DIMEtalk 3.1 Reference Guide

NT108-0305 - Issue 7

Contacting Nallatech:

Support:

WWW:

Go to www.nallatech.com and click 'support'.

Email:

support@nallatech.com

Phone/Fax

Europe and Asia-Pacific:
Phone: +44 (0)1236 789500

North America
Phone: +1-877-44-NALLA

WWW:

www.nallatech.com

Document Name: DIMEtalk 3.1 Reference Guide

Document Number: NT108-0305

Issue Number: Issue 7

Date of Issue: 26/02/07

Revision History:

Date	Issue Number	Revision
06/04/2006	2	Initial release of 3.1 New to 3.1 Reference Guide: "BenBLUE-III Clock Deskew Module for Use with ZBT SRAM" on page 52, "Clock Deskew Component for BenBLUE Modules" on page 83, "DIME-C Link FIFO" on page 93, "DDR2 Memory Clocks Module" on page 102, "DDR-II SRAM Memory Nodes" on page 104, "BenDATA-V4 SDRAM Memory Node" on page 113, "BenDATA2 SDRAM Clock Module" on page 126, "BenDATA2 SDRAM Memory Node - 333MHz" on page 128, "PCI-X Clocks Driver Component" on page 84, "PCI-X Host Interface" on page 4, "Grounding Extra BenBLUE-III Control Pins" on page 53, "ZBT SRAM Testbench for Simulation" on page 55, "Host Testbench" on page 11, New section on component placement - "How to Place Components in DIMEtalk Networks" on page 140.
06/09/2006	3	Updates for DIMEtalk 3.1.1 "PCI-X Host Interface" on page 4: Hostusr_clk signal removed from component block diagram and description table. "PCI-X Clocks Driver Component" on page 84: HostIfClocks: hostif_clk200 signal added. "DDR2 Memory Clocks Module" on page 102: DDR2CoreClocks now DDR2CoreClocks0, DDR2CoreClocks1, DDR2CoreClocks2, DDR2CoreClocks3. Simplified connection of network components in the following examples: "BenDATA-V4 Memory Components" on page 140, "PCI-X Edge and PCI-X Clock Components" on page 141, "PCI-X SRAM Component" on page 141, "BenBLUE-V4 Memory Components" on page 143. New component added: "PCI-X Module Clocks Driver Component" on page 88. "DDR2 Memory Clocks Module" on page 102 contains the following, previously undocumented parameter: Use_with_DDRII_SDRAM (listed in Table 95). New section added on the use of clocks on the BenNUEY-PCI-X-V4: "DIMEtalk Clock Usage on the BenNUEY-PCI-X-V4" on page 146.

Date	Issue Number	Revision
18/10/2006	4	Updates for DIMEtalk 3.1.2 Updates to "PCI-X Host Interface" section: Figure 2 updated Table 4 updated Table 5 updated Component Definition also updated. Updates to "Clock Deskew Component" section: Table 38 updated Updates to "DDR-II SRAM Memory Nodes" section: Table 101 updated
22/11/2006	5	Updates for DIMEtalk 3.1.3 Change to section "DIMEtalk Clock Usage on the BenNUEY-PCI-X-V4": Clock C does not have to be used on the PCI-X motherboard as specified in sub-section "Non Virtex-4 Modules Populated on a BenNUEY-PCI-X-V4 Motherboard". Section updated with correct usage information. Changes to section "DDR-II SRAM Memory Nodes": Table 98 updated with new entry for BenNUEY-PCI-104-V4, Correction to Table 98 to show correct entry for 'IDELAY_CTRLS' value for 'ddr2sram_32_benbluev4_sec' entry, Table 101 updated with list of support files for the BenNUEY-PCI-104-V4 plus corrections to 'idelay_calibrate.vhd' and 'sram_data_iobs.vhd' entries. New entry 'sram_data_path.vhd' added. Table 102 updated with correct device usage information for all entries. "Generic Settings" updated with BenNUEY-PCI-104-V4 details.
27/11/06	6	New components added as follows: "H100 PCI-X Clocks Driver Module", "H100 DDR2 SDRAM Memory Node", "DDR Asynchronous Bridge", "DDR Asynchronous Bridge Clock Components" and "H100 Host Interface". New section added on usage of new components - "H100 Components".
26/02/2007	7	Updates for DIMEtalk 3.1.5 "DIMEtalk Detailed Design" section revised with new VHDL attributes information for DIMEtalk components. Note added to "DDR2 Memory Clocks Module" on use of component with H100 series hardware. New components added as follows: "DIME-C Internal LinkFIFO (dclink_fifo)" and "DIMEtalk DDR to MGT Bridge". Removed "DDR Asynchronous Bridge Clock Components" section - component no longer used. Revised Table 84 with external MGT link information for H100 System Clock component. Removed MGT Interface example from "H100 Components" section, updated example available in *HPC Software Toolkit User Guide*.

Intellectual Property

implication, estoppel or otherwise any licence or right to use any trademark, patent, design right or copyright of Nallatech Limited, or any other third party. Microsoft and Windows are either registered trade marks or trade marks of Microsoft Corporation in the United States and/or other countries.

Disclaimer

This document is for general information purposes only and is not tailored for any specific situations or circumstances. Although Nallatech Limited believes the contents to be true and accurate as at the date of writing, Nallatech Limited makes no assurances or warranty regarding the accuracy, currency or applicability of any contents in relation to specific situations and particular circumstances. As such, the content should not be relied upon and readers should not act on this information without further consultation with Nallatech Limited. Nallatech Limited accepts no responsibility for loss which may arise as a result of relying on the information in this document alone.

This page intentionally blank

Contents

This page intentionally blank

List of Figures

List of Tables

This page intentionally blank

About this Reference Guide

Using this manual

This Reference Guide provides detailed information on DIMEtalk components and the data packet formats used in the software. The manual is designed to provide reference information for users of DIMEtalk and should be read in conjunction with the *DIMEtalk User Guide*.

- For information on how to use the software see the *DIMEtalk User Guide*.
- For additional application notes please visit www.nallatech.com/applicationnotes.
- For information on using the DIMEtalk Application Program Interface (API) see the *FUSE C-C++ API Developer's Guide*.

Symbols Used

Throughout this manual there are symbols to draw attention to important information:

▼ **The red arrow symbol indicates a set of procedures to follow, such as installing software or setting up hardware.**

The blue 'i' symbol indicates useful or important information.

The red '!' symbol indicates a warning, which requires special attention.

Reference Guide Format

The Reference Guide is divided into **Sections**. The sections divide the document as follows:

- DIMEtalk Component Descriptions: The individual components including descriptions, block diagrams and, where relevant, wave forms.
- DIMEtalk Detailed Design: Information on DIMEtalk packet formats and VHDL attributes for user components.

Related Nallatech Documentation

- Nallatech BenDATA-II Reference Guide
- Nallatech BenDATA-V4 Reference Guide
- Nallatech Connecting FPGAs in DIMEtalk Application Note
- Nallatech DDR SDRAM Controller Core Datasheet
- Nallatech DIME-C User Guide
- Nallatech DIMEtalk User Guide
- Nallatech FUSE C-C++ API Developer's Guide
- Nallatech FUSE System Software User Guide
- Nallatech IEEE754 Floating Point Core User Guide
- Nallatech Tcl Plug-In for FUSE Developer's Guide
- Xilinx Xilinx Synthesis and Verification Design Guide
- Xilinx Application Note XAPP645

Abbreviations

- **API:** Application Program Interface
- **DAC:** Digital-to-Analog Converter
- **DIME:** DSP and Image Processing Modules for Enhanced FPGAs
- **ECC:** Error Correction Control
- **FIFO:** First In First Out stack memory
- **FIR:** Finite Impulse Response
- **FPGA:** Field Programmable Gate Array
- **FUSE:** Field Upgradeable System Environment
- **IDE:** Integrated Development Environment
- **I/O:** Input/Output
- **PCI:** Peripheral Component Interconnect
- **SRAM:** Static Random Access Memory
- **TCP/IP:** Transmission Control Protocol/Internet Protocol
- **UCF:** User Constraints File
- **USB:** Universal Serial Bus
- **VHDL:** VHSIC Hardware Description Language

Typographical Conventions

The following typographical conventions are used in this manual:

- Red text indicates a cross-reference to information within the document set you are currently reading. Click the red text to go to the referenced item. To return to the original page, right-click anywhere on the current page and select **Go To Previous View**.

- Blue underlined text indicates a link to a Web page. Click blue-underlined text to browse the specified Web site.
- *Italics* denotes the following items:
  - References to other documents:

    See the *FUSE System Software User Guide* for more information.
  - Emphasis in text:

    Enable Loopback should *not* be enabled until all other registers have been set up.

FUSE Naming Conventions

Please note that the DIMEtalk clocks are named differently in the FUSE System Software compared to this Reference Guide. The clock naming conventions are shown in Table 1.

Clock Names in FUSE	Clock Names in Documentation
System Clock (SYSCLK)	Clock A (CLK A)
DSP Clock (DSPCLK)	Clock B (CLK B)
Pixel Clock (PIXCLK)	Clock C (CLK C)

Table 1: FUSE Naming Conventions

Comments and Suggestions

At the back of this Reference Guide, you will find a remarks form. We welcome any comments you may have on our product or its documentation. Your remarks will be examined thoroughly and taken into account for future versions of Nallatech products.

This page intentionally blank

Section 1

DIMEtalk Component Descriptions

In this section the following DIMEtalk components are described:

- Edges
- Basic Internal FPGA Nodes
- ZBT Nodes
- Bridges
- Routers
- System
- DIME-C
- Virtex-4 DDR2 Memory
- DDR SDRAM

This section also explains how to use the following components:

- BenDATA-V4 memory components
- PCI-X edge and clock components
- BenDATA-II memory and clock components
- BenBLUE-V4 memory components
- H100 components
- DIMEtalk Clock Usage on the BenNUEY-PCI-X-V4

1.1 Edges

PCI Host Interface

Functional Description

This component handles the interface between a PCI host and a DIMEtalk network on Nallatech PCI cards (e.g. BenNUEY). The interface is assumed to be clocked by CLK2. This edge component converts data being transferred into the CLK1 domain. On a multi-FPGA motherboard such as the BenNUEY, the Host Data signals should be mapped to PCI_COMMS(0:31) whilst the Host Control signals should be mapped to PCI_COMMS(32:39). If this interface is instantiated on a module plugged into a BenONE motherboard the Host Data signals should be mapped to LBUS(0:31) whilst the Host Control signals should be mapped to ADJOUT(0:7). The PCI Host Interface signals are shown in Figure 1 and described in Table 2.

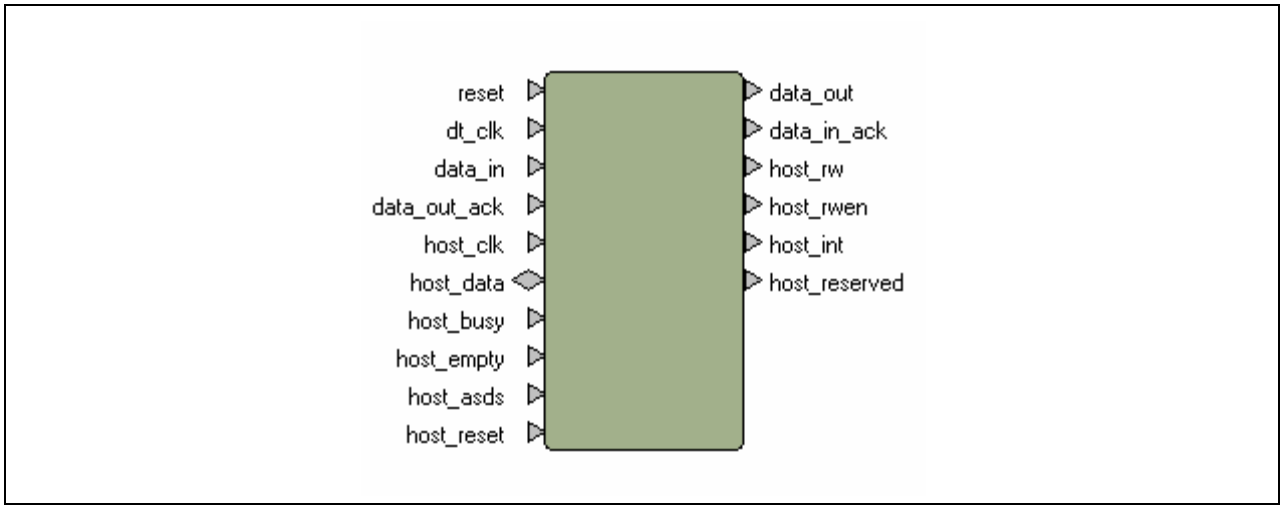

Figure 1: PCI Host Interface Component

Signals

The signal descriptions are provided in Table 2.

Group	Type	Signal Description	Width	Direction
reset	DIMEtalk reset	reset	1-bit	In
dt_clk	DIMEtalk network clock. Def. CLK1	dt_clk	1-bit	In
DIMEtalk		data_in	32-bit	In
DIMEtalk		data_out	32-bit	Out
DIMEtalk		data_in_ack	1-bit	Out
DIMEtalk		data_out_ack	1-bit	In
host_clk	Def. CLK2	host_clk	1-bit	In
Host Data		host_data	32-bit	Inout
Host Control		host_busy	1-bit	In
Host Control		host_empty	1-bit	In
Host Control		host_rw	1-bit	Out

Table 2: PCI Host Signal Descriptions

Group	Type	Signal Description	Width	Direction
Host Control		host_asds	1-bit	In
Host Control		host_rwen	1-bit	Out
Host Control		host_int	1-bit	Out
Host Control		host_reset	1-bit	In
Host Control		host_reserved	1-bit	Out

Table 2: PCI Host Signal Descriptions

Support Files

The support file names and devices used are listed in Table 3.

Name	Device Usage
DIMEtalk\library\Edges\pci_host_interface.support\Virtex2\fifo511x32.edn	Virtex-II
DIMEtalk\library\Edges\pci_host_interface.support\VirtexE\fifo511x32.edn	Virtex-E
DIMEtalk\library\Edges\pci_host_interface.support\All\pci_host_interface.ngc	All
DIMEtalk\library\Edges\pci_host_interface.support\Simulation\pci_host_interface.vhd	Simulation
DIMEtalk\library\Edges\pci_host_interface.support\Simulation_Virtex2\fifo511x32.vhd	Simulation Virtex-II
DIMEtalk\library\Edges\pci_host_interface.support\Simulation_VirtexE\fifo511x32.vhd	Simulation Virtex-E
DIMEtalk\library\Edges\pci_host_interface.support\Virtex4\fifo511x32.ngc	Virtex-4
DIMEtalk\library\Edges\pci_host_interface.support\Virtex4\fifo511x32.vhd	Virtex-4
DIMEtalk\library\Edges\pci_host_interface.support\Virtex4\fifo511x32core.edn	Virtex-4
DIMEtalk\library\Edges\pci_host_interface.support\Virtex4\fifo511x32core.vhd	Virtex-4
DIMEtalk\library\Edges\pci_host_interface.support\Virtex4\fifo511x32core_fifo_generator_v2_2_xst_1.ngc	Virtex-4
DIMEtalk\library\Edges\pci_host_interface.support\Virtex4\fifo511x32core_fifo_generator_v2_2_xst_1_blkmemdp_v6_2_xst.edn	Virtex-4
DIMEtalk\library\Edges\pci_host_interface.support\Simulation_Virtex4\fifo511x32.vhd	Simulation Virtex-4
DIMEtalk\library\Edges\pci_host_interface.support\Simulation_Virtex4\fifo511x32core.vhd	Simulation Virtex-4

Table 3: PCI Host Signal Support Files

Component Definition

Component pci_host_interface_000

Port (

```
-- reset
reset : in std_logic;
-- dt_clk
dt_clk : in std_logic;
```

```
-- DIMEtalk
data_in : in std_logic_vector(31 downto 0);
data_out : out std_logic_vector(31 downto 0);
data_in_ack : out std_logic;
data_out_ack : in std_logic;
-- host_clk
host_clk : in std_logic;
-- Host Data
host_data : inout std_logic_vector(31 downto 0);
-- Host Control
host_busy : in std_logic;
host_empty : in std_logic;
host_rw : out std_logic;
host_asds : in std_logic;
host_rwen : out std_logic;
host_int : out std_logic;
host_reset : in std_logic;
host_reserved : out std_logic
);
```

PCI-X Host Interface

This component, shown in Figure 2, handles the interface between a PCI-X host and a DIMEtalk network on a Nallatech PCI-X card (e.g. BenNUEY-PCI-X). The PCI-X FPGA provides a 200MHz clock source for clocking the PCI-X Host Interface. The component enables communication between the PCI-X and User FPGAs and converts the transferred data into the DIMEtalk clock domain (typically CLK1).

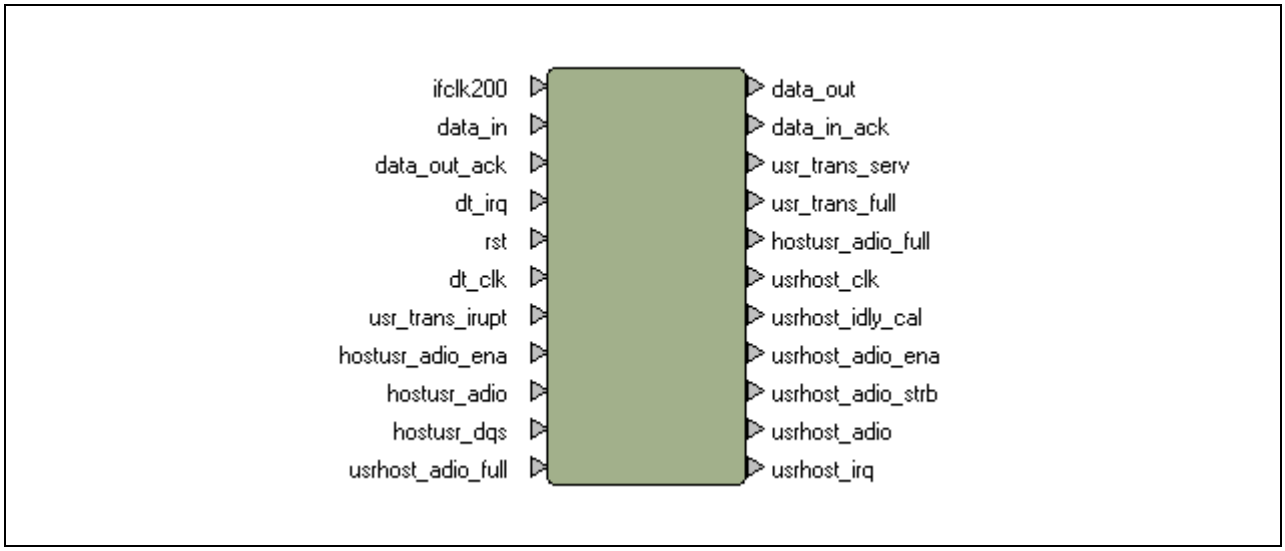

Figure 2: PCI-X Host Interface Component

Signals

The signal descriptions are provided in Table 4.

Group	Type	Signal Description	Width	Direction
Reset	DIMEtalk reset	reset	1-bit	in
DIMEtalk clock	DIMEtalk network clock. Def. CLK1	dt_clk	1-bit	in
Interrupts		dt_irq	4-bits	in
DIMEtalk		data_in	32-bits	in
DIMEtalk		data_out	32-bits	out
DIMEtalk		data_in_ack	1-bit	out
DIMEtalk		data_out_ack	1-bit	in
Host control		usr_trans_irupt	1-bit	in
Host control		usr_trans_serv	1-bit	out
Host control		hostusr_dqs	1-bit	in
Host control		hostusr_adio_ena	1-bit	in
Host control		hostusr_adio_full	1-bit	out
Host data		hostusr_adio	32-bits	in
Host control		usrhost_clk	4-bits	out
Host control		usrhost_idly_cal	1-bit	out
Host control		usrhost_adio_ena	1-bit	out
Host control		usrhost_adio_strb	1-bit	out
Host data		usrhost_adio	32-bits	out
Host control		usrhost_irq	4-bits	out
Host control		usrhost_adio_full	1-bit	in

Table 4: PCI-X Host Interface Signal Descriptions

Support Files

The support file names and devices used are listed in Table 5.

Name	Device Usage
DIMEtalk\library\Edges\pcix_host_interface.support\pcix_host_interface.vhd	Virtex-4
DIMEtalk\library\Edges\pcix_host_interface.support\fifo_64x1024.ngc	Virtex-4
DIMEtalk\library\Edges\pcix_host_interface.support\infifo64x512.ngc	Virtex-4
DIMEtalk\library\Edges\pcix_host_interface.support\synfifo32x512.ngc	Virtex-4
DIMEtalk\library\Edges\pcix_host_interface.support\outfifo32x1024.ngc	Virtex-4
DIMEtalk\library\Edges\pcix_host_interface.support\user_host_if.vhd	Virtex-4
DIMEtalk\library\Edges\pcix_host_interface.support\user_host_ctrl.vhd	Virtex-4

Table 5: PCI-X Host Signal Support Files

Name	Device Usage
DIMEtalk\library\Edges\pcix_host_interface.support\dt_ctrl_if.vhd	Virtex-4
DIMEtalk\common\resync_rise.vhd	Virtex-4
DIMEtalk\library\Edges\pcix_host_interface.support\rx_iob.vhd	Virtex-4
DIMEtalk\library\Edges\pcix_host_interface.support\idelay_cal.vhd	Virtex-4
DIMEtalk\library\Edges\pcix_host_interface.support\tx_iob.vhd	Virtex-4

Table 5: PCI-X Host Signal Support Files

Component Definition

```
component pcix_host_interface
 generic(
  SIM                : boolean := FALSE
  );
 port(
  rst                : in  std_logic;
  ifclk200           : in  std_logic;

  -- Group = DIMEtalk
  dt_clk             : in  std_logic;
  dt_irq             : in  std_logic_vector( 3 downto 0);
  data_in            : in  std_logic_vector(31 downto 0);
  data_out           : out std_logic_vector(31 downto 0);
  data_in_ack        : out std_logic;
  data_out_ack       : in  std_logic;

  -- Group = host interface
  -- host <--> user if target data interrupt & service flags
  usr_trans_irupt    : in  std_logic;
  usr_trans_serv     : out std_logic;
  usr_trans_full     : out std_logic;

  -- host --> user if (Rx)
  hostusr_dqs        : in  std_logic;
  hostusr_adio_ena   : in  std_logic;
  hostusr_adio       : in  std_logic_vector(31 downto 0);
  hostusr_adio_full  : out std_logic;
```

```
-- user --> host if (Tx)
usrhost_clk          : out std_logic_vector( 3 downto 0);
usrhost_idly_cal     : out std_logic;
usrhost_adio_ena     : out std_logic;
usrhost_adio_strb    : out std_logic;
usrhost_adio         : out std_logic_vector(31 downto 0);
usrhost_adio_full    : in  std_logic;

-- user --> host interrupts
usrhost_irq          : out std_logic_vector( 3 downto 0)
);
```

USB Host Interface

Functional Description

This component handles the interface between a USB module connected host and a DIMEtalk network on Nallatech stand-alone motherboards (e.g. BenNUEY and BenONE). The interface is assumed to be clocked by CLK2. This edge component converts data transferred into the CLK1 domain. On a multi-FPGA motherboard such as the BenNUEY, the Host Data signals should be mapped to PCI_COMMS(0:31) whilst the Host Control signals should be mapped to PCI_COMMS(32:39). If this interface is instantiated on a module plugged into a BenONE motherboard the Host Data signals should be mapped to LBUS(0:31) whilst the Host Control signals should be mapped to ADJOUT(0:7). The USB Host Interface signals are shown in Figure 3 and described in Table 6.

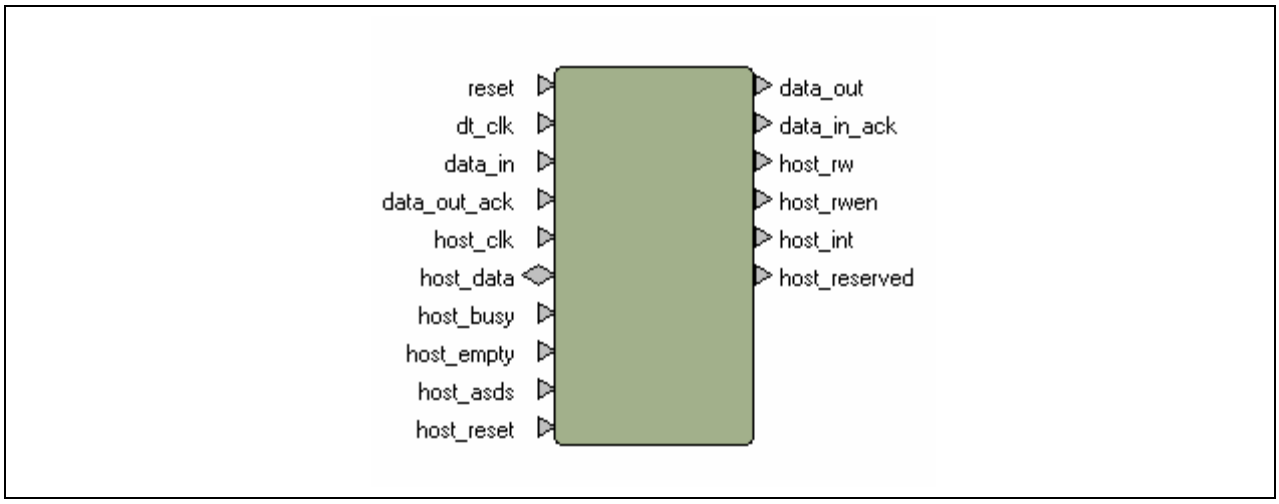

Figure 3: USB Host Interface Component

Signals

The signal descriptions are provided in Table 6.

Group	Type	Signal Description	Width	Direction
reset	DIMEtalk reset	reset	1-bit	In
dt_clk	DIMEtalk network clock. Def. CLK1	dt_clk	1-bit	In
DIMEtalk		data_in	32-bits	In
DIMEtalk		data_out	32-bits	Out
DIMEtalk		data_in_ack	1-bit	Out
DIMEtalk		data_out_ack	1-bit	In
host_clk	Def. CLK2	host_clk	1-bit	In
Host Data		host_data	32-bits	Inout
Host Control		host_busy	1-bit	In
Host Control		host_empty	1-bit	In
Host Control		host_rw	1-bit	Out
Host Control		host_asds	1-bit	In
Host Control		host_rwen	1-bit	Out
Host Control		host_int	1-bit	Out
Host Control		host_reset	1-bit	In
Host Control		host_reserved	1-bit	Out

Table 6: USB Host Interface Signal Descriptions

Support Files

The support file names and devices used are listed in Table 7.

Name	Device Usage
components\Edges\Virtex2\fifo511x32.edn	Virtex-II
components\Edges\VirtexE\fifo511x32.edn	Virtex-E
components\Edges\All\usb_host_interface.ngc	All

Table 7: USB Host Interface Support Files

Component Definition

Component usb_host_interface_000

```
Port (
-- reset
reset : in std_logic;
-- dt_clk
dt_clk : in std_logic;
-- DIMEtalk
```

```
        data_in : in std_logic_vector(31 downto 0);
        data_out : out std_logic_vector(31 downto 0);
        data_in_ack : out std_logic;
        data_out_ack : in std_logic;
        -- host_clk
        host_clk : in std_logic;
        -- Host Data
        host_data : inout std_logic_vector(31 downto 0);
        -- Host Control
        host_busy : in std_logic;
        host_empty : in std_logic;
        host_rw : out std_logic;
        host_asds : in std_logic;
        host_rwen : out std_logic;
        host_int : out std_logic;
        host_reset : in std_logic;
        host_reserved : out std_logic
);
```

Ethernet Host Interface

Functional Description

This component handles the interface between an Ethernet module connected host and a DIMEtalk network on Nallatech stand-alone motherboards (e.g. BenNUEY and BenONE). The interface is assumed to be clocked by CLK2. This edge component converts data being transferred into the CLK1 domain. On a multi-FPGA motherboard such as the BenNUEY, the Host Data signals should be mapped to PCI_COMMS(0:31) whilst the Host Control signals should be mapped to PCI_COMMS(32:39). If this interface is instantiated on a module plugged into a BenONE carrier card the Host Data signals should be mapped to LBUS(0:31) whilst the Host Control signals should be mapped to ADJOUT(0:7). The Ethernet Host Interface signals are shown in Figure 4 and described in Table 8.

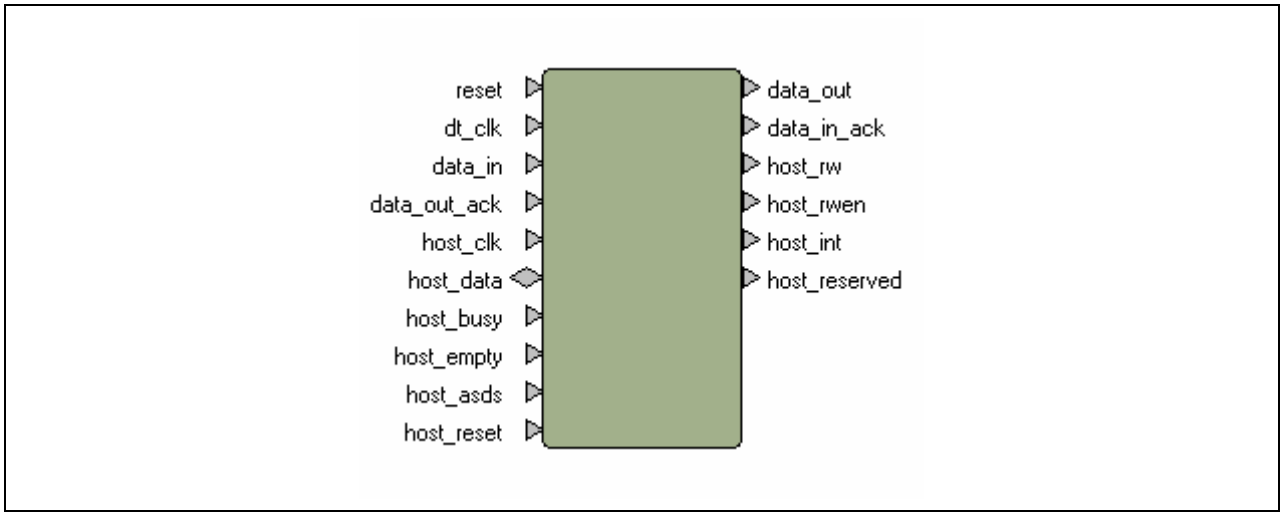

Figure 4: Ethernet Host Interface Component

Signals

The signal descriptions are provided in Table 8.

Group	Type	Signal Description	Width	Direction
reset	DIMEtalk reset	reset	1-bit	In
dt_clk	DIMEtalk network clock. Def. CLK1	dt_clk	1-bit	In
DIMEtalk		data_in	32-bit	In
DIMEtalk		data_out	32-bit	Out
DIMEtalk		data_in_ack	1-bit	Out
DIMEtalk		data_out_ack	1-bit	In
host_clk	Def. CLK2	host_clk	1-bit	In
Host Data		host_data	32-bit	Inout
Host Control		host_busy	1-bit	In
Host Control		host_empty	1-bit	In
Host Control		host_rw	1-bit	Out
Host Control		host_asds	1-bit	In
Host Control		host_rwen	1-bit	Out
Host Control		host_int	1-bit	Out
Host Control		host_reset	1-bit	In
Host Control		host_reserved	1-bit	Out

Table 8: Ethernet Host Signal Descriptions

Support Files

The support file names and devices used are listed in Table 9.

Name	Device Usage
components\Edges\Virtex2\fifo511x32.edn	Virtex-II
components\Edges\VirtexE\fifo511x32.edn	Virtex-E
components\Edges\All\ethernet_host_interface.ngc	All

Table 9: Ethernet Host Interface Support Files

Component Definition

Component ethernet_host_interface_000

Port (

```
-- reset
reset : in std_logic;
-- dt_clk
dt_clk : in std_logic;
-- DIMEtalk
```

```
data_in : in std_logic_vector(31 downto 0);
data_out : out std_logic_vector(31 downto 0);
data_in_ack : out std_logic;
data_out_ack : in std_logic;
-- host_clk
host_clk : in std_logic;
-- Host Data
host_data : inout std_logic_vector(31 downto 0);
-- Host Control
host_busy : in std_logic;
host_empty : in std_logic;
host_rw : out std_logic;
host_asds : in std_logic;
host_rwen : out std_logic;
host_int : out std_logic;
host_reset : in std_logic;
host_reserved : out std_logic
);
```

Host Testbench

Functional Description

This component, shown in Figure 5, allows PCI, USB and Ethernet Edge designs to be exercised in simulation. To run a design in simulation users should connect the Host Testbench to the ports on the edge and to the clock driver module component. When a user's VHDL output is opened in the simulator a tcl version of the DIMEtalk runtime API can be used to write into the simulation model. This is contained in the simutils.tcl output file. See "Simulation Support" for more information.

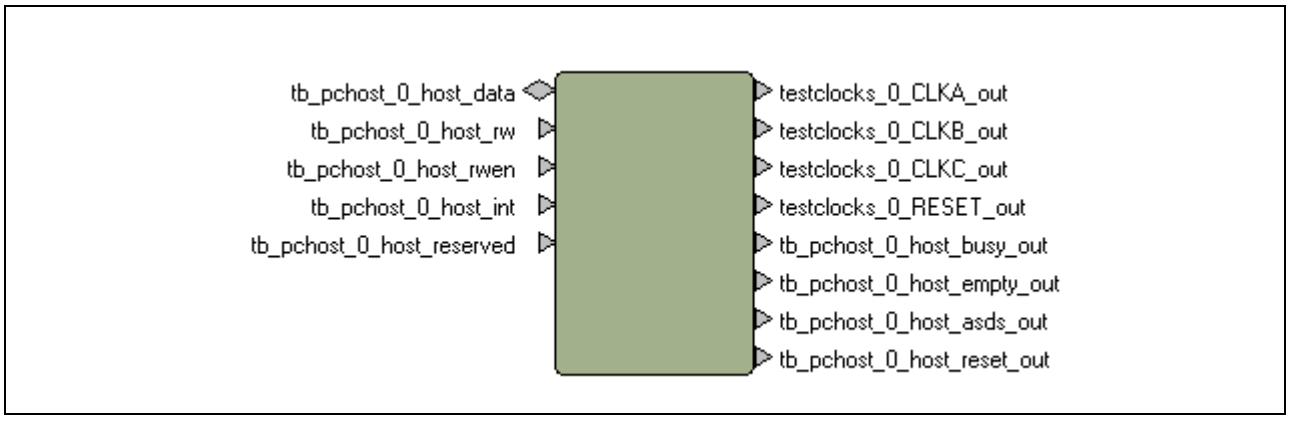

Figure 5: Host Testbench

Signals

The signal descriptions are provided in Table 10.

Group	Type	Signal Description	Width	Direction
clockreset	User Connection	testclocks_0_CLKA_out : out STD_LOGIC;	1	Out
		testclocks_0_CLKB_out : out STD_LOGIC;	1	Out
		testclocks_0_CLKC_out : out STD_LOGIC;	1	Out
		testclocks_0_RESET_out : out STD_LOGIC;	1	Out
host_interface	User Connection	tb_pchost_0_host_data : inout STD_LOGIC_VECTOR(31 downto 0);	32	Inout
		tb_pchost_0_host_busy_out : out STD_LOGIC;	1	Out
		tb_pchost_0_host_empty_out : out STD_LOGIC;	1	Out
		tb_pchost_0_host_rw : in STD_LOGIC;	1	In
		tb_pchost_0_host_asds_out : out STD_LOGIC;	1	Out
		tb_pchost_0_host_rwen : in STD_LOGIC;	1	In
		tb_pchost_0_host_int : in STD_LOGIC;	1	In
		tb_pchost_0_host_reset_out : out STD_LOGIC;	1	Out
		tb_pchost_0_host_reserved : in STD_LOGIC;	1	In

Table 10: Host Testbench Signal Descriptions

Support Files

The support file names and devices used are listed in Table 11.

Name	Device Usage
components \Edges\TestBench.support\All\TestBench.vhd	All
components \Edges\TestBench.support\All\testclocks.vhd	All
components \Edges\TestBench.support\All\pchost.vhd	All

Table 11: Host Testbench Support Files

Component Definition

```
port (
        testclocks_0_CLKA_out : out STD_LOGIC;
        testclocks_0_CLKB_out : out STD_LOGIC;
        testclocks_0_CLKC_out : out STD_LOGIC;
        testclocks_0_RESET_out : out STD_LOGIC;
        tb_pchost_0_host_data : inout STD_LOGIC_VECTOR(31 downto 0);
        tb_pchost_0_host_busy_out : out STD_LOGIC;
```

```
        tb_pchost_0_host_empty_out : out STD_LOGIC;
        tb_pchost_0_host_rw : in STD_LOGIC;
        tb_pchost_0_host_asds_out : out STD_LOGIC;
        tb_pchost_0_host_rwen : in STD_LOGIC;
        tb_pchost_0_host_int : in STD_LOGIC;
        tb_pchost_0_host_reset_out : out STD_LOGIC;
        tb_pchost_0_host_reserved : in STD_LOGIC
);
```

Simulation Support

The DIMEtalk components instantiate Xilinx DCM and BUFG primitives amongst others. In order to enable functional simulation to be performed Xilinx provides a library of simulation models known as the UNISIM library. The following sections detail how to create and access these libraries.

Xilinx Simulation Libraries for ModelSim XE

ModelSim XE is a dedicated Xilinx Edition of the tool and therefore provides a pre-compiled UNISIM library. There is no requirement to map to this library as this is built into the tool. This precompiled library is updated with new releases of ISE and can be downloaded from the software resources section of the Xilinx website.

Xilinx Simulation Libraries for ModelSim SE/PE

As these versions of ModelSim are not linked to any particular technology vendor the Xilinx simulation libraries must be compiled by the user as new versions of ISE become available.

The VHDL source for this library is provided with Xilinx ISE installs, however this code must be complied into a unisim library which the user maps to. Xilinx provide a COMPXLIB utility that automates the compilation of the simulation library. Refer to the *Xilinx Synthesis and Verification Design Guide* for full details on how to use this utility. A brief summary is provided below:-

1. Compile the unisim library. For example, to compile the unisim library for ModelSim SE for all device types the following command line could be used:-

   ```
   compxlib -s mti_pe -arch all -l vhdl -o %XILINX%/vhdl/data/mti -w
   ```

   The libraries will be created in $MYLIBDIRECTORY.

2. Map to the UNISIM library

   ```
   vmap unisim [file join [file normalize $env(XILINX)] vhdl/data/mti]
   ```

   When these steps have been performed the simulator will be able to resolve any references to the Xilinx primitives during the elaboration stage.

H100 Host Interface

The H100 Host Interface component, shown in Figure 6, handles communications between the PCI-X host and a DIMEtalk network on H100. Data is converted from the 200MHz interface clock into the DIMEtalk clock domain.

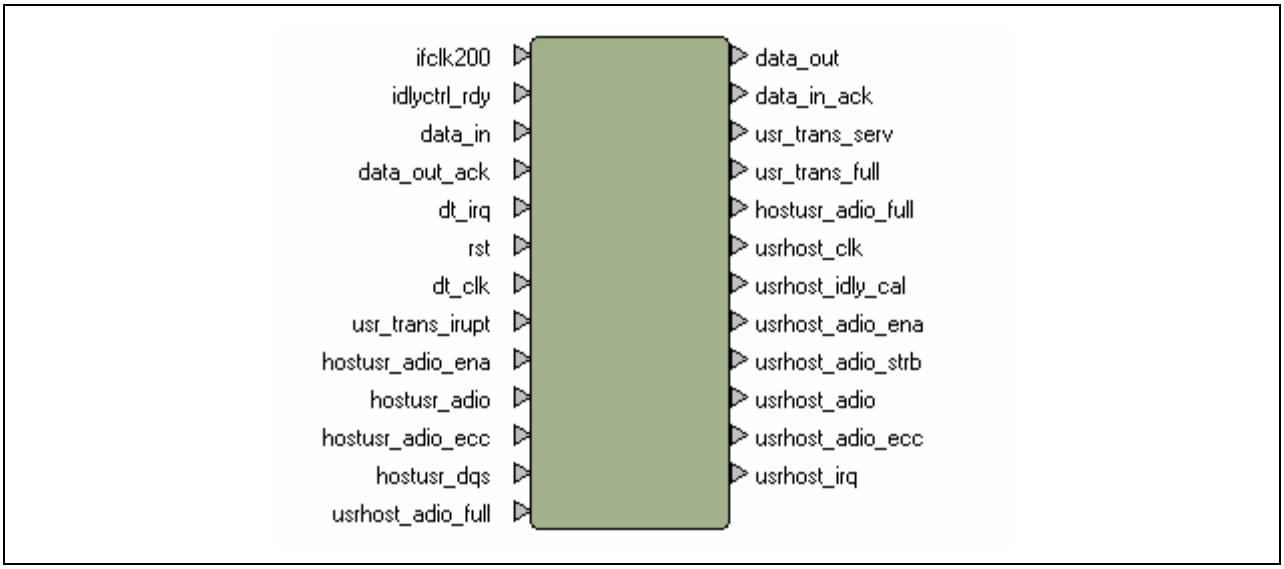

Figure 6: H100 Host Interface Component

Signals

The signal descriptions are provided in Table 12.

Group	Type	Signal Description	Width	Direction
PCIxInterface Clocks	User Connection	ifclk200 : in STD_LOGIC;	1-bit	In
		idlyctrl_rdy : in STD_LOGIC;	1-bit	In
DIMEtalk	DIMEtalk Connection	data_in : in STD_LOGIC_VECTOR(31 downto 0);	32-bits	In
		data_out : out STD_LOGIC_VECTOR(31 downto 0);	32-bits	Out
		data_in_ack : out STD_LOGIC;	1-bit	Out
		data_out_ack : in STD_LOGIC;	1-bit	In
User Signals	User Connection	dt_irq : in STD_LOGIC_VECTOR(3 downto 0);	4-bits	In
rst	Reset Connection	rst : in STD_LOGIC;	1-bit	In
dt_clk	Clock Connection	dt_clk : in STD_LOGIC;	1-bit	In

Table 12: H100 Host Interface Signal Descriptions

Group	Type	Signal Description	Width	Direction
UserPort	User Connection	usr_trans_irupt : in STD_LOGIC;	1-bit	In
		usr_trans_serv : out STD_LOGIC;	1-bit	Out
		usr_trans_full : out STD_LOGIC;	1-bit	Out
		hostusr_adio_full : out STD_LOGIC;	1-bit	Out
		hostusr_adio_ena : in STD_LOGIC;	1-bit	In
		hostusr_adio : in STD_LOGIC_VECTOR(31 downto 0);	32-bits	In
		hostusr_adio_ecc : in STD_LOGIC_VECTOR(3 downto 0);	4-bits	In
		hostusr_dqs : in STD_LOGIC;	1-bit	In
		usrhost_adio_full : in STD_LOGIC;	1-bit	In
		usrhost_clk : out STD_LOGIC;	1-bit	Out
		usrhost_idly_cal : out STD_LOGIC;	1-bit	Out
		usrhost_adio_ena : out STD_LOGIC;	1-bit	Out
		usrhost_adio_strb : out STD_LOGIC;	1-bit	Out
		usrhost_adio : out STD_LOGIC_VECTOR(31 downto 0);	32-bits	Out
		usrhost_adio_ecc : out STD_LOGIC_VECTOR(3 downto 0);	4-bits	Out
		usrhost_irq : out STD_LOGIC_VECTOR(3 downto 0);	4-bits	Out

Table 12: H100 Host Interface Signal Descriptions

Support Files

The support file names and devices used are listed in Table 13.

Name	Device Usage
DIMEtalk\library\Edges\h100_pcix_host_interface.support\All\h100_pcix_host_interface.vhd	H100
DIMEtalk\library\Edges\h100_pcix_host_interface.support\All\fifo_64x1024.ngc	H100
DIMEtalk\library\Edges\h100_pcix_host_interface.support\All\infifo64x512.ngc	H100
DIMEtalk\library\Edges\h100_pcix_host_interface.support\All\outfifo32x1024.ngc	H100
DIMEtalk\library\Edges\h100_pcix_host_interface.support\All\synfifo32x512.ngc	H100
DIMEtalk\library\Edges\h100_pcix_host_interface.support\All\h100_user_host_if.vhd	H100
DIMEtalk\library\Edges\h100_pcix_host_interface.support\All\h100_user_host_ctrl.vhd	H100
DIMEtalk\library\Edges\h100_pcix_host_interface.support\All\h100_dt_ctrl_if.vhd	H100
DIMEtalk\common\resync_rise.vhd	H100
DIMEtalk\library\Edges\h100_pcix_host_interface.support\All\h100_rx_iob.vhd	H100
DIMEtalk\library\Edges\h100_pcix_host_interface.support\All\h100_idelay_cal.vhd	H100

Table 13: H100 Host Interface Support Files

Name	Device Usage
DIMEtalk\library\Edges\h100_pcix_host_interface.support\All\h100_tx_iob.vhd	H100
DIMEtalk\library\Edges\h100_pcix_host_interface.support\All\ecc_encoder_64.vhd	H100
DIMEtalk\library\Edges\h100_pcix_host_interface.support\All\ecc_decoder_64.vhd	H100

Table 13: H100 Host Interface Support Files

1.2 Basic Internal FPGA Nodes

Block RAM Node

Functional Description

The block RAM component creates an area of shared memory. This component is generic in terms of size, and the user specifies a parameter which infers the correct depth. The depth of the generic version is limited by the available resources on the targeted device. The memory is dual port and accessible from the DIMEtalk side (by other DIMEtalk nodes) and the user side. Each has a separate clock although both are wired by default to CLK1. The block RAM signal names are shown in Figure 7 and described in Table 14.

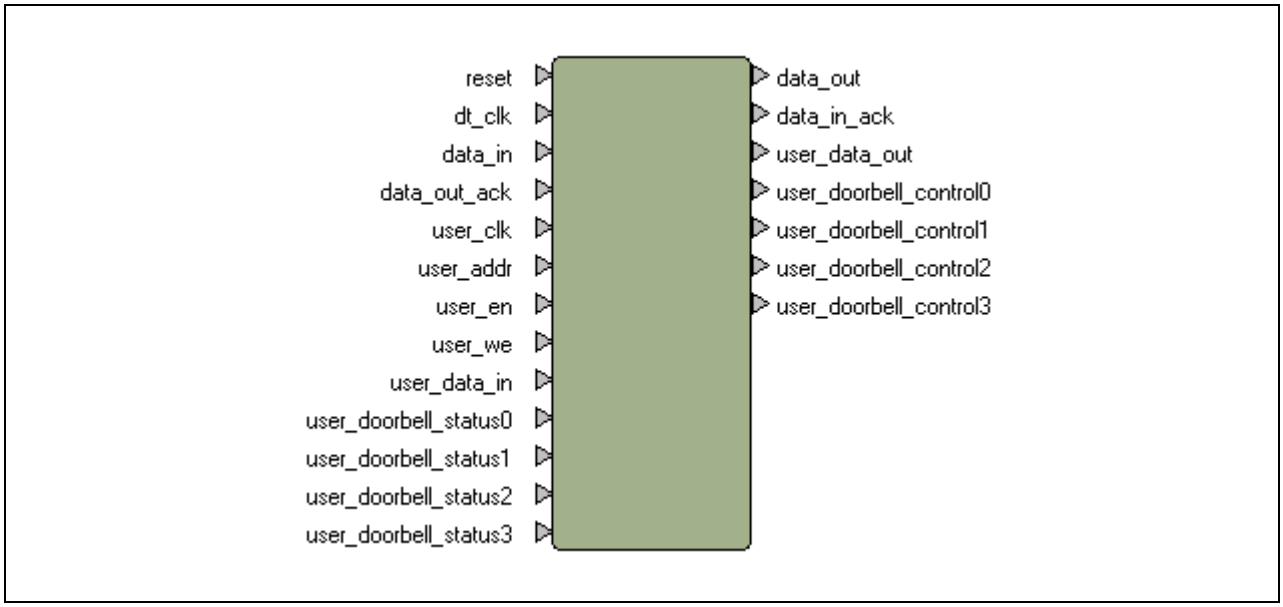

Figure 7: Block RAM Node Component

Extended Doorbell

There are four doorbell status inputs and four doorbell control outputs. The status inputs can be used to trap a change of state on an input whilst the control outputs can be used as a trigger for aspects of a user's design. For more information on packets see "Request Packet Type 10 (Doorbell class)".

Signals

The signal descriptions are provided in Table 14.

Group	Type	Signal Description	Width	Direction
reset	DIMEtalk reset	reset	1-bit	In

Table 14: Block RAM Signal Descriptions

Group	Type	Signal Description	Width	Direction
dt_clk	DIMEtalk network clock. Def.ault CLK1	dt_clk	1-bit	In
DIMEtalk		data_in	32-bits	In
DIMEtalk		data_out	32-bits	Out
DIMEtalk		data_in_ack	1-bit	Out
DIMEtalk		data_out_ack	1-bit	In
User_clk	User clock. Def. CLK1	user_clk	1-bit	In
UserInterface		User_addr	16-bits	In
UserInterface		User_en	1-bit	In
UserInterface		User_we	1-bit	In
UserInterface		User_data_in	32-bits	In
UserInterface		User_data_out	32-bits	Out
Doorbells		user_doorbell_status0	1-bit	In
Doorbells		user_doorbell_status1	1-bit	In
Doorbells		user_doorbell_status2	1-bit	In
Doorbells		user_doorbell_status3	1-bit	In
Doorbells		user_doorbell_control0	1-bit	Out
Doorbells		user_doorbell_control1	1-bit	Out
Doorbells		user_doorbell_control2	1-bit	Out
Doorbells		user_doorbell_control3	1-bit	Out

Table 14: Block RAM Signal Descriptions

User Generics

The user generics are shown in Table 15.

Name	Type	Description
MEM_AWIDTH	Natural	This sets the width of the address bus and hence the depth of the block RAM. MEM_AWIDTH = n infers a depth of 2**n, for example 9 infers a depth of 2**9 = 512 x 32 bit or 12 infers a depth of 2**12 = 4096 x 32 bit.

Table 15: Block RAM User Generics

Support Files

The support file names and devices used are listed in Table 16.

Name	Device Usage
Common/source/pkg_dimetalk_global.vhd	All
Common/source/bretime.vhd	All
Common/source/retime.vhd	All
Common/source/xst_blk_dpram.vhd	All

Table 16: Block RAM Support Files

Name	Device Usage
Common/source/dtnode_slave_control.vhd	All
BlockRam/source/block_ram.vhd	All

Table 16: Block RAM Support Files

Component Definition

```
component block_ram
generic (
        node_ID      : std_logic_vector(NODEID_SZ-1 downto 0);
        mem_awidth   : natural;
        use_synplify : boolean);
port (
        reset              : in  std_logic;
        dt_clk             : in  std_logic;
        data_in            : in  std_logic_vector(DT_DATA_SZ-1 downto 0);
        data_out           : out std_logic_vector(DT_DATA_SZ-1 downto 0);
        data_in_ack        : out std_logic;
        data_out_ack       : in  std_logic;
        user_clk           : in  std_logic;
        user_addr          : in  std_logic_vector(31 downto 0);
        user_en            : in  std_logic;
        user_we            : in  std_logic;
        user_data_in       : in  std_logic_vector(DT_DATA_SZ-1 downto 0);
        user_data_out      : out std_logic_vector(DT_DATA_SZ-1 downto 0);
        user_doorbell_status0  : in  std_logic;
        user_doorbell_status1  : in  std_logic;
        user_doorbell_status2  : in  std_logic;
        user_doorbell_status3  : in  std_logic;
        user_doorbell_control0 : out std_logic;
        user_doorbell_control1 : out std_logic;
        user_doorbell_control2 : out std_logic;
        user_doorbell_control3 : out std_logic);
end component;
```

Waveforms

User port write and read access to the block RAM is illustrated in Figure 8. To perform a write the user presents the write data on user_data_in and asserts user_en and user_we, data is written in to the RAM on the next rising edge.

To perform a read the user asserts user asserts user_en and deasserts user_we, the read data is clocked out on the next rising edge.

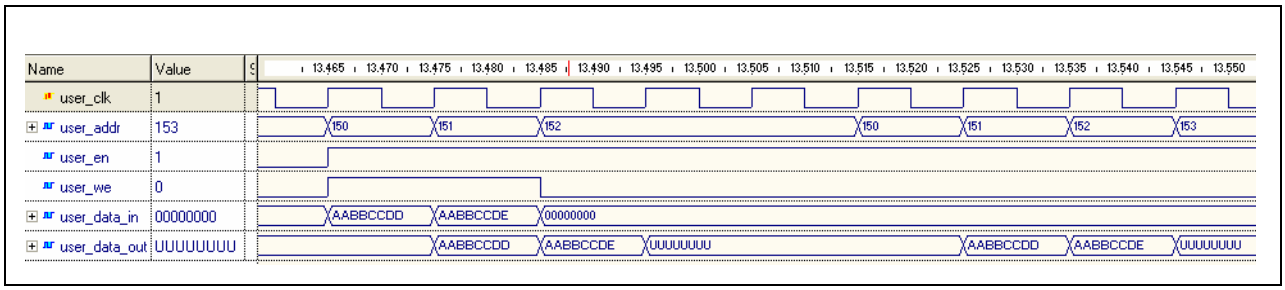

Figure 8: Block RAM Waveform

Read FIFO

Functional Description

This node implements a unidirectional 32-bit wide First In First Out memory. It has two asynchronous datapath interfaces, a DIMEtalk interface which is used to push data into the FIFO and a user read interface which is used to pop data from the FIFO. The depth of the FIFO can be parameterized and is limited only by the availability of block RAMs on the targeted device. Flow control is implemented to ensure that dimetalk data is not accepted by the node unless there is space for a maximal sized packet. The Read FIFO component is shown in Figure 9.

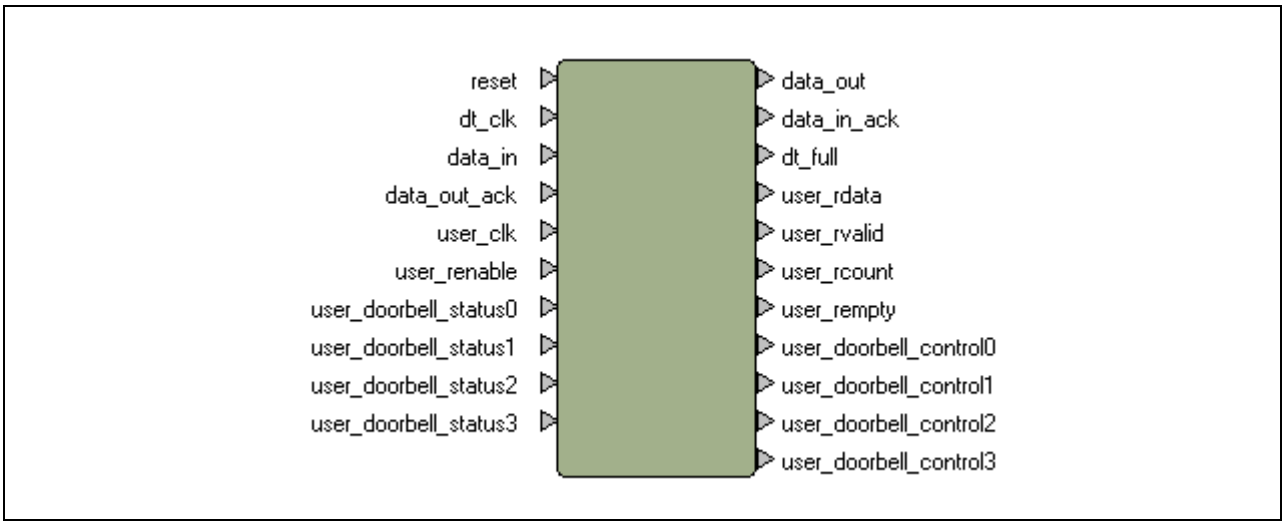

Figure 9: Read FIFO Component

Status

The DIMEtalk user can check the fill status of the FIFO by performing a read access (no particular address required). Additionally a dt_full status bit is provided which may be monitored externally to provide an indication of when the FIFO is full. The user interface is provided with a read count (user_rcount) and an empty flag (user_rempty) indicating FIFO fill in the user_clk domain.

Extended Doorbell

There are four doorbell status inputs and four doorbell control outputs. The status inputs can be used to trap a change of state on an input whilst the control outputs can be used as a trigger for aspects of a user's design. For more information on packets see "Request Packet Type 10 (Doorbell class)".

Signals

The signal descriptions are provided in Table 17.

Group	Type	Signal Description	Width	Direction
reset	DIMEtalk reset	reset	1-bit	In
dt_clk	DIMEtalk network clock. Def. CLK1	dt_clk	1-bit	In
DIMEtalk		data_in	31 downto 0 bits	In
DIMEtalk		data_out	31 downto 0 bits	Out
DIMEtalk		data_in_ack	1-bit	Out
DIMEtalk		data_out_ack	1-bit	In
FillStatus	FIFO fill status in DT clock domain	dt_full	1-bit	Out
ReadPort	User port clock, def CLK1	User_clk	1-bit	In
ReadPort	User read enable	User_renable	1-bit	In
ReadPort	User read data	User_rdata	32-bits	Out
ReadPort	User read count	User_rcount	32-bits	Out
ReadPort	User empty status	User_rempty	1-bit	Out
Doorbells		user_doorbell_status0	1-bit	In
Doorbells		user_doorbell_status1	1-bit	In
Doorbells		user_doorbell_status2	1-bit	In
Doorbells		user_doorbell_status3	1-bit	In
Doorbells		user_doorbell_control0	1-bit	Out
Doorbells		user_doorbell_control1	1-bit	Out
Doorbells		user_doorbell_control2	1-bit	Out
Doorbells		user_doorbell_control3	1-bit	Out

Table 17: Read FIFO Signal Descriptions

User Generics

The user generics are shown in Table 18.

Name	Type	Description
FIFO_PWIDTH	Natural	This indicates the width of the FIFO pointer which sets the depth of the FIFO, n infers a depth of 2**n, for example 9 infers a depth of 2**9 = 512 x 32 bit or 12 infers a depth of 2**12 = 4096 x 32 bit.

Table 18: Read FIFO User Generics

Support Files

The support file names and devices used are listed in Table 19.

Name	Device Usage
common\source\pkg_dimetalk_global.vhd	All
Common\source\bretime.vhd	All
Common\source\retime.vhd	All
Common\source\dtnode_slave_control.vhd	All
Common\source\blk_dpram.vhd	All
Common\source\blk_async_fifo.vhd	All
source\read_fifo.vhd	All

Table 19: Read FIFO Support Files

Component Definition

```
component read_fifo
        generic (
        node_id             :    std_logic_vector(NODEID_SZ-1 downto 0);
        fifo_pwidth         :    natural);
        port (
        reset               : in  std_logic;
        dt_clk              : in  std_logic;
        data_in             : in  std_logic_vector(DT_DATA_SZ-1 downto 0);
        data_out            : out std_logic_vector(DT_DATA_SZ-1 downto 0);
        data_in_ack         : out std_logic;
        data_out_ack        : in  std_logic;
        dt_full             : out std_logic;
        user_clk            : in  std_logic;
        user_renable        : in  std_logic;
        user_rdata          : out std_logic_vector(DT_DATA_SZ-1 downto 0);
        user_rvalid         : out std_logic;
        user_rcount         : out std_logic_vector(USR_FCNT_SZ-1 downto 0);
        user_rempty         : out std_logic;
        user_doorbell_status0  : in  std_logic;
        user_doorbell_status1  : in  std_logic;
        user_doorbell_status2  : in  std_logic;
        user_doorbell_status3  : in  std_logic;
        user_doorbell_control0 : out std_logic;
        user_doorbell_control1 : out std_logic;
```

```
        user_doorbell_control2 : out std_logic;
        user_doorbell_control3 : out std_logic);
end component;
```

Waveforms

User access to the Read FIFO is illustrated in Figure 10. At the start of the access user_rempty is low indicating that there is data in the FIFO. User_rcount indicates that there are six words to be read, user_renable is asserted to read from the FIFO, this read data is validated by user_rvalid in the next cycle, user_rcount is decremented accordingly, note that this count may be delayed by two to three cycles however user_rempty accurately reflects whether there is data to be read on a cycle accurate basis.

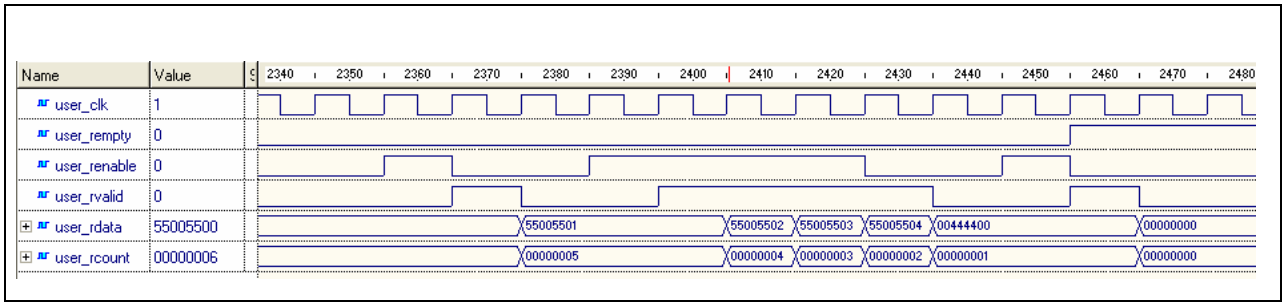

Figure 10: Read FIFO User Access

Write FIFO

Functional Description

This node implements a unidirectional 32-bit wide First In First Out memory. It has two asynchronous datapath interfaces: a DIMEtalk interface which is used to pop data from the FIFO, and a user write interface which is used to push data into the FIFO. The depth of the FIFO can be parameterized and is limited only by the availability of block RAMs on the targeted device. Flow control is implemented to ensure that a DIMEtalk data request is not completed by the node until there is enough data in the FIFO to service the request. The Write FIFO component is shown in Figure 11.

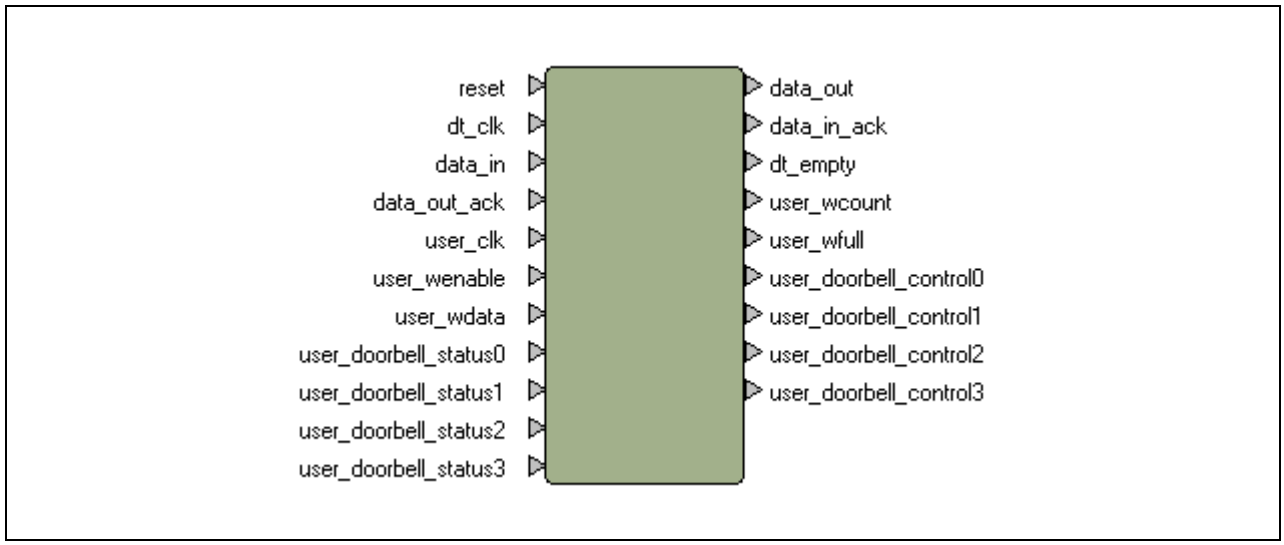

Figure 11: Write FIFO Component

Status

The DIMEtalk user can check the fill status of the FIFO by performing a read access (from address 0x8000_0000). Additionally a dt_empty status bit is provided which may be monitored externally to provide an indication of when the FIFO is empty. The user interface is provided with a write count (user_wcount) and full flag (user_wfull) indicating FIFO fill in the user_clk domain.

Extended Doorbell

There are four doorbell status inputs and four doorbell control outputs. The status inputs can be used to trap a change of state on an input whilst the control outputs can be used as a trigger for aspects of a user's design. For more information on packets see "Request Packet Type 10 (Doorbell class)".

Signals

The signal descriptions are provided in Table 20.

Group	Type	Signal Description	Width	Direction
reset	DIMEtalk reset	reset	1-bit	In
dt_clk	DIMEtalk network clock. Def. CLK1	dt_clk	1-bit	In
DIMEtalk		data_in	31 downto 0 bits	In
DIMEtalk		data_out	31 downto 0 bits	Out
DIMEtalk		data_in_ack	1-bit	Out
DIMEtalk		data_out_ack	1-bit	In
EmptyStatus	FIFO empty status in DT clock domain	dt_empty	1-bit	Out
WritePort	User port clock, def CLK1	User_clk	1 -bit	In
WritePort	User write enable	User_wenable	1-bit	In
WritePort	User write data	User_wdata	32-bits	In
WritePort	User write count	User_wcount	32-bits	Out
WritePort	User full status	User_wfull	1-bit	Out
Doorbells		user_doorbell_status0	1-bit	In
Doorbells		user_doorbell_status1	1-bit	In
Doorbells		user_doorbell_status2	1-bit	In
Doorbells		user_doorbell_status3	1-bit	In
Doorbells		user_doorbell_control0	1-bit	Out
Doorbells		user_doorbell_control1	1-bit	Out
Doorbells		user_doorbell_control2	1-bit	Out
Doorbells		user_doorbell_control3	1-bit	Out

Table 20: Write FIFO Signal Descriptions

User Generics

The user generics are shown in Table 21.

Name	Type	Description
FIFO_PWIDTH	Natural	This indicates the width of the FIFO pointer which sets the depth of the FIFO, n infers a depth of 2**n, for example 9 infers a depth of 2**9 = 512 x 32 bit or 12 infers a depth of 2**12 = 4096 x 32 bit.

Table 21: Write FIFO User Generics

Support Files

The support file names and devices used are listed in Table 22.

Name	Device Usage
common\source\pkg_dimetalk_global.vhd	All
Common\source\bretime.vhd	All
Common\source\retime.vhd	All
Common\source\dtnode_slave_control.vhd	All
Common\source\blk_dpram.vhd	All
Common\source\blk_async_fifo.vhd	All
source\write_fifo.vhd	All

Table 22: Write FIFO Support Files

Component Definition

```
component write_fifo
        generic (
        node_id          :    std_logic_vector(NODEID_SZ-1 downto 0);
        fifo_pwidth      :    natural);
port (
        reset           : in  std_logic;
        dt_clk          : in  std_logic;
        data_in         : in  std_logic_vector(DT_DATA_SZ-1 downto 0);
        data_out        : out std_logic_vector(DT_DATA_SZ-1 downto 0);
        data_in_ack     : out std_logic;
        data_out_ack    : in  std_logic;
        dt_empty        : out std_logic;
        user_clk        : in  std_logic;
        user_wenable    : in  std_logic;
        user_wdata      : in  std_logic_vector(DT_DATA_SZ-1 downto 0);
        user_wcount     : out std_logic_vector(USR_FCNT_SZ-1 downto 0);
```

```
        user_wfull          : out std_logic;
        user_doorbell_status0  : in  std_logic;
        user_doorbell_status1  : in  std_logic;
        user_doorbell_status2  : in  std_logic;
        user_doorbell_status3  : in  std_logic;
        user_doorbell_control0 : out std_logic;
        user_doorbell_control1 : out std_logic;
        user_doorbell_control2 : out std_logic;
        user_doorbell_control3 : out std_logic);
end component;
```

Waveforms

User access to the Write FIFO is illustrated in Figure 12. To write data into the FIFO the user presents the data on user_wdata and asserts user_we. User_wcount indicates how many words are in the FIFO, however this will typically be delayed by two to three cycles due to internal cross clock domain handling. User_wfull gives a timely indication of the FIFO fill status as it is asserted as soon as the last location in the FIFO is filled.

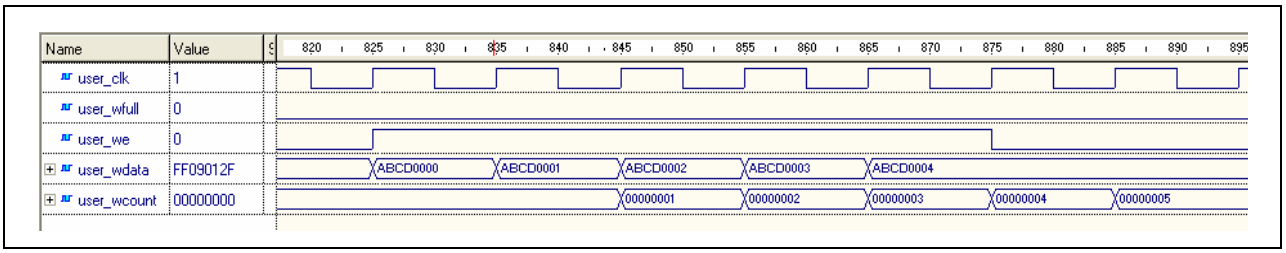

Figure 12: Write FIFO User Access

Memory Map Node

Functional Description

This is a basic slave node which is intended to allow users to develop register blocks or specialist memory schemes. This node assumes that the destination of the data can handle any data that arrives at the node. If this is not the case, a FIFO node would be a more appropriate choice. The user signals are synchronized to the DIMEtalk clock domain. The Memory Map signal names are shown in Figure 13 and described in Table 23.

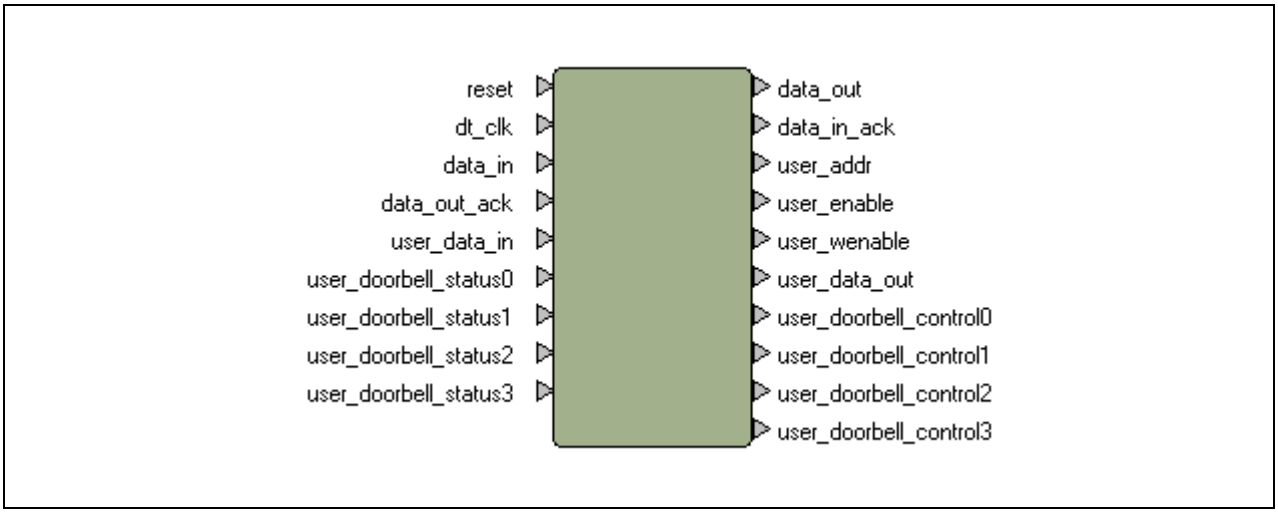

Figure 13: Memory Map Node Component

Extended Doorbell

There are four doorbell status inputs and four doorbell control outputs. The status inputs can be used to trap a change of state on an input whilst the control outputs can be used as a trigger for aspects of a user's design. For more information on packets see "Request Packet Type 10 (Doorbell class)".

Signals

The signal descriptions are provided in Table 23.

Group	Type	Signal Description	Width	Direction
reset	DIMEtalk reset	reset	1-bit	In
dt_clk	DIMEtalk network clock. Def. CLK1	dt_clk	1-bit	In
DIMEtalk		data_in	31 downto 0 bits	In
DIMEtalk		data_out	31 downto 0 bits	Out
DIMEtalk		data_in_ack	1-bit	Out
DIMEtalk		data_out_ack	1-bit	In
UserInterface		User_addr	32-bits	Out
UserInterface		User_enable	1-bit	Out
UserInterface		User_wenable	1-bit	Out
Doorbells		user_doorbell_status0	1-bit	In
Doorbells		user_doorbell_status1	1-bit	In

Table 23: Memory Map Signal Descriptions

Group	Type	Signal Description	Width	Direction
Doorbells		user_doorbell_status2	1-bit	In
Doorbells		user_doorbell_status3	1-bit	In
Doorbells		user_doorbell_control0	1-bit	Out
Doorbells		user_doorbell_control1	1-bit	Out
Doorbells		user_doorbell_control2	1-bit	Out
Doorbells		user_doorbell_control3	1-bit	Out

Table 23: Memory Map Signal Descriptions

User Generics

There are no user generics associated with this module.

Support Files

The support file names and devices used are listed in Table 24.

Name	Device Usage
common\source\pkg_dimetalk_global.vhd	All
Common\source\bretime.vhd	All
Common\source\retime.vhd	All
Common\source\dtnode_slave_control.vhd	All
MemoryMap\source\memory_map.vhd	All

Table 24: Memory Map Support Files

Component Definition

```
component memory_map
        generic (
        node_ID          :    std_logic_vector(NODEID_SZ-1 downto 0));
        port (
        reset           : in  std_logic;
        dt_clk          : in  std_logic;
        data_in         : in  std_logic_vector(DT_DATA_SZ-1 downto 0);
        data_out        : out std_logic_vector(DT_DATA_SZ-1 downto 0);
        data_in_ack     : out std_logic;
        data_out_ack    : in  std_logic;
        user_addr       : out std_logic_vector(DT_DATA_SZ-1 downto 0);
        user_wenable    : out std_logic;
        user_data_out   : out std_logic_vector(DT_DATA_SZ-1 downto 0);
        user_renable    : out std_logic;
        user_data_in    : in  std_logic_vector(DT_DATA_SZ-1 downto 0);
        user_doorbell_status0  : in  std_logic;
```

```
        user_doorbell_status1  : in  std_logic;
        user_doorbell_status2  : in  std_logic;
        user_doorbell_status3  : in  std_logic;
        user_doorbell_control0 : out std_logic;
        user_doorbell_control1 : out std_logic;
        user_doorbell_control2 : out std_logic;
        user_doorbell_control3 : out std_logic);
end component;
```

Waveforms

The memory map user interface follows the functional timing that is required to interface to Xilinx block RAM. Figure 14 shows that for writes the user_addr is qualified by the assertion of user_wenable and user_enable. The example shows a write to five words at locations 0x10 to 0x14.

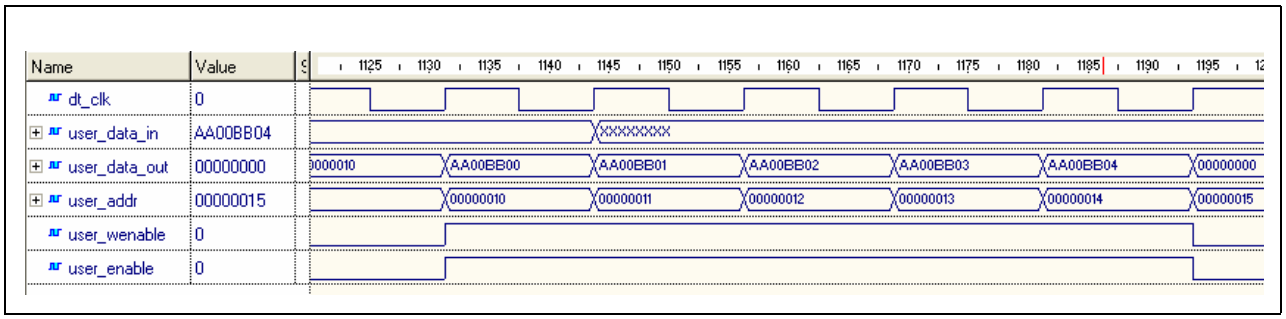

Figure 14: Memory Map Write

For read accesses user_wenable is deasserted but user_enable is used to qualify user_addr as for writes. Once addressed the read data must be returned in the next clock cycle. This is illustrated in Figure 15 where the node addresses 0x10 and the relevant data (0xAA00BB00) is returned in the next clock cycle.

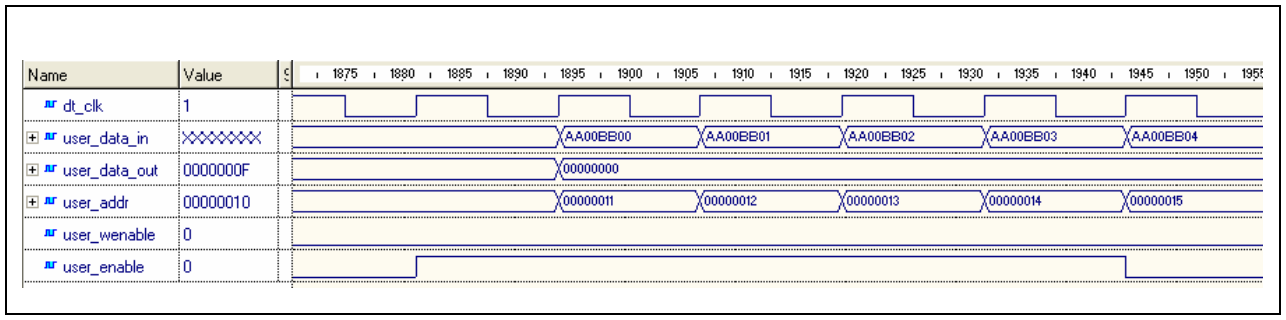

Figure 15: Memory Map Read

FIFO Loopback

Functional Description

This component allows the user interfaces of the Read and Write FIFOs to be connected together in order to provide a loopback test. Once data is written into the Read FIFO by the DIMEtalk host it is read out and written into the Write FIFO to be read by the DIMEtalk host. The FIFO Loopback signals are shown in Figure 16 and described in Table 25.

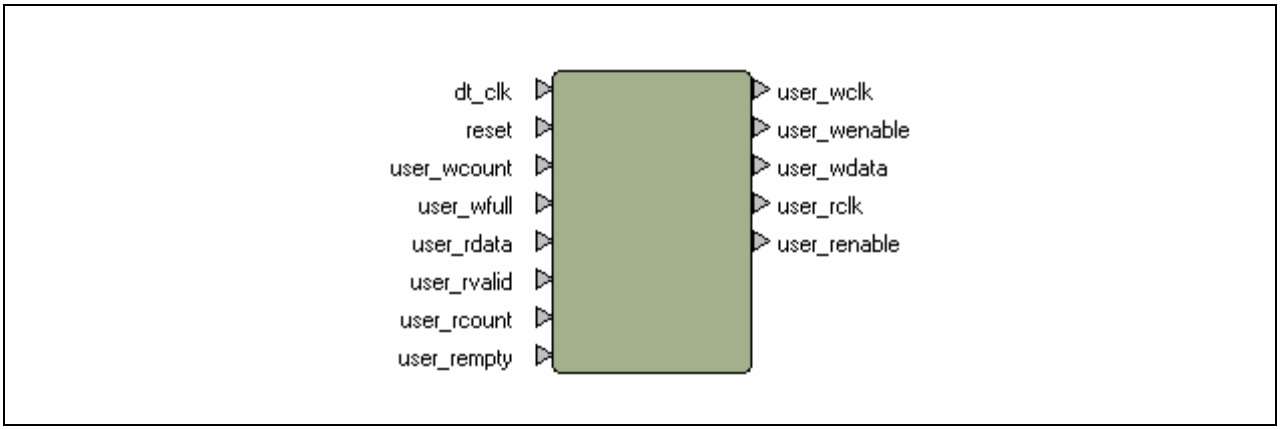

Figure 16: FIFO Loopback Component

Signals

The signal descriptions are provided in Table 25.

Group	Type	Signal Description	Width	Direction
reset	DIMEtalk reset	reset	1-bit	In
dt_clk	DIMEtalk network clock. Def. CLK1	dt_clk	1-bit	In
Write Port	Write clock, sourced from dt_clk	User_wclk	1-bit	Out
Write Port		User_wenable	1-bit	Out
Write Port		User_wdata	32-bits	Out
Write Port		User_wcount	32-bits	In
Write Port		User_wfull	1-bit	In
Read Port	Read clock, sourced from dt_clk	User_rclk	1-bit	Out
Read Port		User_renable	1-bit	Out
Read Port		User_rdata	32-bits	In
Read Port		User_rcount	32-bits	Out
Read Port		User_rempty	1-bit	Out
Read Port		User_rvalid	1-bit	In

Table 25: FIFO Loopback Signal Descriptions

Support Files

The support file names and devices used are listed in Table 26.

Name	Device Usage
fifo_loopback.vhd	All

Table 26: FIFO Loopback Support Files

Component Definition

```
component fifo_loopback is
port(
        dt_clk      : in std_logic;
        reset       : in std_logic;
        -- User port - Write FIFO
        user_wclk   : out std_logic;
        user_wenable : out std_logic;
        user_wdata  : out std_logic_vector(31 downto 0);
        user_wcount :  in  std_logic_vector(15 downto 0);
        user_wfull  : in  std_logic;
        -- User Port - Read FIFO
        user_rclk   : out std_logic;
        user_renable : out std_logic;
        user_rdata  : in  std_logic_vector(31 downto 0);
        user_rvalid  : in  std_logic;
        user_rcount  : in  std_logic_vector(15 downto 0);
        user_rempty  : in  std_logic
);
```

Memory Map Loopback

Functional Description

This component allows the user to check that they can read and write from a memory map node by interfacing it to a 512 x 32-bit Xilinx block RAM. The component is shown in Figure 17.

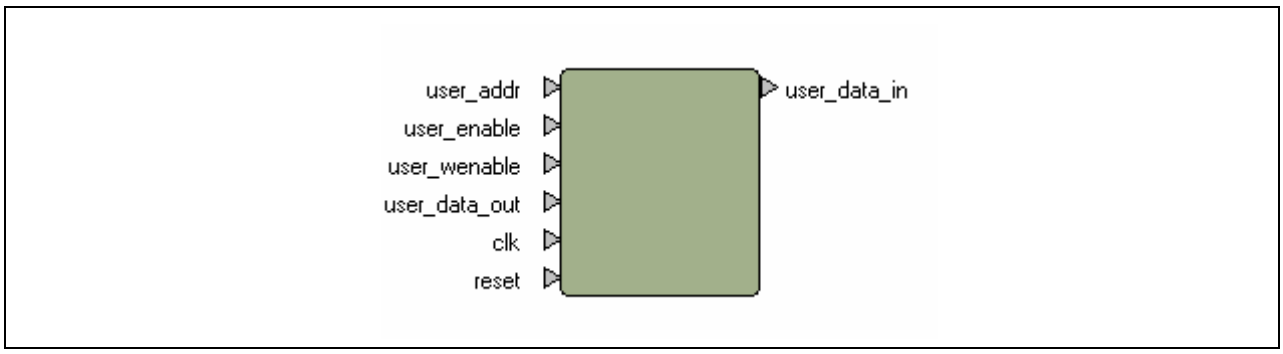

Figure 17: Memory Map Loopback

Signals

The signal descriptions are provided in Table 27.

Group	Type	Signal Description	Width	Direction
reset	DIMEtalk reset	reset	1-bit	In
dt_clk	DIMEtalk network clock. Def. CLK1	clk	1-bit	In
User Port		User_addr	32-bits	In
User Port		User_enable	1-bit	In
User Port		User_wenable	1-bit	In
User Port		User_data_out	32-bits	in
User Port		User_data_in	32-bits	Out

Table 27: Memory Map Loopback Signal Descriptions

Support Files

The support file names and devices used are listed in Table 28.

Name	Device Usage
DIMEtalk\library\Basic internal FPGA nodes\memorymap_test.support\All\memorymap_test.vhd	All
DIMEtalk\library\Basic internal FPGA nodes\memorymap_test.support\Simulation_Virtex2\blockram512x32.vhd	Simulation Virtex-II
DIMEtalk\library\Basic internal FPGA nodes\memorymap_test.support\Simulation_VirtexE\blockram512x32.vhd	Simulation Virtex-E
DIMEtalk\Common\xst_blk_dprw_ram.vhd	All

Table 28: Memory Map Loopback Support Files

Component Definition

```
entity memorymap_test is
port (
        user_addr : in STD_LOGIC_VECTOR(31 downto 0);
        user_enable : in STD_LOGIC;
        user_wenable : in STD_LOGIC;
        user_data_out : in STD_LOGIC_VECTOR(31 downto 0);
        user_data_in : out STD_LOGIC_VECTOR(31 downto 0);
        clk : in STD_LOGIC;
        reset : in STD_LOGIC
);
```

Master DIMEtalk Node

Functional Description

This component allows the user to handle the production of DIMEtalk packets within a DIMEtalk hardware network. It consists of two FIFOs - one receives packets from the network (rx), the other sends packets to the network (tx). To send a packet onto the network, the user places the data words in the tx FIFO (up to 64 words). This is done by presenting data on user_txdata and raising user_txdata_ena. For each user_clk rising edge at which user_txdata_ena is high, one word is written from user_txdata into the FIFO. Once the data has been loaded, the user places the packet header information onto:

user_tx_destination - Destination of packet.

user_tx_msg_type - 000 = read response, 001 = write packet, 010 = read packet.

user_tx_address - The address on the destination to which the packet is destined.

user_tx_datasize - Maximum 64. Note that in a read request this is the number of words requested NOT the packet size.

user_tx_ready is then raised. Once user_tx_ack goes high, the data has been latched in and the packet will be sent at the next available opportunity.

user_tx_busy indicates that a packet is currently underway and has not yet been completely sent.

To receive a packet, the user must wait for user_rx_ready to raise. This indicates that the rx header information is valid.

user_rx_source - source of the packet.

user_rx_msg_type - see user_tx_msg_type for definition.

user_rx_address - address that packet is destined for on this node.

user_rx_datasize - number of datawords (or for a read request, this is the number of words required in response).

Once these have been registered, the user can read the data by putting user_rxdata_ena high. Note that the first word out is on the second clock after raising this signal. Once the user has processed the packet, user_rx_ack should be raised to indicate completion of the packet. This allows the node to receive further packets.

The Master Node signals are shown in Figure 18 and described in Table 29.

.

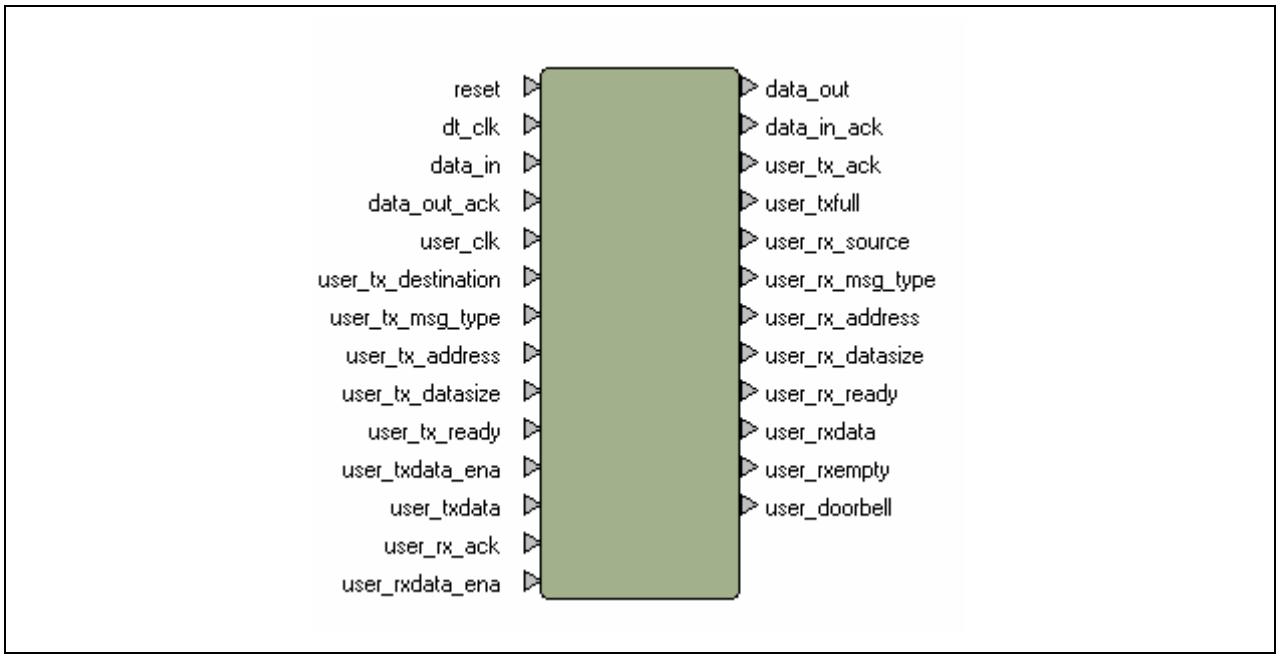

Figure 18: Master DIMEtalk Node

Signals

The signal descriptions are provided in Table 29.

Group	Type	Signal Description	Width	Direction
reset	DIMEtalk reset	reset	1-bit	In
dt_clk	DIMEtalk network clock. Def. CLK1	dt_clk	1-bit	In
DIMEtalk		data_in	32-bits	In
DIMEtalk		data_out	32-bits	Out
DIMEtalk		data_in_ack	1-bit	Out
DIMEtalk		data_out_ack	1-bit	In
user_clk	User clock. Def. CLK1	user_clk	1-bit	In
user_interface		user_tx_destination	8-bits	In
user_interface		user_tx_msg_type	3-bits	In
user_interface		user_tx_address	32-bits	In
user_interface		user_tx_datasize	8-bits	In
user_interface		user_tx_ready	1-bit	In
user_interface		user_tx_ack	1-bit	Out
user_interface		user_tx_busy	1-bit	Out
user_interface		user_txdata_ena	1-bit	In
user_interface		user_txdata	32-bits	In
user_interface		user_txfull	1-bit	Out

Table 29: Master Node Signal Descriptions

Group	Type	Signal Description	Width	Direction
user_interface		user_rx_source	8-bits	Out
user_interface		user_rx_msg_type	3-bits	Out
user_interface		user_rx_address	32-bits	Out
user_interface		user_rx_datasize	8-bits	Out
user_interface		user_rx_ready	1-bit	Out
user_interface		user_rx_ack	1-bit	In
user_interface		user_rxdata_ena	1-bit	In
user_interface		user_rxdata	32-bits	Out
user_interface		user_rxempty	1-bit	Out
user_doorbell		user_doorbell	1-bit	Out

Table 29: Master Node Signal Descriptions

Support Files

The support file names and devices used are listed in Table 30.

Name	Device Usage
DIMEtalk\library\Basic internal FPGA nodes\master_wrapper.support\All\master_wrapper.vhd	All
DIMEtalk\library\Basic internal FPGA nodes\master_wrapper.support\All\master_node.vhd	All
DIMEtalk\library\Basic internal FPGA nodes\master_wrapper.support\All\fifo511x32_ve.edn	All
DIMEtalk\library\Basic internal FPGA nodes\master_wrapper.support\All\fifo511x32.edn	All
DIMEtalk\library\Basic internal FPGA nodes\master_wrapper.support\Simulation_All\fifo511x32_ve.vhd	Simulation
DIMEtalk\library\Basic internal FPGA nodes\master_wrapper.support\Simulation_All\fifo511x32.vhd	Simulation

Table 30: Master Node Support Files

Component Definition

```
Component master_000
        Generic (
        node_address : std_logic_vector( 7 downto 0) := x"00";
        node_id : std_logic_vector( 7 downto 0) := x"00"
        );
        Port (
        -- reset
        reset : in std_logic;
        -- dt_clk
        dt_clk : in std_logic;
        -- DIMEtalk
        data_in : in std_logic_vector(31 downto 0);
        data_out : out std_logic_vector(31 downto 0);
```

```
data_in_ack : out std_logic;
data_out_ack : in std_logic;
-- user_clk
user_clk : in std_logic;
-- user_interface
user_tx_destination : in std_logic_vector(7 downto 0);
user_tx_msg_type : in std_logic_vector(2 downto 0);
user_tx_address : in std_logic_vector(31 downto 0);
user_tx_datasize : in std_logic_vector(7 downto 0);
user_tx_ready : in std_logic;
user_tx_ack : out std_logic;
user_tx_busy : out std_logic;
user_txdata_ena : in std_logic;
user_txdata : in std_logic_vector(31 downto 0);
user_txfull : out std_logic;
user_rx_source : out std_logic_vector(7 downto 0);
user_rx_msg_type : out std_logic_vector(2 downto 0);
user_rx_address : out std_logic_vector(31 downto 0);
user_rx_datasize : out std_logic_vector(7 downto 0);
user_rx_ready : out std_logic;
user_rx_ack : in std_logic;
user_rxdata_ena : in std_logic;
user_rxdata : out std_logic_vector(31 downto 0);
user_rxempty : out std_logic;
-- user_doorbell
user_doorbell : out std_logic
);
```

Example VHDL Component Description

A simple example Master Node controller has been included in the 'Components\Examples' area (Example Master Node Driver) of the DIMEtalk installation. This comes with VHDL showing a simple interface to the Master Node component. There is also an example project in 'projects\example projects\master node' on the DIMEtalk installation. This contains a DIMEscript (master_test.dsc) file that will test the resulting network. The VHDL code (see "Example VHDL Component Code") implements a simple controller for the DIMEtalk Master Node. It expects a write packet from another master (the PC host, for example) containing the following three words:

Address of a block RAM node - DEST

Size of test packet - SIZE

Address of test packet - ADDR

It then generates a read request to DEST for SIZE words at address ADDR (note SIZE must be 64 or less). The destination node returns the words at ADDR. This test code adds one to each and generates a write request to the

destination node with this modified data. Once complete, it sends a write packet to the originator of the initial request. This write contains one word indicating the number of clock cycles between starting to send the read request and receiving the last word from the read response. If a host node with ID 0, a master node with ID 1 and a block RAM node with ID 2 are assumed, the following steps are carried out:

1. Host node sends write packet to node 2 containing 64 words 0-63 to be written to address 0.
2. Host node sends write packet to node 1 containing 2 64 0.
3. Host node awaits write packet from node 1.
4. Master node sends read request to node 2 for 64 words at address 0.
5. Block RAM node sends read response with data 0-63.
6. Master node sends write packet containing data 1-64 to node 2.
7. Master node sends write packet containing cycle count to node 0.
8. Host node sends read request to node 2 for 64 words at address 0
9. Block RAM node sends read response with data 1-64.

Example VHDL Component Code

```
--------------------------------------------------------------------------------
-- Title       : DimeTalk Master Node Test
-- Project     : DimeTalk
--------------------------------------------------------------------------------
-- File        : master_node_test.vhd
-- Author      : Derek McAulay
-- Company     : Nallatech
--------------------------------------------------------------------------------
-- Description : Test harness that is receives a packet from an Edge
-- containing a destination address and packet size
--
--
--------------------------------------------------------------------------------
-- Known Issues and Omissions :
--  - none
--
--------------------------------------------------------------------------------
-- Copyright © 2003 Nallatech Ltd.  All rights reserved
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity master_node_test is
port (
  -- User Port ------------------------------------------------
  reset              : in  std_logic;
  user_clk           : in  std_logic;
  -- User Transmit Port - Control
  user_tx_destination : out std_logic_vector( 7 downto 0);
  user_tx_msg_type    : out std_logic_vector( 2 downto 0);
  user_tx_address     : out std_logic_vector(31 downto 0);
  user_tx_datasize    : out std_logic_vector( 7 downto 0);
  user_tx_ready       : out std_logic;
  user_tx_ack         : in  std_logic;
  user_tx_busy        : in  std_logic;
  -- User Transmit port - Data FIFO
  user_txdata_ena     : out std_logic;
  user_txdata         : out std_logic_vector(31 downto 0);
  user_txfull         : in  std_logic;
  -- User Receive Port - Status
  user_rx_source      : in  std_logic_vector( 7 downto 0);
  user_rx_msg_type    : in  std_logic_vector( 2 downto 0);
```

```
  user_rx_address     : in  std_logic_vector(31 downto 0);
  user_rx_datasize    : in  std_logic_vector( 7 downto 0);
  user_rx_ready       : in  std_logic;
  user_rx_ack         : out std_logic;
  -- User Recieve Port - Data FIFO
  user_rxdata_ena     : out std_logic;
  user_rxdata         : in  std_logic_vector(31 downto 0);
  user_rxempty        : in  std_logic;
  -- User Port - doorbell
  user_doorbell       : in  std_logic
  );
end master_node_test;

architecture rtl of master_node_test is

  signal destination        : std_logic_vector( 7 downto 0);
  signal source             : std_logic_vector( 7 downto 0);
  signal datasize           : std_logic_vector( 7 downto 0);
  signal addr_at_dest       : std_logic_vector(31 downto 0);

  signal packet_count       : std_logic_vector( 6 downto 0);
  signal timer              : std_logic_vector(31 downto 0);
  signal timer_ena          : std_logic;
  signal timer_rst          : std_logic;

  -- master node test state machine
  type t_mnt_state is (IDLE,                RECEIVE_ACK,          GET_DEST,             GET_SIZE,
                       GET_ADDR,            REG_ADDR,             REQ_PACKET_BRN,       WAIT_FOR_BRN_PACKET,
                         ACCEPT_BRN_PACKET,   GET_BRN_PACKET,       MOD_BRN_PACKET1,      MOD_BRN_PACKET2,
                       MOD_BRN_PACKET3,     RETURN_BRN_PACKET,    TX_FINISH,            WRITE_FINISH_PACKET,
                          SEND_FINISH_PACKET,  WRITE_TIMER,          SEND_TIMER);
  signal mnt_state : t_mnt_state;

  -- constant decalaration
  constant read_packet      : std_logic_vector( 2 downto 0) := "010";
  constant write_packet     : std_logic_vector( 2 downto 0) := "001";
  constant read_response    : std_logic_vector( 2 downto 0) := "000";

begin

  mnt_control: process(user_clk, reset)
  begin
    if (reset = '1') then
      mnt_state           <= IDLE;

      timer_ena           <= '0';
      timer_rst           <= '0';
      timer               <= (others => '0');

      packet_count        <= (others => '0');
      destination         <= (others => '0');
      source              <= (others => '0');
      datasize            <= (others => '0');
      addr_at_dest        <= (others => '0');

      user_rx_ack         <= '0';
      user_rxdata_ena     <= '0';

      user_txdata_ena     <= '0';
      user_txdata         <= (others => '0');
      user_tx_destination <= (others => '0');
      user_tx_msg_type    <= (others => '0');
      user_tx_address     <= (others => '0');
      user_tx_datasize    <= (others => '0');
      user_tx_ready       <= '0';

    elsif rising_edge(user_clk) then


      -- timer -->  counter that is active while master node requests
      -- a read from the BRN and then returns the data to the BRN with
      -- a write packet
      if (timer_ena = '1') then
        timer <= timer + 1;
      elsif (timer_rst = '1') then
        timer <= (others => '0');
      end if;

      -- default fsm signal values
      user_rx_ack         <= '0';
      user_rxdata_ena     <= '0';
      user_txdata_ena     <= '0';
```

```
user_tx_ready       <= '0';
packet_count        <= (others => '0');
timer_rst           <= '0';

-- fsm
case(mnt_state) is

  when IDLE =>
    if (user_rx_ready = '1' and user_rx_msg_type = "001") then
      mnt_state <= RECEIVE_ACK;
    elsif (user_rx_ready = '1' and user_rx_msg_type = "010") then
      mnt_state <= WRITE_TIMER;
    end if;

  when RECEIVE_ACK =>
    if (user_rx_ready = '0') then
      mnt_state <= GET_DEST;
      user_rxdata_ena <= '1';
    end if;

    -- acknowledge the receive port and accept the packet
    user_rx_ack <= '1';

  when GET_DEST =>
    mnt_state       <= GET_SIZE;
    user_rxdata_ena <= '1';
    source          <= user_rx_source;
  when GET_SIZE =>
    mnt_state       <= GET_ADDR;
    user_rxdata_ena <= '1';
    destination     <= user_rxdata(7 downto 0);
  when GET_ADDR =>
    mnt_state       <= REG_ADDR;
    datasize        <= user_rxdata(7 downto 0);
  when REG_ADDR =>
    mnt_state       <= REQ_PACKET_BRN;
    addr_at_dest    <= user_rxdata;
    timer_rst <= '1';

  when REQ_PACKET_BRN =>
    if (user_tx_ack = '1') then
      mnt_state     <= WAIT_FOR_BRN_PACKET;
    end if;

    timer_ena           <= '1';

    user_tx_ready       <= '1';

    user_tx_destination <= destination;
    user_tx_msg_type    <= read_packet;
    user_tx_address     <= addr_at_dest;
    user_tx_datasize    <= datasize;

  when WAIT_FOR_BRN_PACKET =>
    if (user_rx_ready = '1') then
      mnt_state <= ACCEPT_BRN_PACKET;
    end if;

  when ACCEPT_BRN_PACKET =>
    if (user_rx_ready = '0') then
      mnt_state <= GET_BRN_PACKET;
    end if;

    user_rx_ack <= '1';

  when GET_BRN_PACKET =>
    mnt_state       <= MOD_BRN_PACKET1;
    user_rxdata_ena <= '1';
    packet_count    <= packet_count + 1;

  when MOD_BRN_PACKET1 =>
    mnt_state       <= MOD_BRN_PACKET2;

    user_rxdata_ena <= '1';
    packet_count    <= packet_count + 1;

  when MOD_BRN_PACKET2 =>
    if (packet_count = user_rx_datasize) then
      mnt_state       <= MOD_BRN_PACKET3;
    end if;
    user_rxdata_ena <= '1';
    user_txdata_ena <= '1';
    user_txdata     <= user_rxdata + 1;
```

```
            packet_count    <= packet_count + 1;

          when MOD_BRN_PACKET3 =>
            mnt_state       <= RETURN_BRN_PACKET;
            user_txdata_ena <= '1';
            user_txdata     <= user_rxdata + 1;
            timer_ena       <= '0';
          when RETURN_BRN_PACKET =>
            if (user_tx_ack = '1') then
              mnt_state         <= TX_FINISH;
            end if;

            user_tx_ready       <= '1';
            user_tx_destination <= destination;
            user_tx_msg_type    <= write_packet;
            user_tx_address     <= addr_at_dest;
            user_tx_datasize    <= datasize;

          when TX_FINISH =>
            if user_tx_busy = '0' then
              mnt_state <= WRITE_FINISH_PACKET;
            end if;

          when WRITE_FINISH_PACKET =>
            mnt_state           <= SEND_FINISH_PACKET;
            user_txdata_ena     <= '1';
            user_txdata         <= timer;

          when SEND_FINISH_PACKET =>
            if (user_tx_ack = '1') then
              mnt_state <= IDLE;
            end if;

            user_tx_ready       <= '1';
            user_tx_destination <= source;
            user_tx_msg_type    <= write_packet;
            user_tx_address     <= (others => '0');
            user_tx_datasize    <= x"01";

          when WRITE_TIMER =>
            mnt_state           <= SEND_TIMER;
            user_txdata_ena     <= '1';
            user_txdata         <= timer;

          when SEND_TIMER =>
            if (user_tx_ack = '1') then
              mnt_state <= IDLE;
            end if;

            user_tx_ready       <= '1';
            user_tx_destination <= source;
            user_tx_msg_type    <= read_response;
            user_tx_address     <= (others => '0');
            user_tx_datasize    <= x"01";

        end case;
      end if;
    end process mnt_control;

end rtl;
```

Waveforms

The DIMEtalk Master Node can transmit and receive packets from other DIMEtalk nodes in the network. The two waveforms in Figure 19 and Figure 20 illustrate the required protocol to transmit packets to other nodes. The data to be transmitted must firstly be written into the user_txdata FIFO port by asserting the user_txdata_ena signal on the rising edge of the user_clk. Note that the user_txfull flag is used to indicate when the tx FIFO is full and cannot accept anymore data.

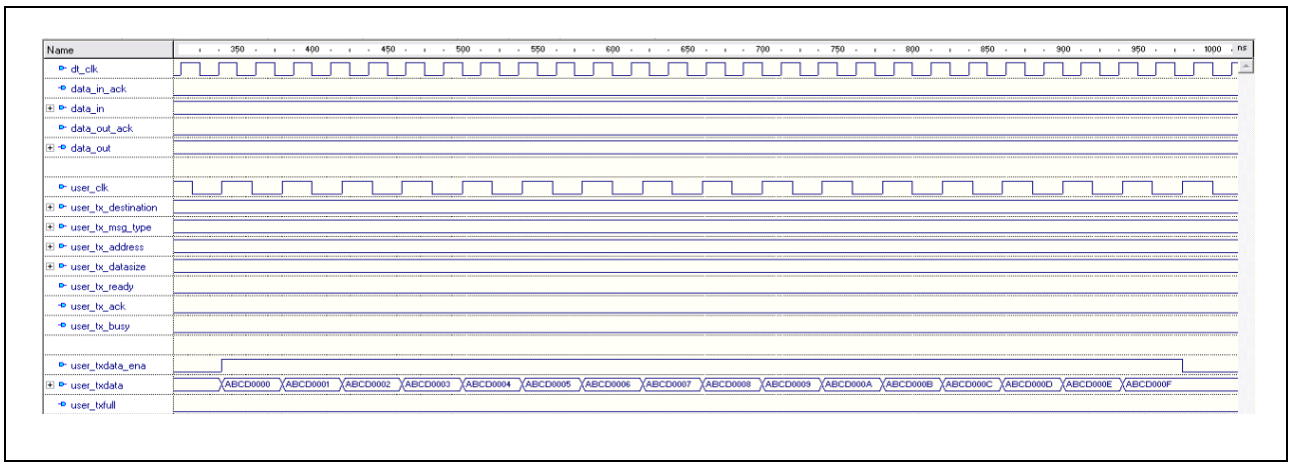

Figure 19: Master Node: tx data write (16 data words)

Once all the data to be transmitted has been written into the user_txdata port, DIMEtalk command data must be passed into the master node in order for the node to successfully construct the packet to be transmitted. To initiate a packet transfer, the user_tx_ready signal must be asserted on the rising edge of the user_clk. Note that the command data must be valid on the assertion of user_tx_ready. If the Master Node is capable of transmitting the data, then it will assert the user_tx_ack signal to acknowledge to the user that the data will be transmitted. The user_tx_busy flag is used to tell the user that the transmit part of the Master Node is currently busy transmitting a packet and cannot service any other transmit requests.

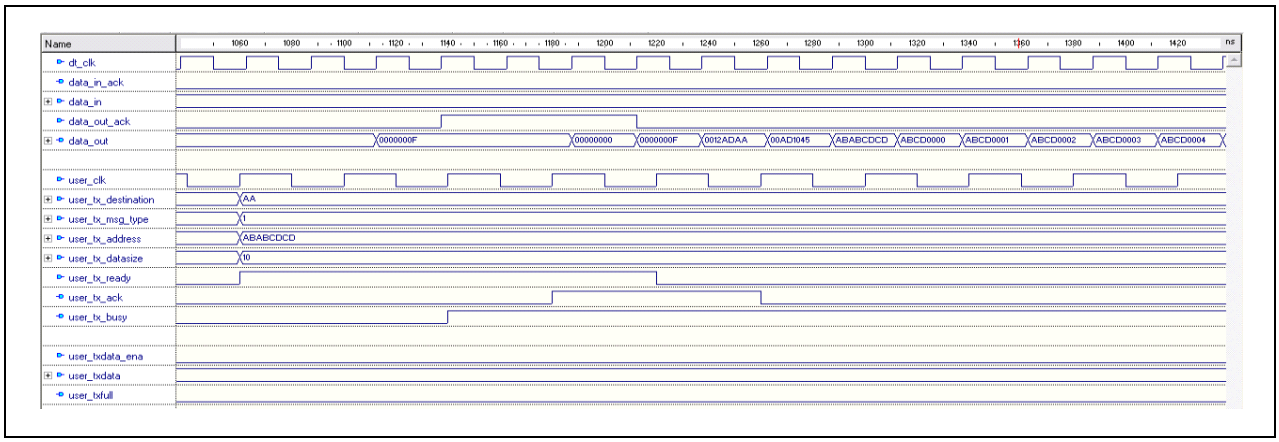

Figure 20: Master Node: tx data to destination

DIMEtalk packets entering the Master Node can be read from receiver port. The following waveforms illustrate a packet entering the Master Node and being read out of the node by the user interface. Figure 21 illustrates a five dataword DIMEtalk write packet with no response being received by the Master Node. The Master Node de-constructs the packet and places the packet source, type, size and the start address of the first dataword on the user_rx_source, user_rx_msg_type, user_rx_datasize and user_rx_address respectively. Once the Master Node successfully receives all datawords in the packet, the user_rx_ready flag is asserted on the rising edge of the user_clk to indicate that the data is ready to be read from the FIFO. When the user_rx_ready flag is asserted, the user must acknowledge the Master Node by asserting user_rx_ack. Following this, the user can then read the data from the

user_rxdata bus by asserting the user_rxdata_ena flag. Note that there is a one clock cycle delay between the user_rxdata_ena being asserted and the data being presented on the user_rxdata bus.

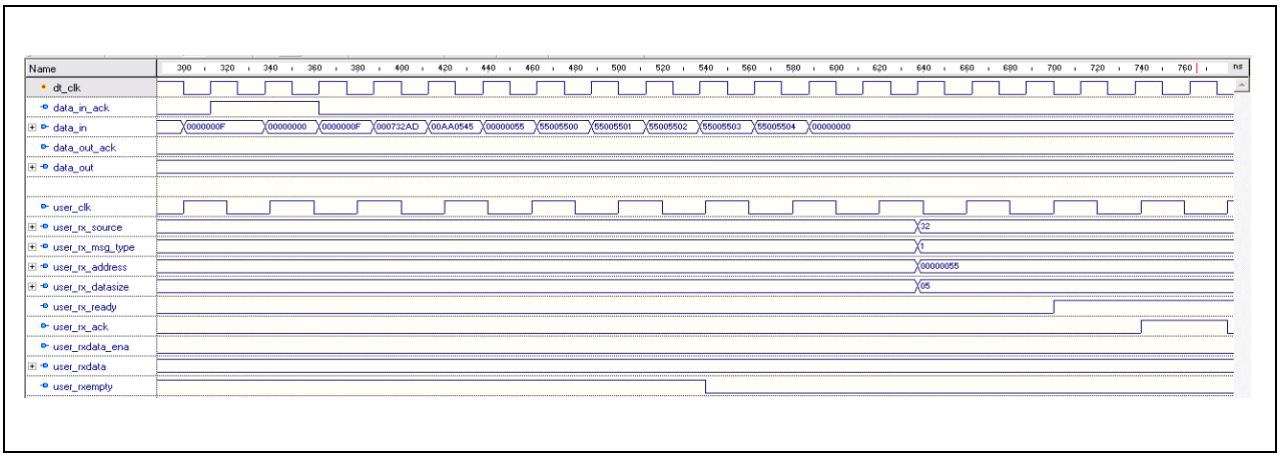

Figure 21: Master Node: write packet - no response (5 data words)

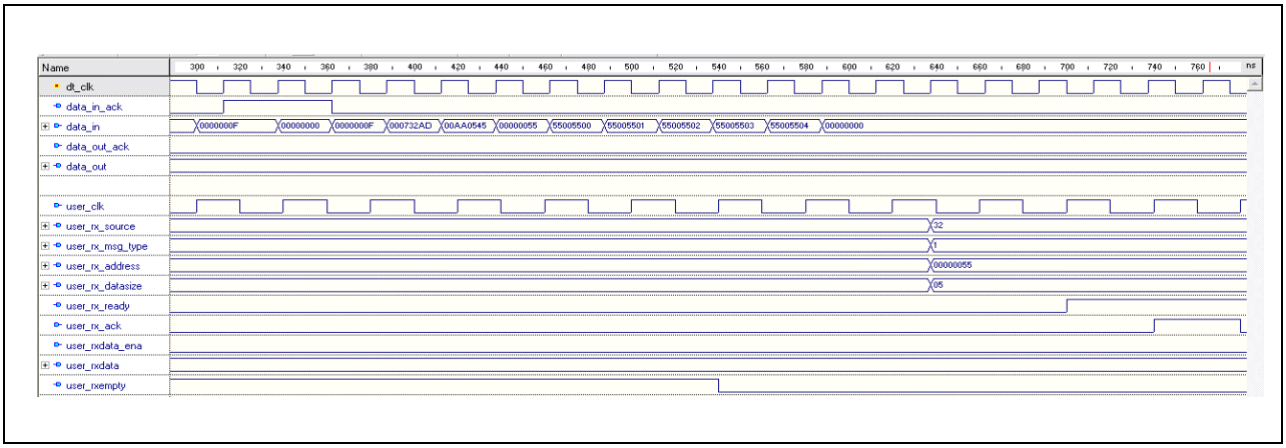

Figure 22: Master Node: rx data

1.3 ZBT Nodes

ZBT Memory Node 32-Bit

Functional Description

This node implements an interface to ZBT SRAM. It can support 32-bit data and different depths of ZBT device. It has two interfaces, DIMEtalk and user, which provide read and write access to ZBT. The node supports independent clock domains for the DIMEtalk and user interface - the user interface runs from the same clock as the ZBT. The user is given fixed deterministic read/write access to the ZBT. In order to provide this any pending DIMEtalk accesses are held off until the user interface is idle. The ZBT Interface component is shown in Figure 23.

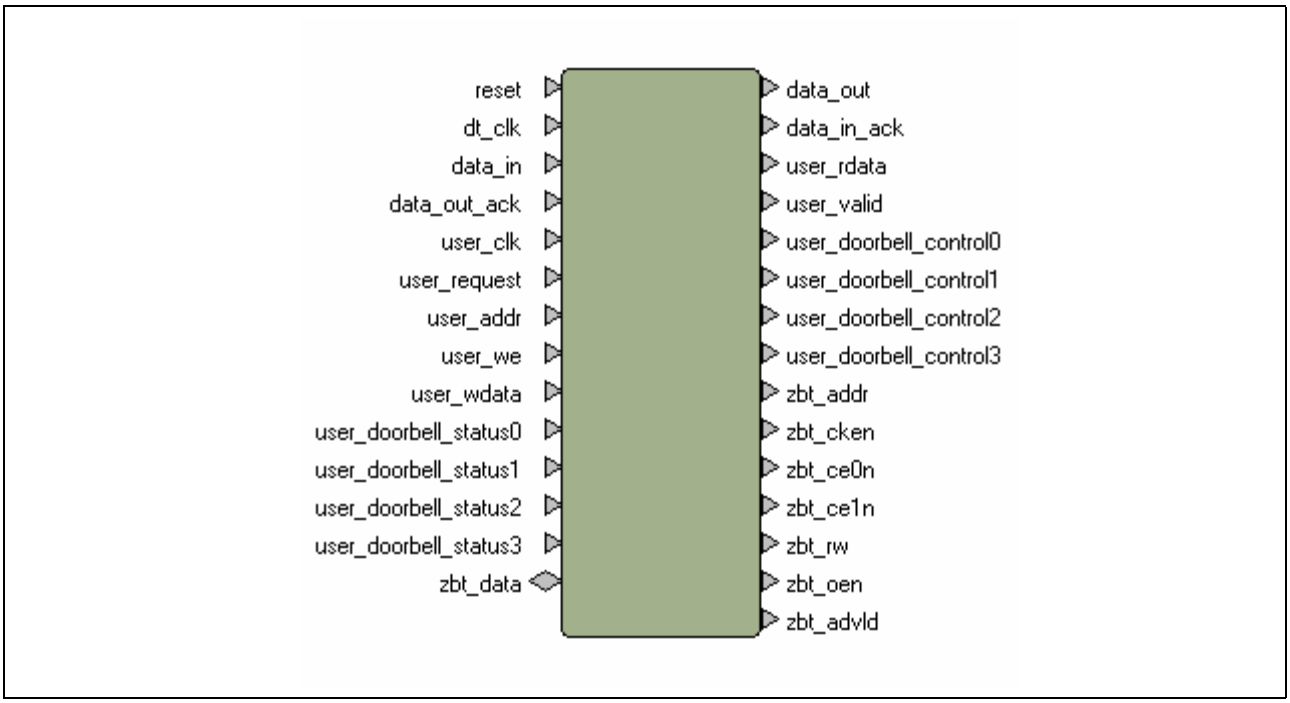

Figure 23: ZBT 32-Bit Interface Component

Additional Modules

This module must be coupled with a Clock Deskew component in order to provide a deskewed clock to the ZBT memory therefore ensuring that the ZBT memory and ZBT interface operate from a synchronous clock.

Extended Doorbell

There are four doorbell status inputs and four doorbell control outputs. The status inputs can be used to trap a change of state on an input whilst the control outputs can be used as a trigger for aspects of a user's design. For more information on packets see "Request Packet Type 10 (Doorbell class)".

Signals

The signal descriptions are shown in Table 31.

Group	Type	Signal Description	Width	Direction
reset	DIMEtalk reset	reset	1-bit	In
dt_clk	DIMEtalk network clock. Def. CLK1	dt_clk	1-bit	In

Table 31: ZBT 32-Bit Interface Signal Descriptions

Group	Type	Signal Description	Width	Direction
DIMEtalk		data_in	31 downto 0 bits	In
DIMEtalk		data_out	31 downto 0 bits	Out
DIMEtalk		data_in_ack	1-bit	Out
DIMEtalk		data_out_ack	1-bit	In
EmptyStatus	FIFO empty status in DT clock domain	dt_empty	1-bit	Out
UserInterface	User port clock, def CLK1	User_clk	1-bit	In
UserInterface	User request (validates user address)	User_request	1-bit	In
UserInterface	User address	User_addr	Mem_awidth	In
UserInterface	User write enable	User_we	1-bit	In
UserInterface	User write data	User_wdata	32-bits	In
UserInterface	User read data	User_rdata	32-bits	Out
UserInterface	User read data valid	User_valid	1-bit	Out
ZBTInterface	ZBT address	Zbt_addr	Mem_awidth	Out
ZBTInterface	ZBT data	Zbt_data	Mem_dwidth	Inout
ZBTInterface	ZBT clock enable	Zbt_cken	1-bit	Out
ZBTInterface	ZBT 0 chip enable	Zbt_ce0n	1-bit	Out
ZBTInterface	ZBT 1 chip enable	Zbt_ce1n	1-bit	Out
ZBTInterface	ZBT read/write	Zbt_rw	1-bit	Out
ZBTInterface	ZBT output enable	Zbt_oen	1-bit	Out
ZBTInterface	ZBT advance/load	Zbt_advld	1-bit	Out
Doorbells		user_doorbell_status0	1-bit	In
Doorbells		user_doorbell_status1	1-bit	In
Doorbells		user_doorbell_status2	1-bit	In
Doorbells		user_doorbell_status3	1-bit	In
Doorbells		user_doorbell_control0	1-bit	Out
Doorbells		user_doorbell_control1	1-bit	Out
Doorbells		user_doorbell_control2	1-bit	Out
Doorbells		user_doorbell_control3	1-bit	Out

Table 31: ZBT 32-Bit Interface Signal Descriptions

User Generics

The user generic data is detailed in Table 32.

Name	Type	Description
Device_asize	Integer	This is used to derive the address at which each chip select is active. For example for two 512k x 32 bit devices used to form a 1M x 32 bit memory space this equals 19.
Mem_awidth	Integer	Width of address bus. This equals 20 in the above example.

Table 32: ZBT 32-Bit Interface User Generics

Support Files

The support files are shown in Table 33.

Name	Device Usage
common\source\pkg_dimetalk_global.vhd	All
Common\source\bretime.vhd	All
Common\source\retime.vhd	All
Common\source\dtnode_slave_control.vhd	All
Common\source\resync_rise.vhd	All
ZBT\source\zbt_if.vhd	All

Table 33: ZBT 32-Bit Interface Support Files

Component Definition

```
component zbt_if_32
        generic (
        node_ID      : std_logic_vector(NODEID_SZ-1 downto 0);
        device_asize : integer range 0 to MAX_DASIZE;
        mem_awidth   : integer range 0 to MAX_MAWIDTH;
        mem_dwidth   : integer range MIN_MDWIDTH to MAX_MDWIDTH);
        port (
        reset               : in    std_logic;
        dt_clk              : in    std_logic;
        data_in             : in    std_logic_vector(DT_DATA_SZ-1 downto 0);
        data_out            : out   std_logic_vector(DT_DATA_SZ-1 downto 0);
        data_in_ack         : out   std_logic;
        data_out_ack        : in    std_logic;
        user_clk            : in    std_logic;
        user_request        : in    std_logic;
        user_addr           : in    std_logic_vector(mem_awidth-1 downto 0);
        user_we             : in    std_logic;
        user_wdata          : in    std_logic_vector(mem_dwidth-1 downto 0);
        user_rdata          : out   std_logic_vector(mem_dwidth-1 downto 0);
        user_valid          : out   std_logic;
        user_doorbell_status0  : in    std_logic;
        user_doorbell_status1  : in    std_logic;
        user_doorbell_status2  : in    std_logic;
        user_doorbell_status3  : in    std_logic;
        user_doorbell_control0 : out   std_logic;
```

```
user_doorbell_control1 : out   std_logic;
user_doorbell_control2 : out   std_logic;
user_doorbell_control3 : out   std_logic;
zbt_addr               : out   std_logic_vector(mem_awidth-1 downto 0);
zbt_data               : inout std_logic_vector(mem_dwidth-1 downto 0);
zbt_cken               : out   std_logic;
zbt_ce0n               : out   std_logic;
zbt_ce1n               : out   std_logic;
zbt_rw                 : out   std_logic;
zbt_oen                : out   std_logic;
zbt_advld              : out   std_logic);
end component;
```

Waveforms

To perform a write access, the user asserts user_request and user_we while presenting the user_addr and user_wdata. In Figure 24 five words are written.

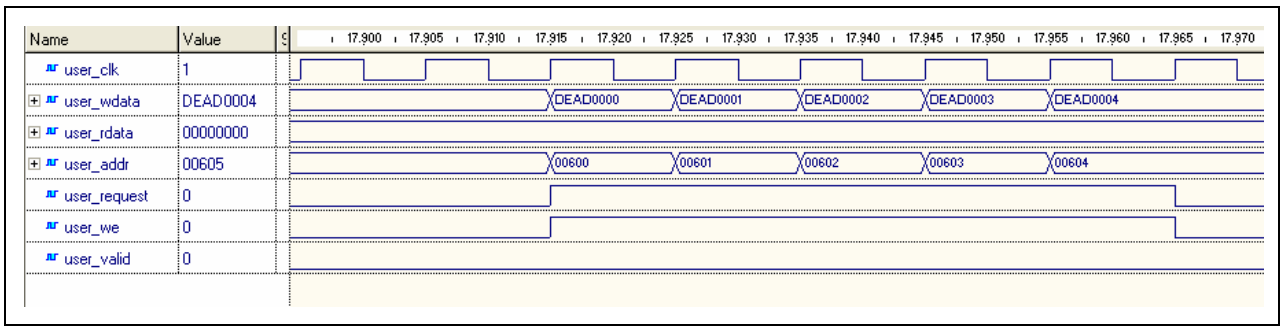

Figure 24: ZBT Write Access

To perform a ZBT read access, shown in Figure 25, the user asserts user_request while presenting user_addr, user_we should be low. A read access is performed and the read data is returned qualified by the assertion of user_valid.

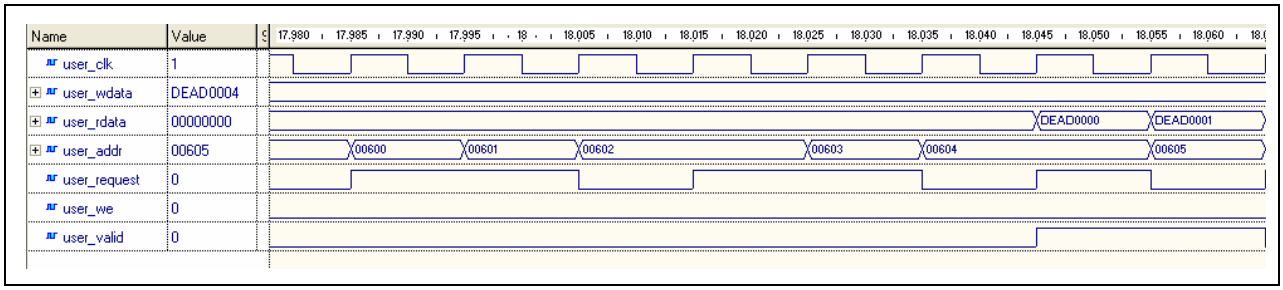

Figure 25: ZBT Read Access

ZBT Memory Node 64-Bit

Functional Description

This node implements an interface to ZBT SRAM. It can support 64-bit data and different depths of ZBT device. It has two interfaces, DIMEtalk and user, which provide read and write access to ZBT. The node supports independent clock domains for the DIMEtalk and user interfaces - the user interface runs from the same clock as the ZBT. The user is given fixed deterministic read/write access to the ZBT. In order to provide this any pending DIMEtalk accesses are held off until the user interface is idle. The ZBT Interface component is shown in Figure 26.

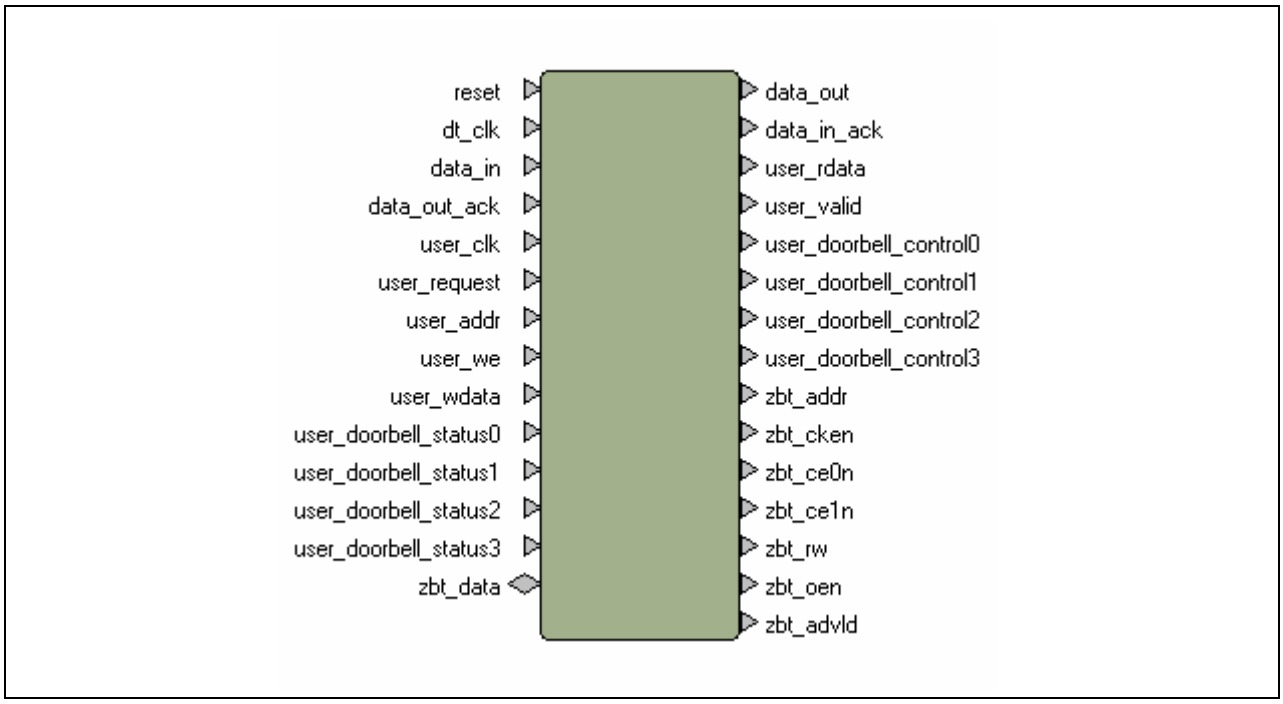

Figure 26: ZBT 64-Bit Interface Component

Additional Modules

This module must be coupled with a Clock Deskew component in order to provide a deskewed clock to the ZBT memory therefore ensuring that the ZBT memory and ZBT Interface operate from a synchronous clock.

Extended Doorbell

There are four doorbell status inputs and four doorbell control outputs. The status inputs can be used to trap a change of state on an input whilst the control outputs can be used as a trigger for aspects of a user's design. For more information on packets see "Request Packet Type 10 (Doorbell class)".

Signals

The signal descriptions are shown in Table 34.

Group	Type	Signal Description	Width	Direction
reset	DIMEtalk reset	reset	1-bit	In
dt_clk	DIMEtalk network clock. Def. CLK1	dt_clk	1-bit	In
DIMEtalk		data_in	31 downto 0 bits	In

Table 34: ZBT 64-Bit Interface Signal Descriptions

Group	Type	Signal Description	Width	Direction
DIMEtalk		data_out	31 downto 0 bits	Out
DIMEtalk		data_in_ack	1-bit	Out
DIMEtalk		data_out_ack	1-bit	In
EmptyStatus	FIFO empty status in DT clock domain	dt_empty	1-bit	Out
UserInterface	User port clock, def CLK1	User_clk	l-bit	In
UserInterface	User request (validates user address)	User_request	1-bit	In
UserInterface	User address	User_addr	Mem_awidth	In
UserInterface	User write enable	User_we	1-bit	In
UserInterface	User write data	User_wdata	64-bits	In
UserInterface	User read data	User_rdata	64-bits	Out
UserInterface	User read data valid	User_valid	1-bit	Out
ZBTInterface	ZBT address	Zbt_addr	Mem_awidth	Out
ZBTInterface	ZBT data	Zbt_data	Mem_dwidth	Inout
ZBTInterface	ZBT clock enable	Zbt_cken	1-bit	Out
ZBTInterface	ZBT 0 chip enable	Zbt_ce0n	1-bit	Out
ZBTInterface	ZBT 1 chip enable	Zbt_ce1n	1-bit	Out
ZBTInterface	ZBT read/write	Zbt_rw	1-bit	Out
ZBTInterface	ZBT output enable	Zbt_oen	1-bit	Out
ZBTInterface	ZBT advance/load	Zbt_advld	1-bit	Out
Doorbells		user_doorbell_status0	1-bit	In
Doorbells		user_doorbell_status1	1-bit	In
Doorbells		user_doorbell_status2	1-bit	In
Doorbells		user_doorbell_status3	1-bit	In
Doorbells		user_doorbell_control0	1-bit	Out
Doorbells		user_doorbell_control1	1-bit	Out
Doorbells		user_doorbell_control2	1-bit	Out
Doorbells		user_doorbell_control3	1-bit	Out

Table 34: ZBT 64-Bit Interface Signal Descriptions

User Generics

The user generic data is detailed in Table 35.

Name	Type	Description
Device_asize	Integer	For 64-bit SRAM banks this is not used.
Mem_awidth	Integer	Width of address bus. This equals 19 in the above example.

Table 35: ZBT 64-Bit Interface User Generics

Support Files

The support files are shown in Table 36.

Name	Device Usage
Common\source\pkg_dimetalk_global.vhd	All
Common\source\bretime.vhd	All
Common\source\retime.vhd	All
Common\source\dtnode_slave_control.vhd	All
Common\source\resync_rise.vhd	All
ZBT\source\zbt_if_64.vhd	All

Table 36: ZBT 64-Bit Interface Support Files

Component Definition

```
component zbt_if_64
        generic (
        node_ID      : std_logic_vector(NODEID_SZ-1 downto 0);
        device_asize : integer range 0 to MAX_DASIZE;
        mem_awidth   : integer range 0 to MAX_MAWIDTH;
        mem_dwidth   : integer range MIN_MDWIDTH to MAX_MDWIDTH);
port (
        reset              : in    std_logic;
        dt_clk             : in    std_logic;
        data_in            : in    std_logic_vector(DT_DATA_SZ-1 downto 0);
        data_out           : out   std_logic_vector(DT_DATA_SZ-1 downto 0);
        data_in_ack        : out   std_logic;
        data_out_ack       : in    std_logic;
        user_clk           : in    std_logic;
        user_request       : in    std_logic;
        user_addr          : in    std_logic_vector(mem_awidth-1 downto 0);
        user_we            : in    std_logic;
        user_wdata         : in    std_logic_vector(mem_dwidth-1 downto 0);
        user_rdata         : out   std_logic_vector(mem_dwidth-1 downto 0);
        user_valid         : out   std_logic;
        user_doorbell_status0  : in    std_logic;
        user_doorbell_status1  : in    std_logic;
        user_doorbell_status2  : in    std_logic;
        user_doorbell_status3  : in    std_logic;
        user_doorbell_control0 : out   std_logic;
```

```
        user_doorbell_control1 : out   std_logic;
        user_doorbell_control2 : out   std_logic;
        user_doorbell_control3 : out   std_logic;
        zbt_addr               : out   std_logic_vector(mem_awidth-1 downto 0);
        zbt_data               : inout std_logic_vector(mem_dwidth-1 downto 0);
        zbt_cken               : out   std_logic;
        zbt_ce0n               : out   std_logic;
        zbt_ce1n               : out   std_logic;
        zbt_rw                 : out   std_logic;
        zbt_oen                : out   std_logic;
        zbt_advld              : out   std_logic);
end component;
```

Waveforms

ZBT 64-bit accesses are identical to ZBT 32-bit accesses except that the data bus is double the width. Refer to "Waveforms" for more details.

Clock Deskew Component

Functional Description

This component implements a clock deskew network and is associated with the ZBT interface. It provides up to eight phase locked clock outputs. The deskew feedback loop is external and takes the clk_fb and clk_fb_o signals from this module to external pins. This module exists as a separate entity as the number of ZBT clock outputs varies between modules. This is a reusable component and designed to cover all ZBT usage cases. The Clock Deskew signals are shown in Figure 27 and described in Table 37.

Note that the *Clock Deskew Component* should not be confused with the *BenBLUE-III Clock Deskew Module for use with ZBT SRAM*. These are two separate components and the *Clock Deskew Component* is not appropriate for use with DIMEtalk networks which contain the BenBLUE-III module.

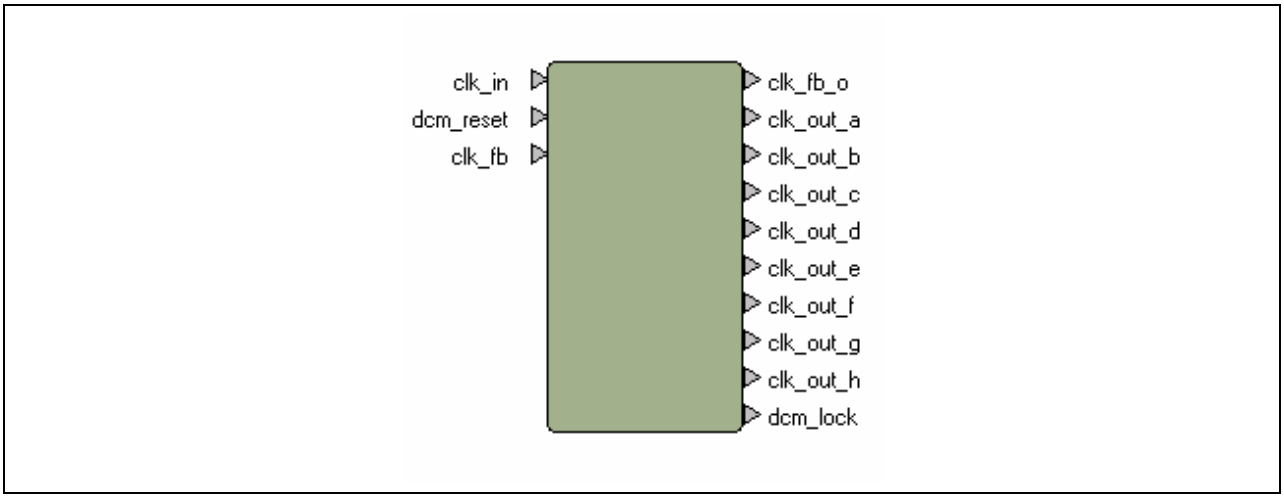

Figure 27: Clock Deskew Module Component

Location Constraint

Note that the DCM in this module needs to be placed physically close to the DCM that deskews the clock for the logic that interfaces to the ZBT. This is done to minimize the skew between the two clocks and is performed via a LOC constraint. The following example illustrates the setup for the BenNUEY-PCI card where Clocks_0 is the instantiation of the system Clock Deskew module and ClockDeskew_0 is the instantiation of the external ZBT memory clock deskew.

```
##############################################

# Placement constraints

##############################################

# DCMs

INST "ClockDeskew_0/dcm1" LOC=DCM_X4Y1;

INST "Clocks_0/clkdll1" LOC=DCM_X4Y0;

INST "Clocks_0/clkdll2" LOC=DCM_X2Y0;

##############################################
```

Signals

The signal descriptions are provided in Table 37.

Group	Type	Signal Description	Width	Direction
reset	DIMEtalk reset, default reset	Dcm_reset	1-bit	In
Clk_in	DIMEtalk network clock, default CLK1	Clk_in	1-bit	In
Clk_fb_in		Clk_fb	l-bit	In
Clk_fb_out		Clk_fb_out	1-bit	Out
Clka_o		Clk_out_a	1-bit	Out
Clkb_o		Clk_out_b	1-bit	Out
Clkc_o		Clk_out_c	1-bit	Out
Clkd_o		Clk_out_d	1-bit	Out
Clke_o		Clk_out_e	1-bit	Out
Clkf_o		Clk_out_f	1-bit	Out
Clkg_o		Clk_out_g	1-bit	Out
Clkh_o		Clk_out_h	1-bit	Out
lock		Dcm_lock	1-bit	Out

Table 37: Clock Deskew Signals

User Generics

The user generics are shown in Table 38.

Name	Type	Description
Num_clk_ops	Integer (1 to 8)	Number of clock outputs required, filled fromclk_out_a to clk_out_h)
BenDVI_2nd	Boolean	Set to true if it's the second Deskew on a BenDVI. Adds DCM placement constraint.

Table 38: Clock Deskew User Generics

Support Files

The support file names and devices used are listed in Table 39.

Name	Device Usage
Clock_deskew\source\clock_deskew.vhd	All

Table 39: Clock Deskew Support Files

Component Definition

```
component ClockDeskew
        generic (
        num_clk_ops : integer range 1 to 8);
        port (
        clk_in    : in  std_logic;
        dcm_reset : in  std_logic;
        clk_fb    : in  std_logic;
        clk_fb_o  : out std_logic;
        clk_out_a : out std_logic;
        clk_out_b : out std_logic;
        clk_out_c : out std_logic;
        clk_out_d : out std_logic;
        clk_out_e : out std_logic;
        clk_out_f : out std_logic;
        clk_out_g : out std_logic;
        clk_out_h : out std_logic;
        dcm_lock  : out std_logic);
        end component;
```

BenBLUE-III Clock Deskew Module for Use with ZBT SRAM

Functional Description

The BenBLUE-III clock deskew component is designed solely for use with the ZBT RAM on the BenBLUE-III Primary and Secondary FPGAs. The BenBLUE-III has four banks of 32-bit wide memory per FPGA. All banks are clocked independently. However, there is only one feedback path. Therefore to enable the correct clocking one BenBLUE-III clock deskew component must be placed down. When looking at the Secondary FPGA, note the clock deskew for BenBLUE modules component on the Primary FPGA. This is used to correctly forward the clocks to the Secondary FPGA. It also forwards the dcm_lock signal which is used by the Secondary FPGA as a reset input. By default only CLKA is used in designs. If other clocks are required, then additional deskew components must be added to the Primary FPGA. The BenBLUE-III clock deskew component is shown in Figure 28.

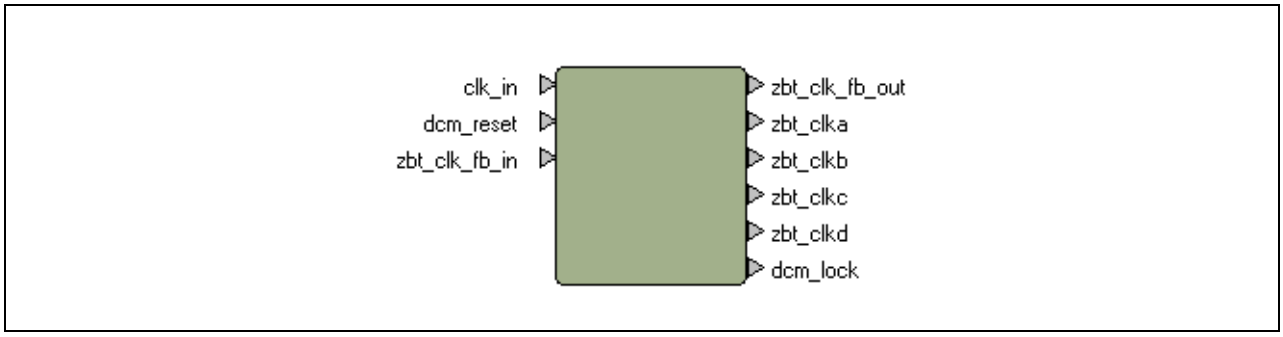

Figure 28: BenBLUE-III Clock Deskew Module

Signals

The signal descriptions are provided in Table 40.

Group	Type	Signal Description	Width	Direction
clk_in	Clock Connection	clk_in : in STD_LOGIC	1-bit	In
dcm_reset	Reset Connection	dcm_reset : in STD_LOGIC;	1-bit	In
clk_fb	User Connection	zbt_clk_fb_in : in STD_LOGIC;	1-bit	Out
		zbt_clk_fb_out : out STD_LOGIC;	1-bit	
		zbt_clka : out STD_LOGIC;	1-bit	
		zbt_clkb : out STD_LOGIC;	1-bit	
		zbt_clkc : out STD_LOGIC;	1-bit	
		zbt_clkd : out STD_LOGIC;	1-bit	
lock	User Connection	dcm_lock : out STD_LOGIC;	1-bit	Out

Table 40: BenBLUE-III Clock Deskew Signals

Support Files

The support file names and devices used are listed in Table 41.

Name	Device Usage
DIMEtalk\library\ZBT Nodes\bb3_clockdeskew.support\bb3_clockdeskew.vhd	All devices

Table 41: BenBLUE-III Clock Deskew Support Files

Component Definition

deskewclocks_0

```
entity deskewclocks is
port(
        clk          : in  std_logic;
        dcm_rst      : in  std_logic;
        -- Secondary FPGA clock & Deskew
        clk_dr       : out std_logic;
        clk_dr_fb_in  : in  std_logic;
        clk_dr_fb_out : out std_logic;
        dcm_lock     : out std_logic
);
```

Grounding Extra BenBLUE-III Control Pins

Functional Description

This is a utility component which allows ZBT components to function on BenBLUE-III FPGAs. One instance of this component should be placed in the Primary and Secondary FPGAs for each ZBT component. The component is shown in Figure 29.

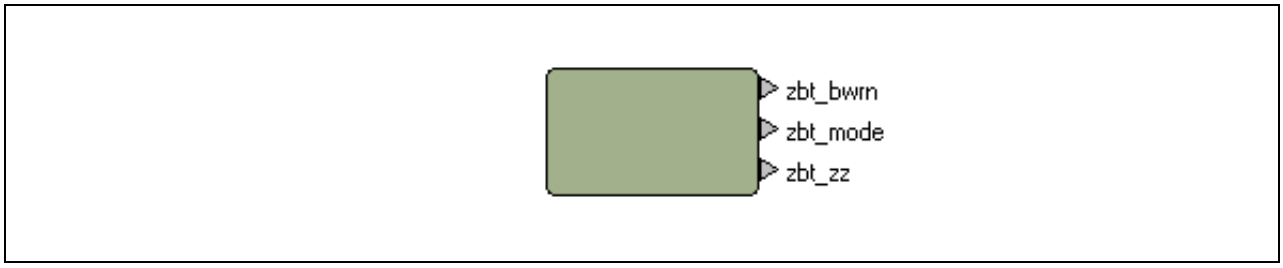

Figure 29: Grounding Extra BenBLUE-III Control Pins Component

Signals

The signal descriptions are provided in Table 42.

Group	Type	Signal Description	Width	Direction
User Signals	User Connection	zbt_bwrn : out STD_LOGIC_VECTOR(3 downto 0);	4-bits	Out
		zbt_mode : out STD_LOGIC;	1-bit	Out
		zbt_zz : out STD_LOGIC;	1-bit	Out

Table 42: BenBLUE-III Clock Deskew Signals

Support Files

The support file names and devices used are listed in Table 43.

Name	Device Usage
components\ZBT Nodes\bb3_zbt.support\all\bb3_zbt.vhd	All devices

Table 43: BenBLUE-III Clock Deskew Support Files

Component Definition

```
port(
        zbt_bwrn : out std_logic_vector(3 downto 0);
        zbt_mode : out std_logic;
        zbt_zz   : out std_logic
);
```

ZBT SRAM Testbench for Simulation

Functional Description

This component allows designs using ZBT to be exercised in a simulator. It describes the physical device so the simulator can model the interaction between this and the VHDL. The component uses Micron Technology Incorporated ZBT SRAM MT55L256L32P (256K x 32, Mode: Pipelined) VHDL model. See "Simulation Support" for more information. The component is shown in Figure 30.

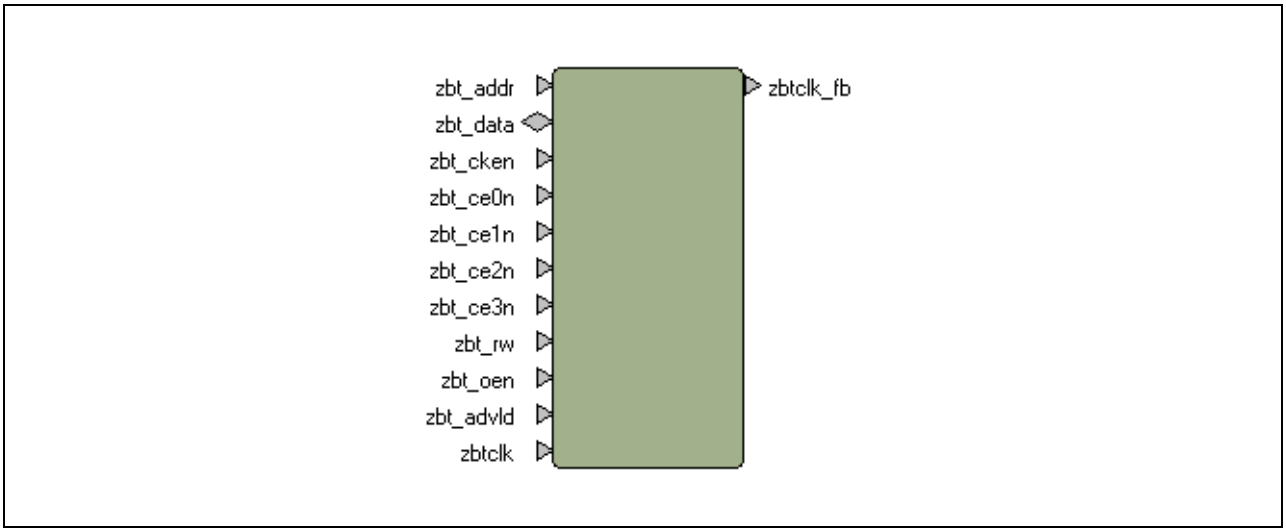

Figure 30: ZBT SRAM Testbench for Simulation Component

Signals

The signal descriptions are provided in Table 44.

Group	Type	Signal Description	Width	Direction
User Signals	User Connection	zbt_addr : in STD_LOGIC_VECTOR(31 downto 0);	32-bits	In
		zbt_data : inout STD_LOGIC_VECTOR(31 downto 0);	32-bits	Inout
		zbt_cken : in STD_LOGIC;	1-bit	In
		zbt_ce0n : in STD_LOGIC;	1-bit	In
		zbt_ce1n : in STD_LOGIC;	1-bit	In
		zbt_ce2n : in STD_LOGIC;	1-bit	In
		zbt_ce3n : in STD_LOGIC;	1-bit	In
		zbt_rw : in STD_LOGIC;	1-bit	In
		zbt_oen : in STD_LOGIC;	1-bit	In
		zbt_advld : in STD_LOGIC;	1-bit	In
clocks	User Connection	zbtclk_fb : out STD_LOGIC;	1-bit	Out
		zbtclk : in STD_LOGIC;	1-bit	In

Table 44: ZBT SRAM Testbench for Simulation Signals

Support Files

The support file names and devices used are listed in Table 45.

Name	Device Usage
components\ZBT Nodes\tb_zbt_if_32.support\All\tb_zbt_if_32.vhd	All devices
components \ZBT Nodes\tb_zbt_if_32.support\all\mt55l256l32p.vhd	All devices

Table 45: ZBT SRAM Testbench for Simulation Support Files

Component Definition

```
port (
        zbt_addr : in STD_LOGIC_VECTOR(20 downto 0);
        zbt_data : inout STD_LOGIC_VECTOR(31 downto 0);
        zbt_cken : in STD_LOGIC;
        zbt_ce0n : in STD_LOGIC;
        zbt_ce1n : in STD_LOGIC;
        zbt_ce2n : in STD_LOGIC;
        zbt_ce3n : in STD_LOGIC;
        zbt_rw : in STD_LOGIC;
        zbt_oen : in STD_LOGIC;
        zbt_advld : in STD_LOGIC;
        zbtclk_fb : out STD_LOGIC;
        zbtclk : in STD_LOGIC
);
```

1.4 Bridges

4-Bit Bidirectional Physical Bridge

Functional Description

This component transfers DIMEtalk packet data from one physical device to another using a 10-bit wide bus. This is used as 4-bits data in each direction plus one bit of flow control. This Bridge is synchronous with the overall DIMEtalk clock and is therefore limited to operating at 1/8 speed of internal 32-bit wide links. The Bridge signals are shown in Figure 31 and described in Table 46.

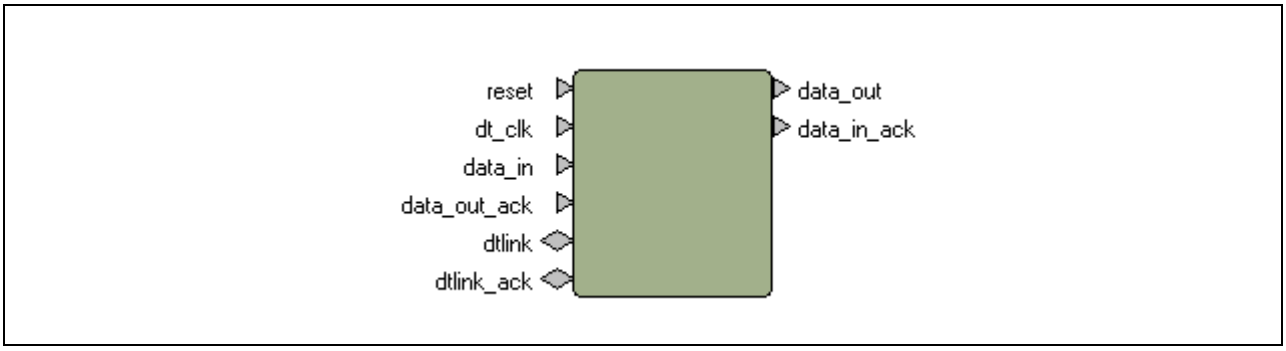

Figure 31: 4-bit Bridge Component

Signals

The signal descriptions are provided in Table 46.

Group	Type	Signal Description	Width	Direction
reset	DIMEtalk reset	reset	1-bit	In
dt_clk	DIMEtalk network clock. Def. CLK1	dt_clk	1-bit	In
DIMEtalk		data_in	31 downto 0 bits	In
DIMEtalk		data_out	31 downto 0 bits	Out
DIMEtalk		data_in_ack	1-bit	Out
DIMEtalk		data_out_ack	1-bit	In
dtlink		dtlink	7 downto 0 bits	Inout
dtlink		dtlink_ack	1 downto 0 bits	Inout

Table 46: 4-Bit Bridge Signal Descriptions

Support Files

The support file names and devices used are listed in Table 47.

Name	Device Usage
DIMEtalk\library\Bridges\4bitbridge.support\All\bridge_4.vhd	All
DIMEtalk\library\Bridges\4bitbridge.support\All\convert32to4.vhd	All
DIMEtalk\library\Bridges\4bitbridge.support\All\convert4to32.vhd	All

Table 47: 4-Bit Bridge Support Files

Name	Device Usage
DIMEtalk\library\Bridges\4bitbridge.support\All\bridge4.vhd	All

Table 47: 4-Bit Bridge Support Files

Component Definition

```
component bridge
        generic(
        plug        : boolean := TRUE;
        linkwidth    : integer := 32
);
port(
        reset       : in  std_logic;
        -- DimeTalk port
        dt_clk      : in  std_logic;
        data_in     : in  std_logic_vector(31 downto 0);
        data_out    : out std_logic_vector(31 downto 0);
        data_in_ack  : out std_logic;
        data_out_ack  : in  std_logic;
        -- Bridge Link port (duplex)
        dtlink      : inout std_logic_vector(linkwidth*2-1 downto 0);
        dtlink_ack   : inout std_logic_vector(1 downto 0)
);
End component;
```

8-Bit Bidirectional Physical Bridge

Functional Description

This component transfers DIMEtalk packet data from one physical device to another using an 18-bit wide bus. This is used as 8-bits data in each direction plus one bit of flow control. The bridge is synchronous with the overall DIMEtalk clock and is therefore limited to operating at ¼ speed of internal 32-bit wide links. The Bridge signals are shown in Figure 32 and described in Table 48.

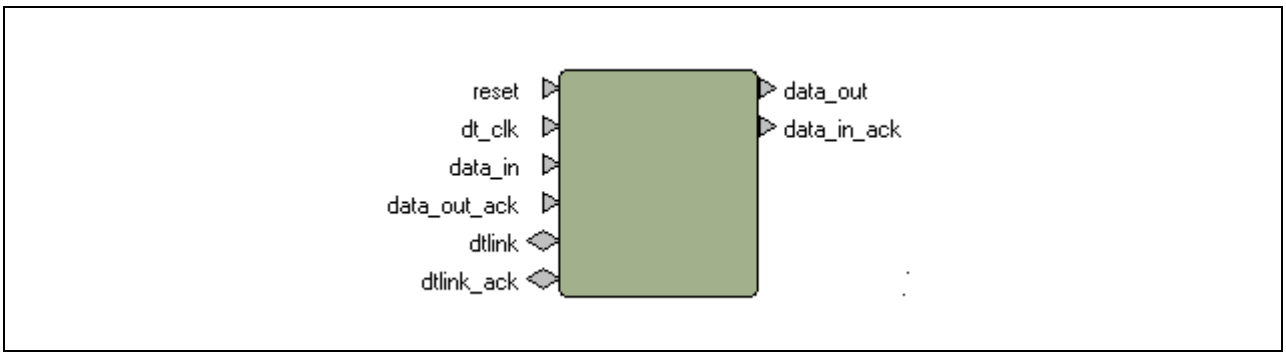

Figure 32: 8-Bit Bridge Component

Signals

The signal descriptions are provided in Table 48.

Group	Type	Signal Description	Width	Direction
reset	DIMEtalk reset	reset	1-bit	In
dt_clk	DIMEtalk network clock. Def. CLK1	dt_clk	1-bit	In
DIMEtalk		data_in	31 downto 0 bits	In
DIMEtalk		data_out	31 downto 0 bits	Out
DIMEtalk		data_in_ack	1-bit	Out
DIMEtalk		data_out_ack	1-bit	In
dtlink		dtlink	15 downto 0 bits	Inout
dtlink		dtlink_ack	1 downto 0 bits	Inout

Table 48: 8-Bit Bridge Signal Descriptions

Support Files

The support file names and devices used are listed in Table 49.

Name	Device Usage
DIMEtalk\library\Bridges\8bitbridge.support\All\bridge_8.vhd	All
DIMEtalk\library\Bridges\8bitbridge.support\All\convert32to8.vhd	All
DIMEtalk\library\Bridges\8bitbridge.support\All\convert8to32.vhd	All
DIMEtalk\library\Bridges\8bitbridge.support\All\bridge8.vhd	All

Table 49: 8-Bit Bridge Support Files

Component Definition

```
component bridge
        generic(
        plug          : boolean := TRUE;
        linkwidth     : integer := 32
);
port(
        reset         : in  std_logic;
        -- DimeTalk port
        dt_clk        : in  std_logic;
        data_in       : in  std_logic_vector(31 downto 0);
        data_out      : out std_logic_vector(31 downto 0);
        data_in_ack   : out std_logic;
        data_out_ack  : in  std_logic;
        -- Bridge Link port (duplex)
```

```
        dtlink      : inout std_logic_vector(linkwidth*2-1 downto 0);
        dtlink_ack  : inout std_logic_vector(1 downto 0)
);
End component;
```

16-Bit Bidirectional Physical Bridge

Functional Description

This component transfers DIMEtalk packet data from one physical device to another using a 32-bit wide bus. This is used as 16-bits data in each direction plus one bit of flow control. The Bridge is synchronous with the overall DIMEtalk clock and is therefore limited to operating at ½ the speed of internal 32-bit wide links. The Bridge signals are shown in Figure 33 and described in Table 50.

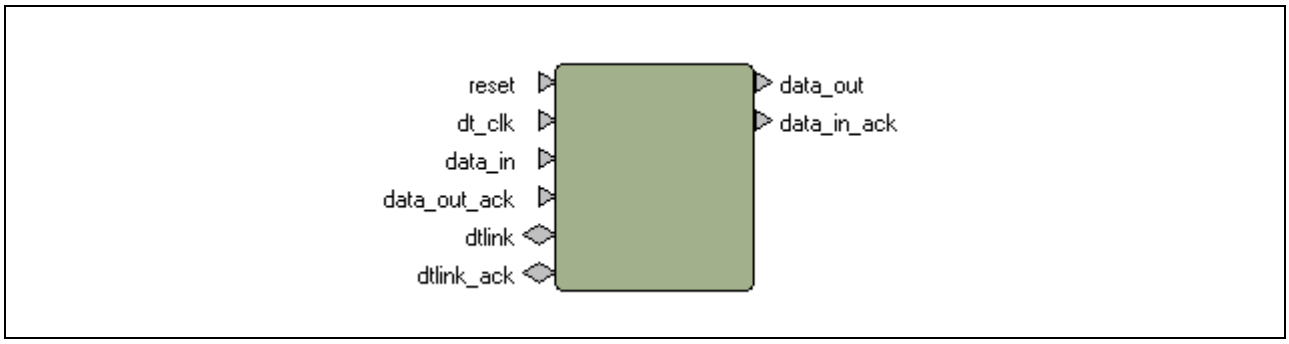

Figure 33: 16-Bit Bridge Component

Signals

The signal descriptions are provided in Table 50.

Group	Type	Signal Description	Width	Direction
reset	DIMEtalk reset	reset	1-bit	In
dt_clk	DIMEtalk network clock. Def. CLK1	dt_clk	1-bit	In
DIMEtalk		data_in	31 downto 0 bits	In
DIMEtalk		data_out	31 downto 0 bits	Out
DIMEtalk		data_in_ack	1-bit	Out
DIMEtalk		data_out_ack	1-bit	In
dtlink		dtlink	31 downto 0 bits	Inout
dtlink		dtlink_ack	1 downto 0 bits	Inout

Table 50: 16-Bit Bridge Signal Descriptions

Support Files

The support file names and devices used are listed in Table 51.

Name	Device Usage
DIMEtalk\library\Bridges\16bitbridge.support\All\bridge_16.vhd	All
DIMEtalk\library\Bridges\16bitbridge.support\All\convert16to32.vhd	All
DIMEtalk\library\Bridges\16bitbridge.support\All\convert32to16.vhd	All
DIMEtalk\library\Bridges\16bitbridge.support\All\bridge16.vhd	All

Table 51: 16-Bit Bridge Support Files

Component Definition

```
component bridge
        generic(
        plug          : boolean := TRUE;
        linkwidth      : integer := 32
);
port(
        reset         : in  std_logic;
        -- DimeTalk port
        dt_clk        : in  std_logic;
        data_in       : in  std_logic_vector(31 downto 0);
        data_out      : out std_logic_vector(31 downto 0);
        data_in_ack   : out std_logic;
        data_out_ack  : in  std_logic;
        -- Bridge Link port (duplex)
        dtlink        : inout std_logic_vector(linkwidth*2-1 downto 0);
        dtlink_ack    : inout std_logic_vector(1 downto 0)
);
End component;
```

32-Bit Bidirectional Physical Bridge

Functional Description

This component transfers DIMEtalk packet data from one physical device to another using a 66-bit wide bus. This is used as 32-bits data in each direction plus one bit of flow control. This Bridge is synchronous with the overall DIMEtalk clock. The Bridge signals are shown in Figure 34 and described in Table 52.

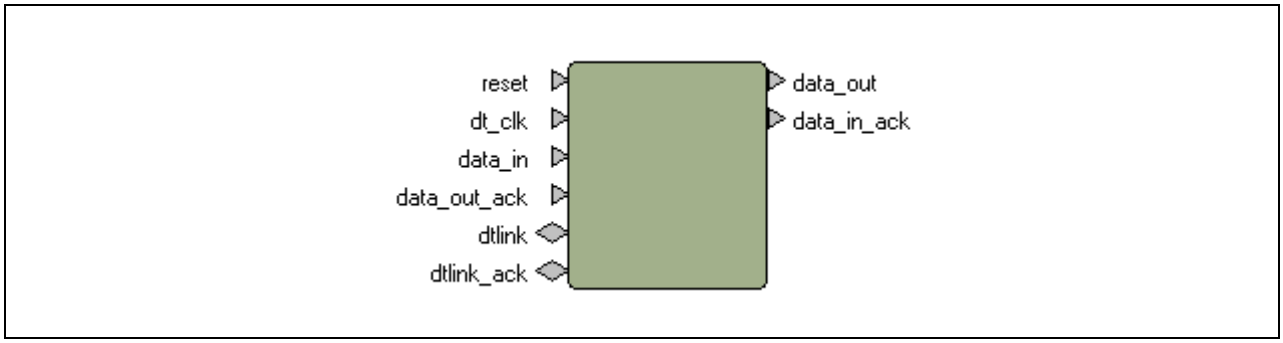

Figure 34: 32-Bit Bridge Component

Signals

The signal descriptions are provided in Table 52.

Group	Type	Signal Description	Width	Direction
reset	DIMEtalk reset	reset	1-bit	In
dt_clk	DIMEtalk network clock. Def. CLK1	dt_clk	1-bit	In
DIMEtalk		data_in	31 downto 0 bits	In
DIMEtalk		data_out	31 downto 0 bits	Out
DIMEtalk		data_in_ack	1-bit	Out
DIMEtalk		data_out_ack	1-bit	In
dtlink		dtlink	63 downto 0 bits	Inout
dtlink		dtlink_ack	1 downto 0 bits	Inout

Table 52: 32-Bit Bridge Signal Descriptions

Support Files

The support file names and devices used are listed in Table 53.

Name	Device Usage
DIMEtalk\library\Bridges\32bitbridge.support\All\bridge_32.vhd	All
DIMEtalk\library\Bridges\32bitbridge.support\All\bridge32.vhd	All

Table 53: 32-Bit Bridge Support Files

Component Definition

```
component bridge
        generic(
        plug          : boolean := TRUE;
        linkwidth     : integer := 32
);
port(
        reset         : in  std_logic;
        -- DimeTalk port
        dt_clk        : in  std_logic;
        data_in       : in  std_logic_vector(31 downto 0);
        data_out      : out std_logic_vector(31 downto 0);
        data_in_ack   : out std_logic;
        data_out_ack  : in  std_logic;
        -- Bridge Link port (duplex)
        dtlink        : inout std_logic_vector(linkwidth*2-1 downto 0);
        dtlink_ack    : inout std_logic_vector(1 downto 0)
);
End component;
```

4-Bit Bidirectional Asynchronous Bridge

Functional Description

This component transfers DIMEtalk packet data from one physical device to another using a 10-bit wide bus. This is used as 4-bits data in each direction plus one bit of flow control. This Bridge is asynchronous with the overall DIMEtalk clock. The speed of the link is governed by the br_clk input and can be faster or slower than the overall network DIMEtalk clock, therefore the Bridge can be used to operate over slower links without slowing the overall network, or faster links to make better use of limited width connections. The Bridge signals are shown in Figure 35 and described in Table 54.

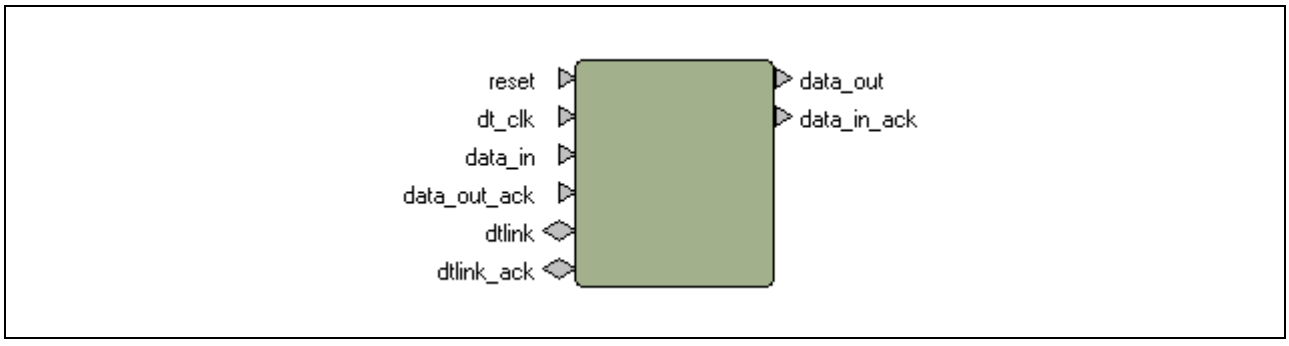

Figure 35: 4-Bit Bridge Component

Signals

The signal descriptions are provided in Table 54.

Group	Type	Signal Description	Width	Direction
reset	DIMEtalk reset	reset	1-bit	In
dt_clk	DIMEtalk network clock. Def. CLK1	dt_clk	1-bit	In
DIMEtalk		data_in	32-bits	In
DIMEtalk		data_out	32-bits	Out
DIMEtalk		data_in_ack	1-bit	Out
DIMEtalk		data_out_ack	1-bit	In
br_clk	Def. CLK3	br_clk	1-bit	In
Link	External connection	brlink	8-bits	Inout
Link	External connection	brlink_ack	2-bits	Inout

Table 54: 4-Bit Bridge Signal Descriptions

Support Files

The support file names and devices used are listed in Table 55.

Name	Device Usage
DIMEtalk\library\Bridges\4bitbridge_async.support\All\async_bridge_wr4.vhd	All
DIMEtalk\library\Bridges\4bitbridge_async.support\All\async_bridge_rd4.vhd	All
DIMEtalk\library\Bridges\4bitbridge_async.support\All\async_bridge_4.vhd	All
DIMEtalk\library\Bridges\async_common.support\bridge_fifo_ve.edn	All
DIMEtalk\library\Bridges\async_common.support\bridge_fifo.edn	All
DIMEtalk\library\Bridges\async_common_simulation.support\bridge_fifo_ve.vhd	Simulation All Devices
DIMEtalk\library\Bridges\async_common_simulation.support\bridge_fifo.vhd	Simulation All Devices

Table 55: 4-Bit Bridge Support Files

Component Definition

```
Component async_bridge
Generic (
        plug : boolean := TRUE;
        linkwidth : integer := 32
);
Port (
        -- reset
        reset : in std_logic;
        -- dt_clk
        dt_clk : in std_logic;
        -- DIMEtalk
```

```
        data_in : in std_logic_vector(31 downto 0);
        data_out : out std_logic_vector(31 downto 0);
        data_in_ack : out std_logic;
        data_out_ack : in std_logic;
        -- br_clk
        br_clk : in std_logic;
        -- Link
        brlink : inout std_logic_vector(linkwidth*2-1 downto 0);
        brlink_ack : inout std_logic_vector(1 downto 0)
);
End component;
```

8-Bit Bidirectional Asynchronous Bridge

Functional Description

This component transfers DIMEtalk packet data from one physical device to another using an 18-bit wide bus. This is used as 8-bits data in each direction plus one bit of flow control. The Bridge is asynchronous with the overall DIMEtalk clock. The speed of the link is governed by the br_clk input and can be either faster or slower than the overall network DIMEtalk clock, therefore the Bridge can be used to operate over slower links without slowing the overall network, or faster links to make better use of limited width connections. The Bridge signals are shown in Figure 36 and described in Table 56.

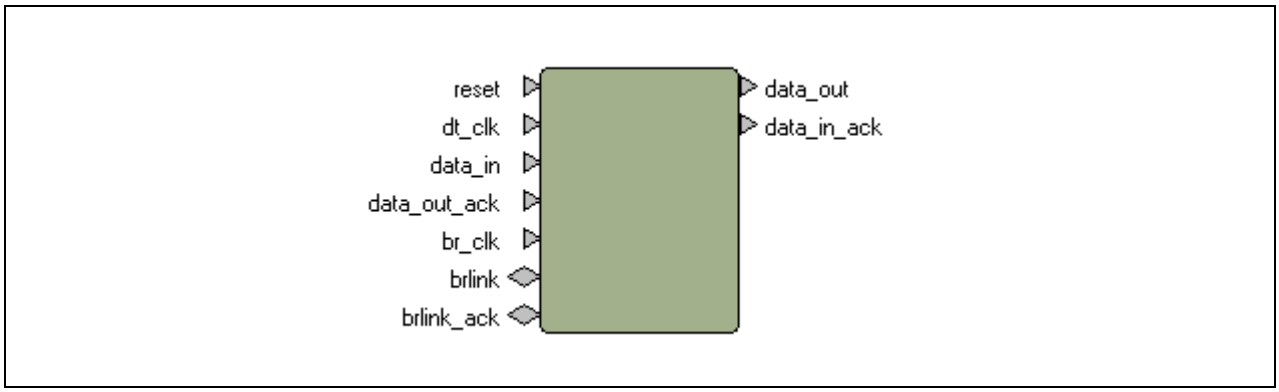

Figure 36: 8-Bit Bridge Component

Signals

The signal descriptions are provided in Table 56.

Group	Type	Signal Description	Width	Direction
reset	DIMEtalk reset	reset	1-bit	In
dt_clk	DIMEtalk network clock. Def. CLK1	dt_clk	1-bit	In
DIMEtalk		data_in	31 downto 0 bits	In

Table 56: : 8-Bit Bridge Signal Descriptions

Group	Type	Signal Description	Width	Direction
DIMEtalk		data_out	31 downto 0 bits	Out
DIMEtalk		data_in_ack	1-bit	Out
DIMEtalk		data_out_ack	1-bit	In
br_clk	Def. CLK3	br_clk	1-bit	In
Link		brlink	15 downto 0 bits	Inout
Link		brlink_ack	1 downto 0 bits	Inout

Table 56: : 8-Bit Bridge Signal Descriptions

Support Files

The support file names and devices used are listed in Table 57.

Name	Device Usage
DIMEtalk\library\Bridges\8bitbridge_async.support\All\async_bridge_8.vhd	All
DIMEtalk\library\Bridges\8bitbridge_async.support\All\async_bridge_wr8.vhd	All
DIMEtalk\library\Bridges\8bitbridge_async.support\All\async_bridge_rd8.vhd	All
DIMEtalk\library\Bridges\async_common.support\bridge_fifo_ve.edn	All
DIMEtalk\library\Bridges\async_common.support\bridge_fifo.edn	All
DIMEtalk\library\Bridges\async_common_simulation.support\bridge_fifo_ve.vhd	Simulation All Devices
DIMEtalk\library\Bridges\async_common_simulation.support\bridge_fifo.vhd	Simulation All Devices

Table 57: 8-Bit Bridge Support Files

Component Definition

```
Component async_bridge
Generic (
        plug : boolean := TRUE;
        linkwidth : integer := 32
);
Port (
        -- reset
        reset : in std_logic;
        -- dt_clk
        dt_clk : in std_logic;
        -- DIMEtalk
        data_in : in std_logic_vector(31 downto 0);
        data_out : out std_logic_vector(31 downto 0);
```

```
        data_in_ack : out std_logic;
        data_out_ack : in std_logic;
        - br_clk
        br_clk : in std_logic;
        -- Link
        brlink : inout std_logic_vector(linkwidth*2-1 downto 0);
        brlink_ack : inout std_logic_vector(1 downto 0)
);
End component;
```

16-Bit Bidirectional Asynchronous Bridge

Functional Description

This component transfers DIMEtalk packet data from one physical device to another using a 34-bit wide bus. This is used as 16-bits data in each direction plus one bit of flow control. This Bridge is asynchronous with the overall DIMEtalk clock. The speed of the link is governed by the br_clk input which can be faster or slower than the overall network DIMEtalk clock. This Bridge can therefore be used to operate over slower links without slowing the overall network, or faster links to make better use of limited width connections. The Bridge signals are shown in Figure 37 and described in Table 58.

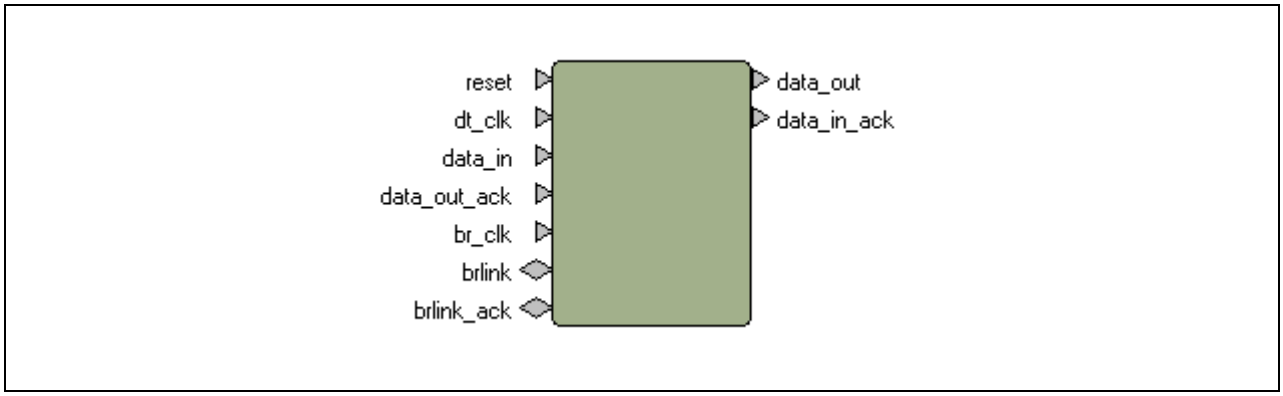

Figure 37: 16-Bit Bridge Component

Signals

The signal descriptions are provided in Table 58.

Group	Type	Signal Description	Width	Direction
reset	DIMEtalk reset	reset	1-bit	in
dt_clk	DIMEtalk network clock. Def. CLK1	dt_clk	1-bit	in
DIMEtalk		data_in	31 downto 0 bits	in
DIMEtalk		data_out	31 downto 0 bits	out
DIMEtalk		data_in_ack	1-bit	out

Table 58: 16-Bit Bridge Signal Descriptions

Group	Type	Signal Description	Width	Direction
DIMEtalk		data_out_ack	1-bit	in
br_clk	Def. CLK3	br_clk	1-bit	in
Link		brlink	31 downto 0 bits	inout
Link		brlink_ack	1 downto 0 bits	inout

Table 58: 16-Bit Bridge Signal Descriptions

Support Files

The support file names and devices used are listed in Table 59.

Name	Device Usage
DIMEtalk\library\Bridges\16bitbridge_async.support\All\async_bridge_16.vhd	All
DIMEtalk\library\Bridges\16bitbridge_async.support\All\async_bridge_wr16.vhd	All
DIMEtalk\library\Bridges\16bitbridge_async.support\All\async_bridge_rd16.vhd	All
DIMEtalk\library\Bridges\async_common.support\bridge_fifo_ve.edn	All
DIMEtalk\library\Bridges\async_common.support\bridge_fifo.edn	All
DIMEtalk\library\Bridges\async_common_simulation.support\bridge_fifo_ve.vhd	Simulation All Devices
DIMEtalk\library\Bridges\async_common_simulation.support\bridge_fifo.vhd	Simulation All Devices

Table 59: 16-Bit Bridge Support Files

Component Definition

```
Component async_bridge
Generic (
        plug : boolean := TRUE;
        linkwidth : integer := 32
);
Port (
        -- reset
        reset : in std_logic;
        -- dt_clk
        dt_clk : in std_logic;
        -- DIMEtalk
        data_in : in std_logic_vector(31 downto 0);
        data_out : out std_logic_vector(31 downto 0);
        data_in_ack : out std_logic;
        data_out_ack : in std_logic;
        -- br_clk
        br_clk : in std_logic;
```

```
        -- Link
        brlink : inout std_logic_vector(linkwidth*2-1 downto 0);
        brlink_ack : inout std_logic_vector(1 downto 0)
);
End component;
```

32-Bit Bidirectional Asynchronous Bridge

This component transfers DIMEtalk packet data from one physical device to another using a 66-bit wide bus. This is used as 32-bits data in each direction plus one bit of flow control. This Bridge is asynchronous with the overall DIMEtalk clock. The speed of the link is governed by the br_clk input which can either be faster or slower than the overall network DIMEtalk clock. The Bridge can therefore be used to operate over slower links without slowing the overall network or faster links to make better use of limited width connections. The Bridge signals are shown in Figure 38 and described in Table 60.

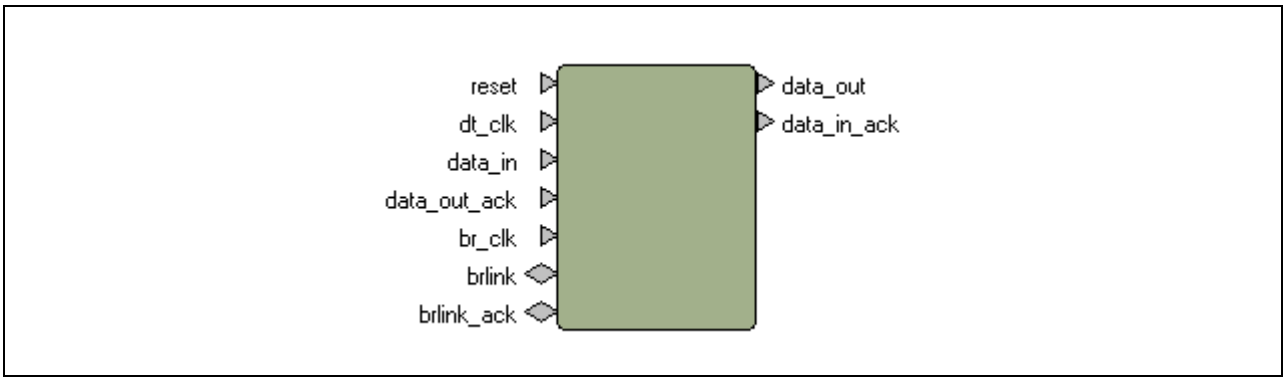

Figure 38: 32-Bit Bridge Component

Signals

The signal descriptions are provided in Table 60.

Group	Type	Signal Description	Width	Direction
reset	DIMEtalk reset	reset	1-bit	In
dt_clk	DIMEtalk network clock. Def. CLK1	dt_clk	1-bit	In
DIMEtalk		data_in	32-bits	In
DIMEtalk		data_out	32-bits	Out
DIMEtalk		data_in_ack	1-bit	Out
DIMEtalk		data_out_ack	1-bit	In
br_clk	Def. CLK3	br_clk	1-bit	In
Link	External connection	brlink	64 -bits	Inout
Link	External connection	brlink_ack	2-bits	Inout

Table 60: 32-Bit Bridge Signal Descriptions

Support Files

The support file names and devices used are listed in Table 61.

Name	Device Usage
DIMEtalk\library\Bridges\32bitbridge_async.support\All\async_bridge_32.vhd	All
DIMEtalk\library\Bridges\async_common_simulation.support\bridge_fifo_ve.vhd	Simulation All Devices
DIMEtalk\library\Bridges\async_common_simulation.support\bridge_fifo.vhd	Simulation All Devices
DIMEtalk\library\Bridges\32bitbridge_async.support\All\async_bridge_wr32.vhd	All
DIMEtalk\library\Bridges\32bitbridge_async.support\All\async_bridge_rd32.vhd	All
DIMEtalk\library\Bridges\async_common.support\bridge_fifo_ve.edn	All
DIMEtalk\library\Bridges\async_common.support\bridge_fifo.edn	All

Table 61: 32-Bit Bridge Support Files

Component Definition

```
Component async_bridge
Generic (
        plug : boolean := TRUE;
        linkwidth : integer := 32
);
        Port (
        -- reset
        reset : in std_logic;
        -- dt_clk
        dt_clk : in std_logic;
        -- DIMEtalk
        data_in : in std_logic_vector(31 downto 0);
        data_out : out std_logic_vector(31 downto 0);
        data_in_ack : out std_logic;
        data_out_ack : in std_logic;
        -- br_clk
        br_clk : in std_logic;
        -- Link
        brlink : inout std_logic_vector(linkwidth*2-1 downto 0);
        brlink_ack : inout std_logic_vector(1 downto 0)
);
End component;
```

RocketIO Bridge

Functional Description

This module transfers DIMEtalk packet data from one physical device to another using the RocketIO high speed serial bus. Transmit and receive FIFOs are implemented in order to provide elasticity in either direction. Flow control is utilized to allow back pressure to be applied to the transmitter as the receive buffer approaches its upper fill level. The RocketIO Bridge component is shown in Figure 39.

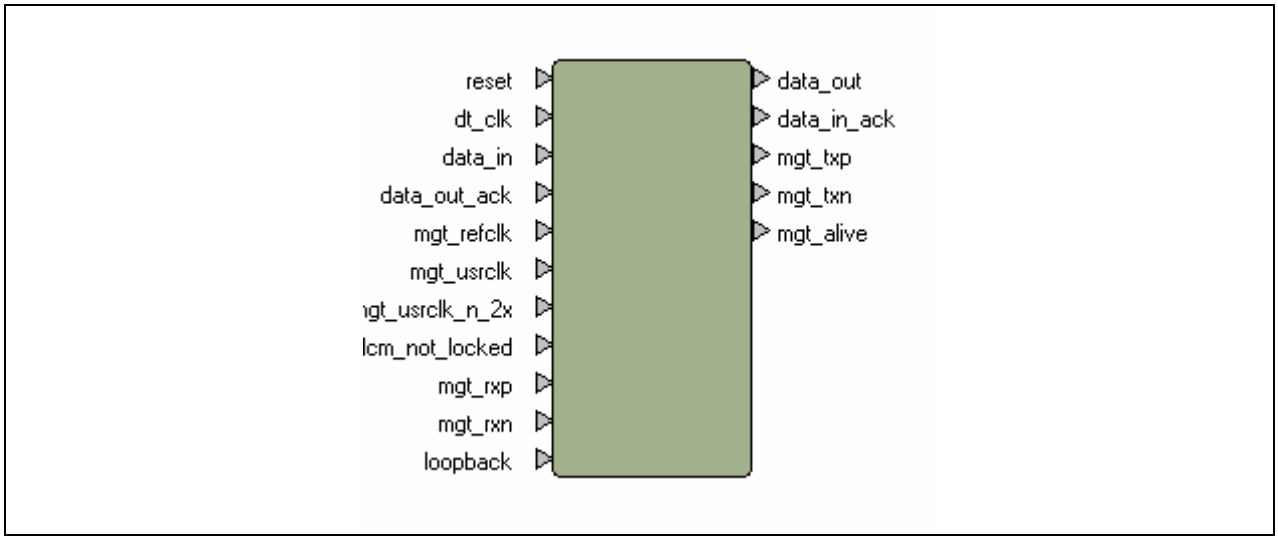

Figure 39: RocketIO Bridge Component

Additional Modules

This module must be coupled with the Rocketio_clocks component which provides the required reference and user clocks.

Signals

The signal descriptions are provided in Table 62.

Group	Type	Signal Description	Width	Direction
reset	DIMEtalk reset	reset	1-bit	in
dt_clk	DIMEtalk network clock. Def. CLK1	dt_clk	1-bit	in
DIMEtalk		data_in	32-bits	in
DIMEtalk		data_out	32-bits	out
DIMEtalk		data_in_ack	1-bit	out
DIMEtalk		data_out_ack	1-bit	in
Refclk		Mgt_refclk	1-bit	In
Usrclk		Mgt_usrclk	1-bit	In
Usrclk_n_2x		Mgt_usrclk_n_2x	1-bit	In
Mgt_rx		Mgt_rxp	1-bit	In
Mgt_rx		Mgt_rxn	1-bit	In
Mgt_tx		Mgt_txp	1-bit	Out

Table 62: RocketIO Bridge Signal Descriptions

Group	Type	Signal Description	Width	Direction
Mgt_tx		Mgt_txn	1-bit	Out
Dcm_status		Dcm_not_locked	1-bit	In
Mgt status		Mgt_alive	1-bit	Out

Table 62: RocketIO Bridge Signal Descriptions

User Generics

The user generics are shown in Table 63.

Name	Type	Description
CLK2_SEL	Boolean	This is combined with REF_CLK_V_SEL (which is set up in the UCF) to ensure that the GT_CUSTOM core uses the correct reference clock as follows clk2sel = true: selects REFCLK2 if REF_CLK_V_SEL = 0 selects BREFCLK2 if REF_CLK_V_SEL = 1 clk2sel = false: selects REFCLK if REF_CLK_V_SEL = 0 selects BREFCLK if REF_CLK_V_SEL = 1

Table 63: RocketIO Bridge User Generics

Support Files

The support file names and devices used are listed in Table 64.

Name	Device Usage
DIMEtalk\library\Bridges\rocketio_bridge.support\All\rocketio_bridge.vhd	All
DIMEtalk\library\Bridges\rocketio_bridge.support\All\aurora32.ngc	All
DIMEtalk\library\Bridges\rocketio_bridge.support\All\dtbridge_egress.vhd	All
DIMEtalk\library\Bridges\rocketio_bridge.support\All\dtbridge_ingress.vhd	All
DIMEtalk\Common\bretime.vhd	All
DIMEtalk\Common\blk_async_fifo.vhd	All
DIMEtalk\Common\blk_dpram.vhd	All
DIMEtalk\Common\retime.vhd	All
DIMEtalk\Common\pkg_dimetalk_global.vhd	All

Table 64: RocketIO Bridge Support Files

Component Definition

```
component rocketio_bridge
        generic (
        clk2sel : boolean);
        port (
        reset          : in  std_logic;
        dt_clk         : in  std_logic;
        data_in        : in  std_logic_vector(DT_DATA_SZ-1 downto 0);
        data_out       : out std_logic_vector(DT_DATA_SZ-1 downto 0);
        data_in_ack    : out std_logic;
        data_out_ack   : in  std_logic;
        mgt_refclk     : in  std_logic;
        mgt_usrclk     : in  std_logic;
        mgt_usrclk_n_2x : in  std_logic;
        mgt_rxp        : in  std_logic;
        mgt_rxn        : in  std_logic;
        mgt_txp        : out std_logic;
        mgt_txn        : out std_logic;
        dcm_not_locked  : in  std_logic;
        loopback       : in  std_logic;
        mgt_alive      : out std_logic);
        end component;
```

RocketIO Clock Component

Functional Description

This module provides the clocks required to drive the RocketIO Bridge. The RocketIO Bridge requires the following clocks:

1. Reference clock

   A low jitter reference clock is required to drive the high speed transmit and receive clock recovery circuits. This can be supplied from REFCLK, REFCLK2, BREFCLK or BREFCLK2. The choice will vary from module to module depending on which input pin the input reference oscillator has been connected to on the device. REFCLK and REFCLK2 are flexible reference clock inputs that can be driven from any global on chip clock resource except DCMs. The main benefit is that one reference clock from REFCLK/ REFCLK2 can supply RocketIO lanes located on the top and bottom edges of the FPGA. However as these use regular clock routing they are jitter-limited to carrying 100MHz, thereby limiting serial line rate to 20x100Mhz = 2Gb/s. BREFCLK and BREFCLK2 are dedicated low jitter differential clock networks for the RocketIO transceivers. They can support serial line rates above 2Gb/s but have one dedicated network for the top edge of the device and one for the bottom. Check the specific *DIME-II Module Reference Guide* to see which reference clocks are supported on the targeted module.

2. User clocks

The RocketIO Bridge requires two parallel clocks that are frequency locked to the reference clock (user_clk and user_clk_2x_n).

This module produces a dcm_not_locked which is connected to the RocketIO Bridge to hold it in reset until the clocks are stable. The RocketIO component is shown in Figure 40.

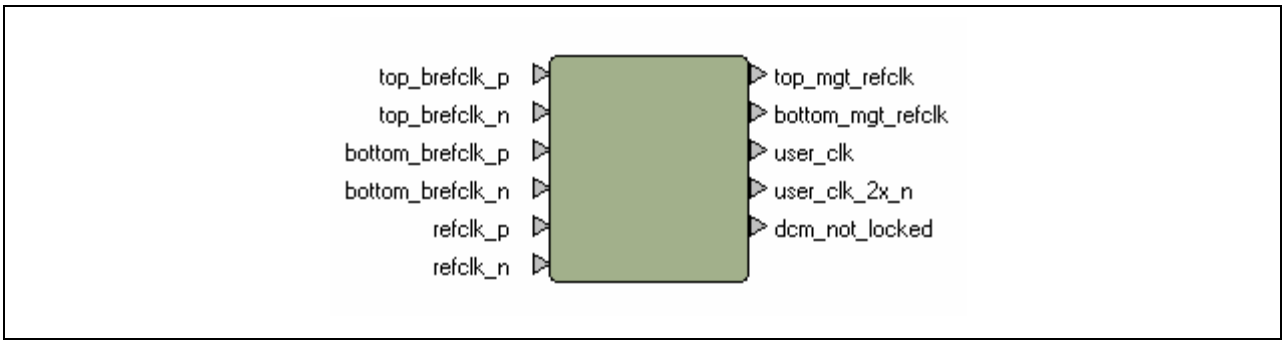

Figure 40: RocketIO Component

Signals

The signal descriptions are provided in Table 65.

Group	Type	Signal Description	Width	Direction
Reset raw	Connects to active high version of system reset	Dcm_reset	1-bit	In
Top_brefclk	Top edge (BREFCLK or BREFCLK2)	Top_brefclk_p	1-bit	In
Top_brefclk	Top edge	Top_brefclk_n	1-bit	In
Bottom_brefclk	Bottom edge (BREFCLK or BREFCLK2)	Bottom_brefclk_p	1-bit	In
Bottom_brefclk	Bottom edge	Bottom_brefclk_n	1-bit	In
Device refclk	Connect to REFCLK or REFCLK2	refclk_p	1-bit	In
Device refclk		refclk_n	1-bit	In
MGT refclk	Connect to top edge RocketIO bridge refclk	Top_mgt_refclk	1-bit	Out
MGT refclk	Connect to bottom edge RocketIO bridge refclk	Bottom_mgt_refclk	1-bit	Out
Usrclk	Connect to RocketIO bridge usr_clk	Usr_clk	1-bit	Out
Usrclk_n_2x	Connect to RocketIO bridge usr_clk_n_2x	Usr_clk_n_2x	1-bit	Out
Dcm_status	Connect to RocketIO bridge dcm_not_locked	Dcm_not_locked	1-bit	In

Table 65: RocketIO Clocks Signal Descriptions

User Generics

The user generics are shown in Table 66.

Name	Type	Description
Top_brefclk_used	Boolean	True if the brefclk from the top edge of the device is used.

Table 66: RocketIO Generics

Name	Type	Description
Bottom_brefclk_used	Boolean	True if the brefclk from the top bottom of the device is used. Note that it is valid to have both brefclk inputs used as this module sources reference clocks for top and bottom edges of the device.

Table 66: RocketIO Generics

Support Files

The support file names and devices used are listed in Table 67.

Name	Device Usage
DIMEtalk\library\Bridges\rocketio_clocks.support\All\rocketio_clocks.vhd	Virtex-II Pro

Table 67: RocketIO Clocks Support Files

Component Definition

```
component rocketio_clocks
        generic (
        top_brefclk_used    : boolean;
        bottom_brefclk_used : boolean);
        port (
        top_brefclk_p     : in  std_logic;
        top_brefclk_n     : in  std_logic;
        bottom_brefclk_p  : in  std_logic;
        bottom_brefclk_n  : in  std_logic;
        refclk_p          : in  std_logic;
        refclk_n          : in  std_logic;
        top_mgt_refclk    : out std_logic;
        bottom_mgt_refclk : out std_logic;
        user_clk          : out std_logic;
        user_clk_2x_n     : out std_logic;
        dcm_not_locked    : out std_logic);
        end component;
```

DIMEtalk DDR to MGT Bridge

Functional Description

The DDR to MGT Bridge, shown in Figure 41, is used to connect DIMEtalk networks together via the MGT links. This component does not directly connect to DIME-C processes. It works in the same way as normal bridges in DIMEtalk. except that the DDR bridge always acts as a plug rather than a socket.

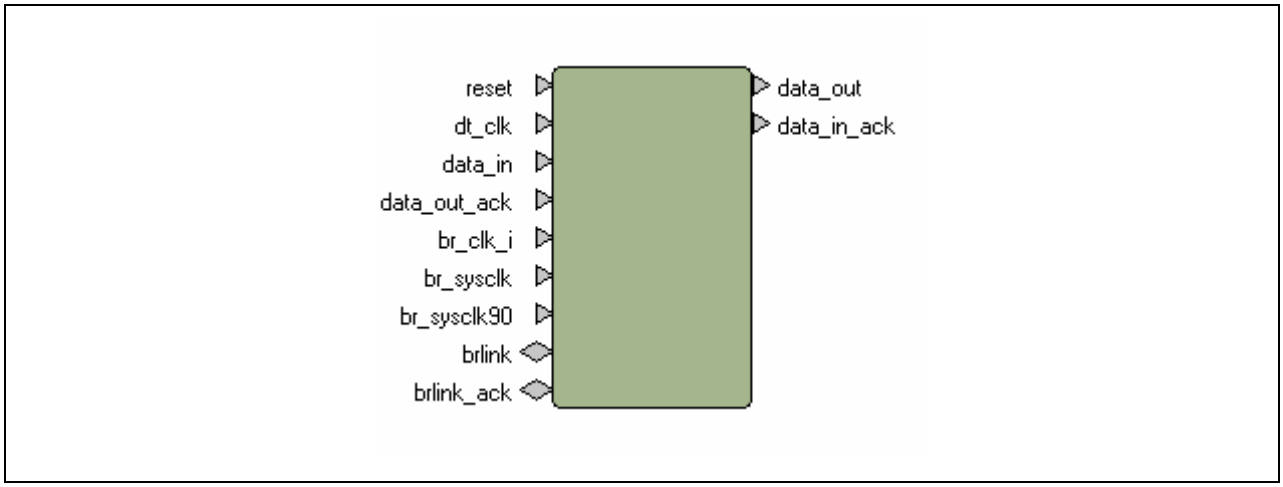

Figure 41: DIMEtalk DDR to MGT Bridge Component

Signals

The signal descriptions are provided in Table 68.

Group	Type	Signal Description	Signal Width	Direction
sysreset	Reset Connection	DIMEtalk Reset	1-bit	IN
dt_clk	Clock Connection	dt_clk : in STD_LOGIC;	1-bit	IN
DIMEtalk	DIMEtalk Connection	data_in : in STD_LOGIC_VECTOR(31 downto 0);	32-bits	IN
		data_out : out STD_LOGIC_VECTOR(31 downto 0);	32-bits	OUT
		data_in_ack : out STD_LOGIC;	1-bit	OUT
		data_out_ack : in STD_LOGIC;	1-bit	IN
UdimeBridgeClocks	User Connection	br_clk_i : in STD_LOGIC;	1-bit	IN
		br_sysclk : in STD_LOGIC;	1-bit	IN
		br_sysclk90 : in STD_LOGIC;	1-bit	IN
BridgeLink	User Connection	brlink : inout STD_LOGIC_VECTOR(15 downto 0);	16-bits	INOUT
		brlink_ack : inout STD_LOGIC_VECTOR(1 downto 0);	2-bits	INOUT

Table 68: DIMEtalk DDR to MGT Bridge Signals

User Generics

The user generics are shown in Table 69.

Name	Type	Value	Read only
setasplug	boolean	true	No
VIRTEXE	boolean	false	Yes
linkwidth	integer	16	Yes
channel	integer	0	No

Table 69: DIMEtalk DDR to MGT Bridge User Generics

Please note that by default the channel parameter is set to 0. When placing this component down users must change the channel parameter to match the required MGT port. The channels are numbered 0,1,2 and 3 - if these are not set users may experience UCF and timing constraint errors at FPGA build time.

Support Files

The support file names and devices used are listed in Table 70.

Location	Device Usage
DIMEtalk\library\Bridges\ddr_async_bridge16.support\ddr_async_bridge_16.vhd	All
DIMEtalk\library\Bridges\ddr_async_bridge16.support\ddr_async_bridge16.vhd	All
DIMEtalk\library\Bridges\ddr_async_bridge16.support\bridge_fifo.edn	All
DIMEtalk\common\bretime.vhd	All
DIMEtalk\common\retime.vhd	All
DIMEtalk\library\Bridges\ddr_async_bridge16.support\async_bridge_wr16.vhd	All
DIMEtalk\library\Bridges\ddr_async_bridge16.support\async_bridge_rd16.vhd	All

Table 70: DIMEtalk DDR to MGT Bridge Support Files

1.5 Routers

Router - Reconfigurable Four Way Non-blocking

Functional Description

This component facilitates the routing of packets around the DIMEtalk network. Routing is based on destination node id of the arriving packet. This is used to access a routing table to give the exit port. Although the routing table is implemented in block RAM the user has the choice of implementing the temporary packet storage associated with this module in either block or distributed RAM. Block RAM is the more appropriate choice unless the user has none

available. The routing table can be dynamically reconfigured via dedicated DIMEtalk packets. The router signal names are shown in Figure 42 and described in Table 71.

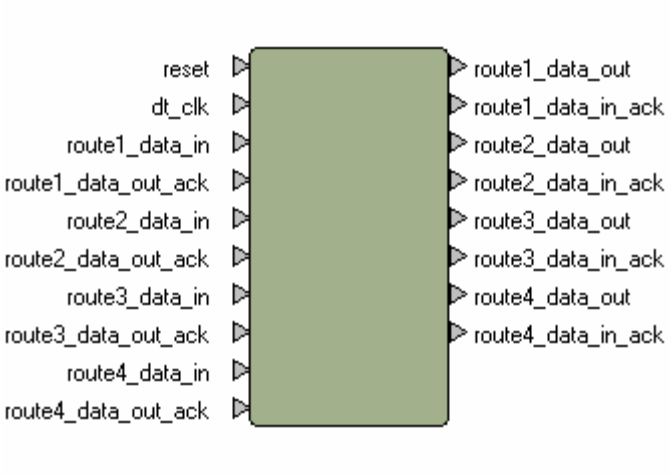

Figure 42: Router Component

Using a Router Component

DIMEtalk routers are 'store and forward' style routers which can handle four incoming/outgoing ports simultaneously. The throughput in a typical system is 32-bits wide at around 100MHz. The routers contain FIFOs with a depth of 16 packets which are treated in a 'round robin' fashion thereby avoiding collisions. Deadlock can be handled with the use of doorbells/semaphores. In order to create router information which allows packets to be propagated from a source router to a destination router, a router table is used in the block RAM instantiated in the router component.

Signals

The signal descriptions are provided in Table 71.

Group	Type	Signal Description	Width	Direction
reset	DIMEtalk reset	reset	1-bit	in
dt_clk	DIMEtalk network clock. Def. CLK1	dt_clk	1-bit	in
Dtport, Port 1		Route1_data_in	32-bits	in
Dtport, Port 1		Route1_data_out	32-bits	out
Dtport, Port 1		Route1_data_in_ack	1-bit	out
Dtport, Port 1		Route1_data_out_ack	1-bit	in
Dtport, Port 2		Route2_data_in	32-bits	in
Dtport, Port 2		Route2_data_out	32-bits	out
Dtport, Port 2		Route2_data_in_ack	1-bit	out
Dtport, Port 2		Route2_data_out_ack	1-bit	in
Dtport, Port 3		Route3_data_in	32-bits	in
Dtport, Port 3		Route3_data_out	32-bits	out
Dtport, Port 3		Route3_data_in_ack	1-bit	out
Dtport, Port 3		Route3_data_out_ack	1-bit	in
Dtport, Port 4		Route4_data_in	32-bits	in

Table 71: Router Signal Descriptions

Group	Type	Signal Description	Width	Direction
Dtport, Port 4		Route4_data_out	32-bits	out
Dtport, Port 4		Route4_data_in_ack	1-bit	out
Dtport, Port 4		Route4_data_out_ack	1-bit	in

Table 71: Router Signal Descriptions

User Generics

The user generics are shown in Table 72.

Name	Type	Description
VIRTEXE	Boolean	True if targeting Virtex-E otherwise false.
Pktramtype	String	"block" or "distributed"
Init00	Bit_vector	routing table initial value
Init01	Bit_vector	routing table initial value
Init02	Bit_vector	routing table initial value

Table 72: Router User Generics

Support Files

The support file names and devices used are listed in Table 73.

Name	Device Usage
DIMEtalk\library\Routers\router.support\All\router.vhd	All
DIMEtalk\library\Routers\router.support\All\router_junction.vhd	All
DIMEtalk\library\Routers\router.support\All\router_inputport.vhd	All
DIMEtalk\Common\dpram.vhd	All
DIMEtalk\Common\pkg_dimetalk_global.vhd	All

Table 73: Router Support Files

Component Definition

```
component router
generic (
        VIRTEXE         :   boolean;
        pktramtype      :   string;
        init00          :   bit_vector;
        init01          :   bit_vector;
        init02          :   bit_vector);
port (
        reset           : in  std_logic;
        dt_clk          : in  std_logic;
        route1_data_in      : in  std_logic_vector(31 downto 0);
        route1_data_out     : out std_logic_vector(31 downto 0);
        route1_data_in_ack  : out std_logic;
        route1_data_out_ack : in  std_logic;
        route2_data_in      : in  std_logic_vector(31 downto 0);
        route2_data_out     : out std_logic_vector(31 downto 0);
        route2_data_in_ack  : out std_logic;
        route2_data_out_ack : in  std_logic;
        route3_data_in      : in  std_logic_vector(31 downto 0);
        route3_data_out     : out std_logic_vector(31 downto 0);
        route3_data_in_ack  : out std_logic;
        route3_data_out_ack : in  std_logic;
        route4_data_in      : in  std_logic_vector(31 downto 0);
        route4_data_out     : out std_logic_vector(31 downto 0);
        route4_data_in_ack  : out std_logic;
        route4_data_out_ack : in  std_logic);
end component;
```

1.6 System Clock Driver

Functional Description

This component should be included in every module to handle the routing and synchronization of the clocks and resets to the module. It takes three basic clocks and a reset signal as input. Each clock (that is used in the design) is routed to a Digital Clock Manager for deskew. The output of this is fed to the user design, as well as a 2xfreq version and the original unskewed clock. The reset signal output from this block is only deasserted when all used clocks have locked. The Clock Driver Signals are shown in Figure 43 and described in Table 74.

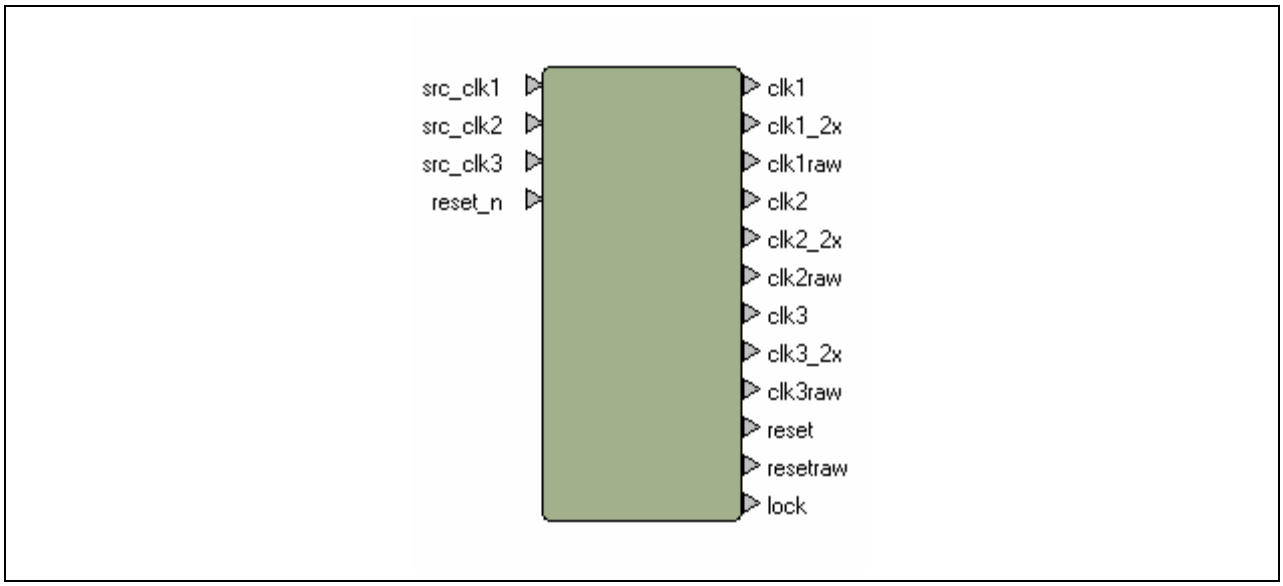

Figure 43: Clock Driver Module

Signals

The signal descriptions are provided in Table 74.

Group	Type	Signal Description	Width	Direction
Clockreset_in	FPGA input	Src_clk1	1-bit	In
Clockreset_in	FPGA input	Src_clk2	1-bit	In
Clockreset_in	FPGA input	Src_clk3	1-bit	In
Clockreset_in	FPGA input	Reset_n	1-bit	In
Clk1	Deskewed clk1	Clk1	1-bit	Out
Clk1x2	Deskewed clk1 2x	Clk1_2x	1-bit	Out
Clk1raw	Copy of src_clk1	Clk1raw	1-bit	Out
Clk2	Deskewed clk2	Clk2	1-bit	Out
Clk2x2	Deskewed clk2 2x	Clk2_2x	1-bit	Out
Clk2raw	Copy of src_clk2	Clk2raw	1-bit	Out
Clk3	Deskewed clk3	Clk3	1-bit	Out
Clk3x2	Deskewed clk3 2x	Clk3_2x	1-bit	Out

Table 74: Clock Driver Signals

Group	Type	Signal Description	Width	Direction
Clk3raw	Copy of src_clk3	Clk3raw	1-bit	Out
Reset	DIMEtalk reset (generated from DCM lock signals). This is used to synchronize the operation of the FPGA with the locking of the FPGA clocks.	Reset	1-bit	Out
Resetraw	Raw reset, this is an inverted version of the input reset_n signal.	Resetraw	1-bit	Out

Table 74: Clock Driver Signals

User Generics

The user generics are shown in Table 75.

Name	Type	Description
clk1_used	Boolean	Clock 1 used
clk2_used	Boolean	Clock 2 used
clk3_used	Boolean	Clock 3 used

Table 75: Clock Driver User Generics

Support Files

The support file names and devices used are listed in Table 76.

Name	Device Usage
System\source\clocks.vhd	All

Table 76: Clock Driver Support Files

Component Definition

```
component clocks
generic(
        clk1_used     : boolean := TRUE;
        clk2_used     : boolean := FALSE;
        clk3_used     : boolean := FALSE
);
port(
        -- source clocks
        src_clk1      : in  std_logic;
        src_clk2      : in  std_logic;
        src_clk3      : in  std_logic;
        -- DLL/DCM reset
        reset_n       : in  std_logic;
        -- Output clocks
```

```
        clk1        : out std_logic;
        clk1_2x     : out std_logic;
        clk2        : out std_logic;
        clk2_2x     : out std_logic;
        clk3        : out std_logic;
        clk3_2x     : out std_logic;
        -- Output System Reset
        reset       : out std_logic;
        -- Clock lock status
        lock        : out std_logic
);
end clocks;
```

Clock Deskew Component for BenBLUE Modules

Functional Description

In the BenBLUE module CLKA is directly connected to the Primary FPGA but not to the Secondary FPGA. Therefore a circuit is required in the Primary FPGA to generate a clock for the internal circuitry but also a de-skewed version to distribute to the logic in the Secondary FPGA. CLKA requires external de-skewing as it must drive the CLKA input on the Secondary FPGA. For more information on how to connect the FPGAs refer to the *Connecting FPGAs in DIMEtalk Application Note* on the DIMEtalk support lounge. The Clock Deskew component is shown in Figure 44.

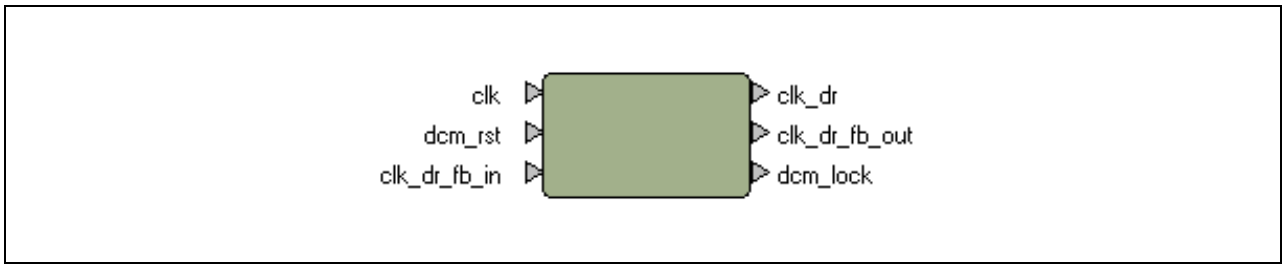

Figure 44: Clock Deskew Component for BenBLUE Module

Signals

The signal descriptions are provided in Table 77.

Group	Type	Signal Description	Width	Direction
Clk	Clock Connection	clk	1-bit	In
Reset	Reset Connection	dcm_rst	1-bit	In
Clock Forwarding	User Connection	clk_dr	1-bit	Out
Clock Forwarding	User Connection	clk_dr_fb_in	1-bit	In
Clock Forwarding	User Connection	clk_dr_fb_out	1-bit	Out
dcm_lock	User Connection	dcm_lock	1-bit	Out

Table 77: Clock Deskew for BenBLUE Signals

Support Files

The support file names and devices used are listed in Table 78.

Name	Device Usage
DIMEtalk\library\System\bb2deskew.support\All\bb2deskew.vhd	All devices

Table 78: Clock Deskew for BenBLUE Support Files

Component Definition

deskewclocks_0

```
entity deskewclocks is
port(
        clk          : in  std_logic;
        dcm_rst      : in  std_logic;
        -- Secondary FPGA clock & Deskew
        clk_dr       : out std_logic;
        clk_dr_fb_in  : in  std_logic;
        clk_dr_fb_out : out std_logic;
        dcm_lock     : out std_logic
);
```

PCI-X Clocks Driver Component

Users should refer to "DIMEtalk Clock Usage on the BenNUEY-PCI-X-V4" for information on how this component is used with the BenNUEY-PCI-X-V4 motherboard.

Functional Description

This component should be included in every PCI-X user design to handle the routing and synchronization of the system and DDR2 SRAM interface clocks and resets. The PCI-X Clocks Driver component is shown in Figure 45.

System Clocks

Each system clock is routed to a Digital Clock Manager for deskew. The output of this is fed to the user design, as well as a 2xfreq version and the original unskewed clock. The reset signal output from this block is only deasserted when all used clocks have locked.

DDR2 SRAM Interface Clocks

In addition to deskewing the system clocks the PCI-X Clocks Driver also provides the clocks required for the SRAM interface including the I-delay clock (idly_clk), phase shifted clocks (mem_clk0, mem_clk90) and a 50MHz clock used by the I-delay calibration state machine (mem_clk50).

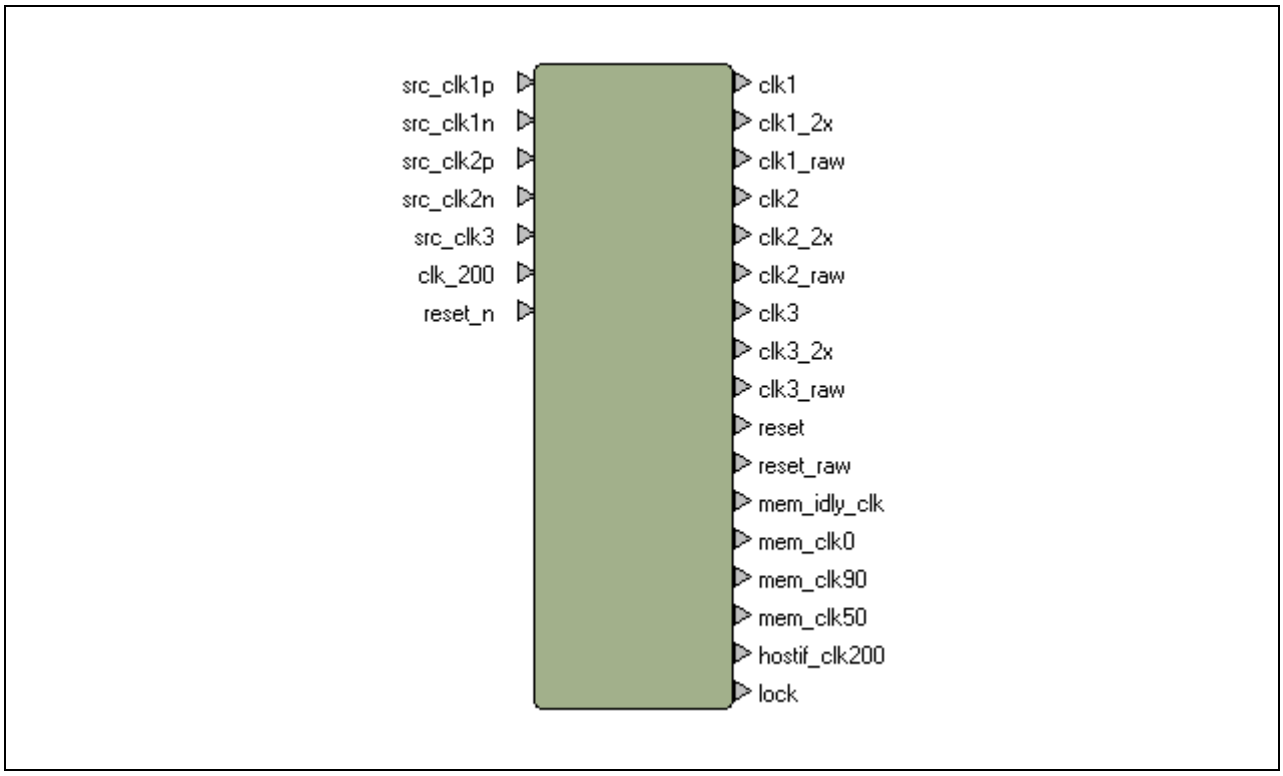

Figure 45: PCI-X Clocks Driver Component

Signals

The signal descriptions are provided in Table 79.

Group	Type	Signal Description	Width	Direction
clockreset_in	User Connection	src_clk1p : in STD_LOGIC;	1-bit	In
		src_clk1n : in STD_LOGIC;	1-bit	In
		src_clk2p : in STD_LOGIC;	1-bit	In
		src_clk2n : in STD_LOGIC;	1-bit	In
		src_clk3 : in STD_LOGIC;	1-bit	In
		clk_200 : in STD_LOGIC;	1-bit	In
		reset_n : in STD_LOGIC;	1-bit	In
clk1	Clock Connection	clk1 : out STD_LOGIC;	1-bit	Out
clk1_2x	Clock Connection	clk1_2x : out STD_LOGIC;	1-bit	Out
clk1_raw	Clock Connection	clk1_raw : out STD_LOGIC;	1-bit	Out
clk2	Clock Connection	clk2 : out STD_LOGIC;	1-bit	Out
clk2_2x	Clock Connection	clk2_2x : out STD_LOGIC;	1-bit	Out
clk2_raw	Clock Connection	clk2_raw : out STD_LOGIC;	1-bit	Out
clk3	Clock Connection	clk3 : out STD_LOGIC;	1-bit	Out
clk3_2x	Clock Connection	clk3_2x : out STD_LOGIC;	1-bit	Out

Table 79: PCI-X Clocks Driver Component Signals

Group	Type	Signal Description	Width	Direction
clk3_raw	Clock Connection	clk3_raw : out STD_LOGIC;	1-bit	Out
reset	Reset Connection	reset : out STD_LOGIC;	1-bit	Out
reset_raw	Reset Connection	reset_raw : out STD_LOGIC;	1-bit	Out
DDR2CoreClocks	User Connection	idly_clk : out STD_LOGIC;	1-bit	Out
		mem_clk0 : out STD_LOGIC;	1-bit	Out
		mem_clk90 : out STD_LOGIC;	1-bit	Out
		mem_clk50 : out STD_LOGIC;	1-bit	Out
HostIfClocks	User Connection	hostif_clk200 : out STD_LOGIC;	1-bit	Out
UserPort	User Connection	lock : out STD_LOGIC;	1-bit	Out

Table 79: PCI-X Clocks Driver Component Signals

Support Files

The support file names and devices used are listed in Table 80.

Location	Device usage
DIMEtalk\library\System\pcix_clocks.support\pcix_clocks.vhd	All devices

Table 80: PCI-X Clocks Driver Component Support Files

Component Definition

```
component pcix_clocks
generic (
clk1_used : boolean;
clk2_used : boolean;
clk3_used : boolean);
port (
        src_clk1p : in  std_logic;
        src_clk1n : in  std_logic;
        src_clk2p : in  std_logic;
        src_clk2n : in  std_logic;
        src_clk3  : in  std_logic;
        clk_200   : in  std_logic;
        reset_n   : in  std_logic;
        clk1      : out std_logic;
        clk1_2x   : out std_logic;
        clk1_raw  : out std_logic;
        clk2      : out std_logic;
```

```
                clk2_2x   : out std_logic;
                clk2_raw  : out std_logic;
                clk3      : out std_logic;
                clk3_2x   : out std_logic;
                clk3_raw  : out std_logic;
                idly_clk  : out std_logic;
                mem_clk0  : out std_logic;
                mem_clk90 : out std_logic;
                mem_clk50 : out std_logic;
                reset     : out std_logic;
                reset_raw : out std_logic;
                lock      : out std_logic);
    end component;
```

PCI-X Module Clocks Driver Component

Users should refer to "DIMEtalk Clock Usage on the BenNUEY-PCI-X-V4" for information on how this component is used with the BenNUEY-PCI-X-V4 motherboard.

Functional Description

This is the clock & reset component for Nallatech Virtex-4 modules populated on a BenNUEY-PCI-X-V4 motherboard. This component is required as the BenNUEY-PCI-X-V4 motherboard provides differential clocks to the modules. If a Virtex-4 module is plugged into a non Virtex-4 motherboard (e.g. a BenNUEY or BenONE) the standard Clock Driver component should be used. On the BenBLUE-V4 module the standard Clock Driver component should be used for the Secondary FPGA as this FPGA receives a single ended clock from the Primary FPGA. The PCI-X Module Clocks Driver component is shown in Figure 46.

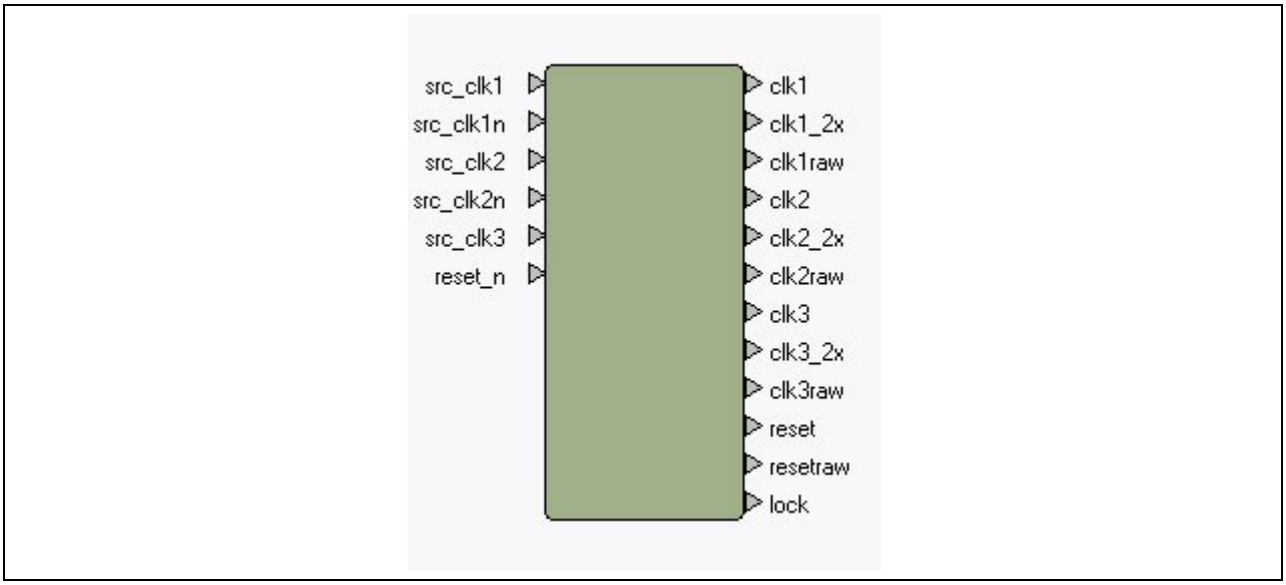

Figure 46: PCI-X Module Clocks Driver Component

System Clocks

Each system clock is routed to a Digital Clock Manager (DCM) for deskew. The output of this is fed to the user design, as well as a 2xfreq version and the original unskewed clock. The reset signal output from this block is only deasserted when all used clocks have locked.

Signals

The signal descriptions are provided in Table 81.

Group	Type	Signal Description	Width	Direction
clockreset_in	User Connection	src_clk1 : in STD_LOGIC;	1-bit	In
		src_clk1n : in STD_LOGIC;	1-bit	In
		src_clk2 : in STD_LOGIC;	1-bit	In
		src_clk2n : in STD_LOGIC;	1-bit	In
		src_clk3 : in STD_LOGIC;	1-bit	In
		reset_n : in STD_LOGIC;	1-bit	In
clk1	Clock Connection	clk1 : out STD_LOGIC;	1-bit	Out
clk1_2x	Clock Connection	clk1_2x : out STD_LOGIC;	1-bit	Out
clk1raw	Clock Connection	clk1_raw : out STD_LOGIC;	1-bit	Out
clk2	Clock Connection	clk2 : out STD_LOGIC;	1-bit	Out
clk2_2x	Clock Connection	clk2_2x : out STD_LOGIC;	1-bit	Out
clk2raw	Clock Connection	clk2raw : out STD_LOGIC;	1-bit	Out
clk3	Clock Connection	clk3 : out STD_LOGIC;	1-bit	Out
clk3_2x	Clock Connection	clk3_2x : out STD_LOGIC;	1-bit	Out
clk3raw	Clock Connection	clk3raw : out STD_LOGIC;	1-bit	Out
reset	Reset Connection	reset : out STD_LOGIC;	1-bit	Out
resetraw	Reset Connection	resetraw : out STD_LOGIC;	1-bit	Out
lock	User Connection	lock : out STD_LOGIC;	1-bit	Out

Table 81: PCI-X Module Clocks Driver Component

Support Files

The support file names and devices used are listed in Table 82.

Location	Device usage
DIMEtalk\library\System\pcix_module_clocks.support\pcix_module_clocks.vhd	All devices

Table 82: PCI-X Module Clocks Driver Component Support Files

H100 PCI-X Clocks Driver Module

Functional Description

This module provides all the clocks required on the H100 User FPGA, which includes SRAM and SDRAM 200MHz, host interface and system clocks. Additionally, an idelayctrl is instantiated in this module to ensure that these are enabled in the required banks. As a result of this, idlyctrl_rdy signals on H100 are distributed to the interfaces which use this feature in place of the idly clock signals on other products. H100 PCI-X clocks driver module is shown in Figure 47.

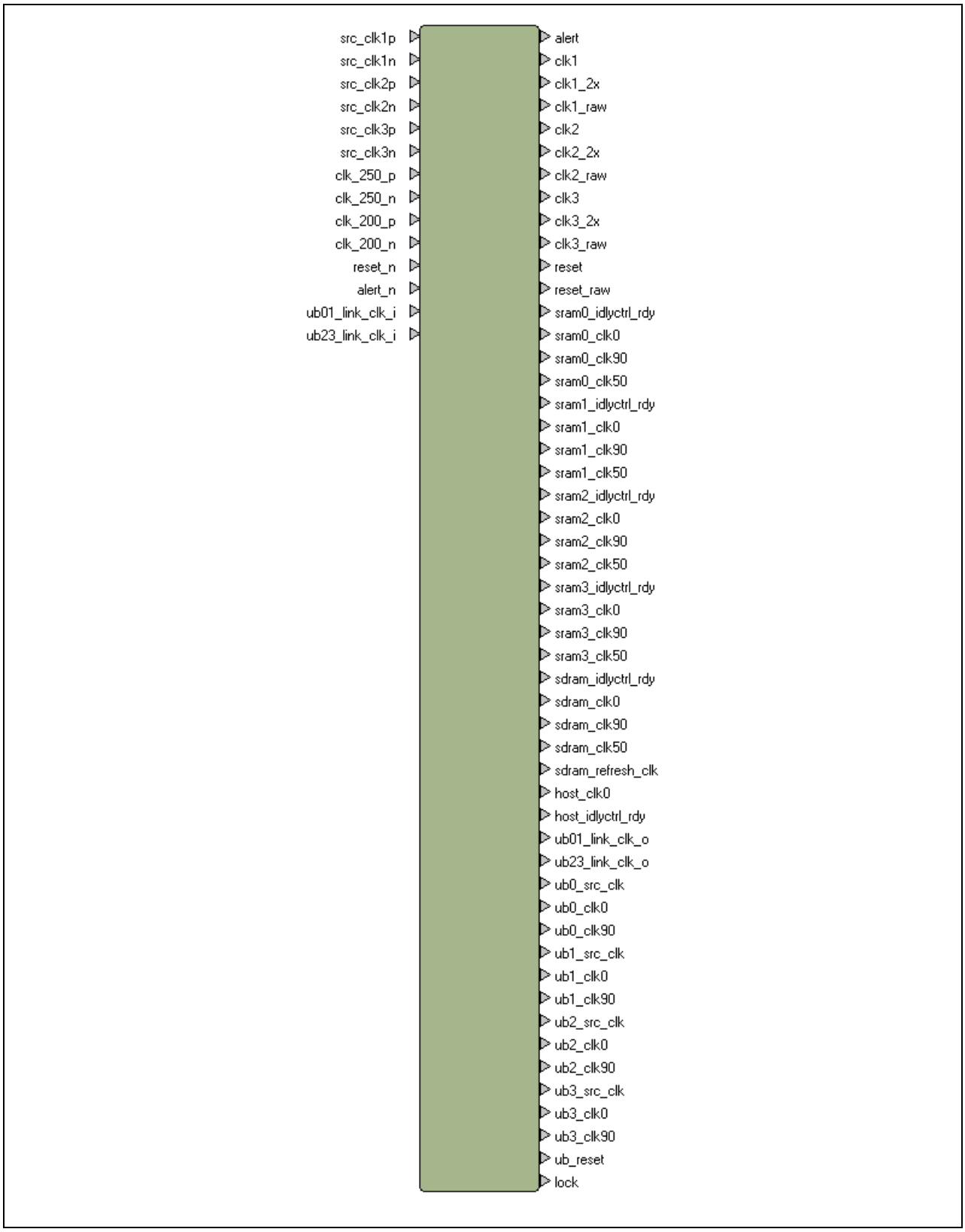

Figure 47: H100 PCI-X Module Clocks Driver Component

User Generics

The user generics are shown in Table 75.

Name	Type	Description
clk1_used	Boolean	Clock 1 used
clk2_used	Boolean	Clock 2 used
clk3_used	Boolean	Clock 3 used

Table 83: H100 PCI-X Module Clocks Driver User Generics

Signals

The signal descriptions are provided in Table 84.

Group	Type	Signal Description	Width	Direction
clockreset_in	User Connection	src_clk1p : in STD_LOGIC;	1-bit	In
		src_clk1n : in STD_LOGIC;	1-bit	In
		src_clk2p : in STD_LOGIC;	1-bit	In
		src_clk2n : in STD_LOGIC;	1-bit	In
		src_clk3p : in STD_LOGIC;	1-bit	In
		src_clk3n : in STD_LOGIC;	1-bit	In
		clk_250_p : in STD_LOGIC;	1-bit	In
		clk_250_n : in STD_LOGIC;	1-bit	In
		clk_200_p : in STD_LOGIC;	1-bit	In
		clk_200_n : in STD_LOGIC;	1-bit	In
		reset_n : in STD_LOGIC;	1-bit	In
clk1	Clock Connection	clk1 : out STD_LOGIC;	1-bit	Out
clk1_2x	Clock Connection	clk1_2x : out STD_LOGIC;	1-bit	Out
clk1_raw	Clock Connection	clk1_raw : out STD_LOGIC;	1-bit	Out
clk2	Clock Connection	clk2 : out STD_LOGIC;	1-bit	Out
clk2_2x	Clock Connection	clk2_2x : out STD_LOGIC;	1-bit	Out
clk2_raw	Clock Connection	clk2_raw : out STD_LOGIC;	1-bit	Out
clk3	Clock Connection	clk3 : out STD_LOGIC;	1-bit	Out
clk3_2x	Clock Connection	clk3_2x : out STD_LOGIC;	1-bit	Out
clk3_raw	Clock Connection	clk3_raw : out STD_LOGIC;	1-bit	Out
reset	Reset Connection	reset : out STD_LOGIC;	1-bit	Out
reset_raw	Reset Connection	reset_raw : out STD_LOGIC;	1-bit	Out
SRAM0 Core Clocks	User Connection	sram0_idlyctrl_rdy : out STD_LOGIC;	1-bit	Out
		sram0_clk0 : out STD_LOGIC;	1-bit	Out
		sram0_clk90 : out STD_LOGIC;	1-bit	Out
		sram0_clk50 : out STD_LOGIC;	1-bit	Out

Table 84: H100 PCI-X Module Clocks Driver Signals

Group	Type	Signal Description	Width	Direction
SRAM1 Core Clocks	User Connection	sram1_idlyctrl_rdy : out STD_LOGIC;	1-bit	Out
		sram1_clk0 : out STD_LOGIC;	1-bit	Out
		sram1_clk90 : out STD_LOGIC;	1-bit	Out
		sram1_clk50 : out STD_LOGIC;	1-bit	Out
SRAM2 Core Clocks	User Connection	sram2_idlyctrl_rdy : out STD_LOGIC;	1-bit	Out
		sram2_clk0 : out STD_LOGIC;	1-bit	Out
		sram2_clk90 : out STD_LOGIC;	1-bit	Out
		sram2_clk50 : out STD_LOGIC;	1-bit	Out
SRAM3 Core Clocks	User Connection	sram3_idlyctrl_rdy : out STD_LOGIC;	1-bit	Out
		sram3_clk0 : out STD_LOGIC;	1-bit	Out
		sram3_clk90 : out STD_LOGIC;	1-bit	Out
		sram3_clk50 : out STD_LOGIC;	1-bit	Out
SDRAM Core Clocks	User Connection	sdram_idlyctrl_rdy : out STD_LOGIC;	1-bit	Out
		sdram_clk0 : out STD_LOGIC;	1-bit	Out
		sdram_clk90 : out STD_LOGIC;	1-bit	Out
		sdram_clk50 : out STD_LOGIC;	1-bit	Out
		sdram_refresh_clk : out STD_LOGIC;	1-bit	Out
PCIxInterfaceClocks	User Connection	host_clk0 : out STD_LOGIC;	1-bit	Out
		host_idlyctrl_rdy : out STD_LOGIC;	1-bit	Out
UserPort	User Connection	lock : out STD_LOGIC;	1-bit	Out
UdimeBridgeIngressClocks	User Connection	ub01_link_clk_i : in STD_LOGIC;	1-bit	
		ub23_link_clk_i : in STD_LOGIC;	1-bit	
UdimeBridgeEgressClocks	User Connection	ub01_link_clk_o : out STD_LOGIC_VECTOR(1 downto 0);	2-bits	
		ub23_link_clk_o : out STD_LOGIC;	1-bit	
UdimeBridge0 Clocks	User Connection	ub0_src_clk : out STD_LOGIC;	1-bit	
		ub0_clk0 : out STD_LOGIC;	1-bit	
		ub0_clk90 : out STD_LOGIC;	1-bit	
UdimeBridge1 Clocks	User Connection	ub1_src_clk : out STD_LOGIC;	1-bit	
		ub1_clk0 : out STD_LOGIC;	1-bit	
		ub1_clk90 : out STD_LOGIC;	1-bit	
UdimeBridge2 Clocks	User Connection	ub2_src_clk : out STD_LOGIC;	1-bit	
		ub2_clk0 : out STD_LOGIC;	1-bit	
		ub2_clk90 : out STD_LOGIC;	1-bit	
UdimeBridge3 Clocks	User Connection	ub3_src_clk : out STD_LOGIC;	1-bit	
		ub3_clk0 : out STD_LOGIC;	1-bit	
		ub3_clk90 : out STD_LOGIC;	1-bit	

Table 84: H100 PCI-X Module Clocks Driver Signals

Group	Type	Signal Description	Width	Direction
UdimeBridgeIngressClocks	User Connection	ub_reset : out STD_LOGIC;	1-bit	

Table 84: H100 PCI-X Module Clocks Driver Signals

Support Files

The support file names and devices used are listed in Table 85.

Location	Device usage
DIMEtalk\library\System\h100_clocks.support\h100_clocks.vhd	All devices

Table 85: H100 PCI-X Module Clocks Driver Support Files

1.7 DIME-C

DIME-C Link FIFO

Functional Description

This is a FIFO that allows users to pass 32-bit data between DIME-C components. The DIME-C Link FIFO is shown in Figure 48.

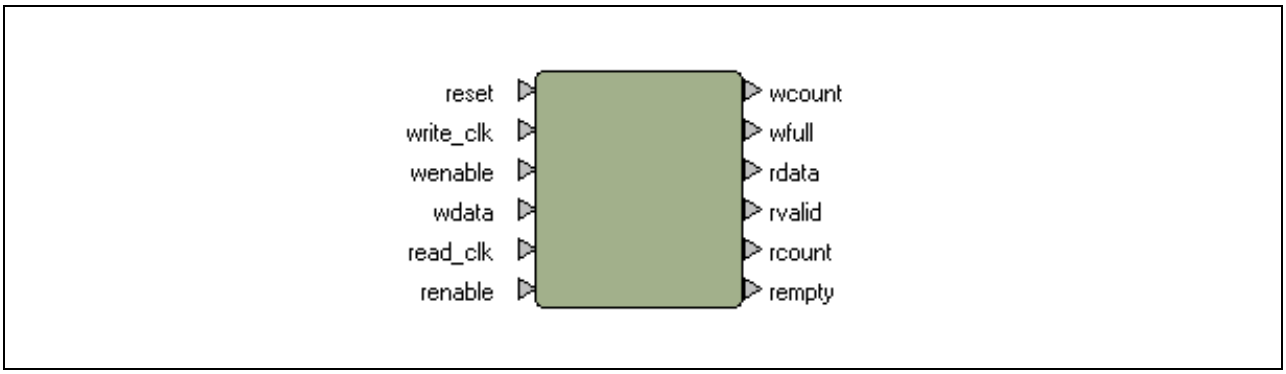

Figure 48: DIME-C Link FIFO Component

Using a DIME-C Link FIFO Component

The Link FIFO is capable of linking channels and pipes directly between different DIME-C blocks, and therefore is typically used to link DIME-C components together. Data is written in by one DIME-C component and read out of the FIFO by another DIME-C component.

Signals

The signal descriptions are shown in Table 86.

Group	Type	Signal Description	Signal Width	Direction
reset	Reset Connection	reset : in STD_LOGIC;	1-bit	In

Table 86: DIME-C Link FIFO Signals

Group	Type	Signal Description	Signal Width	Direction
WritePort	User Connection	write_clk : in STD_LOGIC;	1-bit	In
		wenable : in STD_LOGIC;	1-bit	In
		wdata : in STD_LOGIC_VECTOR(31 downto 0);	32-bits	In
		wcount : out STD_LOGIC_VECTOR(31 downto 0);	32-bits	In
		wfull : out STD_LOGIC;	1-bit	In
ReadPort	User Connection	read_clk : in STD_LOGIC;	1-bit	Out
		renable : in STD_LOGIC;	1-bit	Out
		rdata : out STD_LOGIC_VECTOR(31 downto 0);	32-bits	Out
		rvalid : out STD_LOGIC;	1-bit	Out
		rcount : out STD_LOGIC_VECTOR(31 downto 0);	32-bits	Out
		rempty : out STD_LOGIC;	1-bit	Out

Table 86: DIME-C Link FIFO Signals

Support Files

The support file names and devices used are listed in Table 87.

Location	Device usage
DIMEtalk\library\DIME-C\link_fifo.support\All\link_fifo.vhd	All devices
DIMEtalk\Common\bretime.vhd	All devices
DIMEtalk\Common\blk_async_fifo.vhd	All devices
DIMEtalk\Common\blk_dpram.vhd	All devices
DIMEtalk\Common\retime.vhd	All devices
DIMEtalk\Common\pkg_dimetalk_global.vhd	All devices

Table 87: DIME-C Link FIFO Support Files

User Generics

The user generics are shown in Table 88.

Name	Type	Description
fifo_pwidth	natural	This indicates the width of the FIFO pointer which sets the depth of the FIFO, n infers a depth of 2**n, for example 9 infers a depth of 2**9 = 512 x 32 bit or 12 infers a depth of 2**12 = 4096 x 32 bit.

Table 88: DIME-C Link FIFO User Generics

Component Definition

```
entity link_fifo is
generic(
fifo_pwidth : natural: = 9
```

```
);
port (
        reset                   : in  std_logic;
        -- Write Port
        write_clk                : in  std_logic;
        wenable                  : in  std_logic;
        wdata                    : in std_logic_vector(DT_DATA_SZ-1 downto 0);
        wcount                   : out std_logic_vector(USR_FCNT_SZ-1 downto 0);
        wfull                   : out std_logic;
        -- Read Port
        read_clk                 : in  std_logic;
        renable                  : in  std_logic;
        rdata                   : out std_logic_vector(DT_DATA_SZ-1 downto 0);
        rvalid                  : out std_logic;
        rcount                   : out std_logic_vector(USR_FCNT_SZ-1 downto 0);
        rempty                   : out std_logic
);
end entity link_fifo;
```

DIME-C External MGT LinkFIFO (dimec_ddr2mgtlinkfifo)

Functional Description

The DIME-C external DDR to MGT LinkFIFO component, shown in Figure 49, is intended to be used for connecting DIME-C processes together through the high speed serial channels available on the H100 product range. These are intended for point to point links. The links need to send the data in packets and so certain additional parameters are included to provide enough information for the LinkFIFO component to create appropriate packet sizes. Note also

that when sending a packet the LinkFIFO component will also add a DIMEtalk transport header to the start of each packet.

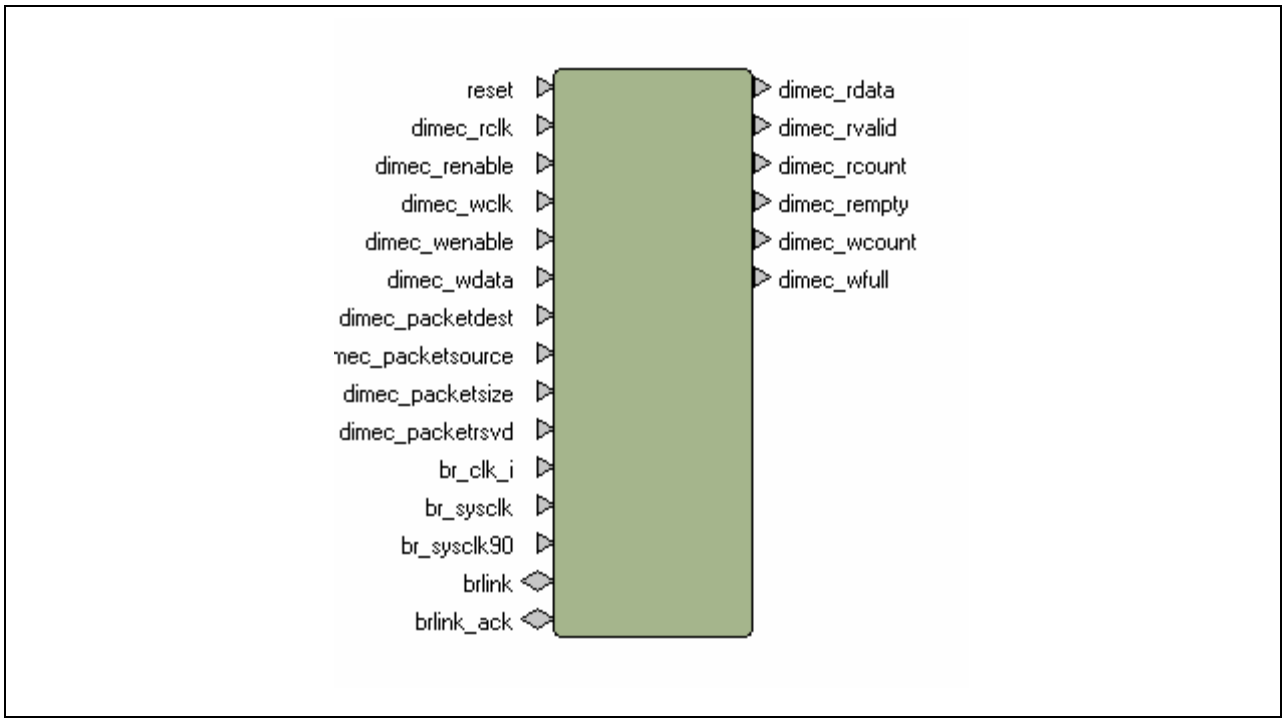

Figure 49: DIME-C External MGT LinkFIFO Component

Signals

The signal descriptions are shown in Table 89.

Group	Type	Signal Description	Signal Width	Direction
sysreset	Reset Connection	reset	1-bit	In
ingress_egress_fifo	User Connection	dimec_rclk : in STD_LOGIC;	1-bit	
		dimec_renable : in STD_LOGIC;	1-bit	
		dimec_rdata : out STD_LOGIC_VECTOR(31 downto 0);	32-bits	
		dimec_rvalid : out STD_LOGIC;	1-bit	
		dimec_rcount : out STD_LOGIC_VECTOR(31 downto 0);	32-bits	
		dimec_rempty : out STD_LOGIC;	1-bit	
		dimec_wclk : in STD_LOGIC;	1-bit	
		dimec_wenable : in STD_LOGIC;	1-bit	
		dimec_wdata : in STD_LOGIC_VECTOR(31 downto 0);	32-bits	

Table 89: DIME-C External MGT LinkFIFO Signals

Group	Type	Signal Description	Signal Width	Direction
		dimec_wcount : out STD_LOGIC_VECTOR(31 downto 0);	32-bits	
		dimec_wfull : out STD_LOGIC;	1-bit	
		dimec_packetdest : in STD_LOGIC_VECTOR(7 downto 0);	8-bits	
		dimec_packetsource : in STD_LOGIC_VECTOR(7 downto 0);	8-bits	
		dimec_packetsize : in STD_LOGIC_VECTOR(7 downto 0);	8-bits	
		dimec_packetrsvd : in STD_LOGIC_VECTOR(7 downto 0);	8-bits	
UdimeBridgeClocks	User Connection	br_clk_i : in STD_LOGIC;	1-bit	
		br_sysclk : in STD_LOGIC;	1-bit	
		br_sysclk90 : in STD_LOGIC;	1-bit	
BridgeLink	User Connection	brlink : inout STD_LOGIC_VECTOR(15 downto 0);	16-bits	
		brlink_ack : inout STD_LOGIC_VECTOR(1 downto 0);	2-bits	

Table 89: DIME-C External MGT LinkFIFO Signals

Support Files

The support file names and devices used are listed in Table 90.

Location	Device Usage
DIMEtalk\library\DIME-C\dimec_ddr2mgtlinkfifo.support\dimec_ddr2mgtlinkfifo.vhd	All
DIMEtalk\library\DIME-C\dimec_ddr2mgtlinkfifo.support\dimec_ddr_async_bridge16.vhd	All
DIMEtalk\library\DIME-C\dimec_ddr2mgtlinkfifo.support\bridge_fifo_511x32.ngc	All
DIMEtalk\common\bretime.vhd	All
DIMEtalk\common\retime.vhd	All
DIMEtalk\library\DIME-C\dimec_ddr2mgtlinkfifo.support\dimec_async_bridge_wr16.vhd	All
DIMEtalk\library\DIME-C\dimec_ddr2mgtlinkfifo.support\dimec_async_bridge_rd16.vhd	All

Table 90: DIME-C External MGT LinkFIFO Support Files

User Generics

The user generics are shown in Table 91.

Name	Type	Value	Read only
setasplug	boolean	true	No
VIRTEXE	boolean	false	Yes
linkwidth	integer	16	Yes
channel	integer	0	No

Table 91: DIME-C External MGT LinkFIFO User Generics

Please note that by default the channel parameter is set to 0. When placing this component the channel parameter must be changed to match the required MGT port. The channels are numbered 0,1,2 and 3.

DIME-C Internal LinkFIFO (dclink_fifo)

Functional Description

The DIME-C Internal LinkFIFO component, shown in Figure 50, is intended to be used for connecting DIME-C processes together internal to the User FPGA on the H100 hardware. Although a general LinkFIFO already exists for DIME-C, this component has been created to mimic the external MGT LinkFIFO connections to DIME-C. This is advanageouos for the user when creating a processing block with the dclink data type as it can easily be connected internally or externally without having to recompile the DIME-C process. This may be of particular use when creating library functions. Note that this component has DIMEtalk control ports such as packet_size but these are dummy ports in order to allow simple connection to the dclink datatype in DIME-C. The actual component is simply two 32-bit FIFOs with a depth that can be set (default is 511 words).

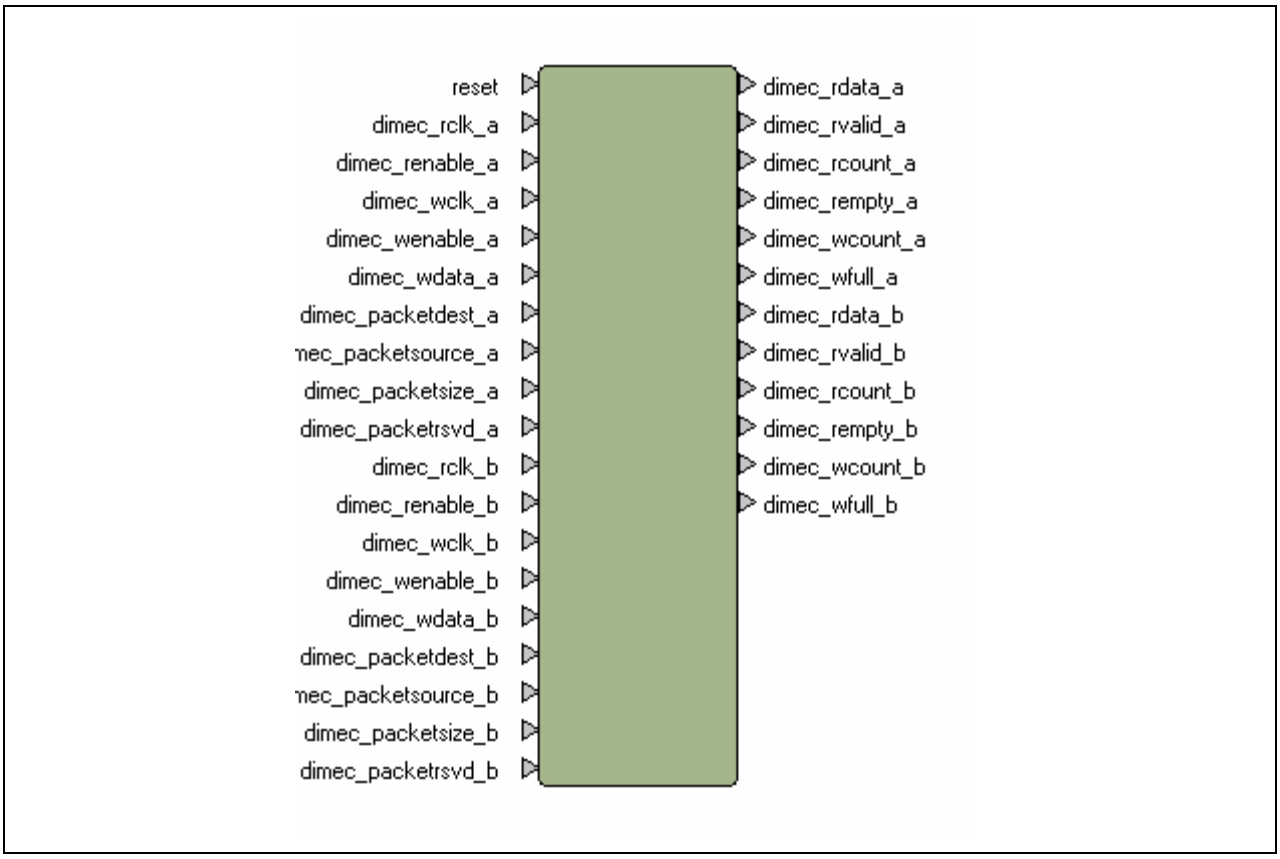

Figure 50: DIME-C Internal LinkFIFO Component

Signals

The signal descriptions are shown in Table 92.

Group	Type	Signal Description	Signal Width	Direction
reset	Reset Connection	DIMEtalk Reset	1-bit	In
ingress_egress_fifo_channela	User Connection	dimec_rclk_a : in STD_LOGIC;	1-bit	In
		dimec_renable_a : in STD_LOGIC;	1-bit	IN
		dimec_rdata_a : out STD_LOGIC_VECTOR(31 downto 0);	32-bits	Out
		dimec_rvalid_a : out STD_LOGIC;	1-bit	Out
		dimec_rcount_a : out STD_LOGIC_VECTOR(31 downto 0);	32-bits	Out
		dimec_rempty_a : out STD_LOGIC;	1-bit	Out
		dimec_wclk_a : in STD_LOGIC;	1-bit	In

Table 92: DIME-C Internal LinkFIFO Signal Descriptions

Group	Type	Signal Description	Signal Width	Direction
		dimec_wenable_a : in STD_LOGIC;	1-bit	In
		dimec_wdata_a : in STD_LOGIC_VECTOR(31 downto 0);	32-bits	In
		dimec_wcount_a : out STD_LOGIC_VECTOR(31 downto 0);	32-bits	Out
		dimec_wfull_a : out STD_LOGIC;	1-bit	Out
		dimec_packetdest_a : in STD_LOGIC_VECTOR(7 downto 0);	8-bits	In
		dimec_packetsource_a : in STD_LOGIC_VECTOR(7 downto 0);	8-bits	In
		dimec_packetsize_a : in STD_LOGIC_VECTOR(7 downto 0);	8-bits	In
		dimec_packetrsvd_a : in STD_LOGIC_VECTOR(7 downto 0);	8-bits	In
ingress_egress_fifo_channelb	User Connection	dimec_rclk_b : in STD_LOGIC;	1-bit	In
		dimec_renable_b : in STD_LOGIC;	1-bit	In
		dimec_rdata_b : out STD_LOGIC_VECTOR(31 downto 0);	32-bits	Out
		dimec_rvalid_b : out STD_LOGIC;	1-bit	Out
		dimec_rcount_b : out STD_LOGIC_VECTOR(31 downto 0);	32-bits	Out
		dimec_rempty_b : out STD_LOGIC;	1-bit	Out
		dimec_wclk_b : in STD_LOGIC;	1-bit	In
		dimec_wenable_b : in STD_LOGIC;	1-bit	In
		dimec_wdata_b : in STD_LOGIC_VECTOR(31 downto 0);	32-bits	In
		dimec_wcount_b : out STD_LOGIC_VECTOR(31 downto 0);	32-bitss	Out
		dimec_wfull_b : out STD_LOGIC;	1-bit	Out

Table 92: DIME-C Internal LinkFIFO Signal Descriptions

Group	Type	Signal Description	Signal Width	Direction
		dimec_packetdest_b : in STD_LOGIC_VECTOR(7 downto 0);	8-bits	In
		dimec_packetsource_b : in STD_LOGIC_VECTOR(7 downto 0);	8-bits	In
		dimec_packetsize_b : in STD_LOGIC_VECTOR(7 downto 0);	8-bits	In
		dimec_packetrsvd_b : in STD_LOGIC_VECTOR(7 downto 0);	8-bits	In

Table 92: DIME-C Internal LinkFIFO Signal Descriptions

User Generics

The following generic shown in Table 93 is used to set the depth of the FIFOs in the link component. The default is 9 which creates FIFOs that are 511 words deep. Note that the component uses two fifos - one for each direction. This equates to two block RAMs used if the depth of each block RAM is set to 511.

Name	Type	Value	Readonly
fifo_pwidth	integer	9	No

Table 93: DIME-C Internal LinkFIFO User Generics

Support Files

The support file names and devices used are listed in Table 94.

Location	Device Usage
DIMEtalk\library\DIME-C\dclink_fifo.support\All\dclink_fifo.vhd	All
DIMEtalk\common\bretime.vhd	All
DIMEtalk\common\blk_async_fifo.vhd	All
DIMEtalk\common\blk_dpram.vhd	All
DIMEtalk\common\retime.vhd	All
DIMEtalk\library\common\pkg_dimetalk_global.vhd	All

Table 94: DIME-C Internal LinkFIFO Support Files

1.8 Virtex-4 DDR2 Memory

DDR2 Memory Clocks Module

This component is not required by users of the H100 series hardware as the clocks for the on-board SRAM memory are integrated into the main H100 System Clock component.

Functional Description

This module is identical to that which is documented in the *BenDATA-V4 Reference Guide* and is summarized here for convenience. This module provides all the clocks required by the DDR2 SDRAM and DDRII SRAM cores. These include a phase shifted version of the 250MHz clock, divided versions of this clock and a 200MHz I-delay calibration clock. The DDR2 memory clock module is shown in Figure 48.

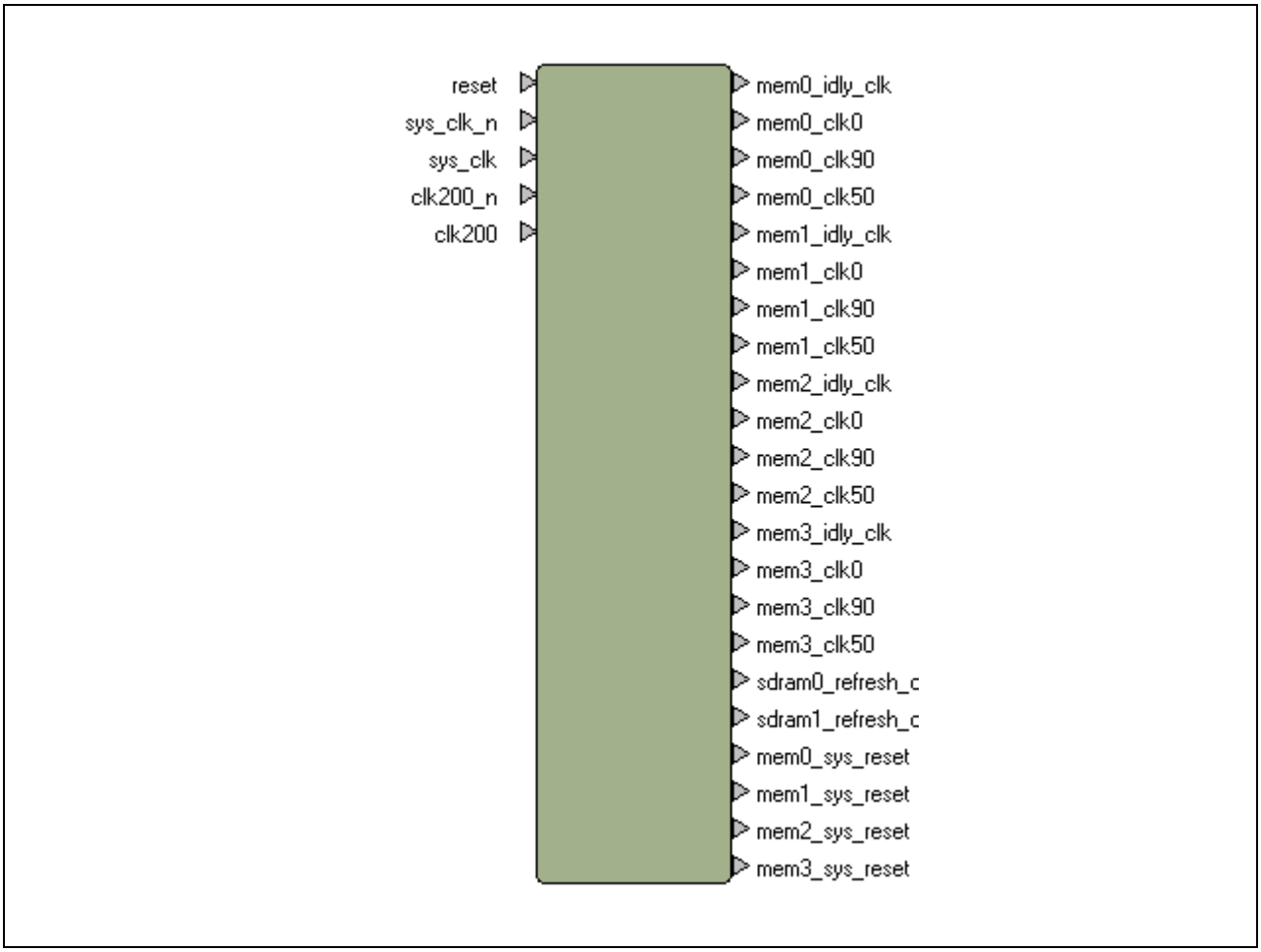

Figure 51: DDR2 Memory Clocks Module Component

User Generics

The user generics are shown in Table 95.

Name	Type	Description
Use_with_DDRII_SDRAM	true or false	Set to false by default, this generic must be set to true if the component is used in conjunction with the BenDATA-V4 SDRAM Memory Node. It is not used in the VHDL but should be set up on a core basis via the GUI by right-clicking on the component instance and editing the component parameter. This allows the software to allocate the correct constraints.

Table 95: DDR2 Memory Clocks Module User Generics

Signals

The signal descriptions are shown in Table 96.

Group	Type	Signal Description	Signal Width	Direction
reset	Reset Connection	reset : in STD_LOGIC;	1-bit	In
DDR2InputClocks	User Connection	sys_clk_n : in STD_LOGIC	1-bit	In
		sys_clk : in STD_LOGIC	1-bit	In
		clk200_n : in STD_LOGIC	1-bit	In
		clk200 : in STD_LOGIC	1-bit	In
DDR2CoreClocks0	User Connection	mem0_idly_clk : out STD_LOGIC;	1-bit	Out
		mem0_clk0 : out STD_LOGIC;	1-bit	Out
		mem0_clk90 : out STD_LOGIC;	1-bit	Out
		mem0_clk50 : out STD_LOGIC;	1-bit	Out
DDR2CoreClocks1	User Connection	mem1_idly_clk : out STD_LOGIC;	1-bit	Out
		mem1_clk0 : out STD_LOGIC;	1-bit	Out
		mem1_clk90 : out STD_LOGIC;	1-bit	Out
		mem1_clk50 : out STD_LOGIC;	1-bit	Out
DDR2CoreClocks2	User Connection	mem2_idly_clk : out STD_LOGIC;	1-bit	Out
		mem2_clk0 : out STD_LOGIC;	1-bit	Out
		mem2_clk90 : out STD_LOGIC;	1-bit	Out
		mem2_clk50 : out STD_LOGIC;	1-bit	Out
DDR2CoreClocks3	User Connection	mem3_idly_clk : out STD_LOGIC;	1-bit	Out
		mem3_clk0 : out STD_LOGIC;	1-bit	Out
		mem3_clk90 : out STD_LOGIC;	1-bit	Out
		mem3_clk50 : out STD_LOGIC;	1-bit	Out
SDRAMRefreshClock0	User Connection	sdram0_refresh_clk : out STD_LOGIC;	1-bit	Out
SDRAMRefreshClock1	User Connection	sdram1_refresh_clk : out STD_LOGIC;	1-bit	Out
dcmlockreset0	User Connection	mem0_sys_reset : out STD_LOGIC;	1-bit	Out
dcmlockreset1	User Connection	mem1_sys_reset : out STD_LOGIC;	1-bit	Out

Table 96: DDR2 Memory Clocks Module Signals

Group	Type	Signal Description	Signal Width	Direction
dcmlockreset2	User Connection	mem2_sys_reset : out STD_LOGIC;	1-bit	Out
dcmlockreset3	User Connection	mem3_sys_reset : out STD_LOGIC;	1-bit	Out

Table 96: DDR2 Memory Clocks Module Signals

Support Files

The support file names and devices used are listed in Table 97.

Location	Device usage
DIMEtalk\library\V4 nodes\ddr2_clocks.support\ddr2_clocks.vhd	All devices

Table 97: DDR2 Memory Clocks Module Support Files

DDR-II SRAM Memory Nodes

The following description applies to all the DDR-II SRAM nodes (designed for Virtex-4) in DIMEtalk. There are currently separate nodes for different hardware products (i.e. BenDATA-V4, BenBLUE-V4, etc). A list of top levels is detailed in Table 98. Some differences exist between nodes and these are due to subtle differences in SRAM configuration or differences in IDELAY_CTRL requirements. For further information on the DDR-II SRAM controller core refer to the *DDR-II SRAM Controller Core section* in the relevant *Reference Guide*.

Top Level Name	Module	Differences
ddr2sram_32_pci104_v4	BenNUEY-PCI-104-V4	2 ICs provide extra memory depth (mem_number=2) 2 IDELAY_CTRLS required (idelayctrl_num=2) 1 Bank of SRAM supported (Bank A)
ddr2sram_32_pcix	BenNUEY-PCI-X	2 ICs provide extra memory depth (mem_number=2) 3 IDELAY_CTRLS required (idelayctrl_num=3) 1 Bank of SRAM supported (Bank A)
ddr2sram_32_benbluev4_pri	BenBLUE-V4 Primary FPGA	2 IDELAY_CTRLS required (idelayctrl_num=2) 4 Banks of SRAM supported (Banks A-D)
ddr2sram_32_benbluev4_sec	BenBLUE-V4 Secondary FPGA	4 IDELAY_CTRLS required (idelayctrl_num=3) 4 Banks of SRAM supported (Banks E-H)
ddr2sram_32_bendatav4	BenDATA-V4, BenIO-V4, BenADC-V4	2 IDELAY_CTRLS required (idelayctrl_num=2) 2 Banks of SRAM supported (Banks A-B)
ddr2sram_if_h100	H100	4 banks of SRAM supported (no bank or idelayctrl generic required).

Table 98: Supported DDR-II Top Levels

Functional Description

These nodes implement an interface to DDR-II SRAM and contain both a DIMEtalk interface and a user interface, which provide full read and write access to the memory. The DIMEtalk interface runs directly from the global DIMEtalk network clock, however the user interface must run from the same clock that is clocking the interface core itself (normally a 200MHz clock). For H100, the interface clocks are driven by the H100 clocks module, all other products use the DDR2 clocks module as described in "DDR2 Memory Clocks Module".

The user interface always receives priority over the DIMEtalk interface. Should the user interrupt mid-transfer, the DIMEtalk interface stops the transfer and starts the packet over. With the exception of an extra signal (user_request)

which gives the user control, the user signals operate in the same fashion as those on the main DDR-II SRAM controller core. The DDR-II SRAM memory node is shown in Figure 52.

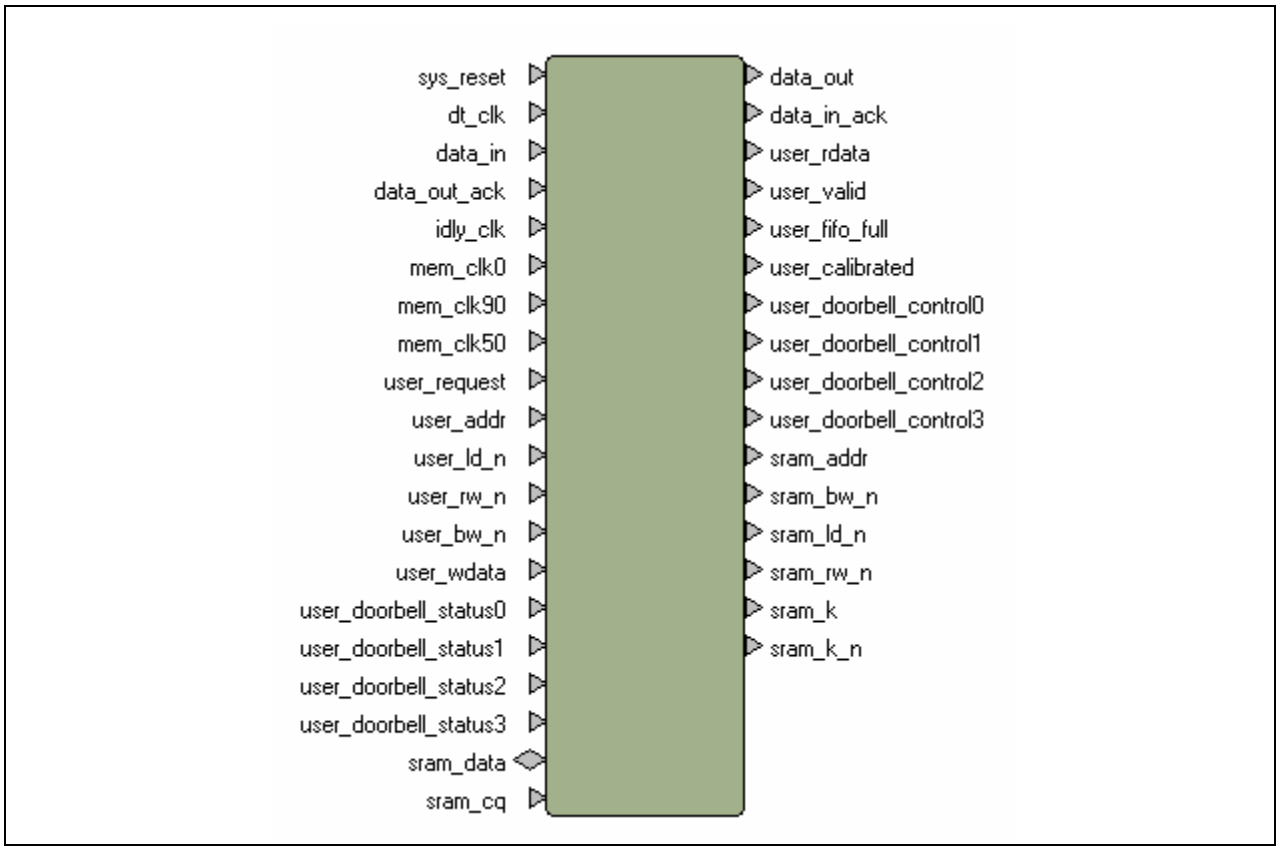

Figure 52: DDR-II SRAM Memory Node

Additional Modules

The module must be coupled with the DDR2_clocks component in order to provide the required clocks for the SRAM interface.

User Generics

The user generics are shown in Table 99.

Name	Type	Description
$bank	Banka/Bankb/Bankc/ etc…	Not used in the VHDL however needs to be set up on a core basis via the GUI by right-clicking on the component instance and editing this component parameter which then allows the software to allocate the correct constraints.

Table 99: DDR-II SRAM Memory Node User Generics

Extended Doorbell

There are four doorbell status inputs and four doorbell control outputs. The status inputs can be used to trap a change of state on an input whilst the control outputs can be used as a trigger for aspects of a user's design. For more information on packets see "Request Packet Type 10 (Doorbell class)".

Signals

The signal descriptions are provided in Table 100.

Group	Type	Signal Description	Signal Width	Direction
dcmlockreset	User Connection	sys_reset : in STD_LOGIC;	1-bit	In
dt_clk	Clock Connection	dt_clk : in STD_LOGIC;	1-bit	In
DIMEtalk	DIMEtalk Connection	data_in : in STD_LOGIC_VECTOR(31 downto 0);	32-bits	In
		data_out : out STD_LOGIC_VECTOR(31 downto 0);	32-bits	Out
		data_in_ack : out STD_LOGIC;	1-bit	Out
		data_out_ack : in STD_LOGIC;	1-bit	In
User Signals	User Connection	idly_clk : in STD_LOGIC;	1-bit	In
		mem_clk0 : in STD_LOGIC;	1-bit	In
		mem_clk90 : in STD_LOGIC;	1-bit	In
		mem_clk50 : in STD_LOGIC;	1-bit	In
UserInterface	User Connection	user_request : in STD_LOGIC;	1-bit	In
		user_addr : in STD_LOGIC_VECTOR(21 downto 0);	22-bits	In
		user_ld_n : in STD_LOGIC;	1-bit	In
		user_rw_n : in STD_LOGIC;	1-bit	In
		user_bw_n : in STD_LOGIC_VECTOR(7 downto 0);	8-bits	In
		user_wdata : in STD_LOGIC_VECTOR(63 downto 0);	64-bits	In
		user_rdata : out STD_LOGIC_VECTOR(63 downto 0);	64-bits	Out
		user_valid : out STD_LOGIC;	1-bit	Out
		user_fifo_full : out STD_LOGIC;	1-bit	Out
		user_calibrated : out STD_LOGIC;	1-bit	Out
Doorbells	User Connection	user_doorbell_status0 : in STD_LOGIC;	1-bit	In
		user_doorbell_status1 : in STD_LOGIC;	1-bit	In
		user_doorbell_status2 : in STD_LOGIC;	1-bit	In
		user_doorbell_status3 : in STD_LOGIC;	1-bit	In
		user_doorbell_control0 : out STD_LOGIC;	1-bit	Out
		user_doorbell_control1 : out STD_LOGIC;	1-bit	Out
		user_doorbell_control2 : out STD_LOGIC;	1-bit	Out
		user_doorbell_control3 : out STD_LOGIC;	1-bit	Out

Table 100: DDR-II SRAM Memory Node Signal Descriptions

Group	Type	Signal Description	Signal Width	Direction
DDR2SRAMInterface	User Connection	sram_addr : out STD_LOGIC_VECTOR(21 downto 0);	22-bits	Out
		sram_data : inout STD_LOGIC_VECTOR(31 downto 0);	32-bits	
		sram_bw_n : out STD_LOGIC_VECTOR(3 downto 0);	4-bits	Out
		sram_ld_n : out STD_LOGIC;	1-bit	Out
		sram_rw_n : out STD_LOGIC;	1-bit	Out
		sram_k : out STD_LOGIC;	1-bit	Out
		sram_k_n : out STD_LOGIC;	1-bit	Out
		sram_cq : in STD_LOGIC;	1-bit	In

Table 100: DDR-II SRAM Memory Node Signal Descriptions

Support Files - Common Between All Implementations

The support file names and devices used are listed in Table 101.

Location	Device Usage
DIMEtalk\library\DDR2 SRAM\ddr2sram.support\Calibration_H100.vhd	H100 only
DIMEtalk\library\DDR2 SRAM\ddr2sram.support\Idelay_calibrate_H100.vhd	H100 only
DIMEtalk\library\DDR2 SRAM\ddr2sram.support\Sram_data_iobs_H100.vhd	H100 only
DIMEtalk\library\DDR2 SRAM\ddr2sram.support\Sram_data_path_H100.vhd	H100 only
DIMEtalk\library\DDR2 SRAM\ddr2sram.support\Calibration_bist_pc104v4.vhd	BenNUEY-PCI-104-V4 only
DIMEtalk\library\DDR2 SRAM\ddr2sram.support\Idelay_calibrate_pc104v4.vhd	BenNUEY-PCI-104-V4 only
DIMEtalk\library\DDR2 SRAM\ddr2sram.support\Resync_fall.vhd	BenNUEY-PCI-104-V4 and H100 only
DIMEtalk\library\DDR2 SRAM\ddr2sram.support\Sram_data_iobs_pc104.vhd	BenNUEY-PCI-104-V4 only
DIMEtalk\library\DDR2 SRAM\ddr2sram.support\Sram_data_path_pc104v4.vhd	BenNUEY-PCI-104-V4 only
DIMEtalk\library\DDR2 SRAM\ddr2sram.support\Wide_prbs_gen	BenNUEY-PCI-104-V4 and H100 only
DIMEtalk\library\DDR2 SRAM\ddr2sram.support\sram_data_path.vhd	All except BenNUEY-PCI-104-V4
DIMEtalk\common\pkg_dimetalk_global.vhd	All
DIMEtalk\library\DDR2 SRAM\ddr2sram.support\ddr2_sram_async_fifo.vhd	All
DIMEtalk\library\DDR2 SRAM\ddr2sram.support\ddr2_sram_controller_core.vhd	All
DIMEtalk\library\DDR2 SRAM\ddr2sram.support\ddr2sram_if_32_v4modules.vhd	All
DIMEtalk\library\DDR2 SRAM\ddr2sram.support\fifo_ram.vhd	All

Table 101: DDR-II SRAM Memory Node Support Files - Common Between Implementations

Location	Device Usage
DIMEtalk\library\DDR2 SRAM\ddr2sram.support\ddr2_sram_fifo.ngc	All
DIMEtalk\library\DDR2 SRAM\ddr2sram.support\idelay_calibrate.vhd	All except BenBLUE-V4 and BenNUEY-PCI-104-V4
DIMEtalk\library\DDR2 SRAM\ddr2sram.support\idelaycontrol.vhd	All
DIMEtalk\library\DDR2 SRAM\ddr2sram.support\pkg_asyncfifo.vhd	All
DIMEtalk\library\DDR2 SRAM\ddr2sram.support\sram_clocks.vhd	All
DIMEtalk\library\DDR2 SRAM\ddr2sram.support\sram_ctrl_iobs.vhd	All
DIMEtalk\library\DDR2 SRAM\ddr2sram.support\sram_ctrl_sm.vhd	All
DIMEtalk\library\DDR2 SRAM\ddr2sram.support\sram_data_front.vhd	All
DIMEtalk\library\DDR2 SRAM\ddr2sram.support\sram_data_iobs.vhd	All except BenNUEY-PCI-104-V4
DIMEtalk\common\retime.vhd	All
DIMEtalk\common\dtnode_slave_control.vhd	All
DIMEtalk\common\resync_rise.vhd	All
DIMEtalk\library\DDR2 SRAM\ddr2sram.support\idelay_calibrate_pri.vhd	BenBLUE-V4 Primary FPGA only
DIMEtalk\library\DDR2 SRAM\ddr2sram.support\idelay_calibrate_sec.vhd	BenBLUE-V4 Secondary FPGA only

Table 101: DDR-II SRAM Memory Node Support Files - Common Between Implementations

Support Files - Individual for Each Implementation

Table 102 lists the top-level VHDL file that will be provided (for each individual implementation) along with the support files listed in Table 101.

Location	Device Usage
DIMEtalk\library\DDR2 SRAM\ddr2sram.support\ddr2sram_if_32_H100.vhd	H100
DIMEtalk\library\DDR2 SRAM\ddr2sram.support\ddr2sram_if_32_bennuey_pci104v4.vhd	BenNUEY-PCI-104-V4
DIMEtalk\library\DDR2 SRAM\ddr2sram.support\ddr2sram_if_32_bennuey_pcix.vhd	BenNUEY-PCI-X-V4
DIMEtalk\library\DDR2 SRAM\ddr2sram.support\ddr2sram_if_32_benblue_v4_primary.vhd	BenBLUE-V4 Primary FPGA only
DIMEtalk\library\DDR2 SRAM\ddr2sram.support\ddr2sram_if_32_benblue_v4_secondary.vhd	BenBLUE-V4 Secondary FPGA only
DIMEtalk\library\DDR2 SRAM\ddr2sram.support\ddr2sram_if_32_bendata_v4.vhd	BenDATA-V4

Table 102: DDR-II SRAM Memory Node Support Files - Individual for Each Implementation

Component Definitions

The main component body (port names) are the same for all implementations of the DDR-II SRAM node. Only the generic values change for each configuration.

component (appropriate name i.e. ddr2sram_32_pcix)

```
port (
    sys_reset : in  std_logic;
    -- DimeTalk port                 ---------------------------------------------
    dt_clk : in  std_logic;
    data_in : in  std_logic_vector(DT_DATA_SZ-1 downto 0);
    data_out : out std_logic_vector(DT_DATA_SZ-1 downto 0);
    data_in_ack : out std_logic;
    data_out_ack : in  std_logic;
    -- User Port                     ------------------------------------------------
    idly_clk  : in  std_logic;
    mem_clk0 : in  std_logic;
    mem_clk90 : in  std_logic;
    mem_clk50 : in  std_logic;
    user_request : in  std_logic;
    user_addr : in  std_logic_vector(mem_awidth+(mem_number/2)-1 downto 0);
    user_ld_n : in  std_logic;
    user_rw_n : in  std_logic;
    user_bw_n : in  std_logic_vector((mem_bwidth*2)-1 downto 0);
    user_wdata : in  std_logic_vector((mem_dwidth*2)-1 downto 0);
    user_rdata : out std_logic_vector((mem_dwidth*2)-1 downto 0);
    user_valid : out std_logic;
    user_fifo_full : out std_logic;
    user_calibrated : out std_logic;
    user_doorbell_status0  : in  std_logic;
    user_doorbell_status1  : in  std_logic;
    user_doorbell_status2  : in  std_logic;
    user_doorbell_status3  : in  std_logic;
    user_doorbell_control0 : out std_logic;
    user_doorbell_control1 : out std_logic;
    user_doorbell_control2 : out std_logic;
    user_doorbell_control3 : out std_logic;
    -- External DDR-II SRAM I/O Port      ------------------------------------
    sram_addr  : out   std_logic_vector(mem_awidth-1 downto 0);
    sram_data  : inout std_logic_vector(mem_dwidth-1 downto 0);
    sram_bw_n   : out std_logic_vector(mem_bwidth-1 downto 0);
    sram_ld_n  : out   std_logic_vector(mem_number-1 downto 0);
    sram_rw_n  : out   std_logic;
```

```
    sram_k  : out   std_logic_vector(clock_number-1 downto 0);
    sram_k_n : out   std_logic_vector(clock_number-1 downto 0);
    sram_cq : in   std_logic
    );
end component;
```

Generic Settings

H100

```
entity ddr2sram_if_h100 is
  generic(
    node_ID     :    std_logic_vector(NODEID_SZ-1 downto 0)   := (others => '0');
    mem_awidth   :    integer range 0 to MAX_MAWIDTH          := 21;
    mem_dwidth   :    integer range MIN_MDWIDTH to MAX_MDWIDTH := 32;
    mem_bwidth   :    integer range 0 to 32          := 4;
    mem_number          :    integer range 1 to 4 := 1;
    clock_number :    integer range 1 to 4 := 1
    );
```

BenNUEY-PCI-104-V4

```
node_ID     :    std_logic_vector(NODEID_SZ-1 downto 0)   := (others => '0');
    mem_awidth   :    integer range 0 to MAX_MAWIDTH          := 21;
    mem_dwidth   :    integer range MIN_MDWIDTH to MAX_MDWIDTH := 32;
    mem_bwidth   :    integer range 0 to 32  := 4;
    mem_number   :    integer range 1 to 4 := 2;
    clock_number :    integer range 1 to 4 := 1;
    idelayctrl_number : integer := 2
```

BenNUEY-PCI-X-V4

```
  generic(
    node_ID : std_logic_vector(NODEID_SZ-1 downto 0)   := (others => '0');
    mem_awidth : integer range 0 to MAX_MAWIDTH          := 21;
    mem_dwidth : integer range MIN_MDWIDTH to MAX_MDWIDTH := 32;
    mem_bwidth : integer range 0 to 32:= 4;
    mem_number : integer range 1 to 4 := 2;
    clock_number : integer range 1 to 4 := 1;
```

```
  idelayctrl_number : integer := 3
  );
```

BenBLUE-V4 Primary

```
 generic(
  node_ID : std_logic_vector(NODEID_SZ-1 downto 0)   := (others => '0');
  mem_awidth : integer range 0 to MAX_MAWIDTH          := 22;
  mem_dwidth : integer range MIN_MDWIDTH to MAX_MDWIDTH := 32;
  mem_bwidth : integer range 0 to 32:= 4;
  mem_number : integer range 1 to 4 := 1;
  clock_number : integer range 1 to 4 := 1;
  idelayctrl_number : integer := 2
  );
```

BenBLUE-V4 Secondary

```
 generic(
  node_ID : std_logic_vector(NODEID_SZ-1 downto 0)   := (others => '0');
  mem_awidth : integer range 0 to MAX_MAWIDTH          := 22;
  mem_dwidth : integer range MIN_MDWIDTH to MAX_MDWIDTH := 32;
  mem_bwidth : integer range 0 to 32:= 4;
  mem_number : integer range 1 to 4 := 1;
  clock_number : integer range 1 to 4 := 1;
  idelayctrl_number : integer := 4
  );
```

BenDATA-V4, BenIO-V4 and BenADC-V4

```
 generic(
  node_ID : std_logic_vector(NODEID_SZ-1 downto 0)   := (others => '0');
  mem_awidth : integer range 0 to MAX_MAWIDTH          := 22;
  mem_dwidth : integer range MIN_MDWIDTH to MAX_MDWIDTH := 32;
  mem_bwidth : integer range 0 to 32:= 4;
  mem_number : integer range 1 to 4 := 1;
  clock_number : integer range 1 to 4 := 1;
```

idelayctrl_number : integer := 2

);

Waveforms

The following figures illustrate write and read accesses to the user port of the SRAM controller. A write access is shown in Figure 53 where user_request goes high at the start of the transfer and user_ld_n is deasserted to load a read/write command (user_rw_n - driven LOW for write). Valid user_addr, user_wdata and user_bw_n must appear on the same clock cycle as user_request, user_ld_n and user_rw_n.

The user_fifo_empty flag will go LOW until all the data is processed. It normally takes four clock cycles between the last write access and the user_fifo_empty_flag to clear. Should the user wish to wait for all write data to be processed prior to reading, this flag must be monitored and a read cycle should only begin after the flag has cleared.

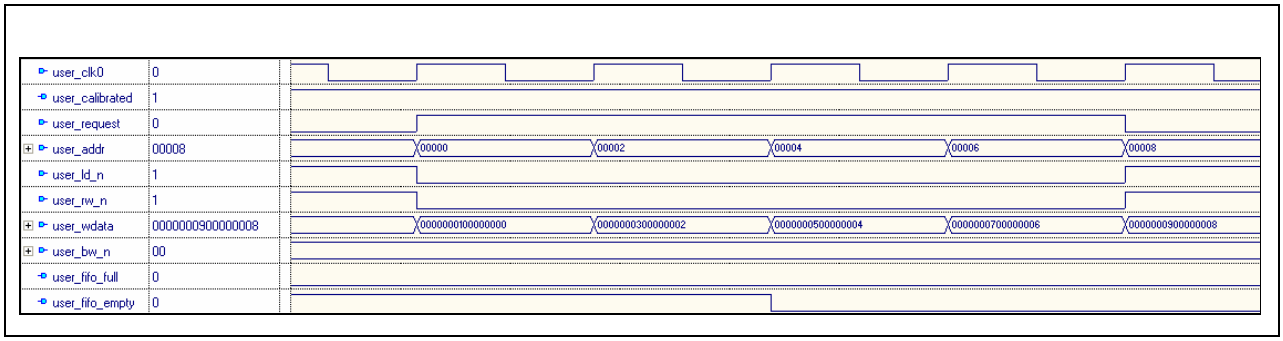

Figure 53: DDR-II SRAM Writes

To perform a read as shown in Figure 54, again the user_request signal must be asserted to make the transaction valid. For a read, user_ld_n must be deasserted whilst user_rw_n remains HIGH. user_addr must be valid on the same clock edge as user_request, user_ld_n and user_rw_n. When the data returns, user_valid indicates valid data. Note that user_fifo_empty is HIGH prior to the read command being sent.

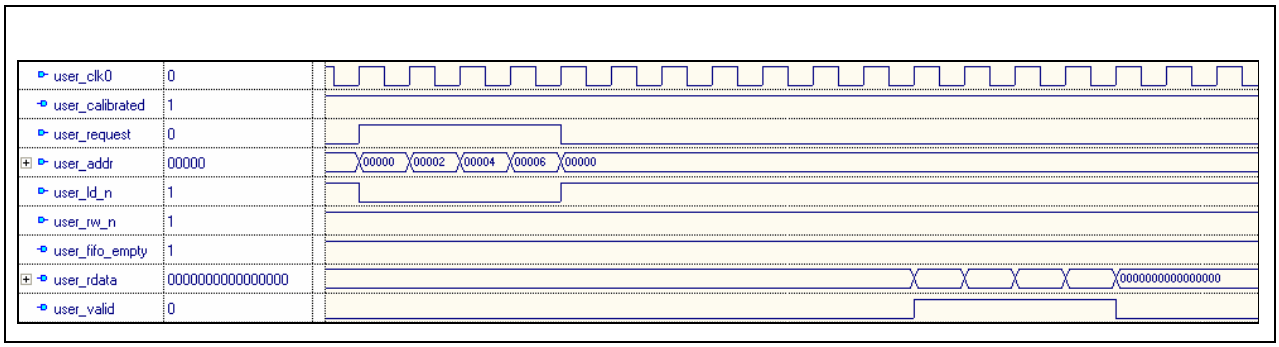

Figure 54: DDR-II SRAM Reads

BenDATA-V4 SDRAM Memory Node

The following description is intended to describe the SDRAM controller core within a DIMEtalk setting. For further information on the core refer to the *DDR SDRAM Controller Core section* of the *BenDATA-V4 Reference Guide*.

Functional Description

The BenDATA2 SDRAM Memory Node, shown in Figure 55, implements an interface to DDR SDRAM as supported on the BenDATA-V4 module. It has two interfaces - a DIMEtalk interface and a user interface - both of which provide read and write access to SDRAM. The node supports independent clock domains for DIMEtalk and user interfaces, the user interface runs from a clock synchronized to the SDRAM. Due to the row paging operation of the SDRAM there is a performance penalty introduced when swapping between rows which occurs when sequential accesses are requested at different pages in the memory map. For this reason the user is given control of DIMEtalk access to the SDRAM so that when access is requested by DIMEtalk it can be granted at the most opportune moment for the user. However, prior to granting control to DIMEtalk, the user must ensure that the input FIFO is empty and the output FIFO has been flushed of data.With this exception the user interface operates in an identical fashion to that described in the *BenDATA-V4 Reference Guide*. For convenience this is summarized below the following figure.

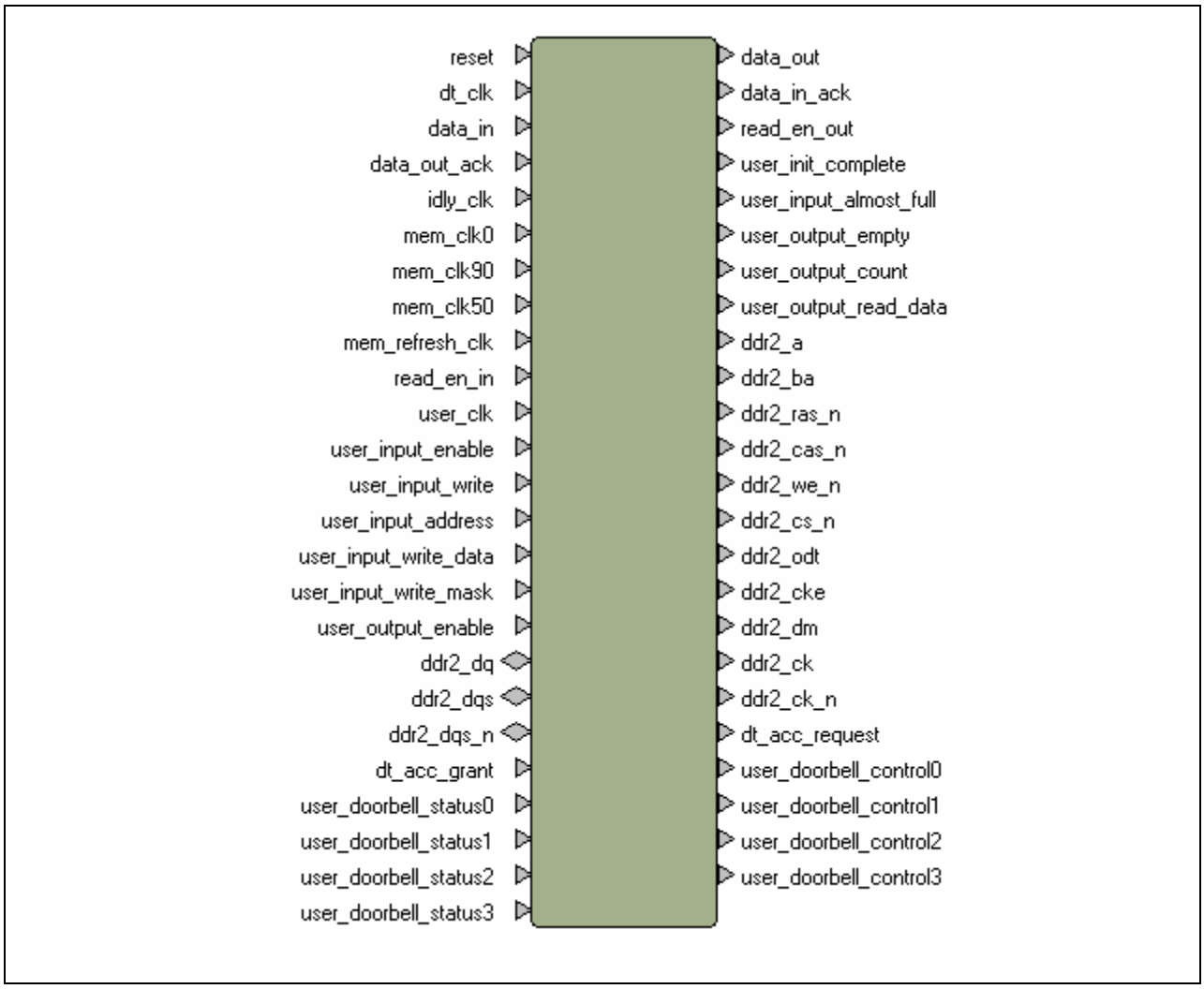

Figure 55: BenDATA-V4 SDRAM Memory Node

The node includes an input FIFO and an output FIFO, write requests are posted into the input FIFO along with associated write data and write mask setup by. This request is strobed into the FIFO by asserting user_input_enable for one cycle. The datapath is 128-bits wide, each 32-bit word is write enabled with an associated mask. Read requests

are posted in the same way although write_data and mask busses have no meaning in this context and can be set to any value. Having posted a read request and following the read access from the SDRAM the read data is returned in the output FIFO. This is indicated by user_output_empty being deasserted and the output data is popped from the FIFO by asserting output_enable for one cycle. The data is available in the next cycle.

Additional Modules

This module must be coupled with the Bendata-4 ddr2_clocks component in order to provide the required clocks for the SDRAM interface.

User Generics

The user generics are shown in Table 103.

Name	Type	Description
$bank	Banka/bankb	Not used in the VHDL however should be set up on a core basis via the GUI by right-clicking on the component instance and editing this component parameter which then allows the software to allocate the correct constraints.

Table 103: BenDATA-V4 SDRAM Memory Node User Generics

Extended Doorbell

There are four doorbell status inputs and four doorbell control outputs. The status inputs can be used to trap a change of state on an input whilst the control outputs can be used as a trigger for aspects of a user's design. For more information on packets see "Request Packet Type 10 (Doorbell class)".

Signals

The signal descriptions are provided in Table 104.

Group	Type	Signal Description	Signal Width	Direction
dcmlockreset	User Connection	reset : in STD_LOGIC;	1-bit	In
dt_clk	Clock Connection	dt_clk : in STD_LOGIC;	1-bit	In
DIMEtalk	DIMEtalk Connection	data_in : in STD_LOGIC_VECTOR(31 downto 0);	32-bits	In
		data_out : out STD_LOGIC_VECTOR(31 downto 0);	32-bits	Out
		data_in_ack : out STD_LOGIC;	1-bit	Out
		data_out_ack : in STD_LOGIC;	1-bit	In
DDR2CoreClocks	User Connection	idly_clk : in STD_LOGIC;	1-bit	In
		mem_clk0 : in STD_LOGIC;	1-bit	In
		mem_clk90 : in STD_LOGIC;	1-bit	In
		mem_clk50 : in STD_LOGIC;	1-bit	In
SDRAMRefreshClock	User Connection	mem_refresh_clk : in STD_LOGIC;	1-bit	In
normalisation	User Connection	read_en_out : out STD_LOGIC;	1-bit	Out
		read_en_in : in STD_LOGIC;	1-bit	In
user_clk	Clock Connection	user_clk : in STD_LOGIC;	1-bit	In

Table 104: BenDATA-V4 SDRAM Memory Node Signals

Group	Type	Signal Description	Signal Width	Direction
DDR2SDRAMUserInterface	User Connection	user_init_complete : out STD_LOGIC;	1-bit	Out
		user_input_enable : in STD_LOGIC;	1-bit	In
		user_input_almost_full : out STD_LOGIC;	1-bit	Out
		user_input_write : in STD_LOGIC;	1-bit	In
		user_input_address : in STD_LOGIC_VECTOR(35 downto 0);	36-bits	In
		user_input_write_data : in STD_LOGIC_VECTOR(127 downto 0);	128-bits	In
		user_input_write_mask : in STD_LOGIC_VECTOR(15 downto 0);	16-bits	In
		user_output_enable : in STD_LOGIC;	1-bit	In
		user_output_empty : out STD_LOGIC;	1-bit	Out
		user_output_count : out STD_LOGIC_VECTOR(11 downto 0);	12-bits	Out
		user_output_read_data : out STD_LOGIC_VECTOR(127 downto 0);	128-bits	Out

Table 104: BenDATA-V4 SDRAM Memory Node Signals

Group	Type	Signal Description	Signal Width	Direction
SDRAMInterface	User Connection	ddr2_a : out STD_LOGIC_VECTOR(13 downto 0);	14-bits	Out
		ddr2_dq : inout STD_LOGIC_VECTOR(31 downto 0);	32-bits	Out
		ddr2_ba : out STD_LOGIC_VECTOR(2 downto 0);	3-bits	Out
		ddr2_ras_n : out STD_LOGIC;	1-bit	Out
		ddr2_cas_n : out STD_LOGIC;	1-bit	Out
		ddr2_we_n : out STD_LOGIC;	1-bit	Out
		ddr2_cs_n : out STD_LOGIC;	1-bit	Out
		ddr2_odt : out STD_LOGIC;	1-bit	Out
		ddr2_cke : out STD_LOGIC;	1-bit	Out
		ddr2_dm : out STD_LOGIC_VECTOR(3 downto 0);	4-bits	Out
		ddr2_dqs : inout STD_LOGIC_VECTOR(3 downto 0);	4-bits	Out
		ddr2_dqs_n : inout STD_LOGIC_VECTOR(3 downto 0);	4-bits	Out
		ddr2_ck : out STD_LOGIC_VECTOR(3 downto 0);	4-bits	Out
		ddr2_ck_n : out STD_LOGIC_VECTOR(3 downto 0);	4-bits	Out
ArbitrationInterface	User Connection	dt_acc_request : out STD_LOGIC;	1-bit	Out
		dt_acc_grant : in STD_LOGIC;	1-bit	In

Table 104: BenDATA-V4 SDRAM Memory Node Signals

Group	Type	Signal Description	Signal Width	Direction
Doorbells	User Connection	user_doorbell_status0 : in STD_LOGIC;	1-bit	In
		user_doorbell_status1 : in STD_LOGIC;	1-bit	In
		user_doorbell_status2 : in STD_LOGIC;	1-bit	In
		user_doorbell_status3 : in STD_LOGIC;	1-bit	In
		user_doorbell_control0 : out STD_LOGIC;	1-bit	Out
		user_doorbell_control1 : out STD_LOGIC;	1-bit	Out
		user_doorbell_control2 : out STD_LOGIC;	1-bit	Out
		user_doorbell_control3 : out STD_LOGIC;	1-bit	Out

Table 104: BenDATA-V4 SDRAM Memory Node Signals

Support Files

The support file names and devices used are listed in Table 105.

Location	Device Usage
DIMEtalk\common\retime.vhd	All
DIMEtalk\common\bretime.vhd	All
DIMEtalk\common\resync_rise.vhd	All
DIMEtalk\common\dtnode_slave_control.vhd	All
DIMEtalk\common\pkg_dimetalk_global.vhd	All
DIMEtalk\library\ddr2_sram\sdram_if_bendata_v4.support\ddr2_sdram.ngc	All
DIMEtalk\library\ddr2_sram\sdram_if_bendata_v4.support\sdram_if_bendata_v4.vhd	All
DIMEtalk\library\ddr2_sram\sdram_if_bendata_v4.support\fifo72.ngc	All
DIMEtalk\library\ddr2_sram\sdram_if_bendata_v4.support\fifo36.ngc	All
DIMEtalk\library\ddr2_sram\sdram_if_bendata_v4.support\fifo128.ngc	All

Table 105: BenDATA-V4 SDRAM Memory Node Support Files

Component Definition

component sdram_if_bendata_v4

generic (

node_ID : std_logic_vector(NODEID_SZ-1 downto 0));

port (

```
reset                : in    std_logic;
dt_clk               : in    std_logic;
data_in              : in    std_logic_vector(DT_DATA_SZ-1 downto 0);
data_out             : out   std_logic_vector(DT_DATA_SZ-1 downto 0);
data_in_ack          : out   std_logic;
data_out_ack         : in    std_logic;
idly_clk             : in    std_logic;
mem_clk0             : in    std_logic;
mem_clk90            : in    std_logic;
mem_clk50            : in    std_logic;
mem_refresh_clk      : in    std_logic;
read_en_out          : out   std_logic;
read_en_in           : in    std_logic;
user_clk             : in    std_logic;
user_init_complete   : out   std_logic;
user_input_enable    : in    std_logic;
user_input_almost_full : out   std_logic;
user_input_write     : in    std_logic;
user_input_address   : in    std_logic_vector(BDIV_AWIDTH-1 downto 0);
user_input_write_data : in    std_logic_vector(BDIV_DWIDTH-1 downto 0);
user_input_write_mask : in    std_logic_vector(BDIV_MWIDTH-1 downto 0);
user_output_enable   : in    std_logic;
user_output_empty    : out   std_logic;
user_output_count    : out   std_logic_vector(BDIV_RCWIDTH-1 downto 0);
user_output_read_data : out   std_logic_vector(BDIV_DWIDTH-1 downto 0);
ddr2_a               : out   std_logic_vector(13 downto 0);
ddr2_dq              : inout std_logic_vector(31 downto 0);
ddr2_ba              : out   std_logic_vector(2 downto 0);
ddr2_ras_n           : out   std_logic;
ddr2_cas_n           : out   std_logic;
ddr2_we_n            : out   std_logic;
ddr2_cs_n            : out   std_logic;
ddr2_odt             : out   std_logic;
```

```
    ddr2_cke            : out   std_logic;
    ddr2_dm             : out   std_logic_vector(3 downto 0);
    ddr2_dqs            : inout std_logic_vector(3 downto 0);
    ddr2_dqs_n          : inout std_logic_vector(3 downto 0);
    ddr2_ck             : out   std_logic_vector(3 downto 0);
    ddr2_ck_n           : out   std_logic_vector(3 downto 0);
    dt_acc_request      : out   std_logic;
    dt_acc_grant_n      : in    std_logic;
    user_doorbell_status0  : in    std_logic;
    user_doorbell_status1  : in    std_logic;
    user_doorbell_status2  : in    std_logic;
    user_doorbell_status3  : in    std_logic;
    user_doorbell_control0 : out   std_logic;
    user_doorbell_control1 : out   std_logic;
    user_doorbell_control2 : out   std_logic;
    user_doorbell_control3 : out   std_logic);
end component;
```

Waveforms

The following figures illustrate write and read accesses to the user port of the sdram controller. Note that dt_acc_grant_n should be low prior to these accesses commencing. Figure 56 shows a write access, where user_input_enable is asserted to strobe in the user_input_address and user_input_write data, with user_input_write asserted high to select a write access.

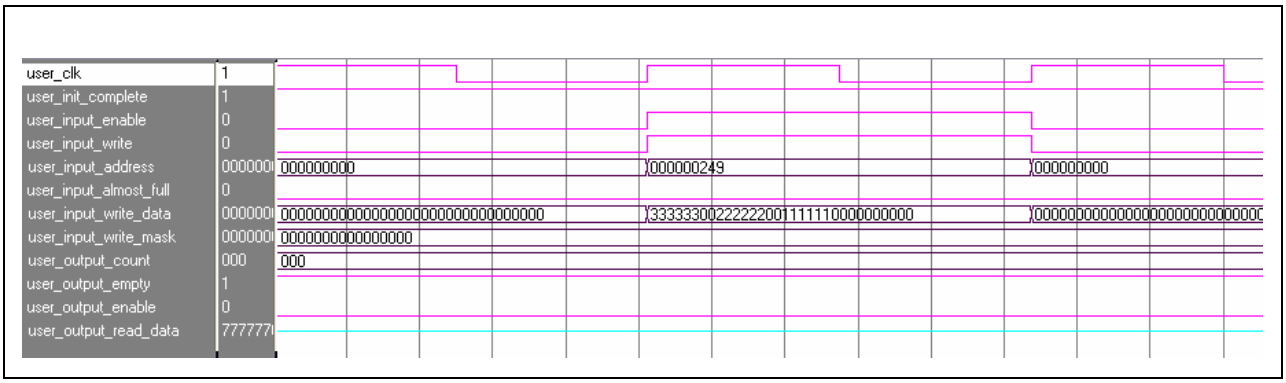

Figure 56: Write Command

A subsequent write followed by two read cycles is shown in Figure 57. Note that for read cycles the user_input_write signal should be asserted low.

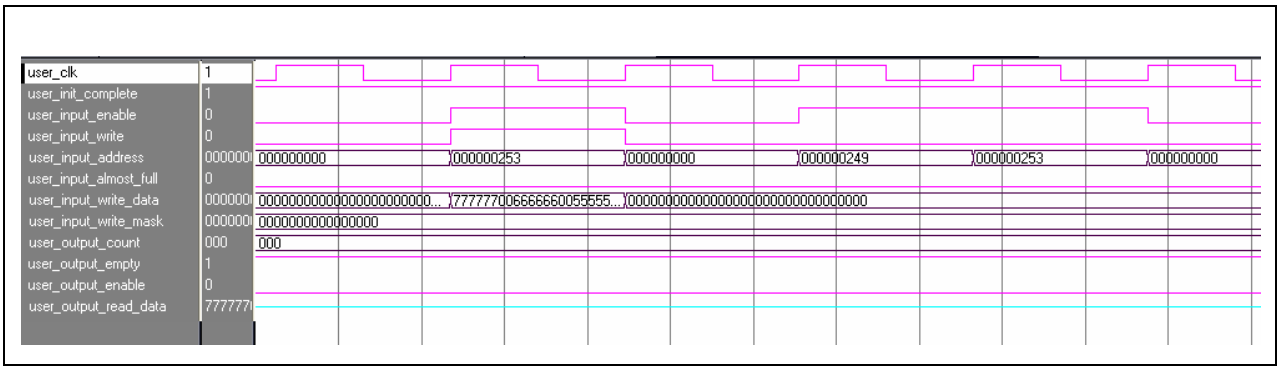

Figure 57: Read Command

Having performed two writes and posted two read commands the user waits for user_output_empty to be deasserted which indicates that the read data is now available in the output FIFO.

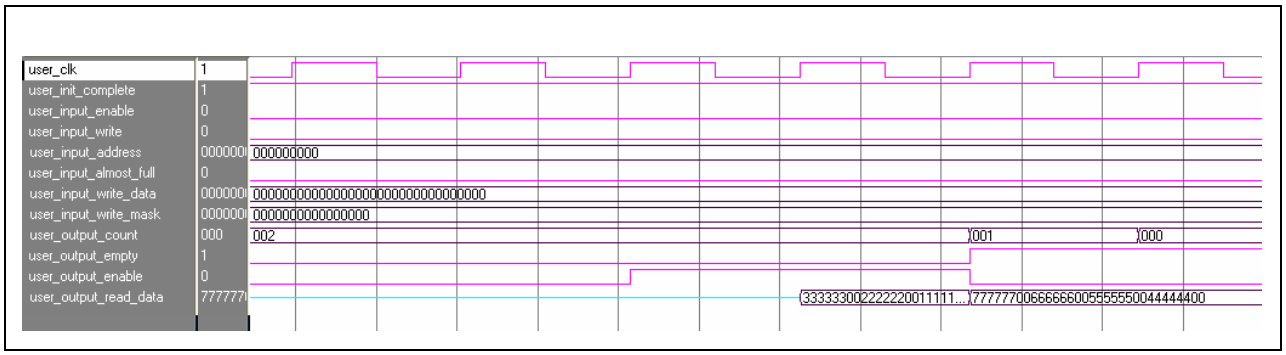

Figure 58: Read Data

The data is read out of the output FIFO by asserting user_output_enable. As shown in Figure 58 the data is valid on the next cycle.

H100 DDR2 SDRAM Memory Node

Variants

There are two variants of this node:

1. sdram_dimm_if_h100_e: used on the entry level H100 which interfaces to a 256MB DIMM
2. sdram_dimm_if_h100_m: used on the mainstream H100 which interfaces to a 512MB DIMM

Functional Description

The H100 DDR2 SDRAM Memory Node shown in Figure 59, interfaces to a registered DIMM as fitted on the H100. It has two interfaces - a DIMEtalk interface and a user interface - both of which provide read and write access to the SDRAM. The node supports independent clock domains for DIMEtalk and user interfaces which are decoupled from the SDRAM clock domain. Due to the row paging operation of the SDRAM there is a penalty introduced when swapping between rows, which occurs when sequential accesses are requested at different pages in the memory map. For this reason the user is given control of DIMEtalk access to the SDRAM so that when access is requested by DIMEtalk it can be granted at the most opportune moment for the user. However, prior to granting control to DIMEtalk, the user must ensure that the input FIFO is empty and the output FIFO has been flushed of data. The user interface is described in "User Interface Waveforms".

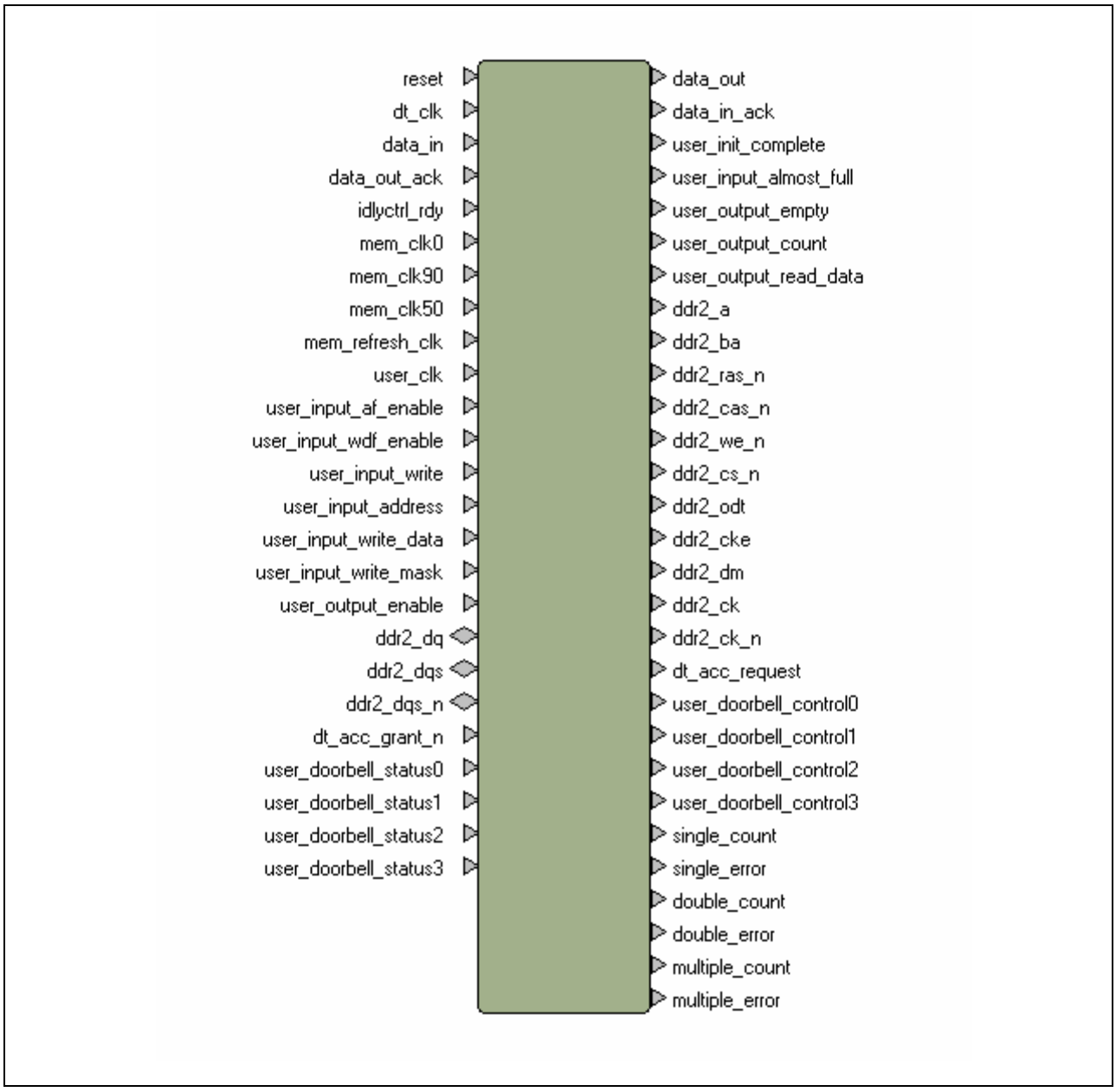

Figure 59: H100 DDR2 SDRAM Memory Node

ECC

The memory interface core features Error Correction Control, which detects and corrects all single bit errors in a code word consisting of 64 data and 8 parity bits. In addition, double and multiple bit errors are detected. Since 64-bits of data is the smallest section of data that can be protected by ECC in this implementation, any writes should by at least 64-bits wide. This is because 8 check bits are required to protect 64 data bits. For applications where non-integer multiples of 64-bits must be modified ECC should be turned off. For further information on ECC users should refer to *Xilinx Application Note XAPP645*.

Addressing/Burst Guidelines

Burst Operation

The minimum burst size on the DIMM is 4 x 64-bit words (256 bits). In order to align with this, DIMEtalk read write accesses should be limited to integer multiples of burst sizes 8, 16, etc to 64.

Address Alignment

The SDRAM devices have a page length of 1024. For a burst length of four each page is divided into 256 addressable segments. Therefore accesses must start on integer multiples of 4, i.e. 0, 4, 8, etc.

Additional Modules

This component must be used with the H100 clocks module.

User Generics

The user generics are shown in Table 106.

Name	Type	Description
CORRECT_ERRORS	Boolean default = false	The core can optionally detect single, double and multiple bit errors and correct single bit errors. ECC only functions when data is modified in 64-bit words, therefore correction is kept as a build time option to allow for applications where non-integer multiples of 64-bit words are modified.

Table 106: H100 DDR2 SDRAM Memory Node User Generics

Signals

The signal descriptions are provided in Table 107.

Group	Type	Signal Description	Signal Width	Direction
dcmlockreset	Reset Connection	reset : in STD_LOGIC;	1-bit	In
dt_clk	Clock Connection	dt_clk : in STD_LOGIC;	1-bit	In
DIMEtalk	DIMEtalk Connection	data_in : in STD_LOGIC_VECTOR(31 downto 0);	32-bits	In
		data_out : out STD_LOGIC_VECTOR(31 downto 0);	32-bits	Out
		data_in_ack : out STD_LOGIC;	1-bit	Out
		data_out_ack : in STD_LOGIC;	1-bit	In
DDR2CoreClocks	User Connection	idlyctrl_ready : in STD_LOGIC;	1-bit	In
		mem_clk0 : in STD_LOGIC;	1-bit	In
		mem_clk90 : in STD_LOGIC;	1-bit	In
		mem_clk50 : in STD_LOGIC;	1-bit	In
		mem_refresh_clk:in STD_LOGIC:	1-bit	In
user_clk	Clock Connection	user_clk : in STD_LOGIC;	1-bit	In

Table 107: H100 DDR2 SDRAM Memory Node Signals

Group	Type	Signal Description	Signal Width	Direction
DDR2SDRAM UserInterface	User Connection	user_init_complete : out STD_LOGIC;	1-bit	Out
		user_input_af_enable : in STD_LOGIC;	1-bit	In
		user_input_wdf_enable : in STD_LOGIC;	1-bit	In
		user_input_almost_full : out STD_LOGIC;	1-bit	Out
		user_input_write : in STD_LOGIC;	1-bit	In
		user_input_address : in STD_LOGIC_VECTOR(35 downto 0);	36-bits	In
		user_input_write_data : in STD_LOGIC_VECTOR(127 downto 0);	128-bits	In
		user_input_write_mask : in STD_LOGIC_VECTOR(15 downto 0);	16-bits	In
		user_output_enable : in STD_LOGIC;	1-bit	In
		user_output_empty : out STD_LOGIC;	1-bit	Out
		user_output_count : out STD_LOGIC_VECTOR(8 downto 0);	9-bits	Out
		user_output_read_data : out STD_LOGIC_VECTOR(127 downto 0);	128-bits	Out
SDRAMInterface	User Connection	ddr2_a : out STD_LOGIC_VECTOR(13 downto 0);	14-bits	Out
		ddr2_dq : inout STD_LOGIC_VECTOR(71 downto 0);	72-bits	Out
		ddr2_ba : out STD_LOGIC_VECTOR(2 downto 0);	3-bits	Out
		ddr2_ras_n : out STD_LOGIC;	1-bit	Out
		ddr2_cas_n : out STD_LOGIC;	1-bit	Out
		ddr2_we_n : out STD_LOGIC;	1-bit	Out
		ddr2_cs_n : out STD_LOGIC;	1-bit	Out
		ddr2_odt : out STD_LOGIC;	1-bit	Out
		ddr2_cke : out STD_LOGIC;	1-bit	Out
		ddr2_dm : out STD_LOGIC_VECTOR(8 downto 0);	9-bits	Out
		ddr2_dqs : inout STD_LOGIC_VECTOR(8 downto 0);	9-bits	Out
		ddr2_dqs_n : inout STD_LOGIC_VECTOR(8 downto 0);	9-bits	Out
		ddr2_ck : out STD_LOGIC;	1-bit	Out
		ddr2_ck_n : out STD_LOGIC;	1-bit	Out
ArbitrationInterface	User Connection	dt_acc_request : out STD_LOGIC;	1-bit	Out
		dt_acc_grant : in STD_LOGIC;	1-bit	In

Table 107: H100 DDR2 SDRAM Memory Node Signals

Group	Type	Signal Description	Signal Width	Direction
Doorbells	User Connection	user_doorbell_status0 : in STD_LOGIC;	1-bit	In
		user_doorbell_status1 : in STD_LOGIC;	1-bit	In
		user_doorbell_status2 : in STD_LOGIC;	1-bit	In
		user_doorbell_status3 : in STD_LOGIC;	1-bit	In
		user_doorbell_control0 : out STD_LOGIC;	1-bit	Out
		user_doorbell_control1 : out STD_LOGIC;	1-bit	Out
		user_doorbell_control2 : out STD_LOGIC;	1-bit	Out
		user_doorbell_control3 : out STD_LOGIC;	1-bit	Out
ECC Status	User Connection	single_count : out STD_LOGIC_VECTOR(7 downto 0);	8-bits	Out
		single_error : out STD_LOGIC;	1-bit	Out
		double_count : out STD_LOGIC_VECTOR(7 downto 0);	8-bits	Out
		double_error : out STD_LOGIC;	1-bit	Out
		multiple_count : out STD_LOGIC_VECTOR(7 downto 0);	8-bits	Out
		multiple_error : out STD_LOGIC;	1-bit	Out

Table 107: H100 DDR2 SDRAM Memory Node Signals

Support Files

The support file names and devices used are listed in Table 108.

Location	Device Usage
DIMEtalk\common\retime.vhd	All
DIMEtalk\common\bretime.vhd	All
DIMEtalk\common\resync_rise.vhd	All
DIMEtalk\common\dtnode_slave_control.vhd	All
DIMEtalk\common\pkg_dimetalk_global.vhd	All
DIMEtalk\library\V4 nodes\sdram_dimm_if_h100_m.support\h100_m_ddr2_sdram_dimm.ngc	H100M
DIMEtalk\library\V4 nodes\sdram_dimm_if_h100_m.support\sdram_dimm_if_h100_m.vhd	H100M
DIMEtalk\library\V4 nodes\sdram_dimm_if_h100_m.support\fifo144.ngc	H100M
DIMEtalk\library\V4 nodes\sdram_dimm_if_h100_m.support\fifo36.ngc	H100M
DIMEtalk\library\V4 nodes\sdram_dimm_if_h100_m.support\fifo128.ngc	H100M
DIMEtalk\library\V4 nodes\sdram_dimm_if_h100_e.support\h100_e_ddr2_sdram_dimm.ngc	H100E
DIMEtalk\library\V4 nodes\sdram_dimm_if_h100_e.support\sdram_dimm_if_h100_e.vhd	H100E
DIMEtalk\library\V4 nodes\sdram_dimm_if_h100_e.support\fifo144.ngc	H100E
DIMEtalk\library\V4 nodes\sdram_dimm_if_h100_e.support\fifo36.ngc	H100E
DIMEtalk\library\V4 nodes\sdram_dimm_if_h100_e.support\fifo128.ngc	H100E

Table 108: H100 DDR2 SDRAM Memory Node Support Files

User Interface Waveforms

The node includes an input FIFO and an output FIFO. Write requests are posted into the input FIFO along with associated write data and mask. Read requests are posted in the same way although write data and mask have no meaning in this context and can be set to any value. Having posted a read request and following the read access from SDRAM the read data is returned in the output FIFO. This is indicated by the deassertion of user_output_empty, data is popped from the FIFO by asserting user_output_enable, data being available in the next cycle. The following figures illustrate write and read accesses to the user port of the H100 DDR2 SDRAM Memory Node. Note that dt_acc_grant_n should be low prior to these accesses commencing.

Figure 60 shows a write access, user_wdf_enable is asserted for two cycles to meet the 256-bit minimum burst size. In the second of these cycles the user_input_address is strobed in with user_af_enable, and user_input_write is asserted high to select a write access.

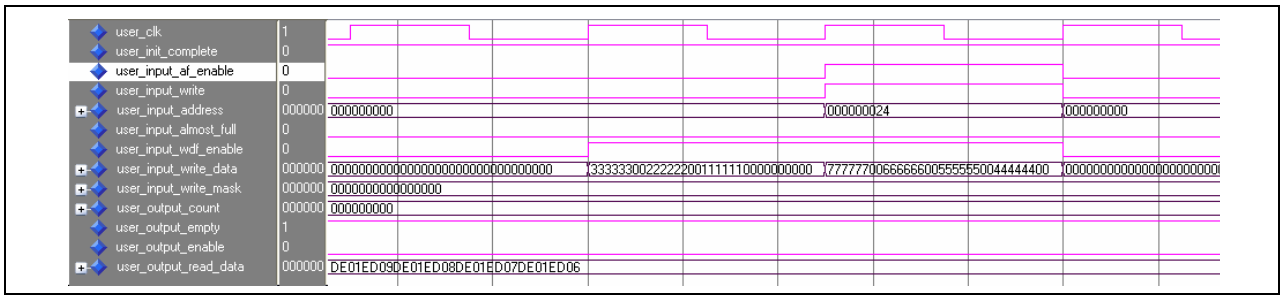

Figure 60: Write Command

Figure 61 shows a read request, where user_input_af_enable is asserted to strobe in the address with user_input_write to select a read.

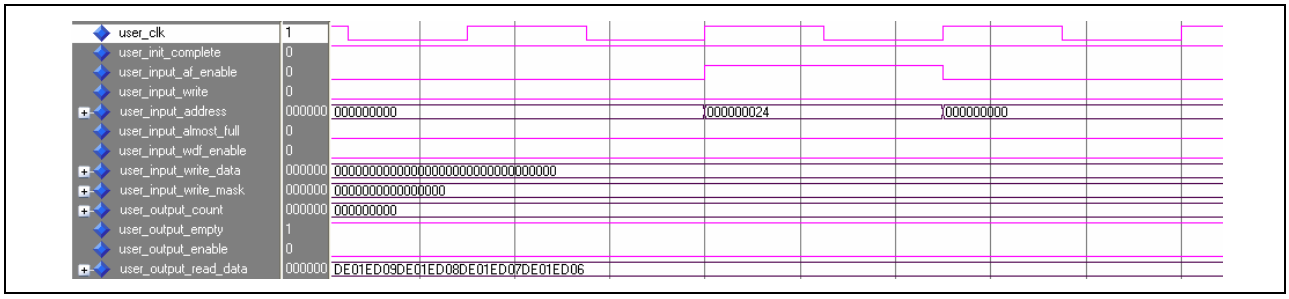

Figure 61: Read Command

The data is read out of the output FIFO by asserting user_output_enable. As shown in Figure 62 the data is valid on the next cycle.

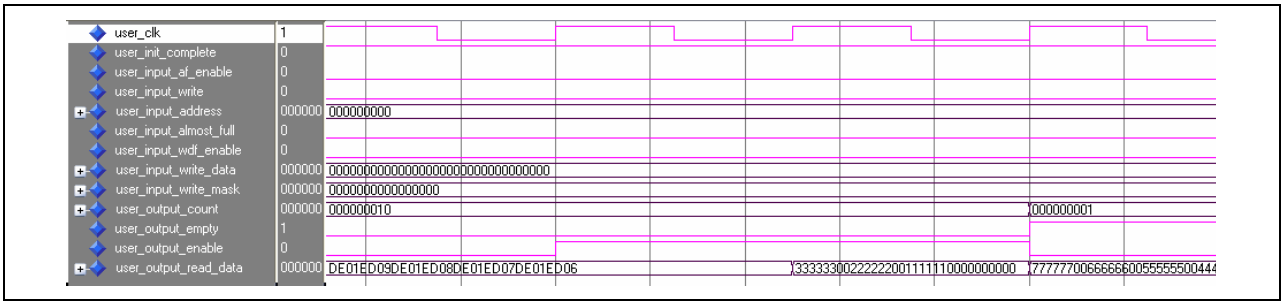

Figure 62: Read Data

1.9 DDR SDRAM

BenDATA2 SDRAM Clock Module

Functional Description

This module is identical to that which is documented in the *DDR SDRAM Controller Core section* of the *BenDATA-II Reference Guide* and is summarized here for convenience. It provides differential clocks to drive the SDRAM in addition to the phase shifted clocks that are required by the SDRAM controller. This module exists as a separate entity to allow it to supply clocks to more than one SDRAM controller. The BenDATA2 SDRAM clock module component is shown in Figure 63.

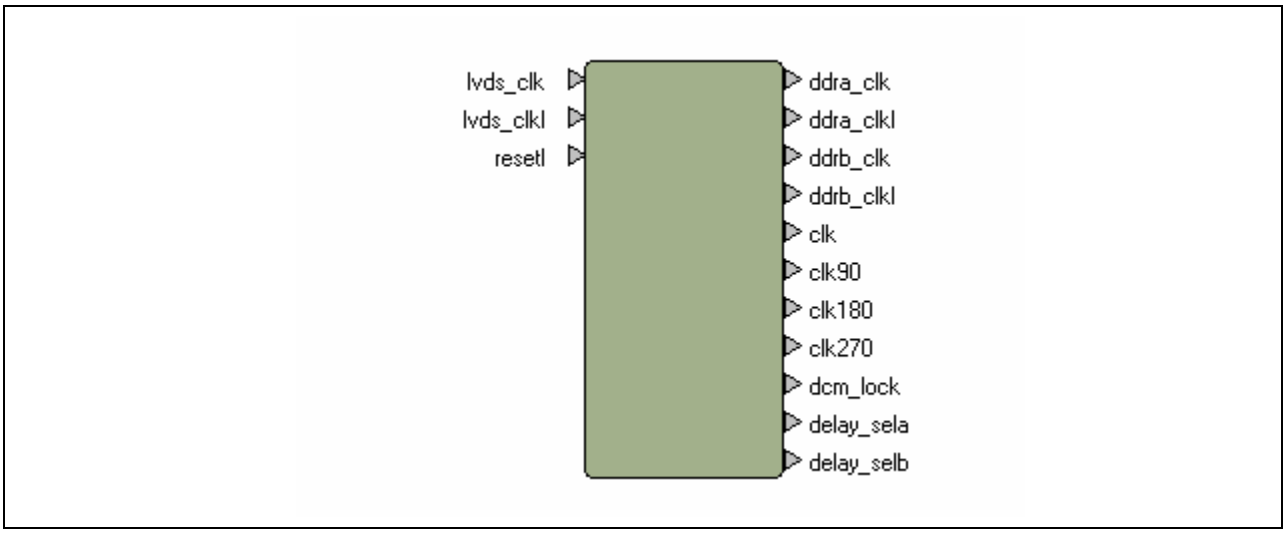

Figure 63: benDATA2 SDRAM Clock Module Component

User Generics

The user generics are shown in Table 109.

Name	Type	Description
DDR400	Boolean	True for DDR400, false for DDR333
XC2VP100	Boolean	True for XC2VP100, false for XC2VP70

Table 109: BenDATA2 SDRAM Clock Module User Generics

Signals

The signal descriptions are provided in Table 110.

Group	Type	Signal Description	Signal Width	Direction
InputClock	User Connection	lvds_clk : in STD_LOGIC	1-bit	In
		lvds_clkl : in STD_LOGIC		In
resetl	Reset Connection	resetl : in STD_LOGIC	1-bit	In
SDRAMBankAClock	User Connection	ddra_clk : out STD_LOGIC_VECTOR(1 downto 0)	2-bits	Out
		ddra_clkl : out STD_LOGIC_VECTOR(1 downto 0)	2-bits	Out
SDRAMBankBClock	User Connection	ddrb_clk : out STD_LOGIC_VECTOR(1 downto 0)	2-bits	Out
		ddrb_clkl : out STD_LOGIC_VECTOR(1 downto 0)	2-bits	Out
SDRAMCoreClocks	User Connection	clk : out STD_LOGIC	1-bit	Out
		clk90 : out STD_LOGIC	1-bit	Out
		clk180 : out STD_LOGIC	1-bit	Out
		clk270 : out STD_LOGIC	1-bit	Out
		dcm_lock : out STD_LOGIC;	1-bit	Out
ClockCalibrationA	User Connection	delay_sela : out STD_LOGIC_VECTOR(4 downto 0)	5-bits	Out
ClockCalibrationB	User Connection	delay_selb : out STD_LOGIC_VECTOR(4 downto 0)	5-bits	Out

Table 110: BenDATA2 SDRAM Clock Module Signals

Support Files

The support file names and devices used are listed in Table 111.

Location	Device usage
DIMEtalk\library\SDRAM\ddr_clocks.vhd	All devices
DIMEtalk\library\SDRAM\dcmx3y0_2vp100.edn	All devices
DIMEtalk\library\SDRAM\cal_top.ngc	All devices
DIMEtalk\library\SDRAM\dcmx1y0_2vp70.edf	All devices

Table 111: BenDATA2 SDRAM Clock Module Support Files

Component Definition

```
component ddr_clocks
generic (
ddr400   : boolean;
xc2vp100 : boolean);
port (
```

```
        lvds_clk   : in  std_logic;
        lvds_clkl  : in  std_logic;
        resetl     : in  std_logic;
        ddra_clk   : out std_logic_vector(1 downto 0);
        ddra_clkl  : out std_logic_vector(1 downto 0);
        ddrb_clk   : out std_logic_vector(1 downto 0);
        ddrb_clkl  : out std_logic_vector(1 downto 0);
        clk        : out std_logic;
        clk90      : out std_logic;
        clk180     : out std_logic;
        clk270     : out std_logic;
        dcm_lock   : out std_logic;
        delay_sela : out std_logic_vector(4 downto 0);
        delay_selb : out std_logic_vector(4 downto 0));
end component;
```

BenDATA2 SDRAM Memory Node - 333MHz

The following description is intended to describe the DDR SDRAM controller core within a DIMEtalk setting. For further information on the core refer to the *DDR SDRAM Controller Core section* of the *BenDATA-II Reference Guide*. The version of the core chosen - DDR333 or DDR400 - is dependent on the speed grade of the FPGA populated on the module.

Functional Description

The BenDATA2 SDRAM Memory Node, shown in Figure 64, implements an interface to DDR SDRAM as supported on the BenDATA-II module. It has two interfaces - a DIMEtalk interface and a user interface - both of which provide read and write access to SDRAM. The node supports independent clock domains for DIMEtalk and user interfaces, the user interface runs from a clock synchronized to the SDRAM. Due to the row paging operation of the SDRAM there is a performance penalty introduced when swapping between rows which occurs when sequential accesses are requested at different pages in the memory map. For this reason the user is given control of DIMEtalk access to the SDRAM so that when access is requested by DIMEtalk it can be granted at the most opportune moment for the user. However, prior to granting control to DIMEtalk, the user must ensure that the input FIFO is empty and the output FIFO has been flushed of data. With this exception the user interface operates in an identical fashion to that described in the *BenDATA-II Reference Guide*. For convenience this is summarized after Figure 64.

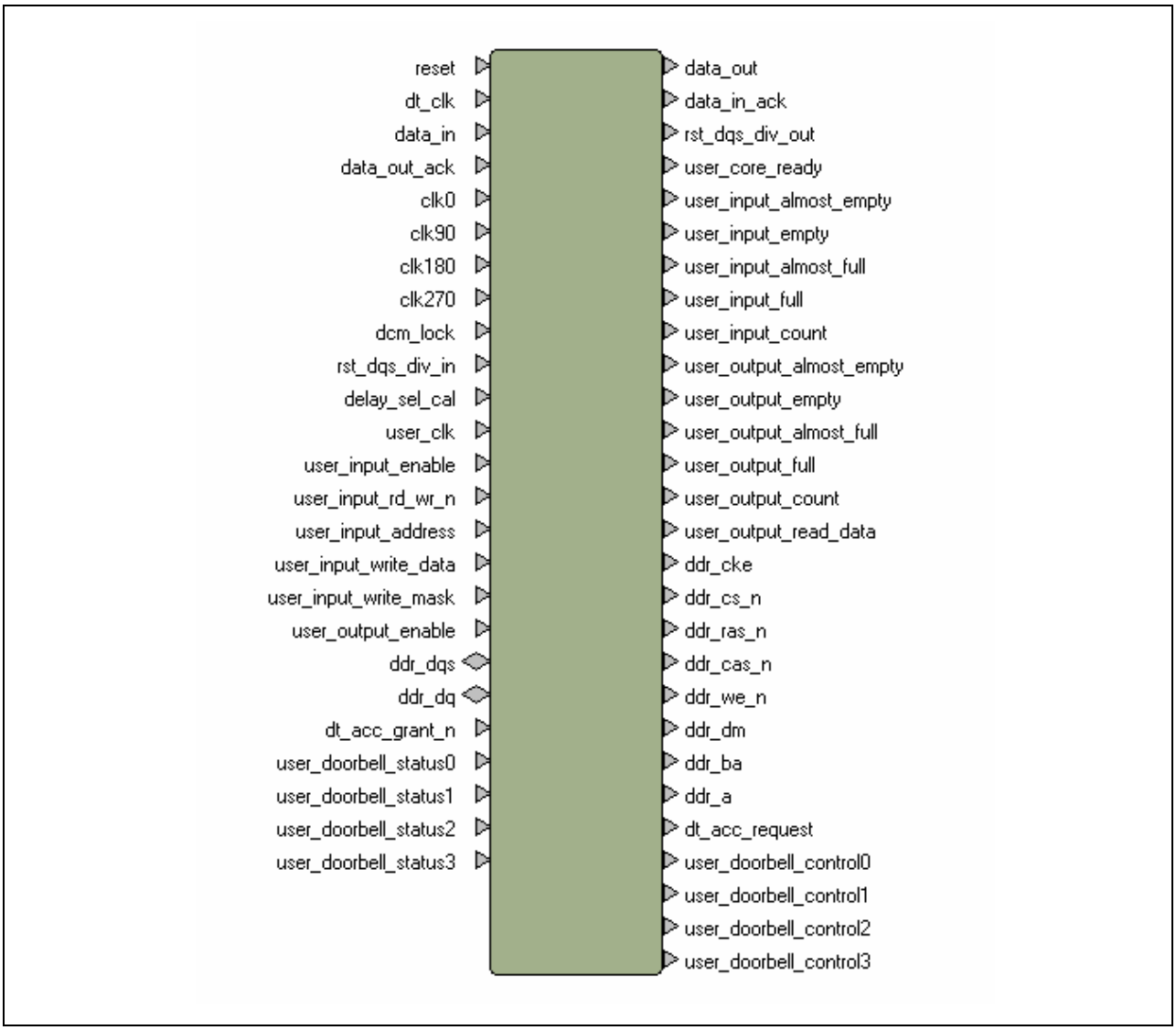

Figure 64: benDATA2 SDRAM Memory Node - 333MHz

The node includes an input FIFO and an output FIFO, write requests are posted into the input FIFO along with associated write data and write mask setup by. This request is strobed into the FIFO by asserting user_input_enable for one cycle. The datapath is 128-bits wide, each 32-bit word is write enabled with an associated mask. Read requests are posted in the same way although write_data and mask busses have no meaning in this context and can be set to any value. Having posted a read request and following the read access from the SDRAM the read data is returned in the output FIFO. This is indicated by user_output_empty being deasserted and the output data is popped from the FIFO by asserting output_enable for one cycle. The data is available in the next cycle.

Additional Modules

This module must be coupled with the BenDATA2 SDRAM Clock Module component in order to provide the required clocks for the SDRAM interface.

Extended Doorbell

There are four doorbell status inputs and four doorbell control outputs. The status inputs can be used to trap a change of state on an input whilst the control outputs can be used as a trigger for aspects of a user's design. For more information on packets see "Request Packet Type 10 (Doorbell class)".

User Generics

The user generics are shown in Table 112.

Name	Type	Description
$bank	Banka/bankb	Not used in the VHDL however should be set up on a core basis via the GUI by right-clicking on the component instance and editing this component parameter which then allows the software to allocate the correct constraints.

Table 112: BenDATA2 SDRAM Memory Node - 333MHz User Generics

Signals

The signal descriptions are provided in Table 113.

Group	Type	Signal Description	Signal Width	Direction
resetl	Reset Connection	resetl : in STD_LOGIC	1-bit	In
dt_clk	Clock Connection	dt_clk : in STD_LOGIC;	1-bit	
DIMEtalk	DIMEtalk Connection	data_in : in STD_LOGIC_VECTOR(31 downto 0);	32-bits	In
		data_out : out STD_LOGIC_VECTOR(31 downto 0);	32-bits	Out
		data_in_ack : out STD_LOGIC;	1-bit	Out
		data_out_ack : in STD_LOGIC;	1-bit	In
SDRAMCoreClocks	User Connection	clk0 : in STD_LOGIC;	1-bit	In
		clk90 : in STD_LOGIC;	1-bit	In
		clk180 : in STD_LOGIC;	1-bit	In
		clk270 : in STD_LOGIC;	1-bit	In
		dcm_lock : in STD_LOGIC;	1-bit	In
ClockCalibration	User Connection	rst_dqs_div_out : out STD_LOGIC;	1-bit	Out
		rst_dqs_div_in : in STD_LOGIC;	1-bit	In
ClockCalibrationDelay	User Connection	delay_sel_cal : in STD_LOGIC_VECTOR(4 downto 0);	5-bits	In
user_clk	Clock Connection	user_clk : in STD_LOGIC;	1-bit	In

Table 113: BenDATA2 SDRAM Memory Node - 333MHz Signals

Group	Type	Signal Description	Signal Width	Direction
UserInterface	User Connection	user_core_ready : out STD_LOGIC;	1-bit	Out
		user_input_enable : in STD_LOGIC;	1-bit	In
		user_input_almost_empty : out STD_LOGIC;	1-bit	Out
		user_input_empty : out STD_LOGIC;	1-bit	Out
		user_input_almost_full : out STD_LOGIC;	1-bit	Out
		user_input_full : out STD_LOGIC;	1-bit	Out
		user_input_count : out STD_LOGIC_VECTOR(8 downto 0);	9-bits	Out
		user_input_rd_wr_n : in STD_LOGIC;	1-bit	In
		user_input_address : in STD_LOGIC_VECTOR(23 downto 0);	24-bits	In
		user_input_write_data : in STD_LOGIC_VECTOR(127 downto 0);	128-bits	In
		user_input_write_mask : in STD_LOGIC_VECTOR(3 downto 0);	4-bits	In
		user_output_enable : in STD_LOGIC;	1-bit	In
		user_output_almost_empty : out STD_LOGIC;	1-bit	Out
		user_output_empty : out STD_LOGIC;	1-bit	Out
		user_output_almost_full : out STD_LOGIC;	1-bit	Out
		user_output_full : out STD_LOGIC;	1-bit	Out
		user_output_count : out STD_LOGIC_VECTOR(8 downto 0);	9-bits	Out
		user_output_read_data : out STD_LOGIC_VECTOR(127 downto 0);	128-bits	Out
SDRAMInterface	User Connection	ddr_dqs : inout STD_LOGIC_VECTOR(3 downto 0);	4-bits	
		ddr_dq : inout STD_LOGIC_VECTOR(31 downto 0);	32-bits	
		ddr_cke : out STD_LOGIC;	1-bit	Out
		ddr_cs_n : out STD_LOGIC;	1-bit	Out
		ddr_ras_n : out STD_LOGIC;	1-bit	Out
		ddr_cas_n : out STD_LOGIC;	1-bit	Out
		ddr_we_n : out STD_LOGIC;	1-bit	Out
		ddr_dm : out STD_LOGIC_VECTOR(3 downto 0);	4-bits	Out
		ddr_ba : out STD_LOGIC_VECTOR(1 downto 0);	2-bits	Out
		ddr_a : out STD_LOGIC_VECTOR(13 downto 0);	14-bits	Out

Table 113: BenDATA2 SDRAM Memory Node - 333MHz Signals

Group	Type	Signal Description	Signal Width	Direction
ArbitrationInterface	User Connection	dt_acc_request : out STD_LOGIC;	1-bit	Out
		dt_acc_grant_n : in STD_LOGIC;	1-bit	In
Doorbells	User Connection	user_doorbell_status0 : in STD_LOGIC;	1-bit	
		user_doorbell_status1 : in STD_LOGIC;	1-bit	
		user_doorbell_status2 : in STD_LOGIC;	1-bit	
		user_doorbell_status3 : in STD_LOGIC;	1-bit	
		user_doorbell_control0 : out STD_LOGIC;	1-bit	
		user_doorbell_control1 : out STD_LOGIC;	1-bit	
		user_doorbell_control2 : out STD_LOGIC;	1-bit	
		user_doorbell_control3 : out STD_LOGIC;	1-bit	

Table 113: BenDATA2 SDRAM Memory Node - 333MHz Signals

Support Files

The support file names and devices used are listed in Table 114.

Location	Device usage
DIMEtalk\library\SDRAM\sdram_if_bendata2_333.vhd	All devices
DIMEtalk\library\SDRAM\sdram_if_bendata2_333.vhd	All devices
DIMEtalk\common\retime.vhd	All devices
DIMEtalk\common\bretime.vhd	All devices
DIMEtalk\common\resync_rise.vhd	All devices
DIMEtalk\common\dtnode_slave_control.vhd	All devices
DIMEtalk\common\pkg_dimetalk_global.vhd	All devices
DIMEtalk\library\SDRAM\333MHz\ddrcore.ngc	All devices
DIMEtalk\library\SDRAM\fifo192x511.edn	All devices
DIMEtalk\library\SDRAM\comparator.edn	All devices
DIMEtalk\library\SDRAM\dcmx1y0_2vp70.edf	All devices
DIMEtalk\library\SDRAM\dcmx3y0_2vp100.edn	All devices
DIMEtalk\library\SDRAM\fifo128x511.edn	All devices

Table 114: BenDATA2 SDRAM Memory Node - 333MHz Support Files

Component Definition

```
component sdram_if
generic (
node_ID              :      std_logic_vector(NODEID_SZ-1 downto 0);
mode400              :      boolean);
port (
    reset              : in    std_logic;
```

```
dt_clk                : in    std_logic;
data_in               : in    std_logic_vector(DT_DATA_SZ-1 downto 0);
data_out              : out   std_logic_vector(DT_DATA_SZ-1 downto 0);
data_in_ack           : out   std_logic;
data_out_ack          : in    std_logic;
clk0                  : in    std_logic;
clk90                 : in    std_logic;
clk180                : in    std_logic;
clk270                : in    std_logic;
dcm_lock              : in    std_logic;
rst_dqs_div_out       : out   std_logic;
rst_dqs_div_in        : in    std_logic;
delay_sel_cal         : in    std_logic_vector(4 downto 0);
user_clk              : in    std_logic;
user_core_ready       : out   std_logic;
user_input_enable     : in    std_logic;
user_input_almost_empty  : out   std_logic;
user_input_empty      : out   std_logic;
user_input_almost_full   : out   std_logic;
user_input_full       : out   std_logic;
user_input_count      : out   std_logic_vector(8 downto 0);
user_input_rd_wr_n    : in    std_logic;
user_input_address    : in    std_logic_vector(BDII_AWIDTH-1 downto 0);
user_input_write_data    : in    std_logic_vector(BDII_DWIDTH-1 downto 0);
user_input_write_mask    : in    std_logic_vector(BDII_MWIDTH-1 downto 0);
user_output_enable    : in    std_logic;
user_output_almost_empty : out   std_logic;
user_output_empty     : out   std_logic;
user_output_almost_full  : out   std_logic;
user_output_full      : out   std_logic;
user_output_count     : out   std_logic_vector(8 downto 0);
user_output_read_data    : out   std_logic_vector(BDII_DWIDTH-1 downto 0);
ddr_dqs               : inout std_logic_vector(3 downto 0);
ddr_dq                : inout std_logic_vector(31 downto 0);
ddr_cke               : out   std_logic;
ddr_cs_n              : out   std_logic;
ddr_ras_n             : out   std_logic;
```

```
    ddr_cas_n              : out   std_logic;
    ddr_we_n               : out   std_logic;
    ddr_dm                 : out   std_logic_vector(3 downto 0);
    ddr_ba                 : out   std_logic_vector(1 downto 0);
    ddr_a                  : out   std_logic_vector(13 downto 0);
    dt_acc_request         : out   std_logic;
    dt_acc_grant_n         : in    std_logic;
    user_doorbell_status0  : in    std_logic;
    user_doorbell_status1  : in    std_logic;
    user_doorbell_status2  : in    std_logic;
    user_doorbell_status3  : in    std_logic;
    user_doorbell_control0 : out   std_logic;
    user_doorbell_control1 : out   std_logic;
    user_doorbell_control2 : out   std_logic;
    user_doorbell_control3 : out   std_logic);
end component;
```

Waveforms

The following figures illustrate write and read accesses to the user port of the sdram controller. Note that dt_acc_grant_n should be low prior to these accesses commencing. Figure 65 shows a write access, where user_input_enable is asserted to strobe in the user_input_address and user_input_write data, with user_input_rd_wr_n asserted low to select a write access.

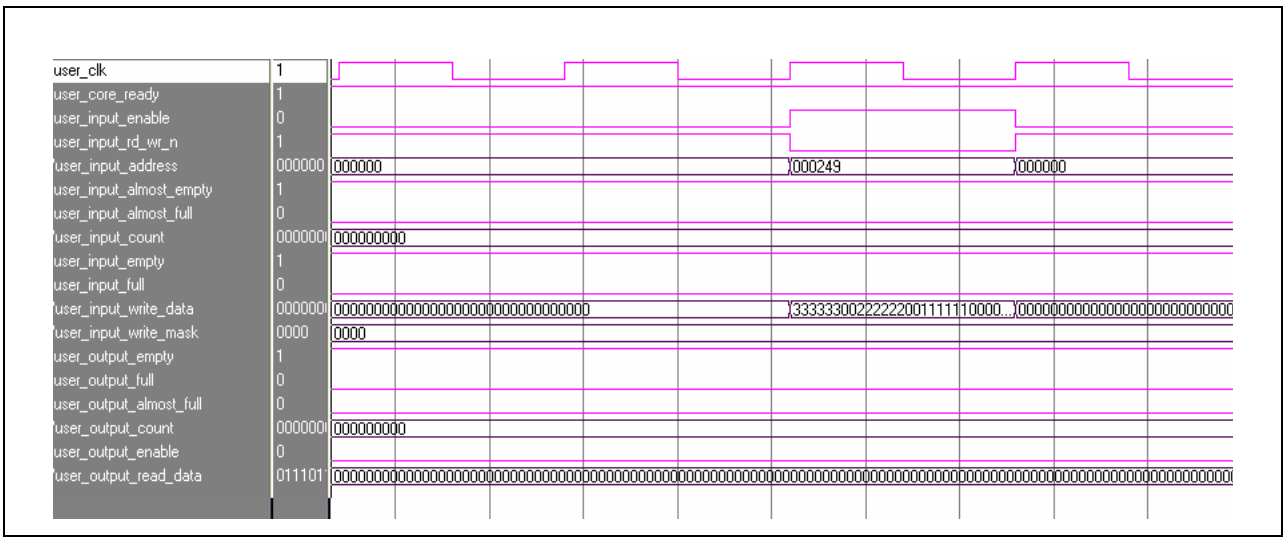

Figure 65: Write Command

A subsequent write followed by two read cycles is shown in Figure 66. Note that for read cycles the user_input_rd_wr_n signal should be asserted high.

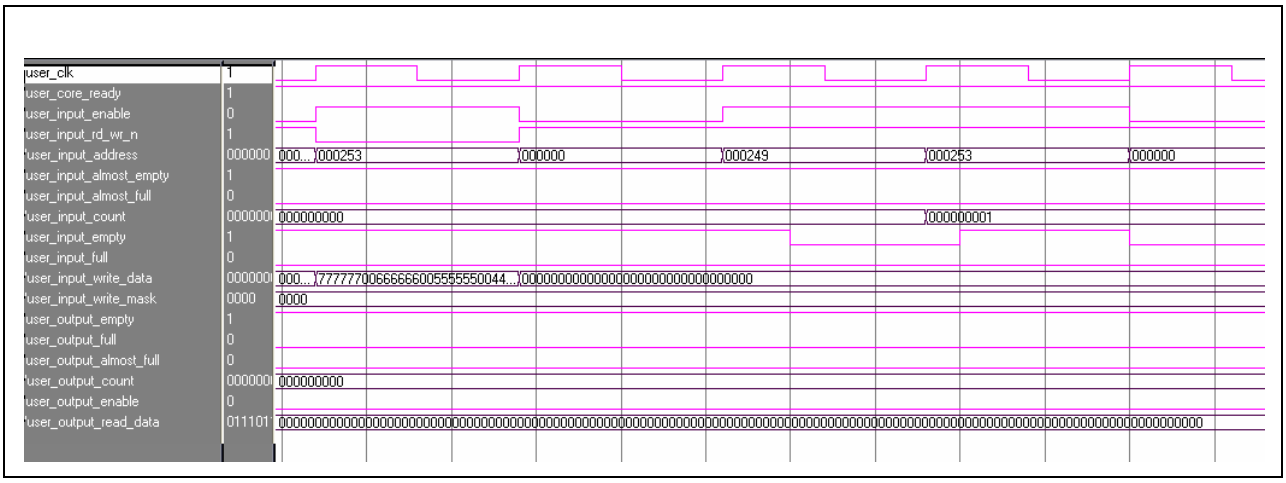

Figure 66: Read Command

Following two writes and two read commands the user waits for user_output_empty to be deasserted which indicates that the read data is now available in the output FIFO.

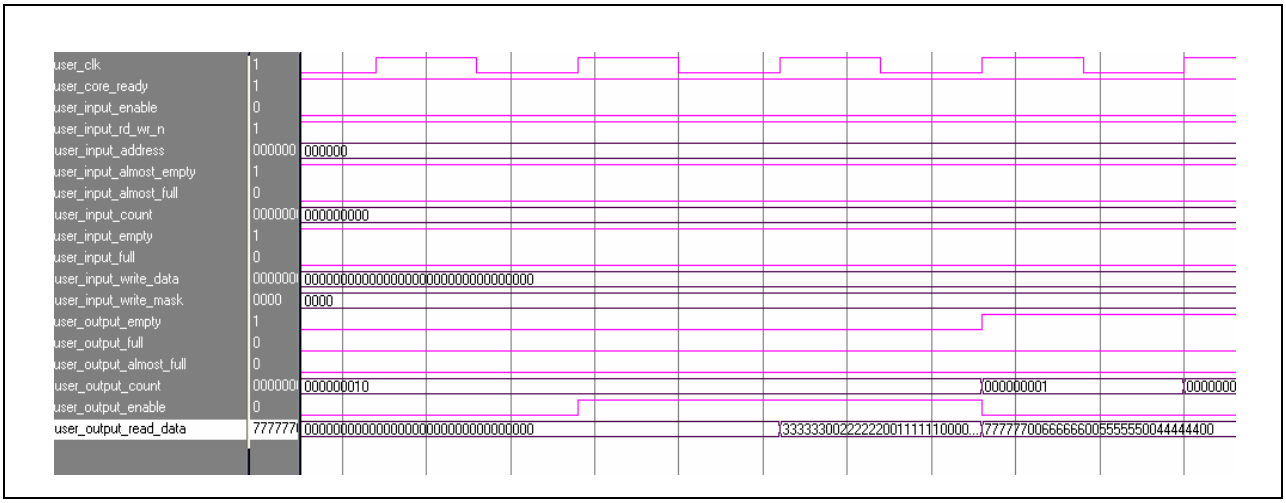

Figure 67: Read Data

The data is read out of the output FIFO by asserting user_output_enable. As shown in Figure 67 the data is valid on the next cycle.

BenDATA2 SDRAM Memory Node - 400MHz

The following description is intended to describe the DDR SDRAM controller core within a DIMEtalk setting. For further information on the core refer to the *DDR SDRAM Controller Core section* of the *BenDATA-II Reference Guide*. The version of the core chosen - DDR333 or DDR400 - is dependent on the speed grade of the FPGA populated on the module.

Functional Description

The BenDATA2 SDRAM Memory Node, shown in Figure 64, implements an interface to DDR SDRAM as supported on the BenDATA-II module. It has two interfaces - a DIMEtalk interface and a user interface - both of which provide read and write access to SDRAM. The node supports independent clock domains for DIMEtalk and user interfaces, the user interface runs from a clock synchronized to the SDRAM. Due to the row paging operation of the SDRAM there is a performance penalty introduced when swapping between rows which occurs when sequential accesses are requested at different pages in the memory map. For this reason the user is given control of DIMEtalk access to the SDRAM so that when access is requested by DIMEtalk it can be granted at the most opportune moment for the user. However, prior to granting control to DIMEtalk, the user must ensure that the input FIFO is empty and the output FIFO has been flushed of data. With this exception the user interface operates in an identical fashion to that described in the *BenDATA-II Reference Guide*. For convenience this is summarized after the following figure.

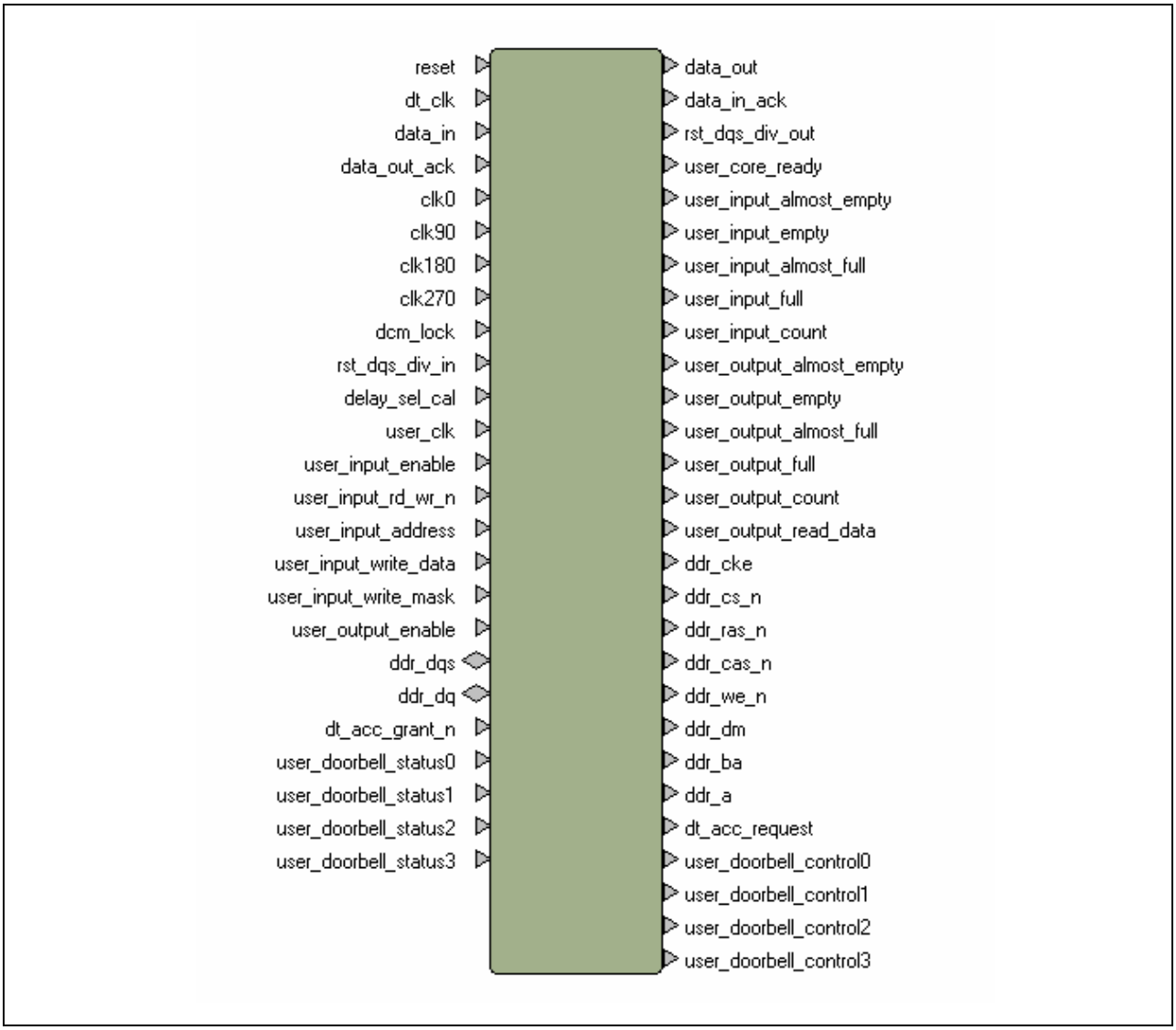

Figure 68: benDATA2 SDRAM Memory Node - 400MHz

The node includes an input FIFO and an output FIFO, write requests are posted into the input FIFO along with associated write data and write mask setup by. This request is strobed into the FIFO by asserting user_input_enable for one cycle. The datapath is 128-bits wide, each 32-bit word is write enabled with an associated mask. Read requests are posted in the same way although write_data and mask busses have no meaning in this context and can be set to any value. Having posted a read request and following the read access from the SDRAM the read data is returned in the output FIFO. This is indicated by user_output_empty being deasserted and the output data is popped from the FIFO by asserting output_enable for one cycle. The data is available in the next cycle.

Additional Modules

This module must be coupled with the BenDATA2 SDRAM Clock Module component in order to provide the required clocks for the SDRAM interface.

Extended Doorbell

There are four doorbell status inputs and four doorbell control outputs. The status inputs can be used to trap a change of state on an input whilst the control outputs can be used as a trigger for aspects of a user's design. For more information on packets see "Request Packet Type 10 (Doorbell class)".

User Generics

The user generics are shown in Table 115.

Name	Type	Description
$bank	Banka/bankb	Not used in the VHDL however should be set up on a core basis via the GUI by right-clicking on the component instance and editing this component parameter which then allows the software to allocate the correct constraints.

Table 115: BenDATA2 SDRAM Memory Node - 400MHz User Generics

Signals

The signal descriptions are provided in Table 116.

Group	Type	Signal Description	Signal Width	Direction
resetl	Reset Connection	resetl : in STD_LOGIC	1-bit	In
dt_clk	Clock Connection	dt_clk : in STD_LOGIC;	1-bit	
DIMEtalk	DIMEtalk Connection	data_in : in STD_LOGIC_VECTOR(31 downto 0);	32-bits	In
		data_out : out STD_LOGIC_VECTOR(31 downto 0);	32-bits	Out
		data_in_ack : out STD_LOGIC;	1-bit	Out
		data_out_ack : in STD_LOGIC;	1-bit	In
SDRAMCoreClocks	User Connection	clk0 : in STD_LOGIC;	1-bit	In
		clk90 : in STD_LOGIC;	1-bit	In
		clk180 : in STD_LOGIC;	1-bit	In
		clk270 : in STD_LOGIC;	1-bit	In
		dcm_lock : in STD_LOGIC;	1-bit	In
ClockCalibration	User Connection	rst_dqs_div_out : out STD_LOGIC;	1-bit	Out
		rst_dqs_div_in : in STD_LOGIC;	1-bit	In

Table 116: BenDATA2 SDRAM Memory Node - 400MHz Signals

Group	Type	Signal Description	Signal Width	Direction
ClockCalibrationDelay	User Connection	delay_sel_cal : in STD_LOGIC_VECTOR(4 downto 0);	5-bits	In
user_clk	Clock Connection	user_clk : in STD_LOGIC;	1-bit	In
UserInterface	User Connection	user_core_ready : out STD_LOGIC;	1-bit	Out
		user_input_enable : in STD_LOGIC;	1-bit	In
		user_input_almost_empty : out STD_LOGIC;	1-bit	Out
		user_input_empty : out STD_LOGIC;	1-bit	Out
		user_input_almost_full : out STD_LOGIC;	1-bit	Out
		user_input_full : out STD_LOGIC;	1-bit	Out
		user_input_count : out STD_LOGIC_VECTOR(8 downto 0);	9-bits	Out
		user_input_rd_wr_n : in STD_LOGIC;	1-bit	In
		user_input_address : in STD_LOGIC_VECTOR(23 downto 0);	24-bits	In
		user_input_write_data : in STD_LOGIC_VECTOR(127 downto 0);	128-bits	In
		user_input_write_mask : in STD_LOGIC_VECTOR(3 downto 0);	4-bits	In
		user_output_enable : in STD_LOGIC;	1-bit	In
		user_output_almost_empty : out STD_LOGIC;	1-bit	Out
		user_output_empty : out STD_LOGIC;	1-bit	Out
		user_output_almost_full : out STD_LOGIC;	1-bit	Out
		user_output_full : out STD_LOGIC;	1-bit	Out
		user_output_count : out STD_LOGIC_VECTOR(8 downto 0);	9-bits	Out
		user_output_read_data : out STD_LOGIC_VECTOR(127 downto 0);	128-bits	Out

Table 116: BenDATA2 SDRAM Memory Node - 400MHz Signals

Group	Type	Signal Description	Signal Width	Direction
SDRAMInterface	User Connection	ddr_dqs : inout STD_LOGIC_VECTOR(3 downto 0);	4-bits	
		ddr_dq : inout STD_LOGIC_VECTOR(31 downto 0);	32-bits	
		ddr_cke : out STD_LOGIC;	1-bit	Out
		ddr_cs_n : out STD_LOGIC;	1-bit	Out
		ddr_ras_n : out STD_LOGIC;	1-bit	Out
		ddr_cas_n : out STD_LOGIC;	1-bit	Out
		ddr_we_n : out STD_LOGIC;	1-bit	Out
		ddr_dm : out STD_LOGIC_VECTOR(3 downto 0);	4-bits	Out
		ddr_ba : out STD_LOGIC_VECTOR(1 downto 0);	2-bits	Out
		ddr_a : out STD_LOGIC_VECTOR(13 downto 0);	14-bits	Out
ArbitrationInterface	User Connection	dt_acc_request : out STD_LOGIC;	1-bit	Out
		dt_acc_grant_n : in STD_LOGIC;	1-bit	In
Doorbells	User Connection	user_doorbell_status0 : in STD_LOGIC;	1-bit	
		user_doorbell_status1 : in STD_LOGIC;	1-bit	
		user_doorbell_status2 : in STD_LOGIC;	1-bit	
		user_doorbell_status3 : in STD_LOGIC;	1-bit	
		user_doorbell_control0 : out STD_LOGIC;	1-bit	
		user_doorbell_control1 : out STD_LOGIC;	1-bit	
		user_doorbell_control2 : out STD_LOGIC;	1-bit	
		user_doorbell_control3 : out STD_LOGIC;	1-bit	

Table 116: BenDATA2 SDRAM Memory Node - 400MHz Signals

Support Files

The support file names and devices used are listed in Table 117.

Location	Device usage
DIMEtalk\library\SDRAM\sdram_if_bendata2_400.vhd	All devices
DIMEtalk\common\retime.vhd	All devices
DIMEtalk\common\bretime.vhd	All devices
DIMEtalk\common\resync_rise.vhd	All devices
DIMEtalk\common\dtnode_slave_control.vhd	All devices
DIMEtalk\common\pkg_dimetalk_global.vhd	All devices
DIMEtalk\library\SDRAM\400MHz\ddrcore.ngc	All devices

Table 117: BenDATA2 SDRAM Memory Node - 400MHz Support Files

Location	Device usage
DIMEtalk\library\SDRAM\fifo128x511.edn	All devices
DIMEtalk\library\SDRAM\fifo192x511.edn	All devices
DIMEtalk\library\SDRAM\comparator.edn	All devices
DIMEtalk\library\SDRAM\dcmx1y0_2vp70.edf	All devices
DIMEtalk\library\SDRAM\dcmx3y0_2vp100.edn	All devices

Table 117: BenDATA2 SDRAM Memory Node - 400MHz Support Files

1.10 How to Place Components in DIMEtalk Networks

This section contains examples of the components which are typically placed down in a DIMEtalk network. The networks shown here illustrate the following components:

- BenDATA-V4 memory components
- PCI-X edge and clock components
- BenDATA-II memory and clock components
- BenBLUE-V4 memory components
- H100 components

1.10.1 BenDATA-V4 Memory Components

Figure 69 shows the Virtex-4 DDR2 memory components connected in an example network. A single DDR2 memory clock module is used for all of the memory banks.

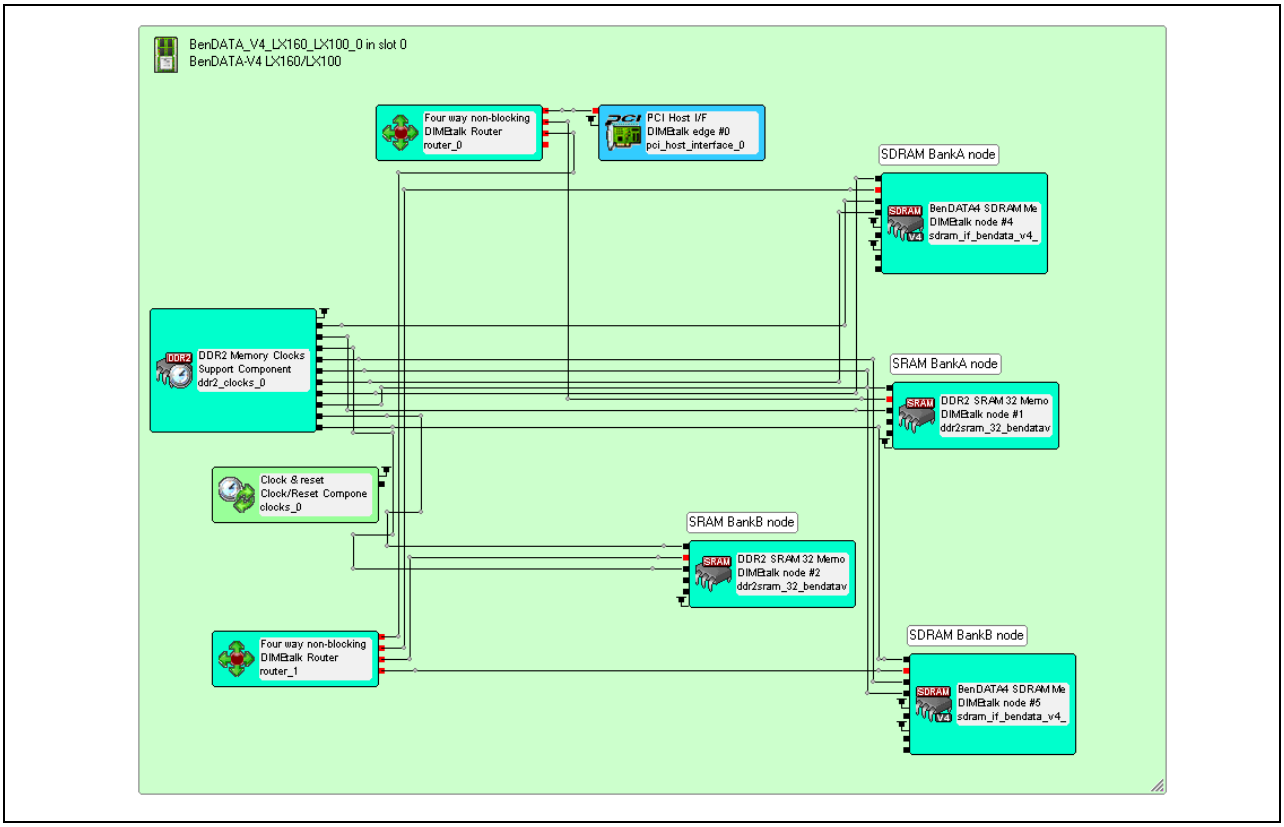

Figure 69: Virtex-4 DDR2 Memory Components

Both the SDRAM and the SRAM components have a bank parameter which defaults to '**banka**'. It is important that the second bank of each memory type placed down has this parameter set to '**bankb**' to allow DIMEtalk to correctly wire up the signals. To do this right-click on the component, select '**edit**' and then click on the 'parameters' tab. This example design is available in the 'projects\Component_Tests\ddr2_memory\bendatav4' directory of the DIMEtalk installation.

1.10.2 PCI-X Edge and PCI-X Clock Components

Figure 70 shows a PCI-X edge component placed down in a network with a PCI-X clock component which provides a clocking mechanism for the PCI-X edge. The PCI-X clock component outputs one signal - hostif_clk200 - which should be connected to the ifclk200 terminal of the PCI-X edge node.

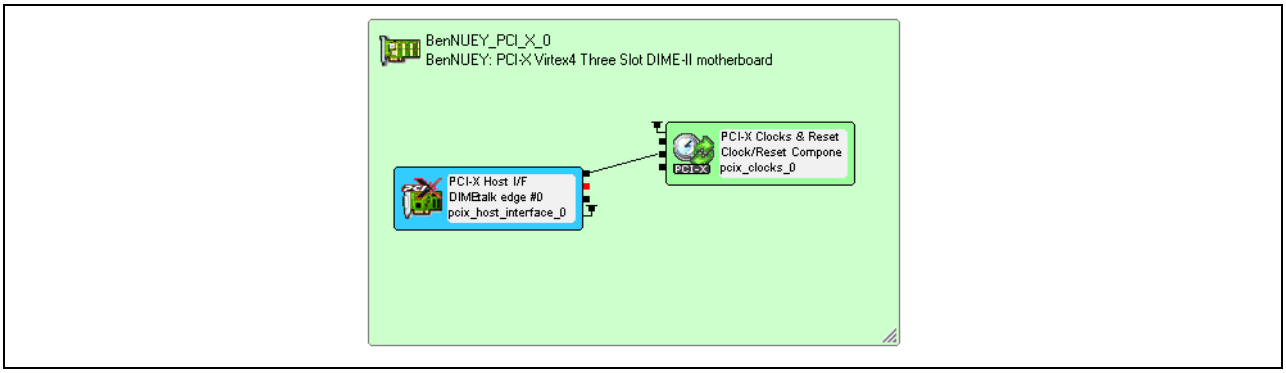

Figure 70: PCI-X Edge and PCI-X Clock Components

1.10.3 PCI-X SRAM Component

The PCI-X clocks component provides four signals: idly_clk, mem_clk0, mem_clk90 and mem_clk50 for use with the PCI-X SRAM component which is shown in Figure 71. These four signals are grouped together in both the PCI-X clocks component and the PCI-X SRAM component.

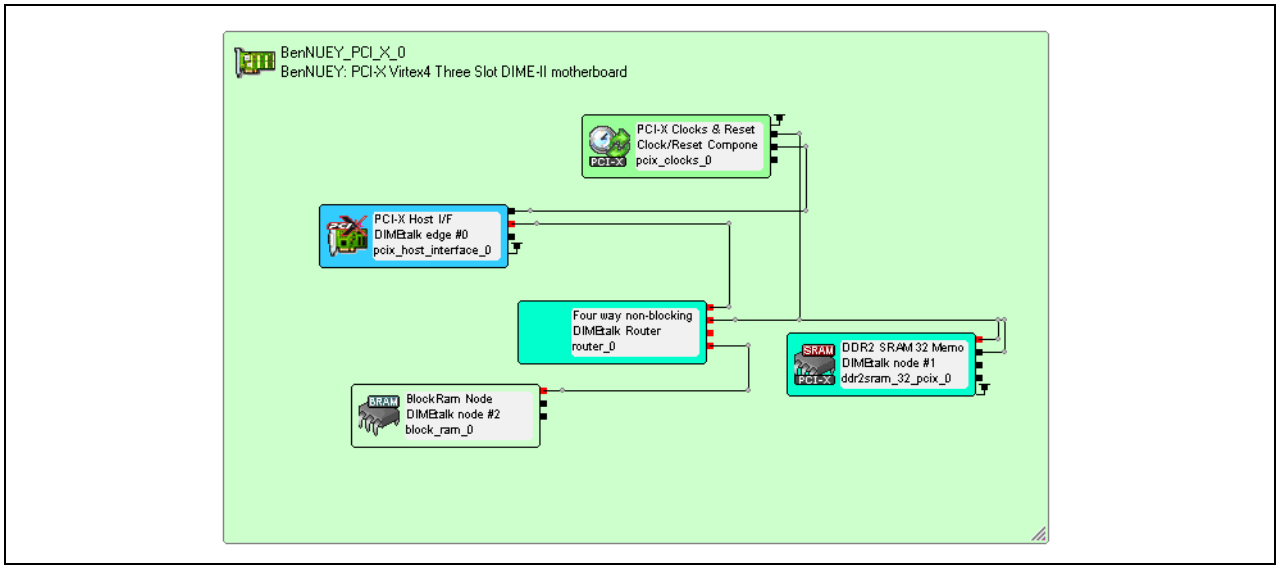

Figure 71: PCI-X SRAM Component

This example design is available in the 'projects\Component_Tests\ddr2_memory\bennuey_pcix' directory of the DIMEtalk installation.

1.10.4 BenDATA-II Memory and Clock Components

Figure 72 shows the BenDATA-II SDRAM memory and clock components connected in an example network. The BenDATA-II SDRAM clock component outputs five signals - clk, clk90, clk180, clk270 and dcm_lock - which are grouped together for convenience and should be connected to both SDRAM components. Users should create a bus which replaces a single terminal with multiple terminals and allows an output terminal to be connected to more than one place.

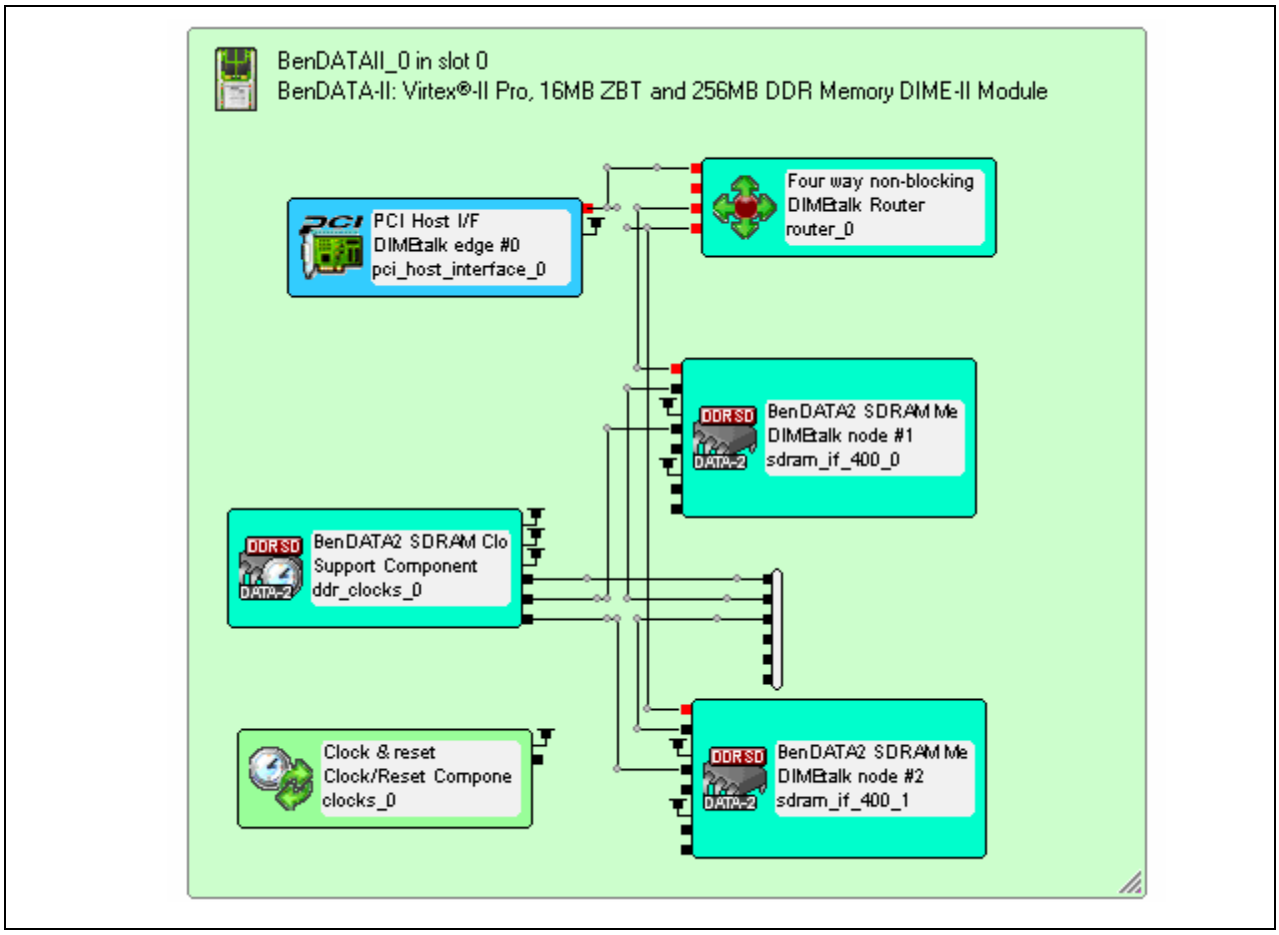

Figure 72: BenDATA-II Memory and Clock Components

The SDRAM components contain a bank parameter which defaults to '**banka**'. It is important that the second bank of memory placed down has this parameter set to '**bankb**' to allow DIMEtalk to wire up the signals correctly. To do this right-click on the component, select '**edit**' and then click on the 'parameters' tab. This example design is available in the 'projects\Component_Tests\ddr_memory' directory of the DIMEtalk installation.

1.10.5 BenBLUE-V4 Memory Components

Figure 73 shows an example network using the DDR2 SRAM memory component and the DDR2 memory clock module for the Primary FPGA on the BenBLUE-V4. Note that a different DDR2 SRAM component is provided for the Primary and Secondary devices. A single DDR2 memory clock module is used for all of the memory banks.

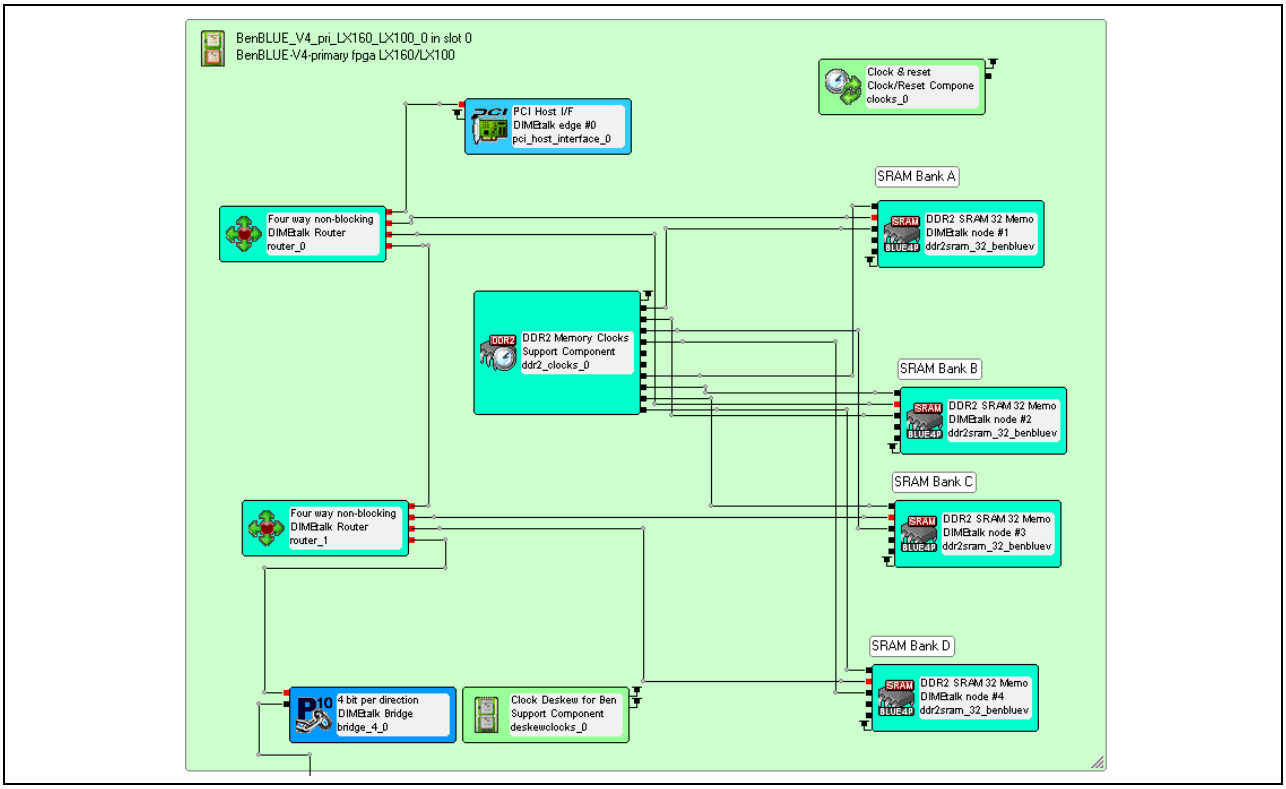

Figure 73: BenBLUE-V4 Primary FPGA Memory Component

On the Primary FPGA the SRAM components have a bank parameter which defaults to '**banka**'. It is important that the second bank of memory placed down has this parameter set to '**bankb**', the third to '**bankc**' and the fourth to '**bankd**' which allows DIMEtalk to correctly wire up the signals. To do this right-click on the component, select '**edit**' and then click on the 'parameters' tab. On the Secondary FPGA the SRAM components have a bank parameter which defaults to '**banke**'. It is important that the second bank of memory placed down has this parameter set to '**bankf**', the third to '**bankg**' and the fourth to '**bankh**'. This example design is available in the 'projects\Component_Tests\ddr2_memory\benbluev4' directory of the DIMEtalk installation. Figure 74 shows the same network for the Secondary FPGA on the BenBLUE-V4.

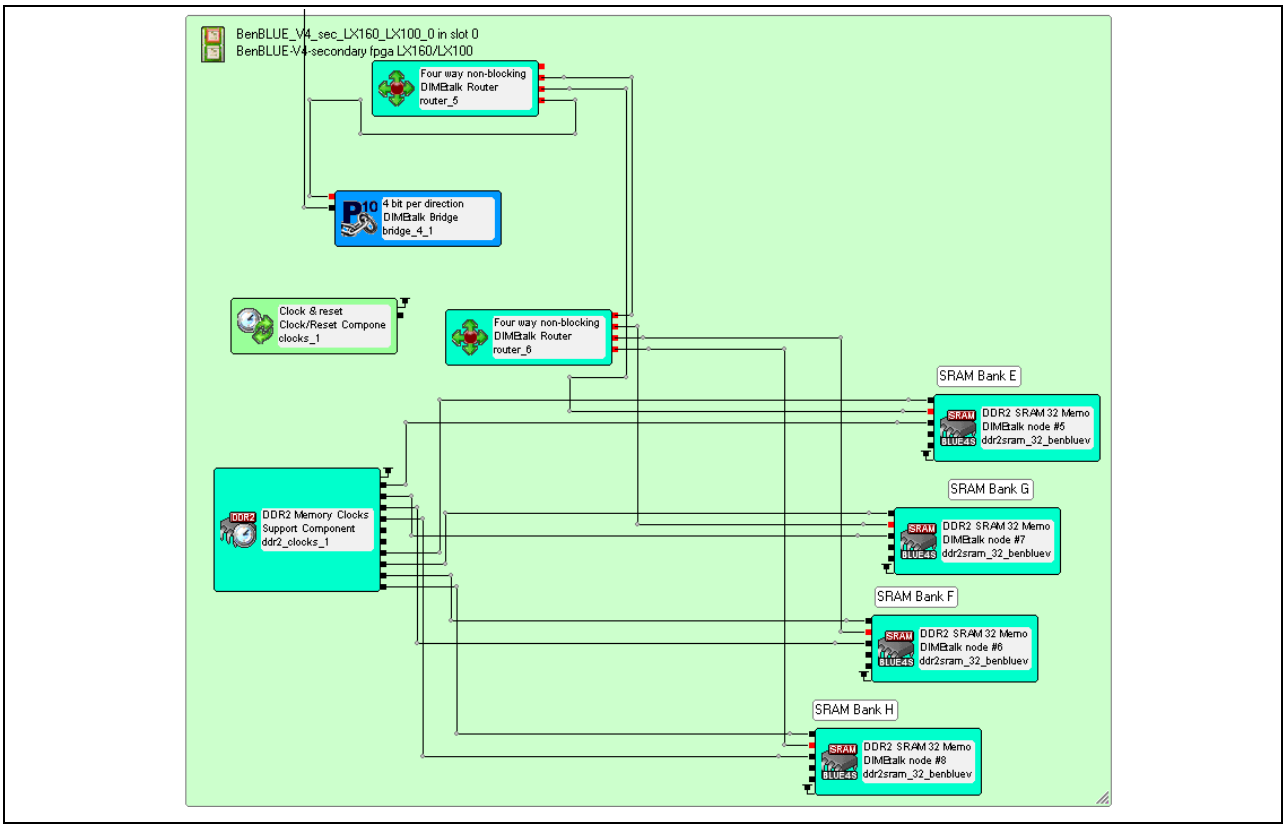

Figure 74: BenBLUE-V4 Secondary FPGA Memory Component

1.10.6 H100 Components

DDR-II SRAM

Figure 75 shows four specific port connections between the H100 clocks component and the SRAM node. These connections provide all the required clocks for each of the SRAM nodes. This example design is available in the 'projects\Component_Tests\ddr2_memory\H100' directory of the DIMEtalk installation.

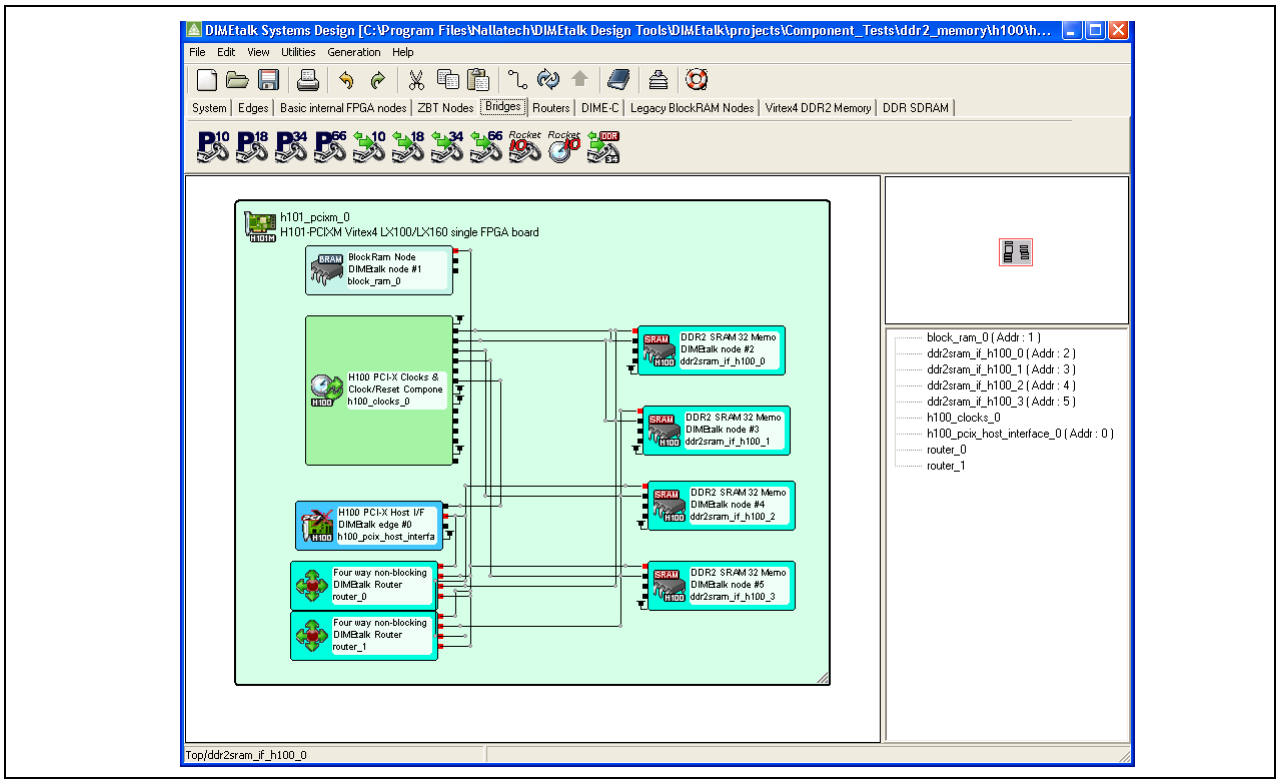

Figure 75: H100 DDR-II SRAM Component

DDR2 SDRAM

Figure 76 shows the specific port connection between the H100 clocks component and the SDRAM node. This connection provides all the required clocks for the SDRAM node. This example design is available in the 'projects\Component_Tests\ddr2_memory\H100' directory of the DIMEtalk installation.

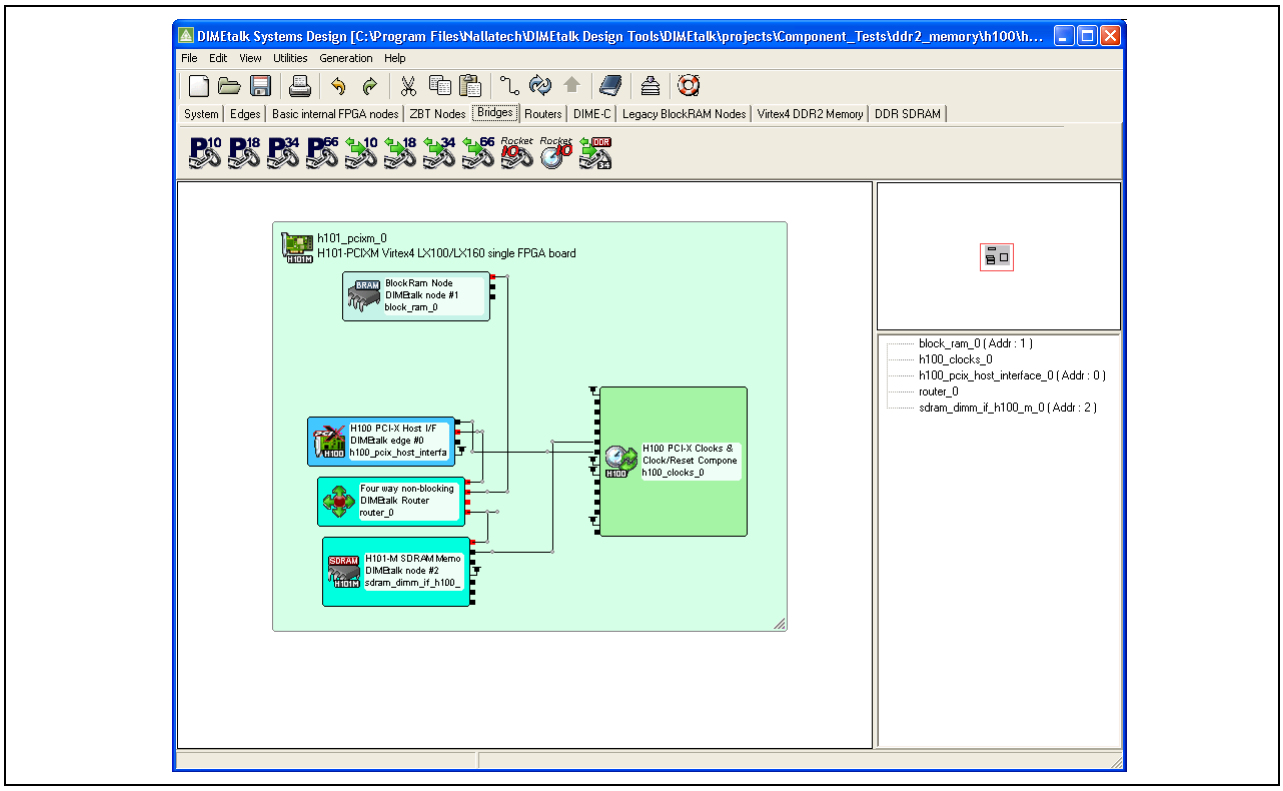

Figure 76: H100 DDR2 SDRAM Component

1.10.7 DIMEtalk Clock Usage on the BenNUEY-PCI-X-V4

The introduction of the BenNUEY-PCI-X-V4 motherboard which uses differential clocks has altered the usage of clock components within DIMEtalk. Previously the Clock Driver component was used for all boards in all circumstances, with additional clock components required for DDR and DDR2 memory, and ZBT SRAM. The Clock Deskew component for BenBLUE modules was required to forward clocking from the BenBLUE Primary FPGA to the Secondary FPGA. These remain unchanged for designs which do not include the BenNUEY-PCI-X-V4, but for designs involving the BenNUEY-PCI-X-V4 please note the following components.

PCI-X Clocks Driver Component

This is used exclusively on the BenNUEY-PCI-X-V4 motherboard. It provides all the clocking required including for the DDR2 SRAM. It should always be used on the BenNUEY-PCI-X-V4 but never on any V4 modules. Note that whilst Clock A & Clock B are differential, Clock C is single ended.

PCI-X Module Clocks Driver Component

This is used exclusively on Virtex-4 modules populated on a BenNUEY-PCI-X-V4 motherboard. It should not be used in any other circumstances. (In particular it should NOT be used on the Secondary FPGA of a BenBLUE-V4.)

Non Virtex-4 Modules Populated on a BenNUEY-PCI-X-V4 Motherboard

Non Virtex-4 modules can be used on a BenNUEY-PCI-X-V4 motherboard but because they do not support the use of differential clocks, Clock C can be used on both the motherboard and the module. Alternatively Clock A can be used with an asynchronous bridge.

Example Designs with Required Clocks Components

Table 118 shows the examples which are available for each DIMEtalk system component, and the motherboards and modules which these components are used with.

Example	Motherboard	System Component	Module	System Component(s)
Example 1	BenERA	Clocks Driver	BenDATA II	Clocks Driver
Example 2	BenERA	Clocks Driver	BenDATA-V4	Clocks Driver
Example 3	BenNUEY-PCI-X-V4	PCI-X Clocks Driver	BenDATA-V4	PCI-X Module Clocks Driver
Example4 – See example file: projects\Component_Tests\ddr2_memory\bennuey_pcix\pcix_benbluev4_sx55_slot0.dt3	BenNUEY-PCI-X-V4	PCI-X Clocks Driver	BenBLUE-V4	**Pri FPGA** - PCI-X Module Clocks, BenBLUE Deskew **Sec FPGA** – Clocks Driver
Example5	BenNUEY-PCI-X-V4	PCI-X Clocks Driver – Use Clock C	BenDATA II	Clocks Driver – Use Clock C
Example6 – See example file: projects\Component_Tests\ddr2_memory\benbluev4\benbluev4_lx100_sram.dt3	BenONE	N/A	BenBLUE-V4	**Pri FPGA** – Clocks Driver, BenBLUE Deskew **Sec FPGA** – Clocks Driver
Example7 - See example file: projects\Component_Tests\ZBT\zbt_benblueIII\benblue3_ben1_zbt.dt3	BenONE	N/A	BenBLUE III	**Pri FPGA** – Clocks Driver, BenBLUE Deskew **Sec FPGA** – Clocks Driver
Example8	BenNUEY-PCI-X-V4	PCI-X Clocks Driver – Use Clock C	BenBLUE III	**Pri FPGA** – Clocks Driver, BenBLUE Deskew – Use Clock C **Sec FPGA** – Clocks Driver – Use Clock C
Example9	BenERA	Clocks Driver	BenBLUE-V4	**Pri FPGA** – Clocks Driver, BenBLUE Deskew **Sec FPGA** – Clocks Driver

Table 118: Example Designs with Required Clocks Components

This page intentionally blank

Section 2

DIMEtalk Detailed Design

In this section:

- Data Packet Format
- dtinfo VHDL Attributes for User Components

2.1 Data Packet Format

2.1.1 Overview

Each DIMEtalk transaction consists of a request followed by a response (if one is required). It is therefore necessary to define request packets and response packets.

2.1.2 Request Packets

General Request Packet Format

Table 119 details the general request packet format. The fields are listed with the corresponding meaning.

Field	Meaning
TransID	General Identifier of transaction type. Differentiates between reads and writes.
Trans	Sub-type. Differentiates between different types of reads or writes.
DataSize	Size of associated data for packet.
SourceID	Source Transaction Identity. Used to identify different transactions.
Address	Address for which data is intended.

Table 119: Request Packet Field Definitions

The request packet bit stream format is shown in Figure 77.

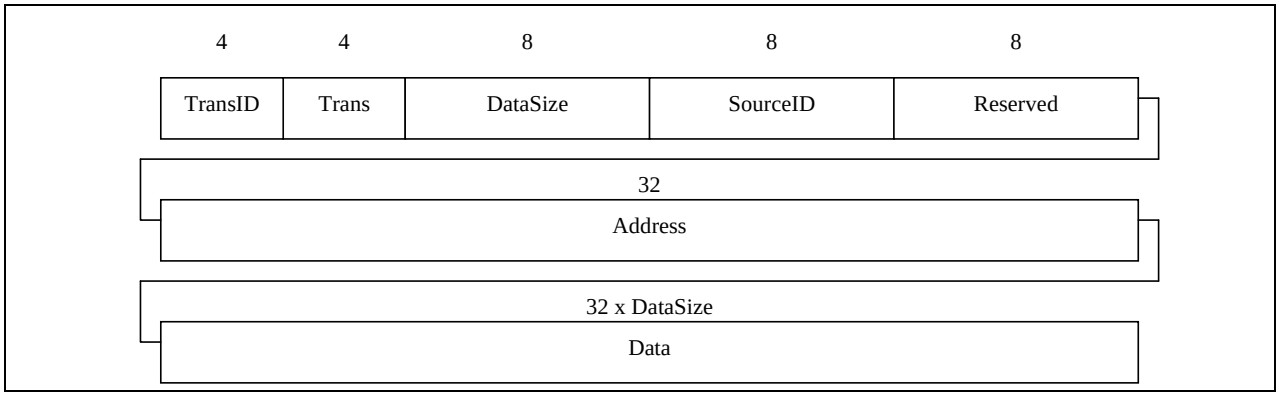

Figure 77: DIMEtalk Request Packets Bit Stream Format

Request Packet Type 2 (Read)

Table 120 details the read packet format. The Trans ID are listed with the corresponding meaning.

Packet type 2 is used for NREAD operations. Type 2 request packets contain no data.

TransID = 0b0010

Trans	Meaning
0b0000-0011	Reserved
0b0100	NREAD transaction
0b010-1011	Reserved
0b1100	ATOMIC inc: post-increment the data
0b1101	ATOMIC dec: post-decrement the data
0b1110	ATOMIC set: set the data (write 0b11111...)
0b1111	ATOMIC set: set the data (write 0b00000...)

Table 120: Trans Values for Reads

The read packet format is shown in Figure 78.

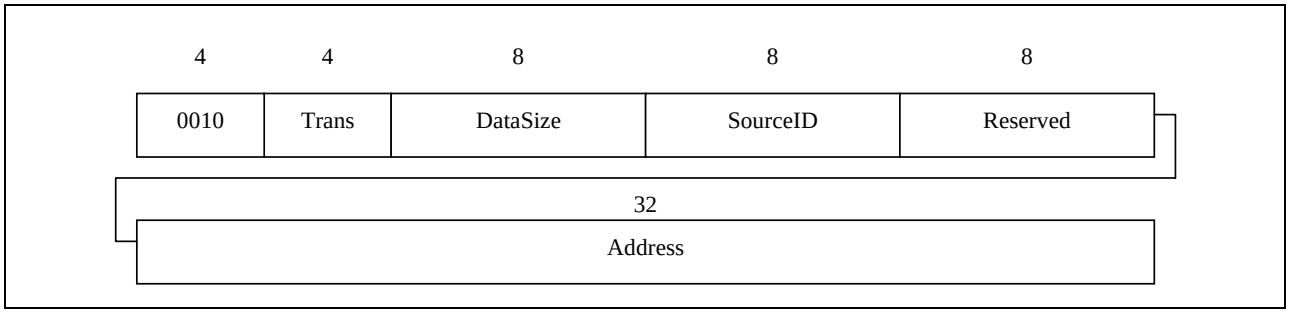

Figure 78: DIMEtalk Read Packet Format

Request Packet Type 5 (Write)

Table 121 details the write packet format. The Trans ID are listed with the corresponding meaning.

Packet type 5 is used for NWRITE, NWRITE_R and atomic operations.

TransID = 0b0101

Trans	Meaning
0b0000	ATOMIC AND
0b0001	ATOMIC OR
0b0010-0011	Reserved
0b0100	NWRITE transaction
0b0101	NWRITE_R transaction
0b0110-1111	Reserved

Table 121: Trans Values for Writes

The write packet format is shown in Figure 79.

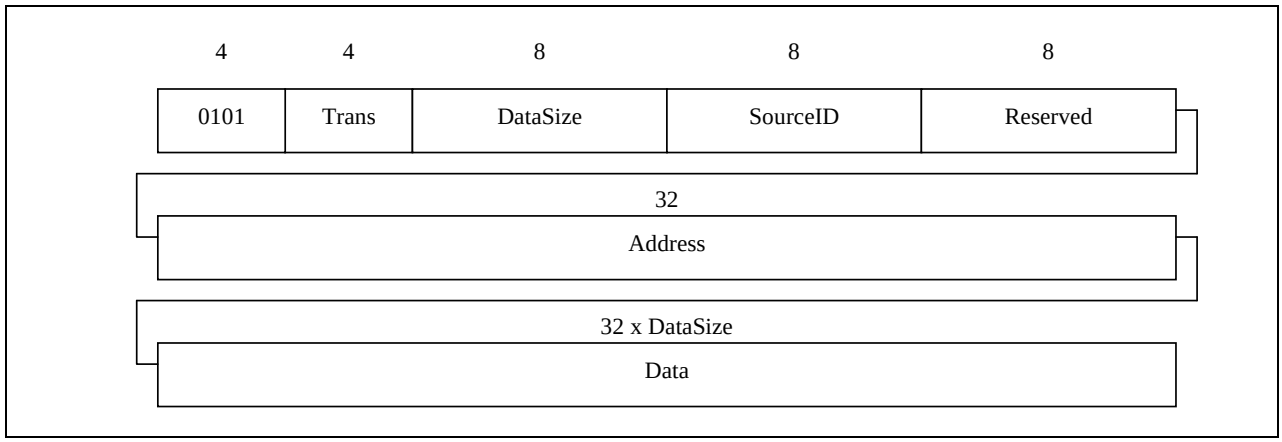

Figure 79: DIMEtalk Write Packet Format

Request Packet Type 8 (Maintenance class)

This packet transaction ID is used to return information on the type of node and the fields supported by returning the DIMETalkR value. The maintenance packet will return a response packet with a data payload of two words. The maintenance packet format is shown in Figure 80.

TransID = 0b1000

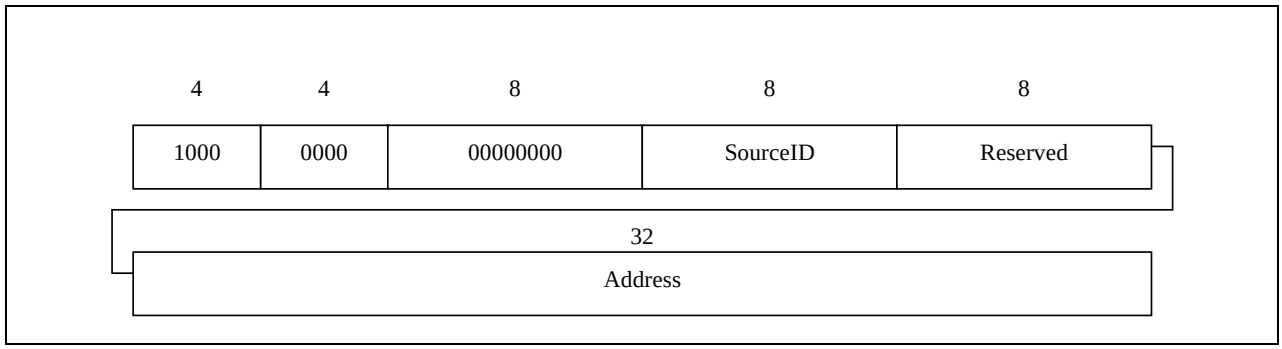

Figure 80: DIMEtalk Maintenance Packet Format

Request Packet Type 10 (Doorbell class)

Doorbell read

The user doorbells are continuously monitored for rise and fall (and a summary change of state bit). If any of the four bits change, the corresponding transition is registered on an SR flip flop (fall, rise, change). If the address is 4 to 7, then the status of one of the user_doorbells (0 to 3) needs to be met before the FSM will proceed, the 'wait on' criteria is the addressed bit changing/rising or falling. Once the bit has changed the corresponding user_db_clear bit is set and the FSM moves to the StallCycle. The doorbell reads are conditions listed in Table 122.

Trans id	Meaning
0x0A	Wait on doorbell bit change, No response.
0x1A	Wait on doorbell bit change, Response.
0x2A	Wait on rising edge on doorbell bit, No response.
0x3A	Wait on rising edge on doorbell bit, Response.

Table 122: Doorbell Read Conditions

Trans id	Meaning
0x4A	Wait on falling edge on doorbell bit, No response.
0x5A	Wait on falling edge on doorbell bit, Response.
Other	Packet not supported, clear all status bits

Table 122: Doorbell Read Conditions

Doorbell write

If the address is set from 0 to 3 and one of the 'DIMEtalk' (i.e. write) doorbells is selected and updated (i.e. address = 0, then dt_doorbell(0) is modified. Depending on the transaction id the doorbells are updated as shown in Table 123.

Trans id	Meaning
0x0A, 0x1A	Invert selected doorbell
0x2A, 0x3A	Set selected doorbell
0x4A, 0x5A	Clear selected doorbell
Other	Clear all (four) doorbells

Table 123: Doorbell Write Conditions

The doorbell packet format is shown in Figure 81.

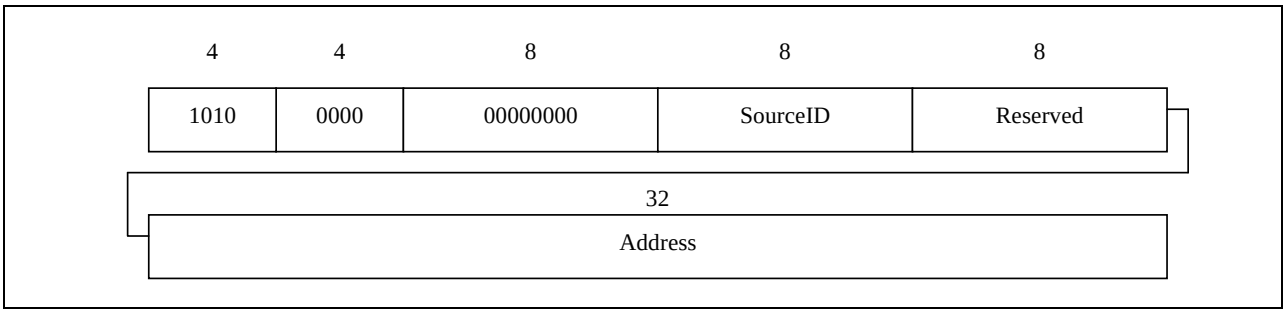

Figure 81: DIMEtalk Doorbell Packet Format

These doorbells are retimed through two stages of pipelining into the user_clk domain. For information on using DIMEtalk Doorbells from software see the DIMEtalk API section of the *FUSE C-C++ API Developer's Guide* or the *TCL Plug-In for FUSE Developer's Guide*.

2.1.3 Response Packets

General Response Packet Format

Table 124 lists the response packet definitions whilst Table 125 lists the status values for the response packets. The fields are listed with their corresponding meaning.

Field	Meaning
TransID	General Identifier of transaction type. Differentiates between types of packets.
Trans	Sub-type. Differentiates between different types of response.
Status	See Table 125 on page 153.
TargetID	Target transactions identity. Used to identify different transactions

Table 124: Response Packet Field Definitions

Field	Meaning
CTransID	Copy of TransID from incoming packet
CTrans	Copy of Trans from incoming packet

Table 124: Response Packet Field Definitions

Field	Meaning
0b0000	DONE
0b0001-0110	Reserved
0b0111	ERROR
0b1000-1011	Reserved
0b1100-1111	Can be used for implementation specific error codes
CTrans	Copy of Trans from incoming packet

Table 125: Status Values for Response Packets

The response packet format is shown in Figure 82.

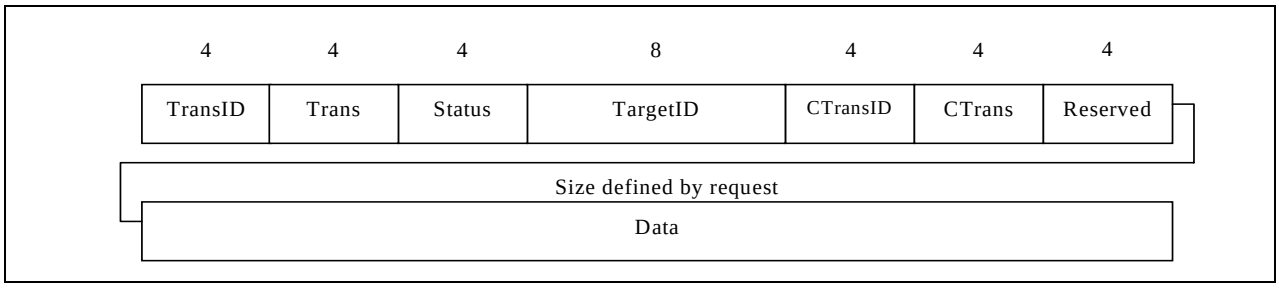

Figure 82: DIMEtalk Response Packets Bit Stream Format

Response Packet Type 13 (Response)

Table 126 lists the response packet type 13 format. The Trans ID are listed with the corresponding meaning.

TransID = 0b1101

Trans	Meaning
0b0000	Response with no payload
0b0001	Response with payload
0b001-1111	Reserved

Table 126: Response Packet Type 13 Format

The response packet type 13 format is shown in Figure 83.

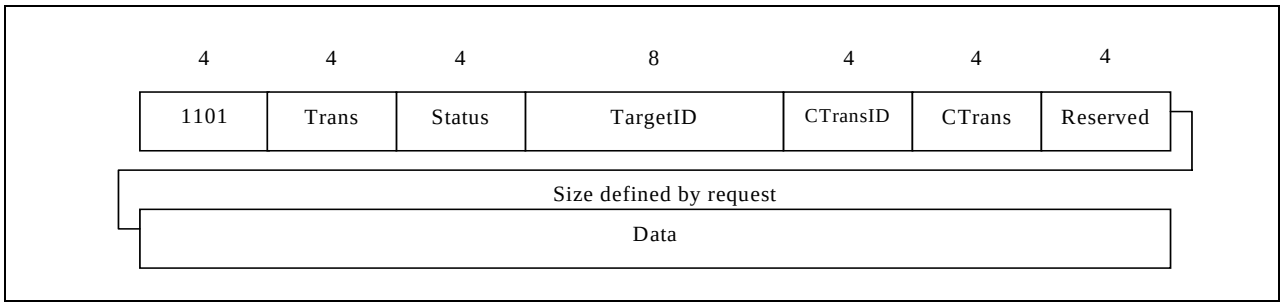

Figure 83: DIMEtalk Response Packet Type 13 Format

2.1.4 Capability Registers

DIMEtalk enables the user to probe each node to determine the type and capability of a DIMEtalk node via its capability register (DIMETalkR) which is listed in Table 127. This is accessed using the Maintenance request packet ID.

Configuration Space Byte Offset	Definition	Bits
0x0	Node type	8
0x1	Reserved	8
0x2	Packet Capability	16
0x4	Request Packet Type 2 capability	16
0x6	Request Packet Type 5 capability	16

Table 127: DIMEtalk Capability Register

This request will return a response packet followed by 8 bytes of data. The registers will have bits set if the node has the defined capability. The packet capabilities of the register bits are listed in Table 128.

Bits	Packet Capability	Default
0-1	Reserved	NA
2	Read	YES
3-4	Reserved	NA
5	Write	YES
6-7	Reserved	NA
8	Maintenance	YES
9	Reserved	NA
10	Doorbell	YES
11-12	Reserved	NA
13	Response	YES
14	Reserved	NA
15	ATOMIC set : set the data (write 0b00000)	NO

Table 128: Packet Capability Register Bits

The packet capabilities of the request packet type 2 register bits are listed in Table 129.

Bits	Packet Capability	Default
0-3	Reserved	NA
4	NREAD transaction	YES
5-11	Reserved	NA
12	ATOMIC inc : post-increment the data	NO
13	ATOMIC dec : post-decrement the data	NO
14	ATOMIC set : set the data (write 0b1111...)	NO
15	ATOMIC set : set the data (write 0b0000...)	NO

Table 129: Request packet type 2 Capability Register Bits

The packet capabilities of the request packet type 5 register bits are listed in Table 130.

Bits	Packet Capability	Default
0	ATOMIC AND	NO
1	ATOMIC OR	NO
2-3	Reserved	NA
4	NWrite transaction	YES
5	NWrite_R transaction	YES
6-15	Reserved	NA

Table 130: Request packet type 5 Capability Register Bits

2.1.5 Transport Format Description

The transport format determines how the network passes data packets from one location to another. In order to do this the transport bit stream must contain a source and destination field and a length to allow routing. This transport header is added to the front of the logical packet (see above). The fields of the transport layer header and their definitions are listed in Table 131.

Fields	Definition	Bits
Destination	The node ID of the data packets destination	8
Source	The node ID of the data packets source	8
Packet size	The size of the data packet, excluding the Transport header, in 32 bit words	8

Table 131: Transport Layer Header

The transport header packet format is shown in Figure 84.

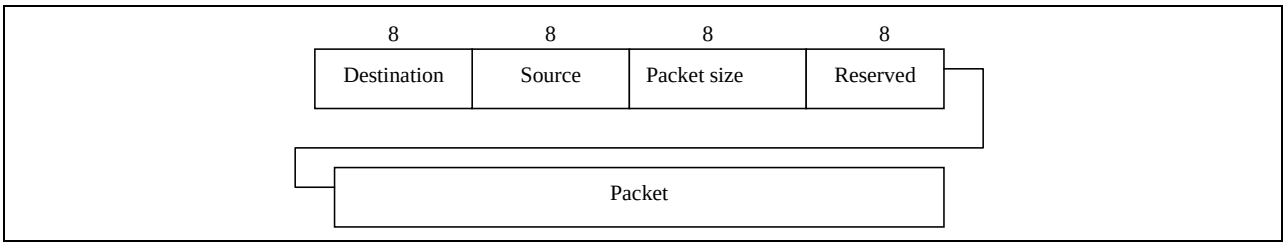

Figure 84: Transport Header Packet Bit Stream Format

2.2 dtinfo VHDL Attributes for User Components

2.2.1 dtinfo Attributes Description

All DIMEtalk components have dtinfo parameters in their VHDL top-level. When the VHDL is imported into DIMEtalk these are parsed into attributes used for the component within the DIMEtalk application. The dtinfo attributes are made up from "dtinfo_", then the component name, followed by an underscore and then the dtinfo attribute name. If a user wishes to create a user VHDL component for seamless use within DIMEtalk they can add dtinfo attributes to their components. The dtinfo attributes fall into two groups - those that are used to describe the general component properties, and those that are used to specify the signal groups and the treatment of the signal groups within DIMEtalk. Table 132 provides information on the first type of dtinfo attribute, including the attribute type, description and an example.

Attribute	Type	Description	Example

icon	string	This specifies the filename of the bitmap to be displayed in the DIMEtalk System Design tool for the component, both in the component bar and when the component is placed in the design. The bitmaps are pulled from the "Icons" folder in the DIMEtalk install directory, typically "C:\Program Files\Nallatech\DIMEtalk Design Tools\DIMEtalk\Icons".	`attribute dtinfo_myvhdlcomp_icon : string;`
			
			`attribute dtinfo_myvhdlcomp_icon of myvhdlcomp : entity is "icon=user3.bmp";`

color	string	This specifies the background color of the component when placed in the DIMEtalk System Design tool. The value is red, green and blue in Hexadecimal.	`attribute dtinfo_myvhdlcomp_color : string;`
			
			`attribute dtinfo_myvhdlcomp_color of myvhdlcomp : entity is "color=0x00f4f0c0";`

description	string	The description is used to describe the component within the DIMEtalk System Design tool.	`attribute dtinfo_myvhdlcomp_description : string;`
			
			`attribute dtinfo_myvhdlcomp_description of myvhdlcomp : entity is "description=FIR Filter VHDL Component";`

shortdesc	string	The shortdesc is an abbreviated version of the description. This is used in the DIMEtalk System Design tool where there is less space for text.	`attribute dtinfo_myvhdlcomp_shortdesc : string;`
			
			`attribute dtinfo_myvhdlcomp_shortdesc of myvhdlcomp : entity is "shortdesc=FIR Filter";`

enumerated	string	This specifies if the component is enumerated. DIMEtalk endpoints are Typically enumerated whilst all other components are not.	`attribute dtinfo_myvhdlcomp_enumerated : string;`
			
			`attribute dtinfo_myvhdlcomp_enumerated of myvhdlcomp : entity is "enumerated=0";`

Table 132: DIMEtalk General Component Attributes - Descriptions

Attribute	Type	Description	Example

type	string	The type attribute is used by DIMEtalk internally to describe the purpose of the component, the types are as follows.
		1: DIMEtalk Router
		2: DIMEtalk Node
		3: DIMEtalk Edge
		4: DIMEtalk Bridge
		5: External Component
		6: Clock/Reset Component
		7: Testbench Component
		8: Support Component
		In practice a user VHDL component should be set to type 5.
			`attribute dtinfo_myvhdlcomp_type : string;`
			
			`attribute dtinfo_myvhdlcomp_type of myvhdlcomp : entity is "type=5";`

Table 132: DIMEtalk General Component Attributes - Descriptions

The second type of attribute is used to describe how the top level ports are grouped, and how that group is treated. Groups are specified by using syntax like "attribute dtinfo of user_myport : signal is "usergroup, myport". When the VHDL is imported, DIMEtalk reads through the top level port list until it comes across a port of the name matching the port named in "dtinfo of" where it then groups them until it sees another port listed in the top level which has a similar "dtinfo of" attribute. As a result of the way it parses the port list from top to bottom, the order of the ports in the top level must match up with the grouping as specified in the dtinfo. The group types are listed in Table 133 and an example of how DIMEtalk processes both group and general component attributes is shown in Figure 85.

Group Name	Type	Visible in DIMEtalk System Design	Port Color in DIMEtalk System Design	Description
reset	Reset	No	n/a	The reset that is used within the component should be set to this group. It will not be shown in the design space.
rawreset	Reset	No	n/a	The rawreset, if used within the component, should be set to this group. It will not be shown in the design space.
clk1	Clock	No	n/a	This specifies that clock1 will drive this port. It will not be shown in the design space.
clk1x2	Clock	No	n/a	This specifies that clock1x2 will drive this port. It will not be shown in the design space.
clk1raw	Clock	No	n/a	This specifies that clock1 raw will drive this port. It will not be shown in the design space.
clk2	Clock	No	n/a	This specifies that clock2 will drive this port. It will not be shown in the design space.
clk2x2	Clock	No	n/a	This specifies that clock2x2 will drive this port. It will not be shown in the design space.

Table 133: DIMEtalk Signal Group Attributes - Descriptions

Group Name	Type	Visible in DIMEtalk System Design	Port Color in DIMEtalk System Design	Description
clk2raw	Clock	No	n/a	This specifies that clock2 raw will drive this port. It will not be shown in the design space.
clk3	Clock	No	n/a	This specifies that clock3 will drive this port. It will not be shown in the design space.
clk3x2	Clock	No	n/a	This specifies that clock3x2 will drive this port. It will not be shown in the design space.
clk3raw	Clock	No	n/a	This specifies that clock3 raw will drive this port. It will not be shown in the design space.
dtport	DIMEtalk	Yes	Red	This specifies that the port will be a DIMEtalk port, and that it has the capability to connect up to a DIMEtalk network. DIMEtalk ports must send data in the DIMEtalk packet format and have the following signals. data_in : in STD_LOGIC_VECTOR(31 downto 0) data_out : out STD_LOGIC_VECTOR(31 downto 0) data_in_ack : out STD_LOGIC data_out_ack : in STD_LOGIC
usergroup	User	Yes	Black	This specifies that the port will be a user port, most commonly used to interface with other components and external pins.

Table 133: DIMEtalk Signal Group Attributes - Descriptions

2.2.2 dtinfo Attributes Example

The general principles of the dtinfo attributes are demonstrated in Figure 85 which shows a Link FIFO user component passing data from one component to another.

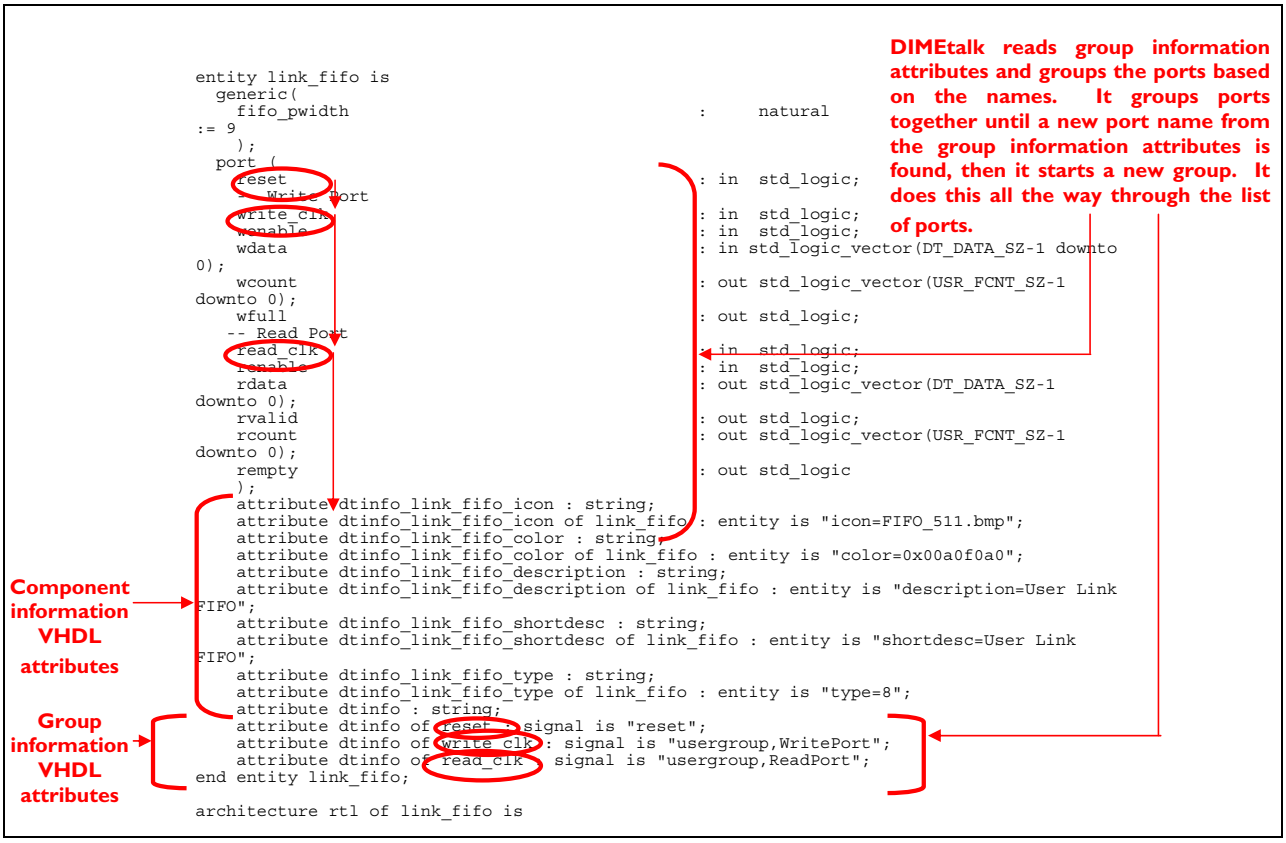

Figure 85: Link FIFO VHDL Example

The example component is shown in Figure 86 as it appears when imported into the DIMEtalk System Design tool, with the attributes added in red. Note the two red boxes highlighted which show the information that appears when the user moves the mouse pointer over the Write Port and Read Port respectively.

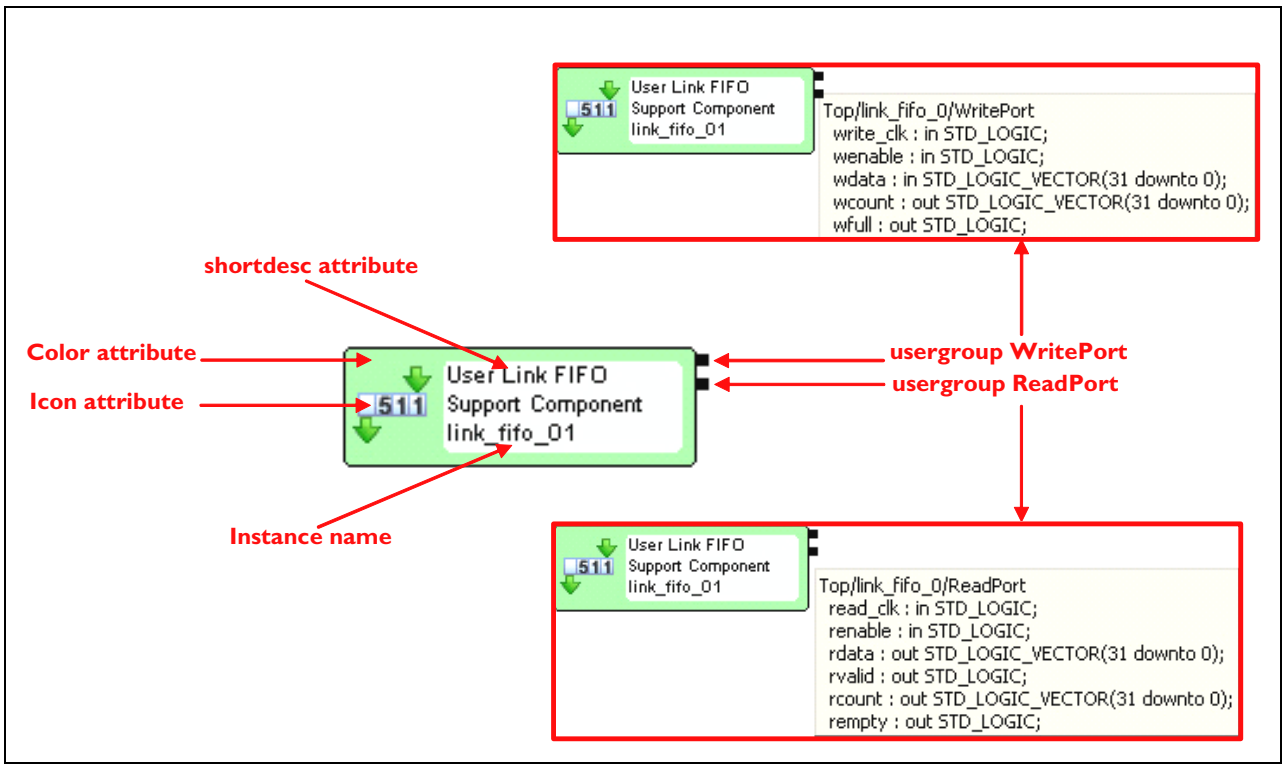

Figure 86: Attributes of Link FIFO DIMEtalk Component

Standard Terms and Conditions

GENERAL

These Terms and Conditions shall apply to all contracts for goods sold or work done by Nallatech Limited. (hereinafter referred to as the "company" or Nallatech) and purchased by any customer (hereinafter referred to as the customer).

Nallatech Limited trading in the style Nallatech (the company), submits all quotations and price lists and accepts all orders subject to the following conditions of contract which apply to all contracts for goods supplied or work done by them or their employees to the exclusion of all other representations, conditions or warranties, express or implied.

The buyer agrees to execute and return any license agreements as may be required by the company in order to authorize the use of those licensable items. If the licensable item is to be resold this condition shall be enforced by the re-seller on the end customer.

Each order received by the company will be deemed to form a separate contract to which these conditions apply and any waiver or any act of non-enforcement or variation of these terms or part thereof shall not bind or prejudice the company in relation to any other contract.

The company reserves the right to re-issue its price list at any time and to refuse to accept orders at a price other than at the price stated on the price list in force at the time of order.

The company reserves the right to vary the specification or withdraw from the offer any of its products without prior warning.

The company reserves the right to refuse to accept any contract that is deemed to be contrary to the company's policies in force at the time.

PRICING

All prices shown on the company's price list, or on quotations offered by them, are based upon the acceptance of these conditions. Any variation of these conditions requested by the buyer could result in changes in the offered pricing or refusal to supply.

All quoted pricing is in Pounds Sterling and is exclusive of Value Added Tax (VAT) and delivery. In addition to the invoiced value the buyer is liable for all import duty as may be applicable in the buyer's location. If there is any documentation required for import formalities, whether or not for the purposes of duty assessment, the buyer shall make this clear at the time of order.

Quotations are made by Nallatech upon the customer's request but there is no obligation for either party until Nallatech accepts the customer's order.

Nallatech reserves the right to increase the price of goods agreed to be sold in proportion to any increase of costs to Nallatech between the date of acceptance of the order and the date of delivery or where the increase is due to any act or default of the customer, including the cancellation or rescheduling by the customer of part of any order.

Nallatech reserves the right (without prejudice to any other remedy) to cancel any uncompleted order or to suspend delivery in the event of any of the customer's commitment with Nallatech not being met.

DELIVERY

All delivery times offered by the company are to be treated as best estimates and no penalty can be accepted for non compliance with them.

Delivery shall be made by the company using a courier service of its choice. The cost of the delivery plus a nominal fee for administration will be added to the invoice issued. Payment of all inward customs duties and fees are the sole responsibility of the buyer. If multiple shipments are requested by the buyer, multiple delivery charges will be made. In the case of multiple deliveries separate invoices will be raised.

If requested at the time of ordering an alternative delivery service can be used, but only if account details are supplied to the company so that the delivery can be invoiced directly to the buyer by the delivery service.

The buyer accepts that any 'to be advised' scheduled orders not completed within twelve months from the date of acceptance of the original order, or orders held up by the buyers lack of action regarding delivery, can be shipped and invoiced by the company and paid in full by the buyer, immediately after completion of that twelve month period.

INSURANCE

All shipments from the company are insured by them. If any goods received by the buyer are in an unsatisfactory condition, the following courses of action shall be taken.

If the outer packaging is visibly damaged, then the goods should not be accepted from the courier, or they should be signed for only after noting that the packaging has sustained damage.

If the goods are found to be damaged after unpacking, the company must be informed immediately.

Under no circumstances should the damaged goods be returned, unless expressly authorized by the company.

If the damage is not reported within 48 hours of receipt, the insurers of the company shall bear no liability.

Any returns made to the company for any reason, at any time shall be packaged in the original packaging, or its direct equivalent and must be adequately insured by the buyer.

Any equipment sent to the company for any purpose, including but not limited to equipment originally supplied by the company must be adequately insured by the buyer while on the premises of the company.

PAYMENT

Nallatech Ltd. terms of payment are 30 days net.

Any charges incurred in making the payment, either

currency conversion or otherwise shall be paid by the buyer.

The company reserves the right to charge interest at a rate of 2% above the base rate of the Bank of Scotland PLC on any overdue accounts. The interest will be charged on any outstanding amount from said due date of payment, until payment is made in full, such interest will accrue on a daily basis.

TECHNICAL SUPPORT

The company offers a dedicompany'scated technical support via telephone and an E-mail address. It will also accept faxed support queries.

Technical support will be given free of charge for 90 days from the date of invoice, for queries regarding the use of the products in the system configuration for which they were sold. Features not documented in the user manual or a written offer of the company will not be supported. Interfacing with other products other than those that are pre-approved by the company as compatible will not be supported. If the development tools and system hardware is demonstrably working, no support can be given with application level problems.

WARRANTY

The company offers as part of a purchase contract 12 months warranty against parts and defective workmanship of hardware elements of a system. The basis of this warranty is that the fault be discussed with the company's technical support staff before any return is made. If it is agreed that a return for repair is necessary then the faulty item and any other component of the system as requested by those staff shall be returned carriage paid to the company. Insurance terms as discussed in the INSURANCE Section will apply.

Returned goods will not be accepted by the company unless this has been expressly authorized.

After warranty repair, goods will be returned to the buyer carriage paid by the company using their preferred method.

Faults incurred by abuse of the product (as defined by the company) are not covered by the warranty.

Attempted repair or alteration of the goods as supplied by the company, by another party immediately invalidates the warranty offered.

The said warranty is contingent upon the proper use of the goods by the customer and does not cover any part of the goods which has been modified without Nallatech's prior written consent or which has been subjected to unusual physical or electrical stress or on which the original identification marks have been removed or altered. Nor will such warranty apply if repair or parts required as a result of causes other than ordinary authorized use including without limitation accident, air conditioning, humidity control or other environmental conditions.

Under no circumstances will the company be liable for any incidental or consequential damage or expense of any kind, including, but not limited to, personal injuries and loss of profits arising in connection with any contract or with the use, abuse, unsafe use or inability to use the company's goods. The company's maximum liability shall not exceed and the customers remedy is limited to, either:

i. repair or replacement of the defective part or product or at the company's option.
ii. return of the product and refund of the purchase price and such remedy shall be the customer's entire and exclusive remedy.

Warranty of the software written by the company shall be limited to 90 days warranty that the media is free from defects and no warranty express or implied is given that the computer software will be free from error or will meet the specification requirements of the buyer.

The terms of any warranty offered by a third party whose software is supplied by the company will be honoured by the company exactly. No other warranty is offered by the company on these products.

Return of faulty equipment after the warranty period has expired, the company may at its discretion make a quotation for repair of the equipment or declare that the equipment is beyond repair.

PASSING OF RISK AND TITLE

The passing of risk for any supply made by the company shall occur at the time of delivery. The title however shall not pass to the buyer until payment has been received in full by the company. And no other sums whatever shall be due from the customer to Nallatech.

If the customer (who shall in such case act on his own account and not as agent for Nallatech) shall sell the goods prior to making payment in full for them, the beneficial entitlement of Nallatech therein shall attach to the proceeds of such sale or to the claim for such proceeds.

The customer shall store any goods owned by Nallatech in such a way that they are clearly identifiable as Nallatech's property and shall maintain records of them identifying them as Nallatech's property. The customer will allow Nallatech to inspect these records and the goods themselves upon request.

In the event of failure by the customer to pay any part of the price of the goods, in addition to any other remedies available to Nallatech under these terms and conditions or otherwise, Nallatech shall be entitled to repossess the goods. The customer will assist and allow Nallatech to repossess the goods as aforesaid and for this purpose admit or procure the admission of Nallatech or its employees and agents to the premises in which the goods are situated.

INTELLECTUAL PROPERTY

The buyer agrees to preserve the Intellectual Property Rights (IPR) of the company at all times and that no contract for supply of goods involves loss of IPR by the company unless expressly offered as part of the contract by the company.

GOVERNING LAW

This agreement and performance of both parties shall be governed by Scottish law.

Any disputes under any contract entered into by the company shall be settled in a court if the company's choice operating under Scottish law and the buyer agrees to attend any such proceedings. No action can be brought arising out of any contract more than 12 months after the completion of the contract.

INDEMNITY

The buyer shall indemnify the company against all claims made against the company by a third party in respect of the goods supplied by the company.

SEVERABILITY

If any part of these terms and conditions is found to be illegal, void or unenforceable for any reason, then such clause or Section shall be severable from the remaining clauses and Sections of these terms and conditions which shall remain in force.

NOTICES

Any notice to be given hereunder shall be in writing and shall be deemed to have been duly given if sent or delivered to the party concerned at its address specified on the invoice or such other addresses as that party may from time to time notify in writing and shall be deemed to have been served, if sent by post, 48 hours after posting.

SOFTWARE LICENSING AGREEMENT

Nallatech Ltd software is licensed for use by end users under the following conditions. By installing the software you agree to be bound by the terms of this license. If you do not agree with the terms of this license, do not install the Software and promptly return it to the place where you obtained it:

1. **License**: Nallatech Ltd grants you a licence to use the software programs and documentation in this package("Licensed materials"). If you have a single license, on only one computer at a time or by only one user at a time;

   if you have acquired multiple licenses, the Software may be used on either stand alone computers or on computer networks, by a number of simultaneous users equal to or less than the number of licenses that you have acquired; and, if you maintain the confidentiality of the Software and documentation at all times.

2. **Restrictions**: This software contains trade secrets in its human perceivable form and, to protect them, except as permitted by applicable law, you may not reverse engineer, disassemble or otherwise reduce the software to any human perceivable form. You may not modify, translate, rent, lease, loan or create derivative works based upon the software or part thereof without a specific run-time licence from Nallatech Ltd.

3. **Copyright**: The Licensed Materials are Copyrighted. Accordingly, you may either make one copy of the Licensed Materials for backup and/or archival purposes or copy the Licensed Materials to another medium and keep the original Licensed Materials for backup and/or archival purposes. Additionally, if the package contains multiple versions of the Licensed Materials, then you may only use the Licensed Materials in one version on a single computer. In no event may you use two copies of the Licensed Materials at the same time.

4. **Warranty**: Nallatech Ltd warrants the media to be free from defects in material and workmanship and that the software will substantially conform to the related documentation for a period of ninety (90) days after the date of your purchase. Nallatech Ltd does not warrant that the Licensed Materials will be free from error or will meet your specific requirements.

5. **Limitations**: Nallatech Ltd makes no warranty or condition, either expressed or implied, including but not limited to any implied warranties of merchantability and fitness for a particular purpose, regarding the Licensed Materials.

   Neither Nallatech Ltd nor any applicable Licenser will be liable for any incidental or consequential damages, including but not limited to lost profits.

6. **Export Control**: The Software is subject to the export control laws of the United States and of the United Kingdom. The Software may not be shipped, transferred, or re-exported directly or indirectly into any country prohibited by the United States Export Administration Act 1969 as amended, and the regulations there under, or be used for any purpose prohibited by the Act.

USER GUIDE CONDITIONS

Information in this User Guide is subject to change without notice. Any changes will be included in future versions of this document. Information within this manual may include technical, typing or printing inaccuracies or errors and no liability will arise therefrom.

This User Guide is supplied without warranty or condition, either expressed or implied, including but not limited to any implied warranties of merchantability and fitness for a particular purpose, regarding the information provided herein.

Under no circumstances will Nallatech Limited be liable for any incidental or consequential damage or expense of any kind, including, but not limited to, loss of profits, arising in connection with the use of the information provided herein.

This page intentionally blank

Index

F

I

P

R

S

T

U

V

Remarks Form

We welcome any comments you may have on our product and its documentation. Your remarks will be examined thoroughly and taken into account for future versions of this product.

DIMEtalk 3.1 Reference Guide NT108-0305 Issue 7	**26/02/07**

Errors Detected

Suggested Improvement

Please send this completed form to:

Nallatech
Boolean House
One Napier Park
Cumbernauld
Glasgow G68 0BH
United Kingdom

If you prefer you may send your remarks via E-mail to support@nallatech.com or by fax to +44 (0) 1236 789599.

If you want Nallatech to reply to your comments, please include your name, address and telephone number.

This page intentionally blank