Projetando um periférico AXI-Stream personalizado

AULA 10

IOUL IIA SKL I AROVA

Fluxo AXI4

Para streaming de dados de alta velocidade em comunicações ponto a ponto.

O AXI4-Stream elimina completamente a necessidade de uma fase de endereço e permite tamanho ilimitado de rajadas de dados.

As interfaces e transferências AXI4-Stream não possuem fases de endereço e, portanto, sãonão considerado mapeado na memória.

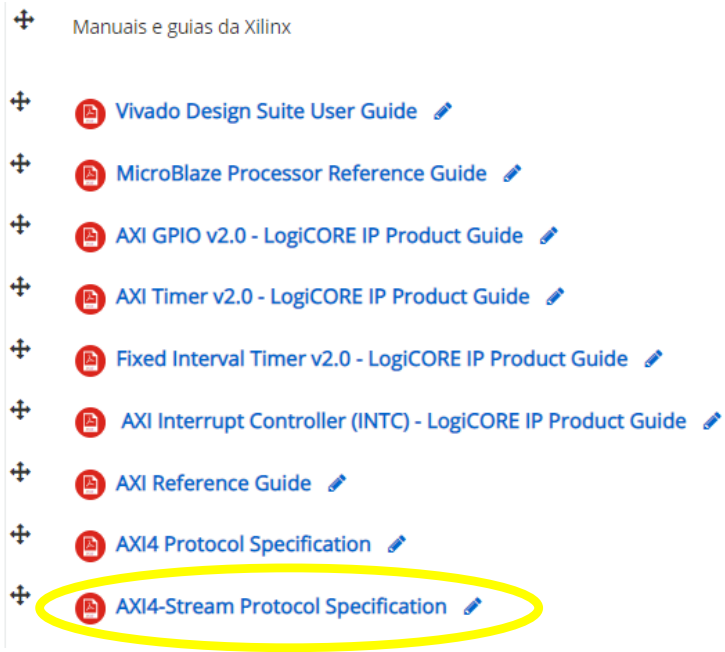
O protocolo AXI4-Stream define um único canal unidirecional para transmissão de dados de streaming (com fluxo de dados de handshake).

O canal AXI4-Stream modela o canal de gravação de dados do AXI4.

Use o protocolo AXI4-Stream para aplicativos que normalmente se concentram em um paradigma centrado em dados e de fluxo de dados, onde o conceito de endereço não está presente ou não é necessário.

Sinais de interface AXI4-Stream

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.



Processo de handshake de fluxo AXI4

O**TVALID**e**PRONTO**handshake determina quando as informações são passadas pela interface.

Um mecanismo de controle de fluxo bidirecional permite que tanto o mestre quanto o escravo controlem a taxa na qual os dados e as informações de controle são transmitidos através da interface.

Para que ocorra uma transferência, tanto o**TVALID**e**PRONTO**sinais devem ser afirmados.

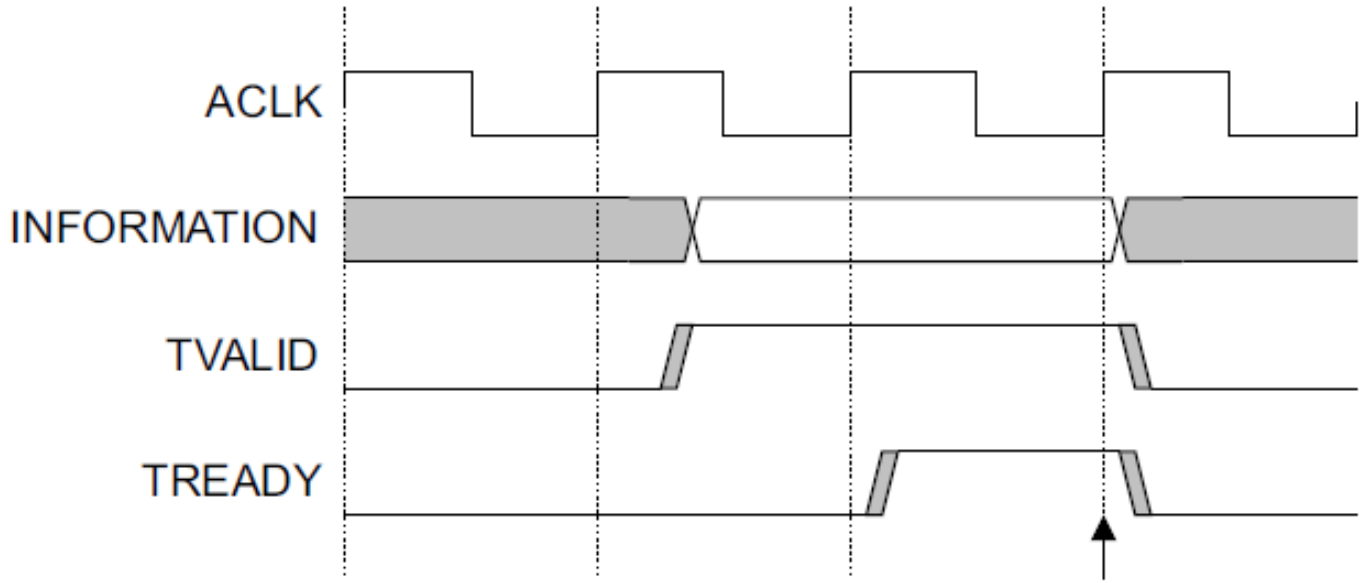
Qualquer**TVALID**ou**PRONTO**podem ser afirmados primeiro ou ambos podem ser afirmados no mesmo**ACLK**ciclo.

Um mestre não está autorizado a esperar até**PRONTO**é afirmado antes de afirmar **TVALID**. Uma vez**TVALID**é afirmado, ele deve permanecer afirmado até que o handshake ocorra.

Um escravo tem permissão para esperar**TVALID**ser afirmado antes de afirmar o correspondente**PRONTO**. Se um escravo afirma**PRONTO**, é permitido desassetar **PRONTO**antes**TVALID**é afirmado.

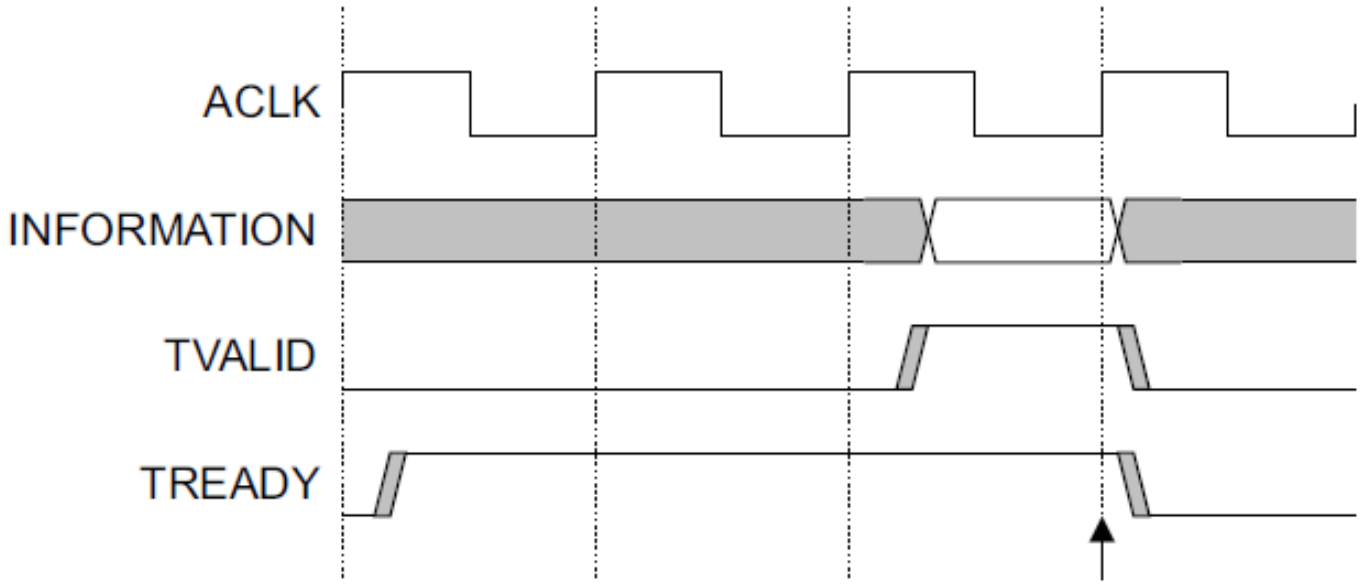
TVALID antes do aperto de mão TREADY

O mestre apresenta os dados e informações de controle e ativa o sinal TVALID HIGH. Uma vez que o mestre tenha afirmado**TVALID**, os dados ou informações de controle do mestre devem permanecer inalterados até que o escravo acione o**PRONTO**sinal ALTO, indicando que pode aceitar os dados e controlar informações. Neste caso, a transferência ocorre quando o escravo afirma **PRONTO**ALTO. A seta mostra quando a transferência ocorre.



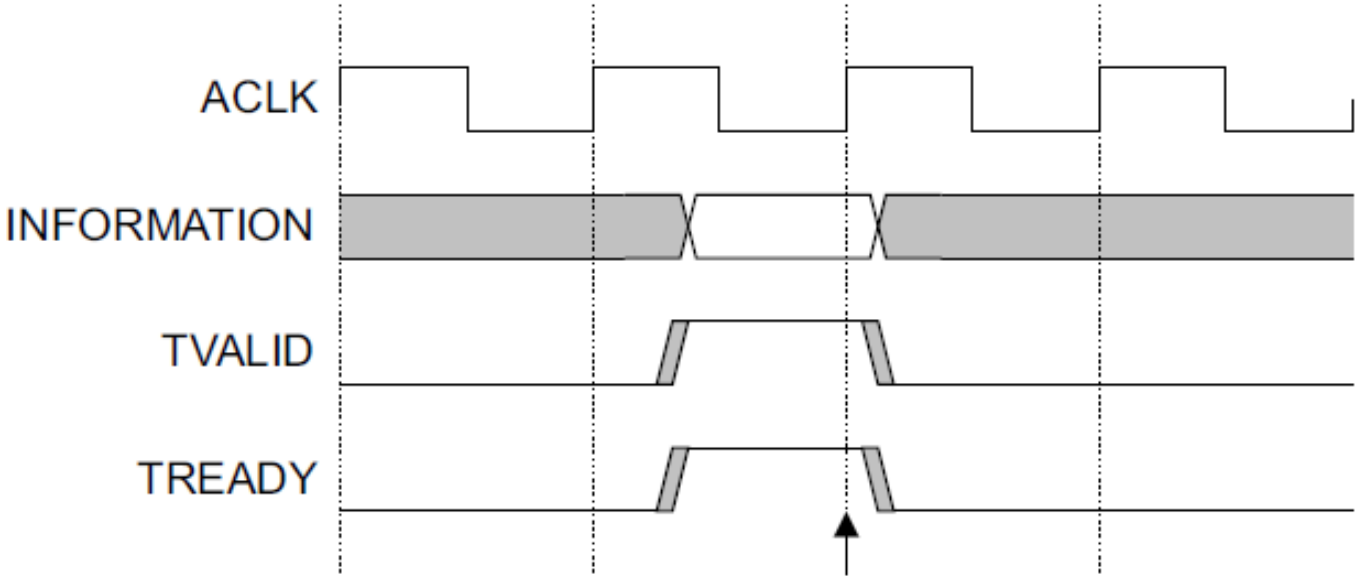
TREADY antes do aperto de mão TVALID

O escravo aciona TREADY HIGH antes que os dados e as informações de controle sejam válidos. Isto indica que o destino pode aceitar os dados e controlar as informações em um único ciclo de ACLK. Neste caso, a transferência ocorre quando o mestre afirma TVALID HIGH. A seta mostra quando a transferência ocorre.



TVALID com aperto de mão TREADY

O mestre afirma TVALID HIGH e o escravo afirma TREADY HIGH no mesmo ciclo de ACLK. Neste caso, a transferência ocorre no mesmo ciclo mostrado pela seta.



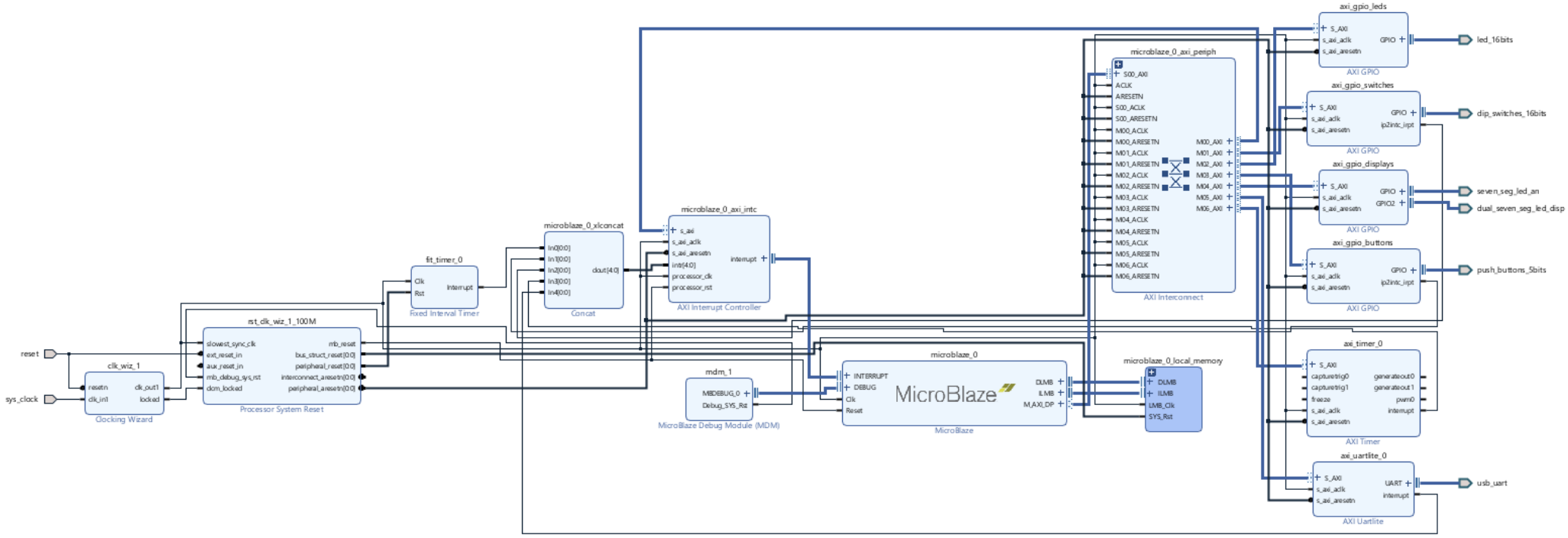
Exemplos

Endianismo reverso

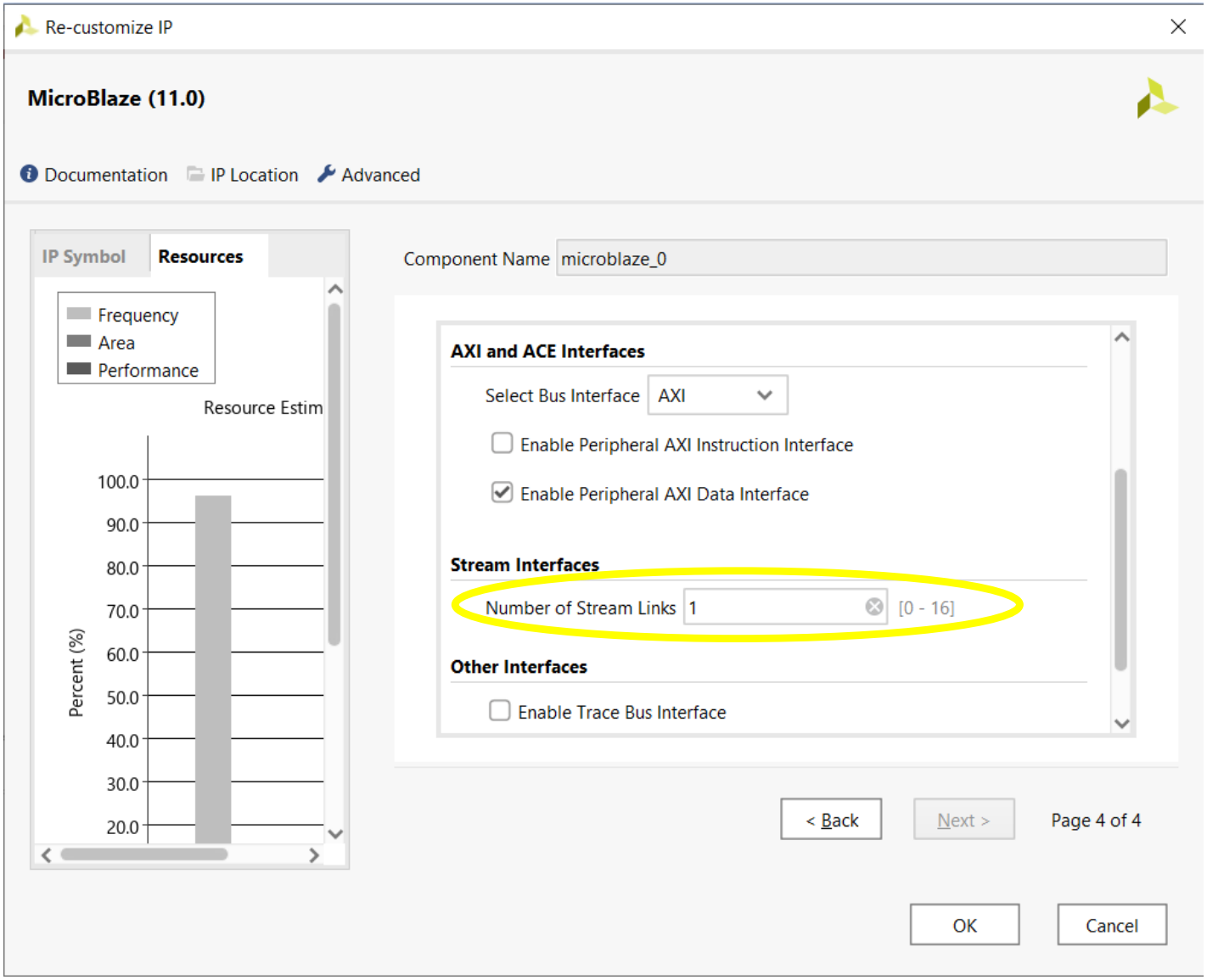
- **Endianismo**é a ordem dos bytes em uma palavra de dados digitais.
- Endianness é expresso principalmente como**big endian**ou**pequeno endian**.
- Um sistema big-endian armazena o byte mais significativo de uma palavra no menor endereço de memória e o byte menos significativo no maior.
- Um sistema little-endian, por outro lado, armazena o byte menos significativo no menor endereço.
- 0xAB347801=> 0x017834AB

Contagem populacional (peso de Hamming)

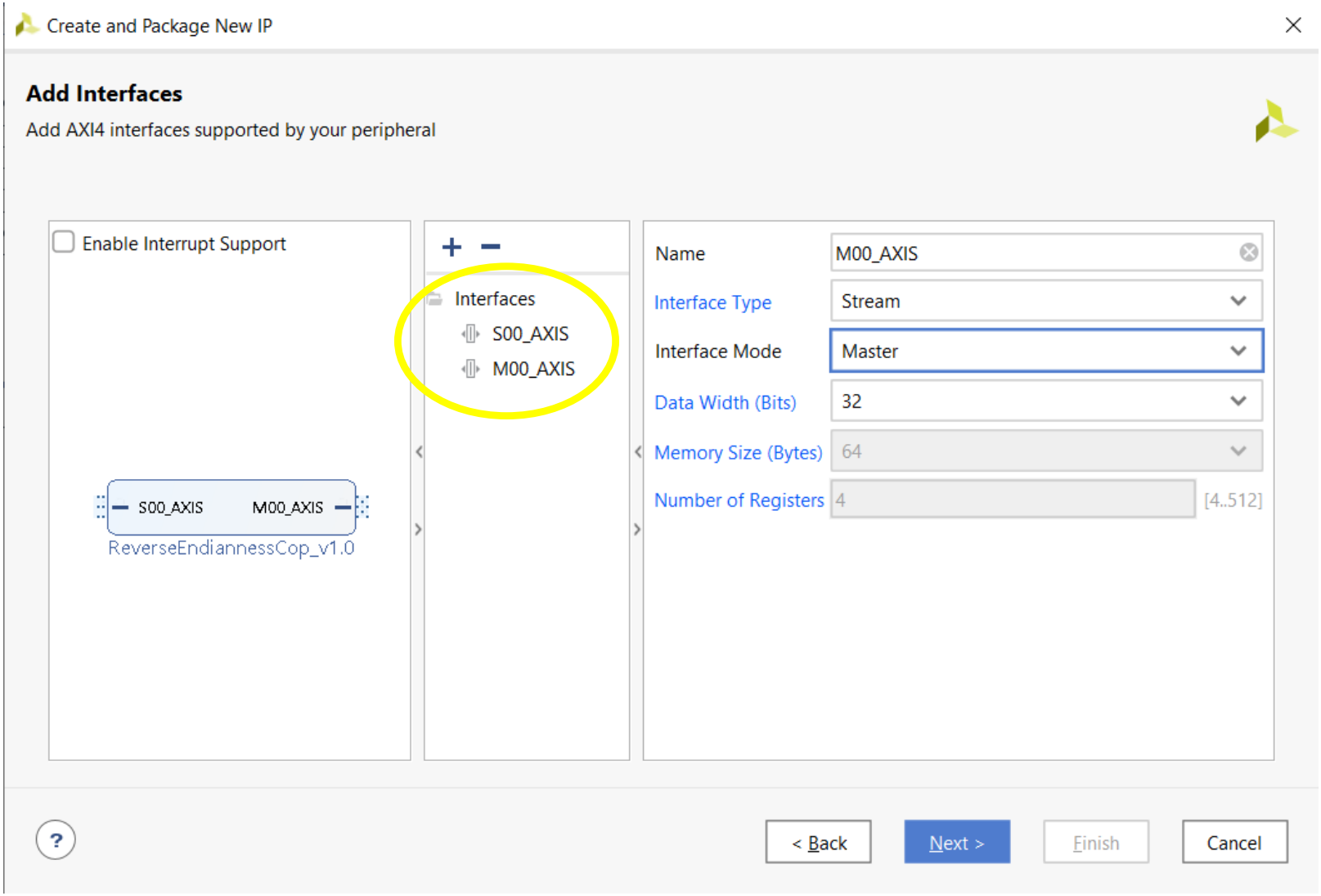
Exemplo 1 – Ponto de Partida



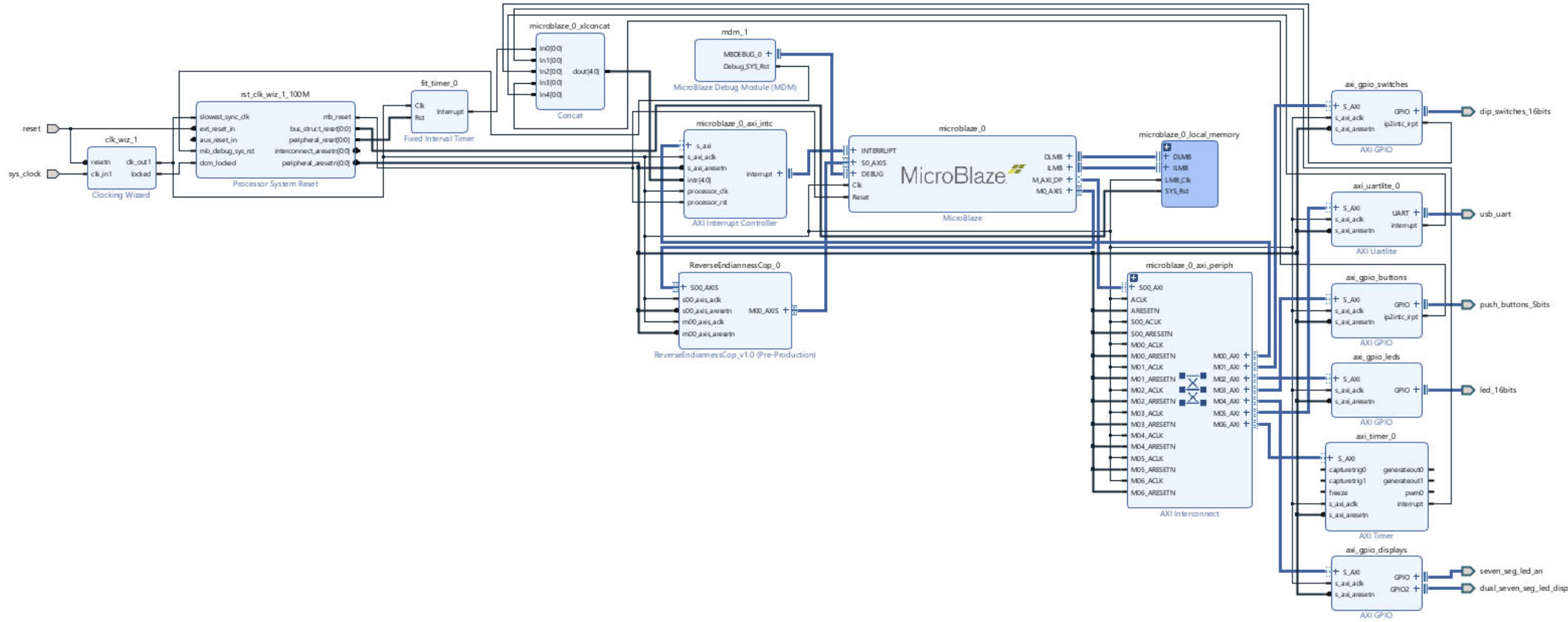
Adicionando links de stream ao MicroBlaze



Criar e empacotar novo IP



Exemplo 1 Design de Bloco



Exemplo 1 – Endianismo reverso

Importar design de bloco

Configure o MicroBlaze para que um par de links de fluxo crie e

empacote um novo IP (*ReverseEndiannessCop*) Adicionar IP

Edite no IP Packager (o código é fornecido no eLearning)

Gere produtos de saída

Criar wrapper HDL

Gerar fluxo de bits

- set_property CONFIG_VOLTAGE 3.3 [get_designs synth_1]
- set_property CFGBVS VCCO [get_designs synth_1]

Exportar hardware

Lançar Vitis

Vitis

O código C é fornecido no eLearning

Não há necessidade de corrigir os makefiles de IP

O código processa uma matriz de N (N=4000) inteiros (32b = 4B) O código

usa o temporizador AXI para medir o tempo

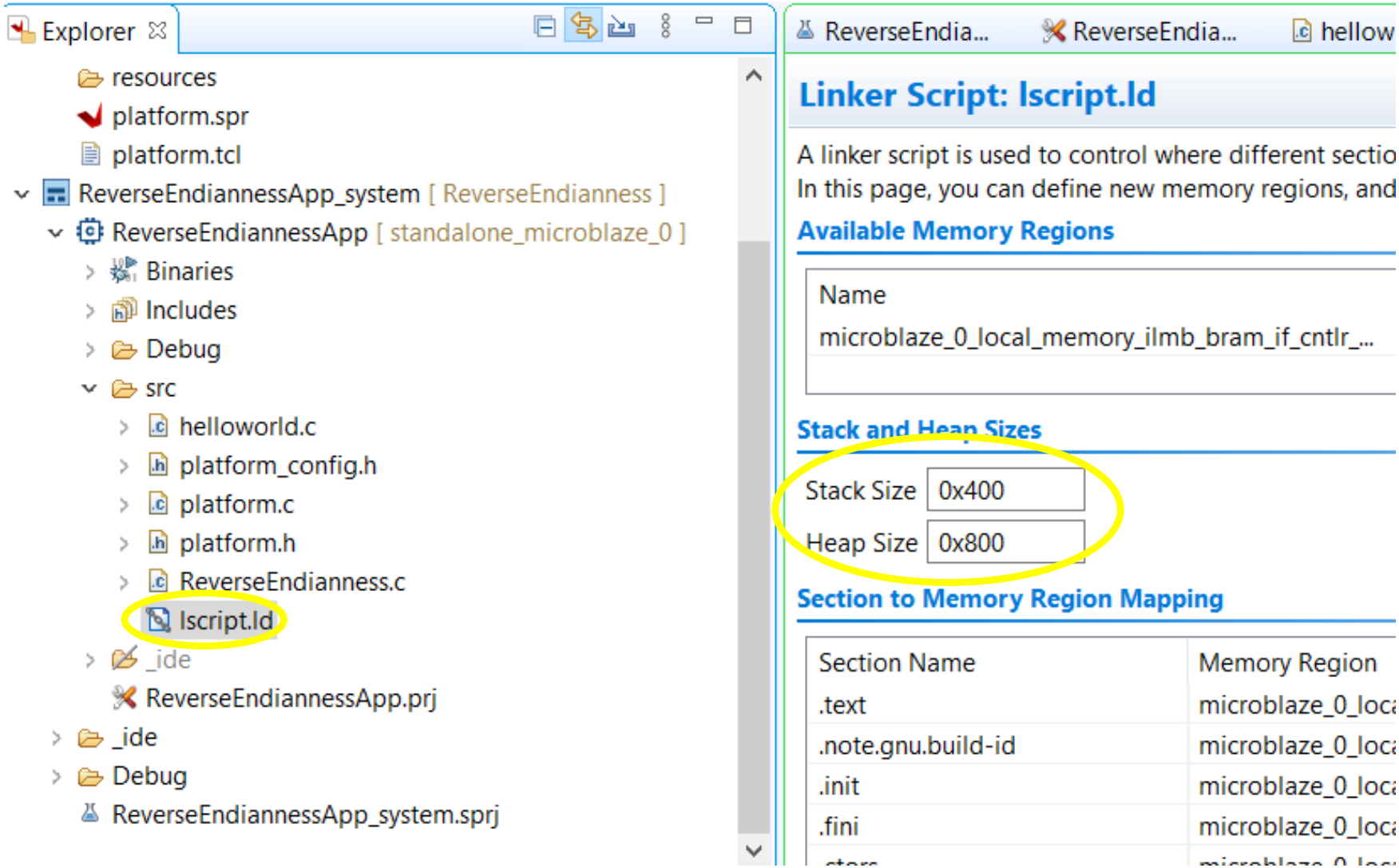
O código não funciona. Por que?

```
# definir N 4000
int srcData[N], dstData[N];
```

Qual é o tamanho dos dados?

Onde esses dados residem?

Vitis – Alterando o tamanho da pilha



Exemplos (coprocessador AXI-Stream)

Endianismo reverso

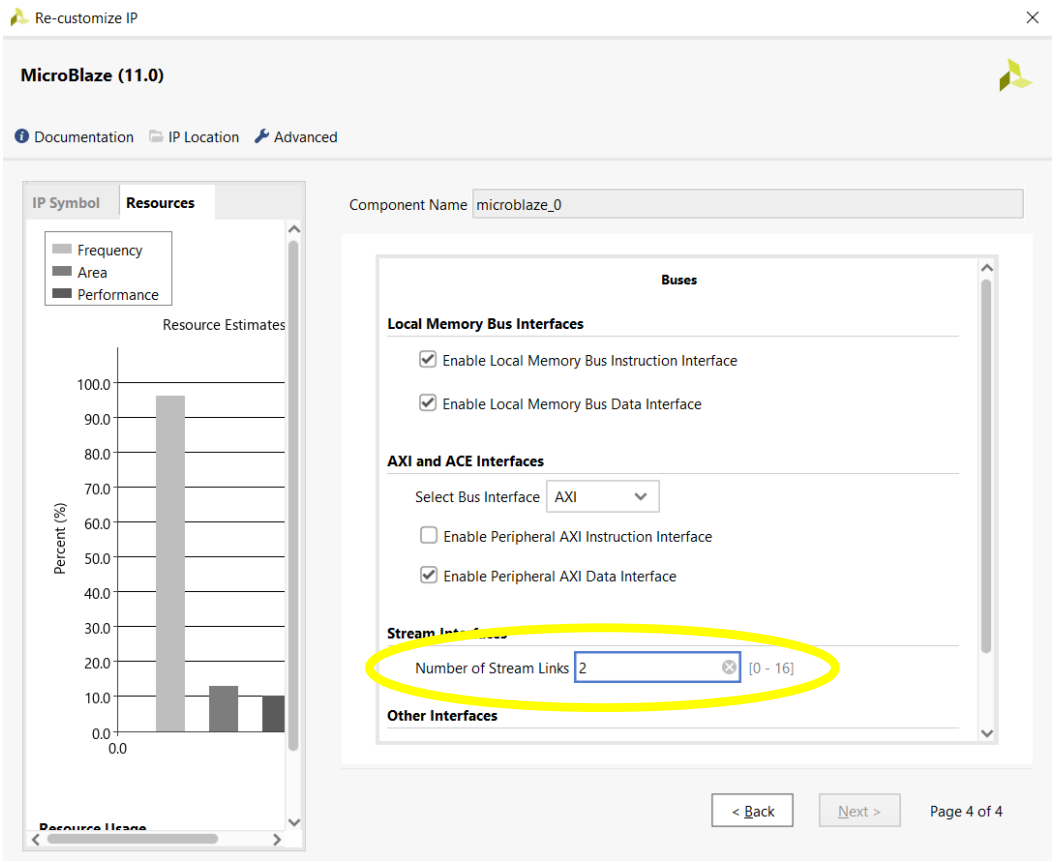
Contagem populacional (peso de Hamming)

- o número de entradas diferentes de zero ('1' bits) em uma palavra de dados.
- 0xAB347801 =>10101011_00110100_01111000_00000001 => 13

Exemplo 2 – Ponto de Partida

Continue trabalhando no mesmo projeto

Altere o número de links de stream no MicroBlaze para 2



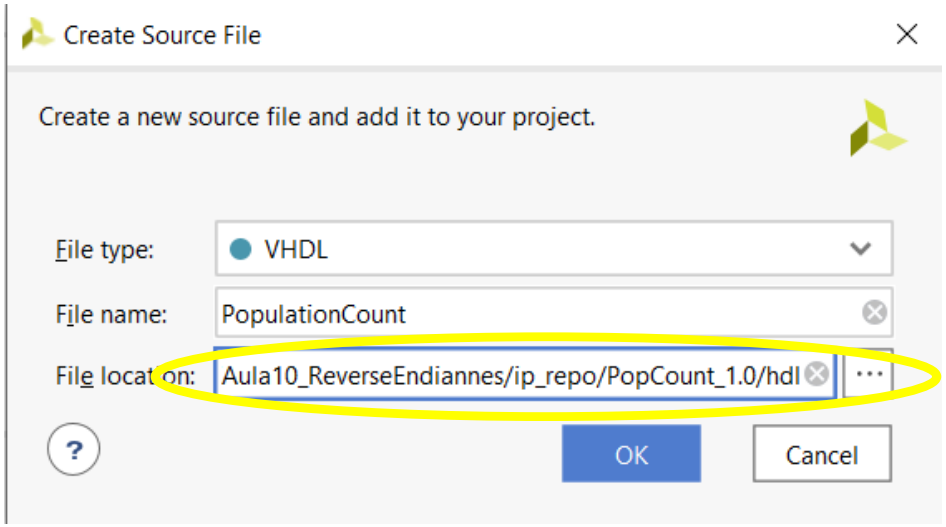
Exemplo 2 – Adicionar novo IP

Criar e empacotar novo IP -**PopCount** Editar

no empacotador IP

Altere os três arquivos “padrão” como no Exemplo 1

Crie uma nova fonte – PopulationCount.vhd (o código é fornecido no eLearning)



Instancie o módulo PopulationCount na interface de fluxo escravo:
cálculo: Contagem Populacional

mapa genérico(N => C_S_AXIS_TDATA_WIDTH)
mapa do porto(dadosIn => ...);

Para loop Declaração VHDL

A**para loop**instrução é uma instrução sequencial que pode ser usada dentro de um processo.

A**para loop**inclui uma especificação de quantas vezes o corpo do loop deve ser executado:

[*laço*_rótulo:]
paraidentificador**em**intervalo_discreto**laço {**
instrução_sequencial**} ciclo final**[*laço*_rótulo];

O**para loop**A instrução é usada sempre que uma operação precisa ser repetida.

O circuito é**desenrolado**estaticamente – o número de iterações do loop deve ser conhecido em tempo de compilação.

Desenrolamento de loopé um método sistemático de alcançar paralelismo que pode ser automatizado.

Isso tem o custo de uma pegada de tecido maior (mais área de FPGA).

Variáveis

Loops For são frequentemente usados com**variáveis**.

Variáveissão declarados na parte de declaração dos processos:

declaração_variável⇐
variávelidentificador { , ... } : subtype_indication [:= expressão] ;

A sintaxe de um**atribuição de variável**afirmação é dada pela regra

declaração_de_atribuição_variável⇐
[rótulo:] nome: = expressão;

Uma atribuição de variável**substitui imediatamente a variável**com um novo valor (uma atribuição de sinal, por outro lado, programa um novo valor para ser aplicado a um sinal em algum momento posterior).

Contagem de população com um loop For

```
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;
```

Uma longa sequência de 31 somadores será gerada.

Para loopcontra**Para gerar**

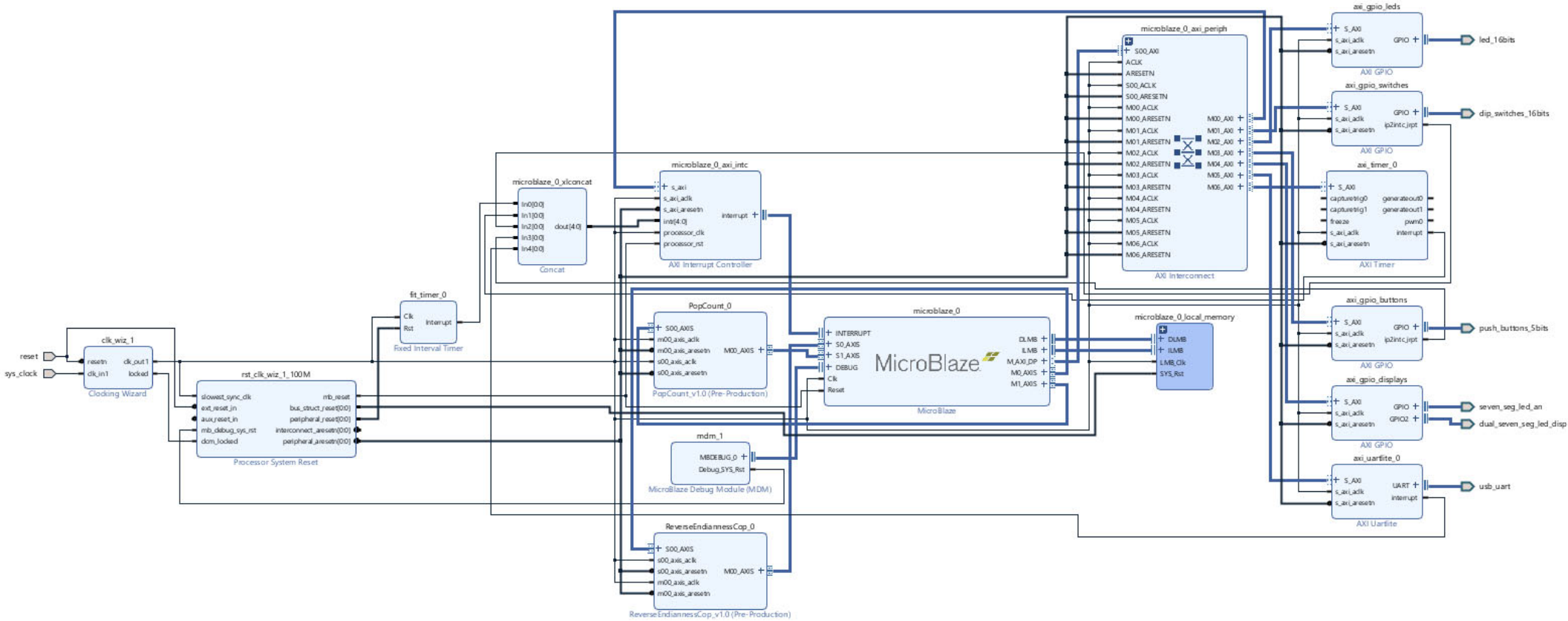
O**para laços**são**declarações sequenciais**,**contendo instruções sequenciais**(ou seja, cada iteração é sequenciada para ser executada após a anterior).

O**loops para gerar**são**simultâneo**declarações,**contendo declarações simultâneas**, e é assim que você pode usá-lo para criar várias instâncias de um componente.

Loops para gerarsão usados ao especificar a estrutura de hardware exata.

Para loopssão mais adequados para descrições comportamentais.

Exemplo 2 Design de Bloco



Exemplo 2 – Etapas Adicionais

Gerar produtos de saída

Criar HDL Wrapper

Gerar fluxo de bits

Exportar hardware

Lançar Vitis

Vitis

Escreva o código C (com base no exemplo ReverseEndianness) Não há necessidade de corrigir os makefiles IP

Configure o tamanho de pilha correto

Considerações finais

Ao final desta palestra você deverá ser capaz de:

- Projete módulos de hardware personalizados interagindo com o MicroBlaze por meio da interface AXI-Stream
- Escreva programas C que fazem uso de hardware personalizado conectado por fluxo

Pendência:

- Construa as plataformas de hardware consideradas
- Teste o aplicativo fornecido no Vitis
- Laboratório completo. 8