Designing a Custom AXI-Stream Peripheral

LECTURE 10

IOULIIA SKLIAROVA

Universidade de Aveiro

Lab. 7 – parts 1-3

Direct the display refresh functions to custom hardware

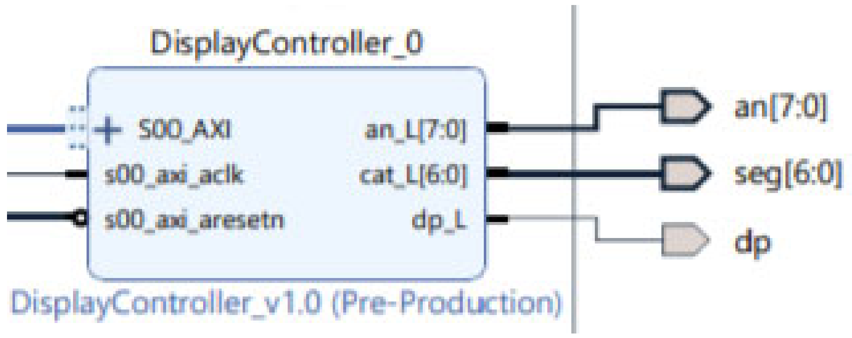
The custom display driver will receive from the MicroBlaze (through AXI-Lite interface):

- Digit enables – 8 bits
- Decimal point enables - 8 bits
- Digit values - 32 bits (8 x 4 bits)

The custom display driver will produce on its outputs:

- an – 8 bits
- seg – 7 bits
- dp – 1 bit

The original GPIO_Displays must be deleted



Lab. 7 – Part 4

Add to the display driver the following configuration options:

- Refresh Rate - 3 bits for configuring the refresh rate to one of the following eight values: 50 Hz, 100 Hz, 200 Hz, 400 Hz, 800 Hz, 1600 Hz, 3200 Hz, 6400 Hz:
  - 0 – 50 Hz
  - 1 – 100 Hz
  - 2 – 200 Hz
  - 3 – 400 Hz
  - 4 – 800 Hz
  - 5 – 1600 Hz
  - 6 – 3200 Hz
  - 7 – 6400 Hz

- Display Brightness - 3 bits for controlling the brightness of the displays (8 levels):
  - 0 – 0%, all displays off
  - 1..6 – equidistant brightness levels
  - 7 – 100%, maximum brightness

Lab. 7 – Refresh Rate

Make the MicroBlaze to set the refresh rate and display brightness based on the values of 6 switches:

- sw[2..0] – refresh rate index -> `refrRate`
- sw[5..3] – brightness index -> `brCtrl`

The external 800 Hz enable signal now does not make sense – remove it.

Generate the respective "enable" internally, based on the requested refresh rate:

- 50 Hz – the enable must be activated once every 2_000_000 clock cycles
- 100 Hz – the enable must be activated once every 1_000_000 clock cycles
- …
- 800 Hz – the enable must be activated once every 125_000 clock cycles
- …
- 6400 Hz – the enable must be activated once every 15_625 clock cycles

Lab. 7 – Refresh Rate (cont.)

Generate the respective "enable" internally from `refrRate`:

- 0 - 50 Hz – the enable must be activated once every 2_000_000 clock cycles
- 1 - 100 Hz – the enable must be activated once every 1_000_000 clock cycles
- …
- 4 - 800 Hz – the enable must be activated once every 125_000 clock cycles
- …
- 7 - 6400 Hz – the enable must be activated once every 15_625 clock cycles

Various encoding alternatives:

```
refrRate : in std_logic_vector(2 downto 0);  …


signal MAX : integer; …
MAX <= 100_000_000 / (2**to_integer(unsigned(refrRate)) * 50);

type TRR is array (0 to 7) of integer range 0 to 2_000_000;
constant REFRESH_RATE_LUT : TRR := (2_000_000, 1_000_000,
500_000, 250_000, 125_000, 62_500, 31_250, 15_625);
```

Lab. 7 – Brightness Control

For eight levels of brightness, let’s assume:

- 0 – 0%, all displays off
- 1 – 14%
- 2 – 29%
- 3 – 43%
- 4 – 57%
- 5 – 71%
- 6 – 86%
- 7 – 100%, maximum brightness

We can control the duty cycle of the *an* output.

Lab. 7 – Refresh Rate & Brightness

For example, for brightness level 2 (29%) and depending on the selected refresh rate:

- 50 Hz
  - the enable must be activated once every 2_000_000 clock cycles (to switch between displays)
  - The selected *an* must be active (0) during 29% of 2_000_000 clock cycles = 580_000 clock cycles and deactivated (1) during the remaining 1_420_000 clock cycles.
- 100 Hz
  - the enable must be activated once every 1_000_000 clock cycles (to switch between displays)
  - The selected *an* must be active (0) during 29% of 1_000_000 clock cycles = 290_000 clock cycles and deactivated (1) during the remaining 710_000 clock cycles.
- 200 Hz
  - the enable must be activated once every 500_000 clock cycles (to switch between displays)
  - The selected *an* must be active (0) during 29% of 500_000 clock cycles = 145_000 clock cycles and deactivated (1) during the remaining 355_000 clock cycles.
- …
- 6400 Hz
  - the enable must be activated once every 15_625 clock cycles (to switch between displays)
  - The selected *an* must be active (0) during 29% of 15_625 clock cycles = 4_531 clock cycles and deactivated (1) during the remaining 11_094 clock cycles.

Lab. 7 – Table-based Implementation

Derive a constant two-dimensional array, specifying the counters' limits.

The first index refers to the brightness level.

The second index refers to the refresh rate.

```
type LUTable is array (0 to 7, 0 to 7) of integer range 0 to 2_000_000;
constant BRIGHTNESS_LUT : LUTable :=
(   (0, 0, 0, 0, 0, 0, 0, 0),
    (280_000, 140_000, …),
    (580_000, 290_000, 145_000, 72_500, 36_250, 18_125, 9_063, 4_531),
     …
    (2_000_000, 1_000_000, 500_000, 250_000, 125_000, 62_500, 31_250, 15_625));
```

Example of use:

```
s_brLimit <= BRIGHTNESS_LUT(to_integer(unsigned(brCtrl)),
                            to_integer(unsigned(refrRate))); -- for duty-cycle
BRIGHTNESS_LUT(7, to_integer(unsigned(refrRate)) - 1) -- for generating enable
```

AXI4-Stream

For high-speed streaming data in point-to-point communications.

AXI4-Stream removes the requirement for an address phase altogether and allows unlimited data burst size.

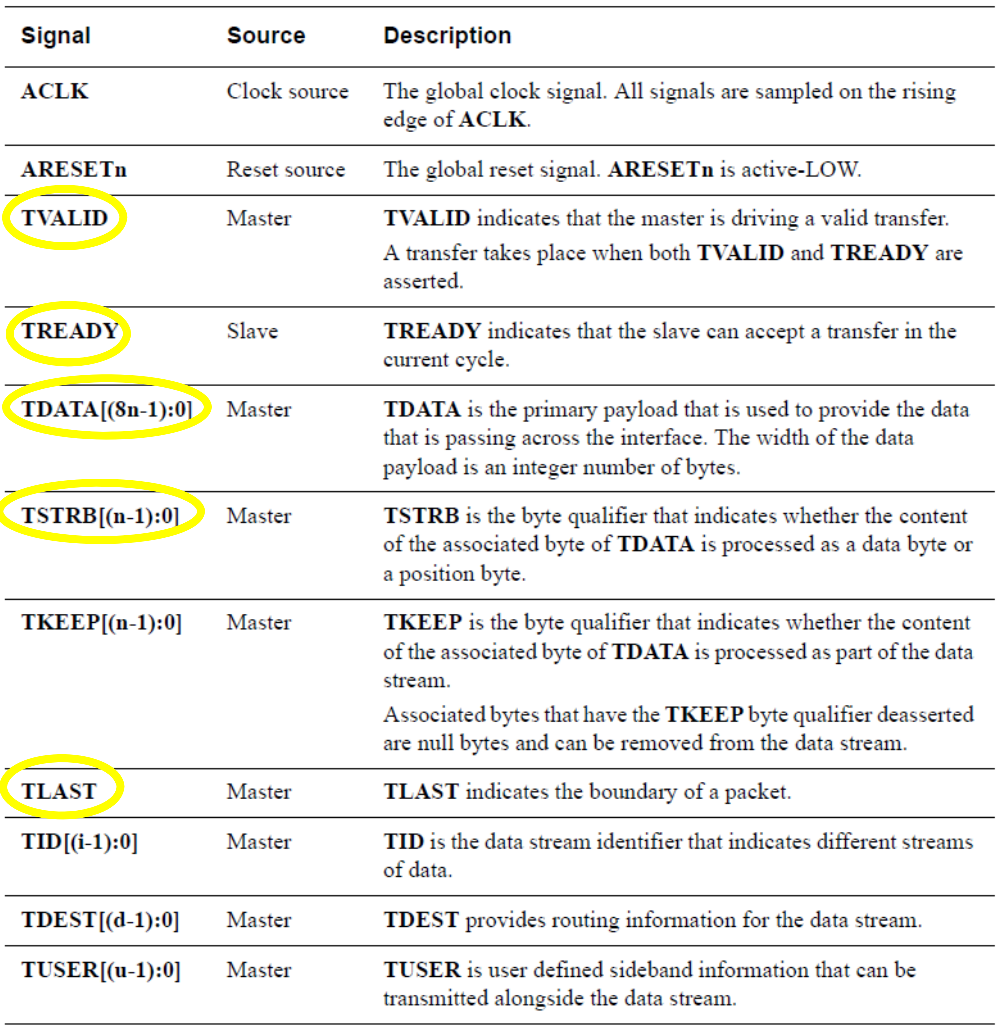
AXI4-Stream interfaces and transfers do not have address phases and are therefore not considered to be memory-mapped.

The AXI4-Stream protocol defines a single unidirectional channel for transmission of streaming data (with a handshaking data flow).

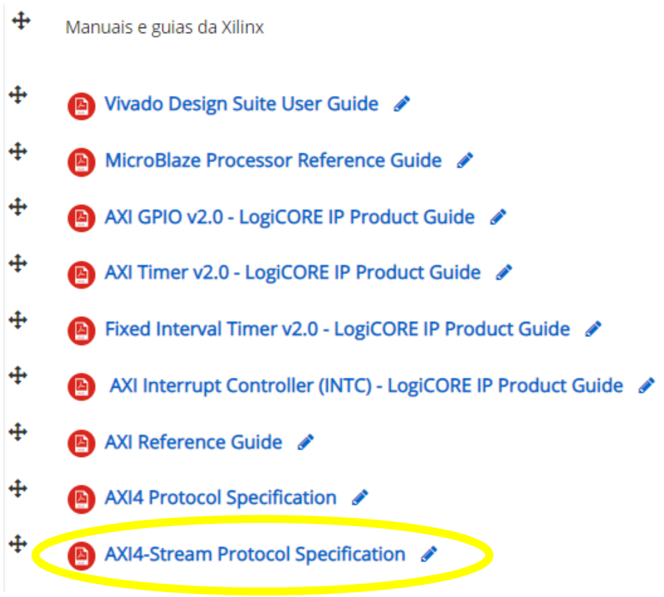
The AXI4-Stream channel models the write data channel of AXI4.

Use the AXI4-Stream protocol for applications that typically focus on a data-centric and data-flow paradigm where the concept of an address is not present or not required.

AXI4-Stream Interface Signals



Signal	Source	Description
ACLK	Clock source	The global clock signal. All signals are sampled on the rising edge of **ACLK**.
ARESETn	Reset source	The global reset signal. **ARESETn** is active-LOW.
TVALID	Master	**TVALID** indicates that the master is driving a valid transfer. A transfer takes place when both **TVALID** and **TREADY** are asserted.
TREADY	Slave	**TREADY** indicates that the slave can accept a transfer in the current cycle.
TDATA[(8n-1):0]	Master	**TDATA** is the primary payload that is used to provide the data that is passing across the interface. The width of the data payload is an integer number of bytes.
TSTRB[(n-1):0]	Master	**TSTRB** is the byte qualifier that indicates whether the content of the associated byte of **TDATA** is processed as a data byte or a position byte.
TKEEP[(n-1):0]	Master	**TKEEP** is the byte qualifier that indicates whether the content of the associated byte of **TDATA** is processed as part of the data stream. Associated bytes that have the **TKEEP** byte qualifier deasserted are null bytes and can be removed from the data stream.
TLAST	Master	**TLAST** indicates the boundary of a packet.
TID[(i-1):0]	Master	**TID** is the data stream identifier that indicates different streams of data.
TDEST[(d-1):0]	Master	**TDEST** provides routing information for the data stream.
TUSER[(u-1):0]	Master	**TUSER** is user defined sideband information that can be transmitted alongside the data stream.



Manuais e guias da Xilinx

- Vivado Design Suite User Guide
- MicroBlaze Processor Reference Guide
- AXI GPIO v2.0 - LogiCORE IP Product Guide
- AXI Timer v2.0 - LogiCORE IP Product Guide
- Fixed Interval Timer v2.0 - LogiCORE IP Product Guide
- AXI Interrupt Controller (INTC) - LogiCORE IP Product Guide
- AXI Reference Guide
- AXI4 Protocol Specification
- AXI4-Stream Protocol Specification

AXI4-Stream Handshake Process

The **TVALID** and **TREADY** handshake determines when information is passed across the interface.

A two-way flow control mechanism enables both the master and slave to control the rate at which the data and control information is transmitted across the interface.

For a transfer to occur both the **TVALID** and **TREADY** signals must be asserted.

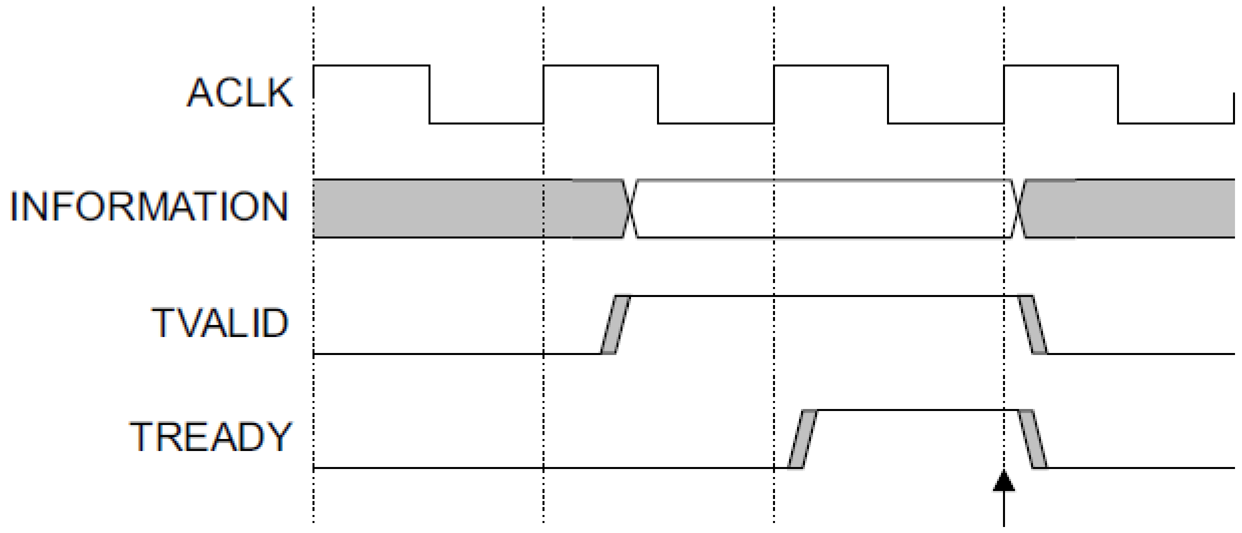
Either **TVALID** or **TREADY** can be asserted first or both can be asserted in the same **ACLK** cycle.

A master is not permitted to wait until **TREADY** is asserted before asserting **TVALID**. Once **TVALID** is asserted it must remain asserted until the handshake occurs.

A slave is permitted to wait for **TVALID** to be asserted before asserting the corresponding **TREADY**. If a slave asserts **TREADY**, it is permitted to deassert **TREADY** before **TVALID** is asserted.

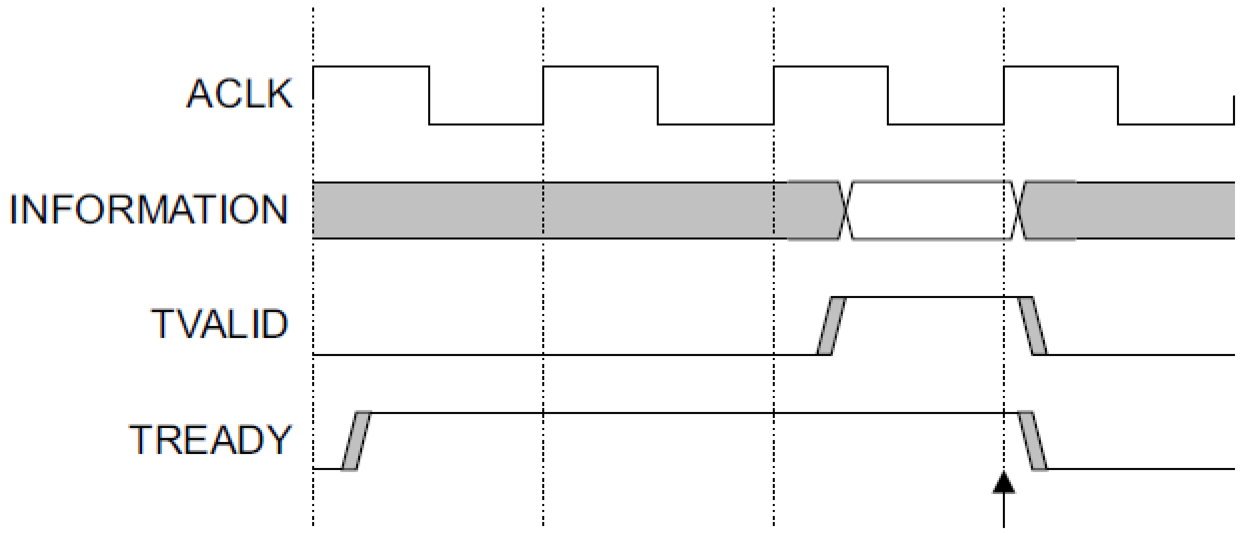
TVALID Before TREADY Handshake

The master presents the data and control information and asserts the TVALID signal HIGH. Once the master has asserted **TVALID**, the data or control information from the master must remain unchanged until the slave drives the **TREADY** signal HIGH, indicating that it can accept the data and control information. In this case, transfer takes place once the slave asserts **TREADY** HIGH. The arrow shows when the transfer occurs.



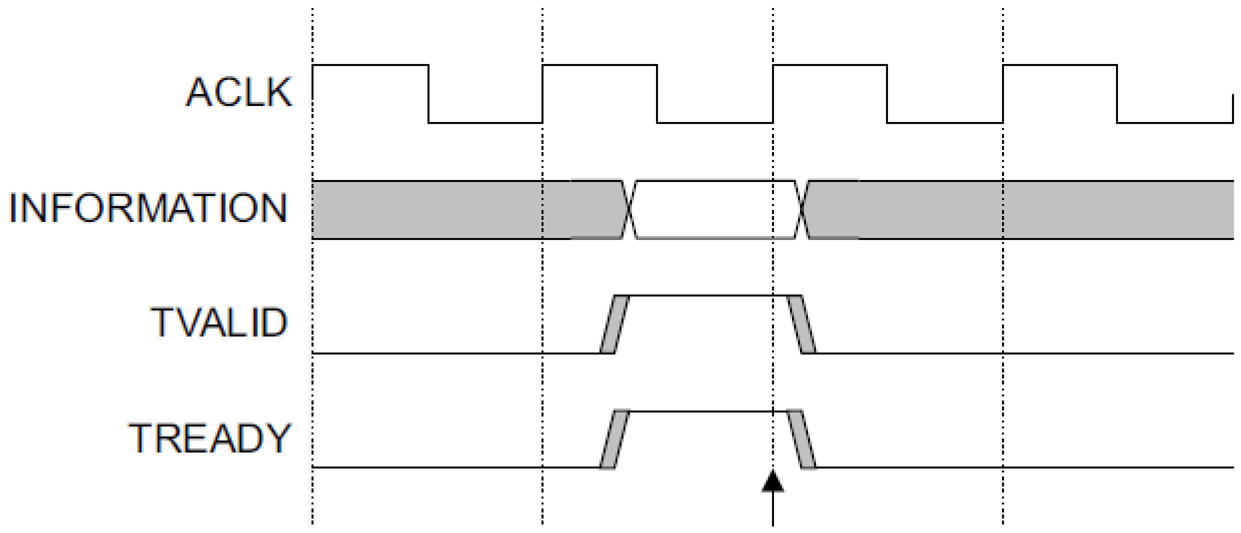
TREADY Before TVALID Handshake

The slave drives TREADY HIGH before the data and control information is valid. This indicates that the destination can accept the data and control information in a single cycle of ACLK. In this case, transfer takes place once the master asserts TVALID HIGH. The arrow shows when the transfer occurs.



TVALID With TREADY Handshake

The master asserts TVALID HIGH and the slave asserts TREADY HIGH in the same cycle of ACLK. In this case, transfer takes place in the same cycle as shown by the arrow.



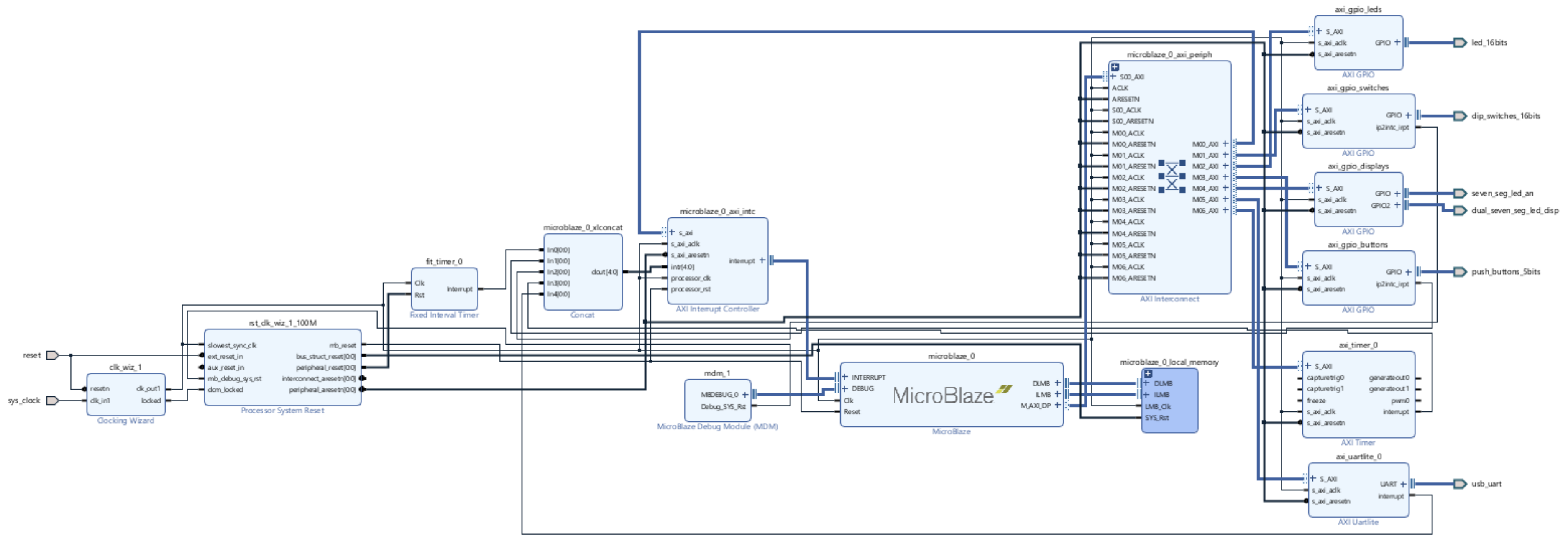
Examples

Reverse endianness

- **Endianness** is the order of bytes in a word of digital data.
- Endianness is primarily expressed as **big-endian** or **little-endian**.
- A big-endian system stores the most significant byte of a word at the smallest memory address and the least significant byte at the largest.
- A little-endian system, in contrast, stores the least-significant byte at the smallest address.
- 0xAB347801 => 0x017834AB

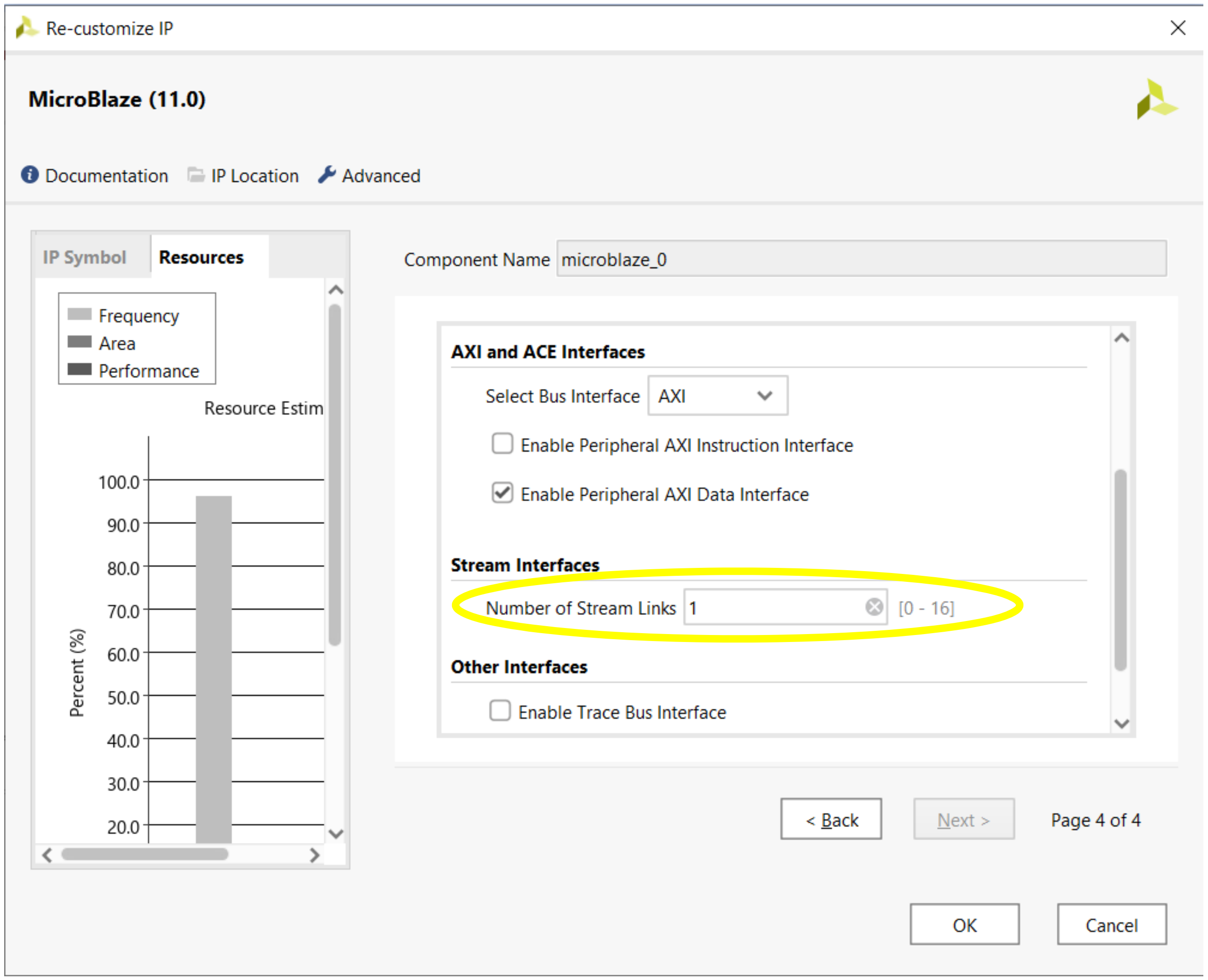
Population count (Hamming weight)

Example 1 – Starting Point



Universidade de Aveiro

Adding Stream Links to MicroBlaze



Re-customize IP

MicroBlaze (11.0)

Documentation | IP Location | Advanced

Component Name microblaze_0

AXI and ACE Interfaces

Select Bus Interface AXI

☐ Enable Peripheral AXI Instruction Interface

☑ Enable Peripheral AXI Data Interface

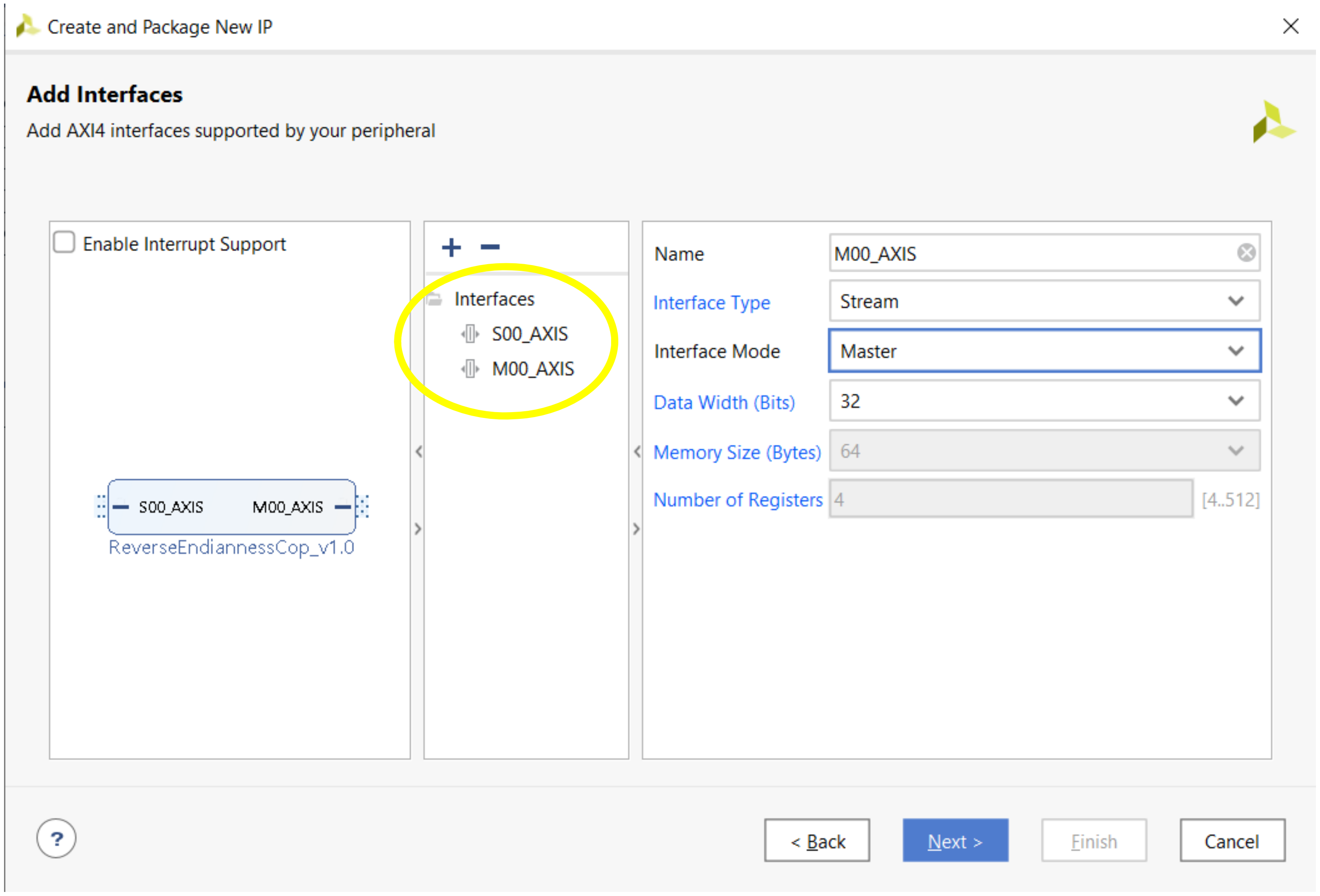
Stream Interfaces

Number of Stream Links 1 [0 - 16]

Other Interfaces

☐ Enable Trace Bus Interface

Create & Package New IP



Create and Package New IP

Add Interfaces

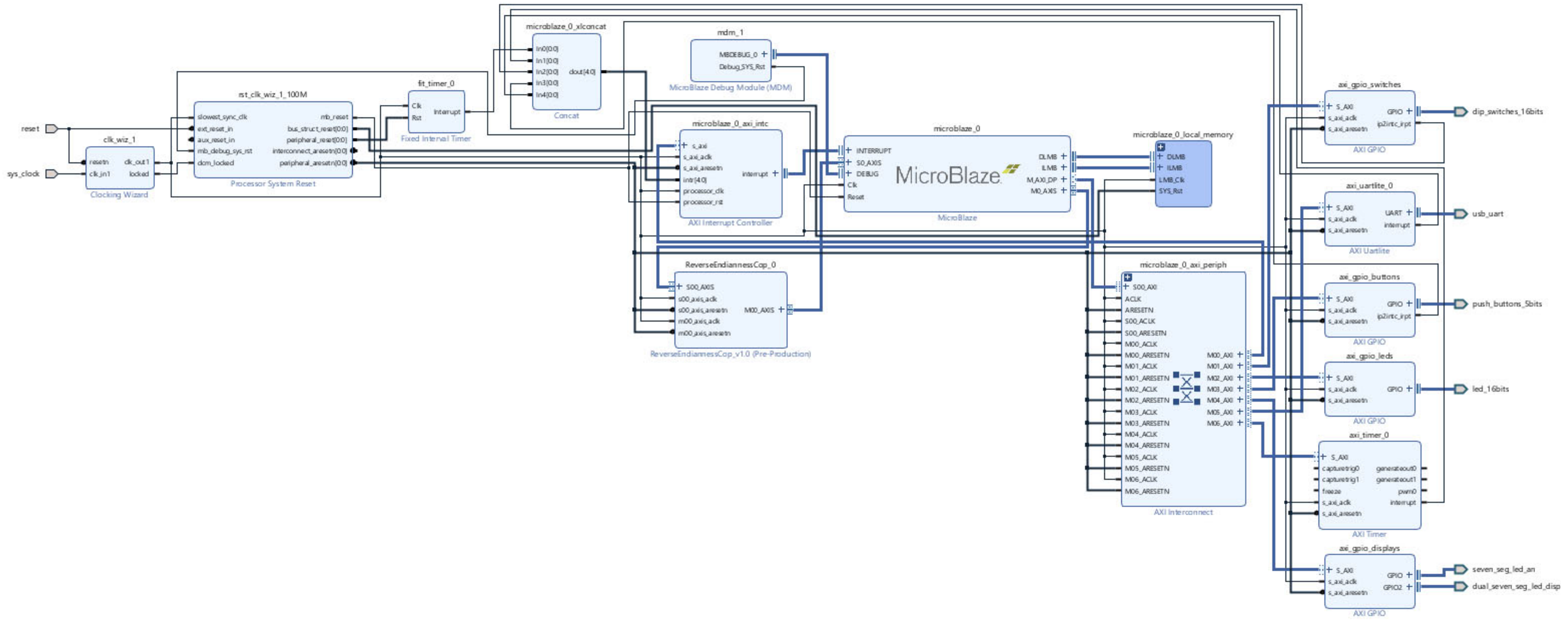
Add AXI4 interfaces supported by your peripheral

Enable Interrupt Support

Interfaces
- S00_AXIS
- M00_AXIS

Name	M00_AXIS
Interface Type	Stream
Interface Mode	Master
Data Width (Bits)	32
Memory Size (Bytes)	64
Number of Registers	4 [4..512]

Example 1 Block Design



Example 1 – Reverse Endianness

Import Block Design

Configure the MicroBlaze to have one pair of stream links

Create and Package new IP (*ReverseEndiannessCop*)

Add IP

Edit in IP Packager (the code is given on eLearning)

Generate output products

Create HDL Wrapper

Generate Bitstream

Export Hardware

Launch Vitis

Vitis

The C code is given on eLearning

There is no need to correct the IP makefiles

The code processes an array of N (N=4000) integers (32b = 4B)

The code uses the AXI timer to measure time

The code does not work. Why?

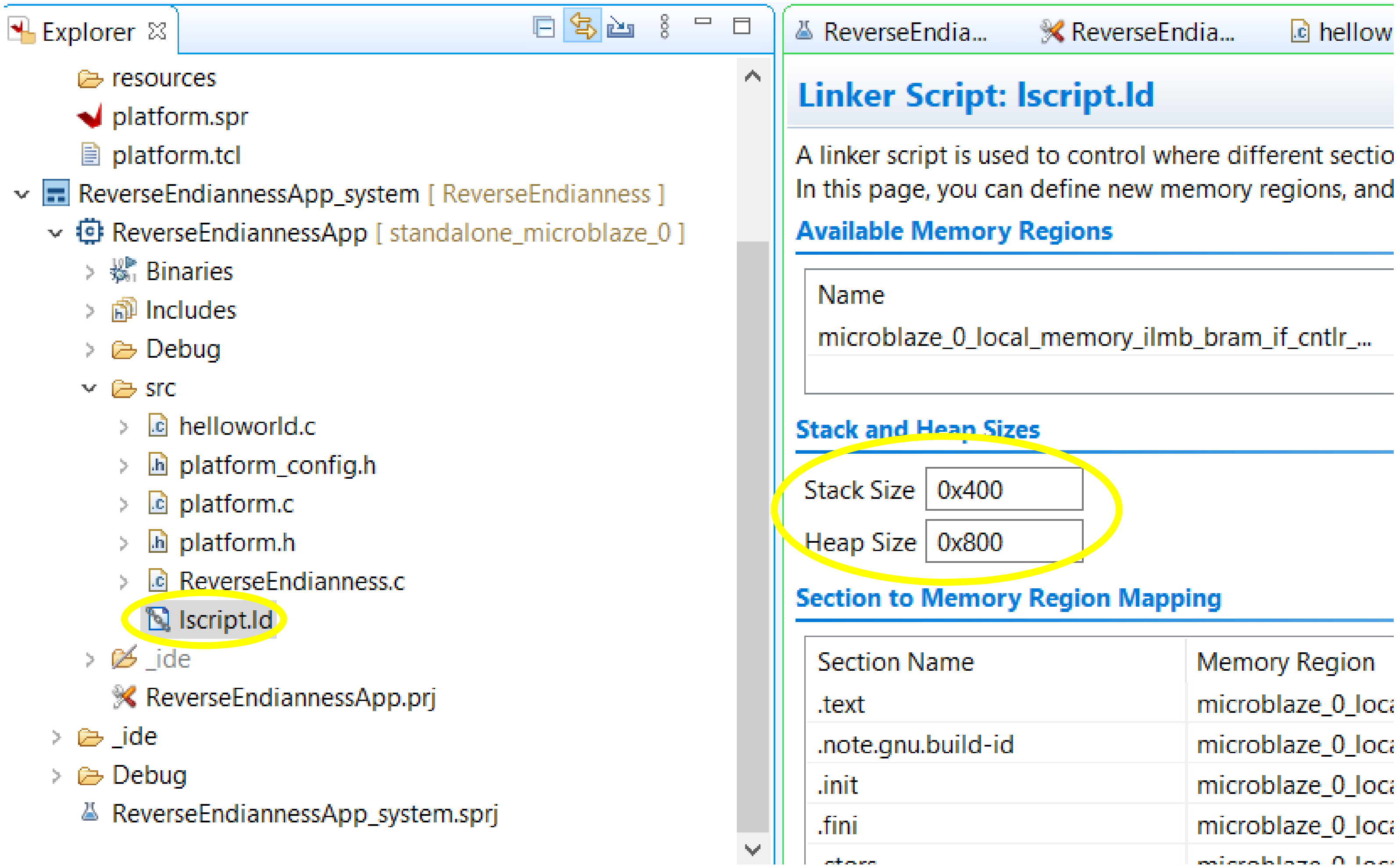
```
#define N 4000

int srcData[N], dstData[N];
```

What is the size of data?

Where do these data reside?

Vitis – Changing the Stack Size


Explorer

- resources
- platform.spr
- platform.tcl
- ReverseEndiannessApp_system [ReverseEndianness]
  - ReverseEndiannessApp [standalone_microblaze_0]
    - Binaries
    - Includes
    - Debug
    - src
      - helloworld.c
      - platform_config.h
      - platform.c
      - platform.h
      - ReverseEndianness.c
      - lscript.ld
    - _ide
    - ReverseEndiannessApp.prj
  - _ide
  - Debug
  - ReverseEndiannessApp_system.sprj

ReverseEndia... ReverseEndia... hellow

Linker Script: lscript.ld

A linker script is used to control where different sectio
In this page, you can define new memory regions, and

Available Memory Regions

Name
microblaze_0_local_memory_ilmb_bram_if_cntlr_...

Stack and Heap Sizes

Stack Size	0x400
Heap Size	0x800

Section to Memory Region Mapping

Section Name	Memory Region
.text	microblaze_0_loca
.note.gnu.build-id	microblaze_0_loca
.init	microblaze_0_loca
.fini	microblaze_0_loca

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

To do:

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