EXPERIMENTO 8 – TPM, DMA e DMAMUX

FEEC | EA871

Thiago Maximo Pavão - 247381
Vinícius Esperança Mantovani - 247395

Terceira parte

3.

Implementaremos os estados PREPARA_INICIO, INICIO, LEITURA, PREPARA_AUDITIVO, ESPERA_ESTIMULO_AUDITIVO, LARGADA_QUEIMADA, ESPERA_REACAO_AUDITIVA e, RESULTADO. O programa inicia no estado PREPARA_INICIO, apesar da máquina de estados impor o início em INICIO. Isso porque, deste modo, conseguiremos controlar melhor o fluxo do programa e seu funcionamento.

Deve-se perceber de início, que implementamos um estado além daqueles apresentados na máquina de estados dada, o estado LARGADA_QUEIMADA. Este estado é responsável por impedir que o usuário pressione subsequentemente a botoeira NMI antes de o buzzer soar, com o intuito de trapacear, uma vez que, desse modo, pressionaria quase que imediatamente após o início do soar do buzzer. Para tanto, esse estado é acionado quando se está no estado PREPARA_AUDITIVO ou ESPERA_ESTIMULO_AUDITIVO, caso a botoeira NMI seja apertada. Essa transição, fará com que uma mensagem seja exibida no LCD e, que o fluxo do programa seja desviado para o estado LEITURA.

Sendo assim, tratemos do estado LEITURA para seguir com a linha de raciocínio anterior. Este estado será responsável por transitar para o estado PREPARA_INICIO e desativar o canal 4 de TPM0 e ativar IRQA12, permitindo que outra rodada seja iniciada com seu pressionamento.

Assim, entramos no estado PREPARA_INICIO, responsável por escrever “Pressione IRQA12” no LCD e, transitar para o estado INICIO, isto será feito pela main.

No estado INICIO, que usaremos para aguardar o pressionamento da botoeira IRQA12, o que ocasionará a transição para o estado PREPARA_AUDITIVO, nesta transição é feita a ativação da botoeira NMI (feita aqui para detectar se a largada foi queimada), a IRQA12 deve ser desativada e a interrupção de overflow do TPM0 deve ser ativada, para que o tempo aleatório seja contado. Sigamos, então para o PREPARA_AUDITIVO.

Vale ressaltar que é nesta transição, feita pela rotina de tratamento de interrupção de TPM1 que o número aleatório (para a espera até soar o buzzer) é gerado, isto foi feito desta forma pois a função que gera o número aleatório faz isso lendo o valor no contador de TPM0, e portanto o valor é mais aleatório quando depende da interação do usuário com o botão. Caso o número fosse gerado no estado PREPARA_INICIO, o valor no contador poderia ser mais previsível, dado que este é o estado inicial do programa, é possível que o valor sorteado no início da execução do programa fosse sempre próximo de um mesmo valor. Esperar a interação do usuário com a botoeira resulta em um valor sempre imprevisível.

Em PREPARA_AUDITIVO, escreveremos a mensagem " Teste Auditivo " no LCD, transitando diretamente para o próximo estado no fim da escrita, o estado ESPERA_ESTIMULO_AUDITIVO sem qualquer condição necessária, apenas que não se queime a largada.

Assim, tratando de ESPERA_ESTIMULO_AUDITIVO, implementaremos este estado para aguardar o soar do buzzer, que é ocasionado pela coincidência do valor de counter com 0, onde counter é uma variável estática definida em ISR.c para determinar o número de overflows até que o buzzer soe. Quando isto ocorre, é necessário ativar o buzzer e o canal 4 de TPM0 em output compare com o valor atual do contador de TPM0, para que seja possível calcular o tempo de reação posteriormente. Além disso, a interrupção de overflow do TPM1 deve ser desativada, já que o tempo de espera aleatória acaba de terminar. Assim, com o soar do buzzer, entra-se no estado ESPERA_REACAO_AUDITIVA.

Neste estado, aguardamos o pressionamento de NMI para que se calcule o valor do tempo de reação do usuário. Enquanto não há pressionamentos, cada passagem do contador de TPM0 pelo mesmo valor que foi armazenado na transição incrementa counter, para posterior uso no cálculo do tempo de reação. Quando a NMI é pressionada, o cálculo é feito, ela é desativada, juntamente com o buzzer e o canal 4, e então é feita a transição para o estado RESULTADO.

Por fim, temos o estado citado acima, cuja função será organizar a string a ser escrita no LCD, contendo tanto a mensagem exibida no roteiro, como o valor calculado para o tempo de reação. Assim, segue-se para o reinício de todo o programa após o fim da escrita e a reconfiguração do que é necessário para o início.

5.

Neste projeto, definimos os 7 estados apresentados na máquina de estados e, adicionamos um estado chamado LARGADA_QUEIMADA, responsável por impedir que uma pessoa aperte a botoeira NMI antes do buzzer soar para que não seja possível trapacear no teste apertando subsequentemente a botoeira. O programa pode entrar neste estado no caso de o usuário pressionar a botoeira NMI antes de o buzzer soar, ou seja, enquanto o estado ainda for PREPARA_AUDITIVO ou ESPERA_ESTIMULO_AUDITIVO e, caso o programa entre neste estado, o que ocorre é a exibição de uma mensagem no LCD ("Largada Queimada"), indicando que o usuário queimou a largada e, em seguida o estado é alterado para LEITURA. O que se afirma anteriormente pode ser visto nas imagens:

```
if (estado == PREPARA_AUDITIVO || estado == ESPERA_ESTIMULO_AUDITIVO) {
    estado = LARGADA_QUEIMADA;
}
```

Imagem 1: condicional em FTM0_IRQHandler() (ISR.c) responsável pela transição para o estado LARGADA_QUEIMADA;

```
case LARGADA_QUEIMADA:
    GPIO_escreveStringLCD(0, (uint8_t *) "Largada Queimada");
    ISR_EscreveEstado(LEITURA);
    SET_Counter(12);
    TPM_CH_config_especifica(0, 4, 0b0100, TPM0_CNT);
    break;
```

Imagem 2: tratamento do estado LARGADA_QUEIMADA em main.c.

A transição é feita da mesma forma que na transição de RESULTADO para LEITURA configurando o counter de ISR para contar de 12 até 0, e ir para PREPARA_INICIO quando terminar, 3 segundos depois.

Expliquemos então o estado PREPARA_INICIO, ele é definido pelo bloco de main.c

```
case PREPARA_INICIO:
    GPIO_escreveStringLCD(0, (uint8_t *) "Pressione IRQA12");
    GPIO_escreveStringLCD(0x42, (uint8_t *) "                    ");
    ISR_EscreveEstado(INICIO);
    break;
```

no qual é escrita a mensagem “Pressione IRQA12” no LCD e, em seguida uma quantia grande de espaços em branco, para impedir que a mensagem escrita no estado RESULTADO continue a ser mostrada. Por fim, o estado é alterado para INICIO.

Em seguida, temos o estado INICIO, no qual se aguarda o pressionamento da botoeira IRQA12 e, quando ocorre esse pressionamento, o estado é transitado para PREPARA_AUDITIVO, conforme as imagens

ISR.c

```
if (estado == INICIO) {
    TPM_CH_config_especifica(1, 0, 0b0000, 0); // desativa IRQA12
    counter = geraNumeroAleatorio(440, 2200);
    TPM_habilitaInterrupTOF(1);

    /*
     * Ativa botoeira NMI
     */
    TPM_CH_config_especifica(0, 1, 0b0010, 0);

    estado = PREPARA_AUDITIVO;
```

Nesta segunda imagem, temos o bloco dentro da IRQ FTM1_IRQHandler() e, trata do estado INICIO, desativando a botoeira IRQA12 para que seu pressionamento não ocasione nenhuma mudança no fluxo do programa após já ter sido pressionado pela primeira vez no estado INICIO. Em seguida, geramos um número aleatório que é armazenado em *counter*, as interrupcoes por overflow neste TPM são habilitadas, até que, por fim, a botoeira NMI é habilitada e o estado é transitado para PREPARA_AUDITIVO.

Entendamos, então, o estado PREPARA_AUDITIVO. Este estado é transitório e serve apenas para escrever a mensagem " Teste Auditivo " no LCD e transitar para o estado ESPERA_ESTIMULO_AUDITIVO, conforme a imagem

```
case PREPARA_AUDITIVO:
    GPIO_escreveStringLCD(0, (uint8_t *) " Teste Auditivo ");
    ISR_EscreveEstado(ESPERA_ESTIMULO_AUDITIVO);
    break;
```

No entanto, este estado pode transitar para o estado LARGADA_QUEIMADA, caso a botoeira NMI seja pressionada antes da transição para ESPERA_REACAO_AUDITIVA.

Sigamos para a descrição do estado ESPERA_ESTIMULO_AUDITIVO. Este é um estado de espera, responsável por aguardar a transição para o estado ESPERA_REACAO_AUDITIVA, após a ativação do buzzer em FTM1_IRQHandler() (ISR.c). Isso ocorre no tratamento de interrupção causado pelo overflow de TPM1 que completa a quantidade de vezes estipulada para ser o tempo aleatório para que o buzzer comece a soar. Este tempo é determinado em FTM1_IRQHandler() quando o estado é INICIO com o valor determinado pela função *geraNumeroAleatorio*():

```
if (TPM1_STATUS & TPM_STATUS_CH0F_MASK) {

    if (estado == INICIO) {
        TPM_CH_config_especifica(1, 0, 0b0000, 0); // desativa IRQA12
        counter = geraNumeroAleatorio(440, 2200);
        TPM_habilitaInterrupTOF(1);
```

→ Configuração do counter com número aleatório.

Então, ainda sobre ESPERA_ESTIMULO_AUDITIVO, este estado pode transitar para o estado ESPERA_REACAO_AUDITIVA, na IRQ FTM1_IRQHandler(), conforme o seguinte:

```
    if (TPM1_STATUS & TPM_STATUS_TOF_MASK) {

        if (estado == PREPARA_AUDITIVO || estado == ESPERA_ESTIMULO_AUDITIVO) {
            counter--;

            if(!counter) {
                /*
                 * Ativa buzzer
                 */
                TPM_CH_config_especifica(1, 1, 0b1001, 750);

                /*
                 * Configura TPM0_CH4 para contar o numero de periodos completos
                 * Isto é feito com output compare e interrupcoes sempre que a c
                 */
                TPM_CH_config_especifica(0, 4, 0b0100, TPM0_CNT);

                /*
                 * Desabilita interrupcoes por overflow, pois ja terminou o peri
                 */
                TPM_desabilitaInterrupTOF(1);

                // counter = 0; Nao e necessario pois ele ja e igual a zero, cas

                estado = ESPERA_REACAO_AUDITIVA;
            }
        }
```

Nesta imagem, vemos que, caso o counter seja 0, ou seja, tenha sido decrescido o suficiente (já houveram interrupções suficientes), o buzzer é ativado, o canal 4 de TPM0 é configurado para contar o número de períodos completos até que NMI seja pressionada, e as interrupções de overflow são desativadas, fazendo com que esta seja a última causada, dado que ela não é mais necessária no novo estado. Assim, por fim, o estado transita para ESPERA_REACAO_AUDITIVA.

Neste estado, aguarda-se o pressionamento da botoeira NMI, tendo tratamento negligenciado na main, mas feito na IRQ FTM0_IRQHandler(), na qual temos o cômputo do tempo de reação do usuário, conforme o que se segue:

```
if (estado == ESPERA_REACAO_AUDITIVA) {
    /*
     * Computo do tempo de reacao
     */
    uint16_t CT1 = TPM0_C4V;
    uint16_t CT2 = TPM0_C1V;
    float max = TPM0_MOD; // definido como float para forçar divisao nao inteira
    if(CT2 >= CT1)
        tempo_reacao = (counter + ((CT2 - CT1) / max)) * 0.25;
    else
        tempo_reacao = (counter + (((max - CT1) + CT2) / max)) * 0.25;

    /*
     * desabilitar buzzer, NMI e TPM0_CH4
     */
    TPM_CH_config_especifica(1, 1, 0b0000, 0); // buzzer
    TPM_CH_config_especifica(0, 1, 0b0000, 0); // NMI
    TPM_CH_config_especifica(0, 4, 0b0000, 0); // TPM0_CH4

    estado = RESULTADO;
}
```

neste bloco, temos CT1 o valor do contador no momento de início do temporizador e CT2 o valor deste contador (de TPM0) no fim do temporizador (momento em que a botoeira NMI é pressionada). Sendo assim, definimos ainda o float *max* cujo valor é igual ao máximo valor que CNT pode assumir em uma contagem de TPM e, seguimos para o caso em que o valor de CT2 >= CT1, que implica que o resíduo é dado por `((CT2 - CT1) / max))`, conforme a imagem e, no caso em que CT2 < CT1, temos que o resíduo é dado por `(((max - CT1) + CT2) / max))`. Logo, o valor do tempo de reação é calculado conforme as fórmulas presentes na imagem, de acordo com o que é adequado para cada um dos dois casos. Finalmente, desabilitam-se o buzzer, NMI e TPM0_CH4 e, parte-se para o estado RESULTADO.

No estado RESULTADO, temos a exibição do tempo medido, acompanhado pela mensagem devida. Isso ocorre em main.c conforme a imagem e a explicação que segue:

```
case RESULTADO:
    GET_TempoReacao(&tempo_racao);
    ftoa(tempo_racao, buffer_saida_2, 2);
    strcat(buffer_saida_2, " segundos");
    GPIO_escreveStringLCD(0x1, (uint8_t *) buffer_saida_1);
    GPIO_escreveStringLCD(0x42, (uint8_t *) buffer_saida_2);
    ISR_EscreveEstado(LEITURA);
    SET_Counter(12);
    TPM_CH_config_especifica(0, 4, 0b0100, TPM0_CNT);
    break;
```

Aqui, usamos a função GET_TempoReacao() definida em ISR.c para obter na variável *tempo_racao* o valor de *tempo_reacao* declarado como estática em ISR.c

```
void GET_TempoReacao(float *tempo) {
    *tempo = tempo_reacao;
```

Assim, usamos esse valor na função *ftoa* para obter uma string contendo os caracteres correspondentes ao valor numérico do tempo de reação do usuário. Desse modo, concatenamos essa string à string buffer_saida1

```
char buffer_saida_1[25] = {'R', 'e', 'a', 0x01, 0x02, 'o', ' ', 'e', 'm', ' ', ' ',' ', ' ', ' ', ' ', ' ', ' ', ' ', '\0'};
char buffer_saida_2[25] = "";
```

E, por fim, concatenamos ambas a string buffer_saida2 à buffer_saida1, escrevendo, finalmente, no LCD a string buffer_saida1, cujo formato termina como “Reação em XXX.XX segundos”. Note que, ao invés de seguirmos veementemente o roteiro, optamos por imprimir duas casas decimais, uma vez que a precisão, dessa maneira, seria maior, considerando que a maioria das pessoas de nossa turma, por exemplo, teriam valores de tempo de reação próximos de 0,2, o que implicaria que a aproximação para uma casa decimal geraria uma igualdade entre todos, sacrificando o propósito comparativo do programa que inferimos que deveria ser preservado.

Em sequência, transitamos para o estado LEITURA, responsável por transitar para o estado PREPARA_INICIO e, desativar o canal 4 de TPM0 e ativar IRQA12. Conforme se observa na imagem do bloco contido em FTM0_IRQHandler():

```
if (estado == LEITURA) {
    counter--;

    if(!counter) {
        estado = PREPARA_INICIO;
        TPM_CH_config_especifica(0, 4, 0b0000, 0); // desativa canal
        TPM_CH_config_especifica(1, 0, 0b0010, 0); // ativa botoeira IRQA12
    }
}
```

Aqui, counter é decrescido de 12 até 0, sempre que um período completo se completa desde a entrada neste estado. Este número é configurado na main, na hora da transição de estado, mas a variável de interesse, *counter* se encontra no arquivo ISR.c, para alterar o valor, criamos a função *SET_counter*, que verifica se o estado é de LEITURA, o único em que faz sentido a escrita em *counter* direta pela main e então realiza a operação.

Vale notar que alguns detalhes do programa foram alterados na versão final, em relação a algumas das imagens acima, de forma que o código final fosse mais sucinto e simples de entender. Uma mudança em destaque foi a do funcionamento da função *GET_TempoReacao*, no printscreen acima a leitura é feita passando um parâmetro de saída. Isto foi feito pois vimos com o *debugger* que utilizar a função para retornar um *float* resultava em um valor aparentemente errado. Porém ao testarmos novamente com todo o código já em funcionamento vimos que nada mudou, e portanto o erro de leitura era puarmente da visualização do *debugger*. Na versão final, o valor do tempo_reacao em ISR.c é resgatado pela main no retorno da função, e não por parâmetro de saída.