Insper Instituto de Ensino e Pesquisa

09 - Projeto Driver

Rafael Corsi
rafael.corsi@insper.edu.br

15 de março de 2017

Entregar o código via github até a próxima aula (15/3)

1. Criar softwares para microcontroladores utilizando suas especificidades (periféricos/ low power);
2. Avaliar e melhorar soluções embarcadas integrando hardware/software levando em conta adequação a uma aplicação;
3. Integrar em um protótipo hardware, software básico, sistema operacional de tempo real e módulos de interfaceamento com usuários, de comunicação e de alimentação;
4. Compreender as limitações de microcontroladores e seus periféricos;
5. Buscar e analisar documentação (datasheet) e extrair informações

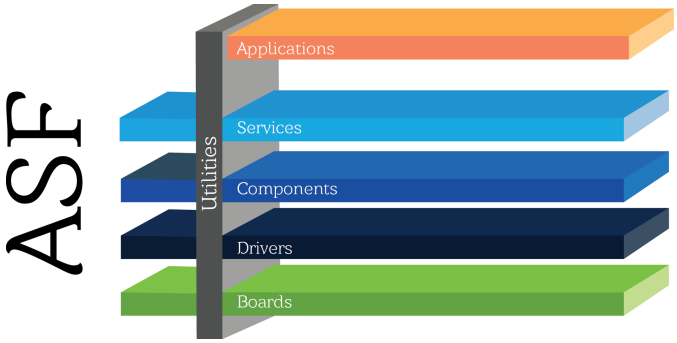
1 Driver

Drivers, APIs, FrameWork ou HAL (Hardware Abstraction Layer) são normalmente funções e defines criados com o intuito de facilitar o desenvolvimento de um projeto ou utilização de uma propriedade do microcontrolador ou de seus periféricos (internos ou externos). Fabricantes de CIs querem facilitar o fluxo de desenvolvimento, diminuir o tempo de prototipação e melhorando a portabilidade para incentivar mais desenvolvedores a utilizar seus produtos.

Imagine a situação em que você projetou um aplicação utilizando o processador SAME70 porém no final da etapa de testes, verificou que o microcontrolador utilizado possui muito mais recurso que o necessário, e que para um otimização de preço do produto final, deseja utilizar um uC mais barato da própria ATMEL (por exemplo o SAMD20, em torno de 2 USD). Se o projeto foi realizado via configuração direta dos registradores, uma etapa muito árdua de migração deve acontecer, onde os dois manuais devem ser analisados a fim de procurar os registradores (e suas funcionalidades) equivalentes de cada uC.

E se, no lugar do acesso direto aos registradores, uma camada de abstração fosse utilizada ? Essa camada poderia ter diferentes implementações que variam com o uC utilizado porém para o usuário as funções chamadas seriam as mesmas. Esse tipo de abstracção é utilizada com frequência a fim de facilitar o desenvolvimento e aumentar o grau de portabilidade de um firmware (lembra do Arduino ? Ele funciona assim !).

A portabilidade entre o mesmo fabricante quando utilizado as funções disponíveis não acontece de forma imediata porém é facilitada como não existe um consenso entre os fabricantes para padronizar essas funções, cada fabricante irá utilizar o seu próprio framework o que dificulta a portabilidade entre diferentes fabricantes. A ATMEL por exemplo, chama essas funções de Atmel Software FrameWork (ASF):



Podemos acessar todas as funções disponíveis para o microcontrolador em uso via o ATMEL STUDIO (Menu: asf -> asf wizard).

Para acessar a documentação a documentação do ASF para a família SAME70, basta entrar no site :

`http://asf.atmel.com/docs/3.24.2/search.html?device=same70`

2 Objetivo

Nessa aula dois drivers deverão ser criados, um para controlar o periférico responsável pelo clock (PMC) e outro para comandar o periférico de controle dos pinos digitais (PIO).

As funções criadas devem ser utilizadas no main.c a fim de remover qualquer acesso direto aos registradores.

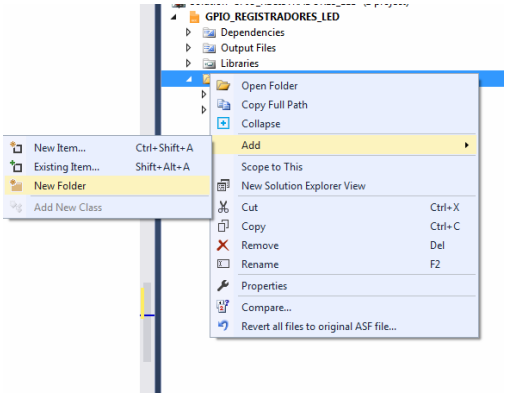
Os drivers são normalmente compostos por arquivos ".c"e ".h". No ".c"estará a implementação das funções e nos arquivos ".h"os includes, defines, varáveis e prototipagem das funções.

A estrutura de arquivos proposta é a seguinte:

```
src
├── main.c
└── Driver
    ├── pio_insper.c
    ├── pio_insper.h
    ├── pmc_insper.c
    └── pmc_insper.h
```

2.1 AtmelStudio

Devemos criar uma pasta "Driver"dentro do src do projeto (/Codigos/09-PIO-DRIVER/Projeto) : src/Driver.



Nessa pasta vamos adicionar 4 arquivos, sendo eles (/Codigos/09-PIO-DRIVER/driver/):

- pmc_insper.c : contém as funções que controlam o PMC
- pmc_insper.h : Header do driver que controla o PMC
- pio_insper.c : contém as funções que controlam o PIO
- pio_insper.h : Header do driver que controla o PIO

Esses arquivos estão localizados no repositório da disciplina, na pasta : */Codigos/09-PIO-DRIVER/Driver*. Para adicionar basta clicarmos com o botão direito sobre a pasta "Driver"→ Add → Existing Item.

2.2 PMC

Duas funções são definidas na biblioteca pmc_insper:

```
1 uint32_t _pmc_enable_periph_clk(uint32_t ID);
2 uint32_t _pmc_disable_periph_clk(uint32_t ID);
```

Essas funções, uma vez incluído no cabeçalho do main.c podem ser utilizadas para configurar o PMC, substituindo a configuração via acesso direto ao registrador:

```
PMC->PMC_PCER0 = (1<<LED_PIO_ID);
```

Verifique como essas funções foram implementadas.

2.2.1 Exemplo: PMC

```
#include <asf.h>

/**
* Inclui a biblioteca de configuracao do PMC
*/
#include "Driver/pmc_insper.h"

int main (void){

/**
* Ativa o clock dos PIOs A e C
*/
_pmc_enable_periph_clock(ID_PIOA);
_pmc_enable_periph_clock(ID_PIOC);
}
```

2.3 PIO

Nessa biblioteca iremos criar uma série de funções que lidam com a configuração e controle do PIO. As funções a serem implementadas já foram definidas no pio_insper.h, porém devemos agora fazer a implementação de sua lógica (pio_insper.c).

Os parâmetros de configuração possíveis para um PIO são:

```
/*  Default pin configuration (no attribute). */
#define PIO_DEFAULT             (0u << 0)

/*  The internal pin pull-up is active. */
#define PIO_PULLUP              (1u << 0)

/*  The internal glitch filter is active. */
#define PIO_DEGLITCH            (1u << 1)

/*  The pin is open-drain. */
#define PIO_OPENDRAIN           (1u << 2)

/*  The internal debouncing filter is active. */
#define PIO_DEBOUNCE            (1u << 3)
```

Essas configurações são aditivas, por exemplo, um pino pode ter configurado o PULL_UP e o PIO_DEBOUNCE , para isso será necessário concatenar as configurações e passar esse valor a função, como no exemplo a seguir:

```
_pio_set_input(LED_PIO, LED_PIN_MASK, PIO_PULLUP | PIO_DEBOUNCE);
```

2.3.1 Exemplo: PISCA LED

Para definirmos um pino em modo de saída, sem pull-up utilizando as funções definidas no driver do PIO (pio_insper.h)

```

/**
 * Includes
 */
#include <asf.h>
#include "Driver/pmc_insper.h"
#include "Driver/pio_insper.h"

/**
 * LED
 */
#define LED_PIO_ID      ID_PIOC
#define LED_PIO         PIOC
#define LED_PIN         8
#define LED_PIN_MASK    (1<<LED_PIN)

main(){
   ...
   ...
   // Ativa clock no periferico que controla o LED
   _pmc_enable_periph_clk(LED_PIO_ID);

   // Configura pino do LED em modo saida
   _pio_set_output(LED_PIO, LED_PIN_MASK, 0, PIO_DEFAULT, PIO_DEFAULT);

   while(1){
     delay_ms(100);
     _pio_clear(LED_PIO, LED_PIN_MASK);
     delay_ms(100);
     _pio_set(LED_PIO, LED_PIN_MASK);
   }

}
```

3 Etapas

A seguir uma sugestão de etapas a serem seguidas a fim de facilitar o desenvolvimento dos drivers. As modificações devem ser feitas nos arquivos dentro da pasta Driver/ e também no main.c.

Sempre que fizer alguma alteração no main.c execute projeto a fim de verificar sua funcionalidade. O código não pode mudar de comportamento.

1. Entenda o que está sendo pedido.
2. Verifique os conteúdos dos arquivos já fornecidos (/09-PIO-DRIVER/driver/)
3. Entenda o que o main.c faz, programe o uC para verificar sua funcionalidade.
4. No main.c inclua os dois .h (pmc_insper.h e pio_insper.h)
5. PMC (pmc_insper.c e pio_insper.h)
   a) Substituir a configuração do PMC do main.c pela função _pmc_enable_periph_clock(...)
6. PIO (pio_insper.c e pio_insper.h)
   a) OutPut
      i. Implementar e usar a função : _pio_set_output(...)
      ii. Implementar e usar as funções : _pio_set(...) e _pio_clear(...)
   b) InPut
      i. Implementar e usar as funções : _pio_pull_up(...) e _pio_pull_down(...)
      ii. Implementar e usar a função : _pio_set_input(...)
      iii. Implementar e usar a função : _pio_get_output_data_status(...)

As funções implementadas devem possuir a mesma funcionalidade e parâmetros da descritas no .h.

4 Avaliação

Estaremos trabalhando nessa etapa os seguintes itens dos objetivos de

1. Faz uso correto de define a fim de melhorar o entendimento/ manipulação do firmware
2. Compreende como as informações extraídas do manual se traduzem para o código
3. Correlaciona as diversas informações contidas em diferentes documentos.
4. Faz uso de comentários
5. Sabe usar corretamente as ferramentas de gravação e depuração
6. Tem claro o fluxo de desenvolvimento

Insatisfatório (I) :	• não apresentou os códigos • contém menos de 70% do total pedido
Em Desenvolvimento (D) :	• apresentou os códigos com até 2 aulas de atraso • contém apenas 70% tópicos do exigido • nem todas as funções implementadas funcionam.
Essencial (C)	• apresentou o código no prazo • entregou mais de 80% do código funcionando • implementou e testou as funções • o código possui comentários mas não em sua totalidade.
Proficiente (B)	• entregou mais de 100% das funções implementadas corretamente • usou e testou todas as funções • não comentou todas as funções
Avançado (A)	• comentou todas as funções e suas chamadas no main.c