



**THE IMAGINATION UNIVERSITY PROGRAMME**

# **RVfpga-SoC Lab 4**

## **Executar o Zephyr no SweRVolf**

**Tabela 1. Termos do RVfpga**

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

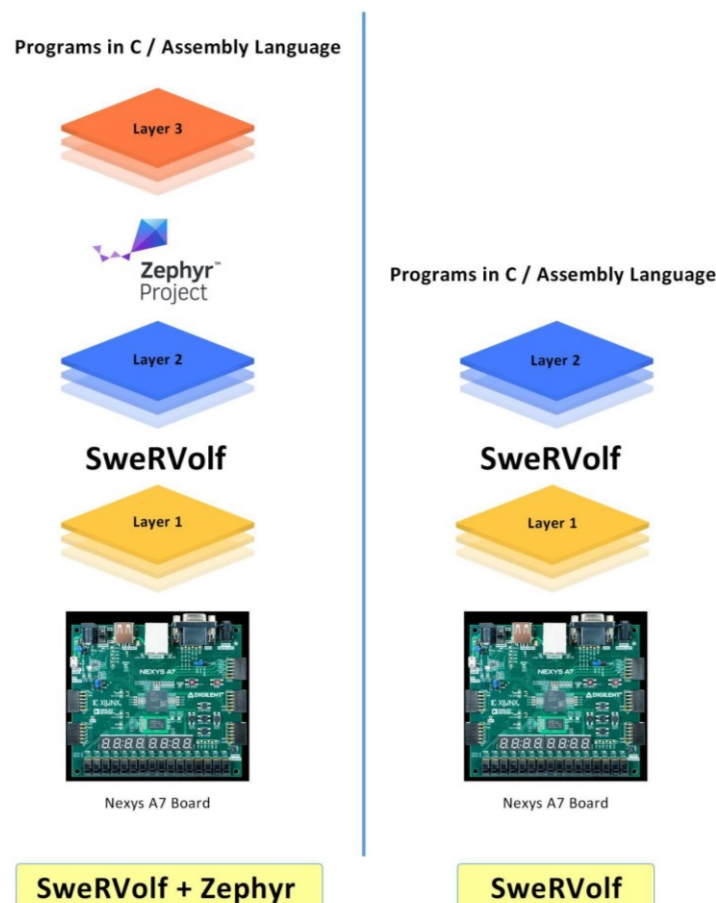
## 1. Introdução

Neste laboratório, irá executar o sistema operativo de tempo real (RTOS) Zephyr no SweRVolf. Um sistema operativo de tempo real (RTOS) é um sistema operativo destinado a aplicações de tempo-real que processam os dados à medida que eles chegam, principalmente sem atraso de buffer.

Nos Labs 2 e 3, executou programas simples escritos em linguagem RISC-V Assembly ou C. Em aplicações práticas, um SoC quase sempre executa um sistema operativo, e as aplicações são executadas sobre o sistema operativo.

Existem duas categorias gerais de sistemas operativos para sistemas embebidos: sistemas operativos baseados em Linux e sistemas operativos de tempo real (RTOS). Quando um SoC é projetado com um CPU específico, o projeto geralmente é ajustado para usar um ou outro tipo de sistema operativo. O SweRVolf foi criado com a intenção de executar um sistema operativo de tempo real. A CPU EH1 do SweRV não tem uma unidade de gestão de memória e, portanto, teria dificuldades para executar o Linux.

A Figura 1 mostra uma ilustração das diferentes camadas de hardware/software no sistema geral.



**Figura 1. Camadas (Layers) sobre as placas FPGA**

Neste laboratório, irá: ver a descrição do Zephyr RTOS, criar e executar o Zephyr RTOS no SweRVolf e criar e executar aplicações Zephyr.

## 2. Requisitos

Para concluir este laboratório, precisará instalar o seguinte:

- Vivado 2019.2 Web Pack (Consulte o Guia de Instalação (Página No.04))
- Verilator (v4.106) (Consulte o Guia de Instalação (página nº 09))
- FuseSoC (Consulte o Guia de instalação (página nº 10))
- OpenOCD (versão para RISC-V) (Consulte o Guia de Instalação (página nº 10))
- Zephyr Pré-requisitos (Consulte o Guia de Instalação (página nº 11))
- Zephyr SDK (v0.12.4) (Consulte o Guia de Instalação (página nº 12))
- PuTTY (Consulte o Guia de Instalação (página nº 12))

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

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

## 3. Resumo do Zephyr

O Zephyr Project é um sistema operativo de tempo real escalável que oferece suporte a várias arquiteturas de hardware, otimizado para dispositivos com recursos limitados e desenvolvido com a segurança em mente. O sistema operativo Zephyr é baseado num kernel de tamanho reduzido, projetado para uso em sistemas com recursos limitados: desde simples sensores ambientais e LEDs em sistemas embebidos vestíveis até sofisticados relógios inteligentes e gateways sem fio de IoT.

O Zephyr oferece uma série de serviços habituais para desenvolvimento: Multi-threading, interrupções, alocação de memória, sincronização entre threads, passagem de dados entre threads e gestão de energia.

O Zephyr oferece suporte a uma ampla variedade de placas com diferentes arquiteturas de CPU e ferramentas de desenvolvimento. Os colaboradores adicionaram suporte a um número cada vez maior de SoCs, plataformas e drivers.

O kernel do Zephyr é compatível com várias arquiteturas, incluindo

- RISC-V (32 e 64 bits)

Para obter informações mais detalhadas sobre o Zephyr Project, leia a documentação em <http://docs.zephyrproject.org>.

Neste laboratório, primeiro irá incorporar a versão 2.4 do Zephyr no Workspace. Em seguida, irá criar o código para alguns exemplos que vêm com o Zephyr. Neste laboratório irá ver exemplos de uso do Zephyr tanto em hardware quanto em simulação.

## 4. Entendendo as Camadas de Hardware/Software

Nos Labs 2 e 3, o processo de execução de programas na placa FPGA seguiu as etapas:

### **Etapas 1. Configuração do SweRVolf na placa FPGA**

Primeiro, descarregou o SweRVolf, o sistema RISC-V realizado para uma FPGA, para a placa FPGA Nexys A7. Fez a configuração do SweRVolf na FPGA carregando o Bitstream para a placa usando o PlatformIO ou usando o comando de execução do FuseSoC, que carrega o Bitstream gerado para a placa se ela estiver conectada.

### **Etapas 2. Crie e execute programas no SweRVolf**

A segunda etapa é criar programas RISC-V e programá-los no SweRVolf.

Neste laboratório, irá alterar essas etapas para adicionar outra camada, o Zephyr RTOS (sistema operativo de tempo real) ao SweRVolf, e executar programas no Zephyr. As etapas para fazer isso são as seguintes:

### **Etapas 1. Faça a configuração do SweRVolf na placa FPGA**

O mesmo que acima.

### **Etapas 2. Compile o Zephyr**

Nesta etapa, crie uma aplicação para o Zephyr. O processo de criação de uma aplicação também cria o Zephyr RTOS subjacente. O resultado é um ficheiro elf.

### **Etapas 3. Carregue os programas no SweRVolf.**

Nesta etapa, carregará o ficheiro elf gerado durante a Etapa 2 no SweRVolf.

A ilustração lado-a-lado dos dois modos de execução de um programa é mostrada na Figura 1 acima.

Agora, irá ver como criar aplicações do Zephyr e, em seguida, executar essas aplicações no Zephyr.

## **5. Adição de Suporte para o Zephyr no SweRVolf**

Nesta seção do laboratório, irá adicionar o Zephyr ao WORKSPACE.

Abra o terminal do Ubuntu e conclua as etapas a seguir:

**Etapas 1.** Vá até o diretório "**SweRVolf**", no qual criou o espaço de trabalho no laboratório anterior, para usá-lo como raiz do projeto. Conhecido como **\$WORKSPACE**. Agora precisa definir as mesmas variáveis de shell novamente. Para fazer isso, execute o seguinte:

- `export WORKSPACE=$(pwd)`
- `export SWERVOLF_ROOT=$WORKSPACE/fusesoc_libraries/swervolf`

também pode escrever o comando "`printenv <variable-name>`" na janela do terminal para verificar se as variáveis da shell foram definidas com êxito ou não.

```
~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ export WORKSPACE=$(pwd)
~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ export SWERVOLF_ROOT=$WORKSPACE/fusesoc_libraries/swervolf
~/RVfpgaSoC/Labs/LabProjects/SweRVolf$
```

**Figura 2. Definir as variáveis da shell**

### **Etapas 2. Adicionar drivers específicos do Zephyr e do SweRVolf**

Crie um espaço de trabalho West (ferramenta de construção do Zephyr) no mesmo diretório que o espaço de trabalho do FuseSoC executando:

➤ `west init`

```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ west init
=== Initializing in /home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf
--- Cloning manifest repository from https://github.com/zephyrproject-rtos/zephyr, rev. master
Initialized empty Git repository in /home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/.west/manifest-tmp/.git/
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 551804 (delta 2), reused 1 (delta 1), pack-reused 551800
Receiving objects: 100% (551804/551804), 375.87 MiB | 435.00 KiB/s, done.
Resolving deltas: 100% (418833/418833), done.
From https://github.com/zephyrproject-rtos/zephyr
* branch                master                                -> FETCH_HEAD
* [new branch]          backport-23821-to-v1.14-branch        -> origin/backport-23821-to-v1.14-branch
* [new branch]          backport-24971-to-v1.14-branch        -> origin/backport-24971-to-v1.14-branch
* [new branch]          backport-25852-to-v1.14-branch        -> origin/backport-25852-to-v1.14-branch
* [new branch]          backport-26571-to-v1.14-branch        -> origin/backport-26571-to-v1.14-branch
* [new branch]          backport-29181-to-v2.4-branch         -> origin/backport-29181-to-v2.4-branch
* [new branch]          backport-31759-to-v2.5-branch         -> origin/backport-31759-to-v2.5-branch
* [new branch]          backport-31908-to-v2.4-branch         -> origin/backport-31908-to-v2.4-branch
* [new tag]             zephyr-v2.2.0                      -> zephyr-v2.2.0
* [new tag]             zephyr-v2.2.1                      -> zephyr-v2.2.1
* [new tag]             zephyr-v2.3.0                      -> zephyr-v2.3.0
* [new tag]             zephyr-v2.4.0                      -> zephyr-v2.4.0
* [new tag]             zephyr-v2.5.0                      -> zephyr-v2.5.0
b0b20112e80187705a08240919613ca9937baae6 refs/remotes/origin/master
Branch 'master' set up to track remote branch 'master' from 'origin'.
Already on 'master'
--- setting manifest.path to zephyr
=== Initialized. Now run "west update" inside /home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf.
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$
```

**Figura 3. West inicializado**

**Etapa 3.** Adicione os drivers específicos do SweRVolf e o Board Support Package (BSP) usando o seguinte comando:

➤ `west config manifest.path fusesoc_libraries/swervolf`

```
~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ west config manifest.path fusesoc_libraries/swervolf
~/RVfpgaSoC/Labs/LabProjects/SweRVolf$
```

**Figura 4. Configuração do West**

➤ `west update`

Isso pode levar vários minutos para concluir o processo de download, dependendo da velocidade da ligação à Internet.

```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ west update
=== updating zephyr (zephyr):
HEAD is now at 7a3b253ced release: Zephyr 2.4.0
WARNING: left behind zephyr branch "master"; to switch back to it (fast forward):
  git -C zephyr checkout master
=== updating cmsis (modules/hal/cmsis):
--- cmsis: initializing
Initialized empty Git repository in /home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/modules/hal/cmsis/.git/
--- cmsis: fetching, need revision 542b2296e6d515b265e25c6b7208e8fea3014f90
remote: Enumerating objects: 563, done.
remote: Counting objects: 100% (563/563), done.
remote: Compressing objects: 100% (291/291), done.
remote: Total 563 (delta 288), reused 528 (delta 267), pack-reused 0
Receiving objects: 100% (563/563), 2.21 MiB | 618.00 KiB/s, done.
Resolving deltas: 100% (288/288), done.
From https://github.com/zephyrproject-rtos/cmsis
* [new branch]      master      -> refs/west/master
HEAD is now at 542b229 DSP: Integrate CMSIS-DSP 1.8.0 (CMSIS 5.7.0)
HEAD is now at 542b229 DSP: Integrate CMSIS-DSP 1.8.0 (CMSIS 5.7.0)
=== updating hal_atmel (modules/hal/atmel):

=====
=== updating trusted-firmware-m (modules/tee/tfm):
--- trusted-firmware-m: initializing
Initialized empty Git repository in /home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/modules/tee/tfm/.git/
--- trusted-firmware-m: fetching, need revision 143df675557305b61f7930a50459a53a8d2bb097
remote: Enumerating objects: 1970, done.
remote: Counting objects: 100% (1970/1970), done.
remote: Compressing objects: 100% (1249/1249), done.
remote: Total 16030 (delta 633), reused 1498 (delta 536), pack-reused 14060
Receiving objects: 100% (16030/16030), 36.00 MiB | 872.00 KiB/s, done.
Resolving deltas: 100% (7538/7538), done.
From https://github.com/zephyrproject-rtos/trusted-firmware-m
* [new branch]      master      -> refs/west/master
HEAD is now at 143df67 CMakeLists.txt: make BL2 configurable
HEAD is now at 143df67 CMakeLists.txt: make BL2 configurable
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$
```

Figura 5. Atualização do West

O espaço de trabalho agora terá a seguinte aparência:

```
$WORKSPACE
├── fusesoc_libraries
│   ├── ...
│   └── swervolf
└── ...
    └── zephyr
```

## 6. Criação e Execução de Aplicações Zephyr no Verilator

Nesta seção, é explicado como criar programas que podem ser executados no Zephyr. Em seguida, irá ver como os simular no Verilator. Irá usar dois exemplos de programas.

### 1. Exemplo de Hello World do Zephyr

Este exemplo exibe "Hello World" + "Nome da placa configurada" no terminal.

Consulte a Figura 6 para ver o código-fonte.

```
1
2 #include <zephyr.h>
3 #include <sys/printk.h>
4
5 void main(void)
6 {
7     printk("Hello World! %s\n", CONFIG_BOARD);
8 }
9
```

Figura 6. main.c do exemplo hello\_world

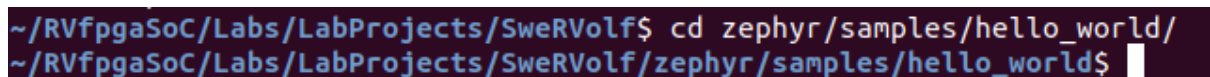


**Etapla 1.** Vá para o diretório deste exemplo, que está localizado em:

```
$WORKSPACE/zephyr/samples/hello_world
```

Para fazer isso, use o comando:

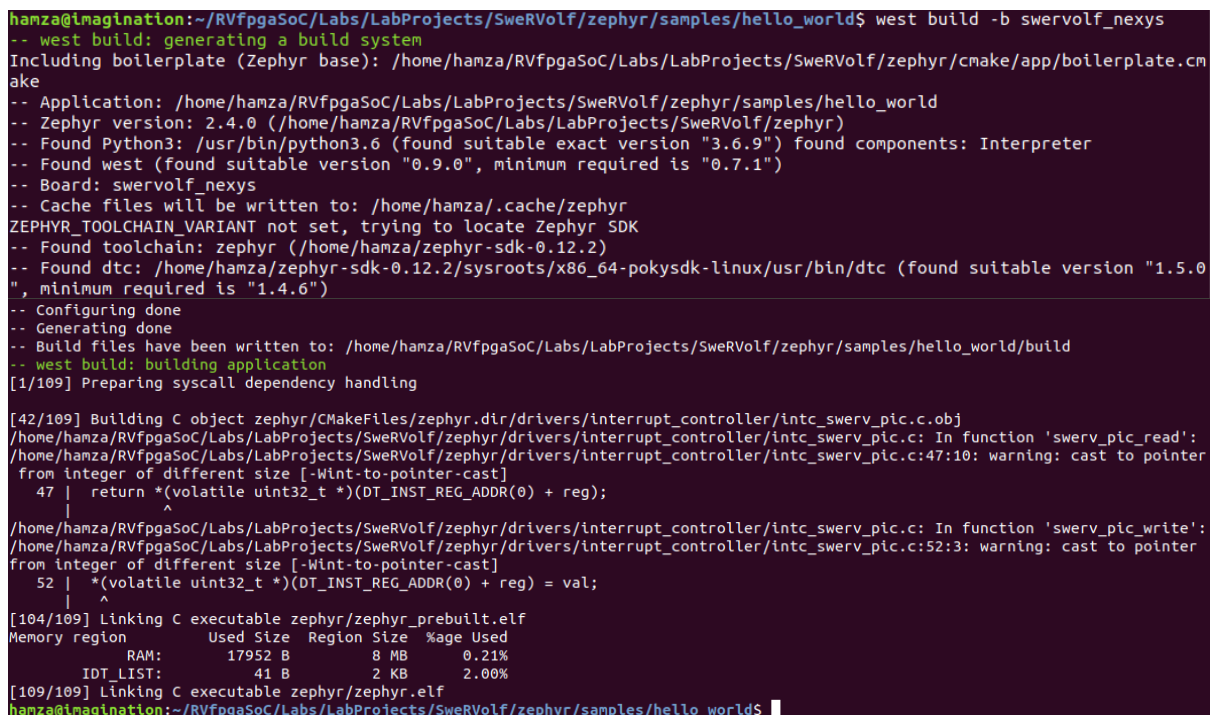
```
> cd zephyr/samples/hello_world
```



**Figura 7.** Navegue até o diretório `hello_world`

**Etapla 2.** Crie o código para o exemplo "hello\_world" usando o comando:

```
> west build -b swervolf_nexys
```



**Figura 8.** Compilação do `hello_world`

Isso criará os ficheiros `zephyr.elf` e `zephyr.bin` para o exemplo **hello\_world**. Irá usar o ficheiro **".bin"** no simulador, mas ele deve primeiro ser convertido num ficheiro Verilog hexadecimal adequado.

**Etapla 3.** Converta o ficheiro **".bin"** num ficheiro **".hex"**:

Para criar o ficheiro **".hex"**, execute o seguinte comando no diretório `hello_world`:

```
> python3 $SWERVOLF_ROOT/sw/makehex.py build/zephyr/zephyr.bin >
/home/<username>/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples/hello_
world/App.hex
```



(Substitua <username> pelo seu nome de utilizador)

```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples/hello_world$ python3 $SWERVOLF_ROOT/sw/makehex.py
build/zephyr/zephyr.bin > /home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples/hello_world/App.hex
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples/hello_world$
```

**Figura 9. Ficheiro hexadecimal hello\_world criado**

**Etapa 4.** Navegue até o diretório WORKSPACE:

➤ `cd $WORKSPACE`

```
~/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples/hello_world$ cd $WORKSPACE
~/RVfpgaSoC/Labs/LabProjects/SweRVolf$
```

**Figura 10. Navegue até o diretório principal do Workspace**

**Etapa 5.** Carregue o ficheiro ".hex" no simulador:

➤ `fusesoc run --target=sim swervolf --ram_init_file=zephyr/samples/hello_world/App.hex`

```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ fusesoc run --target=sim swervolf
--ram_init_file=zephyr/samples/hello_world/App.hex
WARNING: Unknown item compilation_mode in section Xsim
INFO: Preparing ::cdc_utils:0.1-r1
INFO: Preparing chipsalliance.org:cores:SweRV_EH1:1.8
INFO: Preparing fusesoc:utils:generators:0.1.5
INFO: Preparing ::jtag_vpi:0-r5
INFO: Preparing pulp-platform.org::common_cells:1.20.0
INFO: Preparing ::simple_spi:1.6.1
INFO: Preparing ::uart16550:1.5.5-r1
INFO: Preparing ::verilog-arbiter:0-r3
INFO: Preparing ::wb_common:1.0.3
INFO: Preparing pulp-platform.org::axi:0.25.0
INFO: Preparing ::wb_intercon:1.2.2-r1
INFO: Preparing ::swervolf:0.7.3
INFO: Generating ::swervolf-intercon:0.7.3
Found master ifu
Found master lsu
Found master sb
Found slave io
Found slave ram
=====
INFO: Generating ::swervolf-swerv_default_config:0.7.3
INFO: Generating ::swervolf-version:0.7.3
INFO: Generating ::swervolf-wb_intercon:0.7.3
Found master io
Found slave rom
Found slave spi_flash
Found slave sys
Found slave uart
=====
INFO: Setting up project
```

**Figura 11. Execução do fusesoc**

O terminal mostrará a seguinte saída (ver Figura 12).

```
make[1]: Leaving directory '/home/hamza/SweRVolf/build/swervolf_0.7.3/sim-verilator'
INFO: Running
INFO: Running simulation
Loading RAM contents from /home/hamza/SweRVolf/zephyr/samples/hello_world/App.hex
Releasing reset
*** Booting Zephyr OS build zephyr-v2.4.0 ***
Hello World! swervolf_nexys
```

**Figura 12. Exemplo de saída do hello\_world**

Pressione "**CTRL + C**" para interromper o programa.

## 2. Exemplo dos Filósofos no Zephyr

Uma implementação de uma solução para o Dining Philosophers Problem (um problema clássico de sincronização de várias threads). Esta implementação específica demonstra o uso de vários threads preemptivas e cooperativas de prioridades diferentes, bem como mutexes dinâmicos e a suspensão de uma thread.

O filósofo tenta sempre pegar o garfo mais baixo primeiro (f1 e depois f2). Quando terminar, ele devolverá os garfos na ordem inversa (f2, depois f1). Se ele pegar dois garfos, ele está COMENDO. Caso contrário, ele está PENSANDO. Os estados de transição também são mostrados, como FOME, quando o filósofo está com fome, mas os garfos não estão disponíveis, e SEGURANDO UM GARFO, quando o filósofo está esperando que o segundo garfo esteja disponível.

Cada Filósofo alternará aleatoriamente entre os estados de COMER e PENSAR.

Aceda ao caminho a seguir para ver o código-fonte desse exemplo:

```
$WORKSPACE/zephyr/samples/philosophers/src/main.c
```

Para este exemplo, repetiremos o mesmo processo, mas no diretório de filósofos

**Etapa 1.** Esse programa de exemplo está no seguinte diretório:

```
$WORKSPACE/zephyr/samples/philosophers
```

Mude para esse diretório usando o seguinte comando:

```
> cd zephyr/samples/philosophers
```

```
hamza@hamza-lenovo:~/SweRVolf$ cd zephyr/samples/philosophers/
hamza@hamza-lenovo:~/SweRVolf/zephyr/samples/philosophers$
```

**Figura 13. Navegue até o diretório philosophers**

**Etapa 2.** Crie o código para o exemplo dos filósofos usando o seguinte comando:

```
> west build -b swervolf_nexys
```

```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples/philosophers$ west build -b swervolf_nexys
-- west build: generating a build system
Including boilerplate (Zephyr base): /home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/cmake/app/boilerplate.cmake
-- Application: /home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples/philosophers
-- Zephyr version: 2.4.0 (/home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr)
-- Found Python3: /usr/bin/python3.6 (found suitable exact version "3.6.9") found components: Interpreter
-- Found west (found suitable version "0.9.0", minimum required is "0.7.1")
-- Board: swervolf_nexys
-- Cache files will be written to: /home/hamza/.cache/zephyr
ZEPHYR_TOOLCHAIN_VARIANT not set, trying to locate Zephyr SDK
-- Found toolchain: zephyr (/home/hamza/zephyr-sdk-0.12.2)

-- Configuring done
-- Generating done
-- Build files have been written to: /home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples/philosophers/build
-- west build: building application
[1/110] Preparing syscall dependency handling

[44/110] Building C object zephyr/CMakeFi...interrupt_controller/intc_swerv_pic.c.obj
/home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/drivers/interrupt_controller/intc_swerv_pic.c: In function 'swerv_pic_read':
/home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/drivers/interrupt_controller/intc_swerv_pic.c:47:10: warning: cast to pointer from integer of different size [-Wint-to-pointer-cast]
    47 |     return *(volatile uint32_t *) (DT_INST_REG_ADDR(0) + reg);
        |           ^
/home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/drivers/interrupt_controller/intc_swerv_pic.c: In function 'swerv_pic_write':
/home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/drivers/interrupt_controller/intc_swerv_pic.c:52:3: warning: cast to pointer from integer of different size [-Wint-to-pointer-cast]
    52 |     *(volatile uint32_t *) (DT_INST_REG_ADDR(0) + reg) = val;
        |     ^
[105/110] Linking C executable zephyr/zephyr_prebuilt.elf
Memory region      Used Size  Region Size  %age Used
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples/philosophers$
```

**Figura 14. Compilação do exemplo dos Filósofos**

Isso criará os ficheiros zephyr.elf e zephyr.bin para o exemplo dos filósofos. Novamente, irá converter o ficheiro ".bin" num ficheiro Verilog hexadecimal adequado.

**Etapas 3.** Converta o ficheiro ".bin" num **ficheiro ".hex"**.

Para criar o ficheiro ".hex", execute o seguinte comando no diretório philosophers:

- `python3 $SWERVOLF_ROOT/sw/makehex.py build/zephyr/zephyr.bin > /home/<Username>/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples/philosophers/App.hex`

```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples/philosophers$ python3 $SWERVOLF_ROOT/sw/makehex.py build/zephyr/zephyr.bin > /home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples/philosophers/App.hex
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples/philosophers$
```

**Figura 15. Criar ficheiro hexadecimal de filósofos**

**Etapas 4.** Navegue até o diretório WORKSPACE:

- `cd $WORKSPACE`

```
~/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples/philosophers$ cd $WORKSPACE
~/RVfpgaSoC/Labs/LabProjects/SweRVolf$
```

**Figura 16. Diretório principal**

**Etapas 5.** Carregue o ficheiro .hex no simulador:

- `fusesoc run --target=sim swervolf --ram_init_file=zephyr/samples/philosophers/App.hex`

```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ fusesoc run --target=sim swervolf
--ram_init_file=zephyr/samples/philosophers/App.hex
WARNING: Unknown item compilation_mode in section Xsin
INFO: Preparing ::cdc_utils:0.1-r1
INFO: Preparing chipsalliance.org:cores:SweRV_EH1:1.8
INFO: Preparing fusesoc:utils:generators:0.1.5
INFO: Preparing ::jtag_vpi:0-r5
INFO: Preparing pulp-platform.org::common_cells:1.20.0
INFO: Preparing ::simple_spi:1.6.1
INFO: Preparing ::uart16550:1.5.5-r1
INFO: Preparing ::verilog-arbiter:0-r3
INFO: Preparing ::wb_common:1.0.3
INFO: Preparing pulp-platform.org::axi:0.25.0
INFO: Preparing ::wb_intercon:1.2.2-r1
INFO: Preparing ::swervolf:0.7.3
INFO: Generating ::swervolf-intercon:0.7.3
Found master ifu
Found master lsu
Found master sb
Found slave io
Found slave ram
=====
INFO: Generating ::swervolf-swerv_default_config:0.7.3
INFO: Generating ::swervolf-version:0.7.3
INFO: Generating ::swervolf-wb_intercon:0.7.3
Found master io
Found slave rom
Found slave spi_flash
Found slave sys
Found slave uart
=====
INFO: Setting up project
```

**Figura 17. Execução do fusesoc**

Agora irá ver o seguinte resultado:

```
make[1]: Leaving directory '/home/hamza/SweRVolf/build/swervolf_0.7.3/sim-verilator'
INFO: Running
INFO: Running simulation
Loading RAM contents from /home/hamza/SweRVolf/zephyr/samples/philosophers/App.hex
Releasing reset
*** Booting Zephyr OS build zephyr-v2.4.0 ***

Philosopher 0 [P: 3]   HOLDING ONE FORK
Philosopher 1 [P: 2]   EATING [ 125 ms ]
Philosopher 2 [P: 1]   THINKING [ 175 ms ]
Philosopher 3 [P: 0]   EATING [ 325 ms ]
Philosopher 4 [C:-1]   THINKING [ 400 ms ]
Philosopher 5 [C:-2]   STARVING
█

Demo Description
-----
An implementation of a solution to the Dining Philosophers
problem (a classic multi-thread synchronization problem).
This particular implementation demonstrates the usage of multiple
preemptible and cooperative threads of differing priorities, as
well as dynamic mutexes and thread sleeping.
```

**Figura 18. Saída do exemplo dos filósofos do Zephyr**

## 7. Criação da Aplicação Zephyr Para Hardware

Irá criar programas para o SwerVolf executando o Zephyr em hardware.

### 1. Exemplo do Zephyr Blinky

O Blinky é uma aplicação simples que pisca um LED continuamente usando o `GPIO API <gpio\_api>`. O código-fonte mostra como configurar os pinos GPIO como saídas, e depois ligá-los e desligá-los.

```

1
2  /*
3   * Copyright (c) 2016 Intel Corporation
4   *
5   * SPDX-License-Identifier: Apache-2.0
6   */
7
8  #include <zephyr.h>
9  #include <device.h>
10 #include <devicetree.h>
11 #include <drivers/gpio.h>
12
13 /* 1000 msec = 1 sec */
14 #define SLEEP_TIME_MS 1000
15
16 /* O identificador do nó devicetree para o alias "led0". */
17 #define LED0_NODE DT_ALIAS(led0)
18
19 #if DT_NODE_HAS_STATUS(LED0_NODE, okay)
20 #define LED0 DT_GPIO_LABEL(LED0_NODE, gpios)
21 #define PIN DT_GPIO_PIN(LED0_NODE, gpios)
22 #define FLAGS DT_GPIO_FLAGS(LED0_NODE, gpios)
23 #else
24 /* Erro de compilação se a placa não está configurada para piscar um LED. */
25 #error "Placa não suportada: o alias do devicetree led0 não está definido"
26 #define LED0 ""
27 #define PIN 0
28 #define FLAGS 0
29 #endif
30
31 void main(void)
32 {
33     const struct device *dev;
34     bool led_is_on = true;
35     int ret;
36
37     dev = device_get_binding(LED0);
38     if (dev == NULL) {
39         return;
40     }
41
42     ret = gpio_pin_configure(dev, PIN, GPIO_OUTPUT_ACTIVE | FLAGS);
43     if (ret < 0) {
44         return;
45     }
46
47     while (1) {
48         gpio_pin_set(dev, PIN, (int)led_is_on);
49         led_is_on = !led_is_on;
50         k_msleep(SLEEP_TIME_MS);
51     }

```

```
52 }
```

**Figura 19. main.c do exemplo blinky**

O caminho para esse exemplo é:

```
$WORKSPACE/zephyr/samples/basic/blinky/
```

```
> cd zephyr/samples/basic/blinky/
```

Navegue até o caminho acima e, em seguida, execute o seguinte comando no terminal para criar o exemplo e gerar os ficheiros ".elf" e ".bin":

```
> west build -b swervolf_nexys
```

Após a compilação do código, haverá agora um ficheiro executável **.elf** em `build/zephyr/zephyr.elf` e um ficheiro **.bin** em `build/zephyr/zephyr.bin`.

O ficheiro executável pode ser carregado no SweRVolf com um depurador, e o ficheiro binário pode ser convertido num ficheiro **.hex** e carregado na RAM para simulações, conforme descrito na próxima seção.

## 8. Executando a Aplicação Zephyr em Hardware

Para executar as aplicações na placa Nexys A7, precisa de carregar os programas usando o OpenOCD:

**Etapla 1.** Ligue a placa Nexys A7 ao computador e alimente-a. Em seguida, execute o comando de compilação da FPGA no diretório Workspace.

```
> cd $WORKSPACE
```

```
> fusesoc run --target=nexys_a7 --run swervolf
```

```
***** Xilinx cs_server v2019.2.0
**** Build date : Nov 07 2019-10:41:48
** Copyright 2017-2019 Xilinx, Inc. All Rights Reserved.

INFO: Trying to use hardware target localhost:3121/xilinx_tcf/Digilent/210292B0EF83A
INFO: [Labtoolstcl 44-466] Opening hw_target localhost:3121/xilinx_tcf/Digilent/210292B0EF83A
INFO: Opened hardware target localhost:3121/xilinx_tcf/Digilent/210292B0EF83A on try 1.
INFO: Found xc7a100tcs9324-1 as part of xc7a100t_0.
INFO: Programming bitstream to device xc7a100t_0 on target localhost:3121/xilinx_tcf/Digilent/210292B0EF83A.
INFO: [Labtools 27-3164] End of startup status: HIGH
INFO: [Labtoolstcl 44-464] Closing hw_target localhost:3121/xilinx_tcf/Digilent/210292B0EF83A

INFO: SUCCESS! FPGA xc7a100tcs9324-1 successfully programmed with bitstream swervolf_0.7.3.bit.
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$
```

**Figura 20. Executar a configuração da FPGA**

**Etapla 2.** Programe a placa com o OpenOCD.

```
> openocd -c "set BITFILE build/swervolf_0.7.3/nexys_a7-
vivado/swervolf_0.7.3.bit" -f
$SWERVOLF_ROOT/data/swervolf_nexys_program.cfg
```



```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ openocd -c "set BITFILE build/swervolf_0.7.3/nexys_a7-vivado/swervolf_0.7.3.bit"
-f $SWERVOLF_ROOT/data/swervolf_nexys_program.cfg
Open On-Chip Debugger 0.11.0-rc1+dev-01535-g3651cbdfd (2021-01-30-12:10)
Licensed under GNU GPL v2
For bug reports, read
  http://openocd.org/doc/doxygen/bugs.html
build/swervolf_0.7.3/nexys_a7-vivado/swervolf_0.7.3.bit
DEPRECATED! use 'adapter driver' not 'interface'
DEPRECATED! use 'adapter speed' not 'adapter_khz'
Info : ftdi: if you experience problems at higher adapter clocks, try the command "ftdi_tdo_sample_edge falling"
Info : clock speed 10000 kHz
Info : JTAG tap: xc7.tap tap/device found: 0x13631093 (mfg: 0x049 (Xilinx), part: 0x3631, ver: 0x1)
Warn : gdb services need one or more targets defined
shutdown command invoked
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$
```

**Figura 21. Executar o OpenOCD**

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

➤ `openocd -f $SWERVOLF_ROOT/data/swervolf_nexys_debug.cfg`

```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ openocd -f $SWERVOLF_ROOT/data/swervolf_nexys_debug.cfg
Open On-Chip Debugger 0.11.0-rc1+dev-01535-g3651cbdfd (2021-01-30-12:10)
Licensed under GNU GPL v2
For bug reports, read
  http://openocd.org/doc/doxygen/bugs.html
DEPRECATED! use 'adapter driver' not 'interface'
DEPRECATED! use 'adapter speed' not 'adapter_khz'
Info : ftdi: if you experience problems at higher adapter clocks, try the command "ftdi_tdo_sample_edge falling"
Info : clock speed 10000 kHz
Info : JTAG tap: riscv.cpu tap/device found: 0x13631093 (mfg: 0x049 (Xilinx), part: 0x3631, ver: 0x1)
Info : datacount=2 progbufsize=0
Warn : We won't be able to execute fence instructions on this target. Memory may not always appear consistent. (progbufsize=
0, impebreak=0)
Info : Examined RISC-V core; found 1 harts
Info : hart 0: XLEN=32, misa=0x40001104
Info : starting gdb server for riscv.cpu on 3333
Info : Listening on port 3333 for gdb connections
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
```

**Figura 22. OpenOCD conectado**

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

➤ `telnet localhost 4444`

```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ telnet localhost 4444
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Open On-Chip Debugger
>
```

**Figura 23. Ligação telnet**

O OpenOCD suporta o carregamento de ficheiros de programa ELF executando `load_image /path/to/file.elf`. Lembre-se de que o caminho é relativo ao diretório de onde o OpenOCD foi iniciado.

➤ `load_image zephyr/samples/basic/blink/build/zephyr/zephyr.elf`

```
> load_image zephyr/samples/basic/blink/build/zephyr/zephyr.elf
14848 bytes written at address 0x00000000
downloaded 14848 bytes in 1.420706s (10.206 KiB/s)
>
```

Figura 24. Carregar o ficheiro de imagem .elf

Depois que o programa ter sido carregado, defina o Program Counter com o endereço zero usando o seguinte comando:

```
> reg pc 0
```

```
> reg pc 0
pc (/32): 0x00000000
>
```

Figura 25. Colocar o Program Counter a zero

Agora, inicie o programa usando este comando:

```
> resume
```

```
> resume
>
```

Figura 26. Iniciar o programa

Agora verá que o LED mais à direita da placa Nexys A7 começará a piscar.

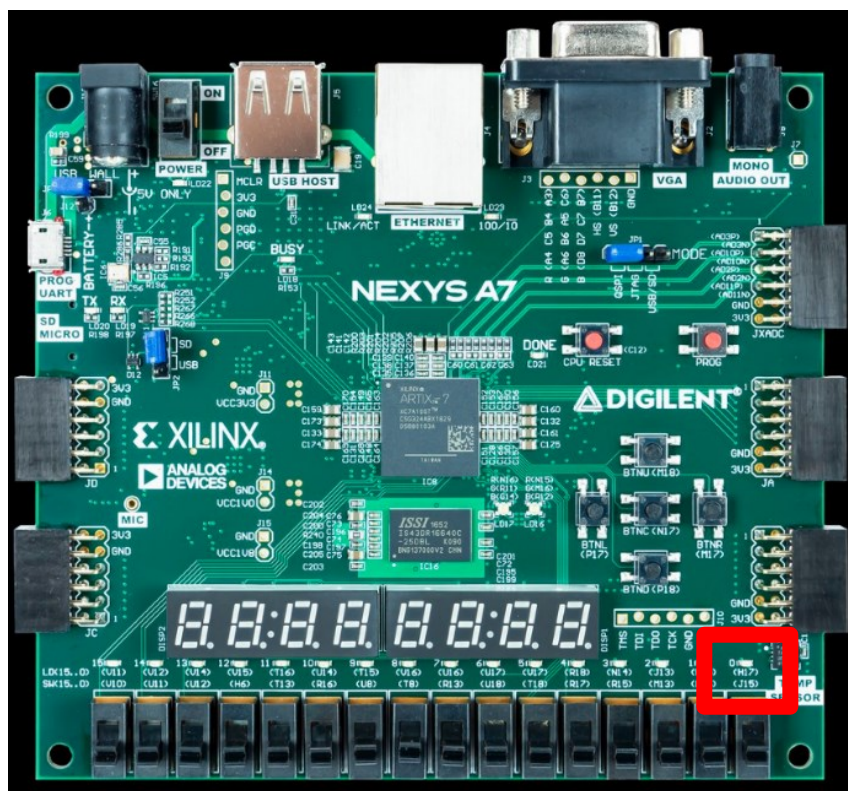


Figura 27: LED piscando

Agora pode pressionar "CTRL + C" para sair do programa.

## 9. Resumo do Desenvolvimento de Aplicações no Zephyr

O sistema de compilação do Zephyr é baseado no CMake. O sistema de compilação é centrado na aplicação e exige que as aplicações baseadas no Zephyr iniciem a compilação da árvore do código-fonte do kernel. A compilação da aplicação controla a configuração e constrói um processo da aplicação e do próprio Zephyr, compilando-os num único binário.

O diretório base do Zephyr contém o código-fonte do Zephyr, as opções de configuração do kernel e as definições de compilação.

Os ficheiros no diretório da aplicação associam o Zephyr à aplicação. Esse diretório contém todos os ficheiros específicos da aplicação, como opções de configuração e código-fonte.

Uma aplicação na sua forma mais simples tem o conteúdo listado aqui e descrito abaixo:

```
/App
├── CMakeLists.txt
├── prj.conf
└── src
    main.c
```

**CMakeLists.txt:** Este ficheiro informa ao sistema de compilação onde encontrar os outros ficheiros da aplicação e associa o diretório da aplicação ao sistema de compilação CMake do Zephyr. Essa ligação fornece recursos suportados pelo sistema de compilação do Zephyr, como ficheiros de configuração do kernel específicos da placa, a capacidade de executar e depurar binários compilados em hardware ou emulado, e outros.

**Ficheiros de configuração do kernel:** Uma aplicação normalmente fornece um ficheiro de configuração Kconfig (geralmente chamado prj.conf) que especifica valores específicos da aplicação para uma ou mais opções de configuração do kernel. Essas configurações da aplicação são combinadas com as configurações específicas da placa para produzir uma configuração do kernel.

**Ficheiros de código-fonte da aplicação:** Uma aplicação normalmente tem um ou mais ficheiros específicos da aplicação escritos em C ou em linguagem Assembly. Esses ficheiros geralmente estão localizados num subdiretório chamado **src**.

## 10. Criação de Uma Nova Aplicação Zephyr

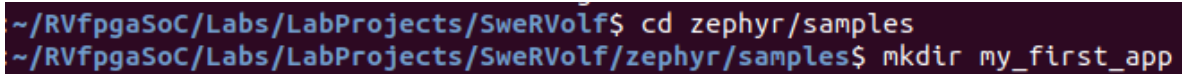
Siga estas etapas para criar um diretório de nova aplicação.

**Etapas 1.** Vá para o diretório Samples:

```
> cd zephyr/samples
```

**Etapa 2.** Crie um novo diretório para a aplicação:

```
> mkdir my_first_app
```



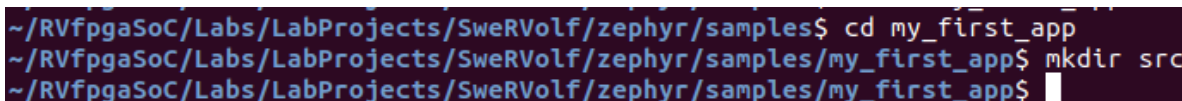
```
~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ cd zephyr/samples  
~/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples$ mkdir my_first_app
```

**Figura 28.** Criar diretório do projeto

**Etapa 3.** Recomenda-se colocar todo o código-fonte da aplicação num subdiretório chamado src.

Isso facilita a distinção entre os ficheiros de projeto e os ficheiros de código-fonte:

```
> cd my_first_app  
> mkdir src
```

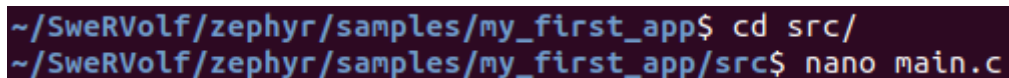


```
~/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples$ cd my_first_app  
~/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples/my_first_app$ mkdir src  
~/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples/my_first_app$
```

**Figura 29.** Criar o diretório src dentro do diretório do projeto

**Etapa 4.** Entre no diretório src e crie o ficheiro de código-fonte principal da aplicação, "main.c".

```
> cd src  
> nano main.c
```



```
~/SweRVolf/zephyr/samples/my_first_app$ cd src/  
~/SweRVolf/zephyr/samples/my_first_app/src$ nano main.c
```

**Figura 30.** Crie o ficheiro "main.c".

O Nano Editor será aberto no terminal do Ubuntu, conforme mostrado na figura abaixo:



**Figura 31. Editor GNU nano**

**Etapla 5.** Copie o código a seguir para o editor nano. Esse código é a mistura dos códigos-fonte de exemplo "hello\_world" e "blinky".

```
#include <zephyr.h>
#include <sys/printk.h>
#include <device.h>
#include <devicetree.h>
#include <drivers/gpio.h>

/* 1000 mseg = 1 segundo */
#define SLEEP_TIME_MS 1000

/* O identificador do nó device-tree para o alias "led0". */
#define LED0_NODE DT_ALIAS(led0)

#if DT_NODE_HAS_STATUS(LED0_NODE, ok)
#define LED0 DT_GPIO_LABEL(LED0_NODE, gpios)
#define PIN DT_GPIO_PIN(LED0_NODE, gpios)
#define FLAGS DT_GPIO_FLAGS(LED0_NODE, gpios)
#else
/* Erro de compilação se a placa não está configurada para piscar um LED. */
#error "Placa não suportada: o alias led0 device-tree não está definido"
#define LED0 ""
#define PIN 0
#define FLAGS 0
#endif

void main(void)
{
    const struct device *dev;
    bool led_is_on = true;
    int ret;

    dev = device_get_binding(LED0);
    if (dev == NULL) {
        return;
    }
}
```

```

ret = gpio_pin_configure(dev, PIN, GPIO_OUTPUT_ACTIVE | FLAGS);
if (ret < 0) {
    return;
}

while (1) {
    gpio_pin_set(dev, PIN, (int)led_is_on);
    led_is_on = !led_is_on;
    k_msleep(SLEEP_TIME_MS);
    printk("Esta aplicação Zephyr está sendo executada em %s\n", CONFIG_BOARD);
}
}

```

**Figura 32. "Código "main.c"**

Quando terminar de escrever o código, pressione **"CTRL + X"** para sair.

```

GNU nano 2.9.3

#include <zephyr.h>
#include <sys/printk.h>
#include <device.h>
#include <devicetree.h>
#include <drivers/gpio.h>

/* 1000 msec = 1 sec */
#define SLEEP_TIME_MS 1000

/* The devicetree node identifier for the "led0" alias. */
#define LED0_NODE DT_ALIAS(led0)

#if DT_NODE_HAS_STATUS(LED0_NODE, okay)
#define LED0 DT_GPIO_LABEL(LED0_NODE, gpios)
#define PIN DT_GPIO_PIN(LED0_NODE, gpios)
#define FLAGS DT_GPIO_FLAGS(LED0_NODE, gpios)
#else
/* A build error here means your board isn't set up to blink an LED. */
#error "Unsupported board: led0 devicetree alias is not defined"
#define LED0 ""
#define PIN 0
#define FLAGS 0
#endif

void main(void)
{
    const struct device *dev;
    bool led_is_on = true;
    int ret;

    dev = device_get_binding(LED0);
    if (dev == NULL) {
        return;
    }

    ret = gpio_pin_configure(dev, PIN, GPIO_OUTPUT_ACTIVE | FLAGS);
    if (ret < 0) {
        return;
    }

    while (1) {
        gpio_pin_set(dev, PIN, (int)led_is_on);
        led_is_on = !led_is_on;
        k_msleep(SLEEP_TIME_MS);
        printk("This Zephyr Application is Running on %s\n", CONFIG_BOARD);
    }
}

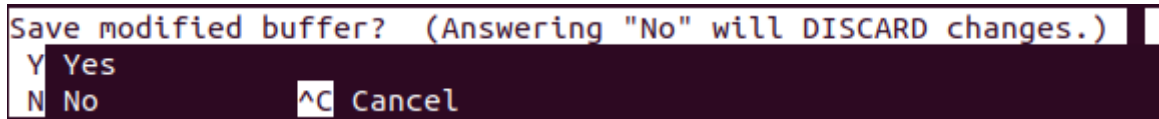
```

<sup>^</sup>G Get Help    <sup>^</sup>O Write Out    <sup>^</sup>W Where Is    <sup>^</sup>K Cut Text    <sup>^</sup>J Justify  
<sup>^</sup>X Exit        <sup>^</sup>R Read File    <sup>^</sup>\ Replace       <sup>^</sup>U Uncut Text    <sup>^</sup>T To Spell

**Figura 33. Código do ficheiro main.c**

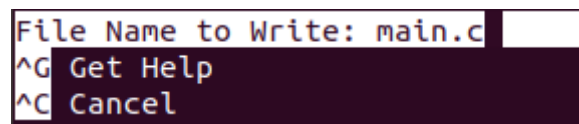


Em seguida, ele perguntará se deseja salvar o ficheiro, e deverá pressionar "y" para Yes (sim).



**Figura 34. Salvar o ficheiro main.c**

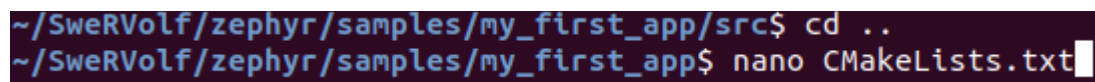
Pressione "**Enter**" para salvar o ficheiro com o nome "main.c".



**Figura 35. Confirme o nome main.c**

**Etapla 6.** Navegue para fora do diretório src e crie os ficheiros "CMakeLists.txt" e "prj.conf":

- cd ...
- nano CMakeLists.txt



**Figura 36. Criar CMakeLists.txt**

Copie o código a seguir para o editor nano:

```
cmake_minimum_required(VERSÃO 3.13.1)

find_package(Zephyr REQUIRED HINTS $ENV{ZEPHYR_BASE})
project(my_first_app)

target_sources(app PRIVATE src/main.c)
```

**Figura 37. Código do ficheiro "CMakeLists.txt".**

Agora, execute as mesmas etapas que executou para salvar o ficheiro "main.c".

```
GNU nano 2.9.3 CMakeLists.txt Modified

cmake_minimum_required(VERSION 3.13.1)

find_package(Zephyr REQUIRED HINTS $ENV{ZEPHYR_BASE})
project(my_first_app)

target_sources(app PRIVATE src/main.c)

^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify
^X Exit      ^R Read File ^_ Replace   ^U Uncut Text ^T To Spell
```

**Figura 38. Editor nano**

Agora, crie o ficheiro de configuração do projeto.

➤ nano prj.conf

```
~/SweRVolf/zephyr/samples/my_first_app$ nano prj.conf
~/SweRVolf/zephyr/samples/my_first_app$
```

**Figura 39. Criar um ficheiro de configuração de projeto**

As opções de configuração da aplicação são definidas no prj.conf no diretório da aplicação. Como está a usar um LED no código-fonte, tem de definir o parâmetro "CONFIG\_GPIO" como sim (yes).

```
CONFIG_GPIO=y
```

**Figura 40. "Código "prj.conf"**

```
GNU nano 2.9.3 prj.conf Modified

CONFIG_GPIO=y

^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text
^X Exit      ^R Read File ^_ Replace   ^U Uncut Text
```

**Figura 41. "Editor nano "prj.conf"**

Agora salve o ficheiro "prj.conf".

## Etapa 7. Crie o código para "my\_first\_app":

➤ `west build -b swervolf_nexys`

```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples/my_first_app$ west build -b swervolf_nexys
-- west build: generating a build system
Including boilerplate (Zephyr base): /home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/cmake/app/boilerplate.cmake
-- Application: /home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples/my_first_app
-- Zephyr version: 2.4.0 (/home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr)
-- Found Python3: /usr/bin/python3.6 (found suitable exact version "3.6.9") found components: Interpreter
-- Found west (found suitable version "0.9.0", minimum required is "0.7.1")
-- Board: swervolf_nexys
-- Cache files will be written to: /home/hamza/.cache/zephyr
ZEPHYR_TOOLCHAIN_VARIANT not set, trying to locate Zephyr SDK
-- Found toolchain: zephyr (/home/hamza/zephyr-sdk-0.12.2)
-- Found dtc: /home/hamza/zephyr-sdk-0.12.2/sysroots/x86_64-pokysdk-linux/usr/bin/dtc (found suitable version "1.5.0",
red is "1.4.6")
-- Configuring done
-- Generating done
-- Build files have been written to: /home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples/my_first_app/build
-- west build: building application
[1/109] Preparing syscall dependency handling

[43/109] Building C object zephyr/CMakeFiles/zephyr.dir/drivers/interrupt_controller/intc_swerv_pic.c.obj
/home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/drivers/interrupt_controller/intc_swerv_pic.c: In function 'swerv_pic_read':
/home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/drivers/interrupt_controller/intc_swerv_pic.c:47:10: warning: cast to pointer
from integer of different size [-Wint-to-pointer-cast]
  47 |     return *(volatile uint32_t *) (DT_INST_REG_ADDR(0) + reg);
      |           ^
/home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/drivers/interrupt_controller/intc_swerv_pic.c: In function 'swerv_pic_write':
/home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/drivers/interrupt_controller/intc_swerv_pic.c:52:3: warning: cast to pointer
from integer of different size [-Wint-to-pointer-cast]
   52 |     *(volatile uint32_t *) (DT_INST_REG_ADDR(0) + reg) = val;
      |     ^
[104/109] Linking C executable zephyr/zephyr_prebuilt.elf
Memory region      Used Size  Region Size  %age Used
RAM:                17968 B      8 MB      0.21%
IDT LIST:           41 B        2 KB      2.00%
[109/109] Linking C executable zephyr/zephyr.elf
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples/my_first_app$
```

**Figura 42. Compilação do "my\_first\_app"**

Os binários foram gerados com sucesso. Agora, executará o programa "my\_first\_app" na placa Nexys A7.

## Etapa 9. Navegue até ao diretório WORKSPACE:

➤ `cd $WORKSPACE`

```
~/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples/my_first_app$ cd $WORKSPACE
~/RVfpgaSoC/Labs/LabProjects/SweRVolf$
```

**Figura 43. Navegue até ao diretório Workspace**

**Etapa 10.** Conecte a placa Nexys A7 ao computador e, em seguida, execute o comando de configuração da FPGA no diretório Workspace.

➤ `fusesoc run --target=nexys_a7 --run swervolf`

```
***** Xilinx cs_server v2019.2.0
**** Build date : Nov 07 2019-10:41:48
** Copyright 2017-2019 Xilinx, Inc. All Rights Reserved.

INFO: Trying to use hardware target localhost:3121/xilinx_tcf/Digilent/210292B0EF83A
INFO: [Labtoolstcl 44-466] Opening hw_target localhost:3121/xilinx_tcf/Digilent/210292B0EF83A
INFO: Opened hardware target localhost:3121/xilinx_tcf/Digilent/210292B0EF83A on try 1.
INFO: Found xc7a100tcsg324-1 as part of xc7a100t_0.
INFO: Programming bitstream to device xc7a100t_0 on target localhost:3121/xilinx_tcf/Digilent/210292B0EF83A.
INFO: [Labtools 27-3164] End of startup status: HIGH
INFO: [Labtoolstcl 44-464] Closing hw_target localhost:3121/xilinx_tcf/Digilent/210292B0EF83A

INFO: SUCCESS! FPGA xc7a100tcsg324-1 successfully programmed with bitstream swervolf_0.7.3.bit.
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$
```

**Figura 44. Executar a configuração da FPGA**

**Etapla 11.** Programe a placa com o OpenOCD.

- `openocd -c "set BITFILE build/swervolf_0.7.3/nexys_a7-vivado/swervolf_0.7.3.bit" -f $SWERVOLF_ROOT/data/swervolf_nexys_program.cfg`

```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ openocd -c "set BITFILE build/swervolf_0.7.3/nexys_a7-vivado/swervolf_0.7.3.bit"
-f $SWERVOLF_ROOT/data/swervolf_nexys_program.cfg
Open On-Chip Debugger 0.11.0-rc1+dev-01535-g3651cbdfd (2021-01-30-12:10)
Licensed under GNU GPL v2
For bug reports, read
  http://openocd.org/doc/doxygen/bugs.html
build/swervolf_0.7.3/nexys_a7-vivado/swervolf_0.7.3.bit
DEPRECATED! use 'adapter driver' not 'interface'
DEPRECATED! use 'adapter speed' not 'adapter_khz'
Info : ftdi: if you experience problems at higher adapter clocks, try the command "ftdi_tdo_sample_edge falling"
Info : clock speed 10000 kHz
Info : JTAG tap: xc7.tap tap/device found: 0x13631093 (mfg: 0x049 (Xilinx), part: 0x3631, ver: 0x1)
Warn : gdb services need one or more targets defined
shutdown command invoked
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$
```

**Figura 45. Executar o OpenOCD**

**Etapla 12.** Conecte o OpenOCD ao SweRVolf.

- `openocd -f $SWERVOLF_ROOT/data/swervolf_nexys_debug.cfg`

```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ openocd -f $SWERVOLF_ROOT/data/swervolf_nexys_debug.cfg
Open On-Chip Debugger 0.11.0-rc1+dev-01535-g3651cbdfd (2021-01-30-12:10)
Licensed under GNU GPL v2
For bug reports, read
  http://openocd.org/doc/doxygen/bugs.html
DEPRECATED! use 'adapter driver' not 'interface'
DEPRECATED! use 'adapter speed' not 'adapter_khz'
Info : ftdi: if you experience problems at higher adapter clocks, try the command "ftdi_tdo_sample_edge falling"
Info : clock speed 10000 kHz
Info : JTAG tap: riscv.cpu tap/device found: 0x13631093 (mfg: 0x049 (Xilinx), part: 0x3631, ver: 0x1)
Info : datacount=2 progbufsize=0
Warn : We won't be able to execute fence instructions on this target. Memory may not always appear consistent. (progbufsize=
0, impebreak=0)
Info : Examined RISC-V core; found 1 harts
Info : hart 0: XLEN=32, misa=0x40001104
Info : starting gdb server for riscv.cpu on 3333
Info : Listening on port 3333 for gdb connections
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
```

**Figura 46. OpenOCD conectado**

**Etapla 13.** Abra um terceiro terminal usando "CTRL + SHIFT + T" e conecte-se à sessão de depuração por meio do OpenOCD usando o seguinte comando:

- `telnet localhost 4444`

```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ telnet localhost 4444
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Open On-Chip Debugger
> █
```

**Figura 47. Ligação telnet**

O OpenOCD suporta o carregamento de ficheiros de programa ELF executando `load_image /path/to/file.elf`. Lembre-se de que o caminho é relativo ao diretório de onde o OpenOCD foi iniciado.

➤ `load_image zephyr/samples/my_first_app/build/zephyr/zephyr.elf`

```
> load_image zephyr/samples/my_first_app/build/zephyr/zephyr.elf
14672 bytes written at address 0x00000000
downloaded 14672 bytes in 1.410859s (10.156 KiB/s)
> █
```

**Figura 48. Carregar o ficheiro de imagem .elf**

Depois que o programa tiver sido carregado, defina o Program Counter com o endereço zero usando o seguinte comando:

➤ `reg pc 0`

```
> reg pc 0
pc (/32): 0x00000000
> █
```

**Figura 49. Colocar o program counter a zero**

Agora, inicie o programa usando este comando:

➤ `resume`

```
> resume
> █
```

**Figura 50. Iniciar o programa**

O LED da placa começará a piscar.

**Etapa 14.** Abra uma nova guia de terminal usando "CTRL + SHIFT + T" e abra o PuTTY usando o seguinte comando:

➤ `sudo putty`

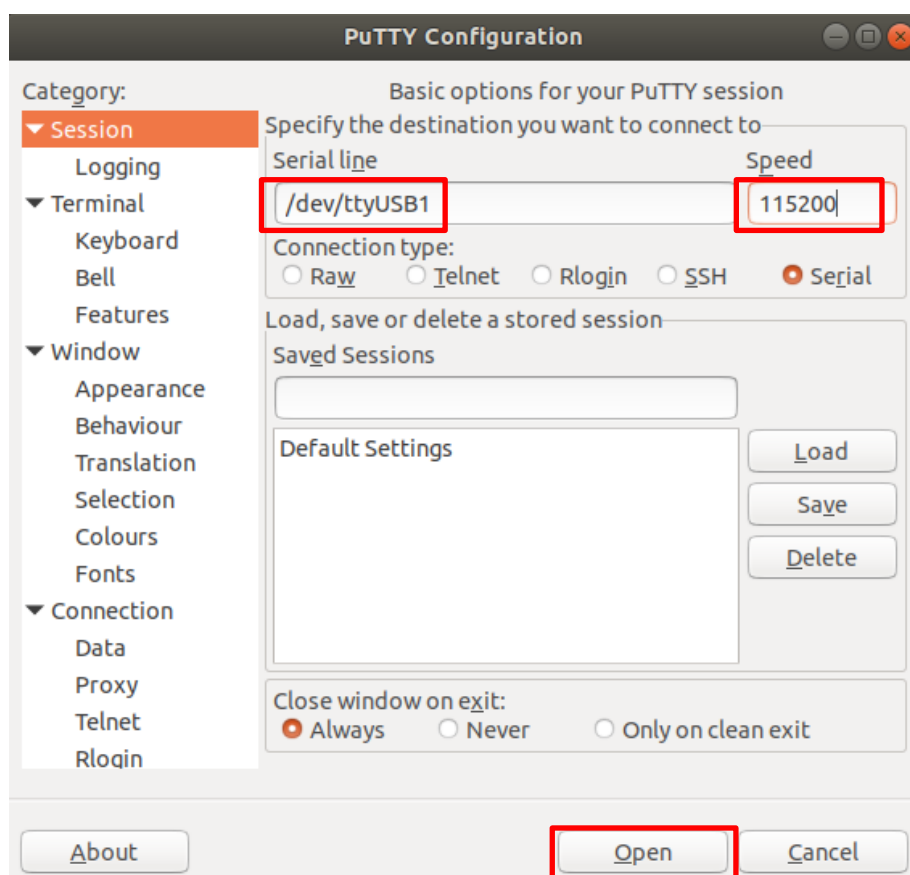
```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ sudo putty
(putty:3263): Gtk-CRITICAL **: 12:18:45.869: gtk_box_gadget_distribute: assertion 'size >= 0' failed in GtkScrollbar
```

**Figura 51. Abrir o PuTTY**

Usará o PuTTY aqui como uma consola de terminal série para a placa Nexys A7.

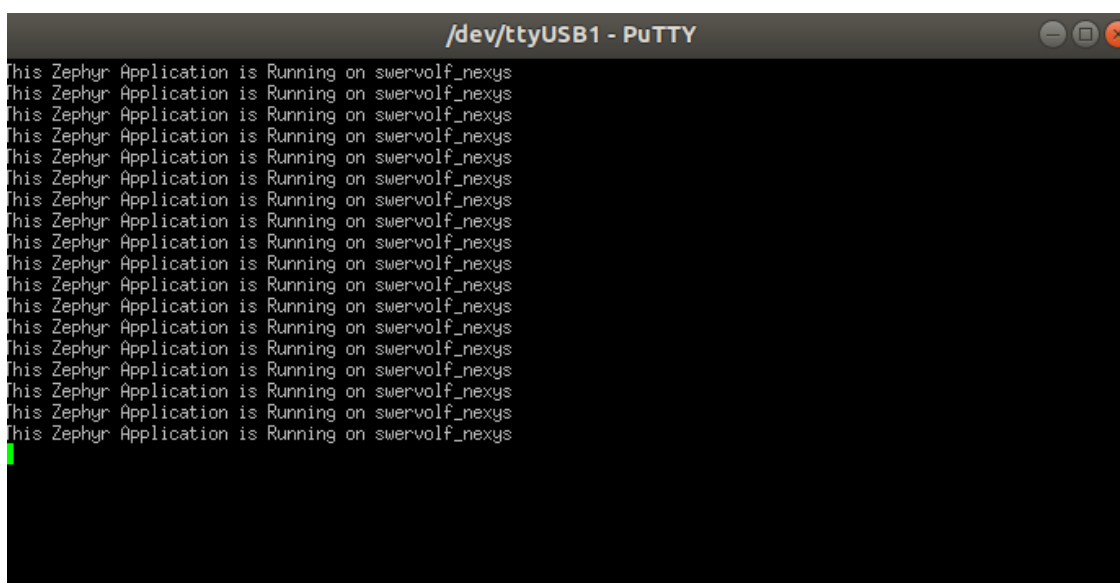
**Etapa 15.** Defina a seguinte configuração:

Selecione o tipo de ligação como **"serial"**, insira **"/dev/ttyUSB1"** como a Serial Line e defina a velocidade como **"115200"**. Agora, clique em **"Open"** para iniciar o consola série.



**Figura 52. Configuração do PuTTY**

No Consola Série, pode ver o output da execução do programa **"my\_first\_app"** (Figura 53).



**Figura 53. Saída do consola série**



**Note:** se não for possível abrir uma consola série, use a linha série `"/dev/ttyUSB0"`.

Como pode ver no texto de saída no consola série, a aplicação zephyr está sendo executada no SweRVolf Nexys.