

Medidor de Indutância e Capacitância

MICROPROCESSADORES E MICROCONTROLADORES

Filipe Alves de Sousa (15/0125429), Flavio Vieira Leão (15/0125682)

Engenharia Eletrônica

Faculdade Gama - Universidade de Brasília

Gama, DF

E-mail: fylypew@gmail.com, flavio.vl@gmail.com

Diogo Caetano Garcia

RESUMO

Este projeto tem o intuito de construir um medidor, pequeno e de simples uso, de Indutância e Capacitância utilizando um microcontrolador MSP430 de baixo consumo de potência desenvolvida pela Texas Instruments.

I. INTRODUÇÃO

De acordo com o Inmetro, instrumentos de medição são dispositivos utilizados para mensurar grandezas físicas, tornando possível a obtenção de dados, em um determinado sistema e em diferentes contextos e aplicações. [1]

O uso de equipamentos que medem de forma precisa, conjuntamente com os avanços em pesquisas e experimentos voltados a Engenharia Eletrônica, possibilitaram operações sofisticadas que auxiliam o contínuo estudo de componentes. Como no caso da análise de capacitores e indutores, cujas propriedades de interesse a serem verificadas são respectivamente, capacidade e indutância.

A capacidade é uma grandeza escalar, dada pela quantidade de energia elétrica que fica armazenada no campo elétrico de um capacitor, cuja unidade de medida é o Farad [F]. E a indutância é uma grandeza, determinada pela quantidade de energia elétrica que fica armazenada no campo magnético de um indutor, medida em uma unidade conhecida como Henry [H]. [2]

Devido à grande demanda por quantificar/medir e contribuir com outras tantas particularidades de circuitos com os quais trabalha-se com estes componentes, foi proposta a implementação de um instrumento capaz de aferir

as grandezas supracitadas.

II. OBJETIVO

O objetivo do projeto é de desenvolver um instrumento de fácil manuseio e baixo custo, utilizando o MSP430, cuja finalidade é medir as grandezas Indutância e Capacitância de cada componente eletrônico a ser analisado, respeitando-se os limites de tolerância em relação aos seus respectivos valores teóricos.

III. REQUISITOS

O medidor deverá realizar, de forma satisfatória, medidas precisas de capacitores e indutores:

Capacitância: Faixa de 20pF até centenas de microfarads com precisão de até $\pm 5\%$.

Indutância: Faixas de 20 μ H até centenas de milihenry com precisão de até $\pm 5\%$.

IV. JUSTIFICATIVA

Os equipamentos de medição são de suma importância para quem trabalha com eletrônica. Visto que eles permitem a análise de defeitos, diagnósticos e aferição de valores reais dos componentes.

Existem muitos multímetros que fazem medidas de indutância e capacidade, porém, geralmente são equipamentos de alto custo, difícil acesso e que, muitas das vezes, não são disponibilizados aos alunos de cursos técnicos e de

graduação.

A motivação deste projeto consiste em complementar os instrumentos básicos de medição, já existentes, para que todo aluno de eletrônica possa comprar ou fazer o seu medidor de capacidade e indutância.

V. BENEFICIOS

O instrumento a ser projetado poderá ser utilizado para auxiliar na verificação de valores de capacidade e indutância de forma prática, facilitando a identificação de cada componente a ser medido. Devido ao MSP430 ser um microcontrolador de baixo consumo energético e por ser um equipamento portátil, o medidor poderá ser alimentado com uma bateria ou com um laptop comum. Além do fácil manuseio, esse dispositivo armazenará os valores medidos, será de baixo custo e poderá melhorar de forma significativa o desempenho em estudos de circuitos eletrônicos.

VI. DESCRIÇÃO DO HARDWARE

Serão utilizados os seguintes componentes:

- Microcontrolador MSP430G2553;
- Display LCD– 16x2;
- Bateria de alimentação 9V;
- Protoboard;
- Resistores: 3x100K, 2x47K e 100;
- Indutor: 82uH
- Capacitores: 2x1nF, 10uF e 0.1uF ;
- LM7805;
- Fios e Jumpers Macho;
- Botões

O circuito analógico foi separado em 3 partes, uma para medir indutância e duas para capacidade (capacidade até 1 nF e a outra para maiores que 1nF).

A medição de capacitores menores e indutores foi usada por meio da abordagem de um circuito ressonante LC (tanque de ressonância). Usando um indutor e capacitor de precisão conhecidos, foi gerada uma frequência de oscilação inicial, após isso, com o oscilador funcionando, foi adicionado um indutor desconhecido em série ou um capacitor desconhecido em paralelo com o

ressonador, gerando uma nova frequência de oscilação. Assim, foi possível medir a variação da frequência provocada adicionando um indutor ou capacitor desconhecido e determinar o valor desconhecido calculando-se a partir das equações de frequência de ressonância abaixo:

$$f_r = \frac{1}{2\pi\sqrt{LC}}$$

$$f_1 = \frac{1}{2\pi\sqrt{L(C+C_x)}}$$

$$f_2 = \frac{1}{2\pi\sqrt{(L+L_x)C}}$$

$$L_x = [(\frac{f_r}{f_2})^2 - 1] L$$

$$C_x = [(\frac{f_r}{f_1})^2 - 1] C$$

Onde:

f_r é a frequência de oscilação sem adicionar nenhum componente desconhecido.

f_1 é frequência de oscilação com o capacitor desconhecido adicionado em série.

f_2 é frequência de oscilação com o indutor desconhecido adicionado em paralelo.

L é a indutância do tanque conhecida.

C é a capacidade do tanque conhecida.

L_x é o valor da indutância desconhecida.

C_x é o valor da capacidade desconhecida.

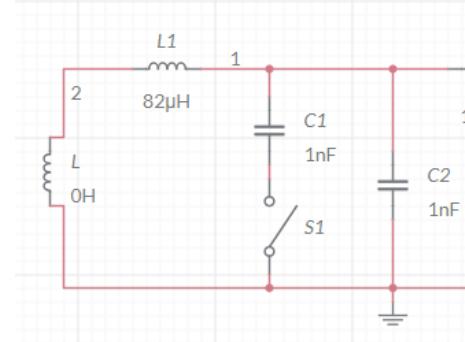


Figura 1. Circuito ressonante LC com indutor desconhecido

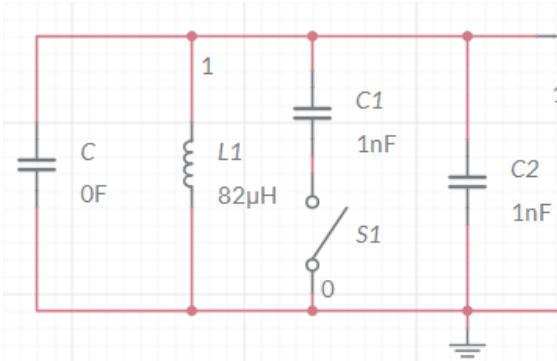


Figura 2. Circuito ressonante LC com capacitor desconhecido

Na medição de capacitores maiores que 1nF, foi usado o ciclo de carga e descarga, associando-se o capacitor a um resistor conhecido, com um timer preciso, onde é medido o tempo de carga e descarga deste capacitor. Como o tempo de carga é linear, ele foi usado para calcular o valor da capacidade a partir da equação da constante de tempo abaixo:

$$\tau = RC$$

$$C = \frac{\tau}{R}$$

Onde: T é a constante de tempo, C é a capacidade e R é a resistência.

Diagrama de blocos funcional do projeto.

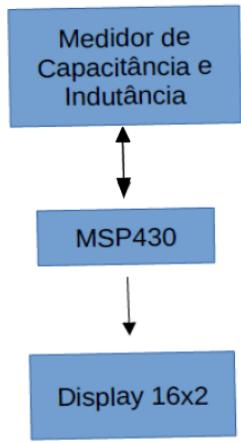


Figura 3. Diagrama de blocos ilustrando o circuito

A parte digital consiste na comunicação entre o microcontrolador e o display. Esse display recebe os pinos de conexão do microcontrolador ao encaixar na LaunchPad.

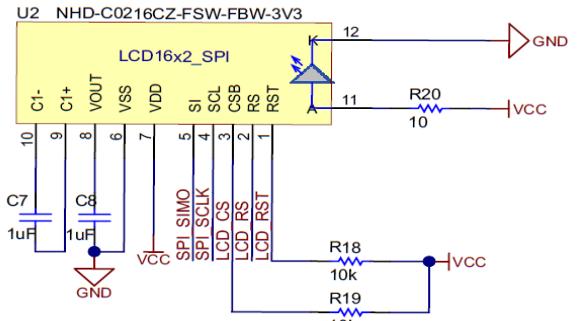


Figura 4. Esquemático do display utilizado

No circuito, a parte do medidor LC (Capacitância até 1nF) utiliza os pinos P1.1, P1.3, P1.6. O comparador A foi usado junto com outros componentes passivos para formar um circuito de tanque LC. Ele oscila em uma frequência com base nos valores do indutor e do capacitor.

Um interruptor foi usado para selecionar o modo de medição de capacidade ou indutância. Com o componente capacitor/indutor conectado, pode-se obter a frequência da oscilação e verificar o valor L ou C, segundo as equações citadas.

A medição de capacitor de alta faixa utiliza os pinos P1.4 e P1.3. O pino P1.4 do microcontrolador foi configurado como entrada inversora do comparador A (-) e utilizou-se uma tensão de 0.55V interna na entrada não-inversora (+).

Configurou-se a interrupção do comparador para disparar o Timer A e medir o tempo que leva para carregar um capacitor a ser medido com tensão de 0.55V. Sabendo que o tempo de carga é linear à capacidade, foi usado um valor calibrado armazenado como referência para calcular a capacidade do elemento testado.

O pino P1.3 foi usado para iniciar a carga do capacitor testado através de um resistor de 47k. Então, devido a demora ocorrida durante as medições de capacitores com valores maiores, foi adicionado um resistor com objetivo de medir o valor da resistência interna dos capacitores, ESR (pelo pino P1.2), obtendo-se assim um maior alcance.

Desta forma, foi possível carregar o capacitor desconhecido através do pino P1.3 (resistor de 47k) sendo que quando ele atingia 50 vezes a quantidade de ciclos, pode-se conectá-lo

ao local de medir via pino P1.2 (com resistor de 100 ohms), para carregar o capacitor testado em menor tempo.

A medição de ESR foi baseada na medição de capacitância de faixa alta (pinos P1.4, P1.3) com um pino P1.2 adicional. Onde utilizou-se o pino P1.4 como entrada ADC. O resistor de 100 ohms foi usado entre os pinos P1.2 e P1.4, com um capacitor de teste no lugar entre o pino P1.4 e o gnd, esses três nós (P1.2, P1.4 e Gnd) formaram um divisor de tensão. Medindo a tensão em P1.4 para encontrar a resistência interna (ou ESR) do capacitor testado.

Conforme a imagem do esquemático a seguir:

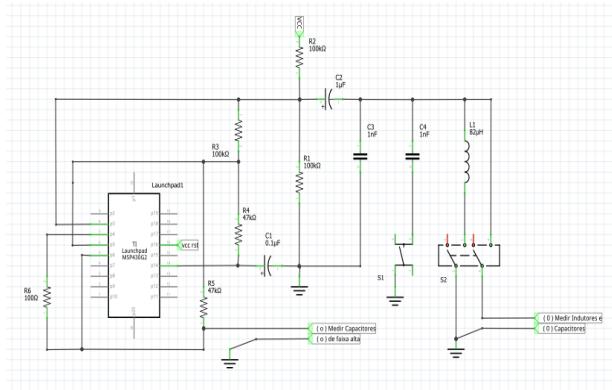


Figura 5. Esquemático do circuito medidor de capacitância

A imagem a seguir mostra o circuito medidor montado:

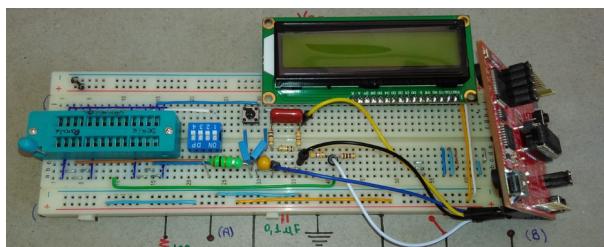


Figura 6. Foto da montagem do medidor implementado em Protoboard

O medidor de capacitância e indutância é composto por um circuito analógico elaborado para fornecer os sinais de tensão ao microprocessador MSP430, que por sua vez, integrará os periféricos utilizados, fará a aquisição e o processamento dos dados e os apresentará no display.

VII. DESCRIÇÃO DO SOFTWARE

Foi possível implementar, em linguagem C, um código que descreve o funcionamento conjunto do circuito analógico conectado ao microprocessador para aferir e exibir as medidas dos indutores e capacitores.

Utilizou-se a plataforma de desenvolvimento Code Composer Studio, versão 7, da Texas Instruments, uma plataforma onde a programação do microcontrolador iniciou-se, após a criação do projeto, com a declaração das bibliotecas (stdio.h, para o uso de funções de captura e impressão de dados, stdint.h, para definir os tipo de dados inteiros e msp430g2553.h, para chamadas de funções de bits e bytes do hardware) e das variáveis usadas no código (conexões de pinos analógicos para medir tensão no circuito). Conforme exposto a seguir:

```
#include <msp430g2553.h>
#include <stdint.h>
#include <stdio.h>
```

Foi definida uma frequência de 16 MHz, e para inicialização o display, foram definidos os pinos de saída e escrita de dados, conforme a pinagem do display configurado no modo 4 pinos de dados. Da seguinte maneira:

```
// define RS high
#define DR P1OUT = P1OUT | BIT0
// define RS low
#define CR P1OUT = P1OUT & (~BIT0)
// define Read signal R/W = 1 for reading
#define READ P1OUT = P1OUT | BIT1
// define Write signal R/W = 0 for writing
#define WRITE P1OUT = P1OUT & (~BIT1)
// define Enable high signal
#define ENABLE_HIGH P1OUT = P1OUT | BIT2
// define Enable Low signal
#define ENABLE_LOW P1OUT = P1OUT & (~BIT2)
```

Após ajustar o tempo de atraso para cada função, foi criada a função lcd_init, possibilitando limpar, escrever e deslocar dados para a inicialização do display.

Foram declaradas funções que usam o tipo unsigned int para realizar a captura de dados.

Na função `uint16_t capture_pulses()`, o (ADC10CTL0 e ADC10CTL1, o registrador de controle e (ADC10AE0) o registrador de habilitação do Comparador A são zerados.

Os pinos P1.3 e P1.4 foram configurados em nível lógico baixo (VSS) e em seguida, direcionados a entrada de dados.

O CACTL1 |= CAIE foi usado para fazer o comparador interromper a contagem de pulsos do tanque LC.

A função 'capture_cnt' foi utilizado para quando o número de pulsos estiver dentro do período de overflow do timerA, sendo que a frequência do tanque LC seria 16Mhz / 64k * 'capture_cnt'

A função `measure_high_cap` foi utilizada para medir capacitores maiores que 1nF. A tensão de referência na entrada não inversora do comparador foi configurada para 0.55V, onde o capacitor seria medido na entrada e pelo tempo de inversão. Nessa estrutura, acontece a verificação de quanto tempo leva para o capacitor desconhecido carregar em 0.55V. O comparador para foi configurado para disparar uma interrupção timerA quando o 0.55V é atingido. Conforme o fluxograma abaixo:

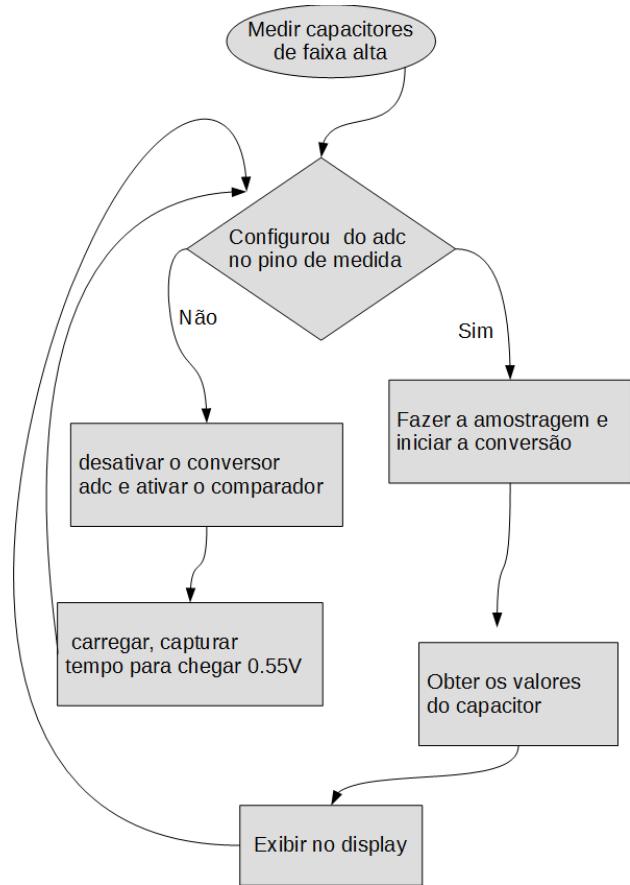


Figura 7. Fluxograma simplificado da função que mede capacidade

A conversão de unidades foi realizada com adc em mVs, 10bit adc em referência de 1,5V, mais ainda 4x em amostras.

O valor do ESR foi obtido via lei de ohm, $V=IR$, onde a corrente de pulso é $(3.6V - pulse pin VDrop) / (R (100ohm used) + pin resistance)$. A tensão de leitura foi dividida com a corrente de pulso para obter ESR.

Foram montadas estruturas lógicas condicionais como os valores recebidos (em ponto flutuante) da parte matemática para as frequências obtidas durante a medição.

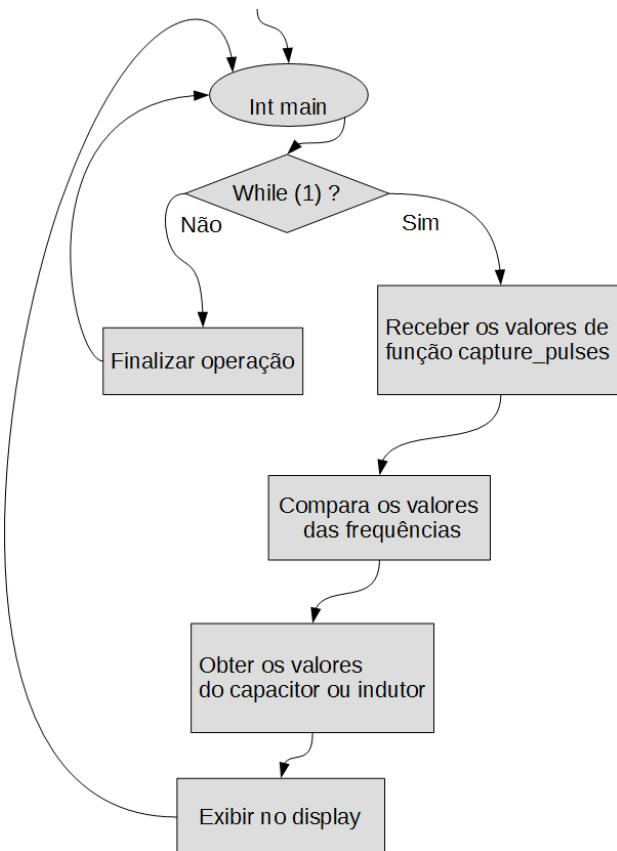


Figura 8. Fluxograma simplificado da função main

Na função main, foram definidos valores de frequências para comparar com os valores das frequências recebidas pela função capture_pulses, bem como foram determinados os dados a serem mostrados no display.

VIII. RESULTADOS

Aqui pode-se verificar a implementação do medidor de capacidade e indutância, onde existe um circuito analógico elaborado para fornecer os sinais de tensão ao microprocessador MSP430, que por sua vez, integrou os periféricos utilizados para fazer a aquisição e o processamento dos dados e os apresentá-los no display.

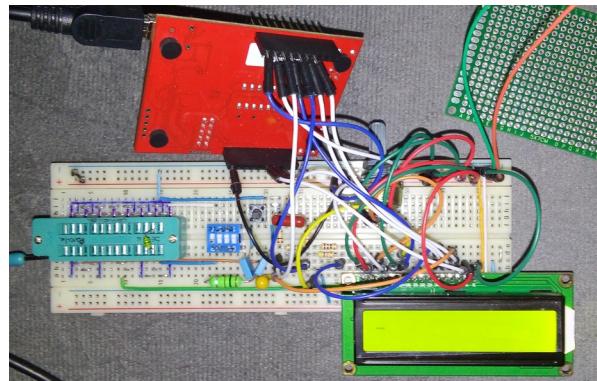


Figura 9. Imagem do projeto sem funcionar

Foi possível verificar que as dificuldades encontradas na implementação desse projeto envolveu a comunicação entre o circuito analógico e o microcontrolador. Isso prejudicou o desenvolvimento do projeto para o ponto de controle 4. Entretanto, é possível compreender quais foram os erros cometidos de forma e buscar solucioná-los de forma a implementar e adequar o projeto.

A partir dos testes realizados, foram obtidos os resultados mostrados a seguir juntamente com os dados obtidos e a implementação de hardware dos circuitos.



Figura 10. Exibição inicial do display LCD

Aqui pode-se notar que os circuitos são simples de implementar, como componentes básicos são usados existindo uma redução na complexidade e também torna-o um dispositivo econômico:

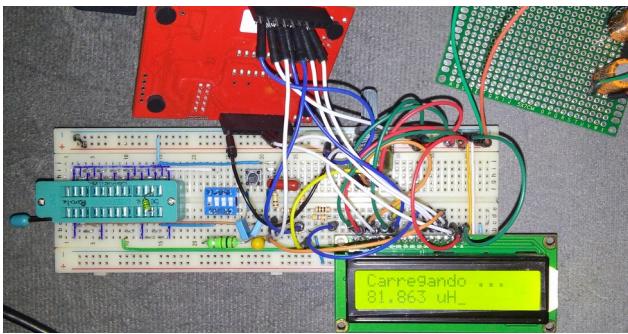


Figura 11. Exibição do valor de indutância medida

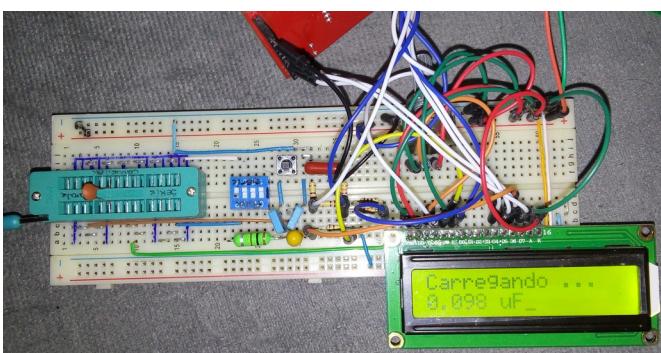


Figura 12. Exibição do valor de capacitor medido

Ao comparar os valores medidos e comerciais, percebeu-se que foram obtidos resultados alguns resultados inesperados, em testes de exibição dos valores aferidos no display.

Isso causou um atraso na conclusão do projeto, sendo possível verificar o alguns resultados esperados posteriormente.

Comparação dos valores obtidos com os valores comerciais do indutor e do capacitor:

Tabela1. Valores comerciais x Valores medidos

Valor	Capacitância	Indutância
Comercial	0.1 [uF]	82 [uH]
Medido	0.098[uF]	81.863 [uH]

É importante levar em consideração que, nesse projeto, as equações e simulações dos circuitos utilizados para modelar o comportamento dos componentes foram formas simplificadas que unem diferentes informações e características do experimento prático. Apesar de estarem próximos dos valores teóricos, alguns valores obtidos não

foram exatamente iguais aos valores obtidos na teoria e nas simulações. Isso aconteceu devido às limitações na modelagem dos circuitos, onde algumas variáveis associadas (desde físicas diretamente relacionadas aos componentes até variáveis ambientais) não foram consideradas no cálculos e simulações.

VIX. CONCLUSÃO

A utilização de indutores e capacitores em circuitos elétronicos é de grande importância, e assim, vê-se necessário o desenvolvimento de medidores de indutância. Esse estudo contribui com informações para projetar um medidor de indutores e capacitores simples e de baixo custo.

O instrumento projetado pode ser adaptado para que se verifique o seu correto funcionamento, onde poderá ser utilizado para verificar de valores de capacitância e indutância de forma prática, facilitando a identificação de cada componente a ser medido. Devido ao MSP430 ser um microcontrolador de baixo consumo energético e por ser um equipamento portátil, o medidor poderá ser alimentado com uma bateria ou com um laptop comum. Além do fácil manuseio, esse dispositivo, é de baixo custo e pode melhorar de forma significativa o desempenho em estudos de circuitos.

Quando comparado com outros projetos, verificou-se que existem algumas coisas faltando, isso poderá afetar a precisão e a usabilidade, no entanto esse é um modelo experimental a qual pode-se acrescentar relés ou mesmo mais interruptores, já que a configuração adotada não tem pinos de IO suficientes disponíveis (quanto conectado ao microcontrolador). Também podem ser acrescentados transistores para acionar o carregamento, desarmar e comutar e para construir um projeto mais confiável, pode-se adicionar recursos de segurança como proteções de diodo, entre outros.

IX. REFERENCIAS

- [1] INMETRO, “Instrumentos de Medição”. Disponível em:<<http://www.inmetro.gov.br/consumidor/instrumentosMedicao.asp>> Acesso em: 08 de abr. 2018

[2] Richard C Dorf e James A. Svoboda. Introducao aos Circuitos Eletricos, LTC, 2012

[3] TEXAS INSTRUMENTS. **MSP420**: Ultra-Low-

Power Microcontrollers. Disponível em:

<<http://www.ti.com/lit/sd/slab034v/slab034v.pdf>>

Acesso em: 08 de abr. 2018.

[4] PEREIRA, Fábio. **Microcontroladores MSP430**

- Teoria e Prática. s.l. : Érica Ltda, 2005.

[5] TEXAS INSTRUMENTS, “User’s Guide
ADS1118boosterpack”. Disponível em:

<www.ti.com/ADS1118boosterpack> Acesso em 10 de jun. de 2018.

[6] NILSSON, J.W.; RIEDEL, S.A. (2009) Circuitos elétricos, 8ª edição, ISBN 9788576051596. Editora Pearson.

[7] IRWIN, J.D. (2013) Análise Básica de Circuitos para Engenharia, 10ª edição, ISBN 9788521621805.

Editora LTC.

[8] G4KGP, “LC meter notes 1 By G4KGP”

.Disponível em:

<https://dochub.com/writzmatz/RNwNED/lcmeter-notes_1-1> Acesso em 04 de jun. De 2018.

[9] Diogo Garcia, Microcontroladores. Disponível em <<https://github.com/DiogoCaetanoGarcia/Microcontroladores>> Acesso em 02 de jun. De 2018.

[10] DataSheet, HD44780U (LCD-II) . Disponível em <<https://www.sparkfun.com/datasheets/LCD/HD44780.pdf>> Acesso em 02 de jun. De 2018.

Apêndice

Código utilizado na programação do microcontrolador

```
1 //--Medidor_LC--MSP430G2553---compilado com CCS v7-----
2
3 #include <msp430g2553.h> //biblioteca especifica do microcontrolador utilizado
4 #include <stdint.h>
5 #include <stdio.h>
6
7 #define MHZ 16
8 // inicializar display 16x2
9 // define RS high
10#define DR P1OUT = P1OUT | BIT0
11
12// define RS baixo
13#define CR P1OUT = P1OUT & (~BIT0)
14
15// define Read signal R/W = 1 para leitura
16#define READ P1OUT = P1OUT | BIT1
17
18// define Write signal R/W = 0 para escrita
19#define WRITE P1OUT = P1OUT & (~BIT1)
20
21// define Enable alto
22#define ENABLE_HIGH P1OUT = P1OUT | BIT2
23
24// define Enable baixo
25#define ENABLE_LOW P1OUT = P1OUT & (~BIT2)
26
27 void configure_clocks()
28 {
29     // parar watchdog
30     WDTCTL = WDTFW + WDTHOLD;
31     BCSCCTL1 = CALBC1_1MHZ;
32     DCOCTL = CALDCO_1MHZ;
33 }
34
35
36 void delay_us(unsigned int us)
37 {
38     while (us)
39     {
40         // 1 for 1 Mhz set 16 for 16 MHz
41         __delay_cycles(1);
42         us--;
43     }
44 }
45
46
47 void delay_ms(unsigned int ms)
48 {
49     while (ms)
50     {
51         // 1000 for 1MHz and 16000 for 16MHz
52         __delay_cycles(1000);
53         ms--;
54     }
55 }
56
57
58 void data_write(void)
59 {
60     ENABLE_HIGH;
61     delay_ms(5);
62     ENABLE_LOW;
63 }
64
65
66 void send_data(unsigned char data)
67 {
```

```

66 void send_data(unsigned char data)
67 {
68     unsigned char higher_nibble = 0xF0 & (data);
69     unsigned char lower_nibble = 0xF0 & (data << 4);
70
71     delay_us(200);
72
73     WRITE;
74     DR;
75     // enviar higher nibble
76     P1OUT = (P1OUT & 0x0F) | (higher_nibble);
77     data_write();
78
79     // enviar lower nibble
80     P1OUT = (P1OUT & 0x0F) | (lower_nibble);
81     data_write();
82 }
83
84
85 void send_string(char *s)
86 {
87     while(*s)
88     {
89         send_data(*s);
90         s++;
91     }
92 }
93
94
95 void send_command(unsigned char cmd)
96 {
97     unsigned char higher_nibble = 0xF0 & (cmd);
98     unsigned char lower_nibble = 0xF0 & (cmd << 4);
99
100    WRITE;
101    CR;
102    // enviar higher nibble
103    P1OUT = (P1OUT & 0x0F) | (higher_nibble);
104    data_write();
105
106    // enviar lower nibble
107    P1OUT = (P1OUT & 0x0F) | (lower_nibble);
108    data_write();
109 }
110
111
112 void lcd_init(void)
113 {
114     P1DIR |= 0xFF;
115     P1OUT &= 0x00;
116
117     delay_ms(15);
118     send_command(0x33);
119
120     delay_us(200);
121     send_command(0x32);
122
123     delay_us(40);
124     send_command(0x28); // modo 4 bits
125
126     delay_us(40);
127     send_command(0x0E); // limpar a tela
128
129     delay_us(40);
130     send_command(0x01); // cursor do display ativo

```

```

130     send_command(0x01); // cursor do display ativo
131
132     delay_us(40);
133     send_command(0x06); // incrementar cursor
134
135     delay_us(40);
136     send_command(0x80); // linha 1 coluna 1
137 }
138
139
140 int main(void)
141 {
142     configure_clocks();
143
144     lcd_init();
145
146     send_string("Medir Indutancia");
147     send_command(0xC0);
148     send_string("ou Capacitancia");
149     delay_us(40);
150     send_command(0x0E); // clear the screen
151     delay_ms(2000);
152
153 }
154
155 //
156 void eblcd_write(uint8_t d, uint8_t cmd) {
157     if (cmd) P2OUT ^= ~EBLCD_RS; // se cmd = 1 --> P2.3 = VSS
158     else P2OUT |= EBLCD_RS; // else cmd = 0 --> P2.3 = VCC
159
160     cmd = 8;
161     while (cmd--) {
162         if (d & 0x80) {
163             if (d & 0x80) P1OUT |= EBLCD_DATA; // se d[7] = 1 --> P2.7 = VCC (mascara)
164             else P1OUT &= ~EBLCD_DATA; // else d[7] = 0 --> P2.7 = VSS

```

```

Getting Started [main.c] [main.c] [main.c]
165     else P1OUT &= ~EBLCD_DATA; // else d[7] = 0 --> P2.7 = VSS
166     d <<= 1; // d desloca 1 a esquerda
167
168     P1OUT &= ~EBLCD_CLK; // --> P1.5 = VSS
169     P1OUT |= EBLCD_CLK; // --> P1.5 = VCC
170     P1OUT &= ~EBLCD_CLK; // --> P1.5 = VSS
171 } //while
172     _delay_cycles(50); // atraso
173 }
174
175 void eblcd_setcg(uint8_t which, uint8_t *dp) {
176     uint8_t i;
177     which <= 3; // which = deslocar 3 a esquerda
178     which |= 0x40; // which[3] antigo ou which [7] novo = 1
179     for (i=0; i<8; i++) {
180         eblcd_write(which+i, 1); // Envia which para função write
181         eblcd_write(*dp++, 0);
182     } //for
183
184 //
185 void eblcd_clear(uint8_t row) {
186     uint8_t i=0;
187     row *= 0x40; // row * 64 ( ou row << 6 )
188     eblcd_write(0x80+row, 1); // Envia (128 + row) para função write
189     for (i=0; i<16; i++) eblcd_write(' ', 0); // Envia (vazio 16 vezes ) para função write
190     eblcd_write(0x80+row, 1); // Envia (128 + row) para função write
191 }
192
193 //
194 void eblcd_putc(char c) {
195     eblcd_write(c, 0);

```

```

196 }
197
198 /**
199 * puts() is used by printf() to display or send a string.. This function
200 * determines where printf prints to. For this case it sends a string
201 * out over UART, another option could be to display the string on an
202 * LCD display.
203 */
204 void puts(
205
206 void eblcd_puts(char *p, uint8_t row) {
207     if (row<2) eblcd_clear(row);
208     while (*p) eblcd_write(*p++, 0);
209 }
210
211 /**
212 const char hex_map[] = "0123456789abcdef";
213 void eblcd_hex8(uint8_t d) {
214     eblcd_write(hex_map[d>4], 0);
215     eblcd_write(hex_map[d&0x0f], 0);
216 }
217
218 /**
219 void eblcd_hex32(uint32_t h) {
220     eblcd_hex8((h & 0xFF000000) >> 24);
221     eblcd_hex8((h & 0x0FF0000) >> 16);
222     eblcd_hex8((h & 0x0000FF00) >> 8);
223     eblcd_hex8(h & 0xFF);
224 }
225
226 /**
227 void eblcd_hex16(uint16_t h) {
228     eblcd_hex8((h & 0xFF00) >> 8);
229     eblcd_hex8(h & 0xFF);
230 }
231
232 /**
233 void eblcd_dec16(uint16_t d) {
234     uint8_t buf[8], i=0;
235
236     do {
237         buf[i++] = (d%10) + '0';
238         d /= 10;
239     } while (d);
240     while (i) eblcd_putc(buf[--i]);
241 }
242
243 /**
244 void eblcd_dec32(uint32_t d, uint8_t dec) {
245     uint8_t buf[8], i=0;
246
247     do {
248         buf[i++] = (d%10) + '0';
249         d /= 10;
250     } while (d);
251     if (i==dec) buf[i++] = '0'; // leading zero
252     while (i<4) {
253         if (i<=dec)
254             buf[i++] = '0';
255         else
256             buf[i++] = ' ';
257     } //while
258     while (i) {
259         if (dec && i==dec) eblcd_putc('.');
260         eblcd_putc(buf[--i]);
261     } //if
262 }
263

```

```

264 /**
265 void eblcd_setup() {
266     PDIR |= EBLCD_CLK|EBLCD_DATA; // P1.5 e P1.7 configura como saída
267     PDIR |= EBLCD_RS; // P2.3 configura como saída
268
269     static const uint8_t lcd_init[] = { 0x30, 0x30, 0x39, 0x14, 0x56, 0x6d, 0x70, 0x0c, 0x06, 0x01, 0x00, };
270
271     eblcd_write(0x30, 1); //lcd port use
272     __delay_cycles(500000);
273
274     const uint8_t *cmd_ptr = lcd_init;
275     while (*cmd_ptr) eblcd_write(*cmd_ptr++, 1); // LCD sequência inicial
276     __delay_cycles(500000);
277
278 /**
279     00000000...00000000...0000...0000...0000...0000...0000...0000...0000...0000...
280     00000000...00000000...0000...0000...0000...0000...0000...0000...0000...0000...
281     00000000...00000000...0000...0000...0000...0000...0000...0000...0000...0000...
282     00000000...00000000...0000...0000...0000...0000...0000...0000...0000...0000...
283     00000000...00000000...0000...0000...0000...0000...0000...0000...0000...0000...
284     00000000...00000000...0000...0000...0000...0000...0000...0000...0000...0000...
285     00000000...00000000...0000...0000...0000...0000...0000...0000...0000...0000...
286     00000000...00000000...0000...0000...0000...0000...0000...0000...0000...0000...
287     00000000...00000000...0000...0000...0000...0000...0000...0000...0000...0000...
288 /**
289     cmd_ptr = (uint8_t*) hello;
290     while (*cmd_ptr) eblcd_write(*cmd_ptr++, 0); // hello display
291     eblcd_write(0x30|0x40, 1);
292     cmd_ptr = (uint8_t*) hello;
293     while (*cmd_ptr) eblcd_write(*cmd_ptr++, 0);
294
295 /**
296     __delay_cycles(5000000);

```

```

297 ADC1OCTL0 = ADC1OCTL1 = ADC1OAE0 = 0; // separar (ADC1OCTL0 e ADC1OCTL1) reg. de controle e (ADC1OAE0) reg. de habilitação do Comparador a
298
299 P1OUT 4 = (FULL_PIN_MEASURE_PIN); // configura P1.3 e P1.4 --> VSS (nível logico baixo)
300 P1DIR 4 = (PULL_UP|MEASURE_PIN); // direciona P1.3 e P1.4 --> Entrada de dados
301
302 CACTL1 = (P2CA1 | (P2CA3|P2CA2)); // C2A1=(+) --> P1.1 e C2A0=(-) --> P1.6 [P1.1]
303 // CACTL2 = CAF; // NÃO use falso, não escalarmos com elas
304 CAPD = BIT0|BIT1|BIT2|BIT4|BIT6; // desligue os pinos para uma leitura mais limpa
305
306 // P1.3 --> CAOUT
307 PDIR |= BIT3; // P1.3 --> Saída
308 PISEL1 |= BIT3;
309 PISEL2 |= BIT3;
310 PIREL1 |= BIT3;
311 PIREL2 |= BIT3;
312 PIREN 4 = BIT3;
313 CACTL1 = CAON;
314
315 .BIS_SR(GIE);
316
317 // ajustar isso para permitir soma de Cb, ou seja, precisar de atraso para caps maiores
318 // P1OUT 2 = (P1ID_2|NC_2|ID_3|TAC1|TAC2); // P1ID_2, NC_2, P1ID_3, P1DIR 2 precisa de interrupção de overflow
319 CACTL1 |= CAIE; // se o capacitor é inversor, só conectar o pino de saída, caso queira C2
320 .BIS_SR(LPM0_bits + GIE); // agora esperamos que o timer1 transduza (overflow), faça loops nos itens
321 .BIS_SR(GIE);
322 CACTL1 |= CAIE; // falso, não mais contando com pulse comparadores
323 // capture_cnt é só para o número de pulsos dentro do período de overflow do timer2
324 // a frequência do tempo IC seria 16Mhz / 64k * 'capture_cnt'
325 TAC0 |= 0; // done, clean-up turn things off
326 // TAC0 |= 0;
327 // CACTL1 = CACTL1 |= CAED = 0; // NÃO desligue o comparador aqui, a leitura continua será afetada
328 .BIS_SR(GIE);
329
330 /**
331
332
333
334 /**
335 /**
336 // esta função é boa para medir capacitores de maior valor, digamos maior que 1nF
337 // nós configuramos a tensão de referência na entrada não inversora do comparador para 0.55V,
338 // nosso capacitor de destino na entrada e no tempo de inversão
339 // quanto tempo leva para carregar o capacitor desconhecido em 0.55V
340 // nós configuraremos o comparador para disparar uma interrupção timerA quando o 0.55V é atingido
341
342 float measure_high_cap(uint8_t mode, float *esr) {
343
344     .BIS_SR(GIE); // pare as coisas, tempo de configuração
345     P1OUT 4 ~ (FULL_PIN_MEASURE_PIN);
346     P1DIR |= (PULL_UP|MEASURE_PIN);
347     PISEL1 |= ~FULL_PIN;
348     PISEL2 |= ~FULL_PIN;
349
350     // configurar o adc no pino de medida, desta vez para determinar o que quer que um limite esteja conectado
351
352     ADC1OCTL0 = ADC1OSEN_2 + ADC1OON + ADC1OIE;
353     ADC1OCTL1 = INCH_4;
354     ADC1OAE0 |= MEASURE_PIN;
355
356     uint16_t last_adc=1023;
357     uint8_t hit=0;
358     while (1) {
359
360 #ifdef DEBUG
361         eblcd_clear(1); eblcd_dec16(ADC1OENM); eblcd_putc('1'); eblcd_dec16(hit);
362 #endif
363
364         P1DIR 4 = ~MEASURE_PIN; // P1.4 --> entrada [ler adc];

```

```

364     FIDIR |= ~MEASURE_PIN; // P1.4 --> Entrada (ler adeo)
365     ADC10C10 |= ENC + ADC10S0; // amostragem e inicio da conversão
366     while (ADC10C10 & ADC10BUSY); // loop
367
368     if ((ADC10MEM<25 && ADC10MEM<10 && !hit) return 0.0; // não considera presente, retorna
369     if (last_ADC10MEM != ADC10MEM) return 0.0; // não desarmar, retornar
370     last_ADC10MEM = ADC10MEM;
371
372     if (ADC10MEM>=10) { // não flutuantes, considere que um capacitor está conectado
373         P1DIR |= MEASURE_PIN; // P1.4 --> Entrada (lendo capacitância através do pino de medição)
374         FIDIR |= PULSE_PIN; // P1.2 --> Saída (Saída digital para acionar o comparador)
375         P1OUT |= PULSE_PIN; // P1.2 --> VSS (terra)
376         CACTL2 |= CACTL1 | CAPD = 0; // Desconecta entrada do Comparador_A
377
378         if (!hit) ebldc_puts("Desacreditando", 1); // Delay - precisa de algum tempo
379         delay_ms(1000000);
380         hit = 1;
381     } // While
382     P1DIR |= ~PULSE_PIN;
383
384 #endif DEBUG
385     ebldc_clear(0); ebldc_deci6(ADC10MEM); ebldc_putc('<');
386 } //endif
387
388 float cx = 1000.0;
389 uint_t charge_pin = PULL_PIN;
390
391 while (charge_pin) { // nós gostaríamos de obter os valores do capacitor ist;
392 // -----DESENLAVAR o CONVERSOR e ATIVAR o COMPARADOR-----
393
394     ADC10C10 = ADC10C11 = ADC10C10 = 0; // Desativa ADC
395     CACTL2 = P2CA2; // P1.4 --> CA4 (-)
396     CACTL1 = CAREF_2|CAREF_1|CAIES|CAON; // 0.55V ligado (+) e liga o Comparador
397
398     CACTL2 = P2CA2;
399     CACTL1 = CAREF_2|CAREF_1|CAIES|CAON; // P1.4 --> CA4 (-) // 0.55V ligado (+) e liga o Comparador
400
401     CAPD = MEASURE_PIN; // CARREGAR, capturar tempo para chegar 0.55V
402
403     ov_cnt = 0; // CARREGAR, capturar tempo para chegar 0.55V
404     P1OUT |= charge_pin; // P1.3 --> VCC
405     FIDIR |= charge_pin; // P1.3 --> Saída
406
407 // setup e start timer, o comparador irá disparar o temporizador quando o cap carregar até 0.55V
408 // note também que, para os temporizadores de grandes capitalizações, o fluxo excedente é muitas vezes
409 // não garantido isso via sinalizadores de interrupção de saída de temporizadores e o usaremos para o cálculo final
410
411     TA0TCL = TA0SEL_1|MC_2|ID_2|TACLR|TAIE; // smclk, cont. div4 need overflow interrupt
412     TA0CCR1 = CM_2|CCIS_1|SCS_1|CAP; // falling edge, CCIXB (i.e. comparador)
413     _BIC_SR(GIE);
414
415     uint8_t over_range = 0;
416 // cada saída overflow é como 2μF, vamos sair em 10000μF, não posso esperar para sempre
417 // idealmente poderíamos usar outro pino para alterar a corrente de fornecimento para impactar o tempo da amostra
418
419     while (!TA0CCR1&CIFG0) {
420         if (ov_cnt>50) { // quando o temporizador atingir 50 unidades de tempo
421             over_range = 1; // saímos
422             break;
423         }
424     } //if
425
426     CAPD = CACTL2 = CACTL1 = 0; // desliga comparador
427     TA0TCL = TA0CCR1 = 0; // desliga timer
428     _BIC_SR(GIE); // desliga as interrupções
429
430
431 #ifndef DEBUG
432
433     ebldc_clear(0); ebldc_deci6(TA0CCR1); ebldc_putc('*');
434     ebldc_hex8(ov_cnt); ebldc_putc('=');
435     if (ov_cnt) {
436         ebldc_clear(0); ebldc_deci6(TA0CCR1); ebldc_putc('*');
437         ebldc_hex16(ov_cnt); ebldc_putc('=');
438     } //if
439     //while (1) { asm("nop"); }
440
441 #endif
442
443     if (over_range) {
444         if (charge_pin == PULL_PIN) // estamos em faixa baixa, tenta a alta
445             charge_pin = PULSE_PIN;
446         else
447             charge_pin = 0;
448     } //if
449     else {
450         cx = (float) TA0CCR1; // violações do comparador de tempo 0.55V
451         while (ov_cnt--) cx += 65536.0; // mais cada estouro (overflow) adiciona 2^16 units
452         cx *= 285.83; // meu valor calibrado usando um inf 1t
453         if (charge_pin == PULSE_PIN) // nós estamos na faixa alta
454             cx *= (47000.0/100.0); // 100cm instead of 47k
455         charge_pin = 0; // não há mais ensaios
456     } //else
457
458 // Fazendo ESR
459 // tenta esgotar a carga acumulada anteriormente via aterrramento tanto medir quanto pulsar
460
461     P1OUT |= ~MEASURE_PIN|PULSE_PIN|FULL_PIN;
462     P1DIR |= (MEASURE_PIN|PULSE_PIN|FULL_PIN);
463

```

```
[1] Getting Started [2] main.c [3] mainmc [4] mainmc.c
196 //ADC10CTL0 ~= ADC10ON;
197 adc |= ADC10MEM;
198 //__delay_cycles(32000);           // 1khz for me, could have been 1Khz, or 100Khz
199 //__delay_cycles(16000);          // 1khz
200 //ADC10CTL0 ~= ADC10ON;
201 //while(1)
202 adc >>= 6;                   // média fora das amostras
203 // converter unidades adc em mVs, 10bit adc em referência de 1.5V, mais ainda 4x em amostras
204
205 float mv = ((float)adc) * 1500 / 1024.0 / 4.0;
206
207 // obter ESR via lei de ohm law V=IR,
208 // a corrente de pulso é (3.6V - pulse pin VDrop) / (R (100ohm used) + pin resistance)
209 // o dividimos nossa tensão de leitura com a corrente de pulso para obter ESR
210
211 // calibração w/ 1, 2.2, 7.5 ohm 1% resistors
212
213 mv = 140.0 / ((3000.0/mv) - 1.0);
214 mv *= 100.0;                  // we want 0.01 ohm units
215 if (mv > 9999) mv = 9999;
216
217 *esr = mv;
218
219 ADC10CTL0 = ADC10CTL1 = ADC10AE0 = 0;
220 PDIR |= ~FULP_PIN;
221
222 return (cx/1000.0);
223
224 }
225
226 int main(void) {
227     WDTCTL = WDTPW + WDTHOLD;
228     BCSCTL1 = CALBC1_16MHZ;
229     DCOCTL = CALDCO_16MHZ;
230 }
```

```

530
531     lcd_setup();
532     ebLCD_clear(0);
533     ebLCD_clear(1);
534
535     uint8_t cm=';
536     uint8_t cnt=0, wait=0, mode=9, last_open=0, open=0;
537
538     uint16_t f1=0, f2=0, fs=0, last_fs=0;
539
540     while (1) {
541         if (c) {
542             switch (c) {
543                 case '=':           // Medir c
544                     {
545                         //__delay_cycles(MHZ*1000);
546                         uint8_t x2 = 0;
547                         float err = 0.0;
548                         if (!f1) // Se (f1 == 0);
549                             f1 = capture_pulses(); // f1 recebe valor da função capture_pulses();
550
551                         if (f1 > 20) { // Se o valor recebido (f1) da função for maior que 20;
552                             if (f1 > 0x8000) { // Se a f1 for maior ou igual 32768 (decimal);
553                                 f1 = 0; // Zera F1;
554                                 ebLCD_puts("selecionar capacitor baixa ", 0); // Envia mensagem LCD;
555                             }
556                         }
557                     }
558                     else {
559                         f1 = 0; // Zera F1;
560                         ebLCD_puts("Escolher Capacitancia", 0); // Envia mensagem LCD;
561                     }
562                 }
563             }
564             if ((f2) { // Se (f2 == 0);

```

```

564     if ((f2 - capture_pulses()) > (f1 - (f1>>3))) {
565         // f2 recebe valor da função capture_pulses();
566         // Se f2 é maior que (f1 - f1/8) { f1 >> 3
567
568         f1 = f2;
569         f2 = 0;
570         ebldc_puts("Press Calibrar ", 0); // Envia mensagem LCD "calibrar";
571         ebldc_clear(1); // LCD
572         ebldc_putc(' '); // LCD
573         ebldc_putc(' '); // LCD
574         ebldc_dec16(wait++); // LCD
575     } //if
576     else {
577         ebldc_puts("Calibrado", 0); // Envia mensagem LCD "calibrado";
578         ebldc_clear(1);
579     } //else
580 } //if
581 else {
582     last_open = open;
583     open = 0;
584     x32 = measure_high_cap(mode, ferror); // x32 recebe valor da função: measure_high_cap();
585     if (x32) { // se x32 é diferente de zero [ if (x32 != 0) ]
586         if (mode != 2) { // se modo é diferente de 2;
587             //ebldc_puts("Capacitance HI", 0);
588             ebldc_puts(" --\|4-- ", 0);
589             mode = 2; // modo recebe 2;
590         } //if
591         last_f3 = 0; // zera last_f3
592         if (x32 > 5000000000000.0) // se x32 maior que 5000000000000.0
593             open = 2;
594     } //if
595     else { // se x32 for igual a zero
596         f3 = capture_pulses(); // f3 recebe valor da função capture_pulses();
597         if (f3 < 20.0) // se f3 menor que 20.0

```

```

598     if (mode != 0) { // se mode diferente de 0:
599         ebldc_puts("-\\1\\1\\1\\1\\1-", 0); // Envia mensagem (custom) LCD;
600         mode = 0;
601     }/if
602     if (mode == 2)
603     last_f3 = 0;
604
605     }/if
606
607     if ((f3>f1-20) && (f3<(f1+20))) i // se f3 maior que (f1 - 20) E f3 menor que (f1 + 20)
608     if (mode != 1) { // se mode diferente de 1:
609         //ebldc_puts("Capacitance", 0);
610         ebldc_puts(" --\\V3-- ", 0); // Envia mensagem (custom) LCD;
611         mode = 1; // mode recebe 1;
612     }/if
613     open = 1; // open recebe 1;
614     last_f3 = 0; // last_f3 recebe 0;
615
616     }/else
617     else {
618         if (mode <= 1) {
619             big fluctuation means trouble
620             if (last_f3) {
621                 if ((f3 > (last_f3-20)) || (f3 > (last_f3+20))) {
622                     open = 2; // mark cut-of-range flag
623                 }/if
624                 last_f3 = f3;
625             }/if
626         }/else
627     }/else
628     }/else
629 }/if
630 //lcd_write(0x80+8, 1); ebldc_dec16(f3/8);
```

```

631 //lcd_write(0x80+8, 1); ebldc_dec16(f3/8);
632 #ifdef DEBUG // verifica se existe a constante (DEFINE DEBUG) existe, se sim executa o código
633     ebldc_clear();
634     ebldc_dec16(f1/8);
635     ebldc_puts("");
636     ebldc_dec16(f2/8);
637     ebldc_puts("");
638     ebldc_dec16(f3/8);
639 #endif
640
641     if (mode <= 2) {
642         if (f3 && mode <= 1) // se f3 diferente de 0 e mode menor ou igual 1:
643             float F1=f1, F2=f2, F3=f3;
644
645         **** Matemática da tanque de ressonância - Diferença das frequências ***
646
647         F1 *= (MHZ*1000000/0x10000/8); // F1 = F1 * 16*10^6 / 65536 / 8
648         F2 *= (MHZ*1000000/0x10000/8);
649         F3 *= (MHZ*1000000/0x10000/8);
650
651         float B1 = F1/F3;
652         float B2 = F1/F2;
653         B1 *= B1; B1 -= 1.0;
654         B2 *= B2; B2 -= 1.0;
655
656         float cx = (B1 / B2) * 1000 * 10; // 0.1pF units
657         float L = 1/(2.0 * 3.14159 * F1);
658         L *= L;
659
660         float lx = (B1 * B2) * L * 1000000000000;
661         lx *= 1000000; // 0.1 nH / unit
662         x32 = mode == 1 ? (uint32_t) cx : (uint32_t) lx;
663     }/if
664
665
666     uint8_t dec=0, scale=0;
667     static const char x_unit[3] = {'H', 'F', 'P' };
668     static const char x_scale[3][3] = {
669         {'n', 'u', 'm' },
670         {'p', 'n', 'u' },
671         {'n', 'u', 'm' },
672     };
673
674     while (x32 > 9999) {
675         x32 /= 10;
676         dec++;
677         if (dec) {
678             dec = 3;
679             scale++;
680         }/if
681     }/while
682
683     if (last_open == 1 && open != 1) {
684         // não quero montar as primeiras leituras como as coisas ainda não estão resolvidas
685         ebldc_puts("Wait", 1);
686     }/if
687     else {
688         if (open) {
689             if (open == 2) {
690                 P1DIR |= PULSE_PIN|MEASURE_PIN; // usar a chance de descarregar grandes capacitores
691                 P1OUT |= ~PULSE_PIN|MEASURE_PIN;
692                 ebldc_puts("Out of Range", 1);
693             }/if
694             else {
695                 // podemos finalizar o comparador agora, nenhum componente anexado
696             }/if
697         // desativá-lo afastará as leituras contínuas no objeto de teste (faixa baixa)
698         // CACTL2 = CACTL1 = CAPD = 0;
699     }/if
700 }
```

```

698 // CACTL2 = CACTL1 = CAPD = 0;
699 ebldc_puts(" ", 1);
700 }/if
701
702 else {
703     ebldc_clear(1);
704     ebldc_dec16(mode);
705     ebldc_puts(scale);
706     ebldc_puts(x_unit[mode]);
707     if (mode==2) { // mostra também essa na capacidade alta da escala
708         lcd_write(0x80+8, 1); ebldc_puts(" -\\(6\\)\\(6\\)- ", 9);
709         lcd_write(0x80+0x8, 1); ebldc_dec2(ex, 2); ebldc_puts('?' );
710     }/if
711 }/else
712 //ebldc_puts(""); ebldc_dec16(f3); // linha não usada
713
714 }/if
715
716 break;
717
718 case 'x': // filter on-off
719     CACTL2 ^= CAP;
720     break;
721
722 }/if
723
724 }/switch
725 c = 0;
726
727 if (!++cnt) {
728     P1DIR &= ~PULSE_PIN; // desconectar
729     c = '='; // mudar de estado para levar a leitura
730 }/if
731 }
```

```

727
728     if (!++cnt) {
729         P1DIR &= ~PULSE_PIN; // desconectar
730         c = '='; // mudar de estado para levar a leitura
731     }/if
732     else {
733         //P1DIR |= PULSE_PIN|MEASURE_PIN;
734         // use chance to discharge large caps // P1OUT &= ~(PULSE_PIN|MEASURE_PIN); //
735     }/else
736     __delay_cycles(50000);
737
738 }/while
739
740
741
742 }
743
744 /**
745 #pragma vector=TIMERO_A1_VECTOR
746 __interrupt void TIMERO_A1_ISR(void) {
747     switch (TАОIV) {
748         case 10:
749             ov_cnt++;
750             __bic_SR_register_on_exit(LPM0_bits);
751             break;
752     }/switch
753 }
754
755 #pragma vector=COMPARATORA_VECTOR
756 __interrupt void COMPARATORA_ISR(void) {
757     capture_cnt++;
758     if (capture_cnt > 0x8000) __bic_SR_register_on_exit(LPM0_bits|GIE);
759 }
```