Trabalho desenvolvido com um acelerómetro ADXL 345

1180919 – Diogo Freitas

1181160 – Manuel Couto

1. Introdução

O presente projeto foi realizado no âmbito da cadeira de Sistemas de Tempo Real (SISTR), do Mestrado de Engenharia Eletrotécnica e de Computadores – Automação e Sistemas (MEEC – AS). Tem como objetivo a recolha de dados de um acelerómetro, através de um microcontrolador, e respetiva interpretação, para o controlo de um personagem virtual. O sensor vai comunicar com o microcontrolador através do protocolo I2C e os dados vão ser enviados para o computador via porta série, com uso do protocolo USART.

2. Acelerómetro [1]

O ADXL345 é um acelerómetro de 3 eixos, fino e ultra potente, de alta resolução (13 bits) até ±16 g. Os dados de saída digital são formatados como um complemento de 16 bits e são acessíveis através de um SPI (3 ou 4 fios) ou de uma interface digital I2C. Para este projeto, foi escolhido o protocolo de comunicação I2C.

a. Configuração de registos

Antes de começar a ler valores do acelerómetro, é preciso configurar alguns registos para que este funcione da forma pretendida.

O primeiro registo é o registo 0x2D (Figura 1), onde é configurado o modo de alimentação do sensor. Assim sendo, com a necessidade de uma medição constante por parte do sensor, foi colocado o valor "1" do bit 3 (Measure). De seguida, é preciso configurar o registo 0x31 (Figura 2), que é utilizado para definir o formato com que os dados são apresentados.

```
1. // Configuração dos registos do acelerómetro
2.
3.  SendMSG_I2C(0x2D, 0b00001000);                  //ADXL345 - measure mode
4.  SendMSG_I2C(0x31, 0b00001000);                  //ADXL345 - data-format
```

Register 0x2D—POWER_CTL (Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0
0	0	Link	AUTO_SLEEP	Measure	Sleep	Wakeup	

Figura 1 – Registo 0x2D [1]

Register 0x31—DATA_FORMAT (Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0
SELF_TEST	SPI	INT_INVERT	0	FULL_RES	Justify	Range	

Figura 2 – Registo 0x31 [1]

Cada bit deste registo contém uma configuração própria, sendo atribuído o valor "0" e "1" consoante a necessidade. Neste caso, os bits são:

Bits [0,1] (Range) → D0 = 0 e D1 = 0, para se obter medições de ±2g;
Bits 2 (Justify) → D2 = 0, valores ajustados à direita;
Bits 3 (FULL_RES) → D3 = 1, máxima resolução;
Bits 4 (0) → D4 = 0, reservado;
Bits 5 (INT_INVERT) → D5 = 0, ativa interrupção no estado lógico alto (contudo a interrupção não é usada);
Bits 6 (SPI) → D6 = 0, não se utiliza o protocolo SPI;
Bits 7 (SELF_TEST) → D7 = 0, não se utiliza o modo de Self-Test.

b. Formatação dos dados recebidos

Os dados são fornecidos em registos de 8 bits, sendo que para cada eixo existem dois registos, um relativo aos bits mais significativos, e outro relativo aos menos significativos. Posteriormente à sua recolha e armazenamento, em variáveis do tipo "uint8_t", este devem ser convertidos em três variáveis do tipo "int16_t", referentes a cada um dos eixos. Para tal, desloca-se o registo mais significativo em 8 bits e adiciona-se o menos significativo. Por fim, para facilitar a compreensão dos resultados, é necessário convertê-los para acelerações. Tendo por base que o valor varia entre -512 e 511, para um alcance de ± 2g, utilizando as capacidades gráficas do Excel, é possível obter uma equação que descreva o sistema, como visível na Figura 3. Os valores são ainda multiplicados por cem com o intuito de não serem utilizadas variáveis com casas decimais.

X	Y
-512	-2
0	0
511	2

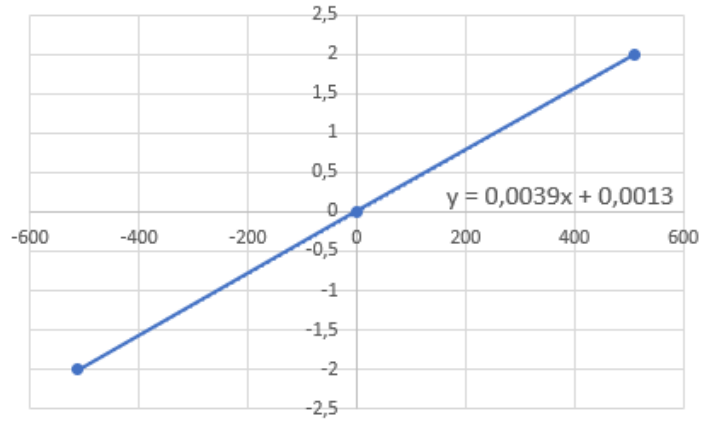

Figura 3 – Equação do Sistema

3. Protocolo I2C

Inter-Integrated Circuit (I2C) é um protocolo de comunicação síncrono que utiliza apenas duas ligações físicas: Serial Data (SDA) e Serial Clock (SCL). SDA é a linha por onde vão circular os dados e SCL é a linha onde é transmitido o sinal de relógio (ou clock) necessário para sincronizar todos os dispositivos. A existência de uma só linha de dados significa que este protocolo apenas permite a comunicação half-duplex (envio e receção de dados de forma alternada) em vez da comunicação full-duplex (envio e receção de dados simultaneamente).

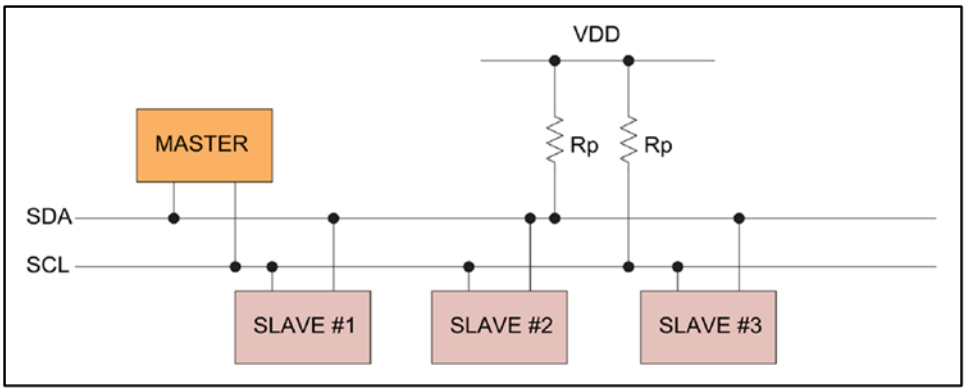

Figura 4 - Protocolo I2C [2]

Como se pode observar na Figura 4, o protocolo I2C permite ligar uma série de dispositivos no mesmo barramento, sendo que pelo menos um tem de funcionar como master. O master é responsável por realizar a coordenação de toda a comunicação, enviando e recebendo informações dos slaves existentes, assim como enviar o sinal de clock. Uma vez que o I2C não possui linhas de seleção, o método de endereçamento é utilizado para definir com qual slave se pretende comunicar. Os dados são enviados por mensagens por I2C, que são divididas por *frames.* Cada mensagem contém o endereço do slave e um ou mais data frames (dados a serem transmitidos). (Figura 5)

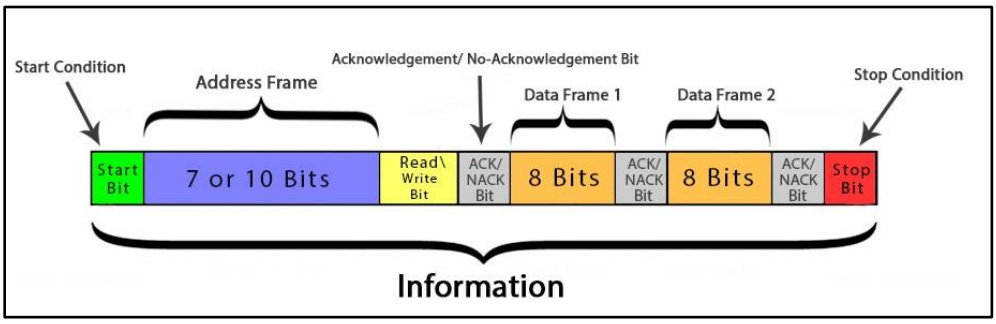

Figura 5 - Diagrama de I2C frames da mensagem [3]

Start Condition: a linha SDA transita do nível lógico alto para o nível lógico baixo, antes de a linha SCL mudar do estado alto para baixo;

Stop Condition: a linha SDA transita do nível lógico baixo para o nível lógico alto, depois de a linha SCL mudar do estado baixo para alto;

Address Frame: um endereço único (7 ou 10 bits) para cada slave, que o identifica quando o master pretende comunicar;

Read/Write Bit: bit que especifica se o master está a enviar ou a ler dados do slave;

ACK/NACK Bit: bit que confirma se os dados são ou não recebidos com sucesso.

a. Configuração do I2C

Para a configuração do I2C, é preciso inicializar os GPIO que estão interligados às entradas I2C do acelerómetro, **SDA (GPIOB9)** e **SCL (GPIOB8)**:

```
1.  GPIO_InitTypeDef GPIO_InitStructure;
2.
3.  //SCL - GPIOB8 | SDA -> GPIOB9
4.  GPIO_InitStructure.GPIO_Pin     = GPIO_Pin_8 | GPIO_Pin_9;
5.  GPIO_InitStructure.GPIO_Speed   = GPIO_Speed_50MHz;
6.  GPIO_InitStructure.GPIO_Mode    = GPIO_Mode_AF_OD;
7.
8.  GPIO_Init(GPIOB, &GPIO_InitStructure);
```

De seguida é preciso configurar os registos relativos ao I2C para escolher o modo de funcionamento e a velocidade do clock, entre outras opções:

```
1.  I2C_InitTypeDef I2C_InitStructure;
2.
3.  I2C_InitStructure.I2C_Mode                = I2C_Mode_I2C;  //Standard Mode
4.  I2C_InitStructure.I2C_DutyCycle                    = I2C_DutyCycle_2;
5.  I2C_InitStructure.I2C_Ack                          = I2C_Ack_Enable;
6.  I2C_InitStructure.I2C_AcknowledgedAddress          =
    I2C_AcknowledgedAddress_7bit;
7.  I2C_InitStructure.I2C_ClockSpeed                   = 400000;
8.
9.  I2C_Init(I2C1, &I2C_InitStructure);
10. I2C_Cmd(I2C1, ENABLE);                                    //Ativa o I2C
11. I2C_AcknowledgeConfig(I2C1, ENABLE);
```

b. Envio de dados

Para enviar dados através do I2C, a mensagem tem de seguir o formato já explicado anteriormente. No datasheet do ADXL345 encontra-se o diagrama (Figura 6) que explica o processo de envio dos dados.

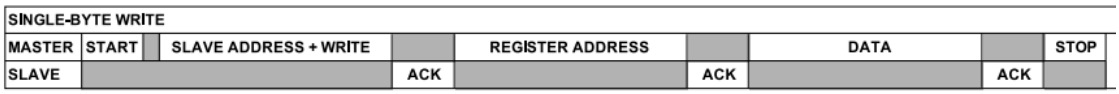

Figura 6 - Trama de envio de dados [1]

Assim, implementou-se uma função com dois parâmetros de entrada (addr e data) responsável por enviar dados para um determinado endereço.

```
1.          while(I2C_GetFlagStatus(I2C1, I2C_FLAG_BUSY));//Aguarda que o I2C esteja
    livre para a comunicação
2.
3.          //START bit, sinaliza uma nova mensagem
4.          I2C_GenerateSTART(I2C1, ENABLE);
5.          while(I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT) != SUCCESS);
6.
7.          //Slave Address + Write
8.          I2C_Send7bitAddress(I2C1, 0xA6, I2C_Direction_Transmitter);
9.          while(I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED) !=
    SUCCESS);
10.
11.         //Register Address
```

```
12.         I2C_SendData(I2C1, addr);            //addr - Endereço de escrita
13.         while(I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED) != SUCCESS);
14.
15.         //Data Frame
16.         I2C_SendData(I2C1, data);            //data - Data a ser escrita no addr
17.         while(I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED) != SUCCESS);
18.
19.         //STOP bit, fim da mensagem
20.         I2C_GenerateSTOP(I2C1, ENABLE);
```

c. Receção e Tratamento de Dados

De forma semelhante, foi também implementada uma função para a leitura de dados do acelerómetro. A Figura 7 apresenta o processo de receção dos dados.

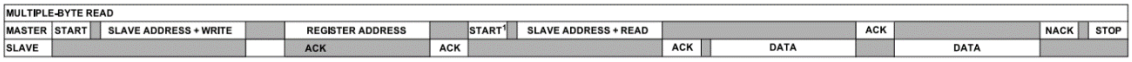

Figura 7 - Trama de receção de dados [1]

O objetivo desta função é a leitura dos registos 0x32 a 0x37 do acelerómetro, que apresentam os valores das acelerações em cada um dos eixos. Cada registo corresponde a 1 byte (8 bits) de informação.

```
1. I2C_AcknowledgeConfig(I2C1, ENABLE);              //Ativa o Acknowledge do
   I2C
2.
3. while(I2C_GetFlagStatus(I2C1, I2C_FLAG_BUSY));    //Aguarda que o I2C esteja
   livre para a comunicação
4. //Construção da mensagem ----------------------------------------------------
   ---------
5. //START bit, sinaliza uma nova mensagem
6. I2C_GenerateSTART(I2C1, ENABLE);
7. while(I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT) != SUCCESS);
8.
9. //Slave Address + Write
10. I2C_Send7bitAddress(I2C1, 0xA6, I2C_Direction_Transmitter); //0xA6 -
   Endereço alternativo de escrita
11. while(I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED) !=
   SUCCESS);
12.
13. //Register Address
14. I2C_SendData(I2C1, 0x32);        //0x32 - 1º endereço de registo (Data x0)
15. while(I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED) != SUCCESS);
16.
17. //Segundo START bit, funciona como um RESTART ou um STOP seguido de um START
18. I2C_GenerateSTART(I2C1, ENABLE);
19. while(I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT) != SUCCESS);
20.
21. //Slave Address + Read
22. I2C_Send7bitAddress(I2C1, 0xA7, I2C_Direction_Receiver); //0xA7 - Endereço
   alternativo de leitura
23. while(I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED) !=
   SUCCESS);
24.
```

```
25. //Eixo X ----------------------------------------------------------------
26. while(I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED) != SUCCESS);
    //Data Frame 1, 8bits menos significativos do X
27. x0 = I2C_ReceiveData(I2C1);      //Leitura dos 8bits menos significativos do
    eixo X
28.
29. while(I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED) != SUCCESS);
    //Data Frame 2, 8bits mais significativos do X
30. x1 = I2C_ReceiveData(I2C1);      //Leitura dos 8bits mais significativos do
    eixo X
31.
32. X = ((x1 << 8) + x0);            //X = shift de 8bits de x1 + x0
33. X = (X  * 0.0039 * 9.81) * 100;  //Conversão de força (g) em aceleração
    (m/s^2)                          //Multiplica por 100 para transformar o
    float num int
34.
35. if(abs(X - X_Average) < ERRO)    //Cálculo da media dos valores lidos no
    eixo X
36.                                  //Se a diferença entre o último valor de X
    e o atual for inferior ao ERRO,
37.                 X_Average = (X_Average + X) /2; //a medição é considerada
    como ruido e o valor retornado é o valor médio das 2 leituras.
38.         else
39.                 X_Average = X;       //Caso contrário é retornado o valor da
    medição atual.
40.         //Eixo Y ----------------------------------------------------------
    -----------------
41.         while(I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED) !=
    SUCCESS); //Data Frame 3
42.         y0 = I2C_ReceiveData(I2C1);
43.
44.         while(I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED) !=
    SUCCESS); //Data Frame 4
45.         y1 = I2C_ReceiveData(I2C1);
46.
47.         Y = ((y1 << 8) + y0) ;
48.         Y = (Y  * 0.0039 * 9.81) * 100;
49.
50.         if(abs(Y - Y_Average) < ERRO)
51.                 Y_Average = (Y_Average + Y) /2;
52.         else
53.                 Y_Average = Y;
54.         //Eixo Z ----------------------------------------------------------
    -----------------
55.         while(I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED) !=
    SUCCESS); //Data Frame 5
56.         z0 = I2C_ReceiveData(I2C1);
57.
58.         while(I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED) !=
    SUCCESS); //Data Frame 6
59.         z1 = I2C_ReceiveData(I2C1);
60.
61.         Z = ((z1 << 8) + z0);
62.         Z = (Z  * 0.0039 * 9.81) * 100;
63.
```

```
64.         if(abs(Z - Z_Average) < ERRO)
65.                 Z_Average = (Z_Average + Z) /2;
66.         else
67.                 Z_Average = Z;
68.         //Fim da Mensagem ----------------------------------------------------
    -----------------
69.
70.         I2C_AcknowledgeConfig(I2C1, DISABLE);     //Desabilita o Acknowledge
    do I2C
71. I2C_GenerateSTOP(I2C1, ENABLE);  //Gera um STOP bit que sinaliza o fim da
    mensagem
```

4. Implementação do projeto

Este projeto está dividido em 2 partes:

- Leitura e apresentação no microcontrolador dos dados obtidos pelo acelerómetro;
- Utilização dos dados obtido para o controlo de um submarino virtual, através da ligação ao computador.

a. Microcontrolador

Na Figura 8, são apresentados os fluxogramas principais do código do projeto: Fluxograma da *main(), init ()* e do *menu().*

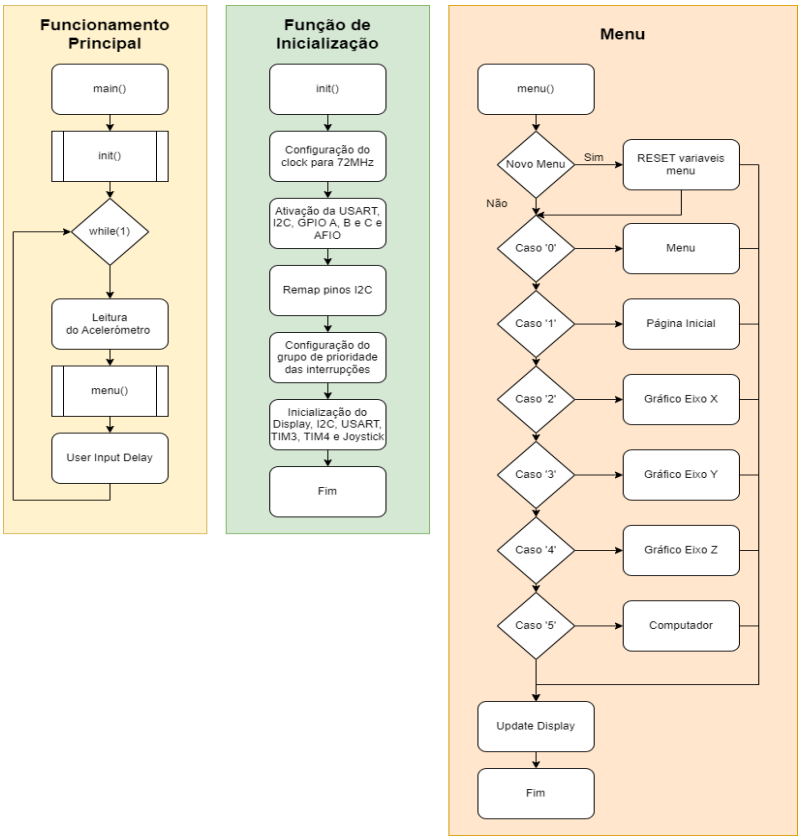

Figura 8 - Fluxogramas principais do código

No microcontrolador, é apresentado um menu para o utilizador poder selecionar as funcionalidades que quer realizar. Essas opções podem ser selecionadas com o joystick presente na placa, Figura 9.

Figura 9 - Display do menu

Nas 3 primeiras funcionalidades, o microcontrolador não precisa de ligação ao computador, pois baseiam-se apenas na leitura dos dados obtidos pelo acelerómetro e apresentação destes, em gráficos definidos pelos eixos cartesianos, Figura 10. O LED STATE é controlado por um PWM gerado pelo TIM4, onde o *duty cicle* corresponde ao valor da aceleração do eixo em estudo, para acelerações negativas o *duty cicle* corresponde a zero.

Figura 10 - Display dos gráficos X, Y e Z

b. Computador

Como mencionado previamente, o microcontrolador tira partido da comunicação serie para enviar os dados do acelerómetro para o computador. Este rege-se pelo protocolo USART, com uma velocidade de comunicação de 9600 bps, 8 bits de mensagem, um STOP bit e sem paridade.

Para que sejam enviados os dados do acelerómetro para o computador, o utilizador deve selecionar, no microcontrolador, a opção “Computador”. Seguidamente, do lado do computador, os dados podem ser lidos através de um terminal, Figura 11, para além disso, foi desenvolvido um jogo em C# no qual o utilizador o pode controlar a movimentação do submarino através da inclinação do microcontrolador, Figura 12. É possível encontrar em anexo um vídeo da interação do microcontrolador com o jogo. [3]

É de salientar, como o objetivo do projeto não era o desenvolvimento de um jogo, o código que gera o mapa é de autoria de Sebastian Lague [4], contudo foi necessário desenvolver dois scripts, um que faz a leitura da porta serie, onde se encontra o microcontrolador conectado, e outro que converte os dados do acelerómetro no movimento do submarino.

Figura 11 - Apresentação dos dados obtidos

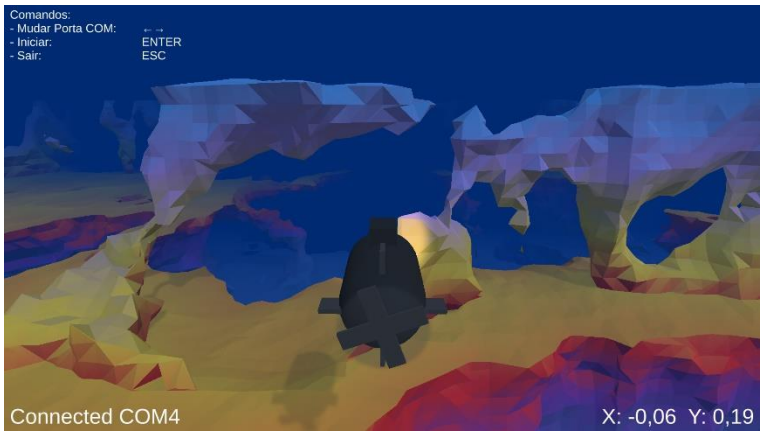

Figura 12 - Imagem da aplicação

5. Teste e resultados

De forma a verificar a leitura e apresentação dos dados obtidos, foi realizado uma medição da aceleração gravítica (aceleração constante igual a 9,81 m/s2), ou seja, em relação ao eixo do Z. Na Figura 13, é visível a leitura constante dos dados, provando-se correta a configuração do sensor.

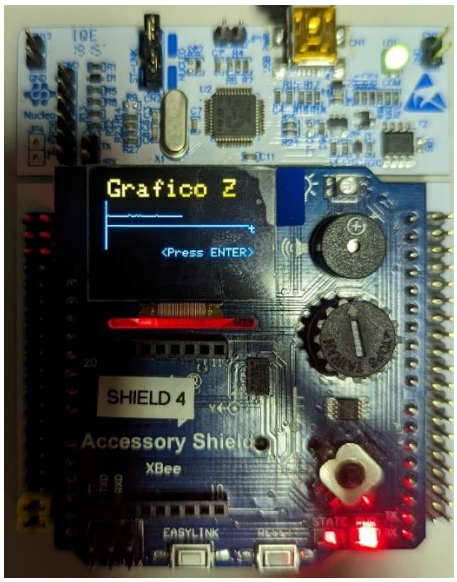

Figura 13 - Teste de medição e apresentação dos dados

Bibliografia

[1] Analog Devices, “ADXL345 Data Sheet,” [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/adxl345.pdf.

[2] S. Afzal, “I2C Primer: What is I2C? (Part 1),” Analog Devices, [Online]. Available: https://www.analog.com/en/technical-articles/i2c-primer-what-is-i2c-part-1.html.

[3] R. Singh, “Difference between I2C and SPI (I2C VS SPI),” 07 fevereiro 2020. [Online]. Available: https://medium.com/@rjrajbir24/difference-between-i2c-and-spi-i2c-vs-spi-c6a68d7242c4.

[4] S. Lague, “Coding Adventure: Marching Cubes,” 06 maio 2019. [Online]. Available: https://www.youtube.com/watch?v=M3iI2l0ltbE&ab_channel=SebastianLague.