Relatório Final

Gabriel W. S. de Mendonça e Felipe R. Sobrinho

I. Resumo

Problemas de análise de circuitos eletrônicos envolvem o uso do osciloscópio como medida de aferição. Medir e avaliar um circuito é uma tarefa trabalhosa tanto pela complexidade na operação dos equipamentos quanto pela dificuldade do aluno em ter acesso a esse tipo de ferramenta. Desta forma, neste trabalho apresentamos uma forma de atenuar esse percalço criando um osciloscópio portátil, mais acessível e que tenha funções necessárias para análises simples de pequenos sinais - como o são em boa parte dos trabalhos de disciplinas práticas iniciais.

II. Introdução

O leque de soluções é imenso para a problemática. Entretanto, exige-se tempo e investimento para a construção de um produto final ou mesmo um modelo mais elaborado. Em princípio, necessitamos de um protótipo para ancorar os aperfeiçoamentos futuros. Dessa forma, foi construído um modelo simples de osciloscópio para atender as necessidades iniciais - com funções um pouco limitadas em relação a um osciloscópio comum - mas que não atenuam a dificuldade do projeto.

Para atender aos requisitos mínimos de um osciloscópio básico de sinais mistos, são necessárias as seguintes características: amostragem de sinais com resolução mínima de 8 bits, ajuste de escala horizontal e ajuste de escala vertical, e visualização do sinal de forma gráfica em tempo real. Há, dentre inúmeros projetos para esse mesmo fim, as seguintes referências que seguem os critérios estabelecidos a cima:

1) http://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/f2017/jw2299_kp394/jw2299_kp394/jw2299_kp394/index.html: projeto de osciloscópio utilizando PIC32 e o protocolo NTSC de circuito de TV fechada Norte-Americana. Foi a maior base para esse projeto, já que usa diretamente manipulação de registradores, como na MSP430.
2) http://www.msp430launchpad.com/2010/12/njcs-launchscope-launchpad-oscilloscope.html: projeto de osciloscópio usando a MSP430G2, com interface serial conectada a um computador pela LaunchPad. Foi boa referência para termos de captura do sinal e configuração dos timers.
3) urlhttps://guilles.website/2016/03/14/25/ : projeto de captura de sinais do ADC e transmissão por UART para a interface USB da Launchpad MSP430G2. Parte do código de configuração dos timers foi tirada daqui.

III. Desenvolvimento

O hardware é composto de elementos bastante comuns na eletrônica que, aliados à MSP430 - e esta como âmago do projeto -, resultaram num conjunto sobremodo casado. Os componentes utilizados estão elencados abaixo:

- 1 MSP 430 F5529;
- 1 rotary encoder;
- 3 resistores de 100 Ω;
- 3 capacitores de cerâmica de 100 μF;
- 1 Arduino UNO;
- 1 display TFT LCD de 2.8" 320x240 pixels;
- Jumpers e headers genéricos;

A. Rotary encoder

Um rotary encoder incremental possui 2 componentes básicos: dois sinais de onda quadrada e switch. Encontramos a necessidade desse componente nas fases iniciais do projeto. Com ele podemos realizar todas as funções necessárias para operação externa do usuário usando apenas três entradas do μC. Além disso, fornece os dados de modo digital, economizando processamento do μC frente ao que teríamos ao utilizar um potenciômetro.

O pino de switch (SW) é acionado quando pressionado o knob. O modo ocioso de todos os pinos operam em nível alto, *i.e.*, os botões vão para o nível lógico baixo quando pressionado. A função desse botão foi usada essencialmente para a troca nos modos de operação: volts por divisão e tempo por divisão.

Os pinos referentes à rotação do encoder são compostos por dois sinais que estão ficam em estado ocioso quando não utilizados. Ao acionar os pinos através de uma rotação, o primeiro pino, CLK, gera uma onda quadrada de diferença. O segundo sinal, DT, gera também uma onda quadrada, porém defasada em relação ao CLK. Caso o haja um adiantamento de uma onda em relação a outra, houve uma rotação no sentido horário; o contrário, no sentido anti-horário.

Manipulamos os três sinais a partir de uma rotina de interrupção. Há várias fontes de interrupção, o que nos levou a criação de várias condicionais para tratar cada caso:

1) Tratamos o switch primeiro pois ele determinará qual modo será manipulado. esse passo consiste em apenas incrementar uma variável do tipo *volatile unsigned char* até um determinado limite para que haja uma modificação no modo de operação.
2) Em fio, para a outra interrupção referente apenas à fase B, intercalamos em uma mesma rotina de interrupção a partir do P1IV, que gerencia a ordem das interrupções. Para essa condicional, apenas verificamos se um sinal era acionado antes ou depois de outro, relacionando isso a uma variável global que era incrementada ou decrementada segundo a forma de rotação do knob.

É importante mencionar o problema de bounce dos encoders. Por se tratar de um sistema mecânico, está sujeito a trepidações

até sua estabilização. Necessitou se de um debouncing. Optamos por fazê-lo via hardware, pois é bem simples como mostrado pela imagem abaixo:

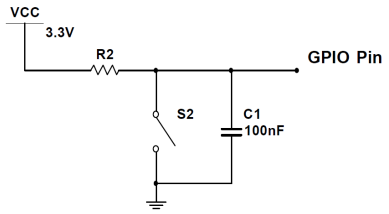

Fig. 1. Circuito para Debouncing.

Dimensionamos R2 como 100Ω, gerando uma constante de tempo $\tau = 100\mu s$, em que $\tau = RC$.

B. Display

Para exibição da forma de onda amostrada pela MSP430, utilizamos um display de TFT LCD 2.8" 320x240 pixels. Devido à necessidade de alta velocidade de exibição dos dados (para que não ocorresse sub-amostragens do sinal) foi necessário modularizar o componente LCD. Por isso quer-se dizer que maior parte do trabalho de exibição de imagem e configuração do display foi isolada do uC principal (MSP430); foi utilizada uma placa Arduino UNO R3, com um uC ATMEGA328P pela fabricante Atmel para controle do display.

O display é controlado por 4 pinos de controle (RESET, CHIP SELECT, COMMAND/DATA, READ/WRITE) e 8 pinos de dados (D0-D7), seguindo o padrão 8080-I da Motorola. Esse display foi escolhido pois uma comunicação em paralelo é notavelmente superior a uma conexão serial para este caso.

O ATMEGA328P foi carregado com as bibliotecas de controle fornecidas pela fabricante do display LCD, em linguagem C++ com variante Arduino. Dessa forma, obtivemos um alto nível de abstração e maior facilidade para configuração do display. Os dados lidos e convertidos pela MSP430 são sequencialmente transmitidos para o ATMEGA328P por comunicação SPI, e por meio de um protocolo pré-definido, a forma de onda e os valores de ms/div (milissegundos por divisão) e mv/div (milivolts por divisão) eram atualizados no display. O uC MSP430, dessa forma, ganhava mais ciclos de processamento para realizar a amostragem e assim nos garante mais precisão.

C. Comunicação SPI

Para a comunicação entre a MSP430 e o ATMEGA328P, foi usada a SPI de 3 pinos. Foi definido um protocolo single-byte para comunicação, por motivos descritos a seguir.

O protocolo SPI de 3 pinos foi implementado por meio dos respectivos periféricos de comunicação de cada uC, usando, em ambos, recepção e transmissão de dados controladas por interrupções. O ATMEGA328P foi configurado como escravo, ligando diretamente seu pino SS (active-low) ao GND, e a MSP430 foi configurada como mestre.

Para o protocolo, utilizamos do fato que a área ativa de exibição é de 317 x 208 pixels. O valor das abscissas não é transmitido pela MSP430, pois o ATMEGA328P foi configurado para receber somente o valor das ordenadas e incrementar automaticamente ao longo do eixo horizontal. Portanto, efetivamente transferimos somente 1 byte para cada ponto que desejamos exibir no display - porém, como sobram (255-208) = 47 bits, temos espaço para 47 comandos. Assim, quando o ATMEGA328P recebe um valor maior que 208, ele interpreta como um comando, que será sucedido de seus parâmetros nos próximos *n* bytes (dependen de como o comando for definido, fica a critério do desenvolvedor). Caso o valor recebido seja menor ou igual a 208, então ele é imediatamente plotado como ordenada e o próximo incremento de tempo é a abscissa.

Para a transmissão dos valores de milivolts/div e segundos/-div, enviamos um valor inteiro de 4 bytes, e um byte inicial adicional de comando para sinalizar que os próximos 4 bytes devem ser concatenados. Como SPI transmite somente 1 byte por vez, utilizamos um tipo de dado *union*, composto por um vetor de 4 bytes de caracteres e um inteiro. Para transmissão, o tipo é lido e transmitido elemento por elemento do vetor de caracteres, vis-a-vis com a recepção, que realiza a mesma operação no ATMEGA328P, concatenando em um mesmo tipo. Ao fim, esse valor é interpretado como um inteiro de 4 bytes.

O mesmo algoritmo pode ser usado para transmissão de outros dados (frequência, valor médio, valor mínimo, etc) com um tamanho qualquer, desde que seja respeitada a condição de haver o máximo de 41 comandos.

D. ADC

A peça fundamental para esse projeto, e causa principal para a escolha da MSP430 F5529 para o projeto, se deu pelo eficiente e altamente manipulável ADC. Para a família F5529, ele vem equipado com uma resolução de até 12 bits, em uma conversão extremamente ágil e manipulável, o que torna-se essencial para o controle dos parâmetros do osciloscópio.

Usamos um timer para realizar o disparo da amostragem, cujo valor de CCR0 é modificado através dos rotary encoders supracitados. Este manipula o tempo de amostragem e, consequentemente, o tempo em que mostramos o valor no display - controlando assim a escala do tempo. Para um tempo de amostragem adequado, realizamos a configuração do ADC para converter os dados amostrados em até 267 ciclos de SMCLK (1 MHz). Há de se considerar os valores estabelecidos pelo fabricante para a carga dos capacitores internos responsáveis pelo armazenamento do valor lido, o qual foi um dos motivos para um tempo de amostragem consideravelmente longo. O período mínimo de amostragem, de acordo com o guia de usuário, foi calculado em 1 us (descrito na seção do ADC12). Com um cálculo rápido, estabelecemos 5 escalas de tempo a partir do valor CCR0, sendo: 19 μs, 25 μs, 50 μs e 100 μs.

O timer foi utilizado em modo up, usando o registrador CCR1. O CCR1 possui uma conexão direta com o ADC12, o que permite o controle imediato dos tempos de amostragem e conversão. Dessa forma, selecionamos a entrada de temporização para o sinal SAMPCON como vinda diretamente do bit OUT do CCR1. Dado que estamos utilizando o SMCLK a 1 MHZ como fonte de clock (sem divisões) para o timer, cada *tick* é equivalente a 1 μs.

E. DMA - Direct Memory Acess

A esse nível de projeto temos a CPU ocupada com diversos cálculos. Para evitar uma sobrecarga e, consequentemente, possíveis erros, decidimos usar o módulo de DMA contido no modelo F5529. Com ela podemos realizar transmissão dos dados em uma média de 4 ciclos de MCLK.

O DMA foi utilizado na forma de transferência única, onde apenas um endereço por vez é enviado. Necessitamos de 2 canais para operação do DMA: um enviava da memória do ADC para um vetor de dados, e outro do ADC para um endereço único.

Observa-se que ambos os sinais utilizam o mesmo endereço de memória, onde a transferência é disparada pela flag de interrupção do ADC. Tivemos então que configurar o parâmetro ROBIN: uma configuração que permite alocar a ordem das transferências, conduzindo-as para que não haja erro ou atrasos. Para manter a integridade dos dados, optamos por não habilitarmos a interrupção através de NMIs(interrupções não mascaráveis).

IV. Conclusão

Para se poder realizar a análise de pequenos sinais, é necessário um equipamento adequado que disponha, de alguma maneira, informações sobre a forma de onda a ser investigada. Tal equipamento é o osciloscópio, para sinais elétricos. Estudantes de engenharia normalmente ficam dependentes de laboratórios e equipamentos caros e pesados para realizar análise de sinais. Como forma de auxiliar no aprendizado e praticidade do meio acadêmico, aqui propusemos um osciloscópio portátil, com as seguintes funções básicas: exibição gráfica da forma de onda sob análise; ajuste de escala de tempo (abscissa); ajuste de escala de tensão (ordenada).

Foram usados dois microcontroladores em conjunto para realização desse projeto: MSP430F5529 e ATMEGA328P, ambos em suas respectivas placas de desenvolvimento (Launchpad e Arduino Uno), comunicando-se por meio da SPI. Para visualização das formas de onda, foi utilizado um display LCD TFT 320x240 pixels de 2.8”, com interface paralela em 8 bits, controlada exclusivamente pelo ATMEGA328P. Para os ajustes de escala horizontal e vertical, foi utilizado um único encoder. Os periféricos utilizados da MSP430 foram o DMA para transferência de dados, ADC12 para amostragem do sinal de entrada, TimerA0 para ajuste de tempo de amostragem, USCI para comunicação SPI, e o PORT1 para interface com encoder.

Obtivemos resultados satisfatórios para o período de tempo estipulado na elaboração do mesmo projeto. Foi possível amostrar sinais de 0 a 2.5 V, com uma frequência máxima de 20 kHz. Há a presença de aliasing caso a frequência de amostragem seja ajustada, por meio do encoder, para menos do dobro da frequência do sinal de entrada, sendo o maior empasse. Há uma série de fatores que ainda podem ser melhorados no projeto, tais como os valores e informações exibidas no display, aumento de frequência de amostragem e resolução, para citar o mínimo. É um excelente projeto como ponto de partida para processamento de sinais.

V. Anexos

** Imagens e vídeos do projeto estão disponíveis na apresentação Apresentacao Osciloscopio Portatil em pptx neste mesmo diretório. **

A. Código MSP430

```
1
2  #include <msp430.h>
3  #include <math.h>
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <string.h>
7
8  #define phasea  BIT5
9  #define phaseb  BIT4
10 #define sw      BIT3
11 #define SIZE    100
12 #define LENGTHX 317
13 #define LENGTHY 208
14
15
16 volatile unsigned char buffer[SIZE] = {0};
17
18 // Variaveis para controle
19 volatile char ymod = 4; // controle vertical
20 // Variavel para sinal do ADC12
21 unsigned char signal = 0;
22
23 // Variaveis para o encoder
24 volatile int adc_conv;
25 volatile int adc_read;
26 volatile int timedivon = 0;
27 volatile int voltsdivon = 0;
28 volatile unsigned int sel_time = 0;
29 volatile unsigned int sel_volts = 0;
30 volatile char mode;
31 volatile unsigned int sel_mode = 0x00;
32 volatile unsigned char disp_val = 0;
33
34 float vrms = 0, vavg=0;
35 int count = 0;
36
37 typedef union _entrada {
38   unsigned int val;
39   unsigned char b[4];
40 } entrada;
41
42 void TIMERA0_CFG();
43 void ADC12_CFG();
44 void ENCODER_CFG();
45 void SPI_CFG();
46 void DMA_CFG();
47 void sendTimeDiv(int timediv);
48 void sendVoltsDiv(int voltsdiv);
49
50 int main(void)
51 {
52
53   WDTCTL = WDTPW | WDTHOLD; // stop watchdog timer
54
55     TIMERA0_CFG();
56     ADC12_CFG();
57     ENCODER_CFG();
58     SPI_CFG();
59    //DMA_CFG();
60
61   __enable_interrupt();
62
63   while(1) {
64   }
65 }
66
67 #pragma vector = ADC12_VECTOR
```

```
68 __interrupt void ADC12_A_ISR(void) {
69     if(timedivon || voltsdivon) return;
70     signal = (ymod*ADC12MEM0*LENGTHY/255)>>2;
71     signal = (signal > LENGTHY) ? 255 : signal;
72     if(UCA0IFG & UCTXIFG) {
73         UCA0TXBUF = signal;
74     }
75 }
76
77 #pragma vector = PORT1_VECTOR
78 __interrupt void sw_push(void){
79     switch (P1IV){
80         case 0x08:                              //Ordenamento dos TAIV para separacao
81                                                 //das funcoes do encoder.
82             P1IFG &= ~0x08;
83             sel_mode++;
84             switch (sel_mode){
85                 case    0x00:
86                     //selecao do tempo//
87                     mode = 0;
88                     break;
89                 case    0x01:
90                     mode= 1;
91                     break;
92                 case    0x02:
93                     mode= 2;
94                     break;
95                 default:
96                     mode= 0;
97                     sel_mode = 0x00;
98                     break;
99             }
100            break;
101
102        case 0x0A: // Caso para a rotacao do encoder
103            P1IFG &= ~0x0A;
104            switch (mode){
105                case 0:
106                    if((P1IN & phasea)){            //phase a e b sao utilizados
107                                                    //para a verificacao
108                        sel_time++;                   //da diferenca de fase.
109                        if(sel_time >= 0x07) sel_time = 0x00;
110                    }
111                    else{
112                        if(sel_time>0)
113                            sel_time--;
114                    }
115                    switch (sel_time){
116                       case 0x00:
117                           TA0CCR0 = 999;
118                           sendTimeDiv(1000);
119                           break;
120                       case 0x01:
121                           TA0CCR0 = 499;
122                           sendTimeDiv(500);
123                           break;
124                       case 0x02:
125                           TA0CCR0 = 249;
126                           sendTimeDiv(250);
127                           break;
128                       case 0x03:
129                           TA0CCR0 = 99;
130                           sendTimeDiv(100);
131                           break;
132                       case 0x04:
133                           TA0CCR0 = 49;
134                           sendTimeDiv(50);
135                           break;
136                       case 0x05:
137                           TA0CCR0 = 24;
138                           sendTimeDiv(25);
139                           break;
140                       case 0x06:
141                           TA0CCR0 = 9;
142                           sendTimeDiv(10);
143                           break;
144                       default:
```

```
145                             break;
146                     }
147                     break;
148                 case 1:
149                     if((P1IN & phasea)){            //phase a e b sao utilizados para a
150                         sel_volts++;                    //verificacao da diferenca de fase.
151                         if(sel_volts >= 0x05) sel_volts = 0x00;
152                     }
153                     else{
154                         if(sel_volts>0)
155                             sel_volts--;
156                     }
157                     switch (sel_volts){
158                        case 0x00:
159                            ymod = 8;
160                            sendVoltsDiv(1);
161                            break;
162                        case 0x01:
163                            ymod = 6;
164                            sendVoltsDiv(2);
165                            break;
166                        case 0x02:
167                            ymod = 4;
168                            sendVoltsDiv(4);
169                            break;
170                        case 0x03:
171                            ymod = 2;
172                            sendVoltsDiv(8);
173                            break;
174                        case 0x04:
175                            ymod = 1;
176                            sendVoltsDiv(16);
177                            break;
178                        default:
179                            break;
180                     }
181                     break;
182                 }
183                 break;
184             default:
185                 break;
186     }
187 }
188
189 void sendTimeDiv(int timediv) {
190     timedivon = 1;
191     UCA0TXBUF = 0xFF; // Envia o comando de modo de tempo
192     while(!(UCA0IFG & UCTXIFG)) {} // Espera terminar de transmitir o comando
193
194     int k=0;
195     entrada tempo;
196     tempo.val = timediv;
197
198     for(k=0; k<=3; k++) {
199         UCA0TXBUF = tempo.b[k];
200         while(!(UCA0IFG & UCTXIFG)) {} // Espera transmitir o byte
201     }
202     timedivon = 0;
203 }
204
205 void sendVoltsDiv(int voltsdiv) {
206     voltsdivon = 1;
207     UCA0TXBUF = 0xFE; // Envia o comando de modo de volt
208     while(!(UCA0IFG & UCTXIFG)) {} // Espera terminar de transmitir o comando
209
210     int k=0;
211     entrada volts;
212     volts.val = voltsdiv;
213
214     for(k=0; k<=3; k++) {
215         UCA0TXBUF = volts.b[k];
216         while(!(UCA0IFG & UCTXIFG)) {} // Espera transmitir o byte
217     }
218     voltsdivon = 0;
219 }
220
221 void SPI_CFG() {
```

```
222     P3SEL |= BIT4 | BIT3; // Enable SPI SOMI and SIMO on pins 3.4, 3.3
223     P2SEL |= BIT7; // Enable SPI Clock output on pin 2.7
224     // Software reset in order to manipulate control registers
225     UCA0CTL1 = UCSWRST;
226     // Configurations for SPI Mode 0
227     // Sample on rising, non-inverted clock polarity, master mode, 3-pin SPI,
228     // synchronous mode
229     UCA0CTL0 = UCCKPH | UCCKPL | UCMST | UCMSB | UCMODE_0 | UCSYNC;
230     // SMCLK as source (1 MHz)
231     UCA0CTL1 |= UCSSEL_2;
232     // SMCLK / 2
233     UCA0BR0 = 0x02;
234     UCA0BR1 = 0x00;
235     // Disable modulation
236     UCA0MCTL = 0;
237     UCA0CTL1 &= ~UCSWRST; // Start SPI state machine
238     // Enable interrupts
239     UCA0TXBUF = 0; // Reset transmit buffer
240     //UCA0IE |= UCTXIE;
241 }
242
243 void TIMERA0_CFG() {
244 //-------------Configuracoes do TIMER0A ------------------//
245     // Seleciona SMCLK como fonte | conta ate CCR0 | limpa o timer
246     TA0CTL = TASSEL_2 | MC_3 | TACLR;
247
248     /*
249      * O timer definira o comprimento div/s, com os seguintes espacamentos
250      * 0.05 s  : CCR0 = 49999
251      * 0.01 s  : CCR0 = 9999
252      * 0.005 s : CCR0 = 4999
253      * 0.001 s : CCR0 = 999
254      * 0.0005 s: CCR0 = 499
255      *
256      * O valor minimo do CCR0 deve ser de 265 para que ocorra uma amostragem correta
257      * no ADC12
258      */
259
260     // Tempo de amostragem: t_sample > (R_S + R_I) * ln(2^(n+1)) * C_I + 800 ns
261     // Para R_S = 10R, R_I = 1k, n=12, C_I = 20pF --> t_sample > 0.984 us ~ 1 us
262     TA0CCR0 = 999;
263
264     // CCR1 OUT --> ADC12SHS_1
265     // Modo de saida Reset/Set.
266     TA0CCTL1 = OUTMOD_7;
267
268 }
269
270 void ADC12_CFG() {
271 //-------------Configuracoes do ADC12_A ------------------//
272     P6SEL |= BIT0; // Selecionar o ADC12_A no pino 6.0
273
274     REFCTL0 &= ~REFMSTR; // Tensao de referencia sera aquela do
275                          // modulo interno do ADC ao invez daquela
276                          // do modulo REF
277
278     ADC12CTL0 &= ~ADC12ENC;
279     // ligar o ADC12 | liga a referencia do ADC | referencia de 2.5 V | 256 ciclos de
280     //clk para amostragem
281     ADC12CTL0 = ADC12ON | ADC12REFON | ADC12REF2_5V | ADC12SHT0_8;
282
283     // fonte da borda de subida do sample e hold | SMCLK como clk do ADC |
284     //Sample-and-hold pelo sampling timer (256 ciclos)
285     // | modo de sequencia de conversao single-channel repetitivo
286     ADC12CTL1 = ADC12SHS_1 | ADC12SSEL_3 | ADC12SHP | ADC12CONSEQ_2;
287     ADC12CTL2 = ADC12RES_0; // Resolucao de 12 bits
288
289     // Habilita o ADC12_A
290     // Mudancas nos parametros do ADC12_A devem ser feitas
291     // ANTES de habilitar o ADC12_A!
292     ADC12CTL0 |= ADC12ENC;
293
294     // Habilita as interrupcoes
295     ADC12IE = ADC12IE0;
296 }
297
298 void ENCODER_CFG() {
```

```
299     P2DIR |= BIT4;                          //P2.4 sendo selecionado para saida
300     P2SEL |= BIT4 | BIT0;                   //P2.4 sendo selecionado para funcao
301     //PWM e P2.0 para funcao de captura
302     P2REN |= BIT0;
303 //    P2OUT |= BIT0;
304
305     P1REN |= BIT5 | BIT4 | BIT3;            //Resistores internos em modo pull-up
306     P1OUT |= BIT5 | BIT4 | BIT3;
307     P1IE  |= phaseb | sw;                    // interrupt enable dos pinos
308     P1IES |= phaseb | sw;
309     P1IFG = 0x00;
310
311     P4DIR |= BIT1 | BIT2;
312     P4OUT = 0x00;
313     P2DIR |= BIT7;
314     P2OUT = 0x00;
315 }
316
317 void DMA_CFG(){
318 //----------------inicializando os enderecos de memoria referentes ao DMA-----------//
319     __data16_write_addr((unsigned short) &DMA0SA,(unsigned long) &ADC12MEM0);
320     //endereco unico de origem
321
322
323     __data16_write_addr((unsigned short) &DMA0DA,(unsigned long) &adc_read);
324     //bloco de enderecos de destino
325
326     DMA0SZ = 50;
327     DMA0CTL |= DMADT_4 | DMADSTINCR_3;
328     DMACTL0 |= DMA0TSEL_24;
329     DMA0CTL |= DMAEN;
330
331     __data16_write_addr((unsigned short) &DMA1SA,(unsigned long) &ADC12MEM0);
332     //endereco unico de origem
333
334     __data16_write_addr((unsigned short) &DMA1DA,(unsigned long) &adc_conv);
335     //endereco unico de destino
336
337     DMA1SZ=1;
338     DMA1CTL |= DMADT_4 | DMAIE;
339     DMACTL0 |= DMA1TSEL_24;
340     DMA1CTL |= DMAEN;
341  }
```

B. Código para o arduino

```
#include <stdlib.h>
#include <SPI.h>
#include <UTFTGLUE.h>                 //use GLUE class and constructor
UTFTGLUE myGLCD(0,A2,A1,A3,A4,A0); //all dummy args

#define PLOTPOSX1   0
#define PLOTPOSY1   14
#define PLOTPOSX2   319 // Internal size of the rectangular area (with borders)
#define PLOTPOSY2   225 // Internal size of the rectangular area (with borders)
#define TDIVPOSX    173
#define TDIVPOSY    2
#define VDIVPOSX    243
#define VDIVPOSY    2
#define LENGTHX     (PLOTPOSX2-PLOTPOSX1-3) / 2
#define LENGTHY     (PLOTPOSY2-PLOTPOSY1-3) / 2
#define SPACING     24 // Space between each dot-1
#define GETMIDDLE(a)  (a & 0x01) ? ((a-1)/2) + 1 : a / 2

const int center[2] = {GETMIDDLE(PLOTPOSX1 + PLOTPOSX2), GETMIDDLE(PLOTPOSY1 + PLOTPOSY2)};

#define gety(y)   (center[1] - y)
#define getx(x)   (center[0] - x)

unsigned char dotsy[240] = {0};
unsigned char dotsx[320] = {0};
char temp[4];

typedef enum _datamode {
  data,
  time,
  volts
} datamode;

unsigned char dado, index;
datamode modo_dado = data;

typedef union _entrada {
  unsigned int val;
  unsigned char b[4];
} entrada;

entrada valorTempoDiv_prev;
entrada valorTempoDiv;
entrada valorVoltsDiv_prev;
entrada valorVoltsDiv;

void atualizaVoltsDiv(int val);
void atualizaTempoDiv(int val);

void setup()
{
  Serial.begin(9600);
  pinMode(SS, INPUT_PULLUP);
  pinMode(MOSI, OUTPUT);
  pinMode(SCK, INPUT);
  SPCR |= _BV(SPE);
  SPI.attachInterrupt();  //allows SPI interrupt

  randomSeed(analogRead(0));
  Serial.print("center: ");
  Serial.print(center[0]);
  Serial.print(center[1]);
  Serial.println();

  // Setup the LCD
  myGLCD.InitLCD();
  myGLCD.setFont(SmallFont);



// Clear the screen and draw the frame
  myGLCD.clrScr();

```

```
  myGLCD.setColor(255, 0, 0); // Retangulo vermelho
  myGLCD.fillRect(0, 0, 319, PLOTPOSY1-1); // Preenche retangulo de cima
  delay(500);
  myGLCD.setColor(64, 64, 64); // Retangulo cinza
  myGLCD.fillRect(0, PLOTPOSY2+1, PLOTPOSX2+1, PLOTPOSY2+14); // Preenche retangulo de baixo
  myGLCD.setColor(255, 255, 255); // Letra branca
  myGLCD.setBackColor(255, 0, 0); // Fundo vermelho
  myGLCD.print("Osciloscopio - MSP430F5529", LEFT, 2);
  atualizaTempoDiv(0);
  atualizaVoltsDiv(0);
  myGLCD.setBackColor(64, 64, 64); // Fundo cinza
  myGLCD.setColor(255,255,0); // Letra amarela

/*   myGLCD.print("f=??.??", 3, 227);
  myGLCD.print("Avg=??.??", 73, 227);
  myGLCD.print("Max=??.??", 143, 227);
  myGLCD.print("Min=??.??", 213, 227);
 */
  myGLCD.setColor(0, 0, 255); // Azul
  myGLCD.drawRect(PLOTPOSX1, PLOTPOSY1, PLOTPOSX2, PLOTPOSY2);
  // Retangulo azul envolvendo o plot

  // Draw crosshairs
  myGLCD.setColor(0, 0, 255); //Azul
  myGLCD.setBackColor(0, 0, 0); // Cor de fundo preta

  for(int i=PLOTPOSX1+((PLOTPOSX2-PLOTPOSX1+1)%(SPACING+1))/2; i<=PLOTPOSX2; i+=SPACING+1){
    dotsx[i] = 1;
    for(int j=PLOTPOSY1+((PLOTPOSY2-PLOTPOSY1+1)%(SPACING+1))/2; j<PLOTPOSY2; j+=SPACING+1){
      if(i == center[0]) {
        myGLCD.drawLine(center[0]-1, j, center[0]+1, j);
        dotsx[center[0]-1] = 1;
        dotsx[center[0]] = 1;
        dotsx[center[0]+1] = 1;
      }
      else if(j == center[1]) {
        myGLCD.drawLine(i, center[1]-1, i, center[1]+1);
        dotsy[center[1]-1] = 1;
        dotsy[center[1]] = 1;
        dotsy[center[1]+1] = 1;
      }
      myGLCD.drawPixel(i, j);
      dotsy[j] = 1;
    }
  }

  myGLCD.setColor(255,255,255);
  myGLCD.drawPixel(center[0], center[1]);
  myGLCD.setColor(0,0,255);
}

unsigned char bufentrada[PLOTPOSX2+1] = {0};
unsigned char buftela[PLOTPOSX2+1] = {0};
int x = 1, i = 0, j;
bool ok=false;

void loop()
{
  while(ok) {
    for(i=1;i<PLOTPOSX2;i++) {
      // Checar se mudaram as divisoes de tempo para mostrar na tela
      if((valorTempoDiv.val != valorTempoDiv_prev.val) && (modo_dado == time)) {
        atualizaTempoDiv(valorTempoDiv.val);
        valorTempoDiv_prev = valorTempoDiv;
      }
      if((valorVoltsDiv.val != valorVoltsDiv_prev.val) && (modo_dado == volts)) {
        atualizaVoltsDiv(valorVoltsDiv.val);
        valorVoltsDiv_prev = valorVoltsDiv;
      }

      if(i + 1 < PLOTPOSX2)
        j = i + 1;
      else/
        j = 1;

      if ((dotsx[j] & dotsy[buftela[j]]))
        myGLCD.setColor(0,0,255); // restaura o ponto
```

```
151       else
152         myGLCD.setColor(0,0,0); // restaura o fundo preto
153       myGLCD.drawPixel(j,buftela[j]); // efetivamente escreve o pixel
154
155       myGLCD.setColor(0,255,255);
156       myGLCD.drawPixel(i,bufentrada[i]);
157       buftela[i] = bufentrada[i];
158     }
159     ok=false;
160   }
161 }
162
163 ISR (SPI_STC_vect) {
164   dado = SPDR; // Receber o dado por SPI
165
166   switch(modo_dado) { // Se estamos esperando terminar um valor inteiro ou se eh so
167   //um ponto para plotar
168     case data: // Se estivermos no modo dado, podemos a qualquer momento receber um valor
169     //de tempodiv ou voltsdiv
170     switch(dado) {
171       case 0xFF: // 0xFF eh o comando para tempodiv
172         modo_dado = time; // Mudamos para o modo tempodiv
173         break;
174       case 0xFE: // 0xFE eh o comando para voltsdiv
175         modo_dado = volts; // Mudamos para o modo voltsdiv
176         break;
177       default: // Se nao for nenhum comando, entao eh so um ponto
178         if(!ok) {
179           if(x == PLOTPOSX2){
180             x = 1;
181             ok = true;
182           }
183           else {
184             x++;
185             bufentrada[x] = 2*center[1]-SPDR-PLOTPOSY1;
186             if(bufentrada[x] == 1) bufentrada[x]++;
187           }
188         }
189     }
190     break;
191
192   case time: // Se estamos no modo tempo, devemos esperar o inteiro ser enviado
193       if(index <= 3) { // Transferimos byte a byte o inteiro a ser lido
194         valorTempoDiv.b[index] = dado;
195         if(index == 3) { // Se ja chegamos no fim
196           index = 0; // Zeramos o indice para a proxima vez
197           modo_dado = data; // Se ja tivermos recebido o inteiro,
198          //mudamos para modo de dados de volta
199         }
200         else index++;
201       }
202   break;
203
204   case volts:
205       if(index <= 3) { // Transferimos byte a byte o inteiro a ser lido
206         valorVoltsDiv.b[index] = dado;
207         if(index == 3) { // Se ja chegamos no fim
208           index = 0; // Zeramos o indice para a proxima vez
209           modo_dado = data; // Se ja tivermos recebido o inteiro,
210           //mudamos para modo de dados de volta
211         }
212         else index++;
213       }
214   break;
215
216   default: break;
217   }
218 }
219
220 void atualizaTempoDiv(int val) {
221   char store[50] = "ms=";
222   myGLCD.setBackColor(255, 0, 0);
223   myGLCD.setColor(255, 0, 0);
224   myGLCD.fillRect(TDIVPOSX+7, TDIVPOSY, VDIVPOSX-1, PLOTPOSY1-1);
225   myGLCD.setColor(0,255,0); // Letra verde
226   itoa(val, temp, 10);
227   strcat(store, temp);
```

```
228   myGLCD.print(store, TDIVPOSX, TDIVPOSY);
229   myGLCD.setColor(0,255,255);
230 }
231
232 void atualizaVoltsDiv(int val) {
233   char store[50] = "mV=";
234   myGLCD.setBackColor(255, 0, 0);
235   myGLCD.setColor(255, 0, 0);
236   myGLCD.fillRect(VDIVPOSX+14, VDIVPOSY, VDIVPOSX+20, PLOTPOSY1-1);
237   myGLCD.setColor(0,255,0); // Letra verde
238   itoa(val, temp, 10);
239   strcat(store, temp);
240   myGLCD.print(store, VDIVPOSX, VDIVPOSY);
241   myGLCD.setColor(0,255,255);
242 }
```