

Luva Tradutora de Libras

Dispositivo que visa facilitar a comunicação entre um deficiente auditivo e uma pessoa que não sabe LIBRAS

Anderson Sales Rodrigues Pinto
Universidade de Brasília - UnB
Brasília-DF, Brasil
aandersonsales@gmail.com

Ítalo Rodrigo Moreira Borges
Universidade de Brasília - UnB
Brasília-DF, Brasil
italrmb@gmail.com

Resumo— Apresenta-se uma solução com auxílio de sensores de flexão, acelerômetro e módulo bluetooth para a implementação de uma luva capaz de traduzir o alfabeto de libras em mensagem, executado por uma pessoa muda. Logo, poderá se comunicar com pessoas que não falam em LIBRAS.

Keywords—*acessibilidade, comunicação, LIBRAS, sensor de flexão.*

I. JUSTIFICATIVA

As pessoas que nascem surdas-mudas, afônicas ou qualquer outro tipo de deficiência auditiva enfrentam grandes dificuldades para se comunicar. De forma a contornar estas dificuldades criou-se a Língua Brasileira de Sinais, que possibilitou os surdos a se comunicarem. Porém as pessoas que não sofrem com esse tipo de deficiência, em sua maioria não entendem essa linguagem, o que dificulta a comunicação. Segundo dados do IBGE no censo de 2000, registrou-se 5.7 milhões de deficientes auditivos no Brasil, já no censo de 2010, registrou-se 9,7 milhões de deficientes auditivos no Brasil. Logo percebe-se o aumento de pessoas com essa deficiência. Tendo isso em mente, teve-se a ideia deste projeto.

Esta luva será capaz de traduzir o movimento de uma das mãos de uma pessoa muda, no qual o movimento refere-se ao alfabeto de LIBRAS, onde será processados em um sistema microcontrolado, transmitir essa informação, traduzi-la e enviar a mensagem para um app de celular (ou display lcd) por meio de bluetooth.

II. OBJETIVO

A. Comunicação entre pessoas surdas e não surdas

- A pessoa surda iria utilizar uma luva capaz de traduzir o alfabeto de Libras e mandar esta tradução para um dispositivo móvel que irá mostrar a letra correspondente ao sinal de libra feito pelo usuário mudo.

B. Integrar um deficiente visual na sociedade

- Tendo em vista a dificuldade que os deficientes auditivos têm em se comunicar, esse dispositivo permitirá a comunicação com pessoas que não sofrem com deficiência auditiva e nem sabem a linguagem de sinais, LIBRAS. E assim, permitirá a integração dessa classe de pessoas na sociedade.

III. REQUISITOS

Como o mínimo necessário para o projeto ser desenvolvido temos:

- uma placa MSP430;
- sensores de flexão;
- Display LCD ou app bluetooth;
- extensômetro;
- acelerômetro;
- módulo bluetooth;
- luvas;

A expectativa é de que as pessoas surdas-mudas usem esta luva para poder se comunicarem com pessoas sem este tipo de deficiência, de modo que a comunicação entre elas se faça de forma mais efetiva.

O produto será restrito apenas a traduzir o alfabeto em LIBRAS e será construído em apenas uma luva. Não será usada a outra luva do par, pois com uma mão já é possível fazer todas as letras do alfabeto de LIBRAS.

A interface do produto se dará basicamente por uma luva que, com todo o sistema microcontrolado construído, irá traduzir o alfabeto em LIBRAS e mandar esta informação a um app de bluetooth, onde o usuário final será a pessoa que não entende libras.

IV. Desenvolvimento

Com base na figura 1, foi necessário fazer o mapeamento de cada dedo que visa diferenciar cada letra do alfabeto, com as combinações entre movimento de cada dedo, consegue-se identificar qual é a letra do alfabeto de libras, porém essas combinações não são suficientes para identificar todo alfabeto. As letras (E e S), (U e V), (F e T), (G e Q), (C, Ç) e (K e H) não identificáveis só com o mapeamento, necessita de um módulo Giroscópio/ Acelerômetro (MPU6050) para diferenciar.

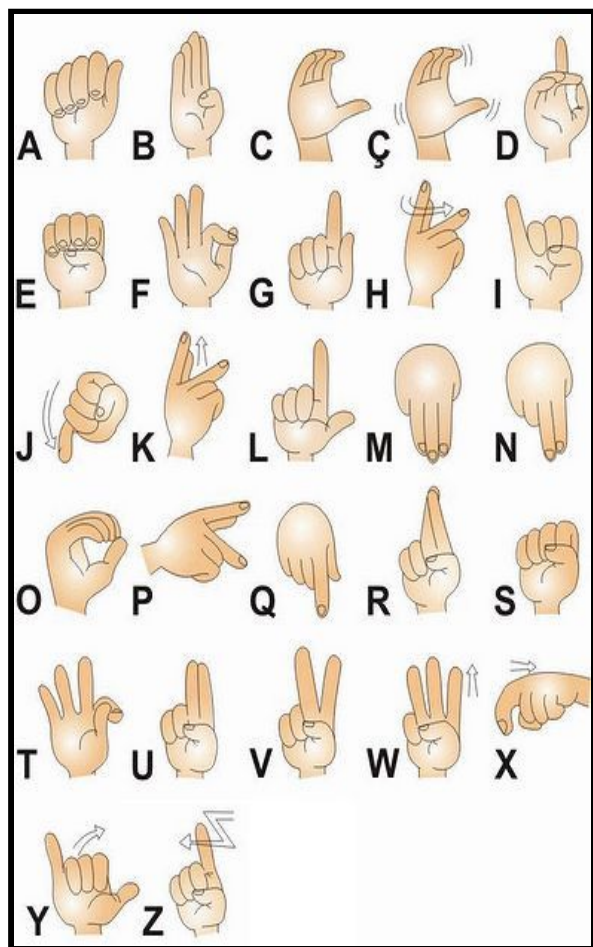


Figura 1 - Alfabeto de LIBRAS.

Mapeamento dos dedos:

Dedo Polegar:

- Polegar relaxado: A, D, F, G, H, K, P, Q, T, O, M, N
- Polegar flexionado: B, E, I, J, R, S, U, V, W, X, Z
- Polegar Esticado: C, Ç, L, Y

Dedo Indicador:

- Indicador Flexionado até a palma: A, E, I, J, S, X, Y
- Indicador Flexionado: C, Ç, F, O, T, X
- Indicador esticado: B, D, G, H, K, L, M, N, P, Q, R, U, V, W, Z

Dedo Médio

- Médio Flexionado até a palma: A, E, I, J, L, Q, S, X, Y, Z, G
- Médio Esticado: B, F, M, N, R, T, U, V, W
- Médio meio Flexionado: H, K, P
- Médio Flexionado: C, Ç, D, O

Dedo Anelar:

- Anelar Flexionado até a palma: A, E, G, H, I, J, K, L, N, P, Q, R, S, U, V, X, Y, Z
- Anelar Esticado: B, F, M, T, W
- Anelar Flexionado: C, Ç, D, O

Dedo Mindinho:

- Mindinho Flexionado até a palma: A, E, G, H, K, L, M, N, P, Q, R, S, U, V, W, X, Z
- Mindinho Flexionado: C, Ç, D, J, O
- Mindinho Esticado: B, F, I, T, Y

A princípio este é um pequeno esboço das ligações do projeto.

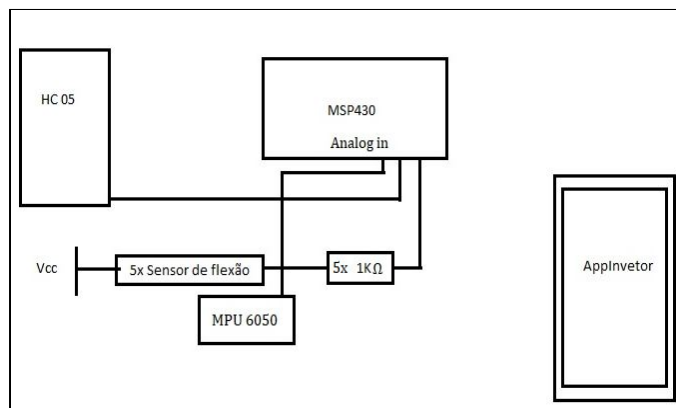


Figura 2 - Esquema de ligações na placa.

Os 5 divisores de tensão serão conectados a 5 entradas analógicas da MSP430. Com a variação da resistência do sensor de flexão a tensão na entrada do pino vai mudar, de forma que com essa variação seja possível mapear os movimentos dos dedos.

O módulo bluetooth irá receber de imediato qual letra vai sair no celular e irá mandar para o aplicativo onde irá mostrar na tela qual letra corresponde àquelas variações de tensão.

Para letras que precisam de movimento para serem reconhecidas será utilizada a MPU-6050, que se trata de um acelerômetro e um giroscópio embutido. Com ela será possível distinguir letras que possuem a mesma variação de dedos, mas com algum movimento.

V. Descrição do hardware

Tabela de Materiais

Material	Fabricante	Modelo
Sensor de Flexão	-	-
MPU6050	-	-
Módulo Bluetooth HC-05	-	-
Resistor 220Ω	-	-
MSP430	Texas Instrument	G2553

O sensor de flexão foi feito de forma artesanal, usando duas folhas de papel alumínio, 3 folhas de papel A4(sendo uma delas pintada com grafite) e jumpers. Obteve-se uma variação de resistência considerável, mas ainda com muita flutuação.

Já com outros teste feitos conseguiu-se uma versão ainda melhor do sensor de flexão. Dessa vez usando duas tiras de cobre adesivas coladas em um pedaço de plástico e um papel pintado com lápis 2B sobreposto a essas duas tiras montou-se um sensor de flexão ainda mais estável e com pouquíssima variação de resistência.

Os resistores de 220Ω serão usados em circuitos divisores de tensão com os 5 sensores de flexão feitos. Para uma melhor coleta de valores de tensão o componentes que irão fornecer estas tensões serão os resistores de 220Ω. Um esquemático de como ficou este circuito encontra-se nos Anexos na figura 10.

O módulo bluetooth HC05 será utilizado para fazer a comunicação entre a MSP e o aplicativo de celular, aplicativo esse que será construído pelo AppInventor, um software livre do MIT.

Módulo MPU6050

O MPU6050 é um sensor de 3 Eixos, contém em um único chip um acelerômetro, um giroscópio do tipo MEMS. São 3 eixos para o acelerômetro e 3 eixos para o giroscópio,

sendo ao todo 6 graus de liberdade (6DOF) e esta placa GY-521 tem um sensor de temperatura embutido no CI MPU-6050 para leituras entre -40 e +85 °C. É um dispositivo de alta precisão e baixo custo.

Esse módulo utiliza um barramento de comunicação que é a I²C, interfaceando com a MSP430 através dos pinos SDA e SCL (pinos analógicos), GND (terra) e alimentação que varia de 3-5 V. Apresenta também I²C auxiliar com os pinos XDA, XCL, ADO que fornece o endereço e o INT que é a interrupção.

Apresenta um conversor analógico digital de 16 bits que permite a leitura das coordenadas x,y,z ao mesmo tempo. As dimensões são 20 x 16 x 1mm.

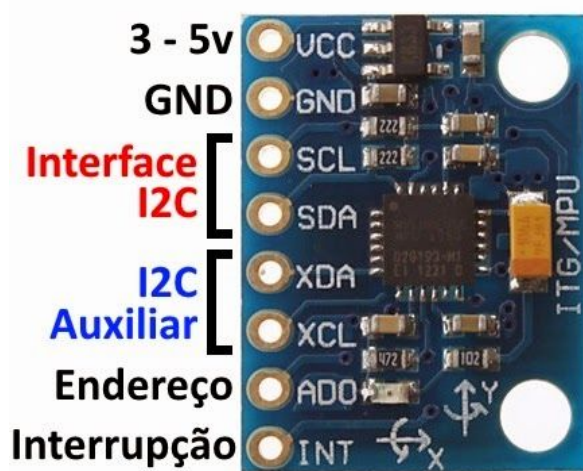


Figura 3 - Módulo MPU-6050.

Módulo Bluetooth HC-05

É um módulo bastante utilizado para comunicações sem fio, permite a comunicação de um microcontrolador e um dispositivo móvel (celular). É configurado por comando AT e tem a possibilidade de funcionar como SLAVE ou MASTER.

Este módulo pode ser alimentado na faixa de 3,3 a 6V, ou seja, a MSP430 atende a esta necessidade. No entanto, os pinos TX e RX utilizam níveis de 3,3V, logo, não permite que sejam conectados diretamente na placas em 5V, mas é possível com a utilização de resistores como divisores de tensão.

Há um pequeno botão para entrar em modo de comando, sendo que também é possível acessar este modo por software utilizando o pino EN. E apresenta um LED incorporado que indica o estado da conexão.

Características:

- Protocolo Bluetooth: v1.1 / 2.0.
- Frequência: banda ISM de 2,4GHz
- Modulação: GFSK

- Potência de transmissão : menos de 4dBm Classe 2
- Sensibilidade: Menos de -84dBm no 0,1% BER
- Razão assíncrona: 2.1Mbps (Max) / 160 kbps
- Síncrono : 1Mbps / 1Mbps
- porta serial Bluetooth (mestre e escravo)
- Alimentação 3,3VCC 50mA (suporta de 3,3 a 6V)
- Temperatura de operação: -5 a 45°C



Figura 4 - Módulo Bluetooth HC-05.

Sensor Flexível

É um sensor artesanal, onde é composto por papel pintado de lápis, duas fitas de cobre, plástico de garrafa pet flexível, fita isolante e jumpers. O carbono contido no lápis fornece uma resistência variável e dependendo do comprimento do papel pintado, terá uma resistência associada respeitando a segunda lei de ohm.

$$R = (\rho \ell) \div (A) \quad \Omega$$

ρ - Resistividade que depende do material

ℓ - Comprimento

A- Área

Diante disto, nota-se que quando o papel for grande, terá resistência alta e quando é pequeno, resistência baixa. Então, quando deforma, o papel fica encolhido, logo, apresentará resistência baixa, caso contrário, não varia.

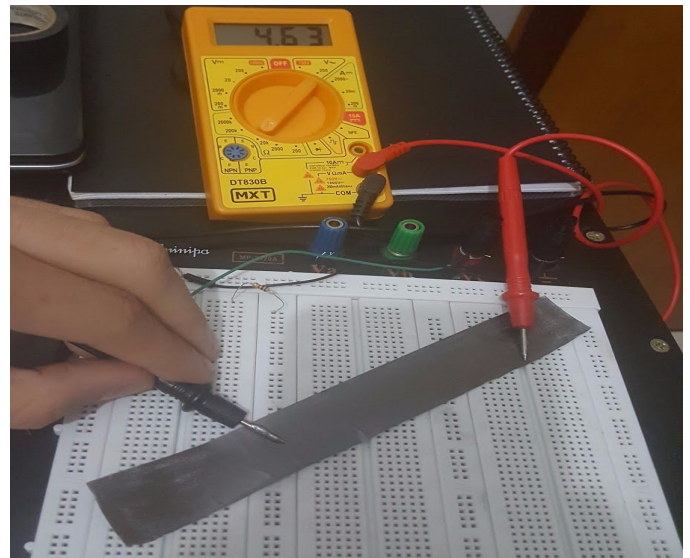


Figura 5 - Papel Pintado de lápis.



Figura 6 - Papel Pintado de lápis e as Fitas de cobre.

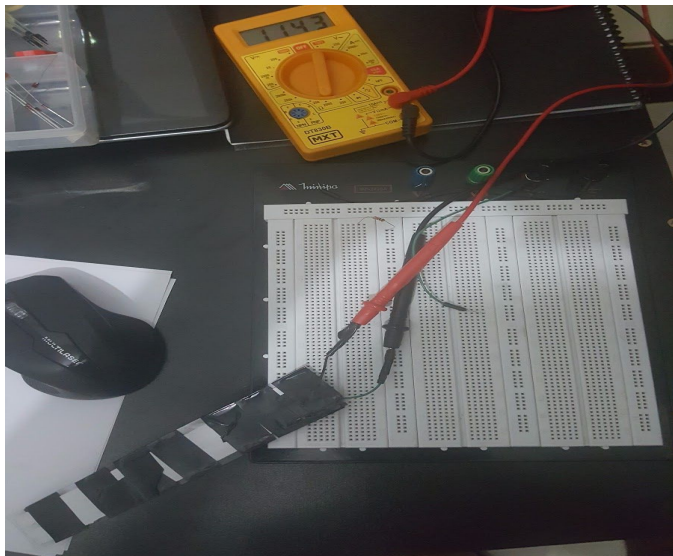


Figura 7 - Sensor Flexível.

VI. Descrição do software

O software utilizado na elaboração dos códigos foi o Code Composer Studio v7 e Energia IDE. Testou-se principalmente os códigos para o módulo bluetooth e para o acelerômetro/giroscópio comentado em anexo.

O código para o bluetooth que foi usado como teste está nos Anexos. O primeiro bloco do código (O que vai até o fim do while) configura o watchdog timer e verifica se o clock está calibrado por meio de um while. Essa verificação é necessária para a operação UART dada a pequena margem de erro que dá graças a tolerância interna de oscilação.

De P1SEL a P1OUT tem-se as configurações das portas de entrada e saída do código. Neste caso está sendo configurada também a função GPIO

Do próximo bloco em diante está toda a configuração USCI (Universal Serial Communication Interface), que é capaz de suportar múltiplas comunicações seriais. Após estas

configurações o código entra em um switch case, onde em cada um dos 2 casos o programa irá jogar uma letra para a saída, sendo elas A ou B.

Na figura 6 temos o código para a conversão AD utilizado nos testes. Por meio do modo de conversões sucessivas e usando apenas 4 entradas analógicas fizemos os testes necessários para testar a funcionalidade do divisor de tensão.

No código da conversão AD tem-se uma função onde configura-se o conversor AD e realizam-se as conversões sucessivas nos pinos P1.0 a P1.4. Esta função será chamada sempre no loop infinito, onde Cada uma dessas conversões será gravada em uma variável chamada *samples* e seus valores posteriormente serão usados para fazer a lógica de saída para cada letra do alfabeto de libras. O código completo encontra-se nos Anexos na figura 11.

A rotina de interrupção que será usada irá setar as flags CPUOFF e GIE, onde após uma leitura a CPU irá desligar por um tempo até que a flag CPUOFF seja zerada e assim volte a rotina principal. A figura 13 mostra um trecho dela.

VII. BENEFÍCIOS

O projeto apresenta uma alternativa tecnológica que trará e/ou melhorará a qualidade de vida dos deficientes auditivos, com uma comunicação que abrange um grande número de pessoas e não fica restrita apenas ao grupo de pessoas que comunicam-se em LIBRAS.

REFERÊNCIAS

- [1] Surdos no Brasil, site: <http://www.surdo.com.br/surdos-brasil.html>.
- [2] Módulo Bluetooth HC-05, site: <https://multilogica-shop.com/modulo-bluetooth-hc-05>
- [3] Apesar de avanços, surdos ainda enfrentam barreiras de acessibilidade, site: <http://www.brasil.gov.br/cidadania-e-justica/2016/09/apesar-de-avancos-surdos-ainda-enfrentam-barreiras-de-acessibilidade>.
- [4] Coder-Tronic, site: <http://coder-tronics.com/msp430-adc-tutorial/>.

VIII. Anexos

```
WDTCTL = WDTPW + WDTHOLD;
if (CALBC1_1MHZ == 0xFF)
{
    while (1);
}

DCOCTL = 0;
BCSCTL1 = CALBC1_1MHZ;
DCOCTL = CALDCO_1MHZ;
P1SEL = BIT1 + BIT2;
P1SEL2 = BIT1 + BIT2;
P1DIR |= BIT6 + BIT0;
P1OUT &= ~(BIT6 + BIT0);

UCA0CTL1 |= UCSSEL_2;
UCA0BR0 = 104;
UCA0BR1 = 0;
UCA0MCTL = UCBRS0;
UCA0CTL1 &= ~UCSWRST;

IE2 |= UCA0RXIE;
__bis_SR_register(LPM0_bits + GIE);
Rx_Data = UCA0RXBUF;
__bic_SR_register_on_exit(LPM0_bits);
switch (Rx_Data)
{
    case 0x41:
        TA0CCTL0 &= ~CCIE;
        P1SEL &= ~BIT6;
        P1OUT |= BIT6 + BIT0;
        break;

    case 0x42:
        TA0CCTL0 &= ~CCIE;
        P1SEL &= ~BIT6;
        P1OUT &= ~(BIT6 + BIT0);
        break;
}
```

Figura 8: Código Bluetooth.

```

/*****
*
*                               DEFINIÇÕES
*
*****/
#ifndef I2C_USCI_H
#define I2C_USCI_H

// Endereços
#define MPU6050_ADDRESS 0x68
#define BQ32000_ADDRESS 0x68
#define DS1307_ADDRESS 0x68
#define LM92_ADDRESS 0x48

/*****
*
*                               Função
*
*****/
void I2C_USCI_Init(unsigned char addr); //Iniciando I2C
void I2C_USCI_Set_Address(unsigned char addr); //Alterar o endereço do escravo
unsigned char I2C_USCI_Read_Byte(unsigned char address); //ler 1 byte
//Ler muitos Byte
unsigned char I2C_USCI_Read_Word(unsigned char Addr_Data,unsigned char *Data, unsigned char Length);
//Escrever 1 Byte
unsigned char I2C_USCI_Write_Byte(unsigned char address, unsigned char Data);

void I2C_USCI_Init(unsigned char addr)
{
    P1SEL |= BIT6 + BIT7;           // Atribua pinos I2C a USCI_B0
    P1SEL2 |= BIT6 + BIT7;          // Atribua pinos I2C a USCI_B0
    UCB0CTL1 |= UCSWRST;            // Enable SW reset
    UCB0CTL0 = UCMST+UCMODE_3+UCSYNC; // I2C Master, modo síncrono
    UCB0CTL1 = UCSSEL_2+UCSWRST;    // USAR SMCLK, Mantenha SW resetada
    UCB0BR0 = 40;                   // fSCL = SMCLK/40 = ~400kHz
    UCB0BR1 = 0;
    UCB0I2CSA = addr;               // Setando endereço escravo
    UCB0CTL1 &= ~UCSWRST;           // Limpar a SW resetada, retomar a operação
}

void I2C_USCI_Set_Address(unsigned char addr)

```

Figura 9.1 - Código MCP6050 parte 1

```

void I2C_USCI_Set_Address(unsigned char addr)
{
    UCB0CTL1 |= UCSWRST;
    UCB0I2CSA = addr;           // Setando endereço escravo
    UCB0CTL1 &= ~UCSWRST;      // Limpar a SW resetada, retomar a operação
}

unsigned char I2C_USCI_Read_Byte(unsigned char address)
{
    while (UCB0CTL1 & UCTXSTP);
    UCB0CTL1 |= UCTR + UCTXSTT; // I2C TX, Iniciando

    while (!(IFG2 & UCB0TXIFG));
    UCB0TXBUF = address;

    while (!(IFG2 & UCB0TXIFG));

    UCB0CTL1 &= ~UCTR;          // I2C RX
    UCB0CTL1 |= UCTXSTT;        // I2C Iniciando
    IFG2 &= ~UCB0TXIFG;

    while (UCB0CTL1 & UCTXSTT);
    UCB0CTL1 |= UCTXSTP;
    return UCB0RXBUF;
}

unsigned char I2C_USCI_Read_Word(unsigned char Addr_Data, unsigned char *Data, unsigned char Length)
{
    unsigned char i=0;
    while (UCB0CTL1 & UCTXSTP); // Loop até I2C STT é enviado
    UCB0CTL1 |= UCTR + UCTXSTT; // I2C TX, start condition

    while (!(IFG2 & UCB0TXIFG));

    IFG2 &= ~UCB0TXIFG;         // Limpar USCI_B0 TX int flag
    if(UCB0STAT & UCNACKIFG) return UCB0STAT;
}

```

Figura 9.2 - Código MCP6050 parte 2


```

while (!(IFG2&UCB0TXIFG));
IFG2 &= ~UCB0TXIFG;           // Limpar USCI_B0 TX int flag
if(UCB0STAT & UCNACKIFG) return UCB0STAT;
UCB0TXBUF = Addr_Data;

while (!(IFG2&UCB0TXIFG));
if(UCB0STAT & UCNACKIFG) return UCB0STAT;

UCB0CTL1 &= ~UCTR;             // I2C RX
UCB0CTL1 |= UCTXSTT;           // I2C Condição de início
IFG2 &= ~UCB0TXIFG;           // Limpar USCI_B0 TX int flag
while (UCB0CTL1 & UCTXSTT);     // Loop until I2C STT is sent
for(i=0;i<(Length-1);i++)
{
    while (!(IFG2&UCB0RXIFG));
    IFG2 &= ~UCB0TXIFG;         // Limpar USCI_B0 TX int flag
    Data[i] = UCB0RXBUF;
}
while (!(IFG2&UCB0RXIFG));
IFG2 &= ~UCB0TXIFG;           // limpar USCI_B0 TX int flag
UCB0CTL1 |= UCTXSTP;           // I2C parando a condição depois do 1º TX
Data[Length-1] = UCB0RXBUF;
IFG2 &= ~UCB0TXIFG;           // limpar USCI_B0 TX int flag
return 0;
}

unsigned char I2C_USCI_Write_Byte(unsigned char address, unsigned char data)
{
    while (UCB0CTL1 & UCTXSTP);
    UCB0CTL1 |= UCTR + UCTXSTT;

    while (!(IFG2&UCB0TXIFG));
    if(UCB0STAT & UCNACKIFG) return UCB0STAT;
    UCB0TXBUF = address;

```

Figura 9.3 - Código MCPU6050 parte 3

```

    while (UCB0CTL1 & UCTXSTT);          // Loop until I2C STT is sent
    for(i=0;i<(Length-1);i++)
    {
        while (!(IFG2&UCB0RXIFG));
        IFG2 &= ~UCB0TXIFG;              // Limpar USCI_B0 TX int flag
        Data[i] = UCB0RXBUF;
    }
    while (!(IFG2&UCB0RXIFG));
    IFG2 &= ~UCB0TXIFG;                  // limpar USCI_B0 TX int flag
    UCB0CTL1 |= UCTXSTP;                  // I2C parando a condição depois do 1º TX
    Data[Length-1] = UCB0RXBUF;
    IFG2 &= ~UCB0TXIFG;                  // limpar USCI_B0 TX int flag
    return 0;
}

unsigned char I2C_USCI_Write_Byte(unsigned char address, unsigned char data)
{
    while (UCB0CTL1 & UCTXSTP);
    UCB0CTL1 |= UCTR + UCTXSTT;

    while (!(IFG2&UCB0TXIFG));
    if(UCB0STAT & UCNACKIFG) return UCB0STAT;
    UCB0TXBUF = address;

    while (!(IFG2&UCB0TXIFG));
    if(UCB0STAT & UCNACKIFG) return UCB0STAT;
    UCB0TXBUF = data;

    while (!(IFG2&UCB0TXIFG));
    if(UCB0STAT & UCNACKIFG) return UCB0STAT;
    UCB0CTL1 |= UCTXSTP;
    IFG2 &= ~UCB0TXIFG;
    return 0;
}
#endif /* I2C_USCI */

```

Figura 9.4 - Código MCPU6050 parte 4

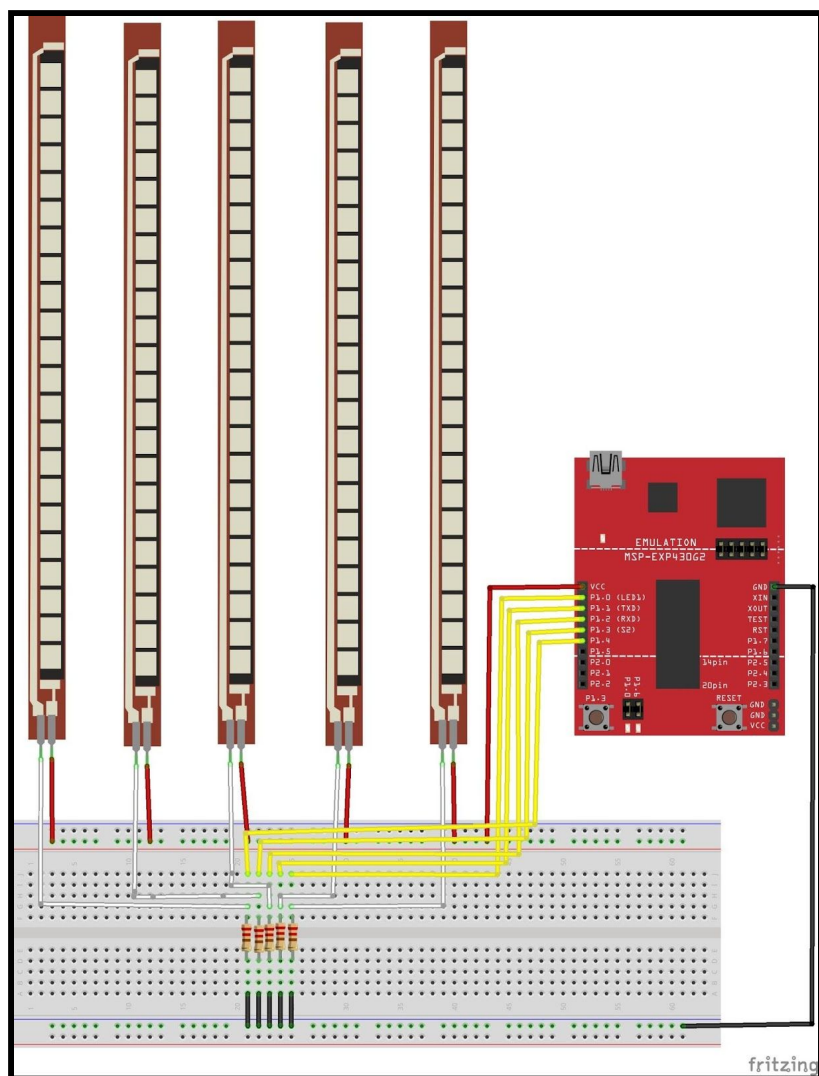


Figura 10 - Configuração para leitura das tensões.

```

#include <msp430.h>
/**
 * main.c
 */
void ConfigureAdc(void);
#define ADC_CHANNELS 4

unsigned int samples[ADC_CHANNELS];

int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;          // stop watchdog timer
    ADC10CTL0 &= ~ENC;

    while(1){
        while (ADC10CTL1 & BUSY);
        ConfigureAdc();
        ADC10SA = (unsigned int)samples;

    }
    return 0;
}

void ConfigureAdc(void)
{
    ADC10CTL1 |= INCH_3 + CONSEQ_1 + ADC10SSEL_3 + SHS_0;
    ADC10CTL0 |= SREF_0 + ADC10SHT_0 + MSC + ADC10ON + ADC10IE;
    ADC10AE0 |= BIT3 + BIT2 + BIT1 + BIT0;
    ADC10DTC1 = ADC_CHANNELS; //ADC_CHANNELS defined to 5
    ADC10CTL0 |= ENC + ADC10SC;
}

```

Figura 11 - Conversão AD

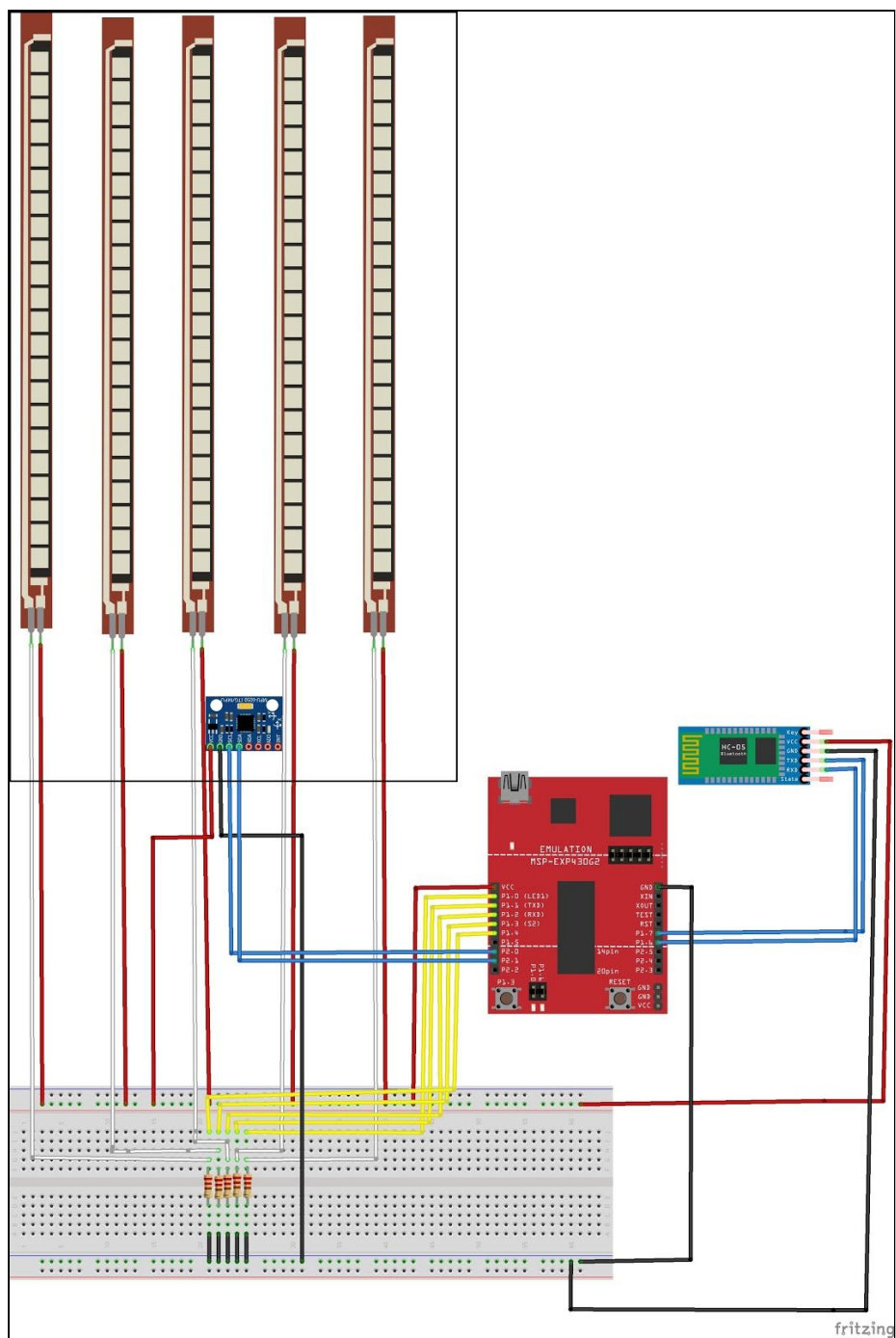


Figura 12 - Esquemático do projeto com seus devidos sensores.


```
.  
. |  
__bis_SR_register(CPUOFF + GIE);  
. |  
. |  
. |  
  
#pragma vector=ADC10_VECTOR  
__interrupt void ADC10_ISR(void)  
{  
    __bic_SR_register_on_exit(CPUOFF);  
}
```

Figura 13 - Código de interrupção.