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.S.M. Course
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 **Pxy**, 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”.
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
- **OTYPE**: 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**

**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

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>
**Output Type Register**

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
Push-Pull vs Open-Drain

**PUSH-PULL MODE**

- OUTPUT PORT 0
  - OUTPUT PIN 0

- OUTPUT PORT 1
  - OUTPUT PIN 1

**OPEN-DRAIN MODE**

- OUTPUT PORT 0
  - OUTPUT PIN 0

- OUTPUT PORT 1
  - OUTPUT PIN 1

Output is "floating"
Output Speed Register

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

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
</tbody>
</table>
### Pull-up/Pull-Down Register

**PUPDR** defines the presence of a pull-up or pull-down resistor (or none) at the GPIO pin.

Each pin has 2 bits that permits the following configurations:

- **00**: No pull-up/pull-down
  - When input is **floating**, state is **unknown**

- **01**: Pull-up
  - When input is **floating**, state is **forced to “1”**

- **10**: Pull-down
  - When input is **floating**, state is **forced to “0”**

<table>
<thead>
<tr>
<th>Bit 31</th>
<th>Bit 30</th>
<th>Bit 29</th>
<th>Bit 28</th>
<th>Bit 27</th>
<th>Bit 26</th>
<th>Bit 25</th>
<th>Bit 24</th>
<th>Bit 23</th>
<th>Bit 22</th>
<th>Bit 21</th>
<th>Bit 20</th>
<th>Bit 19</th>
<th>Bit 18</th>
<th>Bit 17</th>
<th>Bit 16</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit 15</th>
<th>Bit 14</th>
<th>Bit 13</th>
<th>Bit 12</th>
<th>Bit 11</th>
<th>Bit 10</th>
<th>Bit 9</th>
<th>Bit 8</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**Corrado Santoro**

*Digital I/O in STM32 Family with SFR*
Data Input/Output 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 (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 can be also performed through the **BRR** register.

To **reset a pin**, a “1” must be written in the correspondent bit.

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BR15</th>
<th>BR14</th>
<th>BR13</th>
<th>BR12</th>
<th>BR11</th>
<th>BR10</th>
<th>BR9</th>
<th>BR8</th>
<th>BR7</th>
<th>BR6</th>
<th>BR5</th>
<th>BR4</th>
<th>BR3</th>
<th>BR2</th>
<th>BR1</th>
<th>BR0</th>
</tr>
</thead>
<tbody>
<tr>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
</tr>
</tbody>
</table>
```c
#include "stm32_unict_lib.h"

int main()
{
    // pushbutton on PB10; LED on PB8

    // initialize ports
    GPIO_init(GPIOB);

    // configure pin PB10 as input
    GPIO_config_input(GPIOB, 10);

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

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

First Example: Read a Pushbutton and lit the LED
Let’s replace input reading function with SFR

```c
#include "stm32_unict_lib.h"

int main()
{
    // pushbutton on PB10; LED on PB8

    // initialize ports
    GPIO_init(GPIOB);

    // configure pin PB10 as input
    GPIO_config_input(GPIOB, 10);

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

    // infinite loop
    for (;;) {
        int pinval = (GPIOB->IDR & (1 << 10)) != 0;
        /* pinval is "1" when pushbutton is released */
        /* pinval is "0" when pushbutton is pressed */
        GPIO_write(GPIOB, 8, !pinval);
    }
}
```
Let's replace output writing function with SFR

```c
#include "stm32_unict_lib.h"

int main()
{
    // pushbutton on PB10; LED on PB8
    // initialize ports
    GPIO_init(GPIOB);

    // configure pin PB10 as input
    GPIO_config_input(GPIOB, 10);

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

    // infinite loop
    for (;;) {
        int pinval = (GPIOB->IDR & (1 << 10)) != 0;
        /* pinval is "1" when pushbutton is released */
        /* pinval is "0" when pushbutton is pressed */
        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.S.M. Course