Analisador de repetições.

Contador de exercícios para estação de musculação

Erick Antonio Correa dos Reis
15/0034156
Faculdade do Gama
Universidade de Brasília
St. Leste Projeção A - Gama Leste, Brasília - DF
Email: erickcorrareis@gmail.com

Tiago Avelino Ribeiro da Silva
15/0022662
Faculdade do Gama
Universidade de Brasília
St. Leste Projeção A - Gama Leste, Brasília - DF
Email: tiago.avelino1997@gmail.com

RESUMO -

Palavras-chave - MSP430; Academia; Microcontrolador; Monitor exercícios; Acompanhamento de exercícios.

I. Introdução

A estação de musculação é um equipamento muito utilizado em academias e também estão presentes em muitas residências e tem como sua principal vantagem um alto número de exercícios distintos.

Figura 1: Estação de musculação genérica.

Apesar de muito funcionais a estação de musculação pode dificultar a organização de seus utilizadores, principalmente em academias, causado justamente pelo seu ponto forte, a variedade de exercícios, sendo fácil de se perder quando é necessário utilizar o equipamento por um longo período de tempo.

Portanto, visando melhorar o controle dos exercícios, foi desenvolvido um dispositivo capaz de monitorar, de forma completa e mais clara, o andamento da realização dos exercícios, mostrando através de um display as informações necessárias para melhor aproveitamento da estação de musculação. Dentre estas informações as escolhidas para exibição foram: nome do exercício, número de séries e número de repetições.

Entretanto, visando montar um sistema mais simplificado, ao invés de inserir cada uma destas informações optamos pela criação dos níveis de exercícios, ou seja, cada exercício terá algumas opções de níveis que influenciarão no número de séries e repetições. São 6 opções de níveis que por sua vez também serão exibidas no display.

II. Objetivos

Construir um dispositivo capaz de monitorar exercícios e suas repetições, programado através do Launchpad MSP430, buscando facilidade de utilização de estações de musculação.

III. Descrição

Para a implementação deste dispositivo tem-se alguns pontos que merecem destaque, um deles é a forma como as informações são mostradas para o utilizador, visto que estas devem ser claras e de fácil compreensão. Outro ponto é a

precisão da contagem das repetições já que caso contrário poderiam gerar insatisfação e até mesmo lesões por esforço excessivo.

Portanto, buscando satisfazer tais necessidades, os componentes escolhidos para elaboração de dispositivo foram os seguintes:

- *Display LCD*

Figura 2: Display Nokia 5110

Para visualizar os exercícios a serem realizados e os valores obtidos nos sensores será utilizado o Display LCD Nokia 5110 em que se pode ter em uma mesma tela gráficos e textos com uma resolução de 84 X 48 pixels. Por se tratar de um display gráfico e utiliza comunicação serial do tipo SPI, em que se utiliza dois sinais de comunicação de dados, e que toda comunicação de dados acontece em ambas direções e é necessário enviar uma sequência de bits para desenhar no display a forma desejada. Foi encontrada uma biblioteca com a sequências dos vectores que desenham cada caracter (números, letras e símbolos) no display.

Todas as funções para inicialização e envio de dados para display via comunicação serial foram adaptadas para funcionar de maneira mais simples e direta.

Tabela 1: Configuração de pinos Display LCD

Display LCD	**MSP 430**
VCC	VCC
GND	GND
SCE	P1.0
RST	---
D/C	P1.1
DN (MOSI)	P1.7
SCLK	P1.5
LED	VCC

- *Sensor ultrassônico HC-SR04*

Figura 3: Sensor ultrassônico HC-SR04

Com o objetivo de oferecer a integração com a estação de musculação, será instalado um sensor ultrasônico para que possa se obter os dados referentes às repetições a serem realizadas.

Este sensor deve ser posicionado de forma a registrar a movimentação dos pesos que ficam na parte posterior do equipamento, possibilitando assim analisar se uma repetição foi feita. Por conta de sua localização será possível obter essa informação independentemente do exercício realizado na estação já que todos eles proporcionam uma movimentação nos pesos.

O sensor ultrassônico é capaz de medir distância porém sua utilização será restrita apenas para informar de se os pesos se moveram ou ficaram estáticos.

B. Software

Para um funcionamento eficiente do dispositivo alguns detalhes do código podem ser conferidos a seguir:

- *Display LCD*

O código encontrado para utilização do display era muito complexo e extenso, portanto só utilizou-se as funções realmente necessárias, dentre elas temos:

void writeToLCD(unsigned char dataCommand, unsigned char data). Identifica e envia um dado ou um comando para o display através do registrador de envio de dados da comunicação SPI.

void setAddr(unsigned char xAddr, unsigned char yAddr). Move o cursor para a posição desejada do display.

void clearLCD(). Limpa o display.

void initLCD(). Conjunto de instruções que envia dados específicos para inciar o display.

Ainda sobre o display, algumas funções foram modificadas para o melhor funcionamento no código principal.

void writeCharToLCD(char c). Esta função não contava com condição que identificava um *\0.*

void writeIntToLCD(int n). Função não funcionava pra inteiros com mais de um dígito.

Todas as funções, definições e conjunto de caracteres foram postos em uma biblioteca, *"Display.h"* .

- *Sensor ultrassônico*

Embora seu funcionamento seja simples para conseguir medir a distância para algum objeto foi necessário configurar uma interrupção para o pino *ECHO* e também para o Timer A.

int medir_dist(). Gera um pulso de 10 us para o pino TRIG e aguarda a resposta da interrupção gerada no pino ECHO e converte o valor recebido para centímetros.

Interrupção ECHO. Habilita o modo de comparação do Timer e retorna um tempo com alta precisão.

Interrupção Timer A. Contabiliza contagens de 1 ms.

- *Código principal*

Na função *main()* temos inicialmente detalhes básicos, como desabilitar o WDT e definir clocks, além disso para melhorar a organização do código temos as funções:

config_display(). Configura toda a comunicação SPI que será necessária para utilização do display.

config_sensor(). Configura os pinos ECHO e TRIG e também o Timer A em seu modo de comparação.

config_botoes(). Configura os pinos para os botões (reset e nível).

initLCD(). Como já citada inicia o display.

clearLCD(). Limpa o display.

tela_padrao(). Como parte do display sempre vai se manter constante, temos uma função que deixa o display com alguns textos já fixados, visto que não é necessário atualizar estes dados muitas vezes mas sim seus respectivos valores.

reset_ex(). Reinicia a sequência de exercícios, séries e repetições.

Após todas as configurações de pinos e periféricos da MSP430, partimos para a parte do código que vai, efetivamente, contabilizar os exercícios, séries e repetições. Tudo isso é feito através da função *exer(),* onde temos um código que trabalha com condições.

exer(). Determina o que ocorrerá com o display em detrimento das séries e repetições. A determinação ocorre através da comparação da repetição quando esta está com o valor nulo, assim é decrementada a série do exercício a ser executado . Posteriormente é analisado quando ambas série e repetições estão com valores nulos, é caso verdadeira essa afirmação é incrementado o exercício do banco de exercícios após um tempo de descanso predefinido por exercício. O decremento do valores da repetição são realizados pela função *medir dist()*, já explicada, tal valor de distância é ajustado de acordo com a especificação do equipamento. Após a realização de todos os exercícios no equipamento é chamada a função *fim().*

fim(). Tem a finalidade de encerrar o display com uma mensagem de término,ajustando a msp no modo de economia de energia.

interrupção BOTÕES. Determina o botão que ativou a interrupção da porta 2, caso seja o botão de reset zera o watchdog timer, atuando como reset, direccionando o display para a tela inicial. Caso o botão de nível é ativado, é realizado inicialmente o debounce do botão, em seguida é incrementado o nível, modificando as repetições e os exercícios realizados no instrumento.

Foi criada uma biblioteca *display.h* de fácil edição para modificar os valores das repetições de cada exercício tanto quanto a série do exercício e qual exercício é o definido para o treino.

No código *main* são definidas as linhas que serão utilizadas do display, tanto o espaçamento dos caracteres, e a posição no display de cada texto pré definido.

IV. RESULTADOS

Inicialmente foi realizada a montagem de forma simplificada na protoboard, utilizando o próprio vcc da placa do msp, através do circuito foi possível implementar o circuito na placa de circuito impresso (pci) reduzindo o número de mal contato, que ocorriam habitualmente durante a implementação na protoboard.

Com a implementação da pci o sensor de ultrassom apresentou um resultado com maior precisão, por conta da eliminação quase que completa do ruído gerado pela fiação com as entradas da launchpad da msp430, porém uma parte desse ruído permaneceu durante a coleção final dos resultados, isso se deve pelo fato da utilização da protoboard, melhorando o funcionamento do código e consequentemente melhor obtenção dos resultados.

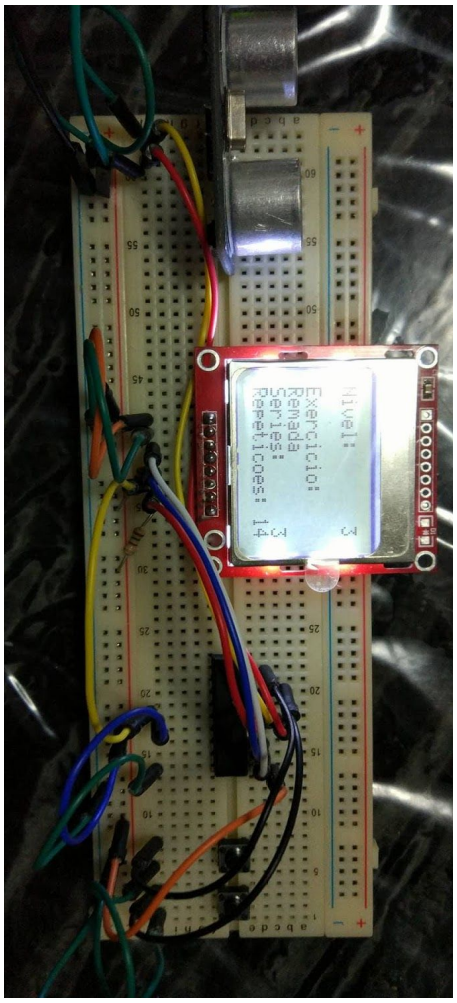

Figura 4: Primeiros testes projeto final.

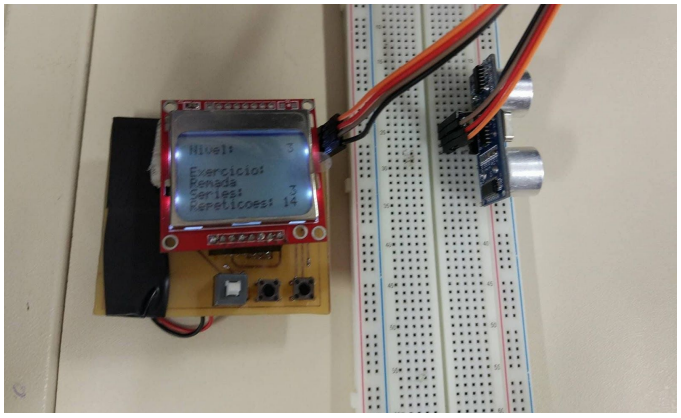

Figura 5: Implementação final do projeto.

Figura 6: Dispositivo final.

Através das imagens é possível visualizar de forma clara todas as informações necessárias durantes a realização de exercícios em uma estação de musculação. O sensor de ultrassom foi apresentado na protoboard, considerando como suporte no momento da apresentação.

Todos os exercícios, séries e repetições podem ser configurados através do arquivo *Exercicios.h*, neste arquivo temos algumas matrizes capazes de armazenar não só os vários exercícios mas também diferentes níveis, possibilitando que o equipamento seja utilizado de forma mais geral sem muitas configurações adicionais.

V. Conclusão

Com os resultados obtidos é possível ver o projeto em sua fase final, embora a implementação de sensor de batimento cardíacos tenha sido cancelada devido ao alto nível de ruído gerado pelo mesmo.

Com relação a comunicação sem fio optamos pela não implementação por conta do número de portas utilizados na msp430, que já haviam sido esgotadas e por conta da complexidade aliada ao tempo de entrega do projeto.

Apesar das modificações realizadas foi possível obter a implementação do projeto final com resultados satisfatórios quando compreendido pelo escopo do projeto, podendo assim melhorar a realização dos exercícios na unidade de musculação analisada e ainda monitorar com certeza o desenvolvimento total das repetições de cada exercício proposto pelo instrutor ao aluno.

VI. Revisão Bibliográfica

[1] GUIMARÃES, Anderson Eduardo; BARSOTTINI, Daniel; VIEIRA, Rodolfo de Paula. **ADEQUAÇÃO DAS ATIVIDADES FÍSICAS PROPOSTAS EM ACADEMIAS EM FUNÇÃO DA AVALIAÇÃO FÍSICA**. 2004. Disponível

em: <http://cronos.univap.br/cd/INIC_2004/trabalhos/epg/pdf/EPG4-10R.pdf>. Acesso em: 02 set. 2017

VII. ANEXOS

- *main.c*

```
#include "msp430g2553.h"
#include <Display.h>
#include <Exercicios.h>
void tela_padrao();
int medir_dist();
void reset_ex();
void exer();
void fim();
void config_display();
void config_sensor();
void config_botoes();
#define BTN  BIT1   //Botão Nivel
#define Reset  BIT0 //Botão Reset
#define Botoes (BTN+Reset)
#define ECHO  BIT4  //Echo sensor ultras.
#define TRIG  BIT3  //Trig sensor ultras.
int miliseconds;
long sensor;
unsigned int nivel, ex, sr, rep;
pvoid main() {
   WDTCTL = WDTPW + WDTHOLD;   // Desabilitar WDT
   BCSCTL1 = CALBC1_1MHZ;   // Clock 1MHz
   DCOCTL = CALDCO_1MHZ;
   config_display();
   config_sensor();
   config_botoes();
   initLCD();
   clearLCD();
   tela_padrao();
   reset_ex();
   while(1){
   exer();
   setAddr(72, 0);
   writeIntToLCD(nivel + 1);
   setAddr(0, 3);
   writeStringToLCD(&nome[ex][0]);
   setAddr(72, 4);
   writeIntToLCD(sr + 1);
   setAddr(72, 5);
   writeIntToLCD(rep);
   }
}
#pragma vector=PORT1_VECTOR
__interrupt void Port_1(void){
   if(P1IFG&ECHO){                  // Interrupção gerada pelo sensor ultrassônico
   if(!(P1IES&ECHO)){
         TA0CTL |= TACLR;
         miliseconds = 0;
         P1IES |= ECHO;
   }
   else sensor = (long)miliseconds*1000 + (long)TAR;
   P1IFG &= ~ECHO;
   }
}
#pragma vector=PORT2_VECTOR
__interrupt void Port_2(void){
   if(P2IFG&Reset){                 // Interrupção gerada pelo botão reset
   while((P2IN&Reset)==0);
   WDTCTL = 0;
   }
   if(P2IFG&BTN){                   // Interrupção gerada pelo botão
   __delay_cycles(100000);
   while((P2IN&BTN)==0);
   nivel++;
   if (nivel==6) nivel = 0;
   reset_ex();
```

```
    P2IFG &= ~BTN;
    }
}
#pragma vector=TIMER0_A0_VECTOR
__interrupt void Timer_A0 (void){    //Contagem de tempo do sensor ultrassônico
    miliseconds++;
    TA0CTL &= ~TAIFG;
}
void config_display(){
    P1OUT |= LCD5110_SCE_PIN + LCD5110_DC_PIN;
    P1DIR |= LCD5110_SCE_PIN + LCD5110_DC_PIN;
    P1SEL |= LCD5110_SCLK_PIN + LCD5110_DN_PIN;
    P1SEL2 |= LCD5110_SCLK_PIN + LCD5110_DN_PIN;
    UCB0CTL0 |= UCCKPH + UCMSB + UCMST + UCSYNC;          // 3-pin, 8-bit SPI master
    UCB0CTL1 |= UCSSEL_2;                                 // SMCLK
    UCB0BR0 |= 0x01;                                      // 1:1
    UCB0BR1 = 0;
    UCB0CTL1 &= ~UCSWRST;                                 // clear SW
}
void config_sensor(){
    CCTL0 = CCIE;
    TA0CCR0 = 1000;
    TA0CTL = TASSEL_2 + MC_1;
    P1DIR |= TRIG;
    P1DIR &= ~ECHO;
    P1IE |= ECHO;
    P1IES &= ~ECHO;
}
void config_botoes(){
    P2DIR &= ~Botoes;
    P2REN |= Botoes;
    P2OUT |= Botoes;
    P2IES |= Botoes;
    P2IE |= Botoes;
    _BIS_SR(GIE);
}
int medir_dist(){
    int distancia;
    P1OUT |= TRIG;                  // gera um pulso
    __delay_cycles(10);             // espera 10 us
    P1OUT &= ~TRIG;                 // para o pulso
    P1IES &= ~ECHO;
    __delay_cycles(30000);          // tempo limite do sensor
    distancia = sensor/58;          // convertendo tempo em cm
    return distancia;
}
void tela_padrao(){
    setAddr(0, 0);
    writeStringToLCD("Nivel:");
    setAddr(0, 2);
    writeStringToLCD("Exercicio:");
    setAddr(0, 4);
    writeStringToLCD("Series:");
    setAddr(0, 5);
    writeStringToLCD("Repeticoes:");
}
void exer(){
    if(sr==0 && rep==0){
    __delay_cycles(5000000);
    ex++;
    sr = series[ex][nivel];
    rep = repeticoes[ex][nivel];
    }
    if(rep==0){
    __delay_cycles(5000000);
    sr--;
    rep = repeticoes[ex][nivel];
    }
    if(medir_dist()<=30){
    rep--;
```

```
    while(medir_dist()<=40) __delay_cycles(500000);
    }
    if(ex>2){
    fim();
    }
}
void reset_ex(){
    ex = 0;
    sr = series[ex][nivel];
    rep = repeticoes[ex][nivel];
}
void fim(){
    __delay_cycles(500000);
    clearLCD();
    reset_ex();
    setAddr(0, 2);
    writeStringToLCD("  Exercicios   ");
    setAddr(0, 3);
    writeStringToLCD("  Concluidos   ");
    _BIS_SR(LPM4_bits);
}
```

- *Display.c*

```
#include "msp430g2553.h"
#include "Display.h"
void writeStringToLCD(const char *string) {
    while(*string) {
    writeCharToLCD(*string++);
    }
}
void writeCharToLCD(char c) {
    unsigned char i;
    if(c != '\0'){
    for(i = 0; i < 5; i++) {
    writeToLCD(LCD5110_DATA, font[c - 0x20][i]);
    }
    }
    writeToLCD(LCD5110_DATA, 0);
}
void writeIntToLCD(int n){
    int casa, dig;
    if(n==0 || n<0){
    writeCharToLCD(' ');
    writeCharToLCD(0x30);
    return;
    }
    if(n<10){
    writeCharToLCD(' ');
    }
    for(casa = 10000; casa>n; casa /= 10);
    while(casa>0)
    {
    dig = (n/casa);
    writeCharToLCD(dig + 0x30);
    n -= dig*casa;
    casa /= 10;
    }
}
void clearLCD() {
    setAddr(0, 0);
    int c = 0;
    while(c < PCD8544_MAXBYTES) {
    writeToLCD(LCD5110_DATA, 0);
    c++;
    }
    setAddr(0, 0);
}
void writeToLCD(unsigned char dataCommand, unsigned char data) {
    LCD5110_SELECT;
    if(dataCommand) {
    LCD5110_SET_DATA;
    } else {
    LCD5110_SET_COMMAND;
```

```
    }
    UCB0TXBUF = data;
    while(!(IFG2 & UCB0TXIFG));
    LCD5110_DESELECT;
}
void setAddr(unsigned char xAddr, unsigned char yAddr) {
    writeToLCD(LCD5110_COMMAND,
PCD8544_SETXADDR | xAddr);
    writeToLCD(LCD5110_COMMAND,
PCD8544_SETYADDR | yAddr);
}
void initLCD() {
    writeToLCD(LCD5110_COMMAND,
PCD8544_FUNCTIONSET                                    |
PCD8544_EXTENDEDINSTRUCTION);
    writeToLCD(LCD5110_COMMAND, PCD8544_SETVOP
| 0x3F);
    writeToLCD(LCD5110_COMMAND,
PCD8544_SETTEMP | 0x02);
    writeToLCD(LCD5110_COMMAND,
PCD8544_SETBIAS | 0x03);
    writeToLCD(LCD5110_COMMAND,
PCD8544_FUNCTIONSET);
    writeToLCD(LCD5110_COMMAND,
PCD8544_DISPLAYCONTROL                                 |
PCD8544_DISPLAYNORMAL);
}
```