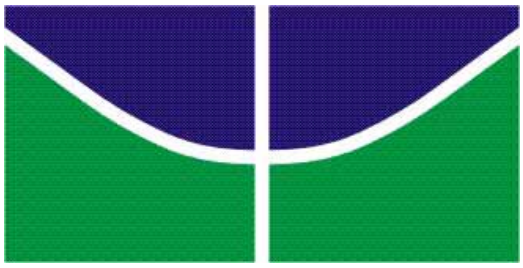
UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA



ELETROCARDIÓGRAFO PORTÁTIL BASEADO NO MICROPROCESSADOR MSP430

RAFAELA GAMELEIRA DA MOTA

ORIENTADOR: ADSON FERREIRA DA ROCHA

PROJETO FINAL DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

BRASÍLIA - DF

1/2012

Universidade de Brasília
Faculdade de Tecnologia
Departamento de Engenharia Elétrica

Eletrocardiógrafo Portátil Baseado no Microprocessador MSP430

Rafaela Gameleira da Mota

Orientador: Adson Ferreira da Rocha

Projeto Final de Graduação em Engenharia Elétrica

Brasília - DF
1/2012

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

ELETROCARDIÓGRAFO PORTÁTIL BASEADO NO MICROPROCESSADOR MSP430

RAFAELA GAMELEIRA DA MOTA

TRABALHO DE CONCLUSÃO DE CURSO SUBMETIDO AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRA ELETRICISTA.

APROVADO POR:

ADSON FERREIRA DA ROCHA, DR., UNB

(ORIENTADOR)

JANAÍNA GONÇALVES GUIMARÃES, DRA., UNB

(EXAMINADORA INTERNA)

LOURDES MATTOS BRASIL, DRA., UNB

(EXAMINADORA INTERNA)

BRASÍLIA, 14 DE SETEMBRO DE 2012

Agradecimentos

A Deus pela vida, pela minha família e pelas oportunidades que me são continuamente oferecidas.

Aos meus pais por sempre estarem disponíveis para os meus problemas cotidianos.

Às minhas irmãs pelo apoio e incentivo.

Às minhas grandes amigas, que de muito perto, ao meu lado na Biblioteca Central, ou de muito longe, por teleconferências, me apoiaram em todos os momentos de dificuldade.

Ao colega da engenharia Luiz Oliveira, cujo auxílio foi indispensável para a conclusão do projeto.

Em especial, ao meu orientador e professor Adson Ferreira da Rocha, pela paciência infinita e apoio, sem os quais o projeto não poderia se concretizar.

Lista de Figuras

Figura 2.1 - Registro das ondas de depolarização e repolarização de uma fibra muscular cardíaca 11

Figura 2.2 - O nodo SA e o sistema de Purkinje do coração 12

Figura 2.3 - Transmissão do impulso cardíaco 13

Figura 2.4 - Forma de onde padrão de um ECG 14

Figura 3.1 – Circuito de captação 15

Figura 3.2 - Diagrama INA 118, Texas Instruments 18

Figura 3.3 – Filtro passa-altas passivo simples 19

Figura 3.4 – Amplificador inversor 20

Figura 3.5 – Filtro passivo passa-baixas 22

Figura 3.6 – Circuito da perna direita 23

Figura 3.7 – Arquitetura interna no MSP420G2553 24

Figura 3.8 – Foto do Lauchpad 24

Figura 4.1 – Foto do sistema completo 32

Figura 4.2 – Onda triangular 33

Figura 4.3 – Sinal do eletrocardiograma 33

SUMÁRIO

1 – Introdução 8

1.1- Motivação do trabalho 8

1.2 - Objetivo do trabalho 8

1.3 - Estruturação do trabalho 9

2 – Funcionamento do coração 10

2.1 - Considerações iniciais 10

2.2 – O coração 10

2.2.1 – Eletrofisiologia da célula cardíaca 10

2.2.2 – Ritmicidade do coração 12

2.3 – O sinal de eletrocardiograma 14

2.4 – Considerações finais 14

3 – O ECG portátil 15

3.1 – Considerações iniciais 15

3.2 – O circuito de captação 15

3.2.1 – O amplificador de instrumentação 16

3.2.2 – O filtro passa-altas 19

3.2.3 – Amplificador inversor 20

3.2.4 – O filtro passa-baixas 21

3.2.5 – Circuito da perna direita 22

3.3 – Parte Digital 23

3.3.1 – O microprocessador MSP430 23

3.3.2 – Conversão analógica-digital 25

3.3.3 – A configuração do LCD 27

3.4 – Considerações finais 30

4 – Resultados obtidos e análise 32

5 – Conclusões e sugestões 34

Referências Bibliográficas **35**

Apêndice - Códigos do sistema usados no Code Composer Studio **36**

A – main.c **36**

B – base.c **38**

C – base.h **41**

D – disp.c **43**

E – disp.h **49**

F – display.c **51**

G – display.h **59**

H – lcd.c **60**

I – lcd.h **66**

J – ploc.c **67**

K – ploc.h **68**

1 – INTRODUÇÃO

1.1- MOTIVAÇÃO DO TRABALHO

O eletrocardiograma, ECG, é um exame conhecido popularmente não invasivo e usado regularmente nos consultórios médicos. Ele é usado tanto para diagnosticar e para acompanhar a evolução de pacientes nos consultórios e hospitais quanto durante cirurgias (GUYTON, 2002).

O ECG portátil apresentado é um primeiro passo para o desenvolvimento de um equipamento portátil com o custo menor do que os que se encontram no mercado, pois foi desenvolvido com componentes de baixo custo. O microprocessador MSP430 do fabricante *Texas Instruments* foi escolhido pela sua acessibilidade e baixo custo. Este é um microprocessador versátil e de baixo consumo de energia, ideal para aplicações portáteis (TEXAS INTRUMENTS, 2000) .

1.2 - OBJETIVO DO TRABALHO

O trabalho tem como objetivo projetar um sistema de captação e visualização do ECG de baixo custo e com baixo consumo de energia.

Esse sistema foi projetado com a intensão de se tornar compatível com outras funções que possam ser incluídas, como aquisição de temperatura e frequência cardíaca no mesmo aparelho.

1.3 - ESTRUTURAÇÃO DO TRABALHO

O trabalho foi divido em 5 capítulos. Após a introdução, no capítulo 2 estão os conceitos necessários para o entendimento do sinal de eletrocardiograma, como ele é gerado e como se propaga até que possa ser medido por meio dos eletrodos.

No capítulo 3 estão as especificações do projeto, assim como componentes usados, todos os estágios pelos quais o sinal passa até a sua amplificação completa. No capítulo 3 também se encontra a descrição da parte digital do projeto, com as configurações principais utilizadas no *firmware* do microprocessador.

No capítulo 4 estão os resultados obtidos com o sistema implementado, fotos do sinal obtido e dos testes e a análise dos resultados obtidos.

No apêndice se encontram os códigos usados no sistema para configurar o microprocessador.

2 – FUNCIONAMENTO DO CORAÇÃO

2.1 - CONSIDERAÇÕES INICIAIS

Este capítulo descreve brevemente o funcionamento do coração e como o mesmo pode ser interpretado pelo registro da variação de tensão provocadas por ele, o ECG.

2.2 – O CORAÇÃO

O coração é um órgão muscular e o centro do sistema circulatório. Ele consiste em duas bombas separadas, uma que bombeia o sangue através dos pulmões, lado direito, e outra, lado esquerdo, que bombeia o sangue através do órgãos periféricos. Cada lado possui duas câmaras, o átrio e o ventrículo. Os átrios funcionam como uma bomba fraca que impulsiona o sangue para o ventrículo, este impulsiona o sangue para os órgão periféricos e para os pulmões (GUYTON, 2002).

O coração tem um sistema de tecidos especiais de excitação e condução que garantem o seu funcionamento rítmico. Esses tecidos possuem estruturas que os permitem a auto-excitação. Esses tecidos produzem e transmitem potenciais através do músculo cardíaco provocando os seus batimentos. Muitas doenças do coração tem base em anormalidades destes sistemas excitador e condutor, especialmente as arritmias cardíacas (GUYTON, 2002).

2.2.1 – ELETROFISIOLOGIA DA CÉLULA CARDÍACA

As células do corpo possuem uma diferença de potencial elétrico entre o interior e o exterior de suas membranas, que é chamado de potencial de membrana (GUYTON, 2002).

Essa diferença de potencial ocorre pois quando a célula está em repouso, a concentração de íons carregados negativamente no seu interior é maior que de íons carregados positivamente no seu exterior. O potencial médio das fibras musculares e neurônios em repouso no interior da célula é de -90 mV. Algumas células, como neurônios e fibras musculares, são capazes de inverter essas concentrações de íons, gerando impulsos que

podem transmitir sinais ao longo das suas membranas. Elas possuem caracterósticas auto-excitáveis, que estão ligadas aos canais presentes nas membranas da célula, por onde os íons atravessam a membrana celular (GUYTON, 2002).

Na Figura 2.1, pode-se observar uma fibra muscular cardíaca no processo de despolarização: o potencial é invertido e o interior da fibra torna-se positivo. Após, o processo de repolarização: o interior da célula volta a ser negativo.

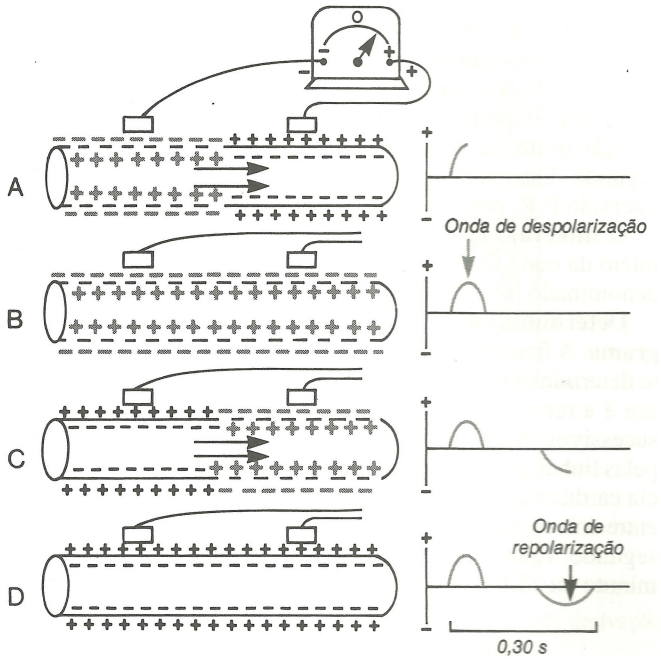


FIGURA 2.1 - REGISTRO DAS ONDAS DE DEPOLARIZAÇÃO E REPOLARIZAÇÃO DE UMA FIBRA MUSCULAR CARDÍACA (GUYTON, 2002)

Quando ocorre a troca do potencial de membrana abruptamente, é gerado um potencial de ação, esse impulso é propagado pelas células excitáveis vizinhas, o que gera um campo elétrico variante no tempo sobre os tecidos próximos desse conjunto de células excitáveis. A variação do campo elétrico pode ser detectada por meio de eletrodos sobre a pele (GUYTON, 2002).

2.2.2 – RITMICIDADE DO CORAÇÃO

O coração de um adulto se contrai em uma frequência de aproximadamente 72 batimentos por segundo. A frequência dos batimentos é controlada por um sistema especial de tecidos que produzem e transmitem o impulso cardíaco através do coração (GUYTON, 2002).

Quando esse impulso passa por determinada parte do coração, ele despolariza as células à sua volta provocando a contração. Esse impulso é gerado pelo nodo sinoatrial, SA, e depois passa pelo coração pelas estrutura mostradas na Figura 2.1 (GUYTON, 2002).

Em um coração normal, o nodo SA funciona como o seu marca-passo primário. Cada estrutura tem uma velocidade de condução, assim o impulso não se propaga com velocidade demasiadamente rápida, o que é benéfico, pois assim, o sangue tem tempo de sair do átrio para o ventrículo antes da contração ventricular (GUYTON, 2002). O tempo, em fração de segundo, que o impulso leva para chegar a cada parte do coração está na Figura 2.2.

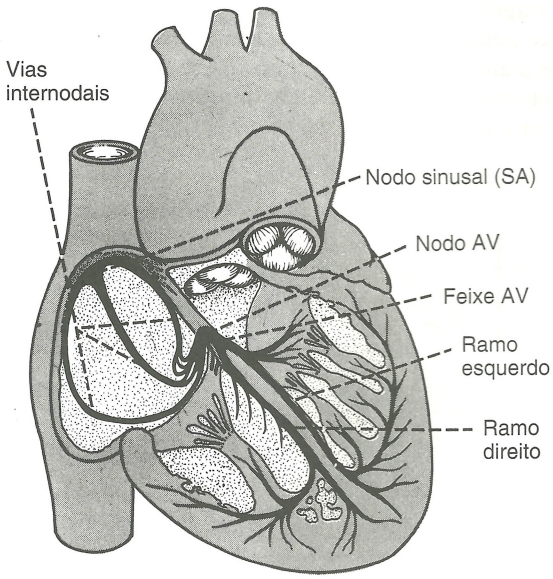


FIGURA 2.2 - O NODO SA E O SISTEMA DE PURKINJE DO CORAÇÃO (GUYTON, 2002)

O impulso percorre os tecidos do coração na ordem seguinte:

- No nodo SA o impulso auto-excitador rítmico normal é gerado;

- As vias intermodais conduzem o impulso do nodo SA para o nodo AV (atrioventricular);
- No nodo AV o impulso dos átrios é retardado antes de passar para os ventrículos;
- O feixe AV conduz o impulso dos átrios aos ventrículos;
- Os feixes esquerdo e direito das fibras de Purkinje conduzem o impulso para todas as partes dos ventrículos (GUYTON, 2002).

A Figura 2.3 mostra a transmissão do impulso cardíaco através do coração com o tempo de aparecimento do impulso em diferentes partes do coração em frações de segundo.

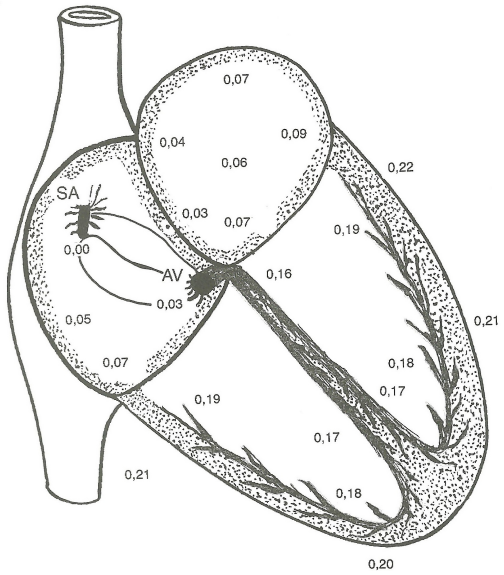


FIGURA 2.3 – TEMPO EM FRAÇÕES DE SEGUNDO DO IMPULSO (GUYTON, 2002)

2.3 – O SINAL DE ELETROCARDIOGRAMA

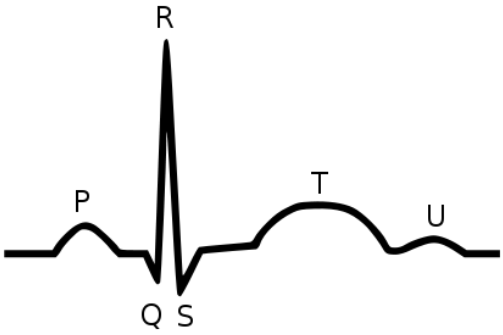


FIGURA 2.4 - FORMA DE ONDE PADRÃO DE UM ECG (GUYTON, 2002)

O ECG é capturado por meio de eletrodos colocados na pele em lugares específicos que registram as variações de tensão em relação ao tempo. As variações resultam da despolarização e repolarização do músculo cardíaco que por sua vez produzem campos elétricos que atingem a superfície do corpo (GUYTON, 2002).

A Figura 2.4 mostra uma representação da forma de onda padrão de um ECG normal.

A primeira onda a aparecer é a onda P, que representa a ativação atrial. Em seguida, o complexo QRS representa a despolarização dos ventrículos. A ondas T é uma onda de repolarização, e representa o seu relaxamento. A onda U não está presente em todos os indivíduos, mas acredita-se que ela também representa a repolarização do miocárdio (GUYTON, 2002).

2.4 – CONSIDERAÇÕES FINAIS

O entendimento básico do funcionamento do coração e da forma como as suas células trabalham é importante para compreender como o sinal de ECG pode ser útil para detectar doenças do sistema circulatório.

As variações detectadas pelo eletrocardiógrafo podem mostrar ao médico onde pode existir anormalidades e assim buscar o tratamento indicado.

3 – O ECG PORTÁTIL

3.1 – CONSIDERAÇÕES INICIAIS

O sistema de ECG portátil proposto no trabalho é composto pela parte analógica: circuito de captação e amplificação do sinal de eletrocardiograma; e a parte digital: conversão AD – analógica-digital - e armazenamento dos dados e transmissão dos resultados para o *display* LCD (*Liquid Crystal Display*).

O circuito de captação foi baseado no amplificador de instrumentação INA118 e a parte digital no microcontrolador MSP430G2552, do fabricante *Texas Instruments*. Este foi configurado por meio do *software Code Composer Studio*. O *display* LCD escolhido foi o S65 LS020, da Siemens (KRANZ, 2005).

3.2 – O CIRCUITO DE CAPTAÇÃO

O circuito de captação do ECG consiste em duas partes: os eletrodos e a amplificação do sinal.

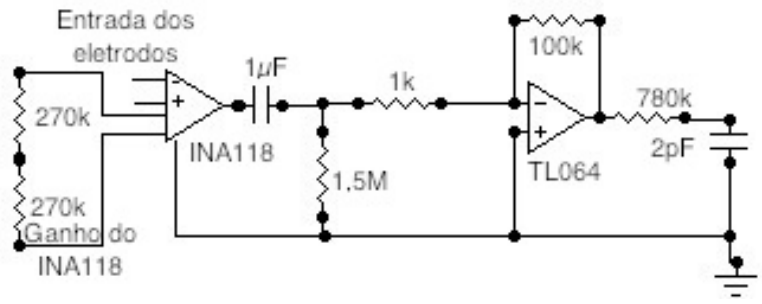


FIGURA 3.1 – CIRCUITO DE CAPTAÇÃO

Após a captação do sinal de ECG, este deve ser amplificado dentro dos parâmetros do conversor AD com o menor ruído possível. Para isso, essa amplificação é feita em 4 etapas:

i) Amplificador de instrumentação (ganho de 10);
ii) Filtro passa-altas;
iii) Amplificador inversor (ganho de 100);
iv) Filtro passa-baixas, um buffer de saída.

O circuito de captação do ECG possui também um circuito chamado circuito da perna direita, *right-leg drive*. O mesmo não tem ligação direta com a amplificação ou com a filtragem do sinal, mas coloca o paciente no mesmo potencial do circuito. Com o mesmo potencial, o erro na entrada do amplificador de instrumentação é reduzido.

3.2.1 – O AMPLIFICADOR DE INSTRUMENTAÇÃO

Como o sinal do ECG tem amplitude em torno de 1mV, é necessário que a amplificação do sinal tenha o menor ruído possível, pois com esse sinal de baixa amplitude, mesmo um ruído pequeno pode ser significativo após a amplificação do sinal. Uma forma de amplificar o sinal de modo a rejeitar satisfatoriamente o ruído é por meio de um amplificador de instrumentação.

A principal característica desse amplificador, importante para o trabalho com sinais de baixa amplitude, é que ele é um amplificador diferencial. O sinal amplificado é a diferença de potencial entre os sinais de entrada. Assim, para dois sinais de baixa amplitude, o ruído externo, que, na teoria, para os dois sinais é o mesmo, é rejeitado. Essa caraterística é chamada de rejeição de modo comum (CMRR-*Commom Mode Rejection*). Na prática o ruído externo pode ser diferente por fatores como o comprimento do fio utilizado, contato diferente entre os eletrodos, etc.

Outras características desse amplificador são: ganho variável, CMRR e impedância de entrada altos, baixa impedância de saída e baixo consumo.

3.2.1.1 – O amplificador de instrumentação INA118 da Texas Instruments

O amplificador de instrumentação escolhido foi o INA118 do fabricante Burr-Brown. Suas principais características extraídas do *datasheet* do fabricante estão presentes na Tabela 3.1 (BURR-BROWN, 1998).

TABELA 3.1 – CARACTERÍSTICAS DO INA118

Característica	**Valor típico**
Consumo de potência	3,5 mW (máximo 10mW)
Tensão de Offset	25 µV (máximo 50 µV)
Corrente de Offset	1nA (máximo 10nA)
Corrente de quiescente	350 µA
Corrente de polarização	5 nA máximo
Ganho ajustável por resistor externo	1 a 10000 V/V
Tensão de alimentação	±1.35 a ±18 V
CMRR	110 dB (no mínimo)
Impedância de entrada	10 GΩ

Um diagrama do INA118 é disponibilizado na Figura 3.2:

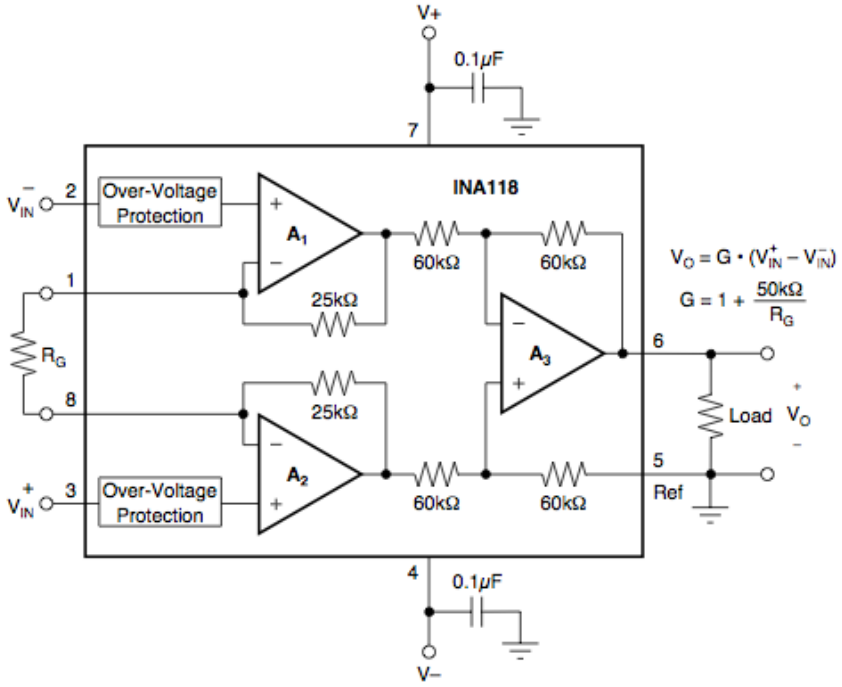


FIGURA 3.2 - DIAGRAMA INA 118, (BURR-BROWN, 1998)

Nesse projeto foi escolhido o ganho de 10 para o amplificador diferencial, de modo que o ruído presente não fosse demasiadamente amplificado. Após a passagem pelo filtro passa-alta, um ganho de 100 será adicionado resultando em um ganho final de 1000 para que o sinal tenha amplitude em torno de 1 V.

O resistor R_G, mostrado na Figura 3.2, é o resistor externo que define o ganho A do amplificador de acordo com a relação dada pela Equação 3.1:

$$A = 1 + \frac{50k\Omega}{R_G}$$

Equação 3.1

Substituindo o ganho de 10 na equação do amplificador operacional, encontra-se o valor de R_G = 5555,55 Ω. O valor comercial disponível mais próximo é de 5,6 kΩ, utilizando

esse valor, o ganho resulta em 9,929 que é suficientemente próximo de 10. No entanto, para o circuito da perna direita devem ser usados dois resistores do mesmo valor em série para se obter o ganho definido. O valor 2,7 kΩ também é um valor comercial. No projeto, então, utilizam-se dois resistores de 2,7kΩ, resultando em uma ganho de 10,25 V/V.

3.2.2 – O FILTRO PASSA-ALTAS

Após o primeiro estágio de amplificação, o sinal de *offset* resultante do ruído pode aparecer amplificado, mesmo que o ganho não seja elevado nesse primeiro estágio. O filtro passa-altas tem a função de retirar a componente DC do sinal de entrada para que essa componente não cause saturação no próximo estágio de amplificação. No circuito de captação do ECG portátil, foi utilizado um filtro passa-altas passivo simples que é composto por um resistor e um capacitor, como na Figura 3.3:

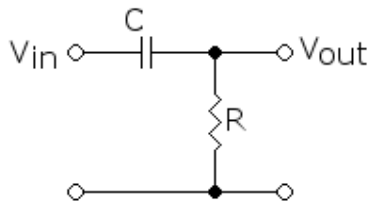


FIGURA 3.3 – FILTRO PASSA-ALTAS PASSIVO SIMPLES

A frequência de corte do filtro passa-altas é dada pela Equação 3.2:

$$f_c = \frac{1}{2\pi RC}$$

Equação 3.2

Onde f_c é a frequência de corte, R o resistor e C o capacitor. Utilizando um resistor de 1,5 MΩ e o capacitor de 1μF obtém-se a f_c= 0,1061 Hz.

3.2.3 – AMPLIFICADOR INVERSOR

O segundo estágio de amplificação é feito por um amplificador inversor visando o ganho de 100, conforme a Equação 3.3.

$$G = \frac{-R_f}{R_{in}}$$

Equação 3.3 – ganho do amplificador inversor

Um esquema do amplificador inversor é exemplificado na Figura 3.4:

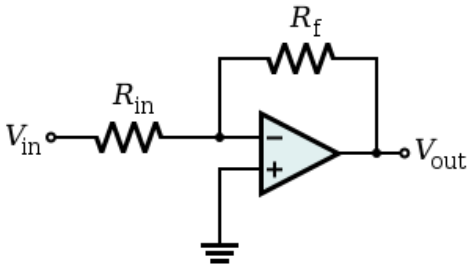


FIGURA 3.4 – AMPLIFICADOR INVERSOR

Para o ganho de -100 foram escolhidos os resistores de R_{in} =1kΩ e R_f =100kΩ.

3.2.3.1 – O amplificador operacional TL064

O amplificador operacional escolhido foi o TL064 no encapsulamento DIP. Ele possui quatro amplificadores operacionais por componente, alta impedância de entrada e baixo consumo de energia. Na Tabela 3.2 são apresentadas as principais características do TL064 (TEXAS INSTRUMENTS, 2004).

TABELA 3.2 – CARACTERÍSTICAS DO TL064

Característica	**Valor típico por amplificador**
Consumo de potência	6 mW (máximo 7,5mW)
Capacidade de fornecimento de corrente	200 μA (máximo 250 μA)
Corrente de *Offset*	30pA (máximo 400pA)
Proteção de curto circuito na saída	Presente
Impedância de entrada	1TΩ
CMRR	86 dB (no mínimo 70dB)

3.2.4 – O FILTRO PASSA-BAIXAS

Após o segundo estágio de amplificação, o sinal tem um ganho de 1000. É necessário filtrar as componentes de alta frequência para reduzir o ruído. O sinal desejado é de baixa frequência, assim, com um filtro passa-baixas com frequência de corte de 100 Hz pode-se reduzir esse ruído das altas frequências. A Figura 3.5 exemplifica um filtro passa-baixas.

A frequência de corte desse filtro é definida pela Equação 3.2.2.

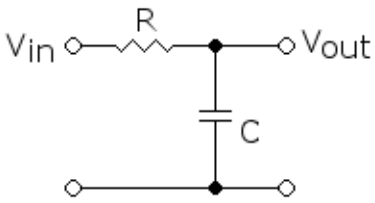


FIGURA 3.5 – FILTRO PASSIVO PASSA-BAIXAS

Os valores de resistor e capacitor escolhidos foram: 780 kΩ e 0,0022 µF. A frequência de corte final é igual a 102,07 Hz, suficientemente próxima de 100 Hz.

3.2.5 – CIRCUITO DA PERNA DIREITA

O circuito da perna direita não tem ligação direta com as fases de amplificação e filtragem do sinal. Esse circuito funciona como um estágio anterior ao tratamento do sinal para diminuir o ruído e também proteger o paciente de uma corrente de escape.

O circuito da perna direita coloca o paciente onde o ECG é medido no mesmo potencial do circuito de captação. No amplificador diferencial, o sinal amplificado é a diferença entre as entradas assim, se o paciente não está no mesmo referencial (aterrado ou no potencial do circuito) a diferença medida tem o potencial de modo comum maior, aumentando assim o ruído do sinal amplificado. Com o paciente no mesmo potencial do circuito o potencial de modo comum é reduzido. Um esquema do circuito da perna direita é exemplificado na Figura 3.6.

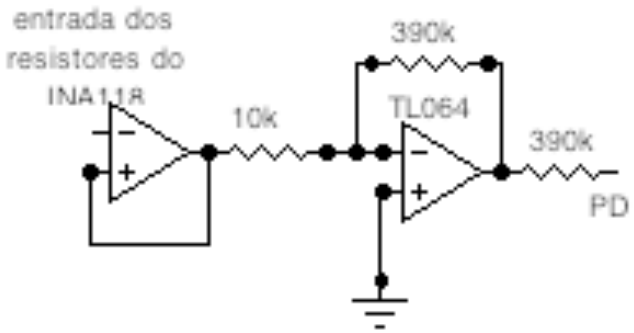


FIGURA 3.6 – CIRCUITO DA PERNA DIREITA

3.3 – PARTE DIGITAL

Após a captação e amplificação do sinal do ECG, este é digitalizado pelo módulo de conversão AD do MSP430. Após a conversão o sinal é armazenado e enviado para o LCD.

3.3.1 – O MICROPROCESSADOR MSP430

O microprocessador MSP430 é um feito pela *Texas Instruments* destinado à aplicações de baixo consumo, com arquitetura Von Neumann e processador de 16 bits. Ele possui vários módulos periféricos que são configurados por meio de software, os módulos usados no ECG portátil foram o conversor ADC10 e o envio de dados pela comunicação serial. A Figura 3.7 é uma representação da arquitetura interna do MSP430G2553.

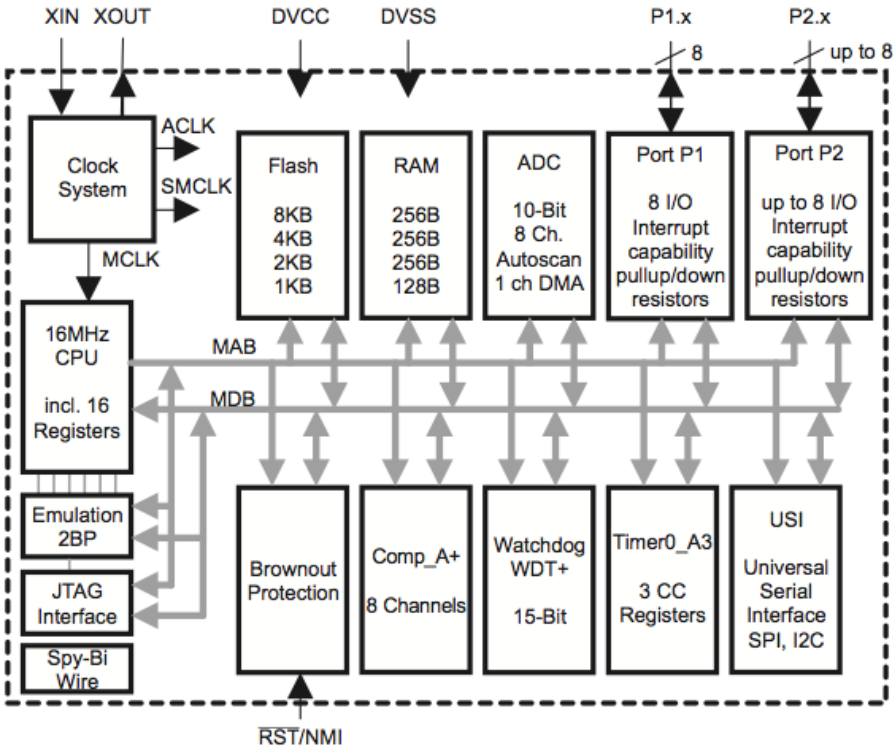


FIGURA 3.7 – ARQUITETURA INTERNA NO MSP430G2553 (TEXAS INSTRUMENTS, 2000)

O microprocessador escolhido no projeto foi o MSP430G2553 em encapsulamento DIP que foi adquirido junto ao *kit Lauchpad*. A Figura 3.8 é uma foto do *kit Launchpad.*

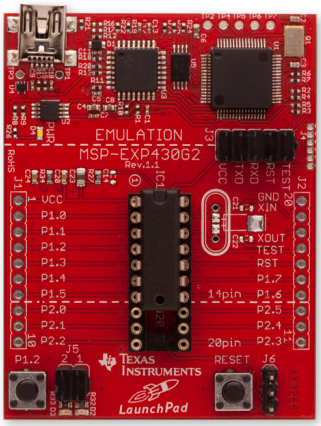


FIGURA 3.8 – FOTO DO LAUNCHPAD

O MSP430 possui um sistema de clock desenvolvido para aplicações alimentadas por bateria.

3.3.2 – CONVERSÃO ANALÓGICA-DIGITAL

O módulo de conversão analógica do MSP430G2553 é o ADC10. O ADC10 é um conversor analógico-digital de 10 bits e alta velocidade. Esse módulo possui gerador de tensão de referência e um controlador de transferência de dados (*Data Transfer Controller* - DTC) (TEXAS INTRUMENTS, 2000).

O DTC permite que as amostras sejam convertidas e armazenadas em qualquer lugar da memória do sem a intervenção da CPU, que permite que o ADC10 opere mesmo em modo de baixo consumo, com o a CPU desligada.

O módulo pode ser configurado por meio de *software* para inúmeras aplicações. Segue algumas caraterísticas do ADC10 (TEXAS INTRUMENTS, 2000):

- Conversão iniciada por software ou pelo *Timer_A;;*
- Máxima taxa de conversão maior que 200K amostras/segundo;
- Conversor monolítico de 12 bits sem perda de códigos;
- *Sample-and-hold* com tempo de amostragem programável;
- A tensão de referência gerada no chip pode ser escolhida por meio do software (1,5 V ou 2,5 V);
- A tensão de referência pode ser escolhida por meio do software para interna ou externa;
- Até 8 canais de entrada externa independentes para a conversão;
- Escolha do clock para a conversão;
- Quatro modos de conversão: canal simples, canal simples repetidas vezes, sequência de canais e sequência de canais repetidas vezes;
- Módulo ADC10 e tensão de referência podem ser desligados separadamente;
- Controle de transferência de dados para armazenamento de resultados de conversão automáticos.

O ADC10 é configurado pelo *software* e possui dois registradores de controle o ADC10CLT0 e o ADC10CTL1. Estes foram configurados de acordo com a Tabela 3.4.

TABELA 3.4 – REGISTRADORES DE CONTROLE DO ADC10

Registrador	Bit	Função
ADC10CTL0	ENC	Habilitar conversão
ADC10CTL0	ADC10SC	Começar conversão
ADC10CTL0	ADC10SHT_2	16 vezes ADC10CLK
ADC10CTL0	MSC	Múltiplas conversões seguidas
ADC10CTL0	ADC10ON	Conversor habilitado
ADC10CTL0	ADC10IE	Interrupções habilitadas
ADC10CTL0	REFON	Gerador de tensão de referencia ligado
ADC10CTL1	INCH_1	Canal de entrada no pino P1.1
ADC10CTL1	CONSEQ_2	Conversão de um canal repetidas vezes
ADC10CTL0	REF2_5V	Tensão de referencia em 2,5V

No registrador ADC10CTL0 foram configurados os bits para habilitar e começar a conversão, o tempo de amostragem, a habilitação para conversões seguidas, a habilitação das interrupções e a referência interna para 2,5 V.

No registrador ADC10CTL1 foram configurados os *bits* para a seleção do canal de entrada e do modo de conversão de um canal repetidas vezes.

O registrador ADC10AE0 é utilizado para escolher o pino de entrada analógica, no caso o pino P1.1

Após a conversão do sinal analógico os resultados do ECG são armazenados no registrador ADC10MEM e na variável *medida* e são chamados na função ploc.c que plota o resultado no display S65 LS020. São armazenados 177 resultados, cerca de 2 segundos do ECG.

3.3.2.1 – Modo de baixo consumo de energia

O microprocessador MSP430 tem diferentes modos de operação, dos quais os quatro seguintes são os mais utilizados: modo ativo, LPM0, LPM3 e LPM4. Os modos de operação de baixo consumo – LPM, *low power mode* – são importantes para a redução da energia necessária para as funções do microprocessador, mas sem perda da qualidade dos dados processados.

No modo ativo, a CPU, todos os *clocks* e todos os módulos periféricos habilitados estão ativos. Esse modo de operação é necessário para iniciar o MSP430 e para que os módulos funcionem, por exemplo, o ADC10.

No projeto, além do modo ativo, o modo de baixo consumo LMP0 foi utilizado. Onde a CPU e o MCLK estão desabilitados, mas os *clocks* SMCLK e o ACLK permanecem ativos.

Na função main.c, onde ocorre a conversão AD, o microcontrolador é colocado em modo de baixo consumo e é acordado por uma interrupção sempre quando há uma conversão.

3.3.3 – A CONFIGURAÇÃO DO LCD

O código do LCD é composto pelos seguintes arquivos:

- disp.c
- disp.h
- lcd.c

- lcd.h

Os arquivos *.h são os headers dos respectivos *.c, ou seja. Neles estão os cabeçalhos para as funções, mas macros e variáveis globais de todas as funções referentes ao LCD.

- lcd.*

No grupo lcd.* encontram-se as funções mais básicas do LCD. É nestes arquivos que estão as rotinas para:

void lcd_init()

Gera a inicialização do LCD. Nela são enviados os comandos básicos que farão o *start up* do LCD, que é dividido em 4 etapas. Todas essas etapas são importantes e devem ser respeitadas para alcançar os devidos estados esperados do LCD

void mswait(uint16_t ms)

Função básica de tempo. Na inicialização do é necessário se dar determinadas pausas para a estabilização dos estados no LCD. Essa função está calibrada para funcionar de acordo com estes tempos. Caso seja utilizado um componente que não consiga operar em 16MHz essa função deve ser revisada.

void pinWrite(unsigned int bit, unsigned char val)

É a função mais básica do LCD. É nela que se encontra o envio bit-a-bit dos dados para o LCD

void writeSPI(unsigned int data)

Esta função usa a pinWrite() para simular uma comunicação SPI - protocolo de comunicação utilizado pelo LCD.

void lcd_wrdata(uint8_t data)

void lcd_wrcmd(uint8_t data

As duas funções estão no mesmo nível de abstração. Elas são responsáveis por enviar um pacote (no caso um número inteiro sem sinal de 8 bits) para a função writeSPI. A diferença entra essas funções é o estado do pino RS, onde quando em nível alto é para envio de comandos e em nível baixo é para envio de data.

void lcd_wrcmd16(uint16_t dat)

void lcd_wrdat16(uint16_t dat)

Devido ao bloco de inicializações ser enviado em grupos de 16 bits, essas funções foram montadas para chamar 2 vezes as respectivas funções de envio de data e comandos de 8 bits.

Por fim, a descrição dos arquivos lcd.* termina com a descrição das macros de cada pino e porta do LCD. São essas macros que possibilitam quem o LCD seja conectado em praticamente qualquer porta do MSP430, tendo como pré-requisito apenas que estejam na mesma porta e que todos os 5 pinos possam assumir as características de entrada e saída.

- disp.*

Este grupo de arquivos contem algumas funções úteis do LCD. São as seguintes:

void lcd_clear(uint16_t color)

Essa função preenche todos os pixes do LCD com uma única cor, dando a impressão de limpar a tela do LCD.

void lcd_set_pixel(uint16_t color,uint8_t x,uint8_t y)

Essa é uma das funções mais básicas deste grupo. Ela é responsável por pintar um pixel da posição (x,y) com a cor "color".

void lcd_retangulo(uint16_t color,uint8_t x,uint8_t dx,uint8_t y,uint8_t dy)

Essa função faz de uso da lcd_set_pixel() para desenhar um retângulo de medidas (x-dx,y-dy) com a cor "color".

void lcd_circulo(uint16_t color,uint8_t xc,uint8_t yc,uint8_t raio)

Essa função foi montada para gerar um circulo de centrado em (xc,yc) com a cor "color". Devido a algumas irregularidades e instabilidades observadas com a função sqrt() - e liberação de memoria de programa, ela esta apenas setando um pixel em (xc,yc).

void put_char(uint8_t x, uint8_t y, char c, uint8_t rot)

Esta função esta comentada por preservar espaço de memoria. Ela contem uma variável com cerca de 10kb contendo toda a tabela ASCII desenhada bit-a-bit, com caracteres com tamanho de 8x14 pixels. O que esta função faz é desenhar 14 sequencias de 8 pixels no LCD. Se for pego um grupo de sequencias, e preenchido um retângulo com 8x14, onde cada bit nível alto representa um pixel escuro, e um nível baixo um pixel claro, poderá ser observado como cada carácter da tabela ASCII foi montado.

Por fim, as macros dos arquivos disp.* possuem as cores do LCD representadas em 12bits, as dimensões do LCD e a sua marca.

Para o uso da biblioteca, basta manter os arquivos disp.c, disp.h, lcd.c, lcd.h no mesmo diretório e incluir apenas o arquivo disp.h no projeto. É importamte observar que nos arquivos *.h esta incluído a biblioteca para o modelo MSP430G2553.

3.4 – CONSIDERAÇÕES FINAIS

O sistema foi dividido em duas partes principais: a parte analógica e a parte digital.

Na parte analógica a maior dificuldade é a remoção do ruído do sinal de ECG captado, assim foram escolhidos 2 filtros e dois estágios de amplificação, além da escolha do amplificador diferencial.

Na parte digital foi escolhido um microprocessador que apresentasse o conversor A/D e memória suficiente para o registro do sinal a ser plotado. A escolha do MSP430 também foi baseada no seu baixo consumo de energia e fácil acesso para estudantes, por ter baixo custo e por possuir muitos guias para quem o adquire.

4 – RESULTADOS OBTIDOS E ANÁLISE

O sistema foi montado e testado no laboratório 1 do SG-11 no *campus* Darcy Ribeiro. Primeiramente foi testado o sistema em tempo real, ou seja, o sinal amostrado é convertido e logo após a conversão esse é plotado no *display*. No entanto, com uma onda senoidal e com uma onda triangular de frequência 1 Hz e amplitude pico a pico de 1 V, a função ploc.c não é suficientemente rápida para a frequência de 1 Hz. Assim o sinal não fica claro, pois a função ploc.c utliza muitos pulsos do *clock*, de forma que a taxa de amostragem fica menor e o sinal menos fiel ao sinal real.

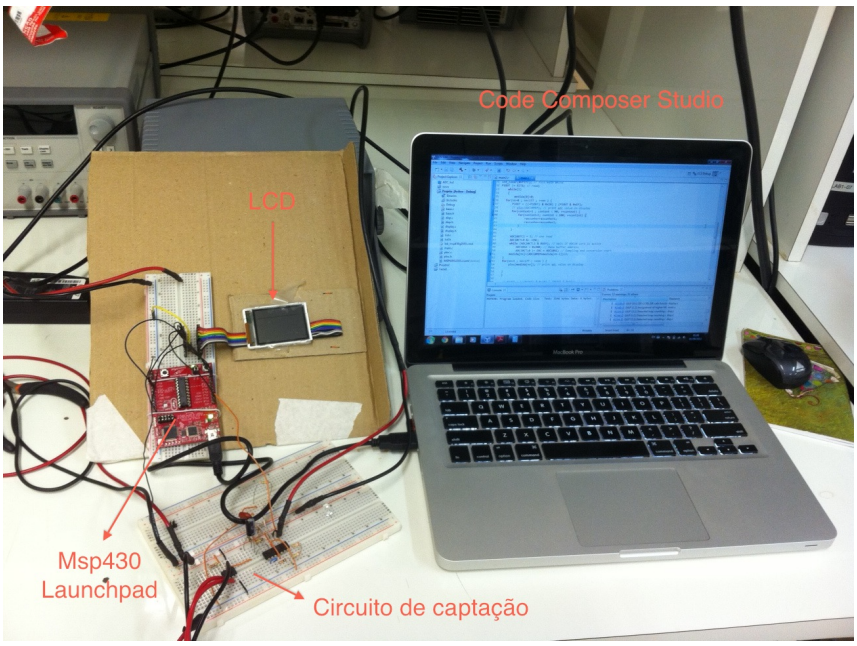


FIGURA 4.1 – FOTO DO SISTEMA COMPLETO

O sistema foi então configurado para capturar o sinal, convertê-lo, armazená-lo e depois plotar os dados convertidos no LCD (código fonte final no apêndice). Primeiramente foi realizado um teste com uma onda triangular gerada pelo gerador de funções com frequência de 1 Hz, Figura 4.1. O computador utilizado para a a configuração do microprocessador foi um *laptop* MacBookPro7,1; com processador de 2,4 GHz Intel Core 2 Duo, com sistema operacional Windows 7.

FIGURA 4.2 – ONDA TRIANGULAR

Este teste mostrou que o sistema faz a conversão e plota o gráfico com sucesso.

O próximo teste foi realizado com o sinal do eletrocardiograma capturado pelo circuito de aquisição implementado no *protoboard*, Figura 4.3.

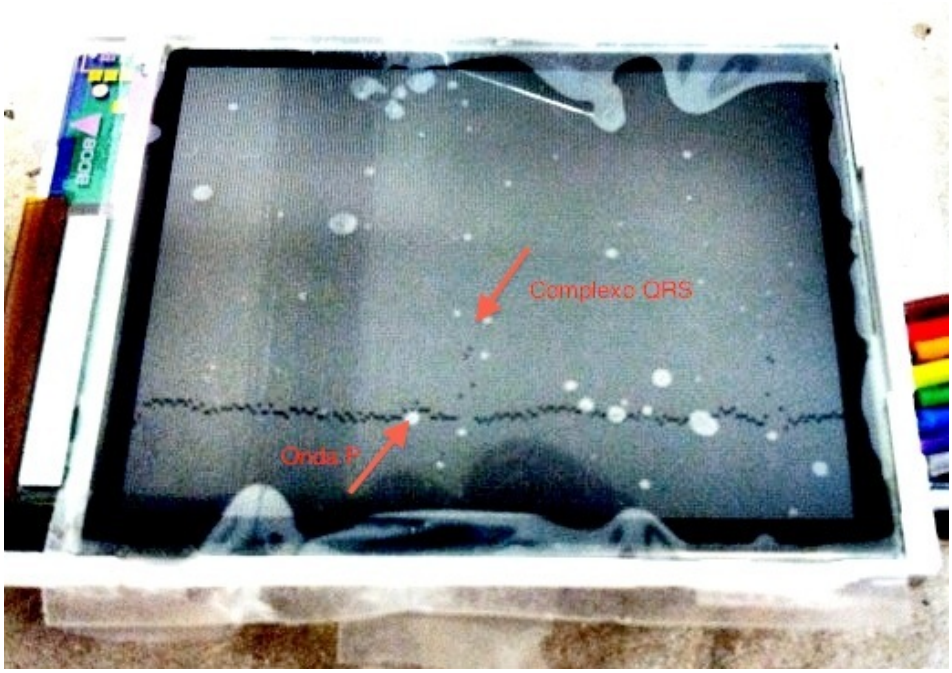


FIGURA 4.3 – SINAL DO ELETROCARDIOGRAMA

Na Figura 4.3, a onda P e o complexo QRS são identificados com clareza, porém as ondas T e U são de menor amplitude e confundem-se com o ruído presente no sinal.

5 – CONCLUSÕES E SUGESTÕES

O projeto acendeu o estudo da fisiologia do coração para que fosse possível o entendimento do sinal do ECG e foram utilizados os conhecimentos do curso de Engenharia Elétrica aplicada à Engenharia Biomédica e o tratamento de sinais bioelétricos. As principais áreas abordadas foram a eletrônica e o uso de microprocessadores.

O projeto consistiu em um sistema de aquisição do sinal do ECG e sua visualização com a finalidade de servir a pacientes que estão fora do ambiente hospitalar.

O sistema funcionou de acordo com o esperado, podemos observar claramente que o sinal obtido foi amplificado corretamente, sem a presença excessiva de ruído e o sinal no display é claramente um sinal de ECG.

Em relação ao custo do sistema obteve grande sucesso, uma vez que o custo total do mesmo é abaixo de sessenta reais. O baixo custo do sistema permite que o seu uso seja mais difundido. Seja no uso doméstico, quanto por médicos que visitam comunidades carentes e aldeias indígenas.

Possíveis melhorias para projetos futuro são: desenvolver o circuito de aquisição com o display e a sua alimentação em uma placa de circuito impresso, o que já viabilizaria a criação de um protótipo e uma significativa redução no ruído do sinal; a melhora do software para a impressão no display também possibilitaria o uso significativo do sistema, para visualizar do sinal do ECG com mais precisão, a fim de detectar mudanças no seu padrão que indiquem a condição do paciente.

REFERÊNCIAS BIBLIOGRÁFICAS

DAVIES, J. H. *MSP430 Microcontroller Basics*. Oxford: Newnes, 2008.

GUYTON, A.; HALL, J. *Tratado de Fisiologia Médica*. 10. ed. Rio de Janeiro: Guanabara Koogan, 2002.

BURR-BROWN CORPORATION.*INA118 Precision Low Power, Intrumentation Amplifier.* Estados Unidos da América, 1998.

MELO, D.; BARBOSA, B. *Sistema de Monitoramento Biomédico,* 2003. Graduação (Engenharia de Redes de Comunicação) – Universidade de Brasília, Brasília, DF.

SEDRA, A.; SMITH, K. *Microeletrônica*. 4. ed. São Paulo: Makron Books, 2000.

TEXAS INSTRUMENTS. *MSP430x2xx Family:* User's Guide. Estados Unidos da América, 2000.

TEXAS INSTRUMENTS. TL061, tl061a, tl061b, tl062, tl062a
Tl062b, tl064, tl064a, tl064b low power JFET-input operational amplifiers. Estados Unidos da América, 2004.

APÊNDICE - CÓDIGOS DO SISTEMA USADOS NO CODE COMPOSER STUDIO

A – MAIN.C

```
/*****************************************************************************
***

* main.c * ADC
*****************************************************************************
***/

#include "disp.h"

 #include "ploc.h"

void main (int argc, char *argv[])

 {

        volatile unsigned int medidapro, rascunho,nn,context,contint;

        volatile unsigned int medida[176];


WDTCTL = WDTPW + WDTHOLD; // Desabilita WDT

BCSCTL1 = RSEL0+RSEL1+RSEL2+RSEL3;

DCOCTL = DCO0+DCO1+DCO2;

ADC10CTL1 = CONSEQ_2 + INCH_1 ; // Repetição de um canal, A1

ADC10CTL0 = ADC10SHT_2 + MSC + ADC10ON + ADC10IE + REFON + REF2_5V;; //
ADC10ON, habilitar interrupções

ADC10DTC1 = 0; // espera

ADC10AE0 |= 0x02; // P1.1 ADC escolha da porta de entrada


 P1DIR |= BIT6; // Setar P1.6 como saída

P1DIR |= BIT4; // Setar P1.0 como saida

lcd_init();

 lcd_clear(0xffff); // preencher com branco o LCD

P1OUT |= BIT6; // pronto para a conversão


         while(1)
```

```
        {
                medida[0]=0;
for(nn=0 ; nn<177 ; ++nn ) {
                P1OUT = ((~P1OUT) & 0x10) | (P1OUT & 0xEF); // toggle da porta após a
        conversão
                for(context=1 ; context < 60; ++context) {
                        for(contint=1; contint < 100; ++contint)
                                rascunho=rascunho+1;
                                rascunho=rascunho+1;
                        }
                }
ADC10DTC1 = 1; // one read
                ADC10CTL0 &= ~ENC;
                while (ADC10CTL1 & BUSY); // Esperar ADC10
                        ADC10SA = 0x200; // Data buffer address
                        ADC10CTL0 |= ENC + ADC10SC; //Começar conversão
                medida[nn]=(ADC10MEM+medida[nn-1])/2; // armazenar 177 medidas na variável
medida
        }
        for(nn=1 ; nn<177 ; ++nn ) {
                ploc(medida[nn]); // plotar medida no display

}
       }
// interrupção ADC10
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR (void)
{
P1OUT |= BIT6; // Mostra o quando está no modo ativo
__bic_SR_register_on_exit(CPUOFF); // Clear CPUOFF bit from 0(SR)

 P1OUT &= ~BIT6;
```

```
}
```

B – BASE.C

```
/*****************************************
*                                                base.c                                                *
*****************************************/
#include "base.h"
void delay(unsigned long d) // max = 8388607
{
        BCSCTL1= CALBC1_16MHZ;          //Setar processador para16MHz
        DCOCTL= CALDCO_16MHZ;
        //d*=512;
  while (d--)
   {
     __delay_cycles(16000);//30
   }
}
/*************************************
                        Interrupções
*************************************/
int init_interrupt (unsigned int code)
{
        switch(code)
        {
                case OFF:
                        __disable_interrupt();
                        return 0;
                break;
                case P1_IO:
                        P1DIR &= ~(B1|B2);
                        P1REN |= B1|B2;
                        P1OUT |= B1|B2;
```

```
                P1IES |= B1|B2;   // alto -> baixo é selecionado para IES.x = 1.
                P1IFG &= ~(B1|B2);  // Limpar flag para previnir interrupção imediatemente
seguinte
                P1IE |= B1|B2;    // Habilita interrupção para P1.3
            break;
            case P2_IO:
                P2DIR &= ~(B1|B2);
                P2REN |= B1|B2;
                P2OUT &= ~(B1|B2);
                P2IES |= B1|B2;   // alto -> baixo é selecionado com IES.x = 1.
                P2IFG &= ~(B1|B2);  // Limpar flag para previnir interrupção imediatemente
seguinte
            P2IE |= B1|B2;    // Habilita interrupções para P1.3
            break;
            case TIMER0:
                BCSCTL1= CALBC1_16MHZ;          //Setar mcu para 16MHZ
                DCOCTL= CALDCO_16MHZ;
                TA0CTL |= TASSEL_2 | MC_1 | ID_3 | TAIE | TACLR;    // Reset it, first.
Up to TACCR
                TA0CCTL0 = CCIE;
            break;
            case TIMER1:
                BCSCTL1= CALBC1_16MHZ;          // Setar mcu para 16MHZ
                DCOCTL= CALDCO_16MHZ;
                TA1CTL |= TASSEL_2 | MC_1 | ID_3 | TAIE | TACLR;    // Resetar
primeiro. Até TACCR
                TA1CCTL0 = CCIE;
            break;
            default:
                return -1;
        }
        __enable_interrupt();               // Habilitar interrupção global
```

```
            return 0;
}
// P1_ISR
/*
interrupt(PORT1_VECTOR) P1_ISR(void)
{
   switch(P1IFG&(B1|B2))
   {
      case B1:
         P1IFG &= ~B1;    // Limpar interrupt flag
         P1OUT &= ~(BIT0|BIT6);
         return;
      case B2:
         P1IFG &= ~B2;    // Limpar interrupt flag
         P1OUT &= ~(BIT0|BIT6);
         return;
          default:                    // Limpar interrup flag
         P1IFG = 0
          return;
   }
}
*/
// P2_ISR
/*
interrupt(PORT2_VECTOR) P2_ISR(void)
{
   switch(P2IFG&(B1|B2))
   {
      case B1:
         P2IFG &= ~B1;    // limpar a interrupt flag
//          P2OUT &= ~(BIT0|BIT6);
```

```
                                estado=verde;
            return;
             case B2:
            P2IFG &= ~B2;    // limpar a interrupt flag
//            P2OUT &= ~(BIT0|BIT6);
                                estado=vermelho;
            return;

default:
            P1IFG = 0;    // limpar flag, se mais de uma está habilitada
            return;
    }
}
*/
//void _RESET(void);    // função reset
/*
interrupt(TIMER0_A0_VECTOR) CCR0_ISR(void)
{
//          LED_OUT ^= (LED0 + LED1); //Toggle both LEDs
            TACTL &= ~TAIFG;    // limpar flag
            TA1CTL |= TACLR;     // limpar TAR0
}
*/
```

C – BASE.H

```
/*****************************************
*                                         base.h                                  *
*****************************************/
#ifndef __BASE_MSP430__
#define __BASE_MSP430__

#include<msp430g2553.h>
```

```
#include <msp430.h>
        #ifndef __MSP430G2553__
                #define __MSP430G2553__
        #endif
        #if defined(__GNUC__)  &&  defined(__MSP430__)
                #include <legacymsp430.h>
                #define _enable_interrupt() __enable_interrupt()
        #else
#endif

#define B1                                      BIT0
#define         B2                                      BIT1
#define    LED_OUT          P1OUT

typedef
enum boolean
{
        false,
        true
}bool;

enum Interrupts
//  definir the interrupções
{
        OFF,
        P1_IO,
        P2_IO,
        TIMER0,
        TIMER1,
        AD0
```

```
};
#define delay_ms( d) delay(d)
extern
void delay(unsigned long d);
extern
int init_interrupt (unsigned int code);
#endif
```

D – DISP.C

```
/*
* disp.c
*
#include "disp.h"
uint16_t backcolor=0x0000;    // cor do background
uint16_t textcolor=0xFFFF;
void lcd_clear(uint16_t color)
{
uint16_t i;
lcd_wrcmd16(0xEF90);
lcd_wrcmd16(0x0500);
lcd_wrcmd16(0x0600);
lcd_wrcmd16(0x0700);
 for (i=0; i<DISP_W*DISP_H; i++)
    lcd_wrdat16(color);
}
void lcd_set_pixel(uint16_t color,uint8_t x,uint8_t y)
{
lcd_wrcmd16(0xEF90);
lcd_wrcmd16(0x0500);
lcd_wrcmd16(0x0600 + y);
lcd_wrcmd16(0x0700 + x);
lcd_wrdat16(color);
```

```
}
void lcd_retangulo(uint16_t color,uint8_t x,uint8_t dx,uint8_t y,uint8_t dy)
{
	uint8_t index_x,index_y;
	for(index_x = x;index_x <= x+dx ; index_x++)
		for(index_y = y; index_y <= y+dy; index_y++)
			lcd_set_pixel(color,index_x,index_y);
}
void lcd_circulo(uint16_t color,uint8_t xc,uint8_t yc,uint8_t raio)
{
	uint8_t index_x = xc,index_y = yc;
//	for(index_x = xc;index_x <= (int)fabs(sqrt((raio*raio) - (index_y*index_y)));index_x++)
//	for(index_y = yc;index_y <= (int)fabs(sqrt((raio*raio) - (index_x*index_x)));index_y++)
	lcd_set_pixel(color,index_x,index_y);
}
/*
void put_char(uint8_t x, uint8_t y, char c, uint8_t rot)
{
	// tabela ascii , começando com o carácter branco (32)
// tamanho 8x14
unsigned char ascii_tab[96][14]={
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}, // space (32)
{ 0x00, 0x00, 0x00, 0x18, 0x3c, 0x3c, 0x3c, 0x18, 0x18, 0x00, 0x18, 0x18, 0x00, 0x00}, //!
{ 0x00, 0x66, 0x66, 0x66, 0x24, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}, //"
{ 0x00, 0x00, 0x00, 0x6c, 0x6c, 0xfe, 0x6c, 0x6c, 0x6c, 0xfe, 0x6c, 0x6c, 0x00, 0x00}, //#
{ 0x00, 0x18, 0x18, 0x7c, 0xc6, 0xc2, 0xc0, 0x7c, 0x06, 0x86, 0xc6, 0x7c, 0x18, 0x18}, //$
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0xc2, 0xc6, 0x0c, 0x18, 0x30, 0x66, 0xc6, 0x00, 0x00}, // %
{ 0x00, 0x00, 0x00, 0x38, 0x6c, 0x6c, 0x38, 0x76, 0xdc, 0xcc, 0xcc, 0x76, 0x00, 0x00}, // &
{ 0x00, 0x18, 0x18, 0x18, 0x30, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}, // '
{ 0x00, 0x00, 0x00, 0x0c, 0x18, 0x30, 0x30, 0x30, 0x30, 0x30, 0x18, 0x0c, 0x00, 0x00}, // (
{ 0x00, 0x00, 0x00, 0x30, 0x18, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x18, 0x30, 0x00, 0x00}, // )
```

```
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x66, 0x3c, 0xff, 0x3c, 0x66, 0x00, 0x00, 0x00, 0x00}, // *
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x18, 0x18, 0x7e, 0x18, 0x18, 0x00, 0x00, 0x00, 0x00}, // +
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x18, 0x18, 0x18, 0x30, 0x00}, // ¥
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xfe, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}, // -
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x18, 0x18, 0x00, 0x00}, // .
{ 0x00, 0x00, 0x00, 0x02, 0x06, 0x0c, 0x18, 0x30, 0x60, 0xc0, 0x80, 0x00, 0x00, 0x00}, // /
{ 0x00, 0x00, 0x00, 0x38, 0x6c, 0xc6, 0xc6, 0xd6, 0xc6, 0xc6, 0x6c, 0x38, 0x00, 0x00}, // 0 (48-32)
{ 0x00, 0x00, 0x00, 0x18, 0x38, 0x78, 0x18, 0x18, 0x18, 0x18, 0x18, 0x7e, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x7c, 0xc6, 0x06, 0x0c, 0x18, 0x30, 0x60, 0xc6, 0xfe, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x7c, 0xc6, 0x06, 0x06, 0x3c, 0x06, 0x06, 0xc6, 0x7c, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x0c, 0x1c, 0x3c, 0x6c, 0xcc, 0xfe, 0x0c, 0x0c, 0x1e, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0xfe, 0xc0, 0xc0, 0xc0, 0xfc, 0x06, 0x06, 0xc6, 0x7c, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x38, 0x60, 0xc0, 0xc0, 0xfc, 0xc6, 0xc6, 0xc6, 0x7c, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0xfe, 0xc6, 0x06, 0x0c, 0x18, 0x30, 0x30, 0x30, 0x30, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x7c, 0xc6, 0xc6, 0xc6, 0x7c, 0xc6, 0xc6, 0xc6, 0x7c, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x7c, 0xc6, 0xc6, 0xc6, 0x7e, 0x06, 0x06, 0x0c, 0x78, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x00, 0x18, 0x18, 0x00, 0x00, 0x00, 0x18, 0x18, 0x00, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x00, 0x18, 0x18, 0x00, 0x00, 0x00, 0x18, 0x18, 0x30, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x0c, 0x18, 0x30, 0x60, 0xc0, 0x60, 0x30, 0x18, 0x0c, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7e, 0x00, 0x00, 0x7e, 0x00, 0x00, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x60, 0x30, 0x18, 0x0c, 0x06, 0x0c, 0x18, 0x30, 0x60, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x7c, 0xc6, 0xc6, 0x0c, 0x18, 0x18, 0x00, 0x18, 0x18, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x7c, 0xc6, 0xc6, 0xde, 0xde, 0xde, 0xdc, 0xc0, 0x7c, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x10, 0x38, 0x6c, 0xc6, 0xc6, 0xfe, 0xc6, 0xc6, 0xc6, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0xfc, 0x66, 0x66, 0x66, 0x7c, 0x66, 0x66, 0x66, 0xfc, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x3c, 0x66, 0xc2, 0xc0, 0xc0, 0xc0, 0xc2, 0x66, 0x3c, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0xf8, 0x6c, 0x66, 0x66, 0x66, 0x66, 0x66, 0x6c, 0xf8, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0xfe, 0x66, 0x62, 0x68, 0x78, 0x68, 0x62, 0x66, 0xfe, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0xfe, 0x66, 0x62, 0x68, 0x78, 0x68, 0x60, 0x60, 0xf0, 0x00, 0x00},

{ 0x00, 0x00, 0x00, 0x3c, 0x66, 0xc2, 0xc0, 0xc0, 0xde, 0xc6, 0x66, 0x3a, 0x00, 0x00},
```

```
{ 0x00, 0x00, 0x00, 0xc6, 0xc6, 0xc6, 0xc6, 0xfe, 0xc6, 0xc6, 0xc6, 0xc6, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x3c, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x3c, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x1e, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0xcc, 0xcc, 0x78, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0xe6, 0x66, 0x6c, 0x6c, 0x78, 0x6c, 0x6c, 0x66, 0xe6, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0xf0, 0x60, 0x60, 0x60, 0x60, 0x60, 0x62, 0x66, 0xfe, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0xc6, 0xee, 0xfe, 0xd6, 0xc6, 0xc6, 0xc6, 0xc6, 0xc6, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0xc6, 0xe6, 0xf6, 0xfe, 0xde, 0xce, 0xc6, 0xc6, 0xc6, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x7c, 0xc6, 0xc6, 0xc6, 0xc6, 0xc6, 0xc6, 0xc6, 0x7c, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0xfc, 0x66, 0x66, 0x66, 0x7c, 0x60, 0x60, 0x60, 0xf0, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x7c, 0xc6, 0xc6, 0xc6, 0xc6, 0xc6, 0xd6, 0xde, 0x7c, 0x0e, 0x00},
{ 0x00, 0x00, 0x00, 0xfc, 0x66, 0x66, 0x66, 0x7c, 0x6c, 0x66, 0x66, 0xe6, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x7c, 0xc6, 0xc6, 0x60, 0x38, 0x0c, 0xc6, 0xc6, 0x7c, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x7e, 0x7e, 0x5a, 0x18, 0x18, 0x18, 0x18, 0x18, 0x3c, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0xc6, 0xc6, 0xc6, 0xc6, 0xc6, 0xc6, 0xc6, 0xc6, 0x7c, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0xc6, 0xc6, 0xc6, 0xc6, 0xc6, 0xc6, 0x6c, 0x38, 0x10, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0xc6, 0xc6, 0xc6, 0xc6, 0xd6, 0xd6, 0xfe, 0x6c, 0x6c, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0xc6, 0xc6, 0xc6, 0x7c, 0x38, 0x7c, 0xc6, 0xc6, 0xc6, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x66, 0x66, 0x66, 0x66, 0x3c, 0x18, 0x18, 0x18, 0x3c, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0xfe, 0xc6, 0x8c, 0x18, 0x30, 0x60, 0xc2, 0xc6, 0xfe, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x3c, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x3c, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x80, 0xc0, 0xe0, 0x70, 0x38, 0x1c, 0x0e, 0x06, 0x02, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x3c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x3c, 0x00, 0x00},
{ 0x10, 0x38, 0x6c, 0xc6, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xff},
{ 0x00, 0x30, 0x18, 0x0c, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x78, 0x0c, 0x7c, 0xcc, 0xcc, 0x76, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0xe0, 0x60, 0x60, 0x78, 0x6c, 0x66, 0x66, 0x66, 0x7c, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7c, 0xc6, 0xc0, 0xc0, 0xc6, 0x7c, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x1c, 0x0c, 0x0c, 0x3c, 0x6c, 0xcc, 0xcc, 0xcc, 0x76, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7c, 0xc6, 0xfe, 0xc0, 0xc6, 0x7c, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x1c, 0x36, 0x32, 0x30, 0x7c, 0x30, 0x30, 0x30, 0x78, 0x00, 0x00},
```

```
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x76, 0xcc, 0xcc, 0xcc, 0x7c, 0x0c, 0xcc, 0x78},
{ 0x00, 0x00, 0x00, 0xe0, 0x60, 0x60, 0x6c, 0x76, 0x66, 0x66, 0x66, 0xe6, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x18, 0x18, 0x00, 0x38, 0x18, 0x18, 0x18, 0x18, 0x3c, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x06, 0x06, 0x00, 0x0e, 0x06, 0x06, 0x06, 0x06, 0x66, 0x66, 0x3c},
{ 0x00, 0x00, 0x00, 0xe0, 0x60, 0x60, 0x66, 0x6c, 0x78, 0x6c, 0x66, 0xe6, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x38, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x3c, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xec, 0xfe, 0xd6, 0xd6, 0xd6, 0xd6, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xdc, 0x66, 0x66, 0x66, 0x66, 0x66, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7c, 0xc6, 0xc6, 0xc6, 0xc6, 0x7c, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xdc, 0x66, 0x66, 0x66, 0x7c, 0x60, 0x60, 0xf0},
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x76, 0xcc, 0xcc, 0xcc, 0x7c, 0x0c, 0x0c, 0x1e},
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xdc, 0x76, 0x66, 0x60, 0x60, 0xf0, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7c, 0xc6, 0x70, 0x1c, 0xc6, 0x7c, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x10, 0x30, 0x30, 0xfc, 0x30, 0x30, 0x30, 0x36, 0x1c, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xcc, 0xcc, 0xcc, 0xcc, 0xcc, 0x76, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xc6, 0xc6, 0xc6, 0x6c, 0x38, 0x10, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xc6, 0xc6, 0xd6, 0xd6, 0xfe, 0x6c, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xc6, 0x6c, 0x38, 0x38, 0x6c, 0xc6, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xc6, 0xc6, 0xc6, 0xc6, 0x7e, 0x06, 0x0c, 0x78},
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xfe, 0xcc, 0x18, 0x30, 0x66, 0xfe, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x0e, 0x18, 0x18, 0x18, 0x70, 0x18, 0x18, 0x18, 0x0e, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x70, 0x18, 0x18, 0x18, 0x0e, 0x18, 0x18, 0x18, 0x70, 0x00, 0x00},
{ 0x00, 0x76, 0xdc, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x10, 0x38, 0x6c, 0xc6, 0xc6, 0xfe, 0x00, 0x00, 0x00},
};
 unsigned char h,ch,p,mask;
 lcd_wrcmd16(0xEF90);
 if (rot)
 {
  lcd_wrcmd16(0x0500);
```

```
 lcd_wrcmd16(0x0800+x);
 lcd_wrcmd16(0x0A00+y);
 lcd_wrcmd16(0x0900+x+CHAR_W-1);
 lcd_wrcmd16(0x0B00+y+CHAR_H-1);
}
else
{
 lcd_wrcmd16(0x0504);
 lcd_wrcmd16(0x0800+y);
 lcd_wrcmd16(0x0A00+x);
 lcd_wrcmd16(0x0900+y+CHAR_H-1);
 lcd_wrcmd16(0x0B00+x+CHAR_W-1);
}
for (h=0; h<CHAR_H; h++) // every column of the character
{
    if (rot)
      ch=ascii_tab[ c-32 ][h];
        else
      ch=ascii_tab[ c-32 ][CHAR_H-h-1];
    mask=0x80;
    for (p=0; p<CHAR_W; p++)  // write the pixels
    {
      if (ch&mask)
      {
        lcd_wrdat16(textcolor);
      }
      else
      {
        lcd_wrdat16(backcolor);
      }
      mask=mask/2;
```

```
      }  // for p
  }
}
*/
```

E – DISP.H

```
/*
 * disp.h
 *
 *  Created on: 07/02/2012
 */
#ifndef DISP_H_
#define DISP_H_
#include "lcd.h"
#include <math.h>
#include <msp430g2553.h>
#define DISP_W 132
#define DISP_H 176
#define CHAR_H 14
#define CHAR_W 8
#define TEXT_COL 16
#define TEXT_ROW 12
extern uint16_t textcolor;  // color of border around text
extern uint16_t backcolor;    // color of background
////////////////////////////// function prototypes ////////////////////////
void lcd_clear(uint16_t color);
//void put_char(uint8_t x, uint8_t y, char c, uint8_t rot);
void lcd_set_pixel(uint16_t color,uint8_t x,uint8_t y);
void lcd_retangulo(uint16_t color,uint8_t x,uint8_t dx,uint8_t y,uint8_t dy);
void lcd_circulo(uint16_t color,uint8_t xc,uint8_t yc,uint8_t raio);
```

```
//#define  PHILLIPS
#define  EPSON
//*****************************************************************
//
//         LCD Dimension Definitions
//
//*****************************************************************
#define ROW_LENGTH  132
#define COL_HEIGHT  132
#define ENDPAGE     132
#define ENDCOL      130
#define DISP_W                  132
#define DISP_H                  176
#define CHAR_H                  14
#define CHAR_W                  8
#define TEXT_COL      16
#define TEXT_ROW     12
//*************************************************
//       12-Bit Color Definitions
//*************************************************
#define WHITE  0xFFFF
#define BLACK  0x0000
#define RED    0xF00
#define GREEN  0x0F0
#define BLUE  0x00F
#define CYAN  0x0FF
#define MAGENTA  0xF0F
#define YELLOW  0xFF0
#define BROWN  0xB22
#define ORANGE  0xFA0
#define PINK  0xF6A
```

```
#endif /* DISP_H_ */
```

F – DISPLAY.C

```
#include <msp430g2553.h>
#include "display.h"
#include "base.h"
char vector_display[] = {'0','0','0','0'};
                                    // vetor que guarda os digitos do display
int index_display = 0;
                                                        // numero que representa o endereco de cada
display
void init_display()
                                                            // Inicia o display
{
        TACCR0 = 500;
                                                                // Seta o tempo do timer
        TACCTL0 = CCIE;
                                                            // Habilita o timer
        TACTL = TASSEL_2 + ID_3 + MC_1 + TACLR;
                                                    // SMCLK, div 8, up mode, clear timer
        _enable_interrupt();
                                                        //
}
void stop_display()
                                                            // Para o display
{       TACCTL0 = !CCIE;        // Enable interrupts for CCR0.
                        // Interrompe o Timer
        P1OUT |= (DIS_A | DIS_B | DIS_C | DIS_D | DIS_E | DIS_F | DIS_G | DIS_PONTO);
        // Desliga todos os displays
}
char itoc(int i) {
                                                        // Converte um numero inteiro para um
caracter que o representa
    switch (i) {
        case 0: return '0';
        case 1: return '1';
        case 2: return '2';
```

```
        case 3: return '3';
        case 4: return '4';
        case 5: return '5';
        case 6: return '6';
        case 7: return '7';
        case 8: return '8';
        case 9: return '9';
        case -1: return 'g';
    }
    return 0;
}
int pow_int(int pow_base , int pow_expoente)
                                    // Funcao de potencia para inteiros
{
        int index_pow;
        int pow_acumulador = 1;
        for(index_pow = 0 ; index_pow < pow_expoente ; index_pow++)
        {
                pow_acumulador *= pow_base;
        }
        return pow_acumulador;
}
void func_numero(char digit)
                                                // Funcao que transforma os digitos em seus
equivalentes de 7 segmentos
{
        switch(digit)
        {
        case '0':
                P1OUT &= ~(DIS_A | DIS_B | DIS_C | DIS_D | DIS_E | DIS_F);
                P1OUT |= DIS_G;
                return;
```

```
case '1':
        P1OUT &= ~(DIS_B | DIS_C);
        P1OUT |= (DIS_A | DIS_D | DIS_E | DIS_F | DIS_G);
        return;
case '2':
        P1OUT &= ~(DIS_A | DIS_B | DIS_D | DIS_E | DIS_G);
        P1OUT |= (DIS_C | DIS_F);
        return;
case '3':
        P1OUT &= ~(DIS_A | DIS_B | DIS_C | DIS_D | DIS_G);
        P1OUT |= (DIS_E | DIS_F);
        return;
case '4':
        P1OUT &= ~(DIS_B | DIS_C | DIS_F | DIS_G);
        P1OUT |= (DIS_A | DIS_D | DIS_E);
        return;
case '5':
        P1OUT &= ~(DIS_A | DIS_C | DIS_D | DIS_F | DIS_G);
        P1OUT |= (DIS_B | DIS_E);
        return;
case '6':
        P1OUT &= ~(DIS_A | DIS_C | DIS_D | DIS_E | DIS_F | DIS_G);
        P1OUT |= (DIS_B);
        return;
case '7':
        P1OUT &= ~(DIS_A | DIS_B | DIS_C);
        P1OUT |= (DIS_D | DIS_E | DIS_F | DIS_G);
        return;
case '8':
        P1OUT &= ~(DIS_A | DIS_B | DIS_C | DIS_D | DIS_E | DIS_F | DIS_G);
        return;
```

```
case '9':
        P1OUT &= ~(DIS_A | DIS_B | DIS_C | DIS_D | DIS_F | DIS_G);
        P1OUT |= (DIS_E);
        return;
case 'A':
        P1OUT &= ~(DIS_A | DIS_B | DIS_C | DIS_E | DIS_F | DIS_G);
        P1OUT |= (DIS_D);
        return;
case 'B':
        P1OUT &= ~(DIS_C | DIS_D | DIS_E | DIS_F | DIS_G);
        P1OUT |= (DIS_A | DIS_B);
        return;
case 'C':
        P1OUT &= ~(DIS_A | DIS_D | DIS_E | DIS_F);
        P1OUT |= (DIS_B | DIS_C | DIS_G);
        return;
case 'D':
        P1OUT &= ~(DIS_B | DIS_C | DIS_D | DIS_E | DIS_G);
        P1OUT |= (DIS_A | DIS_F);
        return;
case 'E':
        P1OUT &= ~(DIS_A | DIS_D | DIS_E | DIS_F | DIS_G);
        P1OUT |= (DIS_B | DIS_C);
        return;
case 'F':
        P1OUT &= ~(DIS_A | DIS_E | DIS_F | DIS_G);
        P1OUT |= (DIS_B | DIS_C | DIS_D);
        return;
case 'G':
        P1OUT &= ~(DIS_A | DIS_B | DIS_C | DIS_D | DIS_F | DIS_G);
        P1OUT |= (DIS_E);
```

```
            return;
case 'H':
            P1OUT &= ~(DIS_B | DIS_C | DIS_E | DIS_F | DIS_G);
            P1OUT |= (DIS_A | DIS_D);
            return;
case 'I':
            P1OUT &= ~(DIS_E | DIS_F);
            P1OUT |= (DIS_A | DIS_B | DIS_C | DIS_D | DIS_G);
            return;
case 'J':
            P1OUT &= ~(DIS_B | DIS_C | DIS_D | DIS_E);
            P1OUT |= (DIS_A | DIS_F | DIS_G);
            return;
case 'L':
            P1OUT &= ~(DIS_D | DIS_E | DIS_F);
            P1OUT |= (DIS_A | DIS_B | DIS_C | DIS_G);
            return;
case 'O':
            P1OUT &= ~(DIS_C | DIS_D | DIS_E | DIS_G);
            P1OUT |= (DIS_A | DIS_B | DIS_F);
            return;
case 'P':
            P1OUT &= ~(DIS_A | DIS_B | DIS_E | DIS_F | DIS_G);
            P1OUT |= (DIS_C | DIS_D);
            return;
case 'Q':
            P1OUT &= ~(DIS_A | DIS_B | DIS_C | DIS_F | DIS_G);
            P1OUT |= (DIS_D | DIS_E);
            return;
case 'U':
            P1OUT &= ~(DIS_B | DIS_C | DIS_D | DIS_E | DIS_F);
```

```
            P1OUT |= (DIS_A | DIS_G);
            return;
case 'a':
            P1OUT &= ~(DIS_A);
            P1OUT |= (DIS_B | DIS_C | DIS_D | DIS_E | DIS_F | DIS_G);
            return;
case 'b':
            P1OUT &= ~(DIS_B);
            P1OUT |= (DIS_A | DIS_C | DIS_D | DIS_E | DIS_F | DIS_G);
            return;
case 'c':
            P1OUT &= ~(DIS_C);
            P1OUT |= (DIS_A | DIS_B | DIS_D | DIS_E | DIS_F | DIS_G);
            return;
case 'd':
            P1OUT &= ~(DIS_D);
            P1OUT |= (DIS_A | DIS_B | DIS_C | DIS_E | DIS_F | DIS_G);
            return;
case 'e':
            P1OUT &= ~(DIS_E);
            P1OUT |= (DIS_A | DIS_B | DIS_C | DIS_D | DIS_F | DIS_G);
            return;
case 'f':
            P1OUT &= ~(DIS_F);
            P1OUT |= (DIS_A | DIS_B | DIS_C | DIS_D | DIS_E | DIS_G);
            return;
case 'g':
            P1OUT &= ~(DIS_G);
            P1OUT |= (DIS_A | DIS_B | DIS_C | DIS_D | DIS_E | DIS_F);
            return;
}
```

```
}
void set_display_digit(char numero,int posicao)
                                        // Funcao externa que possibilta alterar qualquer digito de qualquer
display
{
        vector_display[posicao] = numero;
}
void set_display_number(int numb)
                                                // Funcao externa que possibilita converter um
numero inteiro e escrevelo nos displays
{
        int num = numb;
        int display_number_vector[4] = {0,0,0,0};
        int index_display_number = 0;
                if(!(numb >= -999 && numb <= 9999))
                                                //confere se o numero de entrada esta dentro do
possivel e se nao estiver retorna uma msg de erro
                                        {
                                                set_display_digit('F',3);
                                                set_display_digit('U',2);
                                                set_display_digit('U',1);
                                                set_display_digit('U',0);
                                        }
                else
                        {
                        num = abs(numb);
                        for(index_display_number = 3 ; index_display_number > 0 ;
index_display_number--)         //transforma cada parte do numero para cada um dos displays
                        {
                                if(num >= pow_int(10,index_display_number) )
                                {
                                        display_number_vector[index_display_number] =
num/pow_int(10,index_display_number);
                                        num = num -
display_number_vector[index_display_number]*pow_int(10,index_display_number);
```

```
                        }
                }
                display_number_vector[0] = num;
                if(numb < 0)
                        display_number_vector[3] = -1;
                for(index_display_number = 0;index_display_number < 4 ;
index_display_number++)
        set_display_digit(itoc(display_number_vector[index_display_number]),index_display_numbe
r);
                }
}
void atualizar_display()                        // Atualiza o display a cada periodo do Timer
{
switch(index_display++){
        case 0:
                P2OUT &= ~(DIS_A | DIS_B);
                func_numero(vector_display[0]);
                return;
        case 1:
                P2OUT &= ~(DIS_B);
                P2OUT |= (DIS_A);
                func_numero(vector_display[1]);
                return;
        case 2:
                P2OUT &= ~(DIS_A);
                P2OUT |= (DIS_B);
                func_numero(vector_display[2]);
                return;
        case 3:
                P2OUT |= (DIS_A | DIS_B);
                func_numero(vector_display[3]);
                return;
```

```
        default:
                index_display = 0;
                return;
        }
}
#pragma vector=TIMER0_A0_VECTOR
 __interrupt void CCR0_ISR(void)         // Interrupcao do Timer
{
        atualizar_display();
}
```

G – DISPLAY.H

```
#include <msp430g2553.h>
#include <math.h>
#ifndef DISPLAY_H_
#define DISPLAY_H_
#define abs(x)   fabs(x)
enum DISPLAY
{
        DIS_A=BIT0,
        DIS_B=BIT1,
        DIS_C=BIT2,
        DIS_D=BIT4,
        DIS_E=BIT5,
        DIS_F=BIT6,
        DIS_G=BIT7,
        DIS_PONTO
};
extern
void init_display();
```

```
extern
void stop_display();
extern
void set_display_digit(char numero,int posicao);
extern
void set_display_number(int numb);
extern
char itoc(int i);
void func_numero(char digit);
void atualizar_display();
#endif
```

H – LCD.C

```
#include "lcd.h"
/*
 * lcd.c
 *
 *  Created on: 07/02/2012
 */
void mswait(uint16_t ms)
{
        BCSCTL1= CALBC1_16MHZ;          //Set up the processor to run at a calibrated 16MHz
rate.
        DCOCTL= CALDCO_16MHZ;
        //d*=512;
        uint16_t time = ms;
   while (time--)
   {
      __delay_cycles(16000);//30
   }
}
```

```
void lcd_wrcmd16(uint16_t dat)
{
 lcd_wrcmd((dat>>8));
 lcd_wrcmd(dat);
}
void lcd_wrdat16(uint16_t dat)
{
 lcd_wrdata(dat>>8);
 lcd_wrdata(dat);
}
/*void lcd_init()   //Tentativa de incializar o L2F50
{
        static unsigned char disctl[9] = {0x4C, 0x01, 0x53, 0x00, 0x02, 0xB4, 0xB0, 0x02, 0x00};
        static unsigned char gcp64_0[29] =
{0x11,0x27,0x3C,0x4C,0x5D,0x6C,0x78,0x84,0x90,0x99,0xA2,0xAA,0xB2,0xBA,
0xC0,0xC7,0xCC,0xD2,0xD7,0xDC,0xE0,0xE4,0xE8,0xED,0xF0,0xF4,0xF7,0xFB,
                        0xFE};
        static unsigned char gcp64_1[34] =
                        {0x01,0x03,0x06,0x09,0x0B,0x0E,0x10,0x13,0x15,0x17,0x19,0x1C,0x1E,0x20,
                        0x22,0x24,0x26,0x28,0x2A,0x2C,0x2D,0x2F,0x31,0x33,0x35,0x37,0x39,0x3B,
                        0x3D,0x3F,0x42,0x44,0x47,0x5E};
        static unsigned char gcp16[15] =

{0x13,0x23,0x2D,0x33,0x38,0x3C,0x40,0x43,0x46,0x48,0x4A,0x4C,0x4E,0x50,0x64};
        volatile unsigned int i;
           pinWrite(RESET, 0); // resetar display
           pinWrite(CS, 1);  // CS is high during reset release
           pinWrite(RS, 1);  // RS is set to high
           mswait(10);
           pinWrite(RESET, 1);  //  soltar reset
           mswait(10);
```

```
pinWrite(CS, 0);  // selecionar display
lcd_wrcmd(DATCTL);
lcd_wrdata(0x2A);  // 0x2A=565 mode, 0x0A=666mode, 0x3A=444mode
lcd_cspulse();
lcd_wrcmd(DISCTL);
for (i=0; i<9; i++) {
  lcd_wrdata(disctl[i]);
}
lcd_wrcmd(GCP64);
for (i=0; i<29; i++) {
  lcd_wrdata(gcp64_0[i]);
  lcd_wrdata(0x00);
}
for (i=0; i<34; i++) {
  lcd_wrdata(gcp64_1[i]);
  lcd_wrdata(0x01);
}
lcd_wrcmd(GCP16);
for (i=0; i<15; i++) {
  lcd_wrdata(gcp16[i]);
}
lcd_wrcmd(GSSET);
lcd_wrdata(0x00);
lcd_wrcmd(OSSEL);
lcd_wrdata(0x00);
lcd_wrcmd(SLPOUT);
//  _delay_ms(7);
lcd_wrcmd(SD_CSET);
lcd_wrdata(0x08);
lcd_wrdata(0x01);
lcd_wrdata(0x8B);
```

```
            lcd_wrdata(0x01);
            lcd_wrcmd(SD_PSET);
            lcd_wrdata(0x00);
            lcd_wrdata(0x8F);
            lcd_wrcmd(ASCSET);
            lcd_wrdata(0x00);
            lcd_wrdata(0xAF);
            lcd_wrdata(0xAF);
            lcd_wrdata(0x03);
            lcd_wrcmd(SCSTART);
            lcd_wrdata(0x00);
            pinWrite(RS, 0);
            lcd_wrdata(DISON);
            pinWrite(CS, 1);
}
static void lcd_cspulse() {
   pinWrite(CS, 1);
   pinWrite(CS, 0);
}
*/
void lcd_init()
{
        PORT_DIR |= (RS | DATA | CLK | CS | RESET);
        pinWrite(RESET,0);
        pinWrite(CS,1);
        mswait(100); // esperar
        pinWrite(CS,1);
        pinWrite(RS,1);
        pinWrite(RESET,0);
        mswait(50);
        pinWrite(RESET,1);
```

```
	mswait(50);
	pinWrite(CS,0);
	uint8_t index_init = 0;
	//INIT_0)
	static const uint16_t init_0[] = {0xFDFD,0xFDFD};
	for(index_init = 0 ; index_init < 2 ; index_init++)
		lcd_wrcmd16(init_0[index_init]);
	mswait(50);
			//esperar 50ms
	//INIT_1
	static const uint16_t init_1[] =
{0xEF00,0xEE04,0x1B04,0xFEFE,0xFEFE,0xEF90,0x4A04,0x7F3F,0xEE04,0x4306};
	for(index_init = 0 ; index_init < 10 ; index_init++)
			lcd_wrcmd16(init_1[index_init]);
	mswait(7);
				//esperar 7ms
	//INIT_2
	static const uint16_t init_2[] =
{0xEF90,0x0983,0x0800,0x0BAF,0xA00,0x0500,0x0600,0x0700,0xEF00,0xEE0C,0xEF90,0x0080,
	0xEFB0,0x4902,0xEF00,0x7F01,0xE181,0xE202,0xE276,0xE183};
	for(index_init = 0 ; index_init < 20 ; index_init++)
			lcd_wrcmd16(init_2[index_init]);
	mswait(50);
				// esperar 50ms
	//INIT_3
	static const uint16_t init_3[] = {0x8001,0xEF90,0x0000};
	for(index_init = 0 ; index_init < 3 ; index_init++)
			lcd_wrcmd16(init_3[index_init]);
	pinWrite(CS,1);

}
void lcd_wrdata(uint8_t data)
 {
```

```
        PORT_OUT &=~CS;             //habilitar glcd
        PORT_OUT &= ~RS;
        //cmd(data);
        writeSPI(data);
        //spi_send_data(data);
        PORT_OUT |= CS;             //desabilitar glcd
}
void lcd_wrcmd(uint8_t data)
{
        PORT_OUT &=~CS;             //habilitar glcd
        PORT_OUT |= RS;
       //cmd(data);
        writeSPI(data);
        //spi_send_data(data);
        PORT_OUT |= CS;             //desable glcd
}
void writeSPI(unsigned int data) {
  unsigned int  mask;
  for(mask=0x80; mask != 0; mask = mask >> 1)
  {
    //digitalWrite(CLK, LOW);
    pinWrite(CLK,0);
    if(mask & data)
    {
      //digitalWrite(DATA, HIGH);
      pinWrite(DATA,1);
    }
    else
    {
      //digitalWrite(DATA, LOW);
      pinWrite(DATA,0);
```

```
      }
      //digitalWrite(CLK, HIGH);
      pinWrite(CLK,1);
    }
    //digitalWrite(CLK, LOW);
    pinWrite(CLK,0);
 }
 void pinWrite(unsigned int bit, unsigned char val)
 {
   if (val){
     PORT_OUT |= bit;
   } else {
     PORT_OUT &= ~bit;
   }
 }
```

I – LCD.H

```
/*
 * lcd.h
 *
 */
#ifndef LCD_H_
#define LCD_H_
#include <inttypes.h>
#include <msp430g2553.h>
#define PORT_DIR P2DIR
#define PORT_OUT P2OUT
#define RS        BIT0
#define RESET             BIT1
#define CS        BIT2
```

```
#define CLK     BIT3
#define DATA            BIT4
 void mswait(uint16_t ms);
 void lcd_wrcmd16(uint16_t);
 void lcd_wrdat16(uint16_t);
 void lcd_init();                       // LCD inicializar
 //static void lcd_cspulse();
 //void lcd_clrscr(void);               // LCD desligar
 void lcd_wrdata(uint8_t data);
 void lcd_wrcmd(uint8_t data);
 void writeSPI(unsigned int data);
 void pinWrite(unsigned int bit, unsigned char val);
#endif /* LCD_H_ */
```

J – PLOC.C

```
/*************************************************************************
***
*                   ploc.c                                                *
*************************************************************************
***/
#define COR     0
#define BACK  0xffff
#include "disp.h"
int ploc(unsigned int value)
{
        static int x=0;
        static int x0,x1,x2,x3;
        x3=x2;
        x2=x1;
        x1=x0;
        x0=value/10;
```

```
        #ifdef __GRAFICO_BARRAS__
                lcd_retangulo(BACK,x,0,0,DISP_W);              // limpar último valor
//              lcd_retangulo(COR,x++,0,0,(x0+x1+x2+x3)/4); // colocar último valor e sua cor
                lcd_retangulo(COR,x++,0,0,x0);           // colocar último valor, sem filtro
#else
                lcd_retangulo(BACK,x,0,0,DISP_W);              // limpar último valor
//              lcd_set_pixel(COR,x++,(x0+x1+x2+x3)/4);       // com filtro
                lcd_set_pixel(COR,x++,x0);                                  // sem filtro
#endif

//      antiga = (x0+x1+x2+x3)/4;
        if(x==DISP_H)
        //      fim do display
        {
                x=0;
        }
                return 0;
}
```

K – PLOC.H

```
/*****************************************
*                                       ploc.h                                  *
*****************************************/
#ifndef __PLOC__
#define __PLOC__
extern
int ploc(unsigned int value);
#endif
```