LatticeECP/EC and LatticeXP DDR Usage Guide

September 2012 Technical Note TN1050

Introduction

LatticeECP™, LatticeEC™ and LatticeXP™ devices support various 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 the performance. This document will address in detail how to utilize the capabilities of the LatticeECP/EC and LatticeXP devices to implement both generic DDR and DDR memory interfaces.

DDR SDRAM Interfaces Overview

DDR SDRAM interfaces rely 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 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 SDRAM the memory controller (FPGA) must shift the DQS by 90 degrees to center align with the data signals (DQ). DQ and DQS are bi-directional ports. The same two signals are used for both write and read operations. 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. The skew between CLKP or CLKN and the SDRAM-generated DQS signal is specified in the DDR SDRAM data sheet. Figures 10-1 and 10-2 show DQ and DQS relationships for read and write cycles.

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. SDRAM interfaces typically are implemented with x8, x16 and x32 bits for each DQS signal. 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.

Figure 10-1. Typical DDR Interface

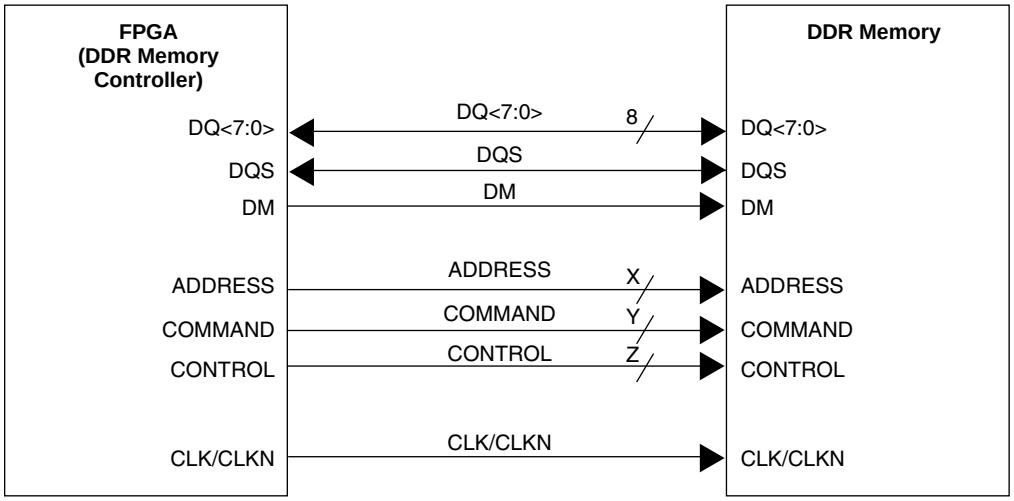


Figure 10-2. DQ-DQS During READ

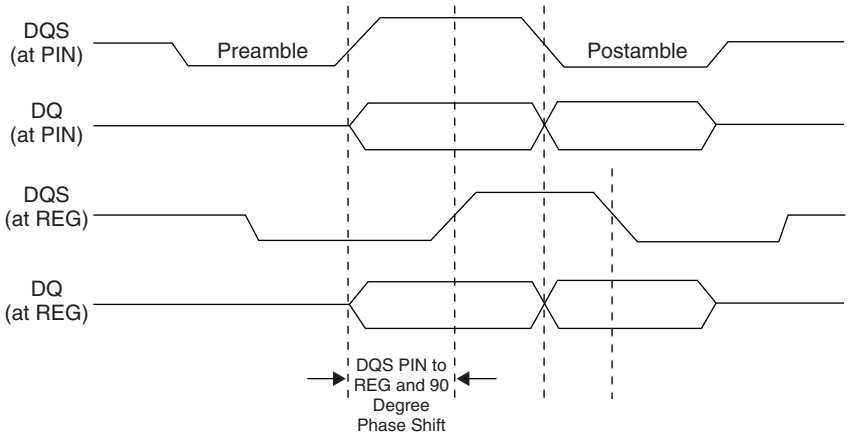
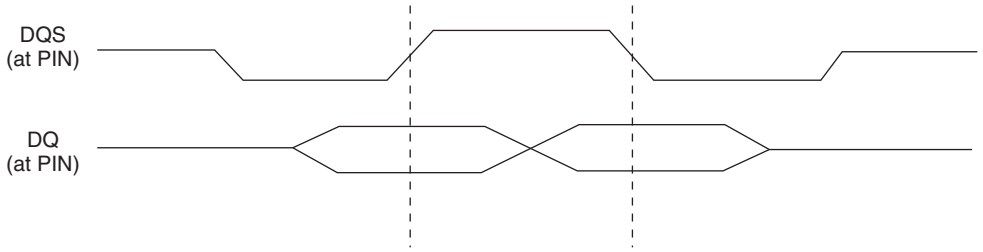


Figure 10-3. DQ-DQS During WRITE



Implementing DDR Memory Interfaces with the LatticeECP/EC Devices

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 LatticeECP/EC and LatticeXP devices.

DQS Grouping

Each DQS group generally consists of at least 10 I/Os (1DQS, 8DQ and 1DM) to implement a complete 8-bit DDR memory interface. In the LatticeECP/EC devices each DQS signal will span across 16 I/Os and in the LatticeXP devices the DQS will span 14 I/Os. Any 10 of these 16 I/Os 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.

The tables below show the total number of DQS groups available per I/O bank for each device and package.

Table 10-1. Number of DQS Banks in the LatticeECP/EC Device

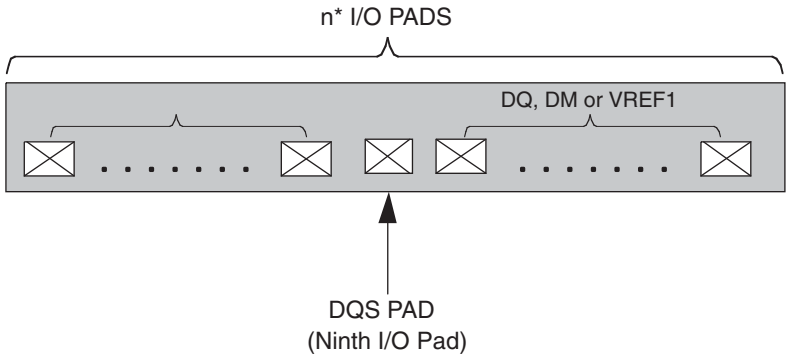
Package	Device	Total x8 DQS Groups	Number of DQS Groups per I/O Bank							
			0	1	2	3	4	5	6	7
100-pin TQFP	LFEC1	3	1	1	0	0	1	0	0	0
	LFEC3	3	1	1	0	0	1	0	0	0
144-pin TQFP	LFEC1	6	1	1	0	1	1	1	1	0
	LFEC3	6	1	1	0	1	1	1	1	0
	LFEC6/LFECP6	6	1	1	0	1	1	1	1	0
208-pin TQFP	LFEC1	6	1	1	0	1	1	1	1	0
	LFEC3	8+2[1]	1+1[1]	1	1	1	1	1+1[1]	1	1
	LFEC6/LFECP6	8+2[1]	1+1[1]	1	1	1	1	1+1[1]	1	1
	LFEC10/LFECP10	8+2[1]	1+1[1]	1	1	1	1	1+1[1]	1	1
256-ball fpBGA	LFEC3	10	2	1	1	1	1	2	1	1
	LFEC6/LFECP6	12	2	1	1	2	1	2	2	1
	LFEC10/LFECP10	12	2	1	1	2	1	2	2	1
	LFEC15/LFECP15	12	2	1	1	2	1	2	2	1
484-ball fpBGA	LFEC6/LFECP6	14	2	2	1	2	2	2	2	1
	LFEC10/LFECP10	18	3	2	2	2	2	3	2	2
	LFEC15/LFECP15	20	3	3	2	2	3	3	2	2
	LFEC20/LFECP20	22	3	3	2	3	3	3	3	2
	LFEC33/LFECP33	22	3	3	2	3	3	3	3	2
672-ball fpBGA	LFEC20/LFECP20	24	4	3	2	3	3	4	3	2
	LFEC33/LFECP33	30	4	4	3	4	4	4	4	3

1. 10 I/Os (1 DQS + 8 DQs + Bank VREF1) can function as a DDR interface in which the FPGA can have a DM output but not a DQS aligned input (in the same DDR bank as the rest of the system).

Table 10-2. Number of DQS Banks in the LatticeXP Device

Package	Device	Total x8 DQS Groups	Number of DQS Groups per I/O Bank							
			0	1	2	3	4	5	6	7
100-pin TQFP	LFXP3C/LFXP3E	2	0	0	0	0	0	1	0	1
	LFXP6C LFXP6E	2	0	0	0	0	0	1	0	1
144-pin TQFP	LFXP3C/LFXP3E	7	1	0	1	1	1	1	1	1
	LFXP6C/LFXP6E	7	1	0	1	1	1	1	1	1
208-pin PQFP	LFXP3C/LFXP3E	8	1	1	1	1	1	1	1	1
	LFXP6C/LFXP6E	8	1	1	1	1	1	1	1	1
256-ball fpBGA	LFXP3C/LFXP3E	12	2	2	1	1	2	2	1	1
	LFXP6C/LFXP6E	12	2	2	1	1	2	2	1	1
	LFXP10C/LFXP10E	16	2	2	2	2	2	2	2	2
	LFXP15C/LFXP15E	16	2	2	2	2	2	2	2	2
	LFXP20C/LFXP20E	20	3	3	2	2	3	3	2	2
388-ball fpBGA	LFXP10C/LFXP10E	16	2	2	2	2	2	2	2	2
	LFXP15C/LFXP15E	20	3	3	2	2	3	3	2	2
	LFXP20C/LFXP20E	20	3	3	2	2	3	3	2	2
484-ball fpBGA	LFXP15C/LFXP15E	20	3	3	2	2	3	3	2	2
	LFXP20C/LFXP20E	20	3	3	2	2	3	3	2	2
672-ball fpBGA	LFXP20C/LFXP20E	24	4	4	2	2	4	4	2	2

Figure 10-4. DQ-DQS Grouping



*For LatticeECP/EC: n = 16, for LatticeXP: n = 14.

Figure 10-4 shows a typical DQ-DQS group for both the LatticeECP/EC device and the LatticeXP device. The ninth I/O of this group of 16 I/Os (for LatticeECP/EC) or 14 I/Os (for LatticeXP) is the dedicated DQS pin. All eight pads before the DQS and seven (for LatticeECP/EC) or four (for LatticeXP) 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. Hence, to implement a 32-bit wide memory interface you would need to use four such DQ-DQS groups.

When not interfacing with the memory, the dedicated DQS pin can be used as a general purpose I/O. Each of the dedicated DQS pin is internally connected to the DQS phase shift circuitry. The pinout information contained in the LatticeECP/EC and LatticeXP device data sheets shows pin locations for the DQS pads. Table 10-2 shows an extract from the LatticeECP/EC data sheet. In this case, the DQS is marked as LDQS6 (L=left side, 6 =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 PL9B. Following this convention, there are eight pads before and seven pads after DQS for DQ available following counter-clockwise for the left and bottom sides of the device, and following clockwise for the top and right sides of the device. The user can assign any eight of these pads to be DQ data signals. The LatticeXP device follows the same method.

Table 10-3. EC20 Pinout (from LatticeECP/EC Family Data Sheet)

Ball Function	Bank	LVDS	Dual Function	484 fpBGA	672 fpBGA
PL2A	7	T	VREF2_7	D4	E3
PL2B	7	C	VREF1_7	E4	E4
PL3A	7	T	—	C3	B1
PL3B	7	C	—	B2	C1
PL4A	7	T	—	E5	F3
PL4B	7	C	—	F5	G3
PL5A	7	T	—	D3	D2
PL5B	7	C	—	C2	E2
PL6A	7	T	LDQS6	F4	D1
PL6B	7	C	—	G4	E1
PL7A	7	T	—	E3	F2
PL7B	7	C	—	D2	G2
PL8A	7	T	LUM0_PLLT_IN_A	B1	F6
PL8B	7	C	LUM0_PLLC_IN_A	C1	G6
PL9A	7	T	LUM0_PLLT_FB_A	F3	H4
PL9B	7	C	LUM0_PLLC_FB_A	E2	G4
PL11A	7	T	—	G5	J4

Table 10-3. EC20 Pinout (from LatticeECP/EC Family Data Sheet)

Ball Function	Bank	LVDS	Dual Function	484 fpBGA	672 fpBGA
PL11B	7	C	—	H6	J5
PL12A	7	T	—	G3	K4

DDR Software Primitives

This section describes the software primitives that can be used to implement DDR interfaces and provides details about how to instantiate them in the software. The primitives described include:

- DQSDLL The DQS delay calibration DLL
- DQSBUF The DQS delay function and the clock polarity selection logic
- INDDRXB The DDR input and DQS to system clock transfer registers
- ODDRXB The DDR output registers

An HDL usage example for each of these primitives is listed in Appendices B and C.

DQSDLL

The DQSDLL will generate a 90-degree phase shift required for the DQS signal. This primitive will implement 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 will generate the delay based on this clock frequency and the update control input to this block. The DLL will update the dynamic delay control to the DQS delay block when this update control (UDDCNTL) input is asserted. Figure 10-5 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 10-5. DQSDLL Symbol

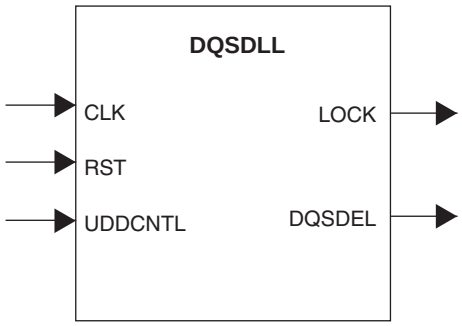


Table 10-4 provides a description of the ports.

Table 10-4. DQSDLL Ports

Port Name	I/O	Definition
CLK	I	System CLK should be at 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 Attributes

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). For operation running at or under 100 MHz it is recommended that the bit be programmed “LOW” (more tolerant). For 133 MHz, the LOCK_SENSITIVITY bit can go either way.

DQSBUF

This primitive implements the DQS Delay and the DQS transition detector logic. Figure 10-6 shows the DQSBUFB function. The preamble detect signal is also generated within this primitive.

Figure 10-6. DQSBUFB Function

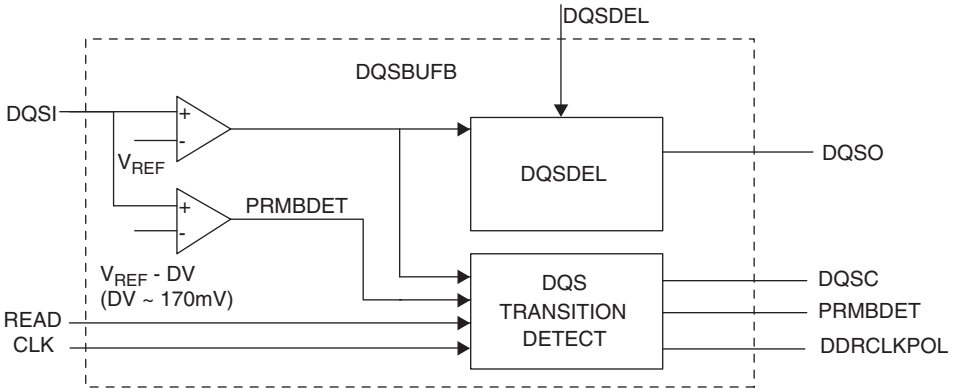


Figure 10-7 shows the primitive symbol and its ports. DQSI is the DQS signal from the memory. PRMBDET is the preamble detect signal that is generated from the DQSI input. READ and CLK are user interface signals coming from the FPGA logic. The DQSDLL block sends digital control line DQSDEL to this block. The DQS is delayed based on this input from the DQSDLL. DQSO is the delayed DQS and is connected to the clock input of the first set of DDR registers.

Figure 10-7. DQSBUFB Symbol

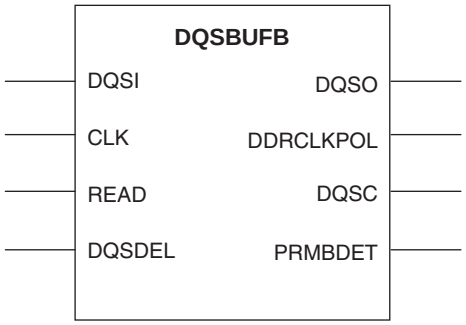


Table 10-5 provides a description of the I/O ports associated with the DQSBUFB primitive.

Table 10-5. DQSBUFB Ports

Port Name	I/O	Definition
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
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

Notes:

1. The DDR Clock Polarity output from this block should be connected to the DDCLKPOL inputs of the input register blocks (IDDRXB).

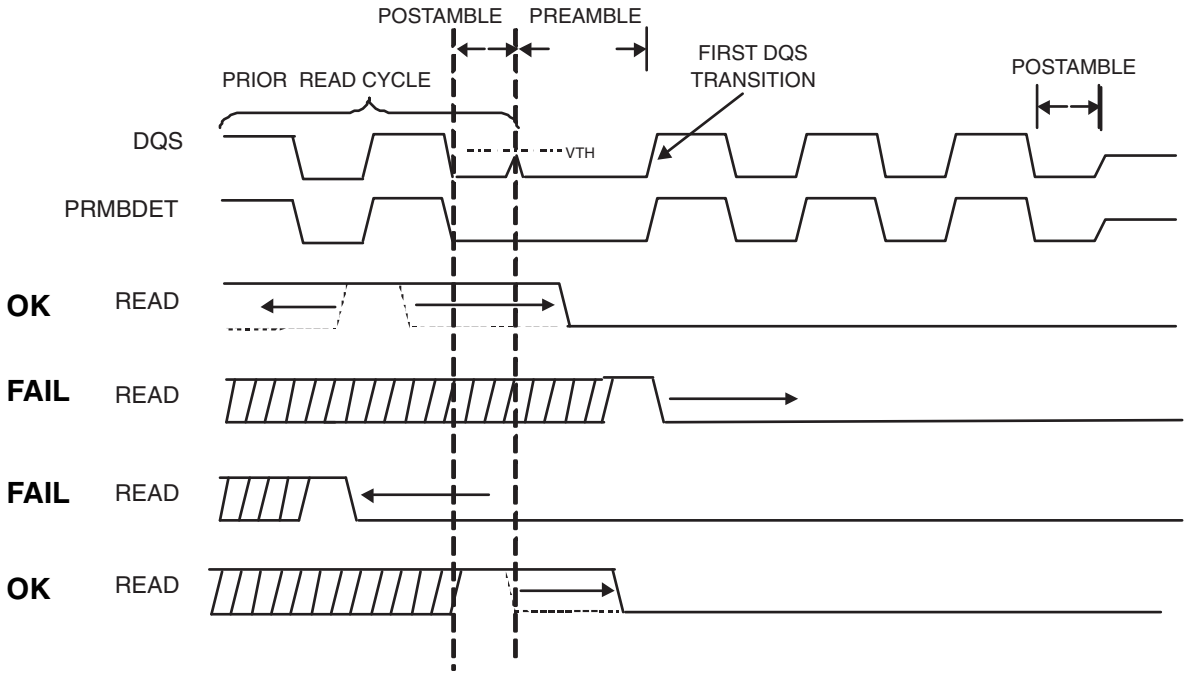
READ Pulse Generation

The READ signal to the DQSBUFB block is internally generated in the FPGA core. The Read signal will go high when the READ command to control the DDR SDRAM is initially asserted. This should normally precede the DQS preamble by one cycle yet may overlap the trailing bits of a prior read cycle. The DQS Detect circuitry of the LatticeECP/EC and LatticeXP devices require 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 needs to coincide with the initial READ Command of the Read Burst and needs to go low before the Preamble goes high.

Figure 10-8 shows the READ Pulse Timing Example with respect to the PRMBDET signal.

Figure 10-8. READ Pulse Generation



IDDRXB

This primitive will implement the input register block. The software defaults to CE Enabled unless otherwise specified. The ECLK input is used to connect to the DQS strobe coming from the DQS delay block (DQSBUFB primitive). The SCLK input should be connected to the system (FPGA) clock. The SCLK and CE inputs to this primitive will be used primarily to synchronize the DDR inputs. DDRCLKPOL is an input from the DQS Clock Polarity tree. This signal is generated by the DQS Transition detect circuit in the hardware. Figure 10-9 shows the primitive symbol and the I/O ports.

Figure 10-9. IDDRXB Symbol

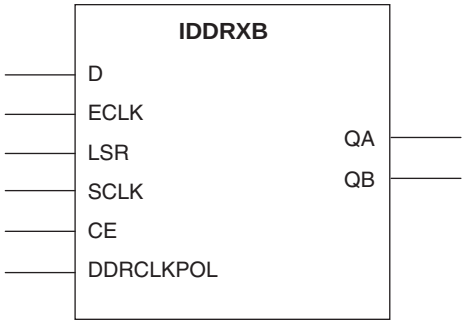


Table 10-6 provides a description of all I/O ports associated with the IDDRXB primitive.

Table 10-6. IDDRXB Ports

Port Name	I/O	Definition
D	I	DDR data
ECLK	I	The phase shifted DQS should be connected to this input
LSR	I	Reset
SCLK	I	System 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:

1. The DDRCLKPOL input to IDDRXB should be connected to the DDRCLKPOL output of DQSBUFB.

ODDRXB

The ODDRXB primitive implements both the write and the tristate functions. This primitive is used to output DDR data and the DQS strobe to the memory. The CKP and CKN can also be generated using this primitive. All the DDR output tristate implementations are also implemented using the same primitive.

Figure 10-10 shows the ODDRXB primitive symbol and its I/O ports.

Figure 10-10. ODDRXB Symbol

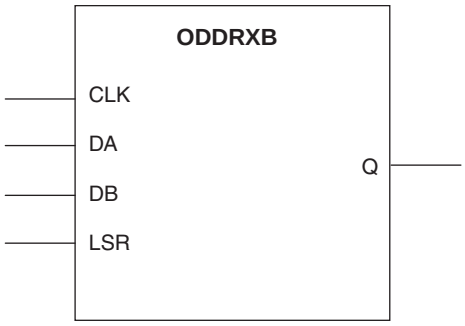


Table 10-7 provides a description of all I/O ports associated with the ODDRXB primitive.

Table 10-7. ODDRXB Ports

Port Name	I/O	Definition
CLK	I	System CLK
DA	I	Data at the positive edge of the clock
DB	I	Data at the negative edge of the clock
LSR	I	Reset
Q	I	DDR data to the memory

Notes:

1. LSR should be held low during DDR Write operation. By default, the software will be implemented CE High and LSR low.
2. DDR output and tristate registers do not have CE support. LSR is available for the tristate DDRX mode (while reading). The LSR will default to set when used in the tristate mode.
3. CE and LSR support is available for the regular (non-DDR) output mode.
4. When asserting reset during DDR writes, it is important to keep in mind that this would only reset the FFs and not the latches.

Memory Read Implementation

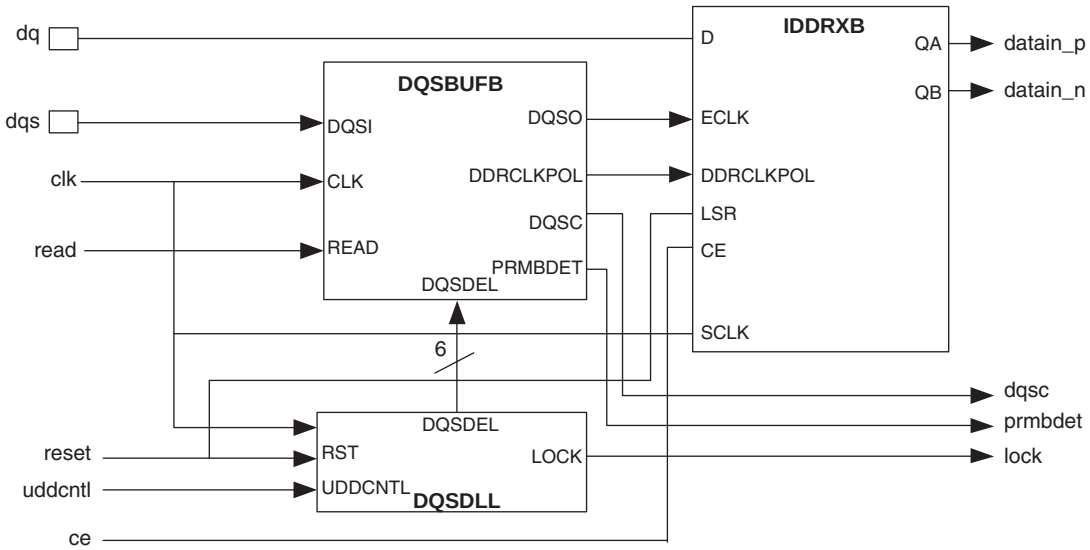
The LatticeECP/EC and LatticeXP devices contain a variety of features to simplify implementation of the read portion of a DDR interface:

- DLL compensated DQS delay elements
- DDR input registers
- Automatic DQS to system clock domain transfer circuitry

The LatticeECP/EC and LatticeXP device data sheets detail these circuit elements.

Three primitives in the Lattice ispLEVER® design tools represent the capability of these three elements. The DQSDLL represents the DLL used for calibration. The IDDRXB primitive represents the DDR input registers and clock domain transfer registers. Finally, the DQSBUFB represents the DQS delay block and the clock polarity control logic. These primitives are explained in more detail in the following sections of this document. Figure 10-11 illustrates how to hook these primitives together to implement the read portion of a DDR memory interface. The DDR Software Primitives section describes each of the primitives and its instantiation in more detail. Appendices A and B provide example code to implement the complete I/O section of a memory interface within a LatticeECP/EC or LatticeXP device.

Figure 10-11. Software Primitive Implementation for Memory READ



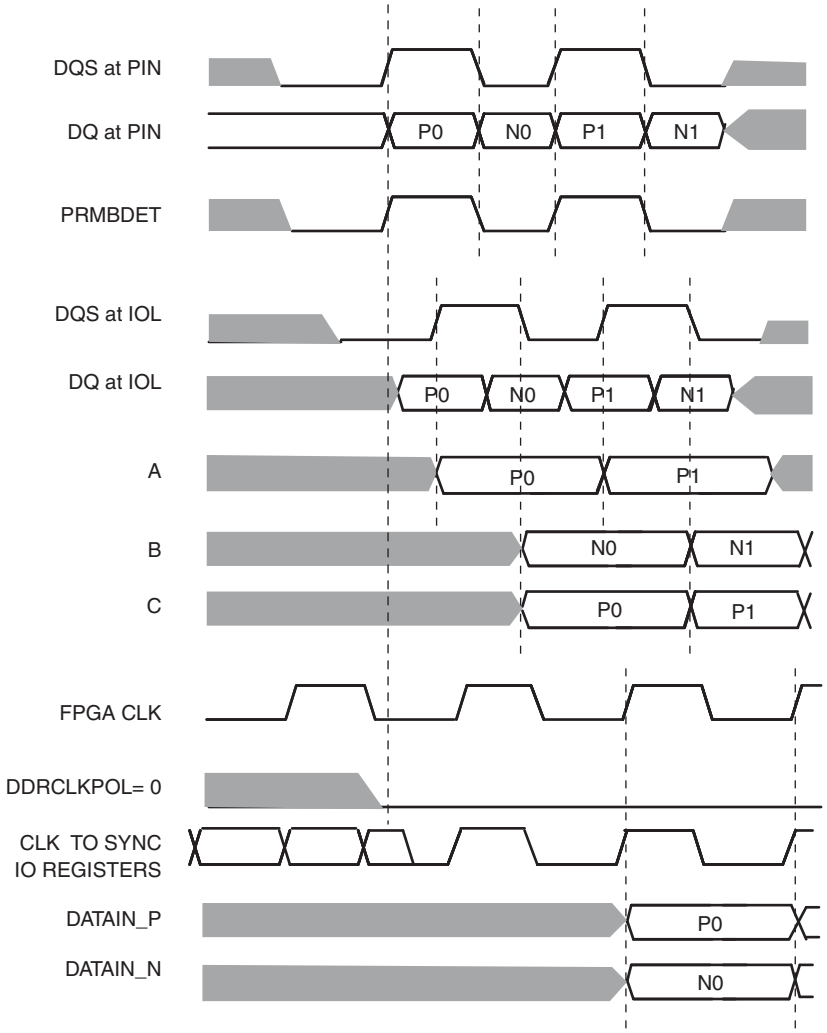
Read Timing Waveforms

Figure 10-12 and Figure 10-13 show READ data transfer for two cases 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 CLK = 0 on the 1st PRMBDET transition, then DDRCLKPOL = 0, hence no inversion required. (Figure 10-12)
- Case 2 – If CLK=1 on the 1st PRMBDET then DDRCLKPOL = 1, the system clock (CLK) needs to be inverted before it is used for synchronization. (Figure 10-13)

The signals A, B and C illustrate the Read Cycle half clock transfer at different stages of IDDRX registers. The first stage of the register captures data on the positive edge as shown by signal A and negative edge as shown by signal B. The data stream A goes through an additional half clock cycle transfers shown by signal C. Phase aligned data streams B and C are presented to the next stage registers clocked by the FPGA CLK

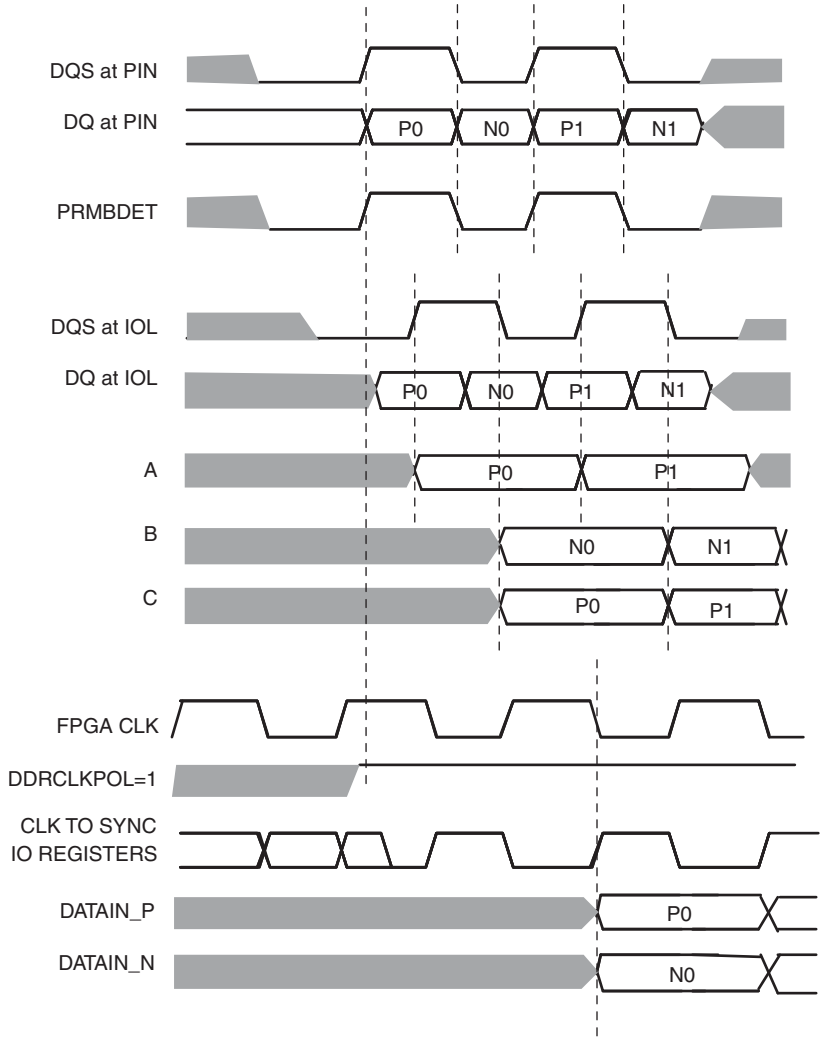
Figure 10-12. READ Data Transfer When DDRCLKPOL=0



Notes -

(1) DDR memory sends DQ aligned to DQS strobe.
(2) The DQS Strobe is delayed by 90 degree 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 IO registers A and B, capture data on the positive edge and negative edge of DQS.
(6) IO register C transfers data so that both data are now aligned to negative edge of DQS.
(7) DDCLKPOL signal generated will determine if the CLK going into the synchronization registers need to be inverted. In this case, the DDRCLKPOL=0 as the CLK is LOW at the 1st rising edge of PRMBDET.
(8) The IO Synchronization registers capture data at on positive edge of the FPGA CLK.

Figure 10-13. Read Data Transfer When DDRCLKPOL=1



Notes -

(1) DDR memory sends DQ aligned to DQS strobe.
(2) The DQS Strobe is delayed by 90 degree 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 IO registers A and B, capture data on the positive edge and negative edge of DQS.
(6) IO register C transfers data so that both data are now aligned to negative edge of DQS.
(7) DDCLKPOL signal generated will determine if the CLK going into the synchronization registers need to be inverted. In this case, the DDRCLKPOL=1 as the CLK is HIGH at the 1st rising edge of PRMBDET.
(8) The IO Synchronization registers capture data at on negative edge of the FPGA CLK.

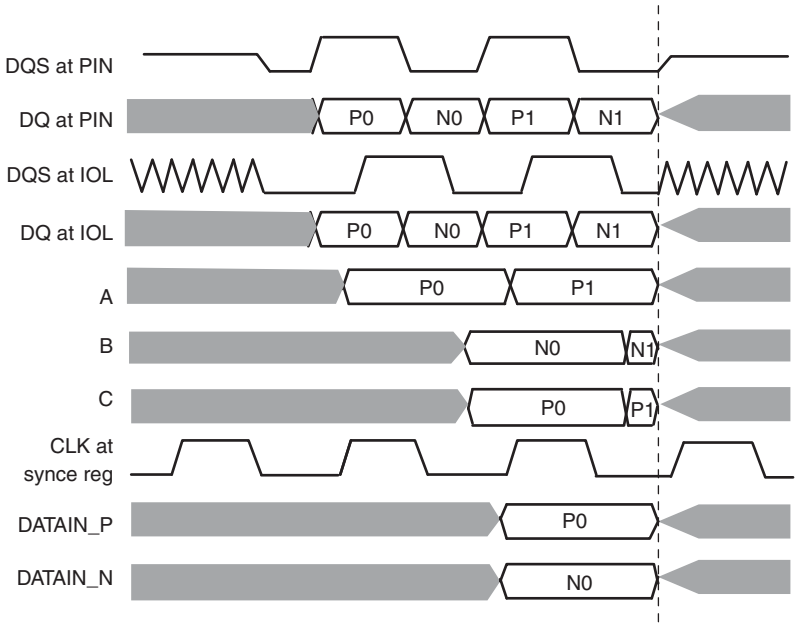
Data Read Critical Path

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 system CLK, this path should be constrained with a half clock transfer. This half clock transfer can be forced in the software by assigning a multicycle constraint (multicycle of 0.5 X) on all the data paths to the 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 10-14. Postamble Effect on READ



LatticeECP/EC and LatticeXP devices have extra dedicated logic in the in the DQS Delay Block that will prevent this postamble problem. The DQS postamble logic is automatically implemented when the user instantiates the DQS Delay logic (DQSBUFB software primitive) in a design.

This postamble solution was implemented in all the devices of the LatticeECP/EC and LatticeXP families except the LFEC20/LFECP20 device. For this device, it is recommended that the user issue an extra READ command to assure correct data has been transferred to the synchronization registers.

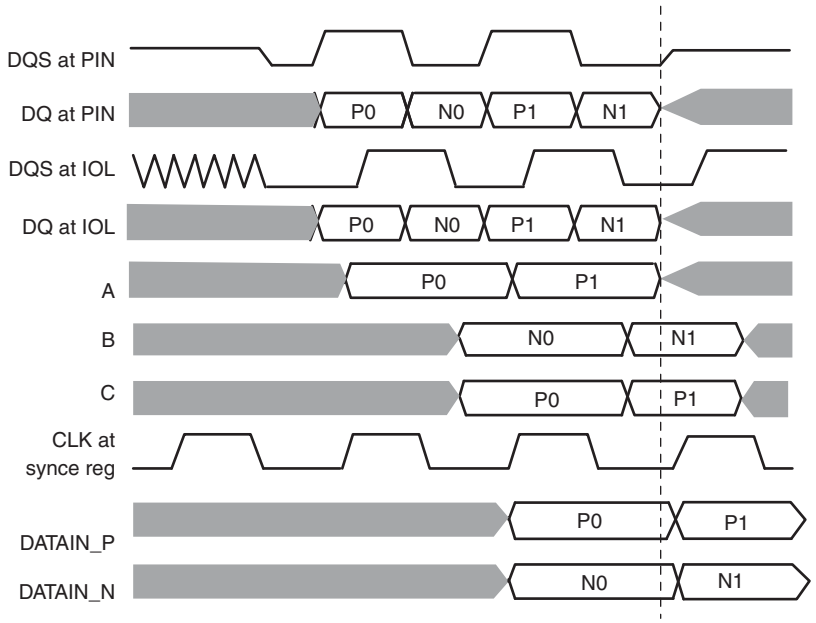
The circumstances under which the extended READ cycle is issued are given in Table 10-8.

Table 10-8. DDR Read Postamble

Current Command	Next Command	Action	Lost Cycles
Read (Row x, Bank y)	Read (Row x, Bank y)	None.	None
Read (any address)	NOP	Extend the current read command.[1]	3
Read (Row x, Bank y)	Read (Row n, Bank y)	Extend the current read to (Row x, Bank y) consecutive to current command	3
Read (Row x, Bank y)	Read (Row x, Bank n)	If the Row x, Bank n was open, do nothing. Else, extend the current read to Row x, Bank y	3
Read (any address)	Write/LMR	Extend the current read command.	3

1. Current read is extended one or more additional clock cycles.

Figure 10-15. Postamble Solution with Extra READ Command



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 data, DQ. Along with the strobe and data this portion of the interface provides the CLKP, CLKN Address/Command and Data Mask (DM) signals to the memory.

LatticeECP/EC and LatticeXP devices contain DDR output and tri-state registers along with PLLs that allow the easy implementation of the write portion of the DDR memory interfaces. The DDR output registers can be accessed in the design tools via the ODDRXB primitive.

All DDR output signals (“ADDR, CMD”, DQS, DQ, DM) are initially aligned to the rising edge of CLK inside the FPGA core. These signals are used for the entire DDR write interface or the controls of DDR read interface. The relative phase of the signals may be adjusted in the IOL logic before departing the FPGA. The adjustments are shown in Figure 16

The adjustments are as follows:

The PLL is used to generate a 90 degree phase shifted clock. This 90 degree phase shifted clock will be used to generate DQS and the differential clocks going to the memory.

The CLKP needs to be centered relative to the ADDR,CMD signal, which is an SDR signal. This is accomplished by inverting the CLKP signal relative to the PLL’s 90 degree phase shifted CLK.

The DDR clock can be generated by assigning “0” to the DA input and “1” to the DB inputs of the ODDRXB primitive as shown in Figure 10-16. This is then fed into a SSTL25 differential output buffer to generate CLKP and CLKN differential clocks. Generating the CLKN in this manner would prevent any skew between the two signals.

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 met by making CLKP and DQS identical in phase. DQS is inverted to match CLKP (= CLK + 270). This is accomplished by routing the positive DQS data in core logic to DB, and negative DQS data in core logic to DA inputs of the ODDRXB primitive.

Internally the DQS and ADDR/CMD signals are clocked using the primary FPGA clock. Therefore, the user will need to do a 1/4 (one-quarter) clock transfer from the core logic to the DDR registers. Timing can be hard to meet, so it is recommended that the user first register these signals with the inverted Clock, so that the transfer from the core logic to I/O registers will only require a 1/2 (half) clock transfer.

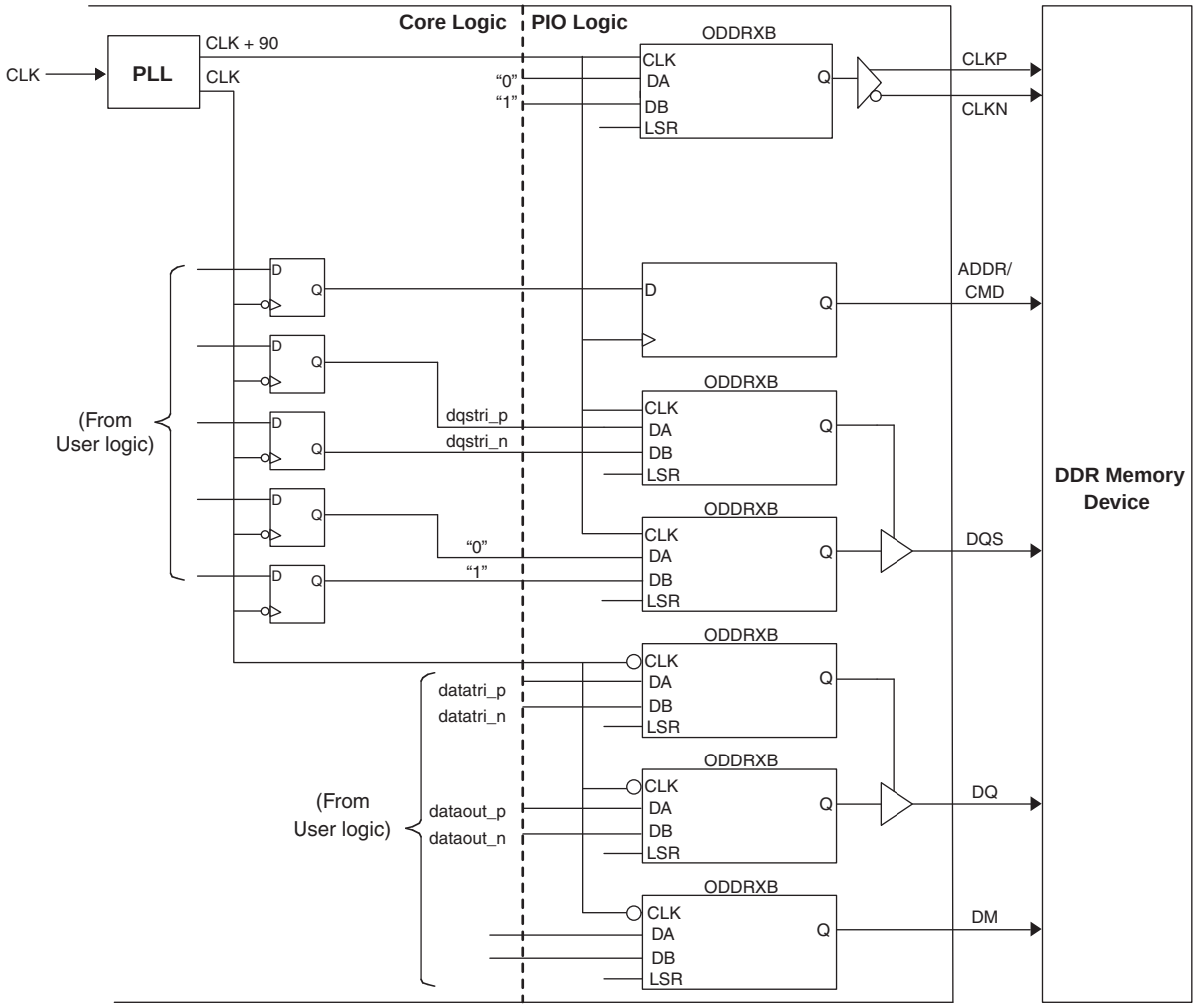
The data DQ and DM needs to be delayed by 90° as it leaves the FPGA. This is to center the data and data mask relative to the DQS when it reaches the DDR memory. This can be accomplished by inverting the CLK to the DQ and DM data.

The DM signal is generated using the same clock as the DQ data pin. The memory masks the DQ signals if the DM pins are driven high.

The tristate control for the data output can also be implemented using the ODDRXB primitive.

Figure 10-16 illustrates how to hook up the ODDRXB primitives and the PLL. The DDR Software Primitives section describes each of the primitives and its instantiation in more detail. Appendix A and Appendix B provide example code for implementing the complete I/O section of a memory interface for a LatticeECP/EC or LatticeXP device.

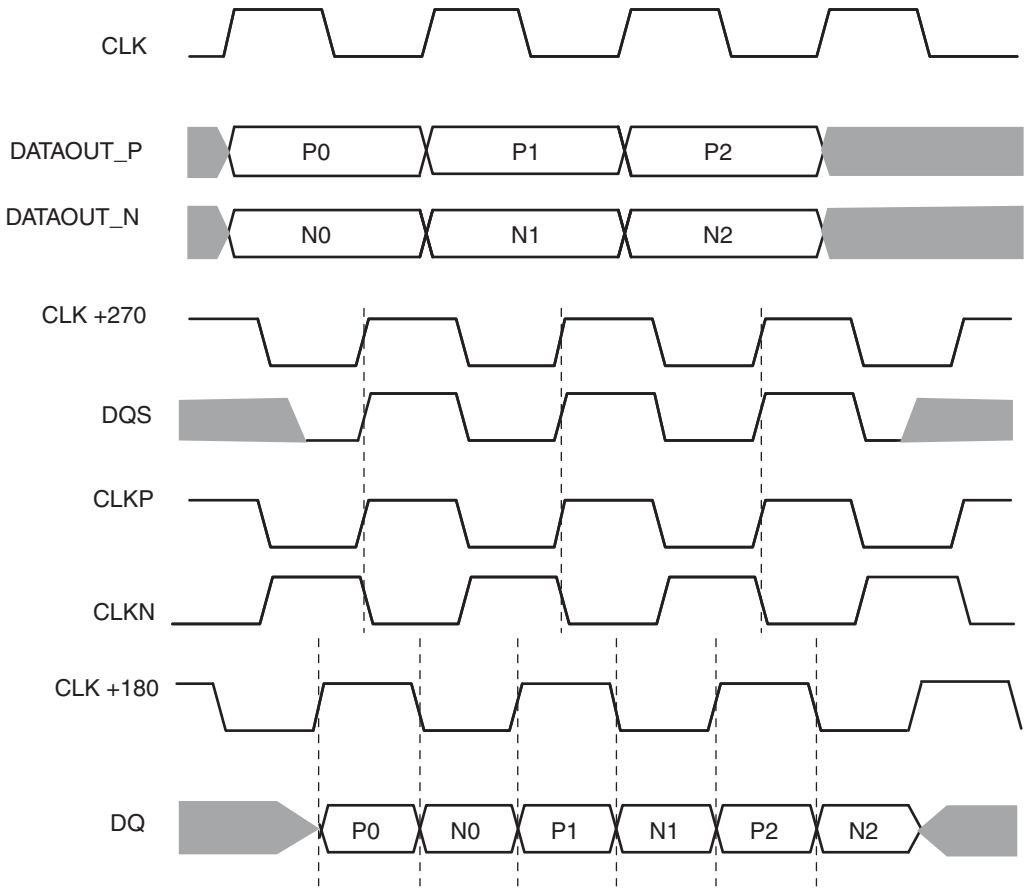
Figure 10-16. Software Primitive Implementation for Memory Write



Write Timing Waveforms

Figure 10-17 shows 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 10-17. DDR Write Data Transfer for DQ Data



Notes -
(1) DATAOUT_P and DATAOUT_N are inputs to the DDR output registers.
(2) DQS is generated at 270 degree phase of CLK.
(3) CLKP is generated simular to DQS and CLKN is the inverted CLKP.
(4) DQ is generated at 180 degree phase of CLK.
(5) DQ is center aligned with the DQS strobe signal when it reaches the memory.

Design Rules/Guidelines

Listed below are some rules and guidelines to keep in mind when implementing DDR memory interfaces in the LatticeECP/EC and LatticeXP devices.

- The LatticeECP/EC and LatticeXP devices have dedicated DQ-DQS banks. Please refer to the logical signal connections of the groups in the LatticeECP/EC and LatticeXP data sheets before locking these pins.
- There are two DQSDLLs on the device, one for the top half and one for the bottom half. Hence, 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 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.

- The DDR SDRAM interface supports the SSTL25 I/O standard, therefore the interface pins should be assigned as SSTL25 I/O type.
- When implementing a DDR interface, the VREF1 of the bank is used to provide the reference voltage for the interface pins.
- Appendix F shows DDR400 implementation results of the LatticeEC Advanced Evaluation Board.

QDR II Interface

QDR II SRAM is a new memory technology defined by a number of leading memory vendors for high-performance and high-bandwidth communication applications. QDR is a synchronous pipelined burst SRAM with two separate unidirectional data buses dedicated for read and write operations running at double data rate. Both the QDR II read and write interfaces use HSTL 1.8V I/O standard.

A QDR II memory controller can be easily implemented using the LatticeECP/EC and LatticeXP devices by taking advantage of the DDR I/O registers. For LatticeECP/EC and LatticeXP devices, ODDRXB primitives are used on the QDR outputs and PFU registers are used on the QDR inputs to implement the DDR interface. To see the details of this implementation refer to Lattice reference design RD1019, *QDR Memory Controller* on the Lattice web site at www.latticesemi.com.

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.

LatticeECP/EC and LatticeXP devices can implement an 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.

Generic High Speed DDR Implementation

In addition to the DDR memory interface, users can use the I/O logic registers to implement a high speed DDR interface. DDR data write operations can be implemented using the DDR output registers similar to the memory interface implementation using the ODDRXB primitives.

On the input side, the read interface can be implemented using the core logic PFU registers. The PFU register next to the I/O cells can be used to de-mux the DDR data to single data rate data. This method of implementing DDR can run at 300 MHz when accompanied by proper preferences in the software. The HDL and the preferences to implement this DDR interface are listed in Appendix D of this document.

Board Design Guidelines

The most common challenge associated with implementing DDR memory interfaces is the board design and layout. Users must strictly follow the guidelines recommended by memory device vendors.

Some common recommendations include matching trace lengths of interface signals to avoid skew, proper DQ-DQS signal grouping, proper termination of the SSTL2 I/O standard, proper VREF and VTT generation decoupling and proper PCB routing.

Some reference documents that discuss 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.
- www.micron.com, TN4607, DDR 333 Design Guide for Two DIMM Systems

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 LatticeECP/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 LatticeECP/EC Devices) User's Guide

Technical Support Assistance

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

Revision History

Date	Version	Change Summary
—	—	Previous Lattice releases.
February 2007	03.2	Updated Generic High Speed DDR Implementation section.
		Updated Appendix D.
September 2012	03.3	Updated document with new corporate logo.

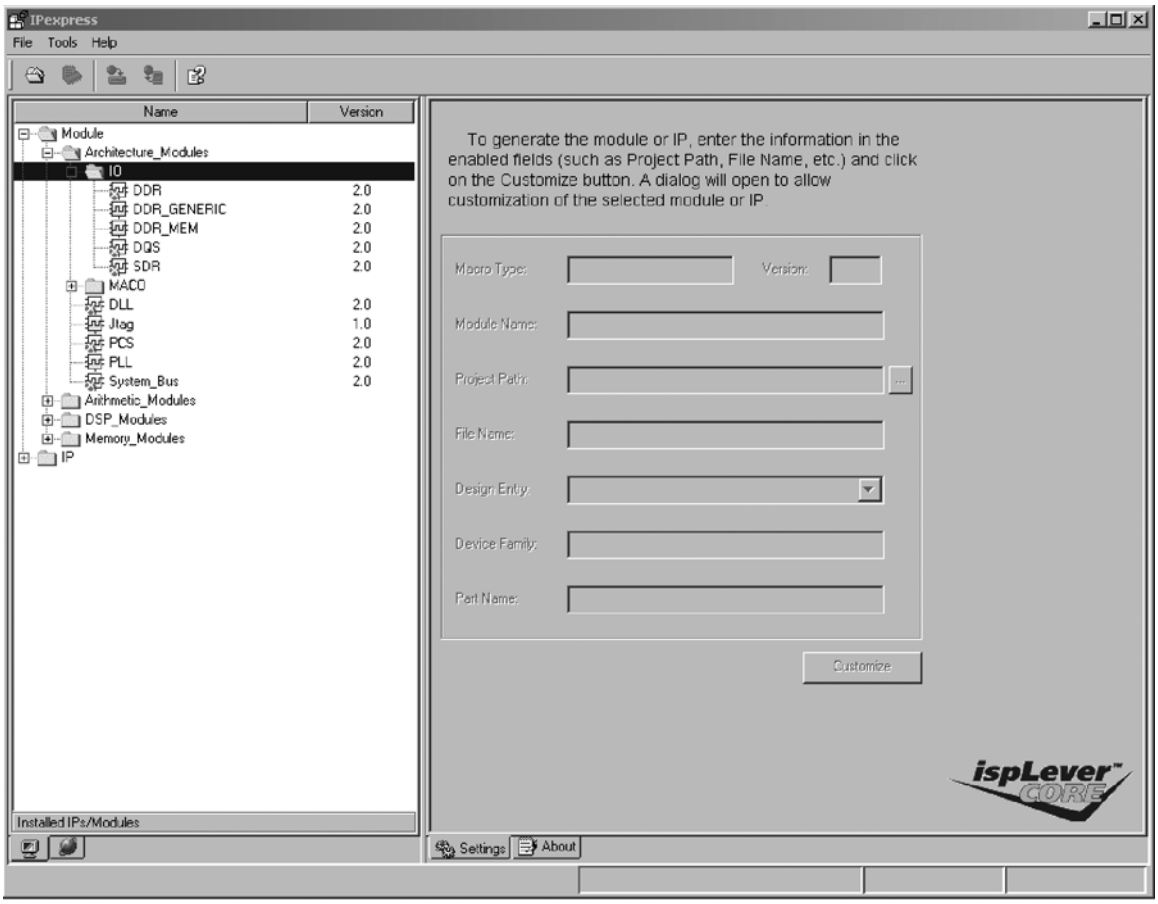
Appendix A. Using IPexpress™ to Generate DDR Modules

The input and output DDR module can be generated using IPexpress. The I/O section under the Architecture modules provides two options to the user:

1. DDR_GENERIC – The option allows generation of a Generic DDR interface, which in the case of LatticeECP/EC and LatticeXP devices, is only the output side DDR. The input side for a Generic DDR interface must be implemented using PFU registers. Appendix D provides the example code for the input side generic DDR.
2. DDR_MEM – This option allows the user to generate a complete DDR memory interface. It will generate both the read and write side interface required to interface with the memory.

IPexpress generates only the modules that are implemented within the IOLOGIC. Any logic required in the FPGA core to complete the memory interface must be implemented by the user.

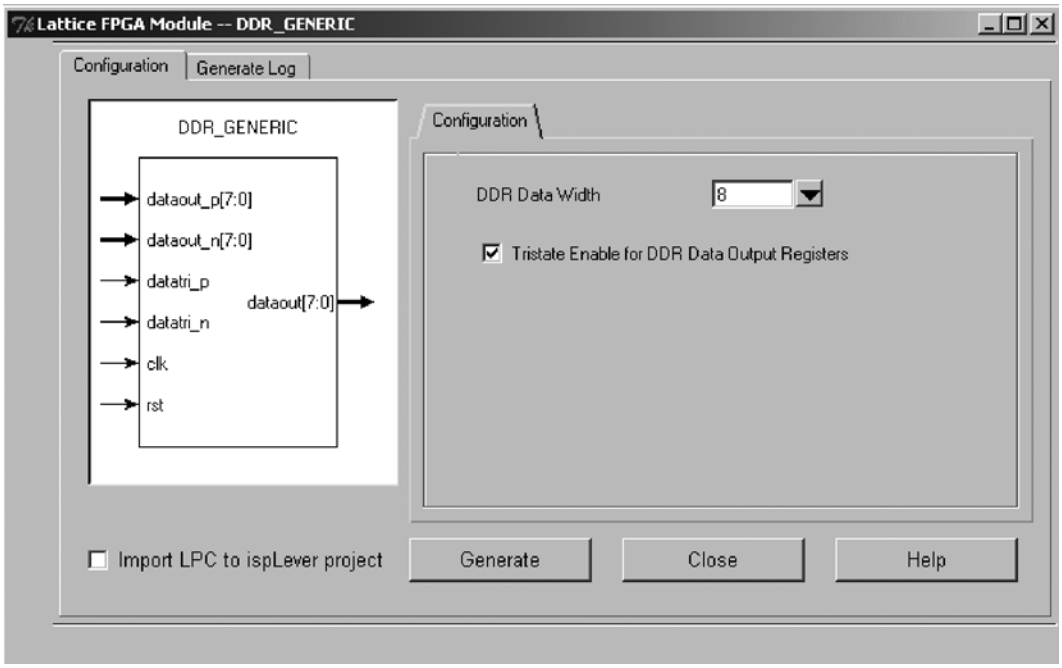
Figure 10-18. IPexpress I/O Section



DDR Generic

DDR Generic will generate the output DDR (ODDRXB) primitives for a given bus width. The user has the option to enable or disable tristate control to the output DDR registers. Figure 10-19 shows the DDR Generic views of IPexpress.

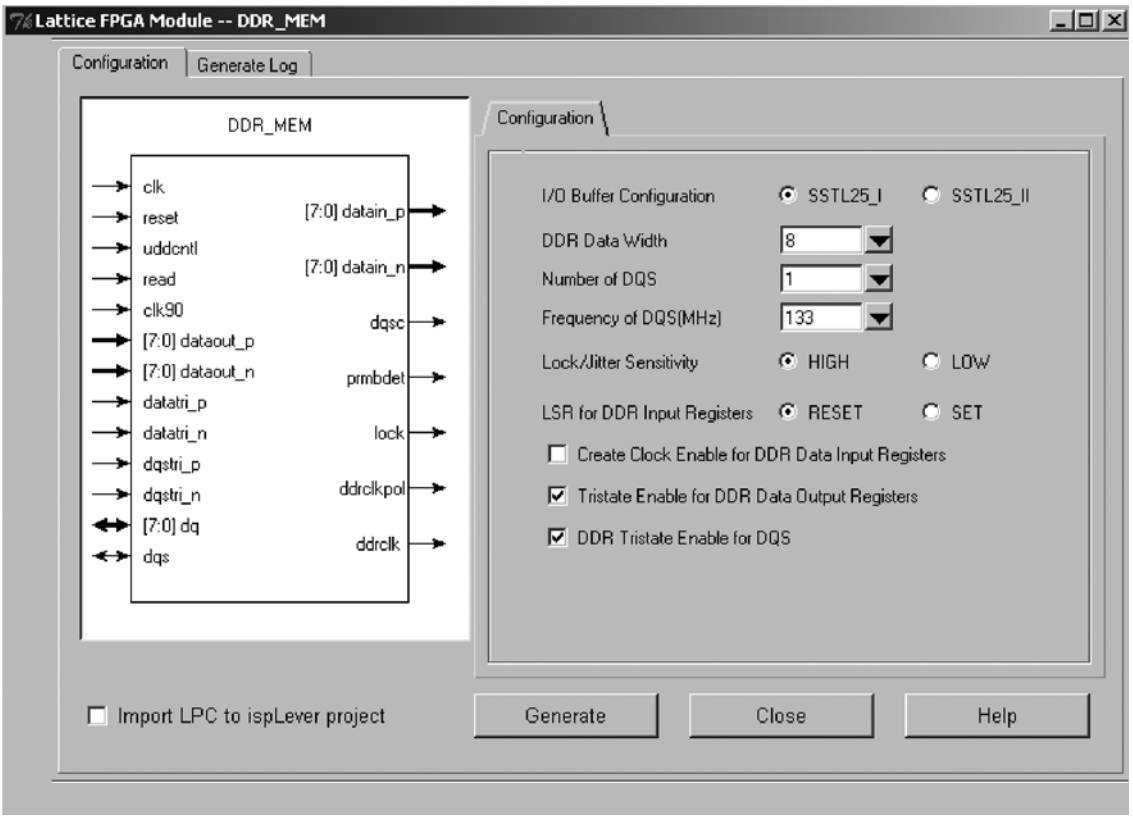
Figure 10-19. DDR Generic Configuration Options



DDR Memory Interface

This IPexpress option generates both read and write interfaces using all DDR primitives for a given bus width. Figure 10-20 shows the options under this section.

Figure 10-20. Configuration Options for DDR Memory Interface



Appendix B. Verilog Example for DDR Input and Output Modules

```
module ddr_mem (dq, dqs, clk, reset, uddcntl, read, datain_p, datain_n, dqsc, prmbdet, lock,
ddrclkpol, clk90, dqstri_p, dqstri_n, datatri_p, datatri_n, dataout_p, dataout_n, ddrclk);

    inout [7:0] dq/* synthesis IO_TYPE="SSTL25_II"*/;
    inout dqs/* synthesis IO_TYPE="SSTL25_II"*/;

    --clk is the core clock and clk90 is the 90 degree phase shifted clock coming from the PLL
    input clk, clk90;
    input reset, uddcntl, read;
    input [7:0] dataout_p, dataout_n;
    input [7:0] datatri_p, datatri_n;
    input dqstri_p, dqstri_n;

    output [7:0] datain_p;
    output[7:0] datain_n;
    output dqsc, prmbdet, lock, ddrclkpol;

    output ddrclk /* synthesis IO_TYPE="SSTL25D_II"*/ ;
    wire vcc_net,gnd_net;
    wire dqsbuf, dqsdel, clk, ddrclkpol_sig;
    wire [7:0] ddrin, ddrout, tridata;
    wire dqsout, tridqs, dqsin, ddrclk;


    assign vcc_net = 1'b1;
    assign gnd_net = 1'b0;
    assign ddrclkpol = ddrclkpol_sig;


    //-------Bidirectional Buffers -----------------------------------------------------
    BB bidiInst0 (.I(ddrout[0]), .T(tridata[0]), .O(ddrin[0]), .B(dq[0]));
    BB bidiInst1 (.I(ddrout[1]), .T(tridata[1]), .O(ddrin[1]), .B(dq[1]));
    BB bidiInst2 (.I(ddrout[2]), .T(tridata[2]), .O(ddrin[2]), .B(dq[2]));
    BB bidiInst3 (.I(ddrout[3]), .T(tridata[3]), .O(ddrin[3]), .B(dq[3]));
    BB bidiInst4 (.I(ddrout[4]), .T(tridata[4]), .O(ddrin[4]), .B(dq[4]));
    BB bidiInst5 (.I(ddrout[5]), .T(tridata[5]), .O(ddrin[5]), .B(dq[5]));
    BB bidiInst6 (.I(ddrout[6]), .T(tridata[6]), .O(ddrin[6]), .B(dq[6]));
    BB bidiInst7 (.I(ddrout[7]), .T(tridata[7]), .O(ddrin[7]), .B(dq[7]));

    //Bidirectional Strobe, DQS
    BB bidiInst8(.I(dqsout), .T(tridqs), .O(dqsin), .B(dqs));
    //-------------------------------------------------------------------------------

    //-----------DDR Input ----------------------------------------------------------


    DQSBUFB    U8    (.DQSI(dqsin),    .CLK(clk),    .READ(read),    .DQSDEL(dqsdel),    .DDRCLK-
POL(ddrclkpol_sig),
                    .DQSC(dqsc), .PRMBDET(prmbdet), .DQSO(dqsbuf));

    DQSDLL U9 (.CLK(clk), .UDDCNTL(uddcntl), .RST(reset), .DQSDEL(dqsdel), .LOCK(lock));

    IDDRXB    UL0    (.D(ddrin[0]),    .ECLK(dqsbuf),    .SCLK(clk),    .CE(vcc_net),    .DDRCLK-
POL(ddrclkpol_sig),
                   .LSR(reset), .QA(datain_p[0]), .QB(datain_n[0]));
```

```
   IDDRXB    UL1    (.D(ddrin[1]),    .ECLK(dqsbuf),    .SCLK(clk),    .CE(vcc_net),    .DDRCLK-
POL(ddrclkpol_sig),
                    .LSR(reset), .QA(datain_p[1]), .QB(datain_n[1]));

   IDDRXB    UL2    (.D(ddrin[2]),    .ECLK(dqsbuf),    .SCLK(clk),    .CE(vcc_net),    .DDRCLK-
POL(ddrclkpol_sig),
                    .LSR(reset), .QA(datain_p[2]), .QB(datain_n[2]));

   IDDRXB    UL3    (.D(ddrin[3]),    .ECLK(dqsbuf),    .SCLK(clk),    .CE(vcc_net),    .DDRCLK-
POL(ddrclkpol_sig),
                    .LSR(reset), .QA(datain_p[3]), .QB(datain_n[3]));

   IDDRXB    UL4    (.D(ddrin[4]),    .ECLK(dqsbuf),    .SCLK(clk),    .CE(vcc_net),    .DDRCLK-
POL(ddrclkpol_sig),
                    .LSR(reset), .QA(datain_p[4]), .QB(datain_n[4]));

   IDDRXB    UL5    (.D(ddrin[5]),    .ECLK(dqsbuf),    .SCLK(clk),    .CE(vcc_net),    .DDRCLK-
POL(ddrclkpol_sig),
                    .LSR(reset), .QA(datain_p[5]), .QB(datain_n[5]));

   IDDRXB    UL6    (.D(ddrin[6]),    .ECLK(dqsbuf),    .SCLK(clk),    .CE(vcc_net),    .DDRCLK-
POL(ddrclkpol_sig),
                    .LSR(reset), .QA(datain_p[6]), .QB(datain_n[6]));

   IDDRXB    UL7    (.D(ddrin[7]),    .ECLK(dqsbuf),    .SCLK(clk),    .CE(vcc_net),    .DDRCLK-
POL(ddrclkpol_sig),
                    .LSR(reset), .QA(datain_p[7]), .QB(datain_n[7]));


  //-------------------------------------------------------------------------------------
  //----TRISTATE Instantiations-----------------------------------------------------------
  // DDR Trisate for data, DQ

  ODDRXB T0 (.DA(datatri_p[0]), .DB(datatri_n[0]), .LSR(reset), .CLK(~clk), .Q(tridata[0]));
  ODDRXB T1 (.DA(datatri_p[1]), .DB(datatri_n[1]), .LSR(reset), .CLK(~clk), .Q(tridata[1]));
  ODDRXB T2 (.DA(datatri_p[2]), .DB(datatri_n[2]), .LSR(reset), .CLK(~clk), .Q(tridata[2]));
  ODDRXB T3 (.DA(datatri_p[3]), .DB(datatri_n[3]), .LSR(reset), .CLK(~clk), .Q(tridata[3]));
  ODDRXB T4 (.DA(datatri_p[4]), .DB(datatri_n[4]), .LSR(reset), .CLK(~clk), .Q(tridata[4]));
  ODDRXB T5 (.DA(datatri_p[5]), .DB(datatri_n[5]), .LSR(reset), .CLK(~clk), .Q(tridata[5]));
  ODDRXB T6 (.DA(datatri_p[6]), .DB(datatri_n[6]), .LSR(reset), .CLK(~clk), .Q(tridata[6]));
  ODDRXB T7 (.DA(datatri_p[7]), .DB(datatri_n[7]), .LSR(reset), .CLK(~clk), .Q(tridata[7]));

  // DDR Trisate for strobe, DQS
  ODDRXB T8 (.DA(dqstri_p), .DB(dqstri_n), .LSR(reset), .CLK(clk90), .Q(tridqs));

  //-------------------------------------------------------------------------------------
-

  //-----------DQ  output-------------------------------------------------------------------
-
  ODDRXB O0 (.DA(dataout_p[0]), .DB(dataout_n[0]), .LSR(reset), .CLK(~clk), .Q(ddrout[0]));
  ODDRXB O1 (.DA(dataout_p[1]), .DB(dataout_n[1]), .LSR(reset), .CLK(~clk), .Q(ddrout[1]));
  ODDRXB O2 (.DA(dataout_p[2]), .DB(dataout_n[2]), .LSR(reset), .CLK(~clk), .Q(ddrout[2]));
  ODDRXB O3 (.DA(dataout_p[3]), .DB(dataout_n[3]), .LSR(reset), .CLK(~clk), .Q(ddrout[3]));
  ODDRXB O4 (.DA(dataout_p[4]), .DB(dataout_n[4]), .LSR(reset), .CLK(~clk), .Q(ddrout[4]));
  ODDRXB O5 (.DA(dataout_p[5]), .DB(dataout_n[5]), .LSR(reset), .CLK(~clk), .Q(ddrout[5]));
  ODDRXB O6 (.DA(dataout_p[6]), .DB(dataout_n[6]), .LSR(reset), .CLK(~clk), .Q(ddrout[6]));
  ODDRXB O7 (.DA(dataout_p[7]), .DB(dataout_n[7]), .LSR(reset), .CLK(~clk), .Q(ddrout[7]));
```

```
  //------------------------------------------------------------------------------------
--
  //--------------- DQS output-----------------------------------------------------------
--
  ODDRXB O8 (.DA(gnd_net), .DB(vcc_net), .LSR(reset), .CLK(clk90), .Q(dqsout));
  //------------------------------------------------------------------------------------
--

  //--------------- CLKOUTP and CLKOUTN Generation---------------------------------------
--
  ODDRXB O9 (.DA(gnd_net), .DB(vcc_net), .LSR(reset), .CLK(clk90), .Q(ddrclk));
  //------------------------------------------------------------------------------------
--
endmodule
```

Appendix C. VHDL Example for DDR Input and Output Modules

```
library IEEE;
use IEEE.std_logic_1164.all;
library ec;
use ec.components.all;

entity ddr_mem is
        port( dq      : inout std_logic_vector(7 downto 0 );
              dqs     : inout std_logic;
              clk     : in std_logic; -- core clock
              clk90      : in std_logic; -- 90 degree phase shifted clock from the pll
              reset      : in std_logic;
              uddcntl    : in std_logic;
              read       : in std_logic;
              dataout_p  : in std_logic_vector(7 downto 0);
              dataout_n  : in std_logic_vector(7 downto 0);
              datatri_p  : in std_logic_vector(7 downto 0);
              datatri_n  : in std_logic_vector(7 downto 0);
              dqstri_p   : in std_logic;
              dqstri_n   : in std_logic;
              ddrclk     : out std_logic;
              datain_p   : out std_logic_vector(7 downto 0);
              datain_n   : out std_logic_vector(7 downto 0);
              dqsc       : out std_logic;
              prmbdet    : out std_logic;
              lock       : out std_logic;
              ddrclkpol  : out std_logic);

--*****DDR interface signals assigned SSTL25 IO Standard *************
ATTRIBUTE IO_TYPE             :  string;
ATTRIBUTE IO_TYPE OF ddrclk   :  SIGNAL IS "SSTL25D_II";
ATTRIBUTE IO_TYPE OF dq       :  SIGNAL IS "SSTL25_II";
ATTRIBUTE IO_TYPE OF dqs      :  SIGNAL IS "SSTL25_II";

end ddr_mem;

architecture structure of ddr_mem is

--*****DDR Input register*********************************************
 component IDDRXB
    port(
       D    : in STD_LOGIC;
       ECLK : in STD_LOGIC;
       SCLK : in STD_LOGIC;
       CE   : in STD_LOGIC;
       LSR  : in STD_LOGIC;
       DDRCLKPOL  : in STD_LOGIC;
       QA   : out STD_LOGIC;
       QB   : out STD_LOGIC);
 end component;
```

```
--*******DDR Output register ******************************************
 component ODDRXB
    port(
       CLK  : in STD_LOGIC;
       DA   : in STD_LOGIC;
       DB   : in STD_LOGIC;
       LSR  : in STD_LOGIC;
       Q    : out STD_LOGIC);
 end component;

--*******Bidirectional Buffer*********************************************
 component BB
    port(
       I  : in STD_LOGIC;
       T  : in STD_LOGIC;
       O  : out STD_LOGIC;
       B  : inout STD_LOGIC);
 end component;

--******DQS DLL Component**************************************************
 component DQSDLL
    port(
       CLK         : in STD_LOGIC;
       RST         : in STD_LOGIC;
       UDDCNTL     : in STD_LOGIC;
       LOCK        : out STD_LOGIC;
       DQSDEL      : out STD_LOGIC);
 end component;

--****** DQS Delay block***************************************************
 component DQSBUFB
     port(
       DQSI        : in STD_LOGIC;
       CLK         : in STD_LOGIC;
       READ        : in STD_LOGIC;
       DQSDEL      : in STD_LOGIC;
       DQSO        : out STD_LOGIC;
       DDRCLKPOL   : out STD_LOGIC;
       DQSC        : out STD_LOGIC;
       PRMBDET     : out STD_LOGIC);
 end component;


 signal dqsbuf : std_logic;
 signal dqsdel : std_logic;
 signal ddrclkpol_sig : std_logic;
 signal ddrin : std_logic_vector(7 downto 0 );
 signal ddrout : std_logic_vector(7 downto 0 );
 signal tridata : std_logic_vector(7 downto 0 );
 signal dqsout : std_logic;
 signal tridqs : std_logic;
 signal dqsin : std_logic;
 signal vcc_net : std_logic;
 signal gnd_net : std_logic;
```

```
 signal clkinv : std_logic;
 signal ddrclk : std_logic;


 begin
   vcc_net <= '1';
   gnd_net <= '0';
   clkinv<= not clk;
   ddrclkpol<=ddrclkpol_sig;

--*************BIDIRECTIONAL BUFFERS*****************************************************
    bidiInst0 : BB PORT MAP( I => ddrout(0),T => tridata(0),O => ddrin(0),B => dq(0));
    bidiInst1 : BB PORT MAP( I => ddrout(1),T => tridata(1),O => ddrin(1),B => dq(1));
    bidiInst2 : BB PORT MAP( I => ddrout(2),T => tridata(2),O => ddrin(2),B => dq(2));
    bidiInst3 : BB PORT MAP( I => ddrout(3),T => tridata(3),O => ddrin(3),B => dq(3));
    bidiInst4 : BB PORT MAP( I => ddrout(4),T => tridata(4),O => ddrin(4),B => dq(4));
    bidiInst5 : BB PORT MAP( I => ddrout(5),T => tridata(5),O => ddrin(5),B => dq(5));
    bidiInst6 : BB PORT MAP( I => ddrout(6),T => tridata(6),O => ddrin(6),B => dq(6));
    bidiInst7 : BB PORT MAP( I => ddrout(7),T => tridata(7),O => ddrin(7),B => dq(7));

    bidiInst8 : BB PORT MAP( I=> dqsout, T=> tridqs, O=> dqsin, B=> dqs);

--*****************************************************************************************

--*************DDRInput*****************************************************************
--DQSDLL, generates the DQS delay
    I0: DQSDLL PORT MAP(CLK=>clk, UDDCNTL=> uddcntl, RST=> reset, DQSDEL=> dqsdel,
                        LOCK => lock);

    I1: DQSBUFB PORT MAP( DQSI=> dqsin, CLK=>clk, READ=> read, DQSDEL=> dqsdel,
                        DDRCLKPOL=> ddrclkpol_sig, DQSC=> dqsc, PRMBDET=> prmbdet,
                        DQSO=> dqsbuf);

--DDR INPUT primitives
    I2 : IDDRXB PORT MAP(D=> ddrin(0), ECLK=> dqsbuf, SCLK => clk, CE => vcc_net,
                        DDRCLKPOL=> ddrclkpol_sig, LSR => reset, QA =>datain_p(0),
                        QB => datain_n(0));
    I3 : IDDRXB PORT MAP(D=> ddrin(1), ECLK=> dqsbuf, SCLK => clk, CE => vcc_net,
                        DDRCLKPOL=> ddrclkpol_sig, LSR => reset, QA =>datain_p(1),
                        QB => datain_n(1));
    I4 : IDDRXB PORT MAP(D=> ddrin(2), ECLK=> dqsbuf, SCLK => clk, CE => vcc_net,
                        DDRCLKPOL=> ddrclkpol_sig, LSR => reset, QA =>datain_p(2),
                        QB => datain_n(2));
    I5 : IDDRXB PORT MAP(D=> ddrin(3), ECLK=> dqsbuf, SCLK => clk, CE => vcc_net,
                        DDRCLKPOL=> ddrclkpol_sig, LSR => reset, QA =>datain_p(3),
                        QB => datain_n(3));
    I6 : IDDRXB PORT MAP(D=> ddrin(4), ECLK=> dqsbuf, SCLK => clk, CE => vcc_net,
                        DDRCLKPOL=> ddrclkpol_sig, LSR => reset, QA =>datain_p(4),
                        QB => datain_n(4));
    I7 : IDDRXB PORT MAP(D=> ddrin(5), ECLK=> dqsbuf, SCLK => clk, CE => vcc_net,
                        DDRCLKPOL=> ddrclkpol_sig, LSR => reset, QA =>datain_p(5),
                        QB => datain_n(5));
```

```
    I8 : IDDRXB PORT MAP(D=> ddrin(6), ECLK=> dqsbuf, SCLK => clk, CE => vcc_net,
                         DDRCLKPOL=> ddrclkpol_sig, LSR => reset, QA =>datain_p(6),
                         QB => datain_n(6));
    I9 : IDDRXB PORT MAP(D=> ddrin(7), ECLK=> dqsbuf, SCLK => clk, CE => vcc_net,
                         DDRCLKPOL=> ddrclkpol_sig, LSR => reset, QA =>datain_p(7),
                         QB => datain_n(7));
--*************************************************************************************

--*****TRISTATE Instantiations*****************************************************
-- DDR Trisate for data, DQ
    T0 : ODDRXB PORT MAP( DA => datatri_p(0), DB => datatri_n(0), LSR => reset,
                         CLK => clkinv, Q => tridata(0));
    T1 : ODDRXB PORT MAP( DA => datatri_p(1), DB => datatri_n(1), LSR => reset,
                         CLK => clkinv, Q => tridata(1));
    T2 : ODDRXB PORT MAP( DA=>  datatri_p(2), DB => datatri_n(2), LSR => reset,
                         CLK => clkinv, Q => tridata(2));
    T3 : ODDRXB PORT MAP( DA => datatri_p(3), DB => datatri_n(3), LSR => reset,
                         CLK => clkinv, Q => tridata(3));
    T4 : ODDRXB PORT MAP( DA => datatri_p(4), DB => datatri_n(4), LSR => reset,
                         CLK => clkinv, Q => tridata(4));
    T5 : ODDRXB PORT MAP( DA => datatri_p(5), DB => datatri_n(5), LSR => reset,
                         CLK => clkinv, Q => tridata(5));
    T6 : ODDRXB PORT MAP( DA => datatri_p(6), DB => datatri_n(6), LSR => reset,
                         CLK => clkinv, Q => tridata(6));
    T7 : ODDRXB PORT MAP( DA => datatri_p(7), DB => datatri_n(7), LSR => reset,
                         CLK => clkinv, Q => tridata(7));

--DDR Trisate for strobe, DQS
    T8: ODDRXB PORT MAP( DA =>dqstri_p, DB=> dqstri_n, LSR=> reset, CLK=> clk90,
                         Q => tridqs);
--*************************************************************************************

--***************DDR Output*********************************************************
--DQ OUTPUT
    O0 : ODDRXB PORT MAP( DA => dataout_p(0), DB  => dataout_n(0), LSR => reset,
                         CLK  => clkinv, Q => ddrout(0));
    O1 : ODDRXB PORT MAP( DA => dataout_p(1), DB  => dataout_n(1), LSR => reset,
                         CLK  => clkinv, Q => ddrout(1));
    O2 : ODDRXB PORT MAP( DA => dataout_p(2), DB  => dataout_n(2), LSR => reset,
                         CLK  => clkinv, Q => ddrout(2));
    O3 : ODDRXB PORT MAP( DA => dataout_p(3), DB  => dataout_n(3), LSR => reset,
                         CLK  => clkinv, Q => ddrout(3));
    O4 : ODDRXB PORT MAP( DA => dataout_p(4), DB  => dataout_n(4), LSR => reset,
                         CLK  => clkinv, Q => ddrout(4));
    O5 : ODDRXB PORT MAP( DA => dataout_p(5), DB  => dataout_n(5), LSR => reset,
                         CLK  => clkinv, Q => ddrout(5));
    O6 : ODDRXB PORT MAP( DA => dataout_p(6), DB  => dataout_n(6), LSR => reset,
                         CLK  => clkinv, Q => ddrout(6));
    O7 : ODDRXB PORT MAP( DA => dataout_p(7), DB  => dataout_n(7), LSR => reset,
                         CLK  => clkinv, Q => ddrout(7));
```

```
--DQS output
    O8: ODDRXB PORT MAP( DA => gnd_net, DB => vcc_net, LSR => reset, CLK => clk90,
                         Q => dqsout);

--clkp and clkn Generation

    O9 : ODDRXB PORT MAP( DA => vcc_net, DB => gnd_net, LSR => reset, CLK => clk90,
                        Q => ddrclk);
--********************************************************************************

end structure;
```

Appendix D. Generic (Non-Memory) High-Speed DDR Interface

The following HDL implements the DDR input interface using PFU registers for non-memory DDR applications.

VHDL Implementation

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

library ec;
use ec.components.all;

entity ddrin is
 port (rst : in std_logic;
       ddrclk: in std_logic;
       ddrdata: in std_logic_vector(7 downto 0);
       datap: out std_logic_vector(7 downto 0);
       datan: out std_logic_vector(7 downto 0));


end ddrin;
architecture structure of ddrin is
-- parameterized module component declaration
component pll90
    port (CLK: in std_logic; RESET: in std_logic; CLKOP: out std_logic;
        CLKOS: out std_logic; LOCK: out std_logic);
end component;


       signal pos0 : std_logic_vector( 7 downto 0 );
       signal pos1 : std_logic_vector( 7 downto 0 );
       signal neg0 : std_logic_vector( 7 downto 0 );

       signal clklock : std_logic;
       signal ddrclk0: std_logic;
       signal ddrclk90: std_logic;
       signal vcc_net : std_logic;
       signal gnd_net: std_logic;

attribute syn_useioff : boolean;
attribute syn_useioff of structure : architecture is false;

begin
       vcc_net <= '1';
       gnd_net <= '0';


       -- parameterized module component instance
       I0 : pll90
       port map (CLK=>ddrclk, RESET=>rst, CLKOP=>ddrclk0, CLKOS=>ddrclk90, LOCK=>clklock);

demux: process (rst, ddrclk90)
begin
       if  rst = '1' then
              pos0    <= (others => '0');
              neg0    <= (others => '0');
              pos1    <= (others => '0');
       elsif rising_edge(ddrclk90) then
              pos0    <= ddrdata;
```

```
        elsif falling_edge(ddrclk90) then
          neg0  <=ddrdata;
         pos1  <=pos0;
       end if;
   end  process demux;

   synch: process (rst, ddrclk90)
   begin
       if  rst = '1' then
          datap  <= (others => '0');
          datan  <= (others => '0');
       elsif rising_edge(ddrclk90) then
          datap<= pos1;
       elsif falling_edge(ddrclk90) then
             datan<= neg0;
       end if;
   end  process synch;

end structure;
```

Verilog Example

```
module ddrin (rst, ddrclk, ddrdata, datap, datan)/*synthesis syn_useioff = 0*/;
// Inputs
input            rst;
input            ddrclk;
input     [7:0]  ddrdata;
// Outputs
output    [7:0]  datap, datan;


reg [7:0] pos0/*synthesis syn_keep=1*/;
reg [7:0] pos1/*synthesis syn_keep=1*/;
reg [7:0] neg0/*synthesis syn_keep=1*/;
reg [7:0] datap, datan/*synthesis syn_keep=1*/;

//PLL signals
wire ddrclk0;
wire ddrclk90;

pll I0 (.CLK(ddrclk), .RESET(rst), .CLKOP(ddrclk0), .CLKOS(ddrclk90), .LOCK(clklock));

always @ ( posedge ddrclk90)
begin
    if (rst)
    begin
    pos0 <= 0;
    end
    else
    begin
       pos0 <= ddrdata;
    end
end

always@ (negedge ddrclk90)
begin
    if (rst)
    begin
       neg0<=0;
       pos1<=0;
    end
    else
    begin
       neg0<=ddrdata;
       pos1<=pos0;
    end
end
always @ (posedge ddrclk90)
begin
    if (rst)
    begin
       datap<= 0;
       datan<= 0;
    end
    else
    begin
       datap<= pos1;
       datan<= neg0;
    end
```

```
end
endmodule
```

Preference File

In order to run the above DDR PFU Implementation at 300MHZ, the following preferences were added to the software preference file.

```
COMMERCIAL;
FREQUENCY NET "ddrclk90" 300.000000 MHz ;
INPUT_SETUP PORT "ddrdata_0" 0.800000 ns CLKNET "ddrclk90" ;
INPUT_SETUP PORT "ddrdata_1" 0.800000 ns CLKNET "ddrclk90" ;
INPUT_SETUP PORT "ddrdata_2" 0.800000 ns CLKNET "ddrclk90" ;
INPUT_SETUP PORT "ddrdata_3" 0.800000 ns CLKNET "ddrclk90" ;
INPUT_SETUP PORT "ddrdata_4" 0.800000 ns CLKNET "ddrclk90" ;
INPUT_SETUP PORT "ddrdata_5" 0.800000 ns CLKNET "ddrclk90" ;
INPUT_SETUP PORT "ddrdata_6" 0.800000 ns CLKNET "ddrclk90" ;
INPUT_SETUP PORT "ddrdata_7" 0.800000 ns CLKNET "ddrclk90" ;
BLOCK ASYNCPATHS ;
```

Appendix E. List of Compatible DDR SDRAM

Below are the criteria used to list the DDR SDRAM part numbers.

1. The memory device should support one DQS strobe for every eight DQ data bits.
2. 4-bit, 8-bit and 16-bit configurations. For 16-bit configurations, each data byte must have an independent DQS strobe.
3. The memory device uses SSTL2 I/O interface standard.
4. Data transfer rate DDR400, DDR333 or DDR266 for LatticeECP/EC devices and DDR333 or DDR266 for LatticeXP devices.
5. Clock transfer rate of 200MHz, 167MHz or 133MHz for LatticeECP/EC devices and 167MHz or 133MHz for LatticeXP devices.

Table 10-9 lists the DDR SDRAM part numbers that can be used with the LatticeECP/EC and LatticeXP devices.

Please note these part numbers are chosen based on the criteria stated above and have not necessary been validated in hardware.

Table 10-9. List of Compatible DDR SDRAM

DDR SDRAM Vendor	Part Number	Configuration	Max Data Rate	Clock Speed
Micron 128MB	MT46V32M4TG	32Mx4	DDR266	133MHz
Micron 128MB	MT46V16M8TG	16Mx8	DDR333 DDR266	167MHz 133MHz
Micron 128MB	MT46V8M16TG	8Mx16	DDR266	133MHz
Micron 256MB	MT46V64M4FG	64Mx4	DDR400 DDR333 DDR266	200MHz 167MHz 133MHz
Micron 256MB	MT46V64M4TG	64Mx4	DDR400 DDR333 DDR266	200MHz 167MHz 133MHz
Micron 256MB	MT46V32M8FG	32Mx8	DDR400 DDR333 DDR266	200MHz 167MHz 133MHz
Micron 256MB	MT46V32M8TG	32Mx8	DDR400 DDR333 DDR266	200MHz 167MHz 133MHz
Micron 256MB	MT46V16M16FG	16Mx16	DDR266	133MHz
Micron 256MB	MT46V16M16TG	16Mx16	DDR400 DDR333 DDR266	200MHz 167MHz 133MHz
Micron 512MB	MT46V128M4FN	128Mx4	DDR400 DDR333 DDR266	200MHz 167MHz 133MHz
Micron 512MB	MT46V128M4TG	128Mx4	DDR266	133MHz
Micron 512MB	MT46V64M8FN	64Mx8	DDR400 DDR333 DDR266	200MHz 167MHz 133MHz
Micron 512MB	MT46V64M8TG	64Mx8	DDR400 DDR333 DDR266	200MHz 167MHz 133MHz
Micron 512MB	MT46V32M16FN	32Mx16	DDR400 DDR333 DDR266	200MHz 167MHz 133MHz

Table 10-9. List of Compatible DDR SDRAM (Continued)

DDR SDRAM Vendor	Part Number	Configuration	Max Data Rate	Clock Speed
Micron 512MB	MT46V32M16TG	32Mx16	DDR400 DDR333 DDR266	200MHz 167MHz 133MHz
Micron 1GB	MT46V256M4TG	256Mx4	DDR266	133MHz
Micron 1GB	MT46V128M8TG	128Mx8	DDR266	133MHz
Micron 1GB	MT46V64M16TG	64Mx16	DDR333 DDR266	167MHz 133MHz
Samsung 128MB E die	K4H280438E-TC/LB3	32Mx4	DDR333	167MHz
	K4H280838E-TC/LB3	16Mx8	DDR333	167MHz
	K4H281638E-TC/LB3	8Mx16	DDR333	167MHz
Samsung 256 Mb E-die	K4H560438E-TC/LB3	64Mx4	DDR333	167MHz
	K4H560438E-NC/LB3	64Mx4	DDR333	167MHz
	K4H560438E-GC/LB3, CC	64Mx4	DDR333 DDR400	167MHz 200MHz
	K4H560838E-TC/LB3, CC	32Mx8	DDR333 DDR400	167MHz 200MHz
	K4H560838E-NC/LB3, CC	32Mx8	DDR333 DDR400	167MHz 200MHz
	K4H560838E-GC/LB3, CC	32Mx8	DDR333 DDR400	167MHz 200MHz
Samsung 512Mb B die	K4H510838B-TC/LB3, CC	64Mx8	DDR333 DDR400	167MHz 200MHz
	K4H510838B-NC/LB3, CC	64Mx8	DDR333 DDR400	167MHz 200MHz
	K4H511638B-TC/LB3, CC	32Mx16	DDR333 DDR400	167MHz 200MHz
	K4H510438B-TC/LB3	128Mx4	DDR333	167MHz
	K4H510438B-NC/LB3	128Mx4	DDR333	167MHz
Infineon 128Mb	HYB25D128400AT	32Mx4	DDR266	133MHz
	HYB25D128400CT	32Mx4	DDR266	133MHz
	HYB25D128400CE	32Mx4	DDR266	133MHz
	HYB25D128800AT	16Mx8	DDR266	133MHz
	HYB25D128800CT	16Mx8	DDR333	167MHz
	HYB25D128800CE	16Mx8	DDR333	167MHz
	HYB25D128160AT	8Mx16	DDR333 DDR266	167MHz 133MHz
	HYB25D128160CT	8Mx16	DDR333 DDR400	167MHz 200MHz
	HYB25D128160CE	8Mx16	DDR333 DDR400	167MHz 200MHz
	HYB25D128160CC	8Mx16	DDR333	167MHz

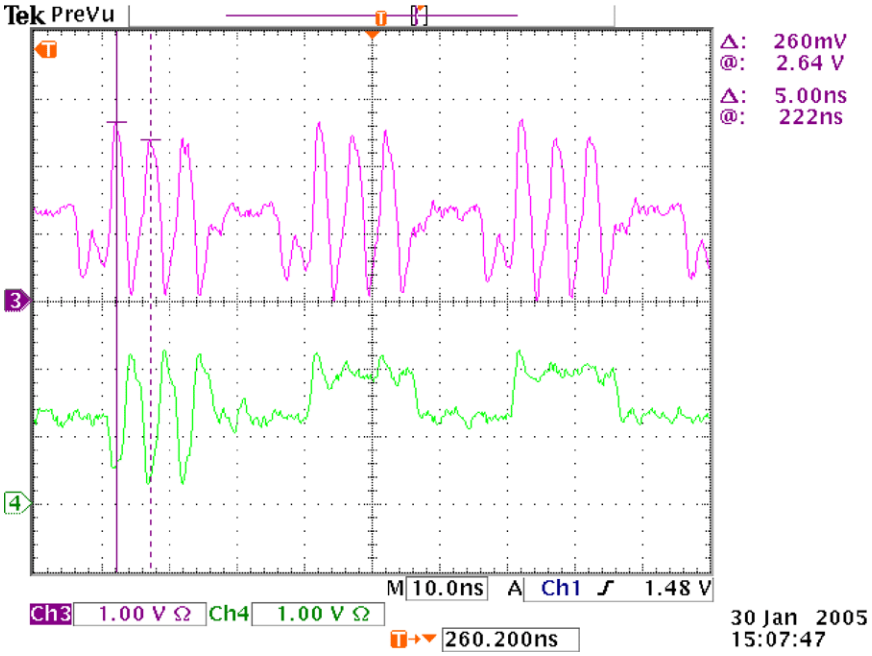
Table 10-9. List of Compatible DDR SDRAM (Continued)

DDR SDRAM Vendor	Part Number	Configuration	Max Data Rate	Clock Speed
Infineon 256Mb	HYB25D256400BT	64Mx4	DDR266	133MHz
	HYB25D256400CT	64Mx4	DDR266	133MHz
	HYB25D256400CE	64Mx4	DDR266	133MHz
	HYB25D256800BT	32Mx8	DDR333	167MHz
	HYB25D256800CT	32Mx8	DDR333	167MHz
	HYB25D256800CE	32Mx8	DDR333 DDR400	167MHz 200MHz
	HYB25D256160BT	16Mx16	DDR400 DDR333 DDR266	200MHz 167MHz 133MHz
	HYB25D256160CT	16Mx16	DDR333 DDR400	167MHz 200MHz
	HYB25D256160CE	16Mx16	DDR333 DDR400	167MHz 200MHz
	HYB25D256400BC	64Mx16	DDR400 DDR333 DDR266	200MHz 167MHz 133MHz
	HYB25D256400CF	64Mx16	DDR333	167MHz
	HYB25D256400CC	64Mx16	DDR333	167MHz
	HYB25D256160BC	16Mx16	DDR333 DDR266	167MHz 133MHz
	HYB25D256160CC	16Mx16	DDR333 DDR400	167MHz 200MHz
Infineon 512Mb	HYB25D512400AT	128Mx4	DDR266	133MHz
	HYB25D512400BT	128Mx4	DDR333	167MHz
	HYB25D512400BE	128Mx4	DDR333	167MHz
	HYB25D1G400BG	256Mx4	DDR266	133MHz
	HYB25D512800AT	64Mx8	DDR333 DDR266	167MHz 133MHz
	HYB25D512800BT	64Mx8	DDR333 DDR400	167MHz 200MHz
	HYB25D512800BE	64Mx8	DDR333 DDR400	167MHz 200MHz
	HYB25D512160AT	32Mx16	DDR333 DDR266	167MHz 133MHz
	HYB25D512160BT	32Mx16	DDR333 DDR400	167MHz 200MHz
	HYB25D512160BE	32Mx16	DDR333 DDR400	167MHz 200MHz
	HYB25D512400BC	128Mx4	DDR333 DDR400	167MHz 200MHz
	HYB25D512400BF	128Mx4	DDR333 DDR400	167MHz 200MHz
	HYB25D512800BC	64Mx8	DDR333	167MHz
	HYB25D512800BF	64Mx8	DDR333 DDR400	167MHz 200MHz
	HYB25D512160BC	32Mx16	DDR333 DDR400	167MHz 200MHz
	HYB25D512160BF	32Mx16	DDR333 DDR400	167MHz 200MHz

Appendix F. DDR400 Interface using the LatticeEC Evaluation Board

The DDR400 interface was implemented using the LatticeEC20 device on the LatticeEC Advanced Evaluation Board. Figures 10-21, 10-22 and 10-23 show the READ, WRITE and WRITE to READ transition operations running at 200MHz. For more information on the evaluation board, refer to *LatticeEC Advanced Evaluation Board User's Guide* available on the Lattice web site at www.latticesemi.com.

Figure 10-21. READ Function Running at 200MHz



Note: An extra READ command is implemented in the LatticeEC20 device to protect the data during postamble. This extra READ is not required for other LatticeEC devices. Refer to the DQS Postamble section of this document for more information.

Figure 10-22. WRITE Function Running at 200MHz

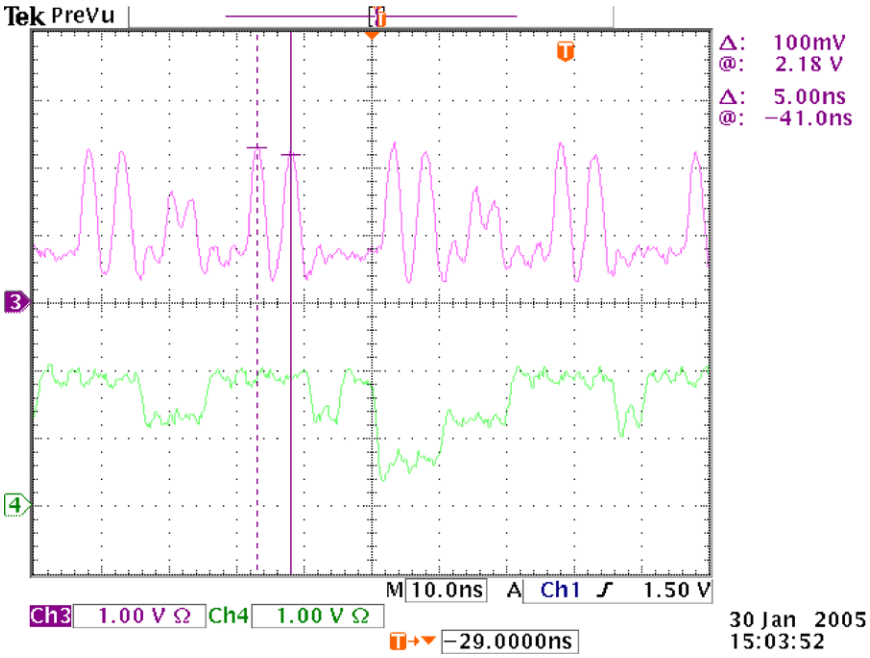
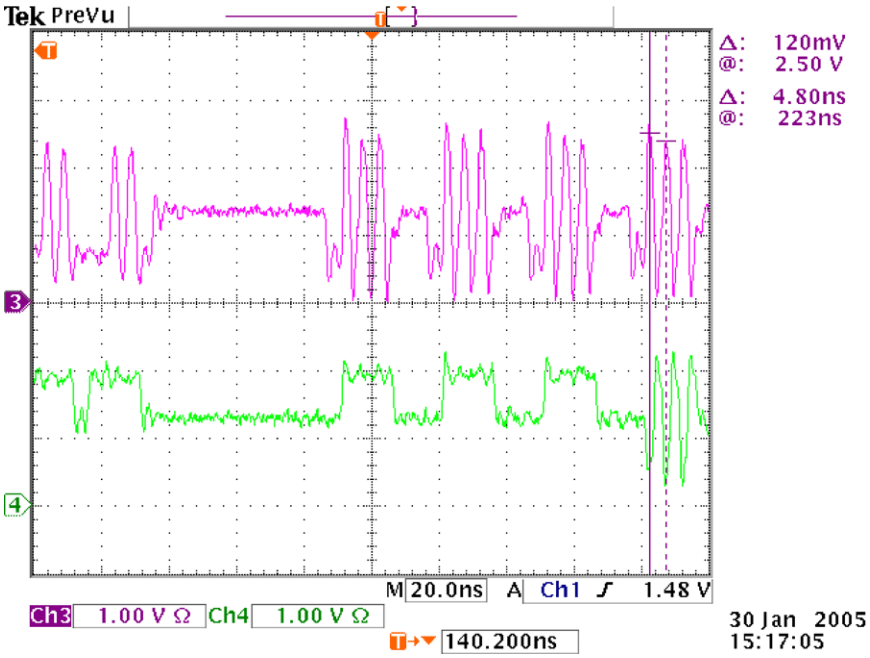


Figure 10-23. WRITE to READ Transition Running at 200MHz



Note: An extra READ command is implemented in the LatticeEC20 device to protect the data during postamble. This extra READ is not required for other LatticeEC devices. Refer to the DQS Postamble section of this document for more information.