



THE IMAGINATION UNIVERSITY PROGRAMME

RVfpga Lab 1

Programação em C

1. INTRODUÇÃO

A maioria dos programas de computador são escritos numa linguagem de alto-nível como o C. Este laboratório mostra-lhe como criar um projeto em C no PlatformIO que pode ser executado no Sistema RVfpga. Fornecemos primeiro um tutorial sobre como criar e executar um programa em C. Depois descrevemos exercícios para que possa praticar a escrita dos seus próprios programas em C.

IMPORTANTE: Antes de iniciar os Labs RVfpga, é necessário ter concluído o Guia de Introdução ao RVfpga fornecido pelo Programa Universitário da Imagination (<https://university.imgtec.com/>).

Por exemplo, se ainda não o fez, instale as ferramentas de software (pelo menos VSCode e PlatformIO) seguindo as instruções do Guia de Introdução ao RVfpga. Além disso, certifique-se de que copiou o diretório **RVfpga** que você obteve do Programa Universitário da Imagination para a sua máquina. Vamos referir-nos ao caminho absoluto do directório onde colocou a pasta RVfpga como `[RVfpgaPath]`. A pasta `[RVfpgaPath]/RVfpga/src` contém os ficheiros fonte Verilog e SystemVerilog para o sistema RVfpga, o SoC RISC-V que usaremos e modificaremos ao longo dos laboratórios. A pasta `[RVfpgaPath]/RVfpga/Labs` contém alguns programas que serão usados nos laboratórios 1 a 20.

2. Programa C para o RVfpga

Irá completar os seguintes passos para criar e executar um programa em C no RVfpgaNexys usando o PlatformIO (lembre-se que também pode executar estes programas em simulação, usando o Verilator e o Whisper):

1. Criar um projeto RVfpga
2. Escrever um programa em C
3. Configurar o RVfpgaNexys na placa FPGA Nexys A7
4. Compilar, descarregar, e executar um programa em C

Passo 1. Criar um projeto RVfpga

Abrir o VSCode (como descrito no Guia de Introdução ao RVfpga). Se o PlatformIO não abrir automaticamente quando iniciar o VSCode, clique no ícone do PlatformIO na barra do menu da esquerda e depois clique em PIO Home → Abrir (ver Figura 1).

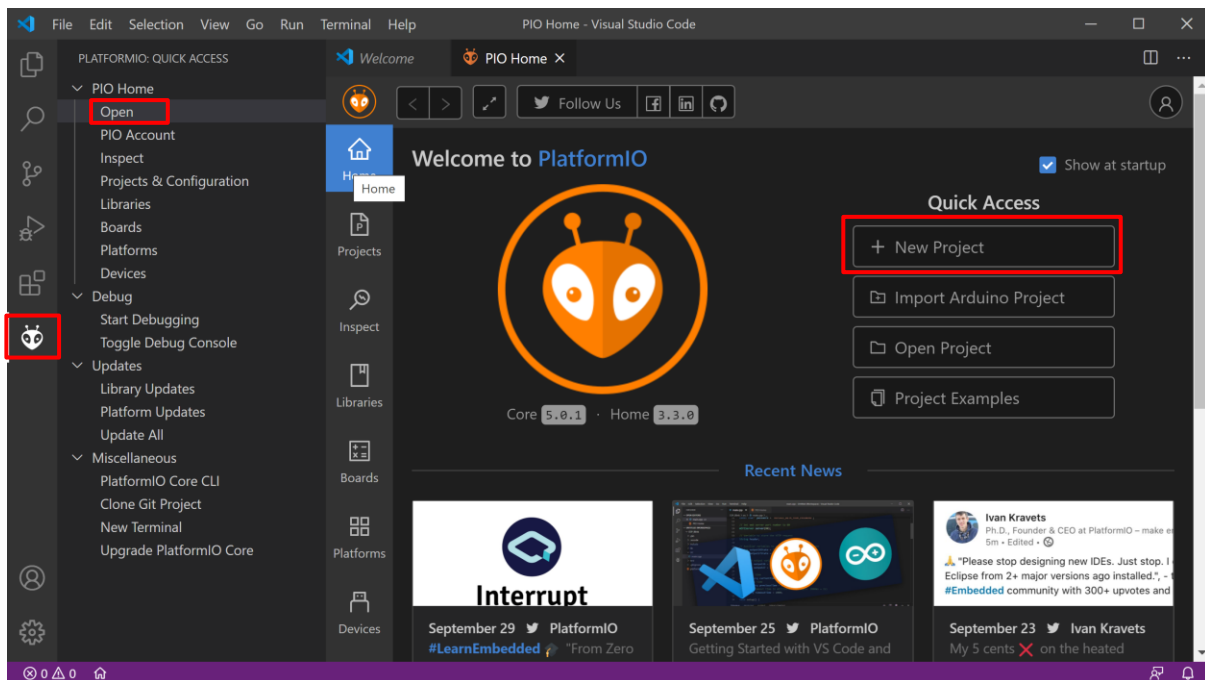


Figura 1. Abrir o PlatformIO e criar um projeto novo

Agora, na janela de boas-vindas ao PIO Home, clique em *New Project* (ver Figura 1).

Como se pode ver na Figura 2, nomear o projeto1 e escolher a placa RVfpga: Digilent Nexys A7 (comece a escrever RVfpga e a placa aparecerá como uma opção). Deixe a plataforma como WD-firmware. O WD-firmware é o firmware da Western Digital, que inclui o Freedom-E SDK gcc e gdb, bem como o PSP e BSP (Processor Support Package e Board Support Package) que utilizamos nestes laboratórios. Desmarque *Use default location* e coloque o seu projeto em:

`[RVfpgaPath]/RVfpga/Labs/Lab01`

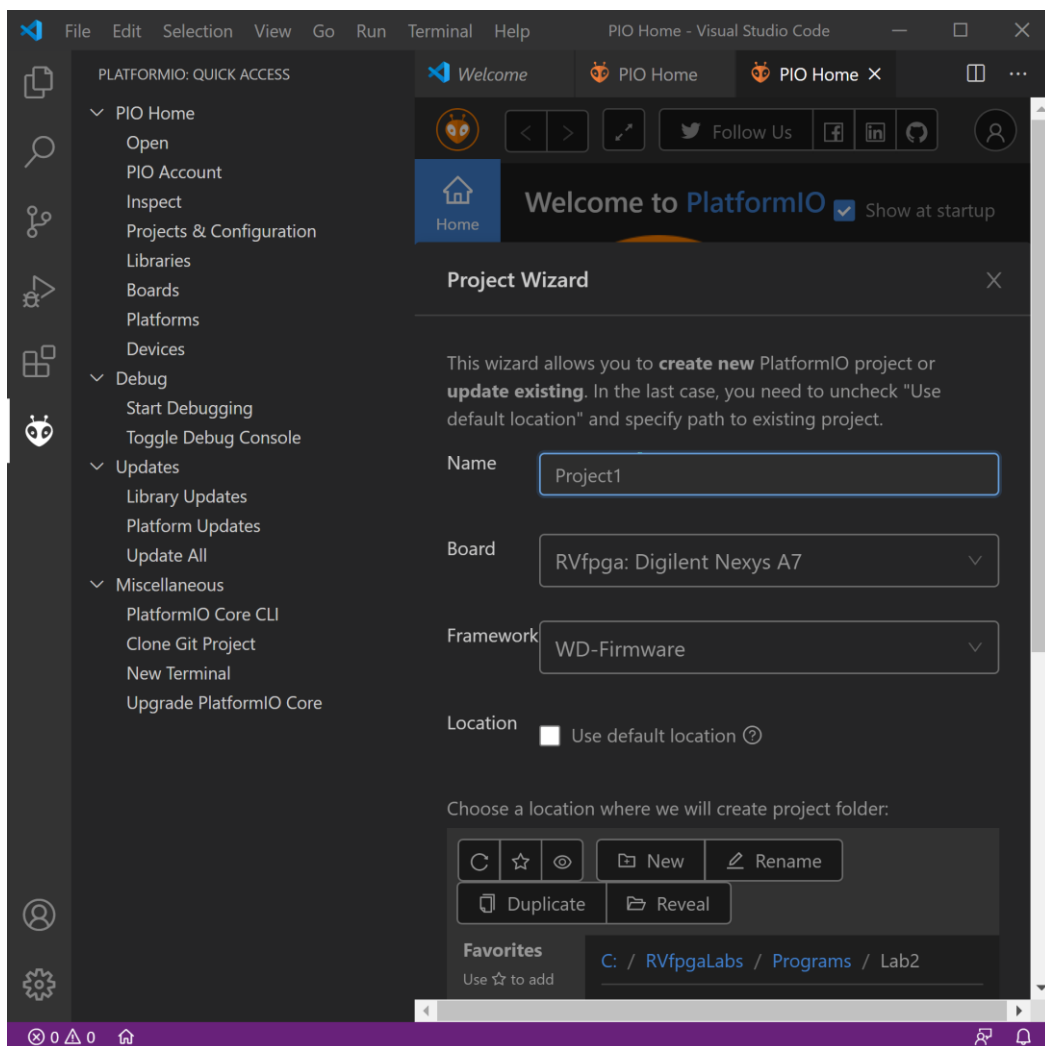


Figura 2. Nomear o projeto e selecionar a placa e a pasta do projeto

Depois clique em Terminar (Finish) na parte inferior da janela (ver Figura 3).

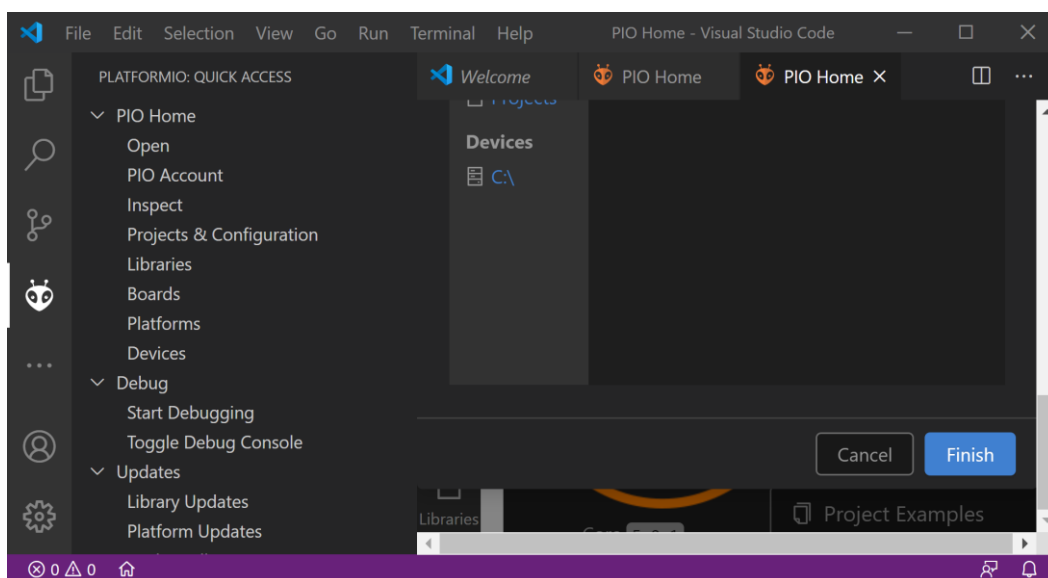


Figura 3. Terminar a criação do projeto

No painel Explorer à esquerda (que poderá ter de expandir), faça duplo clique no `platformio.ini` para o abrir (ver Figura 4). Este é o ficheiro de inicialização da PlatformIO.

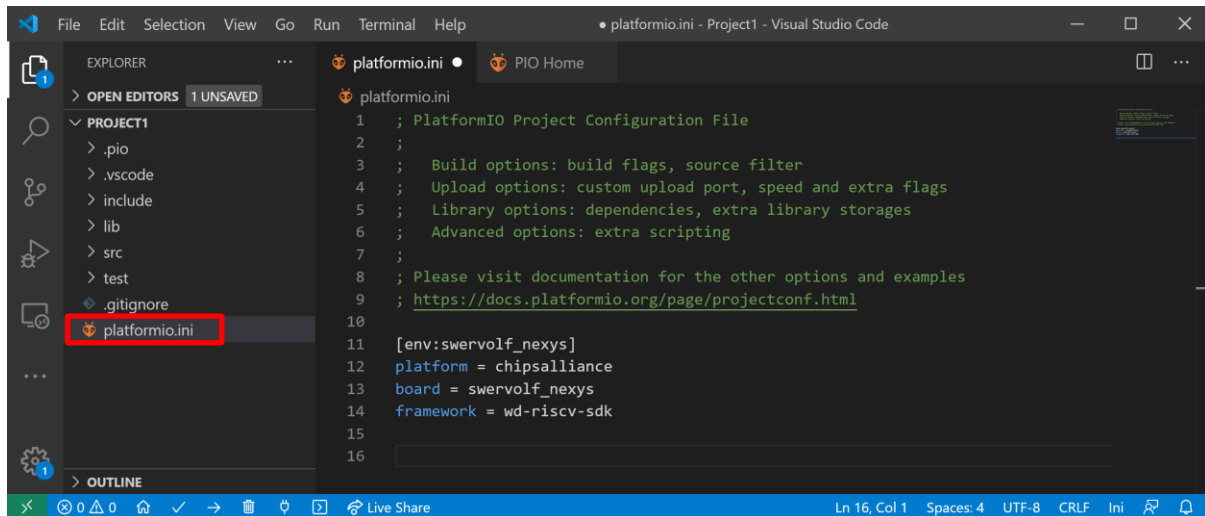


Figura 4. Ficheiro de inicialização do PlatformIO: platformio.ini

Adicione a seguinte linha ao ficheiro `platformio.ini`, como mostrado na Figura 5:

```
board_build.bitstream_file =  
[RVfpgaPath]/RVfpga/Labs/Lab1/Project1/Project1.runs/impl_1/rvfpganexys.bit
```

(Lembre-se de substituir `[RVfpgaPath]` com a localização desta pasta na sua máquina.) Esta linha indica onde o PlatformIO deve encontrar o ficheiro binário (.bit) ou *bitstream* para configurar a FPGA. O caminho acima é a localização do bitstream que criou no Laboratório 1. (Se não completou o Lab 1, pode utilizar o *bitstream* RVfpgaNexys distribuído com o Guia de Iniciação em: `[RVfpgaPath]/RVfpga/src/rvfpganexys.bit`.) Prima Ctrl-S para guardar o ficheiro `platformio.ini`.

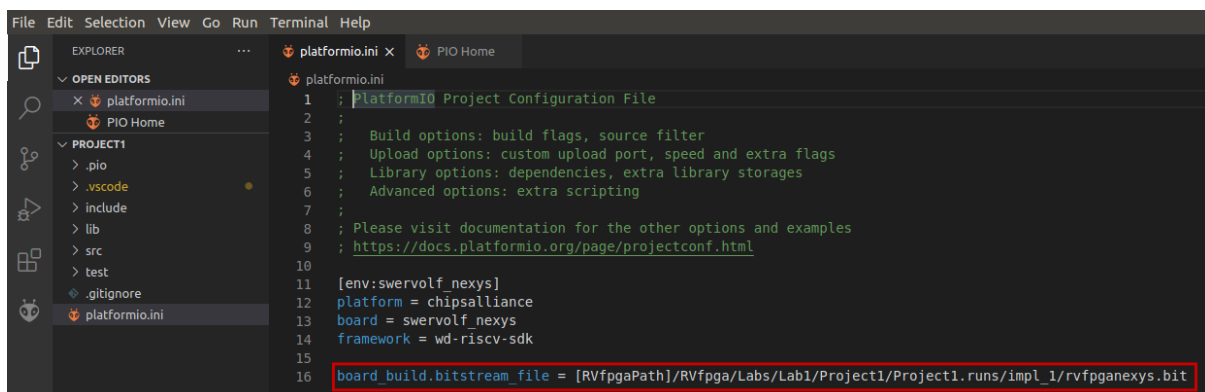


Figura 5. Adicionar localização do ficheiro bitstream do RVfpgaNexys (rvfpganexys.bit)

Lembre-se que um ficheiro `platformio.ini` mais completo foi utilizado nos exemplos utilizados no Guia de Iniciação. Se quiser usar qualquer funcionalidade que exija comandos extra (como o caminho para o simulador Verilator, a configuração da consola série, a ferramenta de depuração Whisper, etc.), pode usar o `platformio.ini` a partir desses exemplos.

Passo 2. Escrever um programa em C

Agora irá escrever um programa em C. Clique em File → New File (ver Figura 6)

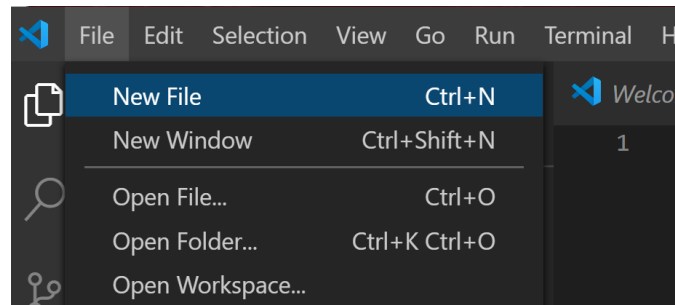


Figura 6. Adicionar ficheiro ao projeto

Abrir-se-á uma janela em branco. Escreva (ou copie/colar) o seguinte programa em C nessa janela (ver Figura 7). Este programa mostra o valor dos interruptores nos LEDs.

```
// Endereços de E/S com memória mapeada
#define GPIO_SWs      0x80001400
#define GPIO_LEDs     0x80001404
#define GPIO_INOUT    0x80001408

#define READ_GPIO(dir) (*(volatile unsigned *)dir)
#define WRITE_GPIO(dir, value) { (*(volatile unsigned *)dir) = (value); }

int main ( void )
{
    int En_Value=0xFFFF, switches_value;

    WRITE_GPIO(GPIO_INOUT, En_Value);

    while (1) {
        switches_value = READ_GPIO(GPIO_SWs); // valor lido nos interruptores
        switches_value = switches_value >> 16; // deslocar para 16 bits
        inferiores
        WRITE_GPIO(GPIO_LEDs, switches_value); // exibir o valor dos
        interruptores nos LEDs
    }

    return(0);
}
```

Este programa está também disponível no seguinte ficheiro para sua conveniência:

[RVfpgaPath]/RVfpga/Labs/Lab01/DisplaySwitches.c

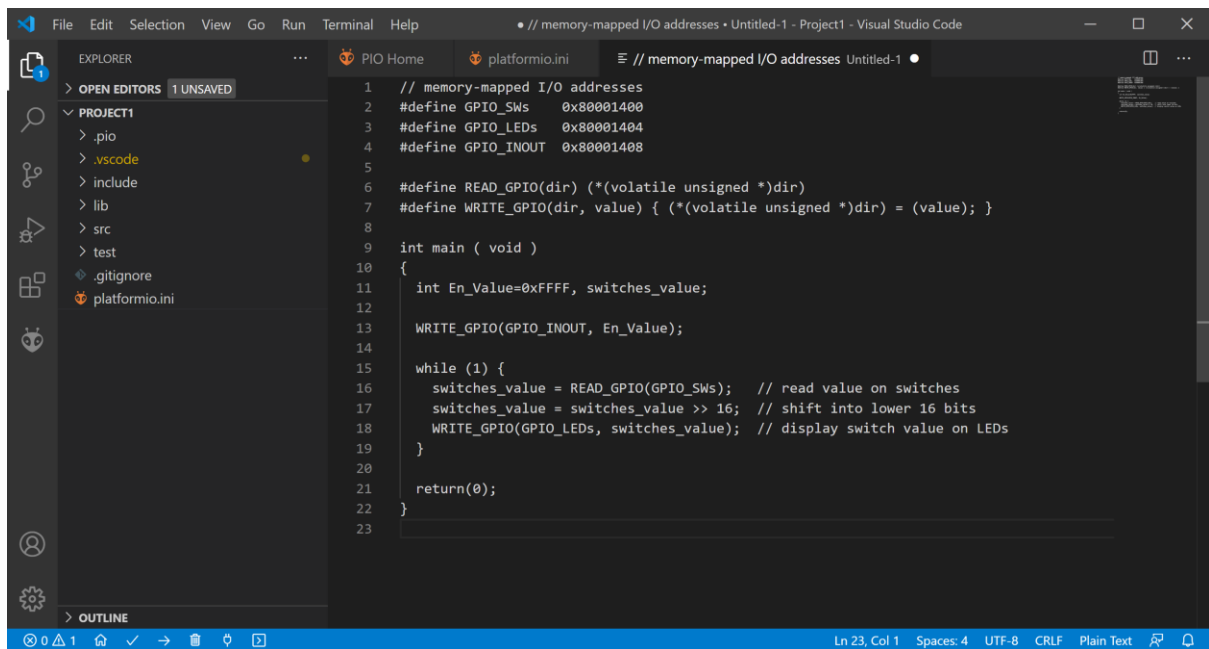


Figura 7. Introduzir programa em C

Depois de introduzir o programa no painel, prima Ctrl-s para guardar o ficheiro. Nomeie DisplaySwitches.c e guarde-o na pasta `src` de `Project1` (ver Figura 8).

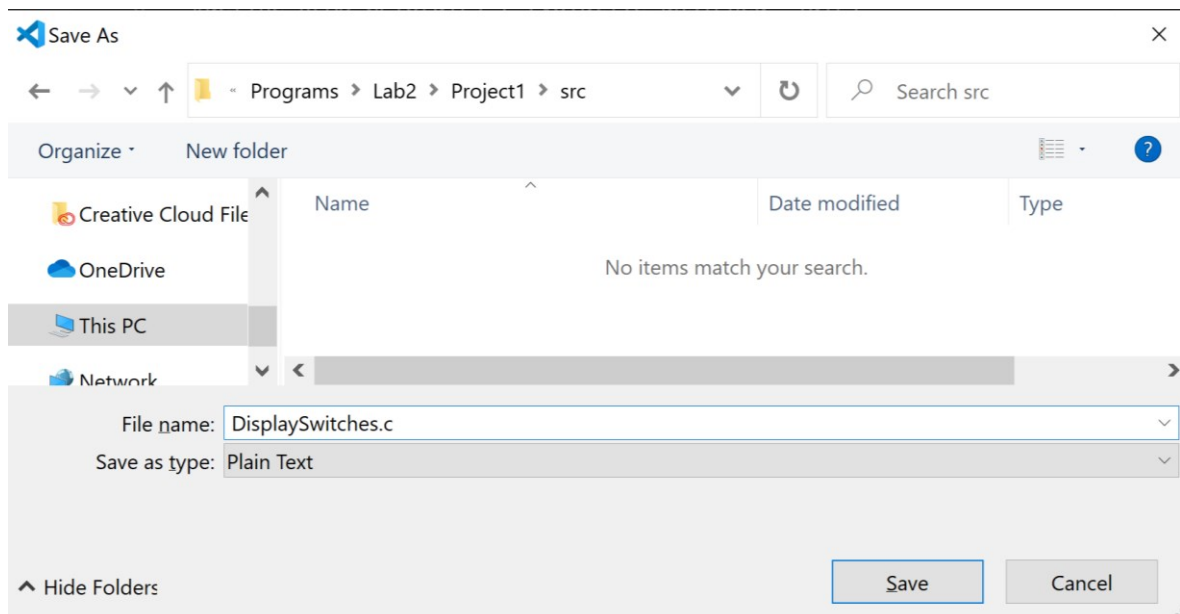


Figura 8. Salvar ficheiro como DisplaySwitches.c

Este programa define em primeiro lugar os endereços dos registos de I/O memorizados ligados aos LEDs e interruptores na placa FPGA Nexys A7 usando as seguintes linhas:

```
#define GPIO_SWs    0x80001400
#define GPIO_LEDS   0x80001404
#define GPIO_INOUT  0x80001408
```

O valor dos interruptores é encontrado lendo o registo mapeado para o endereço 0x80001400 e os valores são exibidos nos LEDs escrevendo o registo mapeado para o endereço 0x80001404. Os valores dos interruptores encontram-se na metade superior do registo, e os LEDs na metade inferior.

O registo GPIO_INOUT define se um bit de E/S de uso geral (GPIO) é uma entrada ou uma saída. Os 16 pinos GPIO menos significativos, 15:0, estão ligados aos 16 LEDs da placa Nexys A7. Os 16 pinos GPIO mais significativos, 31:16, são para os 16 interruptores da placa. Um 0 indica uma entrada e um 1 indica uma saída. Assim, o registo GPIO_INOUT é escrito com 0xFFFF para que os interruptores sejam entradas para RVfpgaNexys e os LEDs sejam saídas acionadas pelo RVfpgaNexys.

A Figura 9 mostra a localização física dos LEDs e interruptores na placa Nexys A7 FPGA, bem como o conector USB, interruptores, botões de pressão e mostradores de 7 segmentos.

Note que no Lab 6 descrevemos em pormenor as funcionalidades do GPIO e o hardware GPIO do RVfpgaNexys. Também discutimos como usar os outros periféricos da placa, como botões e mostradores de 7 segmentos, nos laboratórios seguintes (Labs 6-10).

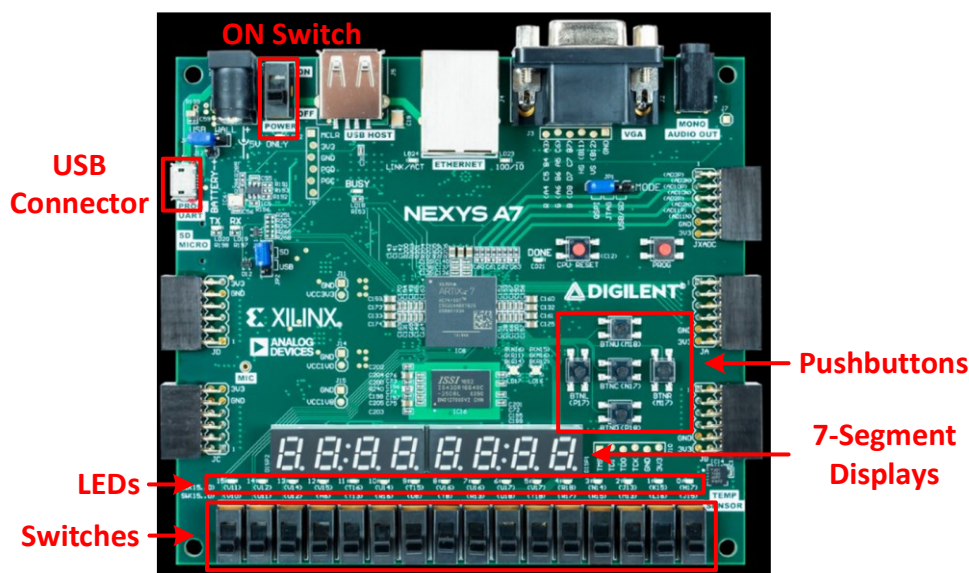


Figura 9. Interfaces de E/S da placa Nexys A7 FPGA da Digilent
(Figura da placa tirada de <https://reference.digilentinc.com/>)

Depois de definir os endereços, de E/S mapeados em memória, dos LEDs e dos interruptores, o programa faz o seguinte:

1. Define os 16 pinos de GPIO mais significativos (que estão ligados aos interruptores) como entradas, definindo a metade superior do registo GPIO_INOUT como 0's e define os 16 pinos de GPIO menos significativos (que estão ligados aos LEDs) como saídas, definindo a metade inferior do registo GPIO_INOUT como 1's, executando o seguinte código:

```
int En_Value=0xFFFF;

WRITE_GPIO(GPIO_INOUT, En_Value);
```

2. Lê repetidamente o valor dos interruptores e escreve esse valor nos LEDs, executando o código abaixo. Lembre-se de que o valor dos interruptores é lido na metade superior do registo de E/S mapeado na memória, pelo que o valor tem de ser deslocado 16 bits para a direita antes de ser escrito no registo de E/S mapeado na memória fisicamente ligado aos LEDs.

```
while (1) {
    switches_value = READ_GPIO(GPIO_SWs);    // lê o valor dos SWs
    switches_value = switches_value >> 16;    // desloca para os 16 LSbits
    WRITE_GPIO(GPIO_LEDs, switches_value);    // mostra valor dos SW nos LED
}
```

As macros `READ_GPIO` e `WRITE_GPIO` leem ou escrevem, respectivamente, um valor no endereço de E/S mapeado na memória especificado.

Passo 3. Configurar o RVfpgaNexys na placa Nexys A7 FPGA

Agora vai configurar o RVfpgaNexys na placa FPGA Nexys A7. Clique no ícone PlatformIO na faixa de opções do menu esquerdo e, em seguida, expanda *Project Tasks* → *env:swervolf_nexys* → *Platform* e clique em *Upload Bitstream*, como mostra a Figura 10.

Nota: se estiver a utilizar **Windows** e ainda não substituiu o controlador da placa FPGA Nexys A7, faça-o seguindo as instruções do Apêndice B do Guia de Iniciação.

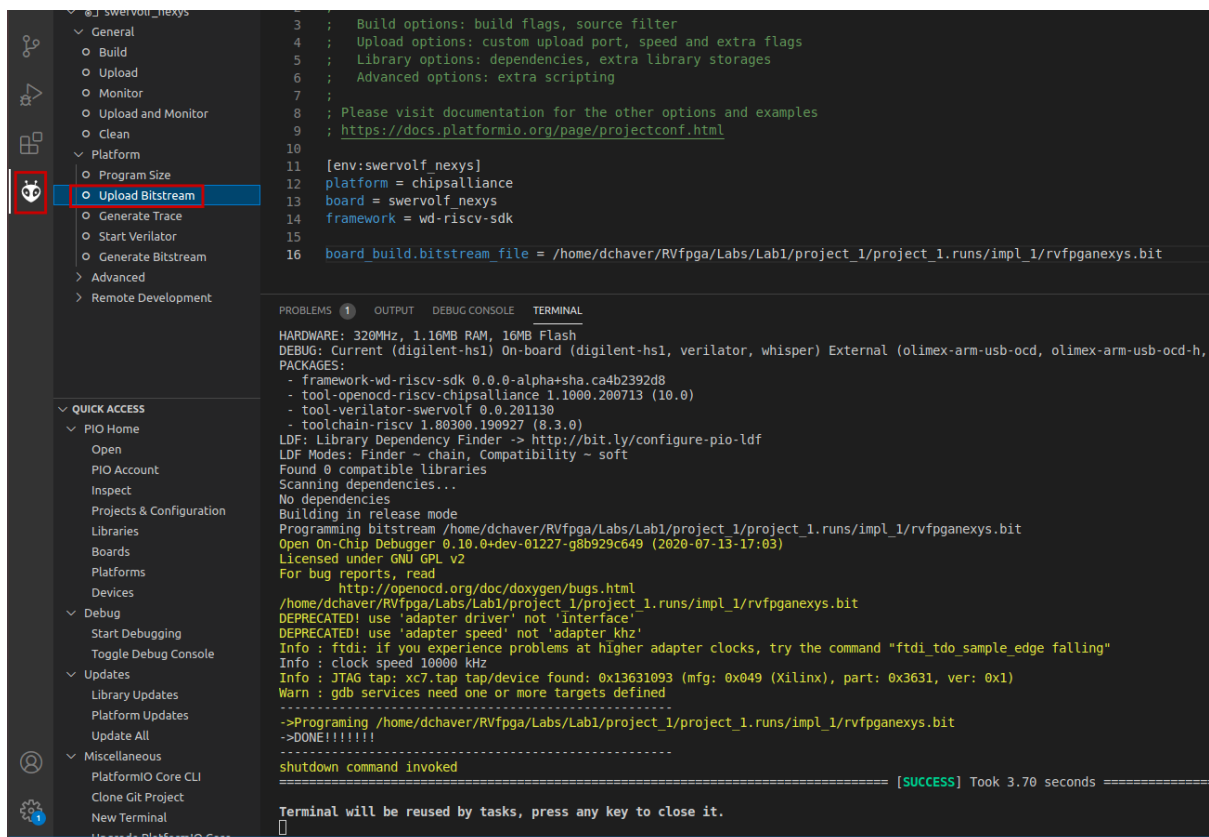



Figura 10. Configurar o RVfpgaNexys na placa Nexys A7 FPGA usando o PlatformIO

Como alternativa, pode configurar o RVfpgaNexys a partir de uma janela de terminal PlatformIO, como mostrado em Figura 11. No PlatformIO clique em: *New Terminal button* (

 na parte inferior da janela do PlatformIO e, em seguida, escreva (ou copie) o seguinte para o terminal do PlatformIO:

```
pio run -t program_fpga
```

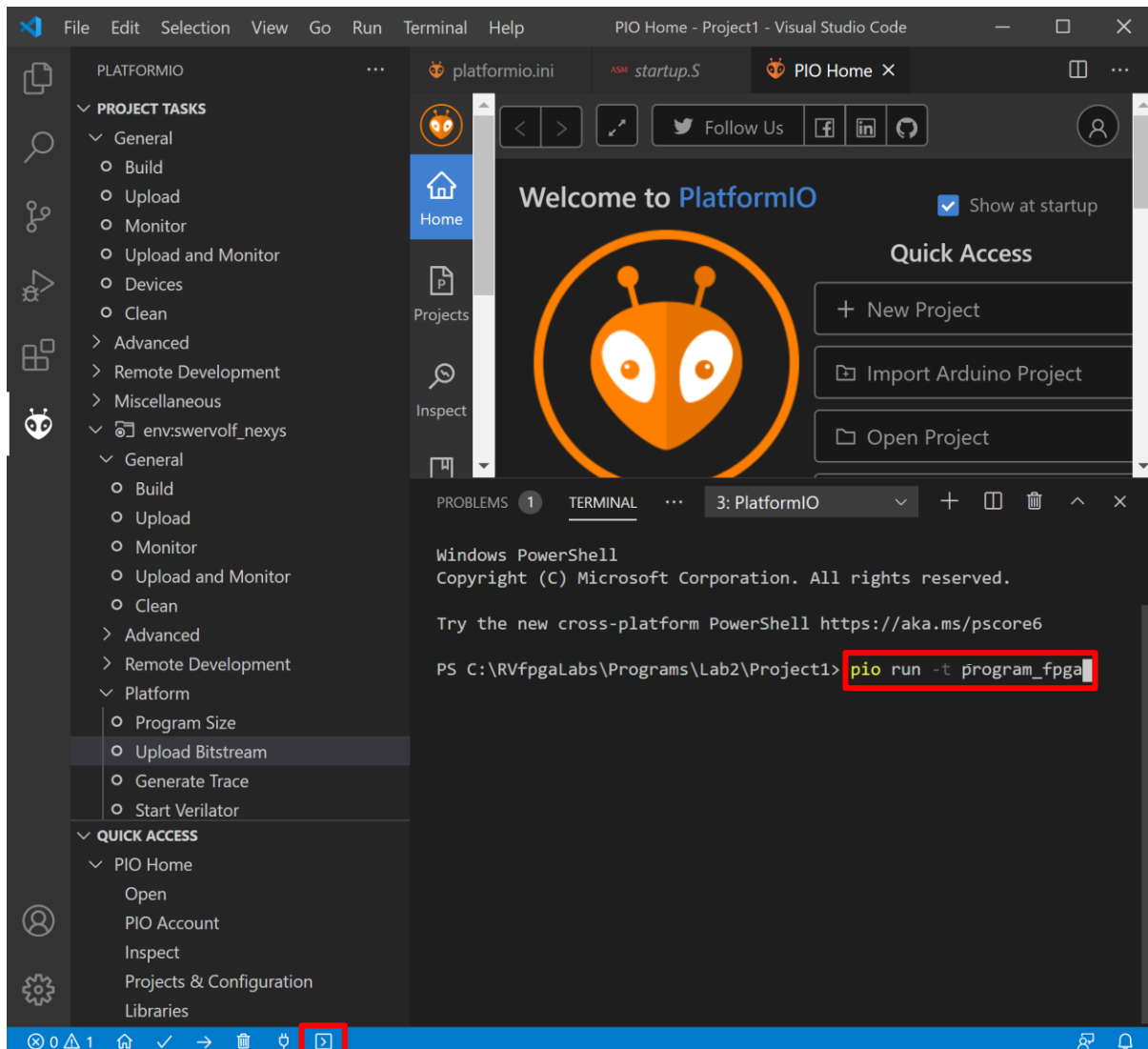


Figura 11. Configurar o RVfpgaNexys na placa Nexys A7 FPGA utilizando o terminal PlatformIO

Passo 4. Compilar, descarregar, e executar o programa C

Agora que o RVfpgaNexys está a correr na placa, irá compilar o seu programa, descarregá-lo para o RVfpgaNexys e executá-lo/depurá-lo. Se o VSCode ainda não estiver aberto, abra-o. O seu último projeto, Project1, deve abrir-se automaticamente. Caso contrário, certifique-se de que a extensão PlatformIO está aberta e clique em *File* → *Open Folder* e selecione (mas não abra) Project1, que criou anteriormente neste laboratório.

Clique no botão *Run* na faixa de opções do menu da esquerda (ver Figura 12).

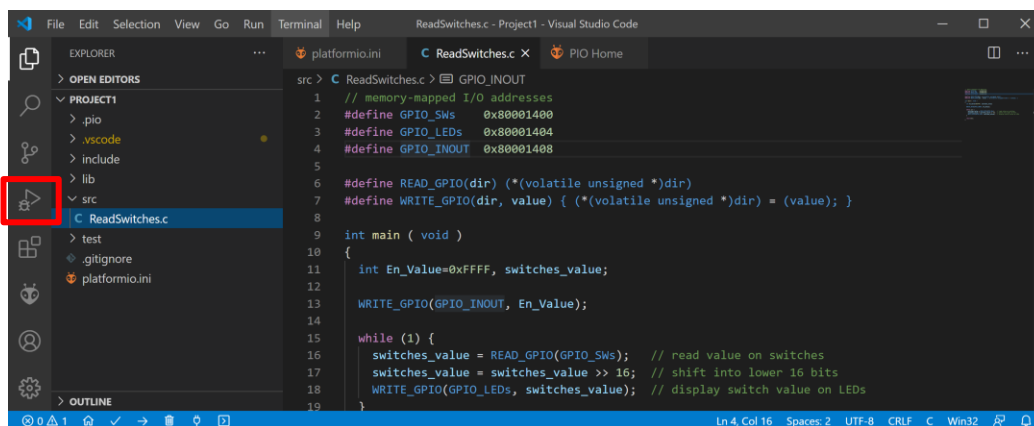


Figura 12. Executar programa no RVfpgaNexys

Agora clique no botão *Start Debugging* (ver Figura 13).

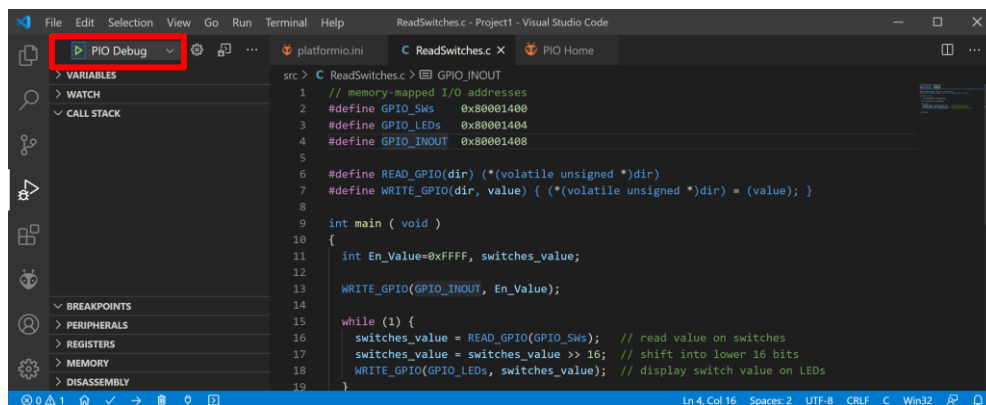


Figura 13. Começar a correr e a depurar o programa

O programa será compilado e depois descarregado para o RVfpgaNexys, que está a correr na FPGA da placa Nexys A7. (Note que pode aparecer no terminal uma mensagem sobre um ficheiro sys/cdefs.h em falta - mas o programa continua a funcionar corretamente). Agora pode começar a executar e a depurar o programa (ver Figura 14).

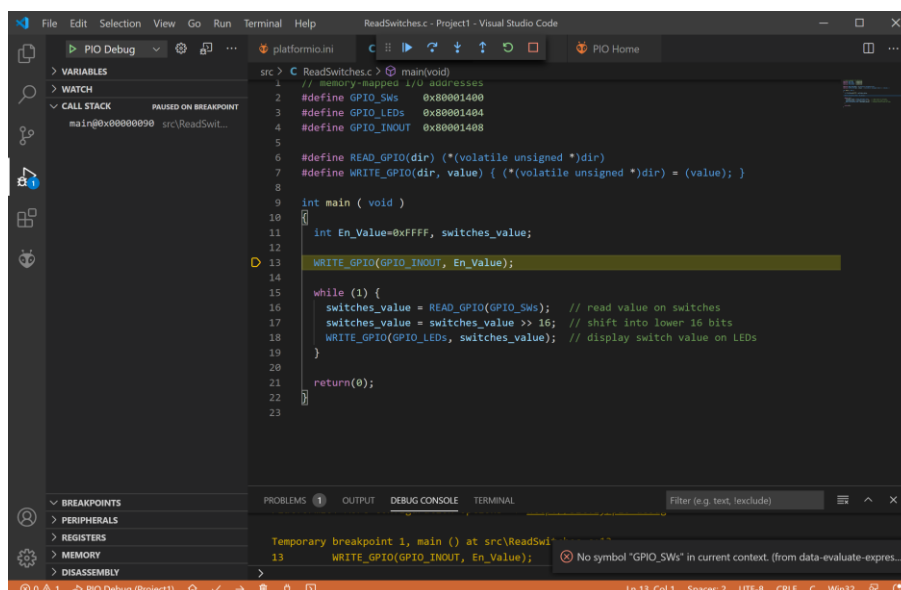


Figura 14. Programa em execução na RVfpgaNexys

Tal como descrito no Guia de Iniciação do RVfpga, para controlar a sua sessão de depuração, utilize a barra de ferramentas de depuração que aparece perto do topo do editor (ver Figura 15). As opções são as seguintes:

1. **Continue** executa o programa até ao próximo ponto de paragem.
2. **Breakpoints** pode ser adicionados clicando à esquerda do número da linha no editor.
3. **Step Over** executa a linha atual e depois pára.
4. **Step Into** executa a linha atual e, se a linha atual incluir uma chamada de função, salta para essa função e pára.
5. **Step Out** executa todo o código na função em que se encontra e pára quando essa função retorna.
6. **Restart** reinicia a sessão de depuração a partir do início do programa.
7. **Stop** pára a sessão de depuração e regressa ao modo de edição normal. Note que quando prime o botão Stop, **o programa continua a ser executado** no RVfpgaNexys, mas a sessão de depuração termina.
8. **Pause** faz uma pausa na execução. Quando o programa está a ser executado, o botão *Continue* (Continuar) é substituído pelo botão *Pause* (Pausa).




Figura 15. Ferramentas de depuração

Na barra lateral esquerda, pode ver as opções do Depurador. 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 em execução, a função de chamada (se existir) e a localização da instrução atual na memória.
- **Breakpoints:** mostra todos os pontos de paragem definidos e realça o seu número de linha. Os pontos de paragem podem ser geridos nesta secção. Os pontos de interrupção também podem ser temporariamente desativados sem serem removidos, bastando para isso ativar a caixa de verificação.
- **Peripherals:** mostra o estado dos registos dos periféricos ligados à memória do dispositivo.
- **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 nível superior, como o C, isto permite-lhe ver o Assembly para depurar as instruções uma a uma.

Por exemplo, clique à esquerda da linha 18 para definir um ponto de interrupção imediatamente antes de o valor dos interruptores ser escrito nos LEDs, como mostrado em

Figura 16. Agora, execute o programa clicando no botão *Continue*  (ou premindo F5). O programa continuará até atingir o ponto de interrupção definido. (Para remover um ponto de interrupção, clique no ponto de interrupção existente, mesmo à esquerda do número da linha.)

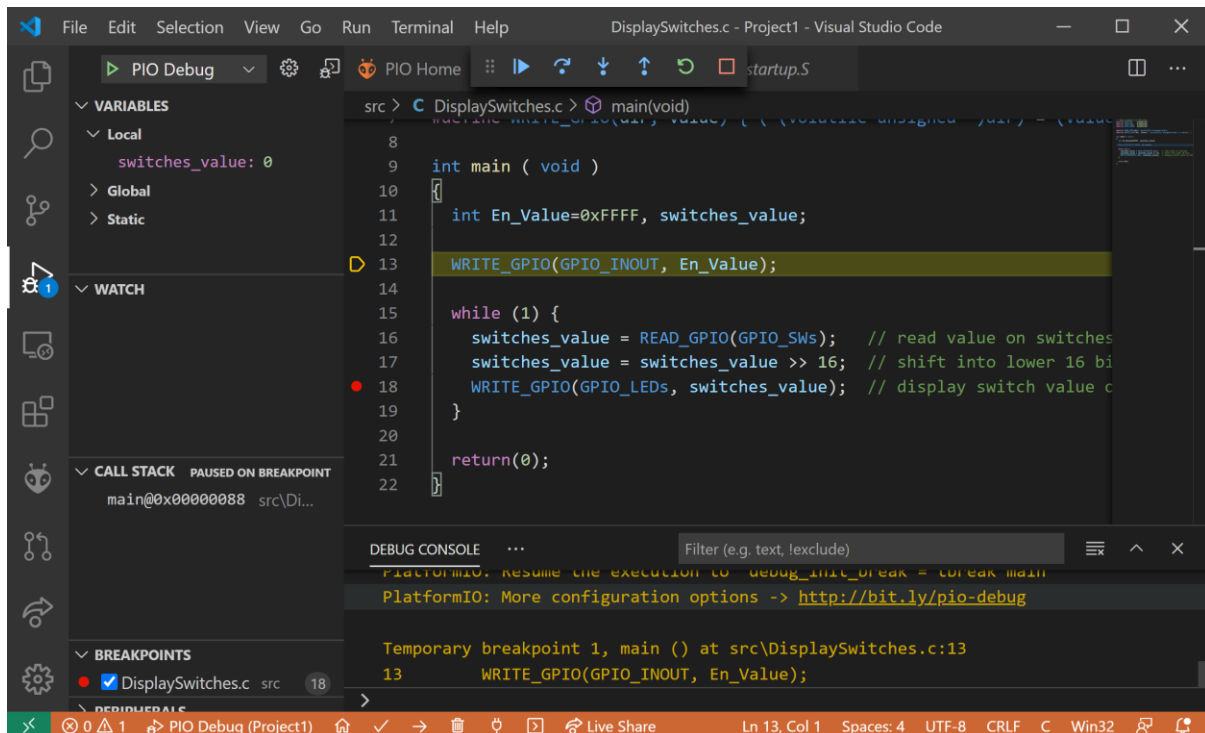


Figura 16. Definir um ponto de interrupção antes de exibir o valor nos LEDs

Depois de atingir o ponto de interrupção, expanda a secção *Variables* no painel esquerdo e veja o valor da variável `switches_value`, como mostrado na Figura 17. O valor dos interruptores é 1029 = 0x405 (em binário 0000_0100_0000_0101), que gera o padrão:

`Switches[15:0] = OFF-OFF-OFF-OFF-OFF-ON-OFF-OFF-OFF-OFF-OFF-OFF-OFF-ON-OFF-ON`

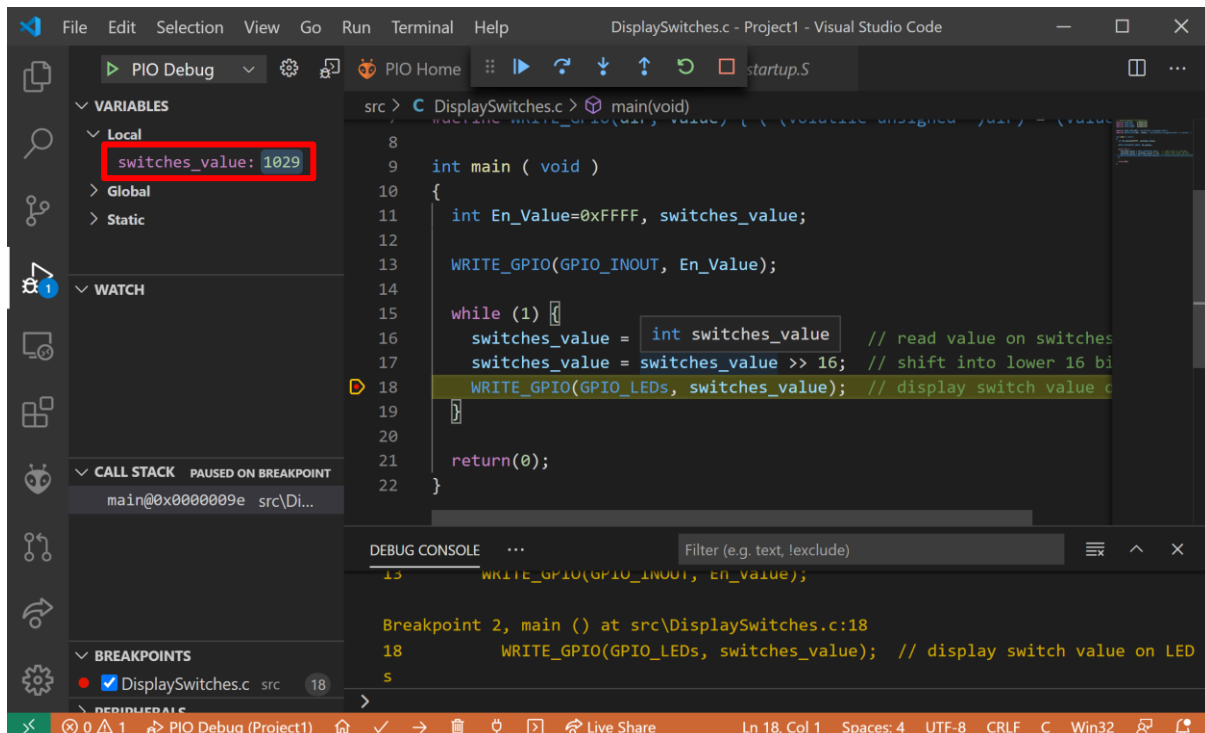


Figura 17. Visualização dos valores das variáveis

Também pode ver o código Assembly RISC-V gerado a partir do programa C. Para isso, clique em *DISASSEMBLY* → *Switch to assembly*, como mostra a Figura 18.

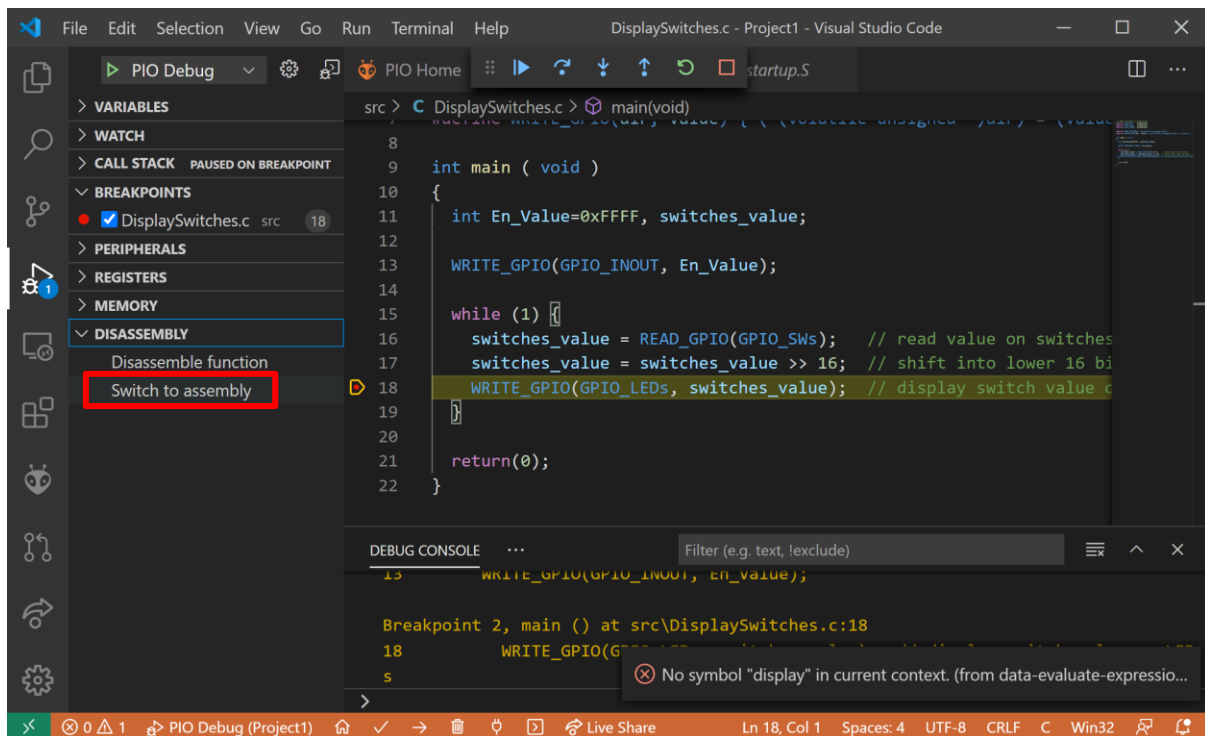


Figura 18. Visualização do código de Assembly RISC-V

Agora o conjunto RISC-V aparece no painel de visualização, como mostrado na Figura 19. O Assembly mostra o endereço de memória da instrução, o código de máquina e o código Assembly. Como pode ver, o código C compila para uma mistura de instruções comprimidas (instruções de 16 bits) e instruções de 32 bits. Abaixo está o código Assembly comentado.

# address	# machine code	# instruction	
0x00000090:	37 17 00 80	lui a4,0x80001	# Base address for I/O
0x00000094:	c1 67	lui a5,0x10	# a5 = 0x10000 - 1 (=0xFFFF)
0x00000096:	fd 17	addi a5,a5,-1	
0x00000098:	23 24 f7 40	sw a5,1032(a4)	# I/O direction: [0x80001408]=0xFFFF
0x0000009c:	37 17 00 80	lui a4,0x80001	# Base address for I/O (redundant)
0x000000a0:	83 27 07 40	lw a5,1024(a4)	# Read switches: a5 = [0x80001400]
0x000000a4:	c1 87	srai a5,a5,0x10	# Move switch value to lower 16 bits
0x000000a6:	23 22 f7 40	sw a5,1028(a4)	# Write LEDs: [0x80001404] = a5
0x000000aa:	cd bf	j 0x9c <main+12>	# repeat

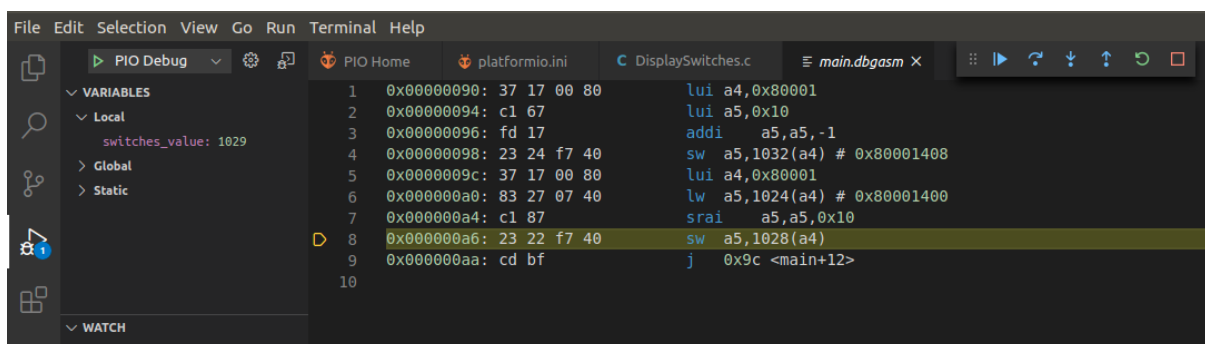



Figura 19. Ver código Assembly RISC-V

Clique em DISASSEMBLY → *Switch to code* para ver o código C novamente.

Depois de terminar a execução/depuração do programa, pare a sessão de depuração

premindo o botão Stop  (ou Shift - F5) e voltar à janela do

Explorer, clicando em  no topo da barra lateral mais à esquerda. Repare que o **programa continua a ser executado** no RVfpgaNexys - apenas a sessão de depuração termina. Feche o projeto clicando em *File* → *Close Folder* na barra de menu superior.

3. Utilização do printf e do monitor série

A utilização de instruções de impressão num programa é uma forma útil de acompanhar o progresso do programa ou fornecer feedback aos utilizadores, por exemplo, resultados de cálculos. Recorde-se do Guia de Iniciação do RVfpga (exemplo *HelloWorld_C-Lang* na Secção 6.F) que pode utilizar a função `printfNexys`, té semelhante à função `printf` em programas C típicos. Para tal, é necessário utilizar o PSP e o BSP (pacote de suporte do processador e pacote de suporte da placa) da Western Digital que fornecem funções comuns para um determinado processador e placa, neste caso o núcleo SweRV EH1 e a placa FPGA Nexys A7.

Crie um projeto PlatformIO chamado *PrintfExample* na pasta *[RVfpgaPath]/RVfpga/Labs/Lab1*. Adicione o seguinte programa a esse projeto (ver Figura 20). Nomeie o ficheiro do programa *PrintfExample.c*.

O programa também está disponível aqui para sua conveniência:

[RVfpgaPath]/RVfpga/Labs/Lab01/PrintfExample.c

```
#if defined(D_NEXYS_A7)
    #include <bsp_printf.h>
    #include <bsp_mem_map.h>
    #include <bsp_version.h>
#else
    PRE_COMPILED_MSG("no platform was defined")
#endif
#include <psp_api.h>

#define DELAY 10000000

int main(void)
{
    int i, j = 0;

    // Initialize UART
    uartInit();

    while (1) {
        printfNexys("Hello RVfpga users! Iteration: %d\n", j);
        for (i=0; i < DELAY; i++) ; // delay between printf's
        j++;
    }
}
```

```

}

src > C PrintfExample.c
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 #define DELAY 10000000
11
12 int main(void)
13 {
14     int i, j = 0;
15
16     // Initialize UART
17     uartInit();
18
19     while (1) {
20         printfNexys("Hello RVfpga users! Iteration: %d\n", j);
21         for (i=0; i < DELAY; i++) ; // delay between printf
22         j++;
23     }
24 }

```

Figura 20. PrintfExample.c

As linhas 1-8 (ver Figura 20) são os ficheiros de inclusão necessários para utilizar a função `printfNexys`. Eles são fornecidos no BSP/PSP da Western Digital. A linha 17 na Figura 20 chama a função `uartInit`; esta linha é necessária para inicializar a ligação UART utilizada para comunicar entre o RVfpgaNexys (em execução na placa Nexys A7) e o monitor série. Finalmente, o ciclo `while` escreve repetidamente no monitor serial usando a função `printfNexys` na linha 20, seguido de um atraso (line 21).

Para utilizar a função `printfNexys` e a UART, o ficheiro `platformio.ini` deve ser modificado para incluir a velocidade da UART. Adicione a seguinte linha ao arquivo `platformio.ini`, como mostrado na Figura 21:

```
monitor_speed = 115200
```


O RVfpgaNexys espera que a UART comunique a 115200 baud (símbolos/segundo), pelo que esta taxa tem de ser definida no `platformio.ini`, como se mostra em Figura 21, linha 16. Lembre-se também de adicionar a localização do seu bitfile RVfpgaNexys usando `board_build.bitstream_file = ...` (como se pode ver na Figura 21, linha 18).

```
platformio.ini
1 ; PlatformIO Project Configuration File
2 ;
3 ; Build options: build flags, source filter
4 ; Upload options: custom upload port, speed and extra flags
5 ; Library options: dependencies, extra library storages
6 ; Advanced options: extra scripting
7 ;
8 ; Please visit documentation for the other options and examples
9 ; https://docs.platformio.org/page/projectconf.html
10
11 [env:swervolf_nexys]
12 platform = chipsalliance
13 board = swervolf_nexys
14 framework = wd-riscv-sdk
15
16 monitor_speed = 115200
17
18 board_build.bitstream_file = /home/dchaver/RVfpga/Labs/Lab1/project_1/project_1.runs/impl_1/rvfpganexys.bit
```

Figura 21. Definição da velocidade da UART

LINUX: Lembre-se que, se estiver a usar Linux, antes de poder usar o monitor de série precisa de preparar o sistema adicionando o seu utilizador aos grupos `dialout`, `tty` e `uucp`, como explicado na Secção 6.F do Guia de Iniciação. Se efetuou este processo no GSG, tudo deverá funcionar; caso contrário, faça-o agora.

Agora carregue o ficheiro de bits (como descrito na Secção 2) e execute/depure o programa.

Depois de o programa começar a correr (e **só depois** de o programa começar a correr), clique no botão do monitor de série  na parte inferior da janela da PlatformIO (ver Figura 22).

Aviso: Se abrir o monitor de série antes de o programa começar a ser executado (e atingir o primeiro ponto de interrupção - temporário), a UART não funcionará corretamente.

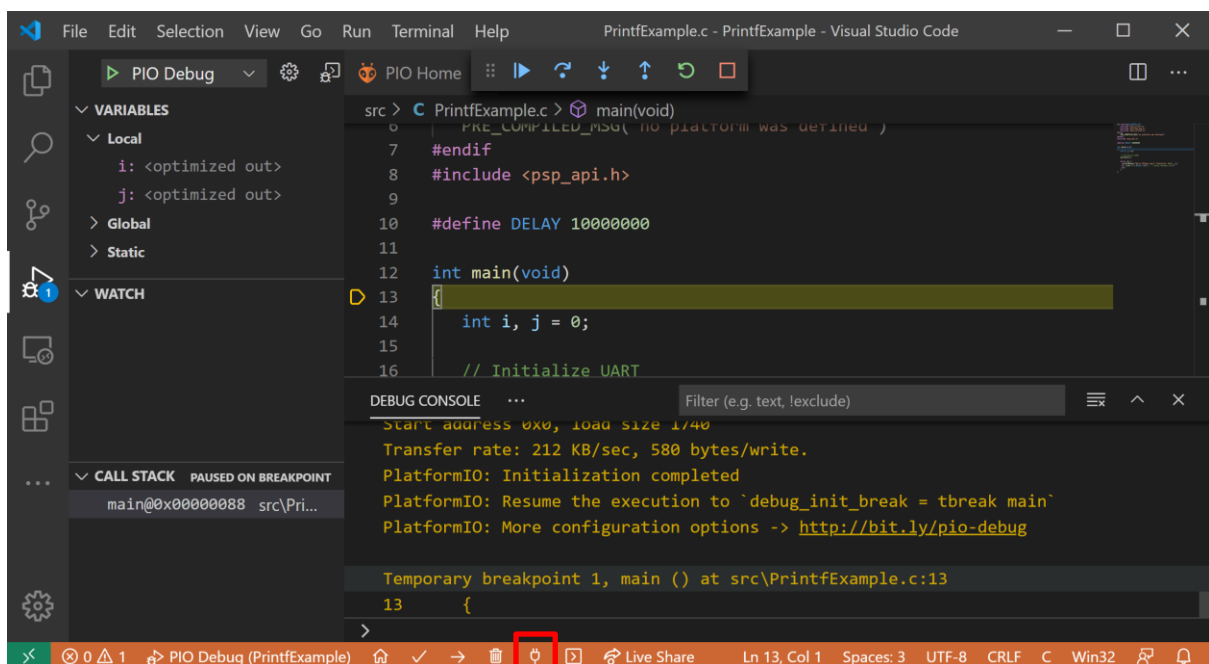


Figura 22. Início do monitor série

Em seguida, execute o programa: .

Verá a impressão da *string* (Hello RVfpga users!) seguida do número de iteração impressa repetidamente no monitor de série, como mostrado na Figura 23.

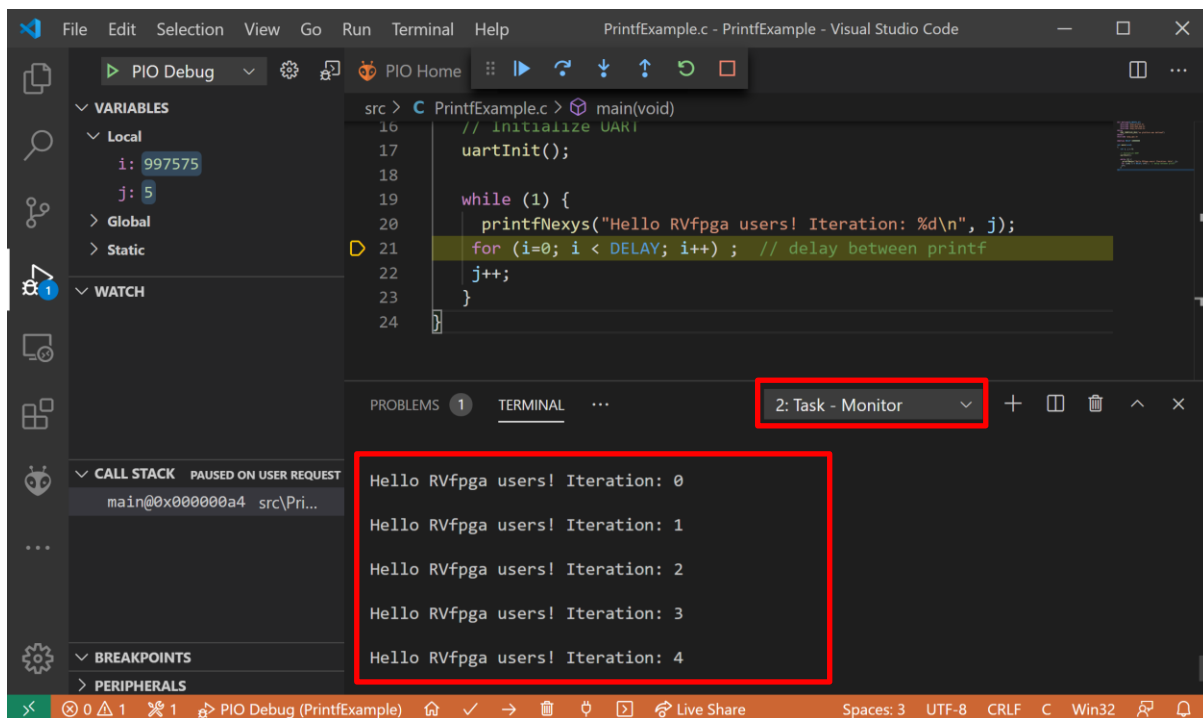


Figura 23. Resultado da função `printfNexys` no monitor série do PlatformIO

4. Exercícios

Crie os seus próprios programas em C completando os seguintes exercícios. Tenha em atenção que se deixar a placa Nexys A7 ligada ao computador e ligada, não é necessário reconfigurar o RVfpgaNexys na placa entre a execução de diferentes programas. No entanto, se desligar a placa Nexys A7, terá de recarregar o RVfpgaNexys na placa utilizando PlatformIO, conforme descrito no passo 3 da Secção 2.

Lembre-se de que pode imprimir qualquer variável utilizando a função BSP da Western Digital `printfNexys` (ver Section 3).

Lembre-se também que pode executar estes programas em simulação, utilizando o Verilator e o Whisper.

Exercício 1. Escreva um programa em C que faça piscar o valor dos interruptores nos LEDs. O valor deve ser activado e desativado de forma suficientemente lenta para que uma pessoa possa ver a intermitência. Nomeie o programa **FlashSwitchesToLEDs.c**.

Exercício 2. Escreva um programa em C que mostre o valor inverso dos interruptores nos LEDs. Por exemplo, se os interruptores forem (em binário): 0101010101010101, então os LEDs devem exibir: 1010101010101010; se os interruptores forem:

11110000111110000, então os LEDs devem exibir: 000011110000111111; e assim por diante. Nomeie o programa **DisplayInverse.c**.

Exercício 3. Escreva um programa em C que deslize um número crescente de LEDs acesos para a frente e para trás até que todos os LEDs estejam acesos. De seguida, o padrão deve repetir-se. Nomeie o programa **ScrollLEDs.c**.

O programa deve provocar o seguinte:

1. Primeiro, um LED aceso deve deslocar-se da direita para a esquerda e depois da esquerda para a direita.
2. Em seguida, dois LEDs acesos devem deslocar-se da direita para a esquerda e depois da esquerda para a direita.
3. Em seguida, três LEDs acesos devem deslocar-se da direita para a esquerda e depois da esquerda para a direita.
4. E assim por diante, até que todos os LEDs estejam acesos.
5. De seguida, o padrão deve repetir-se.

Exercício 4. Escreva um programa em C que mostre a adição sem sinal de 4 bits dos 4 bits menos significativos dos interruptores e dos 4 bits mais significativos dos interruptores. Mostre o resultado nos 4 bits menos significativos (mais à direita) dos LEDs. Dê ao programa o nome de **4bitAdd.c**. O quinto bit dos LEDs deve acender-se quando ocorre um *overflow* sem sinal (ou seja, quando o *carry out* é 1).

Exercício 5. Escreva um programa em C que encontre o maior divisor comum de dois números, *a* e *b*, de acordo com o algoritmo de Euclides. Os valores *a* e *b* devem ser variáveis definidas estaticamente no programa. Dê ao programa o nome de **GCD.c**. Aqui estão algumas informações adicionais sobre o algoritmo de Euclides:
<https://www.khanacademy.org/computing/computer-science/cryptography/modarithmetic/a/the-euclidean-algorithm>. Também pode simplesmente pesquisar no Google “Algoritmo Euclidiano” / “*Euclidean algorithm*”.

Exercício 6. Escreva um programa em C que calcule os primeiros 12 números da sequência de Fibonacci e guarde o resultado num vector finito (i.e. matriz), *V*, de comprimento 12. Esta sequência infinita de números de Fibonacci é definida como:

$$V(0)=0, \quad V(1)=1, \quad V(i)=V(i-1)+V(i-2) \quad (\text{onde } i=0,1,2,\dots)$$

Por outras palavras, o número de Fibonacci correspondente ao elemento *i* é a soma dos dois números de Fibonacci anteriores da série. A Tabela 1 mostra os números de Fibonacci para *i* = 0 a 8.

Tabela 1. Série de Fibonacci

<i>i</i>	0	1	2	3	4	5	6	7	8
<i>V</i>	0	1	1	2	3	5	8	13	21

A dimensão do vector, *N*, deve ser definida no programa como uma constante. Nomear o programa **Fibonacci.c**.

Exercício 7. Dado um vector de *N*-elementos (isto é, array), *A*, gerar outro vector, *B*, de modo a que *B* contenha apenas os elementos de *A* que são números pares maiores que 0. O programa em C deve também contar o número de elementos em *B* e mostrar esse valor

no final do programa. Por exemplo: suponha que $N = 12$ e $A = [0, 1, 2, 7, -8, 4, 5, 12, 11, -2, 6, 3]$, então B será: $N=12$ e $A = [0, 1, 2, 7, -8, 4, 5, 12, 11, -2, 6, 3]$: $B = [2, 4, 12, 6]$. Porque B tem quatro elementos, os seguintes devem ser impressos no final do programa:

Número de elementos em $B = 4$.

Use a função `printfNexys` para tal. Nomear o programa **EvenPositiveNumbers.c**. Teste o seu programa quando A tiver 12 elementos.

Exercício 8. Dados dois vectores de N -elementos (isto é, arrays), A e B , crie outro vector, C , definido como:

$$C(i) = |A[i] + B[N-i-1]|, \quad i = 0, \dots, N-1.$$

Escreva um programa em C que calcula o novo vector. Use vetores de 12 elementos no seu programa. Ao programa dê o nome **AddVectors.c**.

Exercício 9. Implemente o algoritmo de ordenação *bubblesort* em C. Este algoritmo ordena os componentes de um vector em ordem ascendente por meio do seguinte procedimento:

1. Percorra o vector repetidamente até estar terminar.
2. Trocar qualquer par de componentes adjacentes, se $V(i) > V(i+1)$.
3. O algoritmo pára quando cada par de componentes consecutivos estiver em ordem.

Utilize vetores de 12 elementos para testar o seu programa. Dê um nome ao programa **BubbleSort.c**.

Exercício 10. Escreva um programa em C que calcula o fatorial de um dado número não negativo, n , através de multiplicações sucessivas. Enquanto deve testar o seu programa para múltiplos valores de n , a sua submissão final deve ser para $n = 7$. O programa deve imprimir o valor $\text{fatorial}(n)$ no final do programa. n deve ser uma variável definida estaticamente dentro do programa. Nomear o programa **Factorial.c**.