

DMA

Adding Custom IP to DMA

LECTURE 11

IOULIIA SKLIAROVA

DMA – Direct Memory Access

Direct Memory Access transfers the block of data between the memory and peripheral devices of the system, without the participation of the processor.

The unit that controls the activity of accessing memory directly is called a **DMA controller** (DMAC).

Until now, when it was necessary to transfer any data from/to a peripheral device, the MB was fully involved in the data transfer process (it couldn't get involved in any other activity during data transfer).

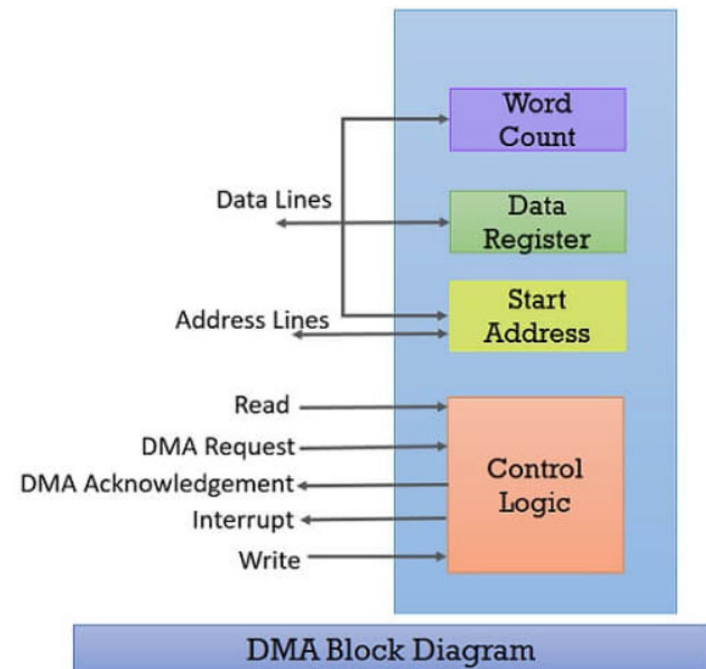
This approach is not useful for transferring large blocks of data.

The DMA controller completes this task at a faster rate and is also effective for transfer of large data blocks.

DMA Controller

The processor instructs the DMA controller by sending the following information:

- Whether the data has to be read from memory or the data has to be written to the memory.
- The **starting address** of/ for the data block in the memory, from where the data block in memory has to be read or where the data block has to be written in memory.
- The **word count**, i.e. how many words are to be read or written.
- **Address of device** that wants to read or write data.



DMA Advantages and Disadvantages

Advantages:

- Transferring the data without the involvement of the processor will **speed up** the read-write task.
- DMA **reduces the clock cycles** required to read or write a block of data.
- Implementing DMA also **reduces the overhead** of the processor.

Disadvantages:

- As it is a hardware unit, it would **cost** to implement a DMA controller in the system.
- Cache **coherence** problem can occur while using DMA controller.

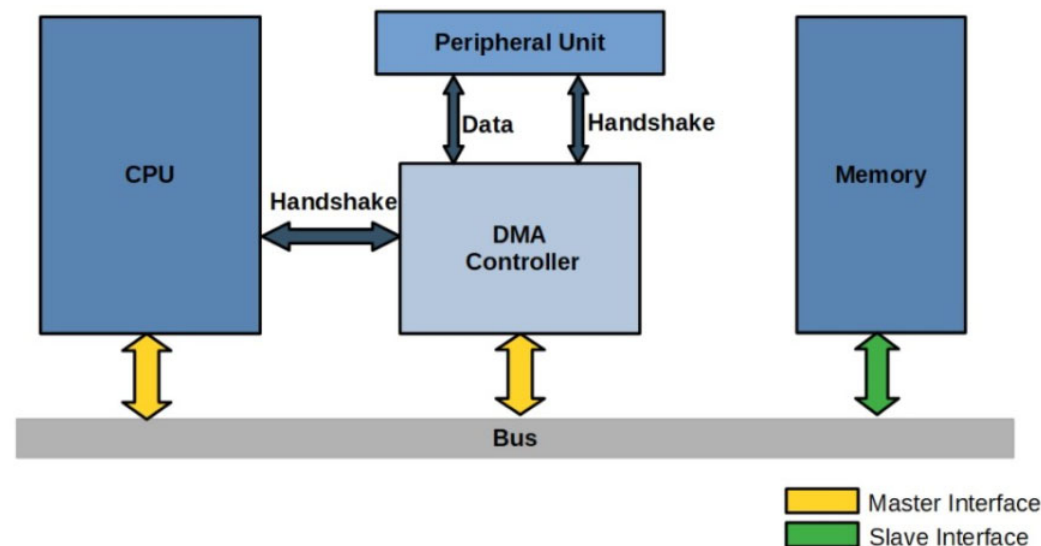
Simplified DMA Block Diagram

All except the peripheral unit are connected on the same bus.

As the CPU and the DMA controller must be able to initiate transfers they have master interfaces.

Although the goal is to have DMA that operates independently, the CPU is the one that has to configure the DMA controller to perform transfers.

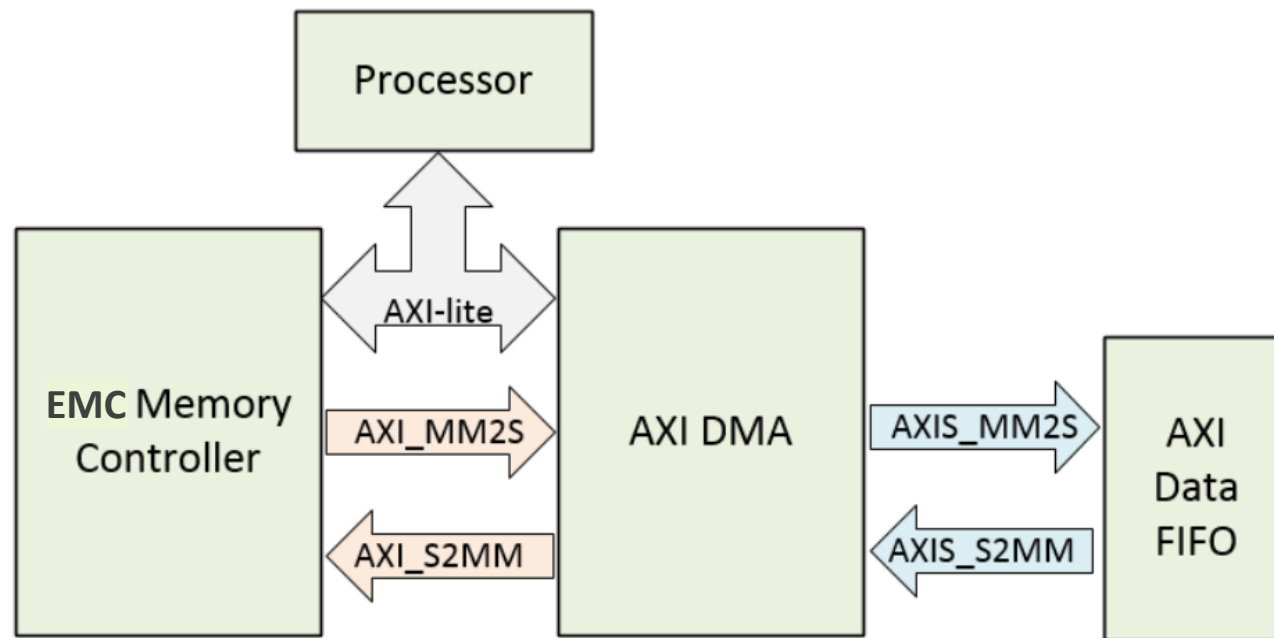
The DMA controller can be **dedicated to a specific DMA-capable peripheral unit** or can be a more general DMA able to access various types of memory-mapped peripherals.



Example 1 - Loopback

We'll use the DMA to transfer data from external memory to an IP block and back to the memory.

The IP block could be any kind of data producer/consumer, but in this example we will use a simple FIFO to create a loopback.



Nexys-4 External Memory

The Nexys-4 board contains two external memories:

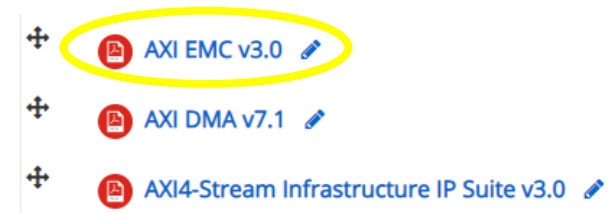
- a **128 Mb Cellular RAM** (pseudo-static DRAM)
- a 128 Mb non-volatile serial Flash device

The Cellular RAM has an SRAM interface.

The 16MB Cellular RAM has a 16-bit bus that supports 8 or 16 bit data access.

AXI **External Memory Controller (EMC)** is a soft Xilinx IP core for use with external memory devices.

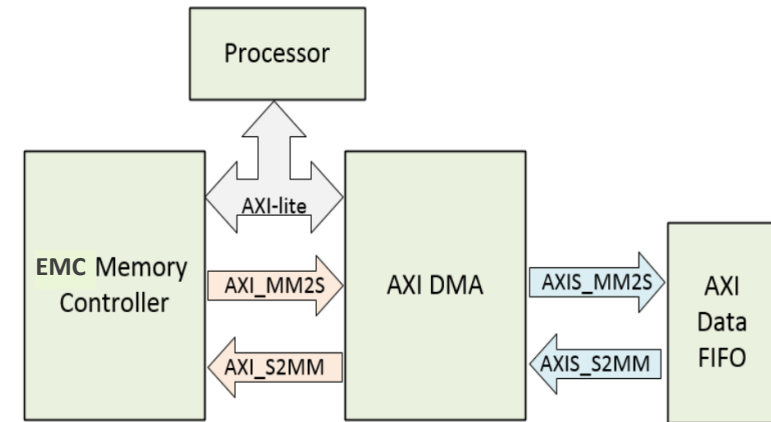
The core provides an AXI4 Slave Interface that can be connected to AXI4 Master or Interconnect devices in the AXI4 systems.



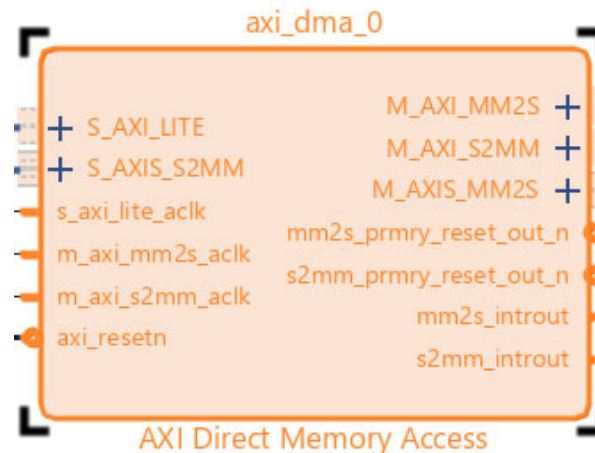
AXI DMA

The **AXI Direct Memory Access** (AXI DMA) IP core provides high-bandwidth direct memory access between the AXI4 memory mapped and AXI4-Stream IP interfaces.

Primary high-speed DMA data movement between system memory and stream target is through the AXI4 Read Master to AXI4 memory-mapped to stream (MM2S) Master, and AXI stream to memory-mapped (S2MM) Slave to AXI4 Write Master.

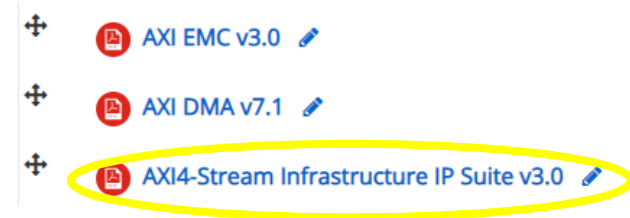


to program the DMA transfer
from peripheral to memory



from memory (interconnect)
to interconnect and then to memory
from DMA to peripheral

FIFO



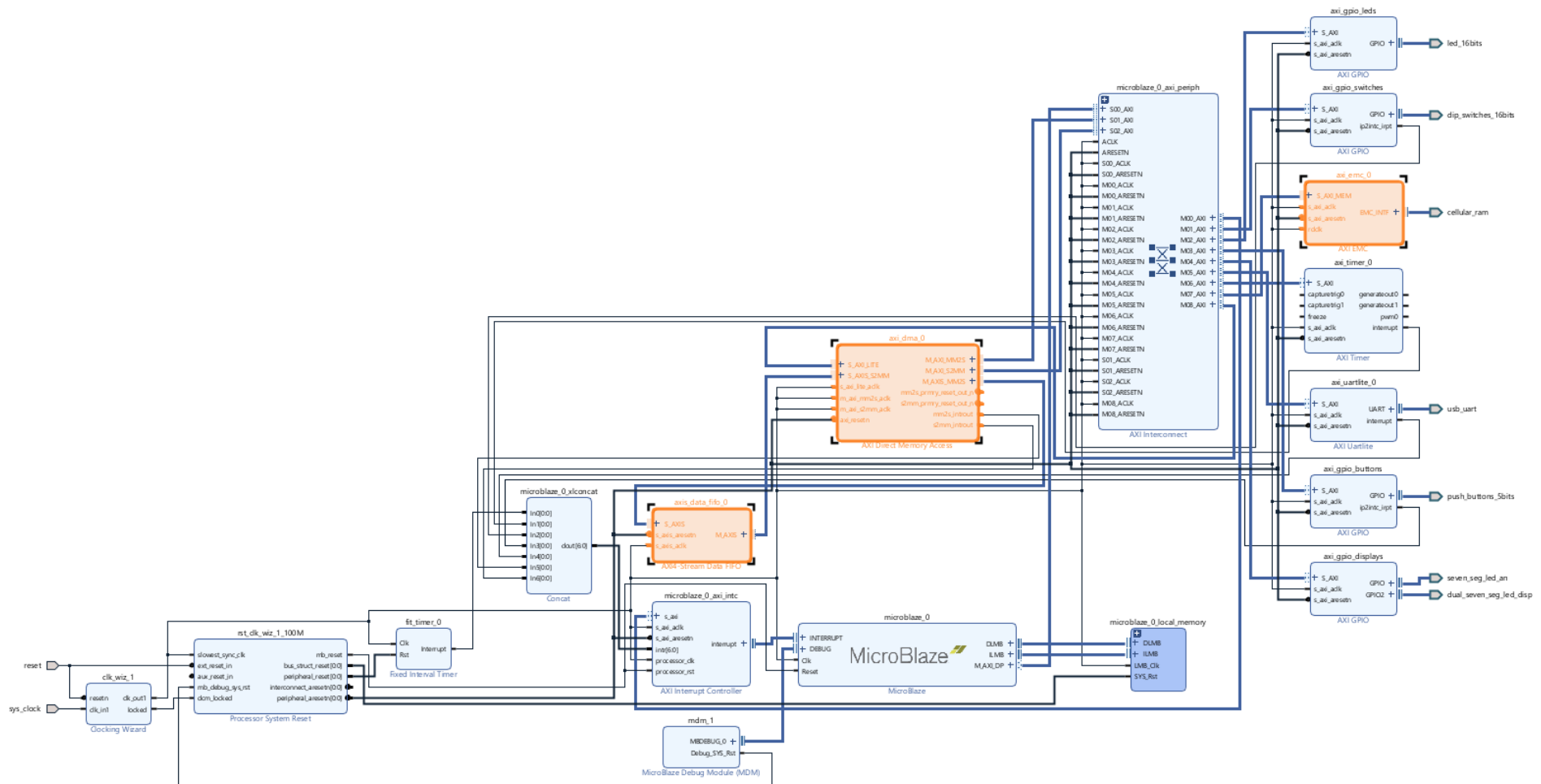
The **AXI4-Stream Infrastructure IP Suite** is a collection of modular IP cores that can be used to rapidly connect AXI4-Stream master/slave IP systems in an efficient manner.

AXI4-Stream Data FIFO - is capable of providing temporary storage (a buffer) of the AXI4-Stream data.

- Supports FIFO depths from 16-32 678 in powers of 2.
- Supports Distributed RAM, Block RAM, and UltraRAM (on select devices) memory primitive types.
- Utilizes Xilinx Parameterized Macros for automatic constraint generation and FIFO implementation.
- Supports independent read/write clocks and ACLKEN conversion.
- Supports Packet Mode (Store and Forward based on TLAST).
- Supports error correction code (ECC) with optional ECC error injection inputs.
- Optional FIFO flags: write data count, almost full, programmable full, read data count, almost empty, and programmable empty.



Block Design



Address Editor

Diagram x Address Editor x Address Map x						
<div> <div> <div>Q</div> <div>≡</div> <div>⇅</div> <div>↴</div> <div>↵</div> </div> <div> <input checked="" type="checkbox"/> Assigned (13) <input checked="" type="checkbox"/> Unassigned (0) <input checked="" type="checkbox"/> Excluded (16) <div>Hide All</div> </div> </div>						
Name	Interface	Slave Segment	Master Base Address	Range	Master High Address	
Network 0						
/axi_dma_0						
/axi_dma_0/Data_MM2S (32 address bits : 4G)						
/axi_emc_0/S_AXI_MEM	S_AXI_MEM	Mem0	0x6000_0000	16M	0x60FF_FFFF	
Excluded (8)						
/axi_dma_0/Data_S2MM (32 address bits : 4G)						
/axi_emc_0/S_AXI_MEM	S_AXI_MEM	Mem0	0x6000_0000	16M	0x60FF_FFFF	
Excluded (8)						
/microblaze_0						
/microblaze_0/Data (32 address bits : 4G)						
/axi_dma_0/S_AXI_LITE	S_AXI_LITE	Reg	0x41E0_0000	64K	0x41E0_FFFF	
/axi_emc_0/S_AXI_MEM	S_AXI_MEM	Mem0	0x6000_0000	16M	0x60FF_FFFF	
/axi_gpio_buttons/S_AXI	S_AXI	Reg	0x4002_0000	64K	0x4002_FFFF	
/axi_gpio_displays/S_AXI	S_AXI	Reg	0x4003_0000	64K	0x4003_FFFF	
/axi_gpio_leds/S_AXI	S_AXI	Reg	0x4001_0000	64K	0x4001_FFFF	
/axi_gpio_switches/S_AXI	S_AXI	Reg	0x4000_0000	64K	0x4000_FFFF	
/axi_timer_0/S_AXI	S_AXI	Reg	0x41C0_0000	64K	0x41C0_FFFF	
/axi_uartlite_0/S_AXI	S_AXI	Reg	0x4060_0000	64K	0x4060_FFFF	
/microblaze_0_axi_intc/S_AXI	s_axi	Reg	0x4120_0000	64K	0x4120_FFFF	
/microblaze_0_local_memory/dlmb_bram_if_cntlr/SLMB	SLMB	Mem	0x0000_0000	64K	0x0000_FFFF	
Network 1						
/microblaze_0						
/microblaze_0/Instruction (32 address bits : 4G)						
/microblaze_0_local_memory/ilmb_bram_if_cntlr/SLMB	SLMB	Mem	0x0000_0000	64K	0x0000_FFFF	

Loopback Example – Further Steps

Generate output products

Create HDL Wrapper

Generate Bitstream

Export Hardware

Launch Vitis

Create a new standalone application – [DMA_App](#)

Board Support Package

A **Board Support Package (BSP)** is a collection of drivers customized to the provided hardware description.

Every application must be associated with a BSP.

The screenshot displays the STM32CubeIDE interface. On the left, the Explorer view shows the project structure with 'platform.spr' highlighted. The main window shows the 'Board Support Package' configuration for 'standalone_microblaze_0'. The 'Board Support Package' section includes buttons for 'Modify BSP Settings...' and 'Reset BSP Sources'. Below this, the 'Operating System' section shows 'Name: standalone' and 'Version: 7.3'. The 'Drivers' tab is active, displaying a table of drivers and their documentation links. The 'axi_dma_0' driver is highlighted, and its 'Import Examples' link is also highlighted.

Name	Driver	Documentation	Examples
axi_dma_0	axidma	Documentation Link	Import Examples
axi_emc_0	emc	Documentation Link	-
axi_gpio_buttons	gpio	-	Import Examples
axi_gpio_displays	gpio	-	Import Examples
axi_gpio_leds	gpio	-	Import Examples
axi_gpio_switches	gpio	-	Import Examples

Xilinx Example C Code

The screenshot displays the Xilinx IDE interface. On the left, the Explorer pane shows the project hierarchy for 'DMA'. The 'xaxidma_example_simple_poll_1_system' project is selected, and its source files are visible in the 'src' directory. The Assistant pane at the bottom shows the project configuration, including the 'DMA_App' application and the 'xaxidma_example_simple_poll_1' application.

On the right, the 'Board Support Package' window is open, showing the 'Import Examples' dialog. The 'xaxidma_example_simple_poll' example is selected. Below the dialog, a table lists the drivers and libraries available for the board.

Name	Driver	Doc
axi_dma_0	axidma	Do...
axi_emc_0	emc	Do...
axi_gpio_buttons	gpio	-
axi_gpio_displays	gpio	-
axi_gpio_leds	gpio	-
axi_gpio_switches	gpio	-

The bottom status bar shows '0 errors, 1 warning, 0 others' and a list of warnings.

Memory Base Address

DMA_Vitis - xaxidma_example_simple_poll_1/src/xaxidma_example_simple_poll.c - Vitis IDE

File Edit Search Xilinx Project Window Help

Explorer

- xtmrctr_ih
- xtmrctr_lh
- xtmrctr.h
- uartlite_ih
- uartlite_lh
- uartlite.h
- bsplib
- system.mss
- DMA.spfm
- DMA.xpfm
- hw
- logs
- microblaze_0
- resources
- platform.spr
- platform.tcl
- DMA_App_system [DMA]
- xaxidma_example_simple_poll_1_system [DMA]
 - Binaries
 - Includes
 - Debug
 - Release
 - xaxidma_example_simple_poll.c

Assistant

- DMA [Platform]
- DMA_App_system [System]
- DMA_App [Application]
- xaxidma_example_simple_poll_1_system [System]
 - xaxidma_example_simple_poll_1 [Application]

helloworld.c

```
76 #define DDR_BASE_ADDR XPAR_AXI_7SDDR_0_S_AXI_BASEADDR
77 #elif defined (XPAR_MIG7SERIES_0_BASEADDR)
78 #define DDR_BASE_ADDR XPAR_MIG7SERIES_0_BASEADDR
79 #elif defined (XPAR_MIG_0_BASEADDR)
80 #define DDR_BASE_ADDR XPAR_MIG_0_BASEADDR
81 #elif defined (XPAR_PSU_DDR_0_S_AXI_BASEADDR)
82 #define DDR_BASE_ADDR XPAR_PSU_DDR_0_S_AXI_BASEADDR
83 #endif
84
85 #ifndef DDR_BASE_ADDR
86 #warning CHECK FOR THE VALID DDR ADDRESS IN XPARAMETERS.H, \
87 #define MEM_BASE_ADDR XPAR_AXI_EMC_0_S_AXI_MEM0_BASEADDR //0x01000000
88 #endif
89
90 #define MEM_BASE_ADDR (DDR_BASE_ADDR + 0x1000000)
91 #endif
92
93 #define TX_BUFFER_BASE (MEM_BASE_ADDR + 0x00100000)
94 #define RX_BUFFER_BASE (MEM_BASE_ADDR + 0x00300000)
95 #define RX_BUFFER_HIGH (MEM_BASE_ADDR + 0x004FFFFFF)
96
97 #define MAX_PKT_LEN 0x20
98
99 #define TEST_START_VALUE 0xC
100
101 #define NUMBER_OF_TRANSFERS 10
102
103 /***** Type Definitions *****/
104
105 /***** Macros (Inline Functions) Definitions *****/
106
107
```

Outline

- xaxidma.h
- xparameters.h
- xdebug.h
- uart550_ih
- DMA_DEV_ID
- DDR_BASE_ADDR
- DDR_BASE_ADDR
- DDR_BASE_ADDR
- DDR_BASE_ADDR
- MEM_BASE_ADDR
- MEM_BASE_ADDR
- TX_BUFFER_BASE
- RX_BUFFER_BASE
- RX_BUFFER_HIGH
- MAX_PKT_LEN
- TEST_START_VALUE
- NUMBER_OF_TRANSFERS
- xi_printf(const char*, ...) : void
- XAxiDma_SimplePollExample(u16)
- CheckData(void) : int
- AxiDma : XAxiDma
- main() : int
- Uart550_Setup(void) : void
- XAxiDma_SimplePollExample(u16)
- CheckData(void) : int

Console

0 errors, 1 warning, 0 others

Description	Resource	Path	Location	Type
Warnings (1 item)				
#warning CHECK FOR THE VALID DDR ADDRESS If xaxidma_exam...		/xaxidma_exam...	line 86	C/C++

Vitis Serial Terminal

Connected to: Serial (COM4, 9600, 0, 8)

Connected to COM4 at 9600
Hello World

Successfully ran Hello World application
--- Entering main() ---
Successfully ran XAxiDma_SimplePoll Example
--- Exiting main() ---

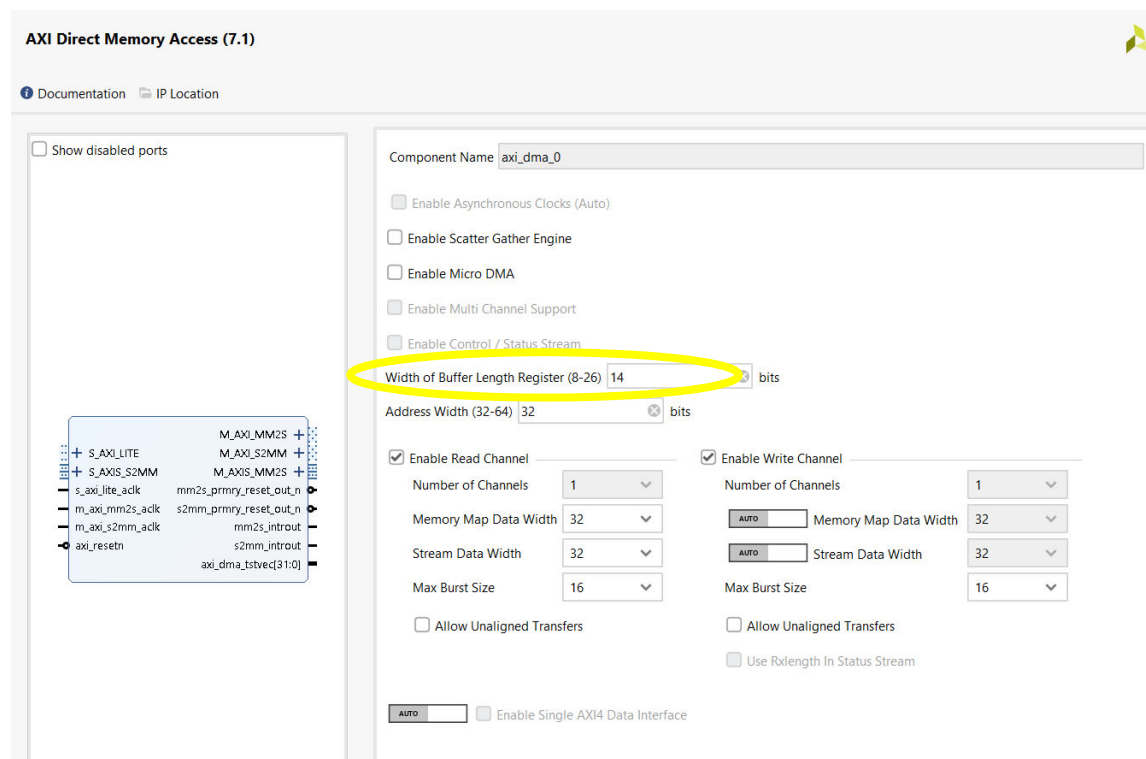
Writable Smart Insert 97 : 29 : 3917 14:50 24/05/2021

Example 2 – Starting Point

Create a new project

Import the BD from the previous class (with EMC, DMA, FIFO (depth=4096)...)

In the DMA you can specify the maximum transfer length (to $2^{14}-1 = 16_383$ Bytes):



Adding the IP and Further Steps

Create a new AXI-Stream connected IP (**ReverseEndiannessCop**) but use a new (modified) VHDL code (available on eLearning)

- processes TLAST and TSTRB signals. Why?

Connect FIFO M_AXIS to **ReverseEndiannessCop** S00_AXIS

Connect **ReverseEndiannessCop** M00_AXIS to DMA S_AXIS_S2MM

Run Connection Automation

Generate Output Products

Create HDL Wrapper

Generate Bitstream

Export Hardware (**DMA_ReverseEndianness.xsa**)

Launch Vitis

Address Editor

Diagram × Address Editor × Address Map × IP Catalog × Project Summary ×						
<input type="checkbox"/> Assigned (13) <input checked="" type="checkbox"/> Unassigned (0) <input checked="" type="checkbox"/> Excluded (16) <input type="button" value="Hide All"/>						
Name	Interface	Slave Segment	Master Base Address	Range	Master High Address	
Network 0						
/axi_dma_0						
/axi_dma_0/Data_MM2S (32 address bits : 4G)						
/axi_emc_0/S_AXI_MEM	S_AXI_MEM	Mem0	0x6000_0000	16M	0x60FF_FFFF	
> Excluded (8)						
/axi_dma_0/Data_S2MM (32 address bits : 4G)						
/axi_emc_0/S_AXI_MEM	S_AXI_MEM	Mem0	0x6000_0000	16M	0x60FF_FFFF	
> Excluded (8)						
/microblaze_0						
/microblaze_0/Data (32 address bits : 4G)						
/axi_dma_0/S_AXI_LITE	S_AXI_LITE	Reg	0x41E0_0000	64K	0x41E0_FFFF	
/axi_emc_0/S_AXI_MEM	S_AXI_MEM	Mem0	0x6000_0000	16M	0x60FF_FFFF	
/axi_gpio_buttons/S_AXI	S_AXI	Reg	0x4002_0000	64K	0x4002_FFFF	
/axi_gpio_displays/S_AXI	S_AXI	Reg	0x4003_0000	64K	0x4003_FFFF	
/axi_gpio_leds/S_AXI	S_AXI	Reg	0x4001_0000	64K	0x4001_FFFF	
/axi_gpio_switches/S_AXI	S_AXI	Reg	0x4000_0000	64K	0x4000_FFFF	
/axi_timer_0/S_AXI	S_AXI	Reg	0x41C0_0000	64K	0x41C0_FFFF	
/axi_uartlite_0/S_AXI	S_AXI	Reg	0x4060_0000	64K	0x4060_FFFF	
/microblaze_0_axi_intc/S_AXI	s_axi	Reg	0x4120_0000	64K	0x4120_FFFF	
/microblaze_0_local_memory/dlmb_bram_if_cntlr/SLMB	SLMB	Mem	0x0000_0000	64K	0x0000_FFFF	
Network 1						
/microblaze_0						
/microblaze_0/Instruction (32 address bits : 4G)						
/microblaze_0_local_memory/ilmb_bram_if_cntlr/SLMB	SLMB	Mem	0x0000_0000	64K	0x0000_FFFF	

Vitis

The C code is given on eLearning
([DMAEndianness.c](#))

There is no need to correct the IP
makefiles

Configure the right stack and heap
size in the linker script and map
memories as indicated:

Linker Script: lscript.ld

A linker script is used to control where different sections of an executable are placed in memory. In this page, you can define new memory regions, and change the assignment of sections to memory regions.

Available Memory Regions

Name	Base Address	Size
microblaze_0_local_memory_ilmb_bram_if_cntlr_...	0x50	0xFFB0
axi_emc_0_MEM0_BASEADDR_Mem0	0x60000000	0x1000000

Stack and Heap Sizes

Stack Size
Heap Size

Section to Memory Region Mapping

Section Name	Memory Region
.text	microblaze_0_local_memory_ilmb_bram_if_cntlr_Mem_micro...
.note.gnu.build-id	microblaze_0_local_memory_ilmb_bram_if_cntlr_Mem_micro...
.init	microblaze_0_local_memory_ilmb_bram_if_cntlr_Mem_micro...
.fini	microblaze_0_local_memory_ilmb_bram_if_cntlr_Mem_micro...
.ctors	microblaze_0_local_memory_ilmb_bram_if_cntlr_Mem_micro...
.dtors	microblaze_0_local_memory_ilmb_bram_if_cntlr_Mem_micro...
.rodata	microblaze_0_local_memory_ilmb_bram_if_cntlr_Mem_micro...
.sdata2	microblaze_0_local_memory_ilmb_bram_if_cntlr_Mem_micro...
.sbss2	microblaze_0_local_memory_ilmb_bram_if_cntlr_Mem_micro...
.data	microblaze_0_local_memory_ilmb_bram_if_cntlr_Mem_micro...
.got	microblaze_0_local_memory_ilmb_bram_if_cntlr_Mem_micro...
.got1	microblaze_0_local_memory_ilmb_bram_if_cntlr_Mem_micro...
.got2	microblaze_0_local_memory_ilmb_bram_if_cntlr_Mem_micro...
.eh_frame	microblaze_0_local_memory_ilmb_bram_if_cntlr_Mem_micro...
.jcr	microblaze_0_local_memory_ilmb_bram_if_cntlr_Mem_micro...
.gcc_except_table	microblaze_0_local_memory_ilmb_bram_if_cntlr_Mem_micro...
.sdata	microblaze_0_local_memory_ilmb_bram_if_cntlr_Mem_micro...
.sbss	microblaze_0_local_memory_ilmb_bram_if_cntlr_Mem_micro...
.tdata	microblaze_0_local_memory_ilmb_bram_if_cntlr_Mem_micro...
.tbss	microblaze_0_local_memory_ilmb_bram_if_cntlr_Mem_micro...
.bss	axi_emc_0_MEM0_BASEADDR_Mem0
.heap	axi_emc_0_MEM0_BASEADDR_Mem0
.stack	axi_emc_0_MEM0_BASEADDR_Mem0

Performance Analysis

AXI-Stream

Software Only vs. Hardware Assisted Reverse Endianness Demonstration

Array initialization time: 53625 microseconds

00000000 5851F42D 40B18CCF 4BB5F646 47033129 30705B04 20FD5DB4 1A8B7F78

Software only reverse endianness time: 4721 microseconds

00000000 2DF45158 CF8CB140 46F6B54B 29310347 045B7030 B45DFD20 787F8B1A

Checking result: OK

Hardware assisted reverse endianness time: 1601 microseconds

00000000 2DF45158 CF8CB140 46F6B54B 29310347 045B7030 B45DFD20 787F8B1A

Checking result: OK

Hw speedup: ~3

DMA

DMA with Reverse Endianness Demo Program - Entering main()...

Filling memory with pseudo-random data...

Memory initialization time: 101733 microseconds

00000000 5851F42D 40B18CCF 4BB5F646 47033129 30705B04 20FD5DB4 1A8B7F78

Software only reverse endianness time: 27330 microseconds

00000000 2DF45158 CF8CB140 46F6B54B 29310347 045B7030 B45DFD20 787F8B1A

Checking result: OK

Configuring DMA...

DMA running...

Max transfer length in bytes = 16383

DMA Hardware assisted reverse endianness time: 1850 microseconds

00000000 2DF45158 CF8CB140 46F6B54B 29310347 045B7030 B45DFD20 787F8B1A

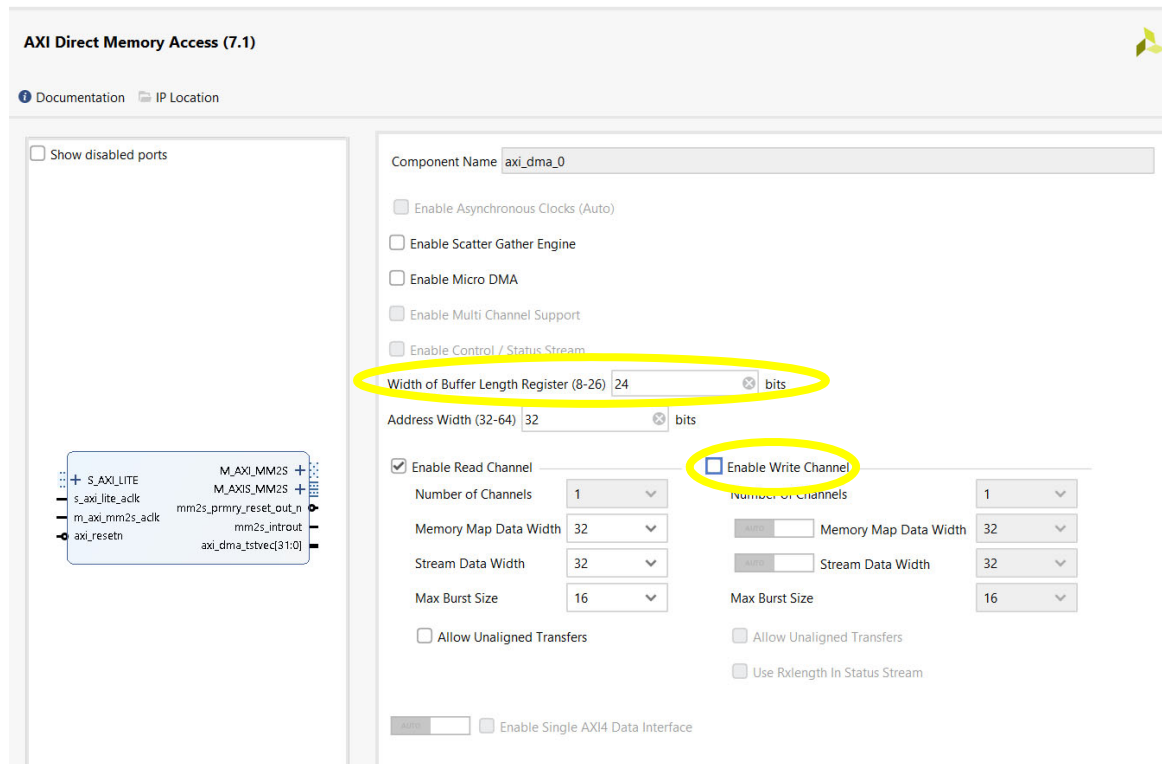
Checking result: OK

Hw speedup: ~15

Example 3 – Starting Point

Create a new project and import the BD from the previous class (having EMC, DMA, FIFO...)

Configure the DMA maximum transfer length (to $2^{24}-1 = 16_777_215$ Bytes) and disable the write channel:



Create and Package New IP (PopCount)

Create and Package New IP

Add Interfaces
Add AXI4 interfaces supported by your peripheral

☐ Enable Interrupt Support

Interfaces

- S00_AXIS
- S00_AXI

S00_AXIS
S00_AXI
PopCount_v1.0

Name: S00_AXIS

Interface Type: Stream

Interface Mode: Slave

Data Width (Bits): 32

Memory Size (Bytes): 64

Number of Registers: 4 [4..512]

< Back Next > Finish Cancel

Adding the IP and Further Steps

Add the **PopCount** IP to the block diagram

Connect FIFO M_AXIS to **PopCount** S00_AXIS

Connect **PopCount** S00_AXI to Interconnect M_09_AXI (add one more master port)

Run Connection Automation

Modify the **PopCount** code

Assign **PopCount** in the address editor

Generate Output Products

Create HDL Wrapper

Generate Bitstream

Export Hardware (**DMA_PopCount.xsa**)

Launch Vitis

Vitis

Design the C code on the basis of the example given on eLearning ([DMAEndianness.c](#))

Correct the IP makefiles (AXI-Lite)

Configure the right stack and heap size in the linker script and map memories as indicated:

Linker Script: lscript.ld

A linker script is used to control where different sections of an executable are placed in memory. In this page, you can define new memory regions, and change the assignment of sections to memory regions.

Available Memory Regions

Name	Base Address	Size
microblaze_0_local_memory_ilmb_bram_if_cntlr_...	0x50	0xFFB0
axi_emc_0_MEM0_BASEADDR_Mem0	0x60000000	0x1000000

Stack and Heap Sizes

Section to Memory Region Mapping

Section Name	Memory Region
.text	microblaze_0_local_memory_ilmb_bram_if_cntlr_Mem_micro...
.note.gnu.build-id	microblaze_0_local_memory_ilmb_bram_if_cntlr_Mem_micro...
.init	microblaze_0_local_memory_ilmb_bram_if_cntlr_Mem_micro...
.fini	microblaze_0_local_memory_ilmb_bram_if_cntlr_Mem_micro...
.ctors	microblaze_0_local_memory_ilmb_bram_if_cntlr_Mem_micro...
.dtors	microblaze_0_local_memory_ilmb_bram_if_cntlr_Mem_micro...
.rodata	microblaze_0_local_memory_ilmb_bram_if_cntlr_Mem_micro...
.sdata2	microblaze_0_local_memory_ilmb_bram_if_cntlr_Mem_micro...
.sbss2	microblaze_0_local_memory_ilmb_bram_if_cntlr_Mem_micro...
.data	microblaze_0_local_memory_ilmb_bram_if_cntlr_Mem_micro...
.got	microblaze_0_local_memory_ilmb_bram_if_cntlr_Mem_micro...
.got1	microblaze_0_local_memory_ilmb_bram_if_cntlr_Mem_micro...
.got2	microblaze_0_local_memory_ilmb_bram_if_cntlr_Mem_micro...
.eh_frame	microblaze_0_local_memory_ilmb_bram_if_cntlr_Mem_micro...
.jcr	microblaze_0_local_memory_ilmb_bram_if_cntlr_Mem_micro...
.gcc_except_table	microblaze_0_local_memory_ilmb_bram_if_cntlr_Mem_micro...
.sdata	microblaze_0_local_memory_ilmb_bram_if_cntlr_Mem_micro...
.sbss	microblaze_0_local_memory_ilmb_bram_if_cntlr_Mem_micro...
.tdata	microblaze_0_local_memory_ilmb_bram_if_cntlr_Mem_micro...
.tbss	microblaze_0_local_memory_ilmb_bram_if_cntlr_Mem_micro...
.bss	axi_emc_0_MEM0_BASEADDR_Mem0
.heap	axi_emc_0_MEM0_BASEADDR_Mem0
.stack	axi_emc_0_MEM0_BASEADDR_Mem0

Final Remarks

At the end of this lecture you should be able to:

- Prepare the hardware platform to support DMA
- Design custom hardware modules supporting DMA

To do:

- Construct the considered hardware platforms
- Test the given applications in Vitis
- Complete lab. 9