UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

# Relatório do Lab 2

## Disciplina de Sistemas Embarcados – Prof. Douglas Renaux

---

Autores: João Felipe Sarggin Machado e Waine Barbosa de Oliveira Junior

**Versão:** 29-Ago-2022

# 1 Introdução

Para o desenvolvimento de uma aplicação embarcada é muito comum o uso de sensores e atuadores. Esses podem ser desde botões e leds até sensores de pH e motores. Tão importante quanto o funcionamento de tais é o tempo em que as ações são feitas, seja medir ou atuar, o que justifica o uso de RTOS (*real time operating system)* para tais aplicações.

Com a proximidade do software e hardware nas soluções embarcadas, os atuadores e sensores usualmente são lidos controlados e lidos diretamente por registradores, I/O ou ferramentas dedicadas para tal. Tal processo requer uma configuração prévia para uso, dependente de plataforma.

Em conjunto com essa proximidade, o requisito de *real time* força que tais aplicações tenham que reagir rapidamente a mudanças no sistema, fazendo medidas ou ajustes em intervalos da ordem de microssegundos de diferença. Logo é de extrema importância que as ações sejam tomadas o mais rápido possível. Por isso, para o tratamento de operações é comum o uso de interrupções, as quais interrompem a execução do código assim que o evento ocorre e tratam tal evento.

Nesse relatório é descrito o desenvolvimento de uma aplicação com o objetivo de englobar vários conceitos comuns em sistemas embarcados. O problema consiste num jogo em que um LED é aceso e então o jogador deve clicar o mais rápido possível num botão. Após isso, o jogador é informado do intervalo de tempo entre o acendimento e o clique.

O experimento tem como objetivo introduzir os conceitos e a aplicação de interrupções, leitura e tratamento de entradas (botão) por meio de interrupções, e o controle de atuadores (LED). Tudo isso no ambiente IAR 9.30 utilizando a placa Tiva EK-TM4C1294XL e o TivaWare.

# 2 Planejamento das fases do processo de desenvolvimento

O desenvolvimento da solução foi dividido em fases, para um melhor gerenciamento com relação às tarefas, permitindo uma definição de ordem e dependências entre elas.

As fases definidas foram:

1. Definição do problema a ser resolvido.

2. Especificação da solução.

3. Estudo da plataforma de HW: placa Tiva e seu processador.

4. Estudo da plataforma de SW: TivaWare.

5. Projeto (design) da solução.

6. Identificação (e entendimento) da funcionalidade do TivaWare e do HW que serão utilizadas na solução.

7. Configuração do projeto na IDE: IAR Embedded Workbench for ARM.

8. Edição do código da solução.

9. Teste e depuração.

10. Escrita do relatório.

Essas foram definidas tal qual a sugestão feita pelo professor. Vale ressaltar que, apesar da escrita do relatório ser listada em último, ela foi feita em conjunto com cada parte da execução (por exemplo, a seção sobre *hardware* foi escrita logo após o estudo sobre tal).

# 3 Definição do problema a ser resolvido

O problema a ser resolvido pode ser visto como um jogo em que o jogador tem que clicar num botão o mais rápido possível após um LED ser aceso.

Um tempo após o sistema ser ligado um LED é aceso. O jogador deve apertar o botão o mais rápido possível após isso. Então é mostrado em tela o intervalo de tempo que o jogador demorou para clicar no botão. Quanto menor o intervalo, melhor.

# 4 Especificação da solução

Para solução do problema, foram definidos os seguintes requisitos funcionais e não funcionais, como sugerido pelo professor:

Os **requisitos funcionais** são:

RF1 - O jogo deve ligar o LED D1 para informar o jogador do início da contagem de tempo.

RF1.1 - O LED deve ser aceso até 1 segundo após o início da operação da placa.

RF2 - O jogo usa o botão SW1 para a entrada de dados pelo usuário.

RF3 - O jogo deve apresentar a contagem de tempo no Terminal do IAR indicando o número de clocks entre o LED acender e o botão SW1 ser pressionado e o valor de tempo correspondente em milissegundos.

Os **requisitos não funcionais** são:

RNF1 - O limite superior de contagem de tempo é o equivalente a 3 segundos.

RNF2 - Usar funções da TivaWare para acesso a I/O, SysTick e temporização.

RNF3 - A solução deve fazer uso de interrupções, obrigatoriamente de GPIO e opcionalmente do SysTick.

RNF4 - O vetor de exceções deve estar em memória Flash e não na RAM.

Algumas outras considerações a serem feitas:

- O ambiente de desenvolvimento utilizado é o IAR Embedded Workbench for ARM 9.30.1 no Windows 10
- A placa utilizada é a Tiva EK-TM4C1294XL
- São utilizados os drivers e funções do TivaWare

# 5 Estudo da plataforma de HW

A placa Tiva TM4C1294NCPDTI usa um cristal de 25 MHz como clock interno principal do circuito, porém, muitas aplicações usam o PLL (Phase-locked loop) interno para multiplicar esse clock para frequências mais altas de até 120 MHz para a temporização do core e dos periféricos.

Os botões de usuário (User Switch 1 e User Switch 2) se encontram no GPIO Port J (Respectivamente nos pinos PJ0 e PJ1), necessitam de um resistor de Pull Up (que pode ser ativado internamente) e possuem lógica negativa, ou seja, quando estão em repouso, seu nível lógico é alto (1) e quando estão pressionados, seu nível lógico é baixo (0).

Já os leds de usuário (User Led 1:4), os dois primeiros (1 e 2) se encontram no GPIO Port N (Respectivamente nos pinos PN1 e PN0), enquanto que os dois ultimos (3 e 4) se encontram no GPIO Port F (Respectivamente nos pinos PF4 e PF0). Possuem lógica positiva, ou seja, quando estão acesos seu nível lógico é alto (1) e quando estão apagados, seu nível lógico é baixo (0).

O Systick é um contador decrescente simples de 24 bits que gera uma interrupção quando seu valor é zerado, reiniciando a contagem automaticamente para o valor em que foi

configurado. Seu valor de contagem máximo é N+1 para o valor que for escrito no registrador "SysTick Reload Value Register (STRELOAD)", ou seja $2^{24}$-1+1 = 16777216 pulsos de clock, que para um clock configurado de 120 MHz, representa um periodo de tempo de aproximadamente $(2^{24})/(1.2 * 10^8) \cong 140$ milisegundos.

Por fim, os temporizadores programáveis de uso geral estão dispostos em blocos de dois contadores de 16 bits (contador A e contador B) que podem ser configurados para operar de maneira distinta ou de modo concatenado, criando assim um contador de 32 bits. No modo de 16 bits, os contadores também podem fazer uso de um prescaler de 8 bits.

Há diversas opções de modos de operação para esses temporizadores: Disparo único, periódico, RTC (Real-Time Clock), PWM (Pulse Width Modulation), contagem de borda de entrada ou captura de tempo. Neste laboratório o modo de disparo único em 32 bits foi o utilizado, onde as principais configurações incluem se o contador deve contar para cima ou para baixo, em qual valor a contagem deve começar, e o valor de correspondência para o qual uma interrupção deve ser gerada quando o contador atingir tal valor.

# 6 Estudo da plataforma de SW

Para configurar o clock do processador será utilizado a função *SysCtlClockFreqSet()*, passando as constantes de configuração *SYSCTL_XTAL_25MHz* (Usará o cristal externo de 25 MHz), *SYSCTL_OSC_MAIN* (indica que será utilizado um cristal externo como fonte de oscilação), *SYSCTL_USE_PLL* (indica que a fonte do clock do sistema será a saída do PLL) e *SYSCTL_CFG_VCO_240* (indica que a saída do PLL VCO será de 240 MHz ou o divisor mais próximo que seja capaz de suportar, nesse caso 120 MHz). Já o valor requisitado de clock será o de 120 MHz.

As configurações do SysTick poderão incluir as seguintes funções:

*SysTickPeriodSet*() - Para definir o período do contador SysTick.

*SysTickIntEnable*() - Para habilitar as interrupções do SysTick.

*SysTickEnable*() - Para iniciar o funcionamento do SysTick.

*SysTickDisable*() - Para desativar o funcionamento do SysTick.

As configurações do GPIO poderão incluir as seguintes funções:

*GPIOPinTypeGPIOInput*() - Para configurar um pino como entrada GPIO.

*GPIOPinTypeGPIOOutput* - Para configurar um pino como saída GPIO.

*GPIOPadConfigSet*() - Será útil para configurar o resistor de pull-up do botão.

*GPIOIntTypeSet*() - Será útil para configurar o interrupção em borda de subida para o botão.

*GPIOIntEnable*() - Para habilitar as interrupções em um determinado pino no GPIO.

As configurações do Temporizador poderão incluir as seguintes funções:

*TimerConfigure*() - Para configurar o modo de operação de um temporizador.]

*TimerMatchSet*() - Para definir o valor de correspondência do temporizador (Definir o limite superior de contagem para o qual uma interrupção deverá ser gerada).

*TimerIntEnable*() - Para habilitar a interrupção do timer dado uma configuração específica. Neste laboratório será utilizado o TIMER_TIMA_MATCH, que indica que uma interrupção deverá ser gerada quando o valor do temporizador atingir o valor de correspondência definido na função TimerMatchSet().

*TimerValueGet*() - Para obter o valor atual do temporizador.

*TimerEnable*() - Para iniciar a contagem do temporizador.

*TimerDisable*() - Para interromper a contagem do temporizador.

Além dessas funções, será necessário o uso da função SysCtlPeripheralEnable(), para habilitar os periféricos GPIO Port J (Botão do usuário 1), GPIO Port N (Led do usuário 1) e TIMER 0 (Temporizador de uso geral).

Por fim, será garantido que o vetor exceções estará na memória flash por meio de uma configuração presente no arquivo Tiva.icf, que é um arquivo de configuração para as regiões de memória. Nele há uma configuração que coloca o vetor de interrupções no início da flash:

"*place at start of FLASH { readonly section .intvec };*".

Adicionalmente, não será chamado nenhuma função do tipo IntRegister(), pois essas funções criam uma cópia do vetor de interrupções presente na flash para a RAM, inserindo nela o endereço da função de tratamento que foi passado para essa função. A habilitação das interrupções será feita por meio da função *IntEnable*(), que exige que referência para a função que tratará a interrupção seja escrita na tabela de vetores de interrupção estática, inicializada no arquivo *startup_ewarm*.

# 7 Projeto (design) da solução

Todas as funções identificadas acima serão utilizadas.

A solução implementada terá três fases:

1) Inicialização dos periféricos.
2) Espera pelas interrupções.
3) Apresentação dos resultados.

A etapa 1 terá 5 sub etapas:

1.1) Ajustar a frequência de clock do sistema.

1.2) Configurar o botão de usuário SW1.

1.3) Configurar o led de usuário D1.

1.4) Configurar o temporizador 0A.

1.5) Configurar e iniciar o temporizador Systick.

A etapa 2 terá 3 sub etapas que se apresentam na forma de tratamento de interrupções:

2.1) Tratar a interrupção do Systick.

2.2) Tratar a interrupção do estouro do temporizador.

2.3) Tratar o pressionamento do botão SW1.

Por fim, a etapa 3 terá apenas uma sub etapa.

3.1) Espera ocupada no valor do temporizador.

Descrevendo cada uma das sub etapas:

1.1) A frequência de clock será definida para um alvo de 120 MHz, utilizando o cristal externo de 25 MHz como fonte de oscilação, e ampliando af frequência máxima por meio do uso do PLL e do VCO de 240 MHZ:

SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ | SYSCTL_OSC_MAIN | SYSCTL_USE_PLL | SYSCTL_CFG_VCO_240), 120000000);

1.2) Será ativado o GPIO Port J:

SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOJ);

O pino 0 será configurado para entrada:

GPIOPinTypeGPIOInput(GPIO_PORTJ_BASE, GPIO_PIN_0);

Um Resistor de Pull Up para 2 mA será ativado:

GPIOPadConfigSet(GPIO_PORTJ_BASE, GPIO_PIN_0, GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU);

O tipo de interrupção será definida para borda de subida:

GPIOIntTypeSet(GPIO_PORTJ_BASE, GPIO_PIN_0, GPIO_RISING_EDGE);

E as as interrupções habilitadas para o pino 0:

GPIOIntEnable(GPIO_PORTJ_BASE, GPIO_INT_PIN_0);

1.3) Será ativado o GPIO Port N:

SysCtlPeripheralEnable(SYSCTL_PERIPH_GPION);

O pino 1 será configurado para saída:

GPIOPinTypeGPIOOutput(GPIO_PORTN_BASE, GPIO_PIN_1);

1.4) Será ativado o TIMER 0:

SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);

O temporizador será ajustado para disparo único para cima:

TimerConfigure(TIMER0_BASE, TIMER_CFG_ONE_SHOT_UP);

O tempo de referência para a interrupção será definida para 3 segundos:

TimerMatchSet(TIMER0_BASE, TIMER_A, 3_SEGUNDOS);

A interrupção por comparação de tempo será habilitada:

TimerIntEnable(TIMER0_BASE, TIMER_TIMA_MATCH);

1.5) O Systick será ajustado para contar na sua capacidade máxima:

SysTickPeriodSet(MAX_SYSTICK);

A interrupção do SysTick será habilitada:

SysTickIntEnable();

O funcionamento do SysTick será habilitado:

SysTickEnable();

2.1) O SysTick será desabilitado:

```
SysTickDisable();
```

O Led D1 será aceso:

```
GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_1, GPIO_PIN_1);
```

O temporizador 0A será ativado:

```
TimerEnable(TIMER0_BASE, TIMER_A);
```

As interrupções do botão SW1 serão ativadas:

```
GPIOIntEnable(GPIO_PORTJ_BASE, GPIO_INT_PIN_0);
```

2.2 e 2.3) Essas duas etapas possuirão a mesma implementação, pois se trata do término do jogo de reação, que pode acabar de duas maneiras: 1) Estourando o limite superior de contagem de 3 segundos, então o tempo imprimido será 3000 ms. 2) Apertando o botão SW1 antes do limite superior de contagem de 3 segundos, então o tempo imprimido será menor que 3000 ms.

O temporizador será desativado (congelar a contagem):

```
TimerDisable(TIMER0_BASE, TIMER_A);
```

O valor da contagem do temporizador 0 será anotado em uma variável global:

```
timerValue = TimerValueGet(TIMER0_BASE, TIMER_A);
```

O Led de usuário D1 será apagado:

```
GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_1, 0x0);
```

As interrupções o botão SW1 sertão desativadas:

```
GPIOIntUnregister(GPIO_PORTJ_BASE);
```

As interrupções para o temporizador 0 serão desativadas:

```
TimerIntUnregister(TIMER0_BASE, TIMER_A);
```

3.1) Para evitar o uso da função printf dentro de uma interrupção, o fim de jogo será detectado por meio de uma espera ocupada na variável global que armazena a contagem do temporizador 0. O valor 0 (zero) será atribuída a essa variável antes da espera ocupada como valor de referência:

```
timerValue = 0;
```

E uma espera ocupa será feita nesta variável:

```
while(1){ if(ui32TimerValue) { // Imprime valores. } }
```

Por fim, os respectivos valores em ciclos de clock e em milissegundos serão apresentados:

```
printf("%d clocks\n", timerValue );
printf("%d ms\n", timerValue /(SysClock/1000));
```

# 8 Configuração do projeto na IDE (IAR).

Nenhuma configuração foi modificada em relação ao template base do projeto do laboratório 2 disponibilizado pelo professor via GitHub.

As configurações básicas incluem a seleção do dispositivo “Texas Instruments TM4C1294NCPDT”, o “VFPv4 single precision” como versão da unidade de ponto flutuante (FPU) e a configuração compacta (“Normal”) da biblioteca de tempo de execução C/C++14 (aonde uma ponto importante a se observar é a falta de suporte à descritor de arquivo, o que pode causar problemas na chamada da função printf dentro das interrupções, algo que naturalmente não é recomendado, mas com um suporte completo - “Full” - poderia ser feito).

As formatações das funções “printf” e “scanf” estão no modo automático (compilador irá ajustar conforme necessidade encontrada). A seleção de Heap também está no modo automático.

Já o compilador está definido para a linguagem C, com o dialeto Standard C e em conformidade com a norma padrão com extensões do IAR. A otimização está definida para o nível máximo, priorizando o tamanho do código de saída.

# 9 Teste e depuração.

Para garantir que a frequência de clock foi ajustada para o valor planejado de 120 MHz, planeja-se um teste onde o temporizador 0 será configurado para gerar uma interrupção quando sua contagem atingir $120.10^6$ (120 milhões) e no tratamento dessa interrupção será desligado o LED D1 e o valor atual do temporizador registrado em uma variável global para ser imprimido futuramente. Antes de habilitar a contagem do temporizador 0, o LED D1 será ligado e espera-se que 1 segundo depois (já que o valor de contagem é igual ao valor da frequência de clock) o LED D1 seja desligado pelo tratamento da interrupção e que um valor próximo a $120.10^6$ seja impresso. Adicionalmente uma câmera de celular poderá ser utilizada para gravar a 30 quadros por segundo a realização desse teste e espera-se a contagem de aproximadamente 30 quadros com o LED D1 acesso.

O teste do requisito não funcional RNF 1 se dará da mesma maneira que o teste descrito acima, podem dessa vez o temporizador será configurado para contar até $360.10^6$ (360 milhões) e espera-se que 3 segundos após o acionamento do temporizador o LED D1 seja apagado pelo tratamento da interrupção. Também espera-se que o valor registrado na variável global responsável por armazenar a contagem do temporizador quando o tratamento de interrupção for chamado seja de aproximadamente $360.10^6$. Analogamente se espera uma contagem de 90 quadros no teste da câmera.

O teste dos requisitos não funcionais RNF 1 e RNF2 são imediatos e sua garantia é intrínseca ao cumprimento do design da solução, ou seja, já está previsto o uso das interrupções, do GPIO e do SysTick por meio das funções da TivaWare.

O teste do requisito não funcional RNF 4 será feito por meio do painel Disassembly do IAR, onde espera-se que ao consultar a zona de memória da Flash, encontre-se o vetor de interrupções (__vector_table) escrito no início dela (A partir do endereço 0x0).

Quanto aos requisitos funcionais, o requisito RF1 será testado ativando-se LED D1 no tratamento de interrupção do SysTick com valor de contagem máxima ($2^{24}$ ciclos de clock) e espera-se, visualmente, que o LED D1 seja acesso cerca de 140 milissegundos após o início de operação da placa.

Já o requisito funcional RF2 será testado escrevendo-se uma interrupção para o botão de usuário SW1 em que o LED D1 seja ligado caso esse botão seja pressionado.

Por fim, o requisito funcional RF3 será garantido em um teste final onde o LED D1 será acesso no tratamento da interrupção do SysTick, juntamente com o início da contagem do temporizador 0. Espera-se que ao pressionar o botão de usuário SW1, uma interrupção seja chamada e anote em uma variável o valor atual do temporizador. Esse valor será imprimido

posteriormente por uma função printf, e espera-se um valor condizente com o tempo decorrido entre o LED D1 acender e o botão SW1 ser pressionado.