Trabalho SPI com FreeRTOS

Sistemas de Tempo Real (SISTR)
MEEC 2020/2021

Sumário

Introdução

Trabalho realizado no âmbito da unidade curricular de Sistemas de Tempo Real (SISTR).

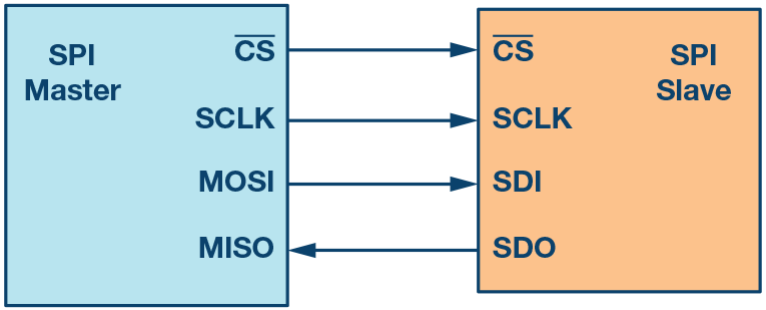
Tem como objetivo o desenvolvimento de um trabalho com FreeRTOS utilizando periférico SPI do STM32.

Para além disso pretende-se desenvolver uma aplicação com o FreeRTOS que pe ler os dados do sensor escolhido e enviá-los para um computador usando uma comunicação série.

SPI

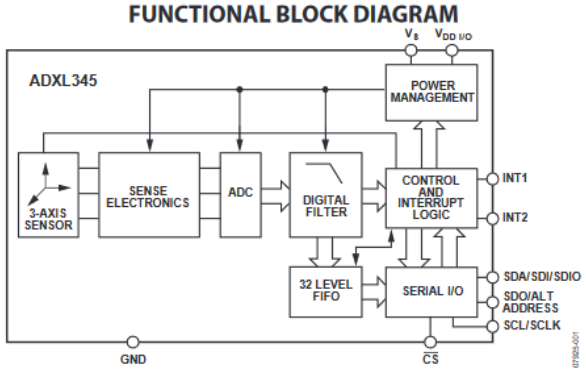
- O que é?

- Características:
  - Comunicação síncrona
  - Arquitetura *master-slave*
  - Comunicação *full-duplex*
  - Comunicação por 4 sinais:
    - *Chip Select* (CS)
    - *Serial Clock* (SCLK)
    - *Master Output Slave Input* (MOSI)
    - *Master Input Slave Output* (MISO)



ADXL345

- Acelerómetro de 3 eixos (x,y,z)
- Tamanho reduzido e baixo consumo
- Saída digital de 16 bits
- Consegue medir aceleração dinâmica e estática



Configuração do Sensor

- Sensor configurado para +/- 2g e 10-bit de resolução (correspondente a 3.9 mg/LSB);
- É possível ter 1024 leituras diferentes para um valor entre -2g e +2g;
- Exemplo:

RAW	BINARY	LSB value for +/-2G	Calc MilliG
16	10000	0.061	0.976
17	10001	0.061	1.037
18	10010	0.061	1.098

- Por este motivo é necessário multiplicar o valor obtido por 3.9mg.

FreeRTOS

- O FreeRTOS é um sistema operativo dedicado a microcontroladores;
- Pode ser usado para permitir uma maior simplicidade e desenvolvimento ráp
- Recorrendo a um RTOS é possível:
  - Separar aplicações complexas em pequenas tarefas podendo levar assir um melhor reaproveitamento do código;
  - Evitar a complexidade da gestão temporal e de sequenciação de tarefas
  - Melhorar a eficiência da comunicação entre as diferentes partes do programa.

SPI Driver

- #include ”drivespiadxl.h”
- void SPI_gpio(void);
- void SPI_config(void);
- void ADXL345(void);
- void SPI_write(uint8_t address, uint8_t data)
- uint8_t SPI_read(uint8_t address);

USART Driver

- #include “driveusart.h”

- void USART_Gpio();
- void USART_Setup(uint32_t baudrate, uint16_t buffer_Tx);
- void USART_Interrupt(uint8_t channel, uint32_t nvic_group, uint8_t priority, uint8_t subpriority);
- void USART_Close();
- void USART_Flush(void);
- void USART_PutChar(char put_char);
- void USART_PutString(char *put_string);

Joystick Driver

- #include “drivejoystick.h”

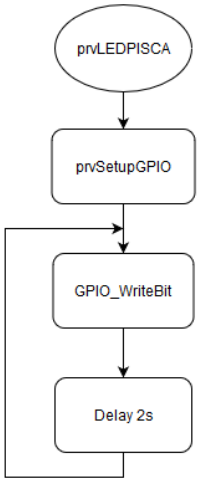
- void joystick_setup(void);
- char joystick_read();
- void joystick_flush(void);

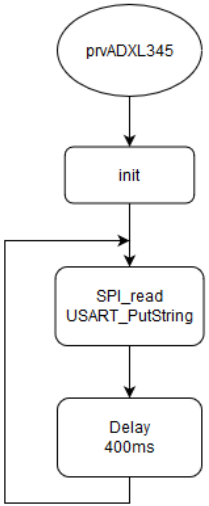
Tarefas FreeRTOS

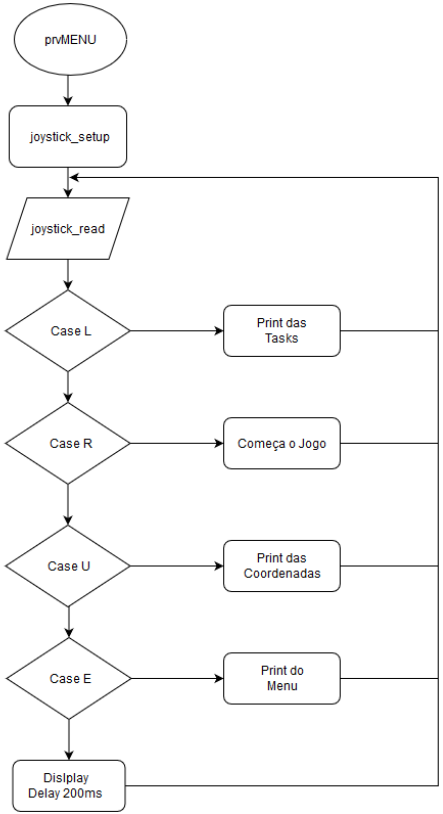
- Neste projeto foram utilizadas 3 tarefas:
  - Led a piscar (prvLEDPISCA);
  - Ler os valores do sensor, enviar para a USART e converter em *string* (prvADXL345);
  - Configurações do Menu e do jogo no LCD (prvMENU);

```
/* Criação das Tarefas */
xTaskCreate( prvLEDPISCA, "LED", configMINIMAL_STACK_SIZE, NULL, mainLEDPISCA_TASK_PRIORITY, &HandleTask1 );
xTaskCreate( prvADXL345, "ADXL345", configMINIMAL_STACK_SIZE*2, NULL, mainADXL345_TASK_PRIORITY, &HandleTask2 );
xTaskCreate( prvMENU, "MENU", configMINIMAL_STACK_SIZE*2, NULL, mainMENU_TASK_PRIORITY, &HandleTask3 );
```

Fluxogramas

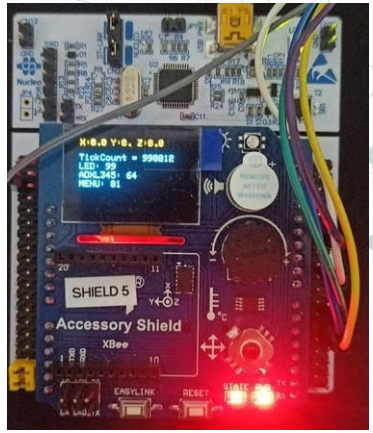






Stack

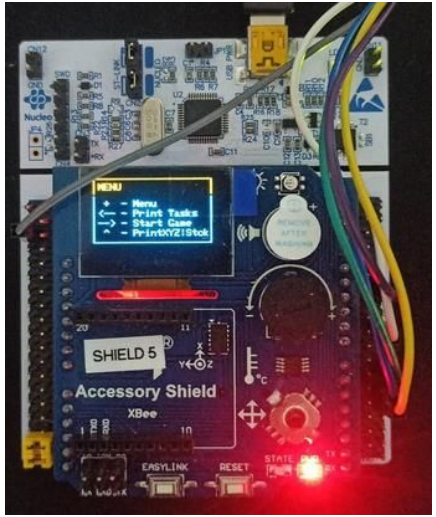
```
UBaseType_t uxHighWaterMark1;
uxHighWaterMark1 = uxTaskGetStackHighWaterMark(HandleTask1);
UBaseType_t uxHighWaterMark2;
uxHighWaterMark2 = uxTaskGetStackHighWaterMark(HandleTask2);
UBaseType_t uxHighWaterMark3;
uxHighWaterMark3 = uxTaskGetStackHighWaterMark(HandleTask3);
```



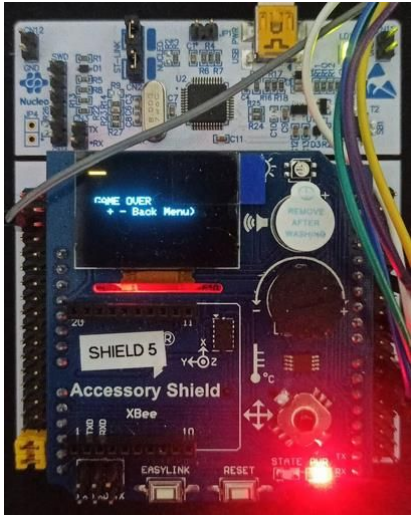
- prvLEDPISCA ➡ configMINIMAL_STACK_SIZE
- prvADXL345 ➡ configMINIMAL_STACK_SIZE*2
- prvMENU ➡ configMINIMAL_STACK_SIZE*2


```
/* Prioridades das tarefas */
#define mainADXL345_TASK_PRIORITY    ( tskIDLE_PRIORITY + 2)
#define mainLEDPISCA_TASK_PRIORITY   ( tskIDLE_PRIORITY + 1)
#define mainMENU_TASK_PRIORITY       ( tskIDLE_PRIORITY + 1)
```

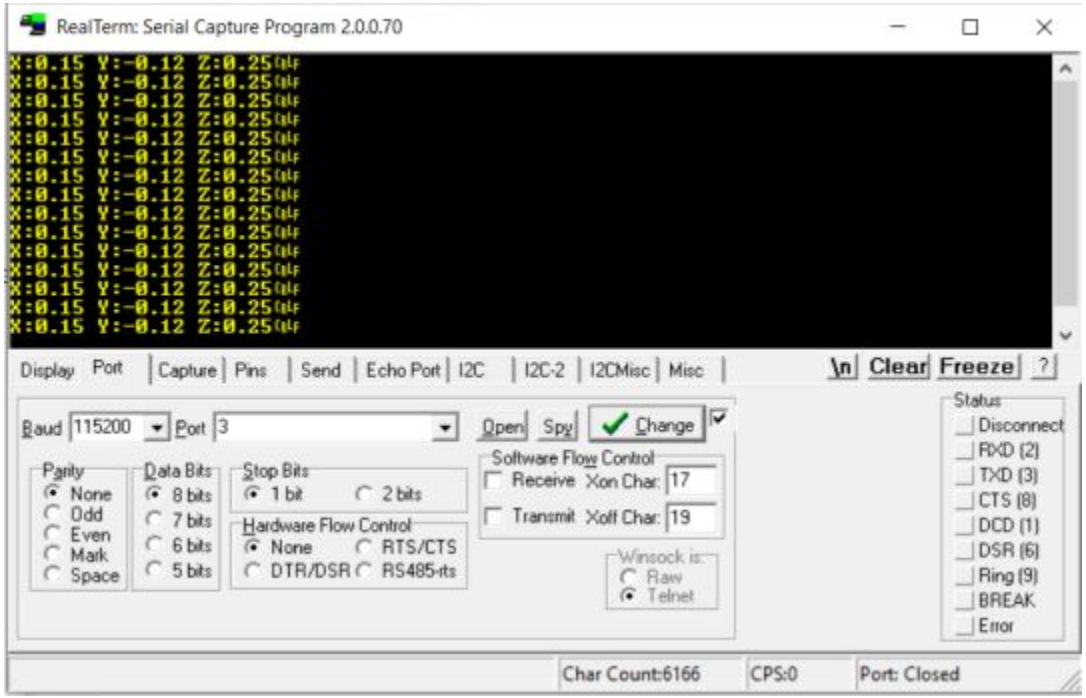
Resultados


MENU
SHIELD 5
Accessory Shield
XBee
EASYLINK
RESET


SHIELD 5
Accessory Shield
XBee
EASYLINK
RESET


SHIELD 5
Accessory Shield
XBee
EASYLINK
RESET

Resultados


RealTerm: Serial Capture Program 2.0.0.70
X:0.15 Y:-0.12 Z:0.25
Display Port Capture Pins Send Echo Port I2C I2C-2 I2CMisc Misc
\n Clear Freeze ?
Baud 115200 Port 3 Open Spy Change
Parity None Odd Even Mark Space
Data Bits 8 bits 7 bits 6 bits 5 bits
Stop Bits 1 bit 2 bits
Hardware Flow Control None RTS/CTS DTR/DSR RS485-rts
Software Flow Control Receive Xon Char: 17 Transmit Xoff Char: 19
Winsock is: Raw Telnet
Status Disconnect RXD (2) TXD (3) CTS (8) DCD (1) DSR (6) Ring (9) BREAK Error
Char Count:6166 CPS:0 Port: Closed

Conclusão

- O trabalho foi realizado com sucesso;
- Criação de uma aplicação utilizando o Sistema operatico de tempo real FreeR
  com os dados do sensor ADXL345;
- Comunicação série dos dados para um computador;
- Problemas com o delay da leitura dos valores do sensor, influenciando assim
  jogo;