERSISTENT serves two purposes.

Setting PERSISTENT ON tells the software's place and route tools that it may not use any of the sysCONFIG pins associated with the parallel port (all of the dual-purpose pins except DI).

Setting PERSISTENT ON also sets a hardware fuse. So, not only are the pins reserved in software, they are also reserved in hardware.

PERSISTENT is set to ON only when the user wants to be able to read the SRAM configuration memory using the Slave Parallel port. In order to perform a read using the parallel port the user must first send a read command, setting PERSISTENT ON allows the parallel port to listen for this command while in user mode (the DONE pin is high). If the design does not require this function, the PERSISTENT option should be set to OFF.

Configuration Mode
Just as the CFG pins tell the hardware which port to configure from, the CONFIG_MODE option tells the software which port will be used. CONFIG_MODE allows the user to protect dual-purpose sysCONFIG pins. For example setting CONFIG_MODE to SPI will keep the Place and Route tools from using the SPI pins as general purpose I/O. The user, however, is still free to assign these pins as GPIO, but a warning will be generated as a reminder that there are certain precautions (see the section above entitled Configuration Pins).

Available options are None, JTAG, SPI, SPIm, Slave Serial, and Slave Parallel. The default is Slave Serial.

DONE OD, DONE EX
During configuration the DONE pin is low. Once configuration is complete, indicated by the setting of the internal Done bit, the device wake-up sequence takes place and then the DONE pin goes high. Under most circumstances this flow is exactly what is needed, however, if there are several devices in one configuration chain, delay of the wake-up sequence may be desirable in order to "synchronize" the wake-up of all devices in the chain. There are two options in the Preference Editor that allow for this synchronization.

DONE OD defaults to ON, DONE OD ON forces the DONE pin type to be open- drain. When connecting multiple DONE pins together all of the pins should be open- drain, however it may be advantageous to have an actively driven DONE pin on the last device. Setting DONE OD to OFF makes the DONE pin an actively driven pin, rather than open-drain.

DONE EX defaults to OFF. Setting DONE EX to ON will cause LatticeECP2 to sample the DONE pin and, if the DONE pin is held low externally, delay the wake-up sequence. Setting DONE EX to OFF will cause the device to wake-up as soon as the internal Done bit is set.

Device Wake-Up

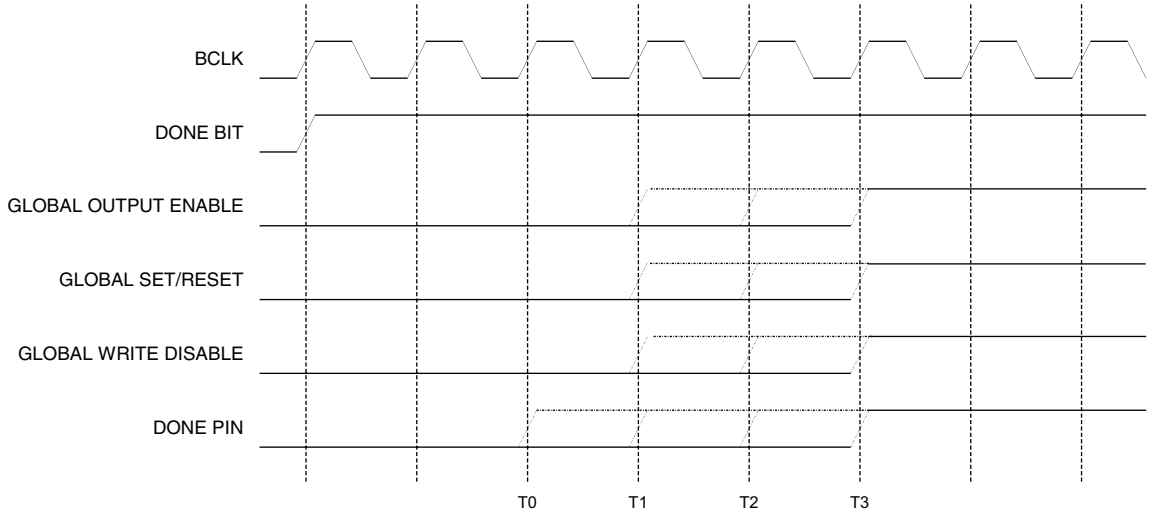
When configuration is complete the device will wake up in a predictable fashion. Wake-Up occurs after successful configuration, without errors, and provides the transition from Configuration Mode to User Mode. The Wake-Up process begins when the internal Done bit is set.

Table 13-9 provides a list of the Wake-Up sequences supported by the LatticeECP2; Figure 13-9 shows the Wake-Up timing. The Wake-Up defaults work fine for the vast majority of applications.

Table 13-9. Wake-Up Options

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 13-9. Wake-Up Timing Diagram



Synchronizing Wake-Up

The internal Wake-Up sequence clock source can be chosen as well as how the device wakes up relative to other devices.

Wake-Up Clock Selection

The Wake-Up sequence is synchronized to a clock source that is user selectable. The clock sources are External (default) and User Clock.

When External is selected the LatticeECP2 will use one of two clocks during the Wake-Up sequence, depending on the configuration data source. If the LatticeECP2 is being configured from JTAG then JTAG's TCK will be used for the Wake-Up sequence, if configuration data is coming from a sysCONFIG port (serial, parallel, or SPI) then CCLK will be used.

When User is selected, any of the design's clock signals can be used as the clock source.

Synchronous to Internal Done Bit

If the LatticeECP2 is the only device in the configuration chain, or the last device in the chain, DONE_EX should be set to the default value (OFF). The Wake-Up process will be initiated by setting of the internal Done bit on successful completion of configuration.

Synchronous to External DONE Pin

The DONE pin can be used to synchronize Wake-Up to other devices in the configuration chain. If DONE_EX (see the DONE OD, DONE EX section above) is ON then the DONE pin is a bi-directional pin. If an external device drives the DONE pin low then the Wake-Up sequence will be delayed; configuration can complete but Wake-Up is delayed. Once the DONE pin goes high the device will follow the selected WAKE_UP sequence.

In a configuration chain, a chain of devices configuring from one source (such as Figure 13-2), it is usually desirable, or even necessary, to delay wake-up of all of the devices until the last device finishes configuration. This is accomplished by setting DONE OD to OFF and DONE_EX to OFF on the last device while setting DONE OD to ON and DONE_EX to ON for the other devices.

Wake-Up Sequence Options

The Wake-Up sequence options shown in Table 13-9 determine the order of application for three internal signals, GSR, GWDIS, and GOE, and one external signal, DONE.

- GSR is used to set and reset the core of the device. GSR is asserted (low) during configuration and de-asserted (high) in the Wake-Up sequence.

- When the GWDIS signal is low it safeguards the integrity of the RAM Blocks and LUTs in the device. This signal is low before the device wakes up.
- When high, the GOE signal prevents the device's I/O buffers from driving the pins.
- When high, the DONE pin indicates that configuration is complete and that no errors were detected.

If DONE_EX (see DONE OD, DONE EX above) is OFF then sequence 21 is the default, but the user can select any sequence from 8 to 25; if DONE_EX is ON the default sequence is 4, but the user can select any sequence from 1 to 7.

Configuration FAQs

Here are some of the more common questions regarding device configuration.

General

- **Q. Other than JTAG, what is the least expensive method of configuration?**

  A. If you already have a processor and extra storage on the board you can use the processor to feed configuration data to the LatticeECP2. The least expensive stand-alone configuration option is SPI Serial Flash.

- **Q. I have created my bitstream, now how do I load the bitstream into the LatticeECP2?**

  A. Use the free Lattice ispVM tool (from www.latticesemi.com), and a Lattice ispDOWNLOAD® cable.

- **Q. I can't read the LatticeECP2 device ID using JTAG. What could be wrong?**

  A. This is the most basic of JTAG operations. If you are having trouble reading the device ID then something basic is wrong. Check that the JTAG connections are correct, and that V_{CCJ} and the download cable V_{CC} are correct (and the same). Make sure that the XRES pin is connected to ground through a 10K resistor. Check that all LatticeECP2 V_{CC} and ground pins are properly connected. Check for noise on the JTAG signals. Sometimes touching a properly grounded 'scope probe to TCK will change the symptoms; if so, you have signal noise issues. Check for excessive noise on the V_{CC} pins.

- **Q. How can I get assistance with configuration issues?**

  A. Use the On-line Assistant feature in ispVM. You will find it under the Help menu.

Mode Specific

SPI/SPIm

- **Q. How do I program the SPI Serial Flash once it's on the board?**

  A. Connect the SPI Serial Flash to the LatticeECP2 as shown in this document, then use ispVM, and a Lattice ispDOWNLOAD cable connected to the JTAG port, to program the bitstream into the Flash.

- **Q. Are there any special requirements for wiring the SPI Flash to the LatticeECP2?**

  A. Other than connecting the Flash to the right pins the only other suggestion is to add a 4.7K pull-down resistor between CCLK and ground. This keeps CCLK quite during V_{CC} ramp-up.

- **Q. Can I use 2.5V to power the SPI Flash?**

  A. Today all SPI Serial Flash of the "25" type are 3.3V, so the Flash, and V_{CCIO8}, must be connected to 3.3V.

- **Q. Can I use something other than a "25" type SPI Serial Flash?**

  A. Only devices that recognize a read op-code of 03h may be used with the LatticeECP2. Please refer to Table 13-6 for a list of vendors.

- **Q. My design is small, can I use a smaller-than-recommended SPI Flash?**

  A. The state of all of the device fuses is contained in the bitstream, whether they are part of the design or not. The size of the design does not affect the size of the bitstream.

Serial

- **Q. Can I use a free running clock for Slave Serial mode?**

  A. The LatticeECP2 clocks data in on every rising edge of CCLK so there should only be one rising clock edge for each data bit.

- **Q. Is the bitstream for the serial modes different from the bitstream for other modes?**

  A. All sysCONFIG bitstreams are the same, they can be different file types, such as hex or binary, but the data is the same.

Parallel

- **Q. My processor is generating all of the proper control signals but the LatticeECP2 won't configure, and INITN goes high and stays high while DONE stays low. What's wrong?**

  A. D0 is the MSb and D7 is the LSb. Try reversing the bit order for each byte in the bitstream. You can do this using your processor or you can generate a bit mirrored file using ispVM. Lattice recommends using your processor so that you don't have to remember to bit mirror the file.

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