



**THE IMAGINATION UNIVERSITY PROGRAMME**

# **RVfpga-SoC Lab 2**

## **Execução de Software no RVfpga-SoC**

**Tabela 1. Termos do RVfpga**

Nome	Descrição
<b>Cursos</b>	
<b>RVfpga</b>	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.
<b>RVfpga-SoC</b>	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.
<b>Núcleos e SoCs</b>	
<b>SweRV EH1 Core</b>	Núcleo RISC-V comercial de código aberto desenvolvido pela Western Digital ( <a href="https://github.com/chipsalliance/Cores-SweRV">https://github.com/chipsalliance/Cores-SweRV</a> ).
<b>SweRV EH1 Core Complex</b>	Núcleo SweRV EH1 com memória adicional (ICCM, DCCM e cache de instruções), controlador de interrupção programável (PIC), interfaces de barramento e unidade de depuração ( <a href="https://github.com/chipsalliance/Cores-SweRV">https://github.com/chipsalliance/Cores-SweRV</a> ).
<b>SweRVolfX</b>	É o System-on-a-Chip usado no curso RVfpga. É uma extensão do SweRVolf. <b>SweRVolf</b> ( <a href="https://github.com/chipsalliance/Cores-SweRVolf">https://github.com/chipsalliance/Cores-SweRVolf</a> ): 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. <b>SweRVolfX</b> : 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.
<b>RVfpgaNexys</b>	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 ( <a href="https://github.com/chipsalliance/Cores-SweRVolf">https://github.com/chipsalliance/Cores-SweRVolf</a> ), exceto que o último é baseado no SweRVolf.
<b>RVfpgaSim</b>	O SoC SweRVolfX tem um encapsulamento de testbench e memória AXI destinados a simulação. O RVfpgaSim é o mesmo que o SweRVolf Sim ( <a href="https://github.com/chipsalliance/Cores-SweRVolf">https://github.com/chipsalliance/Cores-SweRVolf</a> ), exceto pelo fato de que o último é baseado no SweRVolf.

## 1. Introdução

Este laboratório mostra como executar programas escritos em linguagem C ou Assembly no SweRVolfX que criado no Lab 1 usando a ferramenta de Vivado Block Design. Pode optar por simular o projeto usando o Verilator ou executá-lo na placa Nexys A7. Se não tiver acesso a uma placa FPGA, este laboratório poderá ser concluído somente em simulação usando o Verilator. Para concluir este laboratório, usará o ficheiro Verilog **"BD.v"** do Block Design e o ficheiro de bits **"rvfpga.bit"** que foi gerado no Lab 1 usando o Block Design do Vivado.

Neste laboratório, irá ver como gerar os binários de simulação para o **RVfpgaSim**, que serão usados posteriormente para criar o trace de simulação de um programa de exemplo. Também será analisado o trace da simulação usando o GTKWave.

Como etapa opcional, será mostrado como fazer a configuração do **RVfpgaNexys**, conforme definido pelo Bitstream criado no Lab 1, na placa Nexys A7 usando o PlatformIO e, em seguida, a depuração de um programa de exemplo usando o PlatformIO. Essa etapa é opcional, mas recomendada.

## 2. Requisitos

Para concluir este laboratório, precisará de instalar as seguintes ferramentas:

- VSCode (Consulte o Guia de Instalação (página nº 06))
- PlatformIO (Consulte o Guia de instalação (página nº 06))
- GTKWave (Consulte o Guia de Instalação (página nº 09))
- Verilator (Consulte o Guia de Instalação (página nº 09))
- Cygwin (apenas para Windows) (Consulte o Guia de Instalação (página nº 15))

**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 VScode e o Verilator seguindo as instruções do Guia de Instalação do RVfpga-SoC. Certifique-se de que copiou a pasta RVfpga-SoC que descarregou do Imagination University Programme para o computador.

## 3. Execução do SoC Criado no Block Design

No primeiro Lab, criou um SoC SweRVolfX ligando o núcleo do processador, as interconexões e os periféricos entre si usando a ferramenta Block Design do Vivado. Em seguida, o Block Design gera um ficheiro Verilog desse módulo de Block Design como um todo. Neste caso, foi o ficheiro **"BD.v"**.

Agora tem duas opções para executar o SoC do Block Design:

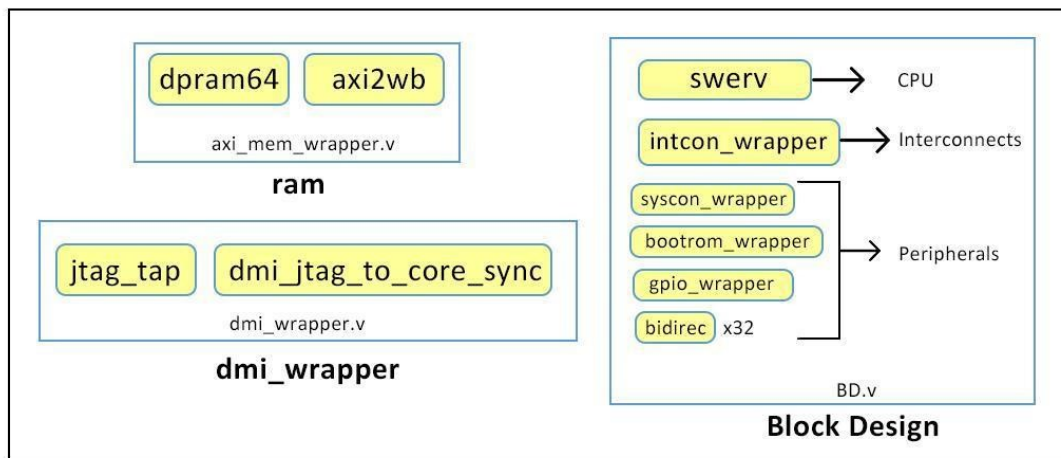
- Usar a placa Nexys A7 100T.
- Usar o simulador Verilator.

O SoC do Block Design tem dois módulos de nível superior que existem para cada uma dessas opções de execução: RVfpgaSim (rvfpgasim) e RVfpga Nexys (rvfpga), conforme descrito abaixo:

## 1. RVfpgaSim (rvfpgasim.v)

O módulo RVfpgaSim é usado como o módulo principal do sistema RVfpga para **simulação**. Usará o Verilator (um simulador de linguagem de descrição de hardware (HDL) que simula o Verilog que define o RVfpga) para simular o sistema RVfpga. A execução do SoC em simulação permite analisar os sinais internos do sistema em detalhe. Mais adiante neste laboratório, quando gerar o binário de simulação para o RVfpgaSim, usará o "rvfpgasim.v" como ficheiro do módulo principal.

A estrutura do módulo principal "rvfpgasim" é ilustrada na Figura 1.



**Figura 1. RVfpgaSim**

Ele inclui três módulos:

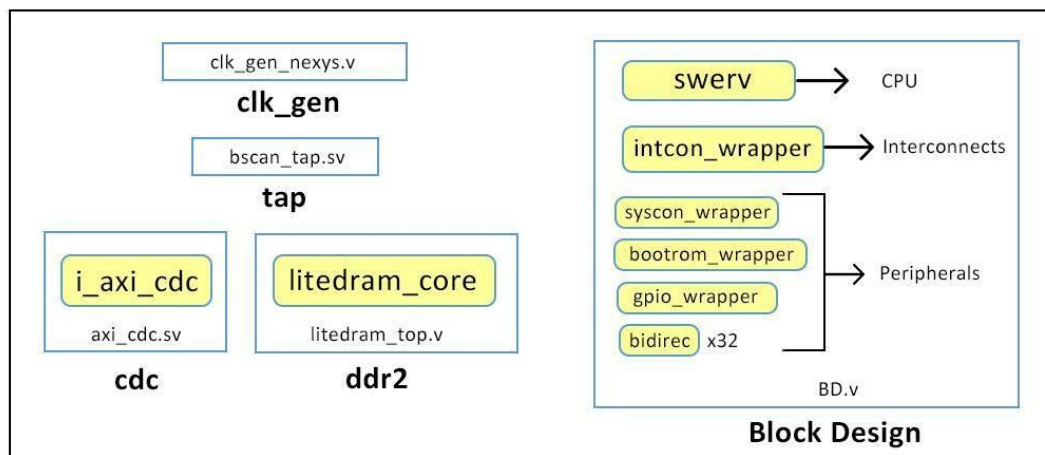
- Block Design (BD.v)  
Este é o módulo SoC que criou usando o Block Design do Vivado.
- ram (axi\_mem\_wrapper.v)  
Esse é o módulo de memória.
- dmi\_wrapper (dmi\_wrapper.v)  
Essa é a interface do módulo Debugging.

No Lab 1, ao conectar os pinos usando o Block Design do Vivado, fez várias ligações a pinos externos. Essas ligações externas do módulo "**Block Design**" são ligadas no módulo principal "**rvfpgasim**" com outros módulos. Por exemplo, as ligações externas "**DMI**" no módulo "**Block Design**" são ligadas ao módulo "**dmi\_wrapper**", e as ligações externas "**RAM**" do módulo "**Block Design**" são ligadas ao módulo "**ram**".

## 2. RVfpga Nexys (rvfpga.sv)

O módulo RVfpga Nexys é usado como o módulo principal do sistema RVfpga para o Hardware (On-Board Implementation), que é implementado para a placa Digilent Nexys A7 (ou, de forma compatível, a antiga placa Nexys 4 DDR).

A estrutura do módulo principal "rvfpga.sv" é ilustrada na Figura 2.



**Figura 2. Nexys RVfpga**

O RVfpga Nexus inclui cinco módulos:

- **Block Design (BD.v)**  
Este é o módulo SoC que criamos usando o Block Design.
- **ddr2 (litedram\_top.v)**  
Esse é o módulo controlador de memória DDR.
- **clk\_gen (clk\_gen\_nexys.v)**  
Esse é o módulo gerador de relógio.
- **tap (bscan\_tap.v)**  
Este é o módulo de depuração JTAG. Para mais informações, ver este [link](#)
- **cdc (axi\_cdc.v)**  
Esse é o módulo Clock Domain Crossing.

No módulo principal "**rvfpga.v**", os pinos externos "**RAM**" do módulo "**Block Design**" são ligados ao módulo "**ddr2**". Os pinos externos "**DMI**" do "**Block Design**" são ligadas ao módulo "**bscan\_tap.v**". O pino externo "**clk**" é ligado ao módulo "**clk\_gen**" (Figura 3).

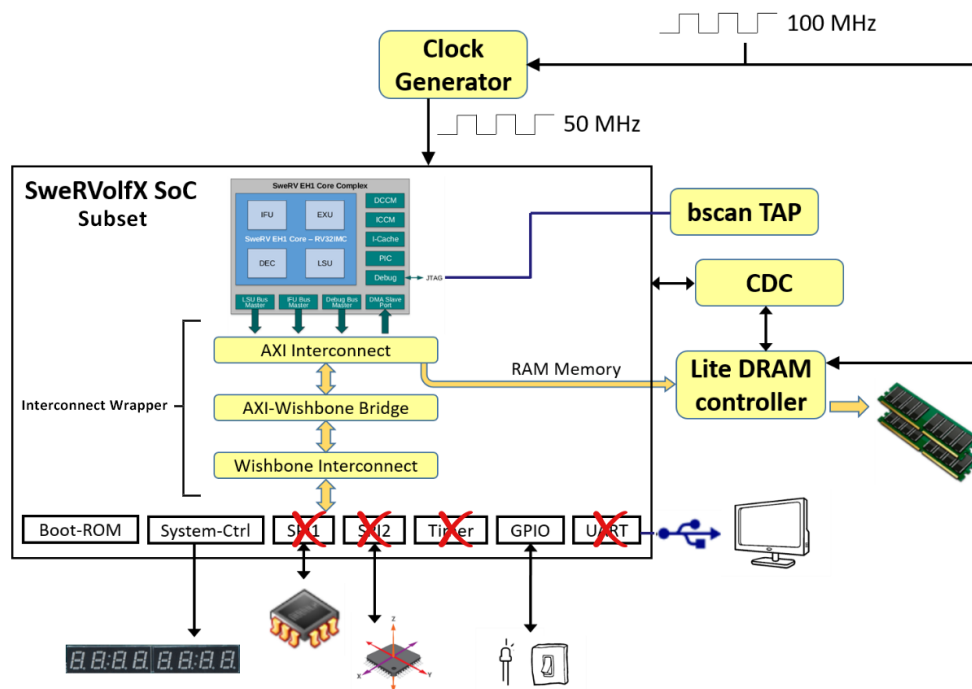


Figura 3. RVfpga Nexys (SoC SweRVolfX reduzido)

#### 4. Executando um Programa no Verilator

Esta seção o guiará pelo processo de execução do seu primeiro programa (*AL\_Operations*) no **RVfpgaSim** usando o Verilator.

**Nota:** todos os módulos de código-fonte Verilog do RVfpga precisam ter o prefixo "BD\_" para funcionar com o ficheiro "BD.v" gerado pelo Block Design. Esse ficheiro "BD.v" é usado pelo módulo principal "rvfpgasim" para simulação no Verilator. Os módulos-fonte são instanciados no ficheiro "BD.v" e isso exige que todos os módulos usados sejam prefixados com "BD\_". Isso foi feito para si numa pasta "src" separada no seguinte caminho: [RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab2/src.

Primeiro, precisa de mover o ficheiro de módulo "**Block Design**", **BD.v**, para a pasta que contém todos os outros ficheiros-fonte, inclusive o ficheiro de módulo superior "rvfpgasim.v".

Quando criou o encapsulamento HDL no laboratório anterior, foi dado o seu caminho completo (ver Figura 4). Precisa de ir até esse caminho e, em seguida, copiar o ficheiro.

[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabProjects/Lab1/Lab1.srcs/sources\_1/bd/BD/synth/BD.v

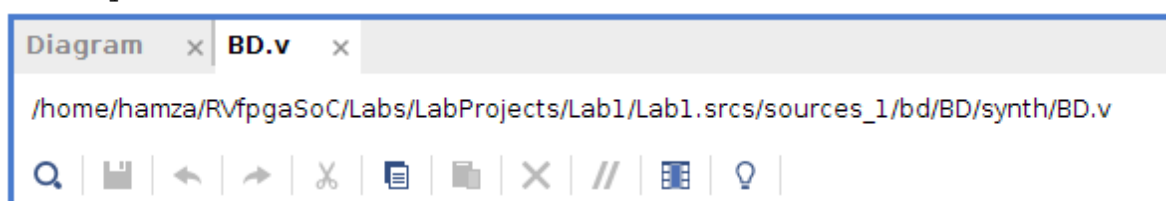
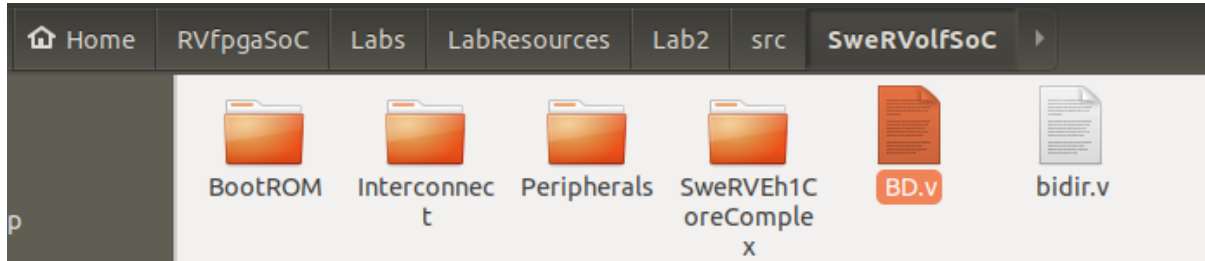


Figura 4. O caminho do ficheiro Verilog "BD.v" do módulo "Block Design".

**Etapa 1.** Copie o ficheiro "BD.v" do caminho fornecido na (Figura 4) e cole o ficheiro "BD.v" no seguinte caminho (Figura 5):

[RVfpgaSoCPath] /RVfpgaSoC/Labs/LabResources/Lab2/src/SweRVolfSoC/



**Figura 5. "BD.v" colado no diretório "SweRVolfSoC".**

**Etapa 2.** Abra o ficheiro "BD.v" e verifique se o nome do módulo a seguir termina com "\_0\_0" (Figura 6).

- BD\_bootrom\_wrapper\_0\_0
- BD\_gpio\_wrapper\_0\_0
- BD\_intcon\_wrapper\_bd\_0\_0
- BD\_swerv\_wrapper\_verilog\_0\_0
- BD\_syscon\_wrapper\_0\_0

**Nota:** Isso é feito para manter a consistência na nomeação dos módulos para simulação. Se eles não forem consistentes, dá erro ao gerar o binário de simulação para o RvfpgaSim na próxima etapa.

```
BD_bootrom_wrapper_0_0 bootrom_wrapper_0
(.i_clk(clk_0_1),
 .i_rst(rst_0_1),
 .i_wb_adr(intcon_wrapper_bd_0_wb_rom_adr_o),
 .i_wb_cyc(intcon_wrapper_bd_0_wb_rom_cyc_o),
 .i_wb_dat(intcon_wrapper_bd_0_wb_rom_dat_o),
 .i_wb_sel(intcon_wrapper_bd_0_wb_rom_sel_o),
 .i_wb_stb(intcon_wrapper_bd_0_wb_rom_stb_o),
 .i_wb_we(intcon_wrapper_bd_0_wb_rom_we_o),
 .o_wb_ack(bootrom_wrapper_0_o_wb_ack),
 .o_wb_rdt(bootrom_wrapper_0_o_wb_rdt));
```

**Figura 6. "BD.v"**

Agora, irá iniciar o processo de execução do programa *AL\_Operations* no SoC do Block Design.

Primeiro, gere o trace da simulação usando o PlatformIO e, em seguida, adicionará o relógio, as instruções para ambas as vias do processador superescalar e os sinais do registro x28 (ou seja, o registro t3) à forma de onda da simulação e visualizará com o GTKWave a alteração dos sinais da instrução e do registro à medida que o programa for executado.

Para fazer isso, conclua as etapas seguintes:

**Etapa 3. Gere o binário de simulação para o RvfpgaSim**

O diretório `[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab2/verilatorSIM` contém o *Makefile* e o *script* (*swervolf\_0.7.vc*) para gerar o binário do simulador para o RVfpgaSim. O *script* contém informações para que o Verilator saiba, entre outras coisas, onde encontrar os ficheiros de código-fonte do SoC, que, neste caso, estão disponíveis em: `[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab2/src`.

Em seguida, gere o binário para o RVfpgaSim, que será usado posteriormente para criar o trace de simulação do programa *AL-Operations* em execução no RVfpga.

Numa janela de terminal, gere o binário do simulador executando os seguintes comandos:

- `cd`  
`[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab2/verilatorSIM`
- `make clean`
- `make`

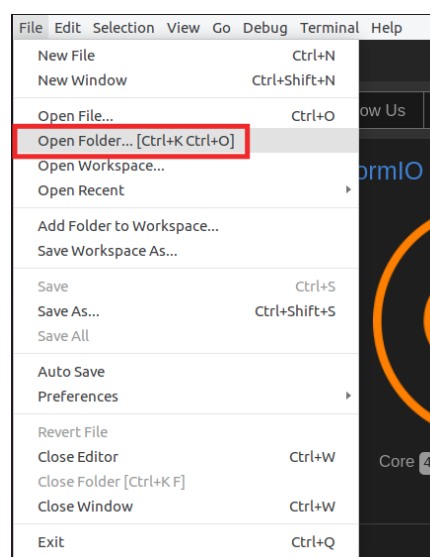
O ficheiro ***Vrvfpgasim*** (o binário de simulação do RVfpga) deve ser gerado dentro do diretório `[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab2/verilatorSIM`.

**Windows:** se estiver em Windows, deverá executar essas mesmas etapas no terminal Cygwin (consulte o Apêndice B do Guia de Introdução do RVfpga-SoC para obter instruções detalhadas). Observe que a pasta `C: Windows` pode ser encontrada dentro do Cygwin em: `/cygdrive/c`. Todas as outras instruções desta seção são as mesmas descritas para o Linux.

#### **Etapas 4. Gere o trace de simulação a partir do PlatformIO**

Depois do binário do simulador (*Vrvfpgasim*) tiver sido gerado, irá usá-lo dentro do PlatformIO para gerar o trace de simulação (*trace.vcd*) do programa *AL\_Operations*.

1. Abra o VSCode e, em seguida, o PlatformIO no computador.
2. Na barra superior, clique em *File*→*Open Folder* (Figura 7) e navegue até o diretório `[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab2/examples/`

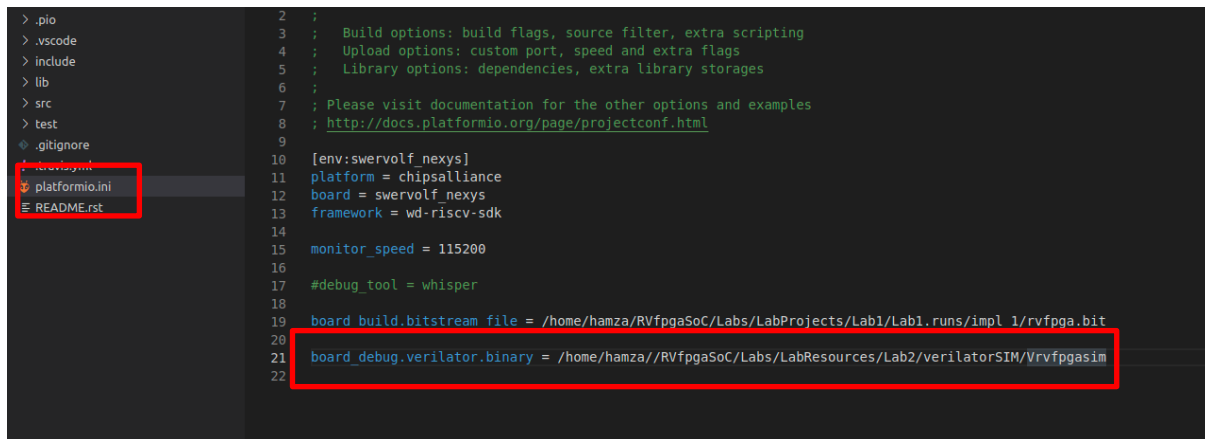


**Figura 7. Abra o exemplo *AL\_Operations.S***



3. Selecione o diretório *AL\_Operations* (mas não o abra, apenas selecione-o) e clique em OK. O exemplo será aberto no PlatformIO.
4. Abra o ficheiro *platformio.ini*. Defina o caminho para o binário de simulação do RVfpga gerado na primeira etapa (*Vrvfpgasim*) editando a seguinte linha (Figura 8).


```
board_debug.verilator.binary =
[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab2/verilatorSIM/Vrvfpgasim
```

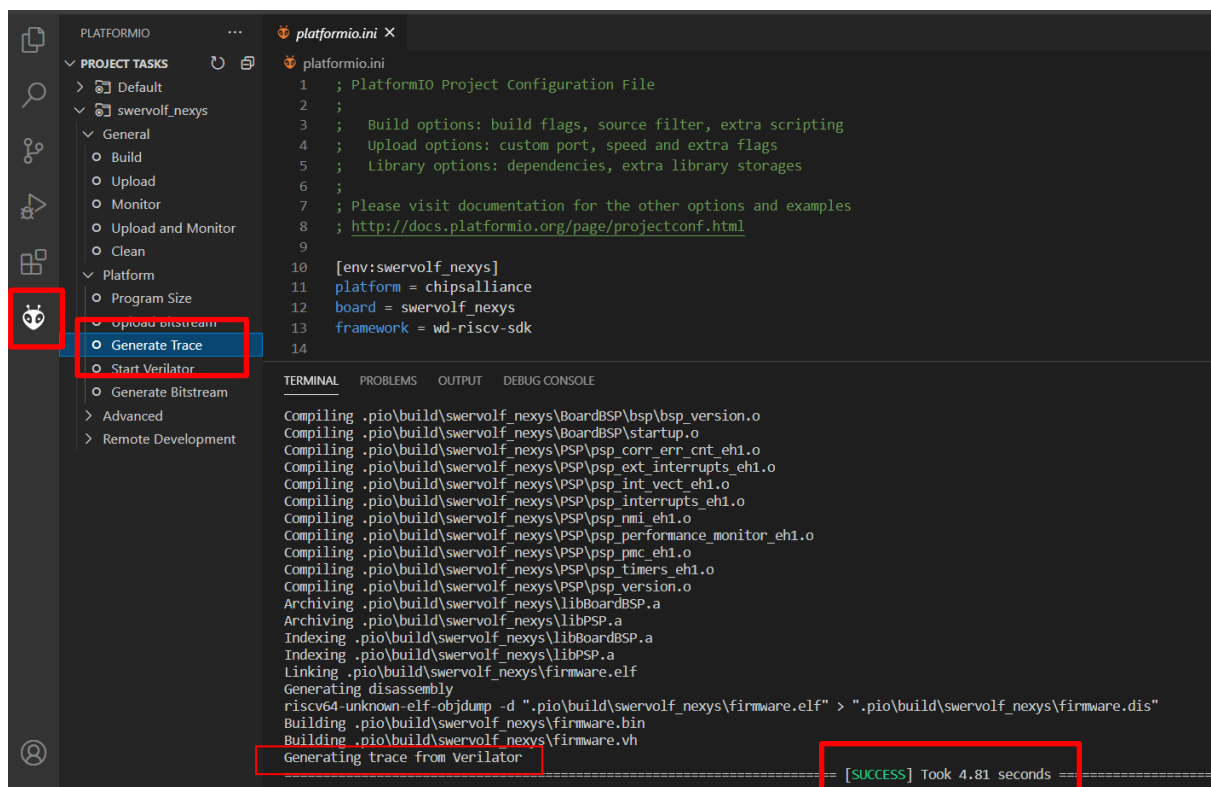


**Figura 8. Ficheiro de inicialização do PlatformIO: platformio.ini**

**Windows:** o executável de simulação do RVfpga é chamado *Vrvfpgasim.exe*. Portanto:


```
board_debug.verilator.binary = [RVfpgaSoCPath]\RVfpgaSoC\Labs\LabResources\Lab2\
verilatorSIM\Vrvfpgasim.exe
```

5. Execute a simulação clicando no ícone PlatformIO na faixa de opções do menu esquerdo  e, em seguida, expanda Project Tasks → env:swervolf\_nexys → Platform e clique em Generate Trace, conforme mostrado na Figura 9.



**Figura 9. Geração de trace do Verilator**

Como alternativa, pode gerar o trace a partir de uma janela de terminal do PlatformIO.

Para isso, clique no botão  (PlatformIO: botão New Terminal) na parte inferior da janela do PlatformIO para abrir uma nova janela de terminal e, em seguida, escreva (ou copie) o seguinte comando no terminal do PlatformIO: `pio run --target generate_trace`

6. Alguns segundos após a etapa anterior, o ficheiro `trace.vcd` é gerado dentro de `[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab2/examples/AL_Operations/.pio/build/swervolf_nexys`, e pode abri-lo com o `GTKWave`. Abra o terminal do Ubuntu e escreva:

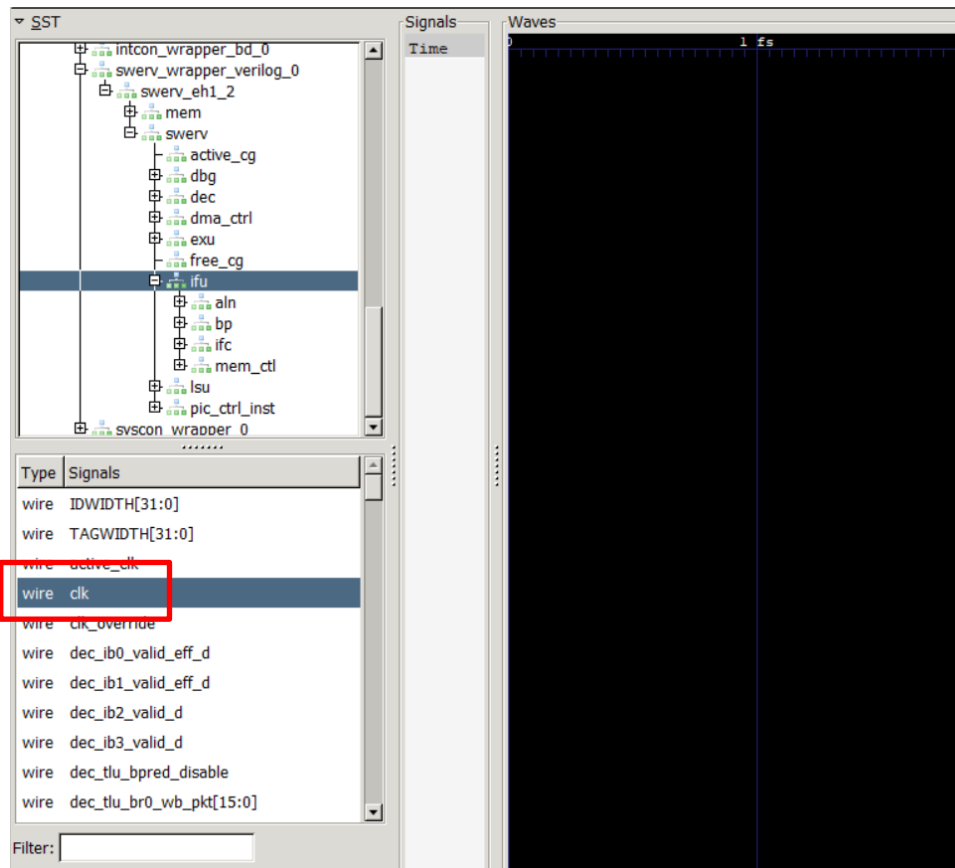
```
gtkwave [RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab2/
examples/AL_Operations/.pio/build/swervolf_nexys/trace.vcd
```

**WINDOWS:** a pasta `gtkwave64` que descarregou inclui uma aplicação chamada `gtkwave.exe` dentro da pasta `bin`. Inicie o `GTKWave` clicando duas vezes nessa aplicação. Na parte superior da aplicação, clique em **File - Open New Tab** e abra o ficheiro `trace.vcd` gerado na pasta `[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab2/examples/AL_Operations/.pio/build/swervolf_nexys`.

## Etapa 5. Analise a simulação no GTKWave

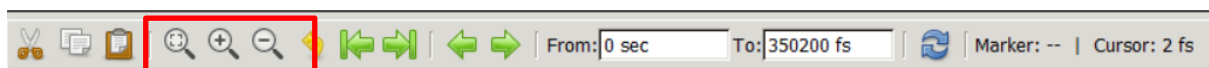
1. Agora, adicionará um sinal de relógio, uma instrução e sinais de registro. No painel superior esquerdo do `GTKWave`, expanda a hierarquia do SoC para que possa adicionar sinais ao gráfico. Expanda a hierarquia em **TOP** → **rvfpgasim** → **swervolf** → **swerv\_wrapper\_verilog\_0** → **swerv\_eh1\_2** → **swerv**, e clique no módulo **ifu** (ele será destacado como mostrado na Figura 10), 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.



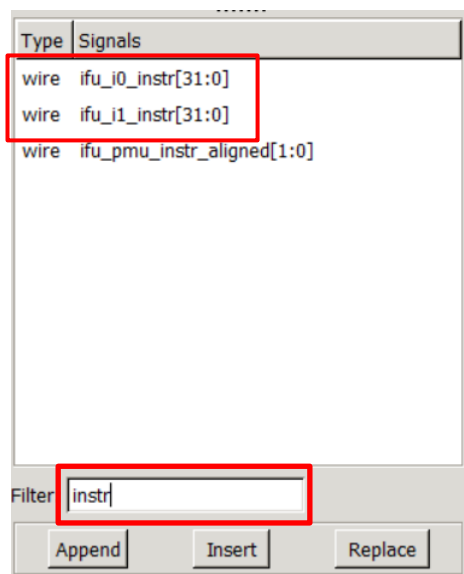
**Figura 10. Adicionar o sinal *clk* ao gráfico**

2. Faça um Zoom Fit e, em seguida, aumente o zoom várias vezes para que possa ver a mudança do sinal do relógio (Figura 11).



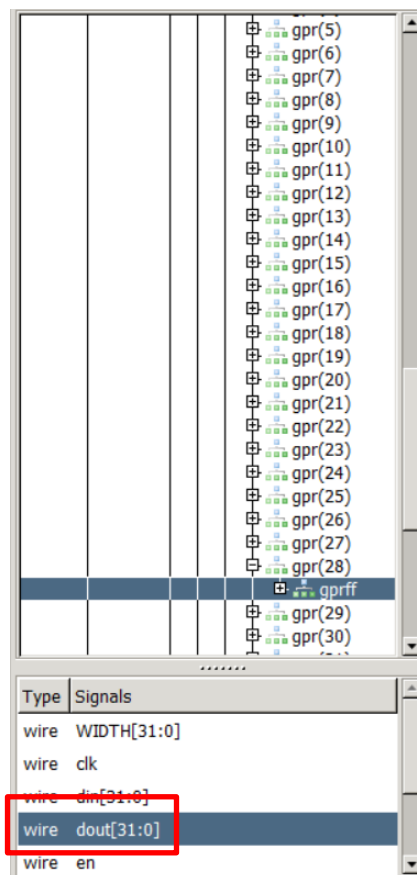
**Figura 10. Aumentar o zoom**

3. Agora adicione os sinais que mostram as instruções que são executadas em cada via do núcleo RISC-V superescalar de duas vias. No mesmo módulo (*ifu*), procure os sinais *ifu\_i0\_instr[31:0]* e *ifu\_i1\_instr[31:0]* (Figura 12) e arraste-os para o painel preto do Waves. O prefixo *ifu* indica a Instruction Fetch Unit, *i0* indica a via superescalar 0 e *i1* indica a via superescalar 1; *instr[31:0]* indica a instrução de 32 bits.
4. Pode usar o filtro de pesquisa para encontrar os sinais rapidamente (Figura 12).



**Figura 12. Adicionar os sinais *ifu\_i0\_instr[31:0]* e *ifu\_i1\_instr[31:0]* à forma de onda**

5. Agora, adicione o sinal que contém o valor do registro  $t_3$  (ou seja, o registro número 28,  $x_{28}$ ). Expanda a hierarquia em **swerv** para **dec** → **arf** → **gpr\_banks(0)** → **gpr(28)** e clique no módulo **gprff** (ele será destacado conforme mostrado na figura a seguir), selecione o sinal *dout[31:0]* (que mostra o conteúdo do registro  $x_{28}$ , usado no exemplo *AL\_Operations.S*) e arraste-o para o painel Waves preto (Figura 13).



**Figura 13. Adicionar o sinal *dout[31:0]* ao gráfico**

6. Outra via para mostrar sinais no GTKWave é usar um ficheiro *.tcl*. O ficheiro *gtkwave\_signals.tcl* é fornecido em *[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab2/*. Abra esse ficheiro e analise-o. Em cada linha, verá o caminho e o nome de cada sinal que queremos mostrar no gráfico.

```
gtkwave::addSignalsFromList rvfpgasim.clk
gtkwave::addSignalsFromList
rvfpgasim.swervolf.swerv_wrapper_verilog_0.swerv_ehl_2.swerv.ifu.ifu_i0_instr
gtkwave::addSignalsFromList
rvfpgasim.swervolf.swerv_wrapper_verilog_0.swerv_ehl_2.swerv.ifu.ifu_i1_instr
gtkwave::addSignalsFromList
rvfpgasim.swervolf.swerv_wrapper_verilog_0.swerv_ehl_2.swerv.dec.arf.gpr_banks(0).gpr(28).gprff.dout
```

Para usar o ficheiro *.tcl* no GTKWave, basta clicar em *File - Read Tcl Script File* e seleccionar o ficheiro *RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab2/gtkwave+signals.tcl*.

A Figura 14 mostra o programa *AL\_Operations.S* e suas instruções com o código-máquina equivalente.

# RISC-V assembly	# comment (t3 = x28)	# machine code
li t3, 0x0	# t3 = 0	# 0x00000E13
<b>REPEAT:</b>		
addi t3, t3, 6	# t3 = t3 + 6	# 0x006E0E13
addi t3, t3, -1	# t3 = t3 - 1	# 0xFFFFE0E13
andi t3, t3, 3	# t3 = t3 AND 3	# 0x003E7E13
beq zero, zero, REPEAT	# Repete o ciclo	# 0xFE000CE3
nop	# nop	# 0x00000013

**Figura 14. AL\_Operations.S com código de máquina equivalente**

Agora veja como os sinais mudam à medida que o tempo de simulação avança e o programa é executado. Procure que as instruções e *t3* (registro *x28*) se tornem os valores mostrados na Figura 15 à medida que o programa é executado:

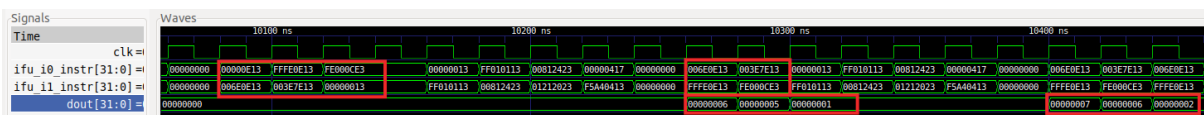
	li t3, 0x0	# t3 = 0	# 0x00000E13
REPEAT:	addi t3, t3, 6	# t3 = 0 + 6 = 6	# 0x006E0E13
	addi t3, t3, -1	# t3 = 5	# 0xFFFFE0E13
	andi t3, t3, 3	# t3 = 5 & 3 = 1	# 0x003E7E13
	beq zero, zero, REPEAT	# Repeat the loop	# 0xFE000CE3
	nop	# nop	# 0x00000013
REPEAT:	addi t3, t3, 6	# t3 = 1 + 6 = 7	# 0x006E0E13
	addi t3, t3, -1	# t3 = 7 - 1 = 6	# 0xFFFFE0E13
	andi t3, t3, 3	# t3 = 6 & 3 = 2	# 0x003E7E13
	beq zero, zero, REPEAT	# Repeat the loop	# 0xFE000CE3
	...		

**Figura 15. Fluxo de instruções e valores do registro *t3* (x28) durante a execução de AL\_Operations**

7. Aproxime o zoom em torno de 10.100 ns, onde analisará a execução das três instruções aritméticas-lógicas da primeira e segunda iterações do ciclo (Figura 16). As duas primeiras instruções (*li t3, 0x0 = 0x00000E13* e *addi t3, t3, 6 = 0x006E0E13*) são carregadas (Fetch) primeiro, uma em cada via do processador RISC-V superescalar, conforme mostrado nos sinais *ifu\_i0\_instr[31:0]* e *ifu\_i1\_instr[31:0]*. As próximas duas instruções (*addi t3, t3, -1 = 0xFFFFE0E13* e *and.i t3, t3, 3*

= 0x003E7E13) são carregadas no próximo ciclo. As duas últimas instruções são carregadas (beq zero, zero, REPEAT = 0xFE000CE3 e nop = 0x00000013) no próximo ciclo.

Devido ao processador pipeline de 9 andares e às dependências do núcleo do SweRV, os efeitos das instruções são vistos oito ou mais ciclos depois que as instruções são carregadas. Oito ciclos depois que a primeira e a segunda instruções são carregadas, x28 (t3) torna-se 0 (o que já era) por causa da primeira instrução: li t3, 0x0 (0x00000E13). Um ciclo depois, x28 é atualizado para 0x6 por causa da próxima instrução: addi t3, t3, 6 (0x006E0E13). Em seguida, x28 é atualizado para 5 por causa da próxima instrução: addi t3, t3, -1 (0xFFFE0E13). Finalmente, x28 é atualizado para 1 por causa da próxima instrução: andi t3, t3, 3 (0x003E7E13). As próximas duas instruções são carregadas: beq zero, zero, REPEAT (0xFE000CE3) e nop (0x00000013); o salto é feito e o ciclo repete-se.



**Figura 16. Execução das três instruções Aritmética-Lógica do exemplo**

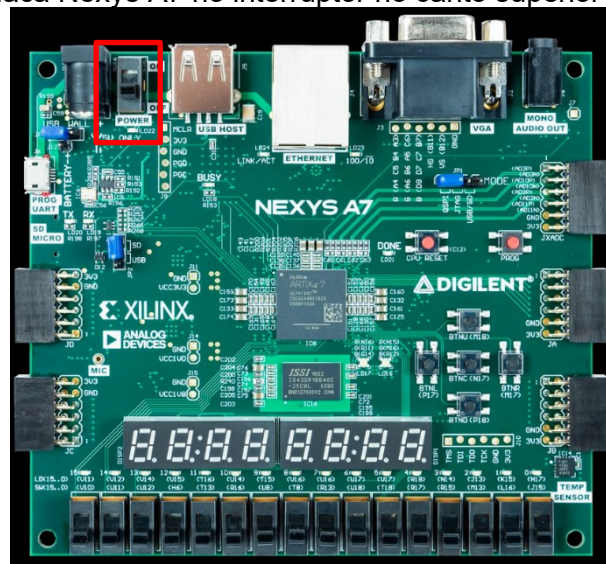
## 5. Execução de um Programa na Placa Nexys A7

Esta seção irá configurar a FPGA com o RVfpgaNexys, com o PlatformIO. Siga as próximas etapas para programar a FPGA com o RVfpgaNexys:

Siga as etapas a seguir:

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

**Etapla 2.** Alimente a placa Nexys A7 no interruptor no canto superior esquerdo. (Figura 17)

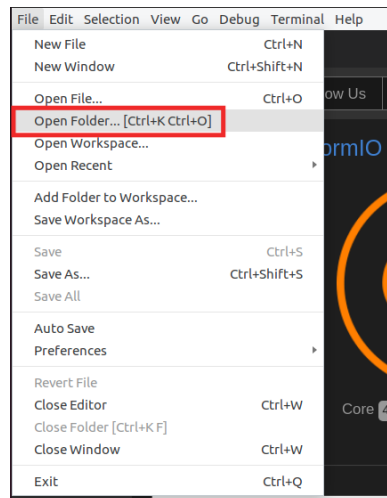


**Figura 17. Botão ON/OFF da placa Nexys A7**

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

**Etapla 4.** Na barra de menu superior, clique em *File* → *Open Folder (abrir pasta)* (Figura 18) e navegue até o diretório

[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab2/examples/



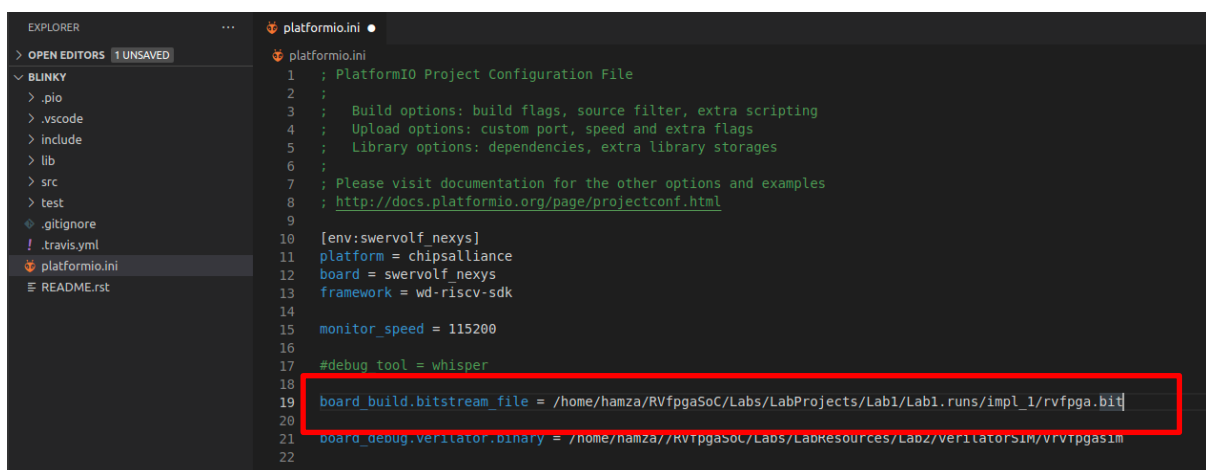
**Figura 18. Abrir pasta**

**Etapa 5.** Selecione o diretório *Blinky* (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 (consulte a Figura 19). Estabeleça o caminho para o fluxo de bits RVfpga em seu sistema editando a seguinte linha (consulte a Figura 19).

**Etapa 7.** O ficheiro "rvfpga.bit" criado com o Vivado Block Design está no seguinte caminho:

```
board_build.bitstream_file =
[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabProjects/Lab1/Lab1.runs/impl_1/
rvfpga.bit
```



**Figura 19. Ficheiro de inicialização do Platformio: platformio.ini**

Há muitos comandos diferentes que podem ser usados no ficheiro de configuração do projeto (*platformio.ini*) e sobre os quais pode encontrar informações em <https://docs.platformio.org/en/latest/projectconf/>.


**Etapa 8.** Clique no ícone PlatformIO  na faixa de opções do menu esquerdo (consulte a

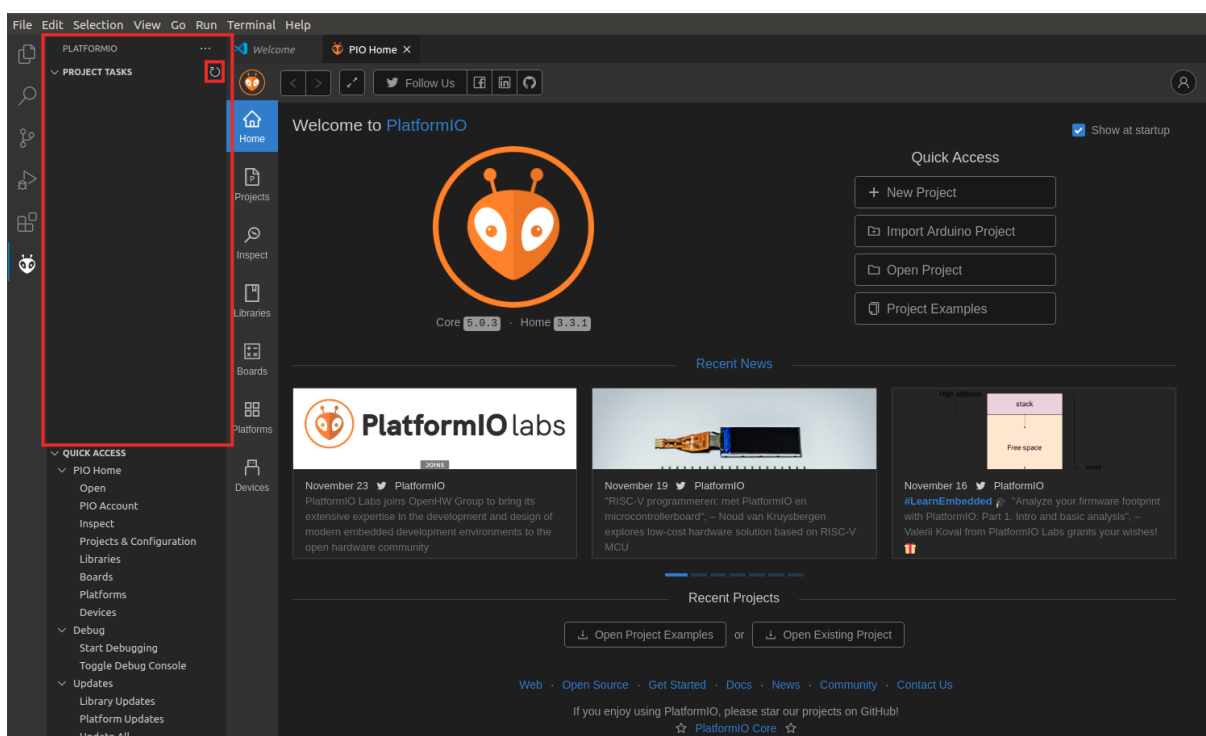


Figura 20).



**Figura 20. Ícone PlatformIO**

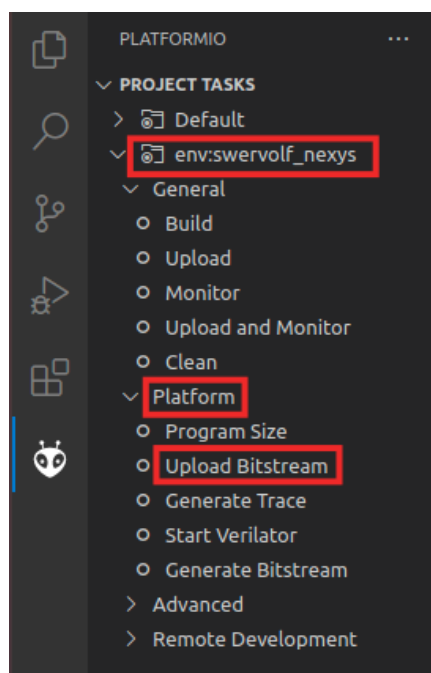
Caso a janela Project Tasks esteja vazia (Figura 21), deve atualizar as Project Tasks primeiro clicando em . Isso pode levar vários minutos.



**Figura 21. Janela PROJECT TASKS vazia - Atualizar**

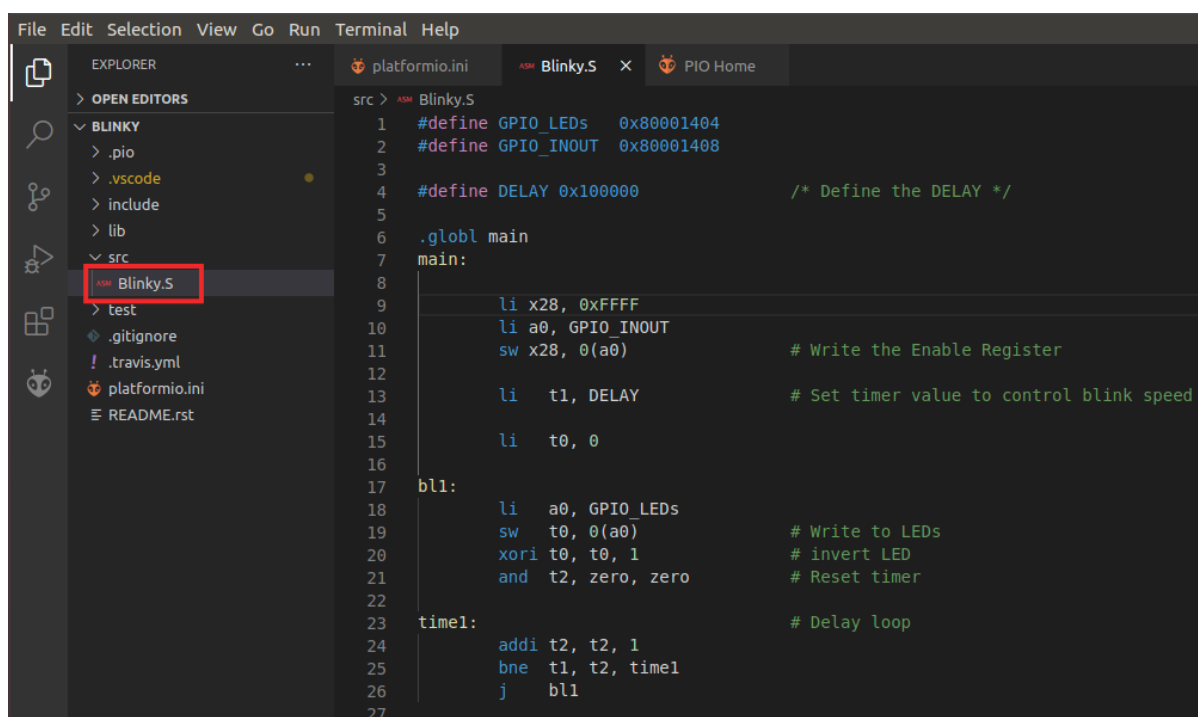
Expanda Project Tasks → env:swervolf\_nexys → Platform e clique em Upload Bitstream, conforme mostrado na Figura 22. **Depois de um ou dois segundos, a FPGA será configurada com o SoC do Block Design** (os mostradores de 7 segmentos disponíveis na placa devem apresentar 8 zeros).








**Figura 22. Carregar bitstream**

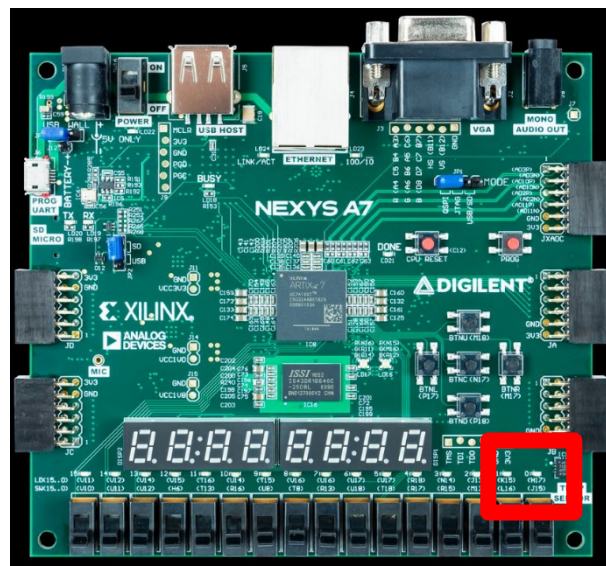
Agora que o bitstream foi carregado, iniciará o processo de depuração.




**Figura 23. blinky.S no PlatformIO**

**Etapla 9.** Clique em  para executar e depurar o programa; em seguida, inicie a depuração clicando no botão play  **PIO Debug**. O PlatformIO define um breakpoint temporário no início da função principal. Portanto, clique no botão Continue (Continuar)  para executar o programa.

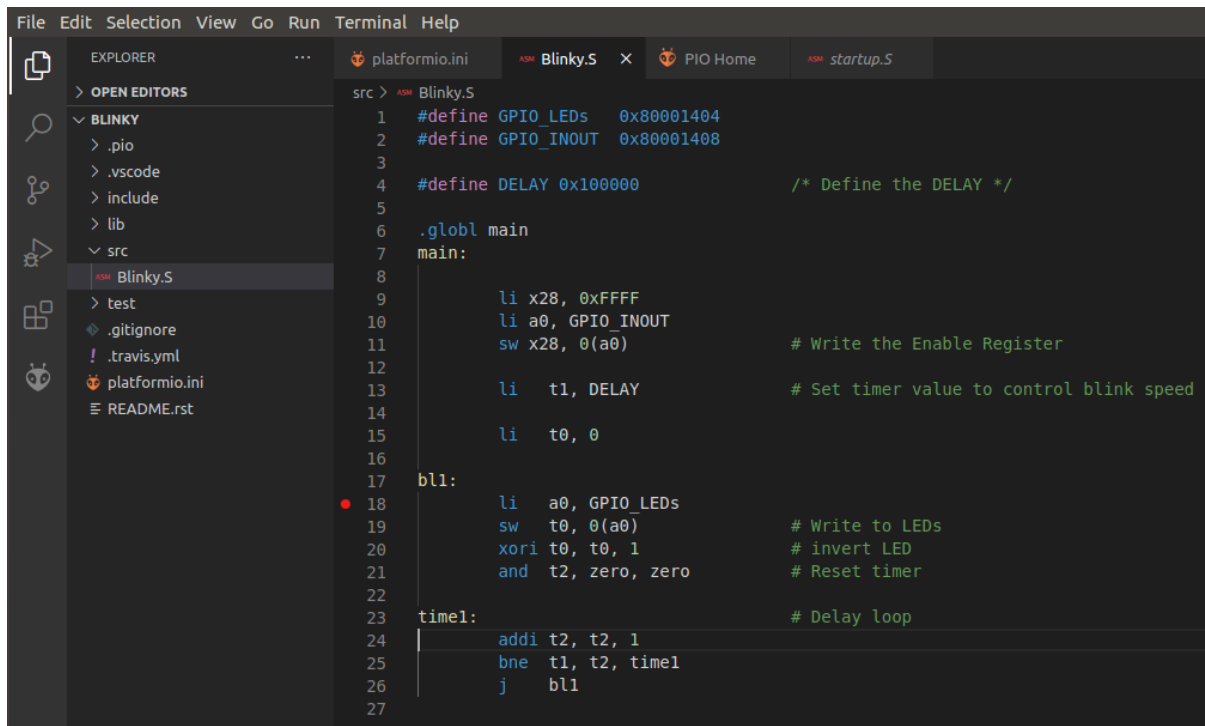
**Etapa 10.** Na placa, irá ver o LED mais à direita começar a piscar.



**Figura 24.** LED mais à direita piscando

**Etapa 11.** Pause a execução clicando no botão de pausa . A execução será interrompida alguns dentro do ciclo infinito (provavelmente, dentro do ciclo de atraso `time1`).

**Etapa 12.** Crie um breakpoint clicando à esquerda da linha número 18. Um ponto vermelho aparecerá e o breakpoint será adicionado à barra BREAKPOINTS (Figura 25).



**Figura 25.** Definição de um breakpoint no `blinky.S`

**Etapa 13.** Em seguida, continue a execução clicando no botão Continue



. A execução continuará e será interrompida após a instrução `store word (sw)`, que escreve 1 (ou 0) no LED mais à direita.

**Etapa 14.** Continue a execução várias vezes. Verá que o valor passado ao LED mais à direita muda de cada vez.



**Etapa 15.** Interrompa a depuração em



do Explorer clicando em . Feche o programa selecionando *File* → *Close Folder* (*Fechar pasta*).

Portanto, executou com êxito o programa de exemplo no RVfpgaSIM e no RVfpgaNexys usando o módulo Block Design que criou no Lab 1.