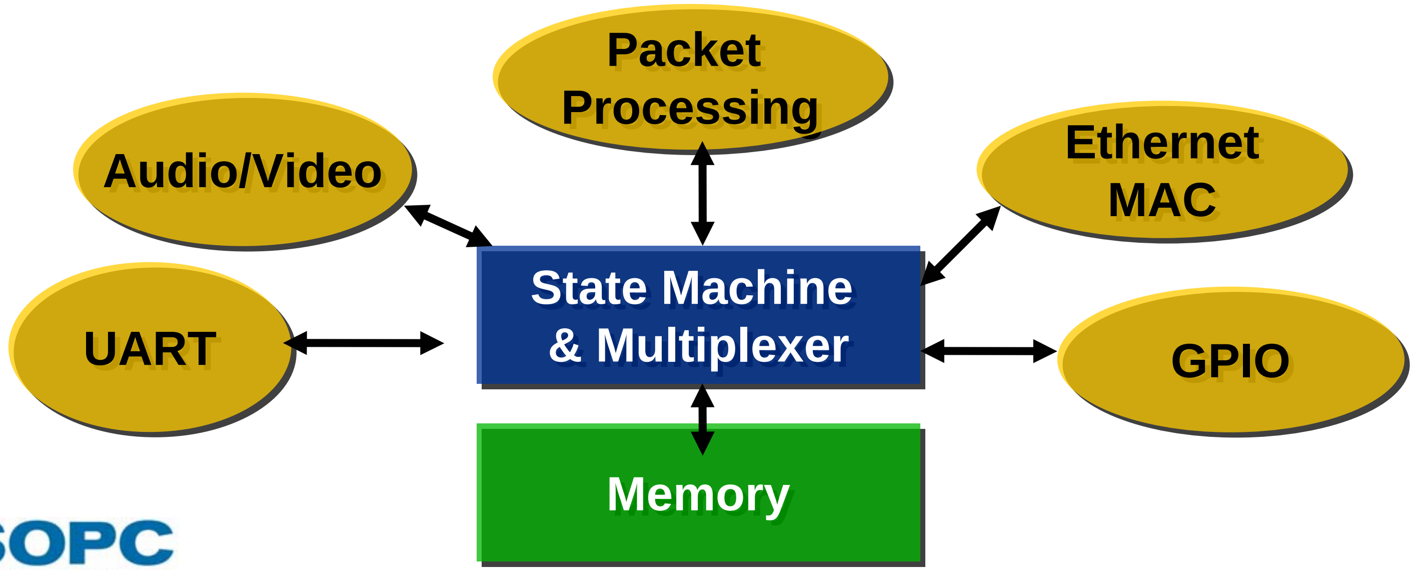
Increase System Performance & Efficiency Using Distributed Direct Memory Access (DMA)

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

Solutions	Problems/Issues
State Machine & Multiplexer	■ Difficult to Manage the Source Code ■ Difficult to Analyze System Efficiency
Direct Wire to Dedicated Memories	■ Cost of Memory Devices ■ Cost of Logic & Routings
Using Processors to Move Data	■ Longer Latencies
Direct Memory Access	■ May Marginally Increase Hardware

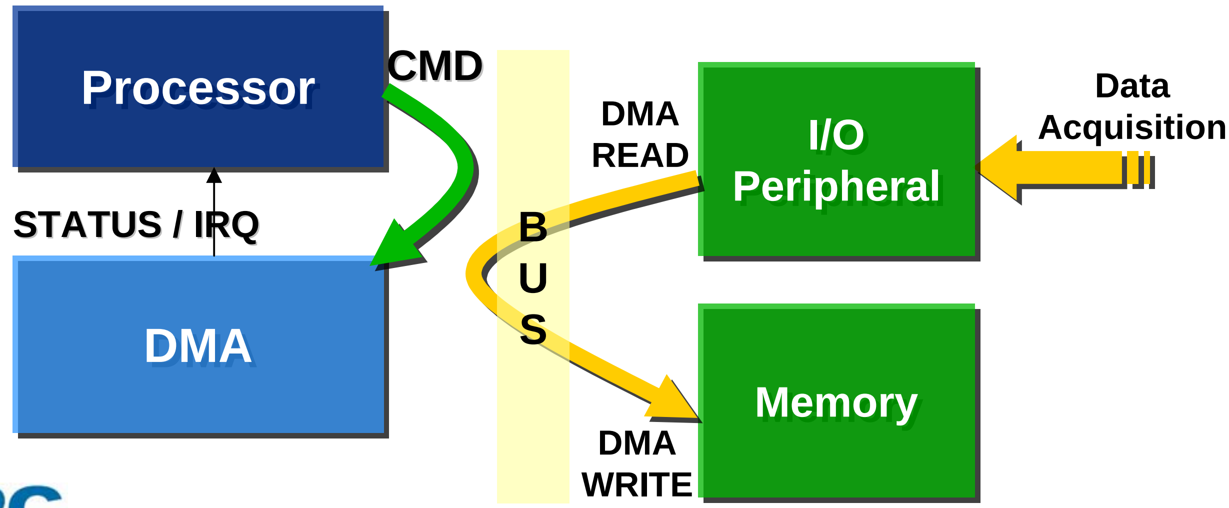
Agenda

- 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

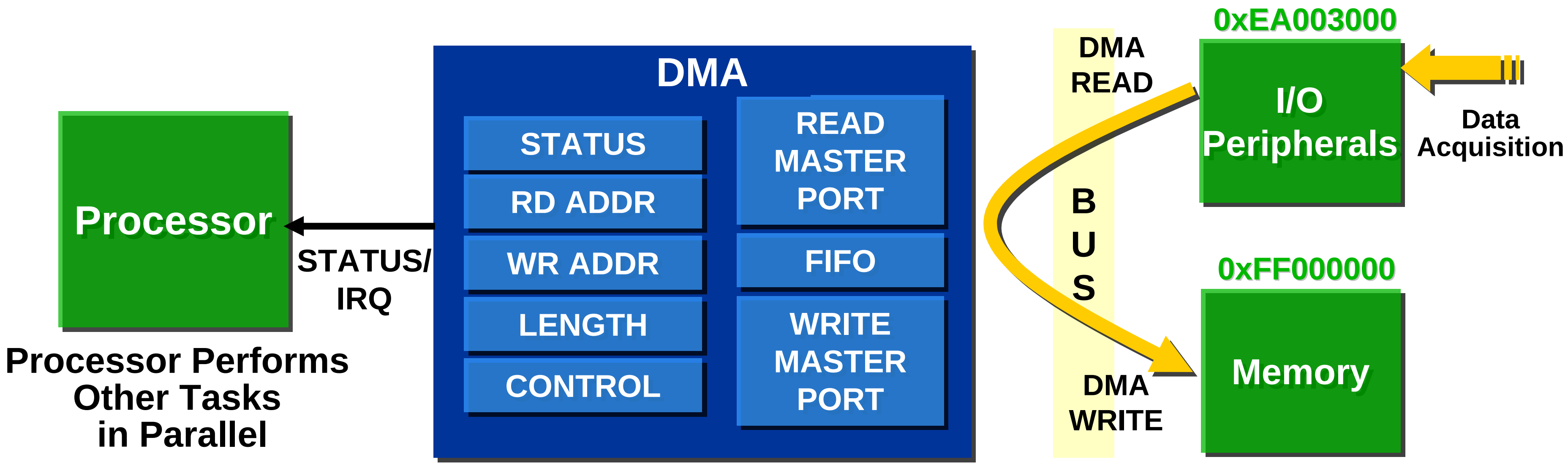


Benefit of Using DMA in FPGAs

- 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

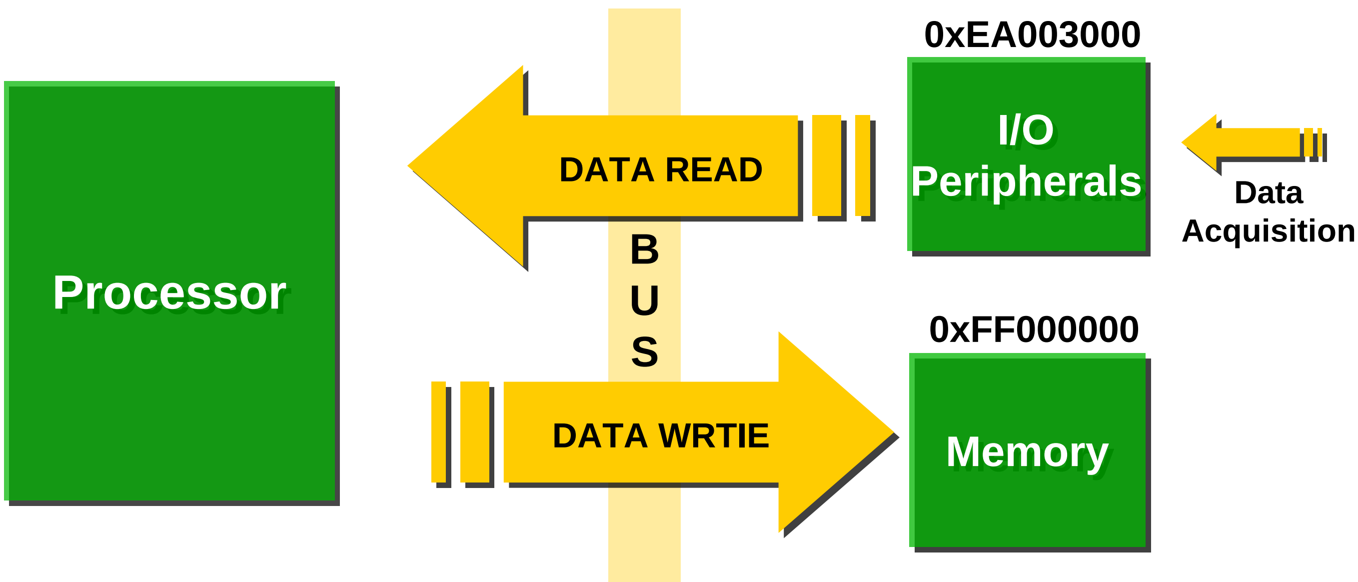
Typical DMA Transaction

- 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)

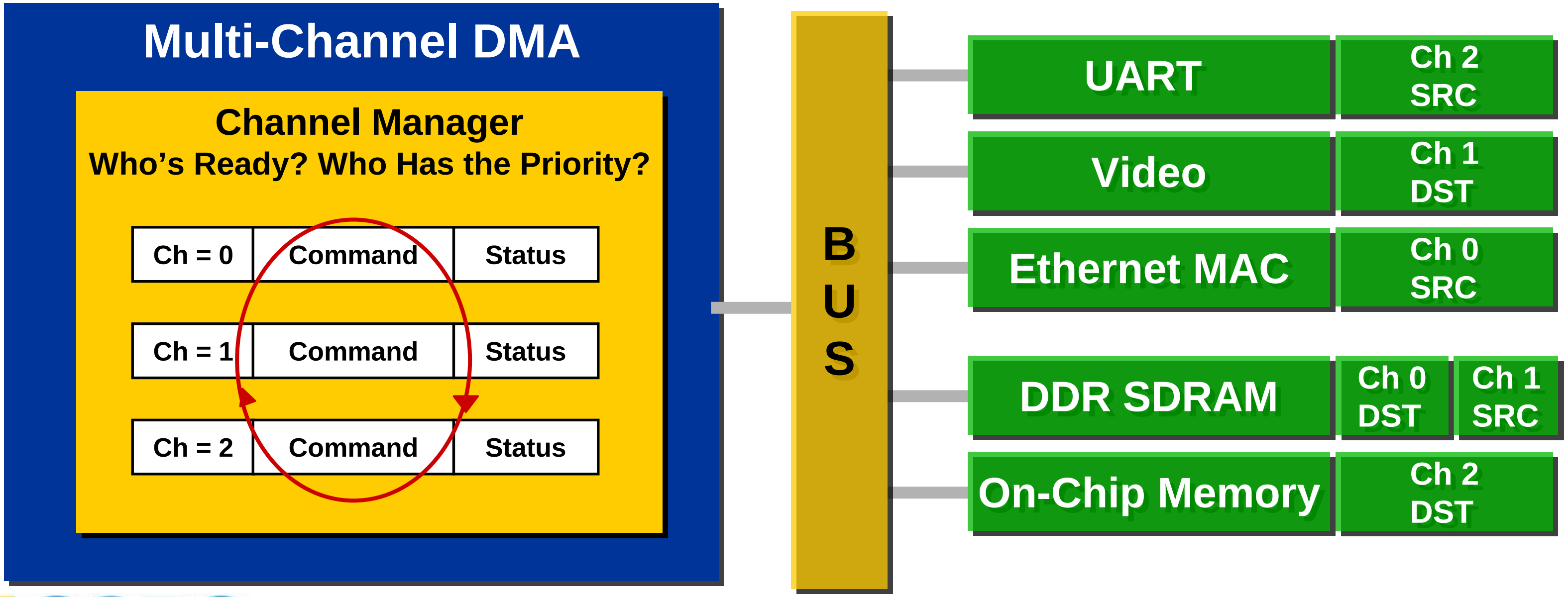
	Design With DMA	Design Without DMA
Aspect	**DMA-Enabled Smart Peripheral (Clock Cycles)**	**Bit-by-Bit Software Algorithm (Clock Cycles)**
8 Byte Message	243	2,838
512 Byte Message	582	181,753
1KB Message	922	363,243
64KB Message	43,925	23,264,648
Hardware Resources Utilized	~975 Additional Logic Elements	Baseline

Other DMA Enhancements

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

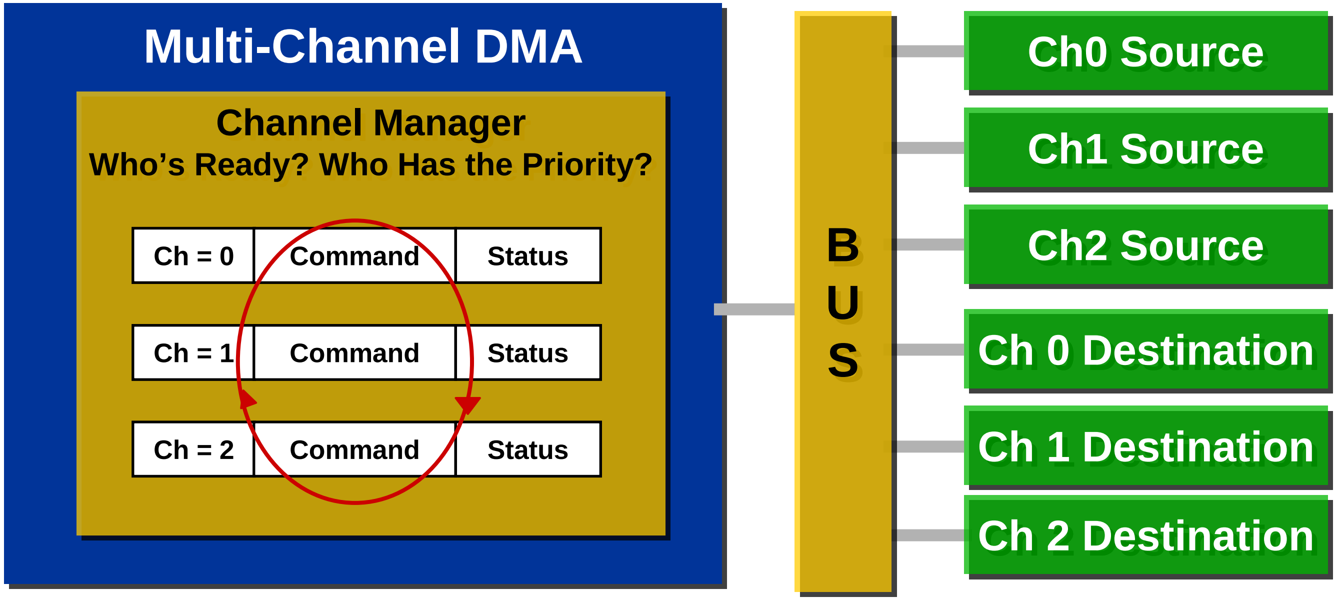
Multi-Channel DMA Controllers

- 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



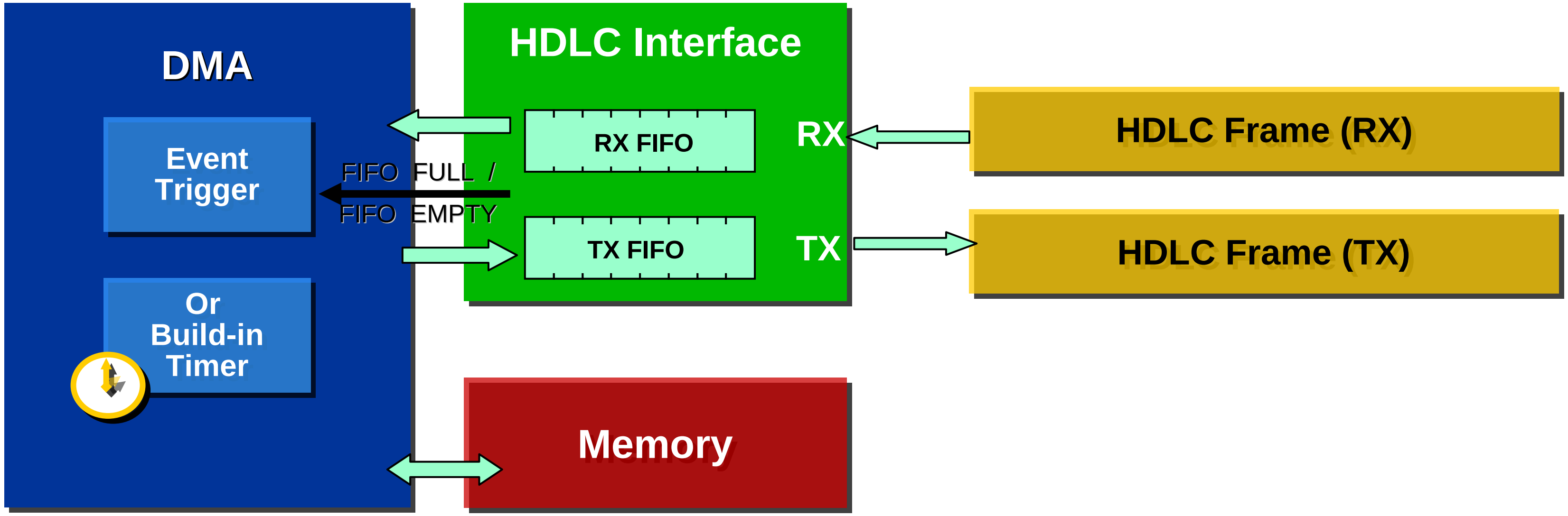
Multi-Channel DMA Controllers

- 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



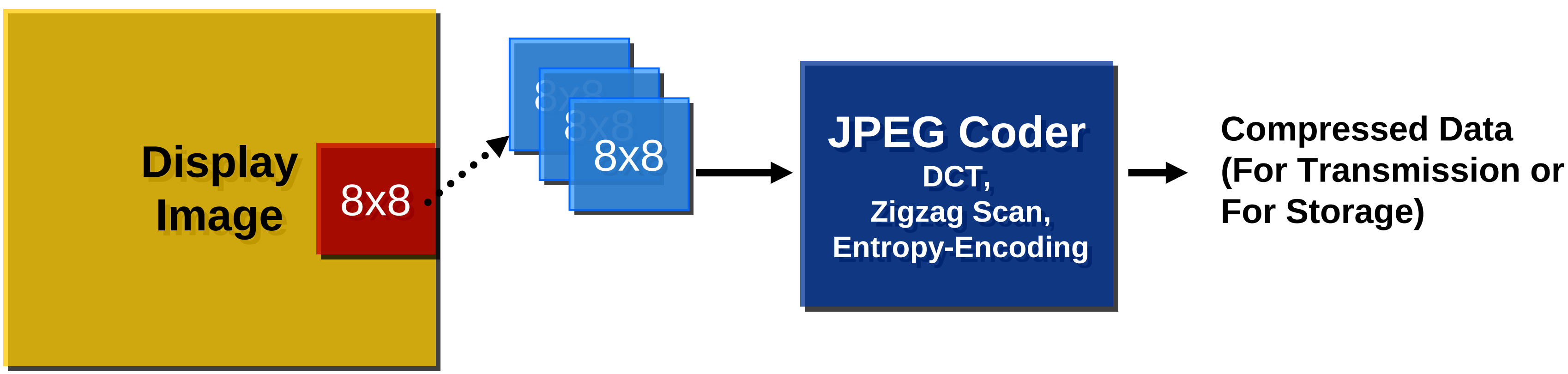
Event- or Time-Triggered DMA

- 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



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)

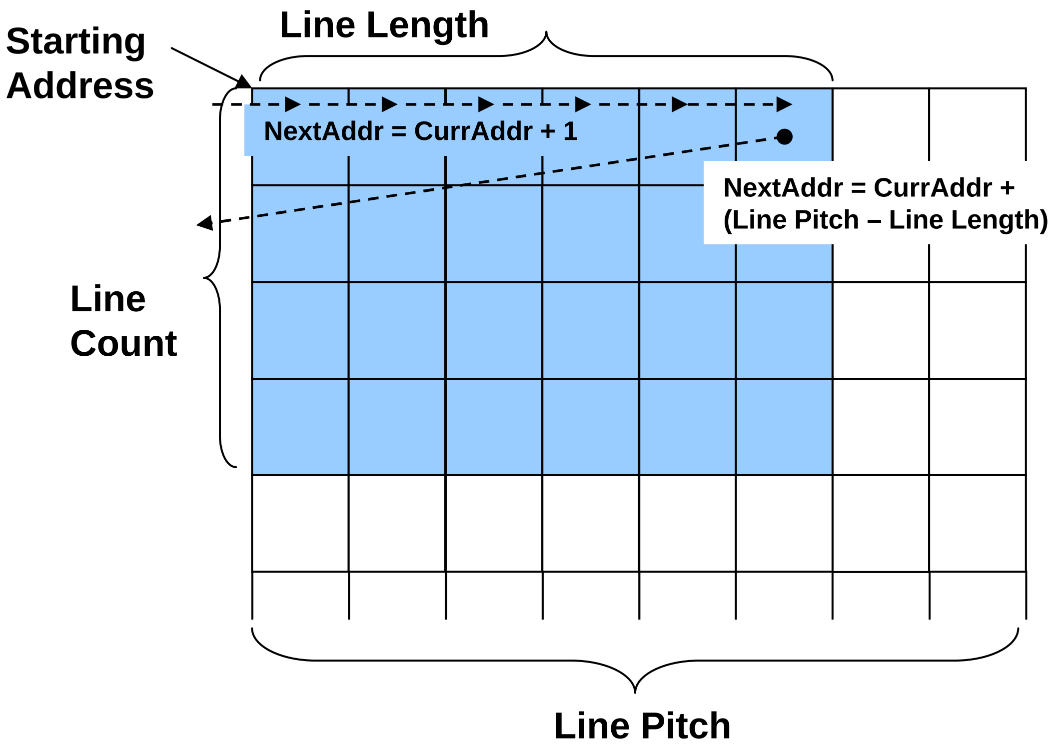
In This example:

Line Length = 6

Line Count = 4

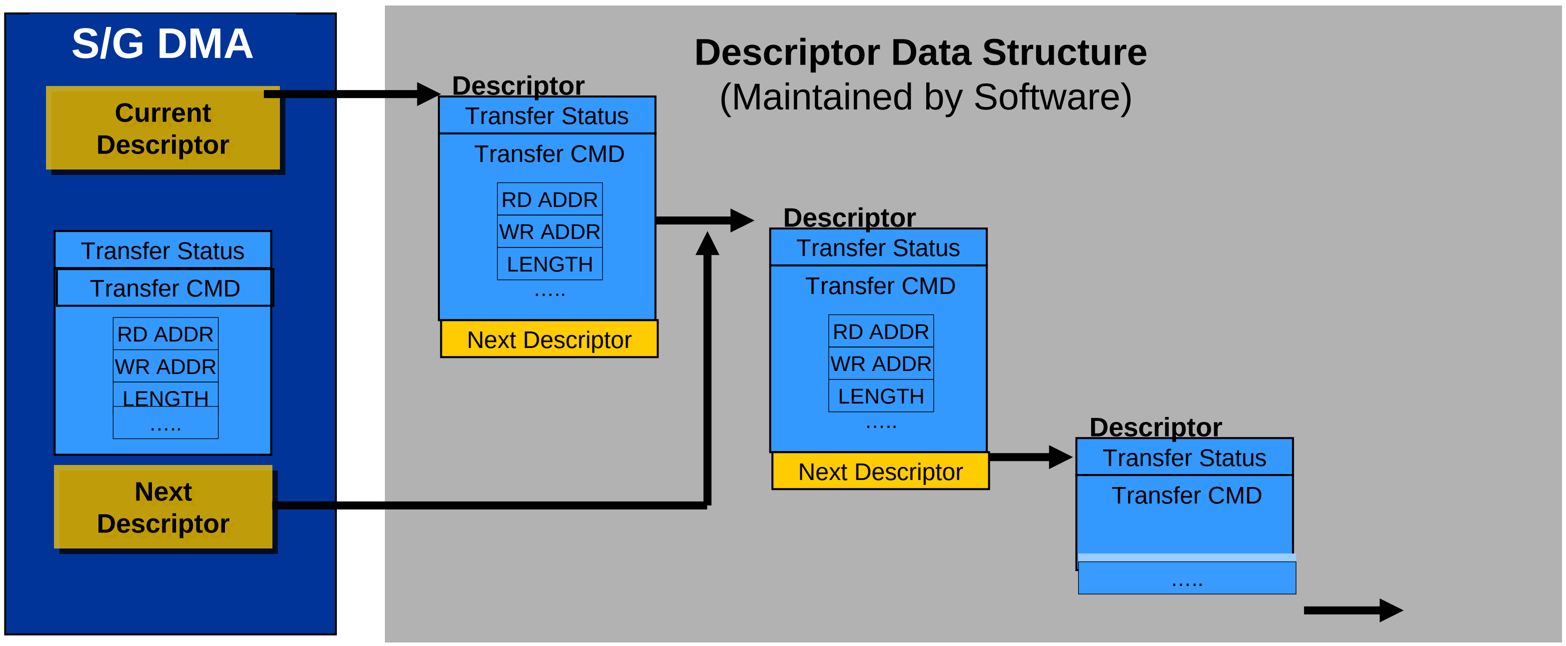
Line Pitch = 8

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

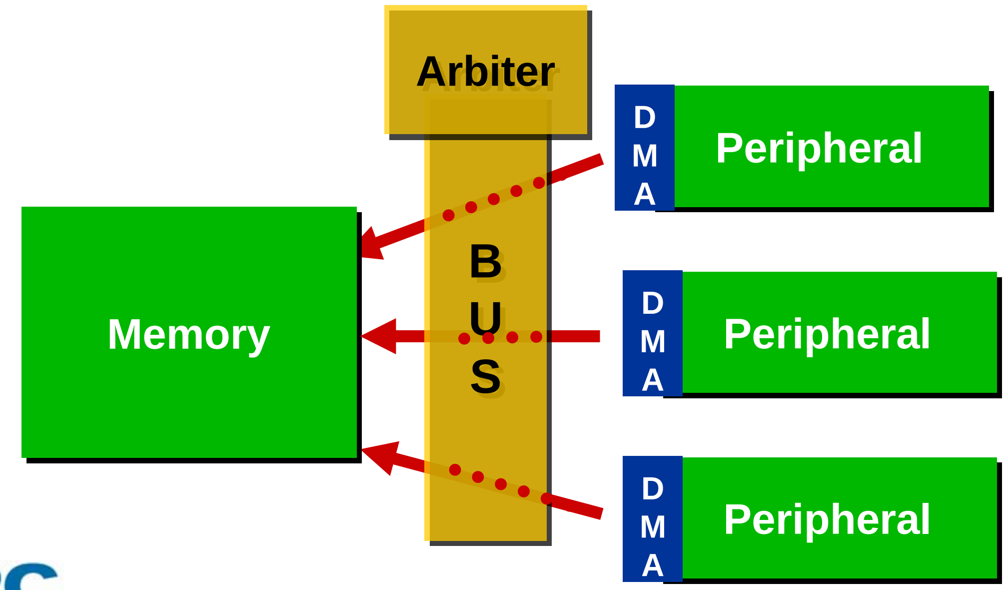


Combine Various DMA Enhancements

- 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)



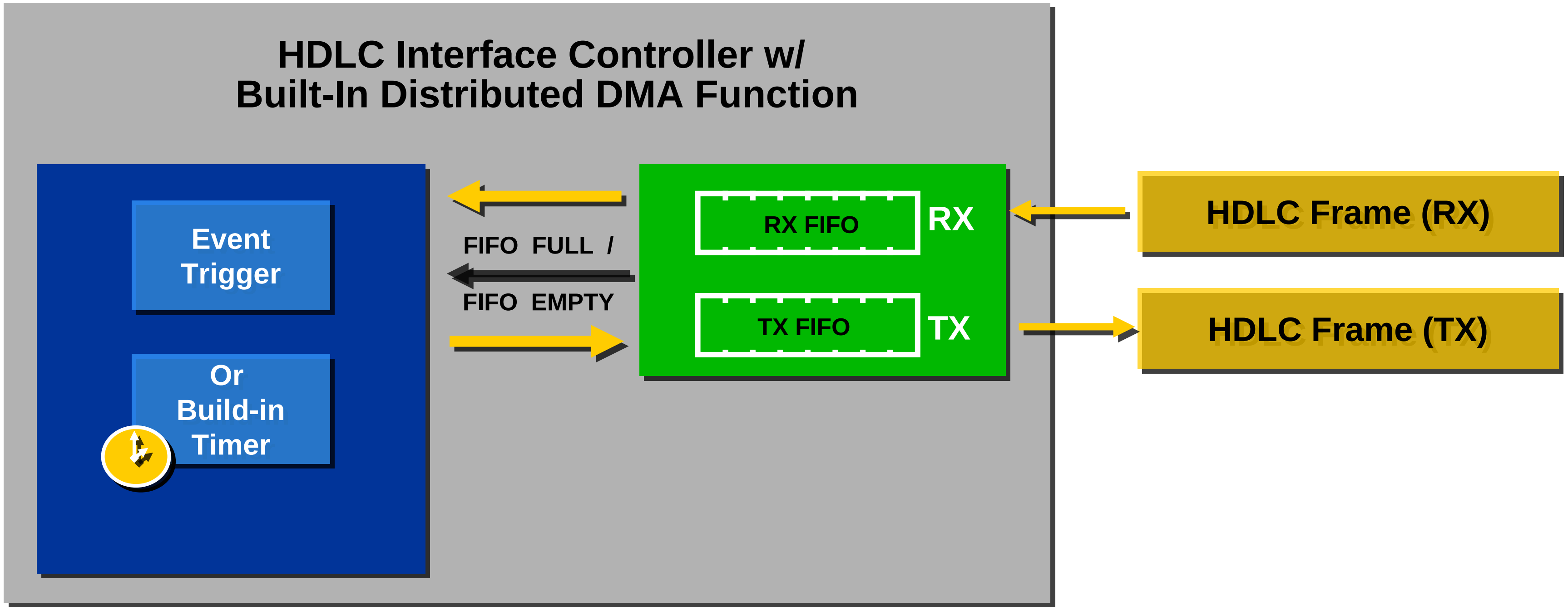
Benefits of Distributed DMA

- 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

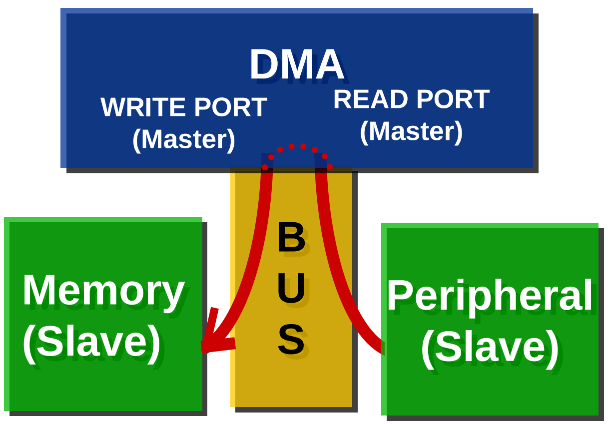
Distributed DMA Example

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

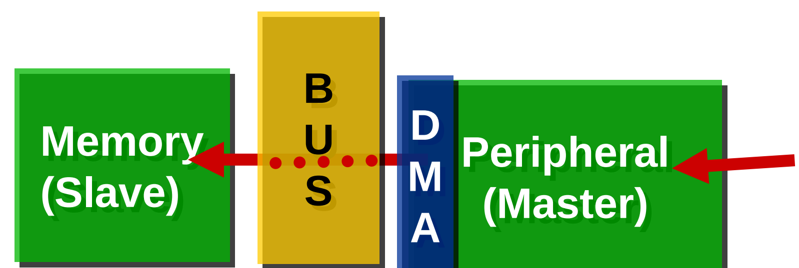


Centralized DMA vs. Distributed DMA

Aspect	Centralized DMA	Distributed DMA
Hardware Resources	Multiple Devices Can Share the Same DMA	Overall System May Consume More Logic
Bus Loading	Double Bus Loading	Single Bus Loading
Master & Slave Consideration	DMA Controller Has 2 Master Ports: Read & Write	Peripheral Must Have Master Capability



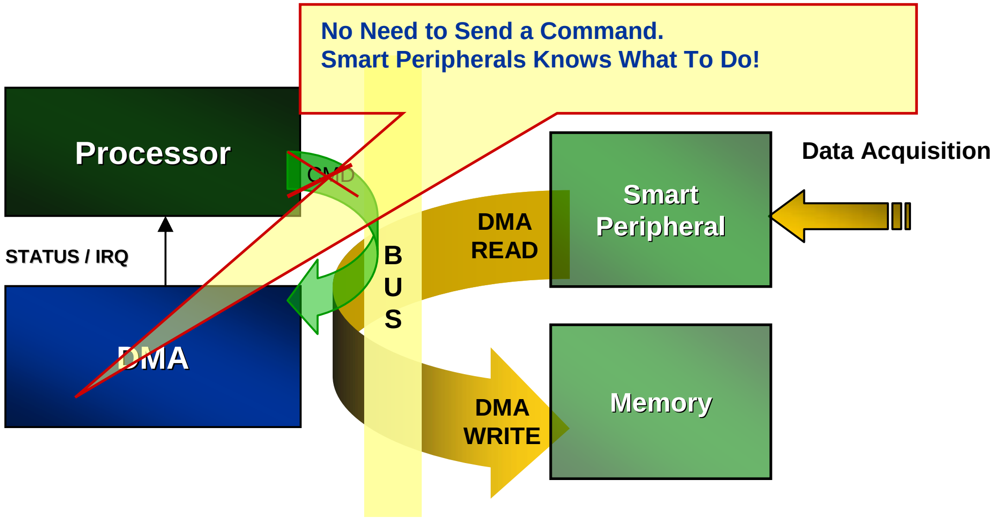
Centralized DMA Mechanism



Distributed DMA Mechanism

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

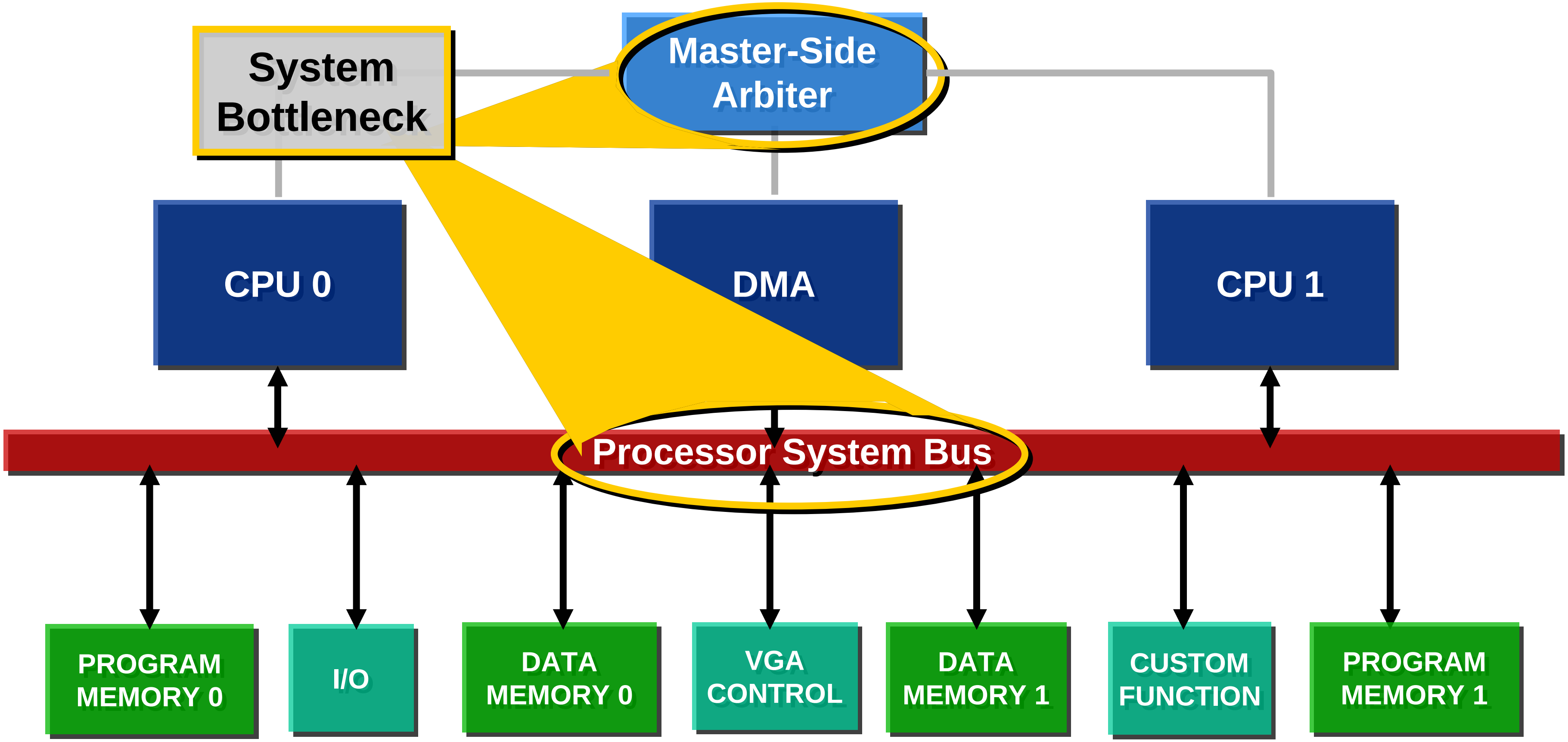
Bus Design Decisions

- 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

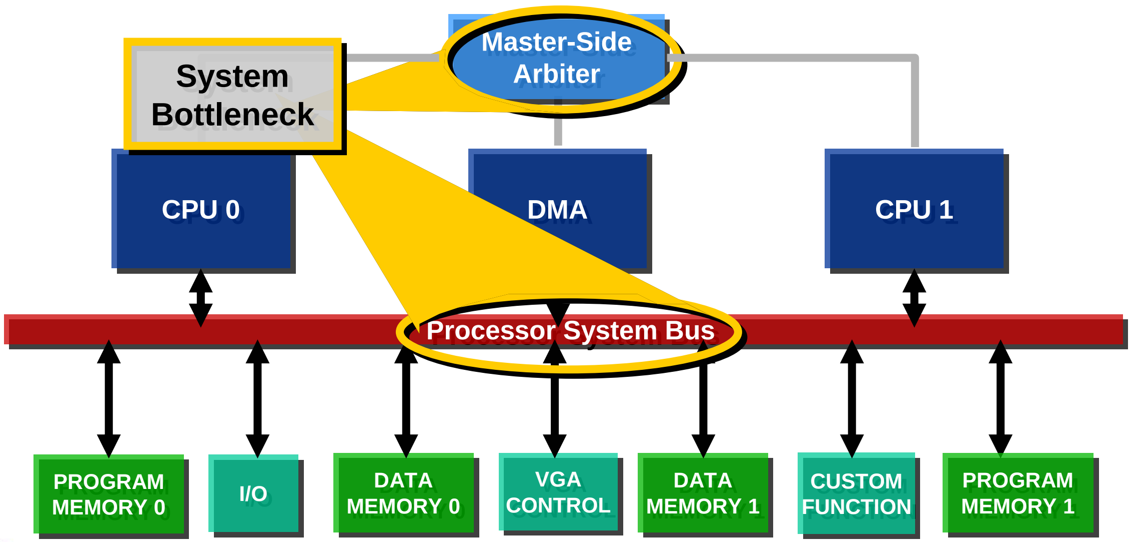
- Master
  - Initiates a Read or Write Transaction
  - Example: Processors
- Slave
  - Responds to a Transaction
  - Example: Memories
- Arbiter
  - Arbitrates in Multiple Masters that Want to Initiate Simultaneous Transactions
- Bridge
  - Connects Buses & Passes the Transaction on a Bus to the Other 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

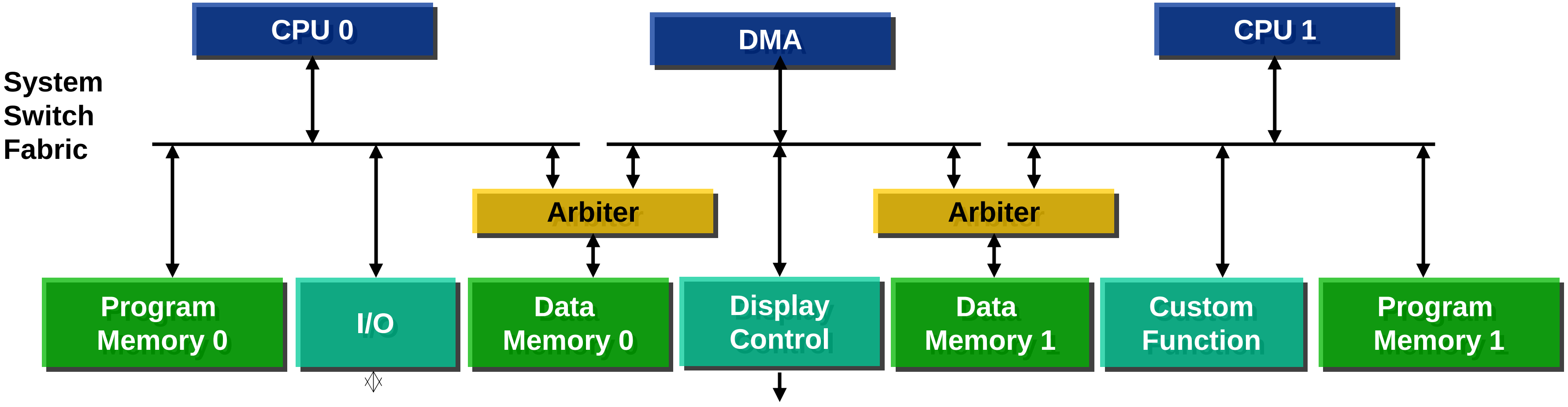


Traditional Shared-Bus System

- 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

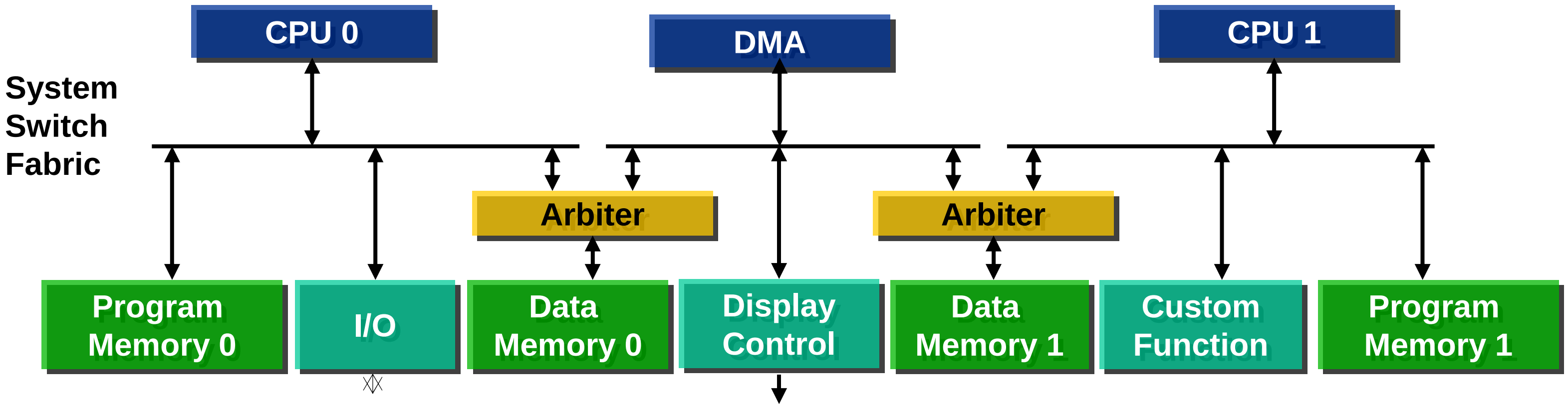


Switch Fabric & Slave-Side Arbitration Scheme



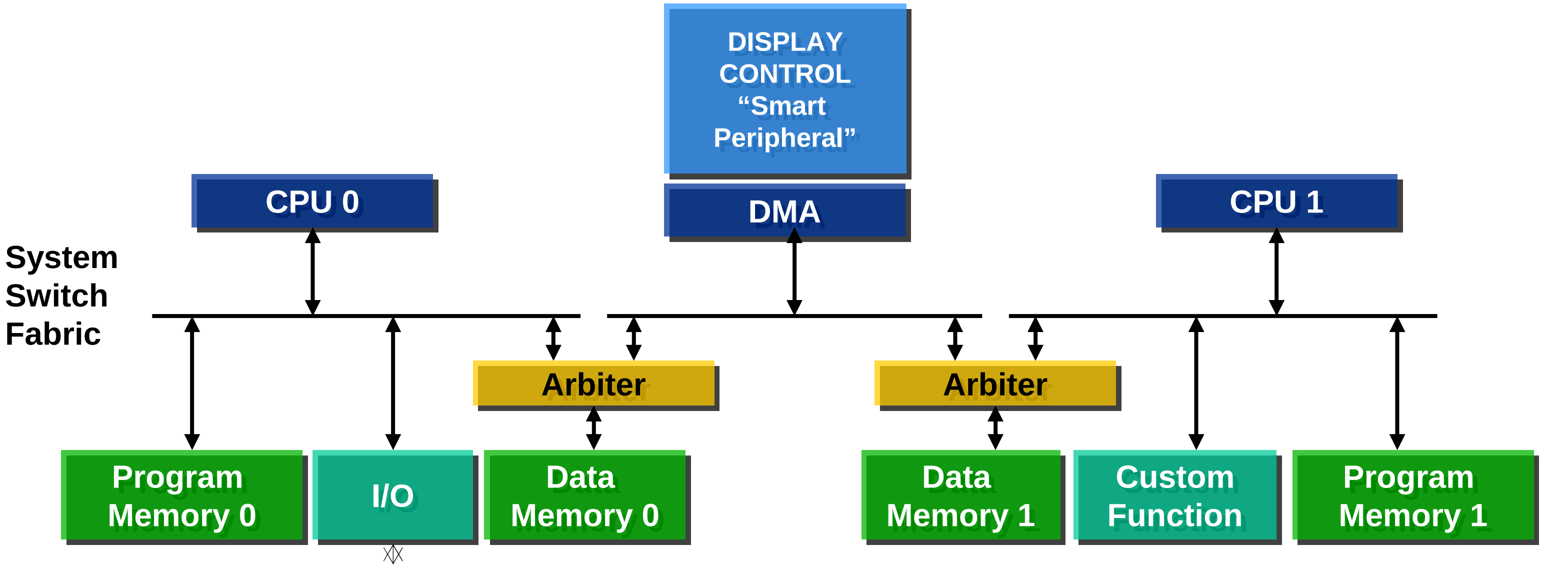
Switch Fabric & Slave-Side Arbitration Scheme

- 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

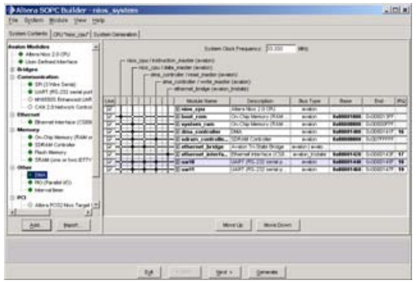
- The Smart Peripheral with DMA Function Reduces Latency for Memory Access



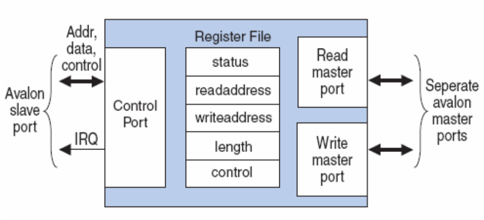
Altera Development Tools & IP Supporting DMA

Altera's DMA System Architecture Solution

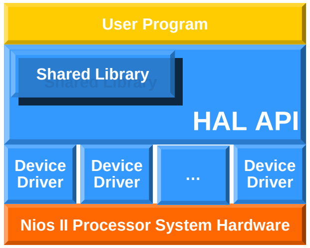
- Development Tool
  - SOPC Builder



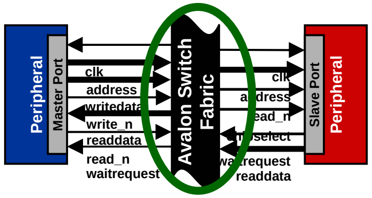
- Hardware
  - DMA Controller IP Core



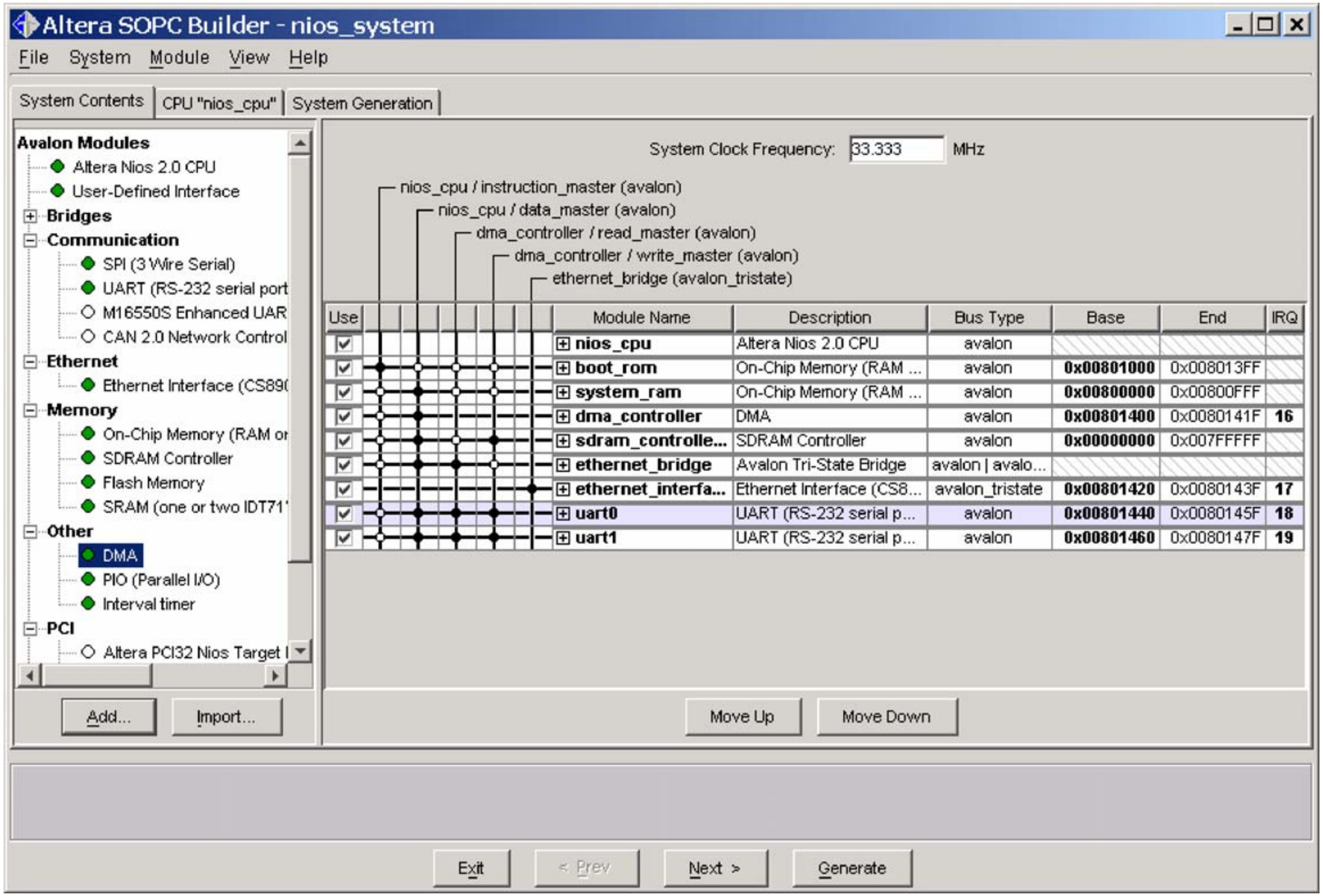
- Software
  - Header Files
  - Hardware Abstraction Layer (HAL)
  - Generic Device Models



- Bus Interconnect
  - Avalon™ Switch Fabric

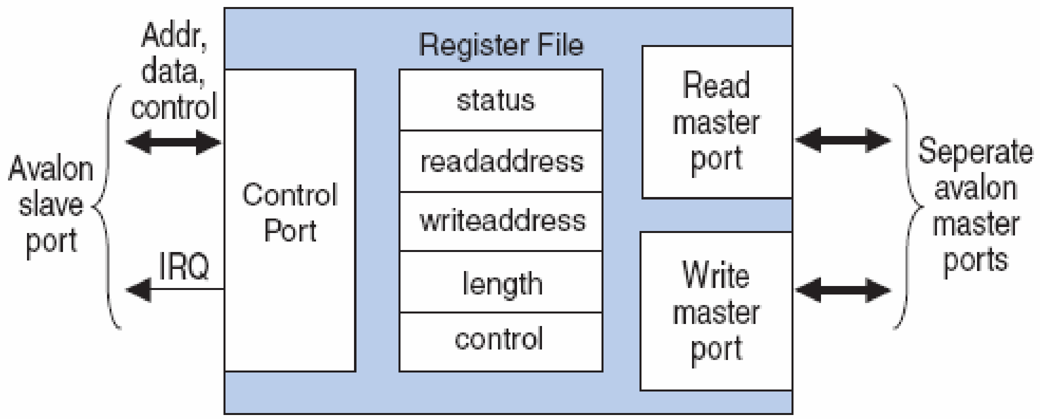


SOPC Builder Development Tool



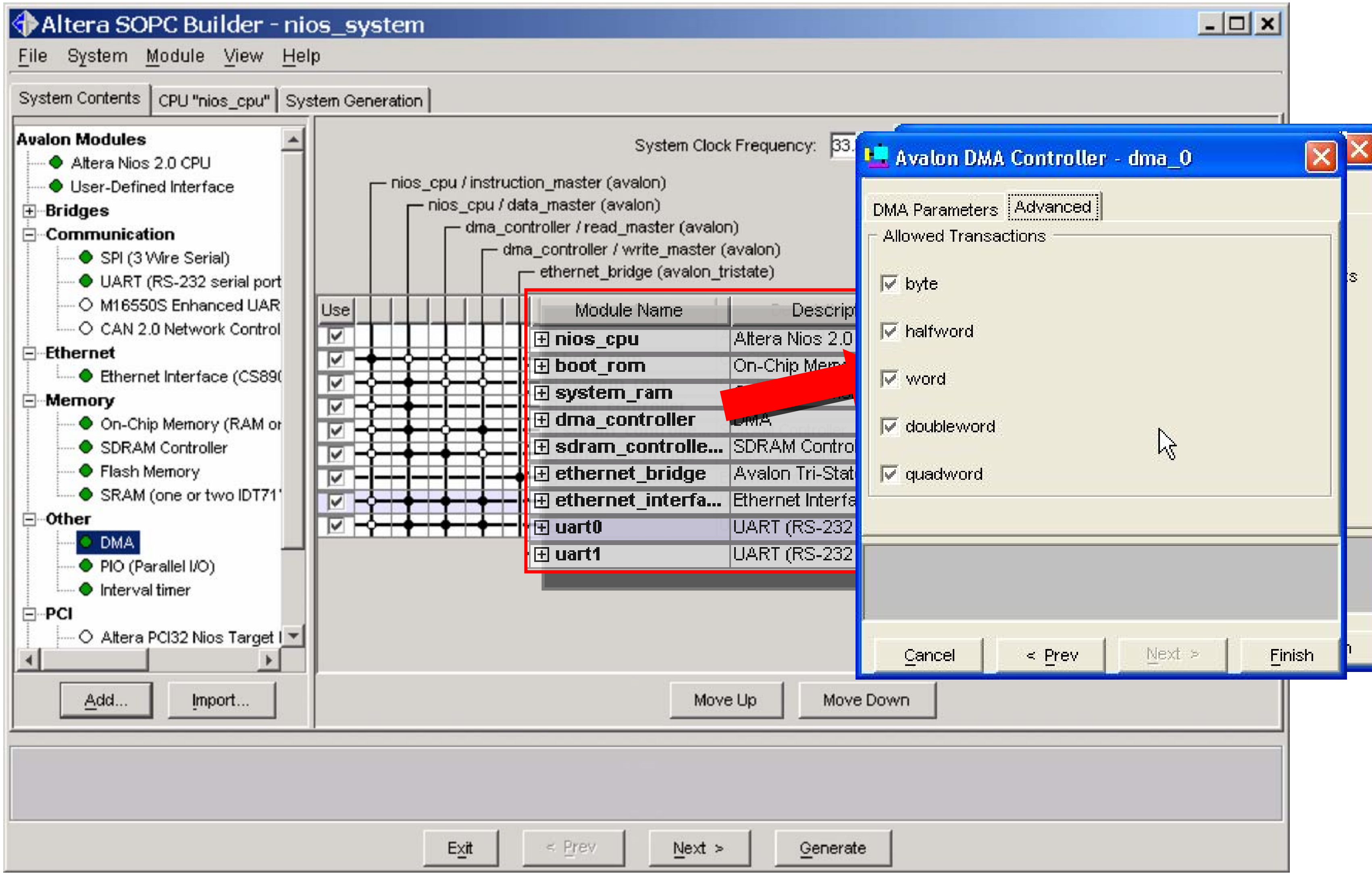
Altera DMA Controller IP Core

- 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

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

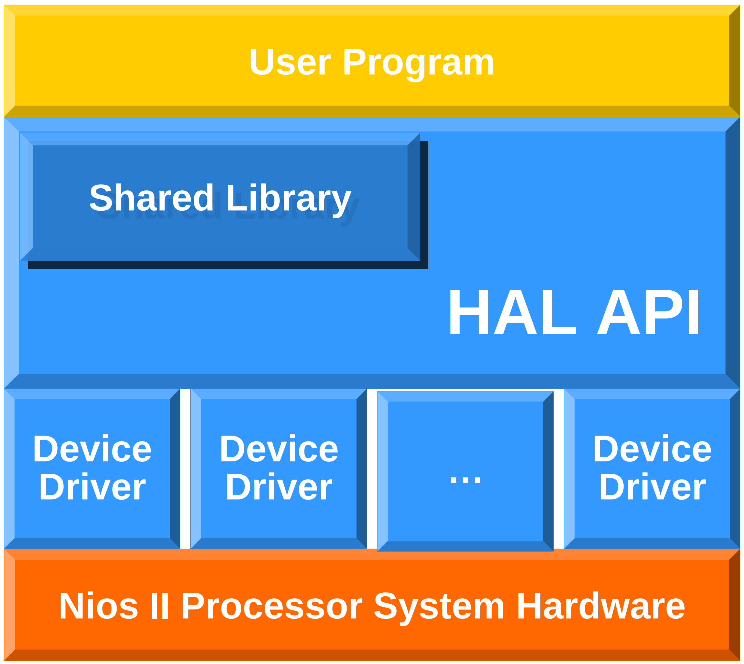
Nios II HAL Architecture

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

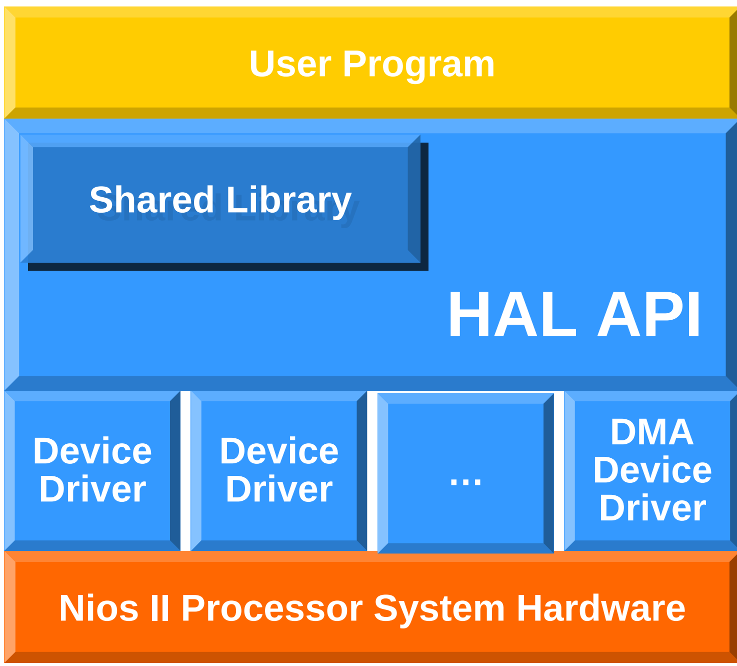
_exit()	open()
close()	opendir
closedir()	read()
fstat()	readdir()
getpid()	rewinddir()
gettimeofday()	sbrk()
ioctl()	settimeofday()
isatty()	stat()
kill()	usleep()
lseek()	wait()
	write()

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



sys/alt_dma.h

```
#ifndef __ALT_DMA_H__
#define __ALT_DMA_H__
…
extern alt_dma_txchan_open();
extern alt_dma_rxchan_open();
extern alt_dma_txchan_send();
extern alt_dma_rxchan_prepare();
```

*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`

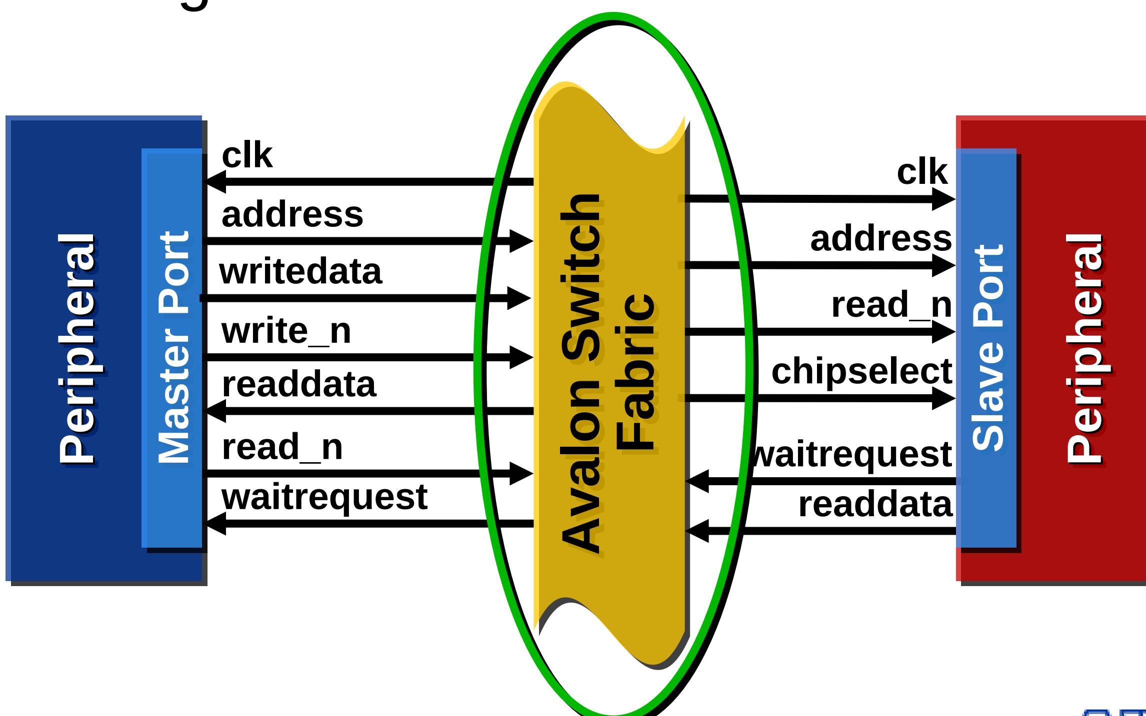
Most Signals Available In Positive or Negative Form

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

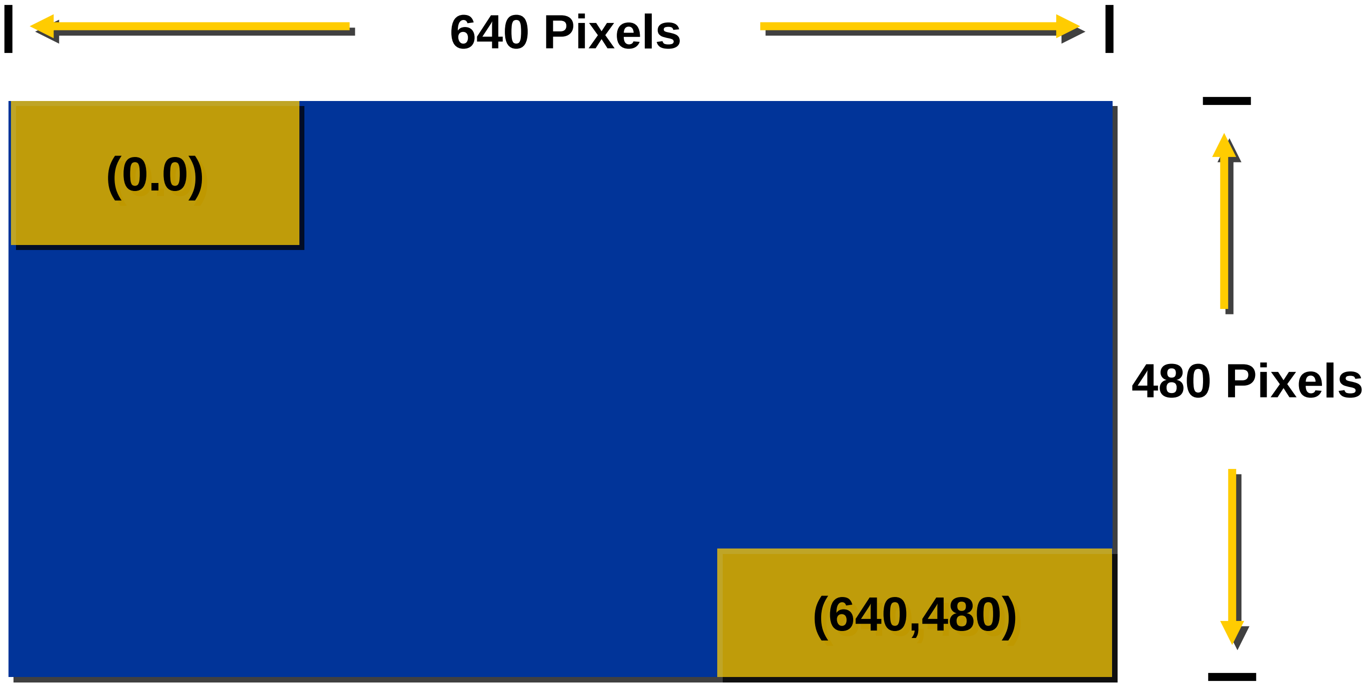


Example:
VGA Controller

Example: VGA Controller

- Requirements
  - High Bandwidth
- Solution
  - Custom VGA Peripheral
  - Avalon Streaming Mode
- *AN 333: Developing Peripherals for SOPC Builder*

VGA Monitor Pixel Organization

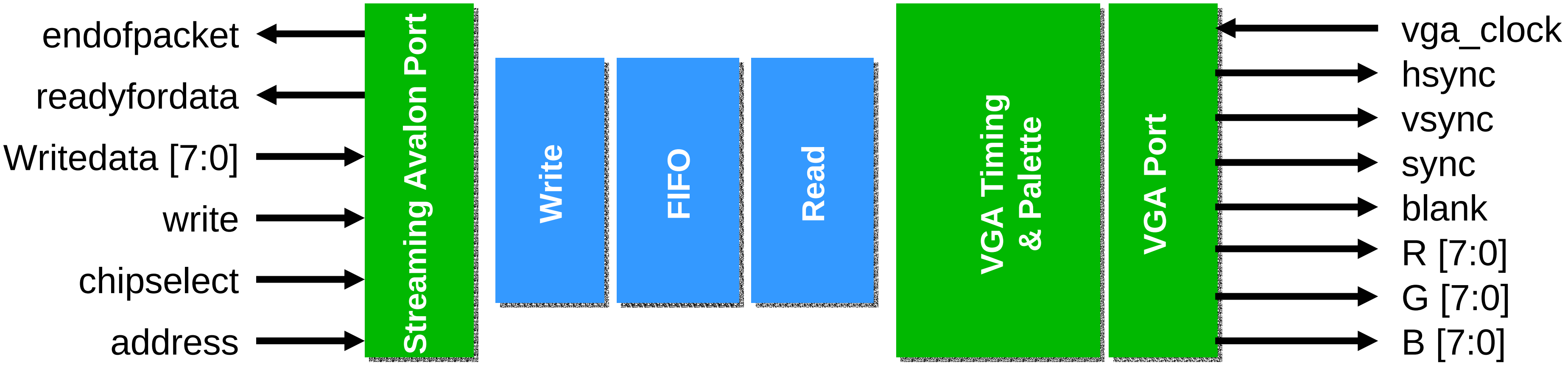


VGA Peripheral

- Peripheral Functional Blocks
  - Peripheral Task Logic
  - Register File
  - Avalon Interface
  - Software Driver Functions

Block Diagram of VGA Display Driver Hardware

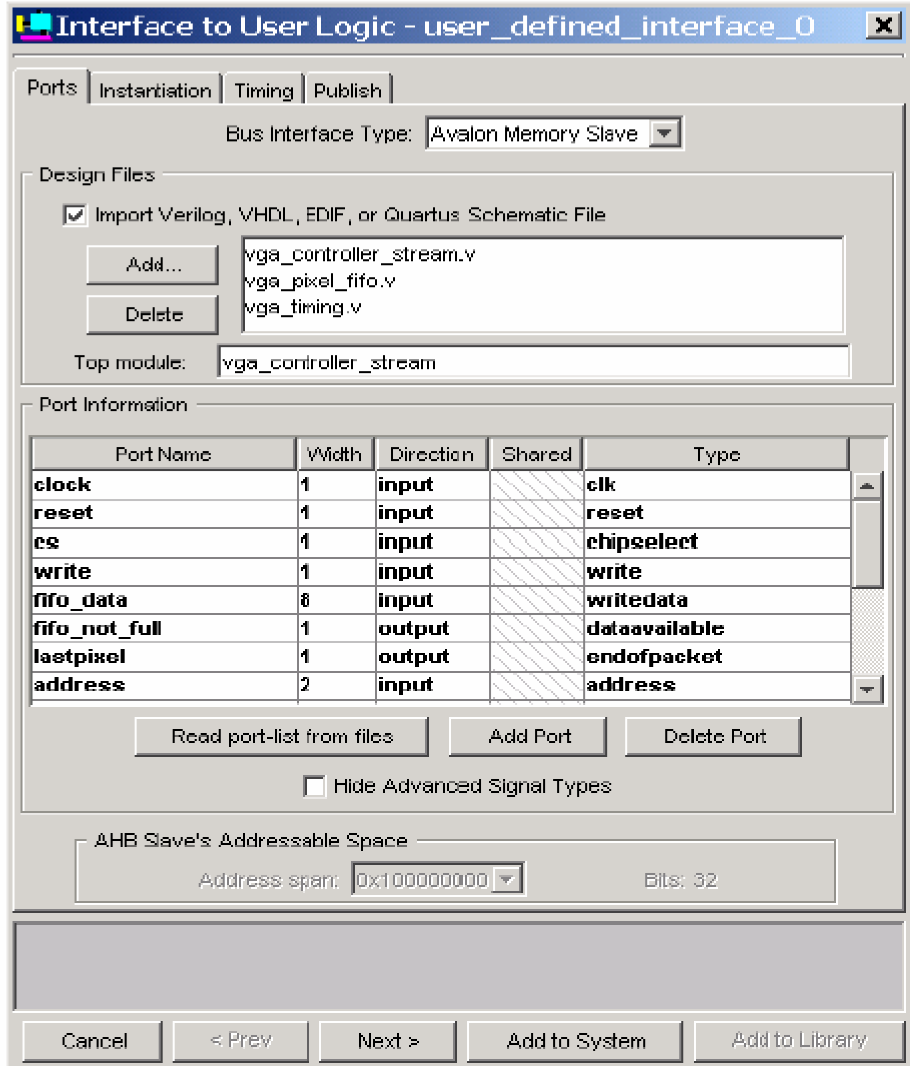
Register File & Address Mapping			
Register Name	**Offset**	**Access**	**Description**
`vga_data`	0x0	Write-Only	Writing to this Register Stores the 8-Bit Value into the FIFO Line-Buffer



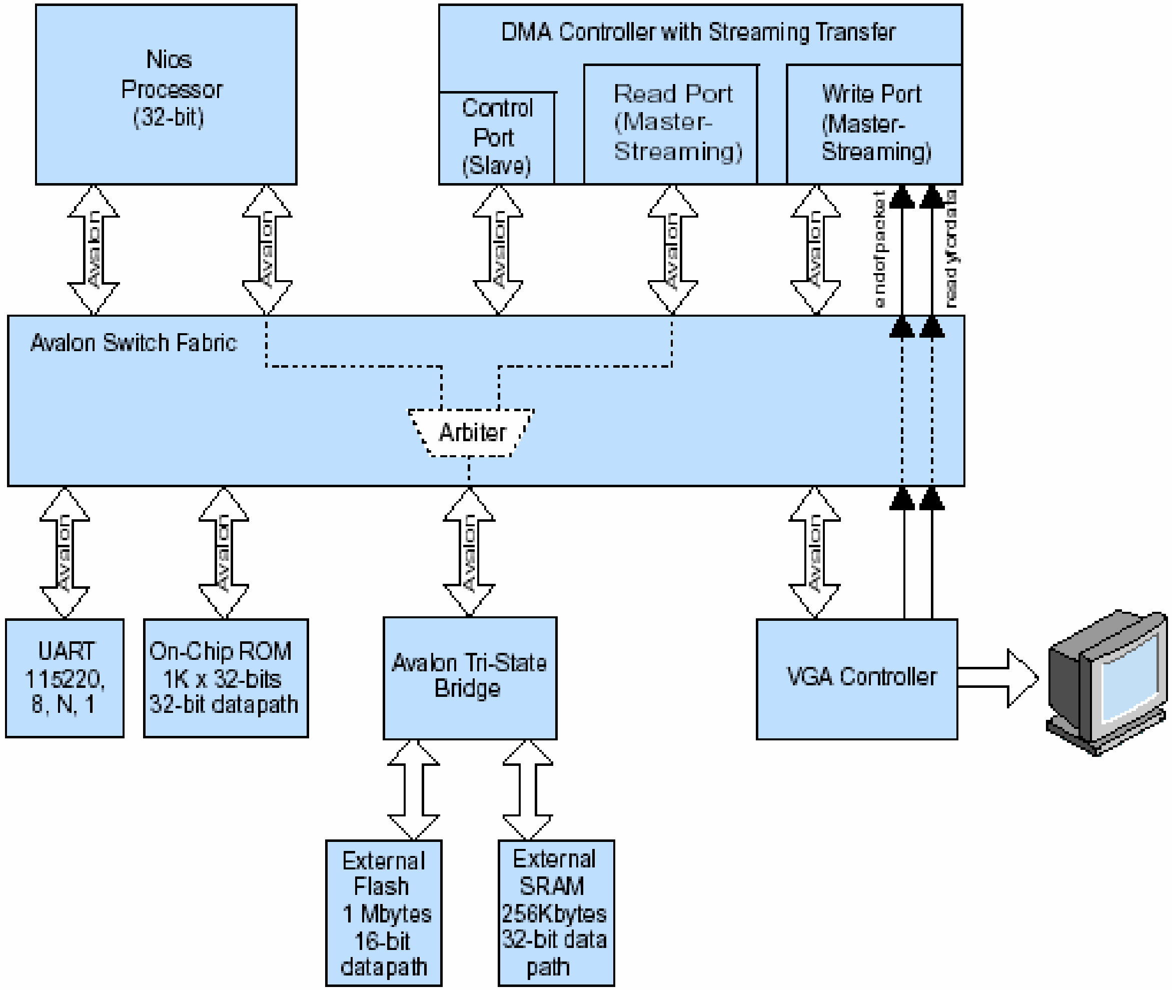
Avalon Signals for VGA Controller Peripheral

Port Name	Avalon Signal Type	Bit-Width	Direction	Description
clock	clk	1	input	Input clock for writing to the FIFO
reset	reset	1	input	Peripheral reset
cs	chipselect	1	input	Chip select
write	write	1	input	Write-enable signal
fifo data	writedata	8	input	8-bit write data
fifo_not_full	readyfordata	1	output	Streaming transfer signal indicating that new data is accepted
lastpixel	endofpacket	1	output	Streaming transfer signal indicating that the last pixel of a frame was received
vga_clock	Export		input	Input clock for VGA timing and reading data from FIFO
hsync	Export	1	output	Horizontal synchronization signal (output)
sync	Export	1	output	Vertical synchronization signal (output)
Blank	Export	1	output	Logical AND of hsync and vsync (output)
R	Export	8	output	Red color (output)
G	Export	8	output	Green color (output)
B	Export	8	output	Blue color (output)

Ports Tab for the VGA Controller Peripheral



Example System with Streaming VGA Controller



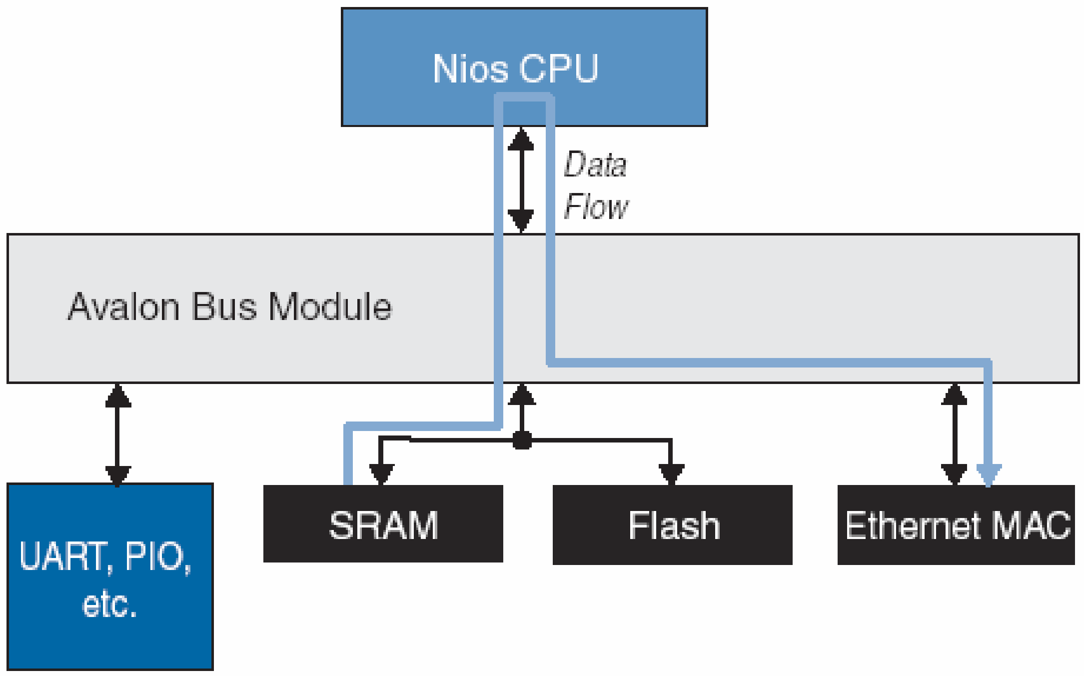
SOPC WORLD 2004

ALTERA

Example:
Ethernet Controller

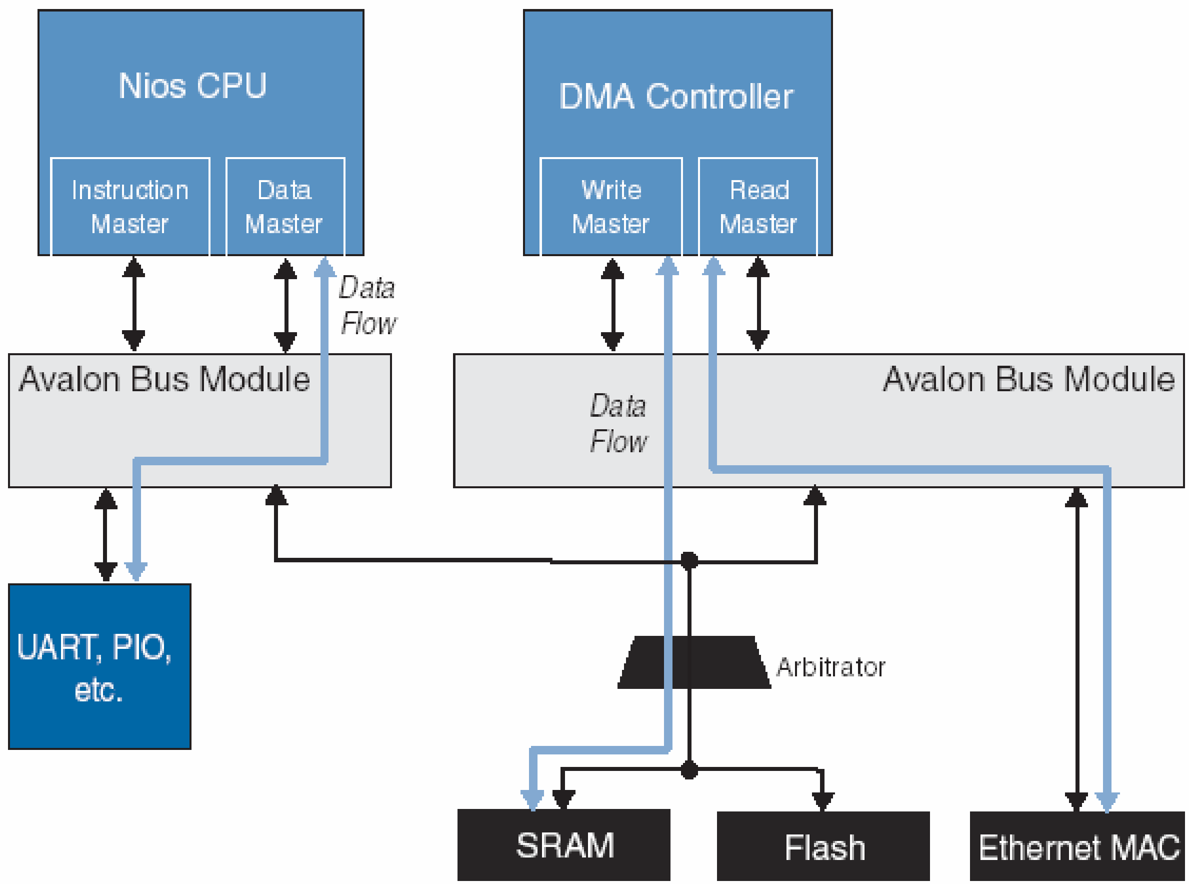
Ethernet Frame Data Transmission Path with Single Master Architecture

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

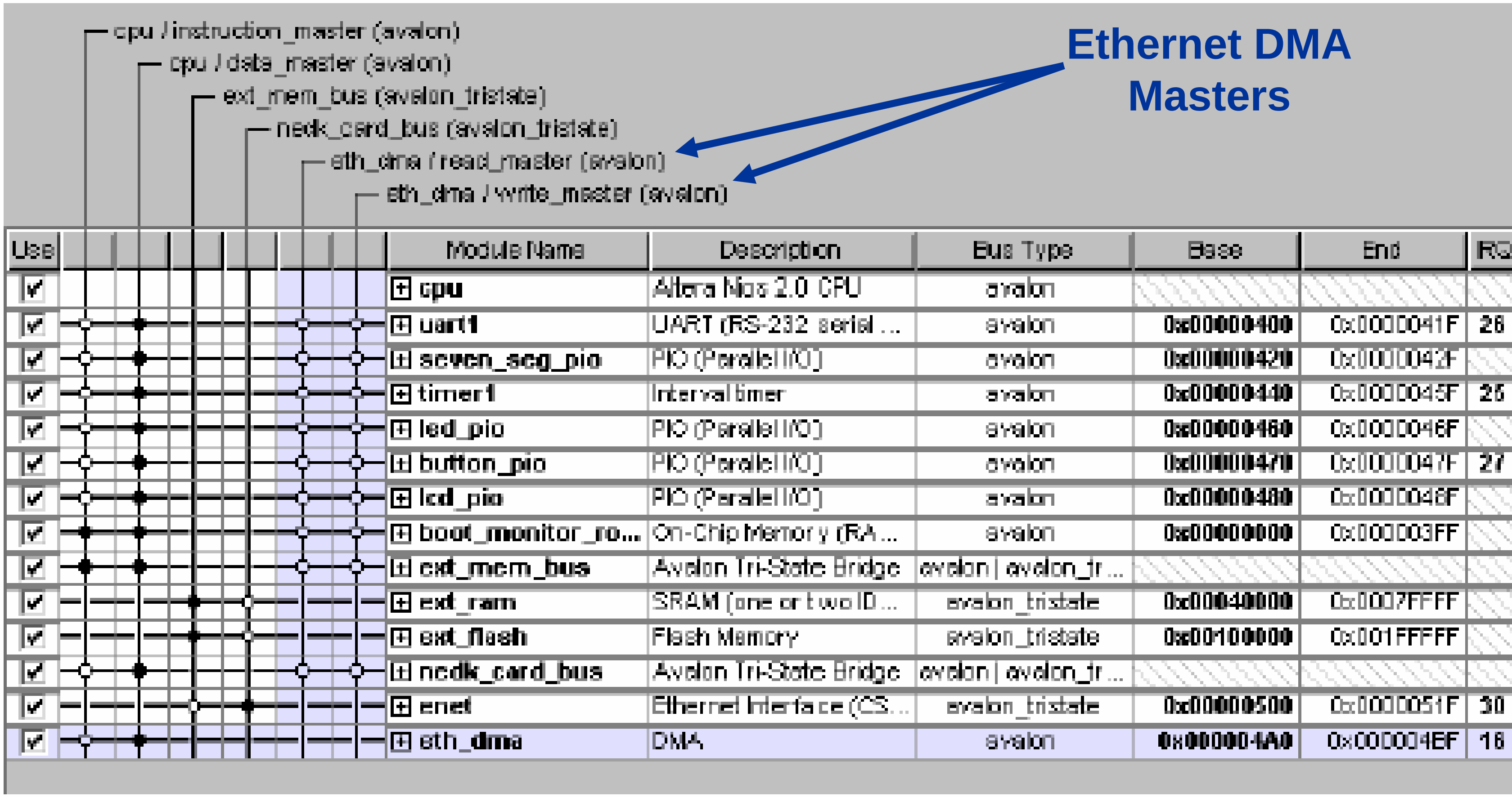


Ethernet Frame Data Transmission Path Using DMA & Simultaneous Multi-Mastering

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



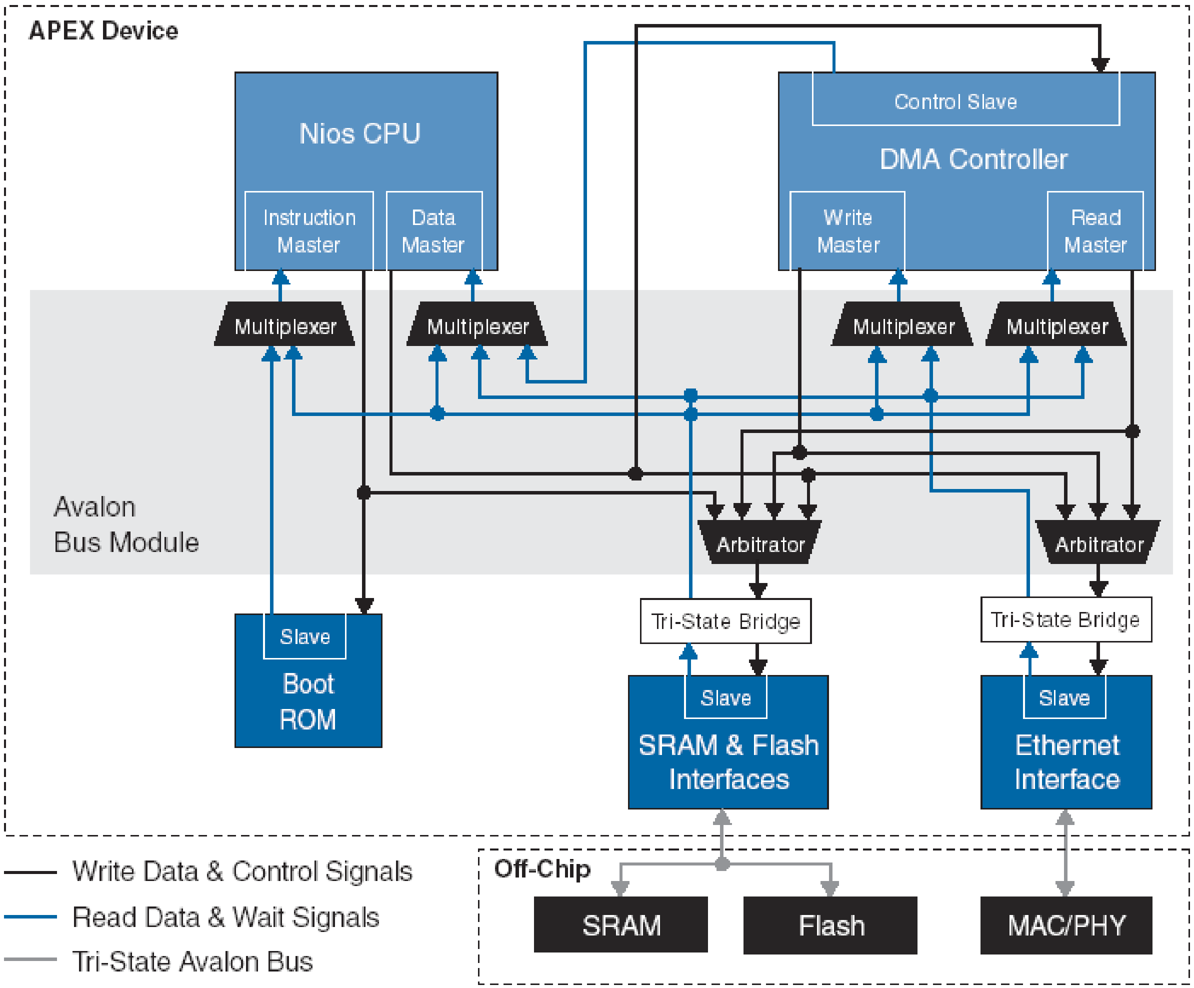
Ethernet Design with DMA Controller



Use	Module Name	Description	Bus Type	Base	End	IRQ
✔	cpu	Altera Nios 2.0 CPU	avalon			
✔	uart1	UART (RS-232 serial ...	avalon	0x00000400	0x0000041F	26
✔	seven_seg_pio	PIO (Parallel I/O)	avalon	0x00000420	0x0000042F	
✔	timer1	Interval timer	avalon	0x00000440	0x0000045F	25
✔	led_pio	PIO (Parallel I/O)	avalon	0x00000460	0x0000046F	
✔	button_pio	PIO (Parallel I/O)	avalon	0x00000470	0x0000047F	27
✔	lcd_pio	PIO (Parallel I/O)	avalon	0x00000480	0x0000048F	
✔	boot_monitor_ro...	On-Chip Memory (RA ...	avalon	0x00000000	0x000003FF	
✔	ext_mem_bus	Avalon Tri-State Bridge	avalon | avalon_tr ...			
✔	ext_ram	SRAM (one or two ID ...	avalon_tristate	0x00040000	0x0007FFFF	
✔	ext_flash	Flash Memory	avalon_tristate	0x00100000	0x001FFFFF	
✔	nedk_card_bus	Avalon Tri-State Bridge	avalon | avalon_tr ...			
✔	enet	Ethernet Interface (CS...	avalon_tristate	0x00000500	0x0000051F	30
✔	eth_dma	DMA	avalon	0x000004A0	0x000004BF	16

System Interconnect Block Diagram

Figure 13. System Interconnect Block Diagram



SOPC WORLD 2004

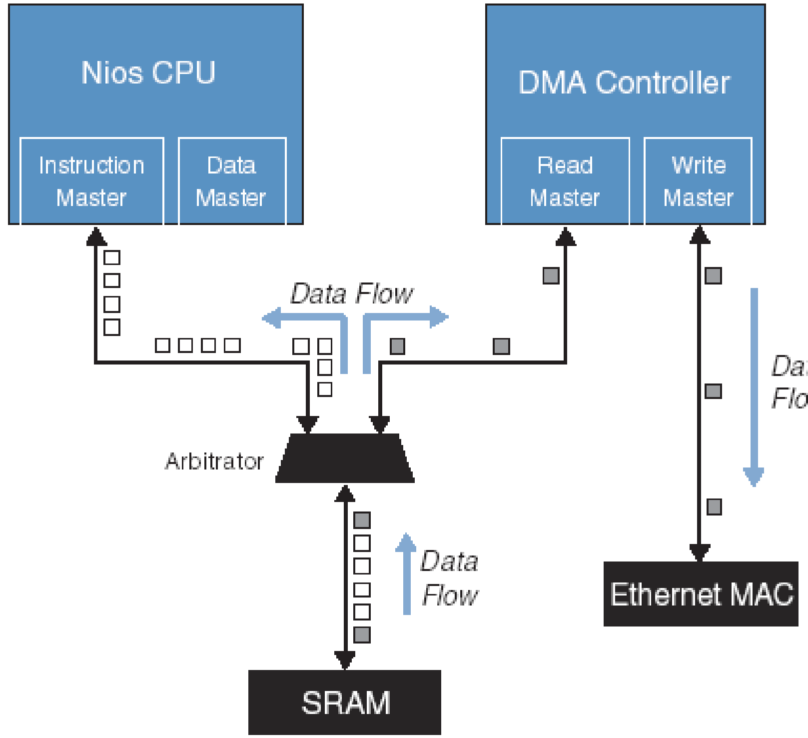
ALTERA

Multi-Master Ethernet Design Arbitration Settings

Use	cpu / instruction_master (avalon)	cpu / data_master (avalon)	ext_mem_bus (avalon_tristate)	nedk_card_bus (avalon_tristate)	eth_dma / read_master (avalon)	eth_dma / write_master (avalon)	Module Name	Description	Bus Type	Base	End	IRQ
✔							⊞ cpu	Altera Nios 2.0 CPU	avalon			
✔		1					⊞ uart1	UART (RS-232 serial po...	avalon	0x00000400	0x0000041F	26
✔		1					⊞ seven_seg_pio	PIO (Parallel I/O)	avalon	0x00000420	0x0000042F	
✔		1					⊞ timer1	Interval timer	avalon	0x00000440	0x0000045F	25
✔		1					⊞ led_pio	PIO (Parallel I/O)	avalon	0x00000460	0x0000046F	
✔		1					⊞ button_pio	PIO (Parallel I/O)	avalon	0x00000470	0x0000047F	27
✔		1					⊞ lcd_pio	PIO (Parallel I/O)	avalon	0x00000480	0x0000048F	
✔	1	1					⊞ boot_monitor_rom	On-Chip Memory (RAM ...	avalon	0x00000000	0x000003FF	
✔	4	4			1	1	⊞ ext_mem_bus	Avalon Tri-State Bridge	avalon | avalon_tri...			
✔			1				⊞ ext_ram	SRAM (one or two IDT7...	avalon_tristate	0x00040000	0x0007FFFF	
✔			1				⊞ ext_flash	Flash Memory	avalon_tristate	0x00100000	0x001FFFFF	
✔		1			1	1	⊞ nedk_card_bus	Avalon Tri-State Bridge	avalon | avalon_tri...			
✔				1			⊞ enet	Ethernet Interface (CS8...	avalon_tristate	0x00000500	0x0000051F	30
✔		1					⊞ eth_dma	DMA	avalon	0x000004A0	0x000004BF	16

Simplified View of Arbitration During Conflict between DMA & CPU

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

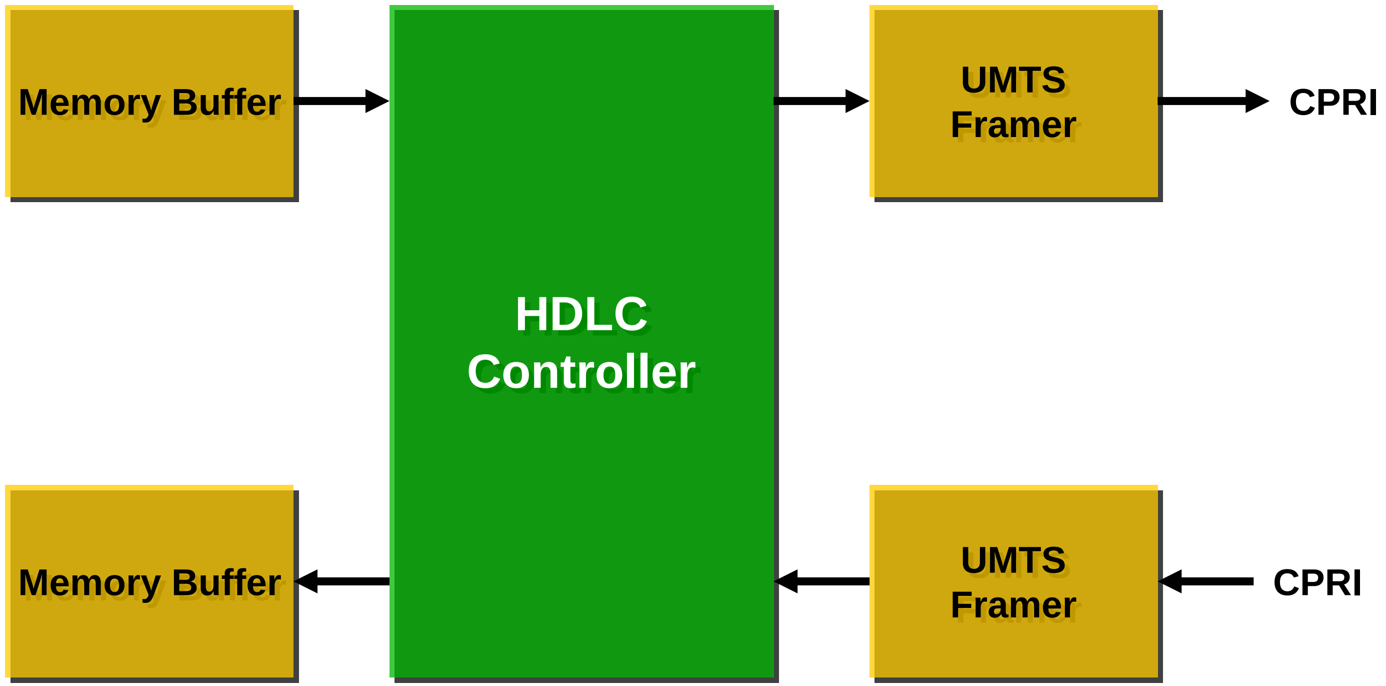
Example: CPRI Controller

- 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

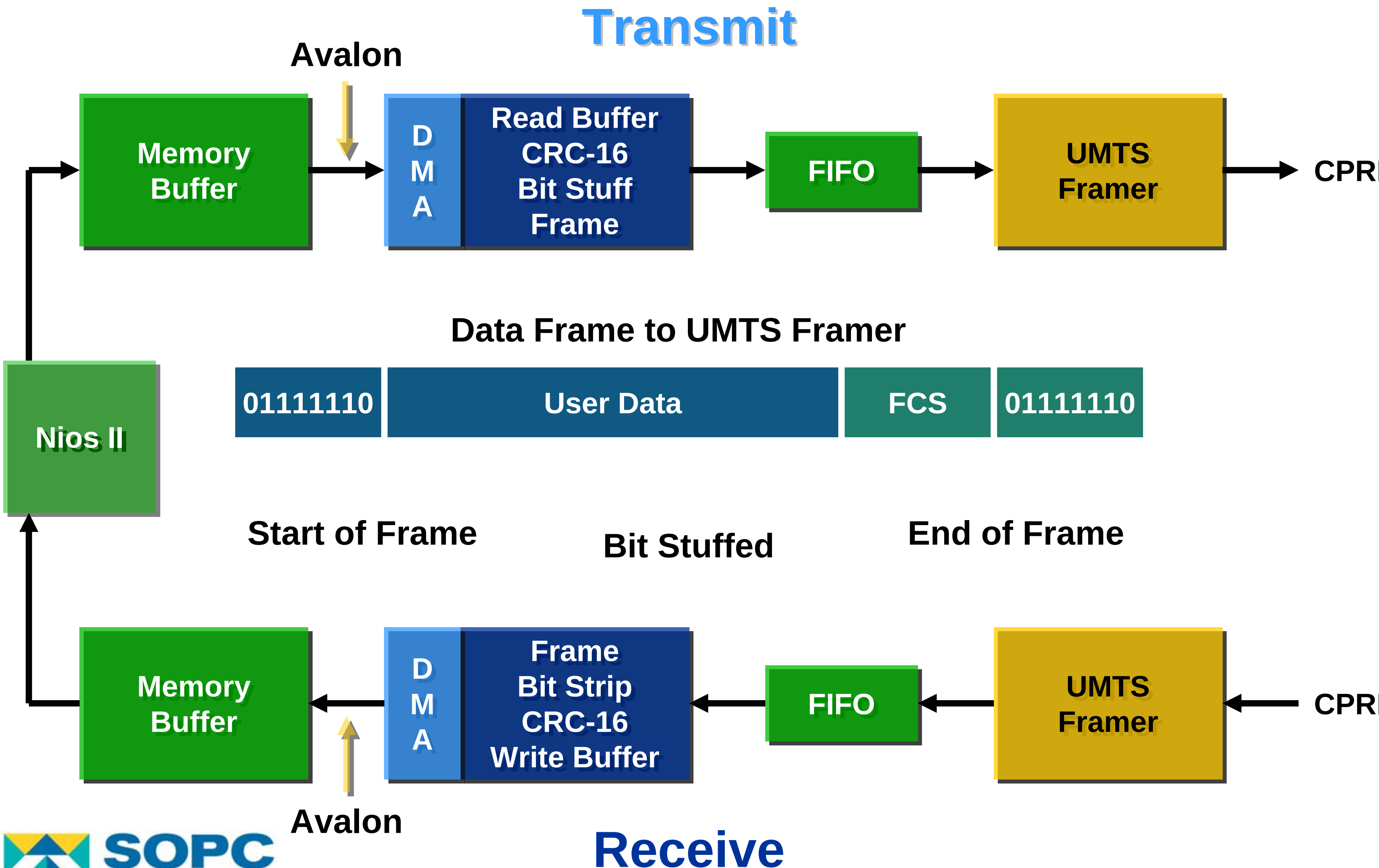
CPRI Controller

- 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



CPRI Interface With Smart Peripheral



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 !