BME3321:Introduction to Microcontroller Programming

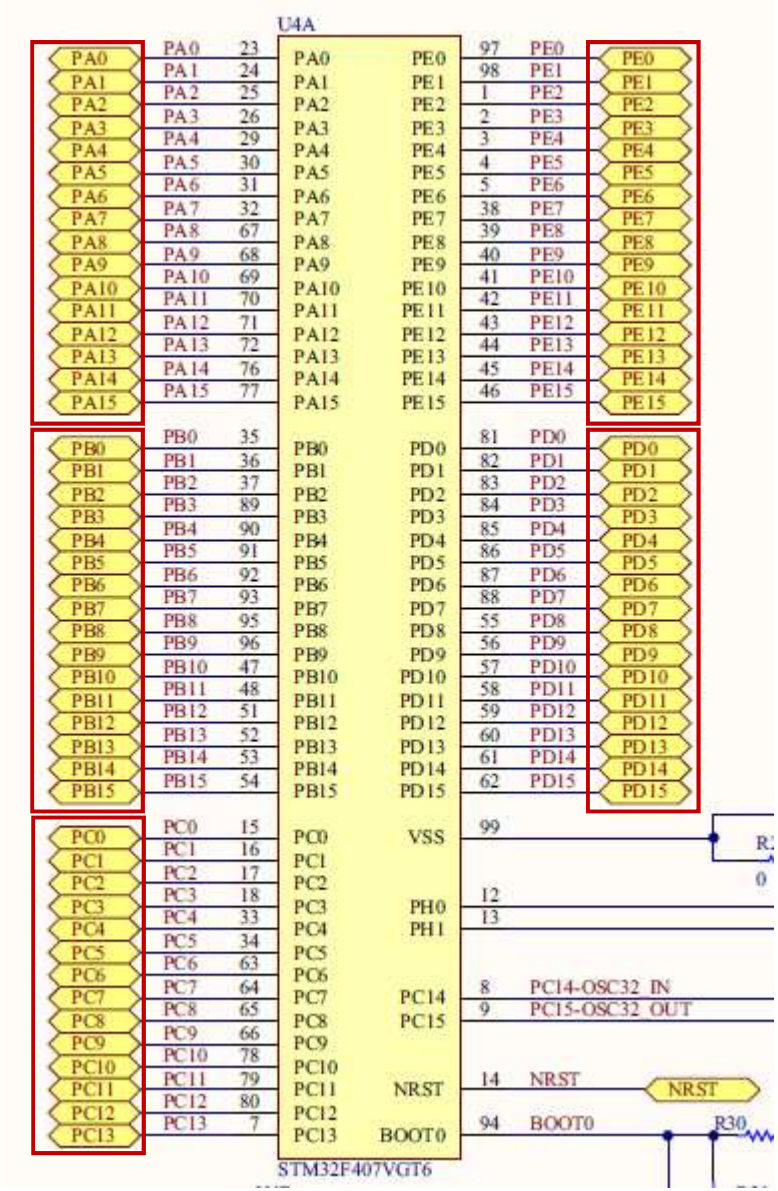
Topic 4: General-purpose I/Os (GPIO)

Asst. Prof. Dr. İsmail Cantürk

Mastering STM32 (ch 6)

STM32F407 reference manual (ch 8)

GPIO of STM32F407



- A,B,C,D,E ports
- Up to 16 I/O pins for each port
- A,B,D,E ports-> 16 pins
- C port->14 pins

GPIO of STM32F407

Each general-purpose I/O port has

four 32-bit configuration registers

- GPIOx_MODER (Mode Register)
- GPIOx_OTYPER (Output Type Register)
- GPIOx_OSPEEDR (Output Speed Register)
- GPIOx_PUPDR (Pull-up/Pull-down Register)

x: port name, it can be A, B, C, D, E

two 32-bit data registers

- GPIOx_IDR (Input Data Register)
- GPIOx_ODR (Output Data Register)

a 32-bit set/reset register

- GPIOx_BSRR (Bit Set/Reset Register)

a 32-bit locking register

- GPIOx_LCKR (Lock Register)

two 32-bit alternate function selection register

- GPIOx_AFRH (Alternate Function High Register)
- GPIOx_AFRL (Alternate Function Low Register)

GPIO

- GPIO registers are used to set up related port's pins for a specific application.
  i.e., input or ouput?, digital or analog?

- Input states: floating, pull-up/down, analog

- Output states: push-pull or open drain + pull-up/down

- Each I/O port bit is freely programmable,

- I/O port registers have to be accessed as 32-bit words, half-words or bytes.

Location GPIO registers in memory map

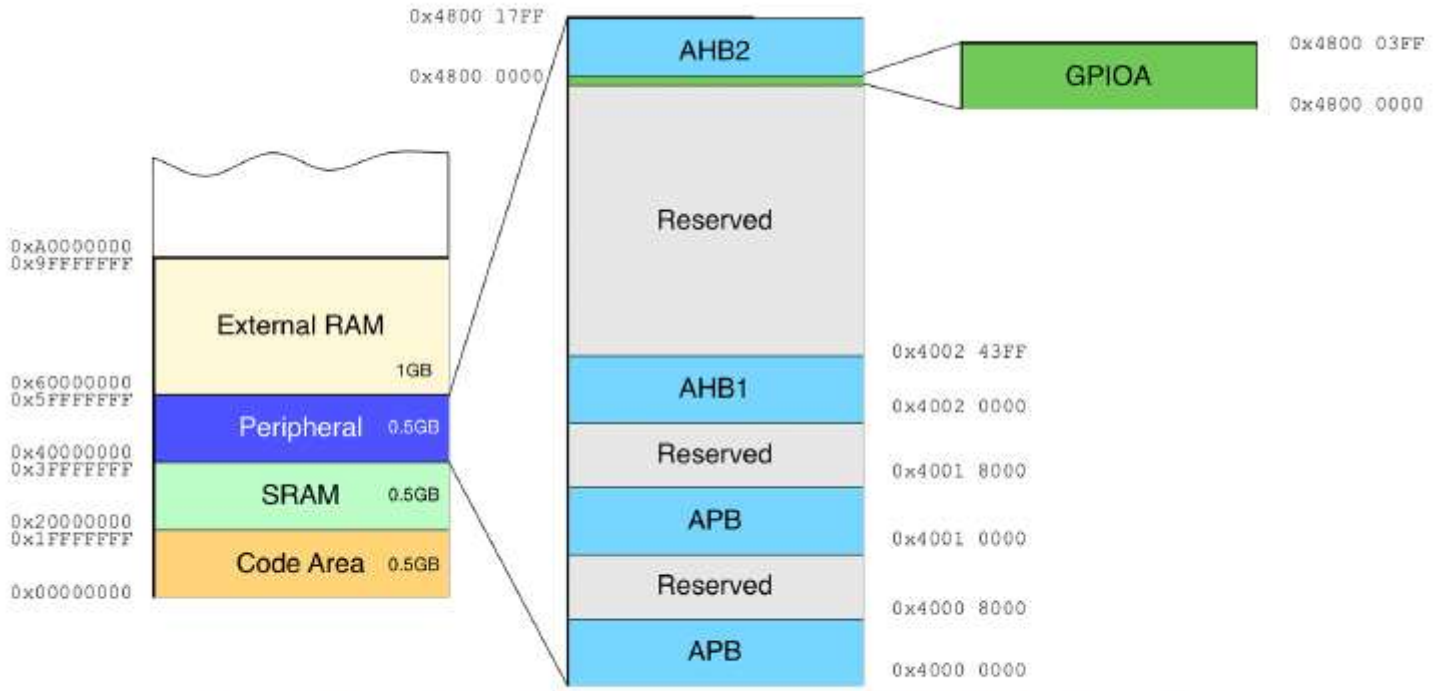


Figure 2: Memory map of peripheral regions for an STM32F030 microcontroller

STM32F407 GPIO register locations

0x4002 3000 - 0x4002 33FF	CRC	AHB1	Section 4.4.4: CRC register map on page 115
0x4002 2800 - 0x4002 2BFF	GPIOK		Section 8.4.11: GPIO register map on page 287
0x4002 2400 - 0x4002 27FF	GPIOJ		
0x4002 2000 - 0x4002 23FF	GPIOI		
0x4002 1C00 - 0x4002 1FFF	GPIOH		
0x4002 1800 - 0x4002 1BFF	GPIOG		
0x4002 1400 - 0x4002 17FF	GPIOF		
0x4002 1000 - 0x4002 13FF	GPIOE		Section 8.4.11: GPIO register map on page 287
0x4002 0C00 - 0x4002 0FFF	GPIOD		
0x4002 0800 - 0x4002 0BFF	GPIOC		
0x4002 0400 - 0x4002 07FF	GPIOB		
0x4002 0000 - 0x4002 03FF	GPIOA		

Boundary addresses of registers for A ports

GPIOA is connected to AHB1 bus

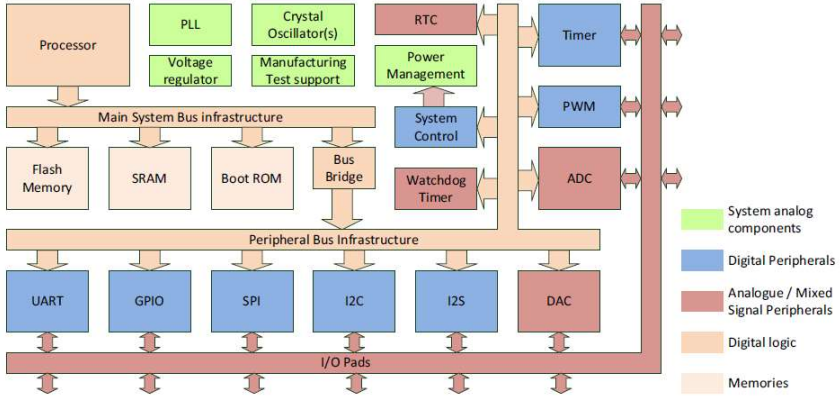
STM32F407 reference manual, memory mapping (ch 2.3)

AHB/APB buses

Buses provide internal connections within the microcontroller.

AHB: Advanced high-performance bus
APB: Advanced peripheral bus

- Each peripheral (i.e., GPIO) is connected to the one of these buses
- In memory mapping of STM32F4, there are AHB1, AHB2, AHB3 and APB1, APB2.
- Speed of peripherals are related to clock frequency of buses



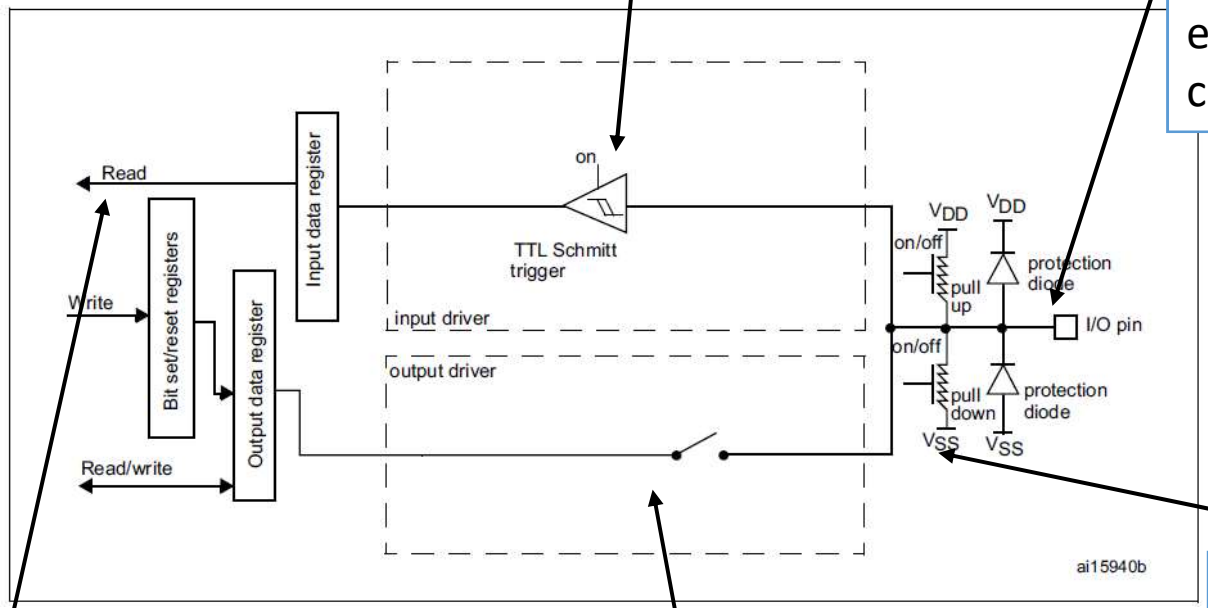
A simple microcontroller.

Check STM32F407 reference manual, memory mapping (ch 2.3), page 64

GPIO input configuration

the Schmitt trigger input is activated

The data present on the I/O pin are sampled into the input data register every AHB1 clock cycle



A read access to the input data register provides the I/O State

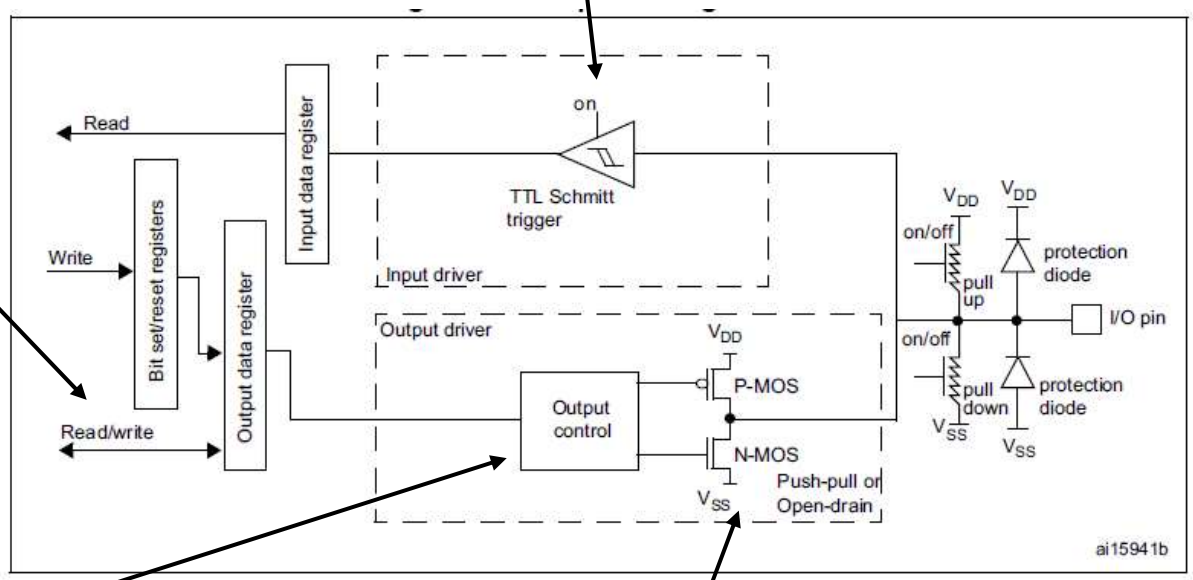
the output buffer is disabled

the pull-up and pull-down resistors are activated depending on the value in the GPIOx_PUPDR register

GPIO output configuration

The Schmitt trigger input is activated, output can be read

A read access to the output data register gets the last written value



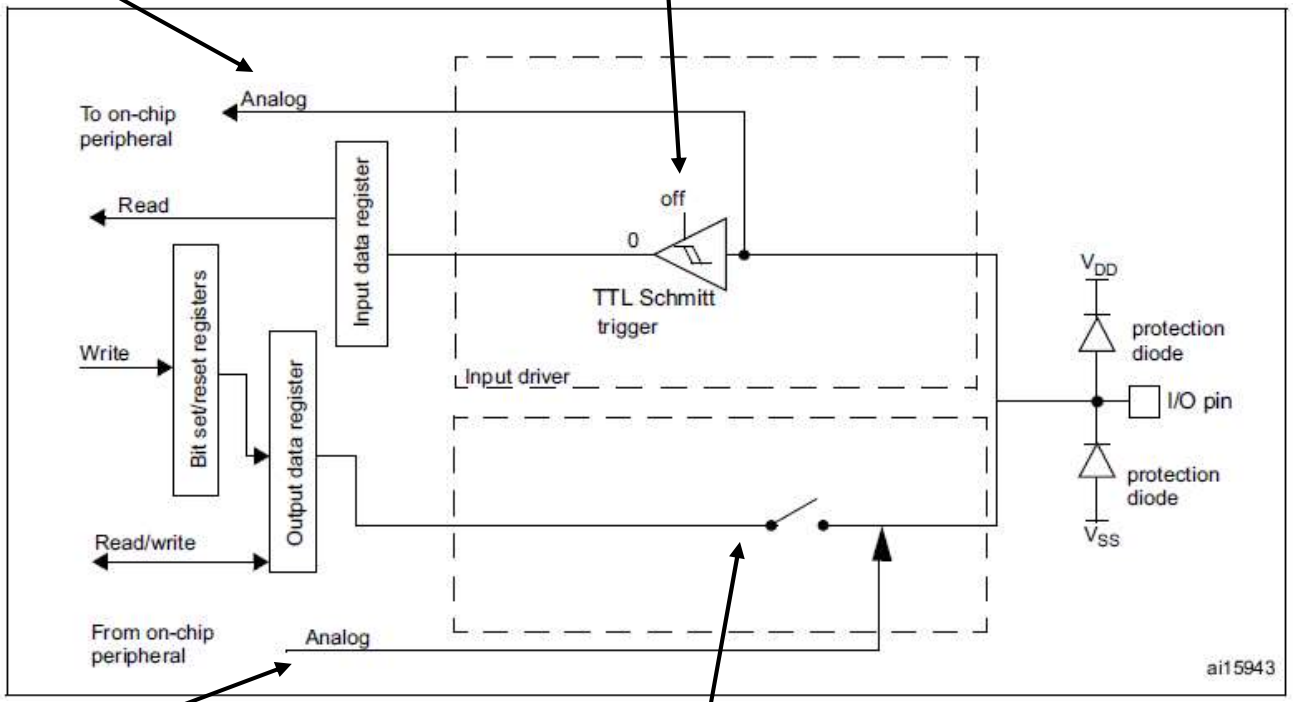
The output buffer is enabled

The output buffer can be configured as open-drain or push-pull

GPIO analog configuration

Analog input

The Schmitt trigger input is deactivated,



Analog ouput

the output buffer is disabled

GPIO port mode register (GPIOx_MODER)

x= A,B,C,D,E

GPIOx_MODER

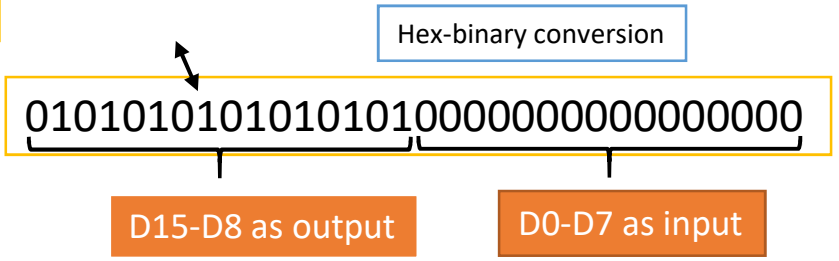
Address offset: 0x00
Reset values:
- 0xA800 0000 for port A
- 0x0000 0280 for port B
- 0x0000 0000 for other ports

These bits are written by software to configure the I/O direction mode.
00: Input (reset state)
01: General purpose output mode
10: Alternate function mode
11: Analog mode

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

```
#define GPIOD_MODER 0x4002 0C00
```

Ex: *GPIOD_MODER= 0x5555 0000 ;



GPIO port output type register (GPIOx_OTYPER)

x= A,B,C,D,E

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **OTy**: Port x configuration bits (y = 0..15)

These bits are written by software to configure the output type of the I/O port.

0: Output push-pull (reset state)

1: Output open-drain

```
#define GPIOD_OTYPER (0x4002 0C00+0x04)
```

Ex: *GPIOD_OTYPER= 0xF000;

1111000000000000

D15-D12 as open drain; rest is push pull assuming all pins are in output mode

GPIO port output speed register (GPIOx_OSPEEDR)

x= A,B,C,D,E

Address offset: 0x08

Reset values:

- 0x0C00 0000 for port A
- 0x0000 00C0 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OSPEEDR15 [1:0]		OSPEEDR14 [1:0]		OSPEEDR13 [1:0]		OSPEEDR12 [1:0]		OSPEEDR11 [1:0]		OSPEEDR10 [1:0]		OSPEEDR9 [1:0]		OSPEEDR8 [1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OSPEEDR7[1:0]		OSPEEDR6[1:0]		OSPEEDR5[1:0]		OSPEEDR4[1:0]		OSPEEDR3[1:0]		OSPEEDR2[1:0]		OSPEEDR1 [1:0]		OSPEEDR0 1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 **OSPEEDRy[1:0]:** Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O output speed.

00: Low speed

01: Medium speed

10: High speed

11: Very high speed

Ex: *GPIOD_OSPEEDR= *GPIOD _OSPEEDR | 0x00000000;

Low speed for all pins

GPIO port pull-up/pull-down register (GPIOx_PUPDR) x= A,B,C,D,E

Address offset: 0x0C

Reset values:

- 0x6400 0000 for port A
- 0x0000 0100 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PUPDR15[1:0]		PUPDR14[1:0]		PUPDR13[1:0]		PUPDR12[1:0]		PUPDR11[1:0]		PUPDR10[1:0]		PUPDR9[1:0]		PUPDR8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PUPDR7[1:0]		PUPDR6[1:0]		PUPDR5[1:0]		PUPDR4[1:0]		PUPDR3[1:0]		PUPDR2[1:0]		PUPDR1[1:0]		PUPDR0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

PUPDRy[1:0]: Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O pull-up or pull-down

00: No pull-up, pull-down

01: Pull-up

10: Pull-down

11: Reserved

GPIO port input data register (GPIOx_IDR)

x= A,B,C,D,E

Address offset: 0x10

Reset value: 0x0000 XXXX (where X means undefined)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **IDRy**: Port input data (y = 0..15)
These bits are read-only and can be accessed in word mode only. They contain the input value of the corresponding I/O port.

Ex:
```
if (*GPIOD _ IDR & 0xF000)
    {
        //do this
    }
```

GPIO port output data register (GPIOx_ODR)

x= A,B,C,D,E

Address offset: 0x14

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **ODRy**: Port output data (y = 0..15)
These bits can be read and written by software.

Ex: *GPIOD_ODR= 0xFFFF;

All pins are high

Other GPIO registers

- GPIO port bit set/reset register (GPIOx_BSRR)
- GPIO port configuration lock register (GPIOx_LCKR)
- GPIO alternate function low register (GPIOx_AFRL)
- GPIO alternate function high register (GPIOx_AFRH)
- Check these ones from reference manual

GPIO registers

- We have each of these registers for each port

- i.e., STM32F4 has A,B,C,D,E ports. Therefore it includes five GPIOx_MODER registers

  GPIOA_MODER
  GPIOB_MODER
  GPIOC_MODER
  GPIOD_MODER
  GPIOE_MODER

GPIO Management with Hardware Abstraction Layer (HAL)

Hardware Abstraction Layer (HAL) : official library to develop STM32 applications

HAL provides many functions, abstract from the specific peripheral mapping, ...

peripheral mapping w/o HAL

```
int main(void) {

uint32_t *GPIOA_MODER = 0x0, *GPIOA_ODR = 0x0;

GPIOA_MODER = (uint32_t*)0x48000000; // Address of the GPIOA->MODER register

GPIOA_ODR = (uint32_t*)(0x48000000 + 0x14); // Address of the GPIOA->ODR register



*GPIOA_MODER = *GPIOA_MODER | 0x400; // Sets MODER[11:10] = 0x1

*GPIOA_ODR = *GPIOA_ODR | 0x20; // Sets ODR[5] = 0x1, that is pulls PA5 high

while(1);
}
```

Use of pointers to access to the GPIOA peripheral mapped in memory of an STM32F030 MCU.

Address mapping is provided by HAL

GPIO registers with HAL

```
typedef struct {
volatile uint32_t MODER;
volatile uint32_t OTYPER;
volatile uint32_t OSPEEDR;
volatile uint32_t PUPDR;
volatile uint32_t IDR;
volatile uint32_t ODR;
volatile uint32_t BSRR;
volatile uint32_t LCKR;
volatile uint32_t AFR[2];
volatile uint32_t BRR;
} GPIO_TypeDef;
```

- GPIO registers are managed with C struct, whose references are used to point to real peripheral address.

- This is a general struct definition for GPIO and it can be used for all I/O peripherals.

```
GPIO_TypeDef *GPIOA = 0x48000000;
GPIOA->MODER |= 0x400;
GPIOA->ODR |= 0x20;
```

GPIOA pointer variable definition in GPIO_TypeDef struct type.

Starting address of GPIOA is assigned

Accessing struct variables

GPIO Configuration

To configure a GPIO, we use this function:

HAL_GPIO_Init (GPIO_TypeDef *GPIOx, GPIO_InitTypeDef *GPIO_Init)

GPIO_InitTypeDef is the C struct used to configure the GPIO, and it is defined in the following way:

```
typedef struct {
uint32_t Pin;
uint32_t Mode;
uint32_t Pull;
uint32_t Speed;
uint32_t Alternate;
} GPIO_InitTypeDef;
```

GPIO Configuration

- Pin: it is the number, starting from 0, of the pins we are going to configure. For example, for PA5 pin it assumes the value GPIO_PIN_5.

- Mode: it is the operating mode of the pin. e.g., input, output, analog...

- Pull: specifies the Pull-up or Pull-Down activation for the selected pins.

- Speed: defines the pin speed.

- Alternate: alternate function definition

These configurations are done with STMCubeMX

Driving a GPIO

- CubeHAL provides four manipulation routines to read, change and lock the state of an I/O.

To read the status of an I/O we can use the function:

```
HAL_GPIO_ReadPin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
```

which accepts the GPIO descriptor and the pin number. It returns GPIO_PIN_RESET when the I/O is low or GPIO_PIN_SET when high.

To change the I/O state, we have the function:

```
HAL_GPIO_WritePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)
```

which accepts the GPIO descriptor, the pin number and the desired state.

Driving a GPIO

If we want to simply invert the I/O state, then we can use this convenient routine:

```
HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
```

we can lock the configuration of an I/O. Any subsequent attempt to change its configuration will fail, until a reset occurs. To lock a pin configuration we can use this routine:

```
HAL_GPIO_LockPin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
```

Driving GPIO examples

```
HAL_GPIO_WritePin(GPIOD,GPIO_PIN_14,GPIO_PIN_SET);

HAL_GPIO_TogglePin(GPIOD,GPIO_PIN_12);

HAL_GPIO_ReadPin(GPIOD,GPIO_PIN_12);
```