Laboratório 1 - Introdução

EA076 - Laboratório de Sistemas Embarcados

Gustavo **CIOTTO PINTON** - **RA 117136**
Grupo **6** - Relatório Individual
Turma **C**

Campinas, 13 de março de 2016

1 Introdução

O principal propósito deste primeiro experimento foi explorar as ferramentas disponibilizadas pelo *software Code Warrior*, uma interface gráfica baseada na *IDE Eclipse*, para o desenvolvimento de *firmwares* compatíveis com microcontrolador *Freescale KL25*. Sendo assim, utilizamos tal plataforma para configurar alguns pinos do microcontrolador como entrada ou saída, dependendo da montagem desejada. Foram realizadas, no total, três montagens distintas.

Na primeira, o objetivo foi construir um programa capaz de piscar o *LED* vermelho, presente no *kit*, em uma frequência de 1Hz. Para tal, foi necessário consultar os manuais do microcontrolador [1] para entender como a comunicação entre ele e o diodo poderia ser realizada e como gerar interrupções em instantes periódicos de tempo. Em seguida, levando em conta que o *kit* possui ainda outros dois *LEDs*, extendemos o programa para piscar também o *LED* verde, a uma frequência de 2Hz.

A segunda tarefa consistiu na introdução de um *LED* externo, de modelo *TLDR490* fabricado pela empresa *Vishay* e cuja documentação é disponibilizada no site do curso [2]. Para realizá-la, foi preciso escolher um pino disponível do microcontrolador para funcionar como *output* e calcular a resistência necessária para limitar a corrente a fim de não danificar nenhum dos dois componentes.

Enfim, a última tarefa envolveu o uso de um *push-button* para fornecer ao usuário a possibilidade de controlar o funcionamento dos *LEDs*. Para isso, tivemos de configurar um pino como entrada, assim como escolher uma resistência para limitar a corrente que passa pelo circuito. Essa resistência é chamada de *pull* em referência à função que desempenha: pode induzir um estado padrão alto (*pull up*) ou baixo (*pull down*) na saída do circuito, que, nosso caso, é o pino que foi configurado como *input*.

As próximas seções visam a análise dessas montagens.

2 Análise

Esta seção é dedicada às características que determinam as três montagens.

2.1 Criação do Projeto

Com base no tutorial fornecido em [3], foi possível realizar um projeto capaz de piscar o *LED* vermelho, conectado ao pino **PTB18**, a uma frequência de 1Hz. A fim de extendê-lo para também piscar o verde, adicionamos um outro pino *BitIO* de entrada ou saída, a partir da aba *Components Library*, e o configuramos de acordo com a informações contidas em [1] sobre o *LED* verde. No campo *Pin for I/O*, introduzimos o pino **PTB19** e, posteriormente, geramos a função *NegVal* da mesma maneira que geramos para o vermelho. Para fazê-lo piscar a uma frequência de 2Hz, utilizamos o mesmo *timer T1*, no entanto alteramos o período de geração de interrupções para $250ms$. Na função tratadora da interrupção, introduzimos uma variável inteira persistente (*static*), isto é, seu valor será conservado ao fim da função e poderá ser acessado em outras chamadas, cujo valor será 0 ou 1 e, a cada interrupção tem seu conteúdo negado. Quando tal variável vale 0, os dois *LEDs* estão acesos e quanto vale 1, somente o verde se acende. Dessa forma, o *LED* vermelho permanecerá acesso ou apagado por 2 interruções, isto é, por $500ms$, tendo assim um período de $1s$. O verde muda de estado a cada 250ms, obtendo, portanto, um período de 500ms.

As funcionalidades acima podem ser resumidas em diagramas de blocos funcionais. Para a configuração contendo somente o *LED* vermelho, obtem-se o seguinte diagrama, em que blocos em azul são implementados como circuito elétricos (*hardware*) e, em vermelho, como *software*:

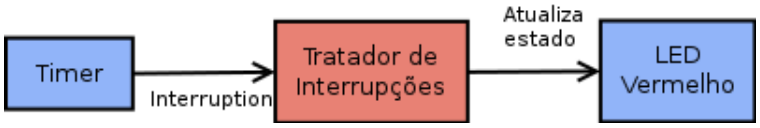

Figura 1: Diagrama para a primeira montagem.

O *timer* é um circuito que possui um registrador que realiza contagens a cada ciclo de *clock*. Quando a contagem chega a um limite, que é calculado em função do intervalo de tempo desejado, uma interrupção é lançada para o processador. O tratador desta interrupção é uma função, escrita em linguagem de alto ou baixo nível, cujo propósito é reagir ao "estímulo". No nosso caso, tal rotina chamará outras funções capazes de alterar o estado lógico dos *LEDs*. O estado lógico precisa ser convertido em características físicas, como corrente e tensão, já que os *LEDs*, sendo diodos, são componentes de um circuito elétrico. Observa-se que o único bloco constituído de *software* realiza um papel centralizador e controlador dos módulos *hardware*, o que permite ao programador uma grande liberdade e escolhas de programação.

Para a montagem com o *LED* verde:

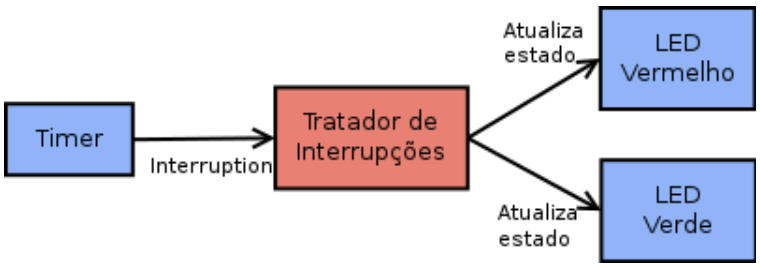

Figura 2: Diagrama para a segunda montagem.

O diagrama é muito parecido com aquele presente na figura 1. Isto quer dizer que é possível utilizarmos praticamente os mesmos componentes utilizados na montagem anterior e configurar somente os parâmetros que necessitam ser alterados, como, por exemplo, o período de geração de interrupções.

2.2 Conexão de LED externo

A primeira etapa para a conexão de um *LED* externo é a escolha da porta que funcionará como saída e que fornecerá a corrente necessária para que ele se acenda. Tal escolha foi baseada na posição das conexões dos pinos nos *headers* da placa, de acordo com [4], e na disponibilidade do respectivo pino de funcionar como *output*. Sendo assim, por estar na primeira linha e na primeira coluna do *header* da esquerda, facilitando assim a sua identificação na placa auxiliar da FEEC, escolhemos o pino **PTA1**. Adicionamos, posteriormente, um novo componente *BitIO* ao projeto e o configuramos para refletir a escolha acima. Modificamos o campo *Init. value* para 1, a fim de que o *LED* externo já se acenda assim que o programa for lançado.

Antes de realizarmos a montagem do circuito, é necessário considerar as correntes e tensões máximas suportadas pelos componentes, cujos valores são descritos nos *datasheets* do microcontrolador [5] e do *LED* externo [2]. Para o primeiro, as linhas 1 e 3 a tabela 7 especificam, respectivamente, que a tensão mínima fornecida no pino em estado lógico alto é $V_{DD} - 0,5$, isto é, 3.3V, e que a corrente máxima total de todas as portas é 100mA. Considerando, que utilizaremos somente uma porta, o valor máximo de corrente suportada pelo microcontrolador é, portanto, 100mA. Para o *LED*, [2] nos diz que a corrente DC direta máxima vale 50mA e que a maior tensão reversa suportada é 6V. Sendo assim, adotaremos um limiar máximo para a corrente de **50mA**.

Considerando o valor mínimo de tensão no pino de saída do microcontrolador, isto é, $V_{pin} = 3.3V$, a figura 5 do *datasheet* do *LED*, que especifica uma relação diretamente entre corrente i e tensão

direta V_{LED}, e a relação $i = \frac{V_{pin} - V_{LED}}{R}$, podemos escolher um resistor de 100Ω. Obtemos, portanto, uma corrente de aproximadamente 13mA.

É importante notar que utilizamos o pino terra da própria placa, isto é, o pino **GND**, presente no 14º pino do *header* inferior esquerdo da figura da página 12 de [5].

O diagrama desta montagem é muito simples: logo que o programa é iniciado, o *LED* externo é aceso. O bloco *LED Externo* engloba todo o circuito (resistência, diodo, fios) que foram necessários para a sua implementação. Observa-se igualmente que este bloco poderia ser facilmente substituído por um dos blocos *LED vermelho* ou *LED verde* das figuras 1 e 2. Esta característica demonstra que programas feitos para este microcontrolador não dependem estritamente de componentes específicos, isto é, são capazes de garantir soluções genéricas que funcionam para diversos tipos de *hardware* distintos, em termos de tensão, corrente, potência *etc.*

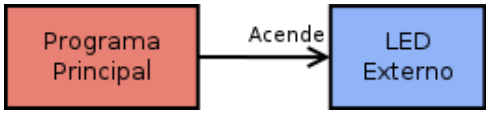

Figura 3: Diagrama para a terceira montagem.

2.3 Chave de controle

A última montagem envolve a utilização de um pino de entrada e de um *push button*, cuja função é controlar a atividade dos *LEDs*, de forma a permitir que eles pisquem quando o botão estiver apertado e apagá-los, caso contrário. Para isso, utilizamos um circuito representado na figura 4 a seguir, em que **P3V3** é um pino que fornece uma tensão de 3.3V, **GND** é a referência do terro na placa e **PTD4** é o pino escolhido como *input*.

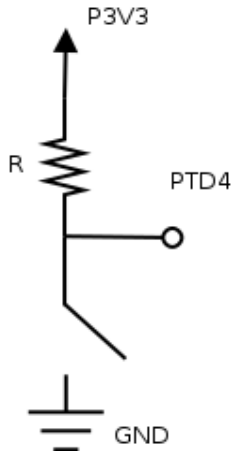

Figura 4: Circuito utilizado na montagem.

O resistor **R** é chamado de *pull up resistor*, visto que ele induz um estado padrão alto na saída do circuito, isto é, **PTD4**. O circuito pode apresentar dois comportamentos distintos, dependendo do estado da chave. Quando está aberta, praticamente há não passagem de corrente entre **P3V3** e **PTD4**, já que a impedância de entrada do pino de entrada é elevada. Por esta razão, o nível lógico deste pino é alto nesta configuração. Em oposição, quando a chave é fechada (*push-button* pressionado), **PTD4** conecta-se ao terra e, portanto, adquire um nível lógico baixo.

O valor de **R** deve garantir as especificações de corrente máxima do fabricante. Sendo assim, utilizamos uma resistência de $1k\Omega$, que garantirá uma corrente de aproximadamente 3.3mA passando pelo circuito quando a chave estiver fechada.

Por fim, adicionamos um novo componente *BitIO* ao projeto, o configuramos para referenciar **PTD4** e funcionar como *input*. Geramos, em seguida, a função *GetVal*, cujo valor de retorno é o estado lógico do respectivo pino. A cada interrupção do *timer*, chamaremos tal função para determinar se o botão foi pressionado ou não. Se o retorno for nulo, isto é, botão apertado, piscamos os *LEDs*, conforme seções anteriores. Senão, os apagaremos colocando níveis lógicos altos em **PTB18** e **PTB19**.

O diagrama da última montagem é representado na figura 5. O bloco *push button* engloba todo o circuito representado pelo figura 4.

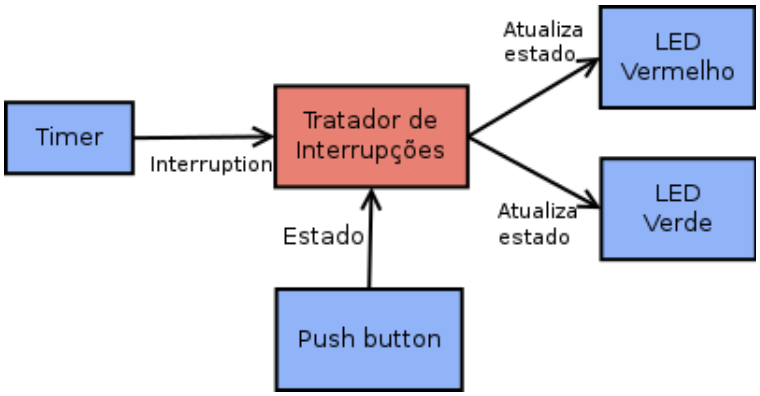

Figura 5: Diagrama para a quarta montagem.

Nota-se que a única diferença entre este diagrama e aquele da figura 2 é a presença do *push button*. Sendo assim, observa-se uma grande portabilidade de código e componentes entre diversas aplicações.

3 Conclusão

Trabalhamos neste experimento alguns conceitos importantes envolvendo a programação de sistemas embarcados. O primeiro foi a necessidade de consulta frequente aos manuais fornecidos pelos fabricantes, a fim de obtermos os limites elétricos dos componentes e, assim, determinar os circuitos que ligaremos ao microcontrolador. O segundo aspecto foi a possibilidade de utilizar um mesmo pino em diversas funções diferentes, seja como entrada/saída ou comunicação serial, por exemplo. Neste quesito, aprendemos também como identificar a localização de um determinado pino na placa.

Exploramos igualmente o ambiente de desenvolvimento *Code Warrior*, que oferece muitos recursos na programação de *firmwares*, tais como a geração automática de código e interfaces de configuração dos pinos. Na disciplina EA871, por exemplo, somos obrigados a configurar todos os registradores individualmente, o que dificulta em algumas ocasiões o *debugging* do programa. Em oposição, em EA076, permitimos que o *Code Warrior* se encarregue destes detalhes.

A construção de diagramas de blocos evidenciou características importantes no desenvolvimento, como a re-utilização de código e portabilidade de componentes (isto é, um mesmo código pode funcionar com diversos *hardwares* distintos). Em escala industriais, tais aspectos são traduzidos em economia de recursos e em uma maior facilidade de implementação tanto de *software* quanto de *hardware*.

Referências

1. *KL25 Sub-Family Reference Manual*, disponível em `ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/ARM/KL25P80M48SF0RM.pdf`.

2. *High Intensity LED in ø 3 mm Clear Package*, disponível em `ftp://ftp.dca.fee.unicamp.br/pub/docs/ea079/datasheet/83002.pdf`.

3. *Introdução ao Freescale Processor Expert para a disciplina EA076*, QUEVEDO A. F., Antonio, disponível em `ftp://ftp.dca.fee.unicamp.br/pub/docs/ea076/complementos/Apostila_PE.pdf`.

4. *FRDM - KL25Z User's Manual*, disponível em `ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/ARM/FRDMKL25Z.pdf`.

5. *Data Sheet Kinetis KL25 Sub-Family*, disponível em `ftp://ftp.dca.fee.unicamp.br/pub/docs/ea076/datasheet/KL25P80M48SF0.pdf`.