





THE IMAGINATION UNIVERSITY PROGRAMME

Guia de Introdução ao RVfpga

Agradecimentos



Imagination
university programme

Imagination

AUTHORS
Prof. Sarah Harris
Prof. Daniel Chaver
Zubair Kakakhel
M. Hamza Liaqat

ADVISER
Prof. David Patterson

CONTRIBUTORS
Robert Owen
Olof Kindgren
Prof. Luis Piñuel
Ivan Kravets
Valerii Koval
Ted Marena
Prof. Roy Kravitz

ASSOCIATES
Prof. José Ignacio Gómez
Prof. Christian Tenllado
Prof. Daniel León
Prof. Katzalin Olcoz
Prof. Alberto del Barrio
Prof. Fernando Castro
Prof. Manuel Prieto

Prof. Francisco Tirado
Prof. Román Hermida
Prof. Ataur Patwary
Cathal McCabe
Dan Hugo
Braden Harwood
Prof. David Burnett

Gage Elerding
Prof. Brian Cruickshank
Deepen Parmar
Thong Doan
Oliver Rew
Niko Nikolay
Guanyang He

Sponsors and Supporters



AUTORES

- Prof. Sarah Harris (<https://www.linkedin.com/in/sarah-harris-12720697/>)
- Prof. Daniel Chaver (<https://www.linkedin.com/in/daniel-chaver-a5056a156/>)
- Zubair Kakakhel (<https://www.linkedin.com/in/zubairlk/>)
- M. Hamza Liaqat (<https://www.linkedin.com/in/muhammad-hamza-liaqat-ab73a0195/>)

ORIENTADOR

- Prof. David Patterson (<https://www.linkedin.com/in/dave-patterson-408225/>)

COLABORADORES

- Robert Owen (<https://www.linkedin.com/in/robert-owen-4335931/>)
- Olof Kindgren (<https://www.linkedin.com/in/olofkindgren/>)
- Prof. Luis Piñuel (<https://www.linkedin.com/in/lpinuel/>)
- Ivan Kravets (<https://www.linkedin.com/in/ivankravets/>)
- Valerii Koval (<https://www.linkedin.com/in/valeros/>)
- Ted Marena (<https://www.linkedin.com/in/tedmarena/>)
- Prof. Roy Kravitz (<https://www.linkedin.com/in/roy-kravitz-4725963/>)

ASSOCIADOS

- Prof. José Ignacio Gómez (<https://www.linkedin.com/in/jos%C3%A9-ignacio-gomez-182b981/>)
- Prof. Christian Tenllado (<https://www.linkedin.com/in/christian-tenllado-31578659/>)
- Prof. Daniel León (<https://www.linkedin.com/in/danileon-ufv/>)
- Prof. Katzalin Olcoz (<https://www.linkedin.com/in/katzalin-olcoz-herrero-5724b0200/>)
- Prof. Alberto del Barrio (<https://www.linkedin.com/in/alberto-antonio-del-barrio-garc%C3%ADa-1a85586a/>)
- Prof. Fernando Castro (<https://www.linkedin.com/in/fernando-castro-5993103a/>)
- Prof. Manuel Prieto (<https://www.linkedin.com/in/manuel-prieto-matias-02470b8b/>)
- Prof. Francisco Tirado (<https://www.linkedin.com/in/francisco-tirado-fern%C3%A1ndez-40a45570/>)
- Prof. Román Hermida (<https://www.linkedin.com/in/roman-hermida-correa-a4175645/>)
- Cathal McCabe (<https://www.linkedin.com/in/cathalmccabe/>)
- Dan Hugo (<https://www.linkedin.com/in/danhugo/>)
- Braden Harwood (<https://www.linkedin.com/in/braden-harwood/>)
- David Burnett (<https://www.linkedin.com/in/david-burnett-3b03778/>)
- Gage Elerding (<https://www.linkedin.com/in/gage-elering-052b16106/>)
- Brian Cruickshank (<https://www.linkedin.com/in/bcruiksh/>)
- Deepen Parmar (<https://www.linkedin.com/in/deepen-parmar/>)
- Thong Doan (<https://www.linkedin.com/in/thong-doan/>)
- Oliver Rew (<https://www.linkedin.com/in/oliver-rew/>)
- Niko Nikolay (<https://www.linkedin.com/in/roy-kravitz-4725963/>)
- Guanyang He (<https://www.linkedin.com/in/guanyang-he-5775ba109/>)
- Prof. Ataur Patwary (<https://www.linkedin.com/in/ataurpatwary/>)

Notas Sobre a Tradução para Português

Este curso RVfpga foi traduzido para português pós-acordo ortográfico. Vale a pena referir que para os públicos do Brasil e de Portugal, mesmo depois do novo acordo ortográfico, não existe uma única concordância possível para vários termos e expressões. Existem vários termos técnicos que continuam a ser diferentes, por exemplo: registos e registros, e ficheiro e arquivo. Como o curso RVfpga é oferecido com os documentos editáveis, é possível substituir automaticamente todas as expressões para adaptar o texto com a terminologia preferida. Uma forma possível de realizar esta substituição é explicada em https://gregmaxey.com/word_tip_pages/vba_find_and_replace.html.

Português de Portugal	Português do Brasil
Registo	Registro
Banco de Registos (do CPU)	Banco de Registradores (do CPU)
Ficheiro	Arquivo
Comutação	Chaveamento
Eletrónica	Eletrônica

Este curso foi traduzido por Rui Policarpo Duarte, professor de Arquiteturas de Computadores do Instituto Superior de Engenharia de Lisboa (ISEL) / Instituto Politécnico de Lisboa (IPL), e também investigador no INESC-ID. Possui um doutoramento em Engenharia Elétrica e Electrónica pelo Imperial College London, e um mestrado em Engenharia Eletrotécnica de e Computadores pelo Instituto Superior Técnico. Os seus principais interesses de investigação são em sistemas digitais reconfiguráveis e sistemas embebidos.

<https://www.linkedin.com/in/ruiapduarte/>

Índice

Agradecimentos	2
Notas Sobre a Tradução para Português	3
0. PREFÁCIO	5
1. INTRODUÇÃO	6
2. GUIA DE INICIAÇÃO RÁPIDA	10
3. RESUMO DA ARQUITETURA RISC-V	20
4. RESUMO DO SISTEMA RVFPGA	22
5. INSTALAÇÃO DAS FERRAMENTAS DE DESENVOLVIMENTO	40
6. CONFIGURAR E PROGRAMAR O RVfpgaNexys	46
7. SIMULAÇÃO COM O VERILATOR	76
8. SIMULAÇÃO COM O WHISPER	82
9. APÊNDICES	85



0. PREFÁCIO

O Curso de arquitetura de computadores RVfpga fornece uma compreensão prática de um processador RISC-V comercial, um SoC RISC-V, e de um ecossistema RISC-V. Este curso faculta a compreensão do Sistema, desde o projeto digital e sinais subjacentes até à arquitetura do conjunto de instruções, e do processador até ao ambiente de desenvolvimento de programas, código de arranque e o compilador. O facto de os estudantes do Curso RVfpga adquirirem uma compreensão de alto a baixo do sistema RISC-V é extraordinário. Não só têm um SoC RISC-V e um ecossistema operacionais, como também conhecimentos para usar e expandir o processador e o sistema RISC-V para futuros projetos e investigação.

O Professor David Patterson, que partilha o prémio Turing (ACM A.M.) com John Hennessy pelas suas contribuições para o RISC afirma, “RISC-V está a transformar o projeto de processadores e o co-projeto de software/hardware. O RISC-V é uma arquitetura aberta, que possibilita realizações de código-aberto. Essa nova opção significa que o desenvolvimento de software pode ocorrer junto com o desenvolvimento de hardware, acelerando a execução do projeto. O curso RVfpga aperfeiçoa a compreensão não apenas dos processadores RISC-V, mas também do ecossistema RISC-V e dos SoCs RISC-V. Este curso fornece uma compreensão profunda de uma arquitetura de processador e sistema de categoria industrial com crescente popularidade, e que será útil nas carreiras académicas e industriais.”

1. INTRODUÇÃO

O RISC-V FPGA, também conhecido como RVfpga, é um pacote que inclui instruções, ferramentas, e laboratórios com vista a ter um processador RISC-V comercial num dispositivo real, uma matriz de portas programáveis em campo/*Field Programmable Gate Array* (FPGA), e num simulador, experimentando-o e expandindo-o para estudar arquiteturas de computadores, projeto de sistemas digitais, sistemas embebidos e programação.

Este Guia de Introdução ao RVfpga é composto pelas seguintes secções, descritas brevemente de seguida:

- **Guia de Iniciação Rápida** (Secção 2)
- **Fundamentos e Resumo**
 - **Arquitetura RISC-V** (Secção 3)
 - **O Sistema RVfpga** (Secção 4)
- **Usando o Sistema RVfpga Físico**
 - **Instalação das Ferramentas** (Secção 5)
 - **Configuração e programação do Sistema RVfpga** (Secção 6)
- **Simulando o Sistema RVfpga**
 - **Usando o Verilator**, Simulador HDL (Secção 7)
 - **Usando o Whisper**, Simulador de Arquiteturas da Western Digital (Secção 8)
- **Apêndices**
 - **Usando o conjunto de ferramentas RISC-V e o OpenOCD** (Apêndice A)
 - **Instalando os drivers Windows para usar o PlatformIO** (Apêndice B)
 - **Instalando o Verilator e o GTKWave em Windows** (Apêndice C)
 - **Instalando o Verilator e o GTKWave em macOS** (Apêndice D)
 - **Usando o Vivado para configurar o Sistema RVfpga numa FPGA** (Apêndice E)
 - **Exemplo: Usando o RVfpga numa aplicação IoT industrial** (Apêndice F)

O Guia de Iniciação Rápida (Seção 2) descreve a instalação mínima de software necessária para o RVfpga, e mostra como descarregar e executar um programa de exemplo simples no Sistema RVfpga. Para compreender o RVfpga de melhor forma, salte a Secção 2 e comece o guia completo, que começa na Secção 3.

A Seção 3 fornece uma breve introdução à arquitetura de computador RISC-V. A Seção 4 descreve o Sistema RVfpga (Seção 4.A – 4.C) e a organização dos ficheiros Verilog que compõem o sistema (Seção 4.D). O Sistema RVfpga é baseado no SoC SweRVolf (<https://github.com/chipsalliance/Cores-SweRVolf>) que, por sua vez, usa o núcleo RISC-V SweRV EH1 em código-aberto (<https://github.com/chipsalliance/Cores-SweRV>) da Western Digital (WD). A Figura 1 e a Tabela 1 ilustram a organização hierárquica do Sistema RVfpga, desde o Núcleo SweRV EH1 até RVfpgaNexys e RVfpgaSim.

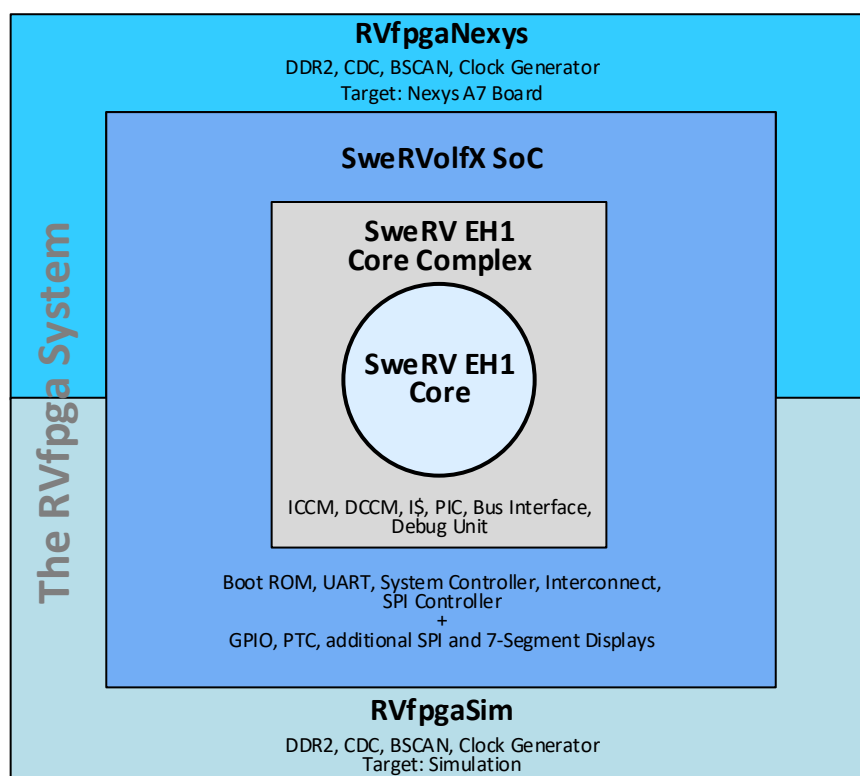


Figura 1. Hierarquia do sistema RVfpga

Tabela 1. Hierarquia do sistema RVfpga

Nome	Descrição
SweRV EH1 Core	Núcleo RISC-V comercial em código-aberto desenvolvido pela Western Digital (https://github.com/chipsalliance/Cores-SweRV).
SweRV EH1 Core Complex	Núcleo SweRV EH1 com memória extra (ICCM, DCCM, e cache de instruções), controlador de interrupção programável (PIC), interfaces de barramentos, e unidade de depuração (https://github.com/chipsalliance/Cores-SweRV).
SweRVolfX (SweRVolf Aumentado)	O System-on-Chip que usamos no curso RVfpga. É uma extensão do SweRVolf. <u>SweRVolf</u> (https://github.com/chipsalliance/Cores-SweRVolf): Um SoC de código-aberto construído em torno do SweRV EH1 Core Complex. Contém uma boot ROM, interface UART, controlador de sistema, barramentos de interconexão (AXI Interconnect, Wishbone Interconnect e ponte AXI-to-Wishbone), e um controlador SPI. <u>SweRVolfX</u> : Acrescenta 4 periféricos novos ao SweRVolf: um GPIO, um PTC, um SPI adicional e um controlador para o mostrador de 8 dígitos de 7 segmentos.
RVfpgaNexys	O SoC SweRVolfX foi realizado para a placa Nexys A7 com os respectivos periféricos. Ele acrescenta uma interface DDR2, uma unidade CDC (cruzamento de domínio de relógio), lógica BSCAN (para a interface JTAG) e um gerador de relógio. O RVfpgaNexys é o mesmo que o SweRVolf Nexys (https://github.com/chipsalliance/Cores-SweRVolf), sendo o segundo baseado no SweRVolf.
RVfpgaSim	O SoC SweRVolfX com um <i>testbench wrapper</i> e uma memória AXI destinada a simulação.

	RVfpgaSim é o mesmo que o SweRVolfSim, (https://github.com/chipsalliance/Cores-SweRVolf), sendo o segundo baseado no SweRVolf.
--	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

As seções restantes mostram como usar o sistema RVfpga no circuito (RVfpgaNexys) e simulação (RVfpgaSim). A seção 5 mostra como instalar as ferramentas de software necessárias para usar o RVfpga. A Seção 6 mostra como usar o PlatformIO para configurar o RVfpgaNexys na placa Nexys A7 FPGA (Seção 6.A) e descarregar e executar vários programas de exemplo nela (Seção 6.B-6.H). As seções 7 e 8 mostram como simular o RVfpgaSim usando o Verilator (seção 7), um simulador HDL de código-aberto, e como usar o Whisper (seção 8), o simulador de conjunto de instruções RISC-V da Western Digital (ISS).

Por fim, os apêndices mostram como usar o RVfpga na linha de comandos do Linux (Apêndice A), como instalar os *drivers* e programas necessários em sistemas Windows e macOS (Anexos B-D), e como usar o Vivado para configurar o RVfpgaNexys numa FPGA usando o Vivado (Apêndice E). O último apêndice, Apêndice F, mostra como usar o RVfpga numa aplicação industrial IoT (Apêndice F).

A Tabela 2 lista os programas e os circuitos necessários para o RVfpga. Este guia mostra como instalar e usar essas ferramentas no sistema operativo Ubuntu 18.04. Outros sistemas operativos (como o Windows ou o macOS), têm passos semelhantes (se não exatamente os mesmos). Quando as instruções diferem, são adicionadas instruções específicas para **Windows** e **macOS** usando estes destaques.

Nota: se não tiver acesso a uma placa FPGA Nexys A7, os laboratórios poderão ser concluídos em simulação usando o Whisper, um simulador de conjunto de instruções (ISS) da Western Digital, e o Verilator, um simulador HDL em código-aberto. Neste caso, não é necessário instalar o Vivado (Seção 5.A); só é necessário instalar o VSCode/PlatformIO (como explicado na Seção 2.A) e o Verilator/GTKWave (como explicado na Seção 5.C).

Tabela 2. Ferramentas necessárias para o RVfpga

Programas		
Nome	Website	Custo
Vivado 2019.2 WebPACK	https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/2019-2.html	grátis
VSCode	https://code.visualstudio.com/Download	grátis
PlatformIO	https://platformio.org/ Instalado dentro do VSCode	grátis
Verilator (simulador HDL) e GTKWave	https://github.com/verilator/verilator http://gtkwave.sourceforge.net/	grátis
Whisper (Simulador de Conjunto de Instruções RISC-V da Western Digital)	https://github.com/chipsalliance/SweRV-ISS Instalado dentro do PlatformIO	grátis
Conjunto de ferramentas RISC-V e OpenOCD	https://github.com/riscv/riscv-gnu-toolchain https://github.com/riscv/riscv-openocd Instalado dentro do PlatformIO	grátis
Circuitos		
Nome	Website	Custo

Placa Nexys A7 FPGA*	https://store.digilentinc.com/nexys-a7-fpga-trainer-board-recommended-for-ece-curriculum/	\$265 (preço acadêmico: \$199)
RISC-V Core e System-on-Chip (SoC)**		
Nome	Website	Custo
SweRV EH1 Core da Western Digital	https://github.com/chipsalliance/Cores-SweRV	grátis
SweRVolf	https://github.com/chipsalliance/Cores-SweRVolf	grátis

*Todas as etapas descritas neste guia também funcionam na placa FPGA Nexys4 DDR da Digilent.

** Fornecido na transferência do RVfpga da Imagination Technologies.

Conhecimentos Prévios Desejados:

Antes de iniciar este curso RVfpga, que inclui este Guia de Introdução ao RVfpga e os Laboratórios RVfpga, espera-se que os estudantes tenham uma compreensão elementar dos seguintes tópicos:

- Projeto de Sistemas Digitais
- Programação em C
- Programação em Assembly
- Arquiteturas de conjunto de instruções
- Microarquiteturas de um Processador
- Sistemas de memória

Estes tópicos são abordados no livro *Digital Design and Computer Architecture: RISC-V Edition*, Harris & Harris, © Morgan Kaufmann 2021. Outros livros didáticos, incluindo *Computer Organization and Design RISC-V Edition*, Patterson & Hennessy, © Morgan Kaufmann 2017, abordam alguns desses tópicos.

2. GUIA DE INICIAÇÃO RÁPIDA

Esta seção mostra como instalar as ferramentas mínimas necessárias para usar o RVfpga e, em seguida, mostra como usar o PlatformIO para configurar o RVfpgaNexys na placa Nexys A7 FPGA e executar um programa no RVfpgaNexys. Para o fazer precisará de adquirir a placa FPGA (Tabela 2). Estes passos também funcionam para a placa Nexys4-DDR FPGA, uma versão anterior da placa.

- A. Instalação mínima: VSCode, PlatformIO e drivers da placa Nexys A7**
- B. Configurar o RVfpgaNexys numa FPGA e executar programas nele**

As instruções abaixo são para um sistema Ubuntu 18.04. Elas também servem para os sistemas operativos Windows 10 e macOS – quando as instruções diferirem do Ubuntu, são adicionadas instruções específicas para **Windows** e **macOS**. Se usar o Ubuntu, pode simplesmente ignorar estas instruções. Os caminhos são escritos como caminhos do Linux usando barras (/), mas os caminhos do Windows são normalmente os mesmos, mas com barras invertidas (\).

A. Instalação mínima: VSCode, PlatformIO e drivers da placa Nexys A7

Nesta etapa, irá instalar os programas e os *drivers* mínimos necessários para usar o RVfpga. Primeiro, instalará o ambiente de programação e, em seguida, os *drivers* da placa Nexys A7 FPGA.

Instalação do VSCode e do PlatformIO: Irá usar o PlatformIO, um ambiente de desenvolvimento integrado (IDE) para configurar o RVfpgaNexys na placa Nexys A7 e para descarregar e executar programas no RVfpgaNexys. O PlatformIO foi desenvolvido como uma extensão do Visual Studio Code (VSCode) da Microsoft. PlatformIO é multiplataforma e inclui um depurador (*debugger*) integrado.

Siga estes passos para instalar o VSCode e o PlatformIO:

1. Instalar o VSCode:
 - a. Descarregue o ficheiro de instalação no endereço:
<https://code.visualstudio.com/Download>
 - b. Abra um terminal, e instale e execute o VSCode:

```
cd ~/Downloads
sudo dpkg -i code*.deb
code
```

Windows / macOS: O VSCode está também disponível para Windows (ficheiro .exe) e macOS (ficheiro .zip) em <https://code.visualstudio.com/Download>. Siga os passos habituais para instalar e executar uma aplicação nestes sistemas operativos.

2. Instalar o PlatformIO sobre o VSCode:

- a. Instalar os utilitários python3 escrevendo o seguinte num terminal:

```
sudo apt install -y python3-distutils python3-venv
```

Windows / macOS: este passo (2.a) não é necessário em Windows. Para o macOS, pode usar o *homebrew* para instalar o python3: `brew install python3`


- b. Se ainda não estiver aberto, inicie o VSCode selecionando o botão Iniciar e escrevendo “VSCode” Na caixa de pesquisa, selecione VSCode ou escreva `code` num terminal Ubuntu.
- c. No VSCode, clique no ícone Extensions  localizado na barra da esquerda do VSCode (Figura 2).



Figura 2. Ícone de extensões do VSCode

- d. Escreva *PlatformIO* na caixa de pesquisa e instale o *IDE PlatformIO* carregando no botão de instalação ao seu lado (Figura 3).

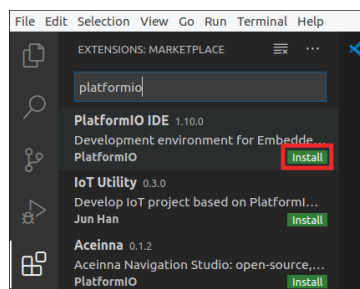


Figura 3. Extensão PlatformIO IDE

- e. A janela OUTPUT no fundo irá informar sobre o processo de instalação. Assim que terminar, clique em “Reload Now” na janela no fundo à direita, e o PlatformIO será instalado dentro do VSCode (Figura 4).

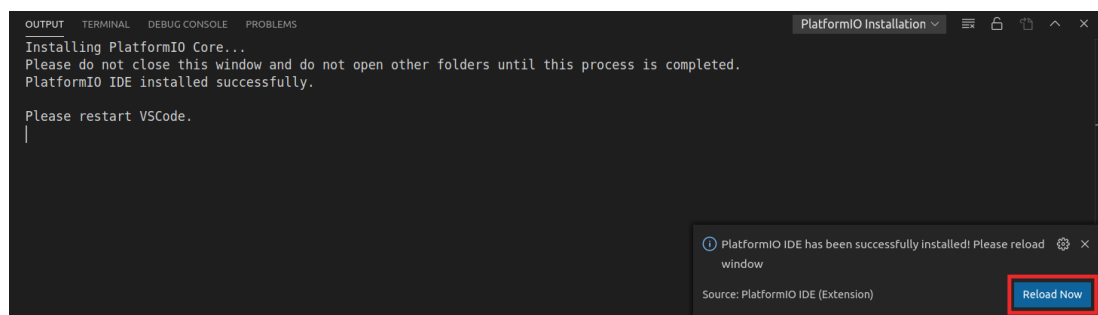


Figura 4. Após instalação do PlatformIO carregar em “Reload Now”

Instalação dos drivers da Nexys A7: precisa instalar manualmente os drivers para a placa Nexys A7.

- Abra um terminal.

- Entre na pasta `[RVfpgaPath]/RVfpga/driversLinux_NexysA7`. (Para simplificar, os *drivers* são fornecidos dentro da pasta RVfpga. Ao instalar o Vivado, na Seção 5 deste guia, pode também encontrar esses *drivers* dentro do ficheiro transferido, conforme descrito nessa seção.)
- Execute o script de instalação:

```
chmod 777 *
sudo ./install_drivers
```
- Desligue a placa Nexys A7 do computador e reinicie o computador para que as alterações surtam efeito.

Windows: siga as instruções fornecidas no Apêndice B para instalação dos *drivers* da placa Nexys A7.

macOS: não é necessário instalar nenhum *driver* adicional.

B. Configurar o RVfpgaNexys numa FPGA e executar programas no RVfpgaNexys

Agora irá configurar o RVfpgaNexys, o sistema RISC-V realizado para FPGA, para a placa FPGA Nexys A7. Embora não o modifiquemos neste Guia de introdução, o Verilog que descreve o sistema RVfpga está disponível em `[RVfpgaPath]/RVfpga/src`. Explicaremos o código-fonte do sistema RVfpga na Seção 4 deste guia, e com mais detalhes nos Labs RVfpga 6-20. Irá modificar o sistema RVfpga em alguns dos exercícios desses laboratórios.

Coloque em funcionamento o RVfpgaNexys na placa FPGA Nexys A7 completando os seguintes passos:

Passo 1. Conecte a placa FPGA Nexys A7 ao computador e ligue a placa

Passo 2. Execute o PlatformIO e o programa em C

Passo 3. Configura o RVfpgaNexys na placa Nexys A7

Passo 4. Descarregue e corra o programa no RVfpgaNexys

Passo 1. Conecte a placa FPGA Nexys A7 a um computador e ligue a placa

Conecte a placa Nexys A7 ao seu computador usando o cabo USB fornecido. A Figura 5 mostra as localizações físicas dos LEDs e interruptores na placa FPGA Nexys A7, bem como o conector USB, interruptores, botões e mostradores de 7 segmentos. Conecte um cabo entre uma porta USB do seu PC e o conector USB da placa Nexys A7, e ligue a placa (On Switch).

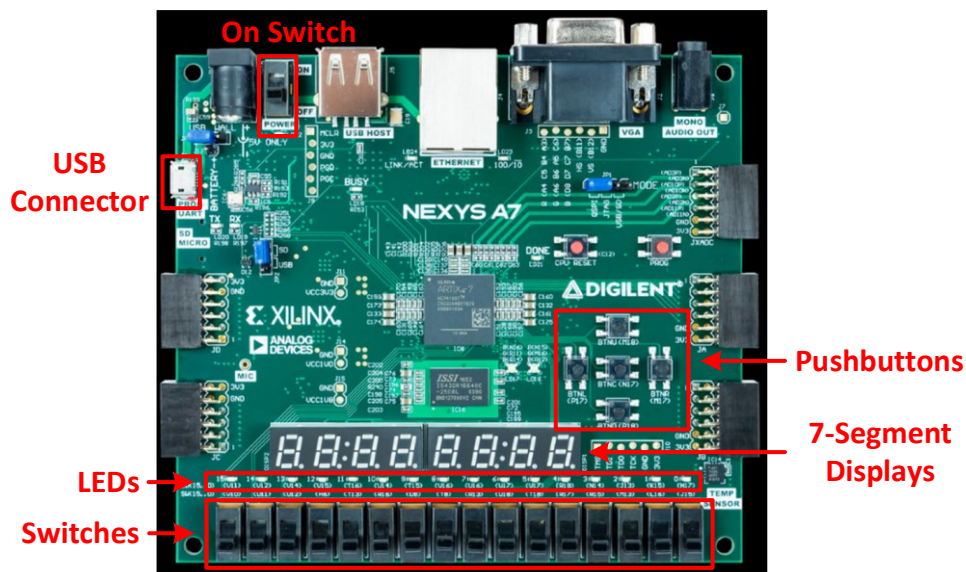


Figura 5. Interfaces de E/S da placa FPGA Nexys A7 da Digilent
(imagem da placa copiada de <https://reference.digilentinc.com/>)

Passo 2. Execute o PlatformIO e o programa em C

Execute o Visual Studio Code (VSCode) escrevendo VSCode no Menu Iniciar (Figura 6) ou escrevendo `code` num terminal.

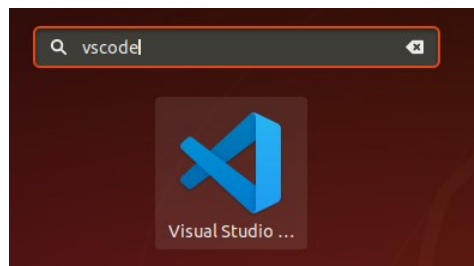



Figura 6. Executar o VSCode

Se a janela PlatformIO Home (PIO Home) não abrir automaticamente, clique no ícone PlatformIO no menu da barra de opções à esquerda: . Em seguida, expanda o PIO Home e clique em Open. Agora o PIO Home abrirá na janela de boas-vindas (Figura 7).

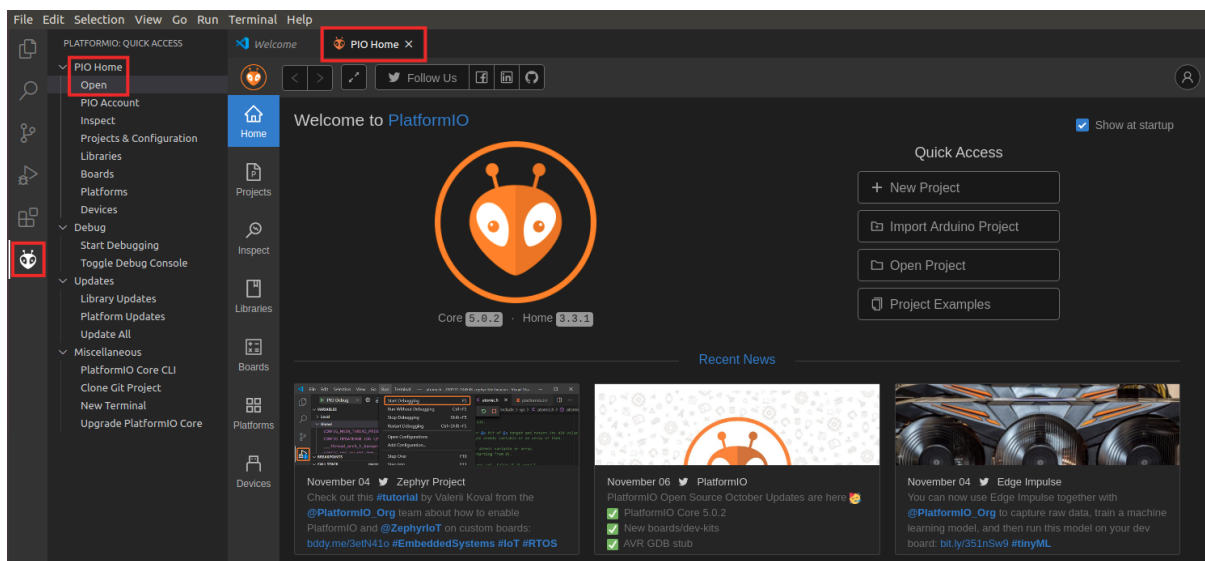


Figura 7. Abrir o PIO Home

Agora clique em File → Open Folder no menu superior e selecione:
[RVfpgaPath]\RVfpga\examples\LedsSwitches_C-Lang

Selecione a pasta, mas não a abra (Figura 8). PlatformIO irá abrir o programa LedsSwitches_C-Lang, que lê os estados dos interruptores na placa Nexys A7 e registra os seus valores nos LEDs da placa.

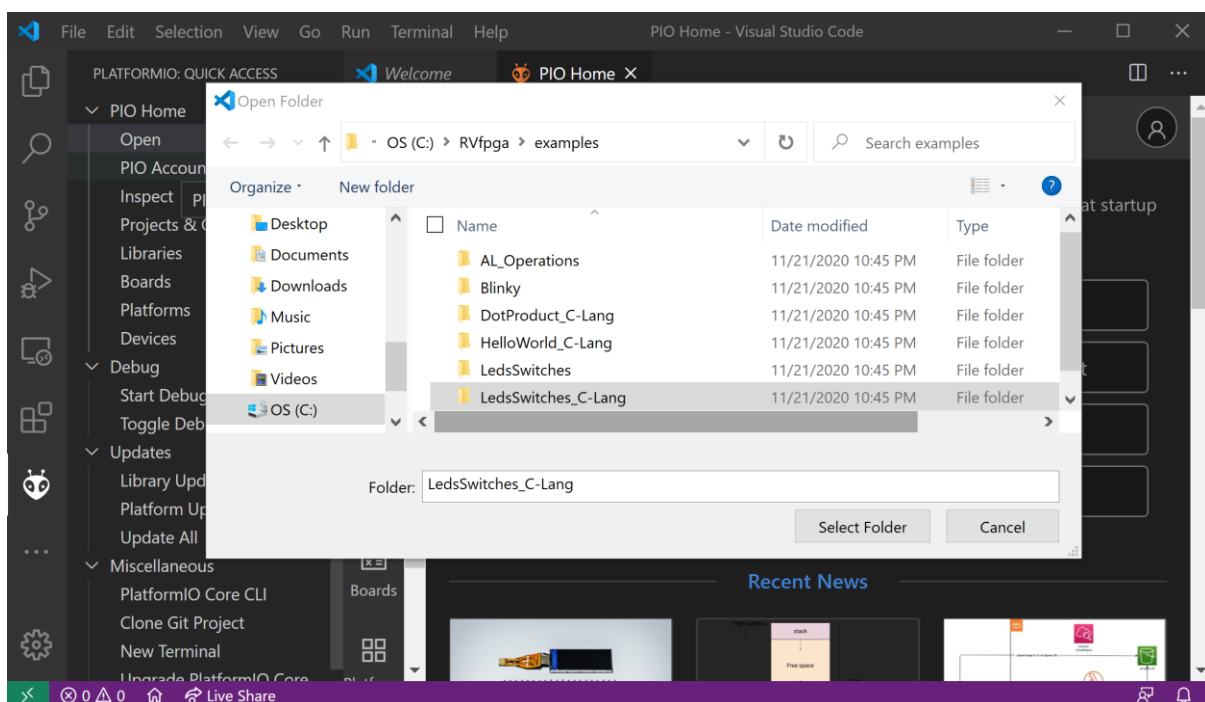
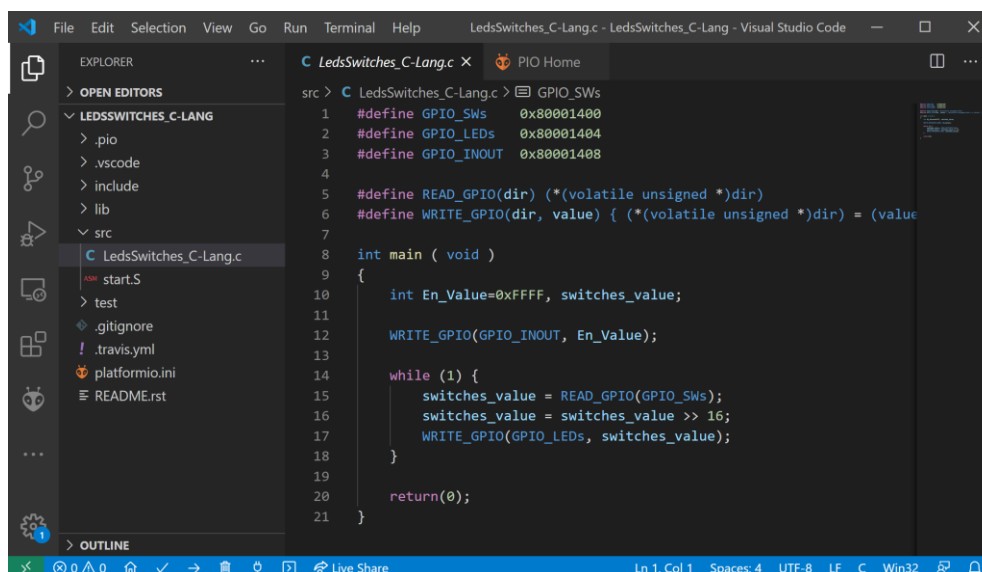


Figura 8. Abrir o exemplo LedsSwitches_C-Lang

Pode visualizar o programa LedsSwitches_C-Lang expandindo a pasta *src* e clicando duas vezes em *LedsSwitches_C-Lang.c* (Figura 9). Discutiremos este programa em detalhe posteriormente no Guia de Introdução. Neste Guia de Iniciação Rápida, simplesmente descarregaremos este programa para o RVfpgaNexys, que estará configurado na placa Nexys A7.



```

src > C LedsSwitches_C-Lang.c x PIO Home
1 #define GPIO_SWS 0x80001400
2 #define GPIO_LEDS 0x80001404
3 #define GPIO_INOUT 0x80001408
4
5 #define READ_GPIO(dir) (*(volatile unsigned *)dir)
6 #define WRITE_GPIO(dir, value) { (*(volatile unsigned *)dir) = (value); }
7
8 int main ( void )
9 {
10     int En_Value=0xFFFF, switches_value;
11
12     WRITE_GPIO(GPIO_INOUT, En_Value);
13
14     while (1) {
15         switches_value = READ_GPIO(GPIO_SWS);
16         switches_value = switches_value >> 16;
17         WRITE_GPIO(GPIO_LEDS, switches_value);
18     }
19
20     return(0);
21 }
    
```

Figura 9. Programa LedsSwitches_C-Lang.c

Na primeira vez que um exemplo RVfpga é aberto no PlatformIO, a plataforma Chips Alliance é instalada automaticamente (isto pode ser visualizado em PIO Home → Platforms, tal como ilustrado na Figura 10). Esta plataforma inclui várias ferramentas que usará posteriormente, como o conjunto de ferramentas RISC-V pré-construído, o OpenOCD para RISC-V, o ficheiro binário (*bitfile*) do RVfpgaNexys, o RVfpgaSim, scripts JavaScript e Python, e vários exemplos. Se, por qualquer motivo, a plataforma Chips Alliance não estiver instalada automaticamente, poderá instalá-la manualmente, conforme explicado na Seção 6.A.

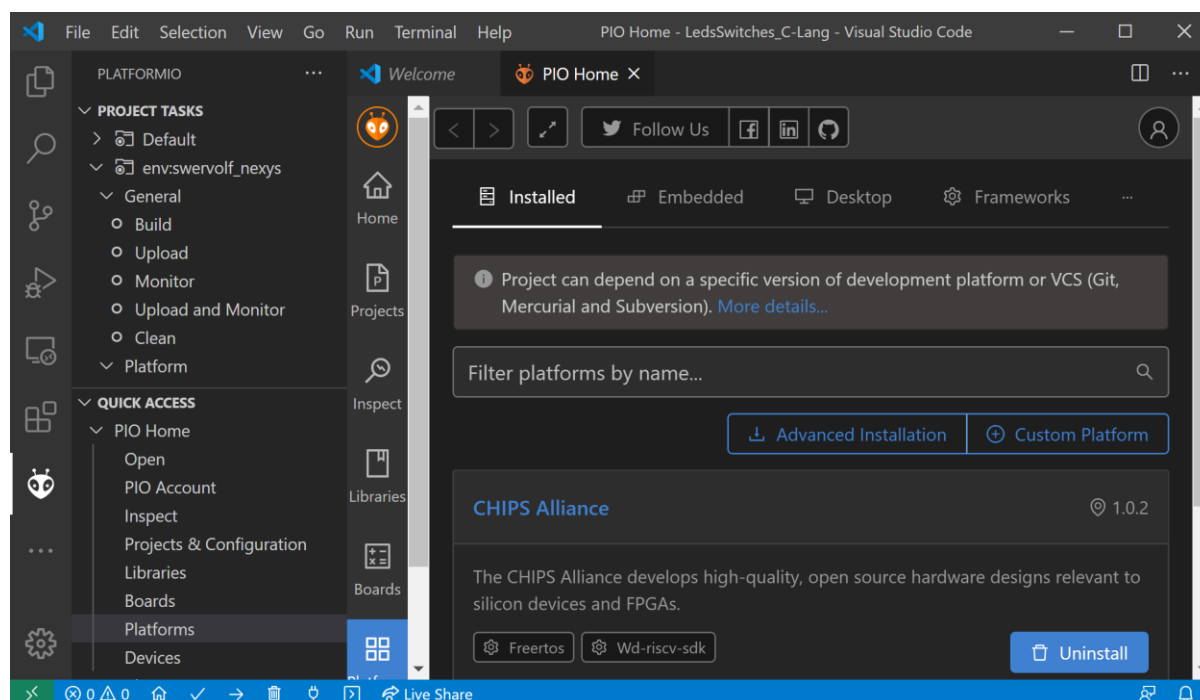



Figura 10. Plataforma Chips Alliance instalada no PlatformIO

Passo 3. Configurar o RVfpgaNexys na placa Nexys A7

Agora está pronto para configurar o RVfpgaNexys, o RISC-V SoC que inclui um processador RISC-V com suporte para periféricos. Abra o ficheiro `platformio.ini` (ficheiro de inicialização do PlatformIO) clicando duas vezes nele na janela EXPLORER, conforme mostrado na Figura 11. (Se a janela do Explorer ainda não estiver aberta, abra-a clicando em  no menu da barra de opções à esquerda.) Agora, adicione o caminho para o local do *bitfile* que define o RVfpgaNexys substituindo o caminho `board_build.bitstream_file` pelo seu próprio caminho (Figura 11):

```
board_build.bitstream_file = [RVfpgaPath]/RVfpga/src/rvfpganexys.bit
```

Guarde o ficheiro `platformio.ini` pressionando Ctrl-s.

Existem vários comandos para o ficheiro de configuração do projeto (*platformio.ini*); mais informações sobre essas opções estão disponíveis em:

<https://docs.platformio.org/en/latest/projectconf/>.

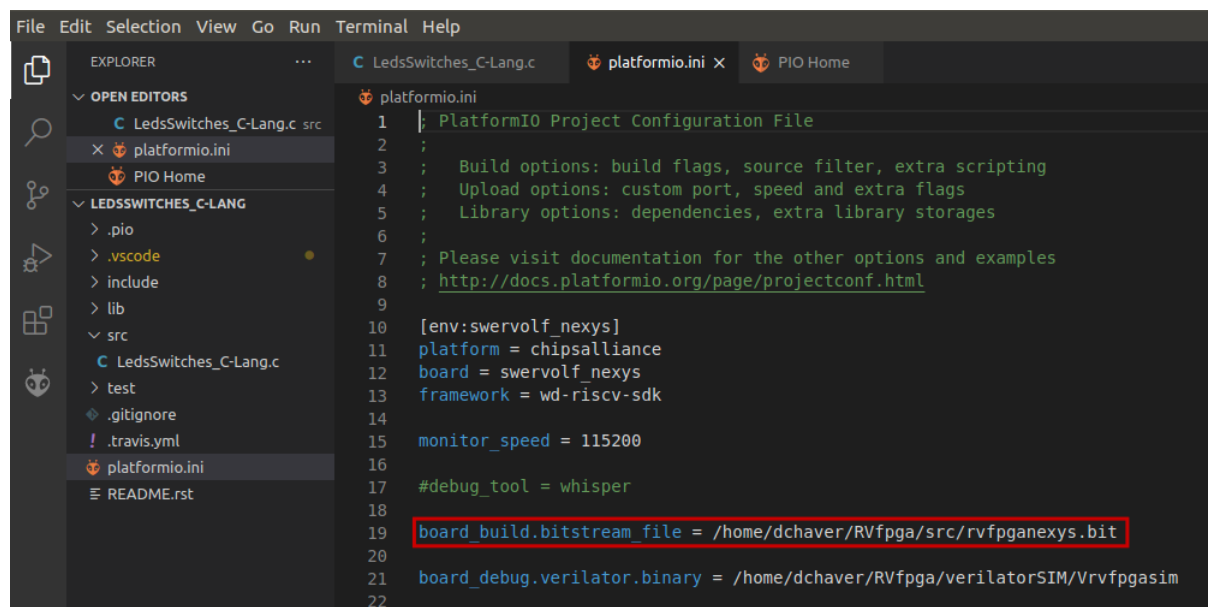


Figura 11. Adicionar o caminho do ficheiro binário (bitfile) RVfpgaNexys

Configure o RVfpgaNexys (tal como especificado por este *bitfile*) na placa Nexys A7:



- Clique no ícone PlatformIO  no menu da barra à esquerda (Figura 12).



Figura 12. Ícone do PlatformIO

- Caso a janela Tarefas do Projeto (Project Tasks) esteja vazia (Figura 13), deverá atualizar as Tarefas do Projeto clicando em . Isso pode demorar vários minutos.

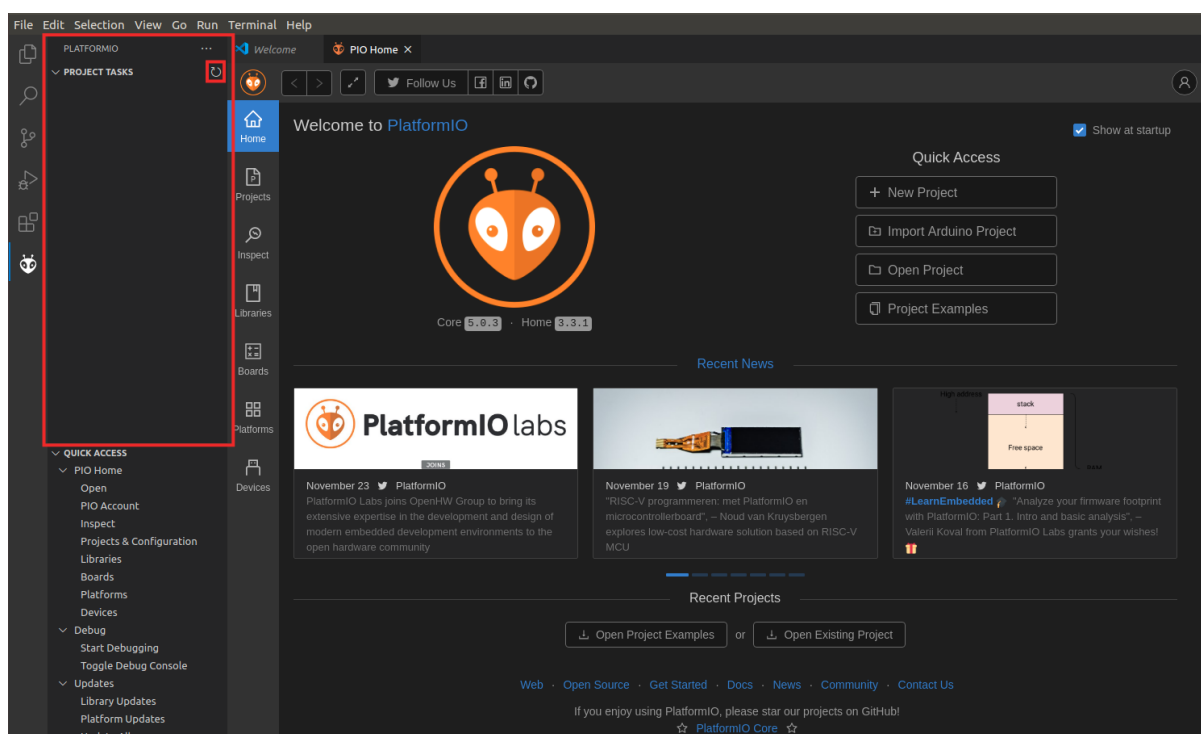


Figura 13. Janela PROJECT TASKS vazia – A atualizar

- Em seguida, expanda Project Tasks → env:swervolf_nexys → Platform e clique em “Upload Bitstream”, tal como ilustrado na Figura 14. **Após um ou dois segundos, a FPGA será configurada com o SoC RVfpgaNexys.**

Por omissão, o processador começa a executar instruções no endereço 0x80000000, onde se encontra a memória de arranque (Boot ROM) no nosso SoC (Tabela 6). A Boot ROM é inicializada com um programa (*boot_main.mem*) que faz piscar os LEDs e os mostradores de 7-segmentos quatro vezes, depois desliga todos os LEDs, e escreve 0s nos 8 mostradores de 7-segmentos, e por fim fica com

a execução presa num ciclo vazio. Este programa encontra-se na pasta: *[RVfpgaPath]/RVfpga/src/SweRVolfSoC/BootROM/sw*. Se quiser alterá-lo e recompilá-lo, faça-o conforme a explicação dada no Apêndice A – Secção III (note que o ficheiro *boot_main.mem* é apenas uma cópia do ficheiro *boot_main.vh*). No Lab 1 mostraremos como a Boot ROM é inicializada com este programa ao criar o *bitstream*.

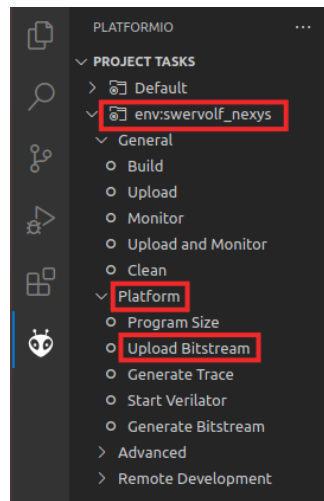

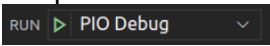


Figura 14. Carregamento da configuração na FPGA (*bitstream*)

Passo 4. Descarregar e executar o programa no RVfpgaNexys

Agora que o RVfpgaNexys foi configurado na placa Nexys A7, pode-se descarregar o programa para a memória do RVfpgaNexys e executar/depurar o programa. Clique no botão “Executar e Depurar”: , que está disponível na barra lateral à esquerda. Inicie o depurador clicando no botão “play”  (certifique-se de que a opção “PIO Debug” está selecionada). Encontra este botão próximo ao topo da janela (Figura 15). O programa irá compilar primeiro e, em seguida, a depuração será iniciada.

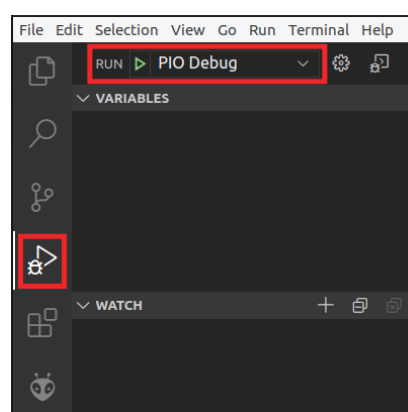



Figura 15. Compilar, descarregar o programa e iniciar o depuramento

Para controlar sua sessão de depuração, pode usar a barra de ferramentas de depuração que aparece na parte superior do editor (Figura 16). Descreveremos e testaremos todas as opções posteriormente neste Guia de Introdução.



Figura 16. Ferramentas de depuração

O PlatformIO cria um ponto de interrupção (*breakpoint*) temporário no início da função principal. Clique no botão “Continue”  para executar o programa. Manipule os interruptores na placa FPGA Nexys A7 e observe como os LEDs correspondentes acendem.

3. RESUMO DA ARQUITETURA RISC-V

O RISC-V é uma Arquitetura de Conjunto de Instruções “Instruction Set Architecture” (ISA) criada em 2011 no Par Lab da Universidade da Califórnia, Berkeley. O objetivo era que o RISC-V se tornasse um “ISA Universal” para processadores usados para toda a gama de aplicações, desde pequenos dispositivos IoT, limitados e de poucos recursos até supercomputadores. Os projetistas do RISC-V estabeleceram cinco princípios para a arquitetura atingir esse objetivo:

- Deve ser compatível com uma ampla gama de aplicações e linguagens de programação.
- A sua implementação deve ser viável em todas as opções de tecnologia, desde FPGAs a ASICs (circuitos integrados específicos de aplicação), bem como tecnologias emergentes.
- Deve ser eficiente nos vários cenários de microarquitetura, incluindo aqueles que implementam microcódigo ou controlo sequencial, *pipelines* em ordem ou fora de ordem, vários tipos de paralelismo, etc.
- Deve poder ser adaptado a tarefas específicas para atingir o desempenho máximo exigido sem inconvenientes impostos pela própria ISA.
- O seu conjunto de instruções básico deve ser estável e duradouro, oferecendo uma estrutura robusta para desenvolvimento.

O RISC-V é uma norma aberta, na realidade, a sua especificação é de domínio público, e é gerido desde 2015 pela **RISC-V Foundation**, agora chamada **RISC-V International**, uma organização sem fins lucrativos que promove o desenvolvimento de circuitos e programas para arquiteturas RISC-V. Em 2018, a Fundação RISC-V Foundation iniciou uma colaboração contínua com a Linux Foundation e, em março de 2020, a RISC-V Foundation tornou-se a RISC-V International com sede na Suíça. Esta mudança eliminou qualquer preocupação que a comunidade pudesse ter sobre a abertura da norma no futuro. Desde 2020, a RISC-V International é apoiada por mais de 200 participantes notáveis da investigação, academia e indústria, incluindo Microchip, NXP, Samsung, Qualcomm, Micron, Google, Alibaba, Hitachi, Nvidia, Huawei, Western Digital, ETH Zurich, KU Leuven, UNLV e UCM.

O RISC-V é um dos poucos, e provavelmente o único, ISA globalmente relevante criado nos últimos 10-20 anos por ser uma norma aberta e modular, em vez de incremental. A sua modularidade torna-o flexível e elegante. Os processadores implementam um ISA básico e apenas as extensões que são usadas. Esta abordagem modular difere dos ISAs tradicionais, como o x86 ou o ARM, que possuem arquiteturas incrementais, onde os ISAs anteriores são expandidos e cada novo processador deve implementar todas as instruções, mesmo aquelas marcadas como “obsoletas”, para garantir a compatibilidade com programas mais antigos. Por exemplo, o x86, que começou com 80 instruções, agora tem mais de 1300, ou 3600 se se considerar todos os *opcodes* diferentes disponíveis no microcódigo. O grande número de instruções e o requisito de compatibilidade com versões anteriores resultaram em processadores grandes que consomem muita energia e precisam suportar instruções longas, porque a maioria dos *opcodes* curtos, ou instruções pequenas, já estão em uso.

O RISC-V possui quatro opções básicas de ISA: duas versões de 32 bits (versões inteiras e embebidas, RV32I e RV32E) e versões de 64 e 128 bits (RV64I e RV128I), conforme mostrado na Tabela 3. Os módulos ISA marcados como *Ratified* foram ratificados. Os módulos marcados como *Frozen* não devem mudar significativamente antes de serem colocados para ratificação. Aceita-se que os módulos marcados como *Draft* sejam alterados antes da ratificação. A capacidade de construir processadores pequenos é um requisito

particularmente importante para dispositivos de custo, espaço e energia limitados. Extensões de instruções podem ser adicionadas sobre os ISAs básicos para habilitar tarefas específicas, por exemplo, operações de vírgula flutuante, multiplicação, divisão e operações vetoriais. Estas extensões de circuitos especializadas também estão incluídas na norma e são conhecidas pelos compiladores, portanto, habilitar as opções desejadas num compilador permitirá a geração de código binário adaptado. Cada uma dessas extensões é identificada por uma letra que deve ser adicionada ao ISA base para representar a capacidade do circuito implementado, conforme mostrado na Tabela 4. Por exemplo, RVM é a extensão multiplicar/dividir, RVF é a extensão de vírgula flutuante e assim por diante.

Tabela 3. ISAs base do RISC-V

(tabela copiada de <https://riscv.org/technical/specifications/>)

Base	Version	Status
RVWMO	2.0	Ratified
RV32I	2.1	Ratified
RV64I	2.1	Ratified
<i>RV32E</i>	<i>1.9</i>	<i>Draft</i>
<i>RV128I</i>	<i>1.7</i>	<i>Draft</i>

Tabela 4. Extensões padrão do RISC-V

(tabela copiada de <https://riscv.org/technical/specifications/>)

Extension	Version	Status
Zifencei	2.0	Ratified
Zicsr	2.0	Ratified
M	2.0	Ratified
A	<i>2.0</i>	Frozen
F	2.2	Ratified
D	2.2	Ratified
Q	2.2	Ratified
C	2.0	Ratified
<i>Ztso</i>	<i>0.1</i>	<i>Frozen</i>
<i>Counters</i>	<i>2.0</i>	<i>Draft</i>
<i>L</i>	<i>0.0</i>	<i>Draft</i>
<i>B</i>	<i>0.0</i>	<i>Draft</i>
<i>J</i>	<i>0.0</i>	<i>Draft</i>
<i>T</i>	<i>0.0</i>	<i>Draft</i>
<i>P</i>	<i>0.2</i>	<i>Draft</i>
<i>V</i>	<i>0.7</i>	<i>Draft</i>
<i>N</i>	<i>1.1</i>	<i>Draft</i>
<i>Zam</i>	<i>0.1</i>	<i>Draft</i>

A letra G, de “geral”, é usada para indicar a inclusão de todas as extensões MAFD. Repare que uma empresa ou um indivíduo pode desenvolver extensões proprietárias usando *opcodes* que são garantidos estarem livres nos módulos padrão. Isso permite que implementações de terceiros sejam desenvolvidas e comercializadas mais rapidamente.

Por exemplo, uma implementação RISC-V de 64 bits, incluindo todas as quatro extensões ISA gerais mais *Bit Manipulation* e *User Level Interrupts*, é conhecida como um ISA RV64GBN. Todos estes módulos estão especificados como sendo sem privilégios ou de utilizador. A RISC-V International abrange também um conjunto de requisitos e instruções para operações privilegiadas necessárias para correr sistemas operativos de uso geral.

4. RESUMO DO SISTEMA RVFPGA

Nesta seção descrevemos todo o sistema RVfpga desde o núcleo até à interface da placa FPGA. A Figura 17 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, a interface do sistema e da placa.

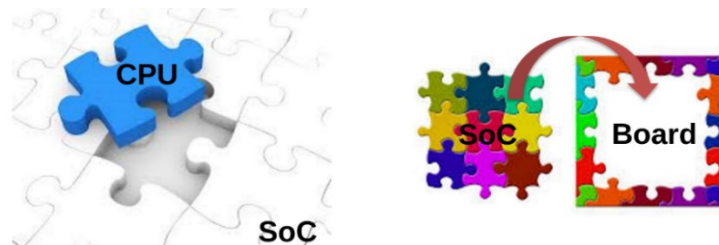


Figura 17. Organização de um Sistema Embebido

A Figura 1 e a Tabela 1 mostram a organização hierárquica do nosso sistema, desde o núcleo SweRV EH1 Core até ao RVfpgaNexys e ao RVfpgaSim. Nas seções seguintes, descrevemos o núcleo do processador (**SweRV EH1 Core da Western Digital**), que executa as instruções RISC-V; então, na Seção B, descrevemos o **SweRVolfX SoC**, que integra os componentes de circuitos do sistema (núcleo, memória e entrada/saída), e as extensões realizadas para uso dentro do RVfpga; na Seção C descrevemos o SoC SweRVolfX implementado na placa FPGA Nexys A7 (**RVfpgaNexys**) e descreve também o SoC SweRVolfX usado na simulação (**RVfpgaSim**). Por fim, explicamos a estrutura de ficheiros de todo o Sistema RVfpga, na Seção D.

A. SweRV EH1 Core e SweRV EH1 Core Complex

A Western Digital desenvolveu três núcleos RISC-V nos últimos anos: **SweRV EH1** (o núcleo usado no Sistema RVfpga), SweRV EH2, e SweRV EL2 (versões futuras do RVfpga poderão incluir estes núcleos). Cada núcleo tem uma licença Apache 2.0. O núcleo SweRV EH1 tem 32 bits e é um superescalar de 2 vias (2-way), e tem um *pipeline* de 9 andares. O núcleo SweRV EH2 baseia-se, e expande, o núcleo SweRV EH1 para adicionar a capacidade de duas tarefas (*dual thread*) simultâneas para aumento de desempenho. O núcleo SweRV EL2 é menor e com um desempenho moderado. A página do RISC-V em <https://www.westerndigital.com/company/innovations/risc-v> descreve os três núcleos disponíveis, cujas principais características são dadas na Tabela 5.

Tabela 5. Principais características dos três núcleos RISC-V da WD

(tabela copiada de <https://www.westerndigital.com/company/innovations/risc-v>)

Core Name	RISC-V Type	Pipeline Stages	Threads	Size @ TSMC	CoreMarks/Mhz
SweRV Core EH1	RV32IMC	9- dual issue	Single	.11mm @ 28nm	4.9
SweRV Core EH2	RV32IMC	9- dual issue	Dual	.067 @ 16nm	6.3
SweRV Core EL2	RV32IMC	4- single issue	Single	.023 @ 16nm	3.6

Dos três núcleos, o **SweRV EH1** (fornecido com o pacote RVfpga e disponível em <https://github.com/chipsalliance/Cores-SweRV>) é preferido pelo seu elevado desempenho/MHz e a sua estrutura de tarefa singular. Além disso, a Chips Alliance, um grupo dedicado a fornecer circuitos em código-aberto, fornece um SoC completo e

verificado, chamado SweRVolf (fornecido com o pacote RVfpga e também disponível em <https://github.com/chipsalliance/Cores-SweRVolf>). O Sistema RVfpga utiliza uma extensão do SweRVolf SoC que, por sua vez, utiliza a versão **1.8** do núcleo **SweRV EH1** da Western Digital.

O **SweRV EH1** é um núcleo de 32 bits que apenas funciona em modo-máquina (M-mode), suporta extensões RISC-V Inteiro (I), instruções comprimidas (C), e operações de multiplicação e divisão (M). O manual de referência do programador (https://github.com/chipsalliance/Cores-SweRV/blob/master/docs/RISC-V_SweRV_EH1_PRM.pdf) descreve em detalhe todos os aspectos do núcleo, desde a sua estrutura até informações de tempo e mapas de memória. O SweRV EH1 é um núcleo superescalar, com um *pipeline dual-issue* de 9 andares (Figura 18) que suporta quatro unidades lógicas aritméticas (ALUs), designadas por EX1 a EX4 em dois *pipelines*, I0 e I1. Ambas as vias do pipeline suportam operações com a ALU. Uma via do *pipeline* suporta leituras e escritas (Load/Store) na memória e a outra via tem um multiplicador com 3 ciclos de latência. O processador também tem um divisor com 34 ciclos de latência fora do pipeline. Existem quatro *Stall Points* no pipeline: 'Fetch 1', 'Align', 'Decode', e 'Commit'. O andar 'Fetch 1' inclui um preditor de saltos do tipo "Gshare". No andar 'Align', as instruções são recuperadas de três *buffers* de carregamento de instruções. No andar 'Decode', até duas instruções de quatro *buffers* de instrução podem ser decodificadas. Na etapa de 'Commit', até duas instruções por ciclo são confirmadas. Por fim, na etapa de 'Writeback', os registos do núcleo são atualizados.

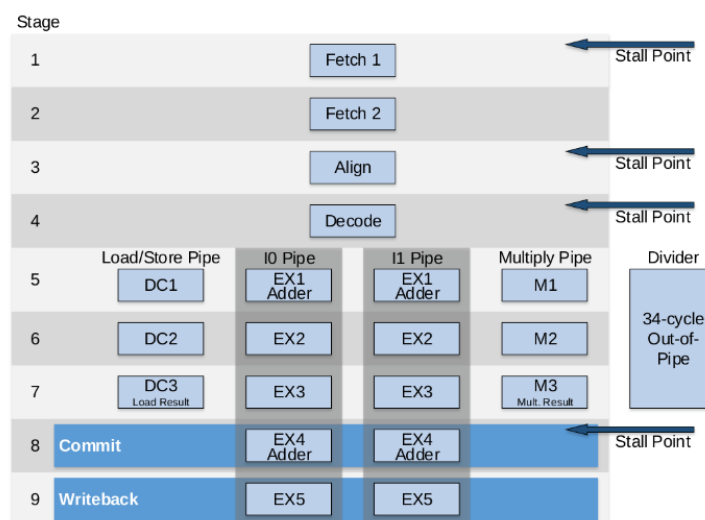


Figura 18. Microarquitetura do núcleo SweRV EH1

(figura copiada de https://github.com/chipsalliance/Cores-SweRV/blob/master/docs/RISC-V_SweRV_EH1_PRM.pdf)

A Figura 19 mostra uma comparação entre os diferentes núcleos e processadores atuais. O desempenho do SweRV EH1 por MHz é impressionantemente alto com 4,9 CM/MHz (CoreMark por MHz): é duas vezes mais rápido que o ARM Cortex A8 e seu desempenho até supera o desempenho do ARM Cortex A15.

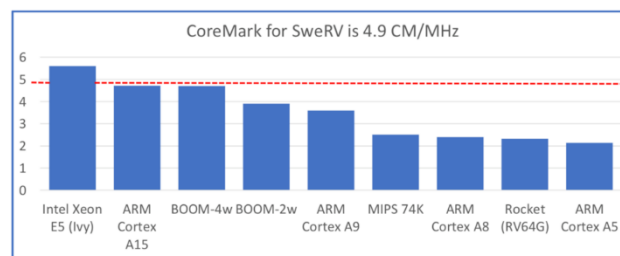


Figura 19. Comparação de desempenho por tarefa e MHz

(figura copiada de https://content.riscv.org/wp-content/uploads/2019/12/12.11-14.20a3-Bandic-WD_SweRV_Cores_Roadmap_v4SCR.pdf)

A Western Digital também fornece uma extensão para o SweRV EH1 chamada de **SweRV EH1 Core Complex** (Figura 20), que acrescenta os elementos ao EH1 descritos acima, e coloridos em azul na figura:

- Duas memórias dedicadas, uma para instruções (ICCM) e outra para dados (DCCM), estão diretamente acopladas ao núcleo. Estas memórias fornecem acessos de baixa latência e proteção SECDED ECC (códigos de correção de erro único e detecção de erro duplo). Cada uma das memórias pode ser configurada com 4, 8, 16, 32, 48, 64, 128, 256 ou 512 KB.
- Uma cache de instruções (opcional) associativa de 4 vias, com verificação de paridade ou proteção ECC.
- Um Controlador de Interrupção Programável (PIC) opcional, que suporta até 255 interrupções externas.
- Quatro interfaces do barramento do sistema para carregamento de instruções (IFU Bus Master), acessos de dados (LSU Bus Master), acessos de depuração (Debug Bus Master) e acessos a DMA externos (Porto de DMA Slave) para memórias diretamente acopladas (configuráveis como AXI4 de 64 bits ou barramento AHB-Lite).
- Unidade de depuração do núcleo (Core Debug Unit) compatível com a especificação RISC-V Debug.

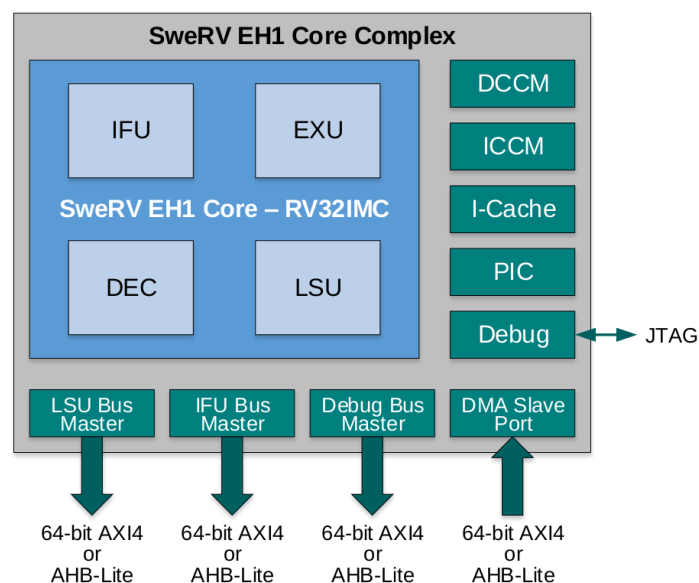


Figura 20. SweRV EH1 Core Complex

(figura copiada de: https://github.com/chipsalliance/Cores-SweRV/blob/master/docs/RISC-V_SweRV_EH1_PRM.pdf)

B. SweRVolfX SoC

O System-on-a-Chip (SoC) usado neste pacote RVfpga, chama-se SweRVolfX e está ilustrado na Figura 21, é baseado na versão 0.7.3 do SweRVolf (<https://github.com/chipsalliance/Cores-SweRVolf/releases/tag/v0.7.3>), que foi construído sobre o SweRV EH1 Core Complex. Em acréscimo ao SweRV EH1 Core Complex (Figura 20), o SweRVolf SoC também inclui a memória ROM de arranque (Boot ROM), uma UART, um Controlador de Sistema e um controlador SPI (a Figura 21 mostra estes elementos a branco). Dado que o SweRV EH1 Core usa um barramento AXI e que os periféricos usam um barramento Wishbone, o SoC também possui uma ponte AXI-Wishbone.

No RVfpga estendemos o SweRVolf SoC com mais algumas funcionalidades, como um controlador SPI (SPI2) adicional, um controlador GPIO (General Purpose Input/Output), um módulo PTC (PWM/Timer/Counter) e um controlador para interface com 8 mostradores de 7 segmentos. A Figura 21 mostra esses novos periféricos em vermelho, exceto o controlador de mostradores de 7 segmentos, que está incluído no Controlador do Sistema. Isto é conhecido como o System-on-a-Chip **SweRVolfX** (o X vem de eXtendido).

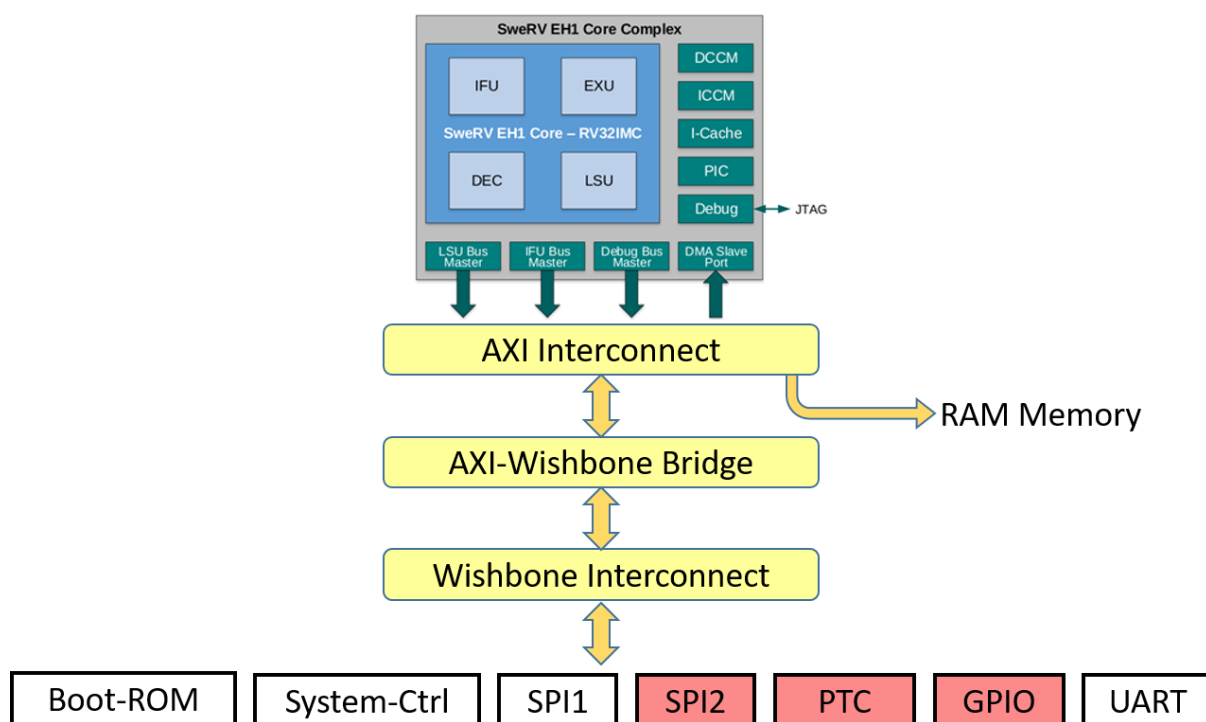


Figura 21. SoC SweRVolfX (SweRVolf eXtended com 4 novos periféricos)

A Tabela 6 mostra os endereços dos periféricos conectados ao núcleo através da interconexão Wishbone e mapeados em memória.

Tabela 6. Endereços dos periféricos do SoC SweRVolfX Extended mapeados em memória

Sistema	Endereço
Boot ROM	0x80000000 - 0x80000FFF
System Controller	0x80001000 - 0x8000103F
SPI1	0x80001040 - 0x8000107F
SPI2	0x80001100 - 0x8000113F
PTC	0x80001200 - 0x8000123F
GPIO	0x80001400 - 0x8000143F
UART	0x80002000 - 0x80002FFF

i. Entrada/Saída

O SweRVolfX SoC usa dois tipos de controladores para comunicação com os periféricos: controladores personalizados escritos em Verilog e controladores de código-aberto da OpenCores [<https://opencores.org/>], uma comunidade *online* dedicada ao desenvolvimento de núcleos (*IP cores*) no espírito de colaboração livre e de código-aberto. O SoC SweRVolfX usado neste curso inclui as interfaces de E/S listadas abaixo, que explicaremos em detalhes e até estenderemos no RVfpga Labs 6-20.

- **Controlador de Sistema:** o controlador do sistema (*System Controller*) contém funcionalidades comuns do sistema, como manter o registo de informação da versão do SoC, estado de inicialização da RAM e o temporizador da máquina RISC-V. Em <https://github.com/chipsalliance/Cores-SweRVolf> encontra o mapa de memória completo. Este módulo foi modificado da seguinte forma:
 - Incluímos um novo controlador para comunicação com os Mostradores de 8 dígitos de 7 segmentos disponíveis na placa Nexys A7, denominado por `SevSegDisplays_Controller`, e incluímos dois novos registos para este controlador mapeados nos endereços 0x80001038 e 0x8000103C.
 - Adicionámos dois registos de 1 bit para atendimento de interrupções do GPIO e do PTC, mapeados no endereço 0x80001018.
 - Removemos os registos GPIO básicos fornecidos pelo SweRVolf e mapeamos nos endereços 0x80001010–0x80001017. Note que adicionamos um controlador GPIO mais completo, conforme descrito em seguida.
- **SPI:** dois controladores SPI de código-aberto (obtidos em https://opencores.org/projects/simple_spi e denominados por SPI1 e SPI2) estão implementados no SweRVolfX. Os seus registos acessíveis (SPI_SPCR, SPI_SPSR, SPI_SPDR, SPI_SPER, SPI_SPSS) estão mapeados entre os endereços 0x80001040 e 0x8000107F (para o SPI1) e entre os endereços 0x80001100 e 0x8000113F (para o SPI2).
- **PTC:** módulo temporizador de <https://opencores.org/projects/ptc>. Os seus registos estão mapeados no intervalo de endereços 0x80001200 a 0x800012FF.
- **GPIO:** controlador de Entrada/Saída (GPIO) de <https://opencores.org/projects/gpio>. Contém 32 portos de E/S mapeados no intervalo de endereços 0x80001400 a 0x800014FF. Cada pino é ligado a um *buffer* de três estados (*tristate buffer*), para que possa ser configurado como entrada ou saída.
- **UART:** controlador UART de código-aberto de <https://opencores.org/projects/uart16550> está disponível no SweRVolfX. Os seus registos acessíveis estão mapeados entre os endereços 0x80002000 e 0x80002FFF.

ii. Memória

O SoC SweRVolfX inclui uma memória ROM de arranque (*Boot ROM*) e os circuitos necessários para permitir que o utilizador inclua memórias RAM e Flash SPI.

- **ROM de arranque:** uma ROM de arranque contém um *first-stage bootloader*. Após arranque do sistema, o SweRVolfX SoC irá carregar as instruções iniciais desta área, que ocupam os endereços 0x80000000 a 0x80000FFF.

- **RAM:** o SoC SweRVofX não inclui um controlador de memória, mas reserva os primeiros 128MiB do seu mapa de memória (0x00000000-0x07FFFFFF) e dá acesso ao barramento AXI, para que o utilizador possa aceder à memória RAM através de um controlador de memória.
- **Flash SPI:** uma memória Flash SPI também pode ser incluída usando o controlador SPI1 descrito na seção anterior (na gama de endereços: 0x80001040-0x8000107F).

iii. Barramentos de Interconexão

O SweRV EH1 Core usa um barramento AXI4 para conectar o núcleo à memória. O barramento também pode ser configurado como barramento AHB-Lite, mas essa opção não é usada neste curso. Todos os periféricos (dispositivos de E/S) são ligados a um barramento Wishbone, um barramento de código-aberto que é muito usado em CPUs e periféricos OpenCore. O sistema inclui uma ponte AXI para Wishbone (como ilustrado na Figura 21) para ligar o núcleo aos periféricos.

Nesta seção, descrevemos brevemente a operação de um barramento AXI4 e um barramento Wishbone. Se estiver interessado em saber mais sobre a especificação desses barramentos, poderá consultar as referências fornecidas abaixo.

O barramento AXI4

O SweRV EH1 Core Complex usa um barramento *AXI4 Interconnect* para comunicação com o mundo exterior (Figura 20). O *Advanced eXtensible Interface* (AXI) é um barramento comum usado por muitos processadores e faz parte da especificação de interconexão on-chip da *Advanced Microcontroller Bus Architecture* (AMBA) ARM.

Nas seções seguintes explicamos brevemente alguns dos principais aspectos do barramento AXI4. Toda a especificação AXI pode ser encontrada no seguinte documento:

https://static.docs.arm.com/ihi0022/e/IHI0022E_amba_axi_and_ace_protocol_spec.pdf

• Principais Características do Barramento AXI

As principais características da tecnologia de barramento AXI são as seguintes:

- É adequado para projetos de elevada largura de banda e baixa latência
- Fornece operações com alta frequência sem usar pontes complexas
- Vai de encontro aos requisitos de interface de uma vasta gama de componentes
- É adequado para controladores de memória com alta latência de acesso inicial
- Proporciona flexibilidade na implementação de arquiteturas de interconexão
- É retrocompatível com as interfaces AHB e APB existentes
- Fornece fases separadas de endereço/controlo e dados
- Inclui suporte para transferências de dados desalinhadas (usando *byte strob*es)
- Permite transações em rajadas indicando apenas o endereço inicial
- Fornece canais de dados de leitura e escrita separados, permitindo DMAs de baixo custo
- Permite que informações de endereço sejam enviadas antes da transferência dos dados
- Suporta o envio de vários endereços pendentes, e conclusão de transações fora de ordem
- Permite adicionar registos de pipeline facilmente, para cumprir com os requisitos de tempo de ciclo

- **Arquitetura AXI**

O protocolo AXI define os seguintes canais de transações independentes:

- Endereço de leitura
- Dados de leitura
- Endereço de escrita
- Dados de escrita
- Resposta de escrita

A Figura 22 mostra como uma transação de leitura usa o endereço de leitura e os canais de dados de leitura. Primeiro, o endereço e os bits de controlo são enviados do dispositivo mestre (*master*), e o escravo (*slave*) responde com os dados no canal de dados lidos (*read data channel*).

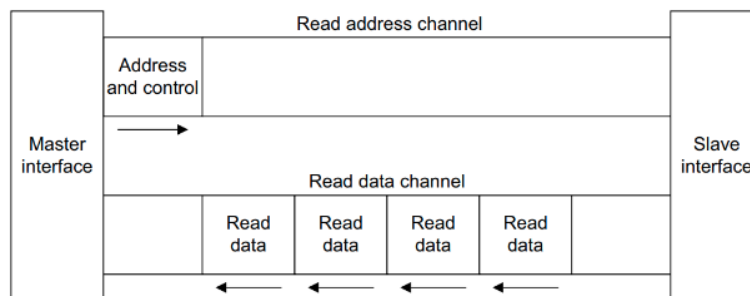


Figura 22. Arquitetura do canal de leitura

(Figura copiada de https://static.docs.arm.com/ihi0022/e/IHI0022E_amba_axi_and_ace_protocol_spec.pdf)

A Figura 23 mostra como uma transação de escrita usa o endereço de escrita, dados de escrita e canais de resposta de escrita. Semelhante a uma leitura, o dispositivo mestre envia o endereço e os bits de controlo. Em seguida, o dispositivo mestre envia os dados no canal de dados de escrita (*write data channel*) e o dispositivo escravo envia uma resposta (*write response*).

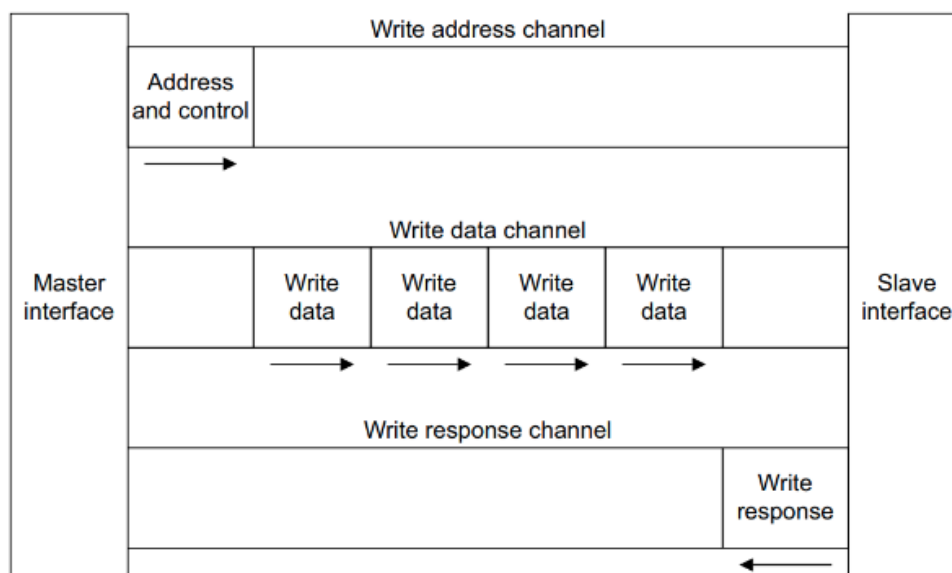


Figura 23. Arquitetura do canal de escrita

(figura copiada de https://static.docs.arm.com/ihi0022/e/IHI0022E_amba_axi_and_ace_protocol_spec.pdf)

O canal do endereço AXI carrega os endereços e as informações de controlo que descrevem a natureza dos dados a serem transferidos. Os dados são transferidos entre o mestre e o escravo usando:

- Um canal de dados de leitura para transferir os dados do escravo para o mestre (Figura 22).
- Um canal de dados de escrita para transferir os dados do mestre para o escravo (Figura 23). Numa transação de escrita, o escravo usa o canal de resposta de escrita para sinalizar a conclusão da transferência para o mestre (Figura 23).

• Sinais AXI

Tabela 7. mostra os principais sinais utilizados no barramento AXI e uma breve descrição de cada um deles. Os sinais são organizados em cinco grupos, que correspondem aos cinco canais descritos na seção anterior:

- Sinais do **Canal do endereço de escrita**, cujos nomes começam com **AW**
- Sinais do **Canal dos dados de escrita**, cujos nomes começam com **W**
- Sinais do **Canal da resposta de escrita**, cujos nomes começam com **B**
- Sinais do **Canal do endereço de leitura**, cujos nomes começam com **AR**
- Sinais do **Canal dos dados de leitura**, cujos nomes começam com **R**

Tabela 7. Sinais AXI

(tabela copiada de https://static.docs.arm.com/ihi0022/e/IHI0022E_amba_axi_and_ace_protocol_spec.pdf)

Signal	Source: master/ slave	Input/ Output	Description
Aclk	Global	Input	Global clock signal.
AResetn	Global	Input	Global reset signal.
AWID[3:0]	Master	Input	Write address ID.
AWADDR[31:0]	Master	Input	Write address.
AWLEN[3:0]	Master	Input	Write burst length.
AWSIZE[2:0]	Master	Input	Write burst size.
AWBURST[1:0]	Master	Input	Write burst type.
AWLOCK[1:0]	Master	Input	Write lock type.
AWCACHE[3:0]	Master	Input	Write cache type.
AWPROT[2:0]	Master	Input	Write protection type.
WDATA[31:0]	Master	Input	Write data.
ARID[3:0]	Master	Input	Read address ID.
ARADDR[31:0]	Master	Input	Read address.
ARLEN[3:0]	Master	Input	Read Burst length.
ARSIZE[2:0]	Master	Input	Read Burst size.
ARLOCK[1:0]	Master	Input	Read Lock type.
ARCACHE[3:0]	Master	Input	Read Cache type.
ARPROT[2:0]	Master	Input	Read Protection type.
RDATA[31:0]	Master	Input	Read data.
WLAST	Master	Input	Write last.
RLAST	Slave	Output	Read last.
AWVALID	Master	Output	Write address valid.
AWREADY	Slave	Output	Write address ready.
WVALID	Master	Output	Write valid.
RAVLID	Slave	Output	Read valid.
WREADY	Slave	Output	Write ready.
BID[3:0]	Slave	Output	Write Response ID.
RID[3:0]	Slave	Output	Read response ID.
BRESP[1:0]	Slave	Output	Write response.
RRESP[1:0]	Slave	Output	Read response.
BVALID	Slave	Output	Write response valid.

Barramento Wishbone

Os periféricos SweRVolfX usam a *Wishbone System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores* (<https://opencores.org/howto/wishbone>). O principal objetivo no uso deste barramento é promover a reutilização do projeto, aliviando os problemas de integração no System-on-Chip. Anteriormente, os núcleos IP usavam esquemas de interconexão sem norma (*ad-hoc*), o que dificultava a sua integração. As interconexões sem norma exigiam a criação de uma lógica de adaptação personalizada adicional para ligar cada um dos núcleos. Ao adotar um esquema de interconexão padrão, como o barramento Wishbone, os núcleos podem ser integrados de forma mais rápida e fácil pelo utilizador final.

- **Principais Características do Barramento Wishbone**

As principais características da tecnologia de barramento Wishbone são as seguintes:

- Suporta metodologias de projeto estruturadas, utilizadas por grandes equipas de projeto.
- Inclui um conjunto completo de protocolos de barramento de transferência de dados populares, incluindo:
 - i. Ciclos de leitura e escrita (READ/WRITE)
 - ii. Ciclos de paragem de transferência (BLOCK)
 - iii. Ciclos de leitura/modificação/escrita (READ/MODIFY/WRITE)
- Fornece barramentos com larguras de dados modulares e tamanhos de operandos até 64 bits.
- Suporta ordenação de dados BIG ENDIAN e LITTLE ENDIAN.
- Suporta vários métodos de interconexão de núcleo, incluindo ponto-a-ponto (*point-to-point*), barramento partilhado (*shared bus*), comutador de barra cruzada (*crossbar switch*) e interconexões em rede (*switched fabric*).
- Inclui protocolos de concordância (*handshaking*) que permitem cada IP Core negociar a sua velocidade de transferência de dados.
- Suporta transferências de dados num único ciclo.
- Suporta finalização normal de ciclo, finalização com nova tentativa e finalização devido a erro.
- Inclui larguras de endereço modulares.
- Fornece um esquema de decodificação de endereço parcial para escravos. Isto facilita a decodificação de endereços a alta velocidade, ao usar menos lógica redundante, e suportar métodos de dimensionamento de endereço variável e de interconexão.
- Oferece suporte para etiquetas (*tags*) definidas pelo utilizador. Elas são úteis para agregar informação a um barramento de dados ou endereço, ou um ciclo de barramento. As etiquetas definidas pelo utilizador são especialmente úteis ao modificar um ciclo de barramento para identificar informações como:
 - i. Transferências de dados,
 - ii. Bits de paridade ou de correção de erros,
 - iii. Vetores de interrupções,
 - iv. Operações de controlo de cache.
- Inclui uma arquitetura Mestre/Escravo para projetos de sistemas flexíveis.
- Possui a capacidade de multiprocessamento (multi-Mestre). Isso permite uma ampla variedade de configurações de SoC.
- Inclui uma metodologia de arbitragem que pode ser definida pelo utilizador final (árbitros de prioridades, rotativo ou *round-robin*, etc.)

• Arquitetura e Sinais Wishbone

A Figura 24 ilustra a conexão habitual entre um mestre (no nosso caso, o SweRV EH1) e um escravo (no nosso caso, um periférico como o GPIO, o SPI...) através de um barramento Wishbone. O barramento Wishbone é muito mais simples que o barramento AXI4 e, como mostrado na Tabela 8, usa menos sinais.

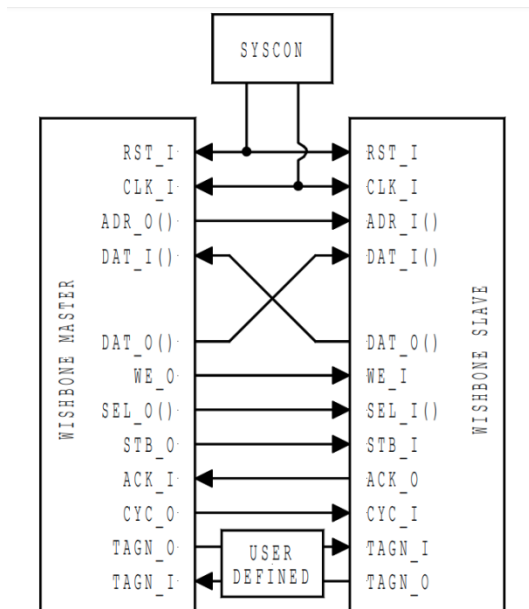


Figura 24. Arquitetura Wishbone

(figura copiada de <https://opencores.org/howto/wishbone>)

Tabela 8. Sinais Wishbone

(tabela copiada de <https://opencores.org/howto/wishbone>)

Signal name	description	Signal name	Description
CLK_O	It coordinates all activities for the internal logic within the WISHBONE interconnect. The INTERCON module connects the [CLK_O] output to the [CLK_I] input on MASTER and SLAVE	CLK_I	All WISHBONE output signals are registered at the rising edge of [CLK_I]. All WISHBONE input signals are stable before the rising edge of [CLK_I].
RST_O	It forces all WISHBONE interfaces to restart. All internal self-starting state machines are forced into an initial state. The INTERCON connects the [RST_O] output to the [RST_I] input on MASTER and SLAVE	DAT_I()	The data input array [DAT_I()] is used to pass binary data. The array boundaries are determined by the port size, with a maximum port size of 64-bits (e.g. [DAT_I(63..0)]).
		DAT_O()	The data output array [DAT_O()] is used to pass binary data. The array boundaries are determined by the port size, with a maximum port size of 64-bits (e.g. [DAT_I(63..0)]).
		RST_I()	The reset input [RST_I] forces the WISHBONE interface to restart
		TGD_I()	Data tag type [TGD_I()] is used on MASTER and SLAVE interfaces. It contains information that is associated with the data input array [DAT_I()], and is qualified by signal [STB_I].
		TGD_O()	Data tag type [TGD_O()] is used on MASTER and SLAVE interfaces. It contains information that is associated with the data output array [DAT_O()], and is qualified by signal [STB_O]

Signal name	Description	Signal name	Description
ACK_I	The acknowledge input [ACK_I], when asserted, indicates the normal termination of a bus cycle	ACK_O	The acknowledge output [ACK_O], when asserted, indicates the termination of a normal bus cycle
CYC_O	The cycle output [CYC_O], when asserted, indicates that a valid bus cycle is in progress	CYC_I	The cycle input [CYC_I], when asserted, indicates that a valid bus cycle is in progress
STALL_I	The pipeline stall input [STALL_I] indicates that current slave is not able to accept the transfer in the transaction queue	STALL_O	The pipeline stall signal [STALL_O] indicates that the slave can not accept additional transactions in its queue
ERR_I	The error input [ERR_I] indicates an abnormal cycle termination	ERR_O	The error output [ERR_O] indicates an abnormal cycle termination
RTY_I	The retry input [RTY_I] indicates that the interface is not ready to accept or send data, and that the cycle should be retried	RTY_O	The retry output [RTY_O] indicates that the interface is not ready to accept or send data, and that the cycle should be retried
STB_O	The strobe output [STB_O] indicates a valid data transfer cycle	STB_I	The strobe input [STB_I], when asserted, indicates that the SLAVE is selected. A SLAVE shall respond to other WISHBONE signals only when this [STB_I] is asserted
WE_O	The write enable output [WE_O] indicates whether the current local bus cycle is a READ or WRITE cycle	WE_I	The write enable input [WE_I] indicates whether the current local bus cycle is a READ or WRITE cycle

C. SoC SweRVolfX na Placa FPGA Nexys A7 e em Simulação

O SoC SweRVolfX (Figura 21) pode ser executado quer (1) na placa FPGA Nexys A7 (ou Nexys4 DDR), cuja configuração é conhecida neste curso como **RVfpgaNexys**, ou (2) em simulação, que é conhecida neste curso como **RVfpgaSim**.

i. RVfpgaNexys

RVfpgaNexys é o SoC SweRVolfX realizado para a placa FPGA Digilent Nexys A7 (Figura 25). RVfpgaNexys é o mesmo que SweRVolf Nexys (<https://github.com/chipsalliance/Cores-SweRVolf>), exceto que o último é baseado no SweRVolf. Os principais elementos usados pelo **RVfpgaNexys** estão representados na Figura 25:

- Circuitos configurados na FPGA:
 - **SoC SweRVolfX** (ilustrado na Figura 21)
 - **Controlador de memória Lite DRAM**
 - **Gerador de Sinal de Relógio (Clock)**: a placa Nexys A7 inclui um oscilador a cristal de **100 MHz** que é usado pelo **Controlador de Memória Lite DRAM**. A frequência deste sinal de relógio é reduzida para **50 MHz** para ser usada pelo **SoC SweRVolfX**.
 - **Módulo de Intersecção de Domínios de Relógio (Clock Domain Crossing)**: ligação entre os dois domínios de relógio: SoC SweRVolfX e DRAM Lite.
 - **Lógica BSCAN para JTAG**: pode encontrar mais informações sobre este módulo em <https://github.com/chipsalliance/Cores-SweRVolf/issues/29>.
- Memória/Periféricos usados no RVfpgaNexys na placa FPGA Nexys A7 (ou Nexys4 DDR):
 - **Memória DDR2** (acessível através do controlador Lite DRAM mencionado acima)
 - **Ligação USB**
 - **Memória Flash SPI**
 - **Acelerómetro SPI**
 - **16 LEDs e 16 Interruptores**
 - **8 mostradores de 7 segmentos**

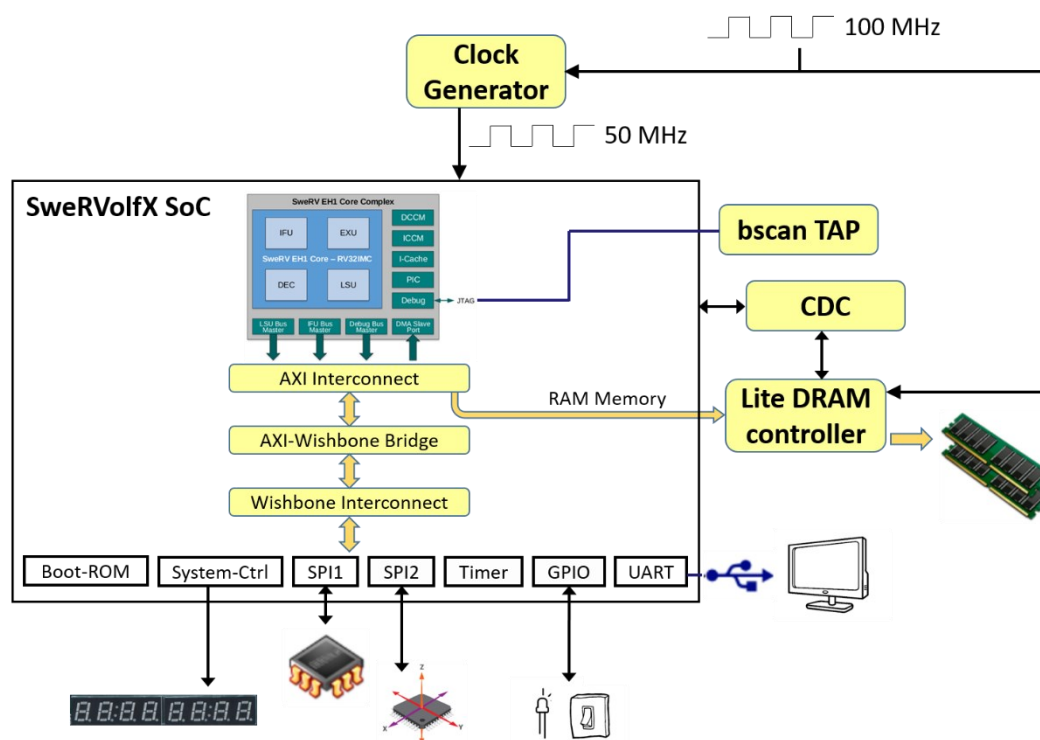


Figura 25. RVfpgaNexys

A placa Nexys A7 (Figura 26) é a recomendada para estudo em cursos de engenharia eletrotécnica e de computadores. Esta placa custa \$265 (preço com desconto para universidades de \$198,75 – é necessária inscrição numa conta Digilent com um endereço de e-mail .edu). A Digilent fornece um manual completo de referência da placa Nexys A7 em: https://reference.digilentinc.com/media/reference/programmable-logic/nexys-a7/nexys-a7_rm.pdf. Esta placa pode ser alimentada a partir de um adaptador de 5V (não fornecido com a placa) ou de um PC através do conector micro-USB na placa. Um microcontrolador Microchip PIC24 gere o processo de configuração da FPGA, tornando esta placa uma opção de fácil utilização. A placa é programável usando o Vivado Design Suite da Xilinx ou o OpenOCD. A configuração desejada pode ser descarregada para a FPGA usando uma das quatro fontes diferentes: um cartão MicroSD formatado em FAT32, uma pen USB formatada em FAT32, a memória flash interna ou uma interface JTAG.

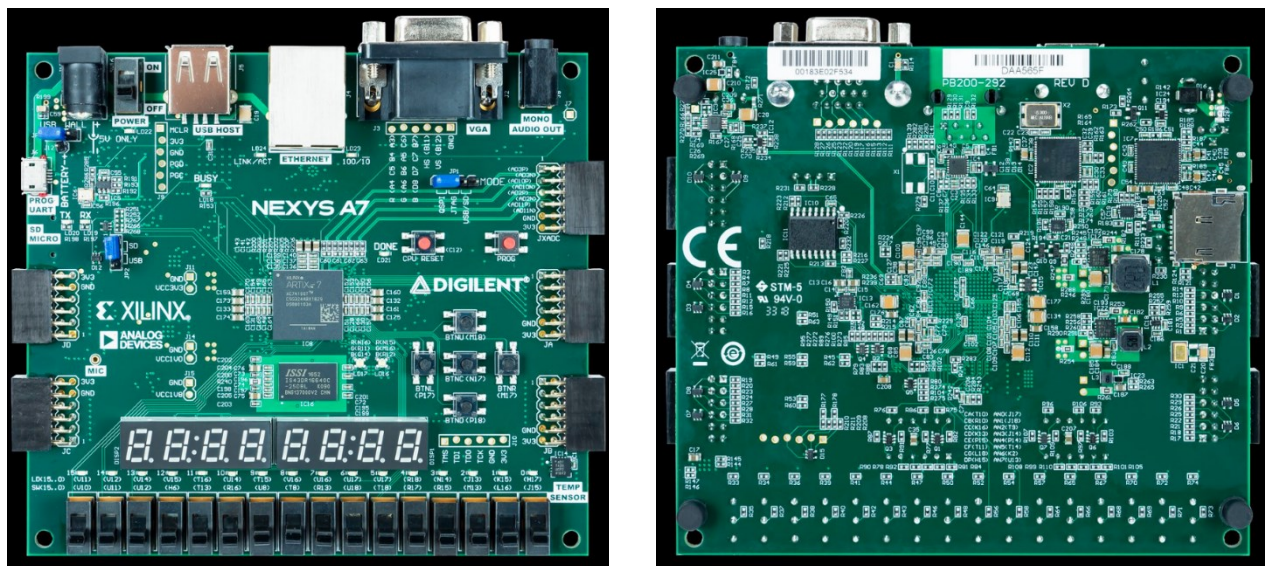


Figura 26. Placa FPGA Nexys A7 da Digilent
(figura copiada de <https://reference.digilentinc.com/>)

A placa FPGA Nexys A7-100T inclui as seguintes interfaces e dispositivos:

- Memória RAM DDR 128 MiB
- Memória Flash SPI 128 Mibit
- 8 Mostradores de 7-Segmentos
- 16 Interruptores
- 16 LEDs
- Sensores e conectores, incluindo um microfone, ficha áudio, saída VGA, porta USB, LEDs RGB, sensor de temperatura I2C, acelerómetro SPI, entre outros.
- FPGA Xilinx Artix-7, que tem as seguintes características:
 - 15.850 *Logic slices* com 4 LUTs de 6 entradas e 8 registos (flip-flops).
 - 4.860 Kibits de *Block RAM*
 - 6 gestores de relógio *clock management tiles* (CMTs)
 - 170 pinos de E/S
 - Frequência do relógio interno até 450 MHz

ii. RVfpgaSim

O SoC SweRVolfX (Figura 21) pode também incluir um *wrapper* em Verilog para permitir a sua simulação. O **RVfpgaSim** é o SoC SweRVolfX encapsulado com um banco de ensaio (*testbench*) para ser utilizado por simuladores HDL. O RVfpgaSim é o mesmo que o SweRVolf sim (<https://github.com/chipsalliance/Cores-SweRVolf>), exceto que este último baseia-se no SweRVolf.

Embora existam muitos simuladores HDL de código-aberto, utilizaremos o Verilator (<https://www.veripool.org/wiki/verilator>). Este simulador HDL aberto e gratuito reconhece Verilog ou SystemVerilog sintetizável e afirma ser o simulador Verilog/SystemVerilog mais rápido. Ele é amplamente utilizado na indústria e no meio académico, fornece suporte imediato dos fornecedores de IPs ARM e RISC-V; e é orientado pela Chips Alliance e pela Linux Foundation.

D. Estrutura de Ficheiros

Nas secções anteriores mostrámos a organização de alto nível do sistema que utilizamos nestes materiais, a partir do **SweRV EH1 Core Complex** (Figura 20), até ao **SoC**

SweRVolfX (Figura 21) e, finalmente, para as implementações **RVfpgaNexys** (Figura 25) e **RVfpgaSim**.

Nesta secção, descrevemos a estrutura do ficheiro de todo o sistema. Enquanto lê estas explicações, abra os ficheiros e visualize-os no seu PC. Os ficheiros estão disponíveis em **[RVfpgaPath]/RVfpga/src**.

i. SweRV EH1 Core Complex

A Figura 27 mostra a estrutura do ficheiro do **SweRV EH1 Core Complex** (Figura 20). O núcleo está organizado em três blocos principais: um *wrapper* SweRV (destacado a cinzento) que inclui o SweRV EH1 Core (destacado a verde) e outros elementos (como o Controlador de Interrupções ou a Unidade de Depuração), e as Memórias de Dados/Instrução e Cache de Instruções (destacado a vermelho).

SweRV EH1 Core Complex (swerv_wrapper_dmi.sv)

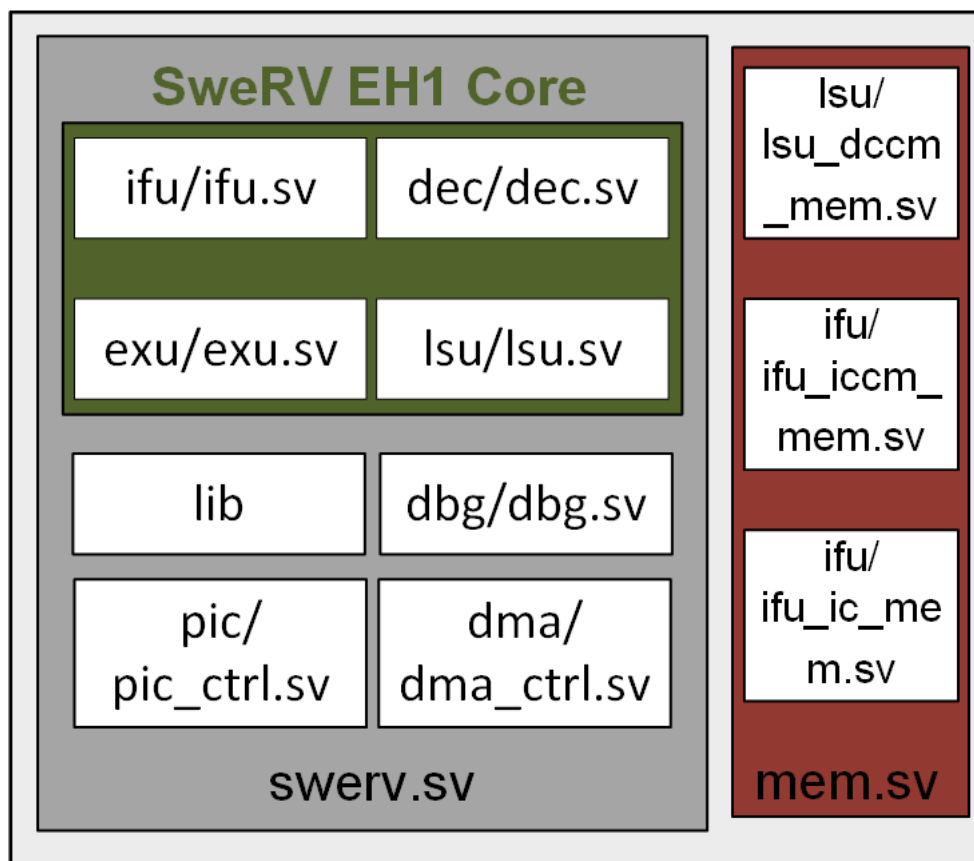


Figura 27. SweRV EH1 Core Complex

Os ficheiros Verilog para o SweRV EH1 Core Complex estão disponíveis na pasta: **[RVfpgaPath]/RVfpga/src/SweRVolfSoC/SweRVEh1CoreComplex**

Procure essa pasta no seu PC, para ver os ficheiros à medida que nos referimos a eles nesta secção.

O ficheiro principal do SweRV EH1 Core Complex é o: *swerv_wrapper.sv*; o módulo de topo chama-se **swerv_wrapper**, e instancia dois módulos que correspondem aos dois blocos destacados a cinzento e vermelho na Figura 27:

- **mem** (implementado no ficheiro *mem.sv*): este módulo instancia os módulos para a implementação do DCCM (**Isu_dccm_mem**, implementado no ficheiro *Isu/Isu_dccm_mem.sv*), o ICCM (**ifu_iccm_mem**, implementado no ficheiro *ifu/ifu_iccm_mem.sv*) e a Cache de Instruções (**ifu_ic_mem**, implementado no ficheiro *ifu/ifu_ic_mem.sv*).
- **swerv** (implementado no ficheiro *swerv.sv*): este módulo instancia as unidades que compõem o núcleo.
O SweRV EH1 Core (realçado a verde na Figura 27) é composto pelas quatro unidades:
 - **ifu** (Instruction Fetch Unit - Unidade de Carregamento de Instrução): esta pasta inclui os ficheiros Verilog (módulo principal disponível dentro do ficheiro *ifu.sv*) para a lcache (*cache* de instruções), Carregamento (Fetch), Preditor de Saltos (Branch Predictor) e Alinhador (Aligner).
 - **dec** (Decode Unit - Unidade de Descodificação): esta pasta inclui os ficheiros Verilog (módulo principal disponível dentro do ficheiro *dec.sv*) do Descodificador de Instruções para Descodificação de Instruções (Instruction Decoding), o Quadro de Dependências (Dependency Scoreboard), e o Banco de Registos (Register File).
 - **exu** (Execution Unit - Unidade de Execução): esta pasta inclui os ficheiros Verilog (módulo principal disponível dentro do ficheiro *exu.sv*) para as unidades aritméticas/lógicas disponíveis no núcleo: duas ALUs com *pipeline*, um Multiplicador com pipeline e um Divisor fora-do-pipeline.
 - **Isu** (Load Store Unit - Unidade de Leitura e Escrita): esta pasta inclui os ficheiros Verilog (módulo principal disponível dentro do ficheiro *Isu.sv*) para a Unidade de Escrita do Resultado com *pipeline* (Load/Store Unit).

Outras unidades incluídas neste módulo são:

- **dbg** (Debug Unit - Unidade de Depuração): esta pasta inclui os ficheiros Verilog (módulo principal disponível dentro do ficheiro *dbg.sv*) da Unidade de Depuração (Debug Unit), que é responsável por colocar o resto do núcleo em modo suspenso, enviar os comandos/endereços, enviar dados de escrita e receber dados lidos, e depois fazer o núcleo retomar o modo normal.
- **lib**: esta pasta inclui os ficheiros Verilog para os barramentos AXI e AHB-Lite.
- **pic**: esta pasta inclui o ficheiro *pic_ctrl.sv*, que implementa o Controlador de Interrupções Programável no módulo **pic_ctrl**.
- **dma**: esta pasta inclui o ficheiro *dma_ctrl.sv*, que implementa o Acesso Direto à Memória (*Direct Memory Access*) no módulo **dma_ctrl**.

ii. SoC SweRVolfX

A Figura 28 mostra a estrutura do ficheiro para o **SweRVolfX SoC** da Figura 21. O SoC está organizado como os módulos que correspondem aos blocos mostrados na Figura 21.

SweRVolfX SoC (swervolf_core.v)

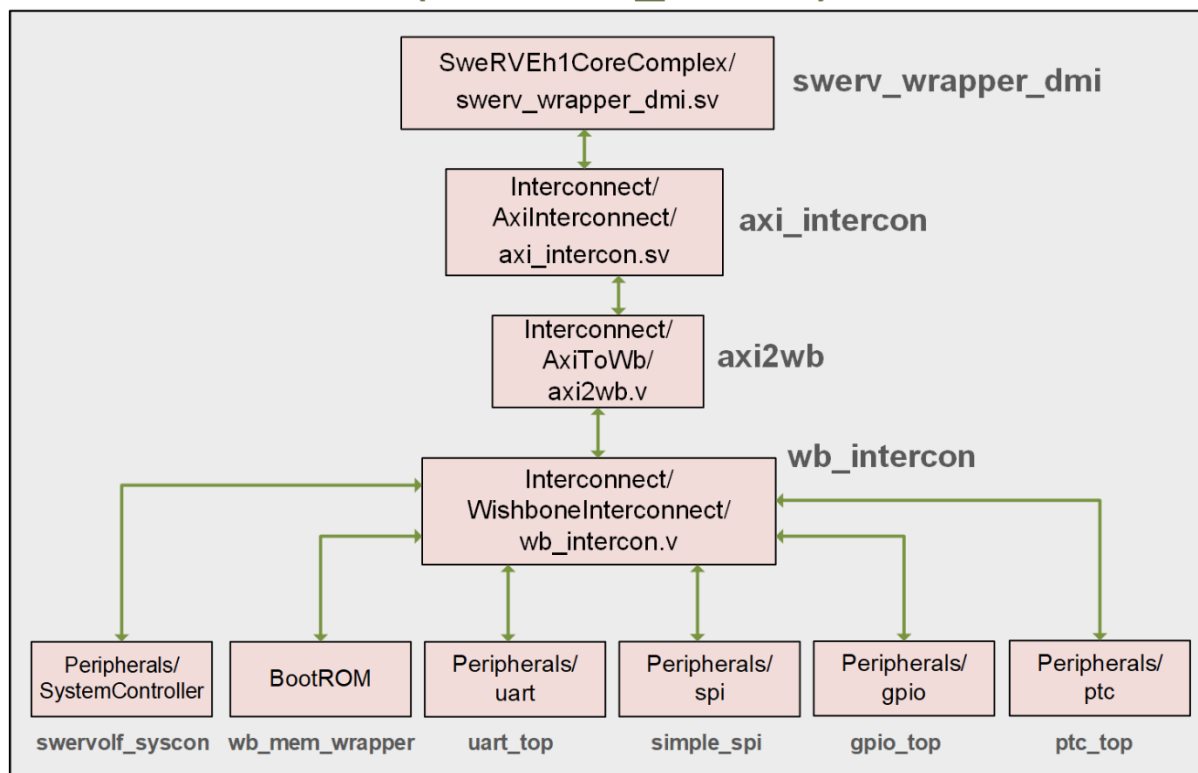


Figura 28. SweRVolfX SoC

Os ficheiros para o SweRVolfX SoC estão em:
[RVfpgaPath]/RVfpga/src/SweRVolfSoC

Encontre essa pasta no seu computador para ver os ficheiros tal como nos referimos a eles nesta secção.

O módulo principal para o **SweRVolfX SoC** está disponível em:
[RVfpgaPath]/RVfpga/src/SweRVolfSoC/swervolf_core.v. Abra esse ficheiro, e verifique que ele inclui os módulos incluídos no SweRVolfX SoC (Figura 21), em particular:

- **axi_intercon** (disponível em *Interconnect/AxiInterconnect/axi_intercon.v*): este módulo é incluído através de outro ficheiro na linha 100 (``include "axi_intercon.vh"`). Liga o SweRV EH1 Core Complex com a AXI-to-Wishbone Bridge.
- **axi2wb** (disponível em *Interconnect/AxiToWb/axi2wb.v*): este módulo, que é instanciado na linha 153 do *swervolf_core.v*, é a AXI-to-Wishbone Bridge que permite a comunicação entre o EH1 Core com AXI e os periféricos do tipo Wishbone.
- **wb_intercon** (disponível em *Interconnect/WishboneInterconnect/wb_intercon.v*): este módulo é incluído através de outro ficheiro na linha 145 (``include "wb_intercon.vh"`). Liga a AXI-to-Wishbone Bridge com os diferentes periféricos através de um multiplexador que analisaremos e modificaremos mais tarde.

- **wb_mem_wrapper** (disponível em *BootROM/wb_mem_wrapper.v*): o encapsulamento para Memória de Arranque (*Boot Memory*) descrito acima é instanciado na linha 197 de *swervolf_core.v*. Instancia o módulo **dpam64** (disponível dentro de *BootROM/dpam64.v*), que é um módulo de RAM básico.
- **swervolf_syscon** (disponível em *Peripherals/SystemController/swervolf_syscon.v*): este módulo, que é instanciado na linha 215 do *swervolf_core.v*, define o Controlador de Sistema (System Controller).
- **simple_spi**: controlador SPI obtido no OpenCores e disponível em *Peripherals/spi/simple_spi_top.v*. É instanciado nas linhas 246 (spi) e 387 (spi2) de *swervolf_core.v*.
- **uart_top**: controlador UART obtido no OpenCores e disponível em *Peripherals/uart/uart_top.v*. É instanciado na linha 272 de *swervolf_core.v*.
- **gpio_top**: controlador GPIO obtido no OpenCores e disponível em *Peripherals/gpio/gpio_top.v*. É instanciado na linha 338 de *swervolf_core.v*.
- **ptc_top**: controlador PTC obtido no OpenCores e disponível em *Peripherals/ptc/ptc_top.v*. É instanciado na linha 361 de *swervolf_core.v*.
- **swerv_wrapper_dmi** (disponível dentro de *SweRVeh1CoreComplex/swerv_wrapper_dmi.v*): instanciação (linha 407 de *swervolf_core.v*) do SweRV EH1 Core Complex da Western Digital, descrito na seção anterior (Figura 27).

iii. Encapsulamentos para execução na placa e em simulação

SIMULAÇÃO:

RVfpgaSim é uma realização para a simulação que encapsula o **SweRVolfX SoC** (Figura 21) num banco de ensaio (*testbench*) que é usado por simuladores HDL. Está disponível em *[RVfpgaPath]/RVfpga/src/rvfpgasim.v*.

ON BOARD EXECUTION:

RVfpgaNexys (disponível em: *[RVfpgaPath]/RVfpga/src/rvfpganexys.v*) encapsula o **SweRVolfX SoC** (Figura 21) para a realização na placa FPGA Nexys A7 com os respectivos periféricos (Figura 25). Este módulo instancia, para além de outros módulos (como o módulo gerador de relógio (*clock generator*), **clk_gen_nexys**, um módulo para passagem de domínios de relógio (*clock domain crossing*), **axi_cdc_intf**, ou o módulo BSCAN para o porto JTAG, **bscan_tap**), e as duas principais estruturas do SoC:

- **swervolf_core**: instanciação do **SweRVolfX SoC** descrito na subseção anterior (Figura 28). Requer o ficheiro de *constraints rvfpganexys.xdc* (disponível em *[RVfpgaPath]/RVfpga/src/*), que define as ligações entre o SoC e a placa.
- **litedram_top**: encapsulamento para o Controlador LiteDRAM DDR2, que liga o SoC SweRVolfX à Memória DDR2, e que é implementado no ficheiro *[RVfpgaPath]/RVfpga/src/LiteDRAM/litedram_top.v*. Requer o ficheiro de *constraints litedram.xdc* (disponível em *[RVfpgaPath]/RVfpga/src/LiteDRAM/*), que define as ligações entre o Controlador de Memória e a Memória DDR2 existente na placa.

Como resumo, a Figura 29 mostra a hierarquia para a implementação completa do RVfpgaNexys na placa FPGA Nexys A7.

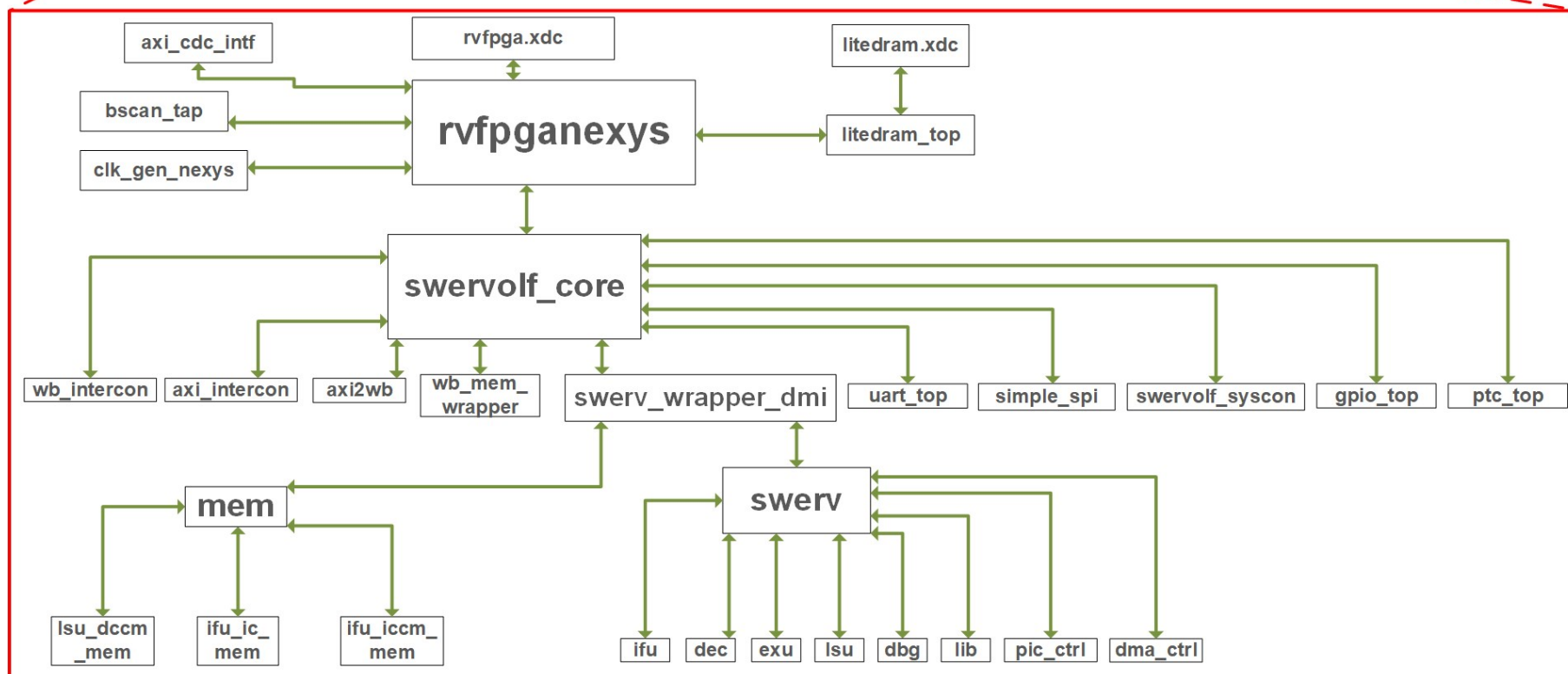
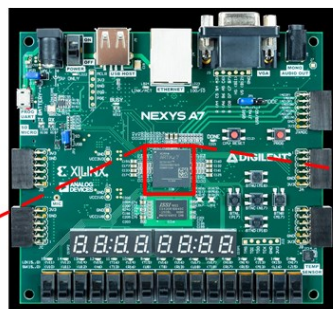


Figura 29. Hierarquia dos Módulos para a implementação na placa FPGA Nexys A7

5. INSTALAÇÃO DAS FERRAMENTAS DE DESENVOLVIMENTO

As instruções abaixo são para o Sistema Operativo Ubuntu 18.04, mas outros sistemas operativos Linux, bem como Windows ou macOS, seguem passos semelhantes (se não exatamente os mesmos). Nalguns casos, inserimos caixas com instruções específicas para esses sistemas operativos diferentes. Se estiver a utilizar o Ubuntu, pode simplesmente ignorar essas caixas.

As instruções mostram como instalar as seguintes ferramentas:

- A. **Vivado**: necessário para ressintetizar o SoC. Isto é algo que fará principalmente nos Laboratórios 6-20, onde serão adicionadas características diferentes no SoC de base.
- B. **VSCode (Visual Studio Code) e PlatformIO**: estes são os principais instrumentos utilizados neste Guia de Introdução e nos laboratórios. São utilizados para programar a FPGA e para executar/depurar programas sobre a mesma.
- C. **Verilator e GTKWave**: necessários para simular o SoC e analisar os diferentes sinais. Mais uma vez, irá utilizar principalmente estas ferramentas nos Laboratórios 6-20.

Note que, para a maioria das coisas que irá fazer neste guia e nos laboratórios, a instalação do VSCode e do PlatformIO seria suficiente. Contudo, recomendamos-lhe que instale agora também as outras ferramentas (Vivado, Verilator e GTKWave), para que mais tarde não sejam necessárias mais instalações.

Este processo poderá demorar várias horas. A maior parte do tempo é gasto à espera que os programas sejam descarregados e instalados.

A. Instalar o Vivado

O Vivado é uma ferramenta da Xilinx para visualizar, modificar e sintetizar o código Verilog do RISC-V FPGA. Utilizá-lo-á extensivamente em laboratórios posteriores. As instruções de instalação estão disponíveis em <https://reference.digilentinc.com/vivado/installing-vivado/start> e são resumidos de seguida.

Windows: a página web referenciada acima (<https://reference.digilentinc.com/vivado/installing-vivado/start>) inclui também instruções detalhadas para a instalação do Vivado no Windows. São adicionadas caixas quando são necessárias instruções específicas para o Windows.

macOS: O Vivado não é suportado em macOS; portanto, é necessária uma máquina virtual Linux/Windows para executar o Vivado neste sistema operativo.

1. Navegue até <https://reference.digilentinc.com/vivado/installing-vivado/start>
2. Lá, será guiado para a página de download do Xilinx: <https://www.xilinx.com/support/download.html>
3. É recomendável que instale o “Self Extracting Web Installer”. No momento da redacção deste documento, a página de download encontra-se nesta ligação: [Xilinx Unified Installer 2019.2: Linux Self Extracting Web Installer](#)

WINDOWS: No momento da redacção deste documento, o “Self Extracting Web Installer” para Windows está nesta ligação na página de download: [Xilinx Unified Installer 2019.2: Windows Self Extracting Web Installer](#)

4. Ser-lhe-á pedido que inicie a sessão na sua conta Xilinx antes de poder descarregar o instalador. Se ainda não tiver uma conta, terá de criar uma.

5. Execute o ficheiro binário. Abra um terminal e torne-se administrador (escreva "sudo su"). Depois arraste o ficheiro binário (Xilinx_Unified_2019.2_1106_2127_Lin64.bin) para o terminal. Se lhe pedir que torne o ficheiro executável e o execute, selecione OK.

- **Resolução de problemas:** Se o terminal indicar “permissão negada”, escreva o seguinte no terminal (na mesma pasta do ficheiro binário):
> sudo chmod +x ./Xilinx_Unified_2019.2_1106_2127_Lin64.bin
> sudo ./Xilinx_Unified_2019.2_1106_2127_Lin64.bin

WINDOWS: No Windows pode simplesmente executar o ficheiro .exe que descarregou nos passos 3 e 4, fazendo duplo clique sobre ele.

6. O instalador do Vivado irá acompanhá-lo através do processo de instalação. Notas importantes:

- Escolha o **Vivado** (*não o Vitis*) como o produto a instalar.
- Escolha o Vivado HL **Webpack** (*não o Vivado HL System Edition*); Webpack é gratuito.
- Nos demais, escolha os valores por omissão.

Sugestão: Se alterou a pasta de instalação do Vivado, terá de modificar o caminho de forma apropriada nos seguintes passos.

WINDOWS: Os passos 7 e 8, não são necessários no Windows. Pode simplesmente ignorar estes dois passos e passar diretamente ao passo 9.

7. Após a instalação do Vivado, é necessário configurar as variáveis de ambiente. Abra um terminal e escreva:

```
source /tools/Xilinx/Vivado2019.2/settings64.sh
```

Acrescente essa linha (source /tools/Xilinx/Vivado2019.2/settings64.sh) ao seu ficheiro ~/.bashrc para que execute cada vez que se lança um terminal.

8. Teste a instalação do Vivado, escrevendo o seguinte num terminal:

```
vivado
```

Resolução de problemas:

- Se o seu sistema não encontrar o executável, terá de acrescentar o seguinte ao seu caminho:

```
/tools/Xilinx/DocNav  
/tools/Xilinx/Vivado/2019.2/bin
```

- Se obter o erro: “application-specific initialization failed...”, num terminal escreva o seguinte:

```
sudo ln -s /lib/x86_64-linux-gnu/libtinfo.so.6 /lib/x86_64-  
linux-gnu/libtinfo.so.5
```

9. Precisar de instalar manualmente os controladores (*drivers*) da placa FPGA Nexys

A7. Escreva o seguinte numa janela terminal:

```
cd  
/tools/Xilinx/Vivado/2019.2/data/xicom/cable_drivers/lin64/ins  
tall_script/install_drivers/  
  
sudo ./install_drivers
```

WINDOWS: A instalação do Vivado no Windows instala automaticamente os controladores para a placa Nexys A7 que não são compatíveis com o PlatformIO. Assim, se estiver a utilizar o Windows, **deve atualizar os controladores tal como explicado no Apêndice B.** Deverá **fazer isto mesmo que já o tenha feito na secção Guia de Início Rápido porque os controladores foram substituídos na instalação do Vivado.**

10. Terá também de instalar manualmente os ficheiros “Digilent Board Files”.

- Descarregue o [ficheiro](#) com as especificações das placas para o Vivado do repositório no Github e extraia-o.
- Abra a pasta extraída do arquivo e navegue até à pasta *new/board_files*. Escolha todas as pastas dentro desta pasta e copie-as (CTRL+C).
- Abra a pasta onde o Vivado foi instalado (*/tools/Xilinx/Vivado* por omissão). Nesta pasta, vá para a pasta *<version>/data/boards/board_files*, depois cole os ficheiros (CTRL+V) da placa nesta pasta.
- Também pode utilizar o terminal, entrando na pasta *new/board_files* e escrevendo:

```
sudo cp -r *  
/tools/Xilinx/Vivado/2019.2/data/boards/board_files
```

WINDOWS: copie/cole as pastas descarregadas, como explicado no passo 10. No Windows, pode encontrar a pasta Vivado's *board_files* em:
C:\Xilinx\Vivado\2019.2\data\boards\board_files

B. Instalar o VSCode e o PlatformIO

Agora vai instalar o VSCode e o PlatformIO. **Se já o fez no Guia Rápido - Secção 1 - não precisa de repetir o processo aqui novamente e pode ir diretamente para a Secção C.**

O PlatformIO é um ambiente de desenvolvimento integrado (IDE) para sistemas embebidos que é construído sobre o Visual Studio (VS) Code da Microsoft. Ele permite programar o processador RISC-V (que está localizado na FPGA) usando C ou Assembly. O PlatformIO é multi-plataforma e inclui um depurador incorporado.

Siga estes passos para instalar o VSCode e o PlatformIO:

Linha de comandos LINUX: embora a utilização do VSCode+PlatformIO seja o método recomendado, o Apêndice A fornece instruções para quem estiver interessado em instalar e utilizar a cadeia de ferramentas nativa RISC-V e o OpenOCD em Linux, e utilizá-las no lugar do PlatformIO. Se vai utilizar o PlatformIO, ignore o Apêndice A.

1. Instalar o VSCode:

Siga estes passos para instalar o VSCode:

- a. Descarregue o ficheiro .deb a partir do seguinte endereço:
<https://code.visualstudio.com/Download>
- b. Num terminal, instale e execute o VSCode, escrevendo o seguinte:

```
cd ~/Downloads  
sudo dpkg -i code*.deb  
code
```

Windows / macOS: Os pacotes de instalação do VSCode também estão disponíveis para Windows (.exe file) e macOS (.zip file) em <https://code.visualstudio.com/Download>. Siga as etapas habitualmente usadas para a instalação e execução de uma aplicação nestes sistemas operativos.

2. Instalar PlatformIO no sobre o VSCode:

Siga estes passos para instalar o PlatformIO:

- a. Instale utilitários do python3 escrevendo o seguinte num terminal:

```
sudo apt install -y python3-distutils python3-venv
```

Windows / macOS: este passo (2.a) não é necessário no Windows. Quanto ao macOS, pode usar o *homebrew* para instalar o python3: `brew install python3`

- b. Se ainda não estiver aberto, inicie o VSCode através do botão Start e escrevendo "VSCode" no menu de pesquisa, e depois selecionar VSCode, ou escrevendo `code` num terminal.

- c. Em VSCode, clique no ícone Extensões  situado na barra lateral esquerda do VSCode (Figura 30).



Figura 30. Ícone de Extensões VSCode

- d. Escreva *PlatformIO* na caixa de pesquisa e instale o PlatformIO IDE clicando no botão de instalação junto a ele (Figura 31).

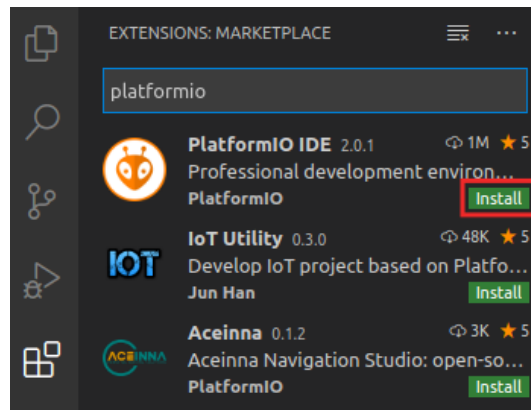


Figura 31. Extensão PlatformIO IDE

- e. A janela OUTPUT na parte inferior irá informá-lo sobre o processo de instalação. Uma vez terminado, clique em "Reload Now" (Recarregar Agora) na janela inferior direita, e PlatformIO será instalado dentro do VSCode (Figura 32).

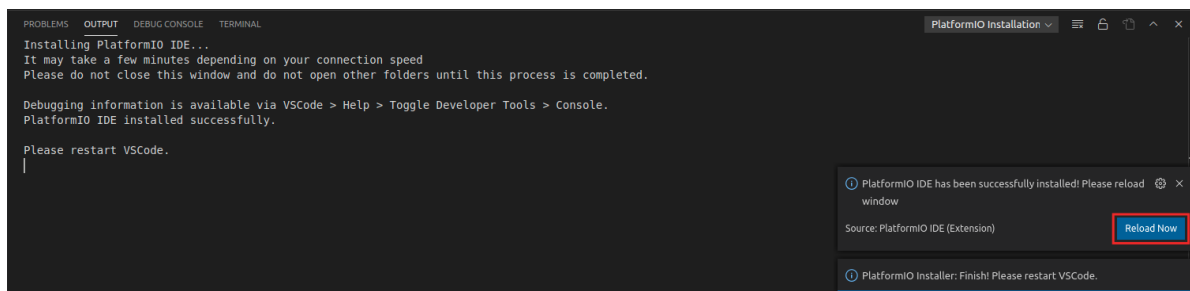


Figura 32. "Reload Now" depois da instalação do PlatformIO

C. Instalar o Verilator e o GTKWave no Ubuntu 18.04

As instruções nesta secção são válidas apenas para sistemas Linux.

Windows: siga o Apêndice C em vez das instruções fornecidas nesta secção.

macOS: siga o Apêndice D em vez das instruções fornecidas nesta secção.

Siga os próximos passos para instalar o Verilator (as instruções estão disponíveis em: <https://www.veripool.org/projects/verilator/wiki/Installing> mas também estão resumidas em baixo) e o GTKWave no seu sistema Linux Ubuntu 18.04. Este processo demora muito tempo.

- `sudo apt-get install git make autoconf g++ flex bison libfl2 libfl-dev`
- `sudo apt-get install -y gtkwave`
- `git clone https://git.veripool.org/git/verilator`
- `cd verilator`
- `git pull`
- `git checkout v4.106`
- `autoconf`
- `./configure`

- `make` (em alternativa, pode usar `make -j$(nproc)` para ser mais rápido)
- `sudo make install`
- `export PATH=$PATH:/usr/local/bin` (mudar o caminho no seu sistema)

Para adicionar `/usr/local/bin` permanentemente ao caminho, adicione a última linha ao ficheiro `~/.bashrc` da sua área.

6. CONFIGURAR E PROGRAMAR O RVfpgaNexys

Nesta secção, mostramos como executar sete programas simples no RVfpgaNexys (Figura 25).

LINUX / Windows / macOS: Todas as instruções descritas nesta secção devem funcionar para os três sistemas operativos, assumindo que todas as ferramentas e controladores (*drivers*) necessários foram instalados corretamente, tal como explicado na Secção 5. Nalguns casos, poderá ser necessário realizar pequenas modificações, tais como a barra (/), utilizada no Linux, para uma barra invertida (\), utilizada no Windows.

Demonstramos como utilizar o RVfpgaNexys mostrando como executar os sete programas de exemplo apresentados na Tabela 9. Os primeiros três programas são escritos em linguagem Assembly RISC-V e os últimos quatro programas são escritos em C. As instruções para executar cada um dos programas no RVfpgaNexys são descritas abaixo.

Tabela 9. Programas de Exemplo RVfpgaNexys

Nome do Programa	Descrição	Linguagem
AL_Operations	exercícios de operação aritmética e lógica	Assembly RISC-V
Blinky	pisca um LED na placa Nexys A7	Assembly RISC-V
LedsSwitches	lê os valores dos interruptores na placa Nexys A7 e escreve esse valor nos LEDs	Assembly RISC-V
LedsSwitches_C-Lang	lê os valores dos interruptores na placa Nexys A7 e escreve esse valor nos LEDs	C
HelloWorld_C-Lang	imprime uma pequena mensagem para uma consola através da porta série	C
VectorSorting_C-Lang	ordena um vetor do maior para o menor	C
DotProduct_C-Lang	calcula o produto interno de dois vetores	C

Repare que, antes de poder executar qualquer um destes sete exemplos, **deve configurar a FPGA com RVfpgaNexys**, como explicado na secção seguinte.

A. Configurar a FPGA com o RVfpgaNexys

Nesta secção, explicamos o método recomendado para a configuração da FPGA com RVfpgaNexys, que utiliza o PlatformIO. Siga os próximos passos para a configuração da FPGA com RVfpgaNexys:

(Se estiver interessado em utilizar o Vivado para a programação da FPGA, pode seguir as instruções fornecidas no Apêndice E deste guia em vez das instruções que se seguem. No

entanto, o método aí descrito só é possível para sistemas Linux e Windows (macOS *não*) – e, em geral, o método de utilizar o Vivado para configurar o RVfpgaNexys na FPGA não é recomendado. Em vez disso, recomenda-se que siga as instruções em baixo e ignore o Apêndice E.)

- Ligue a placa Nexys A7 ao seu computador.
- Ligue a placa Nexys A7 usando o interruptor no canto superior esquerdo.
- Abra o VSCode e o PlatformIO se ainda não estiver aberto.
- Na barra de menu superior, clicar em *File* → *Open Folder* (Figura 33) e encontre a pasta *[RVfpgaPath]/RVfpga/examples/*

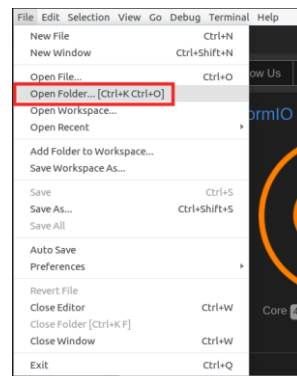


Figura 33. Open Folder

- Selecione o projeto PlatformIO que vai utilizar. Nesta secção, como exemplo, utilizamos *AL_Operations*, o primeiro exemplo mencionado na Tabela 9, que irá depurar na secção seguinte, mas poderá seguir os mesmos passos com qualquer outro exemplo. Selecione a pasta *AL_Operations* (não a abra, mas apenas o selecione – **Figura 34**) e clique em OK na parte superior da janela. PlatformIO irá agora abrir o exemplo.

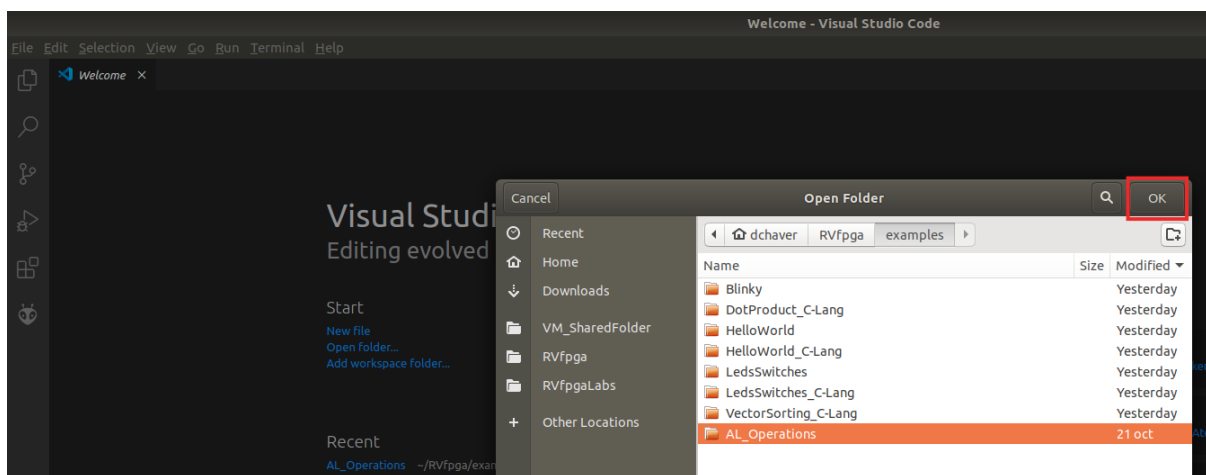


Figura 34. Abrir a pasta AL_Operations

- Abra o ficheiro *platformio.ini*, clicando no *platformio.ini* na barra lateral esquerda (Figura 35). Defina o caminho para o *bitstream* de configuração do RVfpgaNexys no seu sistema, editando a seguinte linha (Figura 35). Note que é fornecido um *bitstream* pré-sintetizado do RVfpgaNexys na pasta do RVfpga em: *[RVfpgaPath]/RVfpga/src/rvfpganexys.bit*.

```
board_build.bitstream_file = [RVfpgaPath]/RVfpga/src/rvfpganexys.bit
```

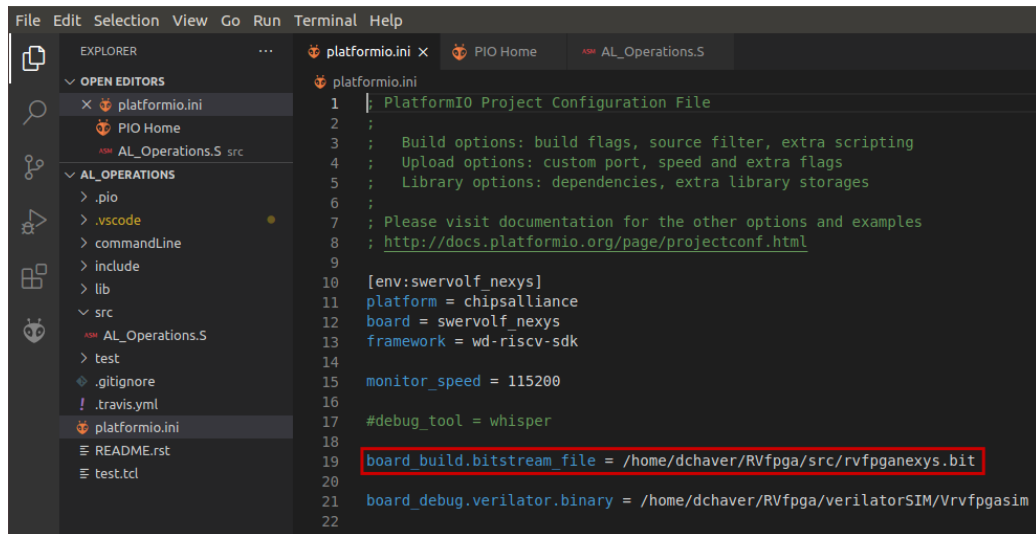


Figura 35. Ficheiro de iniciação do PlatformIO: platformio.ini

Há vários comandos que pode utilizar no Ficheiro de Configuração do Projeto (*platformio.ini*), e para os quais pode encontrar informações em: <https://docs.platformio.org/en/latest/projectconf/>.



- g. Clique no ícone do PlatformIO  na faixa do menu da esquerda (Figura 36).



Figura 36. Ícone do PlatformIO

No caso de a janela de Tarefas do Projeto (Project Tasks) estar vazia (Figura 37), deve atualizar primeiro as Tarefas do Projeto, clicando em . Isto pode demorar vários minutos.

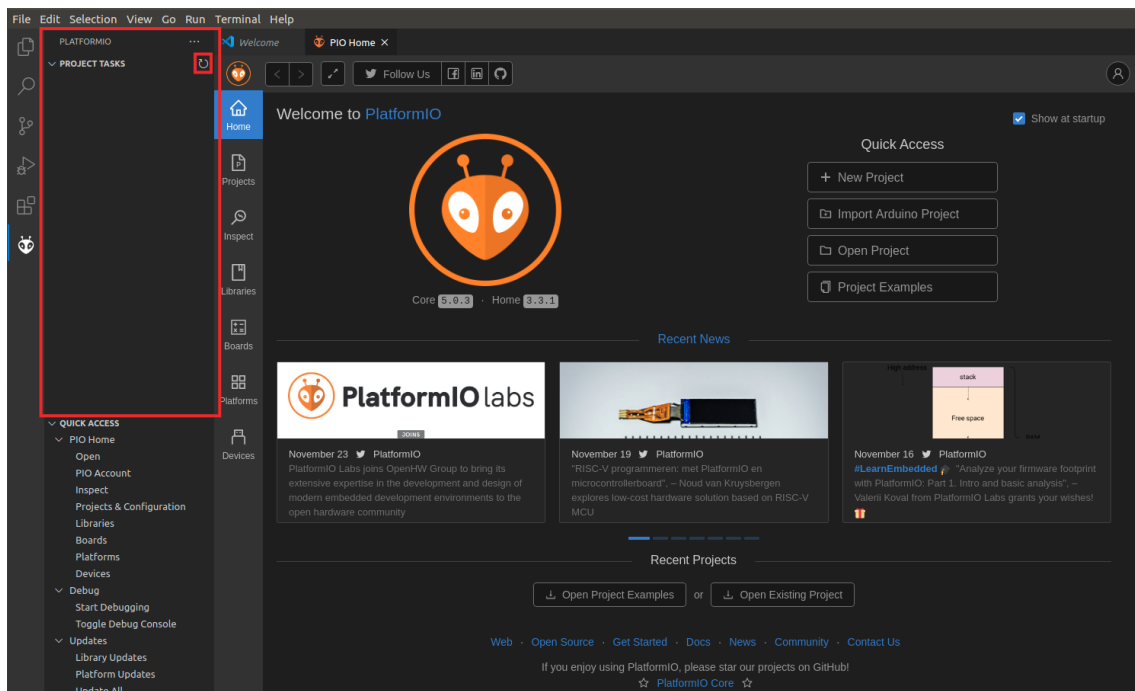


Figura 37. Janela PROJECT TASKS vazia – Atualizar

Em seguida, expandir Project Tasks → env:swervolf_nexys → Platform e clique em “Upload Bitstream”, tal como ilustrado na Figura 38. **Após alguns segundos, a FPGA será configurada com RVfpgaNexys.**

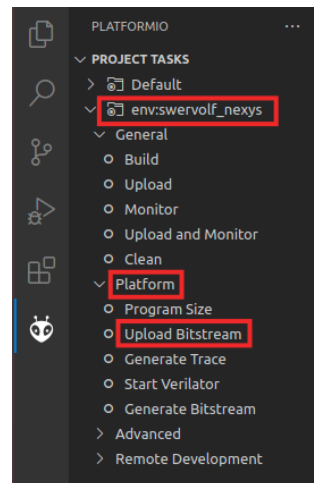



Figura 38. Carregamento do Bitstream

Por omissão, o processador começa a carregar instruções a partir do endereço 0x80000000, onde está a Boot ROM no nosso SoC (Tabela 6). A Boot ROM é iniciada com um programa (*boot_main.mem*) que pisca os LEDs e os mostradores de 7-Segmentos quatro vezes e depois desliga todos os LEDs, escreve 0s para os 8 mostradores de 7-Segmentos e entra num ciclo infinito. Pode encontrar este programa na pasta: *[RVfpgaPath]/RVfpga/src/SweRVolfSoC/BootROM/sw*.

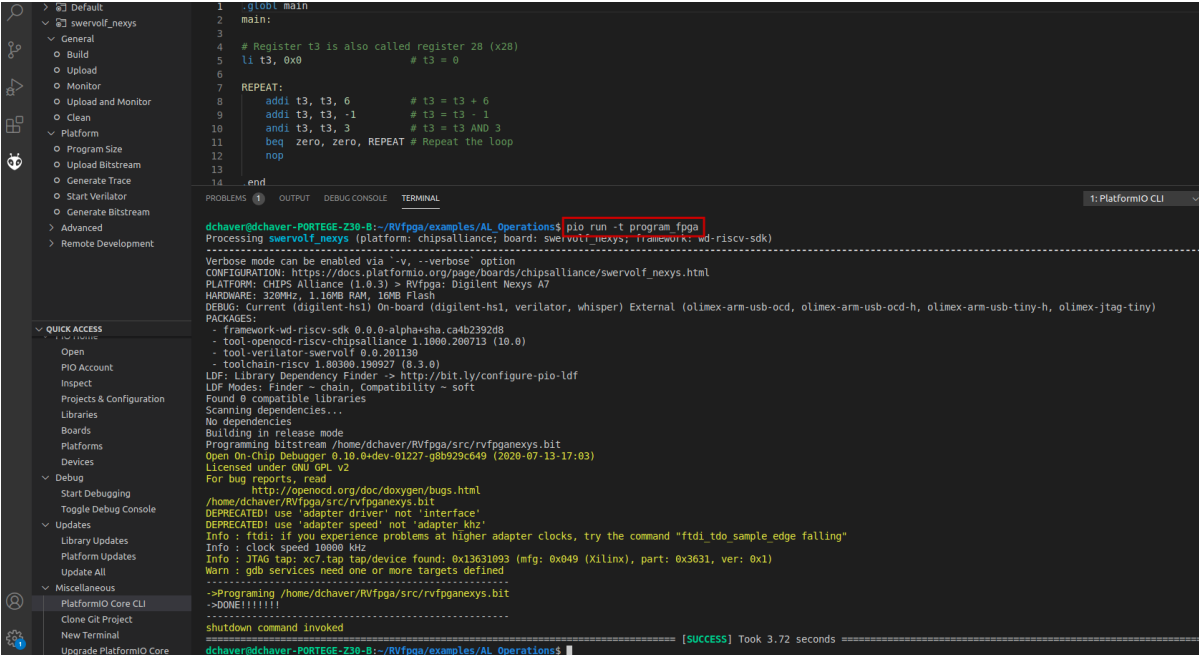
Premindo o botão de Reset do CPU na placa Nexys A7 (Figura 26) faz com que este programa seja de novo executado.

Se quiser alterar e recompilar este programa, faça-o como explicado no Apêndice A –

Secção III (repare que o ficheiro *boot_main.mem* é uma mera cópia do ficheiro *boot_main.vh*). No Lab 1 mostraremos como a Boot ROM é inicializada com este programa quando se cria o bitstream.

- h. Como alternativa ao passo anterior (passo g), pode descarregar o RVfpgaNexys a partir de uma janela de terminal PlatformIO, como se mostra na Figura 39. Clique no botão  (*PlatformIO: New Terminal*) na parte inferior da janela do PlatformIO para abrir uma nova janela de terminal, e depois escrever (ou copiar) o seguinte comando no terminal do PlatformIO:

```
pio run -t program_fpga
```



The screenshot shows the PlatformIO IDE interface. On the left, the 'Project Explorer' shows the project structure for 'RVfpgaNexys'. The main editor displays the 'main.c' file with assembly code. The bottom panel shows the 'TERMINAL' output, which includes the command 'pio run -t program_fpga' and its execution details, such as the platform (Chips Alliance), board (RVfpgaNexys A7), and the successful completion of the build and programming process.

Figura 39. Carregamento do RVfpgaNexys na placa FPGA Nexys A7 usando o PlatformIO

Note que a primeira vez que um exemplo é aberto no PlatformIO, a plataforma Chips Alliance é instalada automaticamente (pode vê-la dentro do PIO Home, como mostrado na Figura 40). Esta plataforma inclui várias ferramentas que irá utilizar mais tarde, tais como a conjunto de ferramentas pré-construídas RISC-V, OpenOCD para RISC-V, um ficheiro *bitstream* RVfpgaNexys e RVfpgaSim, JavaScript e scripts Python, e vários exemplos.

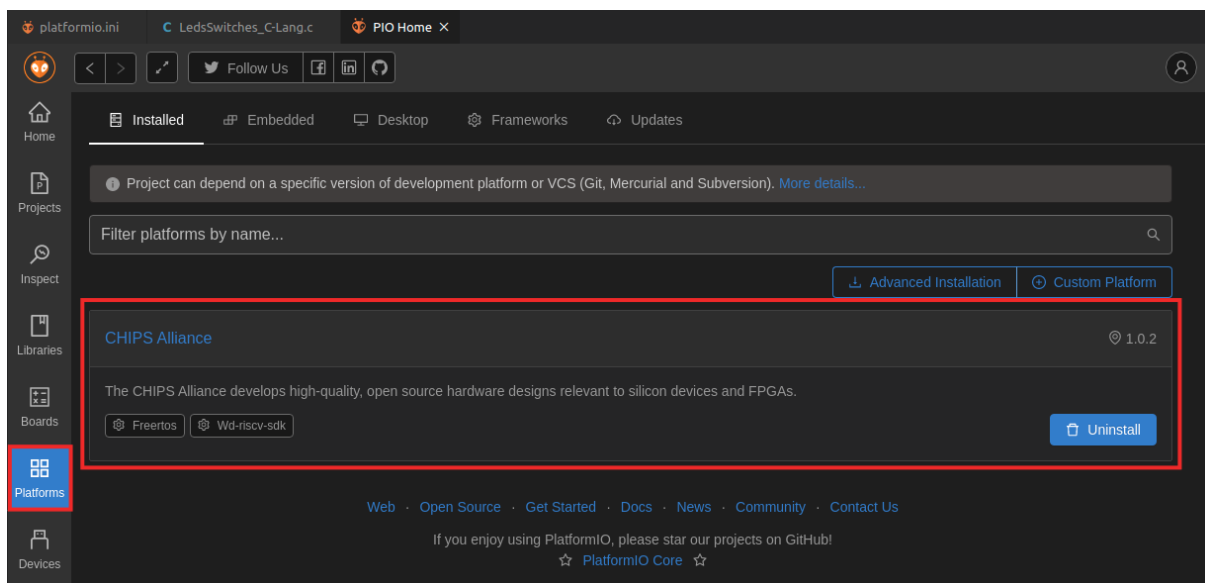





Figura 40. Plataforma Chips Alliance instalada no PlatformIO

Se, por qualquer razão, a plataforma Chips Alliance não se instalou automaticamente, pode instalá-la manualmente seguindo os seguintes passos (normalmente, pode simplesmente saltar este procedimento e continuar com a secção B):

- Veja o menu de Acesso Rápido clicando no botão , localizado na barra lateral esquerda (Figura 41). Depois, no PIO Home, clique no botão  e depois no separador  (Figura 41). Procure **Chipsalliance** (a plataforma que utilizamos no RVfpga) e abra-o clicando no botão **CHIPS Alliance** (Figura 41).

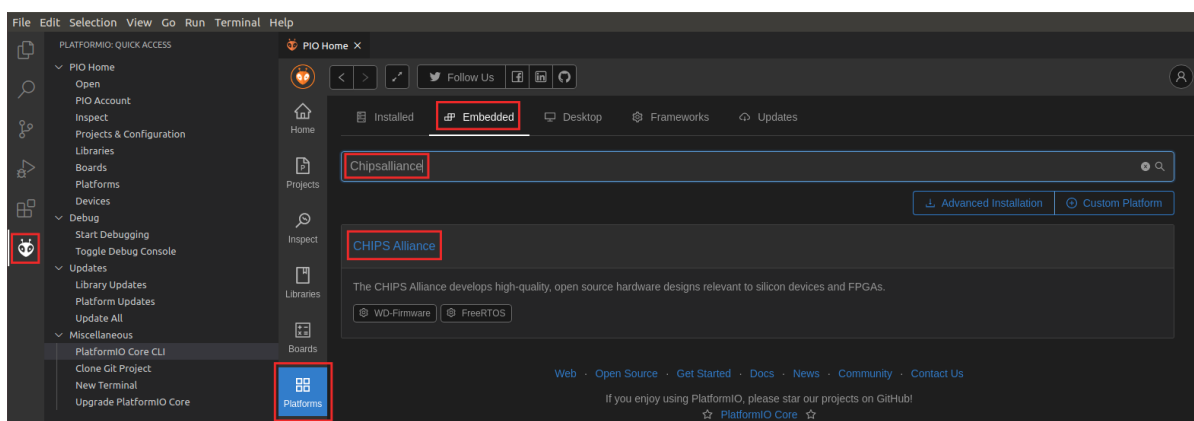



Figura 41. Seleção da Plataforma CHIPS Alliance

- Depois de clicar no botão **CHIPS Alliance**, verá os detalhes da plataforma Chips Alliance (tal como na Figura 42). Instale-o, clicando no botão  (Figura 42).

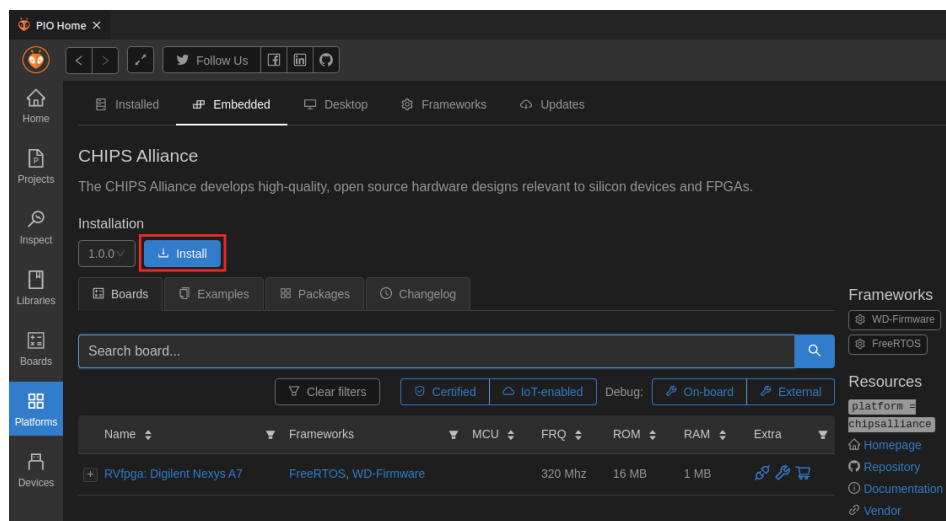



Figura 42. Instalação da plataforma Chips Alliance

- Uma vez concluída a instalação, é apresentado um resumo das ferramentas que foram instaladas, como ilustrado na Figura 43. Clique em  para fechar essa janela.

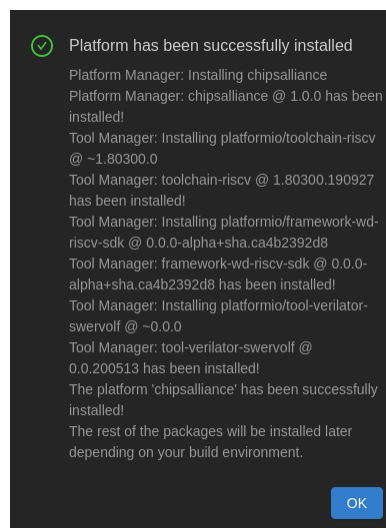


Figura 43. Instalação da Plataforma CHIPS Alliance bem sucedida

B. Programa AL_Operations

O primeiro programa de exemplo, AL_Operations.s (Figura 44), é um programa Assembly que executa três instruções aritmética-lógica (adição, subtração, e lógica AND) no mesmo registo, t3 (também chamado x28), dentro de um ciclo infinito.

```


1  .globl main
2  main:
3
4  # O registo t3 também é designado por registo 28 (x28)
5  li t3, 0x0                # t3 = 0
6
7  REPEAT:
8      addi t3, t3, 6          # t3 = t3 + 6
9      addi t3, t3, -1         # t3 = t3 - 1
10     andi t3, t3, 3          # t3 = t3 AND 3
11     beq zero, zero, REPEAT # Repetir o ciclo

```

```
12    nop
13
14    .end
```

Figura 44. Programa AL_Operations: AL_Operations.S

Siga estes passos para executar e depurar este código na placa A7 FPGA da Nexys utilizando o PlatformIO:

1. Configurar a FPGA como explicado na secção anterior. Repare que já tem o projeto *AL_Operations* aberto no PlatformIO.
2. Abra o programa Assembly, *AL_Operations.S*, clicando no ícone Explorer na fita do menu esquerdo , expandindo *src* debaixo de *AL_OPERATIONS* na barra lateral esquerda e clicando em *AL_Operations.S* (Figura 45).

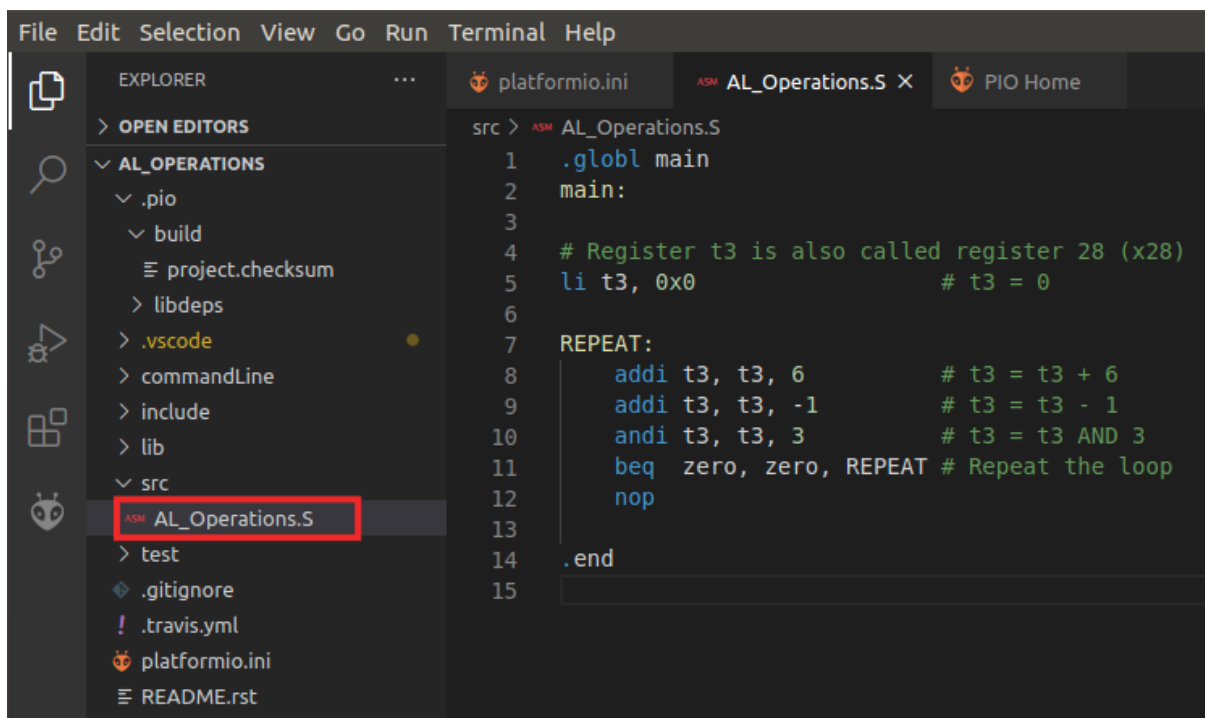
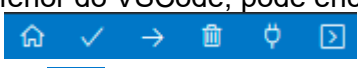




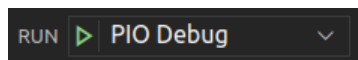


Figura 45. Aspecto do ficheiro Assembly AL_Operations.S

3. O VSCode e o PlatformIO fornecem diferentes formas de compilação, limpeza e depuração do programa. Na parte inferior do VSCode, pode encontrar alguns botões que fornecem funcionalidades úteis: . Por exemplo,  pode ser utilizado para compilar o projeto, ou  pode ser utilizado para o limpar. Na barra lateral esquerda (Figura 30), o botão “Run”  pode ser utilizado para compilar o programa e depois abrir o depurador.

4. Clicar no botão “Run” . Iniciar o depurador clicando no botão “Play”



(certificar-se de que a opção "PIO Debug" está selecionada). Pode encontrar este botão perto da parte superior da janela (Figura 46). O programa compilará primeiro e depois começará a depuração. PlatformIO cria um ponto de paragem (*breakpoint*) temporário no início da função principal, pelo que a execução irá parar aí.

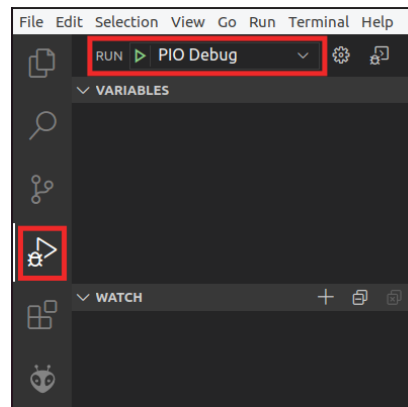


Figura 46. Iniciar o depurador


5. Para controlar a sua sessão de depuração, pode utilizar a barra de ferramentas de depuração que aparece perto do topo do editor (Figura 47). Seguem-se as opções:
 - **Continue** executa o programa até ao próximo *breakpoint*.
 - **Breakpoints** pode ser adicionado clicando à esquerda do número da linha no editor.
 - **Step Over** executa a linha atual e depois é suspenso.
 - **Step Into** executa a linha atual e, se a linha atual incluir uma chamada de função, salta para essa função e é suspenso.
 - **Step Out** executa todo o código na função em que se encontra e é suspenso quando essa função retorna.
 - **Restart** reinicia a sessão de depuração a partir do início do programa.
 - **Stop** termina a sessão de depuração e regressa ao modo de edição normal.
 - **Pause** faz uma pausa na execução. Quando o programa está a ser executado, o botão Continue é substituído pelo botão Pause.



Figura 47. Ferramentas de depuração

6. Na barra lateral esquerda, pode ver as opções do Debugger. Estão disponíveis as seguintes opções:
 - **Variables:** lista as variáveis locais, globais e estáticas presentes no seu programa, juntamente com os seus valores.
 - **Call Stack:** mostra a função atual que está a ser executada, a função de chamada (se existir) e a localização da instrução atual na memória.

- **Breakpoints:** mostra todos os *breakpoints* definidos e realça o seu número de linha. Os *breakpoints* podem ser geridos nesta secção. Os *breakpoints* também podem ser temporariamente desativados sem serem removidos, bastando para isso ativar a caixa de verificação.
- **Peripherals:** mostra o estado dos registos dos periféricos mapeados na memória do dispositivo (abordaremos estes em mais pormenor nos Laboratórios RVfpga).
- **Registers:** lista os valores atuais presentes em cada um dos registos do processador.
- **Memory:** apresenta o conteúdo de um endereço específico da memória.
- **Disassembly:** mostra o código Assembly para uma função específica - para código de alto nível, como o C, isto permite-lhe ver o Assembly para depurar as instruções uma por uma.

7. Expanda a opção Registos na barra lateral do depurador e continue a execução passo a passo . Observará que o registo x28 (também designado por t3, como indicado na secção REGISTERS) armazena os resultados das três operações aritméticas e lógicas: *adição*, *subtração* e *AND lógico*. Ver Figura 48.

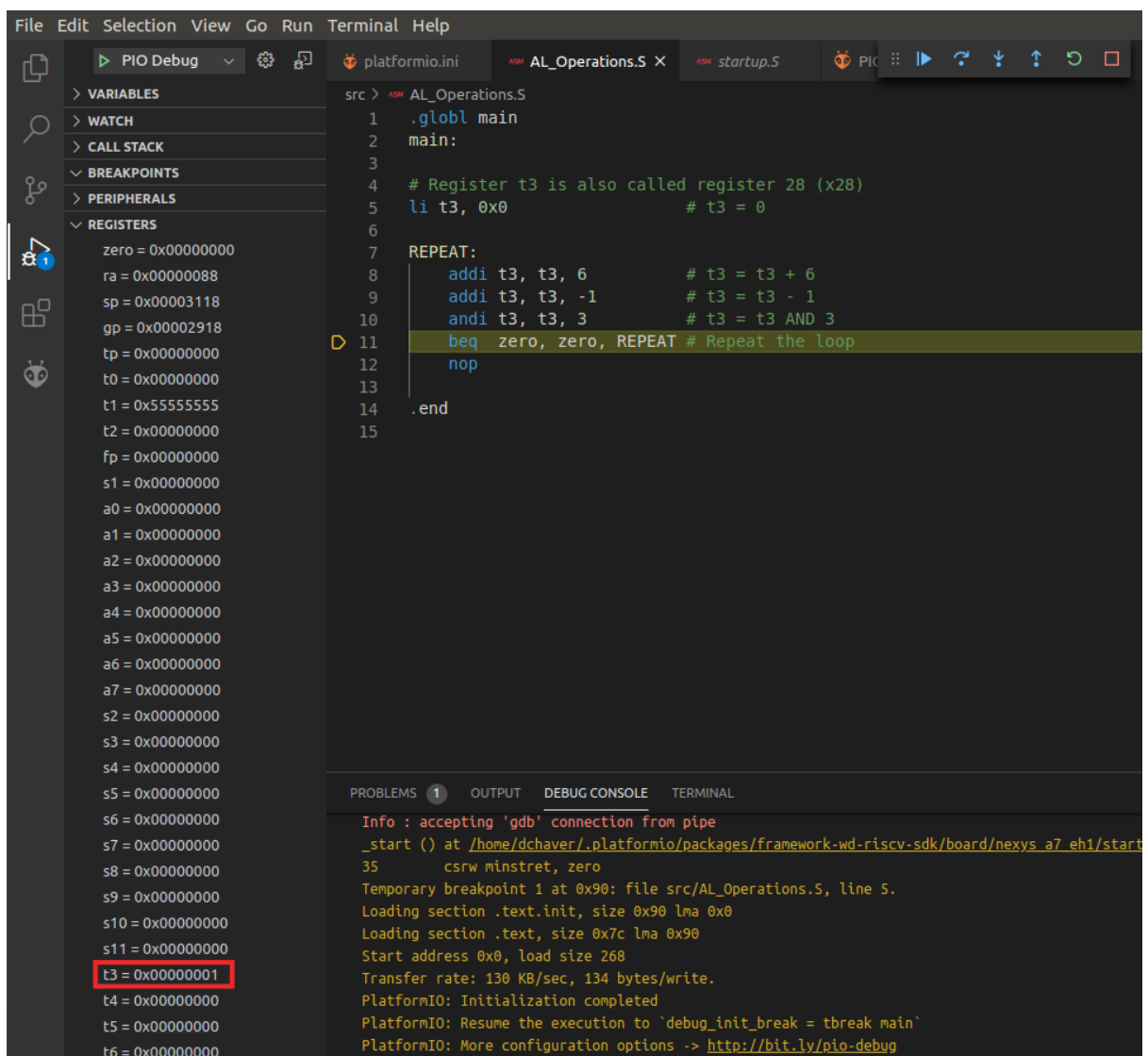




Figura 48. Visualizar o conteúdo dos registos

8. Antes de chamar a função principal (*main*), um ficheiro de arranque, fornecido pela Western Digital em `~/.platformio/packages/framework-wd-riscv-sdk/board/nexys_a7_eh1/startup.S`, é executado. Este ficheiro configura o núcleo: configuração da cache de instruções, inicialização dos registos (como *sp* ou *gp*), etc. Quando a depuração é iniciada, este ficheiro abre-se na janela principal (ver Figura 48), e pode inspecioná-lo lá.

Windows: A pasta `.platformio` está localizada dentro da sua pasta de utilizador (`C:\Users\<USER>`). Observe que talvez seja necessário habilitar o sistema para visualizar ficheiros/pastas ocultos.

macOS: Como no Linux, a pasta do `.platformio` está localizado dentro da sua pasta *home* (`~/.platformio`).

9. De salientar ainda que, na mesma pasta (`~/.platformio/packages/framework-wd-riscv-sdk/board/nexys_a7_eh1/`), é fornecido o ficheiro `link.lds`, que constitui o script de ligação que utilizaremos em todos os nossos projetos. Este ficheiro determina a colocação das secções do Assembly (*text*, *data*, *bss*...) na memória.

10. Finalmente, pare de depurar  (o que fará com que o programa da Boot ROM seja executado novamente) e volte para a janela do Explorer clicando em , que pode encontrar no topo da barra lateral mais à esquerda. Na barra de menu superior, clique em *File* → *Close Folder*.

C. Programa Blinky

O segundo programa de exemplo, *blink.S*, é um programa em Assembly que faz com que o LED mais à direita da placa Nexys A7 pisque (Figura 49). O programa inverte repetidamente o valor ligado ao LED mais à direita com um atraso entre cada inversão.

```

1  #define GPIO_LEDS    0x80001404
2  #define GPIO_INOUT   0x80001408
3
4  #define DELAY 0x100000          /* Definir o atraso */
5
6  .globl main
7  main:
8
9      li x28, 0xFFFF
10     li a0, GPIO_INOUT
11     sw x28, 0(a0)              # Escrever o Enable Register
12
13     li t1, DELAY                # Definir o valor do temporizador para
14                                 # controlar a velocidade de intermitência
15     li t0, 0
16
17 b1:
18     li a0, GPIO_LEDS
19     sb t0, 0(a0)                # Escrever nos LEDs
20     xori t0, t0, 1              # inverter o LED
21     and t2, zero, zero         # Reiniciar o temporizador
22
23 time1:                          # Ciclo de atraso
24     addi t2, t2, 1

```

```
25      bne  t1, t2, time1
26      j    bll
```

Figura 49. blinky.S

Siga os próximos passos para executar e depurar este código no RVfpgaNexys, o SoC RISC-V carregado na placa FPGA:

1. O RVfpgaNexys já está configurado na placa FPGA se tiver executado o primeiro exemplo (*AL_Operations*), por isso não deve ser necessário configurá-la novamente. No entanto, se precisar de reconfigurar novamente o RVfpgaNexys na placa, faça-o como explicado na Secção A, utilizando o exemplo Blinky em vez do exemplo *AL_Operations*.
2. Na barra superior, clique em *File* → *Open Folder*, e navegue até à pasta *[RVfpgaPath]/RVfpga/examples/*

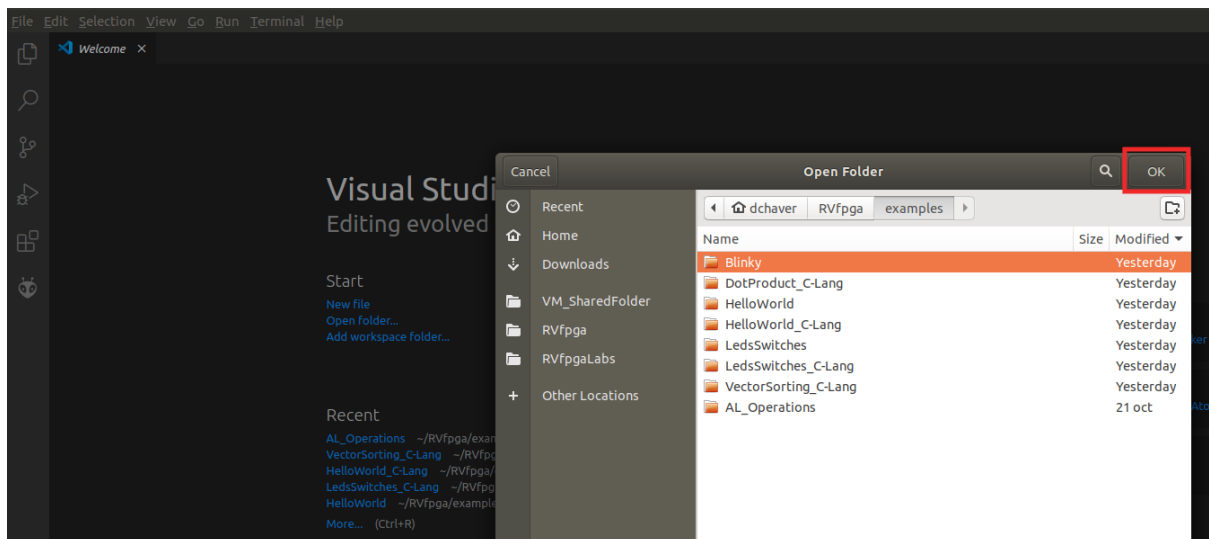
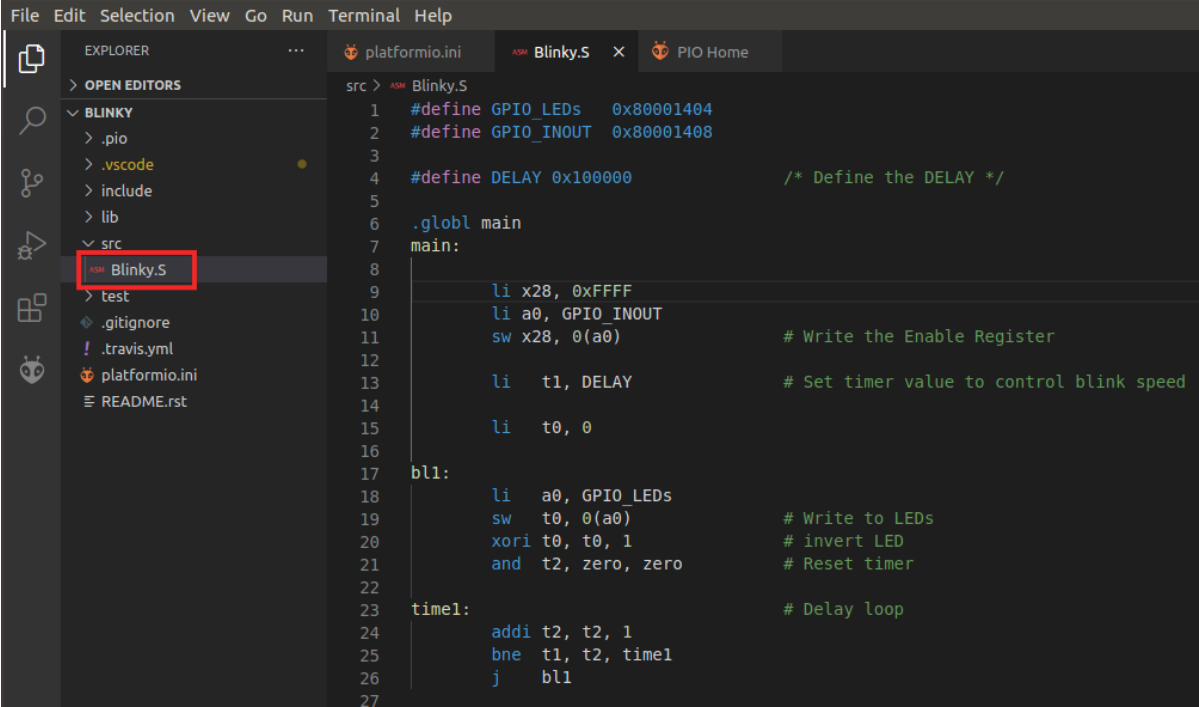


Figura 50. Pasta do programa Blinky

3. Selecione a pasta *Blinky* e clique em OK (Figura 50).
4. Abra o código Assembly do exemplo, o ficheiro *blinky.S*, no editor, clicando nele (Figura 51).








```

1  #define GPIO_LEDS    0x80001404
2  #define GPIO_INOUT   0x80001408
3
4  #define DELAY 0x100000    /* Define the DELAY */
5
6  .globl main
7  main:
8
9      li x28, 0xFFFF
10     li a0, GPIO_INOUT
11     sw x28, 0(a0)        # Write the Enable Register
12
13     li t1, DELAY          # Set timer value to control blink speed
14
15     li t0, 0
16
17 bl1:
18     li a0, GPIO_LEDS
19     sw t0, 0(a0)          # Write to LEDs
20     xori t0, t0, 1        # invert LED
21     and t2, zero, zero   # Reset timer
22
23 time1:
24     addi t2, t2, 1        # Delay loop
25     bne t1, t2, time1
26     j bl1
27

```

Figura 51. blinky.S no PlatformIO

5. Clique em  para executar e depurar o programa; em seguida, inicie a depuração clicando no botão **play**  **PIO Debug** . A PlatformIO define um *breakpoint* temporário no início da função principal. Então, clique no botão *Continue*  para executar o programa.
6. Na placa, verá o LED mais à direita começar a piscar.
7. Faça uma pausa na execução clicando no botão de *pause* . A execução irá parar algures dentro do ciclo infinito (provavelmente, dentro do ciclo de atraso `time1`).
8. Estabeleça um *breakpoint* clicando à esquerda da linha número 18. Aparecerá um ponto vermelho e o ponto de interrupção será adicionado ao separador BREAKPOINTS (Figura 52).

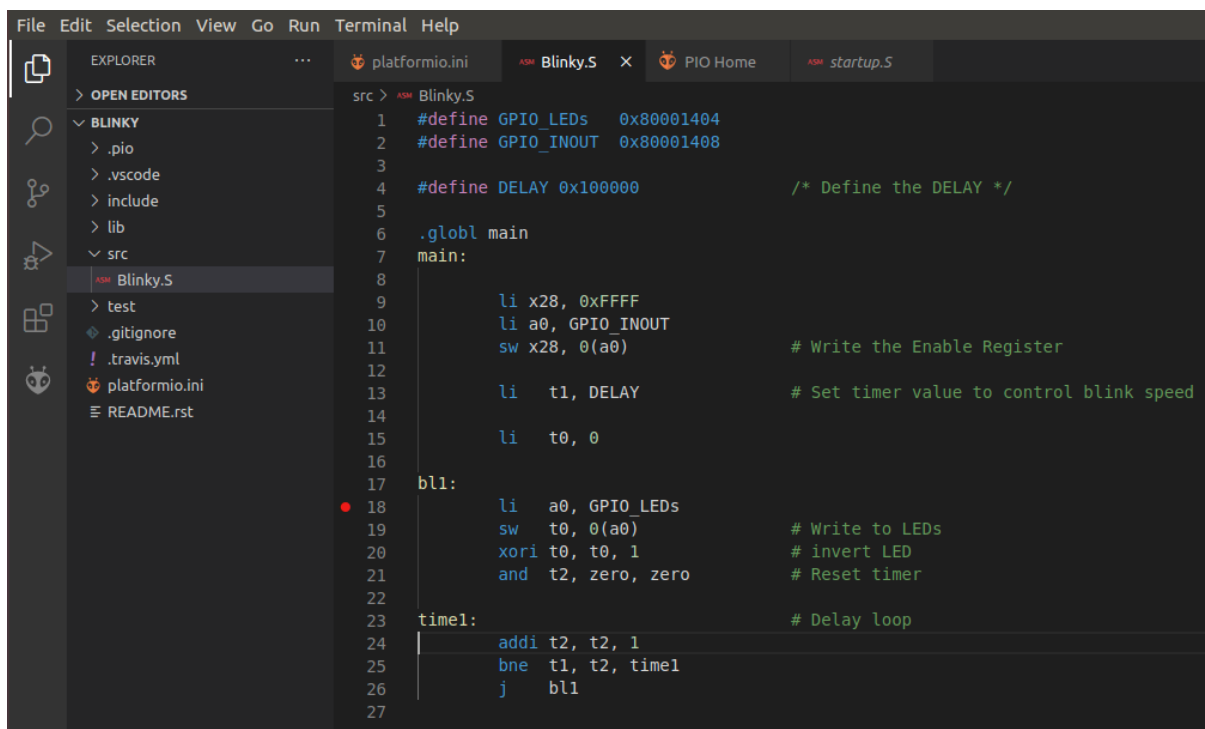


Figura 52. Definir um ponto de paragem (breakpoint) no blinky.S

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



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

10. Continue a execução várias vezes; verá que o valor indicado no LED mais à direita muda de cada vez.



11. 11. Pare a depuração e regresse à janela do Explorer



clicando em . Feche o programa selecionando *File → Close Folder*.

D. Programa LedsSwitches

O terceiro exemplo Assembly comunica com os LEDs e os interruptores disponíveis na placa (Figura 53).

```

1  #define GPIO_SWs      0x80001400
2  #define GPIO_LEDs     0x80001404
3  #define GPIO_INOUT    0x80001408
4
5  .globl main
6  main:
7
8  li x28, 0xFFFF
9  li x29, GPIO_INOUT
10 sw x28, 0(x29)          # Escrever no Enable Register
11
12 next:
13 li a1, GPIO_SWs         # Ler os interruptores
14 lw t0, 0(a1)
15

```

```

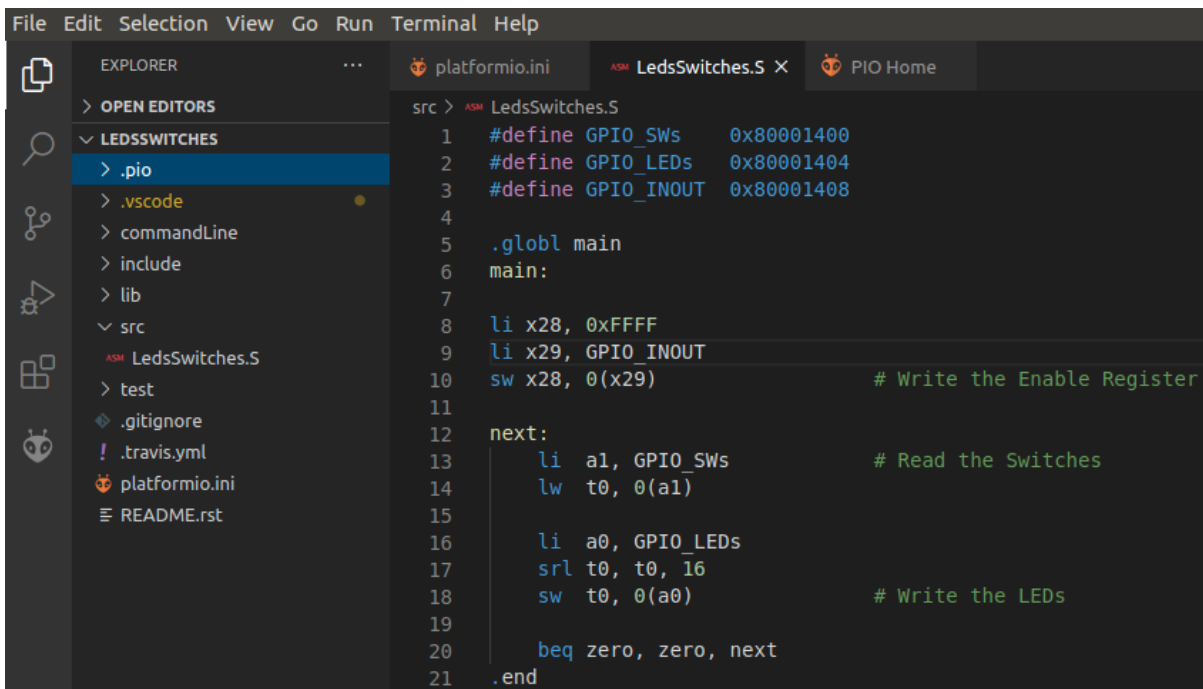
16  li  a0, GPIO_LEDS
17  srl  t0, t0, 16
18  sw  t0, 0(a0)           # Escrever nos LEDs
19
20  beq  zero, zero, next
21  .end

```

Figura 53. LedsSwitches.S

Siga os passos seguintes para executar e depurar este código na placa FPGA:

1. O RVfpgaNexys já está programado na placa FPGA se tiver executado os exemplos anteriores, pelo que não deverá ser necessário programá-lo novamente. No entanto, se precisar de reprogramar o RVfpgaNexys na placa novamente, faça-o como explicado na Secção A, utilizando o exemplo LedsSwitches em vez do exemplo AL_Operations.
2. Na barra superior, clique em *File* → *Open Folder*, e navegue até à pasta *[RVfpgaPath]/RVfpga/examples/*. Selecione a pasta *LedsSwitches* e clique em OK.
3. O programa *LedsSwitches.S* tem um ciclo infinito em que os interruptores são lidos e depois o seu estado é mostrado nos LEDs (Figura 54).




```

File Edit Selection View Go Run Terminal Help
EXPLORER
> OPEN EDITORS
LEDSSWITCHES
  > .pio
  > .vscode
  > commandLine
  > include
  > lib
  > src
    ASM LedsSwitches.S
  > test
  > .gitignore
  > ! .travis.yml
  > platformio.ini
  > README.rst
src > ASM LedsSwitches.S
1  #define GPIO_SWs    0x80001400
2  #define GPIO_LEDS   0x80001404
3  #define GPIO_INOUT  0x80001408
4
5  .globl main
6  main:
7
8  li x28, 0xFFFF
9  li x29, GPIO_INOUT
10 sw x28, 0(x29)           # Write the Enable Register
11
12 next:
13  li a1, GPIO_SWs         # Read the Switches
14  lw t0, 0(a1)
15
16  li a0, GPIO_LEDS
17  srl t0, t0, 16
18  sw t0, 0(a0)           # Write the LEDs
19
20  beq zero, zero, next
21  .end

```

Figura 54. LedsSwitches.S no PlatformIO

4. Depois de lançar o depurador como explicado para os programas anteriores, o programa começa a ser executado. A PlatformIO define um *breakpoint* temporário no início da função principal. Então, clique no botão Continue  para executar o programa.
5. Alterne os interruptores na parte inferior da placa Nexys A7. Verá imediatamente na placa que os LED indicam o novo valor dos interruptores. Pode fazer uma pausa na execução, executar passo a passo e inspecionar os registos como explicado acima. Quando terminar, feche o projeto clicando em *File* → *Close Folder*.
6. Por vezes, pode ser muito útil inspecionar os valores armazenados na memória. Para

isso, o PlatformIO fornece o Memory Display.

- a. Pause a execução e avance até ao início do próximo ciclo. Expanda o *Memory Display* na parte esquerda da janela (Figura 55) e clique em *Enter address...*

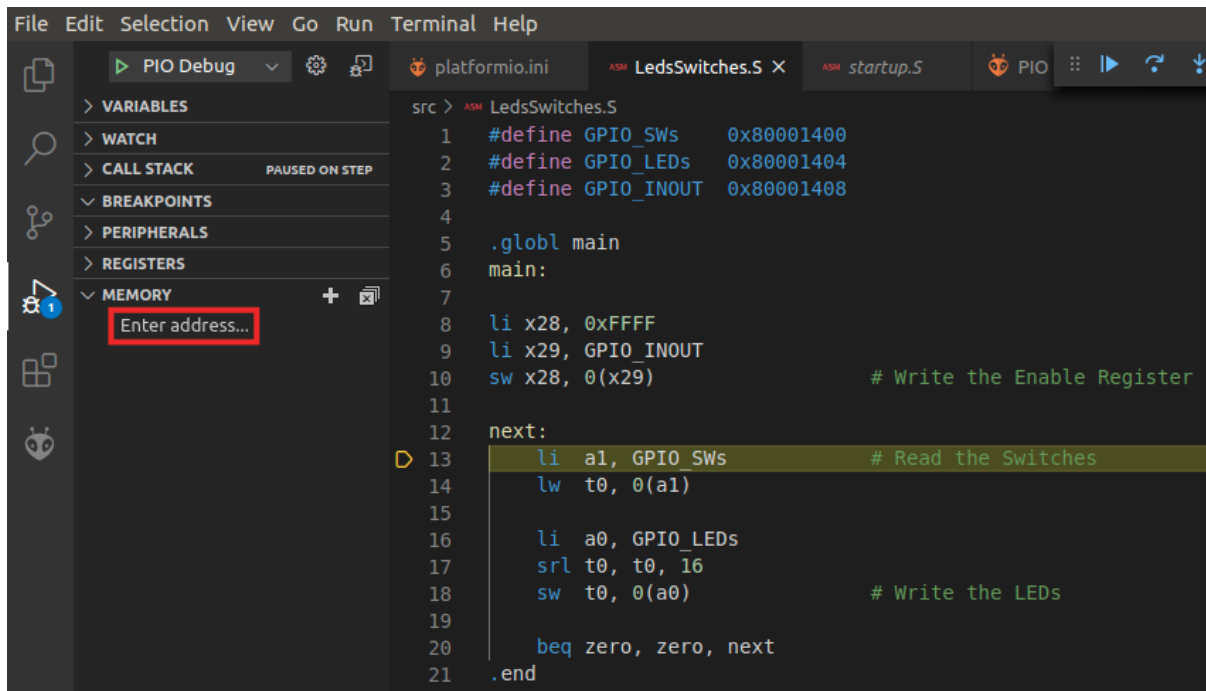


Figura 55. Memory Display

- b. O endereço de memória inicial será pedido (Figura 56). Introduza o endereço inicial onde os Switches estão mapeados, no nosso caso 0x80001400.

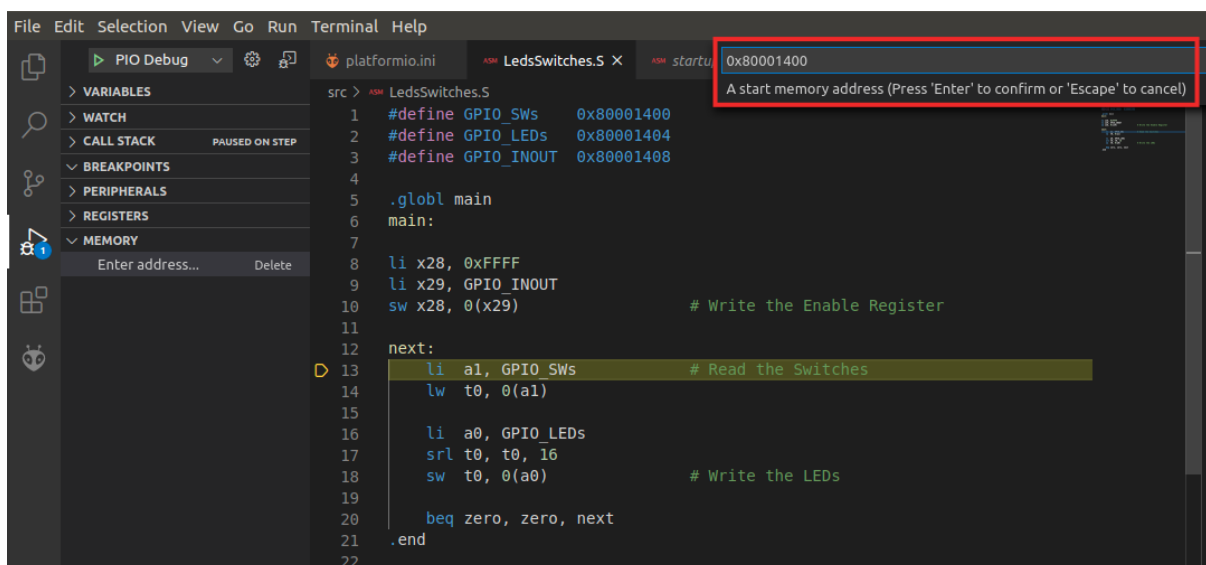


Figura 56. Endereço inicial de memória a mostrar

- c. Em seguida, é pedido o número de bytes que se pretende inspecionar (Figura 57), por isso, insira um valor de 0xc (queremos inspecionar três

registos de E/S de 4 bytes, pelo que precisamos de 12 bytes).

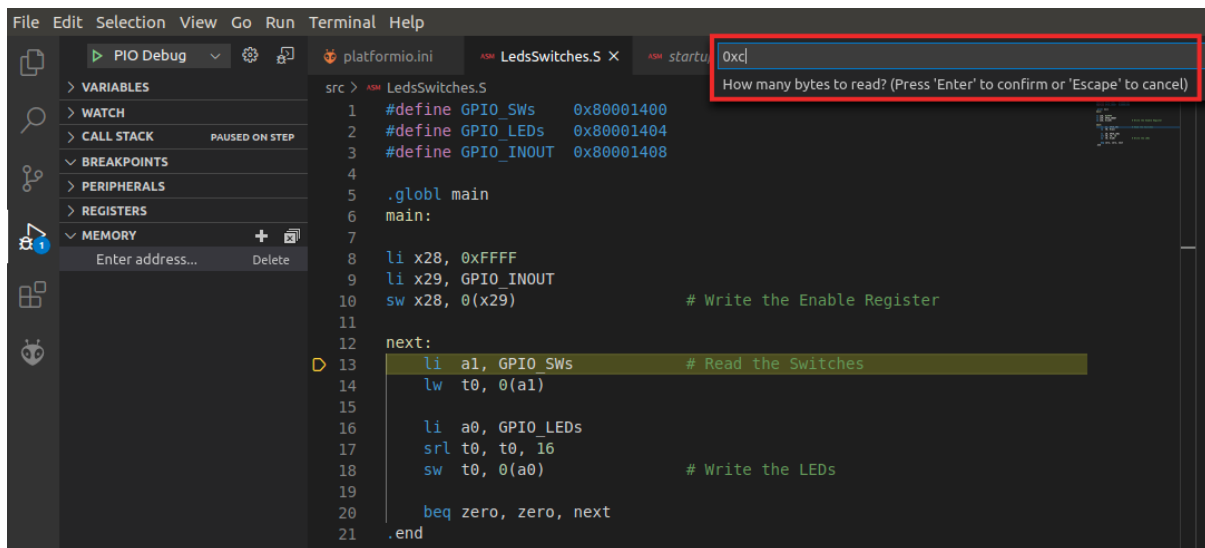


Figura 57. Número de bytes a mostrar

- d. O Memory Display abrir-se-á à direita, mostrando os 12 bytes pedidos (Figura 58). O valor que temos nos 16 interruptores é 0x123C (ver os bytes nos endereços 0x80001402 e 0x80001403). Tendo em conta que a arquitetura RISC-V é *little endian*, o valor apresentado na figura é coerente com isso. Os 16 LEDs (armazenados nos endereços 0x80001404 e 0x80001405) mostram o mesmo valor.

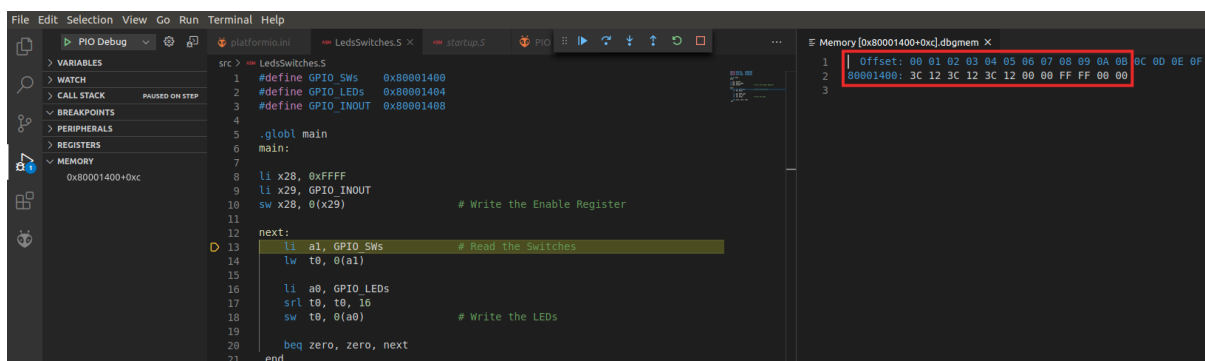


Figura 58. Endereços de memória 0x80001400-0x8000140B

- e. Altere os valores dos interruptores na placa, por exemplo para 0x5555, e execute mais uma iteração do ciclo passo a passo. O valor dos interruptores na memória deve mudar imediatamente após a execução da primeira instrução (Figura 59, topo), e o valor dos LEDs deve mudar em conformidade após a execução da instrução `sw` (Figura 59, fundo).

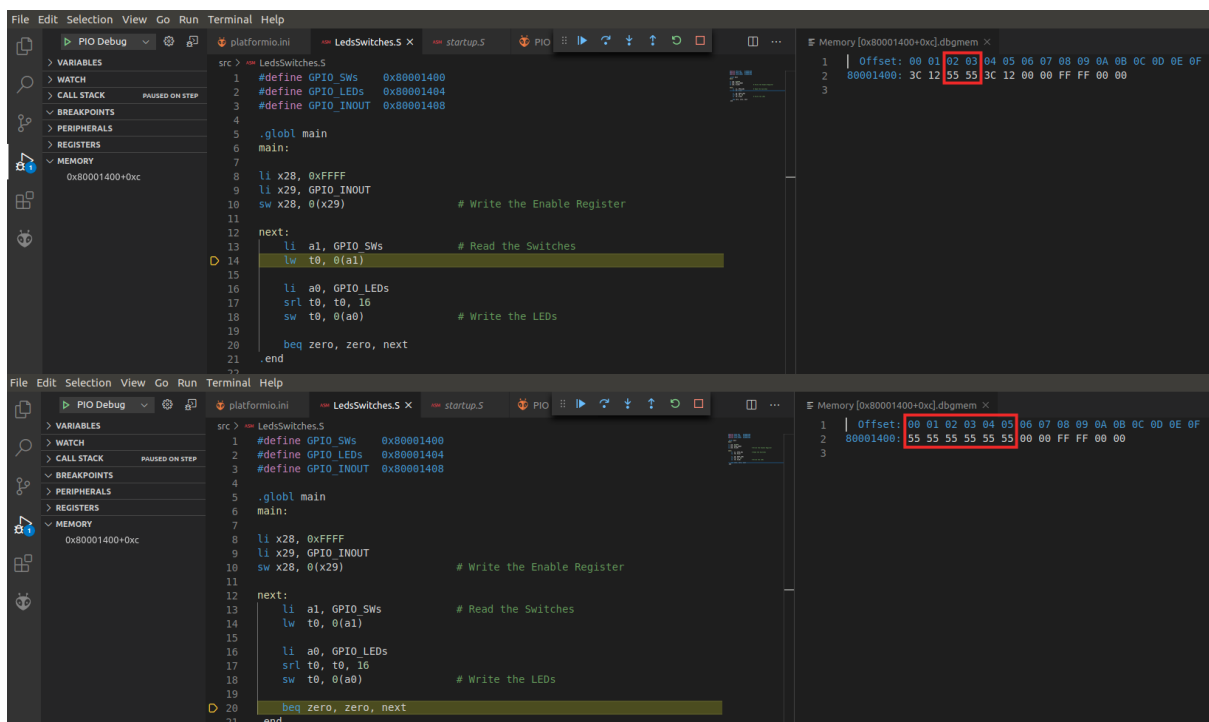


Figura 59. Mudança dos interruptores e LEDs

- f. Pode também visualizar outras posições de memória, como os endereços RAM que armazenam as instruções de máquina do seu programa. Abra outra faixa de memória começando em 0x0 (endereço inicial atribuído à memória RAM) e ocupando 0x100 bytes (Figura 60). Verá as instruções do programa LedsSwitches armazenadas no intervalo de endereços 0x90-0xC4, logo após o programa de arranque (Startup.S).

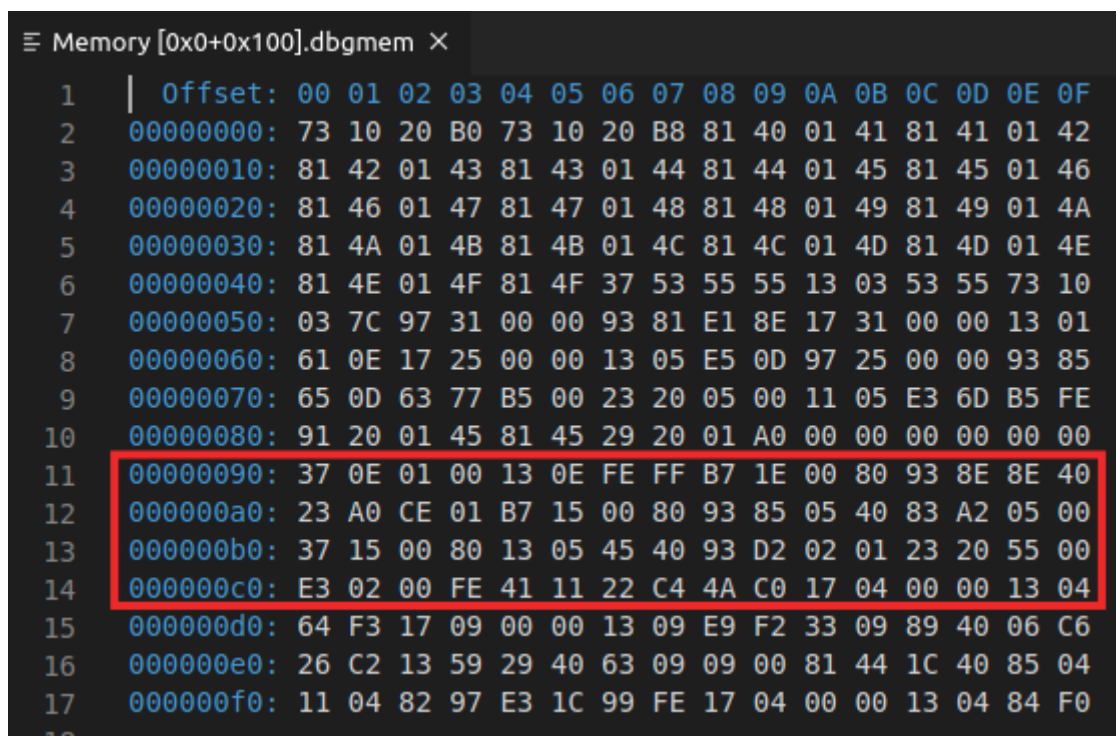


Figura 60. Endereços de memória de 0x0 a 0x100

- g. Pode ver o código de máquina das instruções do programa abrindo o Disassembly do programa disponível em: *[RVfpgaPath]/RVfpga/examples/LedsSwitches/.pio/build/swervolf_nexys/firmware.dis* (see Figura 61). Compara as duas figuras e tenta identificar as instruções do programa.

```

65 Disassembly of section .text:
66
67 00000090 <main>:
68   90: 00010e37      lui t3,0x10
69   94: fffe0e13      addi t3,t3,-1 # ffff <_sp+0xcbbf>
70   98: 80001eb7      lui t4,0x80001
71   9c: 408e8e93      addi t4,t4,1032 # 80001408 <OVERLAY_END_OF_OVERLAYS+0xa0001408>
72   a0: 01cea023      sw t3,0(t4)
73
74 000000a4 <next>:
75   a4: 800015b7      lui a1,0x80001
76   a8: 40058593      addi a1,a1,1024 # 80001400 <OVERLAY_END_OF_OVERLAYS+0xa0001400>
77   ac: 0005a283      lw t0,0(a1)
78   b0: 80001537      lui a0,0x80001
79   b4: 40450513      addi a0,a0,1028 # 80001404 <OVERLAY_END_OF_OVERLAYS+0xa0001404>
80   b8: 0102d293      srli t0,t0,0x10
81   bc: 00552023      sw t0,0(a0)
82   c0: fe0002e3      beqz zero,a4 <next>
83

```

Figura 61. Versão Disassembly do programa LedsSwitches

E. Programa LedsSwitches_C-Lang

O programa LedsSwitches_C-Lang.c (Figura 62) faz o mesmo que o programa LedsSwitches.s já apresentado (Figura 53) mas está escrito em C em vez de Assembly.

```

1  #define GPIO_SWs      0x80001400
2  #define GPIO_LEDs     0x80001404
3  #define GPIO_INOUT    0x80001408
4
5  #define READ_GPIO(dir) (*(volatile unsigned *)dir)
6  #define WRITE_GPIO(dir, value) { (*(volatile unsigned *)dir) = (value); }
7
8  int main ( void )
9  {
10     int En_Value=0xFFFF, switches_value;
11
12     WRITE_GPIO(GPIO_INOUT, En_Value);
13
14     while (1) {
15         switches_value = READ_GPIO(GPIO_SWs);
16         switches_value = switches_value >> 16;
17         WRITE_GPIO(GPIO_LEDs, switches_value);
18     }
19
20     return(0);
21 }

```

Figura 62. LedsSwitches_C-Lang.c

Siga os passos seguintes para executar e depurar este programa na placa FPGA:

1. O RVfpgaNexys já está programado na placa FPGA se tiver executado os exemplos anteriores, pelo que não deverá ser necessário programá-lo novamente. No entanto, se

precisar de reprogramar o RVfpgaNexys na placa novamente, faça-o como explicado na Secção A, utilizando o exemplo LedsSwitches_C-Lang em vez do exemplo AL_Operations.

2. Na barra de menu superior, clique em *File* → *Open Folder*, e navegue até à pasta *[RVfpgaPath]/RVfpga/examples/*. Selecione a pasta *LedsSwitches_C-Lang* e clique em OK.
3. Antes de chamar o depurador, defina um *breakpoint* na linha 15 do código C.
4. Em seguida, inicie a depuração. O programa começa a ser executado e é suspenso no *breakpoint* (Figura 63).

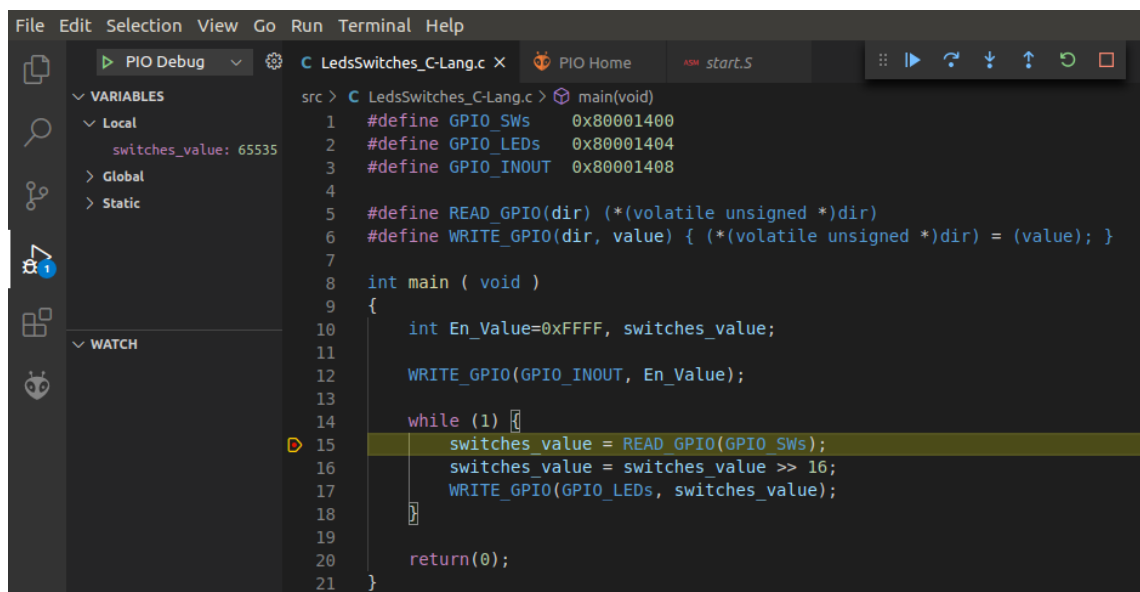



Figura 63. A execução foi interrompida no *breakpoint*

5. Fazer com que o programa continue a ser executado  várias vezes, mas altere os interruptores entre cada clique. Os LEDs devem mostrar o valor dos interruptores.
6. Pode ver a execução do programa em C, como anteriormente, ou pode ver a execução do programa Assembly gerado pelo compilador, clicando em *Switch to Assembly* como destacado na Figura 64.

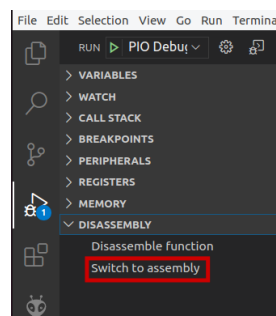


Figura 64. Mudar para Assembly

7. O programa em Assembly (Figura 65) primeiro lê o valor nos Switches com uma instrução `load (lw a5, 1024(a4))` e depois escreve-o nos LEDs com uma instrução de

armazenamento (`sw a5, 1028(a4)`). Execute-o passo a passo, altere os interruptores e verifique se os LEDs mudam para refletir os novos valores dos interruptores.

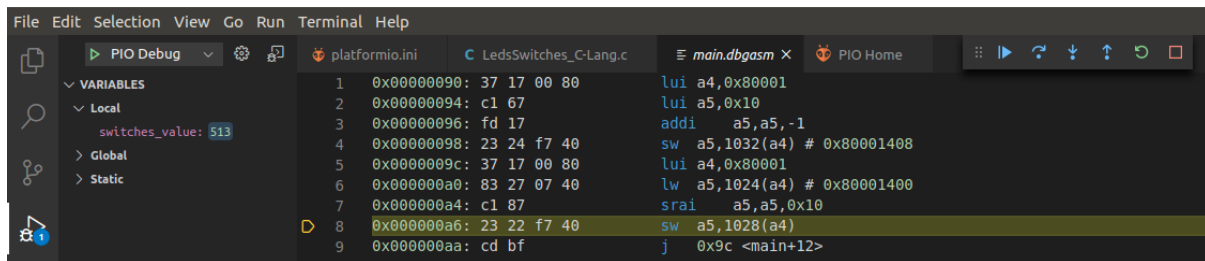


Figura 65. Programa Assembly

F. Programa HelloWorld_C-Lang

O segundo exemplo em C mostra uma mensagem curta para a sua shell através da porta série. Para ver esta mensagem, pode usar qualquer emulador de terminal como o *gtkterm*, *minicom*, etc.; No entanto, o PlatformIO fornece o seu próprio monitor de série, pelo que aqui mostramos como utilizar este monitor.

Para configurar o monitor serial PlatformIO, alguns parâmetros devem ser configurados; especificamente, a taxa de dados (em bits por segundo, ou bauds) para transmissão de dados em série deve ser estabelecida, o que podemos fazer usando o parâmetro *monitor_speed* no ficheiro *platformio.ini* (note que este ficheiro faz parte dos seus projetos PlatformIO). Ver a Figura 66.

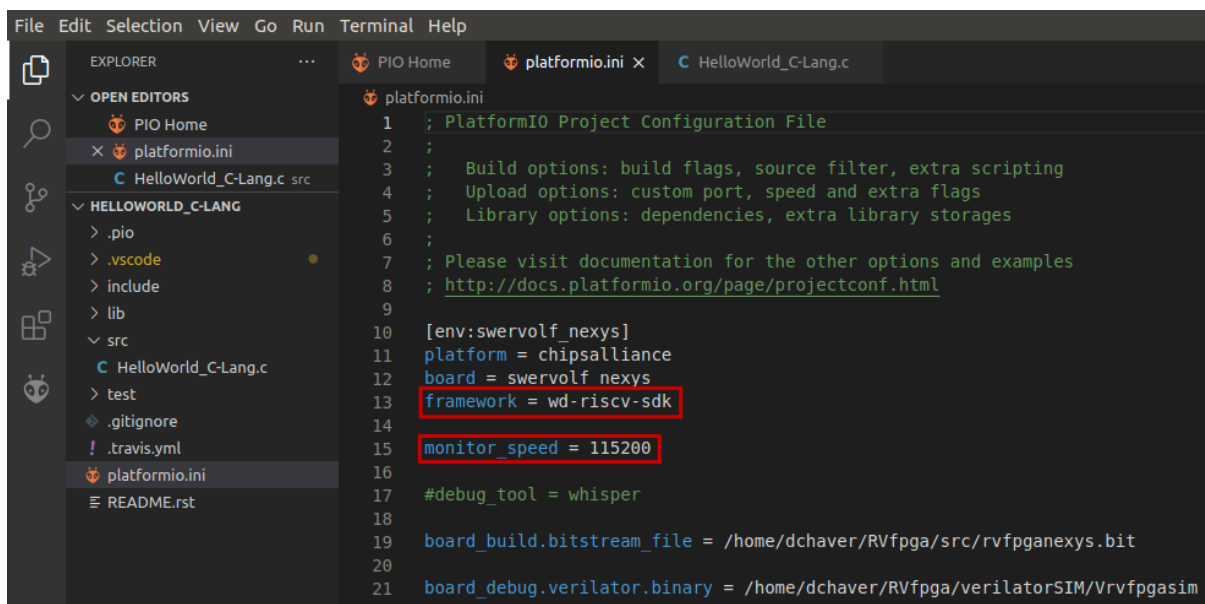


Figura 66. Configuração do Serial monitor

Para além disso, tem de se adicionar aos grupos *dialout*, *tty* e *uucp* escrevendo os seguintes comandos num terminal:

```
sudo usermod -a -G dialout $USER
sudo usermod -a -G tty $USER
sudo usermod -a -G uucp $USER
```

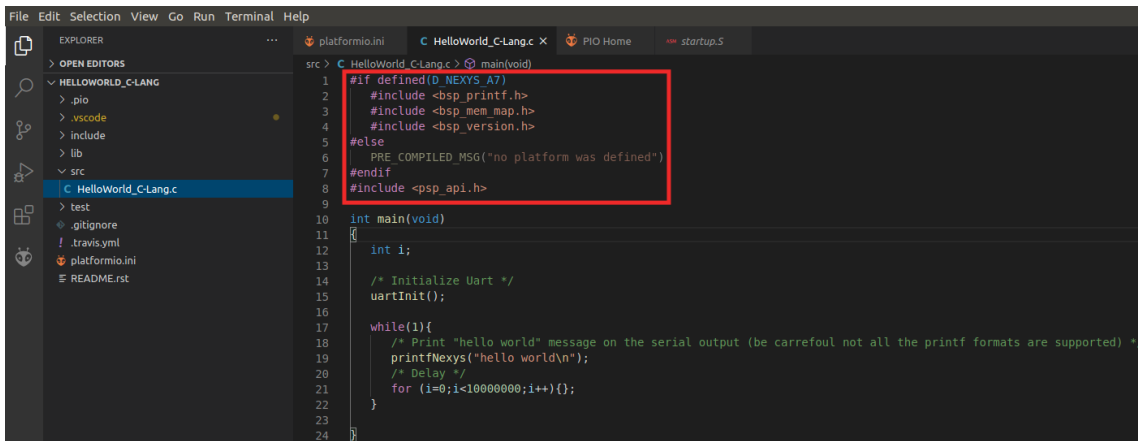
Após os três comandos, reinicie o computador para que as alterações nos grupos tenham efeito.

Windows/macOS: Os utilizadores do Windows e do macOS não precisam de concluir o passo acima.

Além disso, este programa utiliza o Processor Support Package (PSP) e o Board Support Package (BSP) fornecidos pela WD no seu Firmware Package (<https://github.com/westerndigitalcorporation/riscv-fw-infrastructure>). Estas bibliotecas são incluídas no projeto através de um comando específico no `platformio.ini` (`framework = wd-riscv-sdk`), como mostra a Figura 66, e incluindo os ficheiros adequados no início do programa C, como mostra a Figura 67. Pode encontrar as bibliotecas completas no seu sistema nos seguintes caminhos:

- PSP: `~/.platformio/packages/framework-wd-riscv-sdk/psp/`
- BSP: `~/.platformio/packages/framework-wd-riscv-sdk/board/nexys_a7_ehl/bsp/`

Estas bibliotecas fornecem muitas funções e macros que permitem fazer muitas coisas, como utilizar interrupções, escrever uma cadeia de caracteres, ler/escrever registos individuais... Neste exemplo, vamos utilizar a função `printfNexys` para escrever uma mensagem no monitor de série. Nos exemplos subsequentes e nos laboratórios, mostraremos como utilizar outras funções e macros para diferentes fins.



```

src > C: HelloWorld_C-Lang.c > main(void)
1  #if defined(D_NEXYS_A7)
2  #include <bsp_printf.h>
3  #include <bsp_mem_map.h>
4  #include <bsp_version.h>
5  #else
6  PRE_COMPILED_MSG("no platform was defined")
7  #endif
8  #include <psp_api.h>
9
10 int main(void)
11 {
12     int i;
13     /* Initialize Uart */
14     uartInit();
15
16     while(1){
17         /* Print "hello world" message on the serial output (be careful not all the printf formats are supported) */
18         printfNexys("hello world\n");
19         /* Delay */
20         for (i=0; i<100000000; i++){
21             ;
22         }
23     }
24 }

```

Figura 67. Incluir os ficheiros .h em *HelloWorld_C-Lang.c*

Siga os passos seguintes para executar e depurar este código na placa FPGA:

1. O RVfpgaNexys já está programado na placa FPGA se tiver executado os exemplos anteriores, pelo que não deverá ser necessário programá-lo novamente. No entanto, se precisar de reprogramar o RVfpgaNexys na placa novamente, faça-o como explicado na Secção A, utilizando o exemplo `HelloWorld_C-Lang` em vez do exemplo `AL_Operations`.
2. Abra o VSCode. O PlatformIO deve abrir automaticamente dentro do VSCode quando abrir o VSCode. Na barra superior, clique em `File` → `Open Folder`, e navegue até à pasta `[RVfpgaPath]/RVfpga/examples/`. Selecione a pasta `HelloWorld_C-Lang` e clique em `OK`.

- O programa *HelloWorld_C-Lang.C* (Figura 68) inicializa a UART (função **uartInit**) e depois envia a cadeia de caracteres através da porta série, utilizando a função **printfNexys** (pode encontrar a implementação destas funções no ficheiro `~/platformio/packages/framework-wd-riscv-sdk/board/nexys_a7_gh1/bsp/bsp_printf.c`). Em seguida, atrasa algum tempo antes de voltar ao início do ciclo.

```

1  #if defined(D_NEXYS_A7)
2  #include <bsp_printf.h>
3  #include <bsp_mem_map.h>
4  #include <bsp_version.h>
5  #else
6  PRE_COMPILED_MSG("no platform was defined")
7  #endif
8  #include <psp_api.h>
9
10 int main(void)
11 {
12     int i;
13
14     /* Initialize Uart */
15     uartInit();
16
17     while(1){
18         /* Print "hello world" message on the serial output (be careful not all the printf formats are supported) */
19         printfNexys("hello world\n");
20         /* Delay */
21         for (i=0;i<10000000;i++){};
22     }
23 }

```

Figura 68. Função main do HelloWorld_C-Lang.C

- Inicie o depurador na PlatformIO. Quando o programa começar a ser executado, abra o monitor de série, clicando no botão "da ficha" disponível na parte inferior do VS Code (Figura 69).

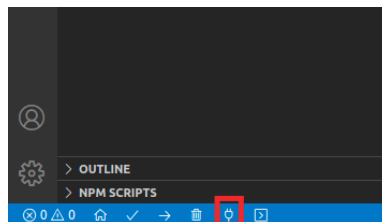
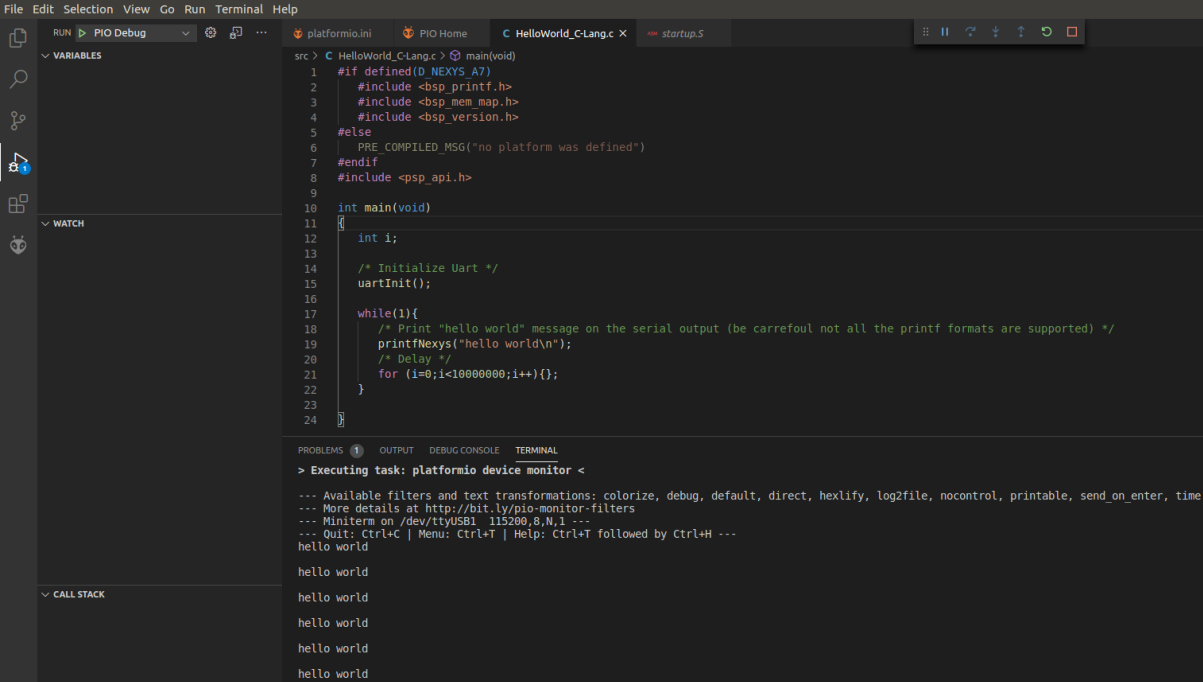


Figura 69. Abrir o serial terminal

- O monitor de série mostra repetidamente a mensagem "HELLO WORLD !!!", como na Figura 70.



```

src > C HelloWorld_C-Lang.c > main(void)
1  #if defined(D_NEXYS_A7)
2  #include <bsp_printf.h>
3  #include <bsp_mem_map.h>
4  #include <bsp_version.h>
5  #else
6  #define PRE_COMPILED_MSG("no platform was defined")
7  #endif
8  #include <bsp_api.h>
9
10 int main(void)
11 {
12     int i;
13
14     /* Initialize Uart */
15     uartInit();
16
17     while(1){
18         /* Print "hello world" message on the serial output (be careful not all the printf formats are supported) */
19         printfNexys("hello world\n");
20         /* Delay */
21         for (i=0; i<100000000; i++){
22             ;
23         }
24     }

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

> Executing task: platformio device monitor <

```

... Available filters and text transformations: colorize, debug, default, direct, hexlify, log2file, nocontrol, printable, send_on_enter, time
... More details at http://bit.ly/pio-monitor-filters
... Miniterm on /dev/ttyUSB1 115200,8,N,1 ---
... Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
hello world
hello world
hello world
hello world
hello world

```

Figura 70. Execução do programa

G. Programa VectorSorting_C-Lang

Finalmente, mostramos outro programa em C que ordena os elementos de um vetor, A, do maior para o menor e coloca os valores ordenados num segundo vetor, B. Os valores do vetor A são substituídos por zeros. A Figura 71 mostra o programa.

```

1  #define N 8
2
3  int A[N]={7,3,25,4,75,2,1,1};
4  int B[N];
5
6  int main ( void )
7  {
8      int max, ind, i, j;
9
10     for(j=0; j<N; j++){
11         max=0;
12         for(i=0; i<N; i++){
13             if (A[i]>max){
14                 max=A[i];
15                 ind=i;
16             }
17         }
18         B[j]=A[ind];
19         A[ind]=0;
20     }
21
22     while(1);
23 }

```

Figura 71. VectorSorting_C-Lang.c

Siga os passos seguintes para executar e depurar este programa na placa FPGA:

1. O RVfpgaNexys já está programado na placa FPGA se tiver executado os exemplos anteriores, pelo que não deverá ser necessário programá-lo novamente. No entanto, se

precisar de reprogramar o RVfpgaNexys na placa novamente, faça-o como explicado na Secção A, utilizando o exemplo VectorSorting_C-Lang em vez do exemplo AL_Operations.

2. Na barra de menu superior, clique em *File* → *Open Folder*, e navegue até à pasta *[RVfpgaPath]/RVfpga/examples/*. Selecione a pasta *VectorSorting_C-Lang* e clique em OK.
3. Coloque um *breakpoint* na linha 10 e inicie a depuração. A execução irá parar no início da linha do ciclo `for` (Figura 72). Expanda a secção *VARIABLES* na barra lateral do depurador e analise os valores dos vetores A e B (destacados a vermelho na Figura 72).

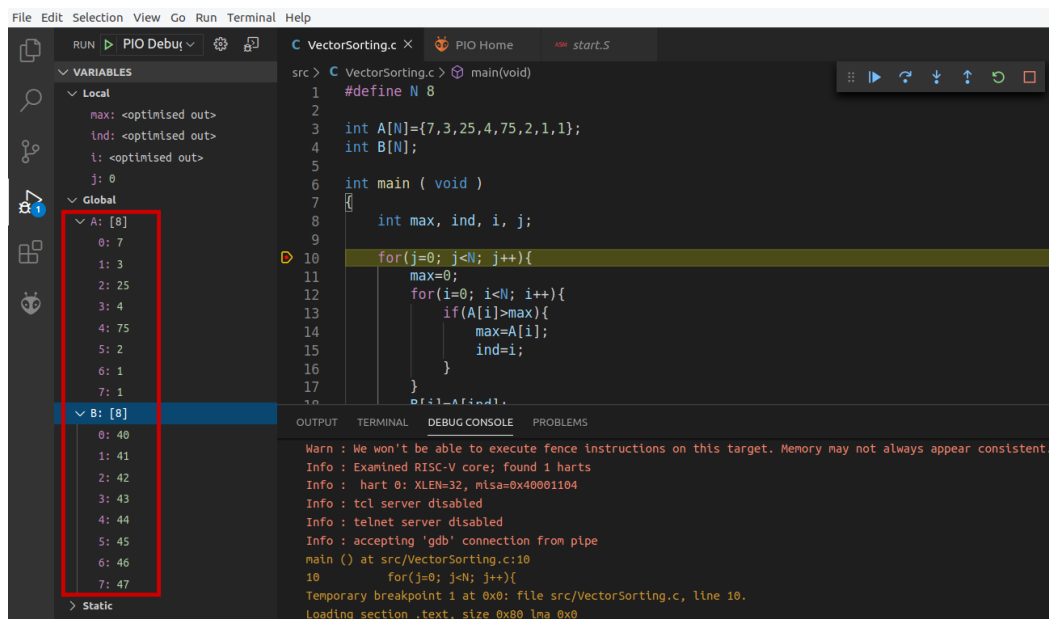



Figura 72. A execução para no início do programa

4. Agora, coloque outro ponto de interrupção na linha 18 e continue a execução clicando em  (Figura 73). Abra o Memory Display (como explicado para o programa LedsSwitches, Figura 55) e veja os 0x50 bytes a partir do endereço 0x2148 (Figura 73), que é o endereço onde o vetor A está armazenado na memória para este programa. É possível ver os valores iniciais dos vetores A (no intervalo 0x2148-0x2167) e B (no intervalo 0x2178-0x2197).

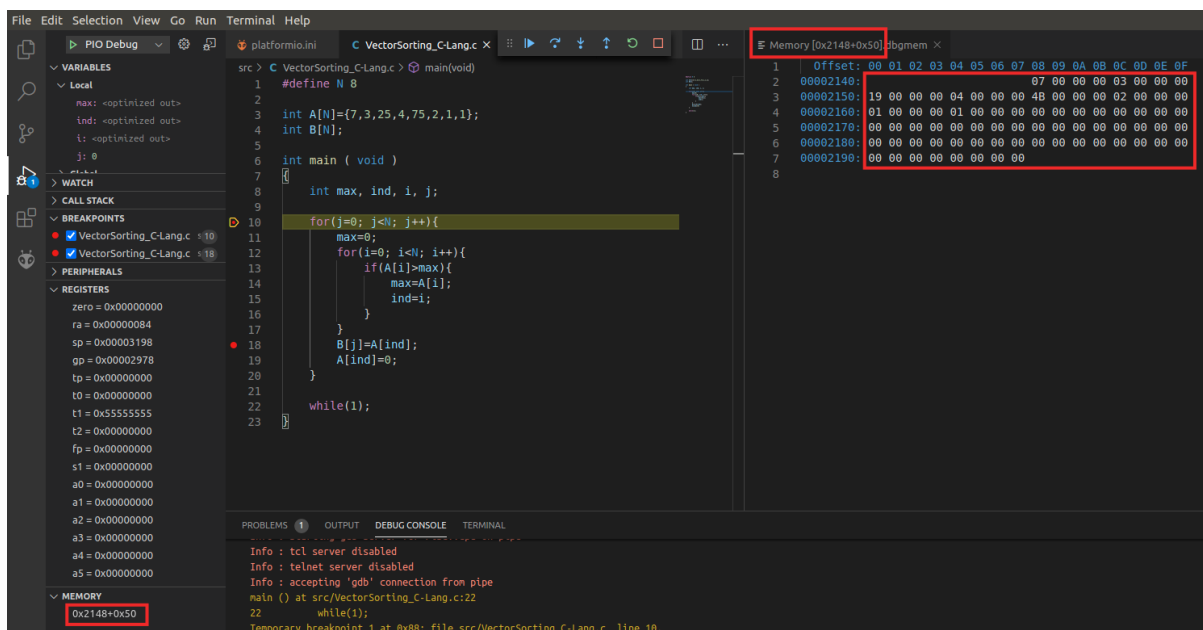


Figura 73. Memory Display para os vetores A e B - Estado inicial.

Note que pode facilmente descobrir o endereço onde os vetores A e B estão armazenados na memória mudando para Assembly, como explicado na Figura 64, e analisando qualquer uma das instruções que acedem a estes vetores (Figura 74). Como se pode ver na figura, na maioria dos casos os comentários fornecem esta informação; no entanto, também se pode ir até essas instruções e ver o valor que é armazenado no registo.

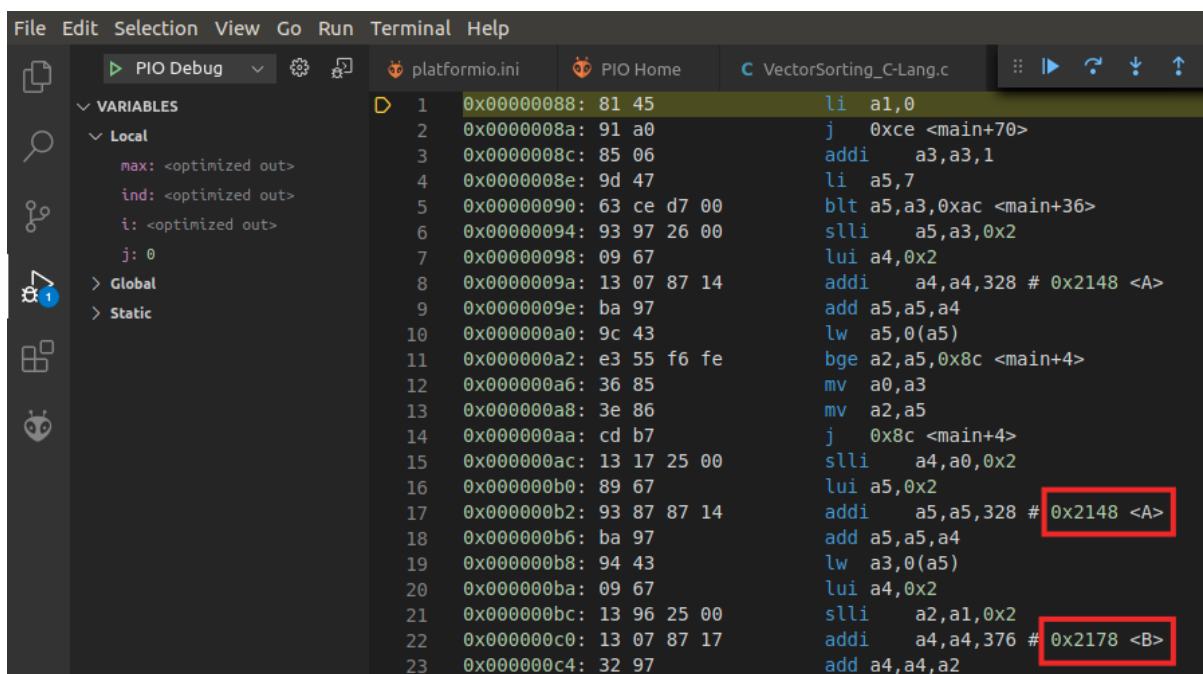
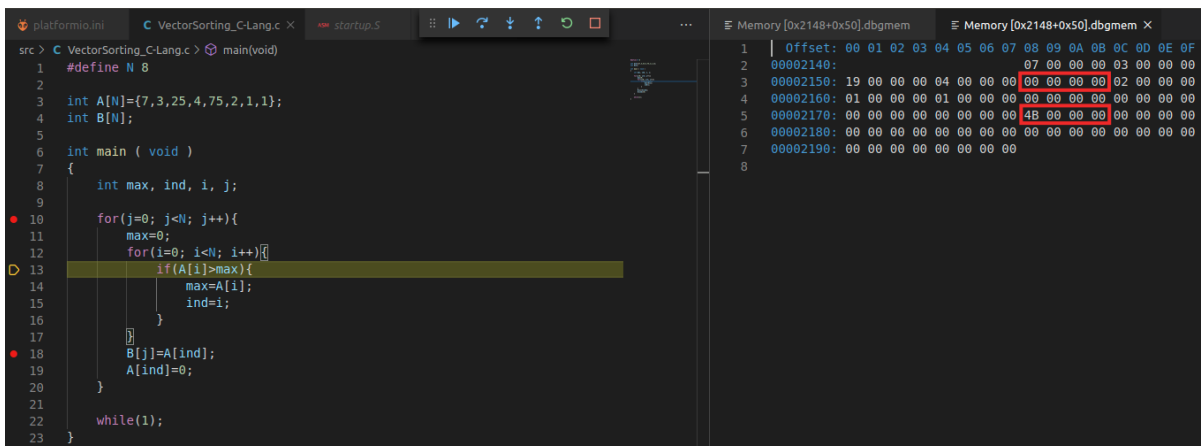


Figura 74. Endereços onde A e B estão armazenados na memória.

Clique duas vezes no botão Step Over (↻), e verá o primeiro elemento de B guardado na memória e o valor correspondente em A definido como 0 (Figura 75).



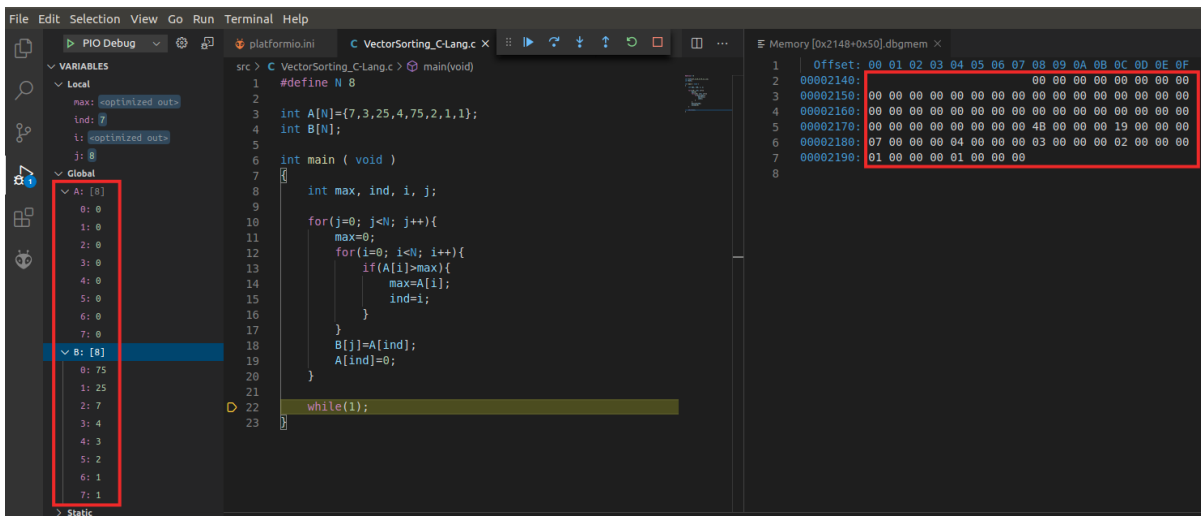
```

src > C VectorSorting_C-Lang.c > main(void)
1 #define N 8
2
3 int A[N]={7,3,25,4,75,2,1,1};
4 int B[N];
5
6 int main ( void )
7 {
8     int max, ind, i, j;
9
10    for(j=0; j<N; j++){
11        max=0;
12        for(i=0; i<N; i++){
13            if(A[i]>max){
14                max=A[i];
15                ind=i;
16            }
17        }
18        B[j]=A[ind];
19        A[ind]=0;
20    }
21
22    while(1);
23 }
  
```

Offset	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00002140:	19	00	00	00	04	00	00	00	00	00	00	00	00	00	00	00
00002150:	01	00	00	00	01	00	00	00	00	00	00	00	00	00	00	00
00002160:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00002170:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00002180:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00002190:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Figura 75. Memory Display para os vetores A e B - Armazena o primeiro elemento de B e anula o elemento correspondente em A.

5. Remova todos os *breakpoints*, continue a execução e faça uma pausa após alguns segundos - altura em que o programa terá terminado a execução. Analise novamente os valores armazenados nos vetores A e B. Como mostrado na Figura 76, O vetor B contém os valores do vetor original A ordenados do maior para o menor e o vetor A contém todos os zeros (pode ver isto tanto na lista de variáveis à esquerda como na consola de memória à direita).



```

src > C VectorSorting_C-Lang.c > main(void)
1 #define N 8
2
3 int A[N]={7,3,25,4,75,2,1,1};
4 int B[N];
5
6 int main ( void )
7 {
8     int max, ind, i, j;
9
10    for(j=0; j<N; j++){
11        max=0;
12        for(i=0; i<N; i++){
13            if(A[i]>max){
14                max=A[i];
15                ind=i;
16            }
17        }
18        B[j]=A[ind];
19        A[ind]=0;
20    }
21
22    while(1);
23 }
  
```

Offset	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00002140:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00002150:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00002160:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00002170:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00002180:	07	00	00	04	00	00	00	03	00	00	00	02	00	00	00	00
00002190:	01	00	00	00	01	00	00	00	00	00	00	00	00	00	00	00

Figura 76. A execução foi interrompida no final do programa

H. Programa DotProduct_C-Lang

O último programa de exemplo, DotProduct_C-Lang.c (Figura 77), calcula o produto escalar de dois vetores. O programa tem duas funções: *main* e *dotproduct*. A primeira função invoca a segunda com três argumentos de entrada: tamanho do vetor e os endereços iniciais de dois vetores. Em seguida, a função *dotproduct* calcula o produto escalar dos dois vetores e devolve o resultado.

```

1  #define DIM 3
2
3  double dot;
4
5  double dotproduct(int n, double a[], double b[]){
6      volatile int i;
7      double sum=0;
8
9      for (i=0; i<n; i++) {
10         sum += a[i]*b[i];
11     }
12     return sum;
13 }
14
15 void main(void) {
16     double x[DIM] = {3.1, 4.3, 5.9};           // x é um vetor de tamanho 3 (DIM)
17     double y[DIM] = {1.4, 2.2, 3.7};         // como x
18
19     dot = dotproduct(DIM, x, y);
20
21     return;
22 }

```

Figura 77. DotProduct_C-Lang.c

Neste exemplo, operamos com números reais (note que o tipo de dados para as variáveis `x`, `y` e `dot`, são `double`). No entanto, o processador SweRV EH1 não inclui suporte para vírgula flutuante. Assim, o exemplo usa emulação de vírgula flutuante através da biblioteca de software fornecida pela `gcc` (<https://gcc.gnu.org/onlinedocs/gccint/Soft-float-library-routines.html>). Esta biblioteca é utilizada sempre que `-msoft-float` é incluído para desativar a geração de instruções de vírgula flutuante.

Siga os passos seguintes para executar e depurar este código na placa FPGA:

1. O RVfpgaNexys já está programado na placa FPGA se tiver executado os exemplos anteriores, pelo que não deverá ser necessário programá-lo novamente. No entanto, se precisar de reprogramar o RVfpgaNexys na placa novamente, faça-o como explicado na Secção A, utilizando o exemplo DotProduct_C-Lang em vez do exemplo AL_Operations.
2. Na barra de menu superior, clique em *File* → *Open Folder*, e navegue até à pasta `[RVfpgaPath]/RVfpga/examples/`. Selecione a pasta `DotProduct_C-Lang` e clique em OK.
3. Antes de chamar o depurador, defina um *breakpoint* na linha 10 e outro na linha 19 (Figura 78).
4. De seguida, inicie a depuração. O programa começará a ser executado; pare-o no primeiro *breakpoint* (Figura 78).
5. Na barra lateral do Depurador, expanda a secção Variables (Variáveis) (Figura 78). Os dois vetores contêm os valores iniciais atribuídos em *main*. A variável `dot` é inicializada em 0.

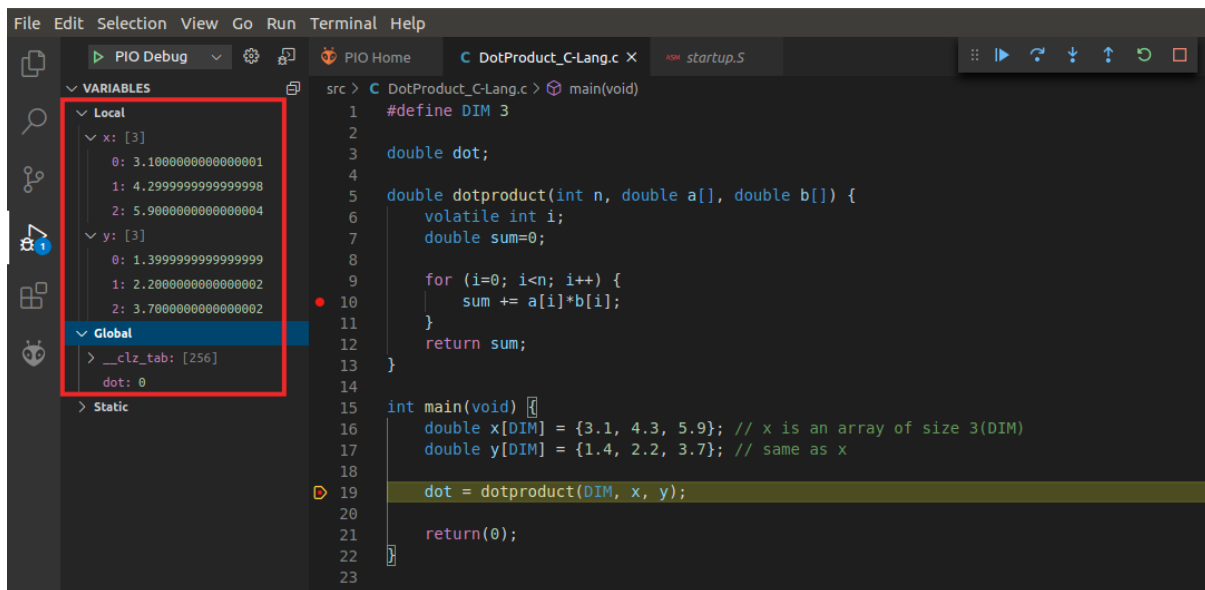



Figura 78. Programa DotProduct_C-Lang: valores das variáveis no primeiro *breakpoint*

6. Fazer com que o programa continue a ser executado . O programa é suspenso no segundo *breakpoint* (linha 10).
7. Mude para Assembly (como fez na Figura 64). É possível ver as rotinas de emulação de ponto flutuante e analisá-las em pormenor entrando nelas (Figura 79).

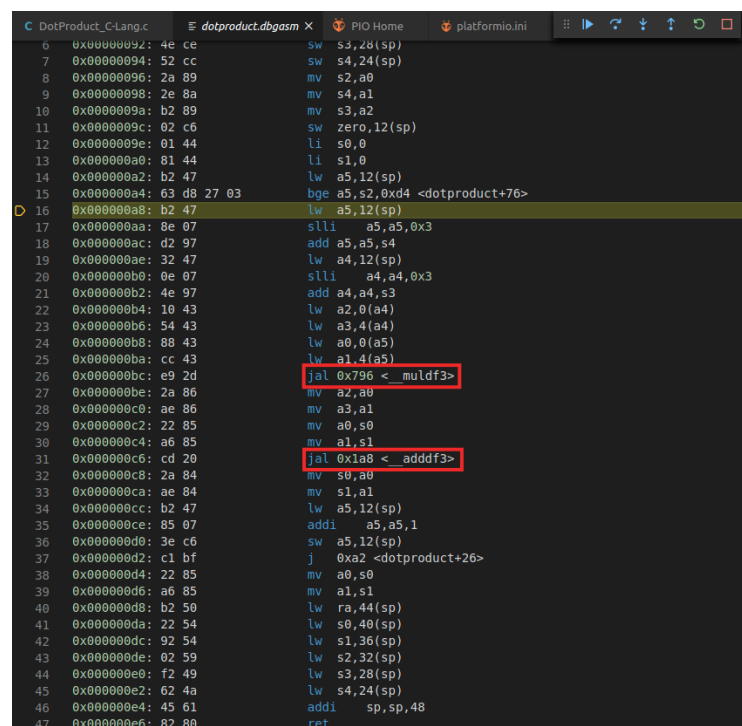
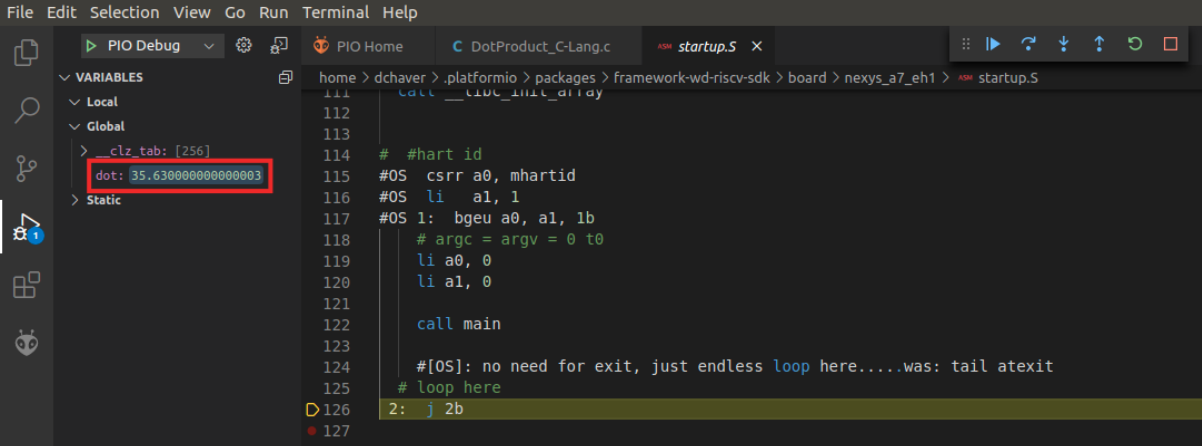


Figura 79. Programa DotProduct_C-Lang: código Assembly no segundo *breakpoint*

8. Regresse ao C e elimine os dois *breakpoints*. Continue a execução e faça uma pausa. Verá que o valor da variável *dot* mudará para o produto escalar dos dois vetores (Figura

80).



The screenshot shows the PIO IDE interface. On the left, the 'VARIABLES' panel is expanded, showing a variable 'dot' with a value of 35.630000000000003, which is highlighted with a red box. The main editor displays the assembly code for 'startup.S'. The code includes comments and instructions for initializing the array, setting the hart id, and calling the main function. The code is as follows:

```

111  call __libc_init_array
112
113
114  # #hart id
115  #OS csrr a0, mhartid
116  #OS li a1, 1
117  #OS 1: bgeu a0, a1, 1b
118  # argc = argv = 0 t0
119  li a0, 0
120  li a1, 0
121
122  call main
123
124  #[OS]: no need for exit, just endless loop here.....was: tail atexit
125  # loop here
126  2: j 2b
127

```

Figura 80. Programa DotProduct_C-Lang: resultado do produto escalar

9. Quando tiver terminado de explorar este programa, feche o projeto clicando em *File* → *Close Folder*.

7. SIMULAÇÃO COM O VERILATOR

Nesta secção, irá executar o primeiro programa utilizado na secção anterior (*AL_Operations*) no RVfpgaSim utilizando o Verilator. O Verilator é um simulador de linguagem de descrição de hardware (HDL) que simula o Verilog que define o SoC (disponível em *[RVfpgaPath]/RVfpga/src*). Esta forma de executar o SoC permite-lhe analisar os sinais internos do sistema, o que é especialmente útil para futuros laboratórios e exercícios em que adicionamos operações internas ou novo hardware ao SoC.

Aqui mostramos como utilizar o Verilator para ver as instruções ciclo-a-ciclo e os valores de registo do *AL_Operations*, o primeiro programa Assembly simples que executou e depurou na Secção 6 (Figura 44). Irá gerar o *trace* de simulação utilizando o PlatformIO e, em seguida, adicionar os sinais do relógio, as instruções para ambas as vias do processador superescalar e o registo *x28* (i.e., registo *t3*) para a forma de onda de simulação e visualizar com o GTKWave a instrução e os sinais de registo que variam à medida que o programa é executado.

GERAR O BINÁRIO DE SIMULAÇÃO, Vrvfpgasim:

A pasta *[RVfpgaPath]/RVfpga/verilatorSIM* contém o ficheiro *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 as fontes para o SoC, que no nosso caso estão disponíveis em *[RVfpgaPath]/RVfpga/src*. Em seguida, mostramos como gerar o binário para o RVfpgaSim, que mais tarde será usado para criar o *trace* de simulação do programa *AL-Operations* em execução no RVfpgaSim.

1. Numa janela de terminal, gerar o binário do simulador executando os seguintes comandos:

```
cd [RVfpgaPath]/RVfpga/verilatorSIM
make clean
make
```

O ficheiro **Vrvfpgasim** (o binário de simulação RVfpgaSim), deve ser gerado dentro da pasta *[RVfpgaPath]/RVfpga/verilatorSIM*.

Windows: se estiver a utilizar o Windows, deve efetuar estes mesmos passos no terminal Cygwin (consulte o Apêndice C para obter instruções detalhadas). Note que a pasta C: Windows pode ser encontrada dentro do Cygwin em: */cygdrive/c*. Todas as outras instruções desta secção são as mesmas que as descritas para Linux.

macOS: Consultar o Apêndice D para obter instruções pormenorizadas.

GERAR O TRACE DA SIMULAÇÃO A PARTIR DO PLATFORMIO, UTILIZANDO Vrvfpgasim:

Quando o binário do simulador (*Vrvfpgasim*) for gerado, utilizá-lo-á no PlatformIO para gerar o *trace* da simulação (*trace.vcd*) do programa *AL_Operations*.

2. Abra o VSCode e, em seguida, o PlatformIO no seu computador.
3. Na barra superior, clique em *File→Open Folder...* (Figura 81), e navegue até à pasta

[RVfpgaPath]/RVfpga/examples/

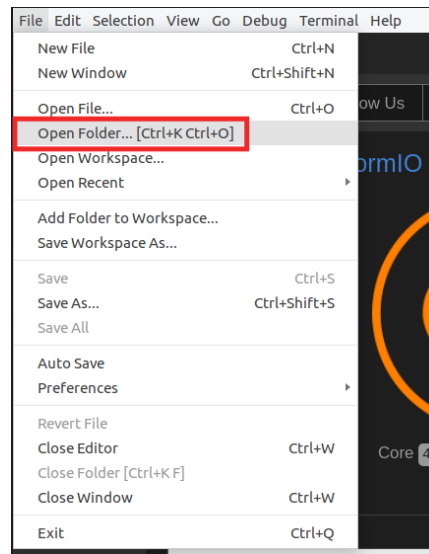


Figura 81. Abrir o exemplo AL_Operations.S

4. Selecionar a pasta *AL_Operations* (não a abra, mas apenas a selecione) e clique em OK. O exemplo será aberto no PlatformIO.
5. Abra o ficheiro *platformio.ini*. Estabeleça o caminho para o binário da simulação RVfpgaSim gerado na primeira etapa (*Vrvfpgasim*) editando a seguinte linha (Figura 82).

```
board_debug.verilator.binary =
[RVfpgaPath]/RVfpga/verilatorSIM/Vrvfpgasim
```

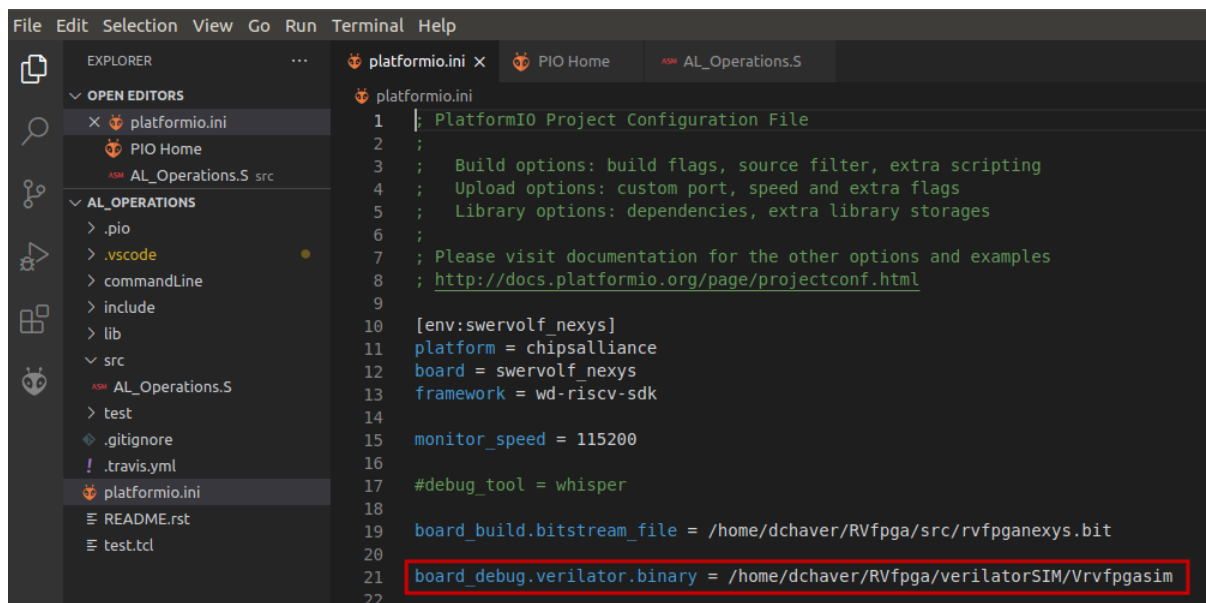


Figura 82. Ficheiro de inicialização do PlatformIO: platformio.ini

Windows: em Windows, o executável de simulação RVfpgaSim é chamado *Vrvfpgasim.exe*. Assim:

```
board_debug.verilator.binary = [RVfpgaPath]\RVfpga\verilatorSIM\Vrvfpgasim.exe
```

- Execute a simulação clicando no ícone PlatformIO na barra de menu à esquerda , depois expanda Project Tasks → env:swervolf_nexys → Platform e clique em Generate Trace, como mostra a Figura 83.

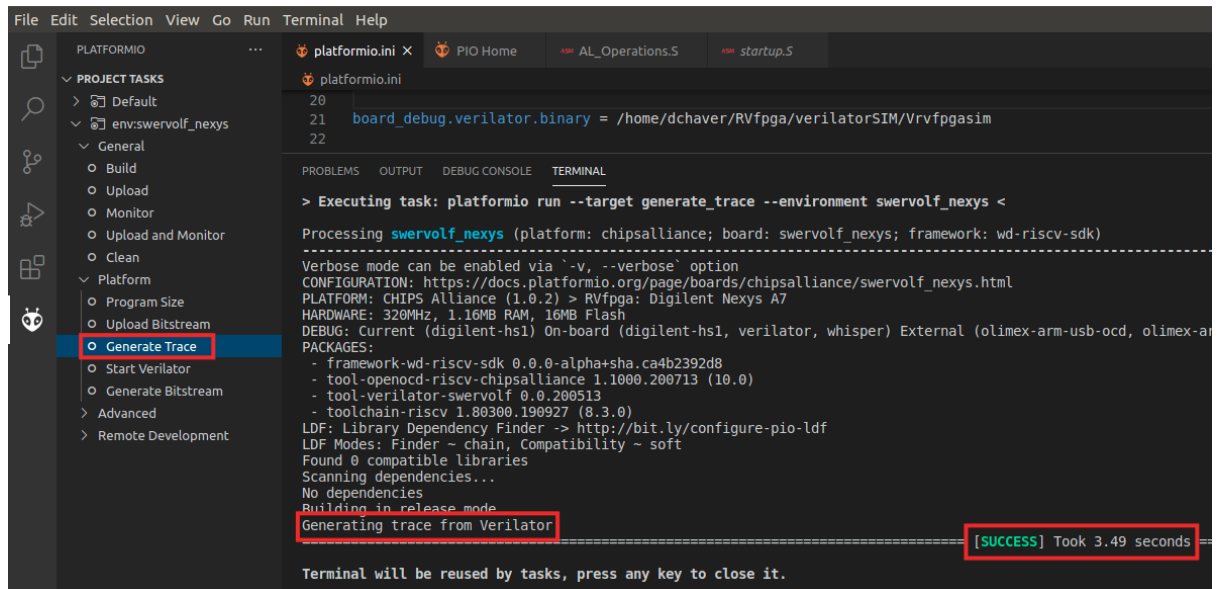



Figura 83. Geração do trace do Verilator

Em alternativa, pode gerar o traço a partir de uma janela do terminal PlatformIO. Para o efeito, clique no botão  (PlatformIO: New Terminal) na parte inferior da janela PlatformIO para abrir uma nova janela de terminal e, em seguida, escreva (ou copie) o seguinte comando no terminal PlatformIO: `pio run --target generate_trace`

- Alguns segundos após o passo anterior, o ficheiro `trace.vcd` deve ter sido gerado dentro do `[RVfpgaPath]/RVfpga/examples/AL_Operations/.pio/build/swervolf_nexys`, e pode abri-lo com o `GTKWave`.

```
gtkwave [RVfpgaPath]/RVfpga/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` fazendo duplo clique nessa aplicação. Na parte superior da aplicação, clique em **File – Open New Tab**, e abra o ficheiro `trace.vcd` gerado na pasta `[RVfpgaPath]/RVfpga/examples/AL_Operations/.pio/build/swervolf_nexys`.

ANÁLISE DO TRACE DA SIMULAÇÃO NO GTKWAVE:

- Agora irá adicionar sinais de relógio, instrução e registo. No painel superior esquerdo do `GTKWave`, expanda a hierarquia do SoC para que você possa adicionar sinais ao gráfico. Expanda a hierarquia em **TOP** → **rvfpgasim** → **swervolf** → **swerv_eh1** → **swerv**, e clique no módulo **ifu** (será realçado como mostra a Figura 84), selecione o sinal `clk` (que é o relógio utilizado para o núcleo) e arraste-o para o painel branco Signals (Sinais) ou para o painel preto Waves (Ondas) à direita.

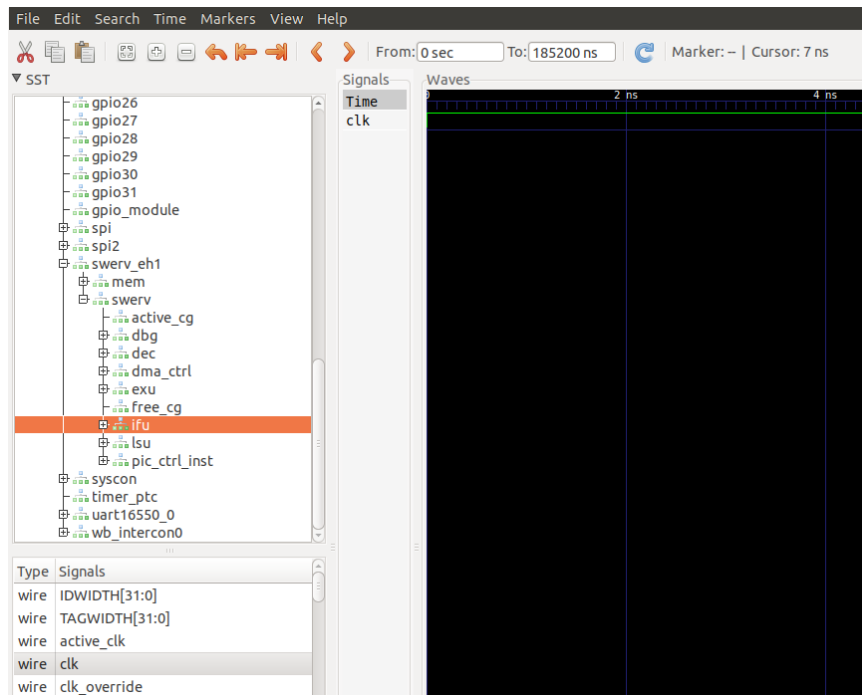


Figura 84. Adicionar o sinal clk ao gráfico

9. Ampliar várias vezes para poder ver a alteração do sinal do relógio (Figura 85).

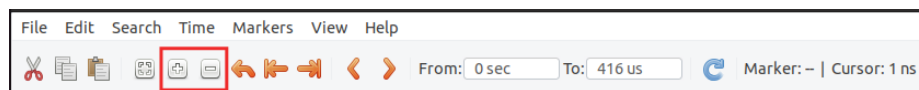


Figura 85. Botões de ampliar e reduzir (zoom)

10. 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 86), e arraste-os para o painel preto Waves. O prefixo *ifu* indica a unidade de carregamento de instruções (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.

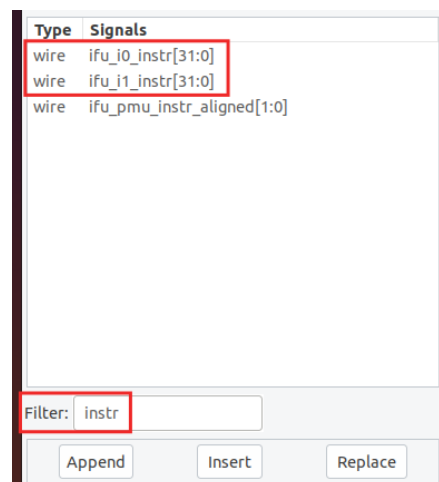


Figura 86. Adicionar os sinais *ifu_i0_instr[31:0]* e *ifu_i1_instr[31:0]* diagrama de forma de onda temporal

11. Agora adicione o sinal que contém o valor do registo `t3` (i.e., registo número 28, `x28`). Expandir a hierarquia em **swerv** para **dec** → **arf** → **gpr_banks(0)** → **gpr(28)** e clicar no módulo **gprff** (será realçado como mostra a figura seguinte), selecione o sinal `dout[31:0]` (que mostra o conteúdo do registo `x28`, usado no exemplo *AL_Operations.S*) e arraste-o para o painel preto Waves (Figura 87).

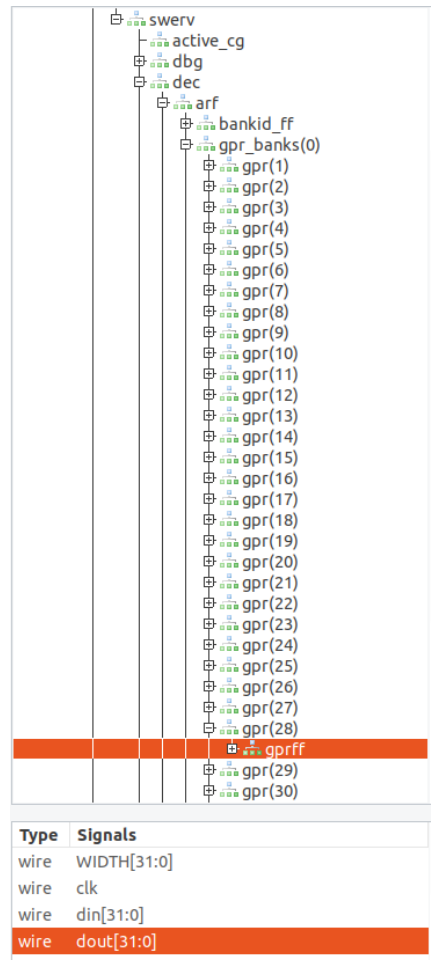


Figura 87. Adicionar o sinal `dout[31:0]` ao gráfico

12. Outra forma de mostrar sinais no GTKWave é usar um ficheiro `.tcl`. O ficheiro `test.tcl` está disponível em `[RVfpgaPath]/RVfpga/examples/AL_Operations`. 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_ehl.swerv.ifu.ifu_i0_instr
gtkwave::addSignalsFromList rvfpgasim.swervolf.swerv_ehl.swerv.ifu.ifu_i1_instr
gtkwave::addSignalsFromList rvfpgasim.swervolf.swerv_ehl.swerv.dec.arf.gpr_banks(0).gpr(28).gprff.dout
```

Para utilizar o ficheiro `.tcl` no GTKWave, pode simplesmente clicar em *File – Read Tcl Script File* e seleccionar o ficheiro `[RVfpgaPath]/RVfpga/examples/AL_Operations/test.tcl`.

13. A Figura 88 mostra o programa *AL_Operations.S* com as suas instruções e o código máquina equivalente.

# RISC-V assembly	# comment (t3 = x28)	# machine code
<code>li t3, 0x0</code>	<code># t3 = 0</code>	<code># 0x00000E13</code>

```

REPEAT:
    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  # Repeat the loop       # 0xFE000CE3
    nop                     # nop                   # 0x00000013
  
```

Figura 88. AL_Operations.S com código de máquina equivalente

Agora veja os sinais mudarem à medida que o programa é executado. É de esperar que as instruções e `t3` (registo `x28`) para tomarem os valores apresentados na Figura 89 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 89. Fluxo de instruções e valores do registo `t3` (`x28`) durante a execução de AL_Operations

14. Amplie em torno de 10100 ps, onde irá analisar a execução das três instruções aritmético-lógicas das primeira e segunda iterações do ciclo (Figura 90). As primeiras duas instruções (`li t3, 0x0 = 0x00000E13` e `addi t3, t3, 6 = 0x006E0E13`) são obtidas em primeiro lugar, uma em cada via do processador RISC-V superescalar, como indicado nos sinais `ifu_i0_instr[31:0]` e `ifu_i1_instr[31:0]`. As duas instruções seguintes (`addi t3, t3, -1 = 0xFFFFE0E13` e `andi t3, t3, 3 = 0x003E7E13`) são obtidas no ciclo seguinte. As duas últimas instruções são obtidas (`beq zero, zero, REPEAT = 0xFE000CE3` e `nop = 0x00000013`) no ciclo seguinte.

Devido ao núcleo do processador SweRV ter um pipeline de 9 fases e às dependências, os efeitos das instruções são vistos oito ou mais ciclos depois de as instruções serem obtidas. Oito ciclos após a primeira e a segunda instruções serem obtidas, `x28` (`t3`) torna-se 0 (que já era) devido à primeira instrução: `li t3, 0x0` (`0x00000E13`). Um ciclo depois, `x28` é atualizado para 0x6 devido à instrução seguinte: `addi t3, t3, 6` (`0x006E0E13`). Depois, `x28` atualiza para 5 devido à instrução seguinte: `addi t3, t3, -1` (`0xFFFFE0E13`). Finalmente, `x28` atualiza para 1 devido à instrução seguinte: `andi t3, t3, 3` (`0x003E7E13`). De seguida, são obtidas as duas instruções seguintes: `beq zero, zero, REPEAT` (`0xFE000CE3`) e `nop` (`0x00000013`), o salto (branch) é tomado e o ciclo repete-se. Tal como previsto na Figura 89. Utilizando um raciocínio semelhante, é possível analisar a segunda iteração, que também é destacada em Figura 90 e prevista na Figura 89.

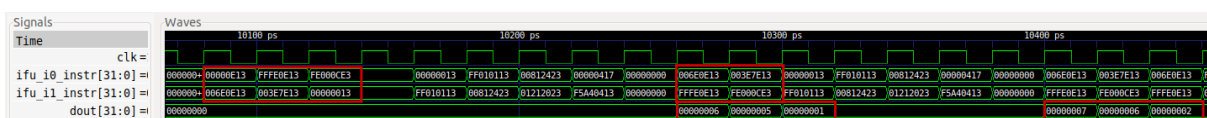


Figura 90. Execução das três instruções Aritmética-Lógica do exemplo

8. SIMULAÇÃO COM O WHISPER

O Whisper (<https://github.com/chipsalliance/SweRV-ISS>) é um simulador de conjunto de instruções "Instruction Set Simulator" (ISS) RISC-V desenvolvido pela Western Digital para a verificação do microcontrolador SweRV. Permite ao utilizador executar código RISC-V sem necessitar de hardware RISC-V subjacente. Usando o Whisper, é possível testar, executar e depurar programas C ou Assembly usando PlatformIO sem precisar da placa FPGA Nexys A7.

Windows: Todas as instruções descritas nesta secção devem funcionar para o Windows (gostaríamos de agradecer a Jean-François Monestier, que foi o primeiro a portar o Whisper para o Windows: <https://jean-francois.monestier.me/porting-western-digital-swerv-iss-to-windows/>). Note que uma janela pop-up pode pedir-lhe para permitir que o Whisper passe pela firewall do Windows.

macOS: Todas as instruções descritas nesta secção também funcionam para o macOS.

O Whisper pode ser executado com a linha de comandos ou com um IDE (ambiente de desenvolvimento integrado) como o Eclipse ou o PlatformIO. Nesta secção demonstramos um exemplo para mostrar como simular um programa com Whisper em PlatformIO. É possível usar os mesmos passos descritos aqui para simular outros programas.

Começamos por utilizar o Whisper ISS para simular AL_Operations, o primeiro programa Assembly simples que executou e depurou na Secção 6 (Figura 44). Siga os próximos passos para executar e depurar este código no Whisper:

1. Abra o VSCode (e o PlatformIO). Na barra de menu superior, clique em *File* → *Open Folder* e navegue até à pasta `[RVfpgaPath]/RVfpga/examples/`, Selecione (mas não abra) a pasta `AL_Operations` e, em seguida, clique em OK.
2. Clique em *File* → *Open File* e faça duplo clique em `[RVfpgaPath]/RVfpga/examples/AL_Operations/platformio.ini`, e defina **whisper** como ferramenta de depuração, descomentando a linha 17 (Figura 91). Guarde o ficheiro (prima Ctrl-s).

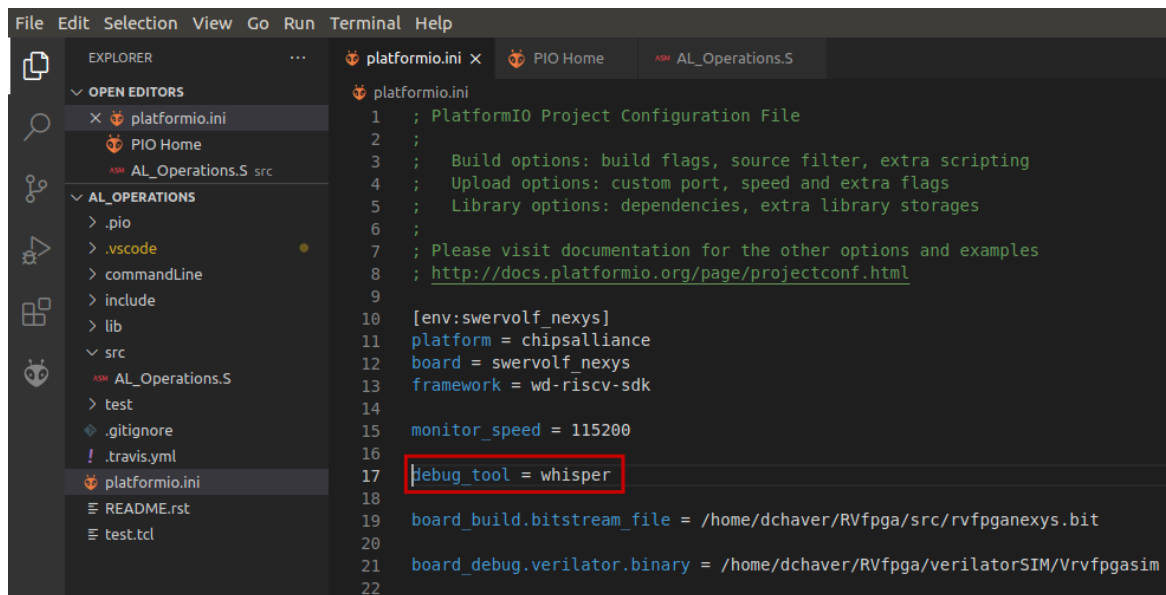

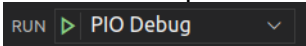
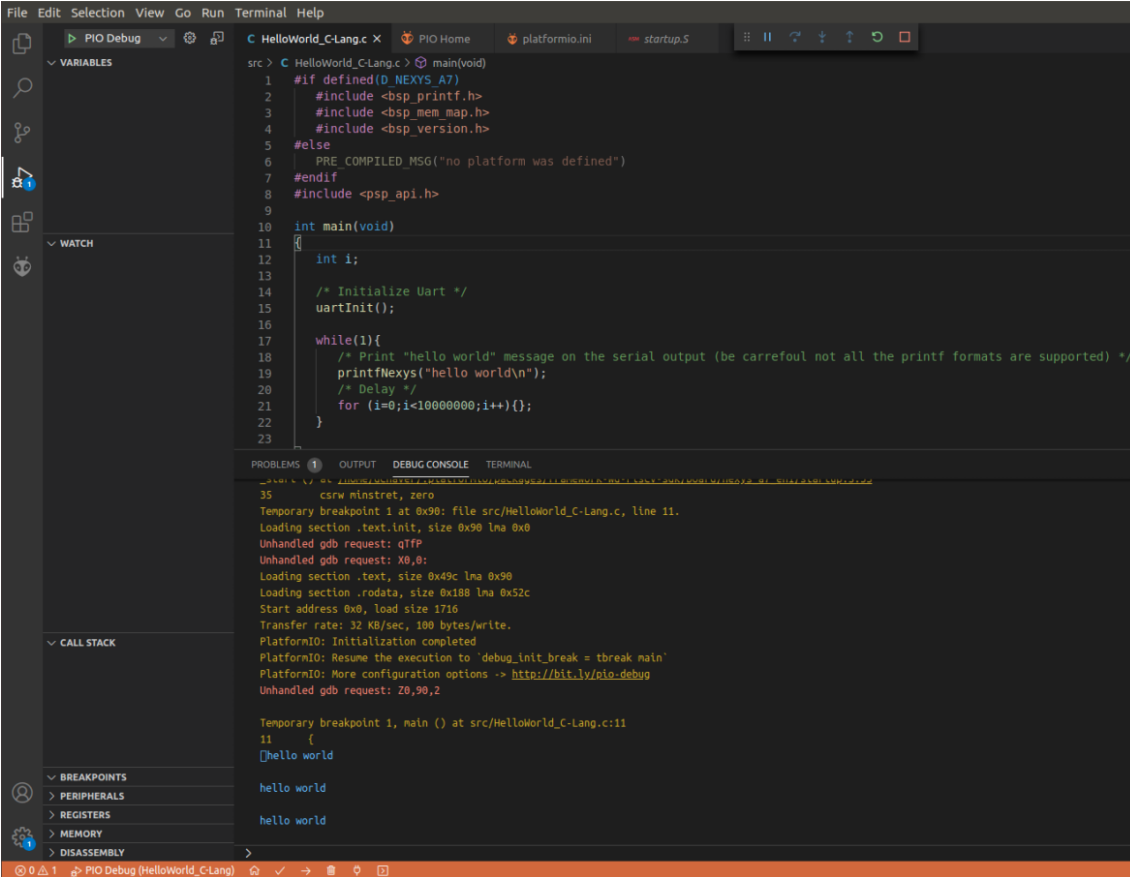


Figura 91. Descomentar a linha 17.

3. 3. Inicie o depurador como habitualmente, clicando em  e depois em 
4. Pode agora depurar o programa exatamente como fez na Secção 6.B, mas desta vez o programa está a ser executado em simulação no Whisper em vez de na placa FPGA Nexys A7.
5. Se um programa usar a função *printfNexys* no Whisper, como o exemplo HelloWorld_C-Lang (Secção 6.F), não deve abrir o monitor série PlatformIO, pois as mensagens são mostradas na consola DEBUG (Figura 92).



```

src > C HelloWorld_C-Lang.c > main(void)
1  #if defined(D_NEXYS_A7)
2  #include <bsp_printf.h>
3  #include <bsp_mem_map.h>
4  #include <bsp_version.h>
5  #else
6  #define PRE_COMPILED_MSG("no platform was defined")
7  #endif
8  #include <csp_api.h>
9
10 int main(void)
11 {
12     int i;
13
14     /* Initialize Uart */
15     uartInit();
16
17     while(1){
18         /* Print "hello world" message on the serial output (be careful not all the printf formats are supported) */
19         printfNexys("hello world\n");
20         /* Delay */
21         for (i=0; i<100000000; i++){
22             ;
23         }
24     }
25 }

```

35 csrw minstret, zero
 Temporary breakpoint 1 at 0x90: file src/HelloWorld_C-Lang.c, line 11.
 Loading section .text.init, size 0x90 lma 0x0
 Unhandled gdb request: q7FF
 Unhandled gdb request: X0:0
 Loading section .text, size 0x49c lma 0x90
 Loading section .rodata, size 0x188 lma 0x52c
 Start address 0x0, load size 1716
 Transfer rate: 32 KB/sec, 100 bytes/write.
 PlatformIO: Initialization completed
 PlatformIO: Resume the execution to 'debug_init_break = tbreak main'
 PlatformIO: More configuration options -> <http://bit.ly/pio-debug>
 Unhandled gdb request: 20,90,2

 Temporary breakpoint 1, main () at src/HelloWorld_C-Lang.c:11
 11 {
 [h]ello world

 hello world

 hello world

Figura 92. Execução do exemplo HelloWorld_C-Lang no Whisper

9. APÊNDICES

Os apêndices a seguir mostram como usar o conjunto de ferramentas nativas do RISC-V e o OpenOCD (em vez do PlatformIO) no Linux, como instalar no Windows os drivers para baixar o bitstream usando o PlatformIO, como instalar o Verilator e o GTKWave em máquinas Windows e Mac OS e como programar o RVfpgaNexys usando o Vivado. A Tabela 10 lista todos os apêndices disponíveis neste Guia de Iniciação ao RVfpga.

Tabela 10. Lista de Apêndices

Apêndice	Descrição	Sistema Operativo
A	Usando a cadeia de ferramentas nativas RISC-V e OpenOCD para RVfpga no Ubuntu 18.04	Linux
B	Instalar <i>drivers</i> no Windows para utilizar PlatformIO	Windows
C	Instalando o Verilator e o GTKWave no Windows	Windows
D	Instalando o Verilator e o GTKWave no macOS	macOS
E	Usando o Vivado para baixar o RVfpgaNexys em um FPGA	Windows e Linux
F	Utilização de RVfpga numa aplicação IoT industrial	All

O Apêndice A deve ser usado por aqueles que querem compilar e executar/depurar programas nativamente usando as ferramentas nativas gcc/gdb e OpenOCD. No entanto, **recomenda-se que os utilizadores do RVfpga utilizem antes o PlatformIO**, como descrito neste Guia de Iniciação.

Os utilizadores de [Windows](#) devem seguir as instruções dos Apêndices B e C. As instruções no Apêndice B mostram como instalar drivers para que os sistemas Windows possam usar o PlatformIO para descarregar programas e configurar o RVfpgaNexys para a placa FPGA Nexys A7. O Apêndice C mostra como instalar o Verilator e o GTKWave para que os utilizadores do Windows possam simular o RVfpgaSim.

Os utilizadores de [macOS](#) devem seguir as instruções do Apêndice D para simular o RVfpgaSim usando o Verilator e o GTKWave.

Recomenda-se a utilização do PlatformIO para configurar o sistema RVfpgaNexys (tal como definido pelo ficheiro *bitfile*, *rvfpganexys.bit*) para a Nexys A7 FPGA board. Este *bitfile* (*rvfpganexys.bit*) pode ser gerado pelo Vivado ou PlatformIO. Também é possível usar o Vivado para configurar o sistema RVfpgaNexys na placa FPGA Nexys A7, conforme descrito no Apêndice E. No entanto, **não é recomendado** usar o Vivado para configurar o RVfpgaNexys na placa – especialmente para utilizadores de [Windows](#), uma vez que exigiria a troca constante de controladores.

Apêndice A: Usando a cadeia de ferramentas nativas RISC-V e o OpenOCD em Ubuntu 18.04

Embora recomendemos o uso do PlatformIO, nesta secção mostramos como instalar, executar e usar a toolchain nativa do RISC-V e usar o OpenOCD para configurar o RVfpgaNexys na placa FPGA Nexys A7 e o gdb para executar e depurar programas no RVfpgaNexys. O conjunto de ferramentas é composto por um compilador gnu, um depurador, um assembler, etc. Mostramos como instalar a toolchain RISC-V e o OpenOCD num sistema operativo (SO) Ubuntu 18.04, mas este processo também deve funcionar para outras distribuições Linux. Estas instruções assumem um sistema Ubuntu fresco.

Os passos a seguir não são necessários se usar o PlatformIO, conforme descrito anteriormente neste guia. Usar PlatformIO, Vivado e Verilator ou Whisper é o método recomendado para executar, depurar e simular programas RISC-V, mas as instruções a seguir são fornecidas para quem estiver interessado em usar a toolchain RISC-V nativa e o OpenOCD no lugar da PlatformIO e do Vivado Hardware Manager.

I. Instalação nativa num Sistema Operativo Linux Ubuntu

Nesta secção descrevemos como instalar nativamente na sua máquina Ubuntu 18.04 a toolchain RISC-V, OpenOCD e Whisper. Essas ferramentas apenas substituem o PlatformIO; a instalação do Vivado e do Verilator ainda é necessária, conforme explicado na Seção 5 deste GSG.

Toolchain RISC-V

Aqui mostramos como instalar a toolchain RISC-V completa - ou seja, compilador gnu, depurador, etc. - no seu computador. As instruções de instalação são fornecidas pela RISC-V International em: <https://github.com/riscv/riscv-gnu-toolchain>. Estas instruções estão resumidas a seguir.

NOTA: A instalação da toolchain RISC-V e do OpenOCD pode levar várias horas - principalmente enquanto a cadeia de ferramentas faz o download, compila e instala.

Num terminal, escreva o seguinte (o processo pode demorar mais de uma hora, mas a maior parte do tempo é passado à espera enquanto os programas são descarregados e instalados):

- `sudo apt-get install git autoconf automake autotools-dev curl libmpc-dev libmpfr-dev libgmp-dev gawk build-essential bison flex texinfo gperf libtool patchutils bc zlib1g-dev libexpat-dev`
- `git clone --recursive https://github.com/riscv/riscv-gnu-toolchain`
- `cd riscv-gnu-toolchain/`
- `./configure --prefix=/opt/riscv --with-arch=rv32imc`
- `sudo make` (Se possível, usar `sudo make -j$(nproc)` uma vez que diminui significativamente o tempo de compilação)
- `export PATH=$PATH:/opt/riscv/bin` (altere o caminho no seu sistema)

OpenOCD

O OpenOCD é um depurador on-chip aberto que permite aos utilizadores programar e depurar dispositivos embebidos. Siga os próximos passos para instalar o RISC-V OpenOCD no seu computador:

- `sudo apt-get install libusb-1.*`
- `sudo apt-get install pkg-config`

- `git clone https://github.com/riscv/riscv-openocd.git`
- `cd riscv-openocd/`
- `./bootstrap`
- `./configure --prefix=/opt/riscv --program-prefix=riscv- --enable-ftdi --enable-jtag_vpi`
- `make`
- `sudo make install`

Whisper

Siga os passos seguintes para instalar a Whisper no seu computador (as instruções estão disponíveis em: <https://github.com/chipsalliance/SweRV-ISS> mas também estão resumidas abaixo):

- `apt-cache policy libboost-all-dev`
- `sudo apt-get install libboost-all-dev`
- `cd [RVfpgaPath]`
- `git clone https://github.com/chipsalliance/SweRV-ISS`
- `cd SweRV-ISS`
- `make BOOST_DIR=/usr/include/boost`
- `export PATH=$PATH:[RVfpgaPath]/SweRV-ISS/build-Linux (substituir [RVfpgaPath] conforme necessário).`

II. Execução de um programa no RVfpgaNexys com a placa FPGA Nexys A7 utilizando o OpenOCD

Passo A. Configurar o RVfpgaNexys (Figura 25) na Nexys A7

1. Vá para a pasta do projeto que contém o ficheiro *bitfile* para o RVfpgaNexys:
`cd [RVfpgaPath]/RVfpga/src`
2. Configurar o RVfpgaNexys para a placa utilizando o OpenOCD:
`riscv-openocd -c "set BITFILE rvfpganexys.bit" -f OtherSources/ConfigFiles/swervolf_nexys_program.cfg`

Passo B. Corra LedsSwitches, o programa que lê os Switches e exibe o seu estado nos LEDs

3. Vá para a pasta LedsSwitches/commandLine:
`cd [RVfpgaPath]/RVfpga/examples/LedsSwitches/commandLine`

Nessa pasta encontrará o Makefile para compilar o código-fonte, o *linker script*, um script python, e o programa *LedsSwitches.S*.

4. Crie o ficheiro .elf:
`make clean`
`make LedsSwitches.elf`
5. Ligue o OpenOCD ao SoC:
`riscv-openocd -f ../../../../src/OtherSources/ConfigFiles/swervolf_nexys_debug.cfg`

Quando o OpenOCD começar a ser executado, verá várias mensagens, incluindo uma que diz:

Info : Listening on port 4444 for telnet connections

6. Abra um novo terminal e vá para a pasta do programa (`cd [RVfpgaPath]/RVfpga/examples/LedsSwitches/commandLine`) e execute o seguinte comando:

```
telnet localhost 4444
```

Em seguida, dentro da ligação telnet, escreva:

```
load_image LedsSwitches.elf
reg pc 0
resume
```

Estes três comandos (1) carregam o programa `LedsSwitches.elf` no `RVfpgaNexys`, (2) colocam o contador de programa (PC) em 0 (a localização do endereço da primeira instrução do programa) e (3) retomam a execução.

O programa começará a ser executado no `RVfpgaNexys`, o SoC RISC-V `SweRVolfX` que já foi configurado na placa FPGA Nexys A7 no Passo 2. O programa faz com que os LEDs mostrem o estado dos interruptores. Ao alternar os interruptores, os LEDs devem mudar imediatamente para refletir o valor dos interruptores.

Passo C. Depurar o programa AL_Operations_CommandLine que executa operações aritméticas-lógicas simples

Agora vamos mostrar como depurar outro programa (`AL_Operations_CommandLine`) usando o `OpenOCD` e o `gdb`.

7. Mantenha a ligação `OpenOCD` aberta (Passo 5).
8. No outro terminal onde a telnet está a correr (Passo 6), saia da ligação telnet escrevendo:

```
exit
```

9. Mude para a pasta do projeto que contém `AL_Operations/commandLine`:

```
cd ../../AL_Operations/commandLine
```

Nessa pasta, encontrará o `Makefile` para compilar as fontes, o *linker script*, um script python e o programa `AL_Operations.S`.

10. Crie o ficheiro `.elf`:

```
make clean
make AL_Operations.elf
```
11. De seguida, neste terminal, inicie o `gdb` escrevendo:

```
riscv32-unknown-elf-gdb AL_Operations.elf
```

12. Na consola `gdb`, digite:

```
target remote localhost:3333
load
```

Isto liga ao `OpenOCD` e carrega o programa `AL_Operations.elf` para a memória.

13. Agora deve ser capaz de depurar o programa. Escreva a seguinte sequência e analise os resultados:

i. `disas 0,20`

Isto mostra o código Assembly do endereço 0 ao 20 (não incluindo o endereço 20). Ver Figura 93.

```
(gdb) disas 0,20
Dump of assembler code from 0x0 to 0x14:
=> 0x00000000 <_start+0>:      li      t3,0
    0x00000004 <REPEAT+0>:      addi     t3,t3,6
    0x00000008 <REPEAT+4>:      addi     t3,t3,-1
    0x0000000c <REPEAT+8>:      andi     t3,t3,3
    0x00000010 <REPEAT+12>:     beqz     zero,0x4 <REPEAT>
End of assembler dump.
```

Figura 93. Visualização do programa Assembly

ii. `i r t3`

Isto mostra o conteúdo do registo `t3`. Em alternativa, pode escrever a versão mais longa: `info reg t3`. Ver Figura 94.

```
(gdb) i r t3
t3          0x0      0
```

Figura 94. Visualização do valor contido no registo `t3`

iii. `i r pc`

Isto mostra o conteúdo do *program counter* (`pc`). Ver Figura 95.

```
(gdb) i r pc
pc          0x0      0x0 <_start>
```

Figura 95. Mostrar o valor contido no registo PC, que aponta para a primeira instrução

iv. `stepi`
`i r t3`
`stepi`
`i r t3`
`stepi`
`i r t3`
`stepi`
`i r t3`

`stepi` faz com que o programa execute uma instrução. `i r t3` e exibe o conteúdo do registo `t3`. Ver Figura 96.

```
(gdb) stepi
0x00000004 in REPEAT ()
(gdb) i r t3
t3                0x0      0
(gdb) stepi
0x00000008 in REPEAT ()
(gdb) i r t3
t3                0x6      6
(gdb) stepi
0x0000000c in REPEAT ()
(gdb) i r t3
t3                0x5      5
(gdb) stepi
0x00000010 in REPEAT ()
(gdb) i r t3
t3                0x1      1
```

Figura 96. Executar várias instruções, uma a uma, e visualizar o registo $t3$

Quando tiver terminado de depurar e explorar o programa e os registos usando o gdb, saia do gdb escrevendo **quit** no terminal do gdb e saia do OpenOCD escrevendo **^C** no terminal do OpenOCD.

III. Simulação de um programa no RVfpgaSim utilizando o Verilator

1. Abrir um terminal no Ubuntu
2. Numa janela de terminal, gerar o binário do simulador executando os seguintes comandos:

```
cd [RVfpgaPath]/RVfpga/verilatorSIM
make clean
make
```

O ficheiro *Vrvfpgasim* (o ficheiro RVfpgaSim binário da simulação), deve ser gerado dentro da pasta *[RVfpgaPath]/RVfpga/verilatorSIM*.

3. Aceder à pasta que contém o programa de exemplo:

```
cd [RVfpgaPath]/RVfpga/examples/AL_Operations/commandLine
```
4. Criar o programa para simulação.

```
make clean
make AL_Operations.elf
make AL_Operations.bin
make AL_Operations.vh
```
5. Executar o simulador.

```
../../../../verilatorSIM/Vrvfpgasim
+ram_init_file=AL_Operations.vh +vcd=1
```

Após alguns segundos, pare a simulação premindo **^C** no terminal. Deve ter sido gerado o ficheiro *trace.vcd*, que pode ser aberto com *GTKWave*.

```
gtkwave trace.vcd
```

6. Siga as instruções fornecidas nos passos 8 a 12 da secção 7 para adicionar sinais ao gráfico e analisá-los.

IV. Simulação de um programa com o Whisper

1. Abra um terminal no Ubuntu
2. Aceda à pasta que contém o programa de exemplo:

```
cd [RVfpgaPath]/RVfpga/examples/AL_Operations/commandLine
```
3. Criar o programa Disassembly.

```
make AL_Operations.dis
```
4. Abrir o ficheiro *AL_Operations.dis* num editor. Isto é o que deve ser visto:

```
<_start>:
    0: 00000e13          li      t3,0
<REPEAT>:
    4: 006e0e13          addi    t3,t3,6
    8: fffe0e13          addi    t3,t3,-1
   c: 003e7e13          andi    t3,t3,3
  10: fe000ae3          beqz    zero,4 <REPEAT>
  14: 00000013          nop
```

5. Execute o simulador em modo interativo.

```
whisper --interactive AL_Operations.elf
```
6. Depure o programa.

```
whisper> step
#1 0 00000000 00000e13 r 1c          00000000  addi      x28, x0, 0x0

whisper> peek r x28
0x00000000

whisper> step
#2 0 00000004 006e0e13 r 1c          00000006  addi      x28, x28, 0x6

whisper> peek r x28
0x00000006

whisper> step
#3 0 00000008 fffe0e13 r 1c          00000005  addi      x28, x28, -0x1

whisper> peek r x28
0x00000005

whisper> step
#4 0 0000000c 003e7e13 r 1c          00000001  andi      x28, x28, 0x3

whisper> peek r x28
0x00000001
```

Quando tiver terminado de depurar e explorar o programa e os registos utilizando o Whisper, saia escrevendo **quit** no terminal.

Apêndice B: Instalar drivers no Windows para usar PlatformIO

Para descarregar o executável Zadig, navegue até ao seguinte sítio Web (Figura 97):

<https://zadig.akeo.ie/>

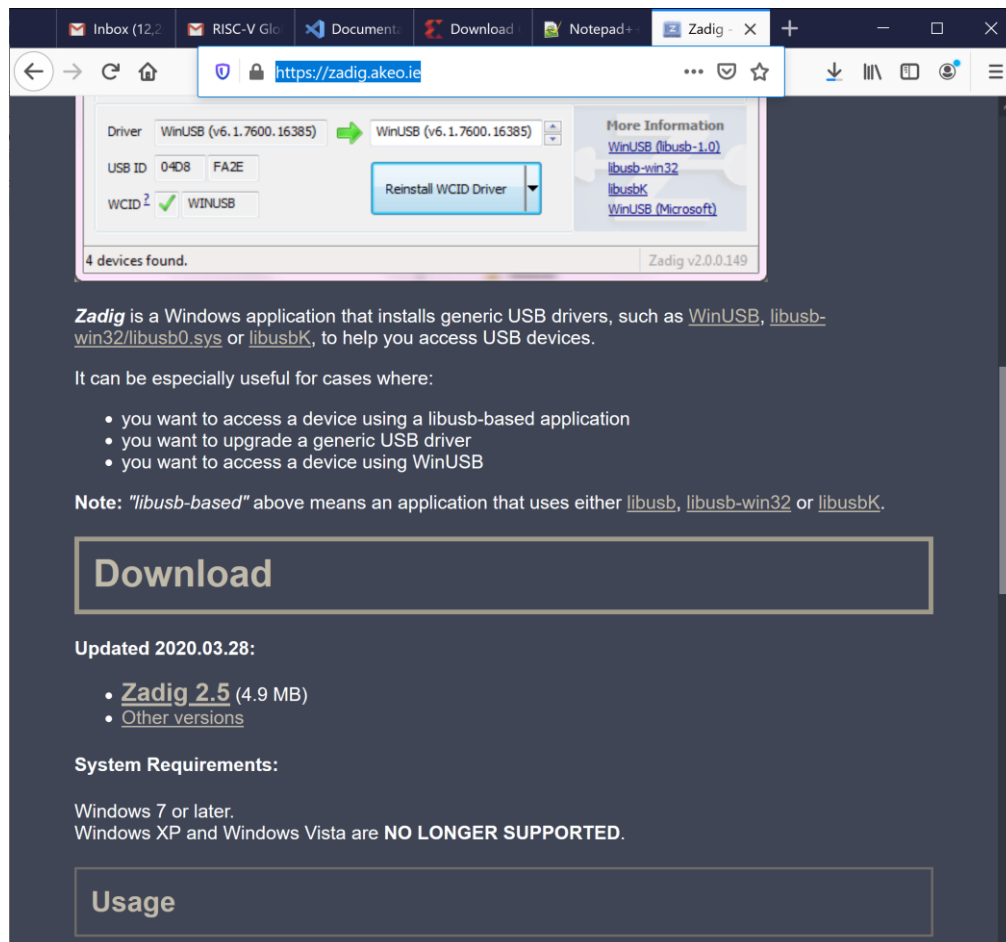


Figura 97. Instalar o controlador da placa Nexys A7 utilizado pelo PlatformIO

Clique em Zadig 2.5 e guarde o executável. Em seguida, execute-o (zadig-2.5.exe), o qual está localizado onde o descarregou. Também pode escrever zadig no menu Iniciar para o encontrar. Provavelmente, ser-lhe-á perguntado se pretende permitir que o Zadig efetue alterações no seu computador e se pretende que este verifique se existem actualizações. Clique em Sim nas duas vezes.

Ligue a placa Nexys A7 ao seu computador e ligue-a. No Zadig, clique em Options → List All Devices (Figura 98).

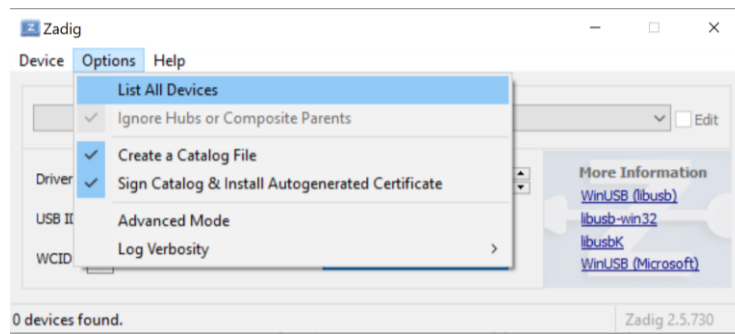


Figura 98. Listar todos os dispositivos no Zadig

Se clicar no menu pendente, verá listados o Dispositivo USB da Digilent (Interface 0) e o Dispositivo USB da Digilent (Interface 1). Irá instalar novos controladores apenas para o Dispositivo USB da Digilent (Interface 0) (Figura 99).

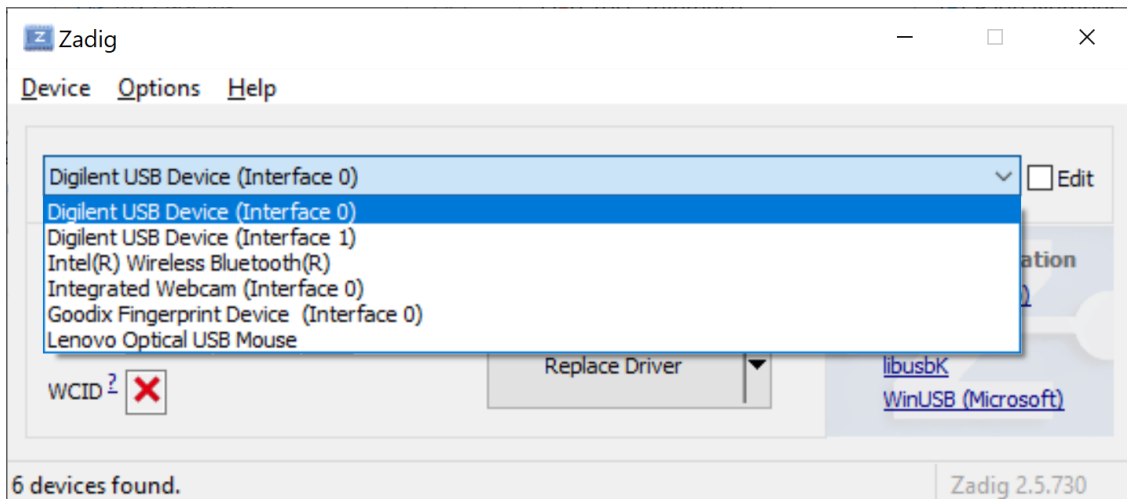


Figura 99. Instalar o controlador WinUSB para o dispositivo USB da Digilent (Interface 0)

Agora, substituirá o controlador FTDI pelo controlador WinUSB, como mostrado na Figura 100. Clique em Replace Driver (ou Install Driver) para o dispositivo USB Digilent (Interface 0). Está a instalar o controlador para a placa Nexys A7 ou, se tiver instalado anteriormente o Vivado, está a substituir o controlador FTDI utilizado pelo Vivado pelo controlador WinUSB utilizado pela PlatformIO.

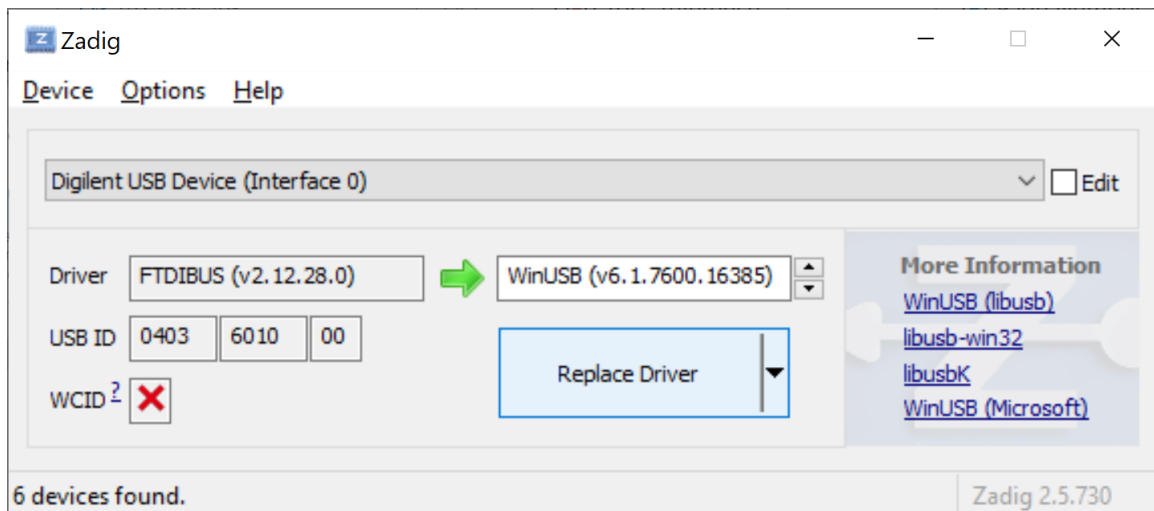


Figura 100. Substituir o driver para a placa Nexys A7

Após algum tempo, normalmente vários minutos, o Zadig indicará que o controlador foi instalado corretamente. Clique em *Close* e, em seguida, feche a janela do Zadig.

Da próxima vez que usar o PlatformIO, não será necessário reinstalar o driver. No entanto, observe que esse driver não é compatível com o Vivado no Windows. Portanto, não é possível usar o Vivado para transferir ficheiros bitfiles para a placa FPGA. Se quiser usar o Vivado para transferir bitfiles (não recomendado), será necessário reverter o driver para o driver original instalado com o Vivado, conforme descrito no Apêndice E.

Apêndice C: Instalando o Verilator e o GTKWave no Windows

Nesta secção, explicamos como instalar o Verilator e o GTKWave no Windows 10. No Windows, é necessário usar o Cygwin para instalar o Verilator, portanto, primeiro explicamos como instalar esse ambiente de programação/execução.

Instalação do Cygwin:

Tal como descrito na sua página Web (<https://www.cygwin.com>), O Cygwin consiste em ferramentas GNU e de código-aberto que fornecem funcionalidade no Windows semelhante à de uma distribuição Linux. Siga os próximos passos para instalar o Cygwin no Windows 10.

1. Navegar para a página Web de instalação (<https://cygwin.com/install.html>) e descarregar o ficheiro de instalação, com o nome `setup-x86_64.exe` (Figura 101).

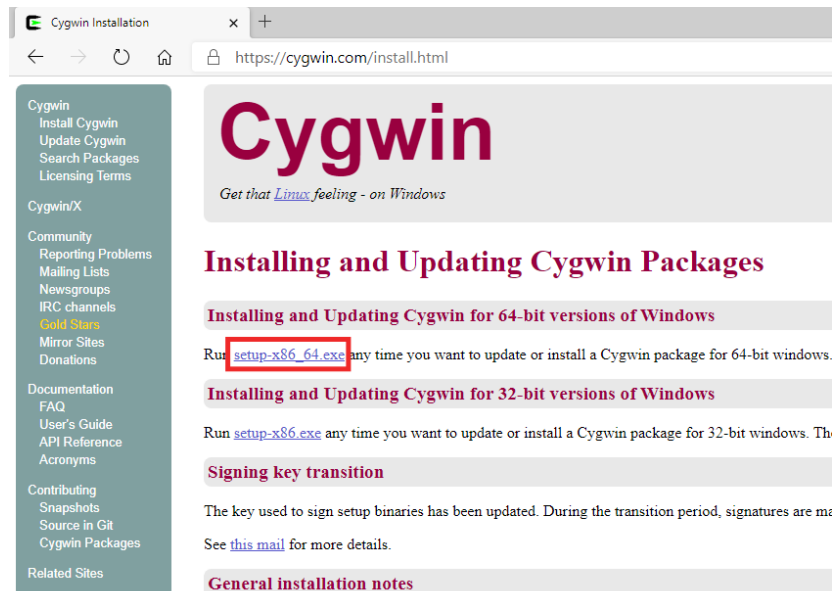


Figura 101. Página web de instalação do Cygwin

2. Execute o ficheiro de instalação na sua máquina fazendo duplo clique sobre ele (Figura 102). Clique em **Next** várias vezes, mantendo as opções predefinidas. O instalador pedir-lhe-á para escolher o local de transferência (**Choose a Download Site**) (Figura 103), pode escolher qualquer um deles.

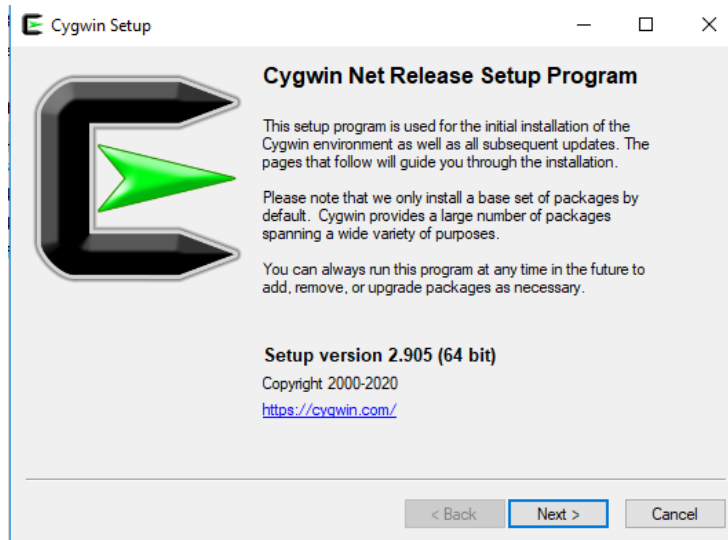


Figura 102. Janela de instalação do Cygwin

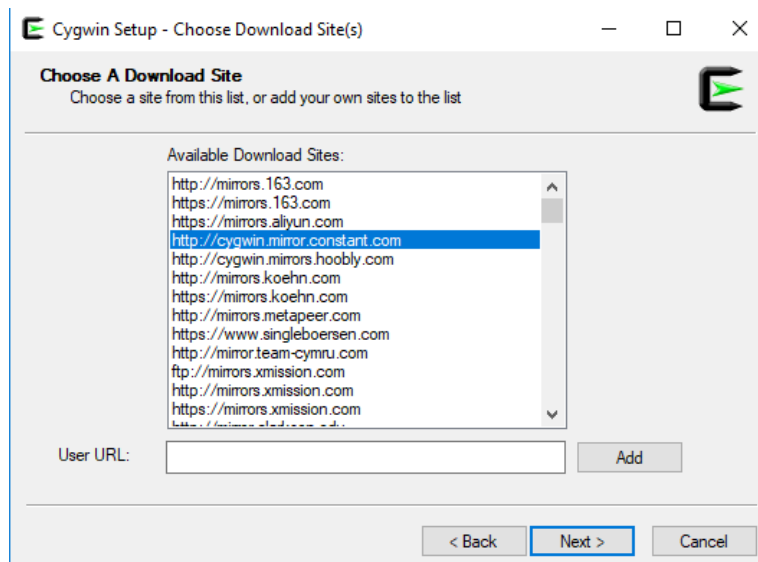


Figura 103. Escolher o local de transferência

3. Após alguns passos, chegará à janela Escolha de Pacotes (**Select Packages**) (Figura 104). Selecionar a vista completa (**Full**), como se pode ver na Figura 104.

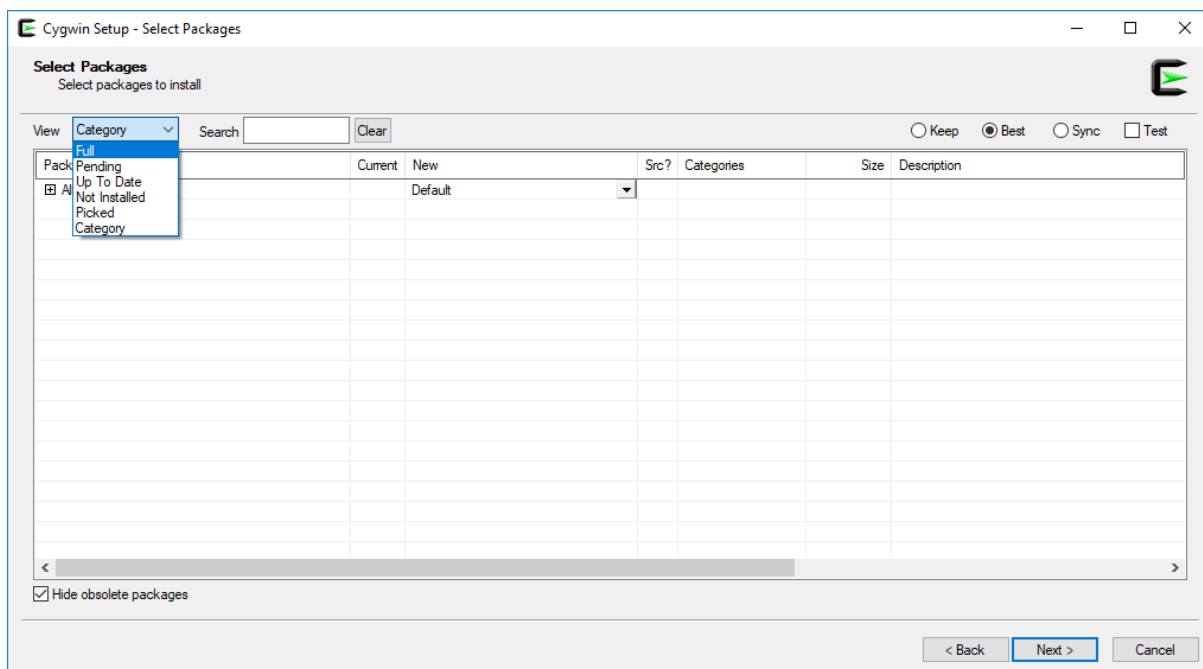


Figura 104. Janela de seleção de pacotes

4. Aparecerá a lista completa de pacotes que pode instalar (Figura 105). Na caixa de pesquisa (**Search**), selecionar os pacotes específicos que pretende instalar.

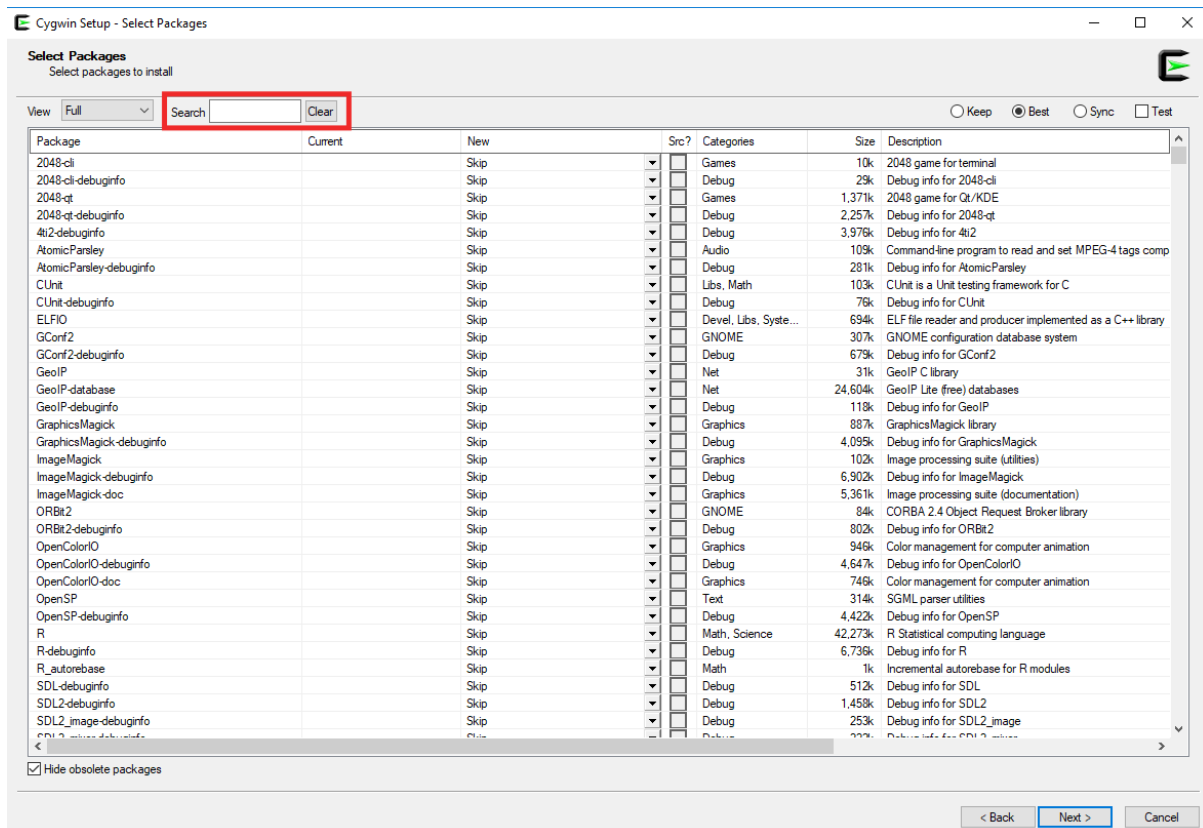


Figura 105. Janela Selecionar pacotes - Vista completa

Para poder compilar o Verilator e gerar um novo binário do simulador, é necessário instalar os seguintes pacotes:

- git
- make
- autoconf
- gcc-core
- gcc-g++
- flex
- bison
- perl
- libargp-devel

Inclua pelo menos estes pacotes na sua instalação do Cygwin. Selecione-os um a um seguindo os passos abaixo (apenas mostramos os passos detalhados para o primeiro pacote da lista, `git`; o processo é idêntico para os outros pacotes):

- Procure o pacote `git` na caixa de pesquisa (**Search**) (Figura 106).

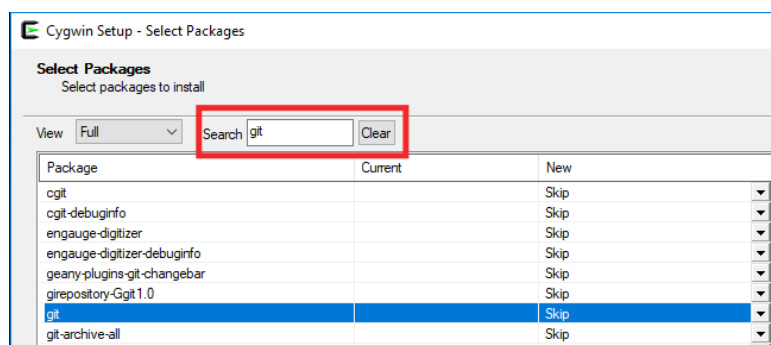


Figura 106. Procura do pacote `git`

- Selecione a versão mais atualizada no menu pendente e marque a caixa (Figura 107).



Figura 107. Selecionar a versão mais atualizada e assinalar a caixa

- Faça o mesmo para os restantes pacotes da lista acima. Na maioria dos casos, pode utilizar a versão mais atualizada, mas para os pacotes `gcc-core` e `gcc-g++` deve utilizar a versão 10.2.0, uma vez que a versão mais atualizada, que no momento da redacção deste documento era a 11.2.0, apresenta alguns conflitos com o Verilator.
5. Depois de ter selecionado os nove pacotes, clique em **Next** nas janelas subsequentes para incluir esses pacotes na sua instalação do Cygwin (o processo de instalação, ver a Figura 108, pode demorar vários minutos) e finalizar a instalação clicando em **Finish** (Figura 109).

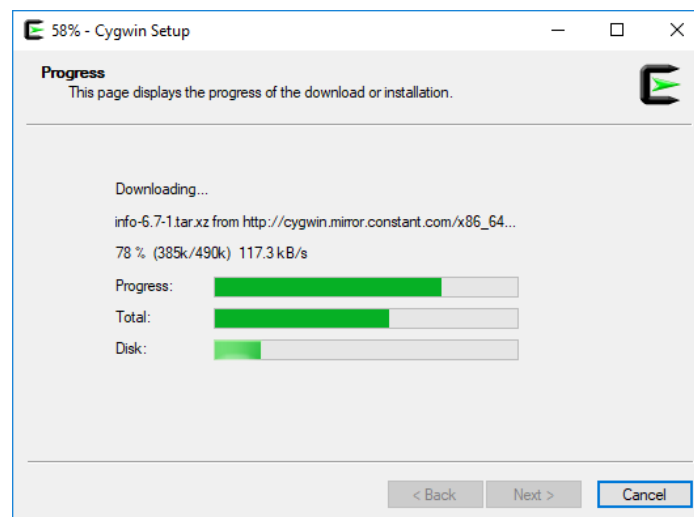


Figura 108. Instalação do Cygwin

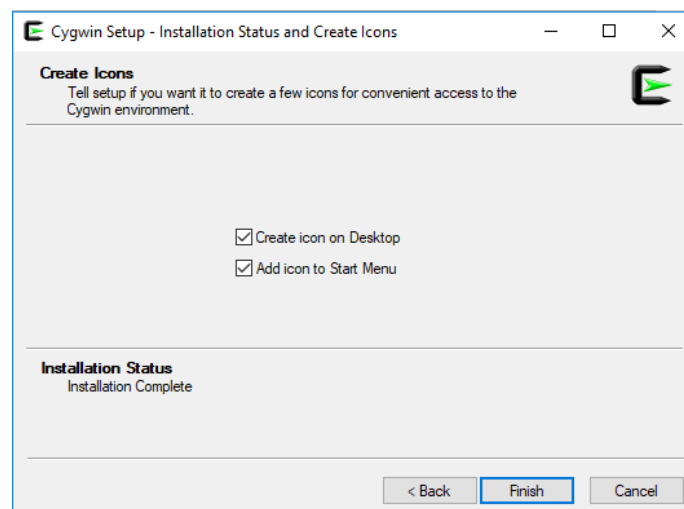


Figura 109. Conclusão da instalação

6. Se precisar de adicionar um pacote à sua instalação do Cygwin, repita os passos 2-5 para esse pacote.

Instalação do Verilator:

Siga os próximos passos para instalar o Verilator no Windows 10.

1. Abra o terminal Cygwin (Figura 110), disponível no ambiente de trabalho do Windows ou no menu Iniciar.

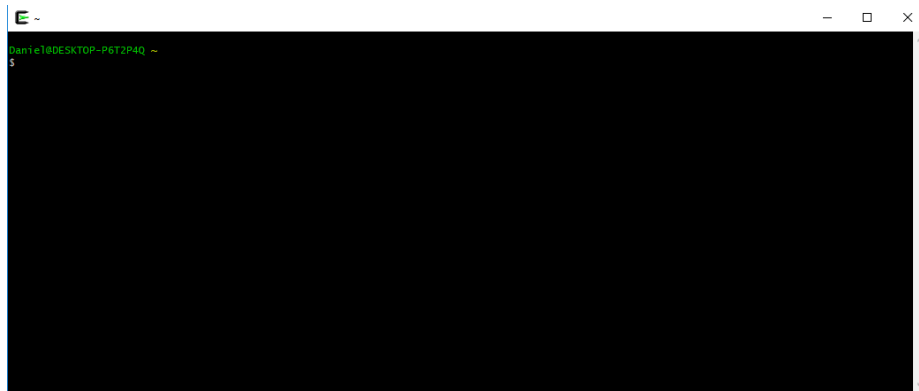


Figura 110. Terminal Cygwin

2. Compile e instale o Verilator seguindo estes passos. Isso pode levar algum tempo (até horas), dependendo da velocidade do seu computador:

- `git clone https://git.veripool.org/git/verilator`
- `cd verilator`
- `git pull`
- `git checkout v4.106`
- `autoconf`
- `./configure`
- `make`
- `make install`

Instalação do GTKWave:

O GTKWave pode ser descarregado como um pacote pré-compilado a partir de <https://sourceforge.net/projects/gtkwave/files/>. Procure o pacote mais recente do Windows (na altura em que este documento foi escrito, era designado por **gtkwave-3.3.100-bin-win64**), e descarregue-o e descomprima-o (uncompress). Pode encontrar um ficheiro executável chamado *gtkwave* dentro da pasta *bin*, que pode executar e utilizar na sua máquina Windows.

Apêndice D: Instalando o Verilator e o GTKWave no macOS

Nesta secção, explicamos como instalar o Verilator e o GTKWave num macOS. As instruções foram testadas com o macOS Catalina 10.15.6, mas espera-se que funcionem noutras versões do SO. O gestor de pacotes Homebrew (<https://brew.sh/>) é utilizado para a instalação. Podem ser encontrados passos semelhantes para o MacPorts, o outro gestor de pacotes amplamente utilizado no macOS (<https://www.macports.org/>).

instalação do gcc:

Para construir um novo simulador usando o Verilator, uma cadeia de ferramentas do compilador precisa ser instalada no sistema. Há muitas maneiras de instalar uma toolchain de compilador válida. Citamos duas delas abaixo:

1. Instale as ferramentas de linha de comando do XCode. Note que isto irá instalar o LLVM, mas um comando gcc estará de qualquer forma disponível após a instalação. Para isso, escreva o seguinte comando numa janela do Terminal:

- `xcode-select -install`

2. Instale o gcc usando o Homebrew. Use o seguinte comando:

- o `brew install gcc@9`

Instalação do Verilator:

Instalar o Verilator com o Homebrew é tão simples quanto escrever o seguinte comando num Terminal:

- `brew install verilator`

instalação do gtkwave:

Mais uma vez, usaremos o Homebrew para instalar o gtkwave. Mas desta vez precisamos de usar o cask porque é uma aplicação GUI do macOS. Escreva os seguintes comandos num Terminal:

- `brew tap homebrew/cask`
- `brew cask install xquartz`
- `brew cask install gtkwave`

Após a instalação, deve aparecer um ícone para *gtkwave.app* na pasta Application. Para o utilizar a partir da linha de comandos, poderá ser necessário instalar o módulo Perl Switch:

- `cpan install Switch`

Apêndice E: Usando o Vivado para configurar o RVfpgaNexys na FPGA

Siga os próximos passos para configurar a FPGA com o RVfpgaNexys usando o Vivado:

WINDOWS: Antes de prosseguir com os próximos passos, no Windows precisa de reverter os *drivers* para os usados pelo Vivado, conforme explicado no final deste Apêndice (Apêndice E).

- a. Ligue a placa Nexys A7 ao seu computador.
- b. Ligue a placa Nexys A7 utilizando o interruptor no canto superior esquerdo.
- c. Abra o Vivado 2019.2.
- d. Abra o *Hardware Manager* disponível no Vivado e destacado na Figura 111.

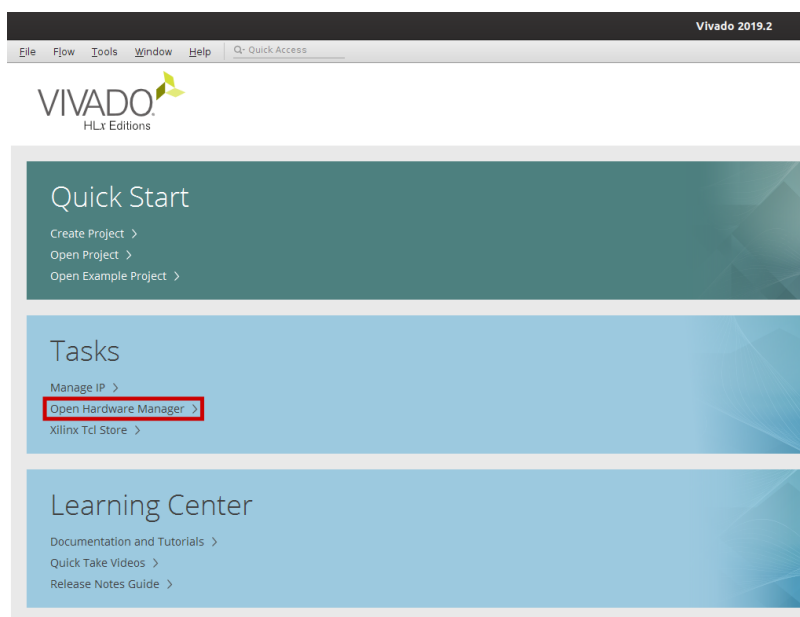


Figura 111. Abrir o *Hardware Manager*

- e. O *Hardware Manager* abre e informa-o de que não existe nenhum dispositivo de hardware aberto. Abra o dispositivo clicando em *Open target – Auto connect* (Figura 112).

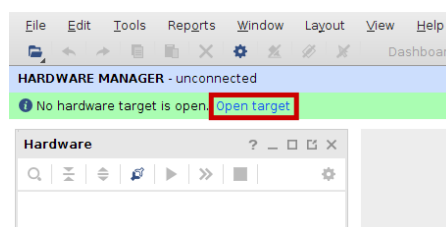


Figura 112. Open target

- f. Selecione *Program device* como mostra a Figura 113. Agora vai carregar o RVfpgaNexys na FPGA. Na nova janela, selecione o ficheiro *Bitstream* de *[RVfpgaPath]/RVfpga/src/rvfpganexys.bit*. Clique em *Program*.

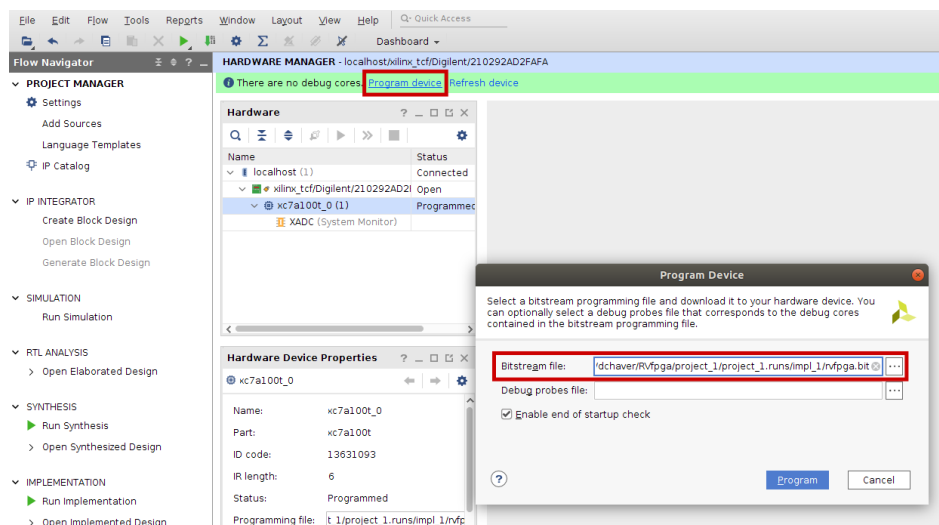


Figura 113. Programar o dispositivo

- g. Após alguns segundos, a FPGA será configurada com o RVfpgaNexys, o **SweRVolfX SoC realizado para FPGA** (Figura 25).
- h. Por fim, **feche o Hardware Manager** clicando no botão X no canto superior direito do painel *Hardware Manager* no Vivado (Figura 114), para que o Vivado liberte a placa.

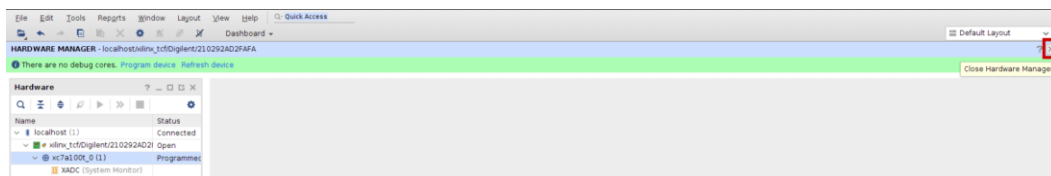


Figura 114. Feche o Hardware Manager

Como reverter os drivers para os usados pelo Vivado no Windows

Infelizmente, no Windows, os drivers para a placa Nexys A7 FPGA são diferentes para o Vivado e o PlatformIO. **Recomenda-se vivamente a utilização da PlatformIO para programar a FPGA, conforme explicado na Secção 5.A deste Guia.** No entanto, se quiser usar o Vivado para programar bitfiles, é preciso reverter os drivers instalados no Apêndice B para os drivers do Vivado (FTDI) para a placa Nexys A7 FPGA. Para isso, abra o Gestor de Dispositivos (*Device Manager*) clicando no menu Iniciar, escrevendo “gestor de dispositivos” ou “device manager” na caixa de pesquisa e clicando em Gestor de Dispositivos ou *Device Manager* (Figura 115).

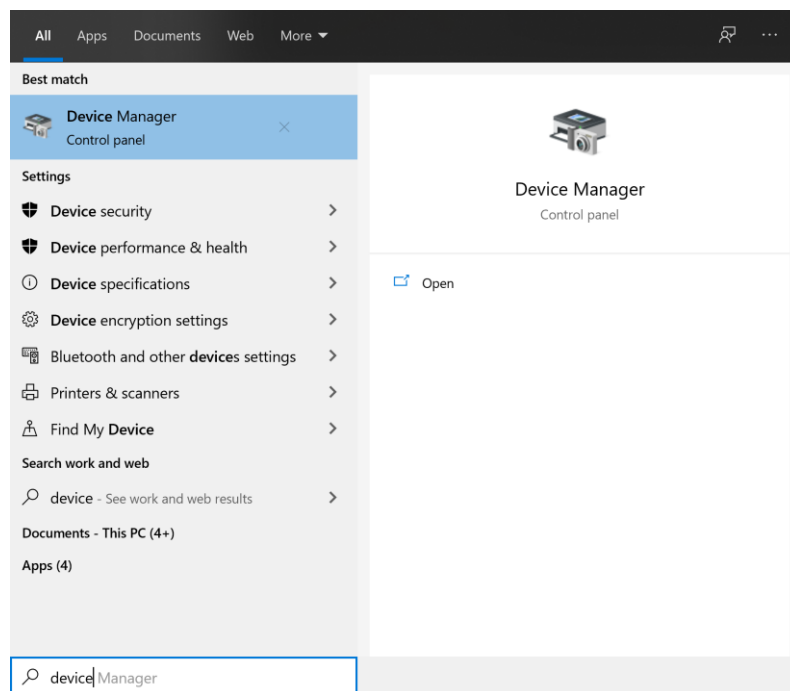


Figura 115. Abrir o gestor de Dispositivos

Em seguida, expanda os dispositivos Universal Serial Bus, clique com o botão direito do rato em “Digilent USB Device” e selecione Properties (Figura 116).

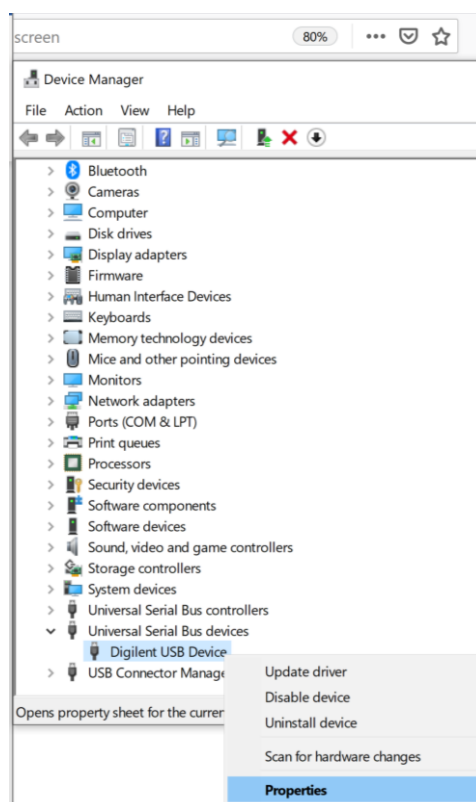


Figura 116. Abrir as propriedades do *driver* para a placa FPGA Nexys A7 da Digilent

Na janela de Propriedades, clique no painel Driver e selecione retorcir *driver* (Figura 117).

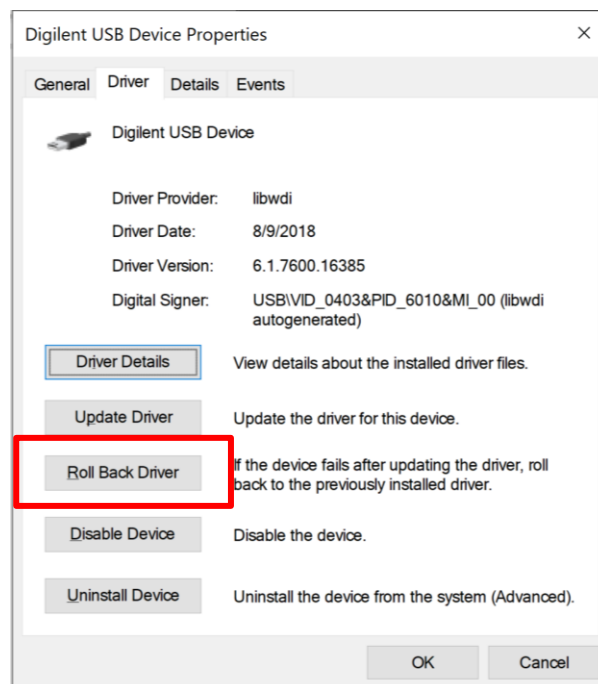


Figura 117. Reverter driver

Uma janela aparecerá perguntando por que está revertendo o *driver*. Selecione um motivo e clique em “Sim” (Figura 118).

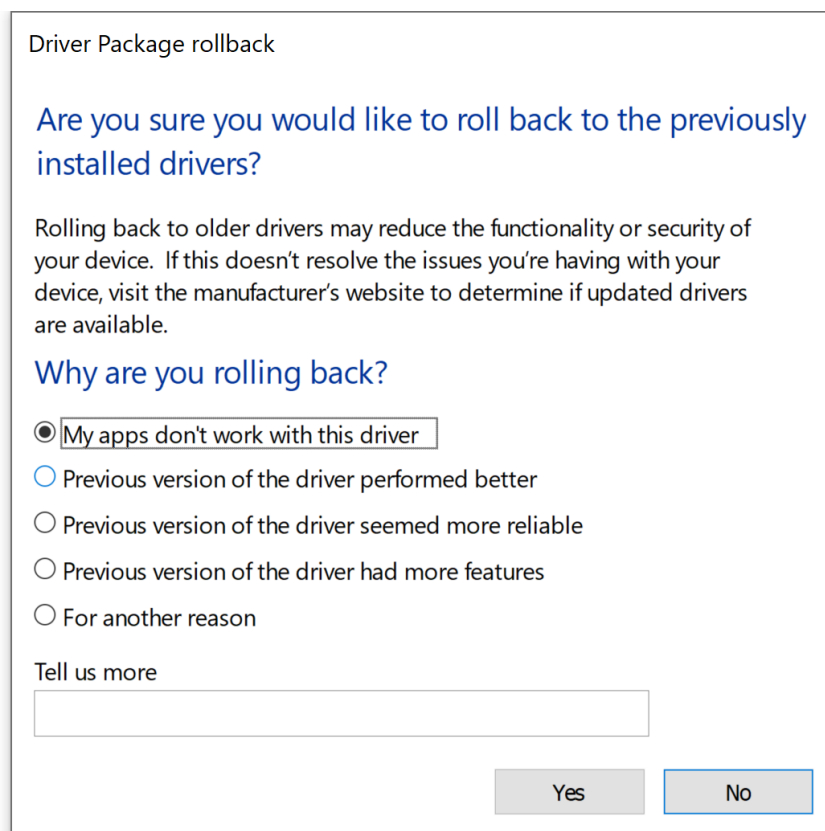


Figura 118. Confirmação do retrocesso

Após o *driver* voltar para o *driver* anterior, o “Driver Provider” deve estar identificado como “FTDI” (Figura 119).

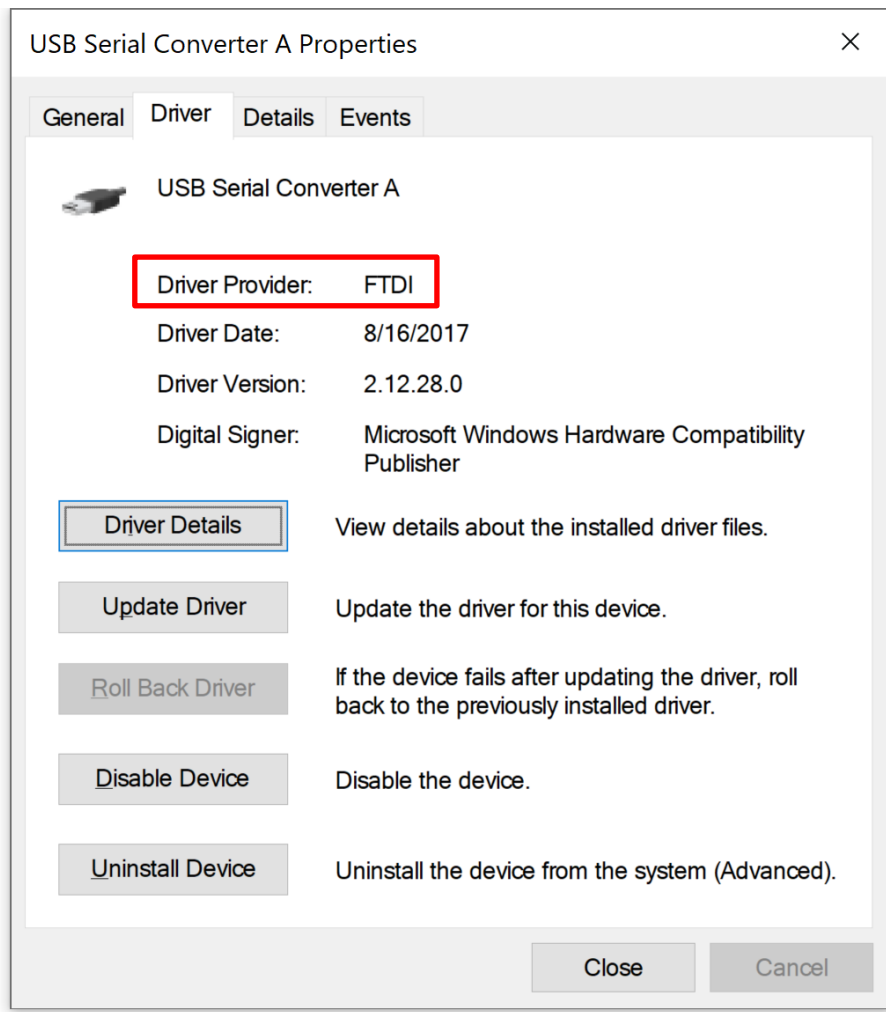


Figura 119. Driver FTDI ilustrado como o *driver* fornecido

Agora pode configurar ficheiros binários (bitfiles) na placa FPGA usando o Vivado. Contudo, precisará usar o Zadig para substituir o *driver* da placa Nexys A7, para que o PlatformIO possa descarregar o programa no RVfpgaNexys. Assim, é recomendável que use o PlatformIO para configurar também os ficheiros binários (bitfiles), em vez de usar o Vivado – isso evitará que tenha de trocar *drivers* constantemente.

Apêndice F: Uso do RVfpga numa Aplicação IoT Industrial

Em Julho de 2020, Daniel León González, aluno de mestrado da Universidade Complutense de Madri, concluiu sua tese de mestrado intitulada “Implementação FPGA de um sistema-on-chip ad-hoc RISC-V para IoT industrial”. Este trabalho mostra o uso do RVfpga em uma aplicação real de IoT industrial. Fornecemos o resumo do projeto abaixo, e a tese completa está disponível em: https://eprints.ucm.es/62106/1/DANIEL_LEON_GONZALEZ_DL_-_FPGA_Implementation_of_an_ad-hoc_RISC-V_SoC_for_Industrial_IoT_Graded_4286351_962908330.pdf.

Implementação FPGA de um System-on-Chip RISC-V ad-hoc para IoT Industrial

Resumo: Os dispositivos para nós IoT precisam de ser eficientes em termos de energia e económicos, mas não exigem um grande desempenho a nível de computação em muitos cenários. Isso muda substancialmente num ambiente de IoT industrial, onde a utilização massiva de sensores e o ritmo acelerado de eventos exigem mais capacidade de processamento. Um nó desenvolvido à medida, usando um processador eficiente e um sistema operativo de elevado desempenho e rico em recursos, pode equilibrar esses requisitos e oferecer uma solução óptima. Este projeto trata da realização do circuito, utilizando uma FPGA Artix-7, para um protótipo de nó IoT baseado na arquitetura do processador RISC-V. O projeto apresenta o SoC personalizado implementado, e o desenvolvimento dos *drivers* Zephyr OS necessários para suportar uma aplicação de demonstração. O nó é ligado numa rede em estrela em torno de um *router* personalizado. Mensagens podem ser enviadas e recebidas de ponta-a-ponta, entre o nó e a plataforma na *cloud* ThingSpeak. Esta tese inclui uma análise das implementações de processadores RISC-V existentes, uma descrição dos elementos necessários e um guia detalhado para configuração do ambiente e design do projeto.