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 e escrita de dados com interface gráfica de controle e flexível quanto à obtenção de dados, a partir da Raspberry Pi. Esses equipamentos, tais 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; gerador de funções; Aquisição de dados; Raspberry-Pi;

I. JUSTIFICATIVA

O osciloscópio é uma importante ferramenta no campo das medições elétricas, sendo muito utilizado por estudantes de Engenharia Elétrica e Física.

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

Esta ferramenta a ser desenvolvida propõe um analisador gráfico de sinais elétricos, bem como sua análise transiente e em frequência.

II. OBJETIVOS

Este projeto tem como objetivo desenvolver uma ferramenta capaz de realizar leituras de tensões com ajuste na visualização da forma de onda e envio da análise para suporte externo.

No caso de liquidação do quadro de planejamento,visto na figura 1, pode ainda ser desenvolvido uma funcionalidade a mais para o projeto, um gerador de funções para complementar a sua utilidade.

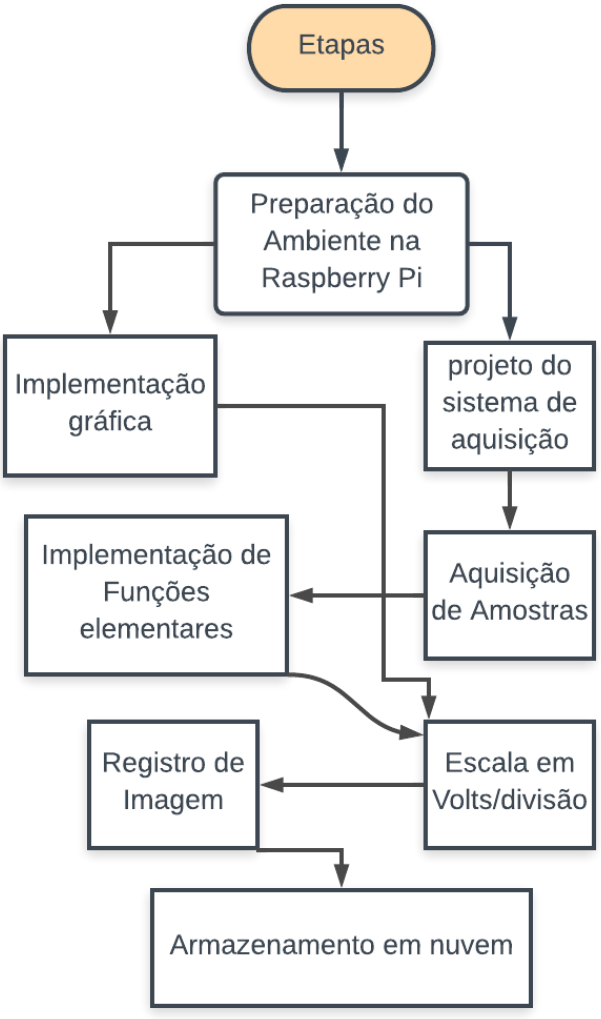

Fig. 1 - Quadro de planejamento.

III. 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 (FFT, escala, cursores, trigger) e conexão compatível com pontas de prova cabos de laboratório. Os requisitos do sistema encontra-se na tabela 1.

TABELA I. REQUESITOS 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 2 ilustra a configuração inicial dos componentes necessários para elaboração do projeto.

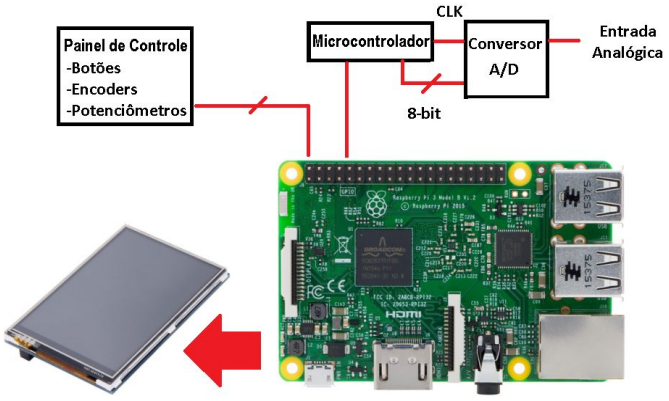

Fig. 2 - Diagrama esquemático do hardware.

IV. 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.

V. PROCEDIIMENTOS EXPERIMENTAIS

A. Aquisição

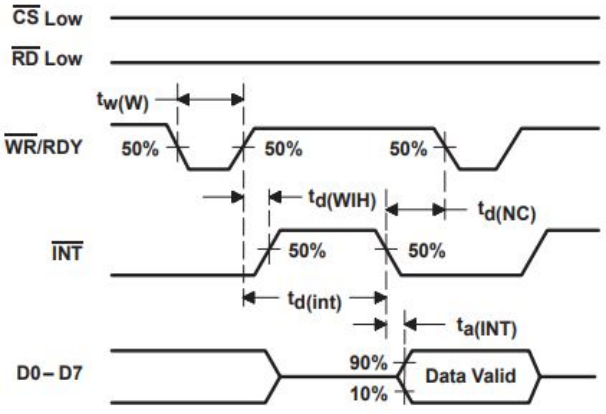

Fig. 3 - Modo de operação Stand-Alone. [6]

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 3.

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 40 cisclos de clock, adequados para frequência de amostragem do conversor ADC.

A aquisição é seguida de transferência de para o raspberry pela comunicação I2C, com devidadas confgurações.

B. 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 4.

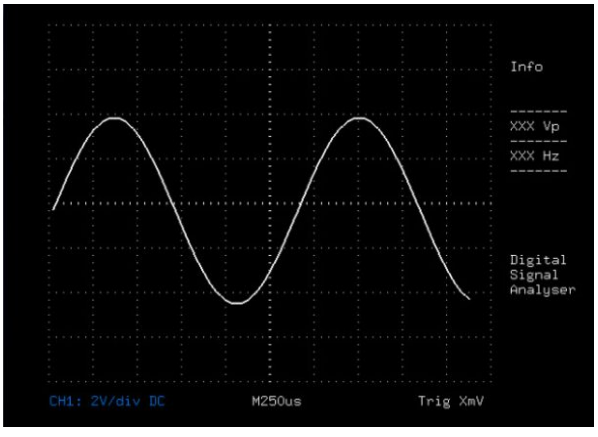

Fig.4 - Teste gráfico com a biblioteca OpenGL.

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 a ferramenta disponível, ferro de solda.

VI. 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	-

REFERENCES

[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.

ANEXOS

```
1   // refresh2_opengl.c - Código teste com a biblioteca OPENGL
2
3   #include <stdio.h>      //  libs para printf e tals
4   #include <stdlib.h>     //
5   #include <unistd.h>     //  biblioteca para sleep
6   #include <GL/glut.h>    //  biblioteca opengl
7   #include <GL/gl.h>      //  biblioteca opengl
8   #include <math.h>       //  para simular uma senoide
9
10  #define COMP_MAX 720
11  #define ALTURA_MAX 400
12  #define MARG_ESQUERDA 50
13  #define MARG_DIREITA COMP_MAX-186
14  #define MARG_CIMA 450
15  #define MARG_BAIXO 58
16  #define pi 3.141592f
17
18  GLfloat buffer_s[500] = {0};  // Armazena os valores anteriores da tela
19  GLfloat Senoid[] = {
20     MARG_ESQUERDA,200, // coordenada iniciais
21     (MARG_ESQUERDA+3.8),200
22  };
23  GLfloat amp = 1.0f;
24
25  GLfloat Background[] = {
26
27     MARG_ESQUERDA,65,MARG_DIREITA,65,   50,MARG_BAIXO,50,MARG_CIMA,
28     MARG_ESQUERDA,113,MARG_DIREITA,113, 98,MARG_BAIXO,98,MARG_CIMA,
29     MARG_ESQUERDA,161,MARG_DIREITA,161, 146,MARG_BAIXO,146,MARG_CIMA,
30     MARG_ESQUERDA,209,MARG_DIREITA,209, 194,MARG_BAIXO,194,MARG_CIMA,
31     MARG_ESQUERDA,256,MARG_DIREITA,256, 242,MARG_BAIXO,242,MARG_CIMA,
32     MARG_ESQUERDA,257,MARG_DIREITA,257, 290,MARG_BAIXO,290,MARG_CIMA,
33     MARG_ESQUERDA,305,MARG_DIREITA,305, 291,MARG_BAIXO,291,MARG_CIMA,
34     MARG_ESQUERDA,353,MARG_DIREITA,353, 338,MARG_BAIXO,338,MARG_CIMA,
35     MARG_ESQUERDA,401,MARG_DIREITA,401, 386,MARG_BAIXO,386,MARG_CIMA,
36     MARG_ESQUERDA,449,MARG_DIREITA,449, 434,MARG_BAIXO,434,MARG_CIMA,
37     MARG_ESQUERDA,258,MARG_DIREITA,258, 482,MARG_BAIXO,482,MARG_CIMA,
38                                 530,MARG_BAIXO,530,MARG_CIMA,
39                                 291,MARG_BAIXO,291,MARG_CIMA,
40                                 289,MARG_BAIXO,289,MARG_CIMA
41  };
42
43  void back_setup() {
44   // ------ Redimentionando a tela para unidades de pixels ---- //
45     glViewport(0.0f, 0.0f, COMP_MAX, ALTURA_MAX);
46     glMatrixMode(GL_PROJECTION);
47     glLoadIdentity();   // update depois de mecher com a matriz
48     glOrtho(0, COMP_MAX, 0, ALTURA_MAX, 0, 1);  // setando os limites da tela
49     glMatrixMode(GL_MODELVIEW); // default values
50     glPushMatrix();
51     glLoadIdentity();   //Sempre atualizar depois de setar funções de Matriz
52  // ------------------------------------------------------------//
53  }
```

```
54
55      int i = 0;
56
57      void main_loop(){
58
59          glClearColor(0,0,0,0);  // Cor preta (nackgroung)
60          glClear(GL_COLOR_BUFFER_BIT);   // resetando a tela(zera todas as linhas)
61          glLineWidth(1); // Largura das linhas das linhas de escala
62          glColor3f(1.0f,1.0f,1.0f);  // cor cinza atribuída ao background
63
64          glEnable(GL_LINE_STIPPLE);  // habilita a pixelização de linha
65              glLineStipple(1,0x8080); // padrão de pixels: 1000000010000000, repete até o término da linha
66              glEnableClientState(GL_VERTEX_ARRAY);   // habilita a escrita de um  array
67                  glVertexPointer(2,GL_FLOAT,0,Background); // Dimensão (x,y) | tipo | gap(0)| *coordenadas
68                  glDrawArrays(GL_LINES,0,50); // Número de pares coordenados (x,y) a ser escrito
69              glDisableClientState(GL_VERTEX_ARRAY);  // Desabilita a escrita em array
70          glDisable(GL_LINE_STIPPLE); //  Desabilita a função "stipple"
71
72          //glFlush();  // sempre que chamar essa função a tela é atualizada
73          //glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24);
74
75          //------------ Essa rotina funcionou----------------
76          glLineWidth(2); // Espessura da linha (3 pixels)
77
78          //glColor3f(1.0f,1.0f,1.0f);  // cor branca
79          //glEnableClientState(GL_VERTEX_ARRAY);
80          //i++;
81          //Senoid[0] = MARG_ESQUERDA;
82          //Senoid[2] = MARG_ESQUERDA+1;
83          //amp = amp*0.85;
84          /*
85              OBS: As amostras são tradadas aos pares, visto que é necessário dois pontos para
86              desenhar um segmento de reta. Dessa forma, sempre a primeira posição de um vetor
87              será o valor da amostra no tempo "x" e a segunda será o valor de   sua amplitude
88              "y". O mesmo ocorre para outras posições de vetores com "N" posições.  Logo,  as
89              posições pares representam dados de "tempo" e as posições ímpares de amplitude.
90          */
91          if( i == 124){
92              i = 0;
93              Senoid[0] = MARG_ESQUERDA;   // Reinicia a escala de tempo para o início da tela
94              Senoid[2] = MARG_ESQUERDA+3.8; // Escala do segundo ponto
95              amp = amp*0.85;             // Atualiza a amplitude para testes de reescrita na tela
96          }
97          //for(i = 0; i < 125; i++){
98              //teste com senoide
99              Senoid[0] += 3.8;   // incrementa a escala de tempo simulando o incremento de "amostras" lidas
100             Senoid[2] += 3.8;
101
102             // OBS: A frequencia abaixo está em função da taxa de amostragem
103             Senoid[1] = (int)(100*amp*sin(2*pi*i/70)+250);     // Amplitude inicial
104             Senoid[3] = (int)(100*amp*sin(2*pi*(i+1)/70)+250); // Amplitude "amostrada" a "x" ms depois
105
106             // adicionando as 2 coordenadas (amostras) no buffer.
107             buffer_s[i<<2] = Senoid[0];
108             buffer_s[(i<<2)+1] = Senoid[1];
```

```
109             buffer_s[(i<<2)+2] = Senoid[2];
110             buffer_s[(i<<2)+3] = Senoid[3];
111
112                                             //printf("senoid[0]:  %.2f  senoid[1]:  %.2f  senoid[2]:  %.2f  senoid[3]:
%.2f\n",Senoid[0],Senoid[1],Senoid[2],Senoid[3]);
113             //printf("senoid[1]: %.2f senoid[3]: %.2f\n",Senoid[1],Senoid[3]);
114             //glBegin(GL_LINES);
115             //printf("buffer[%d]: %.1f buffer[%d]: %.1f\n ", i*2,buffer_s[i*2],i*2+1, buffer_s[i*2-1]);
116             glEnableClientState(GL_VERTEX_ARRAY);   // Escrita dos dados na tela
117                glVertexPointer(2,GL_FLOAT,0,Senoid);   // constrói vértices a partir do vetor Senoid
118                glDrawArrays(GL_LINES,0,2); // Desenha na tela as 2 amostras mais atuais.
119                glVertexPointer(2,GL_FLOAT,0,buffer_s);
120                glDrawArrays(GL_LINES,0,249);   // Desenha os anteriores guardados no buffer.
121             glDisableClientState(GL_VERTEX_ARRAY);
122             i++;
123             //glDisableClientState(GL_VERTEX_ARRAY);
124
125             usleep(20000);  // taxa de amostragem (10ms).
126
127             glColor3f(1.0f, 1.0f, 1.0f);    //cor branca
128             glRasterPos3f(270, 40,0.0);     //Posição do texto
129                glutBitmapString(GLUT_BITMAP_9_BY_15, "M250us");    //Escreve a escala na tela
130             glRasterPos3f(550, 400,0.0);
131             glutBitmapString(GLUT_BITMAP_9_BY_15, "Info\n\n\n-------\nXXX Vp\n-------\nXXX Hz\n-------\n");
132             glutBitmapString(GLUT_BITMAP_9_BY_15,"\n\n\n\n\nDigital\nSignal\nAnalyser");
133             glRasterPos3f(450, 40,0.0);     //Posição do texto
134                glutBitmapString(GLUT_BITMAP_9_BY_15, "Trig XmV");    //Escreve a escala na tela
135             glColor3f(0.0f, 0.5f, 1.0f);    // cor azul
136             glRasterPos3f(50, 40,0.0);
137             glutBitmapString(GLUT_BITMAP_9_BY_15, "CH1: 2V/div DC");
138
139             glFlush();
140             //}
141             //glFlush();
142             //i++;
143          glutPostRedisplay();    // Retorna ao inicio dessa função (Looping infinito)
144          //while(1);
145       }
146
147       void main(int argc, char**argv) {
148          glutInit(&argc, argv);  // É possível alterar as configurações de execução pelos argv e argc
149          glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);    // Bufferização simples, RGB
150          glutInitWindowPosition(0,0);    // Posição inicial da janela em relação ao lado esquerdo/cima
151          glutInitWindowSize(COMP_MAX,ALTURA_MAX);    // Delimitando limites (escalas em pixels)
152          glutCreateWindow("Osciloscope");    // Nome da janela
153          back_setup();   // Configurações iniciais da janela
154          glutDisplayFunc(main_loop); // função que comandará a atualização da tela
155          glutMainLoop(); //Garante que a janela ficará aberta durante a execução do programa.
156       }
```