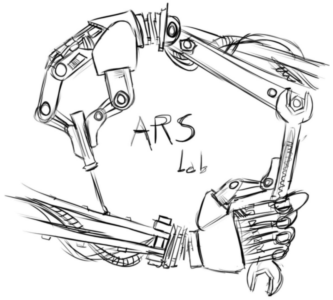
Using the Special Function Registers of the Digital I/O interface of STM32

Corrado Santoro

ARSLAB - Autonomous and Robotic Systems Laboratory
Dipartimento di Matematica e Informatica - Università di Catania, Italy
santoro@dmi.unict.it



L.A.P. 1 Course

The General Purpose I/O (GPIO) Interface of STM32

- MCUs of the STM32 family have several **digital ports**, called **GPIOA**, **GPIOB**, **GPIOC**, **...**,
- Each port has **16 bits** and thus **16 electrical pins**
- Pins are referred as **P*xy***, where *x* is the port name (A, B, ..., E) and *y* is the bit (0, 1, ..., 15).
- As an example, the pin `PC3` is the bit 3 of the port C.
- Each PIN has also an **alternate function**, related to a peripheral e.g. Timer, UART, SPI, etc.
- According to the MCU package, not all bits are mapped to electrical pins. This is a choice “by-design”.

Digital I/O and SFR

Each port *x* has 11 SFRs:

- **MODER**: configures each bit as input or output or other
- **OSPEEDR**: configures the maximum frequency of an output pin
- **PUPDR**: configures the internal pull-up or pull-down register
- **IDR**: the input data register
- **ODR**: the output data register
- **BSRR**: the bit set/reset register
- **AFRL, AFRH**: alternate function configuration registers
- **LCKR**: the bit lock register
- **OTYPER**: output type configuration (push-pull or open-drain)

Accessing is made:

- By using the predefined **structure pointers**:`GPIOA`, `GPIOB`, `GPIOC`
- By accessing the SFR as the structure pointer field: `GPIOA->ODR`

MODE Register

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

- **MODER** allows a programmer to define the functionality of a GPIO pin
- Each pin has **2 bits** that permits the following configurations:
  - **00**: Input
  - **01**: Output
  - **10**: Alternate Function
  - **11**: Analog

Output Type Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
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

- **OTYPER** allows a programmer to configure the output stage of an output GPIO pin
- Each pin has **1 bits** that permits the following configurations:
  - **0**: Push-pull
  - **1**: Open Drain

Output Speed Register

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

- **OSPEEDR** allows a programmer to define the *speed* of an output GPIO pin
- Each pin has **2 bits** that permits the following configurations:
  - **x0**: Low Speed
  - **01**: Medium Speed
  - **11**: High Speed

Pull-up/Pull-Down Register

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

- **PUPDR** defines the presence of a pull-up or pull-down restistor (or none) at the GPIO pin
- Each pin has **2 bits** that permits the following configurations:
  - **00**: No pull-up/pull-down
  - **01**: Pull-up
  - **10**: Pull-down

Data Input/Output Registers

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.

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

- Data Input/Ouput is performed through the **IDR** and **ODR** registers
- Each pin is mapped to the specific bit, so only 16 bits are used in the registers
- Bit set/reset and check operations are performed through logical mask operations

Single-bit Data Output Registers

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

- Single-bit data output (set or reset) can be performed through the **BSRR** register
- The register has two parts: **set part** and **reset part**
- To **set a pin**, a “1” must be written in the correspondent *set part*
- To **reset a pin**, a “1” must be written in the correspondent *reset part*

Single-bit Data Reset Registers

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

- Single-bit data reset can be also performed through the **BRR** register
- To **reset a pin**, a “1” must be written in the correspondent bit

First Example: Read a Pushbutton and lit the LED

```
#include "stm32_unict_lib.h"

int main()
{
    // pushbutton on PA10; LED on PB8

    // initialize ports
    GPIO_init(GPIOA);
    GPIO_init(GPIOB);

    // configure pin PA10 as input
    GPIO_config_input(GPIOA, 10);

    // configure pin PB8 as output
    GPIO_config_output(GPIOB, 8);

    // infinite loop
    for (;;) {
        int pinval = GPIO_read(GPIOA, 10);
        GPIO_write(GPIOB, 8, !pinval);
    }
}
```

First Example: Read a Pushbutton and lit the LED

Let's replace input reading function with SFR

```
#include "stm32_unict_lib.h"

int main()
{
    // pushbutton on PA10; LED on PB8

    // initialize ports
    GPIO_init(GPIOA);
    GPIO_init(GPIOB);

    // configure pin PA10 as input
    GPIO_config_input(GPIOA, 10);

    // configure pin PB8 as output
    GPIO_config_output(GPIOB, 8);

    // infinite loop
    for (;;) {
        int pinval = (GPIOA->IDR & (1 << 10)) != 0;
        GPIO_write(GPIOB, 8, !pinval);
    }
}
```

First Example: Read a Pushbutton and lit the LED

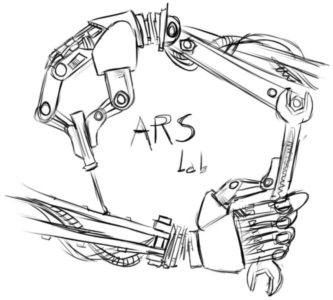
Let's replace output writing function with SFR

```
#include "stm32_unict_lib.h"

int main()
{
    // pushbutton on PA10; LED on PB8

    // initialize ports
    GPIO_init(GPIOA);
    GPIO_init(GPIOB);

    // configure pin PA10 as input
    GPIO_config_input(GPIOA, 10);

    // configure pin PB8 as output
    GPIO_config_output(GPIOB, 8);

    // infinite loop
    for (;;) {
        int pinval = (GPIOA->IDR & (1 << 10)) != 0;
        GPIOB->ODR = (GPIOB->ODR & ~(int32_t)0x100) | (!pinval << 8);
    }
}
```

Using the Special Function Registers of the Digital I/O interface of STM32

Corrado Santoro

ARSLAB - Autonomous and Robotic Systems Laboratory
Dipartimento di Matematica e Informatica - Università di Catania, Italy
santoro@dmi.unict.it



L.A.P. 1 Course