Insper

Computação Embarcada - Aula 03 - Handout - IOs (Input/Output)

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

Fevereiro 2018

Visão Geral

Nessa aula iremos utilizar um projeto de referência *03-IO-Digital/SAME70-XPLD* que foi criado para ser o mais "clean" possível, inclusive, faltando algumas bibliotecas (ASF) básicas para a compilação.

Abra o projeto no AtmelStudio e verifique o conteúdo do arquivo main.c o mesmo deve estar praticamente vazio salvo a inclusão do arquivo asf.h :

```
#include "asf.h"
```

asf.h

O arquivo *header* **asf.h** é criado e atualizado dinamicamente pelo AtmelStudio e contém os frameworks/drivers inseridos no projeto. O Atmel Software Framework (ASF) é uma camada de abstração do acesso ao hardware, possibilitando que configuremos partes específicas do uC em um nível de abstração intermediário.

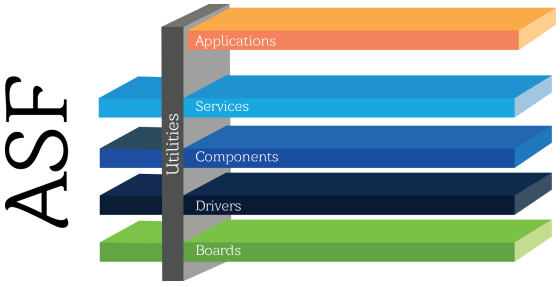

Figure 1: ASF

Esse arquivo não deve ser alterado manualmente por ser atualizado automaticamente pela ferramenta.

int main(void){ ... }

A função main é chamada na inicialização do sistema e deve possuir a configuração do microcontrolador e de seus periféricos. No caso desse exemplo a função main não faz nada.

```
int main(void){
    while (1) {

    }
    return 0;
}
```

Tente compilar o código *BUILD -> BUILD SOLUTION (F7)* o mesmo apresentará erros !

Incluindo dependências no ASF

No Atmel-Studio abra o ASF Wizard clicando em *ASF -> ASF Wizard*, após um tempo (sim demora para abrir) uma janela contendo a esquerda uma lista dos possíveis drivers que podem ser utilizados para o microcontrolador escolhido e na coluna da direita os drivers/bibliotecas já inseridas na solução.

O projeto está configurado para o microcontrolador da placa e com isso o Atmel Studio sabe quais são as bibliotecas disponíveis para o microcontrolador escolhido e irá mostrar somente aquelas que podem ser utilizadas.

As bibliotecas já inseridas no projeto são :

- Generic board support (driver)
  - drivers de compilação para o uC da placa
- System Clock Control (service)
  - funções para controle do clock do uC

No projeto da aula iremos usar as seguintes bibliotecas :

- GPIO - General purpose Input/OutPut (service)
  - funções para configuração do PIO
- IOPORT - General purpose I/O service (service)
  - funções para controle dos pinis
- MPU - Memory Protect Unit (driver)
  - funções para gerenciamento de memória
- PMC - Power Management Controller (driver)

  - funções para configuração do periférico PMC e controle de clock dos periféricos
- PIO - Parallel Input/Output Controller (driver)
  - funções para controle do periférico PIO e controle dos pinos
- Delay routines
  - funções de delay

Passo 1.
Inclua no projeto as bibliotecas listadas anteriormente.

! Adicionar as bibliotecas PMC e PIO é uma redundância que não afeta em nada o projeto, já que as mesmas fazem parte respectivamente da biblioteca System Clock Contol e GPIO.

Para adicionar ou remover bibliotecas da solução utilize a barra inferior :

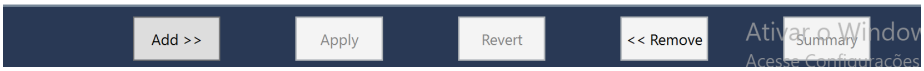

Figure 2: ASF Bar

Adicione as bibliotecas listadas anteriormente, o resultado deve ser :

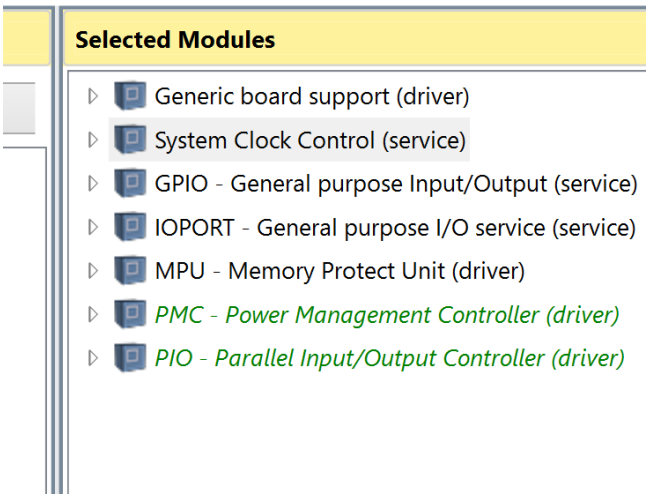

Figure 3: ASF bibliotecas utilizadas

Ao final clique em *APPLY* para salvar as alterações.

MAIN

Passo 2.
Modifique a função main incluindo as seguintes linhas logo na inicialização :

```
int main(void){

    // Initialize the board clock
    sysclk_init();

    // Disativa WatchDog
    WDT->WDT_MR = WDT_MR_WDDIS;
```

A função *sysclk_init()* é responsável por aplicar as configurações do arquivo config/conf_clock.h no clock do microcontrolador. Já a linha WDT->WDT_MR = WDT_MR_MDDIS faz com que o *watch dog* do microcontrolador seja desligado.

! WatchDog como o próprio nome diz é um cão de guarda, responsável por verificar se o código está travado em algum trecho, causando a reinicialização do uC de forma indesejada .

Botão e LED

O kit de desenvolvimento utilizado no curso possui um botão e um led. Abra o manual do kit (manuais/SAME70-XPLD.pdf), na aula de hoje iremos começar usando esses dois periféricos do uC.

! Cuidado com o termo **periférico** ele confunde na maioria das vezes pois depende da referência que estamos tratando.
Dentro do microcontrolador possuímos vários periféricos (PIO/ PMC, Memória, ...), esses são periféricos internos ao uC. Porém podemos chamar também os dispositivos conectados ao uC de periféricos tal como : um mouse USB, um botão, ...

Saída - LED

Vá até a secção **4.4.3 LED** e leia sobre o LED encontrado no kit de desenvolvimento e responda :

Pergunta 1

Como o LED é ativado ?

Pergunta 2

Se colocarmos 0 no pino conectado ao LED ele irá acender ou apagar ?

Pergunta 3

Pela tabela dessa mesma secção preencha :

SAM E70 Pin	Function

O SAM E70 Pin descreve em qual pino e qual PIO será o responsável pelo controle do pino, podemos a partir da informação PC8 extrair as seguintes informações :

O periférico PIO C (existem nesse uC 5 PIOs, cada um controla até 32 pinos) "pino/bit/índice" 8 é responsável por controlar o Liga/Desliga do LED verde da placa.

Passo 3a.
Atualize o #define LED_PIO_PIN para 8.

```
#define LED_PIO_PIN 8
```

As funções ASF pio irão possuir como entrada um parâmetro que indica qual periférico PIO estamos configurando, essa informação é um endereço de memória (periférico mapeado em memória) e pode ser utilizada simplesmente passando o nome do periférico : PIOA, PIOB, PIOC,… .

Para ao longo do código não confundirmos qual PIO é referente a cada entrada, iremos **criar** um novo #define que simplesmente cria um alias para o PIO.

Passo 3b.
Insira no main.c o define a seguir

```
#define LED_PIO PIOC
```

Para isso funcionar deveremos configurar o PIO para tratar esse pino como saída (**um led é saída do uC**). Isso é feito via configuração de uma série de registradores desse periférico (veremos isso na próxima aula técnica).

Ativando periférico PIOC

Antes de configurarmos o PIOC devemos "ligar" esse periférico, ligar no caso é ativarmos o clock. Essa função é de responsabilidade do Power Management Controller (PMC), que é o periférico responsável pela gestão de energia e clock de todo uC.

A função que iremos utilizar para isso é a :

```
uint32_t pmc_enable_periph_clk(uint32_t ul_id)
```

Essa função faz parte do pacote de funções adicionadas quando importamos o PMC no ASF Wizard. A função possui como parâmetro de entrada o ID do periférico a ser ativado o clock e retornar 0 em caso de sucesso e 1 em caso de algum erro de configuração.

Periférico ID

Cada periférico possui um ID único, sendo essa a maneira de indicar ao PMC qual periférico estamos realizando a operação. No manual do uC (SAME70.pdf) vá até a secção **13.1 Peripheral Identifiers** onde é possível encontrar uma relação do periférico e seu ID.

> 13.1 Peripheral Identifiers
>
> Table 13-1 defines the peripheral identifiers of the SAM E70. A peripheral identifier is required for the control of the peripheral interrupt with the Nested Vectored Interrupt Controller and **control of the peripheral clock with the Power Management Controller.**

Pergunta 4

Qual ID do periférico PIOC ?

Periférico	ID
PIOC	

Passo 3b.
Atualize no main.c o **#define LED_PIO_ID** com o ID do PIOC preenchido na tabela anterior.

```
#define LED_PIO_ID xxxx
```

Agora que já sabemos qual o ID do periférico que desejamos ativar, podemos inserir no main a função pmc_enable_periph_clk passando como parâmetro o define recém declarado

Passo 4.
Ative o clock do PIOC via chamada da função pmc_enable_periph_clk(uint32_t ul_id), sendo o parâmetro da função o valor de ID recém encontrado (usar o LED_PIO_ID):

```
int main(void){
...

// Ativa o periférico responsável pelo controle
// do LED
pmc_enable_periph_clk(LED_PIO_ID);

...
```

! A configuração de periféricos deve ser feita fora do while(1), já que nesse caso uma vez configurado como saída o pino sempre será uma saída.

Configurando como saída

Uma vez ativado devemos configurar o pino como saída, cada PIO consegue gerenciar independente até 32 pinos (com qualquer combinação de entrada/ saída). Para isso iremos utilizar agora a biblioteca do PIO e mais especificamente a função pio_configure() que possui a seguinte declaração :

```
uint32_t pio_configure(Pio *p_pio,
                       const pio_type_t ul_type,
                       const uint32_t ul_mask,
                       const uint32_t ul_attribute)
```

Onde :

- Pio *p_pio : é um ponteiro para o PIO em questão
- pio_type : é como o PIO será configurado :
  - PIO_OUTPUT_0 : como saída padrão
  - PIO_INPUT : como entrada padrão

- ul_mask : uma máscara de qual pino do PIO será configurado
  - exemplo (1 << LED_PIO_PIN)
- ul_attribute : parâmetros extras de configuração
  - PIO_DEFAULT : Default pin configuration (no attribute)

```
/*
 *  The #attribute# field is a bitmask that can either be set to PIO_DEFAULT,
 *  or combine (using bitwise OR '|') any number of the following constants:
 *     - PIO_PULLUP
 *     - PIO_DEGLITCH
 *     - PIO_DEBOUNCE
 *     - PIO_OPENDRAIN
 *     - PIO_IT_LOW_LEVEL
 *     - PIO_IT_HIGH_LEVEL
 *     - PIO_IT_FALL_EDGE
 *     - PIO_IT_RISE_EDGE
 */
```

Mais informações sobre as funções do PIO em :

- http://asf.atmel.com/docs/latest/same70/html/group___sam__drivers___pio___group.html

Passo 5a.
Configure o PC8 como sendo saída :

```
//Inicializa PC8 como saída
pio_configure(PIOC, PIO_OUTPUT_0, LED_PIO_PIN_MASK, PIO_DEFAULT);
```

Iremos utilizar muito essa máscara, já que é a maneira utilizada para acessar um pino específico do PIO, devido a isso podemos fazer um novo define para essa operação e usar ao longo do código.

Passo 5b.
Atualize o define LED_PIO_PIN_MASK para :

```
#define LED_PIO_PIN_MASK (1 << LED_PIO_PIN)
```

Acionando o LED com 1 / 0

Agora que o periférico está ativado (energizado via PMC) e que o pino que o LED está conectado já foi configurado como saída, podemos colocar 1/0 no pino do LED, fazendo com que ele acenda ou apague via as duas funções a seguir :

```
// coloca 1 no pino do LED.
pio_set(PIOC, LED_PIO_PIN_MASK);


// colo 0 no pino do LED
pio_clear(PIOC, LED_PIO_PIN_MASK)
```

Passo 6.
A partir dessas funções e fazendo o uso da função delay_ms() conseguimos fazer o LED piscar a uma taxa fixa. Substitua xxxx e yyyy pelas funções corretas declaradas anteriormente.

```
while(1){
     // Set LED
     xxxxxxxx

     delay_ms(200);

     // Clear Led
     yyyyyyyy

     delay_ms(200);

}
```

Brinque com o valor do **delay_ms()**. Além de **delay_ms()** temos : **delay_s()**, **delay_us()** tudo isso graças a inclusão da biblioteca delay no ASF !
Qual seria a frequência de piscar do LED mais rápida que conseguíamos colocar e ainda percebemos o LED piscar?

Entrada - Botão

Passos :

1. Vá até a secção **4.4.2 Mechanical Buttons** e leia sobre os botões da placa.

2. Preencha a tabela :

Periférico	PIO (A/B,C/D)	ID PIO	BIT PIO
Botão SW300			

3. Crie os #defines para o botão

```
#define BUT_PIO
#define BUT_PIO_ID
#define BUT_PIO_PIN
#define BUT_PIO_PIN_MASK
```

4. Ative o clock do periférico via a função do pmc :

```
pmc_enable_periph_clk(BUT_PIO);
```

4. Utilize a função pio_configure porém agora com os parâmetros **PIO_INPUT** no lugar do **PIO_OUTPUT_0** e com o **attribute** configurado para **PIO_PULLUP**.

```
// configura pino como entrada.
pio_configure(BUT_PIO, PIO_INPUT, BUT_PIN_MASK, PIO_PULLUP);
```

5. Busque na documentação do ASF PIO qual função pode ser utilizada para ler se uma entrada é 1 ou 0.

6. Faça com que toda vez que o botão for precionado o LED pisque por 5 vezes e depois pare.