



THE IMAGINATION UNIVERSITY PROGRAMME

RVfpga-SoC Lab1

Introdução ao RVfpga-SoC

Tabela 1. Termos do RVfpga

Nome	Descrição
Cursos	
RVfpga	Um curso que mostra como usar o RVfpgaNexys e o RVfpgaSim, RISC-V system-on-chips (SoCs), para executar programas e ampliar o sistema adicionando periféricos (RVfpga Labs 1-10) e explorar o núcleo e o sistema de memória executando simulações, medindo o desempenho, adicionando instruções e modificando o sistema de memória (RVfpga Labs 11-20). Ao longo do curso, os utilizadores aprenderão a usar a toolchain RISC-V (compiladores e depuradores) e simuladores, o simulador Verilator HDL e o simulador de conjunto de instruções Whisper (ISS) da Western Digital.
RVfpga-SoC	Um curso que mostra como construir um SweRVolfX SoC a partir do zero usando blocos IP como o núcleo SweRV, memórias e periféricos. O curso também mostra como carregar um sistema operativo de tempo real (RTOS) Zephyr no SweRVolf e executar programas, incluindo o exemplo Hello-World do Tensorflow Lite, sobre o sistema operativo.
Núcleos e SoCs	
SweRV EH1 Core	Núcleo RISC-V comercial de código aberto desenvolvido pela Western Digital (https://github.com/chipsalliance/Cores-SweRV).
SweRV EH1 Core Complex	Núcleo SweRV EH1 com memória adicional (ICCM, DCCM e cache de instruções), controlador de interrupção programável (PIC), interfaces de barramento e unidade de depuração (https://github.com/chipsalliance/Cores-SweRV).
SweRVolfX	É o System-on-a-Chip usado no curso RVfpga. É uma extensão do SweRVolf. SweRVolf (https://github.com/chipsalliance/Cores-SweRVolf): Um SoC de código aberto criado em torno do SweRV EH1 Core Complex. Adiciona uma boot ROM, uma interface UART, um controlador de sistema, uma interconexão (AXI Interconnect, Wishbone Interconnect, e AXI-to-Wishbone bridge) e um controlador SPI. SweRVolfX : adiciona quatro novos periféricos ao SweRVolf: um GPIO, um PTC, um SPI adicional e um controlador para os 8 mostradores de 7 segmentos de 8 dígitos.
RVfpgaNexys	O SoC SweRVolfX foi realizado para a placa Nexys A7 e seus periféricos. Ele adiciona uma interface DDR2, unidade CDC (clock domain crossing), lógica BSCAN (para a interface JTAG) e gerador de relógio.

	O RVfpgaNexys é o mesmo que o SweRVolf Nexys (https://github.com/chipsalliance/Cores-SweRVolf), exceto que o último é baseado no SweRVolf.
RVfpgaSim	O SoC SweRVolfX tem um encapsulamento de testbench e memória AXI destinados a simulação. O RVfpgaSim é o mesmo que o SweRVolf Sim (https://github.com/chipsalliance/Cores-SweRVolf), exceto pelo fato de que o último é baseado no SweRVolf.

1. Introdução

1. Introdução a um System-on-a-Chip

Neste laboratório, mostraremos como criar um System-on-a-Chip (SoC) RISC-V a partir de blocos IP. Um **SoC** é um circuito integrado ou um CI que integra todo um sistema eletrônico ou computador nele. Um SoC inclui um núcleo e todos os periféricos e interfaces necessários para carregar um sistema operativo e executar programas. A Figura 1 ilustra a organização hierárquica típica de um sistema embebido, começando com o núcleo do processador, depois o SoC construído em torno do núcleo e, finalmente, o sistema e a interface da placa.



Figura 1. Sistema embebido típico

O processo de projeto de um SoC começa com a criação de protótipos em FPGA. O nosso foco é a realização de um SoC numa FPGA.

O CPU RISC-V que usaremos é o **SweRV EH1 Core Complex** da Western Digital, e o SoC que projetaremos neste laboratório será um **SweRVolfX** reduzido, que será implementado na placa **Nexys A7-100T**. A Figura 2 ilustra os vários componentes e como eles se encaixam.

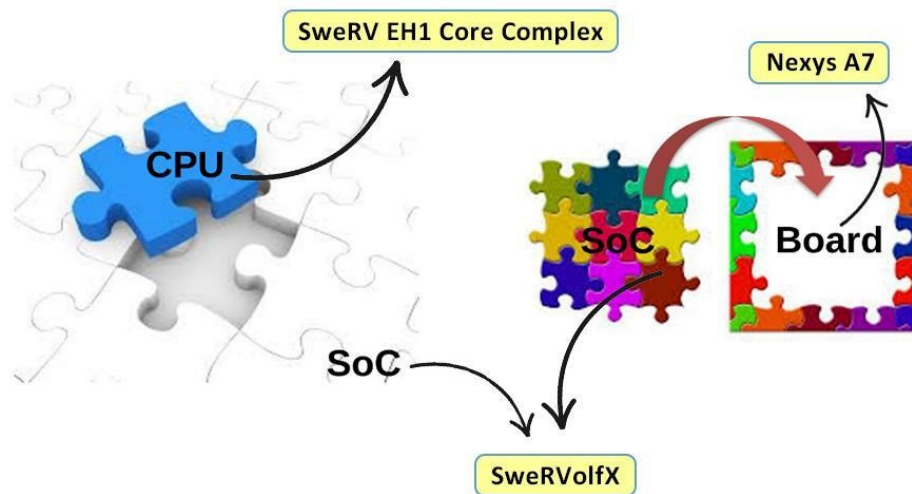


Figura 2. Sistema embebido baseado no sistema RVfpga

2. Introdução ao SweRVolfX e ao sistema RVfpga

Antes de iniciar este laboratório, recomendamos que leia o Guia de Introdução do curso RVfpga e o sistema RVfpga seja entendido como um todo. A seguir, há uma breve descrição do sistema RVfpga apresentado no curso RVfpga.

A Tabela 1 mostra a organização hierárquica do sistema RVfpga, desde o SweRV EH1 Core até ao RVfpgaNexys e ao RVfpgaSim. O System-on-a-Chip (SoC) usado no sistema RVfpga, chamado **SweRVolfX**, ilustrado na Figura 3, é baseado no **SweRVolf** versão 0.7.3 (<https://github.com/chipsalliance/Cores-SweRVolf/releases/tag/v0.7.3>), que é construído sobre o **SweRV EH1 Core Complex**. Além do SweRV EH1 Core Complex, o SoC SweRVolf também inclui uma boot ROM, uma UART, um controlador de sistema e um controlador SPI. O SweRV EH1 Core usa um barramento AXI e os periféricos usam um barramento Wishbone; o SoC também tem uma ponte AXI-Wishbone.

No sistema RVfpga, o SoC SweRVolf é ampliado com mais algumas funcionalidades, como outro controlador SPI (SPI2), um controlador GPIO (General Purpose Input/Output, entrada/saída de uso geral) e um módulo PTC (PWM/Timer/Counter, temporizador/contador). (A Figura 3 mostra esses novos periféricos a vermelho). Esse System-on-a-Chip é chamado **SweRVolfX** (o X significa eXtended).

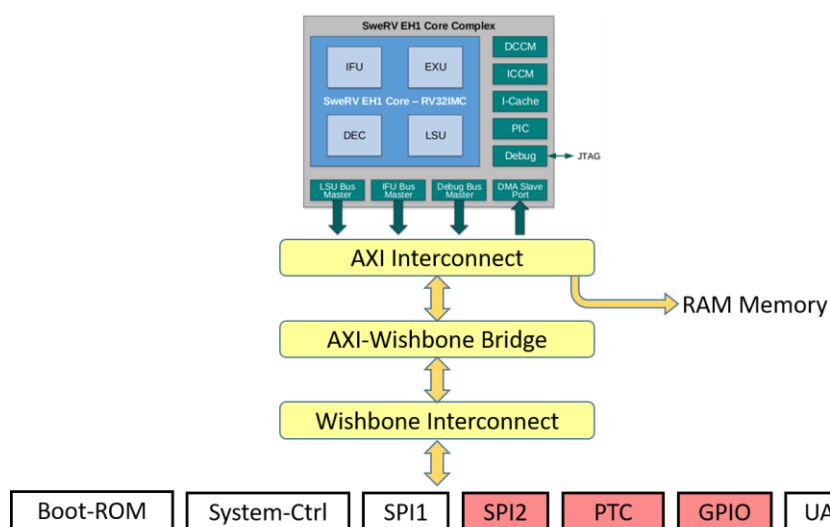


Figura 3. SweRVolfX

A Tabela 4 fornece os endereços dos periféricos mapeados em memória que estão ligados ao núcleo SweRV EH1 por meio da interconexão Wishbone.

Tabela 4. Endereços mapeados em memória do SweRVolfX

Sistema	Endereço
Boot ROM	0x80000000 - 0x80000FFF
Controlador do sistema	0x80001000 - 0x8000103F
SPI1*	0x80001040 - 0x8000107F
SPI2*	0x80001100 - 0x8000113F
Temporizador*	0x80001200 - 0x8000123F
GPIO*	0x80001400 - 0x8000143F
UART	0x80002000 - 0x80002FFF

* Periféricos adicionados no **SweRVolfX**

3. Introdução ao RVfpga-SoC

No RVfpga, o SweRVolfX foi apresentado sem nenhum detalhe sobre como o SweRVolfX foi criado. O curso RVfpga-SoC mostra como construir um subconjunto do SweRVolfX SoC do zero usando blocos IP como o núcleo SweRV, memórias e periféricos.

Este laboratório é um guia passo a passo que mostrará como começar com um CPU (o SweRV EH1 Core Complex) e, em seguida, transformá-lo num SoC. Usaremos a ferramenta de design de blocos do Vivado. A ferramenta de Block Designs do Vivado facilita a ligação dos componentes graficamente, tornando o processo mais fácil de entender e visualizar. Essa abordagem visual também ilustra como cada módulo é conectado aos outros para formar um SoC.

Os módulos podem ser classificados em três blocos ou categorias principais:

1. CPU (SweRV EH1 Core Complex)

2. Interconexão (interconexão AXI, AXI2WB e interconexão WB)
3. Periféricos (boot ROM, controlador GPIO e controlador do sistema)

O SweRVolfX tem muitos módulos diferentes e alguns não são necessários para um SoC RISC-V básico. Portanto, os módulos extras foram removidos para simplificar o Lab e focar na funcionalidade básica necessária para dar vida a um Core de CPU. A Figura 4 mostra os módulos que não serão incluídos (UART, PTC, SPI1 e SPI2).

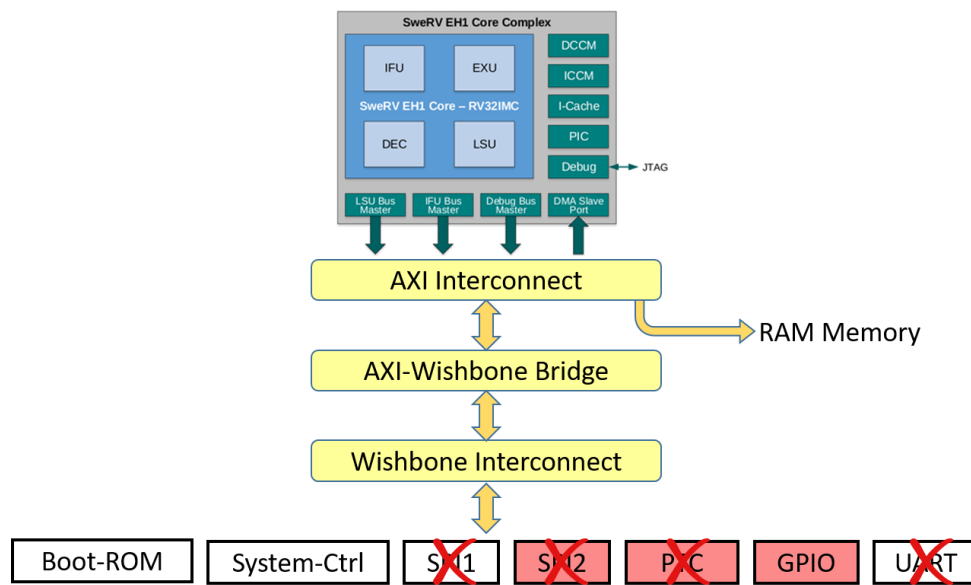


Figura 4. Um subconjunto do SweRVolfX

A Figura 5 mostra um diagrama de blocos de alto-nível do SoC que será implementado.

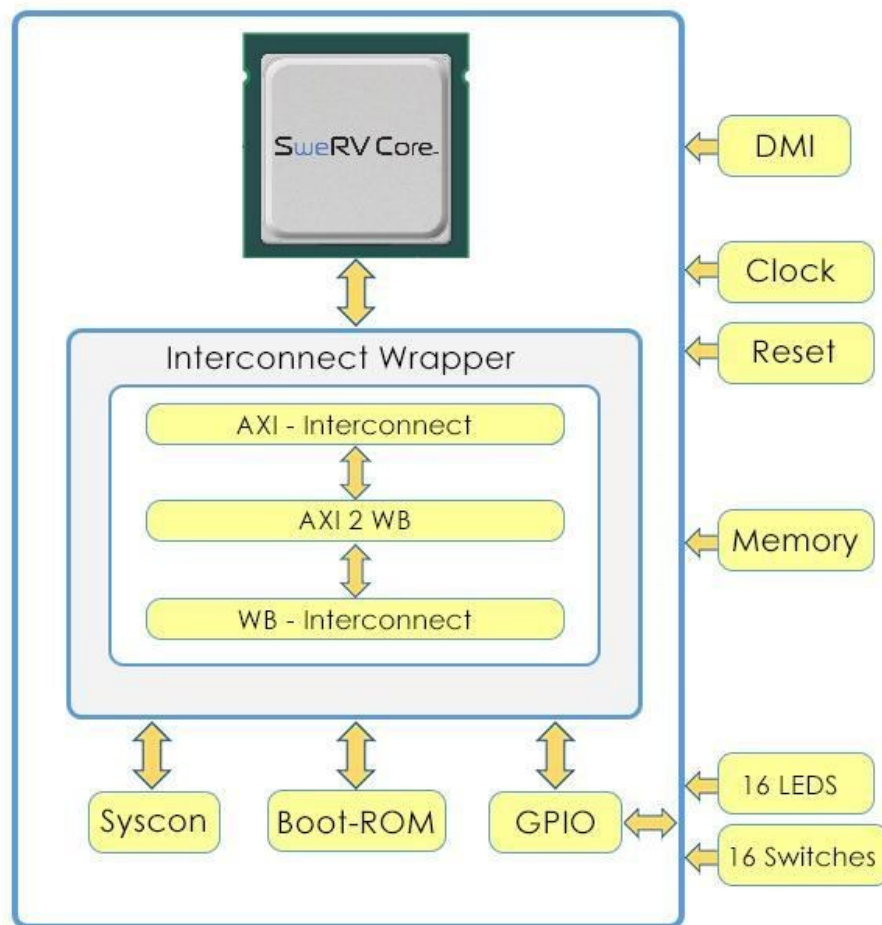


Figura 5. Diagrama de blocos de alto-nível do SoC do Lab 1

Para facilitar a aprendizagem e a compreensão, alguns componentes que fazem parte da interconexão (interconexão AXI, interconexão Wishbone e ponte AXI para Wishbone) foram agrupados num módulo Interconnect Wrapper.

Para os laboratórios que se focam no CPU, e dentro do CPU, consulte o curso RVfpga. O curso RVfpga (também conhecido como RISC-V FPGA) é um pacote que inclui instruções, ferramentas e laboratórios destinados a um processador RISC-V comercial e SoC numa FPGA (Field-Programmable Gate Array) e, em seguida, usá-lo e expandi-lo para aprender sobre arquitetura de computadores, projeto de sistemas digitais, sistemas embebidos e programação.

Para obter mais informações sobre o RVfpga, acesse a

<https://university.imgtec.com/rvfpga/>

2. Requisitos

Para concluir este laboratório, precisará ter o seguinte software instalado:

- Vivado 2019.2 Web Pack (Consulte o Guia de Instalação (Página No.04))
- Ficheiros da placa Digilent (Consulte o Guia de Instalação (Página No.05))

IMPORTANTE: antes de iniciar os Labs do RVfpga-SoC, é altamente recomendável concluir o Guia de Instalação do RVfpga-SoC.

Por exemplo, se ainda não o fez, instale o Vivado da Xilinx seguindo as instruções do Guia de Instalação do RVfpga-SoC. Certifique-se de que copiou a pasta *RVfpgaSoC* que descarregou do Imagination Univeristy Programme.

3. Criar um Projeto no Vivado

Usará o Vivado Design Suite da Xilinx para criar o subconjunto SweRVolfX usando o RTL, os ficheiros Verilog que definem o sistema. Siga estas etapas, detalhadas abaixo, para criar um projeto Vivado.

Etapa 1. Abra o Vivado

Se não instalou o Vivado, conforme descrito no Guia de Instalação do RVfpga-SoC, faça-o agora. Certifique-se de instalar também os Board Files.

Agora, execute o Vivado (no **Linux**, abra um terminal e escreva: **vivado**; no **Windows**, abra o Vivado no menu Iniciar). A janela de boas-vindas do Vivado será aberta. Clique em Create Project (criar projeto) (ver Figura 6).

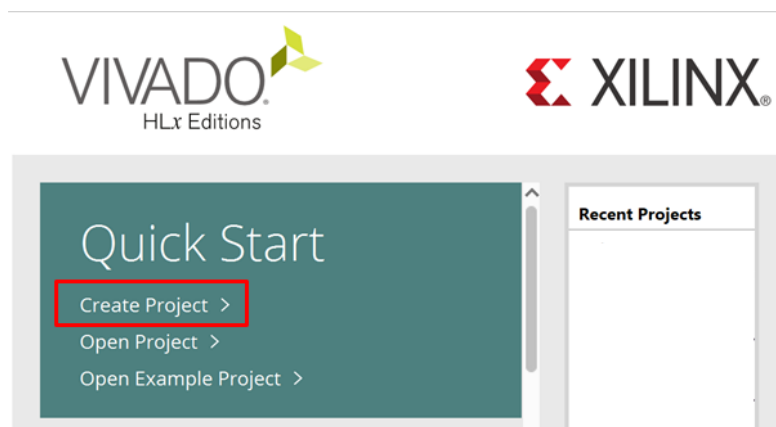


Figura 6. Janela de boas-vindas do Vivado: Criar projeto

Etapa 2. Crie um novo projeto RTL

O Assistente para Criar um Novo Projeto Vivado será aberto (veja a Figura 7). Clique em Next.

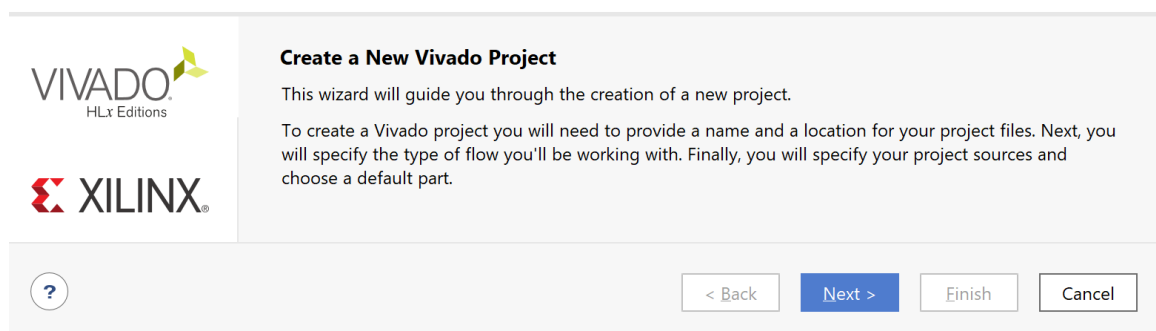


Figura 7. Assistente de criação de um novo projeto Vivado

Escreva o nome do projeto como "**Lab1**", sem espaços. Em seguida, clique em Next (ver Figura 8).

Selecione o seguinte caminho de localização do projeto:
[RVfpgaSoCPath] /RVfpgaSoC/Labs/LabProjects/Lab1

Desmarque a caixa de seleção "create project subdirectory" porque já existe uma pasta chamada "**Lab1**" na pasta "**LabProjects**".

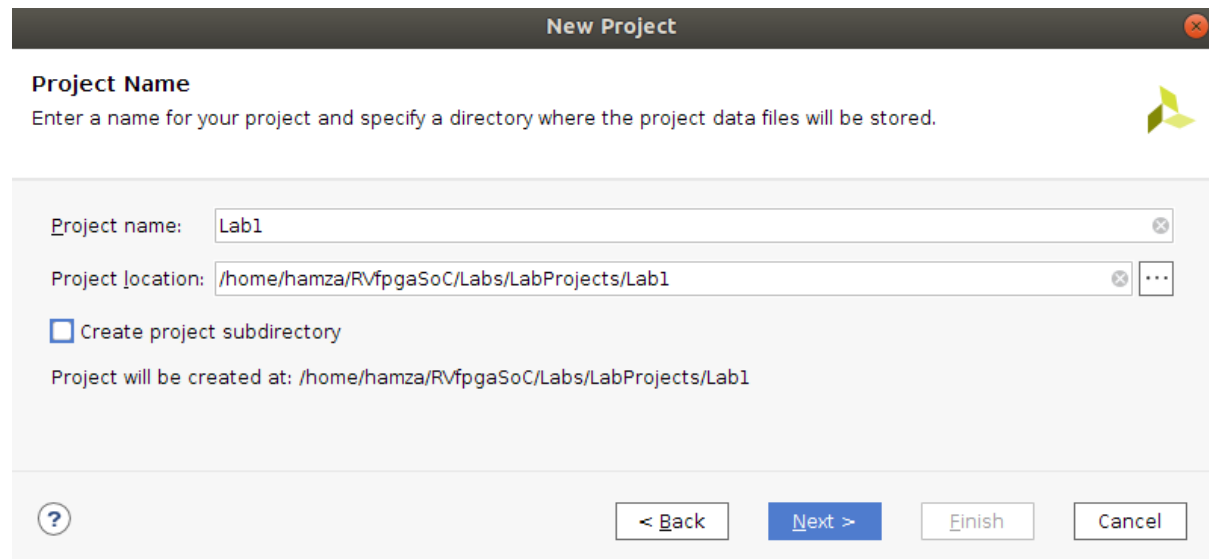


Figura 8. Nome do projeto

Selecione o tipo de projeto como RTL Project e clique em Next (ver Figura 9).

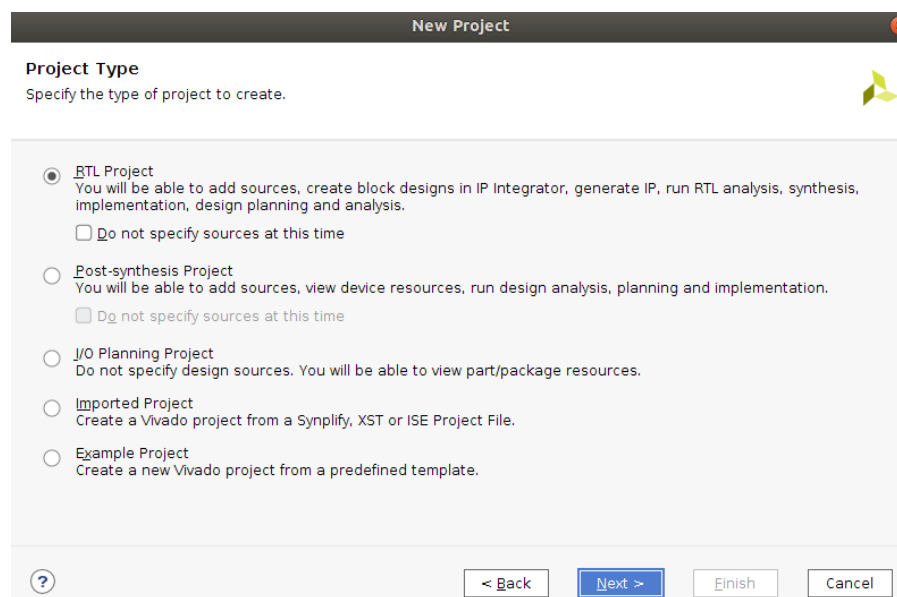


Figura 9. Projeto RTL

Etapas 3. Adicione os ficheiros RTL e os ficheiros de constraints

Na janela Add Sources (Adicionar fontes), clique em **"Add Directories"** (Adicionar diretórios) (ver Figura 10).

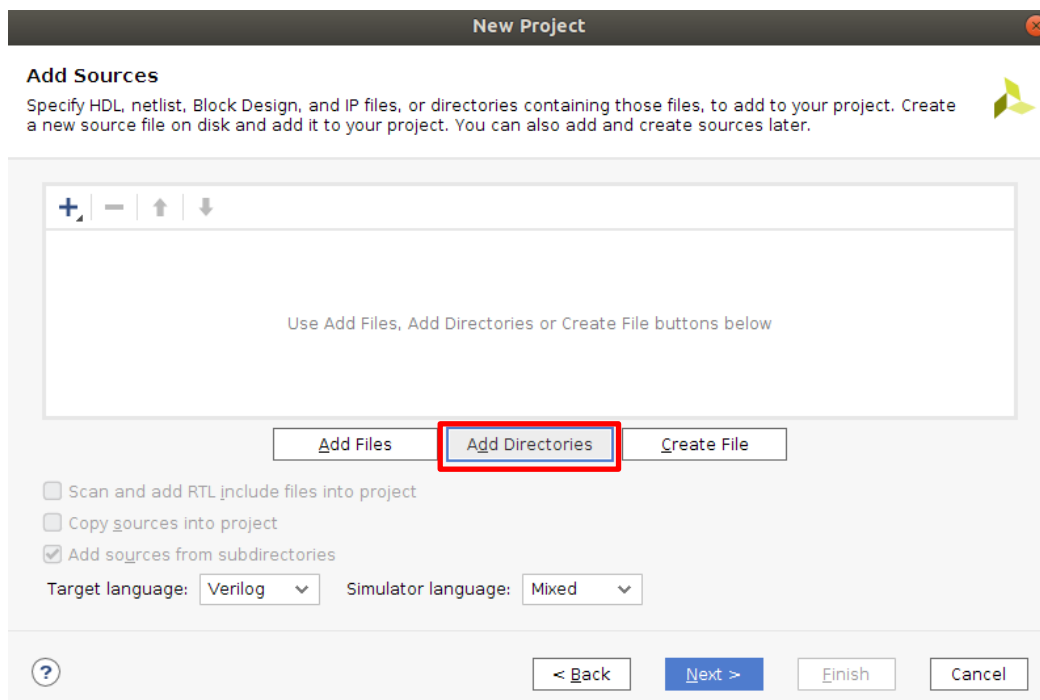


Figura 10. Adicionar diretório Sources

Agora, selecione o diretório "src" no seguinte caminho
[RVfpgaSoCPath] /RVfpgaSoC/Labs/LabResources/Lab1/src (ver Figura 11).

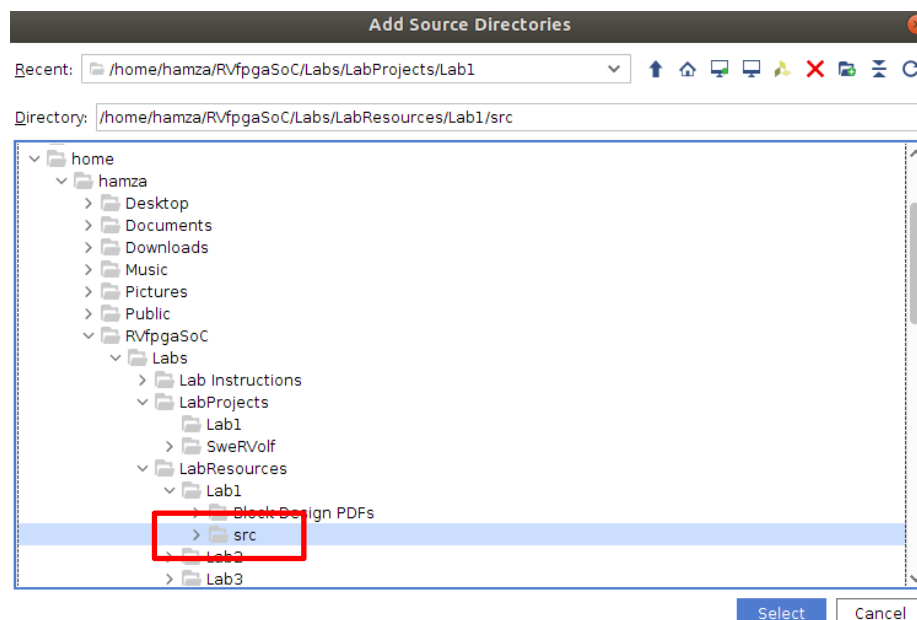


Figura 11. Selecione o diretório "src".

Clique em Select.

Em seguida, clique no botão **"Add Files"** (Adicionar ficheiros).

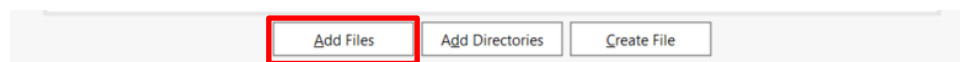


Figura 12. Adicionar ficheiros

Selecione o tipo de ficheiros como **"All Files"**. Agora, navegue até o diretório **LiteDRAM** dentro do diretório **src** que acabamos de adicionar.

- [RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/src/LiteDRAM/mem_1.init
- [RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/src/LiteDRAM/litdram_core.init

Selecione os dois ficheiros **".init"** e clique em OK para adicioná-los (Figura 13).

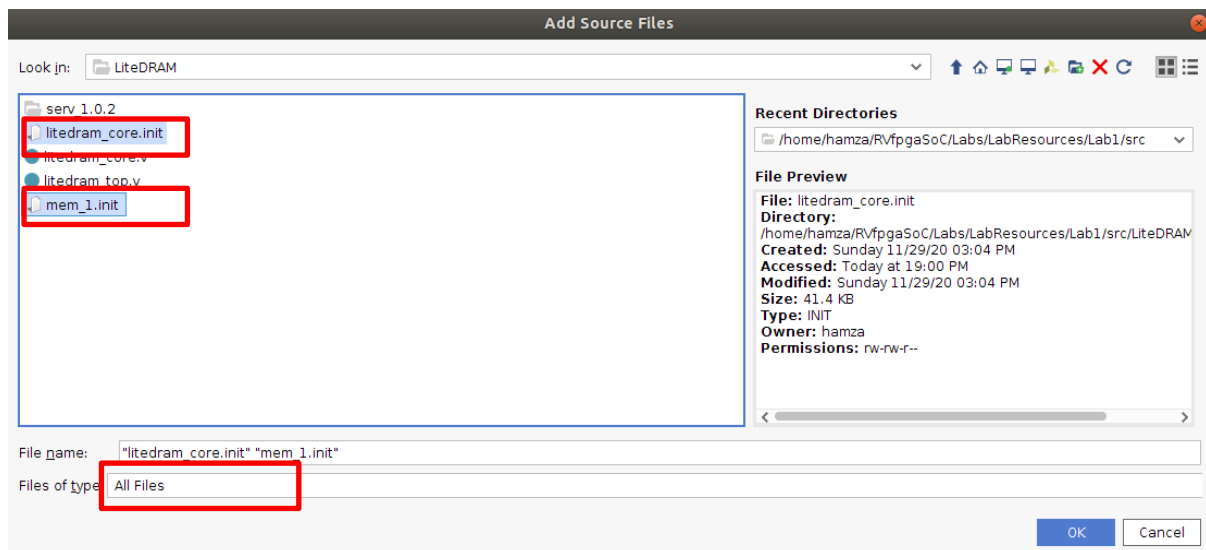


Figura 13. Adicionar os ficheiros fonte do LiteDram

Certifique-se de que todas as três caixas de seleção estejam marcadas (consulte a Figura 14).

Clique em **"Next" (Avançar)** para ir para a próxima etapa.

New Project

Add Sources

Specify HDL, netlist, Block Design, and IP files, or directories containing those files, to add to your project. Create a new source file on disk and add it to your project. You can also add and create sources later.

	Index	Name	Library	HDL Source For	Location
	1	src	xl_defaultlib	Synthesis & Simulation	/home/hamza/RVfpgaSoC/Labs/LabResources/Lab1
	2	litedram_core.init	N/A	N/A	/home/hamza/RVfpgaSoC/Labs/LabResources/Lab1/src/LiteDRAM
	3	mem_1.init	N/A	N/A	/home/hamza/RVfpgaSoC/Labs/LabResources/Lab1/src/LiteDRAM

Add Files
Add Directories
Create File

☒ Scan and add RTL include files into project
☒ Copy sources into project
☒ Add sources from subdirectories

Target language: Verilog
Simulator language: Mixed

? < Back
Next > Finish Cancel

Figura 14. Adicionar os ficheiros-fonte

Agora adicionará os ficheiros de constraints do sistema. Esses ficheiros mapeiam os nomes dos sinais para os pinos da placa. Por exemplo, os LEDs da placa Nexys A7 FPGA são conectados aos pinos da FPGA na placa por meio de pistas na PCB. O Vivado precisa de saber disso para mapear o nome correto do sinal na RTL para o pino correto da FPGA. Por exemplo, a linha seguinte no ficheiro *[RVfpgaSoCPath]/RVfpgaSoC/src/rvfpga.xdc*, um ficheiro de constraints de projeto da Xilinx, indica que o pino H17 da FPGA é mapeado para o LED menos significativo (`o_led[0]`) e que ele usa sinalização LVCMOS 3,3V:

```
set_property -dict { PACKAGE_PIN H17 IOSTANDARD LVCMOS33 } [get_ports { o_led[0] }]
```

Observe que o nome do sinal `o_led` é o nome usado no código Verilog do RVfpga para acionar os LEDs da placa Nexys A7.

Na janela Add Constraints, clique em **"Add Files" (Adicionar ficheiros)** e selecione os dois ficheiros a seguir (ver Figura 15):

[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/src/rvfpga.xdc
[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/src/litedram.xdc

Em seguida, clique em **Next**.

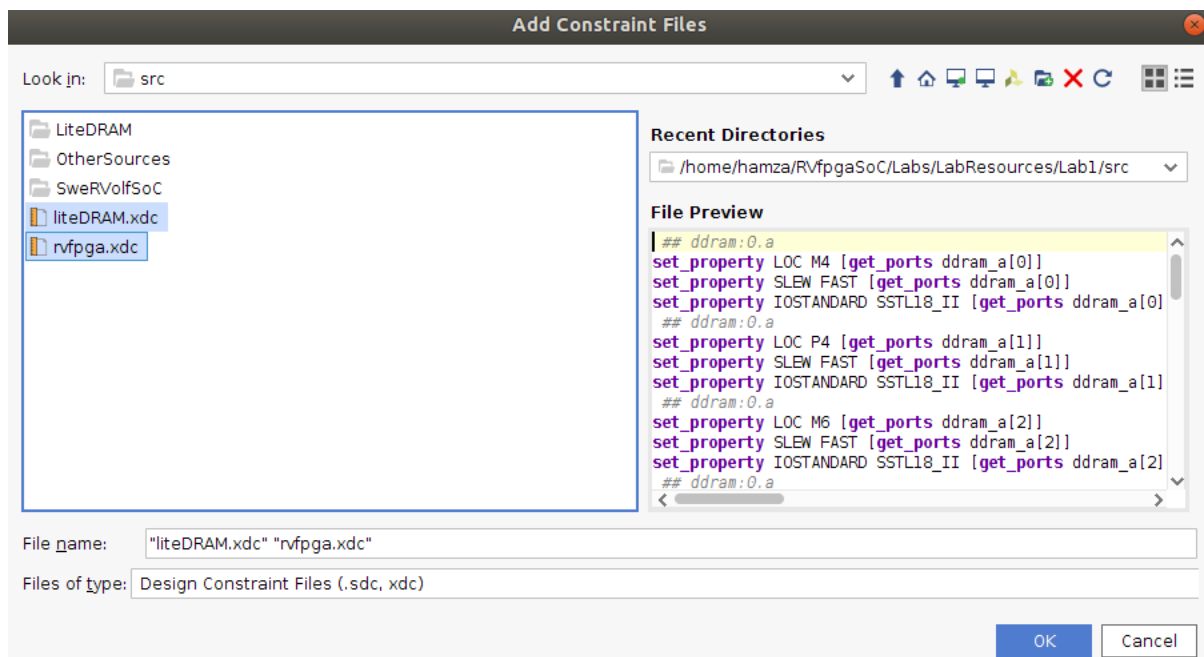


Figura 15. Adicionar ficheiros constraints

Etapa 4. Selecione Nexys A7 como a placa de implementação

Na janela Default Part, clique em Boards e selecione Nexys A7-100T (ver Figura 16). pode usar a caixa Search para restringir os resultados. Também notará que o nome da FPGA a usar está listada na coluna Part: xc7a100tcs324-1. Isso indica que se trata de uma FPGA Xilinx Artix-7 com 100k portas equivalentes com um encapsulamento CSG (chip-scale grid) e 324 pinos.

Clique em Next.

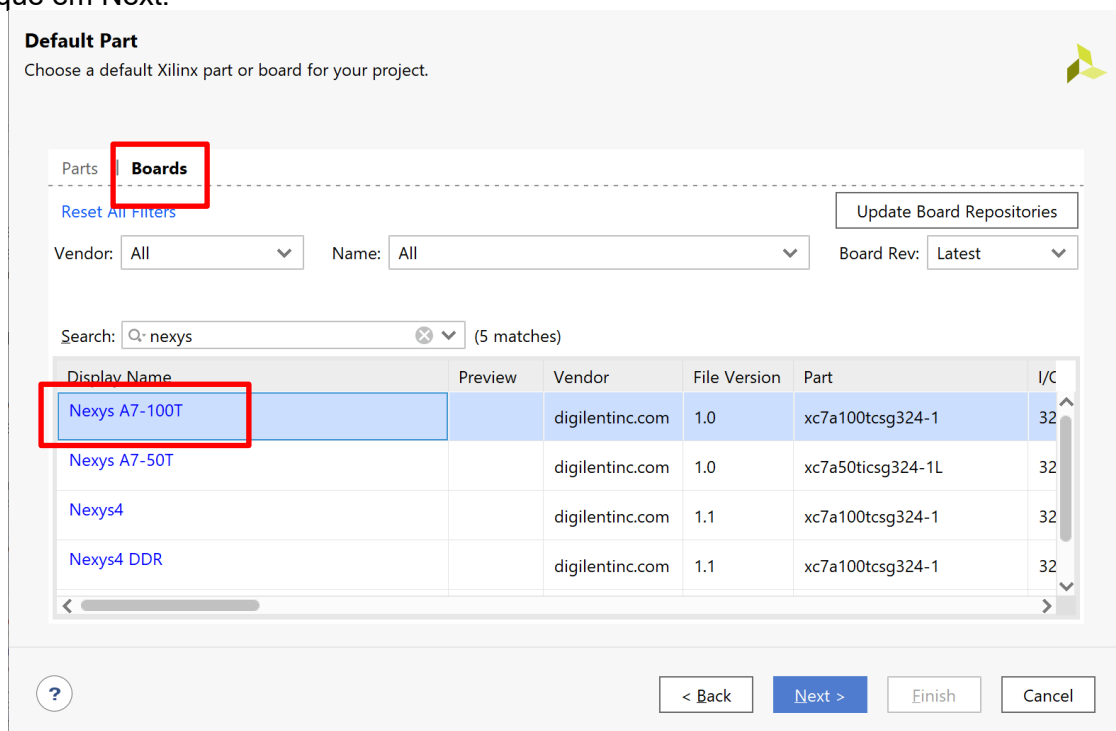


Figura 16. Selecione a placa a usar: Nexys A7-100T

Na janela New Project Summary, clique em **Finish** (ver Figura 17).

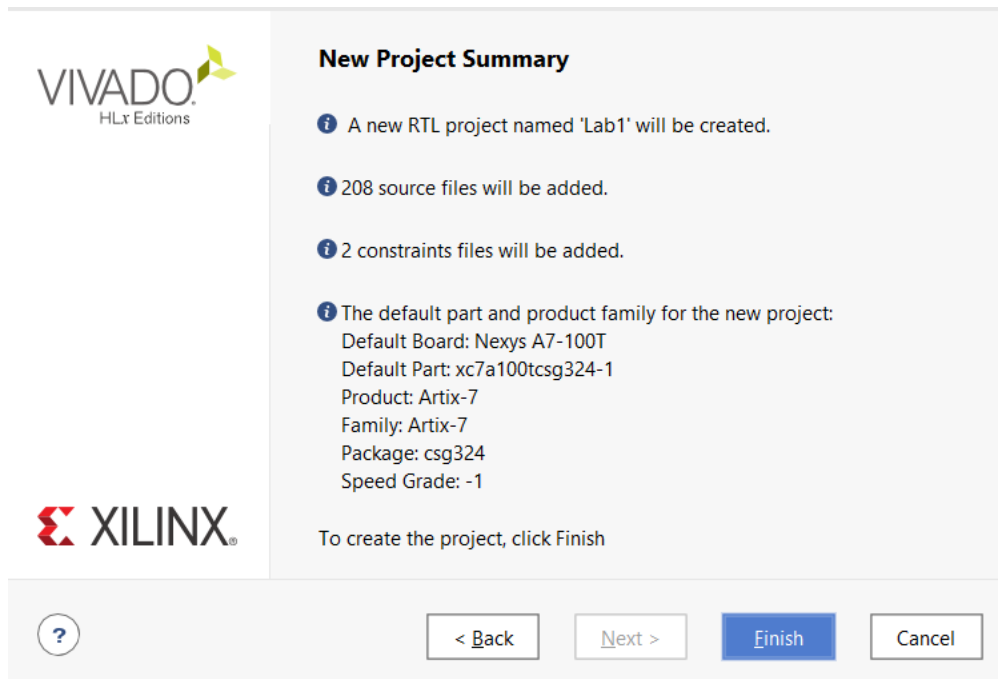


Figura 17. Janela New Project Summary

Observe que, quando o projeto terminar de ser configurado, ele indicará que existem ficheiros com erros de sintaxe - isso será corrigido na próxima etapa.

Etapa 5. Definir rvfpga como módulo principal

O projeto será inicializado. Agora definirá o módulo rvfpga como o módulo principal. No painel Sources, vá para baixo em Design Sources, clique com o botão direito do rato no módulo rvfpga e selecione Set as Top (Figura 18). Também pode encontrar o módulo rvfpga escrevendo esse nome na caixa de pesquisa, conforme mostrado. Isso define o rvfpga como o módulo de nível mais alto (principal) da hierarquia e o que será sintetizado e implementado no FPGA. Depois de definir rvfpga como o módulo principal, a hierarquia será atualizada.

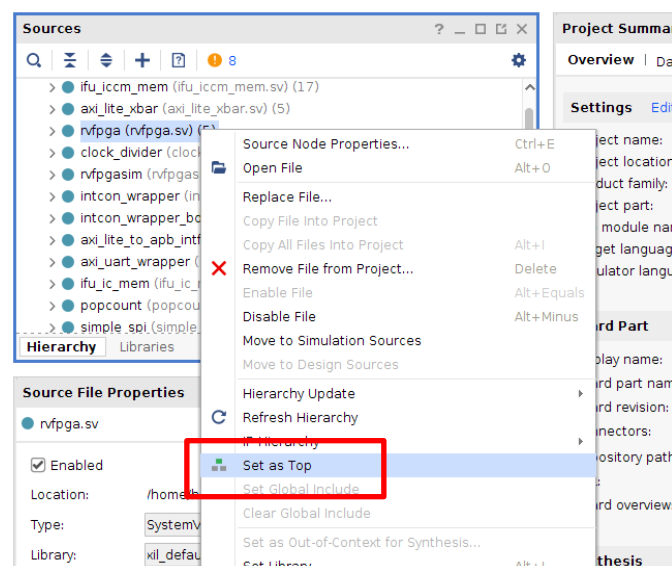


Figura 18. Definir rvfpga como módulo principal

Etapa 6. Defina os ficheiros header Verilog como ficheiros de inclusão globais

Agora, ainda no painel Sources, em Design Sources, expanda o grupo de ficheiros Non-modules e clique em `common_defines.vh`. As propriedades do ficheiro serão abertas no painel Source File Properties, logo abaixo do painel Sources. Clique em Global Include para marcar essa caixa (Figura 19). A hierarquia será atualizada e incluirá esse ficheiro em Design Sources/Global Include.

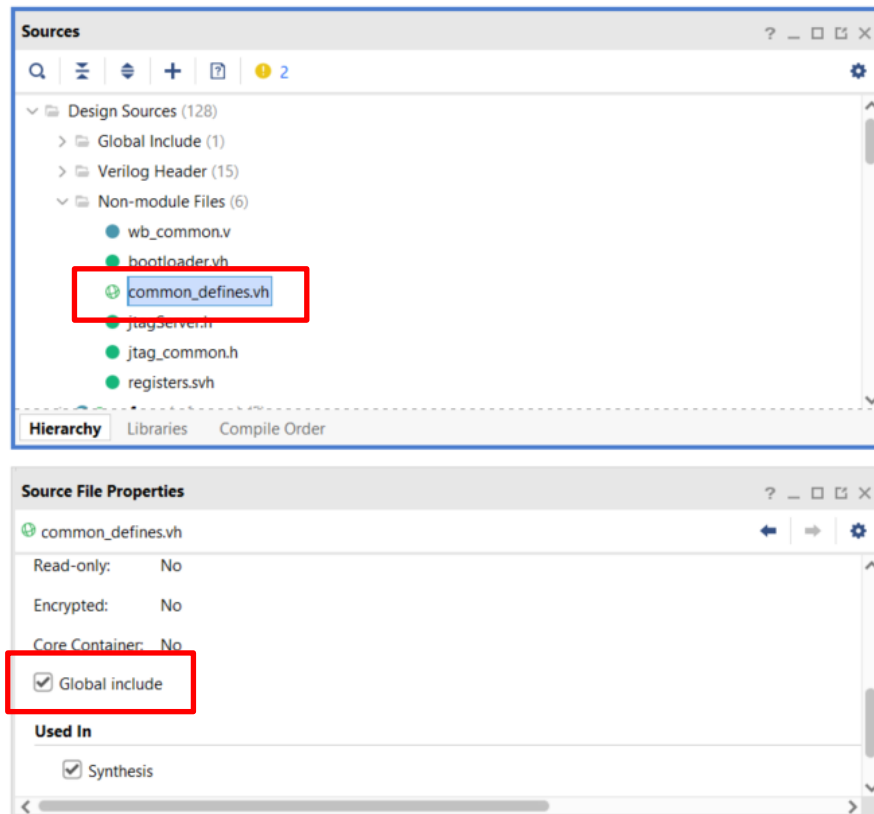


Figura 19. Definir `common_defines.vh` como um ficheiro de inclusão global

Da mesma forma, defina os ficheiros "`assign.svh`", "`registers.svh`" e "`typedef.svh`" do SystemVerilogHeader como ficheiros de inclusão global (veja a Figura 20).

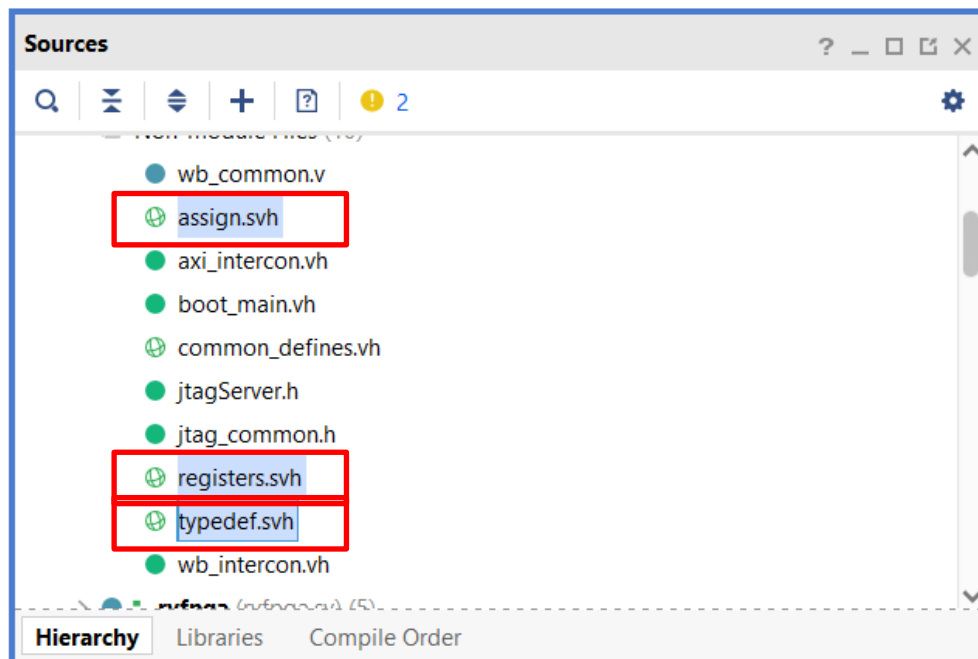


Figura 20. Definir ficheiros ".svh" como um ficheiro de inclusão global

Agora, expanda o grupo de ficheiros "**unknown**" e clique em "**litedram_core.init**". Em seguida, clique no botão Properties ao lado do botão General no painel Source File Properties. Clique em "**IS_Global_INCLUDE**" para marcar essa caixa (Figura 21).

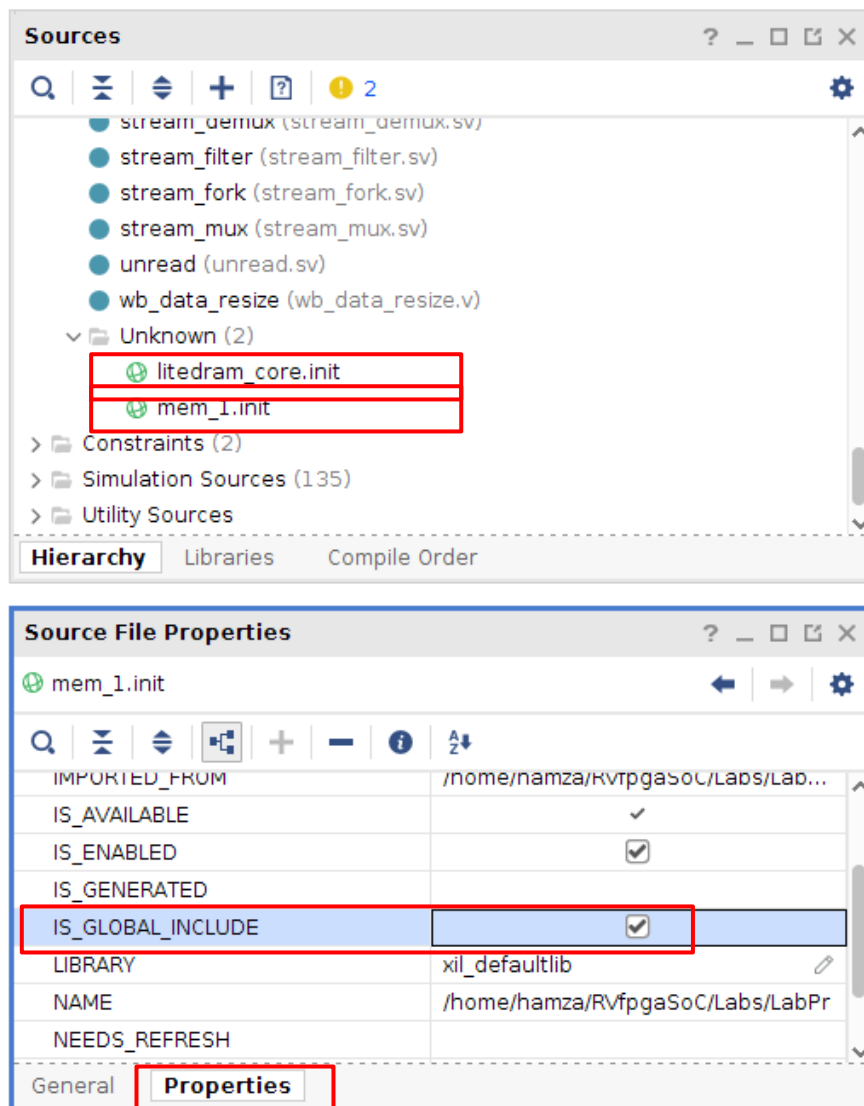


Figura 21. Definir litedram_core.init como um ficheiro de inclusão global

Agora, faça o mesmo com o ficheiro "**mem_1.init**" e defina-o como um ficheiro de inclusão global, assim como fez com o ficheiro "**litedram_core.init**".

Etapas 7. Adicione boot_main.mem ao projeto

No painel Flow Navigator, clique em Add Sources, deixe a opção por omissão ("Add or create design sources") e clique em Add Files (Figura 22). Navegue até *[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/src/SweRVolfSoC/BootROM/sw* e selecione *boot_main.mem* (Figura 22). A hierarquia será atualizada e incluirá esse ficheiro em Design Sources/Memory File.

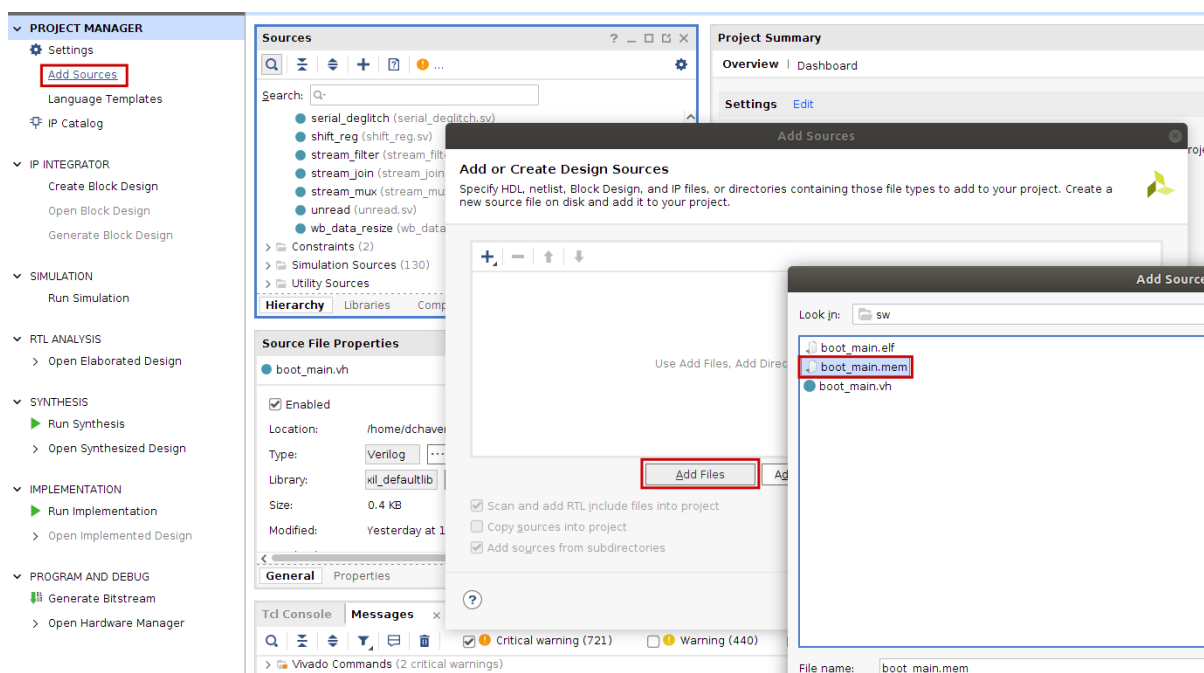


Figura 22. Adicionar ficheiro de memória boot_main.mem

Os ficheiros de origem do design foram adicionados e agora podemos prosseguir e começar a criar o Block Design.

4. Criar o Block Design

Irá usar o Block Design do Vivado para adicionar os módulos necessários para criar o SweRVolfX reduzido e, em seguida, ligar os módulos entre si.

Etapas 1. Clique em Create Block Design

Crie um novo Block Design no Flow Navigator clicando em Create Block Design no cabeçalho IP Integrator (Figura 23).

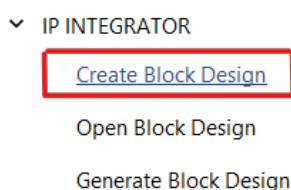


Figura 23. Criar Block Design

Etapas 2. Selecione o nome do Block Design

Selecione o nome do projeto como "BD" para evitar conflitos de nomes posteriormente no laboratório (Figura 24).

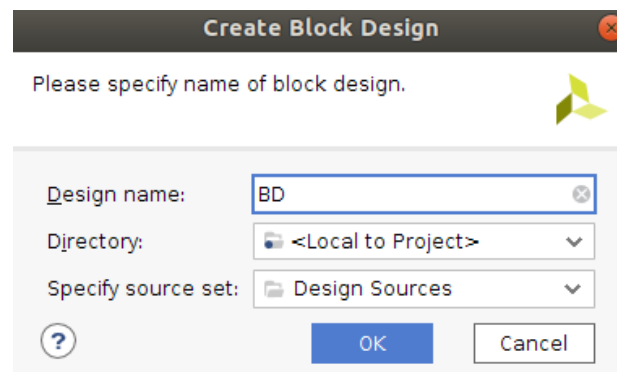


Figura 24. Selecionar o nome e o diretório do Block Design

Agora irá ver o painel Diagram do Block Design em branco. (Figura 25)

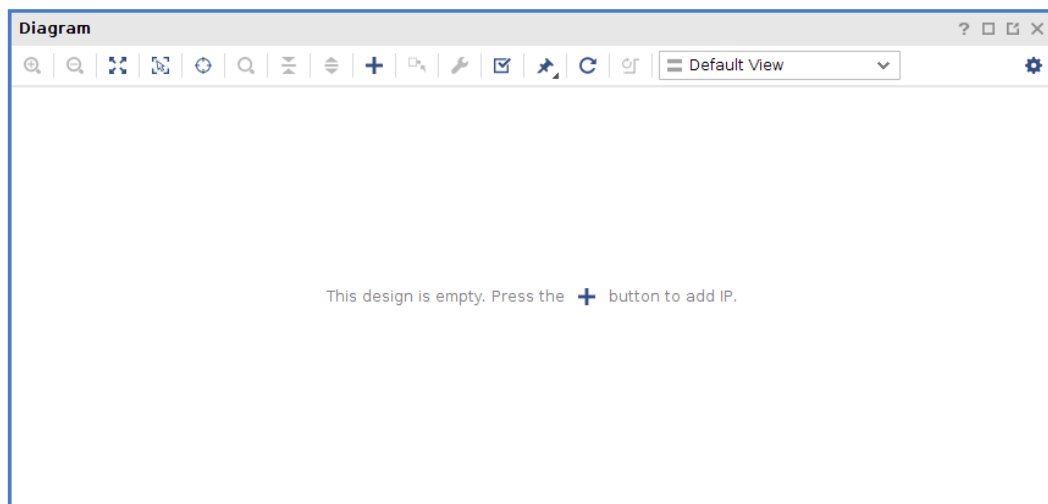


Figura 25. Block Design em branco

Etapa 3. Adicionar módulos ao Block Design

Agora, pode começar a adicionar módulos ao nosso Block Design. Para isso, clique com o botão direito do rato no Block Design em branco e selecione a opção "**Add Module**" (Figura 26).

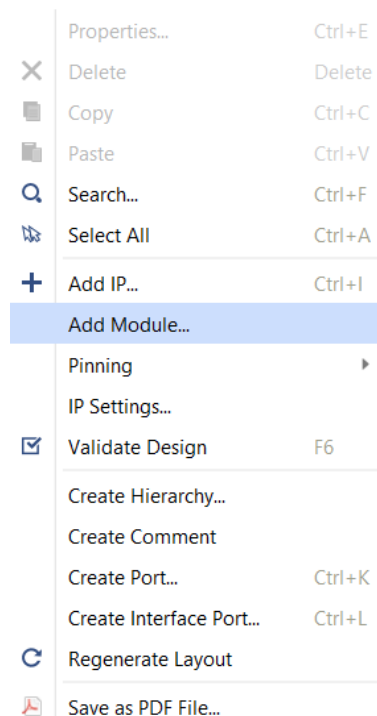


Figura 26. Adicionar módulo

Será exibida uma caixa de diálogo. Vá para baixo ou escreva na caixa de pesquisa o nome dos módulos necessários que gostaria de adicionar. Começará adicionando o SweRV EH1 Core Complex.

Selecione "**swerv_wrapper_verilog**" e clique em OK.

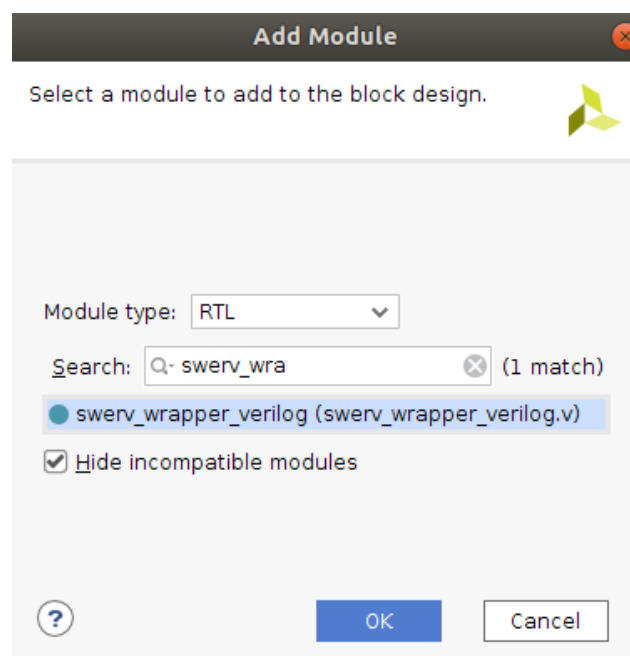


Figura 27. Adicionar o swerv_wrapper_verilog

Uma mensagem de aviso crítico será exibida (ver Figura 28). Clique em OK para ignorar essa mensagem de aviso.

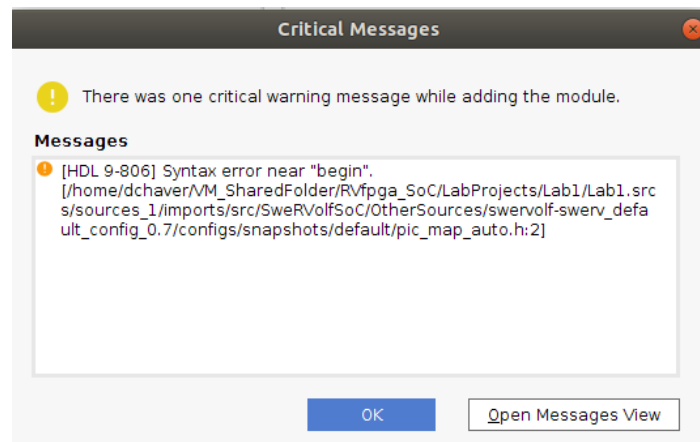


Figura 28. Mensagem de aviso crítico

Depois de adicionar o módulo, pode visualizar e aceder a todos os pinos de "ifu_axi", "lsu_axi" ou "sb_axi" clicando no ícone "+" no módulo.

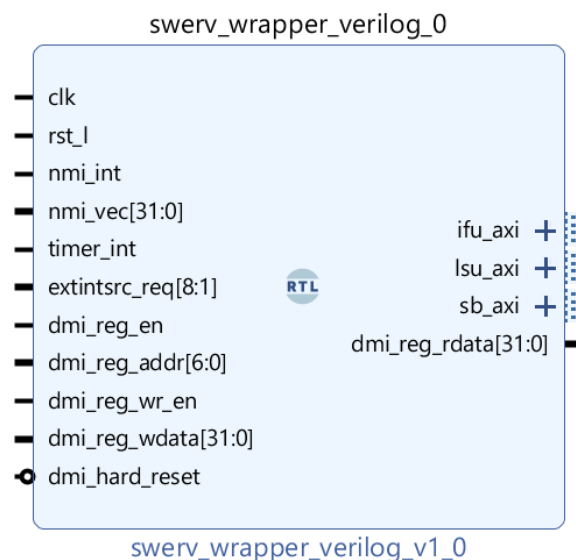
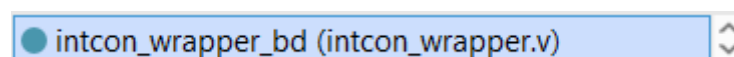


Figura 29. "Módulo "swerv_wrapper_verilog"

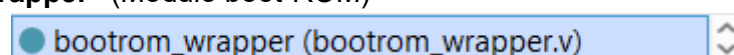
Da mesma forma, adicionará os seguintes módulos:

- **"intcon_wrapper_bd"** (Módulo Interconnect Wrapper): É um módulo de encapsulamento que contém todos os três módulos de interconexão nele encapsulados.



Agora, adicionará os periféricos necessários para nosso SoC:

- **"bootrom_wrapper"** (Módulo boot-ROM)



- **"gpio_wrapper"** (Módulo superior de GPIO)

- **gpio_wrapper** (gpio_wrapper.v)
- **"syscon_wrapper"** (módulo do controlador do sistema)
- **syscon_wrapper** (syscon_wrapper.v)

Adicionará os 32 módulos **"bidirec"** para anexar ao módulo GPIO. 16 deles serão para os LEDs e 16 serão para os interruptores.

- **"bidirec"** (módulo GPIO bidirecional)
- **bidirec** (bidir.v)

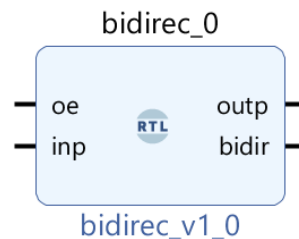


Figura 30. Módulo bidir GPIO

Da mesma forma, adicionará 32 desses módulos ao Block Design.

Uma via rápida para adicionar esses 32 módulos é copiar e colar os blocos no Block Diagram. Primeiro, copie 1 bloco **"bidirec"** e cole-o; depois, copie 2 blocos e cole-os; em seguida, repita o processo de copiar e colar até que tenha 32 blocos do módulo **"bidirec"** adicionados ao Block Design.

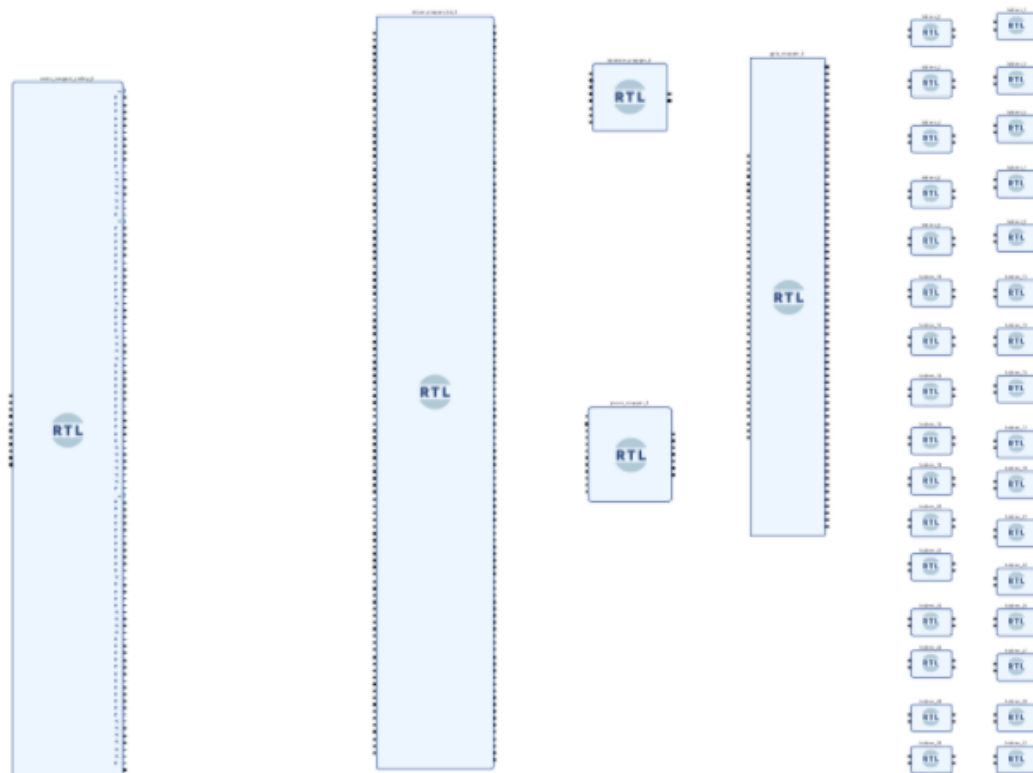


Figura 31. Os módulos necessários adicionados ao Block Design

(Figura 31). Começando pelo lado esquerdo, veja o módulo **SweRV Core** (swerv_wrapper_verilog_0); em seguida, à direita, veja o módulo **Interconnect Wrapper** (intcon_wrapper_bd_0) e os quatro módulos periféricos, que são o módulo **Boot-ROM** (bootrom_wrapper_0), o módulo **System Controller** (syscon_wrapper_0) e o módulo **GPIO** (gpio_wrapper_0). No lado mais à direita, irá ver os 32 módulos **Bidirec** (bidirec_x).

Etapa 4. Ligue os módulos

Agora, ligue os módulos uns aos outros pino-a-pino ou, em alguns casos, barramento-a-barramento. Comece a ligar o "swerv_wrapper_verilog" com o "intcon_wrapper_bd". Três conjuntos diferentes de pinos precisam ser ligados entre esses módulos relacionados aos seguintes submódulos do núcleo:

- **IFU** (Instruction Fetch Unit)
- **LSU** (Load/Store Unit)
- **SB** (Store Byte)

Primeiro, comece conectando os pinos relacionados com a IFU. Conecte o pino "ifu_axi_arid[2:0]" do módulo "swerv_wrapper_verilog" ao pino "i_ifu_arid[2:0]" do "intcon_wrapper_bd".

Da mesma forma,

ifu_axi_araddr[31:0] será conectado a *i_ifu_araddr[31:0]*,

ifu_axi_arsize[2:0] será conectado a *i_ifu_arsize[2:0]*,

ifu_axi_arlen[7:0] será conectado a *i_ifu_arlen[7:0]* e assim por diante (ver Figura 32).

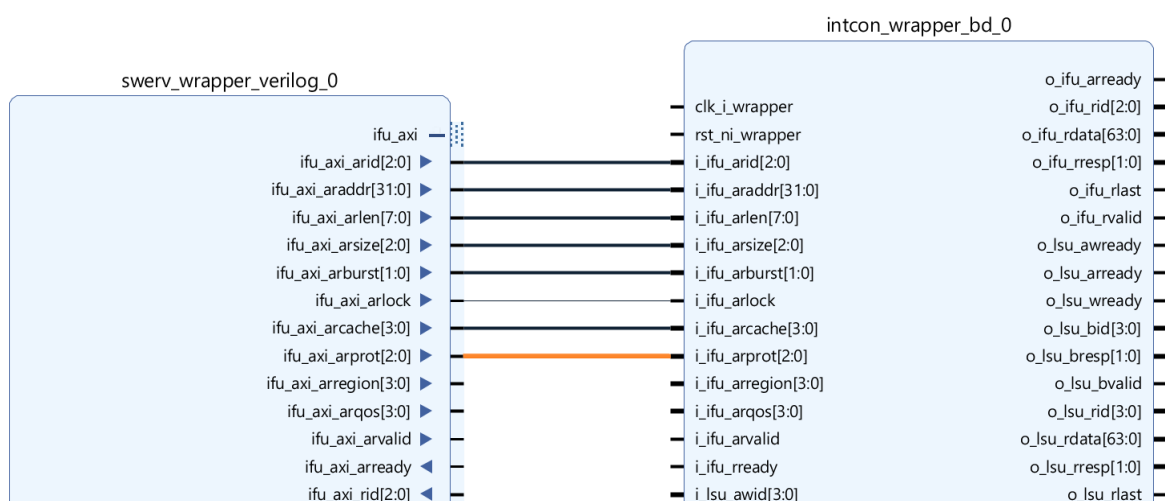


Figura 32. Conecte os pinos relevantes

Da mesma forma, ligue todos os pinos da IFU (Instruction Fetch Unit) do "swerv_wrapper_verilog" com os pinos da IFU do "intcon_wrapper_bd" (Figura 33).

PDF: PDFs de alta qualidade do Block Design, mostrando detalhes de close-up das ligações, estão disponíveis em: [RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs/InternalConnections/1_SwervW_IntconW_IFU.pdf

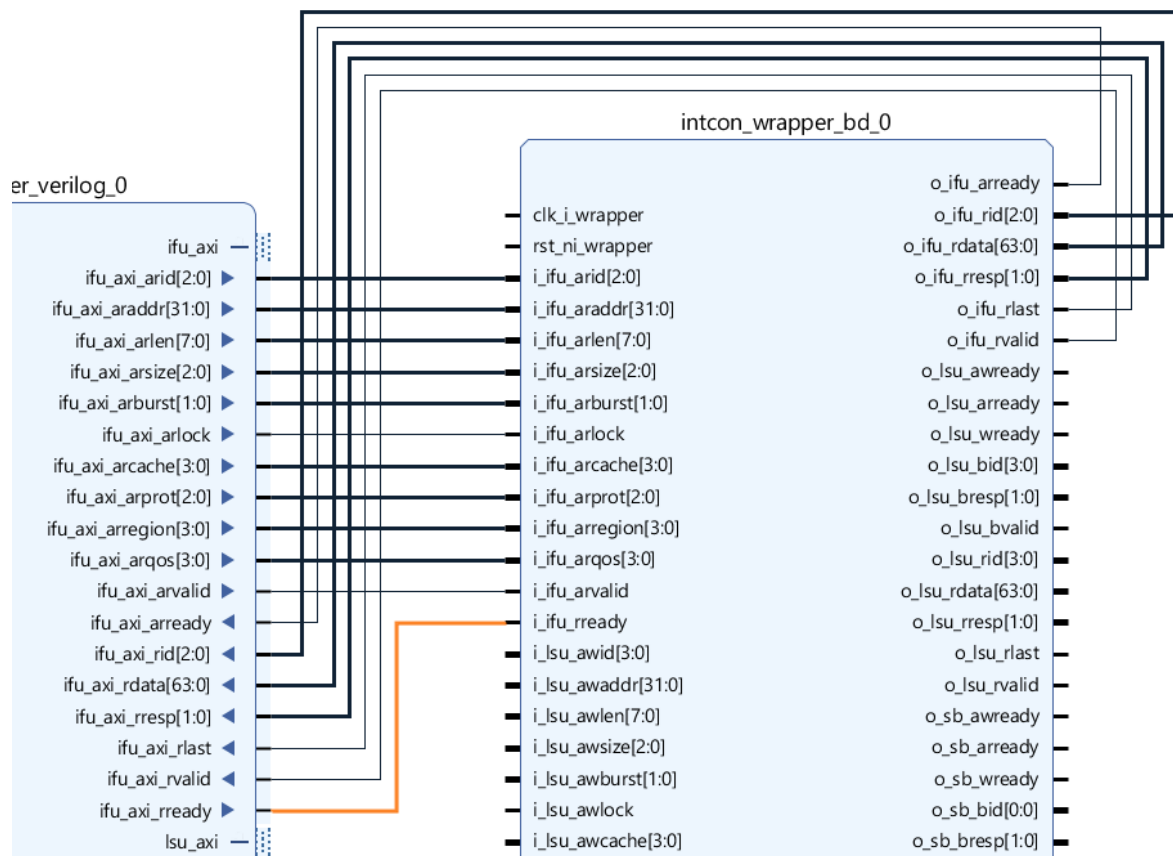


Figura 33. Conecte todos os pinos da IFU

Agora, ligue todos os pinos **LSU (Load Store Unit)** do "`swerv_wrapper_verilog`" aos pinos **LSU** do "`intcon_wrapper_bd`". Realizará o mesmo processo que fez para os pinos da IFU, ligando cada pino da **LSU** do módulo "`swerv_wrapper_verilog`" com seu respectivo pino no módulo "`intcon_wrapper_bd`" (Figura 34).

PDF: PDFs de alta qualidade do Block Design, mostrando detalhes de close-up das ligações, estão disponíveis em: [\[RVfpgaSoCPath\]/RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs/InternalConnections/2_SwervW_IntconW_LSU.pdf](#)

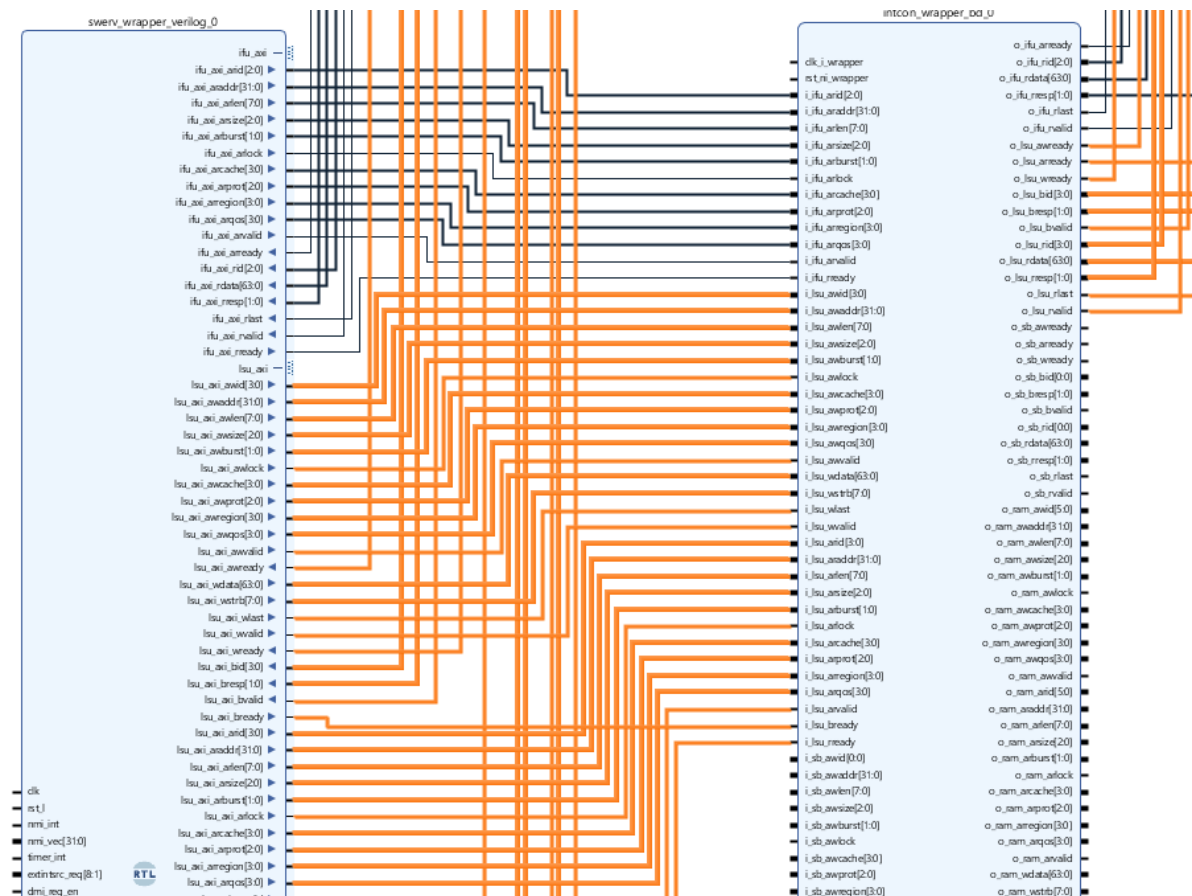


Figura 34. Ligue todos os pinos da LSU

Agora, prossiga com a ligação dos pinos **SB**. Da mesma forma, ligue todos os pinos **SB** do "swerv_wrapper_verilog" com seus respectivos pinos **SB** do "intcon_wrapper_bd" (ver Figura 35).

PDF: PDFs de alta qualidade do Block Design, mostrando detalhes de close-up das ligações, estão disponíveis em: [RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs/InternalConnections/3_SwervW_IntconW_SB.pdf

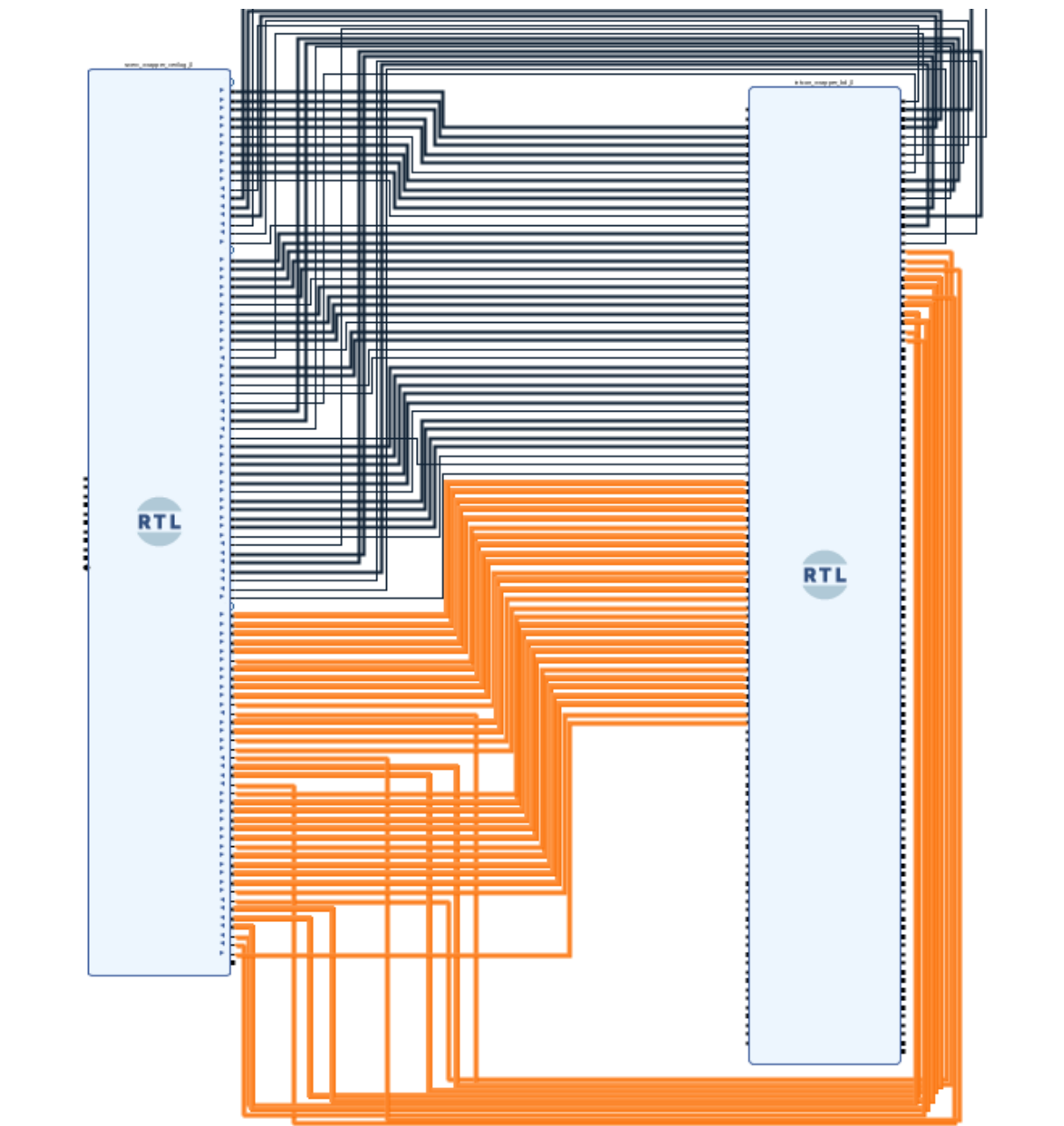


Figura 35. Conecte todos os pinos SB

Em seguida, ligará os periféricos com o "Intcon_wrapper_bd". Comece com o módulo "bootrom_wrapper" unindo os fios "wb_rom_xxx_x" do "Intcon_wrapper_bd" (Figura 36).

PDF: PDFs de alta qualidade do Block Design, mostrando detalhes de close-up das ligações, estão disponíveis em: [\[RVfpgaSoCPath\]/RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs/InternalConnections/4_BootRomW_IntconW.pdf](#)

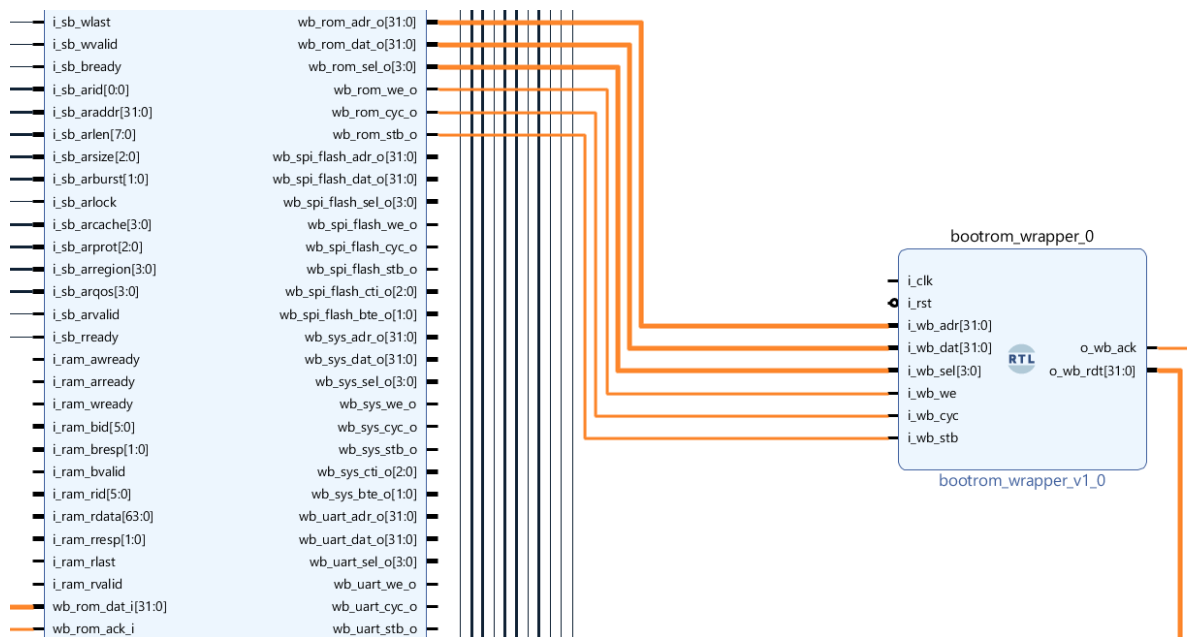


Figura 36. Ligação do módulo BootROM com o módulo de encapsulamento de interconexão

Agora, ligue o módulo "**syscon_wrapper**" com o módulo "**Intcon_wrapper_bd**" (Figura 37).

PDF: PDFs de alta qualidade do Block Design, mostrando detalhes de close-up das ligações, estão disponíveis em: [RVfpgaSoCPath] /RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs/InternalConnections/5_SysconW_IntconW.pdf

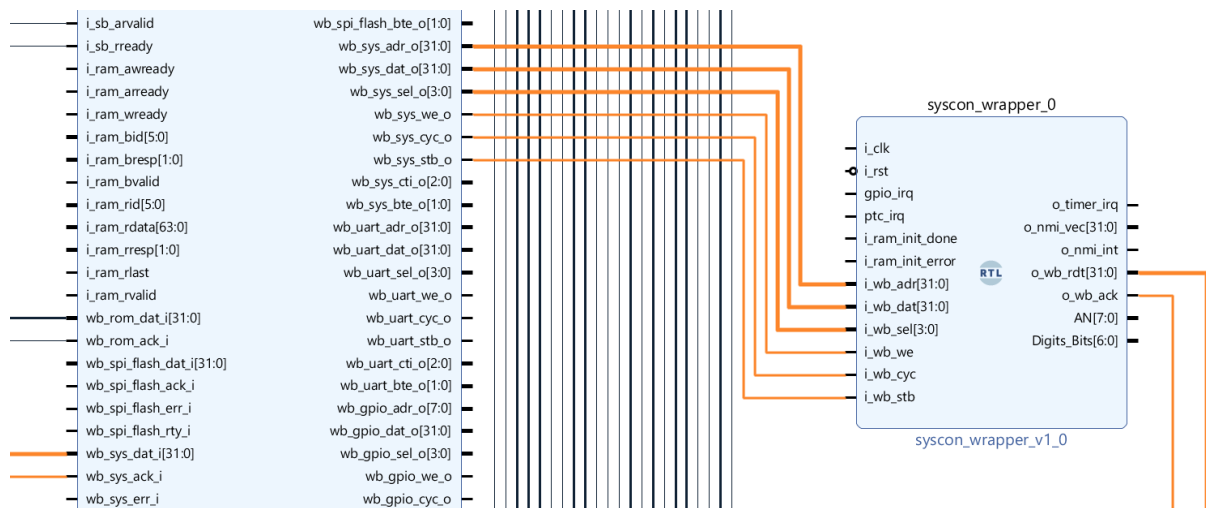


Figura 37. Ligação do Syscon com os pinos de interconexão WB

Os seguintes pinos do "**syscon_wrapper**" serão conectados ao "**swerv_wrapper_verilog**" (Figura 38).

- `o_timer_irq`
- `o_nmi_vec[31:0]`
- `o_nmi_int`

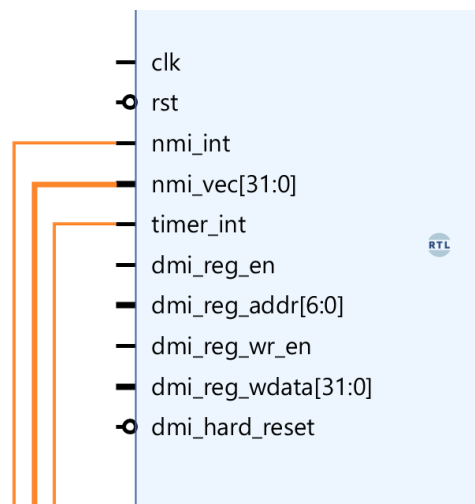


Figura 38. Conecte o syscon_wrapper com os pinos do swerv_wrapper_verilog

Agora vamos conectar o módulo "gpio_wrapper" com o "intcon_wrapper_bd". Conecte os pinos "wb_gpio_xxx_x" do módulo "intcon_wrapper_bd" com os pinos do módulo "gpio_wrapper" (Figura 39).

Conecte o pino "wb_inta_o" do módulo "gpio_wrapper" com o pino "gpio_irq" do módulo "syscon_wrapper".

PDF: PDFs de alta qualidade do Block Design, mostrando detalhes de close-up das ligações, estão disponíveis aqui: [\[RVfpgaSoCPath\] /RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs/InternalConnections/6_GpioW_IntconW.pdf](#)

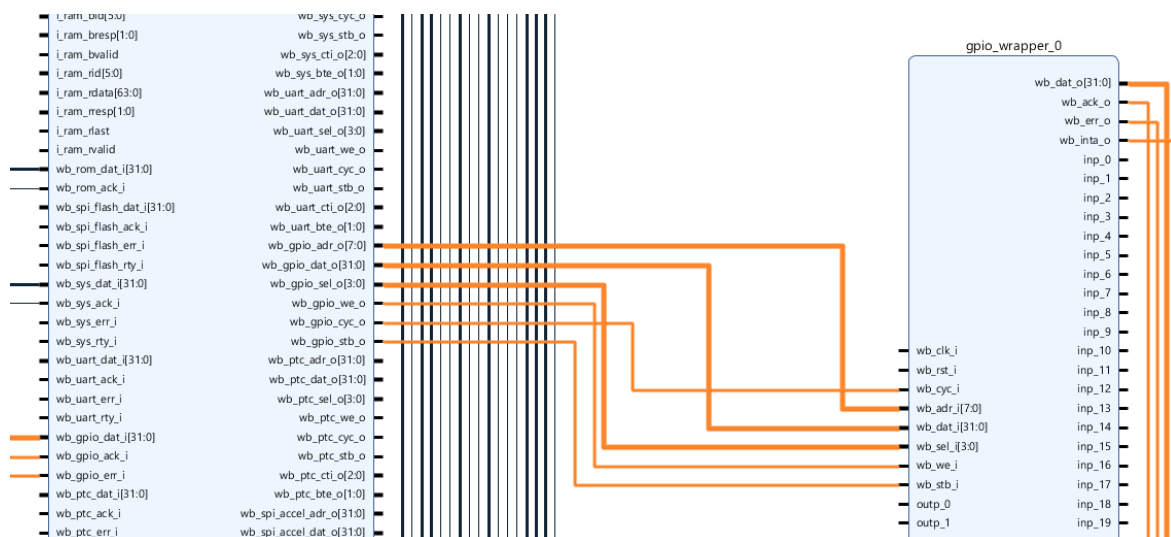


Figura 39. Conecte o gpio_wrapper com os pinos intcon_wrapper

Ligue os 32 módulos GPIO "bidirec" ao módulo "gpio_wrapper" que já ligou. Especificamente, ligue o módulo "gpio_wrapper" com os módulos "bidirec_x", em que x é um número de 1 a 32. As ligações serão feitas da seguinte forma:

O pino "inp_0" de "gpio_wrapper_0" será conectado a "inp" de "bidirec_0",

O pino "**inp_1**" do "**gpio_wrapper_0**" é ligado ao "**inp**" do "**bidirec_1**" e, da mesma forma, essas ligações continuarão até a última ligação "**inp**", que é o "**inp_31**" do "**gpio_wrapper**", que será ligado ao "**inp**" do "**bidirec_31**".

Da mesma forma,

O pino "**oe_0**" de "**gpio_wrapper_0**" é ligado a "**oe**" de "**bidirec_0**",

O pino "**oe_1**" do "**gpio_wrapper_0**" é ligado ao "**oe**" do "**bidirec_1**" e, da mesma forma, essas ligações continuarão até a última ligação "**oe**", que é o "**oe_31**" do "**gpio_wrapper**", que será ligado ao "**oe**" do "**bidirec_31**".

E da mesma forma novamente,

O pino "**outp_0**" de "**gpio_wrapper_0**" é ligado a "**outp**" de "**bidirec_0**",

O pino "**outp_1**" de "**gpio_wrapper_0**" será é ligado a "**outp**" de "**bidirec_1**" e, da mesma forma, essas ligações continuam até a última ligação "**outp**", que é "**outp_31**" de "**gpio_wrapper**", que será ligada a "**outp**" de "**bidirec_31**".

PDF: PDFs de alta qualidade do Block Design, mostrando detalhes de close-up das ligações, estão disponíveis em: [RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs/InternalConnections/7_GpioW_32xBidirec.pdf

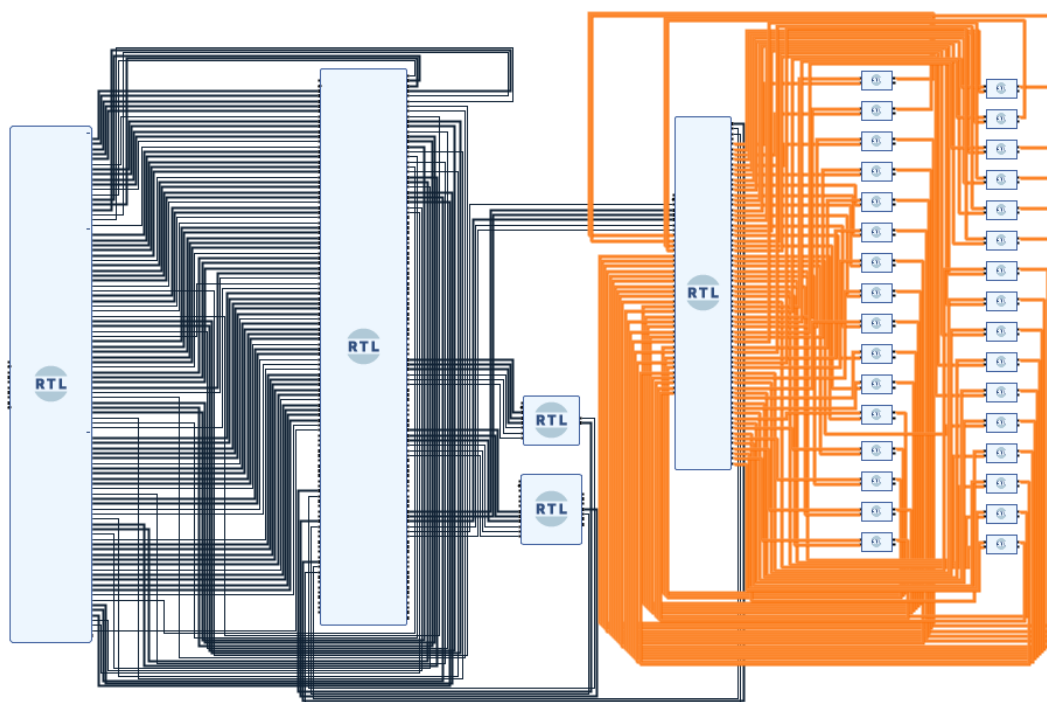


Figura 40. Todos os módulos GPIO Bidirec ligados ao módulo gpio_wrapper

Agora que já ligou todas as ligações internas entre os módulos, faça as ligções externas.

Etapas 5. Faça as ligações externas para os pinos I/OP

Agora ligue os pinos que entram ou saem do Block Design como uma entrada ou saída do Block Design. Ligue esses pinos como pinos/portos externos. Esses pinos externos incluem os pinos de **RAM** (DDR), **CLK** (Clock), **RST** (Reset) e **DMI** (Debug Module interface).

Comece ligando o pino "**clk**". Vá até o módulo "**swerv_wrapper_verilog**", clique com o botão direito do rato no pino "**clk**" e verá um menu *drop-down* (ver Figura 41). Selecione a opção **Make External** entre todas as opções do menu *drop-down*. Também pode clicar com o botão esquerdo do rato no pino e usar a tecla de atalho "**CTRL + T**" para tornar o pino externo.

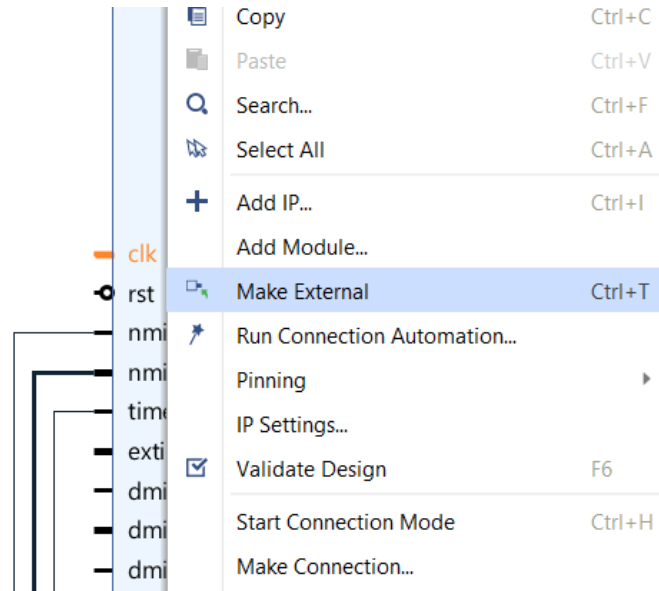


Figura 41. Tornar "clk" uma ligação externa

Agora verá o pino "**clk**" do "**swerv_wrapper_verilog**" ligado a um pino externo "**clk_0**" (ver Figura 42).

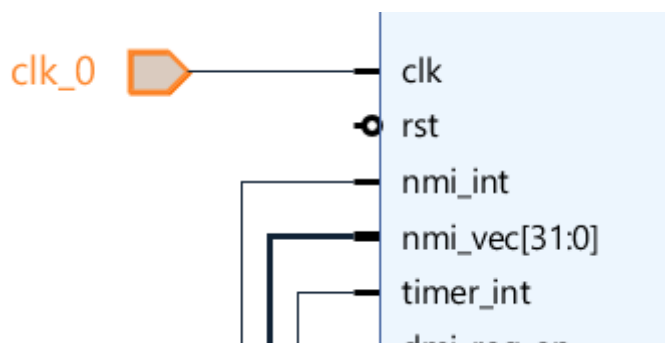


Figura 42. "clk" torna-se uma ligação externa

Agora pode ligar o pino externo "**clk**" aos restantes módulos, incluindo o **intcon_wrapper_bd**, o **syscon_wrapper**, o **bootrom_wrapper** e o **gpio_wrapper**.

PDF: PDFs de alta qualidade do Block Design, mostrando detalhes de close-up das ligações, estão disponíveis aqui: [\[RVfpgaSoCPath\]/RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs/ExternalConnections/1_Clock.pdf](#)

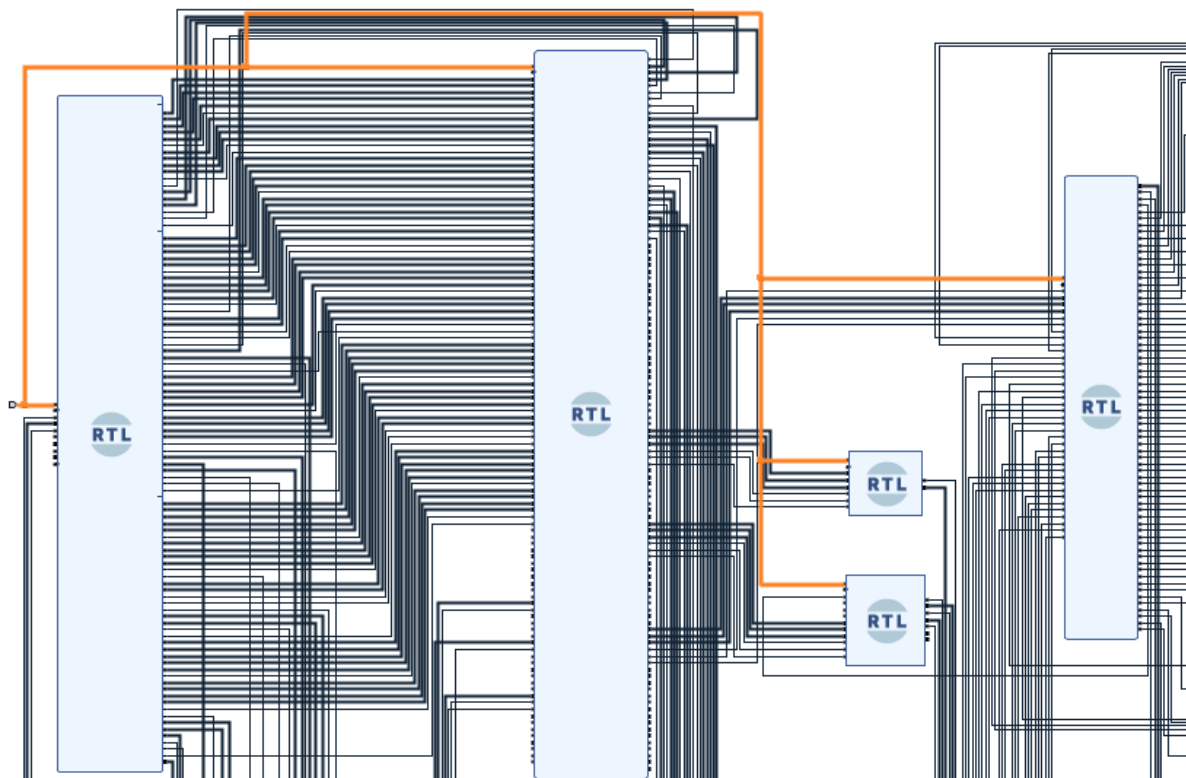


Figura 43. Sinal clk ligado a todos os módulos

Da mesma forma, ligue o pino "rst" a todos os módulos.

Assim como o pino externo que criado para o "clk", crie um para o "rst". Agora, novamente, vá para o módulo "swerv_wrapper_verilog", clique com o botão direito do rato no pino "rst" e torne-o externo.

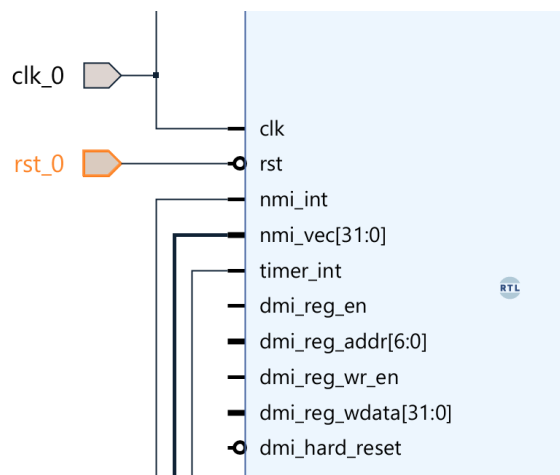


Figura 44. Tornar rst_l como um pino externo

Agora, ligue o pino externo "rst_0" aos restantes módulos, incluindo o intcon_wrapper, o syscon_wrapper, o bootrom_wrapper e o gpio_wrapper.

PDF: PDFs de alta qualidade do Block Design, mostrando detalhes de close-up das ligações, estão disponíveis aqui: [\[RVfpgaSoCPath\]/RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs/ExternalConnections/2_Reset.pdf](#)

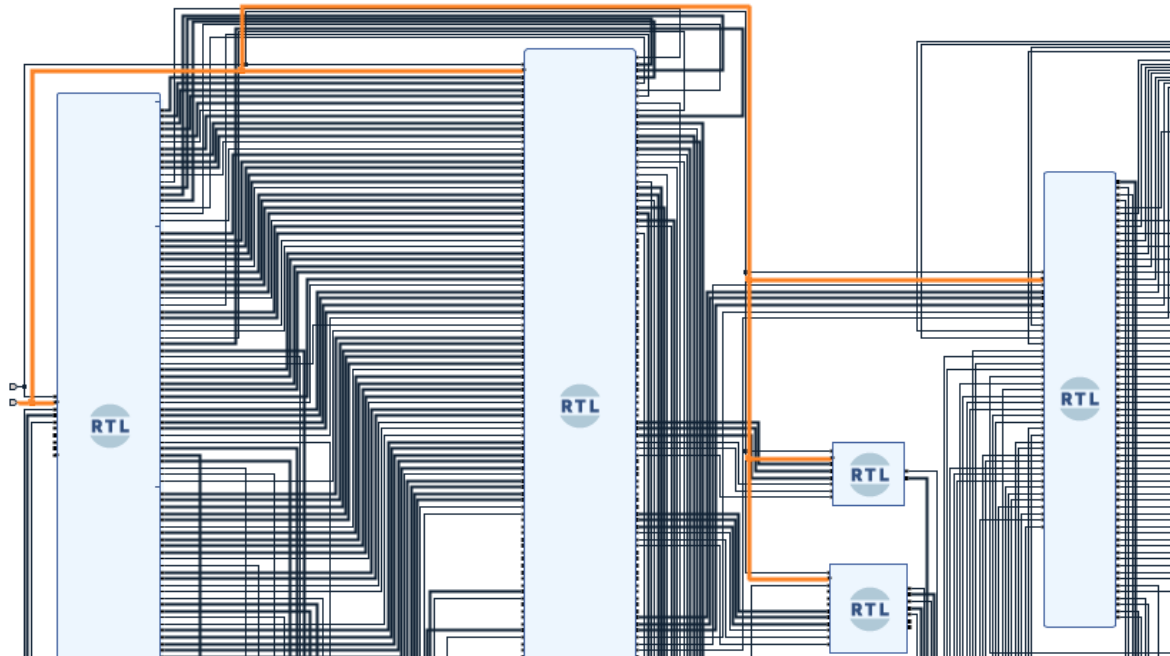


Figura 45. Ligue o pino "rst_0" invertido com o restante dos módulos

Agora, ligue todos os pinos da RAM (DDR) do módulo "Intcon_wrapper_bd" aos pinos da RAM externa seguindo as etapas a seguir.

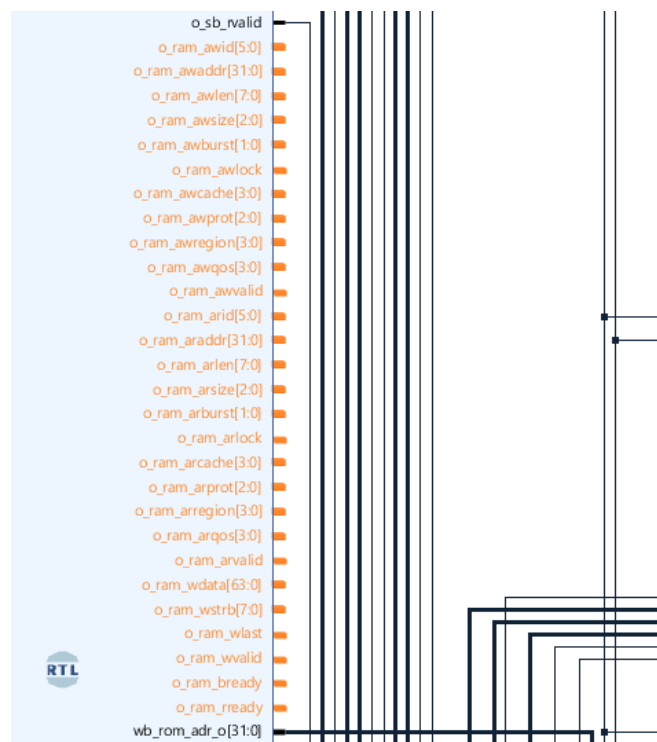


Figura 46. Pinos de RAM do lado direito do wrapper Intcon

Agora torne todos os pinos de RAM do lado direito do módulo "Intcon_wrapper_bd" em pinos externos (Figura 47).

PDF: PDFs de alta qualidade do Block Design, mostrando detalhes de close-up das ligações, estão disponíveis em: [RVfpgaSoCPath] / RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs/ExternalConnections/3_RAM_R.pdf

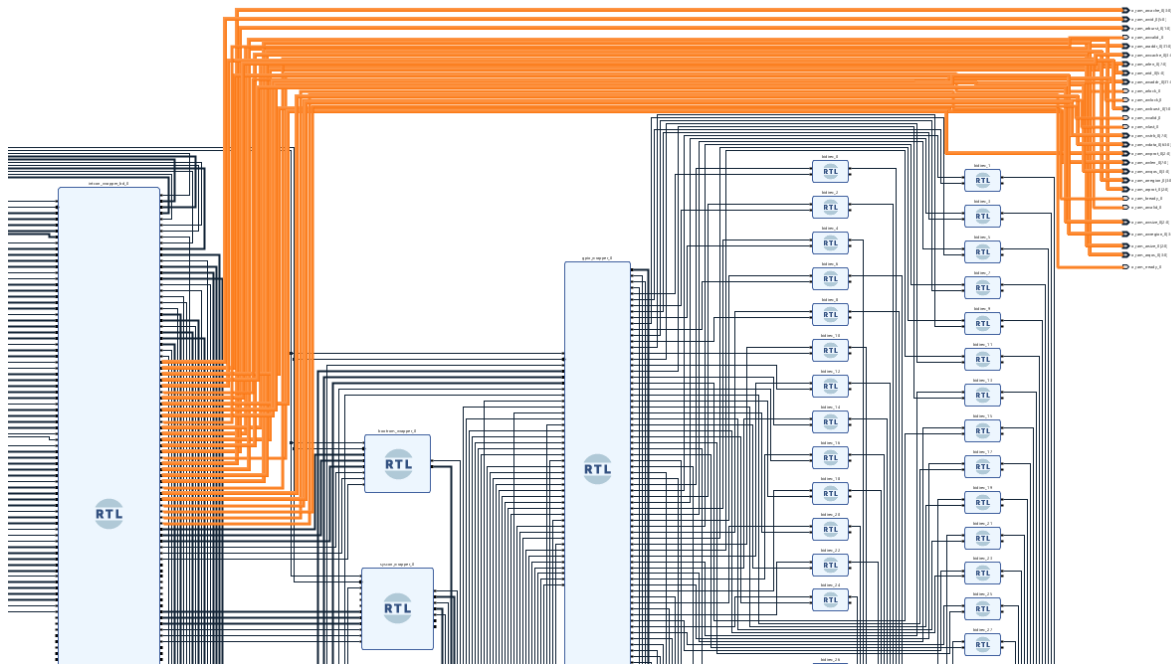


Figura 47. Faça todos os pinos da RAM do lado direito serem External

Transforme os pinos de RAM do lado esquerdo do "Intcon_wrapper_bd" em pinos externos.

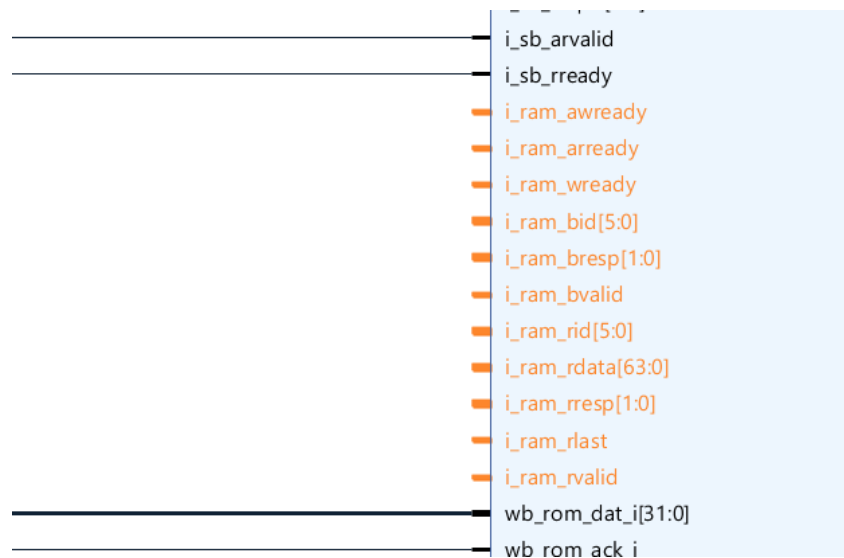


Figura 48. Pinos de RAM do lado esquerdo do wrapper de interconexão

Fará com que todos esses pinos de RAM sejam pinos externos (Figura 49).

PDF: PDFs de alta qualidade do Block Design, mostrando detalhes de close-up das ligações, estão disponíveis em: [RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs/ExternalConnections/4_RAM_L.pdf

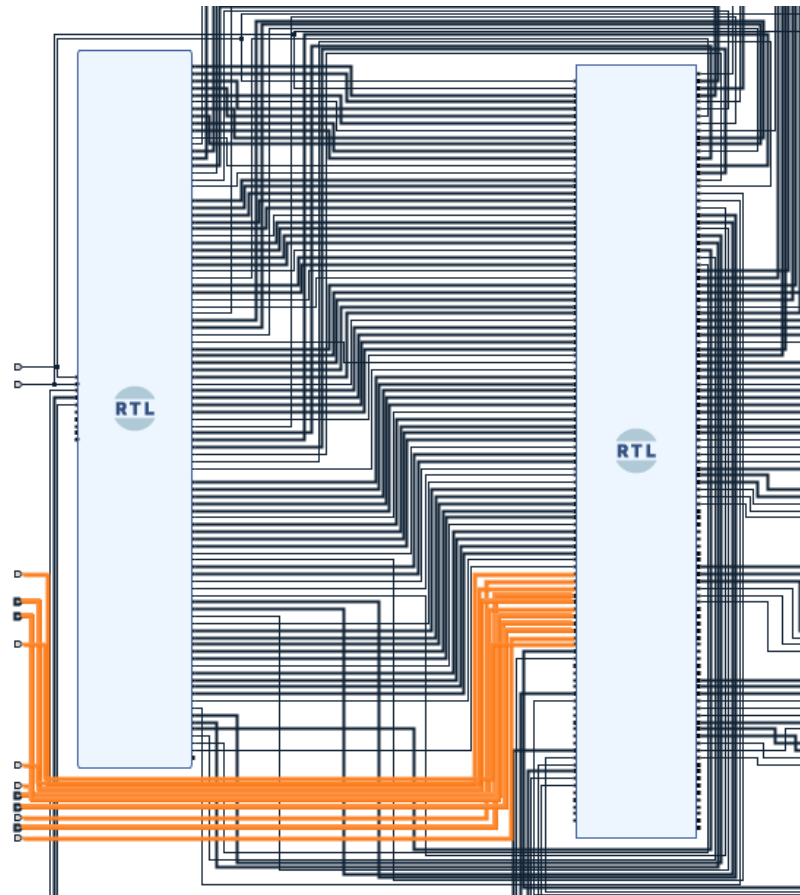


Figura 49. Faça todos os pinos da RAM do lado esquerdo como External

Ligue os pinos **DMI** do módulo "**swerv_wrapper_verilog**" com os pinos externos.

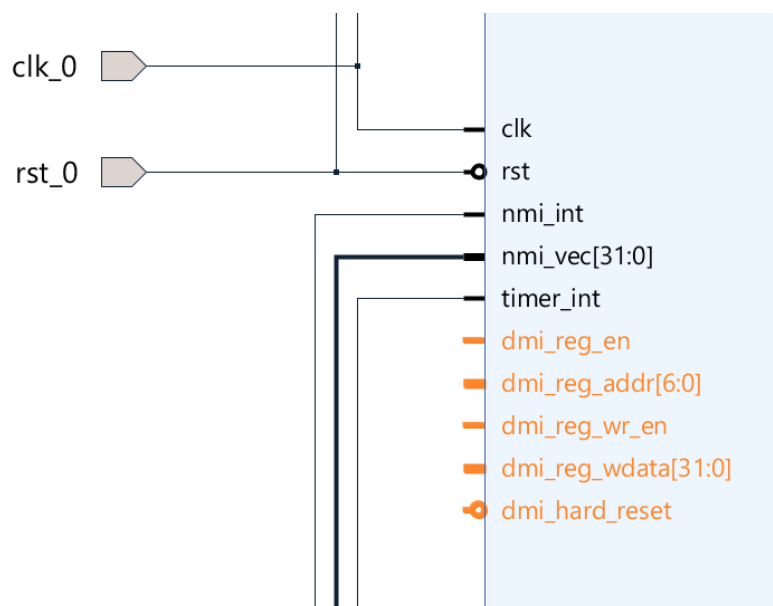


Figura 50. Pinos dmi do swerv_wrapper_verilog (lado esquerdo)

PDF: PDFs de alta qualidade do Block Design, mostrando detalhes de close-up das ligações, estão disponíveis em: [RVfpgaSoCPath] /RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs/ExternalConnections/5_DMI.pdf

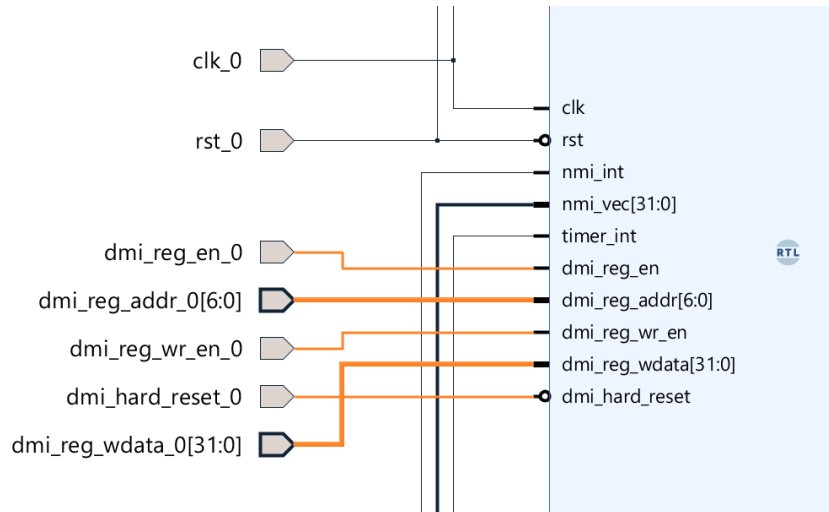


Figura 51. Definição dos pinos dmi como pinos externos

Ligue o pino externo na parte inferior direita do módulo "swerv_wrapper_verilog". Esse pino é "dmi_reg_rdata[31:0]".

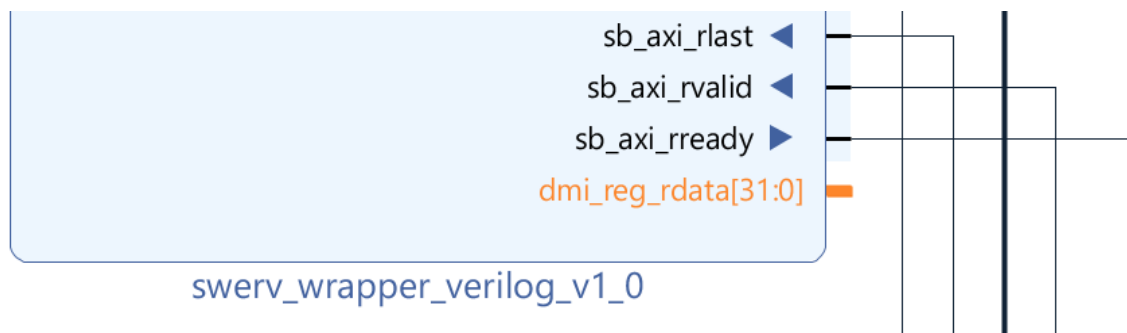


Figura 52. Pino "dmi_reg_rdata[31:0]" (lado direito do swerv_wrapper_verilog)

Faça com que "dmi_reg_rdata[31:0]" também seja um pino externo.

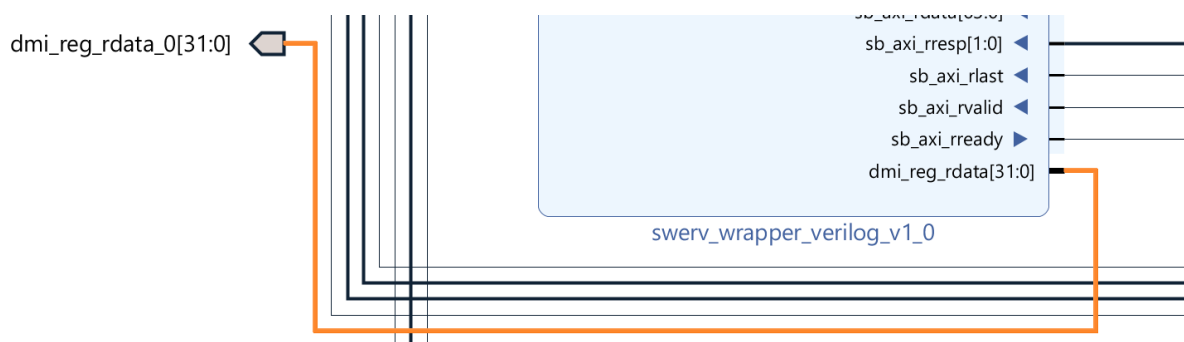


Figura 53. Torne o pino "dmi_reg_rdata[31:0]" como um pino externo

Torne os seguintes pinos do módulo "syscon_wrapper" como pinos externos.

- i_ram_init_done

- i_ram_init_error
- AN[7:0]
- Digital_Bits[6:0]

PDF: PDFs de alta qualidade do Block Design, mostrando detalhes de close-up das ligações, estão disponíveis em: [RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs/ExternalConnections/6_SysconW_External.pdf

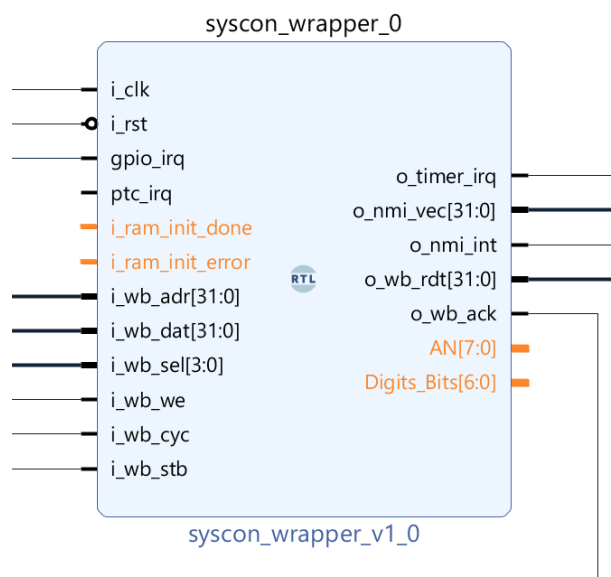


Figura 54. Pinos externos do syscon_wrapper

A última conexão que resta é tornar todos os pinos "bidir" de todos os módulos "bidirec" como pinos externos.

Nota: Faça essas conexões externas uma a uma, começando pelo módulo "bidirec_0" portanto, o pino "bidir" do módulo "bidirec_0" será conectado a um pino externo "bidir_0". Em seguida, vá para o pino "bidir" de "bidirec_1", e assim por diante.

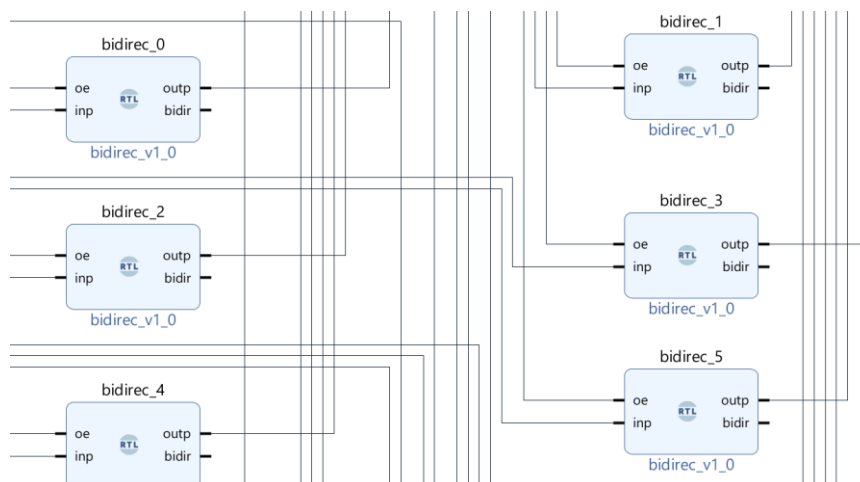


Figura 55. Torne os pinos "bidir" dos módulos GPIO Bidirec como pinos externos

PDF: PDFs de alta qualidade do Block Design, mostrando detalhes de close-up das ligações, estão disponíveis em: [RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs/ExternalConnections/7_Bidir.pdf

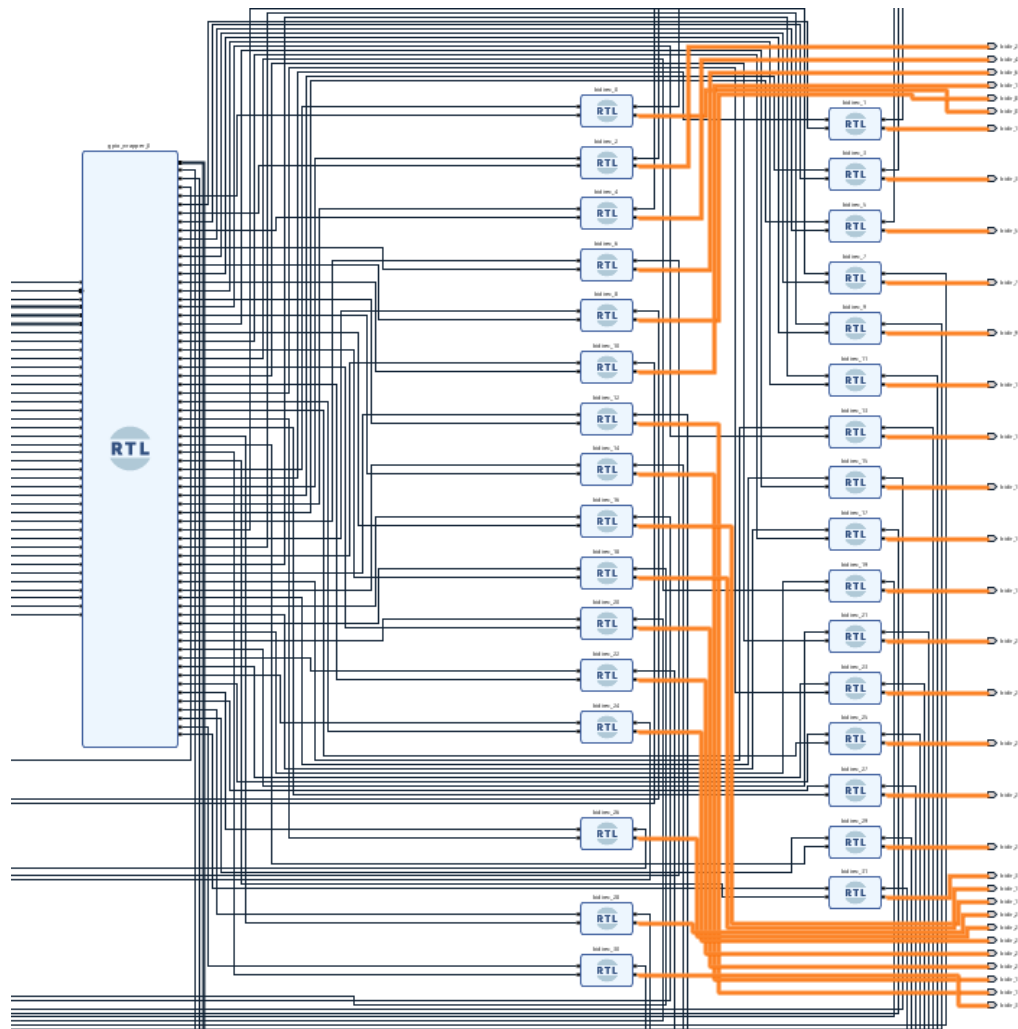


Figura 56. Faça as ligações externas "bidir"

Agora, conclua todas as ligações internas e externas do SoC do Block Design. Pressione **"CTRL + S"** para salvar o Block Design.

O Block Design, que é modelado com base no SoC SweRVofX, foi concluído e agora contém os seguintes módulos ligados:

- | | |
|----------------------------|-------------------------|
| • 1 SweRV Core | (swerv_wrapper_verilog) |
| • 1 Interconnect Wrapper | (intcon_wrapper_bd) |
| • 1 Boot-ROM | (bootrom_wrapper) |
| • 1 GPIO Top Module | (gpio_wrapper) |
| • 1 Controlador de sistema | (syscon_wrapper) |
| • 32 Bidirec Gpio Module | (bidirec) |

(ver Figura 57).

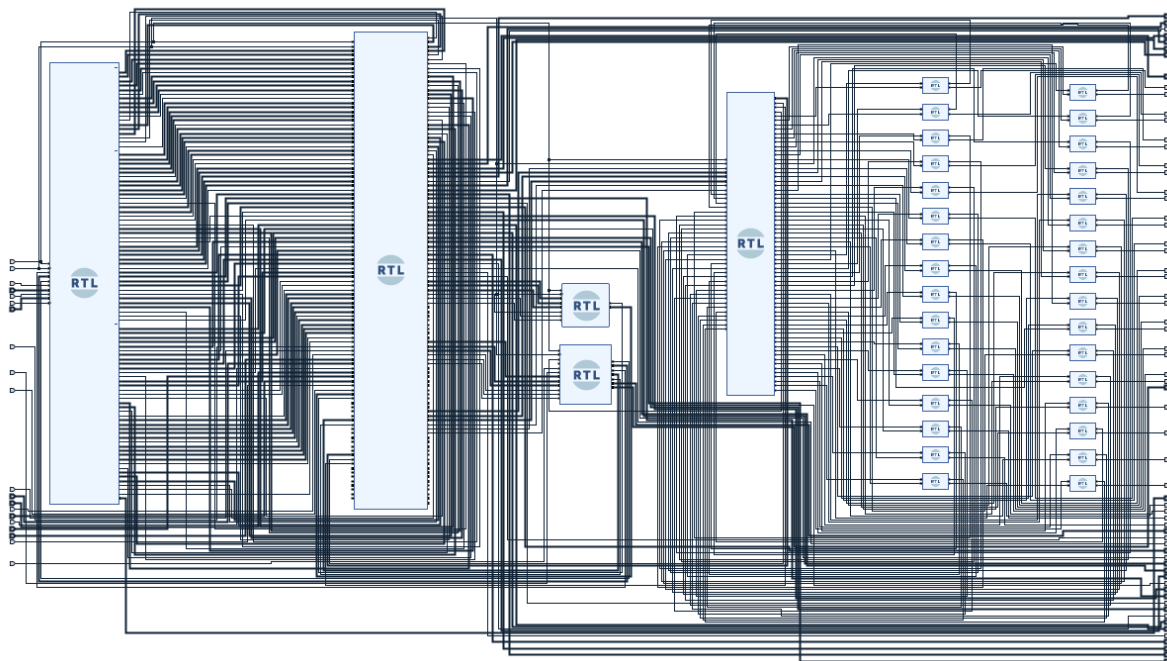


Figura 57. Block Design do SoC concluído

5. Geração do Ficheiro Verilog do Módulo de Block Design

Agora, gere o ficheiro do módulo Verilog do Block Design que criou.

Etapa 1. Navegue até o painel de fontes e localize o módulo de design de bloco "BD" que acabou de criar.

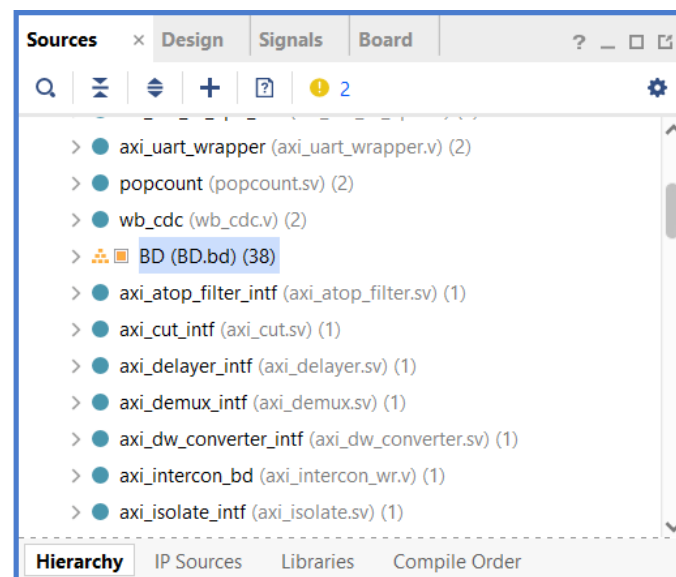


Figura 58. Localizar "BD" em Fontes

Etapa 2. Clique com o botão direito do rato no Block Design (BD) e selecione "Create HDL Wrapper" (Figura 59).

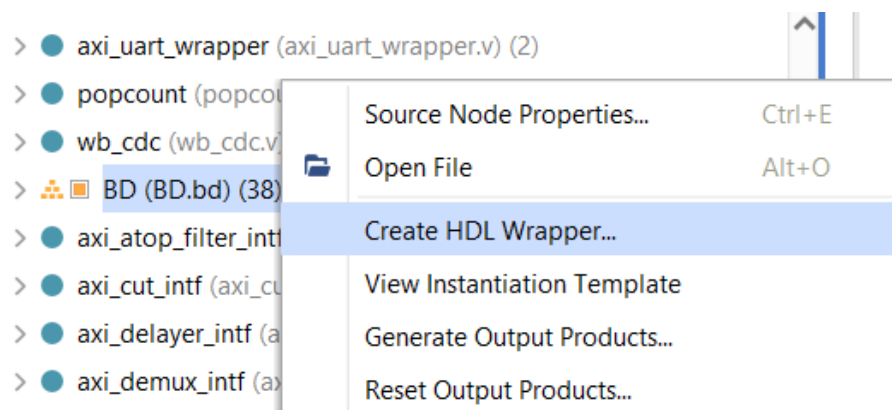


Figura 59. Criar Wrapper HDL

Passo 3. Selecione a opção " **Let Vivado manage wrapper and auto-update** " e clique em OK para continuar.

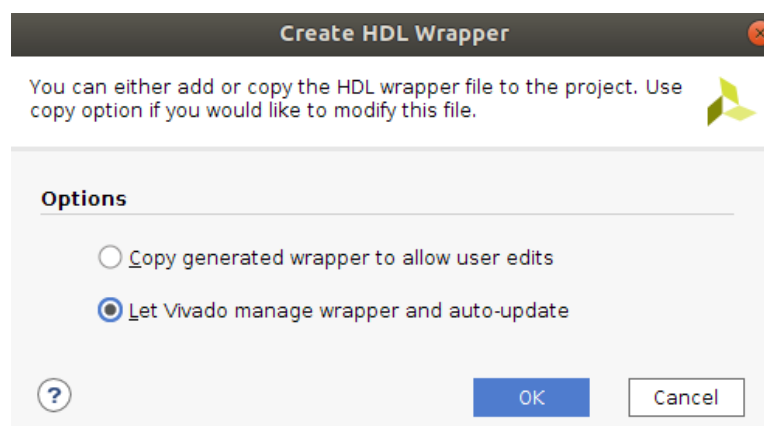


Figura 60. Selecione a segunda opção

Irá ver um pop-up de avisos críticos porque ficaram vários pinos do Block Design desligados, de modo que esses pinos serão automaticamente ligados a "0" (terra).

Clique em OK.

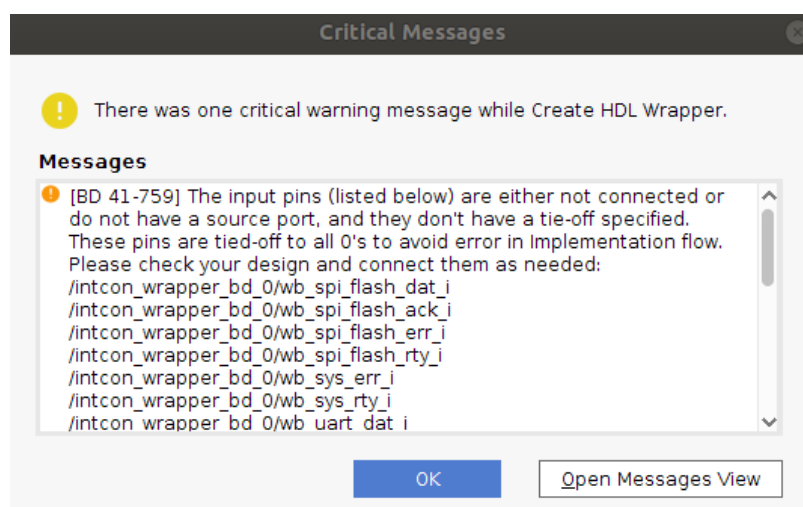


Figura 61. Pop-up de aviso

Agora, o encapsulamento HDL do Block Design foi criado. Pode navegar até o painel Sources e ir para baixo até ver "**BD_wrapper**". Clique no ícone suspenso próximo a ele e depois novamente em "**BD_i**". Abra o ficheiro "**BD (BD.v)**" clicando duas vezes nele (Figura 62).

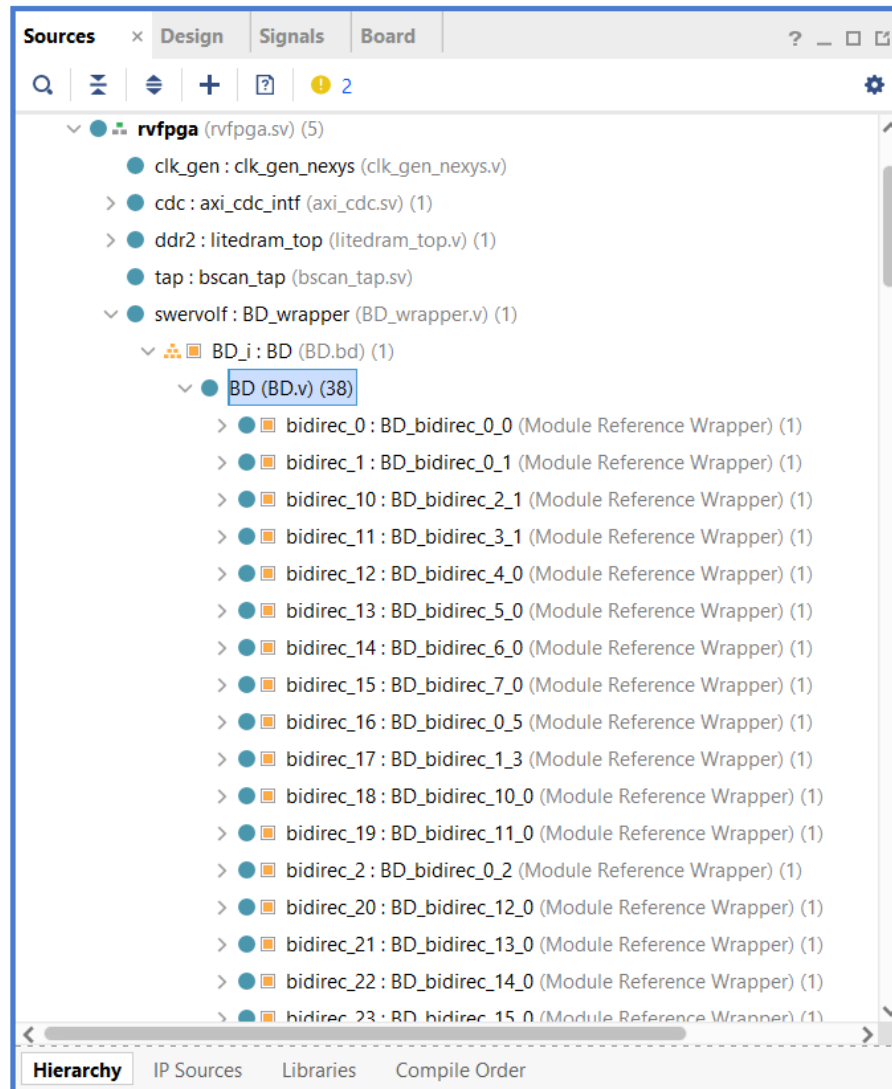
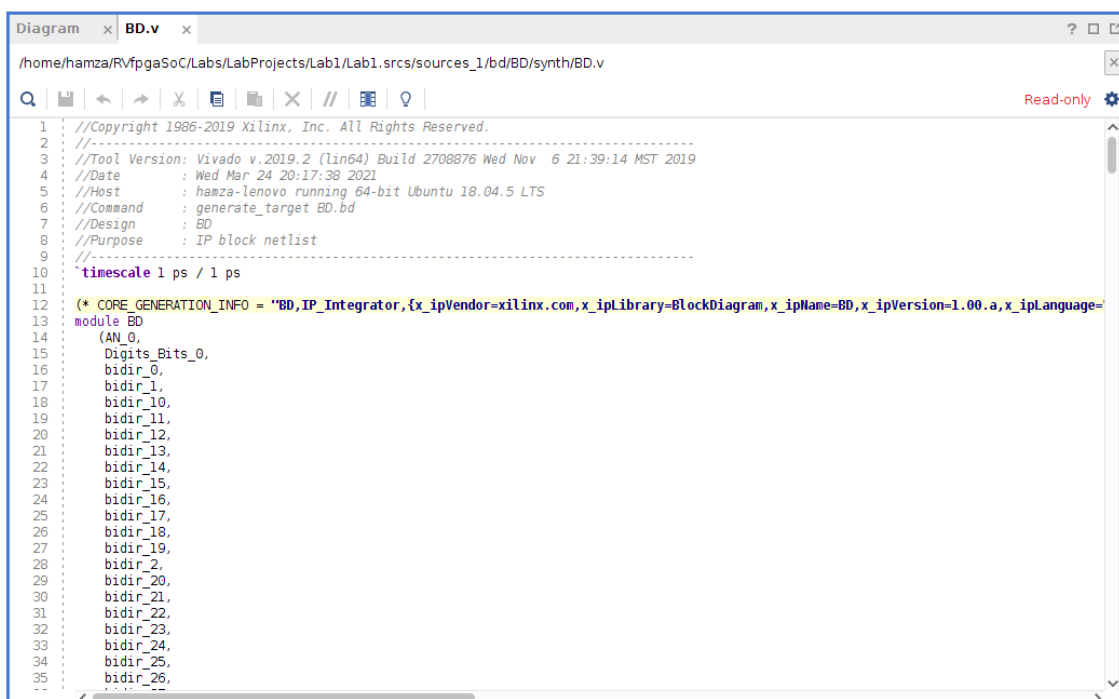


Figura 62. Localize "BD.v" no painel de fontes

Aqui vê o ficheiro Verilog "**BD.v**" que foi criado usando a ferramenta Block Design do Vivado.



```

1 //Copyright 1986-2019 Xilinx, Inc. All Rights Reserved.
2 //
3 //Tool Version: Vivado v.2019.2 (lin64) Build 2708876 Wed Nov 6 21:39:14 MST 2019
4 //Date      : Wed Mar 24 20:17:38 2021
5 //Host      : hamza-lenovo running 64-bit Ubuntu 18.04.5 LTS
6 //Command   : generate_target BD.bd
7 //Design    : BD
8 //Purpose   : IP block netlist
9 //
10 timescale 1 ps / 1 ps
11
12 (* CORE_GENERATION_INFO = "BD,IP_Integrator,{x_ipVendor=xilinx.com,x_ipLibrary=BlockDiagram,x_ipName=BD,x_ipVersion=1.00.a,x_ipLanguage="
13 module BD
14 (AN 0,
15  Digits_Bits_0,
16  bidir_0,
17  bidir_1,
18  bidir_10,
19  bidir_11,
20  bidir_12,
21  bidir_13,
22  bidir_14,
23  bidir_15,
24  bidir_16,
25  bidir_17,
26  bidir_18,
27  bidir_19,
28  bidir_2,
29  bidir_20,
30  bidir_21,
31  bidir_22,
32  bidir_23,
33  bidir_24,
34  bidir_25,
35  bidir_26,
36  )

```

Figura 63. "BD.v"

Pode ver o caminho desse ficheiro recém-criado na parte superior do ficheiro. No próximo laboratório, use esse caminho para aceder ao ficheiro "BD.v".

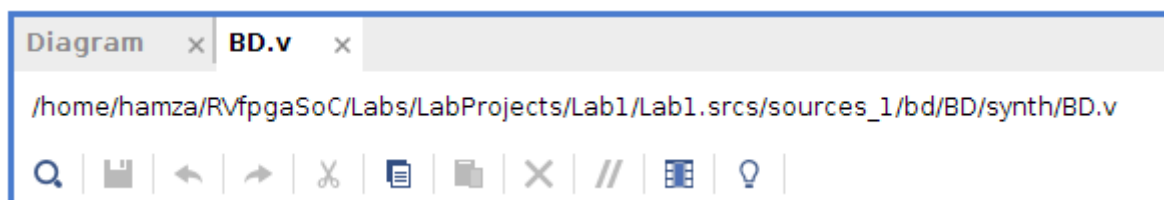


Figura 64. Caminho do ficheiro "BD.v".

6. Gerar Bitstream

Agora que criou o SweRVolfX reduzido use a ferramenta Block Design do Vivado para gerar o wrapper Verilog. Pode gerar o Bitstream que usará para configurar a FPGA. Para gerar o Bitstream, primeiro precisa de ajustar algumas configurações no Vivado, concluindo as etapas a seguir.

Etapas 1. Navegue até Configurações.

Vá a "Tools" (Ferramentas) no lado superior esquerdo da barra de navegação do Vivado e selecione "Settings" (Configurações) nas opções.

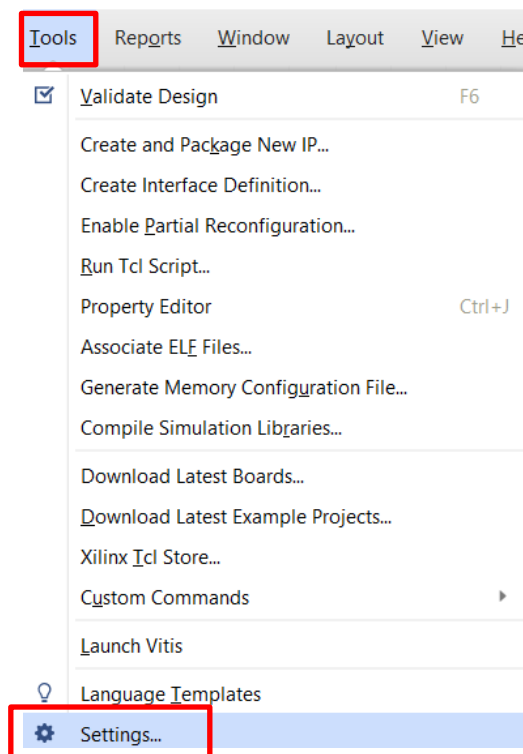


Figura 65. Tools → Configurações

Etapa 2. Navegue até à página General

Vá para a página "General" e seleccione "Verilog Options" na seção de opções de linguagem.

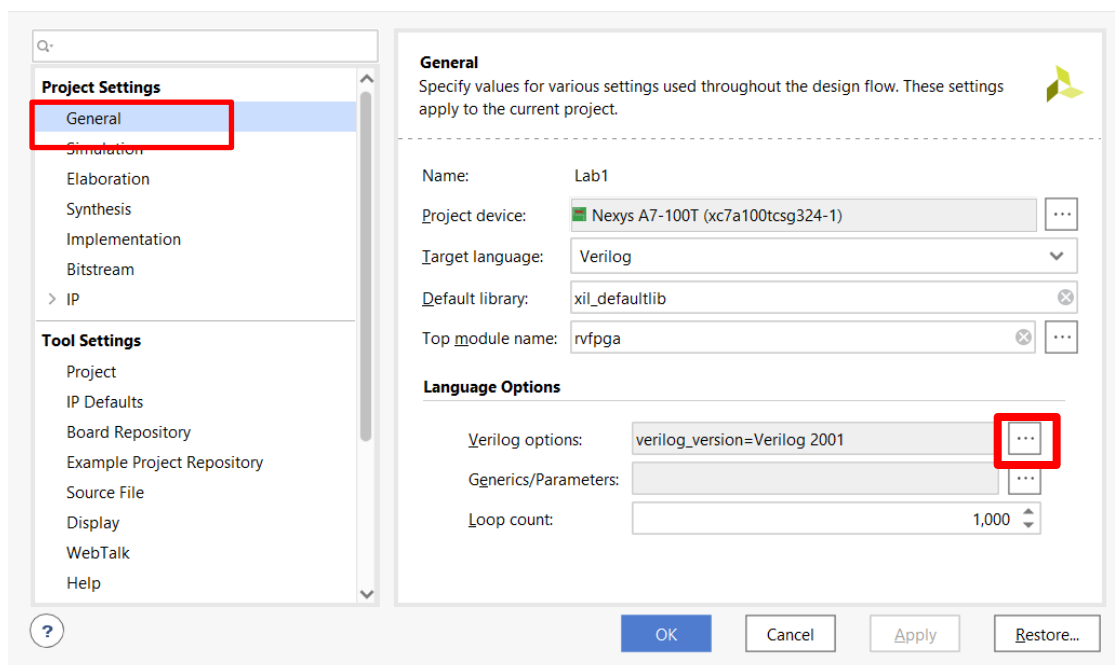


Figura 66. Configurações gerais

Etapa 3. Adicione o caminho para os ficheiros a incluir.

Clique no botão "+" para adicionar o caminho de pesquisa Verilog **Include Files**.

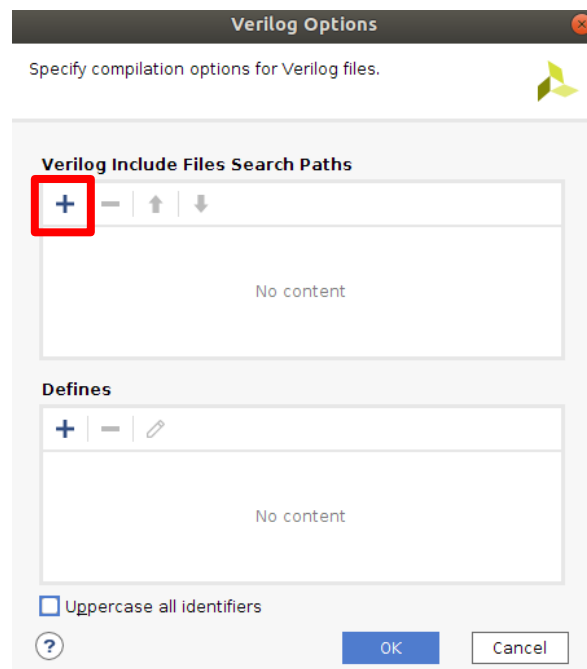


Figura 67. Opções de Verilog

Agora, adicione os três caminhos seguintes:

- [RVfpgaSoCPath]/RvfpgaSoC/Labs/LabProjects/Lab1/Lab1.srcs/sources_1/imports/src/SweRVolfSoC/Interconnect/AxiInterconnect/pulp-platform.org__axi_0.25.0/include
- [RVfpgaSoCPath]/RvfpgaSoC/Labs/LabProjects/Lab1/Lab1.srcs/sources_1/imports/src/OtherSources/pulp-platform.org__common_cells_1.20.0/include
- [RVfpgaSoCPath]/RvfpgaSoC/Labs/LabProjects/Lab1/Lab1.srcs/sources_1/imports/src/SweRVolfSoC/SweRVeh1CoreComplex/include



Figura 68. Caminhos dos ficheiros Verilog a incluir

Clique em OK.

Etapas 4. Navegue até à página Bitstream

Vá para a guia "Bitstream" e clique no botão "tcl.pre".

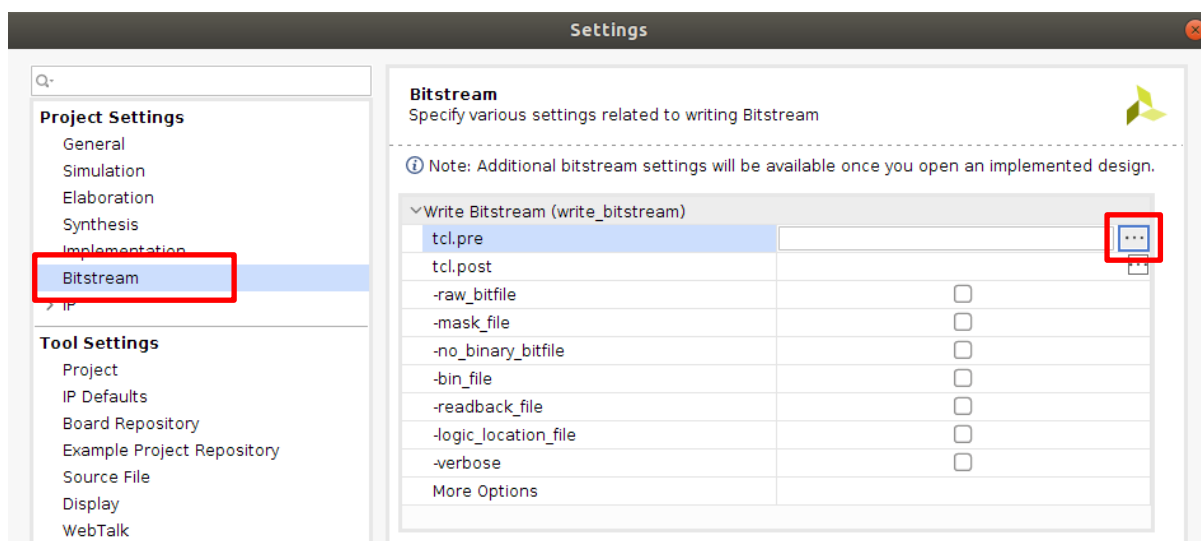


Figura 69. Configuração do Bitstream

Selecione a opção "New script" (Novo script).

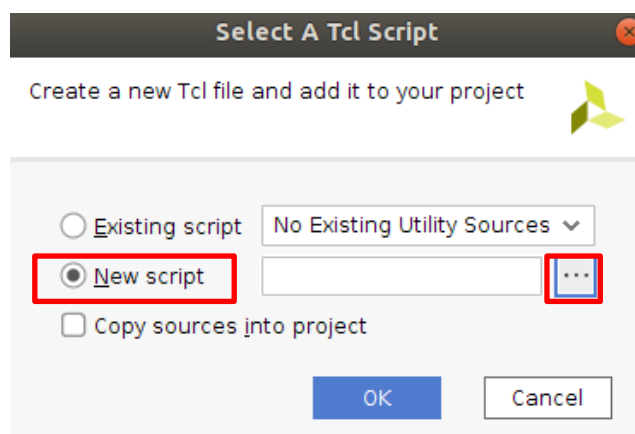


Figura 70. Novo script Tcl

Navegue até o seguinte caminho e selecione o ficheiro "script.tcl". (ver Figura 71)
[RVfpgaSoCPath] /RVfpgaSoC/Labs/LabResources/Lab1/script.tcl

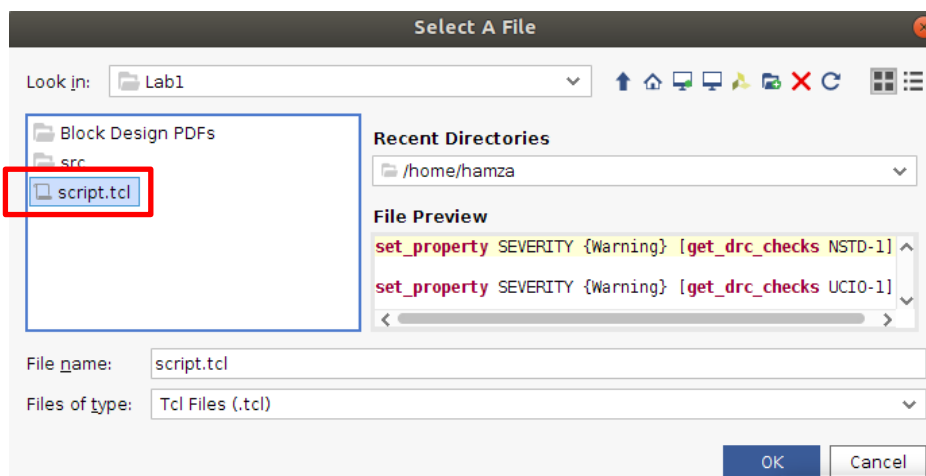


Figura 72. Importar ficheiro "script.tcl"

Clique em OK e aplique as alterações.

Etapa 4. Gerar Bitstream.

Agora, clique em Flow → Generate Bitstream, conforme mostrado na Figura 73.

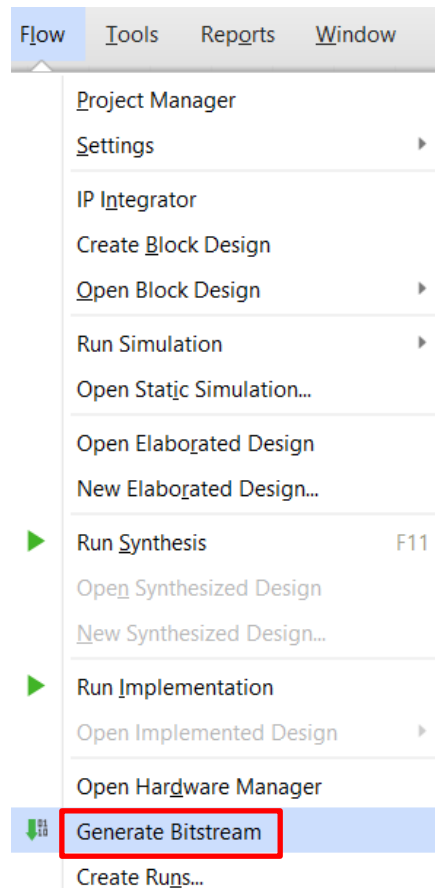


Figura 73. Gerar Bitstream

Poderá aparecer uma janela informando que não há resultados de implementação disponíveis e pedindo para iniciar a síntese e a implementação.

Clique em Yes (Figura 74).

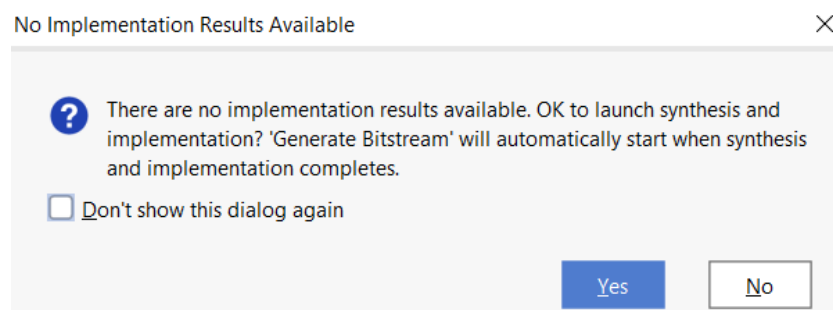


Figura 74. Janela Iniciar síntese e implementação

A janela **Launch Runs** será exibida na tela (Figura 75). Clique em OK.

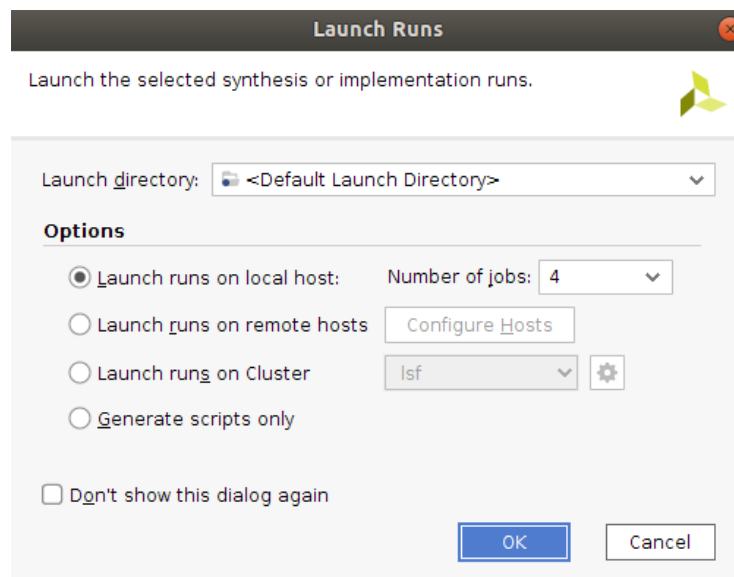


Figura 75. Execuções de lançamento

Agora veremos uma lista de avisos que nos informam que os pinos que deixamos desconectados serão automaticamente conectados a "0". Clicaremos em OK. (consulte a Figura 76).

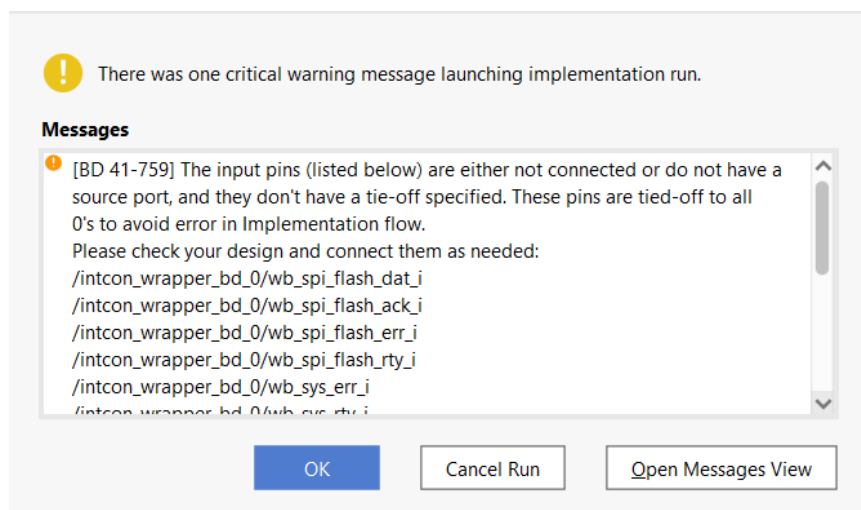


Figura 76. Mensagens de aviso das execuções de inicialização

Essa etapa sintetiza o **RVfpgaNexys** (conforme definido pelos ficheiros Verilog e SystemVerilog no projeto), mapeia-o na FPGA e cria o Bitstream.

Tcl Console Messages Log Reports Design Runs																
Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	DSP	Start	Elapsed
synth_1 (active)	constrs_1	Queued...														00:00:00
impl_1	constrs_1	Queued...														00:00:00
Out-of-Context Module Runs																
BD		Running Submodule Runs													3/3/21, 2:55 PM	00:01:50
BD_bidirec_7	BD_bidirec_7	Running synth_design...													3/3/21, 2:55 PM	00:01:50
BD_bidirec_5	BD_bidirec_5	Running synth_design...													3/3/21, 2:55 PM	00:01:50
BD_bidirec_8	BD_bidirec_8	Running synth_design...													3/3/21, 2:55 PM	00:01:50
BD_bidirec_4	BD_bidirec_4	Running synth_design...													3/3/21, 2:55 PM	00:01:50
BD_bidirec_2	BD_bidirec_2	Running synth_design...													3/3/21, 2:55 PM	00:01:50
BD_bidirec_3	BD_bidirec_3	Running synth_design...													3/3/21, 2:55 PM	00:01:50
BD_bidirec_9	BD_bidirec_9	Running synth_design...													3/3/21, 2:55 PM	00:01:50
BD_bidirec_5	BD_bidirec_5	Running synth_design...													3/3/21, 2:55 PM	00:01:50
BD_bidirec_7	BD_bidirec_7	Queued...														00:00:00
BD_bidirec_0	BD_bidirec_0	Queued...														00:00:00
BD_bidirec_3	BD_bidirec_3	Queued...														00:00:00
BD_bidirec_2	BD_bidirec_2	Queued...														00:00:00

Figura 77. Execuções de projeto

Observação: Se receber o erro: Gtk-Message: Failed to load module "canberra-gtk-module"

Instale um pacote com o seguinte comando para resolver o problema.

```
sudo apt install libcanberra-gtk-module libcanberra-gtk3-module
```

Se estiver usando uma Virtual Machine (VM), o Vivado poderá parar durante a síntese devido à baixa alocação de RAM. Recomenda-se alocar mais RAM para a VM se o Vivado parar.

Esse processo pode levar vários minutos, dependendo da velocidade do computador.

Tcl Console Messages Log Reports Design Runs																	
Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	DSP	Start	Elapsed	Run Strateg
synth_1 (active)	constrs_1	synth_design Complete!								3368	3331	13.0	0	0	5/4/21, 3:22 PM	00:02:13	Vivado Syn
impl_1	constrs_1	write_bitstream Complete!	0.327	0.000	0.050	0.000	0.000	0.934	0	33637	18546	44.0	0	4	5/4/21, 3:25 PM	00:12:32	Vivado Imp
Out-of-Context Module Runs																	
BD		Submodule Runs Complete													5/4/21, 3:03 PM	00:19:17	

Figura 78. Caminho dos ficheiros Verilog a incluir

Depois do Bitstream tiver sido gerado, uma janela será exibida, conforme mostrado na Figura 79. Clique no botão X no canto superior direito para fechar a janela.

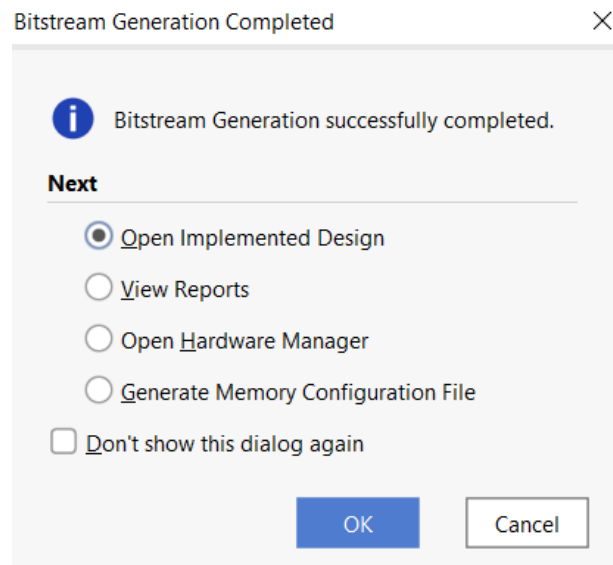


Figura 79. Geração de Bitstream concluída

Agora que o Bitstream foi criado, no próximo laboratório, irá ver como carregar esse bitstream numa placa Nexys A7 via PlatformIO e, em seguida, irá ver como executar programas de exemplo no SweRVolfX reduzido que acabou de criar neste laboratório.