Insper

# Computação Embarcada - IOs (Input/Output)

Rafael Corsi - rafael.corsi@insper.edu.br

Março - 2018

# Projeto

Copie o projeto localizado em **Computacao-Embarcada/Codigos Exemplos/LCD-EXT2/** para a pasta do seu repositório da aula **06-PIO-IRQ**.

## Entrega Parcial :

Ao final da aula, todos os alunos devem fazer um push do status atual do código. Iremos fazer uma avaliação parcial desse código.

1. Porte a parte do código da aula 3 que lê o botão e aciona o LED para o projeto recém copiado.
2. Migre o projeto para usar interrupção e sleep mode como descrito no handout a seguir.

## Entrega final (próxima segunda)

Devemos ter dois botões externo a placa que irão configurar a frequência na qual o LED irá piscar. Um dos botões irá aumentar a frequência do piscar do LED e o outro irá diminuir a frequência que o LED irá piscar. O LCD deverá exibir a frequência atual do led.

Deve-se apresentar junto com o firmware um diagrama de blocos que especifica quais periféricos e pinos foram usados no projeto.

# Handout

Copie o projeto localizado em **Computacao-Embarcada/Codigos Exemplos/LCD-EXT2/** para a pasta do seu repositório da aula **06-PIO-IRQ**.

## Definindo a função de callBack

Devemos definir uma função de callBack que será chamada sempre que acontecer uma mudança no botão. Faça o LED piscar nessa função.

```
void but_callBack(void){...}
```

Insira e implemente essa função no código.

## Configurando o PIO

Para configurarmos o PIO para gerar interrupção é necessário configurarmos duas parte distintas do uC. A primeira é o próprio PIO e a segunda é o NVIC (parte do CORE ARM responsável por receber e gerenciar interrupções).

### PIO

A Atmel disponibiliza a função pio_enable_interrupt que ativa e configura a interrupção do PIO em uma determinada combinações de pinos.

```
// Enable the given interrupt source.
// The PIO must be configured as an NVIC interrupt source as well.
// p_pio Pointer to a PIO instance.
// ul_mask Interrupt sources bit map.
void pio_enable_interrupt(Pio *p_pio, const uint32_t ul_mask);
```

Use essa função para ativar a interrupçõa no PIO: referente ao botão da placa.

! O PIO gera somente uma interrupção: independente de qual pino do PIO foi ativado o código irá para a função *void PIO_Handler(void)* em questão. Porém a ASF fornece uma camada de abstração (pio_handler.c) que possibilita que uma função seja atribuída por pino, dando a sensação que existe uma interrupção por pino quando na verdade isso é tratamento de software.

### pio_handler

Uma vez ativada a interrupção em um determinado periférico, será necessário configurar o tipo de sinal que dará origem a essa interrupção, esses atributos estao definidos no arquivo */src/ASF/sam/drivers/pio.h* :

- Borda de descida (PIO_IT_FALL_EDGE)
- Borda de subida (PIO_IT_RISI_EDGE)
- Nível alto (PIO_IT_HIGH_LEVEL)
- ...

Note que podemos concaternar mais de um atributo por pino, possibilitando que ele gere uma interrupção por exemplo tanto em borda de descida quanto em borda de subida.

Além do PIO, pino e atributo, a função *pio_handler_set* recebe como parâmetro um ponteiro de função que será chamado sempre que a interrupção em um determinado **pino** ocorrer.

```
// Set an interrupt handler for the provided pins.
// The provided handler will be called with the triggering pin as its parameter
//as soon as an interrupt is detected.
// p_pio PIO controller base address.
// ul_id PIO ID.
// ul_mask Pins (bit mask) to configure.
// ul_attr Pins attribute to configure.
// p_handler Interrupt handler function pointer.
// return 0 if successful, 1 if the maximum
// number of sources has been defined.
int32_t pio_handler_set(Pio *p_pio,
                        uint32_t ul_id,
                        uint32_t ul_mask,
                        uint32_t ul_attr,
                        void (*p_handler) (uint32_t,uint32_t));
```

Use essa função para ativar configurar o tipo de interrupção e sua função de callBack no botão da placa. Passe como referência a função but_callback() simplesmente passando o nome da função no ultimo atributo *but_callBack*.

## NVIC

O Nested Vectored Interrupt Controller (NVIC) é a parte do núcleo ARM que lida com interrupções, nele podemos configurar se uma interrupção está ativa ou não, sua prioridade via duas funções definidas no CMSIS :

```
NVIC_EnableIRQ (IRQn_Type IRQn)
NVIC_SetPriority (IRQn_Type IRQn, uint32_t priority)
```

Nessas funções o **IRQn** é o ID do periférico que está sendo configurado e o **priority** é a prioridade que cada periférico terá nas interrupções suportando até 255 diferentes níveis (depende do uC) sendo 0 o maior valor de interrupção.

Use essas duas funções para ativar a interrupção no NVIC configurando como prioridade 0.

## Pisca Led com interrupção

Agora temos todas as partes do firmware / hardware configuradas para podermos operar o botão com interrupção, termine o código para que quando o botão for pressionado uma interrupção é gerada e o LED pisque.

## IRQ - Keep them short and simple

O tempo que um uC deve ficar na interrupção é o mais rápido possível, não é uma boa boa prática gastar muito tempo dentro de uma interrupção, a interrupção só deve executar códigos críticos o resto deve ser processado no loop principal (while(1)) pelos principais motivos a seguir :

1. Outras interrupções de mesma prioridade irão aguardar o retorno da interrupção. O projeto deixará de servir de maneira rápida a uma interrupção.
2. Nem todas as funções são reentrantes. Funções como printf podem não operar corretamente dentro de interrupções (mais de uma chamada por vez).
3. RTOS : As tarefas devem ser executadas em tasks e não nas interrupções, possibilitando assim um maior controle do fluxo de execução do firmware.

### FLAG

A solução a esse problema é sempre que possível devemos realizar o processamento de uma interrupção no loop principal, essa abordagem é muito utilizada em sistemas embarcados. E deve ser feita da forma a seguir :

```
Bool but_flag = false;

void but_callBack(void){
 but_flag = true;
}

void main(void){
  // ...
  // ...

  while(1){

   // trata interrupção do botão
   if(but_flag){
     // código
     // ..
     // ..
```

```
    // reseta flag
    but_flag = false;
   }

  }
}
```

Modifique o firmware para trabalhar com flag. Note que será interessante criar um nova função chamada de pisca_led(), que será chamada para piscar o LED.

## Low power modes

Trabalhar por interrupção possui duas grandes vantagens : 1. Responder imediato a um evento e 2. Possibilitar o uC entrar em modos de baixo gasto energético (*sleep modes*).

Cada uC possui seus modos de baixo consumo energético, no caso do uC utilizado no curso são 4 modos distintos de operação, cada um com sua vantagem - desvantagem.

- Active Mode

  "Active mode is the normal running mode with the core clock running from the fast RC oscillator, the main crystaloscillator or the PLLA. The Power Management Controller can be used to adapt the core, bus and peripheral frequencies and to enable and/or disable the peripheral clocks."

- Backup mode

  "The purpose of Backup mode is to achieve the lowest power consumption possible in a system which is performing periodic wake-ups to perform tasks but not requiring fast startup time."

- Wait mode

  "The purpose of Wait mode is to achieve very low power consumption while maintaining the whole device in a powered state for a startup time of less than 10 us."

- Sleep Mode

  "The purpose of sleep mode is to optimize power consumption of the device versus response time. In this mode, only the core clock is stopped. The peripheral clocks can be enabled. The current consumption in this mode is application-dependent"

Mais informações na secção 6.6 do datasheet

**ASF**

Para termos acesso as funções da atmel que lidam com o sleep mode devemos adicionar a biblioteca no Atmel Studio :

ASF -> ASF Wizard ->

Agora basta adicionar a biblioteca **Sleep manager (service)** ao projeto.

Agora podemos usar as funções de low power, primeiramente iremos utilizar somente o modo : sleep mode via a chamada da função a seguir :

```
void main(void){
  while(1){
     pmc_sleep(SAM_PM_SMODE_SLEEP_WFI);

     ...

  }
}
```

Uma vez chamada essa função o uC entrará em modo sleep WFI (WaitForInterrupt), essa função age como sendo “blocante” ou seja, execução do código é interrompida nela até que uma interrupção “acorde” o uC.