Osciloscópio Embarcado com Raspberry Pi

Ferramenta para Análise Gráfica de Sinais e Gravação de Resultados

Daniel Carvalho de Sousa
Engenharia Eletrônica
Universidade de Brasília - Faculdade do Gama
Brasília/DF. Brasil
danielcsousadf@gmail.com

Gilvan Júnior Pereira Camargo
Engenharia Eletrônica
Universidade de Brasília - Faculdade do Gama
Brasília/DF, Brasil
gilvan.jpc@gmail.com

***Resumo*— A proposta contida neste artigo tem por escopo apresentar uma ferramenta alternativa de aquisição de dados com interface gráfica de controle e flexível quanto à obtenção de dados, a partir da Raspberry Pi. Equipamentos como osciloscópio e geradores de função, são utilizados diariamente no âmbito acadêmico e profissional de engenheiros eletrônicos, público alvo desse projeto. A problemática parte tanto das barreiras verificadas na etapa de obtenção de dados em experimentos funcionais, quanto na análise posterior dos resultados, devido à perdas de informações, seletividade ou devido à leituras imprecisas, podendo ocasionar erros e conclusões equivocadas ao projeto em questão.**

Palavras-chave— Osciloscópio; Aquisição de dados; Raspberry-Pi;

I. INTRODUÇÃO

O osciloscópio é uma importante ferramenta no campo das medições elétricas muito utilizada por estudantes de Engenharia Elétrica e Física para validação e cálculos primários. Porém, observa-se que a necessidade de retirar os dados de forma rápida, seja para uma análise mais precisa, seja para relatórios pós testes, impõe dificuldades e complexidades a esse processo.

Uma função que melhora a experiência de usá-lo em laboratório é obter uma imagem da sua tela, através de um *screen-shot* e enviar ao estudante em nuvem ou dispositivo móvel, por exemplo, sem que seja necessário uma fotografia do aparelho em sala de aula.

A ferramenta proposta caracteriza-se como um sistema embarcado composto por um sistema de aquisisção de dados, condicionamento de sinais, controle de interface gráfica e processamento de dados. A integração entre hardware e software para executar uma função dedicada é notória nesse sistema, de tal forma que esse sistema caracteriza-se como um típico sistema embarcado [7]. Dentre as ferramentas uttilizadas, destacam-se a raspberryPi3, com Linux embarcado, controle online de fluxo de dados, além da circuitaria envolvida com o conversor A/D, tais como condicionadores de sinais, limitadores de amplitude e bufferização.

II. OBJETIVOS

Este projeto tem como objetivo desenvolver um analisador gráfico de sinais elétricos, através de sua análise transiente, com funções de ajuste de escala vertical (tensão) e de posição, além da função de salvar imagens por meio de um print-screen da tela.

III. JUSTIFICATIVA

Muitas vezes não se tem um osciloscópio à sua disposição, nem ao menos para fazer análises modestas no regime temporal, muitas vezes devido à dificuldade de acesso ou pelo preço elevado desses aparelhos. Além disso, geralmente não se tem a opção de salvar a tela automaticamente, fazendo-se necessário tirar fotos, denegrindo a qualidade dos dados e muitas vezes perdendo a resolução dos dados de medição.

IV. REQUISITOS

Os requisitos necessários para a construção desse sistema, levam em conta, não só a funcionalidade mínima exigida mas também a viabilidade ou aceitabilidade do produto pelo público alvo. Dentre os requisitos, destaca-se uma interface gráfica intuitiva, armazenamento de dados e amostras, comunicação com dispositivos eletrônicos cotidianos, funcionalidades padrão necessárias para engenheiros eletrônicos (escala, cursores, trigger) e conexão compatível com pontas de prova cabos de laboratório. Os requisitos do sistema encontra-se na tabela 1.

TABELA I. REQUISITOS NO OSCILOSCÓPIO

Requisitos no Osciloscópio		
Conversão A/D rápida	Alta excursão de tensão de entrada	Botões de menu
Taxa de atualização de pelo menos 200ms	Impedância alta de entrada	Entradas e Saídas com Proteção

A figura 1 ilustra a configuração inicial dos componentes necessários para elaboração do projeto.

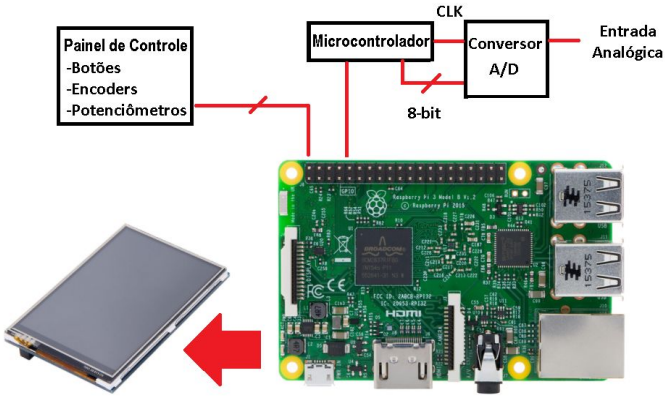

Fig. 1 - Diagrama esquemático do hardware.

V. METODOLOGIA

Com base em pesquisas de mercado em modelos convencionais de osciloscópio digital, observou-se que a taxa de amostragem varia de acordo com a aplicação desejada, podendo atingir valores de 5M até centenas de Giga hertz de taxa de amostragem [5]. Como o sistema operacional em conjunto com o microprocessador presente na Raspberry PI não permite a leitura de sinais a taxas de 5GHz [3], observou-se a necessidade da inserção de registrador com a função de buffer ao sistema capaz armazenar as amostras lidas pelo conversor AD, de modo a enviar para a Raspberry PI apenas pacotes de dados a cada requisição. Dessa forma, seria possível processar amostras e mostrá-las ao display de modo online.

A escolha do conversor AD a ser utilizado levou em conta tanto a máxima frequência utilizada quanto a resolução mínima de tensão para uma aplicação cotidiana de sinais de baixa e média frequência. Dessa forma foi definida uma taxa de amostragem viável de 5MSps com resolução de 10 bits, contemplando sinais com resposta em frequência de até 2.5MHz [4], porém as dificuldades encontradas na soldagem dos *packages* comerciais, tornaram inviáveis a utilização dos registradores FIFO e dos conversores de alta frequência para um protótipo funcional.

O procedimento de inscrição da tela LCD, em conjunto com processamento de dados, posteriores ao processo de aquisição será feito em paralelo com o desenvolvimento do hardware, de forma a identificar os eventuais problemas e solucioná-los para ambas interfaces do projeto. Como o interfaceamento da raspberry Pi com o buffer, optou-se pelo uso de microcontroladores, visto que podem tanto processar dados enviados da raspberry quanto enviar dados formatados, além de servir como buffer na aquisição de dados do conversor AD em questão.

Além disso, o preço estimado para importação tanto dos conversores quanto dos registradores FIFO dobraria o custo estimado pelo projeto. A partir de então, limitou-se a aplicação para apenas sinais de baixa frequência, tais como sinais de áudio, PWM de baixa/média frequência e análise de transitórios em circuitos RLC.

VI. PROCEDIMENTOS EXPERIMENTAIS

A. Condicionamento de Sinais

O condicionamento do sinal de entrada no hardware é feito para adequá-lo à faixa de 0 a 5V, que são as tensões de referência do conversor AD. O projeto do circuito levou em conta a necessidade de manter tanto componentes AC quanto DC, visto que o desacoplamento por meio de um capacitor ocasionaria na perda do sinal DC.

A figura 2 indica o circuito projetado, que é polarizado por meio de um divisor resistivo associado a capacitores para criar um "terra virtual", o qual permitirá excursões positivas e negativas de sinais em sua entrada. Os limites de operação escolhidos foram -5V a 5V.

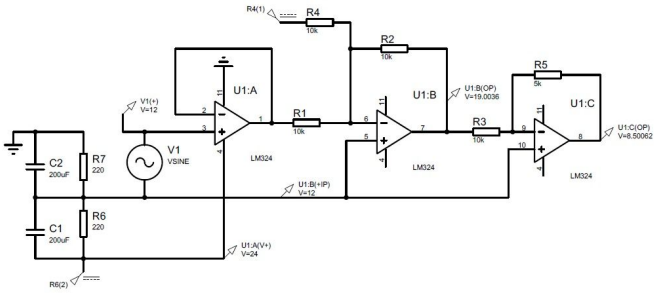

Fig. 2 - Circuito para condicionamento de sinais.

No primeiro estágio o sinal passa por um buffer, o qual garante uma alta impedância de entrada do osciloscópio. No segundo estágio, o sinal é somado com 5V, garantindo tensões positivas em relação ao terra virtual criado. Após esse estágio, observa-se que a tensão poderá atingir até 10V, fazendo-se necessário a atenuação desse sinal antes de conectar ao conversor AD, que opera na faixa de 5V. A atenuação é feita por meio do terceiro amplificador operacional projetado para

ganho -2. Tanto o somador quanto o atenuador estão em suas configurações inversoras, logo, o sinal de saída estará em fase com o sinal de entrada, como observado na Fig.3.

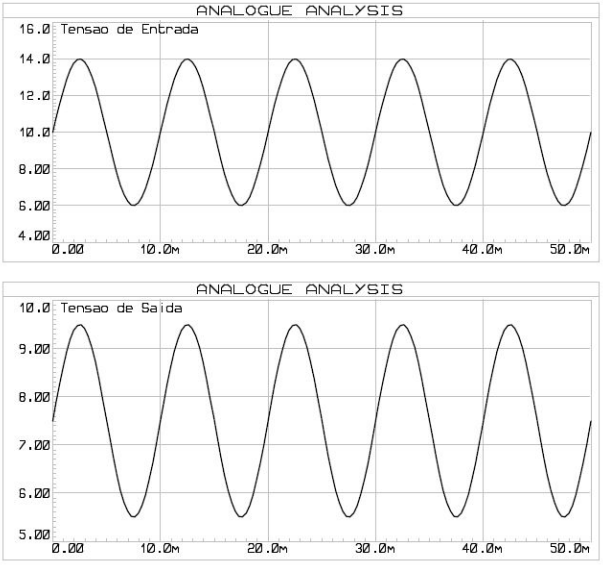

Fig. 3 - Resultado do condicionador de sinais a partir de uma senoide de 8Vpp na entrada, resultando em uma senoide de 2Vpp na saída.

B. Aquisição

O conversor analógico-digital escolhido foi o TCL0820CA da Texas Instruments que possui taxa de amostragem razoável de 400kHz e package CDIP para soldagem viável com as ferramentas disponíveis.

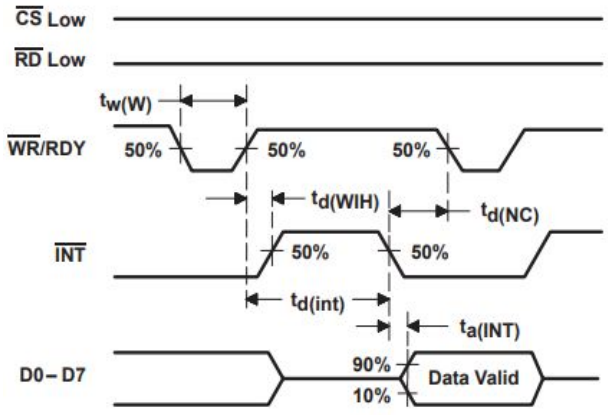

Fig. 4 - Modo de operação Stand-Alone. [6]

Com o pino 7, MODE, ativo, é escolhido o modo Stand-Alone de operação cujo os pinos 13 e 9, [CS] e [INT] são ativados na tensão de referência, terra. Portanto o funcionamento circuito fica simplificado de acordo com o datasheet [6], bastando apenas ser ser realizado um pulso invertido no pino 6, [WR]/RDY, para obtenção do sinal quantizado em 8 bits após o pino 9, [INT] ir ao nível lógico baixo, conforme ilustra a figura 4.

A implementação da aquisição por meio do MSP430 é feita com auxílio do software Code Composer Studio com bibliotecas e headers do "MSPG2553" que possui oito bits de leitura direta no registrador P1IN que pode ser armazenado no "buffer" em 32 ciclos de clock, adequados para frequência de amostragem do conversor ADC.

A aquisição é seguida de transferência de para o raspberry pela comunicação I2C, com as devidas configurações.

C. Implementação Gráfica

A implementação gráfica utiliza-se do conjunto de bibliotecas para desenvolvimento *OpenGL* na linguagem C, escolhida por ser versátil, por atender aos requisitos de criação de ambientes intuitivos, e por ter uma vasta referência bibliográfica.

Como teste de validação da biblioteca *OpenGL*, foi realizado um programa com auxílio das bibliotecas *math.h* e *unistd.h* para escrever os dados de uma senoide em um gráfico tabular. Nesse programa, foram configuradas escalas de referência em forma de grade, linhas pontilhadas, dimensões e espessuras das linhas, cor de fundo, tamanho de janela, dentre outros. A senóide plotada por esse programa é armazenada em um buffer (simulando as aquisições de dados) e a cada par de amostras é mostrado na tela – visto que são necessários pelo menos dois pontos para plotar uma reta. As amostras são atualizadas na tela a partir de uma taxa de aproximadamente 20ms (definida a partir da função *usleep*). O resultado é mostrado pela figura 5.

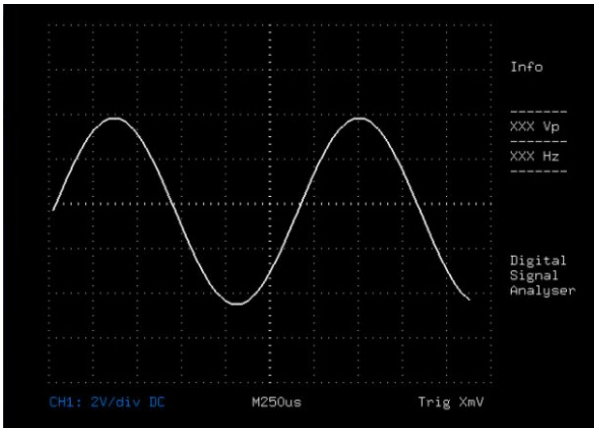

Fig.5 - Teste gráfico com a biblioteca OpenGL.

D. Variação da escala Y

A medida de Volts por divisão(Volts/div) é variada com um encoder rotativo para o usuário ajustar a amplitude da forma de onda no display LCD, o encoder funciona com dois pinos de informação que dizem se a rotação foi para o lado direito, para aumentar a escala de Volts/div ou no sentido

esquerdo, para atenuar a escala, a leitura é feita da seguinte forma:

- Se o pino 1 for levado para nível lógico alto, observa-se o nível lógico no pino 2, se baixo:rotação para esquerda, se alto: rotação para direita. Então espera-se o pino 1 descer e subir novamente para fazer a análise de rotação novamente.

O função no anexo deste documento para a leitura do encoder é realizada em uma thread que possibilita o programa global de realizar mais de uma tarefa ao mesmo tempo [8] e também polling nos pinos 1 e 2 para monitorá-los.

VII. CUSTOS E PRAZOS

A fim de verificar a viabilidade do projeto, foi realizado o levantamento de custo, etapa na qual pode validar ou não a aceitação do público alvo. Nessa etapa, os componentes escolhidos e funções adicionais foram determinadas de tal forma que o preço final estipulado esteja pelo menos 20% menor que os produtos convencionais, de forma que fique acessível tanto para alunos de graduação quanto para instituições de ensino. A tabela 2 contém os preços e custos esperados do projeto de uma unidade.

TABELA II. CUSTOS UNITÁRIOS ESTIMADOS

Produto	**preço**	**Quantidade**
Conversor AD (5MSPS TI)	35,00	2
Display LCD (TFT 7")	80,00	1
Conversor de dados (HDMI- VGA)	25,00	1
Raspberry PI	100,00	1
Microcontrolador (MSP430)	20,00	1
Estrutura Mecânica (MDF)	20,00	1
Conectores e Botões (Variados)	1,00	20
Circuitos adicionais	20,00	1
Total	345,00	-

REFERÊNCIAS

[1] J.P.S. Catalão, "Osciloscópio" http://webx.ubi.pt/~catalao/Osciloscopio.pdf acesso em 04 de abril de 2018.

[2] AGILENT TECHNOLOGIES., Arquiteturas de memória de osciloscópio – Por que toda memória de aquisição não é criada igualmente. 2012.

[3] Matthew Tivnan, Rajan Gurjar, David E. Wolf, Karthik Vishwanath, High Frequency Sampling of TTL Pulses on a Raspberry Pi for Diffuse Correlation Spectroscopy Applications.

[4] Alan V. Oppenheim, Alan S. Willsky, Sinais e Sistemas 2° edição.

[5] M. Takamiya, M. Mizuno and K. Nakamura, "An on-chip 100 GHz-sampling rate 8-channel sampling oscilloscope with embedded sampling clock generator," 2002 IEEE International Solid-State Circuits Conference. Digest of Technical Papers (Cat. No.02CH37315), San Francisco, CA, USA, 2002, pp. 182-458 vol.1.

[6] Texas Instruments. Datasheet: TLC0820, ADC, SEPTEMBER 1986 – REVISED JUNE 1994.

[7] Li, Q. & Yao, C., Real-Time Concepts for Embedded Systems, Editora: CMP Books, 2003.

[8] Mitchell, M., Oldham, J. & Samuel, A., Advanced Linux Programming, Editora: Newriders, 2001.

ANEXOS

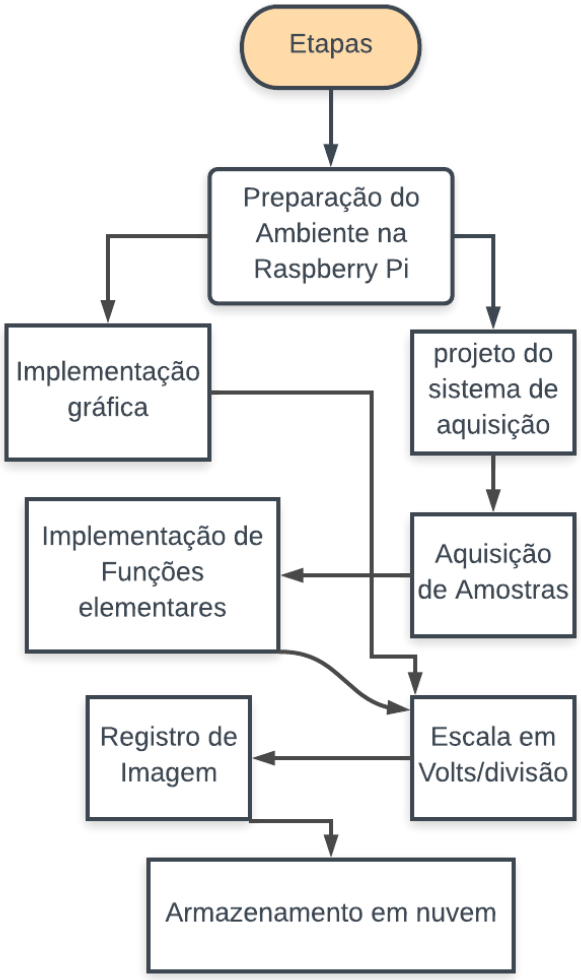

Fig. 6 - Quadro de planejamento.

```
1   //Código para implementação gráfica combinando Amostras, Comunicação I2C e variação de escala vertical com polling nas variáveis de um Encoder.
2   #include <stdio.h>      //  funcoes como printf
3   #include <stdlib.h>     //
4   #include <unistd.h>     //  biblioteca para sleep
5   #include <GL/glut.h>    //  biblioteca opengl
6   #include <GL/gl.h>      //  biblioteca opengl
7   #include <math.h>       //  para simular uma senoide
8   #include <sys/poll.h>
9   #include <fcntl.h>
10  #include <pthread.h>
11  #include <signal.h>
12  #include <sys/ioctl.h>
13  #include <linux/i2c-dev.h>
14  #include <errno.h>
15
16  #define UCB0TXBUF 0x006f
17  #define N 25
```

```
18      #define COMP_MAX 654
19      #define ALTURA_MAX 360
20      #define MARG_ESQUERDA 50
21      #define MARG_DIREITA 534
22      #define MARG_CIMA 338
23      #define MARG_BAIXO 42
24      #define pi 3.141592f
25      #define T0 MARG_ESQUERDA
26      #define T1 MARG_ESQUERDA+20.08
27
28      struct pollfd pfd0,pfd1;
29      char buffer0,buffer1;
30
31      GLfloat buffer_s[500] = {0};  // Armazena os valores anteriores da tela
32      GLfloat Senoid[100] = {0};
33      GLfloat amp = 1.0f;
34
35      unsigned char user_input=1, msp430_ret[N], slave_addr=0x0F;
36      int i2c_fd;
37      GLfloat buffer_s[500] = {0};  // Armazena os valores anteriores da tela
38      GLfloat Time[500] = {0};
39      GLfloat Senoid[4*N] = {0};
40      /*GLfloat Senoid[25] = {
41         MARG_ESQUERDA,200, // coordenada iniciais
42         (MARG_ESQUERDA+3.8),200
43      };*/
44      GLfloat amp = 1.0f;
45
46      GLfloat Background[] = {
47
48         MARG_ESQUERDA,49,MARG_DIREITA,49,   50,MARG_BAIXO,50,MARG_CIMA,

49         MARG_ESQUERDA,97,MARG_DIREITA,97,   98,MARG_BAIXO,98,MARG_CIMA,

50         MARG_ESQUERDA,145,MARG_DIREITA,145, 146,MARG_BAIXO,146,MARG_CIMA,

51         MARG_ESQUERDA,193,MARG_DIREITA,193, 194,MARG_BAIXO,194,MARG_CIMA,

52         MARG_ESQUERDA,194,MARG_DIREITA,194, 242,MARG_BAIXO,242,MARG_CIMA,

53         MARG_ESQUERDA,241,MARG_DIREITA,241, 290,MARG_BAIXO,290,MARG_CIMA,

54         MARG_ESQUERDA,289,MARG_DIREITA,289, 291,MARG_BAIXO,291,MARG_CIMA,

55         MARG_ESQUERDA,337,MARG_DIREITA,337, 338,MARG_BAIXO,338,MARG_CIMA,

56         MARG_ESQUERDA,192,MARG_DIREITA,192, 386,MARG_BAIXO,386,MARG_CIMA,

57                                 434,MARG_BAIXO,434,MARG_CIMA,
58                                 482,MARG_BAIXO,482,MARG_CIMA,
59                                 530,MARG_BAIXO,530,MARG_CIMA,
60                                 289,MARG_BAIXO,289,MARG_CIMA
61
62      };
63
```

```
64      void ctrl_c(int sig)
65      {
66              close(i2c_fd);
67              exit(-1);
68      }
69
70      void back_setup() {
71       // ------ Redimentionando a tela para unidades de pixels ---- //
72         glViewport(0.0f, 0.0f, COMP_MAX, ALTURA_MAX);
73         glMatrixMode(GL_PROJECTION);
74         glLoadIdentity();   // update depois de mecher com a matriz
75         glOrtho(0, COMP_MAX, 0, ALTURA_MAX, 0, 1);  // setando os limites da tela
76         glMatrixMode(GL_MODELVIEW); // default values
77         glPushMatrix();
78         glLoadIdentity();   //Sempre atualizar depois de setar funÃ§Ãµes de Matriz
79      // -------------------------------------------------------//
80      }
81
82      void GPIO_setup(){
83
84         system("echo 4       > /sys/class/gpio/export");
85         system("echo 17      > /sys/class/gpio/export");
86         system("echo falling> /sys/class/gpio/gpio4/edge");
87         system("echo in      > /sys/class/gpio/gpio4/direction");
88         system("echo in      > /sys/class/gpio/gpio17/direction");
89      }
90
91      void* encoder_Read(void* dummy_ptr){
92
93
94         pfd0.fd = open("/sys/class/gpio/gpio4/value", O_RDONLY);
95         pfd1.fd = open("/sys/class/gpio/gpio17/value", O_RDONLY);
96
97         if((pfd0.fd < 0) || (pfd1.fd < 0))
98         {
99            puts("Erro abrindo arquivo");
100                puts("Execute este programa como root.");
101
102        }
103
104        read(pfd0.fd, &buffer0, 1);
105        read(pfd1.fd, &buffer1, 1);
106
107        pfd0.events = POLLPRI | POLLERR;
108        pfd1.events = POLLPRI | POLLERR;
109        pfd0.revents = 0;
110        pfd1.revents = 0;
111
112        for(;;){
113                        lseek(pfd0.fd, 0, SEEK_SET);
114                read(pfd0.fd, &buffer0, 1);
115                poll(&pfd0, 1, -1);
116                lseek(pfd1.fd, 0, SEEK_SET);
117                read(pfd1.fd, &buffer1, 1);
118
```

```
119                 if(buffer1 == '0'){
120                         amp = amp*1.15;
121                         puts("");
122                 }else{
123                         amp = amp*0.85;
124                         puts("");
125                 }
126                 printf("%f",amp);
127         }
128         close(pfd0.fd);
129         close(pfd1.fd);
130
131     }
132     int i = 0;
133
134     void main_loop(){
135                 if((user_input<0) || (user_input>5))
136                         puts("Valor invalido");
137                 else if(user_input>0)
138                 {
139                         if (write(i2c_fd, &user_input, 1) < 0)
140                         {
141                                 fprintf (stderr, "Unable to write serial device: %s\n", strerror (errno)) ;
142                         return 1;
143                         }
144
145                         if (read(i2c_fd, msp430_ret,N*sizeof(char)) < 0)
146                         {
147                                 fprintf (stderr, "Unable to read serial device: %s\n", strerror (errno)) ;
148                                 return 1;
149                         }
150                         for(i=0 ; i<N ; i++)
151                                 printf("MSP430_return %d = %d\n",i,msp430_ret[i]);
152                 }
153                 usleep(80000);
154                 puts("");
155
156         glClearColor(0,0,0,0);  // Cor preta (nackgroung)
157         glClear(GL_COLOR_BUFFER_BIT);   // resetando a tela(zera todas as linhas)
158         glLineWidth(1); // Largura das linhas das linhas de escala
159         glColor3f(1.0f,1.0f,1.0f);  // cor cinza atribuÃda ao background
160
161         glEnable(GL_LINE_STIPPLE);  // habilita a pixelizaÃ§Ã£o de linha
162         glLineStipple(1,0x8080); // padrÃ£o de pixels: 1000000010000000, repete atÃ© o tÃ©rmino da linha

163         glEnableClientState(GL_VERTEX_ARRAY);   // habilita a escrita de um  array
164         glVertexPointer(2,GL_FLOAT,0,Background); // DimensÃ£o (x,y) | tipo | gap(0)| *coordenadas

165         glDrawArrays(GL_LINES,0,44); // NÃºmero de pares coordenados (x,y) a ser escrito
166                 glDisableClientState(GL_VERTEX_ARRAY);  // Desabilita a escrita em array
167         glDisable(GL_LINE_STIPPLE); //  Desabilita a funcao "stipple"
168         glLineWidth(2); // Espessura da linha (3 pixels)
169
170         for(i = 0; i< N-1; i++){
171
```

```
172             Senoid[i<<2] = T0 + i*20.08;
173                  Senoid[(i<<2)+1]  =  (int)msp430_ret[i]*1.15625+MARG_BAIXO;//(100*amp*sin(2*pi*i/12)+250)-50;        //
Amplitude inicial
174             Senoid[(i<<2)+2] = T1 + i*20.08;
175                 Senoid[(i<<2)+3] = (int)msp430_ret[i+1]*1.15625+MARG_BAIXO;//(100*amp*sin(2*pi*(i+1)/12)+250)-50; //
Amplitude "amostrada" a "x" ms depois
176         }
177
178         glEnableClientState(GL_VERTEX_ARRAY);   // Escrita dos dados na tela
179                 glVertexPointer(2,GL_FLOAT,0,Senoid);   // constrÃ³i vÃ©rtices a partir do vetor Senoid

180                 glDrawArrays(GL_LINES,0,50); // Desenha na tela as 2 amostras mais atuais.
181         glDisableClientState(GL_VERTEX_ARRAY);
182
183         glColor3f(1.0f, 1.0f, 1.0f);    //cor branca
184                 glRasterPos3f(270, 20,0.0);     //PosiÃ§Ã£o do texto
185                 glutBitmapString(GLUT_BITMAP_HELVETICA_18, "M250us");    //Escreve a escala na tela

186         glRasterPos3f(550, 320,0.0);
187         glutBitmapString(GLUT_BITMAP_HELVETICA_18, "Info\n\n\n-------\nXXX Vp\n-------\nXXX Hz\n-------\n");

188         glutBitmapString(GLUT_BITMAP_HELVETICA_18,"\nDigital\nSignal\nAnalyser");

189         glRasterPos3f(450, 20,0.0);     //PosiÃ§Ã£o do texto
190                 glutBitmapString(GLUT_BITMAP_HELVETICA_18, "Trig XmV");    //Escreve a escala na tela

191         glColor3f(0.0f, 0.5f, 1.0f);    // cor azul
192         glRasterPos3f(50, 20,0.0);
193         glutBitmapString(GLUT_BITMAP_HELVETICA_18, "CH1: 2V/div DC");
194
195                 glFlush();
196         glutPostRedisplay();    // Retorna ao inicio dessa funÃ§Ã£o (Looping infinito)
197     }
198
199     void main(int argc, char**argv) {
200
201                 pthread_t thread_GPIO;
202         GPIO_setup();
203                 signal(SIGINT, ctrl_c);
204                 i2c_fd = open("/dev/i2c-1", O_RDWR);
205                 ioctl(i2c_fd, I2C_SLAVE, slave_addr);
206
207         glutInit(&argc, argv);  // Ã‰ possÃvel alterar as configuraÃ§Ãµes de execuÃ§Ã£o pelos argv e argc

208         glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);    // BufferizaÃ§Ã£o simples, RGB

209         glutInitWindowPosition(0,0);    // PosiÃ§Ã£o inicial da janela em relaÃ§Ã£o ao lado esquerdo/cima

210         glutInitWindowSize(COMP_MAX,ALTURA_MAX);    // Delimitando limites (escalas em pixels)

211         glutCreateWindow("Osciloscope");    // Nome da janela~]]
212         back_setup();   // ConfiguraÃ§Ãµes iniciais da janela
213                 pthread_create(&thread_GPIO, NULL, &encoder_Read, NULL);
214         glutDisplayFunc(main_loop); // funÃ§Ã£o que comandarÃ¡ a atualizaÃ§Ã£o da tela
```

```
215         glutMainLoop(); //Garante que a janela ficarÃ¡ aberta durante a execuÃ§Ã£o do programa.

216         close(i2c_fd);  // Por fim das contas, fechar o i2c
217     }
```