

**THE IMAGINATION UNIVERSITY PROGRAMME**

**RVfpga-SoC Lab 3**

**Introdução ao SweRVolf**

**e ao FuseSoC**

**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. |

# Introdução

Neste laboratório, analisaremos o SweRVolf em detalhe. Também apresentaremos o FuseSoC e mostraremos como usá-lo para construir o SweRVolf ([https://github.com/chipsalliance/Cores-SweRVolf/](https://github.com/chipsalliance/Cores-SweRVolf/releases/tag/v0.7)).

No Lab 1, os módulos individuais foram escritos em Verilog/SystemVerilog. Em seguida, ligou esses módulos para criar um SweRVolfX reduzido usando a ferramenta de Block Design do Vivado. Na indústria, os projetistas usam métodos gráficos de ligação de módulos ou código em Verilog/SystemVerilog.

A tabela a seguir fornece um breve resumo das vantagens e desvantagens da abordagem Block Design.

**Tabela 2. Prós e contras do Block Design**

|  |  |
| --- | --- |
| **Vantagens do Block Design** | **Desvantagens do Block Design** |
| Fácil de entender | O trabalho em equipa pode ser um desafio |
| Curva de aprendizagem reduzida | É necessária ferramenta específica (Vivado Block Design) |
| Pode combinar módulos complexos num de forma intuitiva | As atualizações podem corromper o sistema |
|  | Pode ter dificuldades para ir além dos FPGAs e chegar a chips SoC reais |
|  | O código gerado por máquina pode ser difícil de ler e entender. |
|  | O ajuste fino do SoC para fabricação ou para obter mais desempenho deve ser feito no nível Verilog/SystemVerilog. |
|  | Os fornecedores de IP Cores fornecem o código-fonte Verilog, mas não necessariamente num formato que funcione com uma ferramenta específica do Vivado Block Design. |

A tabela a seguir fornece um breve resumo das vantagens e desvantagens da abordagem de código ajustado manualmente.

**Tabela 3. Prós e contras do código ajustado manualmente**

|  |  |
| --- | --- |
| **Vantagens do Código Ajustado Manualmente** | **Desvantagens do Código ajustado manualmente** |
| Pode ser ajustado manualmente para ser muito específico para um processo de fabricação de FPGA ou SoC. | Curva de aprendizado acentuada |
| Permite usar sistemas de controle de versões | Requer conhecimento de Verilog/SystemVerilog. |
| Pode trabalhar facilmente em equipas e colaborar (várias pessoas trabalham em módulos diferentes) | Demora um pouco para se familiarizar com a estrutura de todos os ficheiros/pastas/módulos. |

Os módulos do SweRVolf (módulo principal, swervolf\_core, SweRV) foram escritos à mão. O próprio núcleo da CPU do SweRV foi criado pela Western Digital. A interconexão e outros módulos são núcleos IP de vários fornecedores que os fornecem sob licenças de código-aberto. Os módulos principais foram escritos por Olof Kindgren.

É possível ligar todos os diferentes módulos e escrever Verilog manualmente. Mas, com o tempo, isso pode-se tornar tedioso. Especialmente quando equipas maiores trabalham juntas, partilhando algumas partes do código.

Durante o processo de projeto do SoC, várias equipas grandes trabalham juntas simultaneamente. Diferentes equipas concentram-se em diferentes áreas, mas também partilham vários componentes e IP Cores. Por exemplo, a equipa de simulação preocupar-se-á com o mesmo código de IP (SoC+interconexão+CPU+Periférico), mas com encapsulamentos para o simulador de conjunto de instruções ou no Verilator. Enquanto isso, a equipa de FPGA usa o mesmo código IP (SoC+Interconnect+CPU+Peripheral), mas para executar testes em várias placas de desenvolvimento com diferentes periféricos.

Um sistema de compilação facilita o trabalho de diferentes equipas no mesmo código IP (SoC+Interconexão+CPU+Periférico), mas sem que elas se atrapalhem. O sistema de compilação deve ser flexível e dinâmico o suficiente para se adequar a várias equipas e seus requisitos. O SweRVolf foi criado usando um sistema de compilação chamado FuseSoC (pronuncia-se "fuse sock"). O sistema de compilação FuseSoC reúne um projeto de hardware a partir de blocos IP individuais.

Neste laboratório, analisará a criação de versões do SweRVolf no FuseSoC. Irá ver o procedimento passo-a-passo para adicionar a biblioteca "SweRVolf" do FuseSoC e, em seguida, realizá-lo para simulação e implementação na placa. Em seguida, executará programas de exemplo no SweRVolf.

# Requisitos

Para realizar este laboratório, precisará instalar as seguintes ferramentas:

* Vivado 2019.2 Web Pack (consulte o Guia de Instalação (Página nº 04))
* Verilator (V4.106) (consulte o Guia de Instalação (página nº 09))
* GTKWave (consulte o Guia de Instalação (página nº 09))
* FuseSoC (consulte o Guia de instalação (página nº 10))
* OpenOCD (versão RISC-V) (consulte o Guia de Instalação (página nº 10))

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

Por exemplo, se ainda não o fez, instale o Verilator e o Vivado da Xilinx seguindo as instruções do Guia de Instalação do RVfpga-SoC. Certifique-se de ter copiado a pasta RVfpga-SoC que descarregou do Imagination University Programme para o computador.

# O Que é o FuseSoC?

O FuseSoC é um gestor de pacotes premiado e um conjunto de ferramentas de compilação para código HDL (Hardware Description Language). O seu objetivo principal é aumentar a reutilização de IP Cores e criar, construir e simular soluções de SoCs.

Uma entidade fundamental no FuseSoC é o IP core. Os IP Cores podem ser descobertos pelo gestor de pacotes do FuseSoC locais ou remotos e combinados num projeto de hardware completo pelo sistema de compilação.

Um Core FuseSoC não é necessariamente um processador, mas sim uma peça de IP reutilizável e razoavelmente autónoma, como uma implementação de um FIFO. O FuseSoC também se refere a eles como pacotes. Noutros sistemas, essas peças reutilizáveis de hardware são chamadas de módulos.

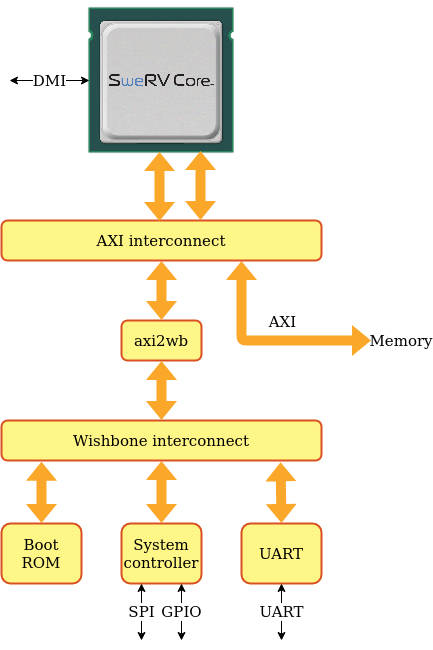
Para obter informações mais detalhadas sobre o FuseSoC, leia a sua documentação em <https://fusesoc.readthedocs.io/en/latest/user/overview.html#understanding-fusesoc>

O FuseSoC fornece o "SweRVolf" (SoC baseado no FuseSoC para o núcleo SweRV RISC-V) como um pacote na sua biblioteca. Pode adicionar a biblioteca "**swervolf**" usando o FuseSoC e, em seguida, construí-la para "sim" (Simulação) ou "Nexys\_a7" (Placa).

# SweRVolf: SoC Baseado no FuseSoC para o SweRV EH1

O SweRVolf é um SoC baseado em FuseSoC para o núcleo SweRV RISC-V. O SweRVolf pode executar os testes de conformidade do RISC-V, o sistema operativo Zephyr ou outro software em simuladores ou em placas FPGA. O FuseSoC tem como objetivo dar suporte e aumentar a portabilidade, a extensibilidade e a facilidade de uso; permitir que os utilizadores do SweRV executem o software rapidamente, modifiquem o SoC de acordo com suas necessidades ou o portem para novos dispositivos hardware.

A Figura 1 mostra um diagrama de blocos de alto nível do SweRVolf.



**Figura 1. Núcleo SweRVolf**

O núcleo do SweRVolf consiste num CPU SweRV com boot ROM, interconexão AXI4, UART, SPI, temporizador RISC-V e GPIO. O núcleo não inclui nenhuma RAM, mas expõe um barramento de memória que o encapsulamento específico do alvo ligará a um controlador de memória apropriado. Outras conexões externas são clock, reset, UART, GPIO, SPI e DMI (Debug Module Interface).

A Tabela 4 fornece os endereços mapeados na memória dos periféricos que estão conectados ao núcleo SweRV EH1 por meio da interconexão Wishbone e da interconexão AXI.

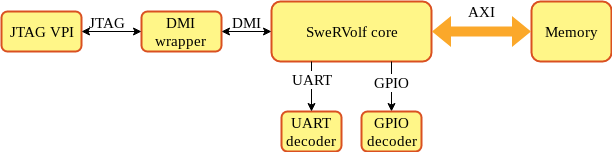
**Tabela 4. Endereços de memória do SweRVolf**

|  |  |
| --- | --- |
| **Sistema** | **Endereço** |
| RAM | 0x00000000-0x07FFFFFF |
| Boot ROM | 0x80000000-0x80000FFF |
| Controlador do sistema | 0x80001000-0x80001FFF |
| UART | 0x80002000-0x80002FFF |
| GPIO | 0x80001010-0x80001013 |

Semelhante aos módulos SystemVerilog RVfpgaSim e RVfpgaNexys no Laboratório 2, o SweRVolf também fornece duas versões equivalentes do SweRVolf: SweRVolf Sim para simulação e SweRVolf Nexys para a placa Digilent Nexys A7.

1. **SweRVolf Sim**

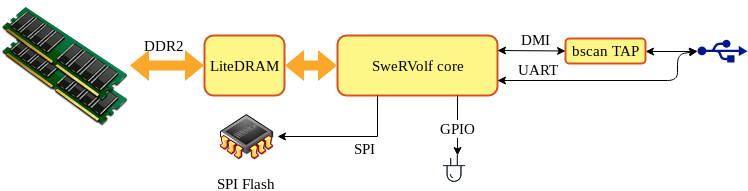
O SweRVolf Sim é destinado a simulação e envolve o IP Core do SweRVolf num testbench a ser usado pelo Verilator ou por outros simuladores orientados a eventos, como o QuestaSim. Ele pode ser usado para simulações de sistema completo que executam programas em um processador SweRV. Ele também suporta a conexão de um depurador por meio do OpenOCD e do JTAG VPI (ver Figura 2).



**Figura 2. Simulação do SweRVolf**

1. **SweRVolf Nexys**

O SweRVolf Nexys é uma versão do SoC SweRVolf para a placa Digilent Nexys A7. Ele usa uma RAM DDR2 de 128 MB integrado na placa, tem um GPIO ligado ao LED, suporta inicialização a partir de SPI Flash e usa o porto micro-USB para comunicação UART e JTAG. O bootloader para o SweRVolf Nexys tentará carregar um programa armazenado na Flash SPI por omissão (ver Figura 3).



**Figura 3. SweRVolf Nexys**

O SweRVolf SoC pode ser executado em simulação ou em hardware, ou seja, na placa Digilent Nexys A7. Em ambos os casos, o FuseSoC pode ser usado para iniciar a simulação ou construir e executar a construção do FPGA.

Tal como o **RVfpgaSim** no laboratório anterior, o **SweRVolf Sim** é destinado a simulação e envolve o IP Core do SweRVolf num testbench a ser usado pelo Verilator. Da mesma forma, o **RVfpgaNexys**,do laboratório anterior, é semelhante ao **SweRVolf Nexys**, que é uma versão do SoC SweRVolf para a placa Digilent Nexys A7.

# Configuração do Ambiente de Desenvolvimento

Nesta seção, mostramos como configurar o ambiente para a compilação do Sim e do Nexys.

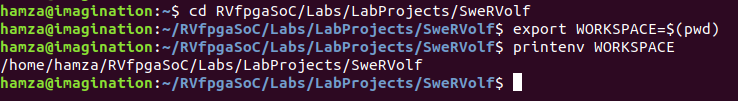
**Etapa 1.** Navegue até o diretório chamado "**SweRVolf**" para usá-lo como raiz do projeto.

De agora em diante, esse diretório será chamado de **$WORKSPACE*.*** Todos os comandos adicionais serão executados a partir de $WORKSPACE, a menos que seja indicado de outra forma. Depois de entrar no diretório do Workspace, execute:

* export WORKSPACE=$(pwd)

Para definir a variável de shell $WORKSPACE (ver Figura 4). Pode usar o seguinte comando para verificar se a variável de shell foi adicionada com êxito.

* printenv WORKSPACE



**Figura 4. Definir o Workspace**

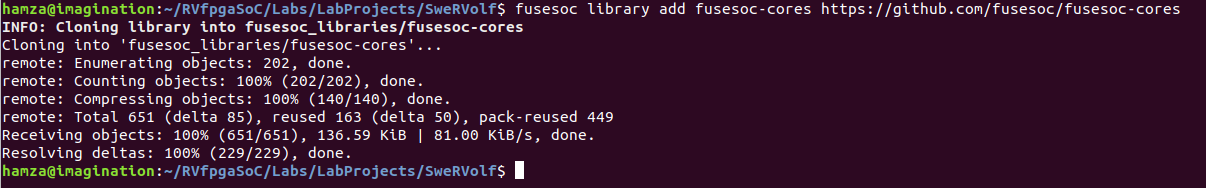
**Etapa 2.** Verifique se já tem o FuseSoC instalado. Se não tiver o FuseSoC

instalado no computador, instale-o executando o seguinte:

* sudo pip3 install fusesoc

**Etapa 3.** Adicione a biblioteca base do FuseSoC ao Workspace:

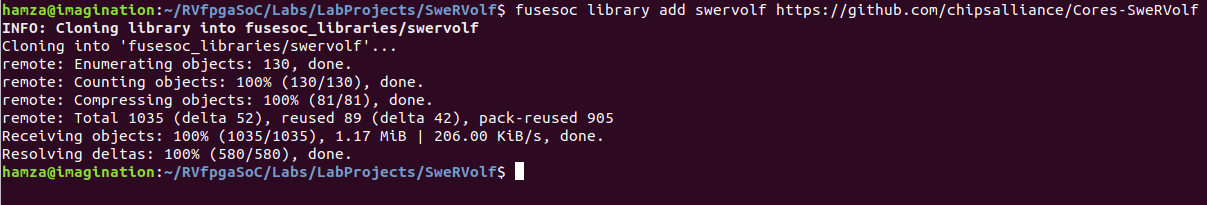
* fusesoc library add fusesoc-cores https://github.com/fusesoc/fusesoc-cores



**Figura 5. Adicionar biblioteca base do FuseSoC**

**Etapa 4.** Adicione a biblioteca swervolf:

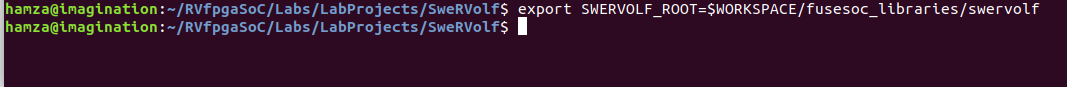
* fusesoc library add swervolf https://github.com/chipsalliance/Cores-SweRVolf

**

**Figura 6. Adicionar a biblioteca Swervolf**

**Etapa 5.** Defina o diretório swervolf como a variável de shell "SWERVOLF\_ROOT":

* export SWERVOLF\_ROOT=$WORKSPACE/fusesoc\_libraries/ swervolf



**Figura 7. Definir variável da shell**

# Compilação Para a Simulação

Nesta seção, irá ver como criar o SweRVolf Sim usando o FuseSoC.

Para selecionar o que será executado, use o comando "**fusesoc run**" com o parâmetro --target.

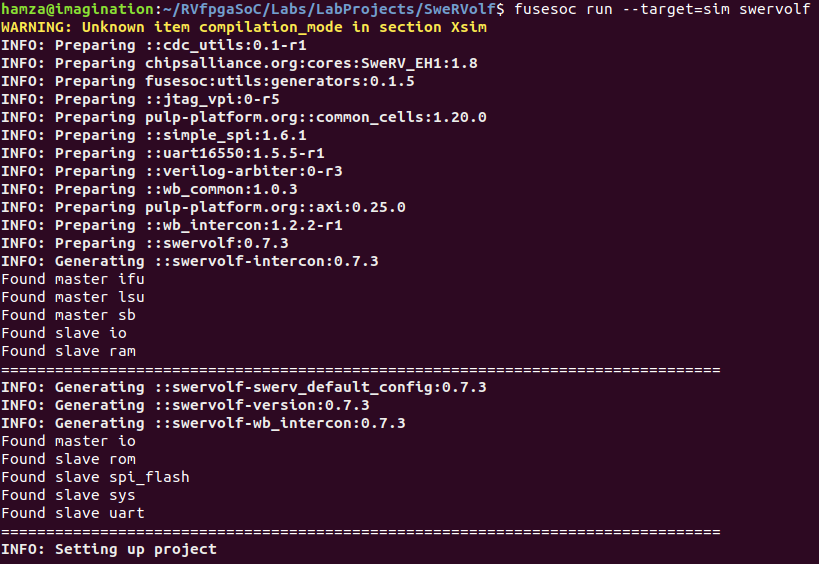
**Etapa 6.** Para executar a simulação, use

* fusesoc run --target=sim swervolf

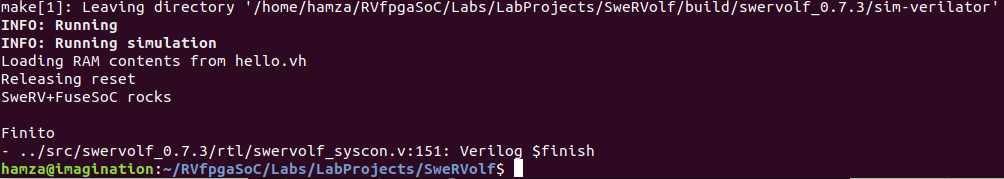
Esse comando gera o binário de simulação chamado "**Vswervolf\_core\_tb**" dentro do seguinte caminho: *SweRVolf/build/swervolf\_0.7.*3/sim-verilator/

Observe que isso é semelhante ao que fez no Lab 2, quando gera o binário de simulação "**Vrvfpgasim**" com o comando make no seguinte caminho: *[RVfpgaSoCPath]*/RVfpgaSoC/Labs/LabResources/Lab2/verilatorSIM

No entanto, o SoC usado no Lab 2 era um SweRVolfX reduzido, enquanto que o SoC usado aqui é o SweRVolf completo.



**.........**

**Figura 8. Construção do simulador SweRVolf**

Esse comando criará a seguinte hierarquia no diretório Workspace:

* $WORKSPACE
  + fusesoc\_libraries
  + construir
    - swervolf\_0.7.3
      * Sim-Verilator
        + **Vswervolf\_core\_tb**
      * src

# Compilação Para a Nexys A7

Nesta seção, irá ver como construir o SweRVolf Nexys usando o FuseSoC.

**Etapa 1.** Para criar (e, opcionalmente, configurar) um Bitstream para uma placa Nexys A7, execute

* fusesoc run --target=nexys\_a7 swervolf

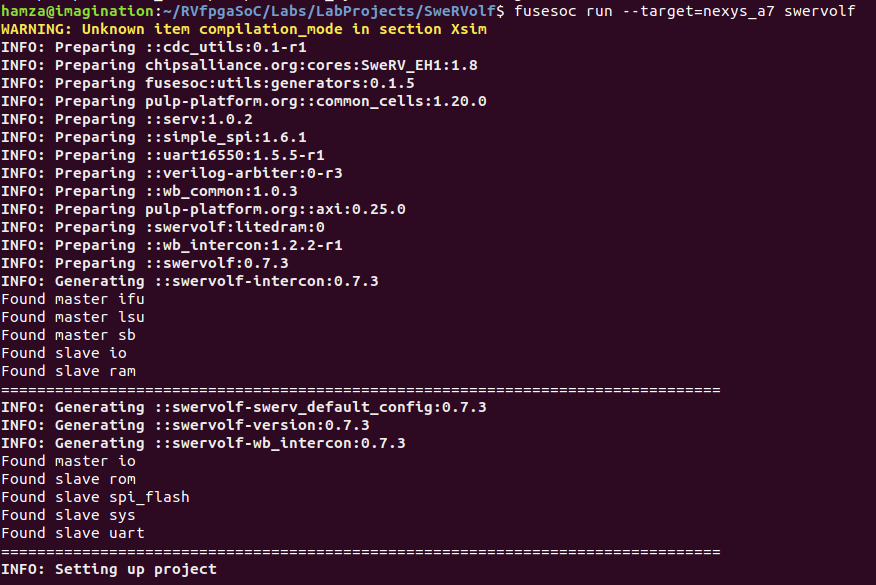
|  |
| --- |
| **Nota**: certifique-se de ligar a placa Nexys A7 ao computador antes de executar esse comando e de que a placa esteja alimentada.  Se receber o seguinte erro, isso significa que não inseriu o caminho do Vivado para o ficheiro "bashrc", conforme mencionado no Guia de Instalação do RVfpgaSoC.    pode adicionar a seguinte linha ao seu ficheiro ~/.bashrc para que seja executada sempre que abrir um terminal ou pode digitá-la no terminal sempre que abrir um novo terminal.   * source /tools/Xilinx/Vivado/2019.2/settings64.sh |

Esse comando gera o ficheiro Bitstream usado para configurar a FPGA (terminado em ".bit") chamado "**swervolf\_0.7.3.bit**" no seguinte caminho: *SweRVolf/build/swervolf\_0.7.3/nexys\_a7-vivado/*

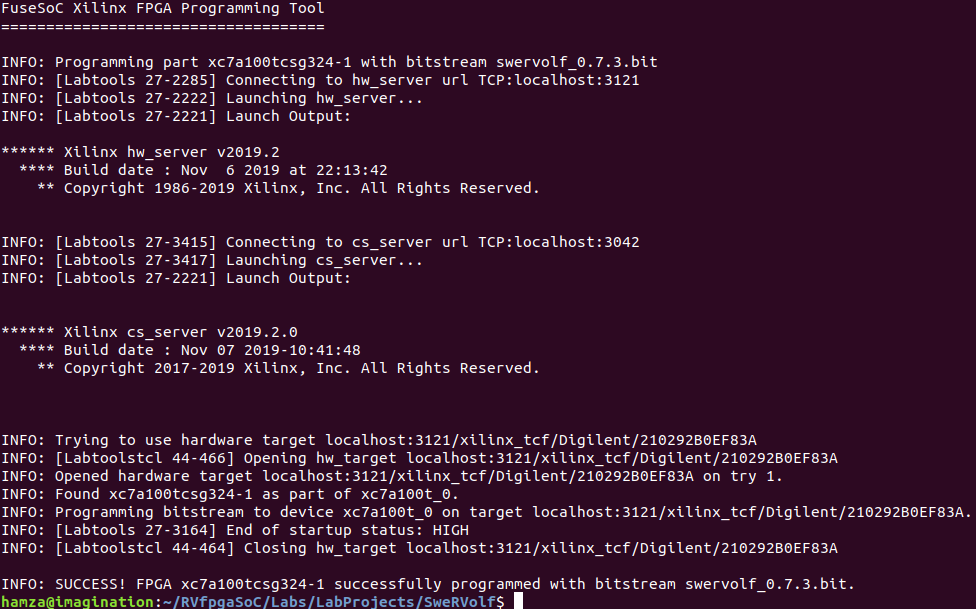
Ele também faz o upload do Bitstream para a placa Nexys A7 ligada ao computador.

Isso é semelhante ao que fez no Lab 1, quando gerou o ficheiro "**rvfpga.bit**" no Vivado após criar o Block Design usando a opção "Generate Bitstream". Mais tarde, no Lab 2, fez o upload do ficheiro "**rvfpga.bit**" para a placa Nexys A7 usando o PlatformIO.

**O FuseSoC faz todo esse trabalho tedioso que fez manualmente no Lab 1 e no Lab 2 com o comando mencionado acima.**



..........



**Figura 9. Compilação e Configuração do SweRVolf Nexys**

Após a conclusão dos comandos mencionados anteriormente, o diretório do Workspace terá a seguinte aparência:

* $WORKSPACE
  + fusesoc\_libraries
  + construir
    - swervolf\_0.7.3
      * Sim-Verilator
      * nexys\_a7-vivado
        + **swervolf\_0.7.3.bit**
      * src

O FuseSoC usa o **Vivado** e cria o projeto, define variáveis globais, adiciona ficheiros de Constraints e gera o Bitstream automaticamente.

Podemos abrir o projeto Vivado para visualizar a hierarquia, como vimos no Lab 1.

**Etapa 2.** Entre no diretório "**nexys\_a7-vivado**" com o seguinte comando:

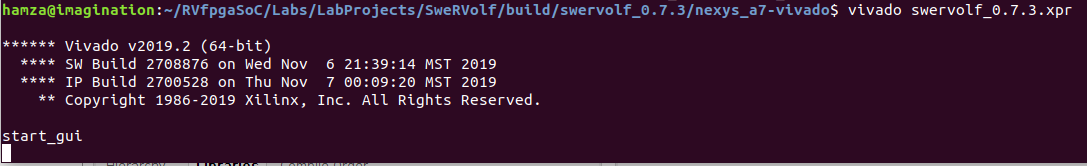
* cd build/swervolf\_0.7.3/nexys\_a7-vivado/

****

**Figura 10. Navegue até o diretório "nexys\_a7-vivado**

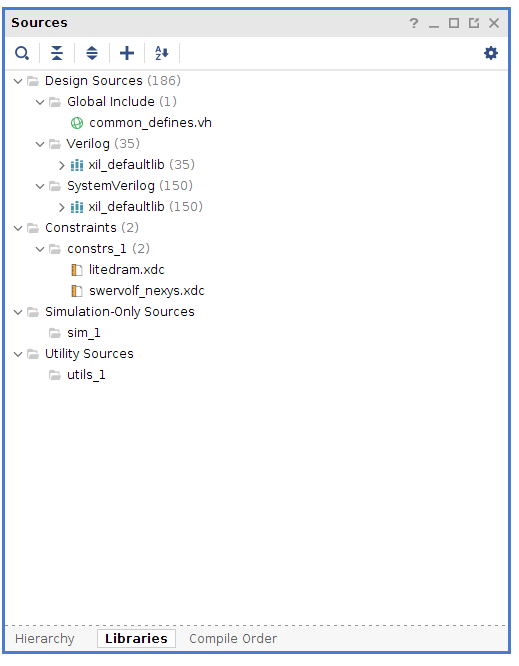
**Etapa 3.** Escreva o seguinte comando para abrir o Projeto Vivado:

* vivado swervolf\_0.7.3.xpr

****

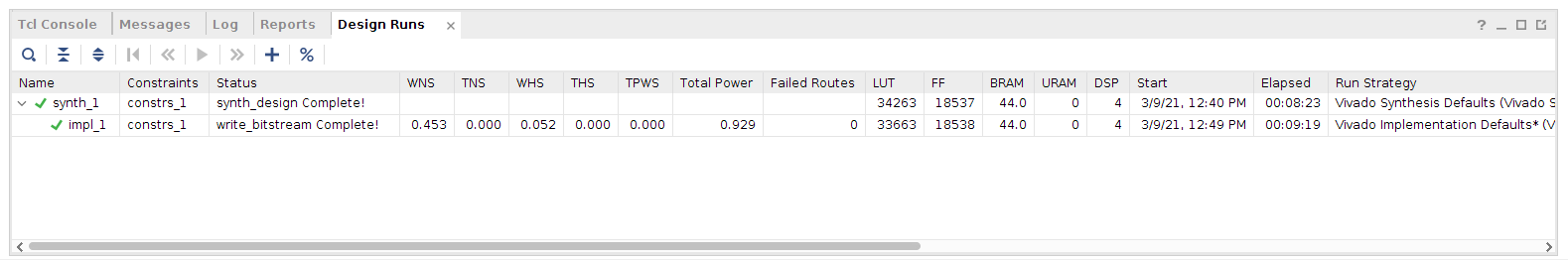
**Figura 11. Abrir projeto Vivado**

Podemos visualizar as variáveis globais e os ficheiros de Constraints definidos no painel de fontes (sources).

****

**Figura 12. Painel Sources**

No painel "Design Runs", pode ver que a síntese e a implementação foram concluídas com êxito.

****

**Figura 13. Execuções de projeto**

# Execução do Exemplo AL\_Operations no Simulador SweRVolf

Nesta seção, irá executar o programa *AL\_Operations* no SweRVolf Sim com o Verilator.

**Etapa 1**. Escreva o diretório de exemplos cujo caminho é:

*[RvfpgaSoCPath]/RvfpgaSoC/Labs/LabResources/Lab3/examples/*

*AL\_Operations/CommandLine/*

* cd ~/RVfpgaSoC/Labs/LabResources/Lab3/examples/AL\_Operations/ commandLine/



**Figura 14. Diretório AL\_Operations**

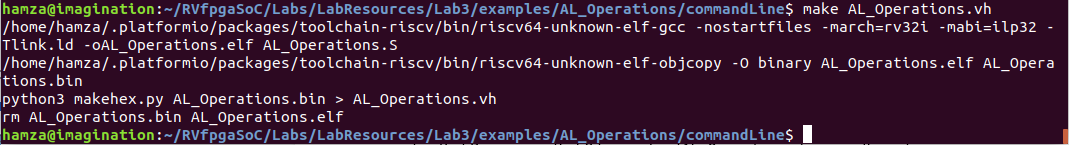
**Etapa 2**. Crie o programa hexadecimal para simulação:

* make clean

****

**Figura 15. make clean**

* make AL\_Operations.vh

****

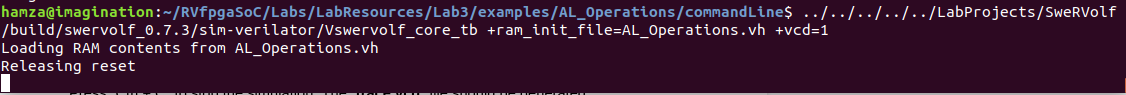
**Figura 16. Compilar o ficheiro AL\_Operations.vh**

|  |
| --- |
| **Nota**: na primeira vez em que um exemplo é aberto no PlatformIO, a plataforma Chips Alliance é instalada automaticamente.  Se vir o seguinte erro, significa que a plataforma Chips Alliance não foi instalada automaticamente.    pode instalá-la manualmente seguindo as etapas mencionadas na seção "Instalação para o Lab 2" do Guia de Instalação do RVfpgaSoC. |

**Etapa 3**. Execute o simulador:

* ../../../../../LabProjects/SweRVolf/build/swervolf\_0.7.3/sim-verilator/Vswervolf\_core\_tb +ram\_init\_file=AL\_Operations.vh +vcd=1

O parâmetro "***ram\_init\_file***" carrega um ficheiro hexadecimal Verilog para ser usado como conteúdo inicial da on-chip RAM. Portanto, carregamos o ficheiro hexadecimal Verilog "**AL\_Operations.vh**" que acabamos de criar usando essa opção (+ram\_init\_file) e executamos o binário de simulação "**Vswervolf\_core\_tb**". Também definimos o vcd (value change dump) como 1 para copiar todas as variáveis dentro do módulo.

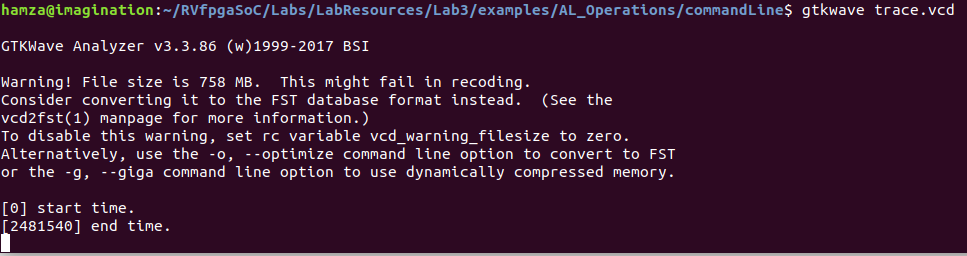
****

**Figura 17. Executar o simulador**

Pressione "CTRL + C" para interromper a simulação. O ficheiro "**trace.vcd**" deverá ser gerado.

**Etapa 4**. Abra o ficheiro trace:

* gtkwave trace.vcd

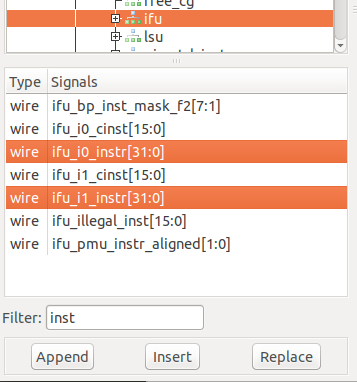
****

**Figura 18. Abrir o ficheiro trace.vcd no gtkwave**

Agora, **o gtkwave** será iniciado. Agora precisará de repetir o processo do Lab 2 para adicionar os sinais ao gráfico e analisá-los.

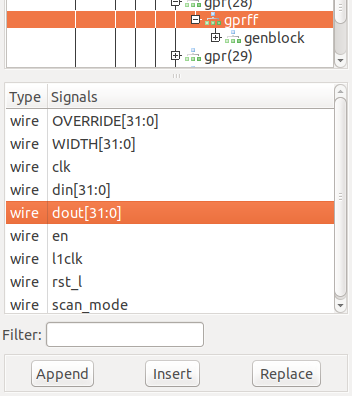
**Etapa 5.** No painel superior esquerdo do *GTKWave*, expanda a hierarquia do SoC para que possa adicionar sinais ao gráfico. Expanda a hierarquia em **TOP → swervolf\_core\_tb→ swervolf → rvtop → swerv,** e clique no módulo **ifu**, selecione o sinal *clk* (que é o relógio usado para o núcleo) e arraste-o para o painel Signals (Sinais) branco ou para o painel Waves (Ondas) preto à direita.

**Etapa 6:** Certifique-sede que o módulo ifu está destacado (consulte a Figura 19), Escreva "inst" na pesquisa do filtro e insira os sinais "ifu\_i0\_instr[31:0]" e "ifu\_i1\_instr[31:0]".



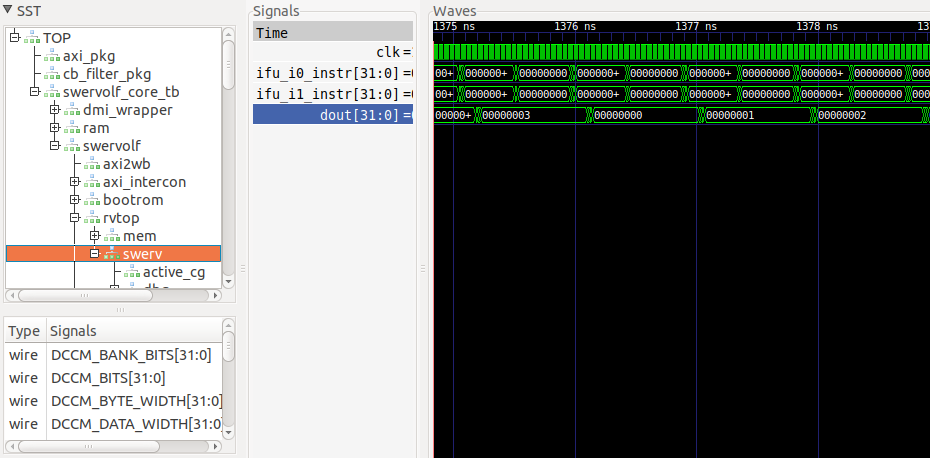
**Figura 19. "Sinais "instr**

**Etapa 7.** Agora, navegue até: **TOP → swervolf\_core\_tb→ swervolf → rvtop → swerv → dec → arf → gpr\_banks(0) → gpr(28) → gprff,** e clique no sinal "dout[31:0]" e insira-o (ver Figura 20).



**Figura 20. "Sinal "dout**

Agora pode ampliar os sinais usando o botão "+" e analisar as formas de onda (Figura 21).



**Figura 21. analisar os sinais**

# Executando o Exemplo Blinky no SweRVolf Nexys

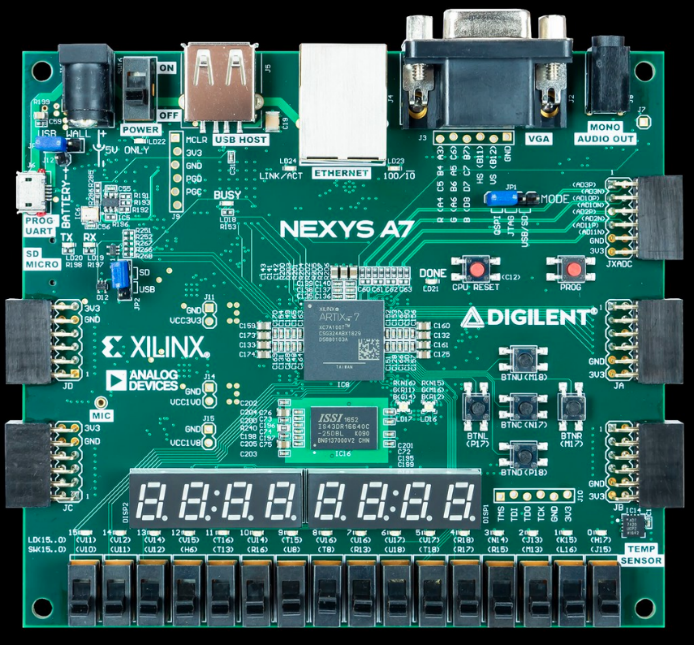
Nesta seção, irá executar o programa Blinky no SweRVolf Nexys. Usará o ficheiro "**.bit**" gerado pelo FuseSoC anteriormente neste laboratório.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Nota:** os mapas de memória de endereços para LEDs e E/S dos módulos SweRVolfX usados no Lab 1 e no Lab 2 são diferentes do SweRVolf (SoC baseado em FuseSoC para SweRV EH1). Portanto, há diretórios de exemplo separados do mesmo programa de exemplo (Blinky) no Lab 2 e no Lab 3 com endereços de memória alterados.  **Tabela 5. Endereços de GPIO MAPEADOS EM MEMÓRIA do SweRVolf e do SweRVolfX**   |  |  |  | | --- | --- | --- | | SoC | Sistema | Endereço | | SweRVolfX (Lab 1) | GPIO | 0x80001400 - 0x8000143F | | SweRVolf (Lab 3) | GPIO | 0x80001010 - 0x80001013 | |

Conclua as etapas a seguir para programar sua placa Nexys A7 com o SweRVolf Nexys e, em seguida, execute o Blinky Program:

**Etapa 1.** Ligue a placa Nexys A7 ao computador.

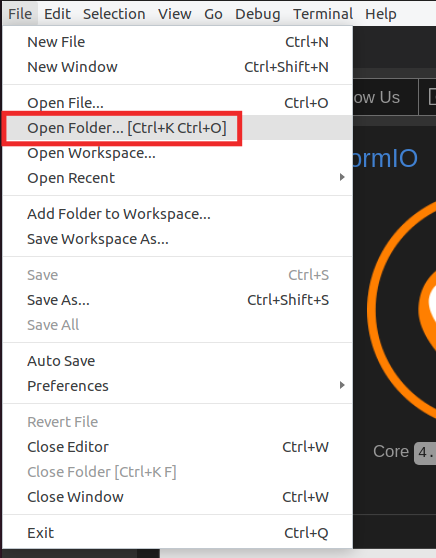
**Etapa 2.** Alimente a placa Nexys A7 no interruptor no canto superior esquerdo. (Figura 22)



**Figura 22. Botão ON/OFF da placa Nexys A7**

**Etapa 3.** Abra o VSCode e o PlatformIO, se ainda não estiverem abertos.

**Etapa 4.** Na barra de menu superior, clique em *File* → *Open Folder* (*abrir pasta)* (Figura 23) e navegue até o diretório *[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab3/examples/*.



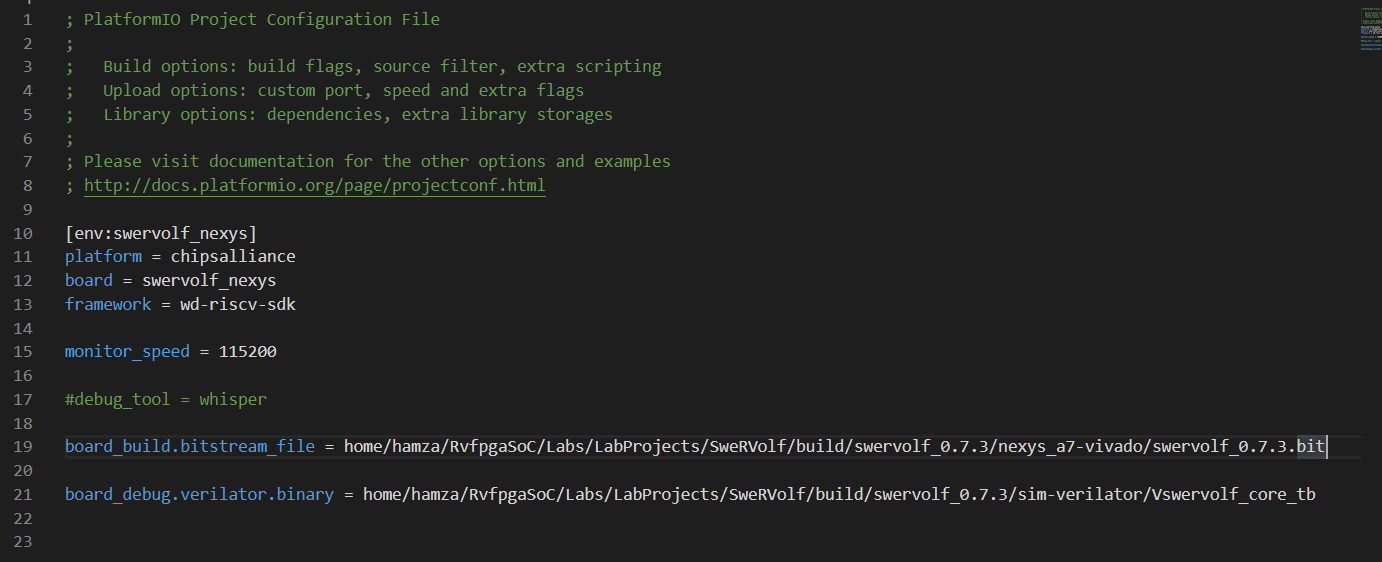
**Figura 23. Abrir pasta**

**Etapa 5.** Selecione o diretório *Blinky\_FuseSoC* (não o abra, apenas selecione-o) e clique em OK na parte superior da janela. O PlatformIO agora abrirá o exemplo.

**Etapa 6.** Abra o ficheiro *platformio.ini clicando* em *platformio.ini* na barra lateral esquerda (Figura 24). Defina o caminho para o Bitstream RVfpga no sistema editando a seguinte linha (Figura 24).

**Etapa 7.** O ficheiro "**swervolf\_0.7.3.bit**" criado com o FuseSoC está no seguinte caminho:

board\_build.bitstream\_file = /home/<Username>/RVfpgaSoC/Labs/LabProjects/SweRVolf/build/swervolf\_0.7.3/nexys\_a7-vivado/swervolf\_0.7.3.bit

****

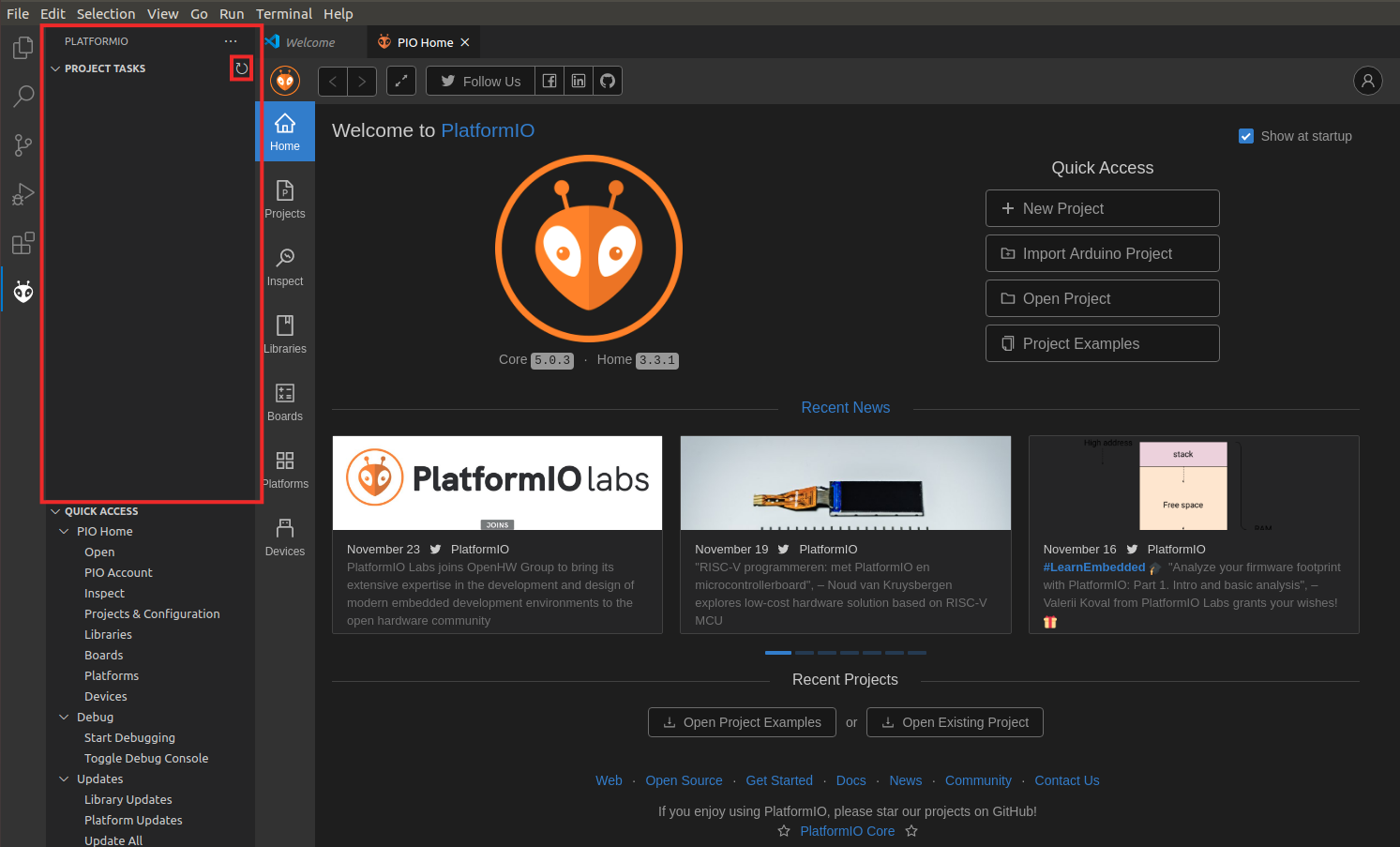
**Figura 24. Ficheiro de inicialização do Platformio: platformio.ini**

**Etapa 8.** Clique no ícone PlatformIO na faixa de opções do menu esquerdo (Figura 25).



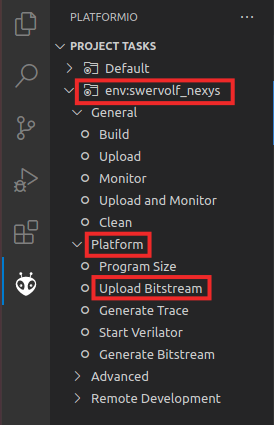
**Figura 25. Ícone PlatformIO**

Caso a janela Project Tasks esteja vazia (Figura 26), primeiro deve atualizar as Project Tasks clicando em . Isso pode levar vários minutos.



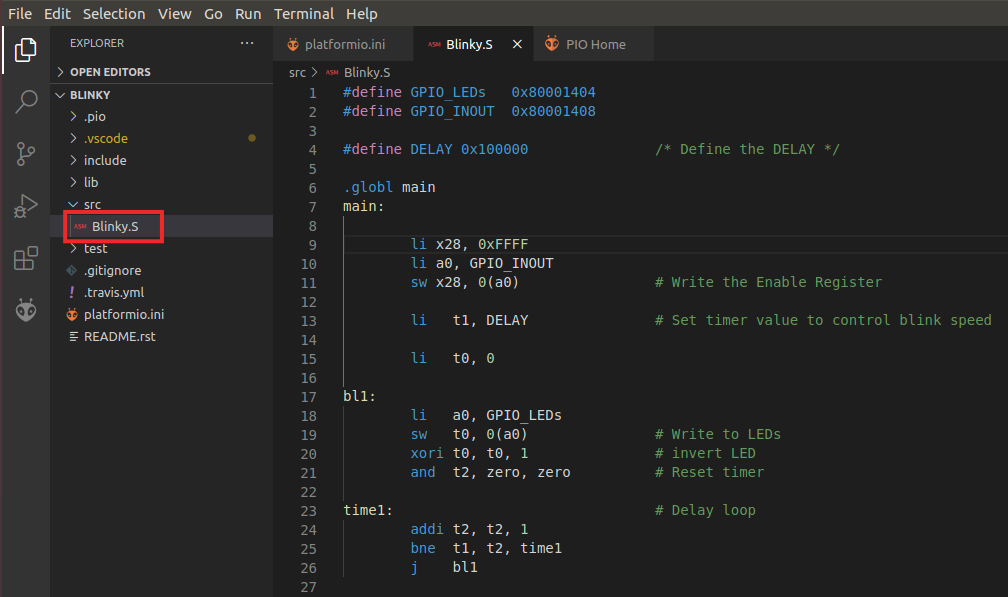
**Figura 26. Janela PROJECT TASKS vazia - Atualizar**

**Etapa 9.** Expanda Project Tasks → env:swervolf\_nexys → Platform e clique em Upload Bitstream, conforme mostrado na Figura 27.



**Figura 27. Carregar bitstream**

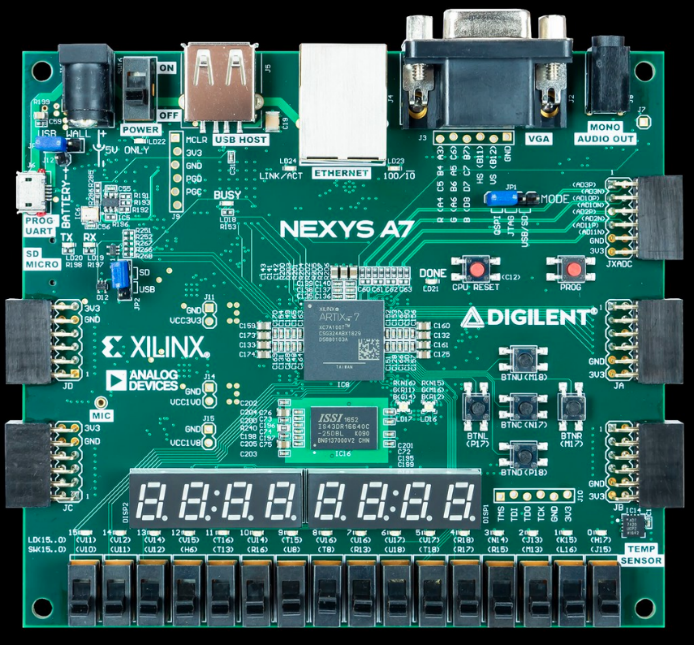
Agora que o Bitstream foi carregado, iniciará o processo de depuração.

****

**Figura 28. blinky.S no PlatformIO**

**Etapa 10.** Clique em para executar e depurar o programa; em seguida, inicie a depuração clicando no botão play . O PlatformIO define um breakpoint temporário no início da função principal. Portanto, clique no botão Continue (Continuar) para executar o programa.

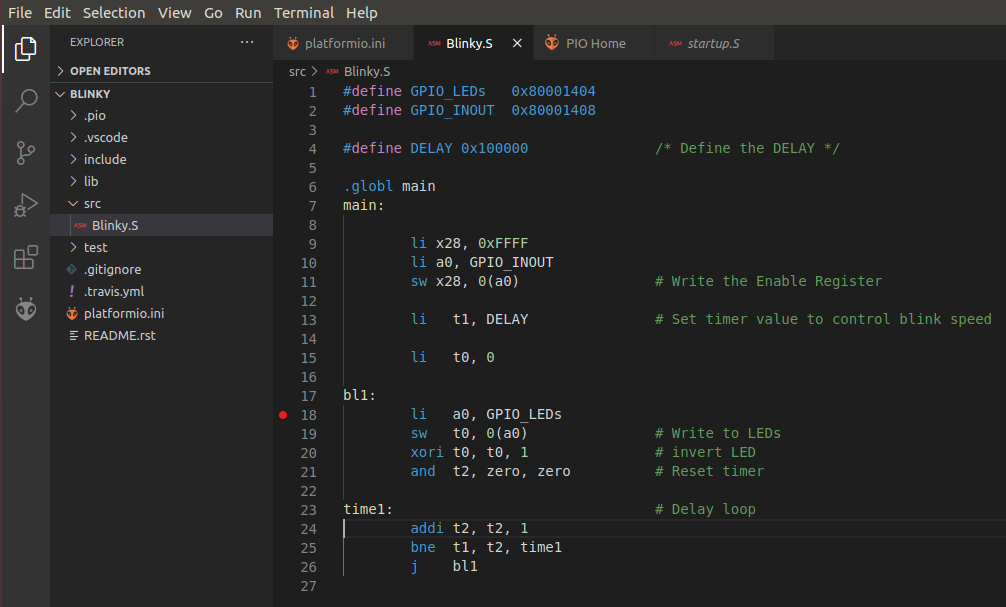
**Etapa 11.** Na placa, verá o LED mais à direita começar a piscar.



**Figura 29. LED mais à direita piscando**

**Etapa 12.** Pause a execução clicando no botão de pausa . A execução será interrompida algures dentro do ciclo infinito (provavelmente, dentro do loop de atraso time1).

**Etapa 13.** Defina um breakpoint clicando à esquerda da linha número 18. Um ponto vermelho aparecerá e o breakpoint será adicionado ao painel BREAKPOINTS (Figura 30).

****

**Figura 30. Definição de um breakpoint no blinky.S**

**Etapa 14.** Em seguida, continue a execução clicando no botão Continue . A execução continuará e será interrompida após a instrução store word (sw), que grava 1 (ou 0) no LED mais à direita.

**Etapa 15.** Continue a execução várias vezes; verá que o valor direcionado ao LED mais à direita muda a cada vez.

**Etapa 16.** Interrompa a depuração em e volte para a janela do Explorer clicando em . Feche o programa selecionando *File* → *Close Folder (Fechar pasta).*

# Depuração do SweRVolf

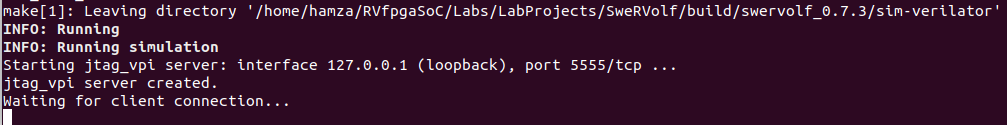
O SweRVolf oferece suporte à depuração tanto no hardware quanto na simulação. Há diferentes procedimentos sobre como conectar o depurador, mas, uma vez conectado, os mesmos comandos podem ser usados (embora os programas sejam executados muito mais lentamente na simulação).

**SIMULAÇÃO:**

**Etapa 1**. Entre no diretório do WORKSPACE "SweRVolf" e, em seguida, execute o comando de simulação:

* fusesoc run --target=sim swervolf --jtag\_vpi\_enable

O parâmetro "***--jtag\_vpi\_enable***" habilita o servidor JTAG ao qual o OpenOCD pode-se conectar. Quando uma simulação do SweRVolf for iniciada com o parâmetro "***--jtag\_vpi\_enable***", ele iniciará um servidor JTAG aguardando que um cliente se conecte e envie comandos JTAG.



**Figura 31. Executar o comando de simulação**

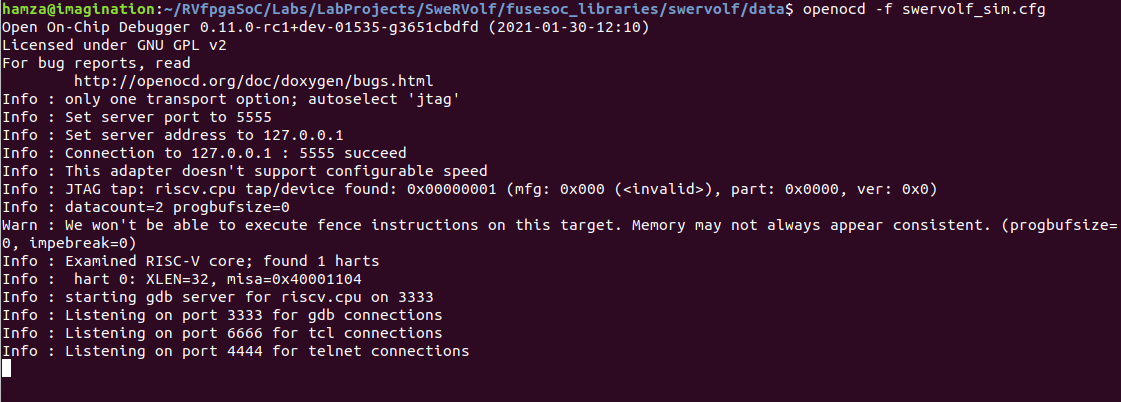
**Etapa 2**. Abra um novo terminal usando "CTRL + SHIFT + T" e, em seguida, no diretório Workspace, navegue até o diretório de dados executando:

* cd fusesoc\_libraries/swervolf/data/

**Etapa 3.** Agora, execute o seguinte comando para ligar o OpenOCD à instância de simulação.

* openocd -f swervolf\_sim.cfg

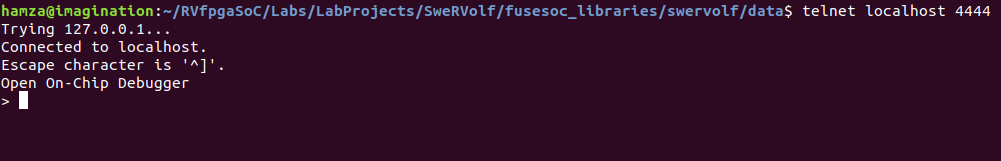
Se for bem-sucedido, o OpenOCD deverá exibir o seguinte (Figura 32).



**Figura 32. Executar o OpenOCD**

**Etapa 4**. Abra um terceiro terminal usando "CTRL + SHIFT + T" e conecte-se à sessão de depuração por meio do OpenOCD.

* telnet localhost 4444



**Figura 33. Ligação telnet**

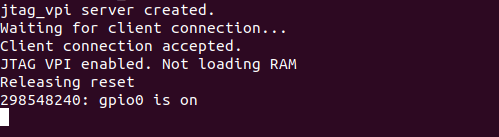
**Etapa 5**. Agora insira comandos para dar instruções. Agora pode fornecer instruções diretamente para o SweRVolf por meio do terceiro terminal.

Nesse SoC, baseado no FuseSoC, 16 LEDs são controlados pelo GPIO mapeado em memória nos endereços **0x80001010-0x80001011.** Esses endereços são diferentes dos módulos do sistema RVfpga que usamos nos Labs 1 e 2.

Portanto, pode escrever um valor de "1" num endereço na memória.

**>** mwb 0x80001010 1

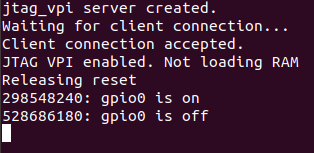
Agora abra o primeiro terminal e verá que o gpio0 está ligado (Figura 34).



**Figura 34. gpio0 ligado.**

Novamente, volte ao terceiro terminal e escreva um valor de "0" no mesmo endereço de memória para desligar o gpio0.

**>** mwb 0x80001010 0



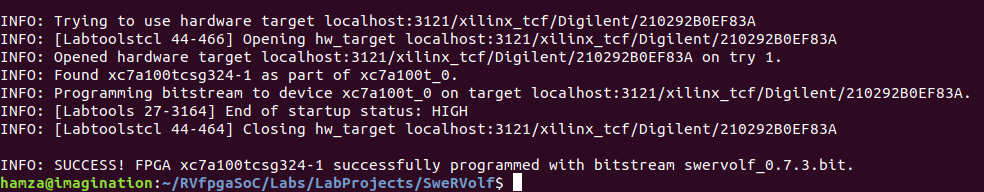
**Figura 35. gpio0 desligado.**

Agora, concluiremos esse mesmo processo na placa Nexys A7.

**HARDWARE:**

**Etapa 1**. Conecte a placa Nexys A7 ao computador e, em seguida, execute o comando de compilação FPGA no diretório Workspace. Para carregar o Bitstream sem recompilar para a placa Nexys A7 novamente, pode adicionar o parâmetro "**--run**".

* fusesoc run --target=nexys\_a7 --run swervolf

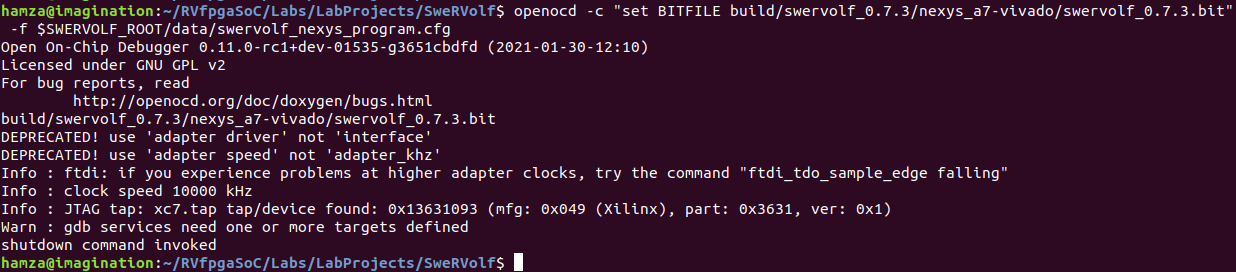


**Figura 36. Executar a configuração da FPGA**

**Etapa 2.** Configurar a FPGA usando o OpenOCD

* openocd -c "set BITFILE build/swervolf\_0.7.3/nexys\_a7-vivado/swervolf\_0.7.3.bit" -f $SWERVOLF\_ROOT/data/swervolf\_nexys\_program.cfg

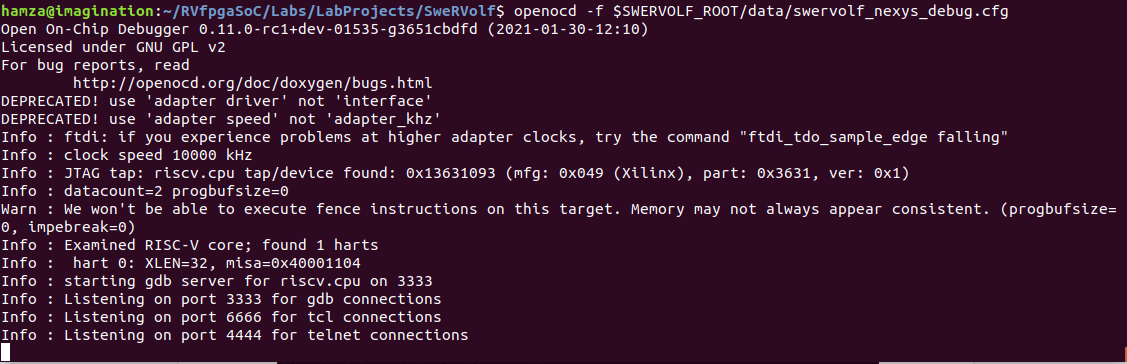
O resultado deve ser como o mostrado na Figura 37.



**Figura 37. Executar o OpenOCD**

**Etapa 3.** Ligar o OpenOCD ao SweRVolf

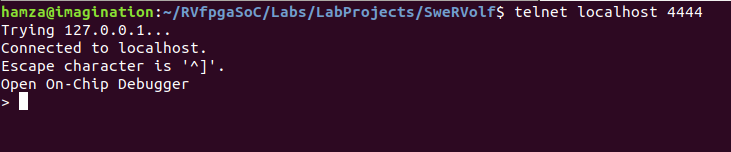
* openocd -f $SWERVOLF\_ROOT/data/swervolf\_nexys\_debug.cfg



**Figura 38. OpenOCD conectado**

**Etapa 4**. Abra um terceiro terminal usando "CTRL + SHIFT + T" e conecte-se à sessão de depuração por meio do OpenOCD:

* telnet localhost 4444



**Figura 39. Ligação telnet**

**Etapa 5**. Agora, escreva os comandos para dar instruções diretamente à FPGA. Agora pode dar instruções diretamente para o SweRVolf por meio do terceiro terminal.

Os 16 LEDs são controlados pelo GPIO mapeado em memória nos endereços **0x80001010 - 0x80001011.**

Escreva o valor "1" em 0x80001010 na memória para ligar o LED mais à direita:

* mwb 0x80001010 1

Agora, execute o seguinte comando para desligar o LED mais à direita:

* mwb 0x80001010 0

Isso desligará o LED mais à direita da placa.

Para recapitular rapidamente este laboratório, primeiro comparou o código do Block Design com o código ajustado manualmente. Em seguida, introduziu-se o FuseSoC, um gestor de pacotes que contém o pacote SweRVolf (SoC baseado no FuseSoC para o SweRV EH1).

Em seguida, explorou as duas versões, "SweRVolf Sim" e "SweRVolf Nexys".

Tal como o **RVfpgaSim** no laboratório anterior, o **SweRVolf Sim** é um sistema para simulação que envolve o núcleo do SweRVolf num testbench a ser usado pelo Verilator. Da mesma forma, o **RVfpgaNexys** do laboratório anterior é semelhante ao **SweRVolf Nexys**, que é uma versão do SoC SweRVolf realizada para a placa Digilent Nexys A7.

Em seguida, criou o SweRVolf Sim para simulação e o SweRVolf Nexys para a placa Nexys A7 usando a execução do FuseSoC. Mais tarde, executou os mesmos exemplos nessas compilações que executou no laboratório anterior no RVfpgaSim e no RVfpgaNexys.

**Nota:** economize tempo nos Labs 4 e 5 armazenando os comandos num ficheiro. Basta exportar os comandos para um ficheiro de texto, salvá-lo (por exemplo, "my\_exports.sh") e usá-lo como fonte em novas sessões de terminal ou adicioná-lo ao .bashrc. Simplifique o fluxo de trabalho e facilmente crie sessões de terminal adicionais quando necessário.