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*— Usando 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. Com isso ela poderá se comunicar com uma pessoa não muda.**

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 deficiente auditivos têm em se comunicar, esse dispositivo permitirá a comunicação com pessoas que não sofre 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.

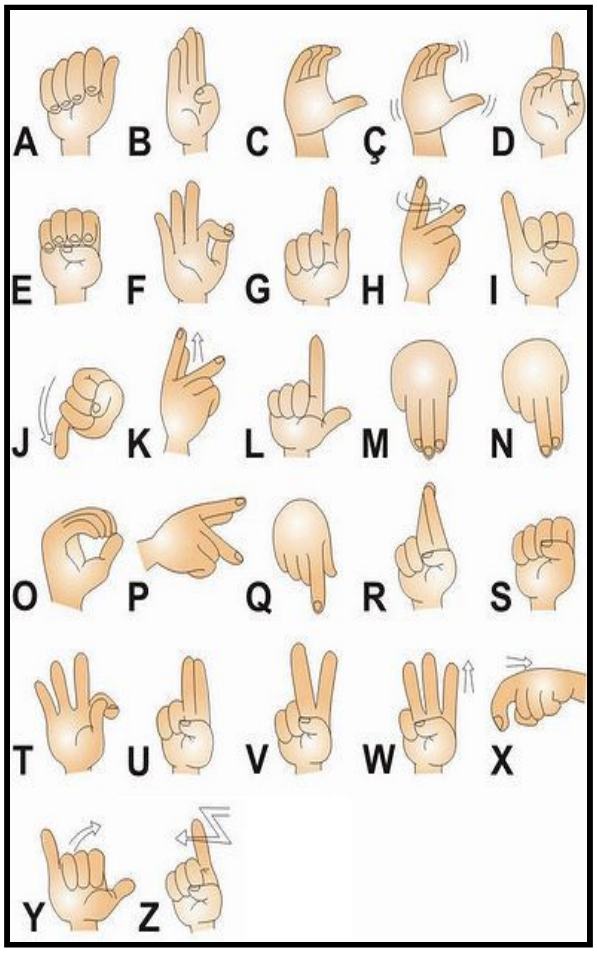

Figura 1 - Alfabeto de LIBRAS

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 esquemático resumido das ligações do projeto.

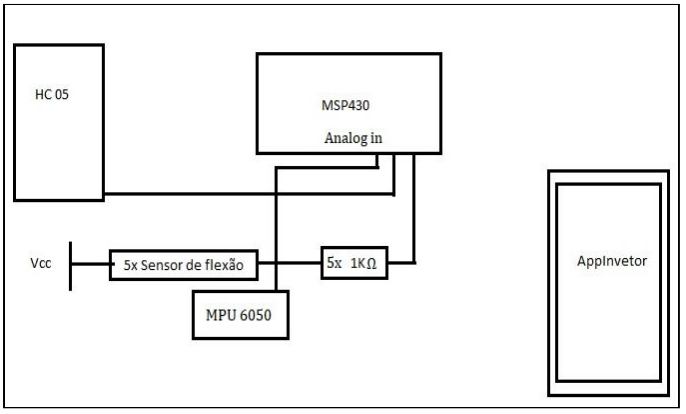

Figura 2 - Esquema de ligações na placa.

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 essas variações de tensão e mandar para o AppInventor, onde elas serão analisadas e o

aplicativo irá mostrar na tela qual letra corresponde àquelas variações de tensão.

Para letras que precisam de movimento para serem reconhecidas será utilizado 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 HC05	-	-
Resistor1kΩ	-	-
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.

Os resistores de 1kΩ serão usados em circuitos divisores de tensão com os 5 sensores de flexão feitos.

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.

VI. Descrição do software

O software utilizado na elaboração dos códigos foi o Code Composer Studio v7 e IAR embedded workbench IDE. Testou-se principalmente os códigos para o módulo bluetooth e para o acelerometro/giroscópio.

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.

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] 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.

[3]

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 3: 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 4.1 - 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,Inciando

	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 4.2 - Código MCPU6050 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 4.3 - 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 4.4 - Código MCPU6050 parte 4