Experimento 2

EA076 - LCD e Periférico de Entrada Analógico

Turma C - Grupo 6
Gustavo Ciotto Pinton RA 117136
Anderson Une Bastos RA 093392

Campinas, 27 de março de 2016

Introdução

No experimento 1, fomos introduzidos ao ambiente de programação *Processor Expert* e às referências disponibilizadas pelo fabricante do *kit* FRDM-KL25Z, que especificam, por exemplo, o descritivo dos registradores, as funções de cada pino e as características elétricas do microprocessador. A partir destes conceitos, extendemos nosso programa a fim de permitir a comunicação com um *display LCD* e a utilização do módulo de conversão analógico-digital.

A conexão com o *display LCD* requiriu, inicialmente, um estudo de sua implementação . Para tal, consultamos seu manual, disponibilizado em [1]. Nas próximas seções, destacamos, portanto, as exigências de temporização dos sinais e os comandos providos pelo componente. Com intuito de testá-lo, escrevemos simplemente o alfabeto e adicionamos a função de rolamento para a esquerda a partir de um *click* no botão.

O conversor analógico digital foi testado com um potenciômetro e, posteriormente, com um sensor de temperatura. Além de um estudo detalhado do respectivo módulo em [2], foi necessário determinar a relação entre temperatura e tensão do sensor. A equação foi encontrada no *datasheet* do componente, disponível em [3].

Display LCD

Esta seção é dedicada às etapas realizadas para configuração do *display* LCD.

Temporização

O fabricante do *display* LCD especifica que os sinais de entrada devem respeitar certos limites de tempo para que ele funcione corretamente. Em [1], estão descritas duas operações possíveis, sendo elas entrada e escrita. Considerando que não utilizaremos as operações de leitura das memórias internas ao componente, vamos nos atentar somente às operações de escrita. A página 10 de [1] resume bem as requisições temporais, porém antes de explicá-las, é necessário entendermos o que cada sinal de entrada realiza. Para tal, fornecemos a tabela 1 abaixo, que relaciona os pinos do LCD com suas respectivas funções. De acordo com a tabela, os dois primeiros pinos são dedicados à alimentação do dispositivo, os pinos 3 *(V0)*, 4 *(RS)*, 5 *(R/W)* e 6 *(E)* são interfaces de controle, os pinos de 7 a 14 são utilizados para envio de dados ou comandos, e, enfim, os dois últimos são reservados para configuração do *backlight* do LCD.

O envio de comandos ou dados deve respeitar, de acordo com a tabela da página 10 de [2], a seguinte ordem, para uma alimentação de 5V:

- *Configuração de RS* (pino 4)*:* caso desejemos enviar um caracter ao LCD, devemos modificar o nível lógico de *RS* para alto. Se quisermos enviar um comando, modificamos o respectivo nível para baixo.

- *Configuração de R/W* (pino 5)*: R*|*W* determina se a operação será de leitura ou de escrita. No nosso caso, como só iremos realizar escritas no componente, podemos ligar tal pino diretamente ao terra da fonte.

- *Configuração de E* (pino 6)*:* de acordo com [1], o dispositivo realiza a operação desejada assim que uma borda de descida for detectada nesta entrada. No mesmo *datasheet*, está indicado que este sinal deve ser configurado t_{AS} segundos após a configuração de RS, em que t_{AS} = 40ns, e que o nível deve ser

mantido em alto por, minimamente, PW_{EH} = 230ns antes de ser modificado para baixo e, assim, habilitar a operação. Além destes tempos, o sinal de RS deve ser mantido por t_{AS} = 40ns após a borda de descida.

- *Configuração dos pinos de dados ou comandos (pinos 7 a 14)*: o *byte* a ser enviado através destes 8 pinos necessita estar pronto, no mínimo, t_{DSW} = 80ns antes da borda de descida do sinal *E* e deve ser mantido por t_H = 10ns após este evento. O primeiro intervalo é chamado de *data set-up time* e o segundo, *data hold time*.

Pino	Função	Descrição
1	Alimentação	Terra ou GND
2	Alimentação	VCC ou +5V
3	V0	**Tensão para ajuste de contraste**
4	RS Seleção:	1 - Dado, 0 - Instrução
5	R/W Seleção:	1 - Leitura, 0 - Escrita
6	E Chip select	1 ou (1 → 0) - Habilita, 0 - Desabilitado
7	B0 LSB	Barramento de Dados
8	B1	
9	B2	
10	B3	
11	B4	
12	B5	
13	B6	
14	B7 MSB	
15	A (qdo existir)	Anodo p/ *LED backlight*
16	K (qdo existir)	Catodo p/ *LED backlight*

Tabela 1: Pinagem do LCD

Características Elétricas

O *datasheet* [1] contem também as especificações elétricas na tabela da página 5. Para uma entrada V_{DD} = 5V, a tensão de entrada considerada alta nos pinos de entrada deve valer, no mínimo, V_{IH} = 0.7 * V_{DD} = 3.5V e a tensão considerada como nível lógico baixo deve ser, no máximo, V_{IL} = 0.3 * V_{DD} = 1.5V. Considerando que o manual de referência [4] do *kit* especifica, na tabela 7 da seção 2.2.3, que a tensão máxima de saída nível lógico baixo é 0.5V e a tensão para nível lógico alto é 3.6V , podemos concluir que as especificações elétricas entre os dois componentes são atendidas.

Implementação da Comunicação entre Microcontrolador e LCD

Com base nas seções anteriores, ligamos os pinos *PTC0 - PTC7* do microcontrolador aos pinos 7 a 14, o *PTC8* no sinal *RS* e *PTC9* em *E*. Conforme comentado anteriormente, *R/W* foi diretamente ligado ao terra e foi utilizada uma fonte de 5V para alimentação do LCD.

Além das restrições comentadas na subseção **Temporização**, [1] também fornece (tabela da página 12) o tempo que deve ser esperado para que cada comando ou escrita de dado seja corretamente executado. Sendo assim, aproveitamos o componente *Timer*, adicionado no experimento 1, para garantir que o programa espere o tempo necessário. Considerando que o menor tempo não nulo presente na tabela é 39µs, vamos configurar o *Timer* para gerar interrupções a cada 60µs, sendo este o menor valor possível de configuração. Dessa forma, por exemplo, um comando de *Clear Display,* que precisa de 1.53ms, necessita de $\frac{1.53ms}{60\mu s} \simeq 26$ interrupções do

Timer. Além dessa alteração, adicionamos outras duas variáveis *globais*, `autorizacao` e `wait_interval`, responsáveis pela comunicação entre a função *main* e a tratadora de interrupções. A primeira é uma variável lógica que indica se o fluxo da *main* pode prosseguir, enquanto que a segunda é um contador decrescente de interrupções que já foram processadas, cujo valor inicial é calculado da mesma forma que no exemplo anterior. A função é, portanto:

Programa 1: Função de tratamento de interrupções geradas pelo timer.

```
void timer_OnInterrupt(void)
{
   if (wait_interval == 0)
     autorizacao  = 1;

   if (wait_interval > 0)   wait_interval--;

}
```

Quando o contador atinge 0, `autorizacao` recebe `true` (1 em C) e o programa principal pode prosseguir. Se o contador ainda não for nulo, decrementamos seu valor, indicando que mais 60µs passaram.

No programa principal, criamos três funções importantes para o funcionamento do LCD. A primeira, `wait_n_interruptions`, é responsável por fazer o fluxo parar até que o contador citado acima seja nulo. Seu código está representado abaixo. Ela recebe como parâmetro o número de interrupções que ela deve esperar.

Programa 2: Função que espera que o contador seja nulo.

```
void wait_n_interruptions (int n_interruption) {

    autorizacao = 0;
    wait_interval = n_interruption;

    while (!autorizacao || wait_interval > 0);  // Aguarda ambas condicoes serem
                                                // atendidas.

}
```

A segunda função envia um caracter para ser escrito no LCD, respeitando a ordem e a temporização exigida pelos sinais, de acordo com a subseção **Temporização**. A penúltima linha da tabela da página 12 de [1] especifica que uma operação de escrita necessita de 43µs para ser realizada. Em termos de interrupções, necessitamos aguardar somente 1. É por esta razão que o parâmetro de `wait_n_interruptions` vale 1.

Programa 3: Função que envia caracteres ao LCD.

```
void send_data(char data) {

    DataCmd_PutVal(data);     // Coloca dado nos pinos PTC0-PTC7
    RS_SetVal();              // 1 -> RS (dados)
    Enable_SetVal();          // 1 -> E
    Enable_ClrVal();          // 0 -> E (borda de descida: ativa LCD.)

    wait_n_interruptions(1);  // Espera somente 1 interrupcao.
}
```

Enfim, a terceira função envia comandos. A única diferença em relação à função anterior é que ela recebe a quantidade de interrupções a ser aguardada como parâmetro.

Programa 4: Função que envia comandos ao LCD.

```
void send_cmd(char cmd, int interval) {

    DataCmd_PutVal(cmd); // Coloca comando nos pinos PTC0-PTC7
    RS_ClrVal();         // 0 -> RS (comandos)
    Enable_SetVal();     // 1 -> E
    Enable_ClrVal();     // 0 -> E (borda de descida: ativa LCD.)

    wait_n_interruptions(interval);
}
```

Um exemplo da utilização destas funções é a inicialização do componente conforme código abaixo:

Programa 5: Função de inicialização do LCD.

```
void init_LCD() {
    // Inicializacao conforme pagina 13 de [1].
    send_cmd(0x3F, 1);
    send_cmd(0x0F, 1);
    send_cmd(0x01, 26);
    send_cmd(0x06, 1);
}
```

Em blocos, o funcionamento do programa pode ser resumido conforme figuras abaixo:

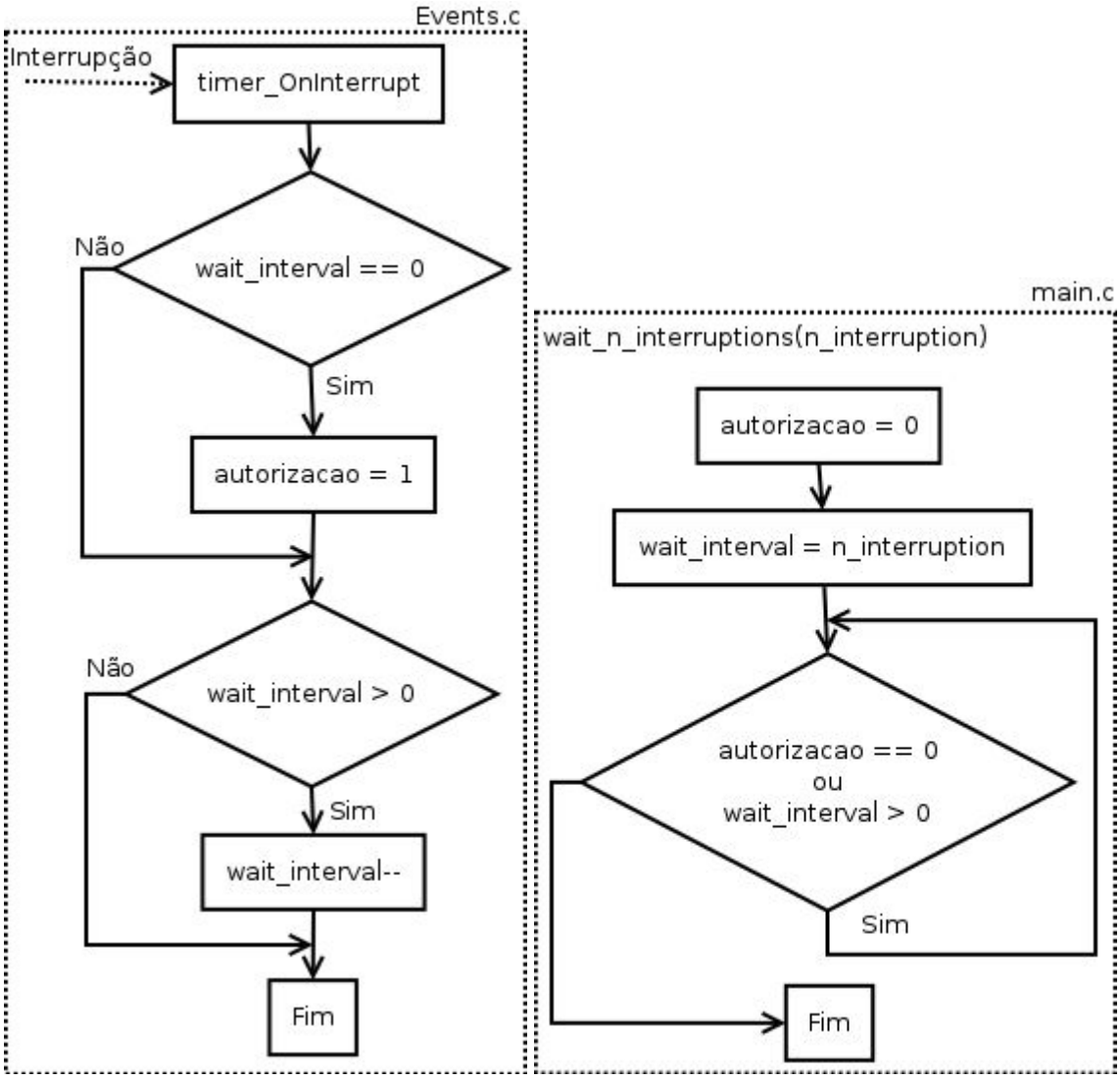


Figura 1: (a) Fluxograma da função tratadora de interrupções e (b) função `wait_n_interruptions`.

A figura 1a representa a função que trata interrupções gerados pelo *timer* e a figura 1b, a função que espera `n_interruption` interrupções.

Escrita do Alfabeto o LCD

A fim de escrevermos o alfabeto no LCD, desenvolvemos a seguinte função:

Programa 6: Função de que escreve o alfabeto no LCD.

```
void write_all_letters(char start_letter, char end_letter) {

    char caracter = start_letter;
    int i, char_count = 0;

    while (caracter <= end_letter) {
      send_data(caracter);
      caracter++;
      char_count++;

      if (!( char_count % 16 ) ) {
           for (i = 0; i < 24; i++)
                send_data(' ');
      }
    }
}
```

A função recebe como parâmetro duas variáveis do tipo `char` (1 byte), representando, respectivamente, a primeira e última letra a ser impresso. A cada iteração escrevemos uma letra através da chamada `send_data(caracter)`. A variável `char_count` conta o número de caracteres que já foram escritos numa linha. Se essa variável possui 16, então precisamos pular de linha. Para tal, adicionamos 14 espaços na memória do LCD. É importante ressaltar que tentamos utilizar o comando `Set DDRAM Address` presente na tabela da página 12 de [1], mas não obtemos sucesso, já que o *cursor* era deslocado corretamente, mas a escrita do caracter não ocorria.

Entrada Analógica

1. O ADC converte valores analógicos para digital através de um algoritmo de aproximações sucessivas. No nosso modo de operação, quando o ADC finaliza a conversão de uma leitura, uma interrupçao é chamada, no caso `void AD1_OnEnd(void)`, e logo após o ADC entra em *hold* para aguardar o próximo valor a ser lido e convertido. É importante lembrar que existem outros modos de operação descritos no manual para leituras de valores seguidos sem que se tenha finalizado a leitura anterior.

   Segundo o manual s*eção 28.4 de [3],* o ADC deve ser calibrado através de uma função *on-chip.* Nenhuma instrução relatava o estado da calibração, portanto foi apenas inspecionada dentro da própria função `AdcLdd1.c` do ADC se existe de fato algum método para realizar a calibração. O resultado foi a identificação do um método chamado `AdcLdd1_GetCalibrationResultStatus`, responsável pela calibração do ADC.

   As tensões de referência são setadas de acordo com o V_{SS} e com o V_{DD} da seguinte maneira: $V_{REFL} = V_{SS}$ e $V_{REFH} = V_{DD}$, no nosso caso $V_{REFL} = 0V$ e $V_{REFH} = 3{,}3V$. Isso garante que o conversor funcione de maneira adequada e não sature ocasionando sempre valores de conversão de 0xFFF.

Apenas a título de informação, o tempo total de conversão do ADC foi estimado de modo a garantir conversões sem erros, e pode ser calculado da seguinte maneira (equação 1):

Total Time = conversion time + sample time + discharging time **(1)**

Em que os fatores acima são dados por:

- **Conversion time:** *12bit resolution: 16 cycles*;
- **Sample time:** pode ser considerado entre 4 – 24 ciclos;
- **Discharging time:** entre 0 a 2 ciclos;

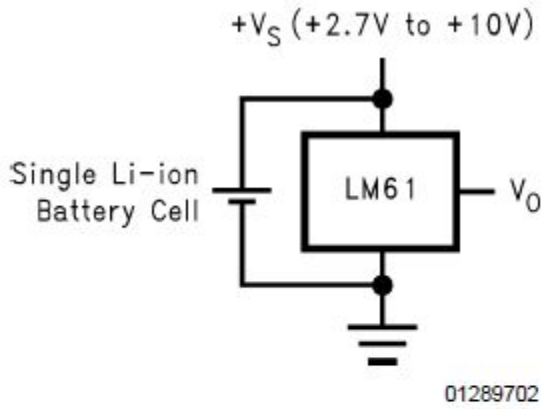
Então, o tempo mínimo e máximo para um ADC de 12 bits de resolução é:

- Tempo mínimo: 16 + 4 + 0 = 20 ciclos
- Tempo máximo: 16 + 24 + 2 = 42 ciclos

Os itens 2 e 3 são mostrados no código fonte, fornecido junto com o relatório.

Sensor de Temperatura

1. Utilizando [3] como base para a nossa aplicação obtemos o seguinte esquemático e sua respectiva função de transferência:



V_O = (+10 mV/°C x T °C) + 600 mV

Temperature (T)	Typical V_O
+100°C	+1600 mV
+85°C	+1450 mV
+25°C	+850 mV
0°C	+600 mV
−25°C	+350 mV
−30°C	+300 mV

Figura 2: Aplicação proposta do sensor de temperatura e sua função de transferência.

Notamos que a cada variação de 1ºC do sensor, o valor de leitura varia de 10mV. Portanto, para obtermos a tensão medida podemos realizar as seguintes operações:

Tensao medida = Tensao de 0ºC + Variação **(2)**
Variação = Temperatura medida * 10mV

Substituindo os valores na equação (2), obtém-se:

$$V_{med} = 600mV + T_{med} * 10mV \quad (3)$$

Em que V_{med} e T_{med} são, respectivamente, a tensão e temperatura medidas.

Ou seja, isolando o termo T_{med}:

$$T_{med} = (V_{med} - 600mV) / 10mV \quad (4)$$

T_{med} então é convertido para valores em `char` através da função `sprintf` e enviado para o LCD pelo método anteriormente explicado. O código pode ser conferido no anexo.

Referências

[1] *Specifications for LCD module*, disponível em `ftp://ftp.dca.fee.unicamp.br/pub/docs/ea079/datasheet/AC162A.pdf`

[2] *KL25 Sub-Family Reference Manual – Freescale Semiconductors (doc. Number KL25P80M48SF0RM)*, disponível em `ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/ARM/KL25P80M48SF0RM.pdf`

[3] *Datasheet – LM61,* disponível em `ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/datasheet/LM61.pd`

[4] *KL25 Sub-Family Datasheet, disponível em* `ftp://ftp.dca.fee.unicamp.br/pub/docs/ea076/datasheet/KL25P80M48SF0.pdf`