Increase System Performance & Efficiency Using Distributed Direct Memory Access (DMA)
Block Data Transfer Becomes a Challenge in a Complex System

- Common Problems with Block Data Transfer Efficiency
  - Meeting Individual Bandwidth Requirement
  - Traffic Priority & Overall Performance
  - Hardware Design Complexity & Maintainability
## Solutions to Improve Block Data Transfer Efficiency

<table>
<thead>
<tr>
<th>Solutions</th>
<th>Problems/Issues</th>
</tr>
</thead>
<tbody>
<tr>
<td>State Machine &amp; Multiplexer</td>
<td>- Difficult to Manage the Source Code</td>
</tr>
<tr>
<td></td>
<td>- Difficult to Analyze System Efficiency</td>
</tr>
<tr>
<td>Direct Wire to Dedicated Memories</td>
<td>- Cost of Memory Devices</td>
</tr>
<tr>
<td></td>
<td>- Cost of Logic &amp; Routings</td>
</tr>
<tr>
<td>Using Processors to Move Data</td>
<td>- Longer Latencies</td>
</tr>
<tr>
<td>Direct Memory Access</td>
<td>- May Marginally Increase Hardware</td>
</tr>
</tbody>
</table>
Agenda

- Direct Memory Access (DMA)
- Switch Fabric & Slave-Side Arbitration
- Altera’s Development Tools & IP Supporting DMA
- Examples
  - VGA Controller
  - Ethernet Controller
  - CPRI
What is DMA?
DMA—Direct Memory Access

- Allows a Bounded Number or Sequential Data Transfer Between Regions in the Address Space
  - Typically Between Memories & Peripherals
    - Memory to Memory
    - Peripheral to Peripheral
    - Memory to Peripheral
    - Peripheral to Memory

- Used in Processor & Bus Architecture
Benefit of Using DMA in FPGAs

- **Simplifies the Hardware Design**
  - Eliminates Low-Level Control Logic for Data-Movements
  - Provides Standardized Interface for Peripherals
  - Enables Re-Useable Hardware Blocks
    - Altera’s DMA block

- **Software Engineer-Friendly**
  - Abstracts the Hardware
    - All Data Movement Controlled by Software
  - Only Design High-Level Drivers Once
    - Never if Using Altera’s DMA Block

- **Increases System Performance**
  - Eliminates Processor Bottleneck in Data Movement
  - Offloads Processors to Perform Other Tasks in Parallel
Typical DMA Transaction

- Step 1: CPU Initializes Transfer Command to the DMA Controller, Then Enables the DMA
  - Assigns RD & WR Starting Address, Transfer Length, Etc.
- Step 2: DMA Begins Data Transfer without Processor Intervention
- Step 3: DMA Completes Data Transfer & Sets Completion Status (or IRQ) to the Processor
Same Transaction without DMA

- The Processor Must Execute a Time-Consuming Software Routine
  - Need Variables to Maintain Read & Write Data Counts
  - Need Pointers for Read & Write Address Increments
  - Need Data Structure for Temporary Buffering
Design with DMA vs. without DMA

• Benchmark Results for 16-bit Cyclic Redundancy Check Algorithm (CRC16-CCITT)

<table>
<thead>
<tr>
<th>Aspect</th>
<th>Design With DMA</th>
<th>Design Without DMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMA-Enabled Smart Peripheral (Clock Cycles)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 Byte Message</td>
<td>243</td>
<td>2,838</td>
</tr>
<tr>
<td>512 Byte Message</td>
<td>582</td>
<td>181,753</td>
</tr>
<tr>
<td>1KB Message</td>
<td>922</td>
<td>363,243</td>
</tr>
<tr>
<td>64KB Message</td>
<td>43,925</td>
<td>23,264,648</td>
</tr>
<tr>
<td>Hardware Resources Utilized</td>
<td>~975 Additional Logic Elements</td>
<td>Baseline</td>
</tr>
</tbody>
</table>
Other DMA Enhancements

- Multi-Channel DMA Controllers
- Event- or Time-Triggered DMA
- Two-Dimensional DMA Transfer
- Scatter-Gather DMA Using Descriptors
Multi-Channel DMA Controllers

- Multiple Peripherals Can Time-Share DMA Controller & Bus Utilization
- In the Following Example:
  - DMA Can Serve All 3 Devices & Meet Bandwidth Requirement
  - Ethernet Can Be Allocated with Higher Priority while UART is Filling the FIFO & Video Controller is Filling the Frame Buffer

![Multi-Channel DMA Diagram]

© 2004 Altera Corporation
Multi-Channel DMA Controllers

- Each Channel is Granted to a Portion of Service Time
- A Channel Manager Arbitrates the Service by Monitoring:
  - Transfer Priority, Data Readiness & Memory Type
  - Available with Most High-End Processors & Digital Signal Processors
Event- or Time-Triggered DMA

- Can be Used in Conjunction with Data Acquisition Buffering, Packet or Frame Processing, Time-Based Counter, etc.
- Events Can Come From Multiple Sources or a “Default”
- Example: High-Level Data Link Control Interface

![Diagram of DMA and HDLC Interface](image)
Two-Dimensional DMA Transfer

- Example: JPEG Still Color Image Coding
  - All JPEG DCT-Based Coders Process 8x8 Blocks of Component at a Time
  - Need to Transfer the 8x8 Blocks for Processing
Two-Dimensional DMA Transfer

- Line Length, Line Count & Line Pitch Can Be Embedded in the Transfer Command
- Address Will Be Incremented Based on the 2D Calculation
  - If “End-of-Line”: Address = Address + (Line Pitch – Line Length)
  - Otherwise: Address = Address + 1
- Transfer Can Mix 2D-to-2D, 2D-to-1D, 1D-to-2D (Source-to-Destination)

In This example:
- Line Length = 6
- Line Count = 4
- Line Pitch = 8
- Line Pitch – Line Length = 2
Scatter-Gather DMA Using Descriptors

- DMA Automatically Executes a Series of Operations Based on the Link List of Descriptors Data Structure
- Reduces Initialization Overhead for Individual Transfer Command
- EX: 3G Channel Element Card, Base Station

S/G DMA

Current Descriptor

Transfer Status
Transfer CMD
- RD ADDR
- WR ADDR
- LENGTH

Next Descriptor

Descriptor Data Structure (Maintained by Software)

Descriptor

Transfer Status
Transfer CMD
- RD ADDR
- WR ADDR
- LENGTH

Next Descriptor
Combine Various DMA Enhancements

To Achieve the Best Performance, Combine Various DMA Enhancements:

- Basic DMA Controllers
- Multi-Channel DMA
- Event- or Time-Triggered DMA
- Two-Dimensional DMA
- Scatter-Gather DMA Using Descriptors
Introduction to Distributed DMA

- Distributed DMA Definition:
  - The Integration of DMA Controllers Into Peripherals; or
  - The Distribution of DMA Controllers Across The Bus Hierarchy (non-centralized)
Benefits of Distributed DMA

- Simplifies the Hardware Design
  - Eliminates Low-Level Control Logic for Data-Movements
  - Provides Standardized Interface for Peripherals
  - Enables Re-Useable Hardware Blocks

- Software Engineer-Friendly
  - Abstracts the Hardware
    - All Data Movement Controlled by Software
  - Only Design High-Level Drivers Once
    - Never if Using Altera’s DMA Block

- Increases System Performance
  - Eliminates Processor Bottleneck in Data Movement
  - Offloads Processors to Perform Other Tasks in Parallel

Note: DMA Benefits in Blue Enabled by Distributed DMA
Distributed DMA Example

- Combining the DMA Controller & HDLC Interface Will Become a Distributed DMA Topology
  - Reduces Latencies

HDLC Interface Controller w/ Built-In Distributed DMA Function

- Event Trigger
- Or Build-in Timer

RX FIFO
- FIFO FULL / FIFO EMPTY

TX FIFO

RX

TX

HDLC Frame (RX)

HDLC Frame (TX)

© 2004 Altera Corporation
Centralized DMA vs. Distributed DMA

<table>
<thead>
<tr>
<th>Aspect</th>
<th>Centralized DMA</th>
<th>Distributed DMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware Resources</td>
<td>Multiple Devices Can Share the Same DMA</td>
<td>Overall System May Consume More Logic</td>
</tr>
<tr>
<td>Bus Loading</td>
<td>Double Bus Loading</td>
<td>Single Bus Loading</td>
</tr>
<tr>
<td>Master &amp; Slave Consideration</td>
<td>DMA Controller Has 2 Master Ports: Read &amp; Write</td>
<td>Peripheral Must Have Master Capability</td>
</tr>
</tbody>
</table>

Centralized DMA Mechanism

Distributed DMA Mechanism

© 2004 Altera Corporation
Smart Peripherals with Distributed DMA

- Smart Peripherals Are Capable of Initiating DMA Data Transactions to & from Memory
- Built-In Intelligence
  - Default Event Trigger
  - Default Source or Destination Address
  - Default Data Count & Transfer Type
Advanced Bus Interconnect Architectures
Bus Architecture

- **Bus**
  - A Shared Communication Link that Connects I/O Pins to Memory & Processor Subsystems

- **Advantages of Applying Bus Architecture**
  - Low Cost: Set of Wires is Shared in Multiple Ways
  - Versatility: Well-Defined Interconnection Scheme Allows Devices to be Added or Removed Easily

- **Caveat of Applying Bus Architecture**
  - Without Proper Design, a Bus Architecture May:
    - Create Data Traffic Bottlenecks
    - Limit Maximum I/O Throughput
Bus Design Decisions

- Bus Width & Data Width
- Number of Masters & Arbitration Scheme
- Type of Devices Connected to the Bus
  - Processors & Co-Processors
  - Memories & Buffers
  - High-Speed I/O Pins
  - Low-Speed I/O Pins
- Bus Hierarchy
- Performance & Cost
Basic Components of Bus

- **Master**
  - Initiates a Read or Write Transaction
  - Example: Processors

- **Slave**
  - Responds to a Transaction
  - Example: Memories

- **Arbiter**
  - Arbitrates in Multiple Masters that Want to Initiate Simultaneous Transactions

- **Bridge**
  - Connects Buses & Passes the Transaction on a Bus to the Other Bus

- **The Bus**
  - Provides Physical Wires for Address, Data & Control Signals
  - Example: Tri-Stated Bus, Multiplexed Bus, And/Or Bus, Etc.
Traditional Shared-Bus System

System Bottleneck

Master-Side Arbiter

CPU 0

DMA

CPU 1

Processor System Bus

PROGRAM MEMORY 0

I/O

DATA MEMORY 0

VGA CONTROL

DATA MEMORY 1

CUSTOM FUNCTION

PROGRAM MEMORY 1

© 2004 Altera Corporation
**Traditional Shared-Bus System**

- **Bottleneck**
  - Any Transaction Demands the Same Master-Side Arbiter & Processor System Bus
  - Only One Master Can Operate at a Time
  - System Bus Can Be Blocked by Processor Cache Line Filling or Any Bulk Data Transfer
  - Centralized DMA Architecture Doubles Bus Loading
Switch Fabric & Slave-Side Arbitration Scheme
Switch Fabric & Slave-Side Arbitration Scheme

- **Benefit**
  - Shared Bus & Shared Arbiter Are No Longer the Bottleneck
  - Multiple Master Transactions Can Operate Simultaneously
    - As Long As They Do Not Access the Same Slave in the Same Bus Cycle
  - I/O Devices Can be Grouped Based on Bandwidth Requirement

- **Trade-Off**
  - Hardware Resource Usage Increases
Use Distributed DMA in the Switch Fabric

- The Smart Peripheral with DMA Function Reduces Latency for Memory Access
Altera Development Tools & IP Supporting DMA
Altera’s DMA System Architecture Solution

- Development Tool
  - SOPC Builder

- Hardware
  - DMA Controller IP Core

- Software
  - Header Files
  - Hardware Abstraction Layer (HAL)
  - Generic Device Models

- Bus Interconnect
  - Avalon™ Switch Fabric

© 2004 Altera Corporation
Altera DMA Controller IP Core

- DMA Controller with Avalon Interface
  - Transfers Data with Maximum Pace Allowed by Source & Destination
  - Capable of Performing Slow Streaming Transfers (e.g., an UART)
  - SOPC Builder-Ready, Easy Integration into Any SOPC Builder-Generated System
  - Device Drivers Provided

- Available with Nios II Embedded Processor Core
  - AMPP (Third Party) Stand-Alone DMA Cores Available -
    www.altera.com/ipmegastore

Avalon DMA Controller

© 2004 Altera Corporation
Standard Parameterized DMA
Software Interface

- Hardware Abstraction Layer
  - Automatically Generated by Nios II Integrated Development Environment (IDE)
  - Allows Using familiar C library
    - printf(), fopen(), fwrite(), etc
  - Provides a Simple Interface for Hardware Device Driver
  - Avoid Direct Access to Hardware Registers for Code Reusability
Nios II HAL Architecture

HAL Details:
- Nios® II Run-Time Library
- Integrated with Newlib ANSI C Library
- Unix-Like API Provided for Development

Provides Following Features:
- Interrupt Handling
- Alarm Facilities
- System & Device Initialization
- Device Access

HAL API
- _exit()
- close()
- closedir()
- fstat()
- getpid()
- gettimeofday()
- ioctl()
- isatty()
- kill()
- lseek()
- open()
- opendir
- read()
- readdir()
- readdir()
- rewinddir()
- sbrk()
- settimeofday()
- stat()
- usleep()
- wait()
- write()
Nios II HAL Architecture

Benefit of Using Nios HAL Architecture
- Tightly Integrated with SOPC Builder to Ensure Software/Hardware Correlation
- Changes in Hardware Propagate to HAL Automatically
- Improve Code Reusability by Avoiding Direct Access to Hardware Registers
DMA Programming Model with HAL

- The HAL API for DMA Access
  - Defined in sys/alt_dma.h, Generated by Nios II IDE
- DMA Device Driver Provided by Altera
  - Integrates to HAL System Library

*Code demonstrated is for illustration purpose*
Avalon Switch Fabric

- High-Performance Interconnect
  - Supporting a Wide Range of Transfer Types Between a Wide Range of Peripherals
  - Parameterizable, Synchronous Operation
  - Scalable Up to 128-Bit Wide Address & Data Path
    - Separate Address & Data Paths
    - Separate Read & Write Data Paths

- Single- & Multi-Mastered Systems

- Optimized for FPGAs

- Complete Specification Available from www.altera.com

Avalon Signal Types

- reset
- chipselect
- address
- byteenable
- read
- readdata
- write
- writedata
- data
- waitrequest
- readyfordata
- dataavailable
- datavalid
- flush
- begintransfer
- endofpacket
- irq
- irqnumber
- clk
- resetrequest

Most Signals Available In Positive or Negative Form

© 2004 Altera Corporation
Avalon Switch Fabric Transfers

- **Fundamental Transfers**
  - Master Read/Write with Switch Fabric Controlled Wait States
  - Slave Read/Write with 0 Wait States

- **Fundamental Transfer Variants**
  - Slave Read/Write with:
    - Fixed Wait States, Peripheral-Controlled Wait States, Setup Time, Setup & Hold Times

- **Advanced Avalon Transfers**
  - Latency-Aware Transfers
  - Streaming Transfers
  - Avalon Tri-State Bridge Transfers for Off-Chip Peripherals
Avalon Switch Fabric

- Interconnect Logic
  - Allows Masters & Slaves to Communicate without Prior Knowledge or Re-Design
  - Supports Independent Development of Peripherals
  - Advances Design Re-Use
Example: VGA Controller
Example: VGA Controller

- Requirements
  - High Bandwidth

- Solution
  - Custom VGA Peripheral
  - Avalon Streaming Mode

AN 333: Developing Peripherals for SOPC Builder
VGA Monitor Pixel Organization

640 Pixels

480 Pixels

(0.0)

(640,480)
VGA Peripheral

- Peripheral Functional Blocks
  - Peripheral Task Logic
  - Register File
  - Avalon Interface
  - Software Driver Functions
Block Diagram of VGA Display Driver Hardware

Register File & Address Mapping

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Offset</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vga_data</td>
<td>0x0</td>
<td>Write-Only</td>
<td>Writing to this Register Stores the 8-Bit Value into the FIFO Line-Buffer</td>
</tr>
</tbody>
</table>

Streaming Avalon Port
- endofpacket
- readyfordata
- Writedata [7:0]
- write
- chipselect
- address

Write
FIFO
Read
VGA Timing & Palette
VGA Port
- vga_clock
- hsync
- vsync
- sync
- blank
- R [7:0]
- G [7:0]
- B [7:0]
## Avalon Signals for VGA Controller Peripheral

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Avalon Signal Type</th>
<th>Bit-Width</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock</td>
<td>clk</td>
<td>1</td>
<td>input</td>
<td>Input clock for writing to the FIFO</td>
</tr>
<tr>
<td>reset</td>
<td>reset</td>
<td>1</td>
<td>input</td>
<td>Peripheral reset</td>
</tr>
<tr>
<td>cs</td>
<td>chipselect</td>
<td>1</td>
<td>input</td>
<td>Chip select</td>
</tr>
<tr>
<td>write</td>
<td>write</td>
<td>1</td>
<td>input</td>
<td>Write-enable signal</td>
</tr>
<tr>
<td>fifo data</td>
<td>writedata</td>
<td>8</td>
<td>input</td>
<td>8-bit write data</td>
</tr>
<tr>
<td>fifo_not_full</td>
<td>readyfordata</td>
<td>1</td>
<td>output</td>
<td>Streaming transfer signal indicating that new data is accepted</td>
</tr>
<tr>
<td>lastpixel</td>
<td>endofpacket</td>
<td>1</td>
<td>output</td>
<td>Streaming transfer signal indicating that the last pixel of a frame was received</td>
</tr>
<tr>
<td>vga_clock</td>
<td>Export</td>
<td></td>
<td>input</td>
<td>Input clock for VGA timing and reading data from FIFO</td>
</tr>
<tr>
<td>hsync</td>
<td>Export</td>
<td>1</td>
<td>output</td>
<td>Horizontal synchronization signal (output)</td>
</tr>
<tr>
<td>sync</td>
<td>Export</td>
<td>1</td>
<td>output</td>
<td>Vertical synchronization signal (output)</td>
</tr>
<tr>
<td>Blank</td>
<td>Export</td>
<td>1</td>
<td>output</td>
<td>Logical AND of hsync and vsync (output)</td>
</tr>
<tr>
<td>R</td>
<td>Export</td>
<td>8</td>
<td>output</td>
<td>Red color (output)</td>
</tr>
<tr>
<td>G</td>
<td>Export</td>
<td>8</td>
<td>output</td>
<td>Green color (output)</td>
</tr>
<tr>
<td>B</td>
<td>Export</td>
<td>8</td>
<td>output</td>
<td>Blue color (output)</td>
</tr>
</tbody>
</table>
Ports Tab for the VGA Controller Peripheral
Example System with Streaming VGA Controller
Example:
Ethernet Controller
Ethernet Frame Data Transmission Path with Single Master Architecture

Figure 5. Ethernet Frame Data Transmission Path with Single Master Architecture

Ethernet Frame Data Transmission
Ethernet Frame Data Transmission Path Using DMA & Simultaneous Multi-Mastering

Figure 6. Ethernet Frame Data Transmission Path Using DMA & Simultaneous Multi-Mastering  Note (1)
Ethernet Design with DMA Controller

Ethernet DMA Masters

<table>
<thead>
<tr>
<th>Usage</th>
<th>Module Name</th>
<th>Description</th>
<th>Bus Type</th>
<th>Base</th>
<th>End</th>
<th>Req</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>cpu</td>
<td>Altera Nios 2.0 CPU</td>
<td>avalon</td>
<td>0x00000400</td>
<td>0x0000041F</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>uart1</td>
<td>UART (RS-232 serial...)</td>
<td>avalon</td>
<td>0x00000420</td>
<td>0x0000042F</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>seven_seg_pio</td>
<td>PIO (Parallel I/O)</td>
<td>avalon</td>
<td>0x00000440</td>
<td>0x0000044F</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>timer1</td>
<td>Interval timer</td>
<td>avalon</td>
<td>0x00000460</td>
<td>0x0000046F</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>led_pio</td>
<td>PIO (Parallel I/O)</td>
<td>avalon</td>
<td>0x00000480</td>
<td>0x0000048F</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>button_pio</td>
<td>PIO (Parallel I/O)</td>
<td>avalon</td>
<td>0x000004A0</td>
<td>0x000004A0</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>led_pio</td>
<td>PIO (Parallel I/O)</td>
<td>avalon</td>
<td>0x000004C0</td>
<td>0x000004C0</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>boot_monitor</td>
<td>On-Chip Memory (RAM)</td>
<td>avalon</td>
<td>0x00000000</td>
<td>0x0000041F</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>ext_mem_bus</td>
<td>Avalon Tri-Static Bridge</td>
<td>avalon</td>
<td>0x00000440</td>
<td>0x0000044F</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>ext_ram</td>
<td>SRAM (one or two ID...)</td>
<td>avalon_tristate</td>
<td>0x00000440</td>
<td>0x0000044F</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>ext_flash</td>
<td>Flash Memory</td>
<td>avalon_tristate</td>
<td>0x00000440</td>
<td>0x0000044F</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>UniqueId</td>
<td>Unique ID</td>
<td>avalon_tristate</td>
<td>0x00000440</td>
<td>0x0000044F</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>nclk_card_bus</td>
<td>Avalon Tri-Static Bridge</td>
<td>avalon</td>
<td>0x00000450</td>
<td>0x0000045F</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>ethernet</td>
<td>Ethernet Interface (CS...)</td>
<td>avalon_tristate</td>
<td>0x00000460</td>
<td>0x0000046F</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>sth_dma</td>
<td>DMA</td>
<td>avalon</td>
<td>0x00000480</td>
<td>0x0000048F</td>
<td>26</td>
</tr>
</tbody>
</table>
System Interconnect Block Diagram

Figure 13. System Interconnect Block Diagram
## Multi-Master Ethernet Design Arbitration Settings

<table>
<thead>
<tr>
<th>Use</th>
<th>Module Name (bus)</th>
<th>Description</th>
<th>Bus Type</th>
<th>Base</th>
<th>End</th>
<th>R#</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>cpu</td>
<td>Altera Nex 2.0 CPU</td>
<td>avalon</td>
<td>0x000000400</td>
<td>0x00000041F</td>
<td>26</td>
</tr>
<tr>
<td>✓</td>
<td>uart</td>
<td>UART (RS-232 serial port)</td>
<td>avalon</td>
<td>0x000000120</td>
<td>0x00000014F</td>
<td>25</td>
</tr>
<tr>
<td>✓</td>
<td>seven_seg_pio</td>
<td>PICO (Parallel I/O)</td>
<td>avalon</td>
<td>0x000000040</td>
<td>0x00000006F</td>
<td>26</td>
</tr>
<tr>
<td>✓</td>
<td>timer</td>
<td>Interval timer</td>
<td>avalon</td>
<td>0x000000040</td>
<td>0x00000004F</td>
<td>25</td>
</tr>
<tr>
<td>✓</td>
<td>led_pio</td>
<td>PICO (Parallel I/O)</td>
<td>avalon</td>
<td>0x000000040</td>
<td>0x00000004F</td>
<td>25</td>
</tr>
<tr>
<td>✓</td>
<td>button_pio</td>
<td>PICO (Parallel I/O)</td>
<td>avalon</td>
<td>0x000000040</td>
<td>0x00000004F</td>
<td>25</td>
</tr>
<tr>
<td>✓</td>
<td>led_pio</td>
<td>PICO (Parallel I/O)</td>
<td>avalon</td>
<td>0x000000040</td>
<td>0x00000004F</td>
<td>25</td>
</tr>
<tr>
<td>✓</td>
<td>boot_monitor_rom</td>
<td>On-Chip Memory (RAM)</td>
<td>avalon</td>
<td>0x000000000</td>
<td>0x00000003FF</td>
<td>25</td>
</tr>
<tr>
<td>✓</td>
<td>ext_mem_bus</td>
<td>Avalon Tri-State Bridge</td>
<td>avalon</td>
<td>0x000000040</td>
<td>0x00000004F</td>
<td>25</td>
</tr>
<tr>
<td>✓</td>
<td>ext_ram</td>
<td>SRAM (one or two IDT)</td>
<td>avalon_tristate</td>
<td>0x0000000400</td>
<td>0x00000007FFFF</td>
<td>25</td>
</tr>
<tr>
<td>✓</td>
<td>ext_flash</td>
<td>Flash Memory</td>
<td>avalon_tristate</td>
<td>0x00000010000</td>
<td>0x0000001FFFF</td>
<td>25</td>
</tr>
<tr>
<td>✓</td>
<td>nic</td>
<td>Ethernet Interface (CSM)</td>
<td>avalon_tristate</td>
<td>0x00000005000</td>
<td>0x000000051FF</td>
<td>25</td>
</tr>
<tr>
<td>✓</td>
<td>eth_dma</td>
<td>DMA</td>
<td>avalon</td>
<td>0x00000004A0</td>
<td>0x00000004EF</td>
<td>16</td>
</tr>
</tbody>
</table>
Figure 15. Simplified View of Arbitration during Conflict between DMA & CPU
The Nios CPU instruction master and DMA controller read master both request continuous access to the shared SRAM.
DMA Routine for Transmitting Frames

// Step 3: write the data out

// Half-word pointer to the data out
w = (r16 *)ethernet_frame;

// Begin new SMM tutorial DMA code
{
    // Declare "ethDMA" as pointer to "eth_dma"
    np_dma *ethDMA = na_eth_dma;

    // Wait for any pending DMA transfers to complete
    while((!(ethDMA->np_dmastatus & np_dmastatus_done_mask) && ethDMA->np_dmastatus != 0));

    // Perform DMA transfer
    nr_dma_copy_range_to_1(na_eth_dma, 2, (void *)w, (void *)e->np_cs8900iodata0, frame_length);
}

// End new SMM tutorial DMA code
DMA Routine for Receiving Frames

// Half-word pointer to the receive data buffer
w = g_frame_buffer;

// Begin new SMM tutorial DMA code
{
    // Declare "ethDMA" as pointer to "eth_dma"
    np_dma *ethDMA = na_eth_dma;

    // Wait for any pending DMA transfers to complete
    while(!((ethDMA->np_dmastatus & np_dmastatus_done_mask) && ethDMA->np_dmastatus != 0));

    // Perform DMA transfer
    nr_dma_copy_1_to_range(na_eth_dma, 2, (void *)e->np_cs8900iodata0, (void *)w, frame_length);
}
// End new SMM tutorial DMA code
Example: CPRI Controller
Example: CPRI Controller

- **CPRI**
  - Common Packet Radio Interface
  - Open Standard Between Radio Equipment & Radio Equipment Controller

- **System Requirements**
  - Data Management Function for Base-Station
  - HDLC-Like Framer for Control Frames
CPRI Controller

- HDLC Controller as a Smart Peripheral
- Implemented as Half DMA Engine
  - Separate Buffers for Read Data & Write Data
  - Reduced Design Size: Went from >2,000 Logic Elements (LEs) to <200 LEs
CPRI Architectural Solution

- Memory Buffer
- HDLC Controller
- Memory Buffer
- UMTS Framer
- CPRI
- UMTS Framer
- CPRI

© 2004 Altera Corporation
CPRI Interface With Smart Peripheral

**Transmit**

- **Memory Buffer**
- **DMA**
- **Read Buffer**
  - CRC-16
  - Bit Stuff
  - Frame
- **CRC-16**
- **Bit Stuff**
- **Frame**
- **UMTS Framer**
- **CPRI**

**Receive**

- **Memory Buffer**
- **DMA**
- **Frame**
  - Bit Strip
  - CRC-16
  - Write Buffer
- **FIFO**
- **UMTS Framer**
- **CPRI**

**Data Frame to UMTS Framer**

- **Start of Frame**
- **User Data**
- **FCS**
- **End of Frame**

© 2004 Altera Corporation
Benefits of this Methodology

- Simplifies Core
  - Uses Nios II CPU as a System Component
  - Uses Nios II CPU (Already Present) + Simple Peripherals

- Removes Clocking Constraints
  - No Re-Timing for UMTS Framer Necessary

- Abstracts the Hardware
  - Software Engineer-Friendly
  - Only Design High-Level Drivers Once
Related Documentation

- Application Notes & Tutorials
  - AN 333: Developing Peripherals for SOPC Builder
  - AN 184: Simultaneous Multi-Mastering with the Avalon Bus
  - Tutorial: Simultaneous Multi-Mastering with the Nios Processor
Thank You !