Contents

Revision History ................................................................. v
How to Contact Altera .......................................................... v
Typographic Conventions ..................................................... vi

Chapter 1. About These Megafunctions
Device Family Support ......................................................... 1–1
Introduction ............................................................................. 1–1
Features .................................................................................... 1–2
General Description .............................................................. 1–2
   Common Applications .......................................................... 1–2
       DDR SDRAM, DDR2 SDRAM & RLDRAM II Memory Interfaces ........................................ 1–3
       QDR SRAM & QDRII SRAM Memory Interfaces ......................................................... 1–3
       High-Speed Interface Applications ............................................................................ 1–3
DDR Device Configuration .................................................... 1–4
   Input Configuration .............................................................. 1–4
   Output Configuration ............................................................ 1–5
   Bidirectional Configuration .................................................. 1–8
DDR I/O Timing ........................................................................ 1–10
Resource Utilization & Performance ...................................... 1–11

Chapter 2. Getting Started
System & Software Requirements .......................................... 2–1
MegaWizard Plug-In Manager Customization ............................ 2–1
MegaWizard Page Description .............................................. 2–2
   altddio_in Megafuction Configuration .................................................... 2–4
   altddio_out Megafuction Configuration ..................................................... 2–5
   altddio_bidir Megafuction Configuration ................................................. 2–6
   altddio_in, altddio_out & altddio_bidir Simulation Libraries ................................. 2–8
Inferring Megafunctions from HDL Code .............................. 2–9
Instantiating Megafunctions in HDL Code .............................. 2–10
   EDA Simulation ........................................................................ 2–10
SignalTap II Embedded Logic Analyzer .................................. 2–10
Design Example 1: 8-Bit DDR Divider Using altddio_in & altddio_out ........................................ 2–11
   Design Files .......................................................................... 2–11
   Example .............................................................................. 2–11
   Generate a Divider Using altddio_in & altddio_out .............................................. 2–11
       Create the altddio_in Module ...................................................................... 2–11
       Create the altddio_out Module .................................................................. 2–16
       Create the lpm_divide Module .................................................................. 2–20
   Create a Divider ....................................................................... 2–25
   Implement the Divider Design ................................................................ 2–26
Contents

Chapter 3. Specifications

Ports & Parameters .......................................................... 3–1
  Ports & Parameters for the altddio_in Megafunction .................. 3–1
  Ports & Parameters for the altddio_out Megafunction .................. 3–3
  Ports & Parameters for the altddio_bidir Megafunction ................. 3–5
Revision History
The table below displays the revision history for the chapters in this User Guide.

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Date</th>
<th>Document Version</th>
<th>Changes Made</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>July 2006</td>
<td>4.0</td>
<td>Updated to reflect Quartus II 6.0 release, added ModelSim simulation information, updated design examples</td>
</tr>
<tr>
<td>All</td>
<td>March 2005</td>
<td>3.0</td>
<td>Updated to reflect new GUI changes</td>
</tr>
<tr>
<td>All</td>
<td>December 2004</td>
<td>2.0</td>
<td>Updated to reflect new document organization and GUI changes</td>
</tr>
</tbody>
</table>

How to Contact Altera
For the most up-to-date information about Altera® products, go to the Altera web site at www.altera.com. For technical support about this product, go to www.altera.com/mysupport. For additional information about Altera products, consult the sources shown below.

<table>
<thead>
<tr>
<th>Information Type</th>
<th>USA &amp; Canada</th>
<th>All Other Locations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(800) 800-EPLD (3753) (7:00 a.m. to 5:00 p.m. Pacific Time)</td>
<td>+1 408-544-8767/(1) 7:00 a.m. to 5:00 p.m. (GMT -8:00) Pacific Time</td>
</tr>
<tr>
<td>Product literature</td>
<td><a href="http://www.altera.com">www.altera.com</a> (1)</td>
<td><a href="http://www.altera.com">www.altera.com</a> (1)</td>
</tr>
<tr>
<td>Alreta literature services</td>
<td><a href="mailto:literature@altera.com">literature@altera.com</a></td>
<td><a href="mailto:literature@altera.com">literature@altera.com</a></td>
</tr>
<tr>
<td>Non-technical customer service</td>
<td>(800) 767-3753</td>
<td>+ 1 408-544-7000 7:00 a.m. to 5:00 p.m. (GMT -8:00) Pacific Time</td>
</tr>
<tr>
<td>FTP site</td>
<td>ftp.altera.com</td>
<td>ftp.altera.com</td>
</tr>
</tbody>
</table>

Note to table:
(1) You can also contact your local Altera sales office or sales representative.
Typographic Conventions

This document uses the typographic conventions shown below.

<table>
<thead>
<tr>
<th>Visual Cue</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bold Type with Initial Capital Letters</strong></td>
<td>Command names, dialog box titles, checkbox options, and dialog box box options are shown in bold, initial capital letters. Example: <strong>Save As</strong> dialog box.</td>
</tr>
<tr>
<td><strong>bold type</strong></td>
<td>External timing parameters, directory names, project names, disk drive names, filenames, filename extensions, and software utility names are shown in bold type. Examples: (f_{MAX}), \qdesigns\ directory, (d:) drive, \chiptrip.gdf\ file.</td>
</tr>
<tr>
<td><strong>Italic Type with Initial Capital Letters</strong></td>
<td>Document titles are shown in italic type with initial capital letters. Example: <em>AN 75: High-Speed Board Design</em>.</td>
</tr>
<tr>
<td><strong>Italic type</strong></td>
<td>Internal timing parameters and variables are shown in italic type. Examples: (t_{PIA}), (n + 1).</td>
</tr>
<tr>
<td></td>
<td>Variable names are enclosed in angle brackets (&lt; &gt;) and shown in italic type. Example: &lt;file name&gt;, &lt;project name&gt;.pof file.</td>
</tr>
<tr>
<td>Initial Capital Letters</td>
<td>Keyboard keys and menu names are shown with initial capital letters. Examples: Delete key, the Options menu.</td>
</tr>
<tr>
<td>&quot;Subheading Title&quot;</td>
<td>References to sections within a document and titles of on-line help topics are shown in quotation marks. Example: &quot;Typographic Conventions.&quot;</td>
</tr>
<tr>
<td><strong>Courier type</strong></td>
<td>Signal and port names are shown in lowercase Courier type. Examples: data1, tdi, input. <strong>Active-low signals are denoted by suffix n, e.g., resetn.</strong></td>
</tr>
<tr>
<td></td>
<td>Anything that must be typed exactly as it appears is shown in Courier type. For example: \c:\qdesigns\tutorial\chiptrip.gdf. Also, sections of an actual file, such as a Report File, references to parts of files (e.g., the AHDL keyword \SUBDESIGN), as well as logic function names (e.g., TRI) are shown in Courier.</td>
</tr>
<tr>
<td>1., 2., 3., and a., b., c., etc.</td>
<td>Numbered steps are used in a list of items when the sequence of the items is important, such as the steps listed in a procedure.</td>
</tr>
<tr>
<td></td>
<td><strong>Bullets</strong> are used in a list of items when the sequence of the items is not important.</td>
</tr>
<tr>
<td></td>
<td>The checkmark indicates a procedure that consists of one step only.</td>
</tr>
<tr>
<td></td>
<td>The hand points to information that requires special attention.</td>
</tr>
<tr>
<td></td>
<td>The caution indicates required information that needs special consideration and understanding and should be read prior to starting or continuing with the procedure or process.</td>
</tr>
<tr>
<td></td>
<td>The warning indicates information that should be read prior to starting or continuing the procedure or processes.</td>
</tr>
<tr>
<td></td>
<td>The angled arrow indicates you should press the Enter key.</td>
</tr>
<tr>
<td></td>
<td>The feet direct you to more information about a particular topic.</td>
</tr>
</tbody>
</table>
Device Family Support

Megafunctions provide either full or preliminary support for target Altera® device families, as described below:

- **Full support** means the megafunction meets all functional and timing requirements for the device family and may be used in production designs.
- **Preliminary support** means the megafunction meets all functional requirements, but may still be undergoing timing analysis for the device family; it may be used in production designs with caution.

Table 1–1 shows the level of support offered by the Altera double data rate (DDR) megafunctions, altddio_in, altddio_out, and altddio_bidir, for each Altera device family.

<table>
<thead>
<tr>
<th>Device Family</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stratix® II</td>
<td>Full</td>
</tr>
<tr>
<td>Stratix GX</td>
<td>Full</td>
</tr>
<tr>
<td>Stratix II GX</td>
<td>Full</td>
</tr>
<tr>
<td>Stratix</td>
<td>Full</td>
</tr>
<tr>
<td>HardCopy Stratix</td>
<td>Full</td>
</tr>
<tr>
<td>HardCopy II</td>
<td>Full</td>
</tr>
<tr>
<td>Cyclone™ II</td>
<td>Full</td>
</tr>
<tr>
<td>Cyclone</td>
<td>Full</td>
</tr>
<tr>
<td>APEX™ II</td>
<td>Full</td>
</tr>
</tbody>
</table>

Introduction

As design complexities increase, use of vendor-specific intellectual property (IP) blocks has become a common design methodology. Altera provides parameterizable megafunctions that are optimized for Altera device architectures. Using megafunctions instead of coding your own logic saves valuable design time. Additionally, the Altera-provided functions may offer more efficient logic synthesis and device implementation. You can scale the megafunction’s size by simply setting parameters.
**Features**

The **altddio** megafuctions implement a DDR interface and offer many additional features, which include:

- The **altddio_in** megafuction receives data on both edges of the reference clock
- The **altddio_out** megafuction transmits data on both edges of the reference clock
- The **altddio_bidir** megafuction transmits and receives data on both edges of the reference clock
- Asynchronous clear and asynchronous set input options available
- **inclock** signal to sample the DDR input
- **outclock** signal to register the data output
- Clock enable signals
- Bidirectional port for **altddio_bidir** megafuctions
- An output enable input for **altddio_out** and **altddio_bidir** megafuctions

**General Description**

The Altera DDR megafuctions let you configure the DDR I/O registers in Stratix and Cyclone series, APEX II, and Mercury devices. You can also use the megafuctions to implement DDR registers in the logic elements (LEs). In Stratix series and APEX II devices, the DDR registers are implemented in the I/O element (IOE). In Cyclone series devices, the megafuctions automatically implement the DDR registers in the LEs closest to the pin. In Mercury devices, DDR inputs and outputs are implemented in the IOE, and bidirectional DDR uses a combination of IOEs and LEs. The **altddio_in** megafuction implements the interface for DDR inputs. The **altddio_out** megafuction implements the interface for DDR outputs. The **altddio_bidir** megafuction implements the interface for bidirectional DDR inputs and outputs.

**Common Applications**

DDR registers capture and/or send data at twice the rate of the clock or data strobe to interface with a memory device or other high-speed interface application in which the data is clocked at both edges of the clock. The DDR registers interface with DDR SDRAM, DDR2 SDRAM, RLDRAM II, QDR SRAM, and QDRII SRAM memory devices. You can also use the DDR I/O registers as a SERDES bypass mechanism in LVDS applications. This section provides information about the following DDR I/O applications:

- DDR SDRAM, DDR2 SDRAM, and RLDRAM II memory interfaces
- QDR SRAM and QDRII SRAM memory interfaces
- High-speed interface applications
**DDR SDRAM, DDR2 SDRAM & RLDRAM II Memory Interfaces**

DDR SDRAM, DDR2 SDRAM, and RLDRAM II write and read data at twice the clock rate by capturing data on both the positive and negative edge of a clock. DDR and DDR2 SDRAM are JEDEC standards. RLDRAM II devices have minimal latency to support designs that require fast response times. These DDR memory interfaces use a variety of I/O standards such as SSTL-II, 1.8-V HSTL, LVTTL, and LVCMOS.

The DDR and DDRII SDRAM controller is available by downloading the Altera DDR SDRAM Controller MegaCore® function from www.altera.com.

**QDR SRAM & QDRII SRAM Memory Interfaces**

The QDR and QDRII SRAM standard is defined jointly by Cypress, IDT, and Micron. QDR and QDRII SRAMs have separate DDR read and write ports that pass data concurrently. The combination of concurrent transactions and DDR signaling allows data to be passed four times faster than conventional SRAMs. The I/O standards used for QDR SRAM devices are 1.5-V HSTL class I and II. QDRII SRAMs use both 1.5-V and 1.8-V HSTL class I.

The QDR II SDRAM controller is available by downloading the Altera QDR II SDRAM Controller MegaCore function from www.altera.com.

**High-Speed Interface Applications**

High-speed interface applications use various differential standards such as LVDS, LVPECL, PCML, or HyperTransport technology to transfer data. These standards often use DDR data. Stratix series devices implement high-speed standards either by using the dedicated differential I/O SERDES blocks or by bypassing SERDES and using the DDR I/O circuitry in SERDES bypass mode. DDR megafunctions, PLLs, and shift registers are all used in SERDES functionality.

For more information, refer to the following resources:

- The *External Memory Interfaces in Stratix II & Stratix II GX Devices* chapter in volume 2 of the *Stratix II Device Handbook*
- The *Implementing Double Data Rate I/O Signaling in Cyclone Devices* chapter in volume 1 of the *Cyclone Device Handbook*
- *AN 167: Using Flexible-LVDS I/O Pins in APEX II Devices*
- *AN 186: Using Flexible Circuitry in Mercury Devices*
The following sections describe how the DDR registers are configured in the Stratix series and APEX II devices.

Input Configuration

When the IOE is configured as an input pin, input registers AI, BI, and latch CI implement the input path for DDR I/O. Figure 1–1 shows an IOE configured for DDR inputs for a Stratix series or APEX II device.

Figure 1–1. Input DDR I/O Path Configuration for a Stratix Series or APEX II Device

Note to Figure 1–1:
(1) On the falling edge of the clock, the negative-edge triggered register B₁ acquires the first data bit. On the corresponding rising edge of the clock, the positive-edge triggered register A₁ acquires the second data bit. For a successful data transfer to the logic array, the latch C₁ synchronizes the data from register B₁ to the positive edge of the clock.
Figure 1–2 shows an IOE configured for DDR inputs for a Stratix or Stratix II device.

**Figure 1–2. Stratix II IOE in DDR Input I/O Configuration Notes (1), (2), (3)**

Notes to Figure 1–2:
(1) All input signals to the IOE can be inverted at the IOE.
(2) This signal connection is only allowed on dedicated DQ function pins.
(3) This signal is for dedicated DQS function pins only.
(4) The optional PCI clamp is only available on column I/O pins.

**Output Configuration**

The dedicated output registers for Stratix series and APEX II devices are labeled \( A_O \) and \( B_O \). These positive-edge triggered registers and a multiplexer are used to implement the output path for DDR I/O. **Figure 1–3** shows the IOE configuration for DDR outputs in Stratix series and APEX II devices.
**Figure 1–3. Output DDR I/O Path Configuration for Stratix Series & APEX II Devices**

**Notes to Figure 1–3:**
(1) The OE is active low, but the Quartus® II software implements this as active high and automatically adds an inverter before the input to the AOE register during compilation. If desired, you can change the OE back to active low.
(2) Register AOE generates the enable signal for general-purpose DDR I/O applications.
(3) This select line corresponds to the delay switch-on by a half clock cycle option in the MegaWizard® Plug-In Manager.
(4) Register BOE generates the delayed enable signal for DDR SDRAM applications.
(5) The tri-state is active high by default. However, you can design it to be active low.
About These Megafunctions

On the positive edge of the clock, a high data bit and a low data bit are captured in registers \(A_O\) and \(B_O\). The outputs of these two registers are fed to the input of a 2-to-1 multiplexer, which uses the output register clock as its control signal. A high clock selects the data in register \(B_O\), and a low level of the clock selects the data in register \(A_O\). This process doubles the data at the I/O pin.

Figure 1–4 shows the IOE configuration for DDR outputs in Stratix series devices.
Bidirectional Configuration

Input and output registers are independent of each other, enabling the bidirectional DDR I/O path to be implemented entirely in the I/O element for Stratix, Stratix GX, and APEX II devices. The bidirectional configuration includes an input path, an output path, and two output enable registers.

The bidirectional path consists of two data flow paths:

- Input path active
- Output path active

When the input path is active, the output enable disables the tri-state buffer, which prevents data from being sent out on the output path. Disabling the tri-state buffer prevents contention at the I/O pin. The input path behaves like the input configuration as shown in Figure 1–1 on page 1–4. When the output path is active, the output enable register AOE controls the flow of data from the output registers. During outgoing transactions, the bidirectional configuration behaves like the output configuration as shown in Figure 1–3 on page 1–6. The second output enable register (BOE) is used for DDR SDRAM interfaces. This negative-edge register extends the high-impedance state of the pin by a half clock cycle. This option is useful to provide the write preamble for the DQS strobe in the DDR SDRAM interfaces. This feature is enabled by using the Delay switch-on by a half clock cycle option in the altddio_bidir megafunction in the Quartus II software. You can bypass the input registers and latch to get a combinational output (combout) from the pin going into the APEX II or Stratix series device. Furthermore, the input data ports (dataout_h and dataout_l) can be disabled. These features are especially useful for generating data strobes like DQS.

Figure 1–5 shows the bidirectional DDR I/O configuration for Stratix series and APEX II devices.
About These Megafunctions

Figure 1–5. Bidirectional DDR I/O Path Configuration *Note (1)*

Notes to Figure 1–5:

1. All control signals can be inverted at the I\(_{\text{OE}}\).
2. The OE signal is active low, but the Quartus II software implements this as active high and automatically adds an inverter before input to the A\(_{\text{OE}}\) register during compilation. If desired, you can change the OE back to active low.
3. The A\(_{\text{OE}}\) register generates the enable signal for general-purpose DDR I/O applications.
4. This line selects whether the OE signal should be delayed by half-a-clock cycle.
5. The B\(_{\text{OE}}\) register generates the delayed enable signal for the write strobes or write clocks for memory interfaces.
6. The tri-state enable is by default active low. You can, however, design it to be active high.
7. You can also have combinational output to the I/O pin. This path is not shown in the diagram.
For more information about clock signals and output enable signals for Stratix series or APEX II devices, refer to the following sources:

- The *Stratix II Architecture* chapter in volume 1 of the *Stratix II Device Handbook*
- *APEX II Programmable Logic Device Family Data Sheet*

For more information about the DDR registers in Cyclone and Mercury devices, refer to the following sources:

- The *Implementing Double Data Rate I/O Signaling in Cyclone Devices* chapter in volume 1 of the *Cyclone Device Handbook*
- *Mercury Programmable Logic Device Data Sheet*

**DDR I/O Timing**

Figure 1–6 shows the functional timing waveform for the input path. The signal names are the port names used in the *altddio_in* megafunction. The *datain* signal is the input from the pin to the DDR circuitry. The output of register $B_I$ is $neg\_reg\_out$. The output of latch $C_I$ is $dataout\_h$, and the output of register $A_I$ is $dataout\_l$. $dataout\_h$ and $dataout\_l$ feed the logic array and show the conversion of the data from a DDR implementation to positive-edge triggered data.

**Figure 1–6. DDR I/O Input Timing Waveform**

Figure 1–7 shows a functional timing waveform example for the output path with the output enable registered. In this example, the delay switch-on by a half clock cycle is not turned on, so the second output enable register ($BOE$) is not used. The output enable signal $OE$ is active high and can be driven from a pin or internal logic. The data signals $datain\_l$ and $datain\_h$ are driven from the logic array to output registers $A_O$ and $B_O$. The $dataout$ signal is the output from the DDR circuitry to the pin.

Figure 1–7 shows a functional timing waveform example for the output path with the output enable registered. In this example, the delay switch-on by a half clock cycle is not turned on, so the second output enable register ($BOE$) is not used. The output enable signal $OE$ is active high and can be driven from a pin or internal logic. The data signals $datain\_l$ and $datain\_h$ are driven from the logic array to output registers $A_O$ and $B_O$. The $dataout$ signal is the output from the DDR circuitry to the pin.
About These Megafunctions

Figure 1–7. DDR I/O Output Timing Waveform

The waveform in Figure 1–7 reflects the software simulation results. The OE signal is active low in silicon; however, the Quartus II software implements this as active high and automatically adds an inverter before the D input of the OE register AOE. You can change the OE back to active low, if desired.

Resource Utilization & Performance

In Stratix series and APEX II devices, the DDR registers are implemented in the IOE of the I/O elements.

Table 1–2 summarizes the resource usage of the altddio megafunctions when implementing DDR functions in Stratix and Stratix II devices with the default settings in the MegaWizard Plug-In Manager.

<table>
<thead>
<tr>
<th>Device family</th>
<th>Optimization (1)</th>
<th>Width</th>
<th>Megafuction</th>
<th>Logic Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stratix</td>
<td>Balanced</td>
<td>8-bit</td>
<td>altddio_in</td>
<td>16 registers</td>
</tr>
<tr>
<td></td>
<td>Balanced</td>
<td>8-bit</td>
<td>altddio_out</td>
<td>16 registers</td>
</tr>
<tr>
<td></td>
<td>Balanced</td>
<td>8-bit</td>
<td>altddio_bidir</td>
<td>24 LEs</td>
</tr>
<tr>
<td>Stratix II</td>
<td>Balanced</td>
<td>8-bit</td>
<td>altddio_in</td>
<td>8 registers</td>
</tr>
<tr>
<td></td>
<td>Balanced</td>
<td>8-bit</td>
<td>altddio_out</td>
<td>8 registers</td>
</tr>
<tr>
<td></td>
<td>Balanced</td>
<td>8-bit</td>
<td>altddio_bidir</td>
<td>24 ALUTs</td>
</tr>
</tbody>
</table>

Note to Table 1–2:
(1) Choose a design implementation that balances high performance with minimal logic usage. This setting is available for APEX 20K, Cyclone, Cyclone II, MAX® II, Stratix, and Stratix II devices only. The balanced optimization logic option is set in Analysis and Synthesis settings (Assignments menu).
In Cyclone devices, the megafunctions automatically implement the DDR registers in the LEs closest to the pin.

Table 1–3 summarizes the resource usage of the altddio megafunctions when implementing DDR functions in Cyclone series, APEX II, and Mercury devices with the default settings in the MegaWizard Plug-In Manager.

<table>
<thead>
<tr>
<th>Device family</th>
<th>Optimization (1)</th>
<th>Width</th>
<th>Megafuntion</th>
<th>Logic Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cyclone II</td>
<td>Balanced</td>
<td>8-bit</td>
<td>altddio_in</td>
<td>24 LEs</td>
</tr>
<tr>
<td></td>
<td>Balanced</td>
<td>8-bit</td>
<td>altddio_out</td>
<td>16 LEs</td>
</tr>
<tr>
<td></td>
<td>Balanced</td>
<td>8-bit</td>
<td>altddio_bidir</td>
<td>40 LEs</td>
</tr>
<tr>
<td>Cyclone</td>
<td>Balanced</td>
<td>8-bit</td>
<td>altddio_in</td>
<td>24 LEs</td>
</tr>
<tr>
<td></td>
<td>Balanced</td>
<td>8-bit</td>
<td>altddio_out</td>
<td>16 LEs</td>
</tr>
<tr>
<td></td>
<td>Balanced</td>
<td>8-bit</td>
<td>altddio_bidir</td>
<td>32 LEs</td>
</tr>
<tr>
<td>APEX II</td>
<td>Balanced</td>
<td>8-bit</td>
<td>altddio_in</td>
<td>8 registers</td>
</tr>
<tr>
<td></td>
<td>Balanced</td>
<td>8-bit</td>
<td>altddio_out</td>
<td>8 registers</td>
</tr>
<tr>
<td></td>
<td>Balanced</td>
<td>8-bit</td>
<td>altddio_bidir</td>
<td>32 registers</td>
</tr>
<tr>
<td>Mercury</td>
<td>Speed (2)</td>
<td>8-bit</td>
<td>altddio_in</td>
<td>16 registers</td>
</tr>
<tr>
<td></td>
<td>Speed (2)</td>
<td>8-bit</td>
<td>altddio_out</td>
<td>16 registers</td>
</tr>
<tr>
<td></td>
<td>Speed (2)</td>
<td>8-bit</td>
<td>altddio_bidir</td>
<td>40 registers</td>
</tr>
<tr>
<td></td>
<td>Area (3)</td>
<td>8-bit</td>
<td>altddio_in</td>
<td>16 registers</td>
</tr>
<tr>
<td></td>
<td>Area (3)</td>
<td>8-bit</td>
<td>altddio_out</td>
<td>16 registers</td>
</tr>
<tr>
<td></td>
<td>Area (3)</td>
<td>8-bit</td>
<td>altddio_bidir</td>
<td>40 registers</td>
</tr>
</tbody>
</table>

Notes to Table 1–3:
(1) Choose a design implementation that balances high performance with minimal logic usage. This setting is available for APEX 20K, Cyclone, Cyclone II, MAX II, Stratix, and Stratix II devices only. The balanced optimization logic option is set in Analysis and Synthesis settings (Assignments menu).
(2) Select Speed for a design implementation with the fastest fMAX. This optimization logic option is set in Analysis and Synthesis settings (Assignments menu).
(3) Select Area to make your design as small as possible, minimizing resource usage. This optimization logic option is set in Analysis and Synthesis settings (Assignments menu).
2. Getting Started

System & Software Requirements

The instructions in this section require the following hardware and software:

- A PC running Windows NT/2000/XP, Red Hat Linux 7.3 or 8.0, or Red Hat Linux Enterprise 3, or an HP workstation running the HP-UX 11.0 operating system, or a Sun workstation running the Solaris 8 or 9 operating system
- Quartus® II software beginning with version 6.0

MegaWizard Plug-In Manager Customization

Use the MegaWizard® Plug-In Manager to create or modify design files containing custom megafunction variations which can then be instantiated in a design file. The MegaWizard Plug-In Manager is a wizard to set the `altddio_in`, `altddio_out`, and `altddio_bidir` megafunction features in the design.

Start the MegaWizard Plug-In Manager in one of the following ways:

- On the Tools menu, choose the MegaWizard Plug-In Manager command.
- When working in the Block Editor, on the Edit menu, in the Symbol dialog box, click MegaWizard Plug-In Manager.
- Start the stand-alone version of the MegaWizard Plug-In Manager by typing the following command at the command prompt:

  `qmegawiz`
This section provides descriptions of the options available on the individual pages of the DDR MegaWizard Plug-In Manager.

In page 1 of the MegaWizard Plug-In Manager, you can create, edit, or copy a custom megafuntion variation (Figure 2–1).

**Figure 2–1. Create, Edit, or Copy a Megafuntion Variation**

On page 2a (Figure 2–2), you specify the device family you want to use, type of output file to create, and the name of the output file.

Select the megafuntion you’re creating from the list on the left of the window. Choose AHDL (.tdf), VHDL (.vhd), or Verilog HDL (.v) as the output file type (Figure 2–2).
Generate the required variation of each DDR megafunction using the wizard. Click **Documentation** (Figure 2–3) for more information.

Each DDR megafunction has its own sequence of pages in the wizard where you specify the required configuration details. These pages are described in the following sections.
**altddio_in Megafuction Configuration**

On page 3 of the altddio_in MegaWizard Plug-In Manager, you specify customizable parameters for the device family, data bus width, the type of reset, and the clock enable option (Figure 2–3). A description of each function is shown in Table 2–1.

---

**Figure 2–3. altddio_in Wizard, Page 3**

![altddio_in Wizard, Page 3](image)

**Table 2–1. altddio_in MegaWizard Plug-in Manager Page 3 Options**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Which device family will you be using?</td>
<td>Specify the Altera® device family you are using.</td>
</tr>
<tr>
<td>How wide should the data buses be?</td>
<td>Specify the width of the data buses.</td>
</tr>
<tr>
<td>Which asynchronous reset port would you like?</td>
<td>Use asynchronous clear (aclr) or asynchronous preset (aset) as asynchronous reset. If you do not use either clear option, you must specify whether registers should power up high or low.</td>
</tr>
<tr>
<td>Create a clock enable port.</td>
<td>You can add a clock enable port to control when data is clocked in. This signal prevents data from being passed through.</td>
</tr>
<tr>
<td>Invert input clock</td>
<td>When enabled, the first bit of data is captured on the rising edge of the input clock. If not enabled, the first bit of data is captured on the falling edge of the input clock.</td>
</tr>
</tbody>
</table>
altddio_out Megafuinction Configuration

On page 3 of the altddio_out MegaWizard Plug-In Manager, you specify customizable parameters for the device family, data bus width, and the type of reset, clock enable port, an output enable port with the option to register the port, and the ability to extend the high-impedance state for a half clock cycle (Figure 2–4). A description of each function is shown in Table 2–2.

**Figure 2–4. altddio_out Wizard, Page 3**

![Figure 2–4. altddio_out Wizard, Page 3](Image)

**Table 2–2. altddio_out MegaWizard Plug-in Manager Page 3 Options (Part 1 of 2)**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Currently selected device family</td>
<td>Specify the Altera device family you are using.</td>
</tr>
<tr>
<td>How wide should the buses be?</td>
<td>Specify width of data buses.</td>
</tr>
<tr>
<td>Which asynchronous reset port would you like?</td>
<td>You can use asynchronous clear (aclr) or asynchronous preset (aset) as asynchronous reset. If you do not use either option, you must specify whether the registers should power up high or low.</td>
</tr>
</tbody>
</table>
The `altddio_bidir` megafunction combines `altddio_in` and `altddio_out` into a single megafunction, which instantiates bidirectional DDR ports. On page 3 of the `altddio_bidir` wizard, specify the device family, bus width, and port options (Figure 2-5).

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create a clock enable port for each clock</td>
<td>Creates a clock enable port for each clock port.</td>
</tr>
<tr>
<td>port.</td>
<td></td>
</tr>
<tr>
<td>Create an output enable port, register output</td>
<td>Creates an output enable port. You can also register the port and extend</td>
</tr>
<tr>
<td>output enable, and set delay switch-on.</td>
<td>the high-impedance state for a half clock cycle.</td>
</tr>
</tbody>
</table>

**altddio_bidir Megafuction Configuration**

Create a clock enable port for each clock port.

Create an output enable port, register output enable, and set delay switch-on.

![Figure 2-5. altddio_bidir Wizard, Page 3](image)
The options for `altddio_bidir` are the same as for `altddio_out` with the following additions:

- An option for an unregistered data port
- Ability to implement the `altddio_bidir` megafunction input path in logic elements (LEs)
- Ability to register and delay the OE signal (by a half clock cycle) if you use the OE port. The `altddio_bidir` megafunction behaves similarly to the `altddio_out` megafunction. This does not apply to Mercury devices.

If you use the `altddio_bidir` megafunction for your DQS signal in an external memory interface, you route the undelayed DQS signal to the LE (Stratix and Stratix II devices only).

Table 2–3 shows the features and settings on page 3 of the `altddio_bidir` megafunction wizard.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create for which device family?</td>
<td>Specify the Altera device family you are using.</td>
</tr>
<tr>
<td>How wide should the buses be?</td>
<td>Specify width of data buses.</td>
</tr>
<tr>
<td>Which asynchronous reset port would you like?</td>
<td>You can use asynchronous clear (<code>aclr</code>) or asynchronous preset (<code>aset</code>) as asynchronous reset. If you do not use either clear option, you must specify whether the should power up high or low.</td>
</tr>
<tr>
<td>Create a clock enable port for each clock port.</td>
<td>You can add a clock enable port to control when data is clocked in. This signal prevents data from being passed through.</td>
</tr>
<tr>
<td>Create an output enable port, register output enable, and set delay switch-on.</td>
<td>This option creates an output enable port. You can also register the port and extend the high-impedance state for a half clock cycle.</td>
</tr>
<tr>
<td>Add unregistered output port.</td>
<td>Optional data port <code>combout</code> is included. The <code>combout</code> port sends data to the core, bypassing the DDR I/O input registers. You can also disable <code>dataout_h</code> and <code>dataout_l</code> ports. For bidirectional operation, you must enable <code>dataout_h</code> and <code>dataout_l</code> ports, <code>combout</code> port, or both.</td>
</tr>
<tr>
<td>Implement input registers in LEs.</td>
<td>Implements input path in logic elements.</td>
</tr>
<tr>
<td>Use <code>dataout_h</code>, <code>dataout_l</code> ports.</td>
<td>Enables <code>dataout_h</code> and <code>dataout_l</code> ports.</td>
</tr>
<tr>
<td>Invert ‘padio’ port (when driving output)</td>
<td>The ‘padio’ port is inverted whenever driven as an output. This option is only applicable when using the Cyclone II device.</td>
</tr>
</tbody>
</table>
altddio_in, altddio_out & altddio_bidir Simulation Libraries

In the wizard, page 4 (for the altddio_in and altddio_out megafunctions) or page 5 (for the altddio_bidir megafunction) shows a list of the libraries needed to properly simulate the design files (Figure 2–6). This applies to all forms of the altddio megafuction.

Figure 2–6. altddio_in & altddio_out Wizard, Page 4, or altddio_bidir Wizard, Page 5
On the final page of the wizard, specify the files you wish to have generated for your custom megafunction. The gray check marks indicate files that are always generated; the other files are optional and are generated only if selected (indicated by a red check mark) (Figure 2–7).

Click Finish to create an instance of the megafunction.

**Figure 2–7. MegaWizard Plug-In Manager, Final Page**

Inferring Megafunctions from HDL Code

Synthesis tools, including the Quartus II software integrated synthesis, recognize certain types of HDL code and automatically infer the appropriate megafunction when a megafunction will provide optimal results. In other words, the Quartus II software uses the Altera megafunction code when compiling your design, even though you may not have specifically instantiated the megafunction. The Quartus II software infers megafunctions because they are optimized for Altera devices, so the area and/or performance may be better than generic HDL code. Additionally, you must use megafunctions to access certain Altera architecture-specific features, such as memory, DSP blocks, and shift registers. These features generally provide improved performance compared to basic logic elements.

Refer to the Recommended HDL Coding Styles chapter of the Quartus II Handbook for specific information about your particular megafunction.
Instantiating Megafunctions in HDL Code

When you use the MegaWizard Plug-In Manager to set up and parameterize a megafunction, it creates either a VHDL or Verilog HDL wrapper file that instantiates the megafunction (a black-box methodology). For some megafunctions, you can generate a fully synthesizable netlist for improved results with EDA synthesis tools such as Synplify and Precision RTL Synthesis (a clear-box methodology). Both clear-and black-box methodologies are described in the third-party synthesis support chapters in volume 1 of the Quartus II Handbook.

EDA Simulation

Depending on the third-party simulation tool you are using, refer to the Simulation section of volume 3 of the Quartus II Handbook. These chapters show you how to perform functional and gate-level timing simulations that include the megafunctions, with details on the files that are needed and the directories where those files are located.

SignalTap II Embedded Logic Analyzer

The SignalTap® II embedded logic analyzer provides you with a method of debugging all of the Altera megafunctions within your design. With the SignalTap II embedded logic analyzer, you can capture and analyze data samples for the top-level ports of the Altera megafunctions in your design while your system is running at full speed.

To monitor signals from your Altera megafunctions, you must first configure the SignalTap II embedded logic analyzer in the Quartus II software, and then include the analyzer as part of your project. The Quartus II software will then seamlessly embed the analyzer along with your design in the selected device.

For more information about using the SignalTap II embedded logic analyzer, refer to the Design Debugging Using the SignalTap II Embedded Logic Analyzer chapter in volume 3 of the Quartus II Handbook.
This section presents a design example that uses the DDR megafunction to generate a divider. This example uses the MegaWizard Plug-In Manager in the Quartus II software. As you go through the wizard, each page is described in detail. When you are finished with this example, you can incorporate it into the overall project.

Design Files

The design files are available in the Literature section on the User Guide page of the Altera web site (www.altera.com). Select the specific examples for altddio Megafunction User Guide link from the page to download the design files.

Example

In this example, you perform the following activities:

- Create a divider using the altddio_in, altddio_out, and lpm_divide megafunction and the MegaWizard Plug-in Manager
- Implement the design and assign the Stratix EP1S10F780C6 device to the project
- Compile and simulate the design

Generate a Divider Using altddio_in & altddio_out

The new megafunction created in this example is added to the top-level file in your Quartus II project.

Create the altddio_in Module

Follow these steps to create the altddio_in module:

1. In the Quartus II software, open the ex1.qar project.
2. On the Tools menu, select MegaWizard Plug-In Manager.
3. In the Installed Plug-Ins folder, select ALTDDIO_IN.
4. In the MegaWizard Plug-In manager dialog box, select Create a new custom megafuction variation, and click Next. The MegaWizard Plug-In Manager page displays (Figure 2–1).
5. To answer Which device family will you be using?, select Stratix.
6. To answer Which type of output file do you want to create?, select VHDL.
7. Specify the output file `ddin`.

8. Click Next. Figure 2–8 shows the wizard after you have made these selections.

![Figure 2–8. MegaWizard Plug-In Manager, Page 2a](image)

9. Click Next. Page 3 appears (Figure 2–9).
10. To answer **How wide should the buses be?** list, select 8.

11. To answer **Which asynchronous reset port would you like?**, click **None**.

12. To answer **How should the registers power up?**, click **Low**.

13. Turn off **Create a clock enable port** and **Invert input clock**.

14. Click **Next**. Page 4 (Figure 2–10) appears.
15. Click **Next**. Page 5 (**Figure 2–11**) shows the wizard summary.
16. Turn on VHDL Component declaration file and Instantiation template file.

17. Turn off Quartus symbol file and AHDL Include file.

18. Click Finish.

The altddio_in module is now built.
Create the altddio_out Module

Follow these steps to create the altddio_out module:

1. On the Tools menu, select MegaWizard Plug-In Manager.
2. In the Installed Plug-Ins folder, select ALTDDIO_OUT.
3. In the MegaWizard Plug-In manager dialog box, select Create a new custom megafunction variation, and click Next. The MegaWizard Plug-In Manager page displays (Figure 2–12).

4. In the Installed Plug-Ins folder, select ALTDDIO_OUT.
5. To answer Which device family will you be using?, select Stratix.
6. To answer Which type of output file do you want to create?, select VHDL.
7. Specify the output file ddout.
8. Click Next. Page 3 (Figure 2–13) appears.

Figure 2–13. ALTDDIO_OUT Wizard, Page 3

9. To answer How wide should the buses be?, select 8.

10. To answer Which asynchronous reset port would you like?, select None.

11. To answer How should the registers power up?, select Low.

12. Turn off the Create a clock enable port for each clock port and Create an output enable port options.

13. Click Next. Page 4 (Figure 2–14) appears.
14. Click Next. The Summary page appears. (Figure 2–15)
15. Turn on the **VHDL Component declaration file** and **Instantiation template file** options.

16. Turn off the **Quartus symbol file** and **AHDL Include file** options.

17. Click **Finish**.

The `altddio_out` module is now built.
Create the lpm_divide Module

Follow these steps to create the lpm_divide module:

1. On the Tools menu, select MegaWizard Plug-In Manager.

2. In the Installed Plug-Ins folder, select LPM_DIVIDE.

3. In the MegaWizard Plug-In manager dialog box, select Create a new custom megafunction variation, and click Next. The MegaWizard Plug-In Manager page displays (Figure 2–16).

4. In the Arithmetic folder, select LPM_DIVIDE.

5. To answer Which device family will you be using?, select Stratix.

6. To answer Which type of output file do you want to create?, select VHDL.

7. Specify the output file lp_div.
8. Click Next. Page 3 (Figure 2–17) appears.

Figure 2–17. lpm_divide Wizard, Page 3

9. To answer the questions How wide should the ‘numerator’ input bus be? and How wide should the ‘denominator’ input bus be?, select 8.

10. Under both the Numerator Representation and Denominator Representation options, select Unsigned.

11. Click Next. Page 4 (Figure 2–18) appears.
12. To answer **Do you want to pipeline the function?**, select **Yes, I want an output latency of** and type 1 in the **Clock cycles** box.

13. Turn off both the **Create an Asynchronous Clear input** and **Create a Clock Enable input** options.

14. To answer **Which do you wish to optimize?**, select **Default Optimization**.

15. To answer the question **Always return a positive remainder?**, select **Yes**.

16. Click **Next**. Page 5 (Figure 2–20) appears.
17. Click **Next**. The Wizard Summary appears.
18. Turn on the **VHDL Component declaration file** and **Instantiation template file** options.

19. Turn off the **Quartus Symbol file** and **AHDL Include file** options.

20. Click **Finish**.

The `lpm_divide` module is now built.
Create a Divider

Use the following steps to combine \texttt{altddio\_in}, \texttt{altddio\_out}, and \texttt{lpm\_divide} to create a divider.

1. In the Quartus II software, open the file \texttt{ex1.vhd}.

2. On the Project menu, click Add/Remove File in Project. The File Settings dialog box displays (Figure 2–21).

3. In the File Settings dialog box, click (...) after File name and browse for \texttt{ex1.vhd} in the project folder.

4. Select \texttt{ex1.vhd} and click Add.

5. Click OK.
The top-level file is added to the project. You have now created the complete design file.

The overall operation of the design is described in Equation 1.

The DDRIN8_OUT_H [7..0] signals are the numerator and the DDRIN8_OUT_L [7..0] signals are the denominator; in other words, the equation can be indicated thusly:

\[ DDRIN8\_OUT\_H[7..0] = Quotient[7..0] + Remain[7..0] \]
\[ DDRIN8\_OUT\_L[7..0] \]

The DDRIN8_OUT_H [7..0] and DDRIN8_OUT_L [7..0] sets of signals are passed into the divide8 function where the quotient and remainder are calculated. The divider calculates the quotient and remainder through a one-stage pipeline. The quotient and remainder are then fed via signals quotient [7..0] and remain [7..0] into the DDROUT8 module. The DDROUT8 module then drives the data out through pins DDROUT8_OUT [7..0] at double the data rate.

**Implement the Divider Design**

This section describes how to assign the Stratix EP1S10F780C6 device to the project and how to compile the project.

1. With the EX1.vhd file open, on the Assignments menu, click Settings. The Settings window displays.

2. Under the Category list, select Device. The Device Selection dialog box displays (Figure 2–22).
3. From the Family list, select Stratix.

4. Under Show in ‘Available devices’ list, select FBGA as the Package, Pin count of 780, Speed grade of 6, and turn on Show advanced devices.

5. Click OK.

6. On the Processing menu, click Start Compilation.

7. When the Full compilation was successful box displays, click OK.
Functional Results—Simulate the Divider Design in Quartus II

This section describes how to verify the design example you just created by simulating the design using the Quartus II Simulator.

To set up the Quartus II Simulator, perform the following steps:

1. With the EX1.vhd file open, on the Processing menu, click Generate Functional Simulation Netlist.
2. When the Functional Simulation Netlist Generation was successful box displays, click OK.
3. On the Assignments menu, click Settings. The Settings dialog box displays.
4. In the Category list, select Simulator Settings.
5. In the Simulation mode list, select Functional.
6. In the Simulation input box, type EX1_ip.vwf, or click Browse (...) to select the file in the project folder.
7. Turn on the Run simulation until all vector stimuli are used, Automatically add pins to simulation output waveforms, and Simulation coverage reporting options.
8. Turn off Overwrite simulation input file with simulation results.
9. Click OK.
10. On the Processing menu, click Start Simulation.
11. When the Simulation was successful box displays, click OK.
12. The Simulation Report window displays. Verify the simulation waveform results (Figure 2–23).
For Figure 2–23, the numerator (100) and denominator (5) are captured at 100 Mbps through pin DDRIN8_IN.

■ On the rising edge of clk at 15 ns, the numerator (100) drives onto the signal DDRIN8_OUT_H and the denominator (5) drives onto the signal DDRIN8_OUT_L.

■ At 35 ns the quotient (20) and the remainder (0) are calculated and driven onto signals REG_DDROUT8_IN_H and REG_DDROUT8_IN_L.

■ The high level of clk, starting at 55 ns, selects the quotient (20) to drive the DDROUT8_OUT pins, and the low level of clk selects the remainder (0) to drive the same pins.

The waveform shown in Figure 2–23 contains calculations for two more sets of numbers. The latency (15 to 55 ns) exists because of a one-stage pipeline in the divider.

Functional Results—Simulate the Divider Design in ModelSim-Altera

Simulate the design in ModelSim to compare the results of both simulators.

This User Guide assumes that you are familiar with using ModelSim-Altera before trying out the design example. If you are unfamiliar with ModelSim-Altera, refer to www.altera.com/support/software/products/modelsim/mod-modelsim.html, a support page for ModelSim-Altera. There are links to topics such as installation, usage, and troubleshooting.
Set up the ModelSim- Altera simulator by performing the following steps.

1. Unzip the **altddio_ex1_msim.zip** file to any working directory on your PC.

2. Browse to the folder in which you unzipped the files and open the **altddio_ex1.do** file in a text editor.

3. In line 1 of the **altddio_ex1.do** file, replace `<insert_directory_path_here>` with the directory path of the appropriate library files. For example, C:/Modeltech_ae/altera/verilog/stratix

4. On the File menu, click **Save**.

5. Start **ModelSim- Altera**.

6. On the File menu, click **Change Directory**.

7. Select the folder in which you unzipped the files. Click **OK**.

8. On the Tools menu, click **Execute Macro**.

9. Select the **altddio_ex1.do** file and click **Open**. This is a script file for ModelSim that automates all the necessary settings for the simulation.

10. Verify the results by looking at the **Waveform Viewer** window.

You may need to rearrange signals, remove redundant signals, and change the radix to suit the results in the Quartus II Simulator. **Figure 2–24** shows the expected simulation results in ModelSim.
This section presents a design example that uses the DDR megafunction to generate a divider. This example uses the MegaWizard Plug-In Manager in the Quartus II software. As you go through the wizard, each page is described in detail. When you are finished with this example, you can incorporate it into your overall project.

**Design Files**

The design files are available in the Quartus II Projects section on the Design Examples page of the Altera web site:

http://www.altera.com/support/examples/quartus/quartus.html

Select the Examples for altddio Megafuncton User Guide link from the examples page to download the design files.

In this example, you perform the following tasks:

- Create a divider using the altddio bidir and lpm divide megafunctions and the MegaWizard Plug-in Manager
- Implement the design and assign the Stratix EP1S10F780C6 device to the project
- Compile and simulate the design
Design Example 2: 8-Bit DDR Divider Using altddio_bidir

Generate a Divider Using altddio_bidir

The new megafunction created in this example is added to the top-level file in your Quartus II project.

Create the altddio_bidir Module

Follow these steps to create the lpm_divide module:

1. With the ex2.qar project open, in the Tools menu, select MegaWizard Plug-In Manager.

2. In the Installed Plug-Ins folder, select LPM_DIVIDE.

3. In the MegaWizard Plug-In manager dialog box, select Create a new custom megafunction variation, and click Next. The MegaWizard Plug-In Manager page displays (Figure 2–25).

4. In the Installed Plug-Ins folder, select ALTDDIO_BIDIR.
5. To answer **Which device family will you be using?**, select **Stratix**.

6. Under **Which type of output file do you want to create?**, select **VHDL**.

7. Specify the output file **alt_bid**.

8. Click **Next**. Page 3 (Figure 2–26) appears.

![Figure 2–26. altddio_bidir Wizard, Page 3](image)

9. To answer **How wide should the buses be?**, select **8**.

10. To answer **Which asynchronous reset port would you like?**, select **None**.

11. To answer **How should the registers power up?**, select **Low**.

12. Click **Next**. Page 4 (Figure 2–27) appears.
13. Turn off the Create a clock enable port for each clock port and turn on the Create an output enable port options.

14. Under Output ports, turn off Add unregistered output (‘combout’), turn on Use dataout_h, dataout_l ports and turn off Implement input registers in LE’s box.

15. Click Next. Page 5 (Figure 2–28) appears.
16. Click Next. The MegaWizard Summary page appears (Figure 2–29).
17. Turn on the **VHDL Component declaration file** and **Instantiation template file** options.

18. Turn off the **Quartus symbol file** and **AHDL Include file** options.

19. Click **Finish**.

The `altddio_bidir` module is now built.

To generate a divider, follow the steps shown in “Create the lpm_divide Module” on page 2–20.
Create a Divider

Use the following steps to combine altddio_bidir and lpm_divide to create a divider.

Follow these steps to create a top-level VHDL file:

1. With the ex2.qar project open, on the File menu, open the file ex2.vhd.

2. On the Project menu, click Add/Remove File in Project. The File Settings window displays.

3. In the File Settings window, click (...) after File name and browse for ex2.vhd in the project folder (Figure 2–21).
4. Select `ex2.vhd` and click Add to add the top-level file to the project.

5. Click OK.

The top-level file is added to the project. You have now created the complete design file.

This design implements the same divider example as described in Example 1 on page 2–26, but the functionality of `altddio_in` and `altddio_out` are implemented in a single megafunction, `altddio_bidir`. The bidirectional pins `DDR_BIDIR8 [7..0]` receive data at double the clock rate. The `DDRBIDIR8_OUT_H [7..0]` signals are the numerator and the `DDRBIDIR8_OUT_L [7..0]` signals are the denominator. These two sets of signals are passed into `lpm_divide` where the quotient and remainder are calculated. The divider calculates the quotient and remainder with a one-stage pipeline. The quotient and remainder are then fed via signals `quotient [7..0]` and `remain [7..0]` back into the `altddio_bidir` megafunction. The `altddio_bidir` megafunction then drives the data out through pins `DDR_BIDIR8 [7..0]` at double the data rate.

Implement the Divider Design

This section describes how to assign the Stratix EP1S10F780C6 device to the project and compile the project.

1. With the `ex2.qar` project open, on the Assignment menu, click Settings. The Settings dialog box displays.

2. In the Category list, select Device.

3. To answer Which device family will you be using?, select Stratix.

4. Under Target device, select Specific device selected in ‘Available devices’ list.

5. In the Available devices list, select EP1S10F780C6.

6. Under Show in ‘Available devices’ list, select FBGA as the Package, Pin count of 780, Speed grade of 6, and turn on Show Advanced Devices.

7. Click OK.

8. On the Processing menu, click Start Compilation.

9. When the Full compilation was successful box displays, click OK.
**Functional Results—Simulate the Divider Design in Quartus II**

This section describes how to verify the design example you just created by simulating the design using the Quartus II Simulator. To set up the Quartus II Simulator, perform the following steps:

1. On the Processing menu, click **Generate Functional Simulation Netlist**.

2. When the **Functional Simulation Netlist Generation was successful** box displays, click **OK**.

3. On the Assignments menu, click **Settings**. The **Settings** dialog box displays.

4. In the **Category** list, select **Simulator Settings**.

5. In the **Simulation mode** list, select **Functional**.

6. Type `ex2_ip.vwf` in the **Simulation input** box, or click **Browse (...)** to select the file in the project folder.

7. Turn on the **Run simulation until all vector stimuli are used**, **Automatically add pins to simulation output waveforms**, and **Simulation coverage reporting** options.

8. Turn off **Overwrite simulation input file with simulation results**.

9. Click **OK**.

10. On the Processing menu, click **Start Simulation**.

11. When the **Simulator was successful** box displays, click **OK**.

12. The **Simulation Report** window displays. Verify the simulation waveform results (**Figure 2–31 on page 2–40**).
In Figure 2–31, the numerator (100) and denominator (5) are captured at 100 Mbps through pin DDRBIDIR8.

- On the rising edge of clk at 15 ns, the numerator (100) drives onto the signal DDRBIDIR8_OUT_H and the denominator (5) drives onto the signal DDRBIDIR8_OUT_L.
- At 35 ns the quotient (20) and the remainder (0) are calculated and driven to signals REG_DDRBIDIR8_IN_H and REG_DDRBIDIR8_IN_L.
- The high level of clk, starting at 60 ns, selects the quotient (20) to drive the DDROUT8_OUT pins, and the low level of clk selects the remainder (0) to drive the same pins.

The waveform shown in Figure 2–31 contains calculations for two more sets of numbers. The latency (15 ns to 60 ns) exists because of a one-stage pipeline in the divider.

To allow the data to be driven out of the bidirectional pin in the simulation, make sure the input signal part of the bidirectional pin is set to a weak unknown, allowing the simulation to overwrite the value at the specific time interval. The Quartus II software creates an additional signal to emulate the output part of the bidirectional pin. This signal is named `<pin name>~result`. 
**Functional Results—Simulate the Divider Design in ModelSim-Altera**

Simulate the design in ModelSim to compare the results of both simulators.

This User Guide assumes that you are familiar with using ModelSim-Altera before trying out the design example. If you are unfamiliar with ModelSim-Altera, refer to [www.altera.com/support/software/products/modelsim/mod-modelsim.html](http://www.altera.com/support/software/products/modelsim/mod-modelsim.html), a support page for ModelSim-Altera. There are links to topics such as installation, usage, and troubleshooting.

Set up the ModelSim-Altera simulator by performing the following steps.

1. Unzip the `altddio_ex2_msim.zip` file to any working directory on your PC.

2. Browse to the folder in which you unzipped the files and open the `altddio_ex2.do` file in a text editor.

3. In line 1 of the `altddio_ex2.do` file, replace `<insert_directory_path_here>` with the directory path of the appropriate library files. For example, `C:/Modeltech_ae/altera/verilog/stratix`.

4. On the File menu, click **Save**.

5. Start **ModelSim-Altera**.

6. On the File menu, click **Change Directory**.

7. Select the folder in which you unzipped the files. Click **OK**.

8. On the Tools menu, click **Execute Macro**.

9. Select the `altddio_ex2.do` file and click **Open**. This is a script file for ModelSim that automates all the necessary settings for the simulation.

10. Verify the results by looking at the **Waveform Viewer** window.

You may need to rearrange signals, remove redundant signals, and change the radix to suit the results in the Quartus II Simulator. **Figure 2–32** shows the expected simulation results in ModelSim.
Conclusion

The Quartus II software provides parameterizable megafunctions ranging from simple arithmetic units, such as adders and counters, to advanced phase-locked loop (PLL) blocks, multipliers, and memory structures. These megafunctions are performance-optimized for Altera devices, and therefore provide more efficient logic synthesis and device implementation, because they automate the coding process and save valuable design time. You should use these functions during design implementation so you can consistently meet your design goals.
The Quartus® II software provides three megafunctions that support DDR functionality: altddio_in, altddio_out, and altddio_bidir. This chapter describes the ports and parameters for the DDR megafunctions.

The parameter details are only relevant for users who bypass the MegaWizard® Plug-In Manager interface and use the megafunction as a directly parameterized instantiation in their design. The details of these parameters are hidden from MegaWizard Plug-In Manager interface users.

Refer to the latest version of the Quartus II Help for the most current information about the ports and parameters for these megafunctions.

**Ports & Parameters for the altddio_in Megafuction**

*Figure 3–1* shows the ports for the altddio_in megafuction.

**Figure 3–1. altddio_in Ports**

```
+-----+-----+
|     |     |
| datain[ ] | dataout_h[ ] |
+-----+-----+
|     |     |
| inclock | dataout_l[ ] |
+-----+-----+
|     |     |
| inclocken |     |
+-----+-----+
|     |     |
| aclr |     |
+-----+-----+
|     |     |
| aset |     |
+-----+-----+
```
Table 3–1 shows the input ports, Table 3–2 shows the output ports, and Table 3–3 shows the \texttt{altddio\_in} megafuction parameters. The options listed in these tables are valid for Stratix\textsuperscript{®} series, Cyclone\textsuperscript{™} series, APEX\textsuperscript{™} II, and Mercury\textsuperscript{™} devices except where noted.

<table>
<thead>
<tr>
<th>Name</th>
<th>Required</th>
<th>Description</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>datain[]</td>
<td>Yes</td>
<td>DDR input data port.</td>
<td>Input port WIDTH wide. The datain port should be directly fed from an input pin in the top-level design.</td>
</tr>
<tr>
<td>inclock</td>
<td>Yes</td>
<td>Clock signal to sample the DDR input.</td>
<td>The datain port is sampled on each clock edge of the inclock signal.</td>
</tr>
<tr>
<td>inclocken</td>
<td>No</td>
<td>Clock enable for the data clock.</td>
<td>—</td>
</tr>
<tr>
<td>aclr</td>
<td>No</td>
<td>Asynchronous clear input.</td>
<td>The aclr and aset ports cannot be connected at the same time.</td>
</tr>
<tr>
<td>aset</td>
<td>No</td>
<td>Asynchronous set input.</td>
<td>The aclr and aset ports cannot be connected at the same time.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataout_h[]</td>
<td>Yes</td>
<td>Data sampled from datain[] port at the rising edge of the inclock signal.</td>
</tr>
<tr>
<td>dataout_l[]</td>
<td>Yes</td>
<td>Data sampled from datain[] port at the falling edge of the inclock signal.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIDTH</td>
<td>Integer</td>
<td>Yes</td>
<td>Width of datain, dataout_h, and dataout_l ports.</td>
</tr>
<tr>
<td>POWER_UP_HIGH</td>
<td>String</td>
<td>No</td>
<td>When both aset and aclr ports are unused, POWER_UP_HIGH parameter is available to specify power-up state of output ports. Values are ON and OFF. The default setting is OFF.</td>
</tr>
<tr>
<td>INTENDED_DEVICE_FAMILY</td>
<td>String</td>
<td>No</td>
<td>This parameter is used for modeling and behavioral simulation. Create altddio_in megafuction with the wizard to calculate value for this parameter. If not created with the wizard, the default family is Mercury.</td>
</tr>
</tbody>
</table>
Ports & Parameters for the altddio_out Megafunction

Ports for the altddio_out megafunction are shown in Figure 3–2.

**Figure 3–2. altddio_out Ports**

![Diagram of altddio_out Ports]

Table 3–4 shows the input ports, Table 3–5 shows the output ports, and Table 3–6 shows the altddio_out megafunction parameters. The options listed in these tables are valid for Stratix series, Cyclone, APEX II, and Mercury devices.

<table>
<thead>
<tr>
<th>Name</th>
<th>Required</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>datain_h[]</td>
<td>Yes</td>
<td>Input data for rising edge of outclock port.</td>
<td>Input port WIDTH wide.</td>
</tr>
<tr>
<td>datain_l[]</td>
<td>Yes</td>
<td>Input data for falling edge of outclock port.</td>
<td>Input port WIDTH wide.</td>
</tr>
<tr>
<td>outclock</td>
<td>Yes</td>
<td>Clock signal to register data output.</td>
<td>dataout port outputs DDR data on each level of outclock signal.</td>
</tr>
<tr>
<td>outclocken</td>
<td>No</td>
<td>Clock enable for outclock port.</td>
<td>—</td>
</tr>
<tr>
<td>aclr</td>
<td>No</td>
<td>Asynchronous clear input.</td>
<td>The aclr and aset ports cannot be connected at the same time.</td>
</tr>
<tr>
<td>aset</td>
<td>No</td>
<td>Asynchronous set input.</td>
<td>The aclr and aset ports cannot be connected at the same time.</td>
</tr>
<tr>
<td>oe</td>
<td>No</td>
<td>Output enable for the dataout port.</td>
<td>Active-high signal. You can add an inverter if you need an active-low oe.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Required</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataout[]</td>
<td>Yes</td>
<td>DDR output data port.</td>
<td>Output port WIDTH wide. dataout port should directly feed an output pin in top-level design.</td>
</tr>
</tbody>
</table>
### Table 3–6. altddio_out Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIDTH</td>
<td>Integer</td>
<td>Yes</td>
<td>Sets the width of the <code>datain_h</code>, <code>datain_l</code>, and <code>dataout</code> ports.</td>
</tr>
<tr>
<td>POWER_UP_HIGH</td>
<td>String</td>
<td>No</td>
<td>When both the <code>aset</code> and <code>aclr</code> ports are unused, the POWER_UP_HIGH parameter is available to specify the power-up state of the output ports. Values are ON and OFF. The default setting is OFF.</td>
</tr>
<tr>
<td>INTENDED_DEVICE_FAMILY</td>
<td>String</td>
<td>No</td>
<td>This parameter is used for modeling and behavioral simulation. Create the altddio_out megafuction wizard to calculate the value for this parameter. If not created with the wizard, the default family is Mercury.</td>
</tr>
<tr>
<td>OE_REG</td>
<td>String</td>
<td>No</td>
<td>Specifies whether the oe port is registered. Values are REGISTERED, UNREGISTERED, and UNUSED. The default setting is UNUSED. (1)</td>
</tr>
<tr>
<td>EXTEND_OE_DISABLE</td>
<td>String</td>
<td>No</td>
<td>This specifies whether the second oe register should be used. When the second oe register is used, the output pin is held at high impedance an extra half clock cycle after the oe port goes high. Values are ON, OFF, and UNUSED. The default setting is UNUSED. (1)</td>
</tr>
<tr>
<td>LPM_HINT</td>
<td>String</td>
<td>No</td>
<td>Allows you to assign Altera®-specific parameters in VHDL Design Files (.vhd). The default is UNUSED.</td>
</tr>
<tr>
<td>LPM_TYPE</td>
<td>String</td>
<td>No</td>
<td>Identifies the library of parameterized modules (LPM) entity name in VHDL Design Files.</td>
</tr>
</tbody>
</table>

**Note for Table 3–6:**

(1) This parameter is not available for the Mercury device family.
Ports & Parameters for the altddio_bidir Megafunction

Figure 3–3 shows the ports for the altddio_bidir megafunction.

Table 3–7 shows the input ports, Table 3–8 shows the output ports, Table 3–9 shows the bidirectional port, and Table 3–10 shows the altddio_bidir megafunction parameters. The options listed in these tables are valid when targeting Stratix series, Cyclone, APEX II, and Mercury devices.

<table>
<thead>
<tr>
<th>Name</th>
<th>Required</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>datain_h[]</td>
<td>Yes</td>
<td>Input data to be output to the padio port at the rising edge of the outclock port.</td>
<td>Input port ([(WIDTH) - (1)\ldots0]) wide.</td>
</tr>
<tr>
<td>datain_l[]</td>
<td>Yes</td>
<td>Input data to be output to the padio port at the falling edge of the outclock port.</td>
<td>Input port ([(WIDTH) - (1)\ldots0]) wide.</td>
</tr>
<tr>
<td>inclock</td>
<td>Yes</td>
<td>Clock signal to sample the DDR input.</td>
<td>The padio port is sampled on each clock edge of the inclock signal.</td>
</tr>
<tr>
<td>inclocken</td>
<td>No</td>
<td>Clock enable for the inclock port.</td>
<td>—</td>
</tr>
<tr>
<td>outclock</td>
<td>Yes</td>
<td>Clock signal to register the data output.</td>
<td>The padio port outputs the DDR data on each edge of the outclock signal.</td>
</tr>
<tr>
<td>outclocken</td>
<td>No</td>
<td>Clock enable for the outclock port.</td>
<td>—</td>
</tr>
<tr>
<td>aclr</td>
<td>No</td>
<td>Asynchronous clear input.</td>
<td>The aclr and aset ports cannot be connected at the same time.</td>
</tr>
</tbody>
</table>
### Ports & Parameters

#### Table 3–7. altddio_bidir Input Ports (Part 2 of 2)

<table>
<thead>
<tr>
<th>Name</th>
<th>Required</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>aset</td>
<td>No</td>
<td>Asynchronous set input.</td>
<td>The aclr and aset ports cannot be connected at the same time.</td>
</tr>
<tr>
<td>oe</td>
<td>No</td>
<td>Output enable for the bidirectional padio port.</td>
<td>If the oe port is not connected, then the padio port is an output port.</td>
</tr>
</tbody>
</table>

### Table 3–8. altddio_bidir Output Ports

<table>
<thead>
<tr>
<th>Name</th>
<th>Required</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataout_h[]</td>
<td>Yes</td>
<td>Data sampled from the padio port at the rising edge of the inclock signal.</td>
<td>Output port ([(\text{WIDTH}) - 1..0]) wide.</td>
</tr>
<tr>
<td>dataout_l[]</td>
<td>Yes</td>
<td>Data sampled from the padio port at the falling edge of the inclock signal.</td>
<td>Output port ([(\text{WIDTH}) - 1..0]) wide.</td>
</tr>
<tr>
<td>combout[]</td>
<td>No</td>
<td>Combinational output directly fed by the padio port.</td>
<td>(1)</td>
</tr>
<tr>
<td>dqsundelayedout[]</td>
<td>No</td>
<td>Undelayed output from the DQS pins.</td>
<td>Output port ([\text{WIDTH-1..0}]) wide. (2)</td>
</tr>
</tbody>
</table>

**Notes for Table 3–8:**
1. This port is available for Stratix series, HardCopy Stratix, Cyclone series, and APEX II devices only.
2. This port is available for Stratix and HardCopy Stratix devices only.

### Table 3–9. altddio_bidir Bidirectional Port

<table>
<thead>
<tr>
<th>Name</th>
<th>Required</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>padio[]</td>
<td>Yes</td>
<td>Bidirectional DDR port that should directly feed a bidirectional pin in the</td>
<td>The DDR data is transmitted and received on this bidirectional port.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>top-level design.</td>
<td>Bidirectional port ([(\text{WIDTH}) - 1..0]) wide.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Type</td>
<td>Required</td>
<td>Comments</td>
</tr>
<tr>
<td>----------------------------</td>
<td>----------</td>
<td>----------</td>
<td>-------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>WIDTH</td>
<td>Integer</td>
<td>Yes</td>
<td>Width of the datain_h, datain_l, and dataout ports.</td>
</tr>
<tr>
<td>POWER_UP_HIGH</td>
<td>String</td>
<td>No</td>
<td>When both the aset and aclr ports are unused, the POWER_UP_HIGH parameter is available to specify the power-up state of the output ports. Values are ON and OFF. The default setting is OFF.</td>
</tr>
<tr>
<td>INTENDED_DEVICE_FAMILY</td>
<td>String</td>
<td>No</td>
<td>This parameter is used for modeling and behavioral simulation purposes. Create the altddio Bidir megafuction with the MegaWizard Plug-in Manager to calculate the value for this parameter. If not created with the wizard, the default family is Mercury.</td>
</tr>
<tr>
<td>LPM_TYPE</td>
<td>String</td>
<td>No</td>
<td>Identifies the library of parameterized modules (LPM) entity name in VHDL Design Files.</td>
</tr>
<tr>
<td>OE_REG</td>
<td>String</td>
<td>No</td>
<td>Specifies whether the oe port is registered. Values are REGISTERED, UNREGISTERED, and UNUSED. The default setting is UNUSED. (1)</td>
</tr>
<tr>
<td>IMPLEMENT_INPUT_IN_LCELL</td>
<td>String</td>
<td>No</td>
<td>Specifies whether the input channels should be implemented using logic cells. Values are ON, OFF, and UNUSED. The default is UNUSED. (1)</td>
</tr>
<tr>
<td>EXTEND_OE_DISABLE</td>
<td>String</td>
<td>No</td>
<td>This specifies whether the second oe register should be used. When the second oe register is used, the output pin is held at high impedance an extra half clock cycle after the oe port goes high. Values are ON, OFF, and UNUSED. The default setting is UNUSED. (1)</td>
</tr>
</tbody>
</table>

*Note for Table 3–10:*
(1) This parameter is not available for the Mercury device family.