Designing a Custom AXI-Stream Peripheral (for loops and variables)

Final Project

DMA

LECTURE 11

IOULIIA SKLIAROVA

Examples (AXI-Stream Coprocessor)

Reverse endianness

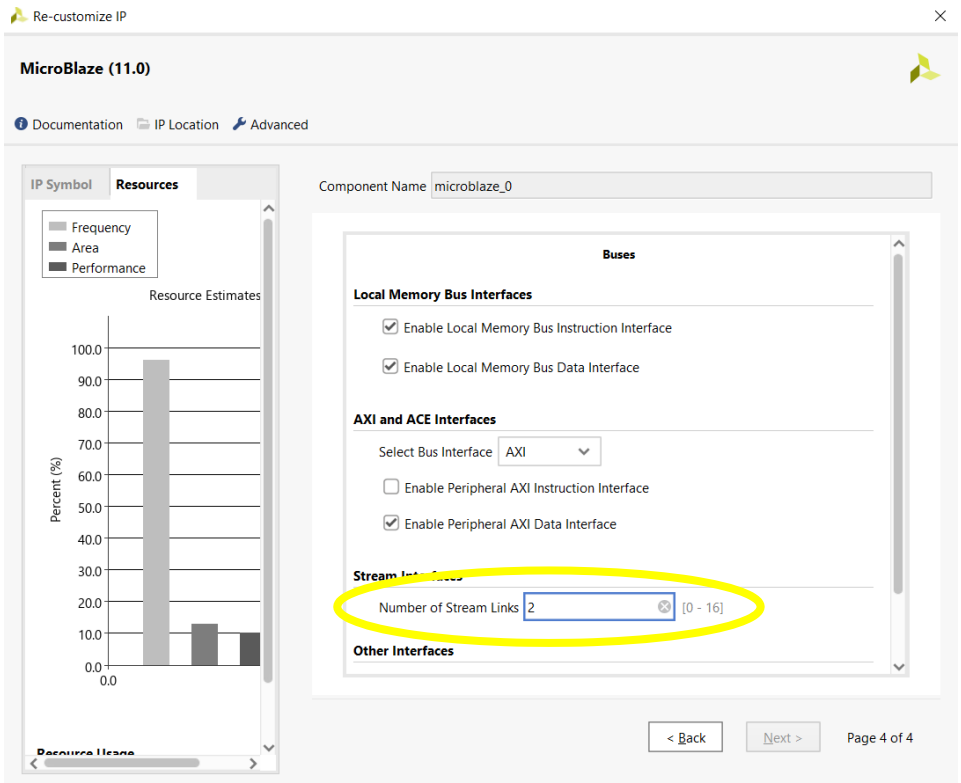
Population count (Hamming weight)

- the number of non-zero entries ('1' bits) in a word of data.
- 0xAB347801 => 10101011_00110100_01111000_00000001 => 13

Example 2 – Starting Point

Continue to work on the same project (as in the previous class)

Change the number of stream links in the MicroBlaze to 2



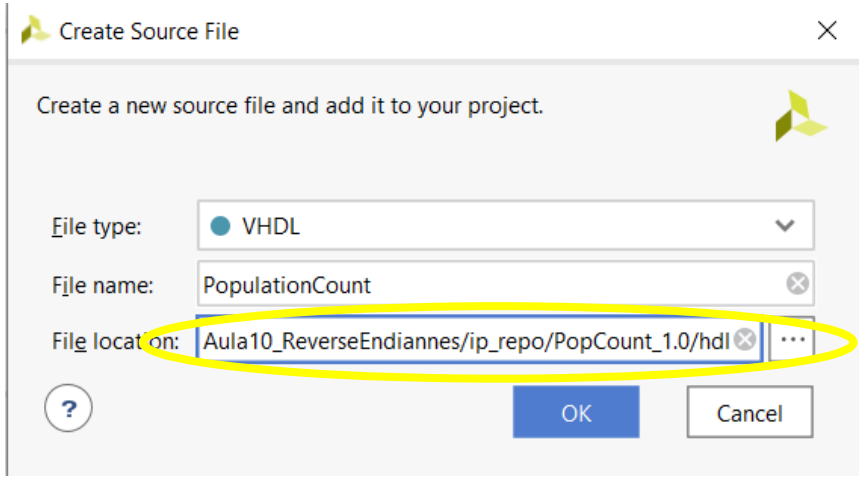
Example 2 – Add New IP

Create and Package new IP - **PopCount**

Edit in IP packager

Change the three “default” files like in Example 1

Create a new source – PopulationCount.vhd (the code is given on eLearning)



Instantiate the PopulationCount module in the slave stream interface:

```
calc: PopulationCount
  generic map(N      => C_S_AXIS_TDATA_WIDTH)
  port map( dataIn   => …);
```

For Loop VHDL Statement

A **for loop** statement is a sequential statement that can be used inside a process.

A **for loop** includes a specification of how many times the body of the loop is to be executed:

```
[loop_label:]
for identifier in discrete_range loop
{ sequential_statement }
end loop [loop_label];
```

The **for loop** statement is used whenever an operation needs to be repeated.

The loop is **unrolled** statically – the number of loop iterations must be known at compile time.

Loop unrolling is a systematic method of achieving parallelism that can be automated.

This comes at a cost of a larger fabric footprint (more FPGA area).

For Loop vs For Generate

The **for loops** are **sequential statements**, **containing sequential statements** (i.e. each iteration is sequenced to be executed after the previous one).

The **for-generate loops** are **concurrent** statements, **containing concurrent statements**, and this is how you can use it to make several instances of a component.

For-generate loops are used when specifying the exact hardware structure.

For loops are more suited for behavioral descriptions.

Variables

For loops are often used with **variables**.

Variables are declared in the declaration part of processes:

```
variable_declaration ⇐
variable identifier { , … } : subtype_indication
[:= expression] ;
```

The syntax of a **variable assignment** statement is given by the rule

```
variable_assignment_statement ⇐
[label :] name := expression ;
```

A variable assignment **immediately overwrites the variable** with a new value (a signal assignment, on the other hand, schedules a new value to be applied to a signal at some later time).

Population Count With a For Loop

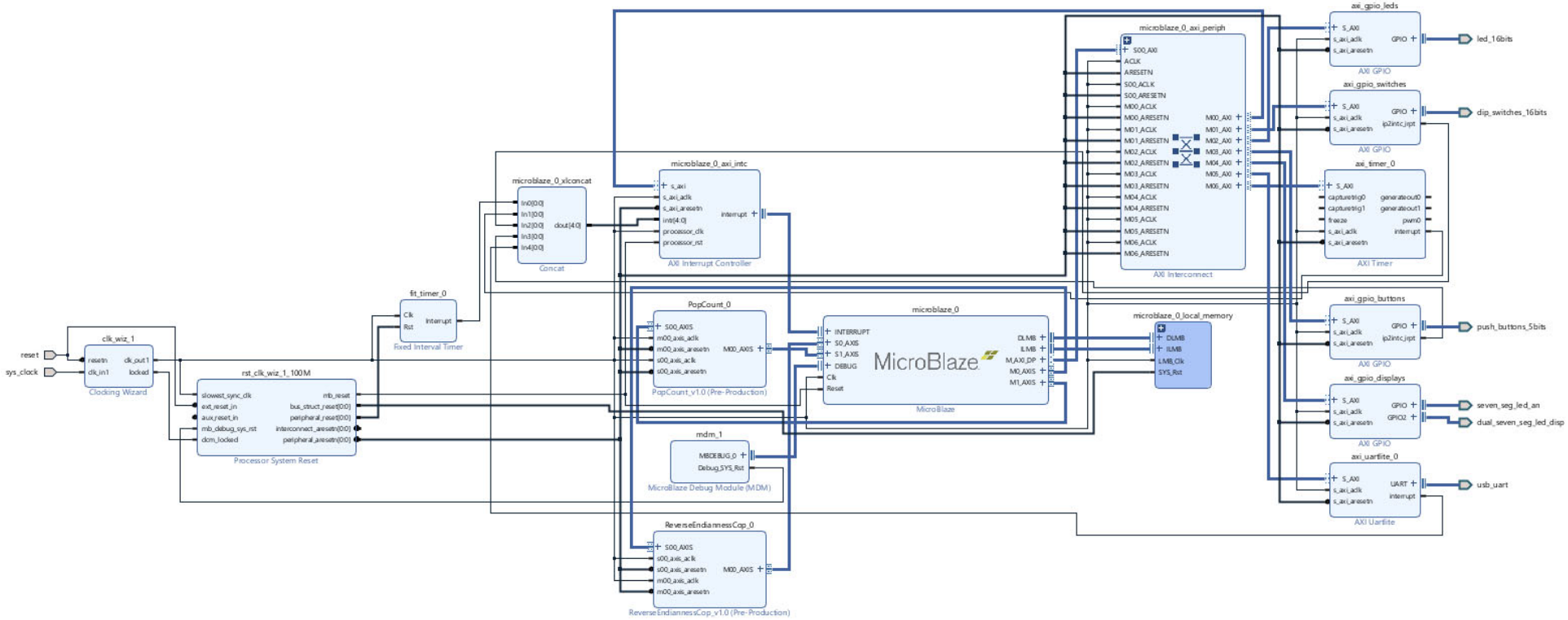
```
entity PopulationCount is
  generic(N     : positive := 4);
  port(dataIn   : in  std_logic_vector(N-1 downto 0);
       cntOut   : out std_logic_vector(N-1 downto 0));
end PopulationCount;

architecture Behavioral of PopulationCount is
    signal s_cnt : natural range 0 to N;
begin
    process(dataIn)
        variable v_cnt : natural range 0 to N;
    begin
        v_cnt := 0;
        for i in 0 to N-1 loop
            if dataIn(i) = '1' then
                v_cnt := v_cnt + 1;
            end if;
        end loop;
        s_cnt <= v_cnt;
    end process;

    cntOut <= std_logic_vector(to_unsigned(s_cnt, N));
end Behavioral;
```

A long sequence of 31 adders will be generated.

Example 2 Block Design



Example 2 – Further Steps

Generate output products

Create HDL Wrapper

Generate Bitstream

Export Hardware

Launch Vitis

Vitis

Write the C code (on the basis of the ReverseEndianness example)

There is no need to correct the IP makefiles

Configure the right stack size

Final Project

No formal guidelines will be given

Hardware + software

Hardware must include MB, memory, standard peripherals, and custom modules

Select an **operation** suitable for hardware (to increase the performance, to have a clearer implementation)

Compare software and hardware implementations of the selected operation

Demonstrate that you are familiar with the design flow:

- write VHDL code
- simulate with a testbench
- incorporate to block design
- do synthesis, implementation, report analysis
- determine critical path, optimize operating frequency
- write software

Final Project - Operation

Select an **operation** suitable for hardware (to increase the performance, to have a clearer implementation)

- an instruction/function not supported by MB (~~popcnt~~, vector operations, complex bitwise/shift logic, specific peripheral, cryptography…)

Repeated operations are not allowed among students

Operations considered during classes are not allowed

Either bring your proposals to labs on June 1/2 or send them to me by e-mail

- title of the project
- brief description of the functionality (one phrase)
- brief description of the proposed architecture
- why to use the suggested custom hardware module?
- test procedure and user interaction

Final Project – Proposal Example

Title of the project:

- Accelerating Population Count with a Hardware Co-Processor

Brief description of the functionality (one phrase):

- System with a DMA-assisted hardware accelerator for calculating population count over a configurable-length array of 32-bit vectors

Brief description of the proposed architecture:

- The system will include a custom hardware module executing the population count operation over a 32-bit input. The module will be used with a DMA controller and an accumulator to process a considerable number (up to 2^{12}) of 32-bit values stored in external cellular RAM. The performance of software and hardware implementations will be analyzed and compared.

Why to use the suggested custom hardware module?

- To reduce the processing time.

Test procedure and user interaction

- Input data will be randomly generated. Software will check the hardware results. Testbench for the accelerator. User interaction through UARTLite and serial terminal.

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 could't get involved in any other activity during data transfer).

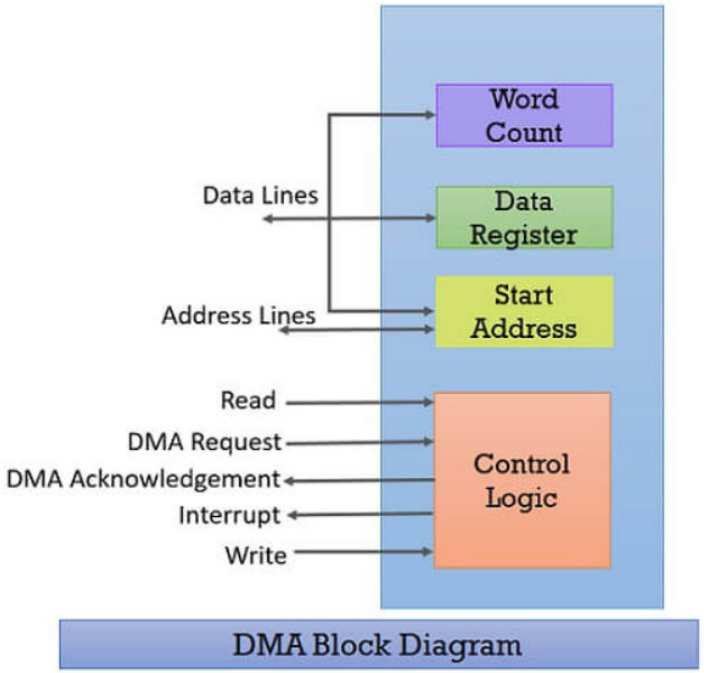
This approach is not useful for transferring large blocks of data.

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 Block Diagram

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.

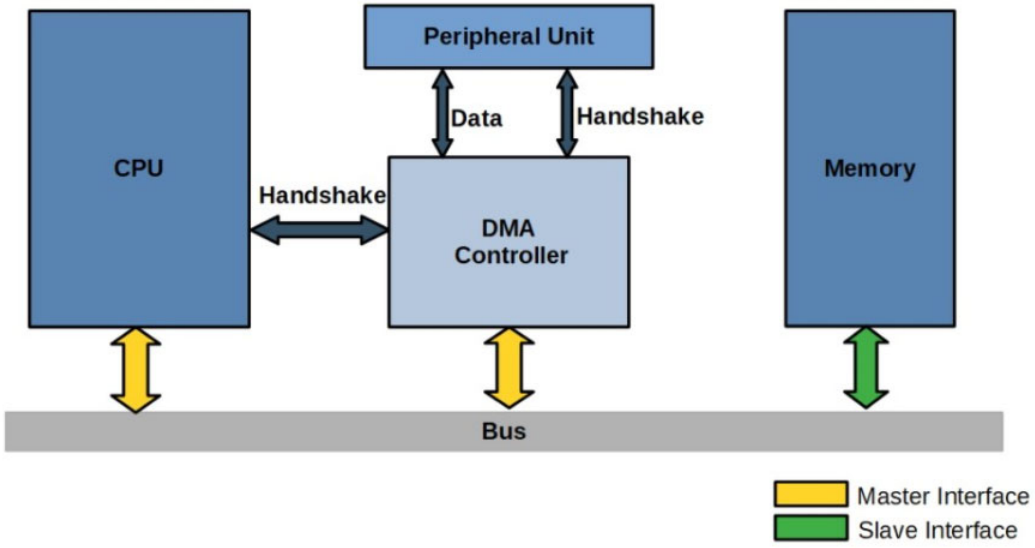
Simplified DMA Block Diagram

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.

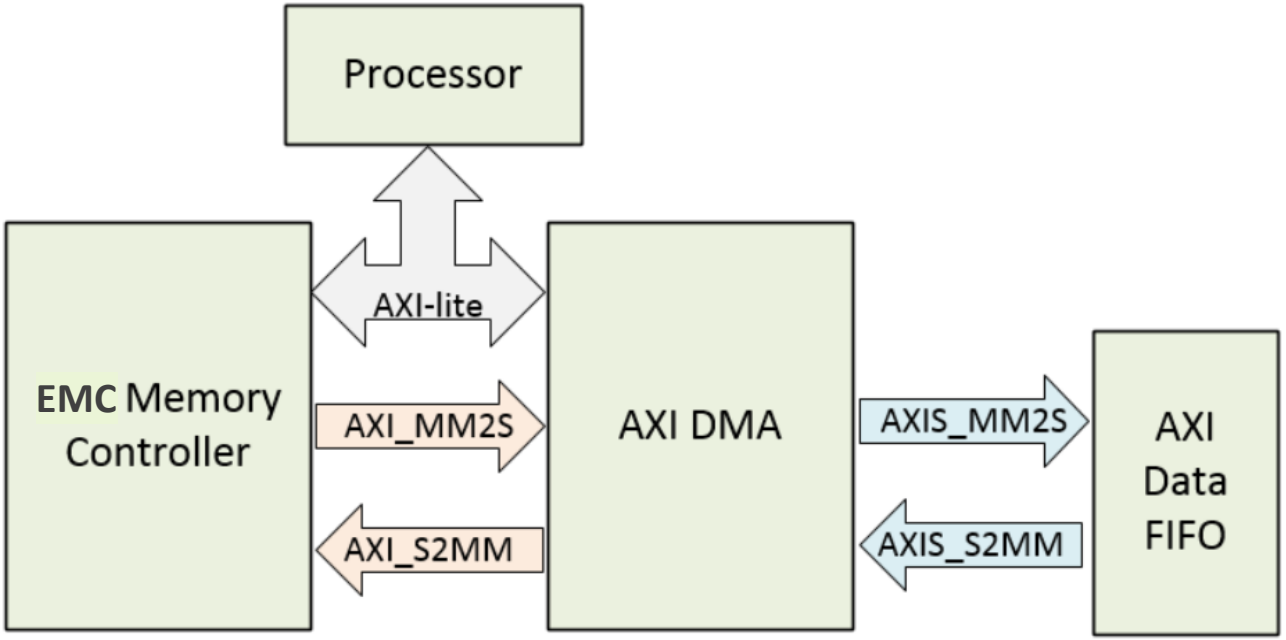
The DMA controller can be **dedicated to a specific DMA-capable peripheral unit** or can be a more general DMA able to access various types of memory-mapped peripherals.



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:

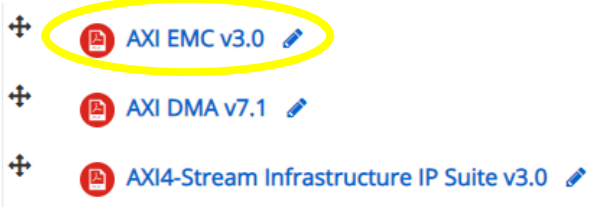
- a **128 Mb Cellular RAM** (pseudo-static DRAM)
- a 128 Mb non-volatile serial Flash device

The Cellular RAM has an SRAM interface.

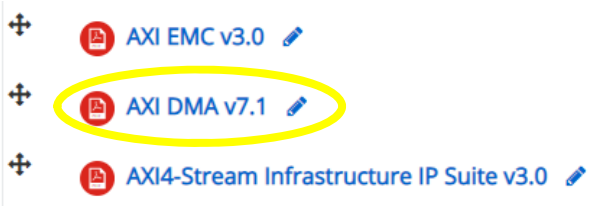
The 16MB Cellular RAM has a 16-bit bus that supports 8 or 16 bit data access.

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.

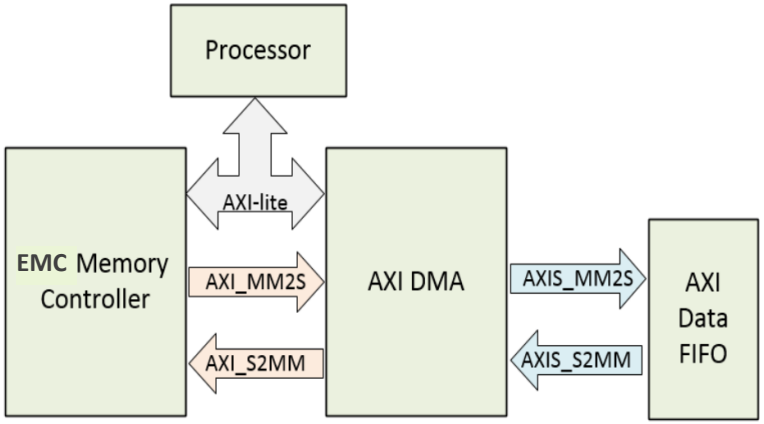


AXI DMA

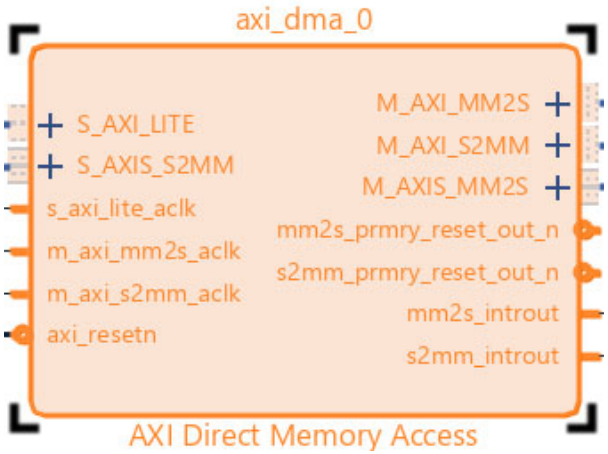


The **AXI Direct Memory Access** (AXI DMA) IP core provides high-bandwidth direct memory access between the AXI4 memory mapped and AXI4-Stream IP interfaces.

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.

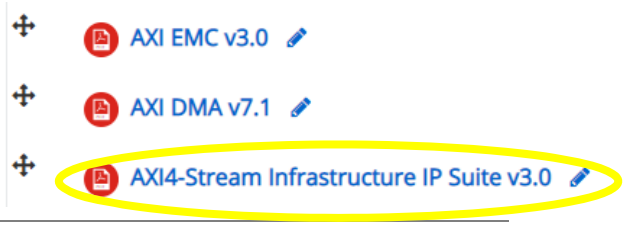


to program the DMA transfer from peripheral to memory



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

FIFO

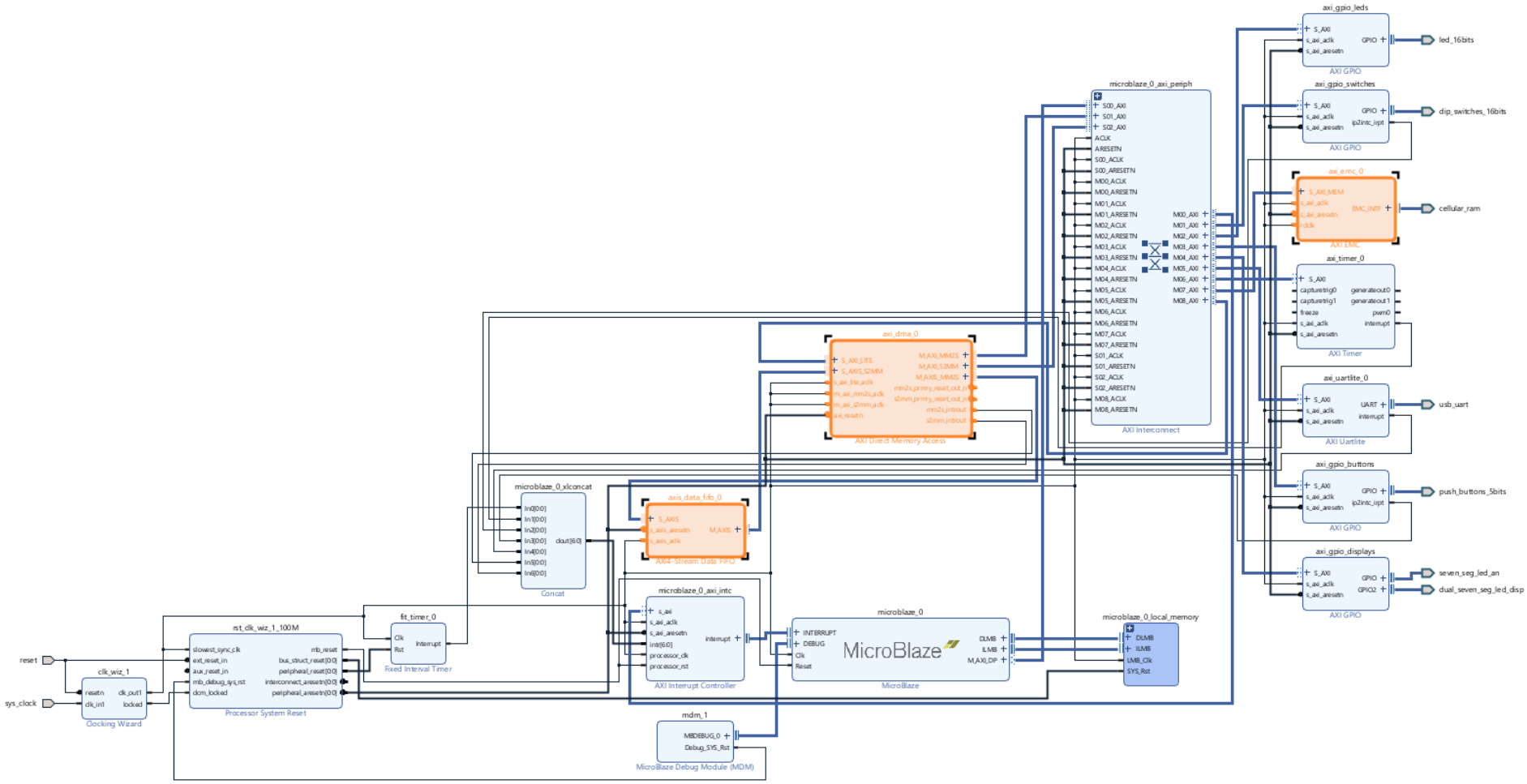


The AXI4-Stream Infrastructure IP Suite is a collection of modular IP cores that can be used to rapidly connect AXI4-Stream master/slave IP systems in an efficient manner.

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 | Address Editor | Address Map

Assigned (13) Unassigned (0) Excluded (16) 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

Loopback Example – Further Steps

Generate output products

Create HDL Wrapper

Generate Bitstream

Export Hardware

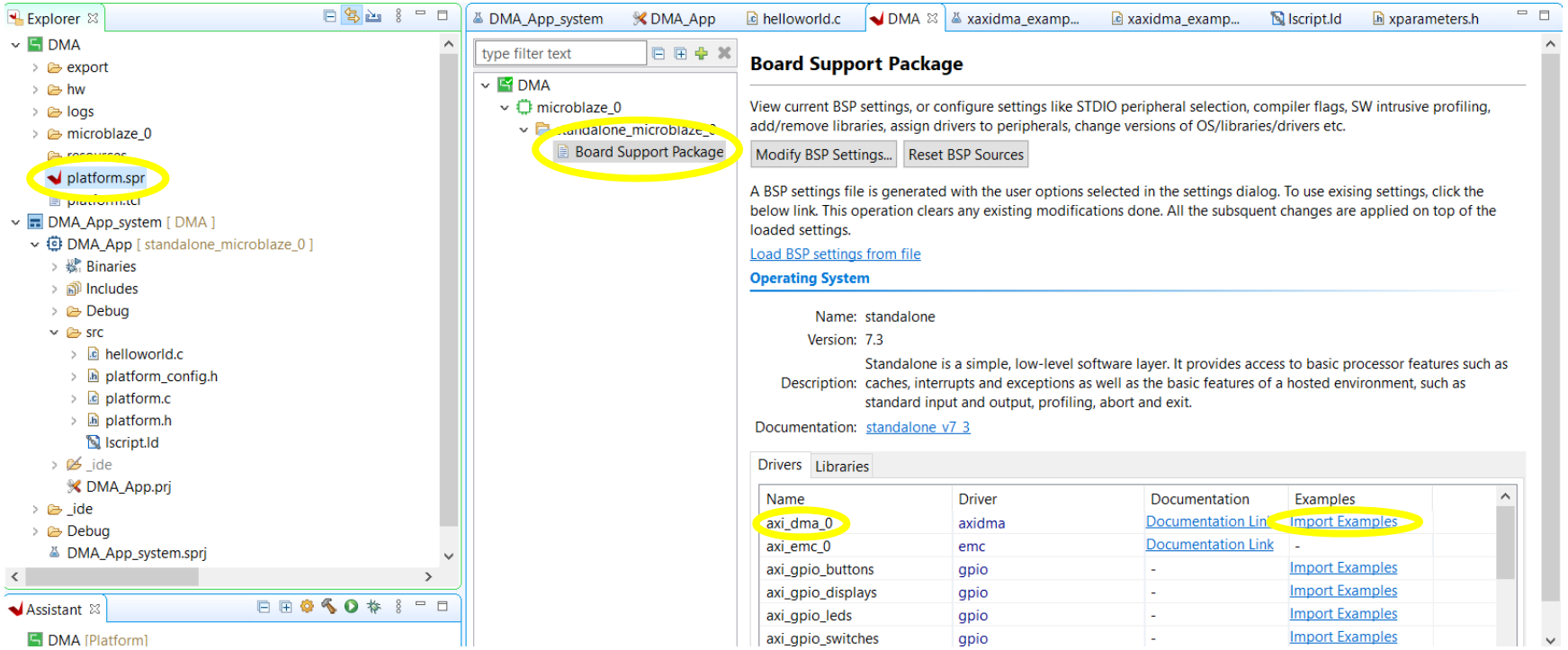
Launch Vitis

Create a new standalone application – DMA_App

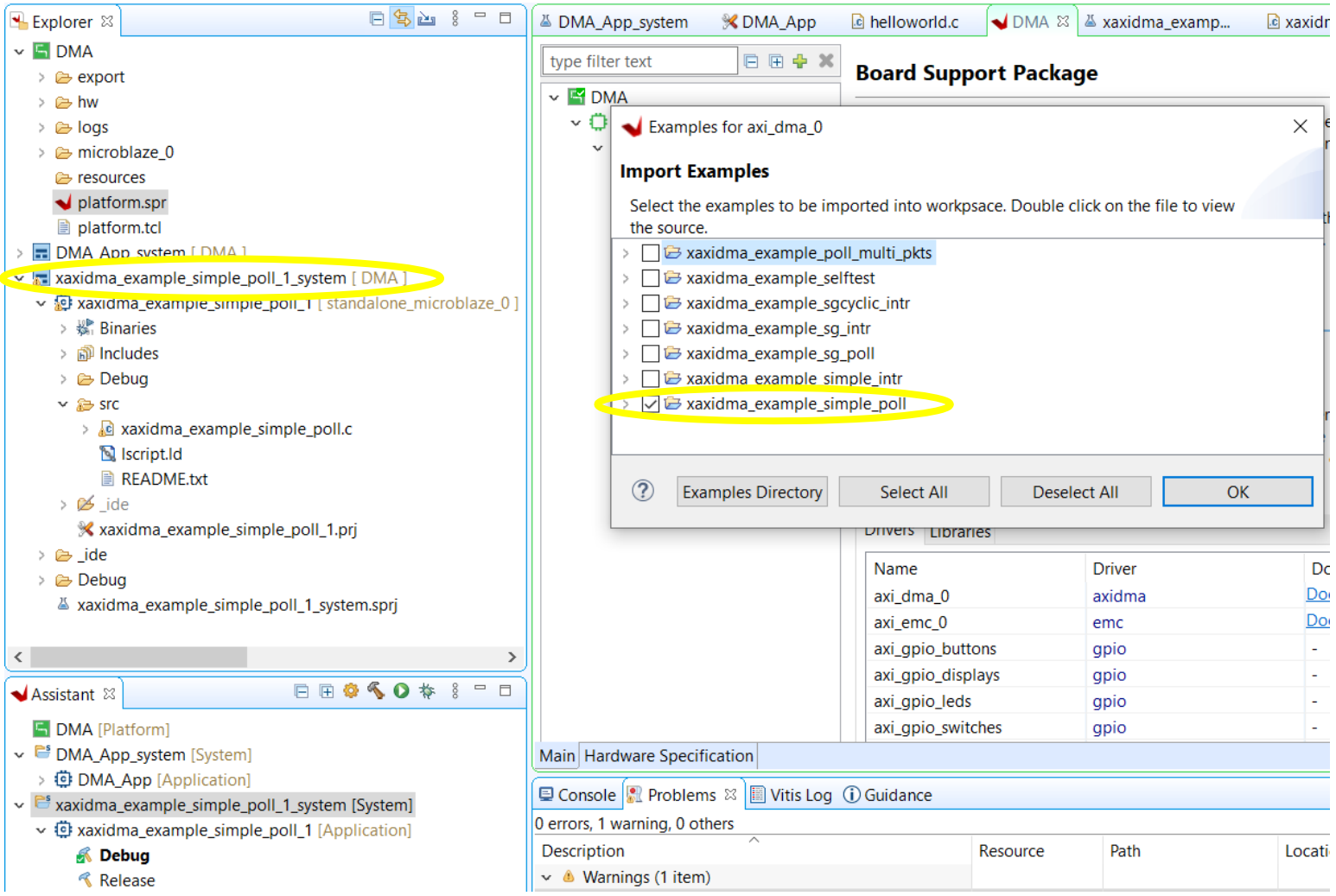
Board Support Package

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

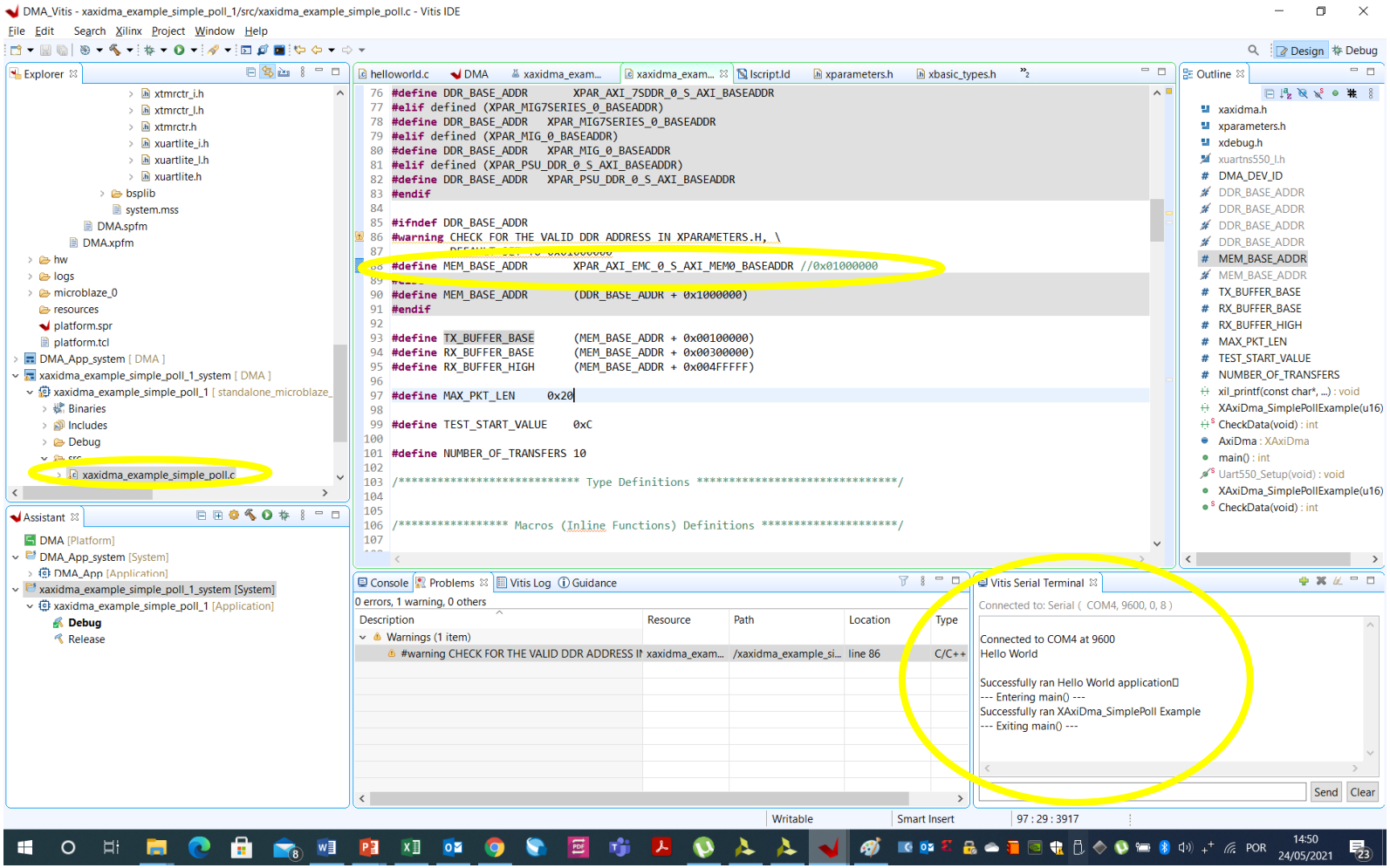
Every application must be associated with a BSP.



Xilinx Example C Code



Memory Base Address



Final Remarks

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

- Design custom hardware modules interacting with the MicroBlaze through AXI-Stream interface
- Write C programs that make use of stream-connected custom hardware
- Prepare your project proposal
- Prepare the hardware platform to support DMA

To do:

- Construct the considered hardware platforms
- Test the given applications in Vitis