LABORATÓRIO TPSE I

Prática 05: Implementando Interrupção para GPIO e Timers

Prof. Francisco Helder

23 de agosto de 2024

1 Funcionalidade Interrupts

Uma interrupção pode ser usada para sinalizar ao processador todos os tipos de eventos - por exemplo, porque os dados chegaram e podem ser lidos, um usuário acionou uma chave ou um determinado período de tempo se passou.

As interrupções permitem que os desenvolvedores separem as operações de tempo crítico do programa principal para garantir que sejam processadas de maneira priorizada. Como as interrupções são eventos assíncronos, elas podem ocorrer a qualquer momento durante a execução do programa principal.

Logo, uma interrupção é o mecanismo que o processador usa para pausar o que está fazendo e atender um objetivo paralelo. Esse objetivo pode ser algo como "oh, ei, alguém apertou o botão. Vamos ver o que é". Ou a busca paralela pode ser a troca de contexto entre diferentes threads. Existem 128 eventos de interrupção exclusivos que podem ocorrer na Beaglebone Black.

Vamos usar interrupções para receber a ação de clicar no botão e contagem cronológica, e assim realizar alteração na lógica do seu programa.

1.1 Interrupt Vector Table

Uma tabela de vetores de interrupção é uma lista de instruções e geralmente fica como as primeiras linhas de código em seu aplicativo. O processador é codificado para chamar a enésima instrução para um evento de interrupção específico. Aqui está um exemplo de tabela de vetores de interrupção (*Interrupt Vector Table* - **IVT**) em assembly:

```
/* Vector table */
_vector_table:
        ldr   pc, _reset      /* reset - _start           */
        ldr   pc, _undf       /* undefined - _undf        */
        ldr   pc, _swi        /* SWI - _swi               */
        ldr   pc, _pabt       /* program abort - _pabt    */
        ldr   pc, _dabt       /* data abort - _dabt       */
        nop                   /* reserved                 */
        ldr   pc, _irq        /* IRQ - read the VIC       */
        ldr   pc, _fiq        /* FIQ - _fiq               */

_reset: .word _start
_undf:  .word 0x4030CE24 /* undefined               */
_swi:   .word 0x4030CE28 /* SWI                     */
_pabt:  .word 0x4030CE2C /* program abort           */
_dabt:  .word 0x4030CE30 /* data abort              */
nop
_irq:   .word 0x4030CE38  /* IRQ                     */
_fiq:   .word 0x4030CE3C  /* FIQ                     */
```

Dado o **IVT**, você pode implementar essas diferentes funções para lidar com cada operação de interrupção respectiva.

Para configurarmos nossa própria tabela de vetores de interrupção, nós copiaremos o endereço da função que queremos invocar para um ponto específico da memória. Essa abstração significa que seu código pode ser executado em qualquer lugar e você ainda poderá configurar o **IVT**. De acordo com a seção 26.1.4.2 do manual de referência técnica, podemos armazenar o endereço de nosso manipulador **Handler** nos locais apresentados na Figura 1:

Table 26-3. RAM Exception Vectors

Address	Exception	Content
4030CE00h	Reserved	Reserved
4030CE04h	Undefined	PC = [4030CE24h]
4030CE08h	SWI	PC = [4030CE28h]
4030CE0Ch	Pre-fetch abort	PC = [4030CE2Ch]
4030CE10h	Data abort	PC = [4030CE30h]
4030CE14h	Unused	PC = [4030CE34h]
4030CE18h	IRQ	PC = [4030CE38h]
4030CE1Ch	FIQ	PC = [4030CE3Ch]
4030CE20h	Reserved	20090h
4030CE24h	Undefined	20080h
4030CE28h	SWI	20084h
4030CE2Ch	Pre-fetch abort	Address of default pre-fetch abort handler(1)
4030CE30h	Data abort	Address of default data abort handler(1)
4030CE34h	Unused	20090h
4030CE38h	IRQ	Address of default IRQ handler
4030CE3Ch	FIQ	20098h

Figura 1: Tabela de Vetores de exceção de RAM retirado do manual de referência.

Para demonstrar como alguém pode carregar um endereço para uma função, devemos ir para o código em Assembly. Nosso manipulador de interrupção (Handler) precisará de pelo menos um pouco de assembly para capturar os registradores.

```
/* Startup Code */
_start:
        ...
        /* IRQ Handler */
        ldr r0, =_irq
        ldr r1, =.irq_handler
        str r1, [r0]

        bl main

        b .

.irq_handler:
        stmfd sp!, {r0-r12, lr}
        MRS r11, spsr
        bl ISR_Handler
        dsb
        msr spsr, r11
        ldmfd sp!, {r0-r12, lr}
        subs pc, lr, #4
```

O que esse código faz é carregar dois registradores separados com os valores contidos nas variáveis globais **_irq** e **.irq_handler**. Em seguida, ele armazena o valor de **r1** no endereço contido em **r0**. Com isso, podemos escrever a localização da função **.irq_handler** na tabela de abstração do vetor de interrupção. Sempre que um evento de IRQ ocorrer, o Beaglebone invocará nossa função Assembly.

Um manipulador de interrupção (handler) deve realizar algumas tarefas específicas imedia-

tamente. A primeira coisa que ele deve fazer é armazenar todos os registradores atuais (r0–r12), bem como a instrução branch-back (lr). Isso será efetivamente um snapshot do que estava fazendo antes de ser interrompido. Onde o sistema armazena esses valores, exatamente? Bem, a resposta está no ponteiro da pilha. Precisamos inicializar o stack pointer para que, quando o sistema tentar tirar esses snapshot, tenha um local para armazenar os valores.

1.2 Configurando Interrupção para GPIO

Sequência de Inicialização:

1. Programe o registro MPU_INTC.INTC_SYSCONFIG: Se necessário, habilite o autogating do clock da interface configurando o bit AUTOIDLE.

2. Programe o registro INTC_IDLE: Se necessário, desabilite o autogating funcional do clock ou habilite o autogating do sincronizador configurando o bit FUNCIDLE ou o bit TURBO de acordo.

3. Programe o registro INTC_ILRm para cada linha de interrupção: Atribua um nível de prioridade e defina o bit FIQNIRQ para uma interrupção FIQ (por padrão, as interrupções são mapeadas para IRQ e a prioridade é 0x0 [mais alta]).

4. Programe o registrador INTC_MIRn: Habilita interrupções (por padrão, todas as linhas de interrupção são mascaradas).

NOTE: Para programar o registro INTC_MIRn, os registros INTC_MIR_SETn e INTC_MIR_CLEARn são fornecidos para facilitar o mascaramento, mesmo que seja possível para compatibilidade com versões anteriores escrever diretamente no registro INTC_MIRn.

Para programar a interrupção no GPIO precisamos definir qual grupo de interrupções vamos utilizar, no processador AM335x temos duas interrupções por módulo de GPIO (GPIOINTXA e GPIOINTXB - com X sendo o modúlo GPIO), que permite rotear dois grupos de pinos GPIO para interrupções separadas, respectivamente ISRs separados que os atendem.

Agora precisamos definir qual registrador INTC_MIR_CLEARn configurar, e para isso, devemos descobrir o número da interrupção do grupo do módulo GPIO escolhido, como exemplo vamos escolher o GPIO módulo 1, e escolher o registrador de grupo de interrupção GPIOINT1A, consultando a tabela “ARM Cortex-A8 Interrupts” na subseção 6.3, temos que o número da interrupção desse registrador é 98, então realizamos um deslocamento de 5 ($98 \gg 5$) para descobrir qual valor de n do registrador MIR_CLEARn, que é 3, logo temos que configurar o registrador INTC_MIR_CLEAR3 (0x4820_00E8), que tem base INTC (0x4820_0000) mais o offset (0xE8).

Agora precisamos saber qual bit será configurado para esse número de interrupção (98 = 0x62 em hexa), e para isso realizamos uma lógica (and) do número da interrupção com o valor (0x1F). Logo, para configurar a interrupção para os GPIOs do módulo 1 no grupo GPIOINT1A temos que setar o bit 2 (0x62&0x1F) do registrador INTC_MIR_CLEAR3, como mostrado na seguinte linha no código depois de configurar o clock do GPIO1:

```
/* Interrupt mask */
HWREG(INTC_MIR_CLEAR3) |= (1<<2); //98->bit 2 on rergister MIR_CLEAR3
```

Para gerar uma solicitação de interrupção para um evento definido (nível ou transição lógica) ocorrendo em um pino GPIO, os registradores de configuração GPIO devem ser programados da seguinte forma:

- As interrupções do GPIO devem ser habilitadas nos registros GPIO_IRQSTATUS_SET_0 e/ou GPIO_IRQSTATUS_SET_1.
- Os eventos esperados no GPIO de entrada para acionar a solicitação de interrupção devem ser selecionados nos registradores GPIO_LEVELDETECT0, GPIO_LEVELDETECT1, GPIO_RISINGDETECT e GPIO_FALLINGDETECT.

Por exemplo, a geração de interrupção em ambas as bordas é configurada definindo como 1 nos registradores GPIO_RISINGDETECT e GPIO_FALLINGDETECT junto com a habilitação de interrupção para uma ou ambas as linhas de interrupção (GPIO_IRQSTATUS_SET_n).

Então para habilitar a interrupção no GPIO gpio1_12 com detecção por borda de subida, precisamos configurar as seguintes linhas de códigos:

```
/* Setting interrupt GPIO pin. */
HWREG(GPIO1_IRQSTATUS_SET_0) |= 1<<12;

/* Enable interrupt generation on detection of a rising edge.*/
HWREG(GPIO1_RISINGDETECT) |= 1<<12;
```

NOTE: podemos habilitar a característica de **debouncing** na detecção do evento, somente setando 1 no bit específico do registardor GPIO_DEBOUNCENABLE.

1.2.1 Rotina de Serviço de Interrupção - ISR

Quando o evento de interrupção ocorrer (nesse caso o botão ser apertado), o processador invoca o tratador de interrupção, que no nosso sistema é definido por "irq_handler", que por sua vez chamada a rotina de serviço de interrupção (ISR), representado pela função **ISR_Handler** no código em C. quando e ISR é executada, primeiro se verifica qual tipo de interrupção (definido na tabela 6.1 da subseção 6.3 do manual técnico), para isso devemos fazer uma operação lógica (and) do registrado INTC_SIR_IRQ com 0x7F (em que os seis primeiros bits 0-6 definem o número da interrupção ativa). Com isso tomar alguma decisão e em seguida reconhecer a IRQ, como mostrado nas linhas de código abaixo:

```
/* verify active IRQ number */
unsigned int irq_number = HWREG(INTC_SIR_IRQ) & 0x7f;

if(irq_number == 98)
        gpioIrqHandler();

/* acknowledge IRQ */
HWREG(INTC_CONTROL) = 0x1;
```

na função "gpioIsrHandler" deve-se limpa o status dos flags de interrupção e também realizar alterações necessárias para ser utilizado em seu código, como visto na linha de código abaixo:

```
/* Clear the status of the interrupt flags */
HWREG(GPIO1_IRQSTATUS_0) = 0x1000;

flag_gpio = true;
```

Que nesse caso é mudado a flag de GPIO para alterar a sequência do pisca LEDs.

1.3 Configurando Interrupção para Timer

Sequência de Inicialização:

1. Programe o registro MPU_INTC.INTC_SYSCONFIG: Se necessário, habilite o autogating do clock da interface configurando o bit AUTOIDLE.
2. Programe o registro INTC_IDLE: Se necessário, desabilite o autogating funcional do clock ou habilite o autogating do sincronizador configurando o bit FUNCIDLE ou o bit TURBO de acordo.
3. Programe o registro INTC_ILRm para cada linha de interrupção: Atribua um nível de prioridade e defina o bit FIQNIRQ para uma interrupção FIQ (por padrão, as interrupções são mapeadas para IRQ e a prioridade é 0x0 [mais alta]).
4. Programe o registrador INTC_MIRn: Habilita interrupções (por padrão, todas as linhas de interrupção são mascaradas).

NOTE: Para programar o registro INTC_MIRn, os registros INTC_MIR_SETn e INTC_MIR_CLEARn são fornecidos para facilitar o mascaramento, mesmo que seja possível para compatibilidade com versões anteriores escrever diretamente no registro INTC_MIRn.

Para programar a interrupção do Timer precisamos definir qual grupo de Timer vamos utilizar e então habilitar sua interrupção, no processador AM335x temos uma interrupções por gruo de Timer, que permite rotear a interrupção separadamente, respectivamente ISRs separados que os atendem.

Agora precisamos definir qual registrador INTC_MIR_CLEARn configurar, e para isso, devemos descobrir o número da interrupção do grupo do Timer escolhido, como exemplo vamos escolher o DMTIMER7, consultando a tebela “ARM Cortex-A8 Interrupts” na subseção 6.3, temos que o número da interrupção desse registrador é 95, então realizamos um deslocamento de 5 ($95 \gg 5$) para descobrir qual valor de n do rergistrador MIR_CLEARn, que é 2, logo temos que configurar o registrador INTC_MIR_CLEAR2 (0x4820_00C8), que tem base INTC (0x4820_0000) mais o offset (0xC8).

1.3.1 Função Delay com Interrupção

```
        unsigned int countVal = TIMER_OVERFLOW - (mSec *
           TIMER_1MS_COUNT);

        /* Wait for previous write to complete */
        DMTimerWaitForWrite(0x2);
```

```
        /* Load the register with the re-load value */
        HWREG(DMTIMER_TCRR) = countVal;

        flag_timer = false;

        /* Enable the DMTimer interrupts */
        HWREG(DMTIMER_IRQENABLE_SET) = 0x2;

        /* Start the DMTimer */
        timerEnable();

        while(flag_timer == false);

        /* Disable the DMTimer interrupts */
        HWREG(DMTIMER_IRQENABLE_CLR) = 0x2;
```

Quando habilitamos o TIMER com interrupção, então devemos carregar o registrador TCRR com o valor de tempo que deve ser contado. Com isso habilitamos a interrupção do timer (TIMER7) para que o hardware possa contar e no fim da contagem acionar a interrupção certa, como visto na rotina de interrupção apresentado na próxima subseção. Por fim desabilitamos a interrupção no fim da contagem, como vista no código acima.

1.3.2 Rotina de Serviço de Interrupção - ISR

Quando o evento de interrupção ocorrer (nesse caso o botão ser apertado), o processador invoca o tratador de interrupção, que no nosso sistema é definido por "irq_handler", que por sua vez chamada a rotina de serviço de interrupção (ISR), representado pela função **ISR_Handler** no código em C. quando e ISR é executada, primeiro se verifica qual tipo de interrupção (definido na tabela 6.1 da subseção 6.3 do manual técnico), para isso devemos fazer uma operação lógica (and) do registrado INTC_SIR_IRQ com 0x7F (em que os seis primeiros bits 0-6 definem o número da interrupção ativa). Com isso tomar alguma decisão e em seguida reconhecer a IRQ, como mostrado nas linhas de código abaixo:

```
        /* verify active IRQ number */
        unsigned int irq_number = HWREG(INTC_SIR_IRQ) & 0x7f;

        if(irq_number == 95)
                timerIrqHandler();

        /* acknowledge IRQ */
        HWREG(INTC_CONTROL) = 0x1;
```

na função "timerIsrHandler" deve-se limpa o status dos flags de interrupção e também realizar alterações necessárias para ser utilizado em seu código, como visto na linha de código abaixo:

```
    /* Clear the status of the interrupt flags */
        HWREG(DMTIMER_IRQSTATUS) = 0x2;

        flag_timer = true;

        /* Stop the DMTimer */
        timerDisable();
```

Que nesse caso é mudado a flag de TIMER para alterar a sequência do pisca LEDs.

1.4 Circuito Completo

Nessa prática você irá demonstra o conhecimento adquirido na disciplina, durante as práticas realizada, que envolve os procedimentos para configurar pino de GPIO (general purpose input/output) como entrada e saída, e no caso do pino de entrada utilizar o conhecimento de interrupção (nesse caso usando o pino 60). Por final realizar a configuração do modulo de TIMER, para criar uma função Delay com valores de tempos passado pelo usuário, num menu.

1.5 Circuito Completo

Para conectar os botões, e os LEDs na BeagleBone, siga estes passos e consulte o diagrama para o circuito mostrado na Figura 2.

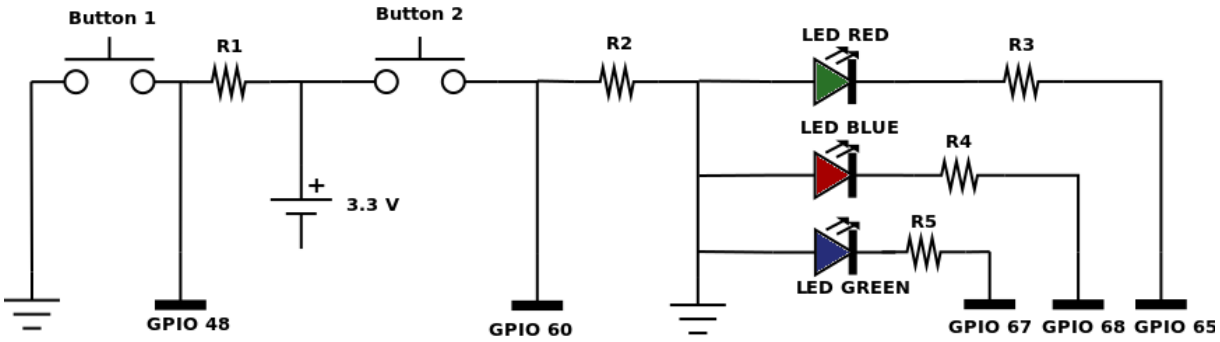

Figura 2: Valor do tempo no registrador TCRR.

Para conectar botões e LEDs para o BeagleBone, construa na protoboard um circuito resistivo acima. Em seguida faça os próximos passos para configurar o sistema completo:

1. **Configure o terra**: Conectar o pino GND do BeagleBone, por exemplo, pinos 1 e 2 em ambos os expansores - a faixa negativa da protoboard.

2. **Ligue o pino GPIO 60 para a protoboard**: Este GPIO 60 (gpio1_28) será utilizado como Button_2 (faça um pullDown) e terá a finalidade de alternar uma sequência qualquer de blink dos 3 LEDs, essa configuração deve ser realiza obrigatoriamente com interrupção.

3. **Conecte o GPIO_67, GPIO_68 e GPIO_65 para a protoboard**: Esses GPIOs serão utilizados para realizar uma sequência qualquer de blink definida pelo aluno.

4. **Ligue o pino GPIO_48 para a protoboard**: Este GPIO_48 (gpio1_16) será utilizado como Button_1 (faça um pullUp) e terá a finalidade de mostrar um menu para o usuário escolher qual o tempo em milissegundos da frequência do blink.

2 Atividades Práticas

pratica 1:

Baseado no código exemplo “gpio_int”, faça uma atualização do seu sistema para receber interrupção de botão.

pratica 2:

Utilizando o seu sistema de timer, faça um sistema que receba informação da uart para alterar em tempo real a frequencia de oscilação do LED.