Fisoterapia ludica com MSP430

Uso de MSP430 para criação de uma fisoterapia ludica

Autor: Antônio Aldísio de Sousa Alves Fereira Filho
Universidade de Brasília
Microcontroladores e Microprocessadores
Gama- DF
aldisiofilho@gmail.com

Autor: Daniel Carvalho de Sousa
Universidade de Brasília
Microcontroladores e Microprocessadores
Gama- DF
danielcsdf@gmail.com

I. Introdução

A fisioterapia é uma das principais formas de reabilitação para uma pessoa vítima de doenças, traumas ou acidentes. Com ela, é possível obter a ajuda necessária para que essa pessoa volte a executar tarefas que, sem medidas auxiliares, seriam muitas vezes impossíveis e as cujas chances de recuperação seriam cada vez mais improváveis.

De acordo com o Conselho Nacional de Fisioterapia (COFFITO), podemos definir a fisioterapia: *"É uma ciência da saúde que estuda, previne e trata os distúrbios cinéticos funcionais intercorrentes em órgãos e sistemas do corpo humano, gerados por alterações genéticas, por traumas e por doenças adquiridas, na atenção básica, média complexidade e alta complexidade. Fundamenta suas ações em mecanismos terapêuticos próprios, sistematizados pelos estudos da biologia, das ciências morfológicas, das ciências fisiológicas, das patologias, da bioquímica, da biofísica, da biomecânica, da cinesia, da sinergia funcional, e da cinesia patológica de órgãos e sistemas do corpo humano e as disciplinas comportamentais e sociais."*. A partir dessa definição, observa-se a vasta aplicação da fisioterapia no tratamento ou na prevenção de diversos tipos de doenças e traumas.

Dentre essas aplicações, destaca-se neste artigo aquele referente ao tratamento de traumas ou de doenças que impossibilitaram os movimentos dos membros superiores tais como ombros, cotovelos e pulsos. As soluções fisioterapêuticas atuais, nesses casos, têm como um dos principais obstáculos a falta de aparelhos ou métodos que incentivam pacientes ao tratamento ativo correto, especialmente ao público infantil e aos portadores de necessidades especiais, o que dificulta o tratamento das patologias e impõem necessidades de fisioterapeutas com especialidade em pedagogia até casos não muito complexos, o que em muitas ocasiões se torna desnecessário e exigem demanda desses profissionais, o que pode elevar os custos desses tratamentos.

A partir dessa problemática, o projeto em questão propõe uma solução lúdica que consiste em um aparelho eletrônico-visual em forma de jogo interativo controlado por meio de movimentos específicos de acordo com o tratamento fisiológico dos membros superiores, de modo que a progressão e a evolução do tratamento avançam à medida que se completam os níveis e cada fase do jogo.

Dessa forma, espera-se obter uma solução eficiente no tratamento fisiológico de contusões ou de sequelas geradas a partir de alguma doença.

II. Justificativa

Como a fisioterapia é um trabalho de repetição de movimentos. Para crianças e idoso que realização algumas fisioterapias podem considerar que essa atividade como monótona e entediante.

A partir disso existe uma linha de trabalho na fisioterapia que desenvolve atividades lucidas durante o atendimento medico assim o paciente não visualiza que estar realizando atividade. Segundo Rafael Dias, a atividade lúdica apresenta um elemento motivador dentro do processo de reabilitação do paciente. O paciente que desenvolve atividade lúdica não encara o tratamento tão seriamente como se fosse ao modelo tradicional.

Boa parte dos estudos realizados utiliza o console Wii U para desenvolvimento das atividades lúdicas. O valor monetário e acessibilidade do console dentro de um centro de fisioterapia ou hospital pode ficar inviável para o tratamento. O desenvolvimento de um sistema que utiliza o MSP430 e a possibilidade criação de mecanismo de baixo custo e sendo portátil. A diferença de portabilidade pode ser mostrada nas imagens abaixo:

Figura 1 – pessoas jogando wii U

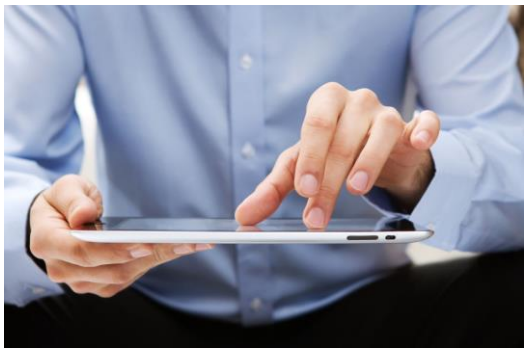

Figura 2 – ideia de como será o sistema com MSP430

O projeto não visa construir um tablete para a fisioterapia, mas a ideia de portabilidade é comparada a um tablet, por esse motivo utilizamos uma foto com tablet para mostrar a ideia do projeto.

Porque não usar um tablet? Um sistema pronto e que é necessário apenas criar um software em vez de criar uma a partir do uso de MSP430. A utilização do msp430 baseia-se em um sistema único dentro do produto no qual podemos criar uma adaptação melhor para o usuário e proporcionar um baixo custo de produção, para o publico alvo que iremos trabalhar.

Por fim, iremos utilizar o MSP430 para desenvolver o sistema de fisioterapia lúdica para atividades nos membros do pulso e cotovelo. Esse sistema será um jogo cuja interface gráfica é uma matriz de *leds* onde o paciente irar guiar o *led* a partir de movimentos rastreados via acelerômetro.

III. **Objetivo**

O desenvolvimento desse projeto tem como objetivo de construir um jogo que auxilie na fisioterapia de membros superiores, pulso, cotovelo e ombro. Esse projeto visa transmitir uma sensação ao usuário a fim de que ele não perceba que esteja realizando a fisioterapia.

Assim construindo um sistema que possa trabalhar com grupo de pessoas que tenham dificuldades com a fisioterapia tradicional.

Portanto, objetivo desse projeto é criar uma imersão para o paciente, onde ele possa trabalhar movimentos fisioterápicos sem que perceba que esteja realizando a atividade de fisioterapia.

IV. **Requisitos**

Tendo em vista que o projeto visa implementar de forma eletrônica um tratamento lúdico fisioterapêutico para auxiliar o paciente que possui dificuldades de seguir o tratamento convencional, além de outras características, os requisitos são:

1.Conter um programa atrativo ao publico alvo.

2.Conter sistema de sensor eletrônico que detecta os movimentos corretos do tratamento.

3. Acompanhar a evolução do paciente conforme o uso do equipamento.

4. O Produto deve ser constituído de um material leve e portátil.

V. **Benefício**

O beneficio desse projeto é desenvolver um jogo de baixo custo onde possamos implementar em hospitais públicos para o tratamento de pacientes. Além do baixo custo temos a questão da mobilidade do equipamento que pode ser carregador igualmente ao um tablet.

Agora pensando no desenvolvimento da matéria iremos poder compreender melhor o uso do MSP430 na pratica, utilizando comunicação i2c para adiquirir dados, e utilizando uma interface gráfica..

VI. **Desenvolvimento**

A metodologia principal utilizada para a elaboração dos códigos foi por meio de engenharia reversa, utilizando soluções sugeridas por desenvolvedores e complementando com o uso de algoritmos aprendidos na disciplina, auxiliados com esquemáticos de ligação e de interfaceamento, para tornar o programa estéticamente agradável e funcional. Em todos os códigos foi utilizada a ferramenta Energia para testar as funcionalidades iniciais do sistema antes de configurá-lo a nível de registradores por meio do software Code Composer Studio.

Nos códigos envolvendo o controle das matrizes de leds foram testados o funcionamentode cada matriz, para identificar como os dados foram enviados. Foram compilados trechos de códigos programas de exemplos base disponibilizados pela companhia desenvolvedora do Energia e, a partir da ferramenta Code Composer Studio, foram traduzidos para o nível de registradores com algumas adaptações nos códigos originais e com alguns complementos.

Na parte de hardware foi realizada a construção de um circuito com mosfet para realizar um elevador de tensão, a fim de alimentar as entradas de dados e enables do CI MAX7219. A partir desse CI foram controlados, via transmissão serial, os dados que seriam carregados em cada registrador responsável pela multiplexação das linhas e colunas da matriz.

A. Materiais utilizados

O módulo com MAX7119 e matriz 8x8 de Led vermelho O processamento será feito a partir da lauchpad contendo MSP430G2553. A Aquisição de dados referentes aos membros superiores será feita a partir de do módulo MPU-6050 e, no circuito elevador de tensão, será utilizado o BS170.

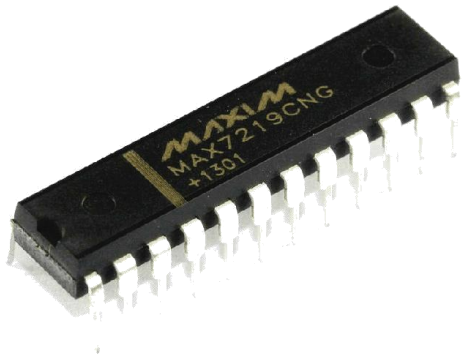

Figura 3 *- CI controlador da matriz de led (MAX7219).*

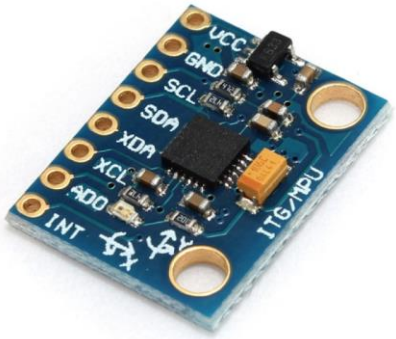

Figura 4 *– Módulo giroscópio/acelerômetro (MPU6050).*

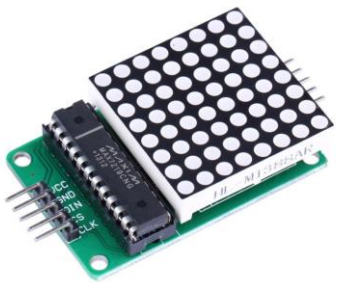

Figura 5 *– Modulo da Matriz de led*

B. Métodos

Os componentes foram dispostos em protoboard e conectados por meio de "jumpers". Na ligação do CI MAX7219 deve ser ultizada alimentação de 5V, bem como as entradas, cuja tensão mínima suportada é 3,5V. Para isso, foi conectado à saída do MSP transistor para amplificar a tensão das saídas do MSP430.

Foi desenvolvida uma lógica na qual em apenas uma matriz de led, onde acedemos todos os led dos cantos, um led fica apagado em posições aleatórias, dependendo de cada fase, indicando o buraco onde o jogador tem que encaminhar a bolinha na matriz.

O desenvolvimento do MPU-6050 no Code Composer, a comunicação de I2C foi adaptada de um desenvolvedor, no qual baseia-se no uso de interrupção para realizar a escrita e leitura de dados pelo protocolo I2C.

A montagem da interface gráfica se deu a partir de jumpers, nos quais conectam os dois módulos utilizados, um para a exibição do tempo e nível e outro o "mapa" do jogo, isto é, o jogo é exibido em uma matriz enquanto a outra exibe a contagem do tempo para resolver todas as 8 fases do jogo.

VII. **Resultados e discussões**

Em relação à montagem do circuito no protoboard, encontra-se na Figura 6.

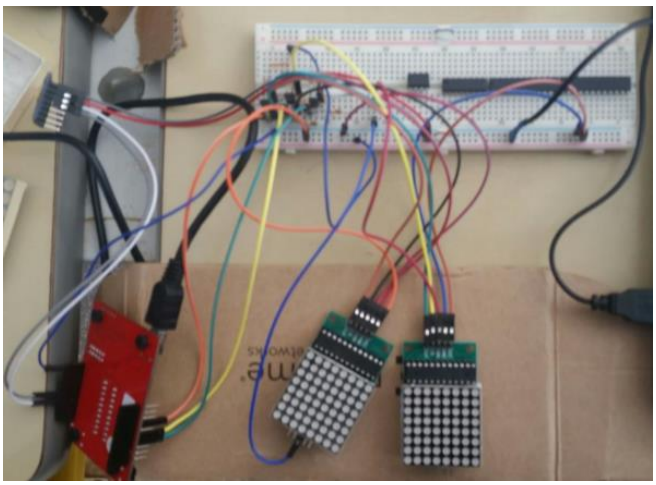

Figura 6 *– Circuito montando*

No código foi necessário definir alguns registradores no MAX 7219 para conseguir fazer com que ele recebesse os dados via comunicação serial, sem as quais o módulo não entra em funcionamento. A principal delas é a definição do registrador ShutDown, que deve ser inicializado como "0". Outro registrador fundamental do CI é o displayTest, o qual deve também ser setado para "0" para seu funcionamento normal. Os demais registradores, ilustrados pela Figura 7 são responsáveis por configurações de exibição nos displays.

```
setReg(max7219_reg_scanLimit, 0x07);
setReg(max7219_reg_decodeMode, 0x00);
setReg(max7219_reg_shutdown, 0x01);
setReg(max7219_reg_displayTest, 0x00);
setReg(max7219_reg_intensity, 0x0A & 0x0F);
```

Figura 7 *–definição dos registradores no MAX 7219*

Como necessitamos criar uma bolinha andando colocamos a função *delay* para atualizar a posição da

bolinha dependendo da inclinação/aceleração do módulo acelerômetro.

O MSP 430 controla o MAX 7219da seguinte forma: primeiramente, através da função scanByte ele varre um byte, que é o valor enviado aos registradores do CI. Isso se dá a partir do deslocamento de uma mascara “0x01” e com a operação AND com a posição que é deslocada pelo laço, varrendo dessa forma bit a bit enviado para o pino “DIN” do MAX7219. O pino CLK é acionado depois de setar um bit para sincronizar o envio de dados com o clock de envio. Posteriormente a isso, para o envio de uma palavra, o pino LOAD, é acionado e o conteúdo enviado por meio do pino DIN é carregado nos registradores do CI.

```
void scanByte(unsigned int sDATA)
{
  int i = 8;
  unsigned mask;
  while(i > 0) {
    mask = 0x01 << (i - 1);
    P1OUT &= ~CLK;
    if (sDATA & mask){
    P1OUT |= DIN;
    }else{
    P1OUT &= ~DIN;
    }
    P1OUT |= CLK;
    --i;
  }
} // end scanByte;
```

Figura 8 *–varrendo byte a byte do MAX 7219*

Após varrer cada bit a ser enviado, utilizamos a função setReg para enviar um conjunto de 16 bits ao MAX7219. Vale ressaltar que o conjunto de 16 bits contém o endereço da instrução e o dado a ser carregado no registrador em questão, ilustrado pela Figura 9 (b)

```
void setReg(unsigned int reg, int col)    // initialize  all  MAX7219's in the system
{
  int c = 0;
  P1OUT &= ~LOAD;   // begin
  for ( c = 1; c <= maxInUse; c++) {
      scanByte(reg);    // specify register
      scanByte(col);    //((data & 0x01) * 256) + data >> 1); // put data
  }
  P1OUT |= LOAD;
}
```

(a)

Table 1. Serial-Data Format (16 Bits)

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
X	X	X	X	ADDRESS				MSB			DATA				LSB

(b)

Figura 9 *–Inicializando MAX 7219 com a função setReg (a) e configuração dos dados enviados ao MAX7219 (b).*

Para o MSP saber onde está cada ponto acesso precisamos criar um mapa da matriz, que será trocado pela função changeMap, que ler ponto a ponto e salva os dados em um vetor chamado *limits.*

```
void changeMap()
{
int i = 0;
    for(i = 0; i < 8; i++){
        if(i == portalIndex) // Se a coluna mover-se
        {
            if((portalIndex > 0) && (portalIndex < 7))
                limits[portalIndex] = portalX;
            else
                //if((portalY != 128) && (portalY != 1))
                    limits[portalIndex] = ~portalY;
        }
        else
            limits[i] = limits0[i];
    }
}
```

Figura 10 *– Função changeMap*

Para construir a borda achamos o endereço nos registradores e deixamos sempre ligado assim para construir a borda ligada e como o *changeMap* identifica sempre a borda acessa e o único led que encontra-se apagado.

```
limits0[0] = 0b11111111;
limits0[1] = 0b10000001;
limits0[2] = 0b10000001;
limits0[3] = 0b10000001;
limits0[4] = 0b10000001;
limits0[5] = 0b10000001;
limits0[6] = 0b10000001;
limits0[7] = 0b11111111;
```

Figura 11 *–Definindo borda*

Criamos um laço responsável para enviar as coordenada atual, a partir da leitura envida pelo módulo MPU6050 para conseguir fazer a movimentação da bolinha. Isso se deu a partir da análise da linha que foi mudada (deslocada) pelos valores obtidos do MPU6050, atualizando apenas essa linha.

```
for(;;){


    // Laço responsável por enviar as coordenadas certas ao LED
    // Ele atualiza o valor enviado para o buffer[8] com as coordenadas atuais
    for(i = 0; i < 8; i++){
        if(i == coluna) // Se a coluna mover-se
            buffer[coluna] = limits[i] | linha; // Analise feita a partir da coluna, essa posição que receberá a nova posição
        else                                    // da linha ... isto é, será enviado o par (linha,coluna) para a matriz.
            buffer[i] = limits[i];  // Nas demais colunas, a matriz será preenchida com as bordas do jogo.
    }
```

Figura 12 *– Movimentação*

Como é um jogo precisamos criar uma parte randômica para quando finalizar um jogo ele automaticamente definir uma nova fase. Fizemos isso a partir da interrupção de leitura e escrita do MPU6050, pelo motivo de essa interrupção ter um tempo variável, dependendo das condições satisfeitas no laço infinito, isto é, como essa interrupção não intervalos de tempos definidos, conseguimos uma aleatoriedade mais fiel.

```
#pragma vector = USCIAB0TX_VECTOR
__interrupt void USCIAB0TX_ISR(void)
{
    //__bis_SR_register(GIE);
    if(UCB0CTL1 & UCTR)
    {
        if (TX_ByteCtr)
        {
            TX_ByteCtr--;                                /
            UCB0TXBUF = TX_Data[TX_ByteCtr];
        }
        else
        {
            UCB0CTL1 |= UCTXSTP;
            IFG2 &= ~UCB0TXIFG;

            __bic_SR_register_on_exit(CPUOFF);

          //-------------change_map-------------//
            portalIndex ++;
            if(portalIndex == 8)
            {
                portalIndex = 0;
            }
            portalY = (portalY << 1);
            if(portalY == 128)
                portalY = 0x02;
            portalX ^= 0b10000001;
          //----------------------------------//
        }
    }
    else // (UCTR == 0)                    // RX mode
    {
```

Figura 13 *– Função de leitura e escrita funcionando como gerador randomico*

O protocolo de comunicação do MPU 6050 que é baseado em I2C. No qual o MSP é o mestre e acelerômetro é o escravo. Ele é dividido em 3 funções na qual fazemos a inicialização da comunicação, uma na qual fazemos o sincronismo dos clock do MSP com o MPU, outra referente a escrita dos dados do acelerômetro no microcontrolador e a ultima, a leitura desses dados dentro do módulo.

A função i2cInt, figura 14, ele sincroniza o clock do acelerômetro com o MSP e seta os registradores para iniciar a comunição I2C.

```
void i2cInit(void)
{
    // set up I2C module
    UCB0CTL1 |= UCSWRST;                    // Enable SW reset
    UCB0CTL0 = UCMST + UCMODE_3 + UCSYNC;           // I2C Master, synchronous mode
    UCB0CTL1 = UCSSEL_2 + UCSWRST;          // Use SMCLK, keep SW reset
    UCB0BR0 = 10;                       // fSCL = SMCLK/12 = ~100kHz
    UCB0BR1 = 0;
    UCB0CTL1 &= ~UCSWRST;                   // Clear SW reset, resume operation
}
```

Figura 14 *– Função i2cInt*

Já na função de i2cRead (Figura 15) e i2cWrite (figura 16) temos a troca de informação entre o escravo e o servo. O Read trabalha limpando os dados antigos recebidos e trava a nova aquisição de dados, para não pegar metade do sinal do ponto 1 e outra parte do sinal ser do ponto 2. Com isso conseguimos trabalhar com uma certa precisão. Para Write ele desabilita a interrupção feita pela função Read e salva novos valores dentro dos registradores.

```
void i2cRead(unsigned char address)
{
    //__disable_interrupt();
    IE2 &= ~UCB0RXIE;                       // limpa a flad  do interrupt i2c
    UCB0I2CSA = address;                    // Load slave address
    IE2 |= UCB0RXIE;                    // Enable RX interrupt
    while(UCB0CTL1 & UCTXSTP);              // Ensure stop condition sent
    UCB0CTL1 &= ~UCTR;                  // RX mode
    UCB0CTL1 |= UCTXSTT;                    // Start Condition
    //__bis_SR_register(GIE);
    //while((IFG2 & UCB0RXIFG) == 0);
    __bis_SR_register(LPM0_bits + GIE);         // sleep until UCB0RXIFG is set ...
}
```

Figura 15 *– Função i2cRead*

```
void i2cWrite(unsigned char address)
{
    //__disable_interrupt();
    IE2 &= ~UCB0TXIE;                       // desabilita o interrupt
    UCB0I2CSA = address;                    // Load slave address
    IE2 |= UCB0TXIE;                    // Enable TX interrupt
    while(UCB0CTL1 & UCTXSTP);              // Ensure stop condition sent
    UCB0CTL1 |= UCTR + UCTXSTT;             // TX mode and START condition
    //__bis_SR_register(GIE);
    //while((IFG2 & UCB0TXIFG) == 0);
    __bis_SR_register( LPM0_bits + GIE);        // sleep until UCB0TXIFG is set ...
}

void i2cRead(unsigned char address)
```

Figura 16 *– Função i2cWrite*

Dentro da função do Main utilizamos primeiro a função i2cInt e colocamos o registrador Tx_Data com o endereço 0 e 1 com os registradores do MPU. E cada endereço tem o tamanho de dois bytes.

```
i2cInit();
slaveAddress = 0x68;                    // Endereço do módulo MPU-6050/
TX_Data[1] = 0x6B;                      // Endereço do registrador PWR_MGMT_1
TX_Data[0] = 0x00;                      // limpa o registrador(inicializa o MPU-6050)
TX_ByteCtr = 2;
i2cWrite(slaveAddress);                 // Envia os dados de TX_data para o MPU-6050

slaveAddress = 0x68;                    // MPU-6050 address
TX_Data[1] = 0x3C;                      // calibrando a escala de aceleração
TX_ByteCtr = 2;
i2cWrite(slaveAddress);
```

Figura 17 *– Inicialização da função i2cInt*

Como identificar a direção de cada valor para uma direção? A partir disso, criamos duas variáveis xAccel e yAccel e colocamos o endereço 5 e 4 para aquele, 6 e 7 para este. Com isso, realizamos a mesma ideia do botão, porém agora tempos que calibrar o sensor para que quando mais rápido incline ele mais rápido a bolinha mexe e para ela não passar do espaço da matriz.

Não realizamos uma calibração mais precisa, pois não tivemos tempo focado para poder melhorar o código.

```
slaveAddress = 0x68;                    // MPU-6050 address
RX_ByteCtr = 6;
i2cRead(slaveAddress);

xAccel  = RX_Data[5] << 8;              // MSB
xAccel |= RX_Data[4];                   // LSB
yAccel  = RX_Data[3] << 8;
yAccel |= RX_Data[2];

left = map(xAccel,0,20000,limite_medio,limite_superior); // new range: 0 -> limite_medio -> limite-superior
right = map(xAccel,0,-20000,limite_medio,limite_superior);
up = map(yAccel,0,20000,limite_medio,limite_superior);
down = map(yAccel,0,-20000,limite_medio,limite_superior);
```

Figura 17 *– Posicionando as direções*

O projeto ele trabalha com 2 interrupções diferentes, a primeira acontece para alocação dos dados nos registradores do MPU, a segunda é a realização da contagem de tempo mostrada em uma das matriz.

```
#pragma vector = USCIAB0TX_VECTOR
__interrupt void USCIAB0TX_ISR(void)
{
    //__bis_SR_register(GIE);
    if(UCB0CTL1 & UCTR)                 // TX mode (UCTR == 1)
    {
        if (TX_ByteCtr)                     // TRUE if more bytes remain
        {
            TX_ByteCtr--;               // Decrement TX byte counter
            UCB0TXBUF = TX_Data[TX_ByteCtr];    // Load TX buffer
        }
        else                        // no more bytes to send
        {
            UCB0CTL1 |= UCTXSTP;            // I2C stop condition
            IFG2 &= ~UCB0TXIFG;         // Clear USCI_B0 TX int flag

            __bic_SR_register_on_exit(CPUOFF);  // Exit LPM0

          //-------------change_map-------------//
            portalIndex ++;
            if(portalIndex == 8)
            {
                portalIndex = 0;
            }
            portalY = (portalY << 1);
            if(portalY == 128)
                portalY = 0x02;
            portalX ^= 0b10000001;
          //------------------------------------//
        }
    }
    else // (UCTR == 0)                 // RX mode
    {
        RX_ByteCtr--;                       // Decrement RX byte counter
        if (RX_ByteCtr)                     // RxByteCtr != 0
        {
            RX_Data[RX_ByteCtr] = UCB0RXBUF;    // Get received byte
            if (RX_ByteCtr == 1)            // Only one byte left?
            UCB0CTL1 |= UCTXSTP;            // Generate I2C stop condition
        }
        else                        // RxByteCtr == 0
        {
            RX_Data[RX_ByteCtr] = UCB0RXBUF;    // Get final received byte
            __bic_SR_register_on_exit(CPUOFF);  // Exit LPM0
        }
```

Figura 17 *– Interrupção de leitura e escrita*

Para o contador do jogo precisamos realizar uma outra interrupção. Porque não colocamos as interrupções juntas? O motivo pelo qual não fizemos isso é, pelo fato que a leitura era feita, a contagem de tempo mudava de forma não síncrona, além disso, a utilização de delay não é tão eficiente para contar tempo. Para isso, utiliazmos o timer A e, com a realização de outra interrupção, solucionamos o problema. O contador que criamos ele faz a contagem do de 0 a 99.

```
#pragma vector=TIMER0_A1_VECTOR
__interrupt void Timer_A0(void){
    LSB ++; // incrementa um algarismo no display
    if(LSB == 10){
        if(MSB == 9){   // A contagem não reinicia, isto é de 0 -> 99
            MSB = 9;
            LSB = 9;
        }else{
            LSB = 0;
            MSB++;  // incrementa outro algarismo do display
        }
    }
    TA0CTL &= ~TAIFG;
}
```

Figura 18 *– Interrupções do contador*

VIII. **Conclusão**

O projeto teve como ideia criar um sistema portátil de fisioterapia para pulso de uma forma lúdica. Não conseguimos fazer teste em um paciente ou um profissional da área que certifique nosso projeto para atividade realizada e, além disso, não tivemos um tempo hábil para construção de uma caixa onde pudemos embarcar tudo e o usuário tivesse acesso visual apenas a matriz de led.

Já dentro da área da engenharia o desenvolvimento do projeto aconteceu com uma certa dificuldade para entender como ligar a matriz e a utilização do MAX7249 a partir disso o projeto conseguiu andar tranquilamente, pois ficamos focado no código e na construção otimizada do sistema, principalmente na parte das interrupções.

Pelo lado do software a grande dificuldade que tivemos era realizar a alocação certa do registrador no MAX7219 e saber controlar onde acender o led do contador de fase

Em suma, foram desenvolvidas habilidades relacionadas a microcontroladores, comunicação I2C. Para um engenheiro eletrônico esse conhecimento é de grande importante em trabalhamos que podemos realizar no futuro relacionados a utilização de microcontroladores

IX. **Revisão Biobibliográfica**

[1] Datasheet MAX7219 , link: https://datasheets.maximintegrated.com/en/ds/MAX7219-MAX7221.pdf

[2] Datasheet Matrix 8X8, link: http://www.circuitstoday.com/wp-content/uploads/2016/04/8X8-Matrix-Pinout.png

[3] Datasheet MPU 6050, link https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf

[4] User guide MSP430, link http://www.ti.com/lit/ug/slau144j/slau144j.pdf

[5] https://www.coffito.gov.br/