Nessa aula iremos utilizar como projeto referência o LAB-1. Vocês devem fazer uma cópia desse projeto para a pasta `Labs/PIO-Driver`, iremos modificar esse projeto.

Entrega

O objetivo desse laboratório é o do entendimento das funções utilizadas para configurar o PIO. Como um pino é configurado como saída e entrada? Como o firmware manipula o periférico PIO? Entender o que o PIO é capaz de fazer. Para isso iremos aqui implementar nossas próprias funções de interface com o PIO.

Pasta
`Labs/PIO-Driver`

- Ao final da aula:
  - [ ] _pio_set()
  - [ ] _pio_clear()
  - [ ] _pio_pull_up()
  - [ ] _pio_set_input()
  - [ ] _pio_set_output()

Lab

Vamos implementar uma série de funções que irão configurar o periférico PIO via a escrita em seu banco de registradores. Para isso será necessário ler o manual do uC mais especificamente a secção do PIO.

_pio_set()

Iremos começar com essa função que é uma das mais simples. Crie uma função no `main.c` com a seguinte estrutura :

```
/**
 * \brief Set a high output level on all the PIOs defined in
ul_mask.
 * This has no immediate effects on PIOs that are not output, but
the PIO
 * controller will save the value if they are changed to outputs.
 *
 * \param p_pio Pointer to a PIO instance.
 * \param ul_mask Bitmask of one or more pin(s) to configure.
 */
void _pio_set(Pio *p_pio, const uint32_t ul_mask)
{

}
```

Na primeira etapa iremos substituir a função que a Atmel/Microchip já nos disponibiliza por uma criada por nós, em todo lugar no código que você faz o uso da função `pio_set(...)` substitua a chamada por essa recém criada `_pio_set(...)` .

> Lembre que essa funcão serve para acionarmos um pino digital (se ele for saída)

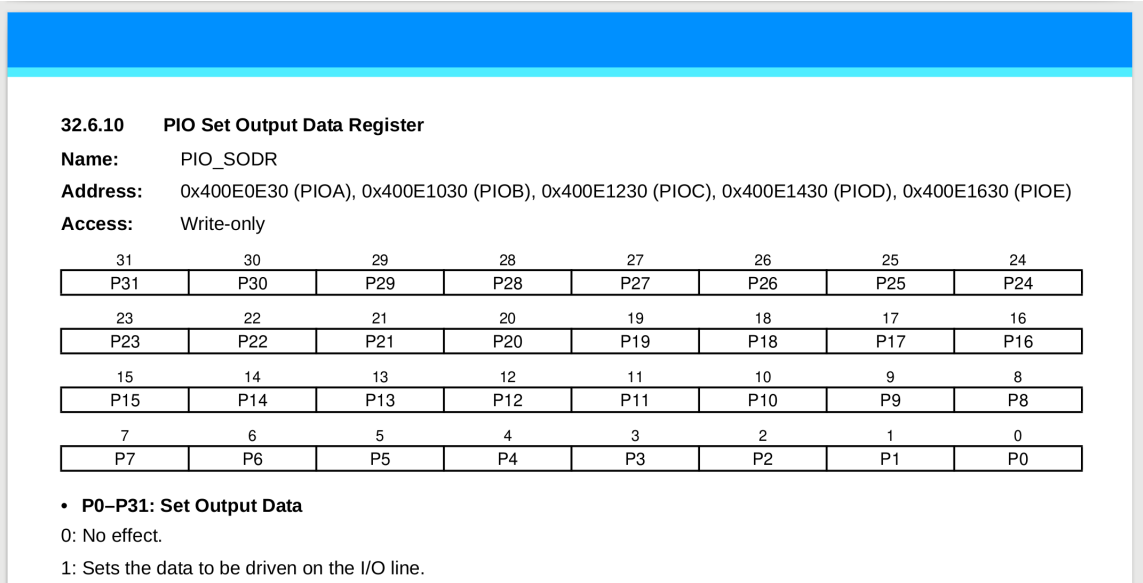
Agora será necessário entender como o PIO controla os pinos e o que deve ser feito para que ele atue sobre o pino como desejamos. A parte da secção do manual que fala sobre o PIO e suas saídas/entradas é a secção 32 do (`manual SAME70`), vamos analisar:

32.5.4 Output Control

> ...
>
> The level driven on an I/O line can be determined by writing in the Set Output Data Register (PIO_SODR) and the Clear Output Data Register (PIO_CODR). These write operations, respectively, set and clear the Output Data Status Register (PIO_ODSR), which represents the data driven on the I/O lines**.

> Writing in PIO_OER and PIO_ODR manages PIO_OSR whether the pin is configured to be controlled by the PIO Controller or assigned to a peripheral function. This enables configuration of the I/O line prior to setting it to be managed by the PIO Controller.

Agora sabemos que para termos 1 no pino devemos escrever no registrador **PIO_SODR**, no manual tem mais detalhes sobre todos os registradores do PIO. Vamos analisar a documentação desse registrador (`SODR`):



32.6.10 PIO Set Output Data Register

Name: PIO_SODR

Address: 0x400E0E30 (PIOA), 0x400E1030 (PIOB), 0x400E1230 (PIOC), 0x400E1430 (PIOD), 0x400E1630 (PIOE)

Access: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0–P31: Set Output Data**

0: No effect.

1: Sets the data to be driven on the I/O line.

Repare que esse registrador é do tipo **write-only** ou seja ele não pode ser lido, somente escrito. Cada bit desse registrador representa um pino, se pegarmos por exemplo o bit 30 desse registrador (pensando no PIOA) estaríamos nos referindo ao PA30, qualquer alteração **ESCRITA** nesse bit influenciará esse **SOMENTE** pino.

> Todos os registradores estão listados e explicados no datasheet, de uma olhada na página **362**, a descrição começa ai.

Agora que já sabemos o que deve ser feito para colocarmos acionarmos um pino (ativar) e considerando que ele já foi configurado como saída podemos escrever a implementação da função:

```
void _pio_set(Pio *p_pio, const uint32_t ul_mask)
{
```

```
    p_pio->PIO_SODR = ul_mask;
}
```

- `*p_pio` : é um endereço recebido do tipo Pio, ele indica o endereço de memória na qual o PIO (periférico) em questão está mapeado (vamos ver isso em detalhes).
- `ul_mask` : é a máscara na qual iremos aplicar ao registrador que controla os pinos para colocarmos 1 na saída.

O que isso significa? Significa que estamos acessando o periférico passado como referência a função (um dos 5 PIOs: PIOA, PIOB, PIOC, ...) e estamos aplicando a máscara `ul_mask` no seu registrador `PIO_SODR` .

> A função está pronta, agora precisamos testar. Com a modificação no código faça a gravação do uC e nada deve mudar na execução do código. Já que a função implementada possui a mesma funcionalidade daquela fornecida pelo Atmel.

```
+ Embarque o código e o mesmo deve funcionar normalmente caso
+ a função implementada esteja correta.
```

_pio_clear(..)

Faça o mesmo para a função clear:

```
/**
 * \brief Set a low output level on all the PIOs defined in
ul_mask.
 * This has no immediate effects on PIOs that are not output, but
the PIO
 * controller will save the value if they are changed to outputs.
 *
 * \param p_pio Pointer to a PIO instance.
 * \param ul_mask Bitmask of one or more pin(s) to configure.
 */
void _pio_clear(Pio *p_pio, const uint32_t ul_mask)
{

}
```

Vocês deverão descobrir pelo manual qual o periférico que deve ser acessado. Releia a secção 32.5.4

> Teste a função implementada substituindo a função **pio_clear()** pela função **_pio_clear()** e embarque o código. Ele deve se comportar igual.

```
+ Embarque o código e o mesmo deve funcionar normalmente caso
+ a função implementada esteja correta.
```

_pio_pull_up(...)

Vamos implementar uma função que faz a configuração do `pullup` nos pinos do PIO, esse pullup é utilizado no botão da placa. Para isso declare a função a seguir:

```
/**
 * \brief Configure PIO internal pull-up.
 *
 * \param p_pio Pointer to a PIO instance.
 * \param ul_mask Bitmask of one or more pin(s) to configure.
 * \param ul_pull_up_enable Indicates if the pin(s) internal pull-
up shall be
 * configured.
 */
void _pio_pull_up(Pio *p_pio, const uint32_t ul_mask,
        const uint32_t ul_pull_up_enable){

 }
```

Essa função recebe o PIO que irá configurar, os pinos que serão configurados e como último parâmetro se o pullup estará ativado (1) ou desativado (0). Para implementar leia **a secção 32.5.1**.

> Teste a função implementada substituindo a função **pio_pull_up()** pela função **_pio_pull_up()** e embarque o código. Ele deve se comportar igual.

```
+ Embarque o código e o mesmo deve funcionar normalmente caso
+ a função implementada esteja correta.
```

_pio_set_input(...)

Agora vamos criar uma nova função para configurar um pino como entrada, para isso inclua os seguintes defines que serão utilizados como forma de configuração da função:

```
/*  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)
```

Esses defines serão passados como configuração da função `_pio_set_input()` no parâmetro `ul_attribute`. Declare no seu código a seguinte função:

```
/**
 * \brief Configure one or more pin(s) or a PIO controller as
inputs.
 * Optionally, the corresponding internal pull-up(s) and glitch
filter(s) can
 * be enabled.
 *
 * \param p_pio Pointer to a PIO instance.
 * \param ul_mask Bitmask indicating which pin(s) to configure as
input(s).
 * \param ul_attribute PIO attribute(s).
 */
void _pio_set_input(Pio *p_pio, const uint32_t ul_mask,
        const uint32_t ul_attribute)
{

}
```

> Leia a secção do `datasheet 32.5.9` para verificar os registradores necessários para implementar a função.

Para testar essa função substitua o seguinte trecho de código que configura um pino como entrada + o pull-up

```
pio_set_input(BUT_PIO, BUT_PIO_MASK, _PIO_DEFAULT);
_pio_pull_up(BUT_PIO, BUT_PIN_MASK, 1);
```

Para :

```
_pio_set_input(BUT_PIO, BUT_PIO_MASK, _PIO_PULLUP);
```

```
+ Embarque o código e o mesmo deve funcionar normalmente caso
+ a função implementada esteja correta.
```

_pio_set_output(...)

Na aula passada utilizamos a função `pio_set_output` para configurarmos que o pino é uma saída. Iremos aqui definir uma nova função chamada de `_pio_set_output()` que implementa essa função.

Defina no seu código a função a seguir:

```
/**
 * \brief Configure one or more pin(s) of a PIO controller as
outputs, with
 * the given default value. Optionally, the multi-drive feature
can be enabled
 * on the pin(s).
 *
 * \param p_pio Pointer to a PIO instance.
 * \param ul_mask Bitmask indicating which pin(s) to configure.
 * \param ul_default_level Default level on the pin(s).
 * \param ul_multidrive_enable Indicates if the pin(s) shall be
configured as
 * open-drain.
 * \param ul_pull_up_enable Indicates if the pin shall have its
pull-up
 * activated.
 */
void _pio_set_output(Pio *p_pio, const uint32_t ul_mask,
        const uint32_t ul_default_level,
        const uint32_t ul_multidrive_enable,
```

```
        const uint32_t ul_pull_up_enable)
{

}
```

Essa função é um pouco mais complexa, e deve executar as seguintes configurações:

1. Configurar o PIO para controlar o pino
    - secção 32.5.2

> When a pin is multiplexed with one or two peripheral functions, the selection is controlled with the Enable Register (PIO_PER) and the Disable Register (PIO_PDR). The Status Register (PIO_PSR) is the result of the set and clear registers and indicates whether the pin is controlled by the corresponding peripheral or by the PIO Controller.

1. Configurar o pino em modo saída
    - secção `32.5.4`
2. Definir a saída inicial do pino (1 ou 0)
    - aqui você pode fazer uso das duas funções recentes implementadas.
3. Ativar ou não o multidrive :
    - Leia a secção `32.5.6`
4. Ativar ou não o pull-up :
    - utilize a função `_pio_pull_up()` recém declarada.

> Uma vez implementada a função, utilize ela no seu código substituindo a função `pio_set_output()` por essa função `_pio_set_output()` . Teste se o LED continua funcionando, se continuar quer dizer que sua função foi executada com sucesso.

```
+ Embarque o código e o mesmo deve funcionar normalmente caso
+ a função implementada esteja correta.
```

Extras

_pio_get(...)

*/ * \brief Return 1 if one or more PIOs of the given Pin instance currently have * a high level; otherwise returns 0. This method returns the actual value that * is being read on the pin. To return the supposed output value of a pin, use * pio_get_output_data_status() instead. * * \param p_pio Pointer to a PIO instance. * \param ul_type PIO type. * \param ul_mask Bitmask of one or more pin(s) to configure. * * \retval 1 at least one PIO currently has a high level. * \retval 0 all PIOs have a low level. / uint32_t pio_get(Pio* p_pio, const pio_type_t ul_type, const uint32_t ul_mask) {}*