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.

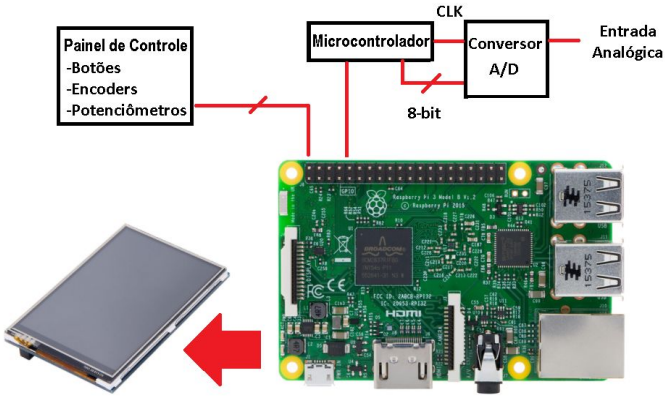

Fig. 1 - Diagrama esquemático do hardware.

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.

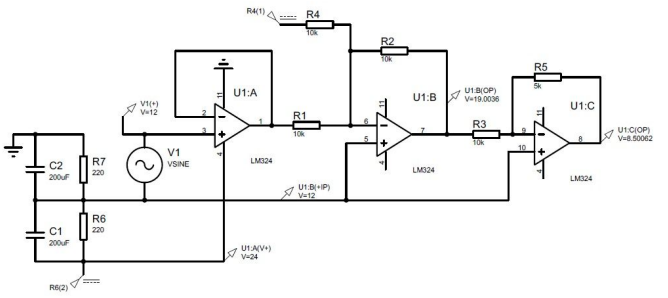

Fig. 2 - Circuito para condicionamento de sinais.

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.

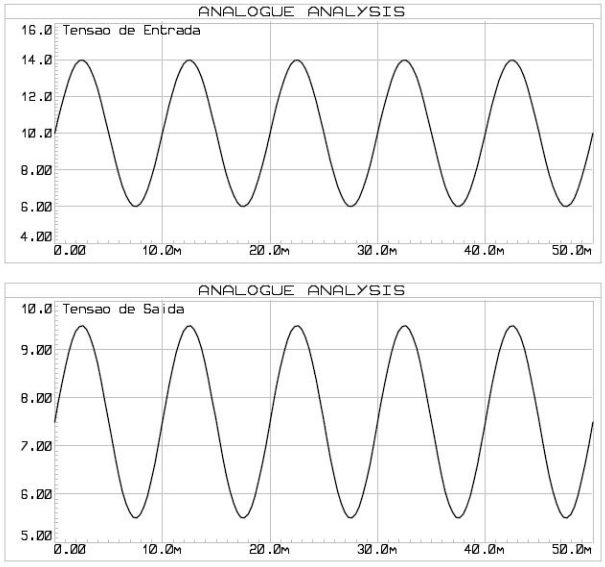

Fig. 3 - Resultado do condicionador de sinais a partir de uma senoide de 8Vpp na entrada, resultando em uma senoide de 2Vpp na saída.

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.

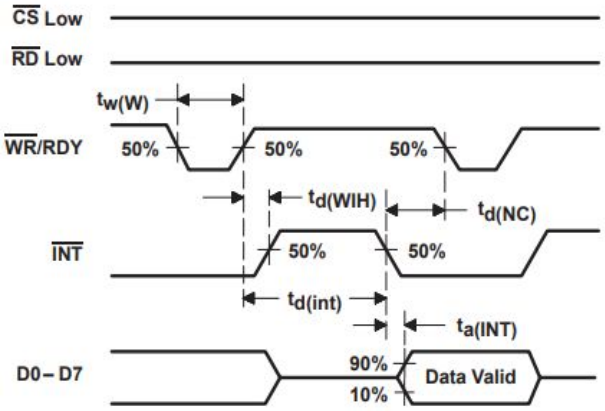

Fig. 4 - Modo de operação Stand-Alone. [6]

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.

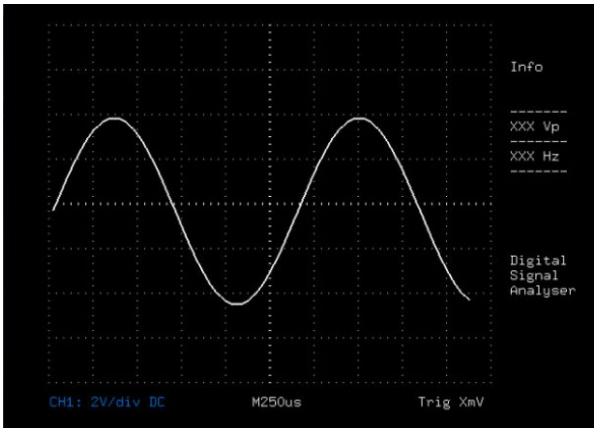

Fig.5 - Teste gráfico com a biblioteca OpenGL.

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.

Considerou-se os 8 bits do conversor A/D que totaliza 256 níveis distintos de quantização com as seis divisões para a escala de amplitudes no display, utilizou-se a o nível 128 para ser a referência, 0V. Então identificaram-se quais seriam os níveis para 1,500m,200m,50m e 10m (V/div), e mapearan-se estes valores nos limites inferior e superior da tela.

E. Variação da Escala Temporal

Variou-se a frequência de amostragem para cada uma das escalas de visualização no display, tendo em vista que aumentando a frequência de amostragem, teriam mais períodos num mesmo intervalo de amostras, e vice-versa, diminuir a frequência de amostragem implica menos períodos de um mesmo sinal considerando o tamanho o buffer de amostras constante.

F. Print-screen automático

O print-screen ocorre com pooling em um botão que acionas a thread que executa o script .sh que usa a função *scrot()* para gravar a imagem e o comando *mv* para mover a imagem para um pendrive conectado a raspberry.

Com o auxílio dos comandos *findmnt* e *awk*, localiza-se o nome do pendrive montado na pasta */etc/sda1*.,

O programa geral é iniciado automaticamente com a inserção de seu caminho completo acompanhado por um '*&*' no arquivo */etc/profile*.

A figura 6 é um print-screen salvo no pendrive.

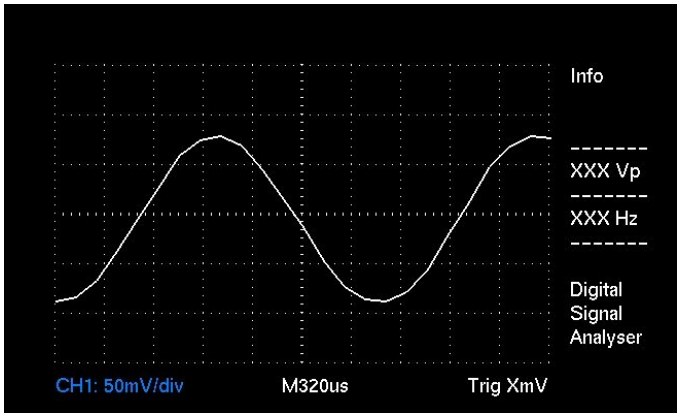

Fig. 6 - Print-screen do display do osciloscópio.

G. Trigger

O trigger serve para informar ao dispositivo onde está a onda no display. E esta ferramenta foi construída varrendo-se a onda de interesse em um laço *for* para plotá-la somente qando se visse o valor do trigger na onda. Escolhe-se o valor de trigger a partir de um potenciômetro, assim como a escala de amplitudes e temporização por divisão.

H. Inicialização automática com @reboot

Com intuito de automatização da inicialização do programa no raspberrypi, inseriu-se a seguinte linha de código no arquivo /etc/profile:

lxterminal --command=/home/Documets/./oscilocpio.out

Este comando inicializa o terminal executando o programa compilado. Este requisito ajuda a desequipar mouse e o teclado do sistema.

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	-

VIII. RESULTADOS

A acomodação do microcontrolador, da raspberry e do display foram foi feita em uma case confeccionada em madeira MDF e acrílico. A figura 7 mostra o conjunto em sua

visão externa e interna de componentes. São destacados detalhes como os encoders reguladores de amplitude e temporização e na parte interna a disposição das placas no protótipo.

Fig. 7a - Visão externa do oscilóscopio.

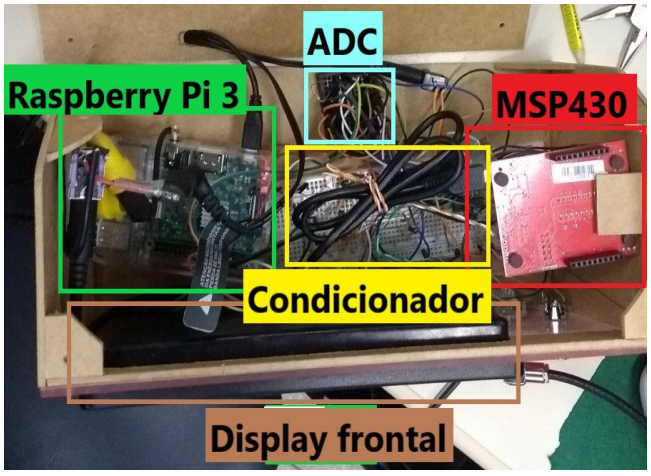

Fig. 7b - Visão interna do osciloscópio.

A frequência de máxima de input com resultados satisfatórios acontecem em algumas dezenas de hertz, que seria suficiente para aplicações em áudio.

Não é apresentado na figura 7, mas na lateral da case estão os slots para pen drive que armazenam os print-screens com data e hora quando solicitados pelo push-button.

A inicialização do sistema se dá com início automático do programa do "osciloscópio", com as ressalva de ser necessária a troca de páginas para trazer a janela com a forma de onda para frente, com "Alt-tab" pelo teclado.

IX. CONCLUSÃO

Este projeto evidencia a complexidade de se criar um osciloscópio, como lidar controlar a frequência de amostragem ou até mesmo a quantidade de volts por divisão no display.

Para uma análise concreta de um sinal é necessária acurácia e precisão no instrumento de medida, bem como a preservação do seu nível de offset a cada instante,para esta finalidade foi implementado um circuito condicionador na entrada, evitando que se utiliza-se apenas um simples capacitor que desprezasse o DC. Esta é uma prática comum e essencial também para garantir a integridade do equipamento.

Como este sistema vem sendo implementado em uma plataforma com diversas ferramentas, ainda há o que ser melhorado em relação a automatização do código, backup em nuvem, mais níveis de amplitude e principalmente suporte a frequências mais altas, bem como análises espectrais.

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

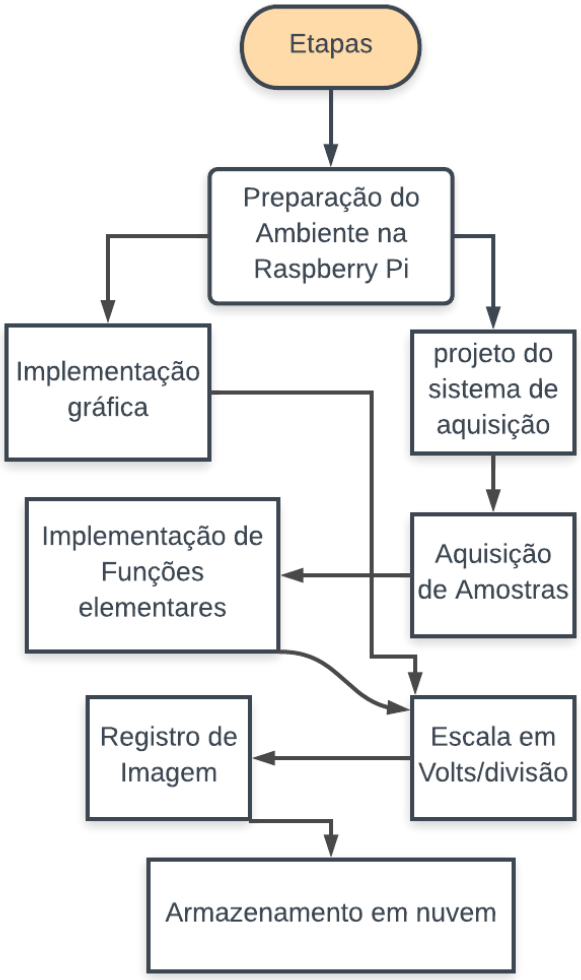

Fig. 7 - Quadro de planejamento.

```
1   // Código para inicialização Gráfica e comunicação com microcontrolador
2   #include <stdio.h> // libs para printf e tals
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 include
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 100
18  #define COMP_MAX 654
```

```
#define ALTURA_MAX 396
#define MARG_ESQUERDA 50
#define MARG_DIREITA 534
#define MARG_CIMA 338
#define MARG_BAIXO 42
#define pi 3.141592f
#define T0 MARG_ESQUERDA
#define T1 MARG_ESQUERDA+20.08
#define Nbounce 500
#define Nlim (Nbounce*3)/4

struct pollfd pfd0,pfd1,pfd2,pfd3;
char buffer0,buffer1,buffer2,buffer3;
char v_pico[10] = {'\0'};
int val_pico = 0;   // Valor da tensÃ£o de pico do sinal
int tempo = 1;
int amp = 5;
int Aux = 0;
int index = 0;
int trigger = 100;
float in_min;
float in_max;
GLfloat last_value = 999;
GLfloat buffer_s[500] = {0};  // Armazena os valores anteriores da tela
GLfloat Senoid[100] = {0};
//GLfloat amp = 1.0f;

unsigned char user_input=1, msp430_ret[N], slave_addr=0x0F;
int i2c_fd;
GLfloat Time[500] = {0};

GLfloat Background[] = {

   MARG_ESQUERDA,49,MARG_DIREITA,49,   50,MARG_BAIXO,50,MARG_CIMA,
   MARG_ESQUERDA,97,MARG_DIREITA,97,   98,MARG_BAIXO,98,MARG_CIMA,
   MARG_ESQUERDA,145,MARG_DIREITA,145, 146,MARG_BAIXO,146,MARG_CIMA,
   MARG_ESQUERDA,193,MARG_DIREITA,193, 194,MARG_BAIXO,194,MARG_CIMA,
   MARG_ESQUERDA,194,MARG_DIREITA,194, 242,MARG_BAIXO,242,MARG_CIMA,
   MARG_ESQUERDA,241,MARG_DIREITA,241, 290,MARG_BAIXO,290,MARG_CIMA,
   MARG_ESQUERDA,289,MARG_DIREITA,289, 291,MARG_BAIXO,291,MARG_CIMA,
   MARG_ESQUERDA,337,MARG_DIREITA,337, 338,MARG_BAIXO,338,MARG_CIMA,
   MARG_ESQUERDA,192,MARG_DIREITA,192, 386,MARG_BAIXO,386,MARG_CIMA,
                              434,MARG_BAIXO,434,MARG_CIMA,
                              482,MARG_BAIXO,482,MARG_CIMA,
                              530,MARG_BAIXO,530,MARG_CIMA,
                              289,MARG_BAIXO,289,MARG_CIMA

};

void ctrl_c(int sig)
{


}

```

```
74      int map(int Accel, float in_min, float in_max, float out_min, float out_max)
75      {
76        return (int)(((float)Accel - in_min)*(out_max - out_min)/(in_max - in_min)+out_min);
77      }
78
79      void back_setup() {
80       // ------ Redimentionando a tela para unidades de pixels ---- //
81          glViewport(0.0f, 0.0f, COMP_MAX, ALTURA_MAX);
82          glMatrixMode(GL_PROJECTION);
83          glLoadIdentity();   // update depois de mecher com a matriz
84          glOrtho(0, COMP_MAX, 0, ALTURA_MAX, 0, 1);  // setando os limites da tela
85          glMatrixMode(GL_MODELVIEW); // default values
86          glPushMatrix();
87          glLoadIdentity();   //Sempre atualizar depois de setar funÃ§Ãµes de Matriz
88      // ------------------------------------------------------------//
89      }
90
91      void GPIO_setup(){
92          system("echo 23       > /sys/class/gpio/export");
93          system("echo 18       > /sys/class/gpio/export");
94          system("echo 4       > /sys/class/gpio/export");
95          system("echo 17      > /sys/class/gpio/export");
96          system("echo falling> /sys/class/gpio/gpio4/edge");
97          system("echo falling> /sys/class/gpio/gpio18/edge");
98          system("echo in      > /sys/class/gpio/gpio4/direction");
99          system("echo in      > /sys/class/gpio/gpio17/direction");
100         system("echo in      > /sys/class/gpio/gpio18/direction");
101         system("echo in      > /sys/class/gpio/gpio23/direction");
102     }
103
104     void* encoder_Read1(void* dummy_ptr){
105
106         pfd0.fd = open("/sys/class/gpio/gpio4/value", O_RDONLY);
107         pfd1.fd = open("/sys/class/gpio/gpio17/value", O_RDONLY);
108
109         if((pfd0.fd < 0) || (pfd1.fd < 0))
110         {
111             puts("Erro abrindo arquivo");
112
113         }
114
115         read(pfd0.fd, &buffer0, 1);
116         read(pfd1.fd, &buffer1, 1);
117
118
119         pfd0.events = POLLPRI | POLLERR;
120         pfd1.events = POLLPRI | POLLERR;
121         pfd0.revents = 0;
122         pfd1.revents = 0;
123
124         for(;;){
125
126
127
128
```

```
129
130
131
132
133
134
135
136
137
138             }
139         }
140         close(pfd0.fd);
141         close(pfd1.fd);
142
143     }
144
145     void* encoder_Read2(void* dummy_ptr){
146
147         pfd2.fd = open("/sys/class/gpio/gpio18/value", O_RDONLY);
148         pfd3.fd = open("/sys/class/gpio/gpio23/value", O_RDONLY);
149
150         if((pfd2.fd < 0)||(pfd3.fd < 0))
151         {
152             puts("Erro abrindo arquivo");
153
154         }
155
156         read(pfd2.fd, &buffer2, 1);
157         read(pfd3.fd, &buffer3, 1);
158
159         pfd2.events = POLLPRI | POLLERR;
160         pfd3.events = POLLPRI | POLLERR;
161         pfd2.revents = 0;
162         pfd3.revents = 0;
163
164         for(;;){
165
166
167
168
169             lseek(pfd3.fd, 0, SEEK_SET);
170
171
172                 if(tempo < 5)
173
174
175                 if(tempo > 1)
176
177
178         }
179         close(pfd2.fd);
180         close(pfd3.fd);
181
182     }
183
```

```
184
185     void poll_btn(struct pollfd *pfd)
186     {
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206     }
207
208     void set_polling()
209     {
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233     }
234     int i = 0;
235
236     void main_loop(){
237
238
```

```
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254         for(i=0 ; i<N ; i++)
255         {
256
257         }
258
259
260         glClearColor(0,0,0,0);  // Cor preta (nackgroung)
261         glClear(GL_COLOR_BUFFER_BIT);   // resetando a tela(zera todas as linhas)
262         glLineWidth(1); // Largura das linhas das linhas de escala
263         glColor3f(1.0f,1.0f,1.0f);  // cor cinza atribuÃda ao background
264
265         glEnable(GL_LINE_STIPPLE);  // habilita a pixelizaÃ§Ã£o de linha
266            glLineStipple(1,0x8080); // padrÃ£o de pixels: 1000000010000000, repete atÃ© o tÃ©rmino da linha
267            glEnableClientState(GL_VERTEX_ARRAY);   // habilita a escrita de um  array
268               glVertexPointer(2,GL_FLOAT,0,Background); // DimensÃ£o (x,y) | tipo | gap(0)| *coordenadas
269               glDrawArrays(GL_LINES,0,44); // NÃºmero de pares coordenados (x,y) a ser escrito
270            glDisableClientState(GL_VERTEX_ARRAY);  // Desabilita a escrita em array
271         glDisable(GL_LINE_STIPPLE); //  Desabilita a funÃ§Ã£o "stipple"
272
273         glLineWidth(2); // Espessura da linha (3 pixels)
274
275          for(i = 0; i< N/2 -1; i++){
276            Aux = (int)msp430_ret[i]*(11-amp)/5+49;
277            if(abs(Aux - trigger) < last_value){
278               last_value = Aux - trigger;
279               index = i;
280            }
281            if(Aux > val_pico)
282               val_pico = Aux;
283          }
284
285        for(i = index; i< index + 24; i++){
286
287
288               Senoid[((i - index)<<2)+1] = (int)(msp430_ret[i]-128)*(6-amp)/3+193;//(100*amp*sin(2*pi*i/12)+250)-50;      //
Amplitude inicial
289            Senoid[((i- index)<<2)+2] = T1 + (i-index)*20.08;
290              Senoid[((i - index)<<2)+3] = (int)(msp430_ret[i+1]-128)*(6-amp)/3+193;//(100*amp*sin(2*pi*(i+1)/12)+250)-50;
// Amplitude "amostrada" a "x" ms depois
291
```

```
292         if(Senoid[((i - index)<<2)+1] > MARG_CIMA)    // Limitando o sinal para a escala do grÃ¡fico
293            Senoid[((i - index)<<2)+1] = MARG_CIMA;
294         else if(Senoid[((i - index)<<2)+1] < 50)
295            Senoid[((i - index)<<2)+1] = 50;
296         if(Senoid[((i - index)<<2)+3] > MARG_CIMA)
297            Senoid[((i - index)<<2)+3] = MARG_CIMA;
298         else if(Senoid[((i - index)<<2)+3] < 50)
299            Senoid[((i - index)<<2)+3] = 50;
300
301
302         glEnableClientState(GL_VERTEX_ARRAY);   // Escrita dos dados na tela
303            glVertexPointer(2,GL_FLOAT,0,Senoid);   // constrÃ³i vÃ©rtices a partir do vetor Senoid
304            glDrawArrays(GL_LINES,0,50); // Desenha na tela as 2 amostras mais atuais.
305         glDisableClientState(GL_VERTEX_ARRAY);
306         glColor3f(1.0f, 1.0f, 1.0f);    //cor branca
307         glRasterPos3f(270, 20,0.0);     //PosiÃ§Ã£o do texto
308
309         {
310            case 1:
311            glutBitmapString(GLUT_BITMAP_HELVETICA_18, "M20us");
312            break;// 25 amostras
313            case 2:
314            glutBitmapString(GLUT_BITMAP_HELVETICA_18, "M40us");
315            break; // 100 amostras
316            case 3:
317            glutBitmapString(GLUT_BITMAP_HELVETICA_18, "M80us");
318            break; // 200 amostras
319            case 4:
320            glutBitmapString(GLUT_BITMAP_HELVETICA_18, "M160us");
321            break; // 1000 amostras
322            case 5:
323            glutBitmapString(GLUT_BITMAP_HELVETICA_18, "M320us");
324            break; // 2000 amostras
325            case 6:
326            glutBitmapString(GLUT_BITMAP_HELVETICA_18, "M0.64ms");
327            break; // 2M amostras
328            default:
329            glutBitmapString(GLUT_BITMAP_HELVETICA_18, "Error");
330         }
331         glRasterPos3f(550, 320,0.0);
332
333         sprintf(v_pico, "%d",val_pico); // Transforma inteiro para const *char
334
335         glutBitmapString(GLUT_BITMAP_HELVETICA_18, "Info\n\n\n-------\n");
336         glutBitmapString(GLUT_BITMAP_HELVETICA_18, v_pico);
337         glutBitmapString(GLUT_BITMAP_HELVETICA_18, "Vp\n-------\n");
338         glutBitmapString(GLUT_BITMAP_HELVETICA_18,"\nDigital\nSignal\nAnalyser");
339
340         glRasterPos3f(450, 20,0.0);     //PosiÃ§Ã£o do texto
341
342         glColor3f(0.0f, 0.5f, 1.0f);    // cor azul
343         glRasterPos3f(50, 20,0.0);
344
345         {
346            case 1:
```

```
347             glutBitmapString(GLUT_BITMAP_HELVETICA_18, "CH1: 50mV/div");
348             in_min = 56;
349
350             break;
351             case 2:
352             glutBitmapString(GLUT_BITMAP_HELVETICA_18, "CH1: 100mV/div");
353             in_min = 88;
354
355             break;
356             case 3:
357             glutBitmapString(GLUT_BITMAP_HELVETICA_18, "CH1: 200mV/div");
358             in_min = 112;
359
360             break;
361             case 4:
362             glutBitmapString(GLUT_BITMAP_HELVETICA_18, "CH1: 500mV/div");
363             in_min = 120;
364
365             break;
366             case 5:
367             glutBitmapString(GLUT_BITMAP_HELVETICA_18, "CH1: 1V/div");
368             in_min = 124;
369
370             break;
371             default:
372             glutBitmapString(GLUT_BITMAP_HELVETICA_18, "CH1: 1V/div", amp);
373           }
374           glFlush();
375           val_pico = 0;
376           //}
377           //glFlush();
378           //i++;
379           glutPostRedisplay();    // Retorna ao inicio dessa funÃ§Ã£o (Looping infinito)
380
381         //while(1);
382     }
383
384     void main(int argc, char**argv) {
385       pthread_t thread_GPIO1, thread_GPIO2,thread_id;
386       GPIO_setup();
387       signal(SIGINT, ctrl_c);
388       i2c_fd = open("/dev/i2c-1", O_RDWR);
389       ioctl(i2c_fd, I2C_SLAVE, slave_addr);
390       glutInit(&argc, argv);  // Ã‰ possÃvel alterar as configuraÃ§Ãµes de execuÃ§Ã£o pelos argv e argc
391       glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);    // BufferizaÃ§Ã£o simples, RGB
392       glutInitWindowPosition(0,0);    // PosiÃ§Ã£o inicial da janela em relaÃ§Ã£o ao lado esquerdo/cima
393       glutInitWindowSize(COMP_MAX,ALTURA_MAX);    // Delimitando limites (escalas em pixels)
394       glutCreateWindow("Osciloscope");    // Nome da janela~]]
395       back_setup();   // ConfiguraÃ§Ãµes iniciais da janela
396       pthread_create(&thread_GPIO1, NULL, &encoder_Read1, NULL);
397       pthread_create(&thread_GPIO2, NULL, &encoder_Read2, NULL);
398       pthread_create (&thread_id, NULL, &set_polling, NULL);
399       glutDisplayFunc(main_loop); // funÃ§Ã£o que comandarÃ¡ a atualizaÃ§Ã£o da tela
400       glutMainLoop(); //Garante que a janela ficarÃ¡ aberta durante a execuÃ§Ã£o do programa.
401       close(i2c_fd);  // Por fim das contas, fechar o i2c
```