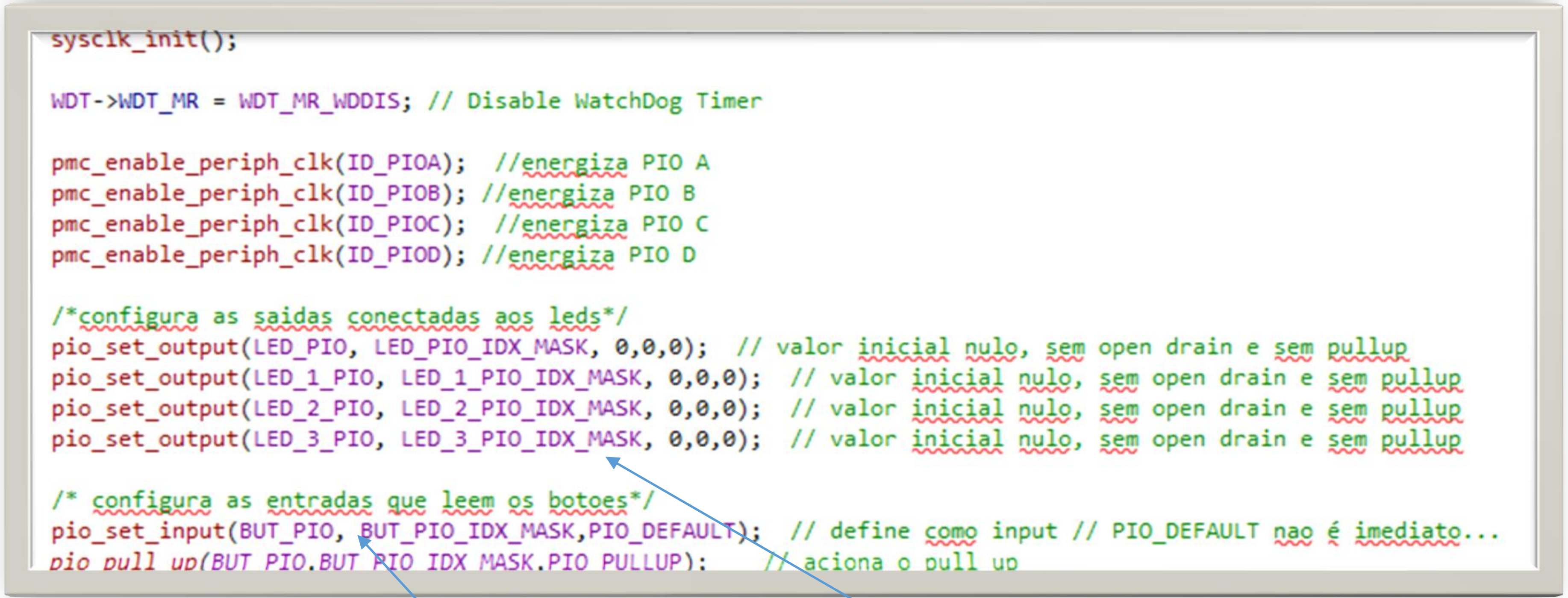
# Computação embarcada 2023-1

## LAB 2 – PIO DRIVER

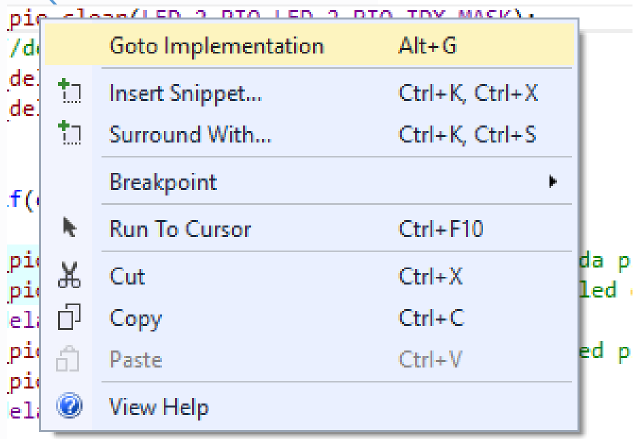


```
sysclk_init();

WDT->WDT_MR = WDT_MR_WDDIS; // Disable WatchDog Timer

pmc_enable_periph_clk(ID_PIOA);  //energiza PIO A
pmc_enable_periph_clk(ID_PIOB); //energiza PIO B
pmc_enable_periph_clk(ID_PIOC);  //energiza PIO C
pmc_enable_periph_clk(ID_PIOD); //energiza PIO D

/*configura as saidas conectadas aos leds*/
pio_set_output(LED_PIO, LED_PIO_IDX_MASK, 0,0,0);  // valor inicial nulo, sem open drain e sem pullup
pio_set_output(LED_1_PIO, LED_1_PIO_IDX_MASK, 0,0,0);  // valor inicial nulo, sem open drain e sem pullup
pio_set_output(LED_2_PIO, LED_2_PIO_IDX_MASK, 0,0,0);  // valor inicial nulo, sem open drain e sem pullup
pio_set_output(LED_3_PIO, LED_3_PIO_IDX_MASK, 0,0,0);  // valor inicial nulo, sem open drain e sem pullup

/* configura as entradas que leem os botoes*/
pio_set_input(BUT_PIO, BUT_PIO_IDX_MASK,PIO_DEFAULT);  // define como input // PIO_DEFAULT nao é imediato...
pio_pull_up(BUT_PIO,BUT_PIO_IDX_MASK,PIO_PULLUP);    // aciona o pull up
```

## Configurações

O PIO suporta as seguintes configurações:

- Interrupção ao nível ou borda em qualquer I/O
- Filtragem de "glitch"
- Deboucing
- Open-Drain
- Pull-up/Pull-down
- Capacidade de trabalhar de forma paralela

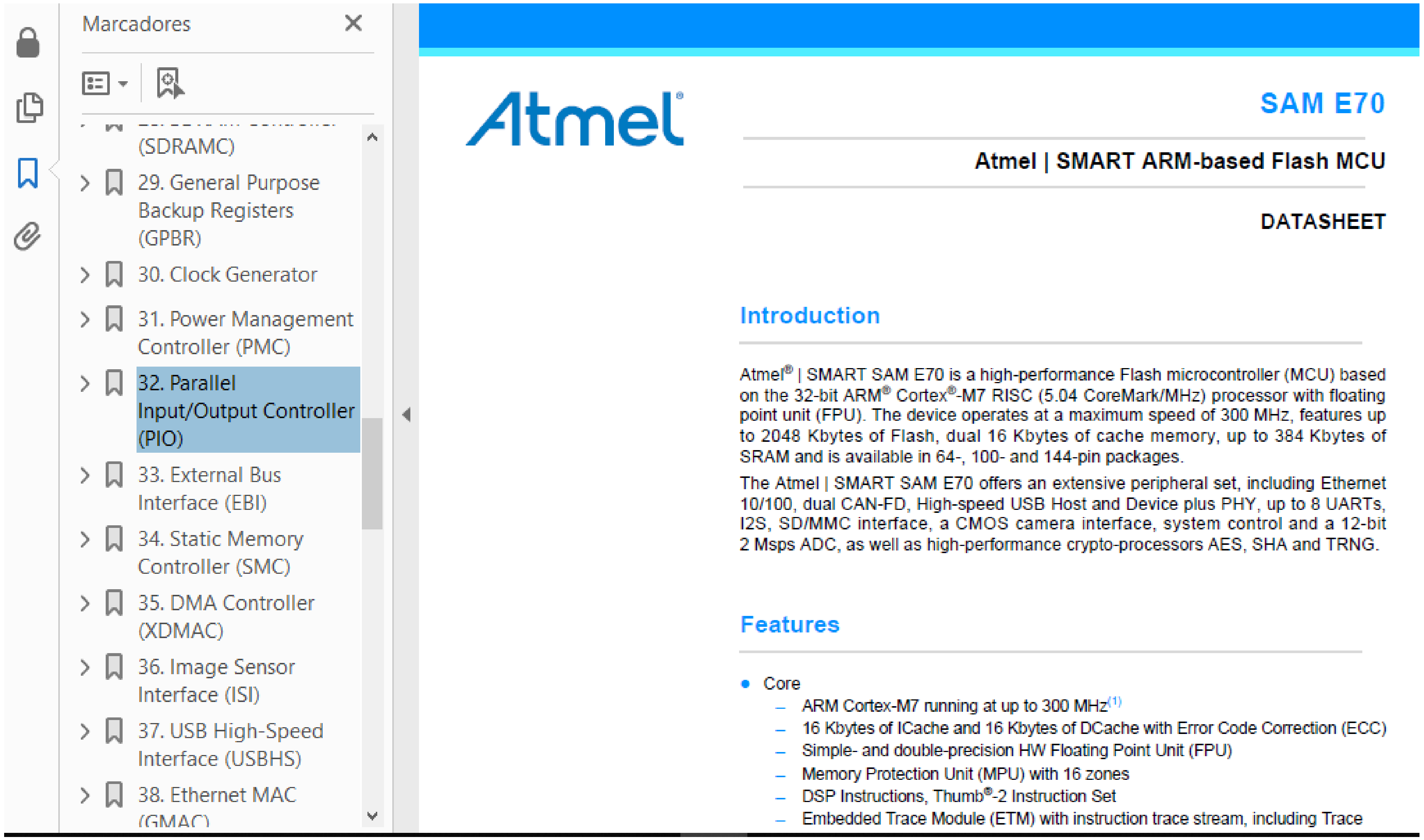

Goto Implementation Alt+G
Insert Snippet... Ctrl+K, Ctrl+X
Surround With... Ctrl+K, Ctrl+S
Breakpoint
Run To Cursor Ctrl+F10
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
View Help

Iremos ver para que serve algumas dessas configurações ao longo do curso.

## HAL

Hardware Abstraction Layer (HAL)

## *NO LAB 2 FAREMOS NOSSA PRÓPRIA CAMADA DE ABSTRAÇÃO !*



Atmel

SAM E70

Atmel | SMART ARM-based Flash MCU

DATASHEET

Introduction

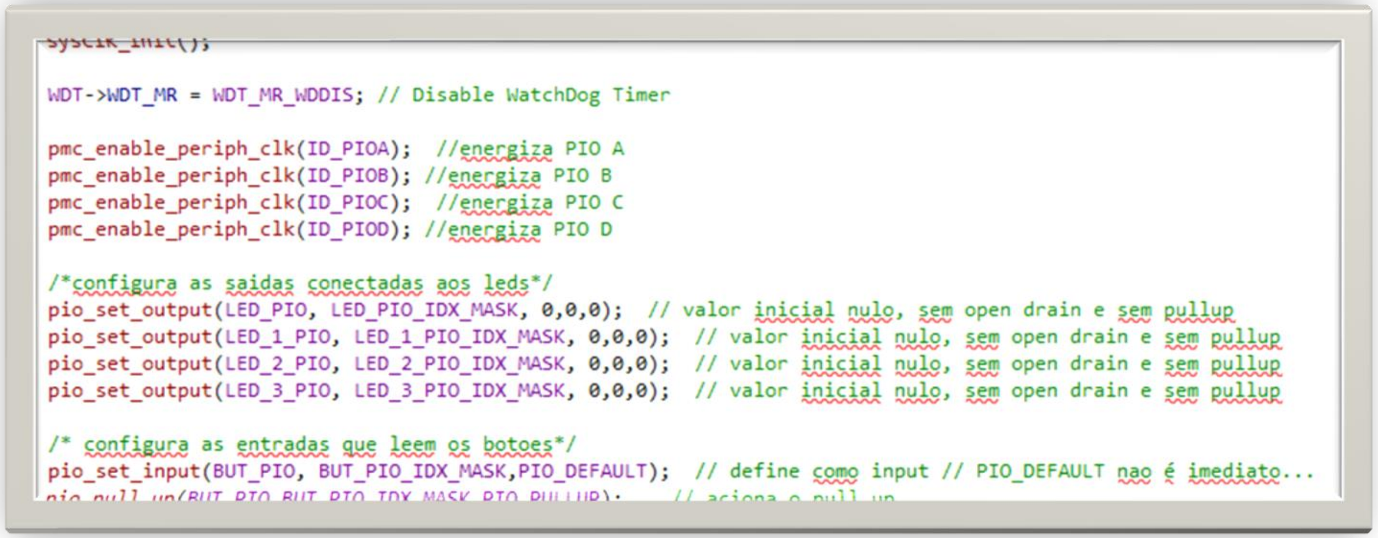
Atmel® | SMART SAM E70 is a high-performance Flash microcontroller (MCU) based on the 32-bit ARM® Cortex®-M7 RISC (5.04 CoreMark/MHz) processor with floating point unit (FPU). The device operates at a maximum speed of 300 MHz, features up to 2048 Kbytes of Flash, dual 16 Kbytes of cache memory, up to 384 Kbytes of SRAM and is available in 64-, 100- and 144-pin packages.

The Atmel | SMART SAM E70 offers an extensive peripheral set, including Ethernet 10/100, dual CAN-FD, High-speed USB Host and Device plus PHY, up to 8 UARTs, I2S, SD/MMC interface, a CMOS camera interface, system control and a 12-bit 2 Msps ADC, as well as high-performance crypto-processors AES, SHA and TRNG.

Features

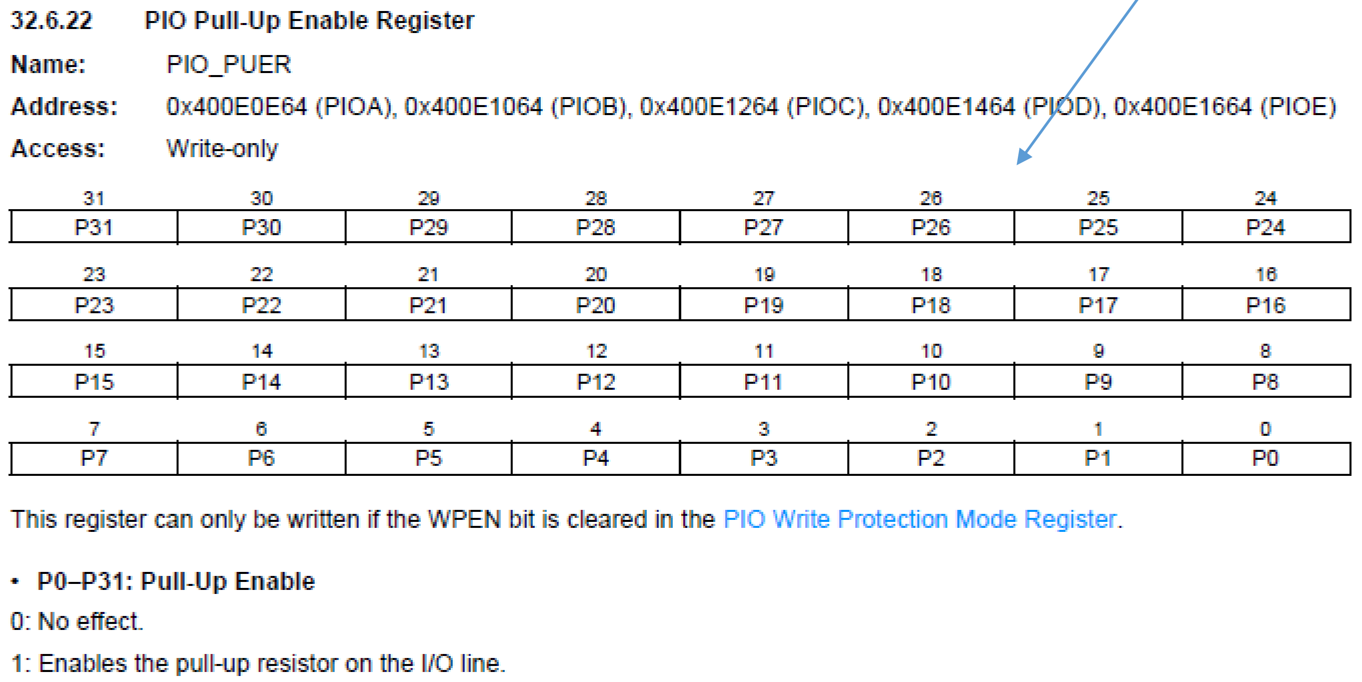
- Core
  - ARM Cortex-M7 running at up to 300 MHz(1)
  - 16 Kbytes of ICache and 16 Kbytes of DCache with Error Code Correction (ECC)
  - Simple- and double-precision HW Floating Point Unit (FPU)
  - Memory Protection Unit (MPU) with 16 zones
  - DSP Instructions, Thumb®-2 Instruction Set
  - Embedded Trace Module (ETM) with instruction trace stream, including Trace

***Cada PIO possui 89 registradores!***



```
sysclk_init();

WDT->WDT_MR = WDT_MR_WDDIS; // Disable WatchDog Timer

pmc_enable_periph_clk(ID_PIOA);  //energiza PIO A
pmc_enable_periph_clk(ID_PIOB); //energiza PIO B
pmc_enable_periph_clk(ID_PIOC);  //energiza PIO C
pmc_enable_periph_clk(ID_PIOD); //energiza PIO D

/*configura as saidas conectadas aos leds*/
pio_set_output(LED_PIO, LED_PIO_IDX_MASK, 0,0,0);  // valor inicial nulo, sem open drain e sem pullup
pio_set_output(LED_1_PIO, LED_1_PIO_IDX_MASK, 0,0,0);  // valor inicial nulo, sem open drain e sem pullup
pio_set_output(LED_2_PIO, LED_2_PIO_IDX_MASK, 0,0,0);  // valor inicial nulo, sem open drain e sem pullup
pio_set_output(LED_3_PIO, LED_3_PIO_IDX_MASK, 0,0,0);  // valor inicial nulo, sem open drain e sem pullup

/* configura as entradas que leem os botoes*/
pio_set_input(BUT_PIO, BUT_PIO_IDX_MASK,PIO_DEFAULT);  // define como input // PIO_DEFAULT nao é imediato...
pio_pull_up(BUT_PIO,BUT_PIO_IDX_MASK,PIO_PULLUP);    // aciona o pull up
```

***Exemplo de um registrador***



**32.6.22 PIO Pull-Up Enable Register**

**Name:** PIO_PUER

**Address:** 0x400E0E64 (PIOA), 0x400E1064 (PIOB), 0x400E1264 (PIOC), 0x400E1464 (PIOD), 0x400E1664 (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 |

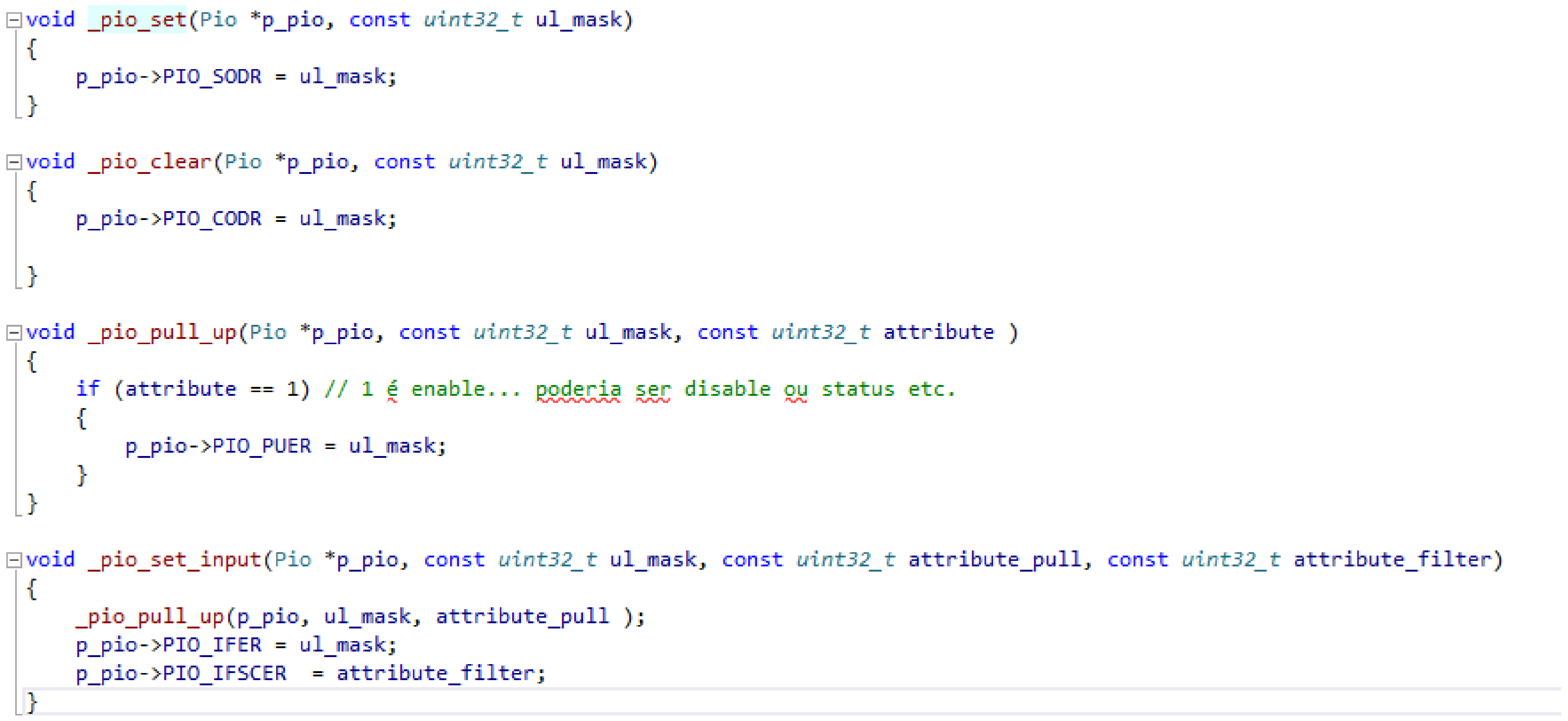
This register can only be written if the WPEN bit is cleared in the PIO Write Protection Mode Register.

- **P0–P31: Pull-Up Enable**

0: No effect.

1: Enables the pull-up resistor on the I/O line.

# Exemplo de construção de funções



```c
void _pio_set(Pio *p_pio, const uint32_t ul_mask)
{
    p_pio->PIO_SODR = ul_mask;
}

void _pio_clear(Pio *p_pio, const uint32_t ul_mask)
{
    p_pio->PIO_CODR = ul_mask;

}

void _pio_pull_up(Pio *p_pio, const uint32_t ul_mask, const uint32_t attribute )
{
    if (attribute == 1) // 1 é enable... poderia ser disable ou status etc.
    {
        p_pio->PIO_PUER = ul_mask;
    }
}

void _pio_set_input(Pio *p_pio, const uint32_t ul_mask, const uint32_t attribute_pull, const uint32_t attribute_filter)
{
    _pio_pull_up(p_pio, ul_mask, attribute_pull );
    p_pio->PIO_IFER = ul_mask;
    p_pio->PIO_IFSCER  = attribute_filter;
}
```

## 32.6 Parallel Input/Output Controller (PIO) User Interface

Each I/O line controlled by the PIO Controller is associated with a bit in each of the PIO Controller User Interface registers. Each register is 32-bit wide. If a parallel I/O line is not defined, writing to the corresponding bits has no effect. Undefined bits read zero. If the I/O line is not multiplexed with any peripheral, the I/O line is controlled by the PIO Controller and PIO_PSR returns one systematically.

**Table 32-5. Register Mapping**

| Offset | Register | Name | Access | Reset |
|---|---|---|---|---|
| 0x0000 | PIO Enable Register | PIO_PER | Write-only | – |
| 0x0004 | PIO Disable Register | PIO_PDR | Write-only | – |
| 0x0008 | PIO Status Register | PIO_PSR | Read-only | [1] |
| 0x000C | Reserved | – | – | – |
| 0x0010 | Output Enable Register | PIO_OER | Write-only | – |
| 0x0014 | Output Disable Register | PIO_ODR | Write-only | – |
| 0x0018 | Output Status Register | PIO_OSR | Read-only | 0x00000000 |
| 0x001C | Reserved | – | – | – |
| 0x0020 | Glitch Input Filter Enable Register | PIO_IFER | Write-only | – |
| 0x0024 | Glitch Input Filter Disable Register | PIO_IFDR | Write-only | – |
| 0x0028 | Glitch Input Filter Status Register | PIO_IFSR | Read-only | 0x00000000 |
| 0x002C | Reserved | – | – | – |
| 0x0030 | Set Output Data Register | PIO_SODR | Write-only | – |
| 0x0034 | Clear Output Data Register | PIO_CODR | Write-only | |
| 0x0038 | Output Data Status Register | PIO_ODSR | Read-only or[2] Read/Write | – |
| 0x003C | Pin Data Status Register | PIO_PDSR | Read-only | [3] |
| 0x0040 | Interrupt Enable Register | PIO_IER | Write-only | – |
| 0x0044 | Interrupt Disable Register | PIO_IDR | Write-only | – |
| 0x0048 | Interrupt Mask Register | PIO_IMR | Read-only | 0x00000000 |
| 0x004C | Interrupt Status Register[4] | PIO_ISR | Read-only | 0x00000000 |
| 0x0050 | Multi-driver Enable Register | PIO_MDER | Write-only | – |
| 0x0054 | Multi-driver Disable Register | PIO_MDDR | Write-only | – |
| 0x0058 | Multi-driver Status Register | PIO_MDSR | Read-only | 0x00000000 |
| 0x005C | Reserved | – | – | – |
| 0x0060 | Pull-up Disable Register | PIO_PUDR | Write-only | – |
| 0x0064 | Pull-up Enable Register | PIO_PUER | Write-only | – |
| 0x0068 | Pad Pull-up Status Register | PIO_PUSR | Read-only | [1] |
| 0x006C | Reserved | – | – | – |

### 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.

## 32.6 Parallel Input/Output Controller (PIO) User Interface

Each I/O line controlled by the PIO Controller is associated with a bit in each of the PIO Controller User Interface registers. Each register is 32-bit wide. If a parallel I/O line is not defined, writing to the corresponding bits has no effect. Undefined bits read zero. If the I/O line is not multiplexed with any peripheral, the I/O line is controlled by the PIO Controller and PIO_PSR returns one systematically.

**Table 32-5. Register Mapping**

| Offset | Register | Name | Access | Reset |
|---|---|---|---|---|
| 0x0000 | PIO Enable Register | PIO_PER | Write-only | – |
| 0x0004 | PIO Disable Register | PIO_PDR | Write-only | – |
| 0x0008 | PIO Status Register | PIO_PSR | Read-only | (1) |
| 0x000C | Reserved | – | – | – |
| 0x0010 | Output Enable Register | PIO_OER | Write-only | – |
| 0x0014 | Output Disable Register | PIO_ODR | Write-only | – |
| 0x0018 | Output Status Register | PIO_OSR | Read-only | 0x00000000 |
| 0x001C | Reserved | – | – | – |
| 0x0020 | Glitch Input Filter Enable Register | PIO_IFER | Write-only | – |
| 0x0024 | Glitch Input Filter Disable Register | PIO_IFDR | Write-only | – |
| 0x0028 | Glitch Input Filter Status Register | PIO_IFSR | Read-only | 0x00000000 |
| 0x002C | Reserved | – | – | – |
| 0x0030 | Set Output Data Register | PIO_SODR | Write-only | – |
| 0x0034 | Clear Output Data Register | PIO_CODR | Write-only | |
| 0x0038 | Output Data Status Register | PIO_ODSR | Read-only or(2) Read/Write | – |
| 0x003C | Pin Data Status Register | PIO_PDSR | Read-only | (3) |
| 0x0040 | Interrupt Enable Register | PIO_IER | Write-only | – |
| 0x0044 | Interrupt Disable Register | PIO_IDR | Write-only | – |
| 0x0048 | Interrupt Mask Register | PIO_IMR | Read-only | 0x00000000 |
| 0x004C | Interrupt Status Register(4) | PIO_ISR | Read-only | 0x00000000 |
| 0x0050 | Multi-driver Enable Register | PIO_MDER | Write-only | – |
| 0x0054 | Multi-driver Disable Register | PIO_MDDR | Write-only | – |
| 0x0058 | Multi-driver Status Register | PIO_MDSR | Read-only | 0x00000000 |
| 0x005C | Reserved | – | – | – |
| 0x0060 | Pull-up Disable Register | PIO_PUDR | Write-only | – |
| 0x0064 | Pull-up Enable Register | PIO_PUER | Write-only | – |
| 0x0068 | Pad Pull-up Status Register | PIO_PUSR | Read-only | (1) |
| 0x006C | Reserved | – | – | – |

### 32.6.7 PIO Input Filter Enable Register

**Name:** PIO_IFER

**Address:** 0x400E0E20 (PIOA), 0x400E1020 (PIOB), 0x400E1220 (PIOC), 0x400E1420 (PIOD), 0x400E1620 (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 |

This register can only be written if the WPEN bit is cleared in the PIO Write Protection Mode Register.

- **P0–P31: Input Filter Enable**

0: No effect.

1: Enables the input glitch filter on the I/O line.

## 32.6 Parallel Input/Output Controller (PIO) User Interface

Each I/O line controlled by the PIO Controller is associated with a bit in each of the PIO Controller User Interface registers. Each register is 32-bit wide. If a parallel I/O line is not defined, writing to the corresponding bits has no effect. Undefined bits read zero. If the I/O line is not multiplexed with any peripheral, the I/O line is controlled by the PIO Controller and PIO_PSR returns one systematically.

**Table 32-5. Register Mapping**

| Offset | Register | Name | Access | Reset |
|---|---|---|---|---|
| 0x0000 | PIO Enable Register | PIO_PER | Write-only | – |
| 0x0004 | PIO Disable Register | PIO_PDR | Write-only | – |
| 0x0008 | PIO Status Register | PIO_PSR | Read-only | [1] |
| 0x000C | Reserved | – | – | – |
| 0x0010 | Output Enable Register | PIO_OER | Write-only | – |
| 0x0014 | Output Disable Register | PIO_ODR | Write-only | – |
| 0x0018 | Output Status Register | PIO_OSR | Read-only | 0x00000000 |
| 0x001C | Reserved | – | – | – |
| 0x0020 | Glitch Input Filter Enable Register | PIO_IFER | Write-only | – |
| 0x0024 | Glitch Input Filter Disable Register | PIO_IFDR | Write-only | – |
| 0x0028 | Glitch Input Filter Status Register | PIO_IFSR | Read-only | 0x00000000 |
| 0x002C | Reserved | – | – | – |
| 0x0030 | Set Output Data Register | PIO_SODR | Write-only | – |
| 0x0034 | Clear Output Data Register | PIO_CODR | Write-only | |
| 0x0038 | Output Data Status Register | PIO_ODSR | Read-only or[2] Read/Write | – |
| 0x003C | Pin Data Status Register | PIO_PDSR | Read-only | [3] |
| 0x0040 | Interrupt Enable Register | PIO_IER | Write-only | – |
| 0x0044 | Interrupt Disable Register | PIO_IDR | Write-only | – |
| 0x0048 | Interrupt Mask Register | PIO_IMR | Read-only | 0x00000000 |
| 0x004C | Interrupt Status Register[4] | PIO_ISR | Read-only | 0x00000000 |
| 0x0050 | Multi-driver Enable Register | PIO_MDER | Write-only | – |
| 0x0054 | Multi-driver Disable Register | PIO_MDDR | Write-only | – |
| 0x0058 | Multi-driver Status Register | PIO_MDSR | Read-only | 0x00000000 |
| 0x005C | Reserved | – | – | – |
| 0x0060 | Pull-up Disable Register | PIO_PUDR | Write-only | – |
| 0x0064 | Pull-up Enable Register | PIO_PUER | Write-only | – |
| 0x0068 | Pad Pull-up Status Register | PIO_PUSR | Read-only | [1] |
| 0x006C | Reserved | – | – | – |

**32.6.54 PIO Parallel Capture Interrupt Status Register**

**Name:** PIO_PCISR

**Address:** 0x400E0F60 (PIOA), 0x400E1160 (PIOB), 0x400E1360 (PIOC), 0x400E1560 (PIOD), 0x400E1760 (PIOE)

**Access:** Read-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – | – |
| **23** | **22** | **21** | **20** | **19** | **18** | **17** | **16** |
| – | – | – | – | – | – | – | – |
| **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** |
| – | – | – | – | – | – | – | – |
| **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| – | – | – | – | – | – | OVRE | DRDY |

- **DRDY: Parallel Capture Mode Data Ready**

0: No new data is ready to be read since the last read of PIO_PCRHR.

1: A new data is ready to be read since the last read of PIO_PCRHR.

The DRDY flag is automatically reset when PIO_PCRHR is read or when the parallel capture mode is disabled.

- **OVRE: Parallel Capture Mode Overrun Error**

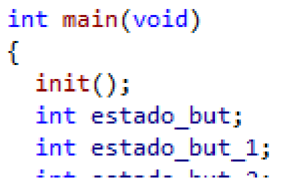
0: No overrun error occurred since the last read of this register.

1: At least one overrun error occurred since the last read of this register.

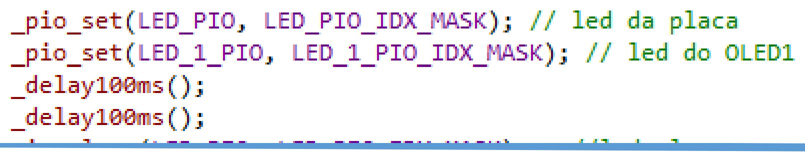
The OVRE flag is automatically reset when this register is read or when the parallel capture mode is disabled.

# Organizando o código

**main.c**

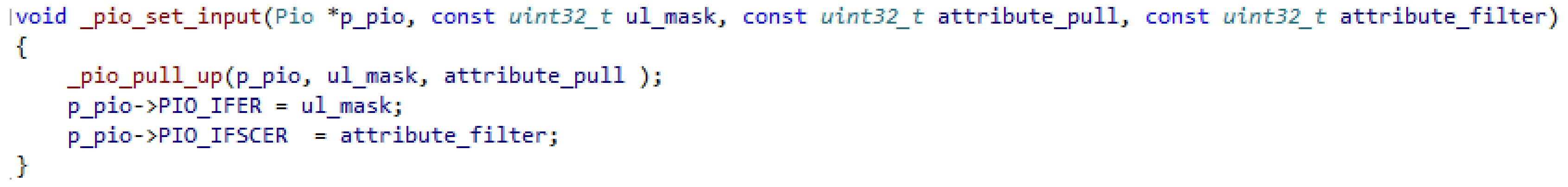


```c
int main(void)
{
  init();
  int estado_but;
  int estado_but_1;
```



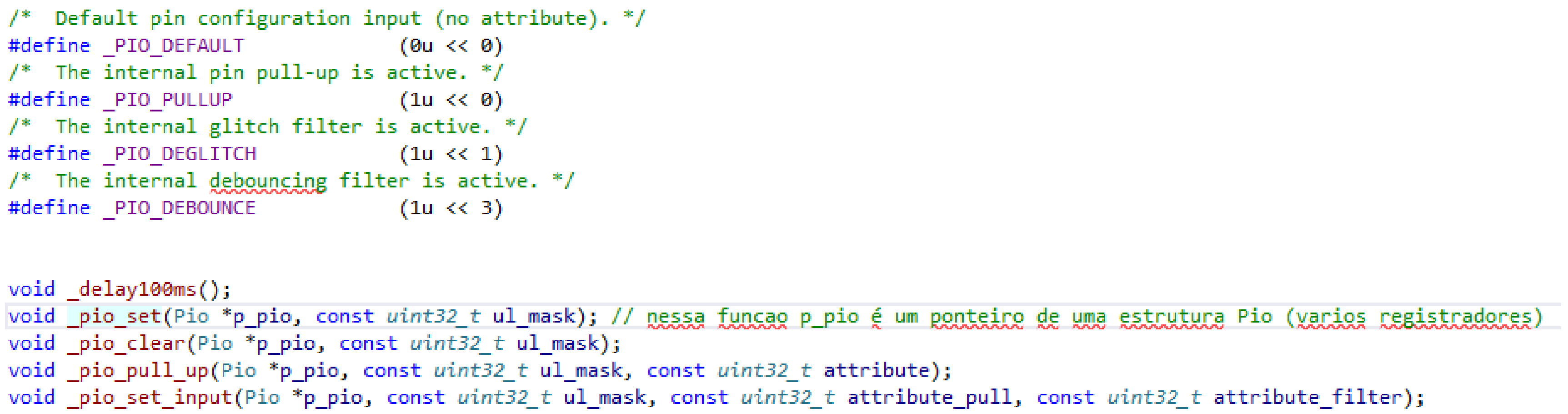
```c
_pio_set(LED_PIO, LED_PIO_IDX_MASK); // led da placa
_pio_set(LED_1_PIO, LED_1_PIO_IDX_MASK); // led do OLED1
_delay100ms();
_delay100ms();
```

**myfunc.c**



```c
void _pio_set_input(Pio *p_pio, const uint32_t ul_mask, const uint32_t attribute_pull, const uint32_t attribute_filter)
{
    _pio_pull_up(p_pio, ul_mask, attribute_pull );
    p_pio->PIO_IFER = ul_mask;
    p_pio->PIO_IFSCER  = attribute_filter;
}
```

**myfunc.h**



```c
/*  Default pin configuration input (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 internal debouncing filter is active. */
#define _PIO_DEBOUNCE            (1u << 3)


void _delay100ms();
void _pio_set(Pio *p_pio, const uint32_t ul_mask); // nessa funcao p_pio é um ponteiro de uma estrutura Pio (varios registradores)
void _pio_clear(Pio *p_pio, const uint32_t ul_mask);
void _pio_pull_up(Pio *p_pio, const uint32_t ul_mask, const uint32_t attribute);
void _pio_set_input(Pio *p_pio, const uint32_t ul_mask, const uint32_t attribute_pull, const uint32_t attribute_filter);
```