

# ***Luva Tradutora de Libras***

## ***SMART GLOVE***

*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. INTRODUÇÃO

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 led) 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.

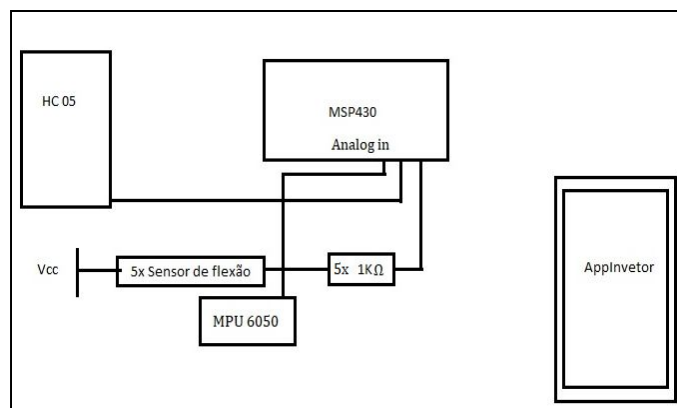
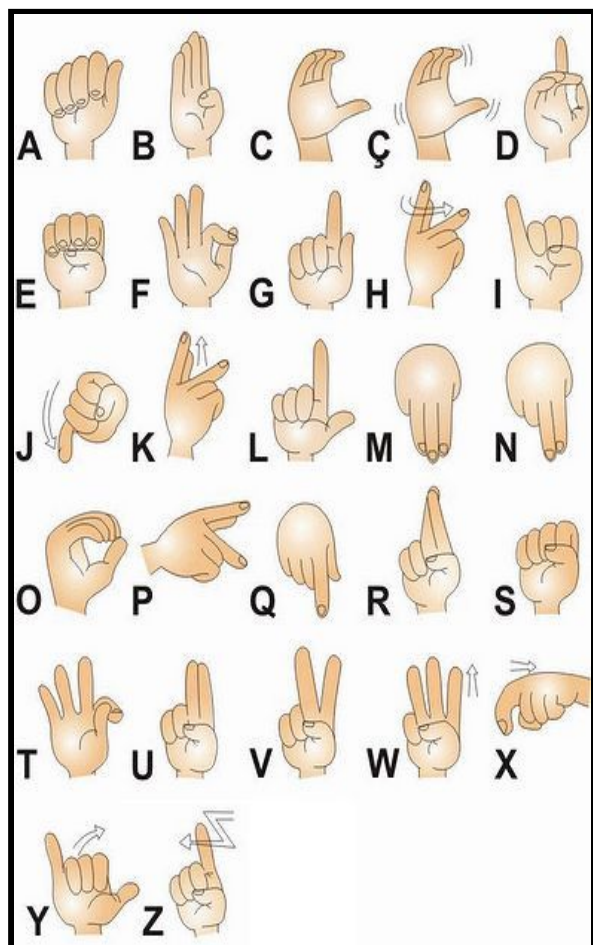


Figura 2 - Esquema de ligações na placa.

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	-	-
Resistores de 220Ω e de 2k2Ω	-	-
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 I<sup>2</sup>C, interfaceando com a MSP430 através dos pinos SDA e SCL ( pinos analógicos), GND (terra) e alimentação que varia de 3-5 V. Apresenta também I<sup>2</sup>C 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.

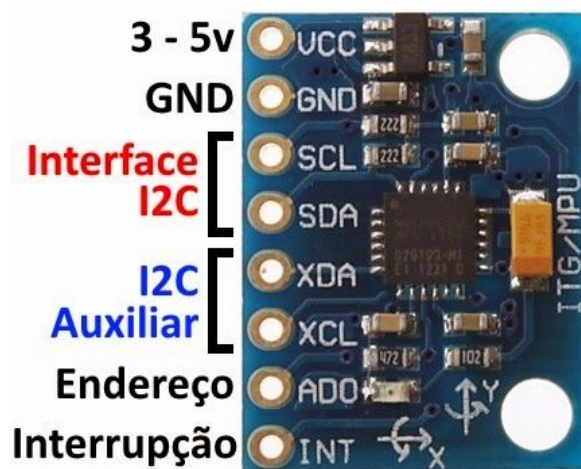


Figura 3 - Módulo MPU-6050.

## 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 papel adquirido pelo lápis fornece uma resistência variável e dependendo do comprimento do papel pintado, terá uma resistência associada respeitando a segunda lei de ohm.

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

$\rho$  - Resistividade que depende do material

$\ell$  - 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.

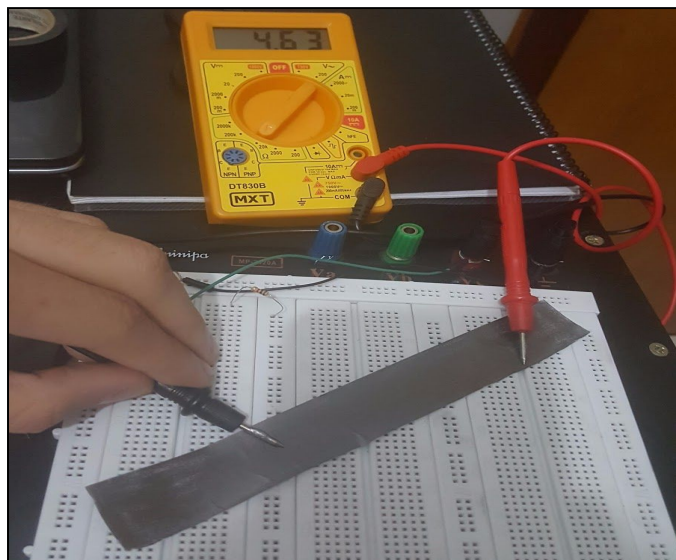


Figura 5 - Papel Pintado de lápis.



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



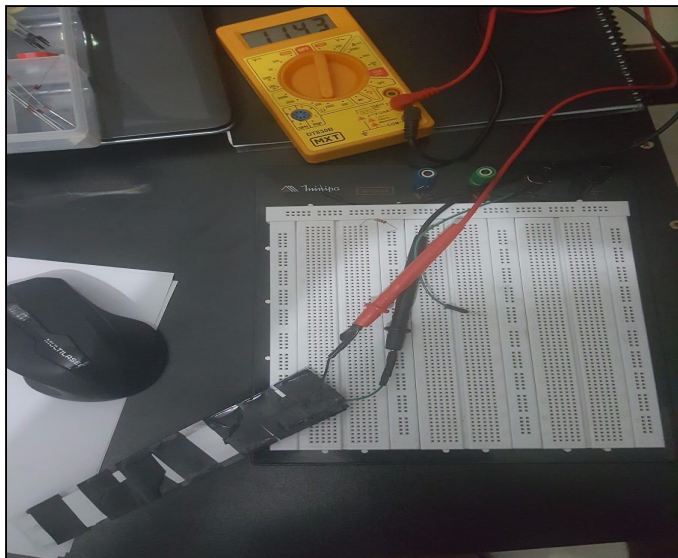


Figura 7 - Sensor Flexível.

## VI. Descrição do software

O software utilizado na elaboração dos códigos foi o Code Composer Studio v7 e IAR workbench 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.

Por fim temos o código completo na figura 14.1 até a figura 14.8, já contendo a rotina de decisão para cada letra a ser mandada. Essa rotina foi feita baseada nos dados lidos do divisor de tensão feito com o sensor de flexão e os resistores. Depois dessa rotina temos praticamente algumas rotinas de interrupção e declaração de outras funções.

## VII. RESULTADOS

Na implementação do módulo MPU6050, que é o Acelerômetro e giroscópio, irá utilizar as portas analógicas P1.6 e P1.7. Obteve-se êxito na criação das funções, que são: Iniciar a comunicação, Leitura de dados e Escrita de dados, porém houve dificuldade da implementação do protocolo de comunicação, que com o tempo foram resolvidas e o módulo funcionou normalmente. Algumas vezes acontecia de o módulo “travar” a MSP, mas isso parou de acontecer durante outros testes

Ao que se diz respeito à conversão AD os resultados foram bem satisfatórios. Conseguiu-se ler muitos valores de tensão diferentes ao variar o sensor de flexão. A variação de tensão era bem baixa, o que possibilitaria futuramente colocar uma faixa de valores para cada dedo na hora de fazer a lógica de saída de cada letra, não precisando assim fazer nenhuma média. Nessa parte a conversão de múltiplos pinos acabava por atrapalhar na comunicação UART, que acabava mandando algum lixo para a tela do celular. Depois de muitas tentativas a solução encontrada foi bem simples: escrever 0x00 no UCA0TXBUF dentro da interrupção, o que resolveu esse problema por completo.

Na comunicação bluetooth foi possível ver pelo menos duas letras ao realizar uma simples lógica com um sensor de flexão onde para um valor menor que 512 mostrava a letra A e para maior do que isso a letra Z. Para ver se a letra saiu mesmo foi usado um app chamado Bluetooth ssp. Há a possibilidade da porta p1.1 e p1.2 atrapalharem a conversão AD, mas no futuro isso foi sanado.

Com o código todo pronto e compilado foi possível mandar várias letras para a tela do celular. Na mudança de uma letra para outra poderia acontecer de mandar uma outra letra aleatória devido ao fato das conversões estarem acontecendo em um intervalo de tempo muito curto e até mesmo à variação dos sensores de flexão. Porém esses empecilhos não atrapalharam tanto na visualização das letras e pode-se distinguir muito bem as mais variadas letras.

## VII. CONCLUSÃO

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. Levando em consideração os aspectos técnicos do projeto, pode-se dizer que o projeto utiliza conceitos que envolve funções, entradas analógicas, saídas analógicas, comunicação serial síncrona e assíncrona.

Mesmo com todos esses conceitos envolvidos, o que tornava o projeto um tanto quanto ousado, conseguiu-se cumprir os requisitos do projeto. Fomos capazes de mostrar várias letras no display do celular.

No mais o projeto fez com que o nosso conhecimento sobre microcontroladores aumentasse mais, possibilitando assim fazer projetos até um tanto quanto mais complexos futuramente.

## VIII. 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/>.
- [5] Let's draw a flex sensor 1º florinas, site: <https://www.youtube.com/watch?v=1oF6iY-OO4Q>.

## IX. 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 MCPU6050 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



```

#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;

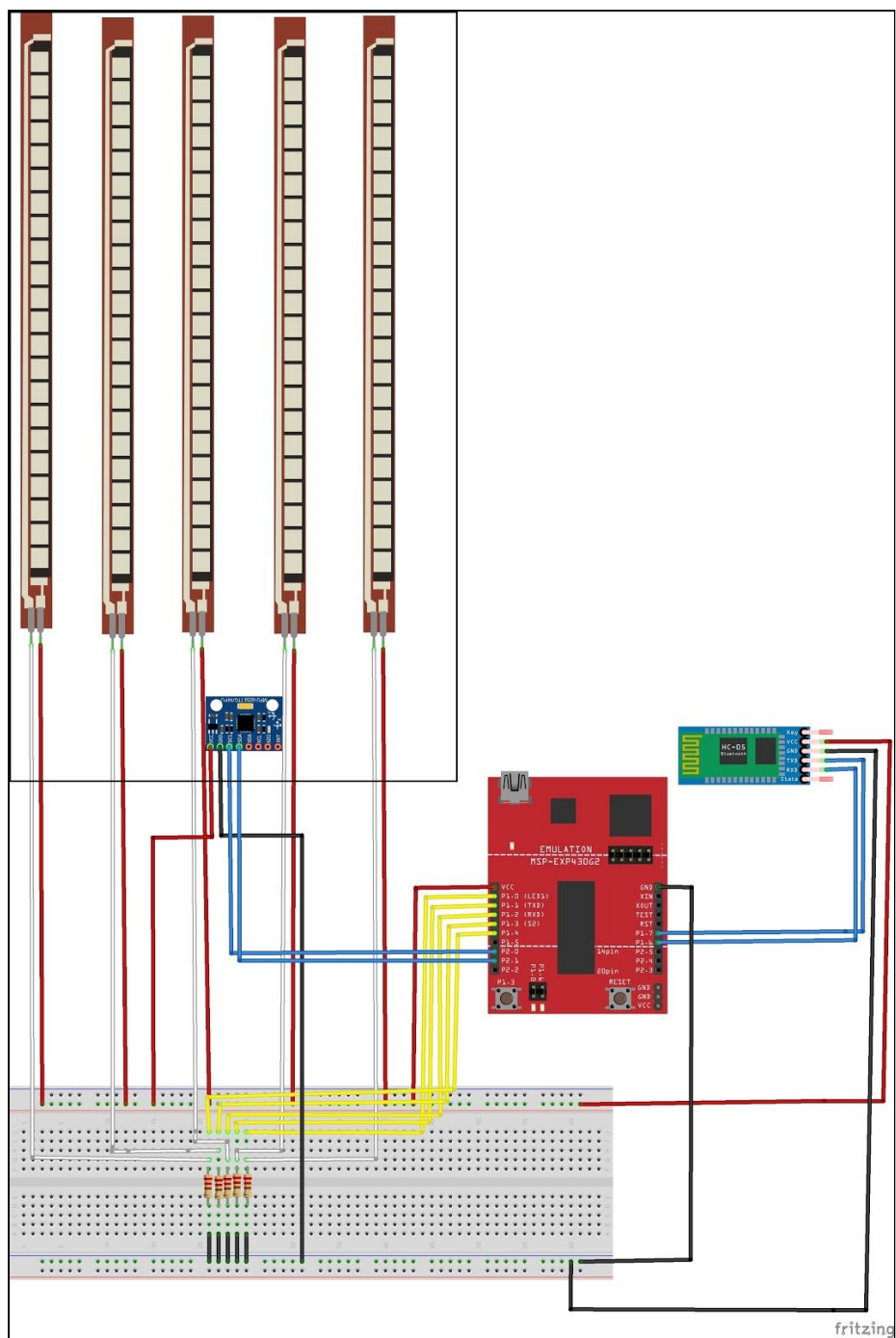
    }
    return 0;
}

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





```

.
.
.|
__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.

```

1 #include <mcp430g2553.h>
2 #define CANAIS_ADC 6
3
4 unsigned int amostras[CANAIS_ADC];
5 unsigned int POLEGAR, MEDIO, MINDINHO, INDICADOR;
6
7 // Acelerômetro
8 unsigned char RX_Data[6];
9 unsigned char TX_Data[2];
10 unsigned char RX_ByteCtr;
11 unsigned char TX_ByteCtr;
12 int xAccel;
13 int sel_xAccel;
14 int yAccel;
15 int sel_yAccel;
16 int zAccel;
17 int sel_zAccel;
18 unsigned char slaveAddress = 0x68; // Seta endereço para a MPU-6050
19 // 0x68 para ADD pin=0
20 // 0x69 para ADD pin=1
21
22 const unsigned char PWR_MGMT_1 = 0x6B; // MPU-6050 registrador de endereço
23 const unsigned char ACCEL_XOUT_H = 0x3B; // MPU-6050 registrador de endereço
24 const unsigned char ACCEL_XOUT_L = 0x3C; // MPU-6050 registrador de endereço
25 const unsigned char ACCEL_YOUT_H = 0x3D; // MPU-6050 registrador de endereço
26 const unsigned char ACCEL_YOUT_L = 0x3E; // MPU-6050 registrador de endereço
27 const unsigned char ACCEL_ZOUT_H = 0x3F; // MPU-6050 registrador de endereço
28 const unsigned char ACCEL_ZOUT_L = 0x40; // MPU-6050 registrador de endereço
29
30 //Funções do Acelerômetro
31
32 void i2cInit(void);
33 void i2cWrite(unsigned char);
34 void i2cRead(unsigned char);
35 void coleta_valores_mpu(void);
36 int coleta_valores_MPU(int sel_xAccel,int sel_yAccel,int sel_zAccel);
37
38 void ConfigureAdc(void)
39 {
40     {
41
42         ADC10CTL1 |= INCH_5 + CONSEQ_3 + ADC10SSEL_3; // tamanho 6, multiplas conversões sucessivas, modo de conversão de multi canais
43         ADC10CTL0 |= SREF_0 + ADC10SHT_0 + MSC + ADC10ON + ADC10IE;
44         ADC10AEO0 |= BIT5 + BIT4 + BIT3 + BIT0; // bits correspondentes a cada pino da MSP
45         ADC10DTC1 = CANAIS_ADC; //4 canais
46         ADC10CTL0 |= ENC + ADC10SC; // inicializa a conversão
47     }
48 }

```

Figura 14.1 - Código Projeto Final - Declarações de funções e variáveis.

```

49 void main(void)
50 {
51     WDTCTL = WDTPW + WDTHOLD;           //Parar o WDT
52 // clock de 1Mhz
53 DCOCTL = 0;
54 BCSCTL1 = CALBC1_1MHZ;
55 DCOCTL = CALDCO_1MHZ;
56
57 P1SEL |= BIT1 + BIT2 + BIT6 + BIT7; //P1.1TX e P1.2RX
58 P1SEL2 |= BIT1 + BIT2 + BIT6 + BIT7;
59 P1DIR &=~BIT2;
60 P1DIR &=~BIT1;
61 // setar a baud rate para 9800bps
62 UCA0CTL1 |= UCSSEL_2;
63 UCA0BR0 = 104;
64 UCA0BR1 = 0;
65 UCA0MCTL = UCBRS0;
66 UCA0CTL1 &= ~UCSWRST;
67 ConfigureAdc(); // configura o adc
68
69 while (1)
70 {
71     // carrega as amostras para as variáveis correspondentes a cada dedo.
72     POLEGAR = amostras[0];
73     INDICADOR = amostras[1];
74     MEDIO = amostras[2];
75     MINDINHO = amostras[5];
76     ADC10CTL0 &= ~ENC; // Encerra a conversão AD
77     while (ADC10CTL1 & BUSY); // garante que o conversor AD não esteja ocupado para iniciar novas conversões
78     ADC10SA = (unsigned int)amostras; // ADC10SA recebe o endereço da variável amostra, que no fim receberá também os valores da conversão AD
79     ADC10CTL0 |= ENC + ADC10SC;
80
81     __bis_SR_register(CPUOFF + GIE);
82 //Coleta os Valores do MPU, Acelerometro
83     sel_xAccel=1;
84     sel_yAccel=0;
85     sel_zAccel=0;
86     xAccel = coleta_valores_MPU(sel_xAccel,sel_yAccel,sel_zAccel);
87
88     sel_xAccel=0;
89     sel_yAccel=1;
90     sel_zAccel=0;
91     yAccel = coleta_valores_MPU(sel_xAccel,sel_yAccel,sel_zAccel);
92
93     sel_xAccel=0;
94     sel_yAccel=0;
95     sel_zAccel=1;
96     zAccel = coleta_valores_MPU(sel_xAccel,sel_yAccel,sel_zAccel);

```

Figura 14.2 - Código Projeto Final - Configurações de comunicação, pinos e loop principal.

```

98 //logica de saída para as letras
99
100 if ( (POLEGAR >490) && (INDICADOR < 300)&&(MEDIO < 200 ) &&(MINDINHO< 200))
101 {
102     UCA0TXBUF = 'A';
103 }
104 else if ( (POLEGAR < 360) && (INDICADOR > 400)&&(MEDIO > 319) && (MINDINHO > 400))
105 {
106     UCA0TXBUF = 'B';
107 }
108 else if ( (POLEGAR > 490) && ((INDICADOR >= 300) && (INDICADOR <= 400)) && ((MEDIO >= 200) && (MEDIO <= 319)) &&((MINDINHO <= 400) && (MINDINHO >= 200)) && (xAccel < 0) )
109 {
110     UCA0TXBUF = 'C';
111 }
112 else if ( ((POLEGAR >= 360 && POLEGAR<=490)) && (INDICADOR > 400) && ((MEDIO >= 200) && (MEDIO <= 319)) &&((MINDINHO <= 400) && (MINDINHO >= 200)))
113 {
114     UCA0TXBUF = 'D';
115 }
116 else if ( (POLEGAR < 360) && (INDICADOR < 300)&& (MEDIO < 200 ) &&(MINDINHO< 200) && (xAccel > 0))
117 {
118     UCA0TXBUF = 'E';
119 }
120 else if ( (POLEGAR >= 360 && POLEGAR<=490) && ((INDICADOR >= 300) && (INDICADOR <= 400)) && (MEDIO > 319) &&(MINDINHO > 400) && (xAccel > 0) )
121 {
122     UCA0TXBUF = 'F';
123 }
124 else if ( (POLEGAR >= 360 && POLEGAR<=490) && (INDICADOR > 400) && (MEDIO < 200 ) &&(MINDINHO< 200))
125 {
126     UCA0TXBUF = 'G';
127 }
128 else if ( (POLEGAR >= 360 && POLEGAR<=490) && (INDICADOR > 400) &&(((MEDIO >= 200) && (MEDIO <= 319))) &&(MINDINHO< 200) && (xAccel < 0) && (yAccel< 0) && (zAccel< 0))
129 {
130     UCA0TXBUF = 'H'; //USAR O GIROSCOPIO PARA DIFERENCIAR
131 }
132 else if ( (POLEGAR < 360) && (INDICADOR < 300) && (MEDIO < 200 ) &&(MINDINHO > 400))
133 {
134     UCA0TXBUF = 'I';
135 }
136 else if ( (POLEGAR < 360) && (INDICADOR < 300) && (MEDIO < 200 ) &&((MINDINHO <= 400) && (MINDINHO >= 200)))
137 {
138     UCA0TXBUF = 'J';
139 }
140 else if ( (POLEGAR >= 360 && POLEGAR<=490) && (INDICADOR< 300) &&((MEDIO >= 200) && (MEDIO <= 319)) &&(MINDINHO< 200))
141 {
142     UCA0TXBUF = 'K'; // USAR O GIROSCOPIO PARA DIFERENCIAR
143 }

```

Figura 14.3 - Código Projeto Final - Continuação do loop principal já com as condições de saída.

```

144 else if ( (POLEGAR > 490) && (INDICADOR > 400) && (MEDIO < 200 ) &&(MINDINHO< 200))
145 {
146     UCA0TXBUF = 'L';
147 }
148 else if ( (POLEGAR >= 360 && POLEGAR<=490) && (INDICADOR > 400) && (MEDIO > 319) &&(MINDINHO< 200) && (yAccel < 0) && (zAccel < 0))
149 {
150     UCA0TXBUF = 'M';
151 }
152 else if ( (POLEGAR >= 360 && POLEGAR<=490) && (INDICADOR > 400) && (MEDIO > 319) &&(MINDINHO< 200) && (yAccel > 0) && (zAccel > 0))
153 {
154     UCA0TXBUF = 'N';
155 }
156 else if ( (POLEGAR >= 360 && POLEGAR<=490) && ((INDICADOR >= 300) && (INDICADOR <= 400)) &&((MEDIO >= 200) && (MEDIO <= 319)) && ((MINDINHO <= 400) && (MINDINHO >= 200)))
157 {
158     UCA0TXBUF = 'O';
159 }
160 else if ( (POLEGAR >= 360 && POLEGAR<=490) && (INDICADOR > 400)&& ((MEDIO >= 200) && (MEDIO <= 319)) &&(MINDINHO< 200) && (xAccel < 0) && (yAccel > 0) && (zAccel > 0))
161 {
162     UCA0TXBUF = 'P'; // USAR O GIROSCOPIO PARA DIFRENCIAR
163 }
164 else if ( (POLEGAR >= 360 && POLEGAR<=490) && (INDICADOR > 400) && (MEDIO < 200 ) && (MINDINHO< 200))
165 {
166     UCA0TXBUF = 'Q';
167 }
168 else if ( (POLEGAR < 360) && (INDICADOR > 400) && ((MEDIO > 264) && (MEDIO <= 380)) &&(MINDINHO< 200))
169 {
170     UCA0TXBUF = 'R';
171 }
172 else if ( (POLEGAR < 360) && (INDICADOR < 300) && (MEDIO < 200 ) && (MINDINHO< 200) && (xAccel < 0))
173 {
174     UCA0TXBUF = 'S';
175 }
176 else if ( (POLEGAR >= 360 && POLEGAR<=490) && (INDICADOR < 300)&& (MEDIO > 319) &&(MINDINHO > 400) && (xAccel < 0))
177 {
178     UCA0TXBUF = 'T';
179 }
180 else if ( (POLEGAR < 360) && (INDICADOR > 400) && (MEDIO > 319) && (MINDINHO< 200) && (xAccel < 0) && (yAccel< 0) && (zAccel< 0))
181 {
182     UCA0TXBUF = 'U';
183 }
184 else if ((POLEGAR < 360) && (INDICADOR > 400) && (MEDIO > 319) &&(MINDINHO< 200) && (xAccel > 0) && (yAccel < 0 ) && (zAccel < 0))
185 {
186     UCA0TXBUF = 'V';
187 }
188 else if ( (POLEGAR < 360) && ((INDICADOR >= 300) && (INDICADOR <= 400))&&(MEDIO < 200 ) &&(MINDINHO< 200))
189 {
190     UCA0TXBUF = 'X';
191 }

```

Figura 14.4 - Código Projeto Final - Continuação das condições de saída.



```

192 else if ((POLEGAR < 360) && (INDICADOR > 400) && (MEDIO > 319) && (MINDINHO > 400) && (xAccel < 0) && (yAccel > 0) && (zAccel > 0))
193 {
194     UCA0TXBUF = 'W';
195 }
196 else if ((POLEGAR > 490) && (INDICADOR < 300) && (MEDIO < 200) && (MINDINHO > 400))
197 {
198     UCA0TXBUF = 'Y';
199 }
200 else if ((POLEGAR < 360) && (INDICADOR > 400) && (MEDIO < 200) && (MINDINHO < 200))
201 {
202     UCA0TXBUF = 'Z';
203 }
204 else if ((POLEGAR > 490) && ((INDICADOR <= 670) && (INDICADOR > 498)) && ((MEDIO >= 200) && (MEDIO <= 319)) && ((MINDINHO <= 400) && (MINDINHO >= 200)) && (xAccel > 0))
205 {
206     UCA0TXBUF = 'Ç';
207 }
208 else {
209     UCA0TXBUF = '.';
210 }
211 }
212 }
213 }
214 }
215 #pragma vector = ADC10_VECTOR //Interrupção do ADC
216 __interrupt void ADC10_ISR(void)
217 {
218     UCA0TXBUF = 0x00; //escreve 0 no TXBUF para evitar que a conversão AD jogue algum lixo aqui.
219 }
220 __bic_SR_register_on_exit(CPUOFF);
221 }
222 }
223 void i2cInit(void)
224 {
225     // set up I2C module
226     UCB0CTL1 |= UCSWRST; // Habilita reset SW
227     UCB0CTL0 = UCMST + UCMODE_3 + UCSYNC; // I2C Master, modo síncrono
228     UCB0CTL1 = UCSSEL_2 + UCSWRST; // Use SMCLK, keep SW reset
229     UCB0BR0 = 10; // fSCL = SMCLK/12 = ~100kHz
230     UCB0BR1 = 0;
231     UCB0CTL1 &= ~UCSWRST; // Clear SW reset, resume operation
232 }

```

Figura 14.5 - Código Projeto Final - Fim das condições de saída, início das interrupções e de outras funções.

```

234 void i2cWrite(unsigned char address)
235 {
236     __disable_interrupt();
237     UCB0I2CSA = address;           // Carregar endereço do escravo
238     IE2 |= UCB0TXIE;              // Habilita a interrupção do TX
239     while(UCB0CTL1 & UCTXSTP);     // garante que a condição de parada seja mandada
240     UCB0CTL1 |= UCTR + UCTXSTT;    // modo TX e condição de start
241     __bis_SR_register(CPUOFF + GIE); // dorme enquanto UCB0TXIFG é setado
242 }
243
244 void i2cRead(unsigned char address)
245 {
246     __disable_interrupt();
247     UCB0I2CSA = address;           // Carregar endereço do escravo
248     IE2 |= UCB0RXIE;              // Habilita a interrupção do RX
249     while(UCB0CTL1 & UCTXSTP);     // garante que a condição de parada seja mandada
250     UCB0CTL1 &= ~UCTR;             // modo RX
251     UCB0CTL1 |= UCTXSTT;           // Condição de Start
252     __bis_SR_register(CPUOFF + GIE); // dorme enquanto UCB0RXIFG é setado
253 }
254 // USCIAB0TX_ISR
255 #pragma vector = USCIAB0TX_VECTOR
256 __interrupt void USCIAB0TX_ISR(void)
257 {
258     if(UCB0CTL1 & UCTR)             // modo TX (UCTR == 1)
259     {
260         if (TX_ByteCtr)             // Verdadeiro se tiver bytes
261         {
262             TX_ByteCtr--;            // Decrementa RX_ByteCtr
263             UCB0TXBUF = TX_Data[TX_ByteCtr]; // Carrega o TX buffer
264         }
265         else                         // nenhum byte para mandar
266         {
267             UCB0CTL1 |= UCTXSTP;     // I2C stop condition
268             IFG2 &= ~UCB0TXIFG;     // Limpar a flag int USCI_B0 TX
269             __bic_SR_register_on_exit(CPUOFF); // Sair do LPM0
270         }
271     }
}

```

Figura 14.6 - Código Projeto Final - Funções i2cWrite e i2cRead.

```

272 else // (UCTR == 0) // modo RX
273 {
274     RX_ByteCtr--; // Decrementa RX_ByteCtr
275     if (RX_ByteCtr) // RxByteCtr != 0
276     {
277         RX_Data[RX_ByteCtr] = UCB0RXBUF; // gravar o byte recebido
278         if (RX_ByteCtr == 1) // Apenas um byte sobrando?
279             UCB0CTL1 |= UCTXSTP; // Gerar uma condição de parada para I2C
280     }
281     else // RxByteCtr == 0
282     {
283         RX_Data[RX_ByteCtr] = UCB0RXBUF; // Pegar o último bit recebido
284         __bic_SR_register_on_exit(CPUOFF); // Sair do LPM0
285     }
286 }
287 }
288
289 int coleta_valores_MPU(int sel_xAccel,int sel_yAccel,int sel_zAccel){
290
291
292
293     // Inicializa a I2C
294     i2cInit();
295
296     // Acorda the MPU-6050
297     slaveAddress = 0x68; // Endereco da MPU-6050
298     TX_Data[1] = 0x6B; // Endereco do registrador PWR_MGMT_1
299     TX_Data[0] = 0x00; // Setar o registrador para 0 (acorda a MPU-6050)
300     TX_ByteCtr = 2;
301     i2cWrite(slaveAddress);
302
303
304     // apontar para o registrador ACCEL_ZOUT_H na MPU-6050
305     slaveAddress = 0x68; // Endereco da MPU-6050
306     TX_Data[0] = 0x3B;
307     TX_ByteCtr = 1;
308     i2cWrite(slaveAddress);

```

Figura 14.7 - Código Projeto Final - fim das funções i2cRead e i2cWrite e começo da função de coleta de valores..

```

309
310 // Lê os dois bytes do data e armazena eles nos eixos
311 slaveAddress = 0x68; // Endereço da MPU-6050
312 RX_ByteCtr = 6;
313 i2cRead(slaveAddress);
314 xAccel = RX_Data[5] << 8; // MSB
315 xAccel |= RX_Data[4]; // LSB
316 yAccel = RX_Data[3] << 8; // MSB
317 yAccel |= RX_Data[2]; // LSB
318 zAccel = RX_Data[1] << 8; // MSB
319 zAccel |= RX_Data[0]; // LSB
320
321
322
323 __no_operation(); // Setar o breakpoint >>aqui<< e ler
324
325 //Lógica para retorno de valor da função
326 if((sel_xAccel==1) && (sel_yAccel==0) && (sel_zAccel==0)){
327 return xAccel;
328 }
329 else if((sel_xAccel==0) && (sel_yAccel==1) && (sel_zAccel==0)){
330 return yAccel;
331 }
332 else if((sel_xAccel==0) && (sel_yAccel==0) && (sel_zAccel==1)){
333 return zAccel;
334 }
335
336
337 return 0;
338 }

```

Figura 14.8 - Código Projeto Final - Fim da função de coleta de valores.



Figura 15 - Smart Glove.