EA871 – LAB. DE PROGRAMAÇÃO BÁSICA DE SISTEMAS DIGITAIS

EXPERIMENTO 6 – *Polling*

Profa. Wu Shin-Ting

OBJETIVO: Apresentação de um projeto de interface entre um microcontrolador e uma chave simples.

ASSUNTOS: Técnicas de *debounce*, técnica de *polling* e técnicas de programação.

O que você deve ser capaz ao final deste experimento?

Entender o problema de *bounce* em chaves e alternativas para contorná-lo.

Saber a técnica de *debounce* adotado no shield FEEC 871.

Saber aplicar a técnica de *polling* na captura de eventos externos.

Saber aplicar tipos de dados enum e struct para tornar o código mais legível.

INTRODUÇÃO

Chaves SPST (com um pólo e uma posição), momentâneas (passa momentaneamente para um outro estado quando são acionadas) e normalmente abertas (estado normal é aberto) são muito comuns em sistemas embarcados. Exemplos típicos são as botoeiras “Start”, “Stop” e “Reset” que se encontram na maioria dos sistemas. Como toidos os projetos de interface com periféricos, o projeto de interface destas chaves com um microcontrolador deve levar em conta as **compatibilidades funcionais** (funções de sinais), **elétricas** (níveis de sinais), **temporais** (velocidade dos sinias) e **mecânicas** (pinagens/conexões fpisicas). Funcionalmente, os dois níveis lógicos (0/1) correspondem a dois estados que uma chave pode assumir (aberta/fechada). Sob o ponto de vista de conexões, as duas pontas de uma chave SPST podem ser conectadas, respectivamente, no pino de entrada do microcontrolador e o terra (Figura 1).

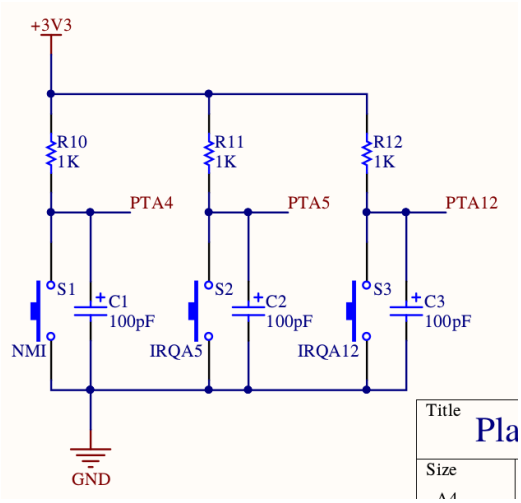

Figura 1: Esquemático da conexão de chaves no *shield* FEEC871.

Em relação à compatibilidade elétrica, devemos considerar ***bounces*** dos sinais elétricos nas chaves por um intervalo médio de 20ms até se estabilizar num estado, tanto no instante de mudar para o estado momentâneo quanto no instante de voltar para o estado normal. E em relação à compatibilidade temporal, temos o problema de que a ação dos usuários é totamente **assíncrona**

em relação ao ritmo de execução do processador. Não se sabe exatamente o instante em que a chave pode ser acionada nem o intervalo em que ela fica acionada.

Há diversas soluções de *debounce*. Elas podem ser por *hardware* e por *software*. A solução por *hardware* é baseada no princípio de filtragem de *bounces* pelo descarregamento lento do capacitor em relação à fonte de *bounces*. Enquanto a solução por *software* é a inserção de um atraso a partir da detecção da primeira borda de *bounce*, na expectativa de que o valor lido seja um valor no estado estabilizado. Analisando o esquemático do *shield* FEEC 871 (Figura 1), é possível constatar que a solução adotada é por *hardware* utilizando um capacitor de 100pF e um resistor de 1K. Com isso, garante-se a **compatibilização elétrica** dos sinais.

Em relação à **compatibilização temporal**, podemos adotar uma solução conhecida por *polling* em que se programa o processador para verificar periodicamente o estado das chaves, como ilustra a Figura 2. Embora não seja uma solução perfeitamente sincronizada, a peobabilidade de amostrar todas as mudanças de estados é grande quando a frequênci a de amostragem for alta.

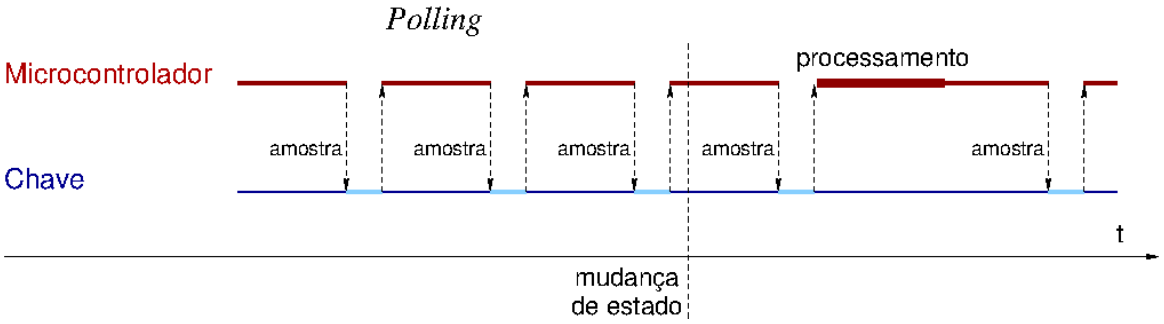

Figura 2: Técnica de *polling* para monitorar mudança de estado de uma chave.

Assim, usando uma solução por *hardware* e outra solução por *software* conseguimos assegurar os quatro tipos de compatibilidade conforme sintetiza a Tabela 1.

	Microcontrolador	Chave
Funcional	Sinal digital de propósito geral. Níveis lógicos 1/0	Aberta/Fechada
Elétrico	3,3V / 0V	Com debouncer por *hardware*: 3.3V / 0V
Temporal	1 ciclo de instrução: ~50ns	Com *polling*: o estado é amostrado sincronamento pelo processador.
Mecânico	Pino de entrada e terra	Um pólo e uma posição

Tabela 1: Compattibilização entre um microcontrolador e uma chave SPST.

EXPERIMENTO

Neste experimento vamos desenvolver um projeto “escolha_cor” dentre uma paleta de cores pré-definidas **cor []** = {R, G, B, R+G, G+B, B+R, R+G+B, sem luz} (não precisa ser necessariamente esta sequência) a serem geradas pelo *led* RGB conectado nos pinos PTE21 (R), PTE22 (G) e PTE23 (B) da porta E. O processo de escolha consiste em acionar a botoeira IRQ12. (conectado no pino PTA12 de PORTA) [1]. A cada acionamento, a cor é mudada para a próxima cor na sequência da paleta até chegar na cor desejada. A sequência de cores se repete em cada 8 acionamentos, ou seja, o valor **indice_apropriado** no seguinte pseudocódigo deve se repetir ciclicamente:

```
GPIO_initLedRGB();

GPIO_initIRQ12();

Laço:

    Se (GPIO_sampleSwIRQ12() == HABILITA) então

        GPIO_LedRGB(cor[indice_apropriado]);
```

1. Qual são as macros que manipulam os 3 *bits* do campo MUX de um registrador POTRx_PCRn? Explique como elas operam.

2. Em qual endereço o registrador GPIOA_PDIR está mapeado? Qual é a função de cada *bit* deste registrador?

3. Em C podemos organizar a sequência de cores da paleta num vetor/arranjo de cores em que cada elemento seja referenciado pelo seu índice. Por exemplo, o vetor **cor[]**. Ao invés de usar índices numéricos, como você poderia substituí-los pelos nomes de cores, VERMELHO, VERDE, AZUL, AMARELO, CIANO, MAGENTA, BRANCO, PRETO, com os quais estamos mais familiarizdos?

4. Para acessar os elementos do vetor **cor[]** sequencialmente, você pode definir uma variável de 32 *bits*, por exemplo **contador**, inicializada em zero e em cada acesso ao vetor **cor**[], incremente **contador** de 1. Qual operação você poderia aplicar a **contador** para gerar valores de acesso apropriados para a variável **indice_apapriado**? E o que acontece quando o conteúdo de **contador** fique maior que 0xFFFF_FFFF?

5. Reusando as funções implementadas no aplicativo **polling** [2], implemente o aplicativo **escolha_cor** em C. Utilize a técnica *polling* para amostrar o estado da botoeira.

6. Documente todas as funções que não foram geradas pelo IDE CodeWarrior.

RELATÓRIO

Para este experimento, responda as questões 1 a 4 e submeta o projeto exportado no ambiente IDE CodeWarrior, o aplicativo **escolha_cor** documentado, no sistema *Moodle*.

REFERÊNCIAS

[1] KL25 Sub-Family Reference Manual, Rev. 3, September 2012

ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/ARM/KL25P80M48SF0RM.pdf
[2] polling.zip
http://www.dca.fee.unicamp.br/cursos/EA871/2s2020/ST/codes/polling.zip

Setembro de 2020