通用输入输出1

程晨闻
东南大学电气工程学院

- TM4C1294XL开发板架构
  - 供电，复位，时钟，调试器，按键，LED
- TM4C1294控制器
  - 32位架构，三级流水线，哈佛总线，浮点型，位操作，120MHz…
- 开发软件CCS的安装
- Tivaware的安装
  - 延时，位或，位与
- CCS编程环境

作业点评

- 1：十进制数10（0x0000 000A），-100（0xFFFF FF9C），-1000（0xFFFF FC18），10000（0x0000 2710）转换为16进制（补码，32位存储）。
- 2：定义两个变量 int8_t m=100, n=-100;他们在计算机中是以__二__进制的形式存储的，具体存储的数据为：m:__0110 0100b__，n:__1001 1100b__。
- 3：定义一个字符串：char hello[] = “Hello”;则存储器hello地址处存放的数据是_0x48_，hello+1处存放的数据是__0x65__，hello+2处存放的数据是__0x6C___，hello+3处存放的数据是___0x6C___，hello+4处存放的数据是__0x6F__，hello+5处存放的数据是__0x00___。（为方便书写，可用16进制填写，ASCII码）。
- 4：定义一个变量 int16_t k=1230。 把这个变量，从一个嵌入式系统传到另一个嵌入式系统，实际上传递的数据是__0x04CE__。(为方便书写，可用16进制填写）。

作业点评

- 5：存储器一般由_存储体_、_控制电路_、_地址译码电路_、_数据缓冲电路_四部分组成。
- 6：SRAM的基本存储单元是_触发器_、DRAM的基本存储单元是_存储电容_、U盘的基本存储单元是_浮栅场效应管_。
- 7：TM4C1294的程序存储器是_NOR_类型的FLASH。
- 8：TM4C1294的FLASH中，一个uint32_t类型的数据存放在地址0x0000 0000处，的内容是 0x2000 036C，那么0x0000 0000处存放的字节是_0x6C_，0x0000 0001处存放的字节是_0x03_，0x0000 0002处存放的字节是_0x00_，0x0000 0003处存放的字节是_0x20_。

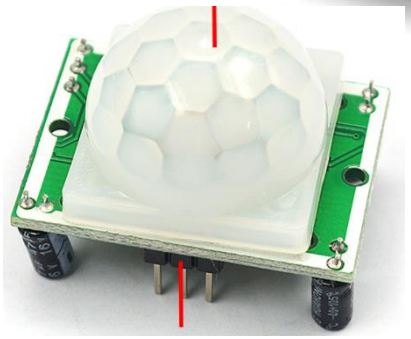
- 通用输入输出端口的基本概念
- TM4C1294的GPIO
- 控制寄存器的访问
- 控制寄存器的功能

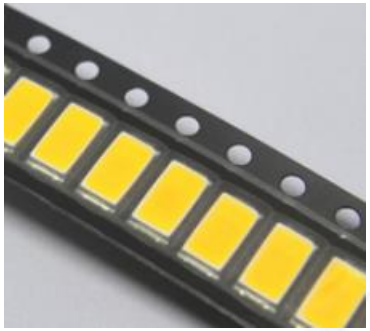
- 数据手册的阅读
- API文档的阅读

- **通用输入输出接口（GPIO）**
  - GPIO=General Purpose Input Output，通用输入输出。有时候简称为“IO口”
  - MCU的最基本外设
- **端口（Port）**
  - CPU和外设进行通信的媒介；一个端口**一般**有8个引脚
  - TM4C1294有15个端口：(Port A, Port B, Port C, Port D, Port E, Port F, Port G, Port H, Port J, Port K, Port L, Port M, Port N, Port P, Port Q)
  - 共90个引脚
- **引脚（管脚,Pin）**
  - MCU与外界连接的独立的导线，隶属于某一个端口
  - 对其中一个引脚进行操作的时候不会影响到其他引脚
  - 可以设置为多种功能，单独设置为数字输入或者数字输出，有些引脚还可以用作模拟输入和外设中断

GPIO的应用

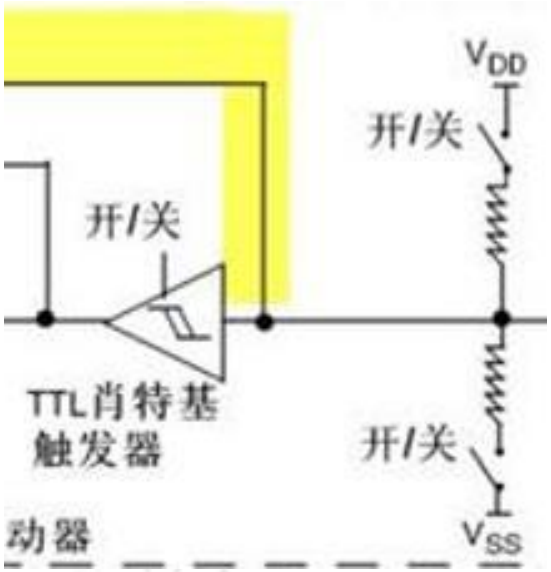
- 常用的IO驱动电路
- 简单按键开关
- 矩阵键盘
- 继电器
- LED灯
- 数码管
- 传感器输入

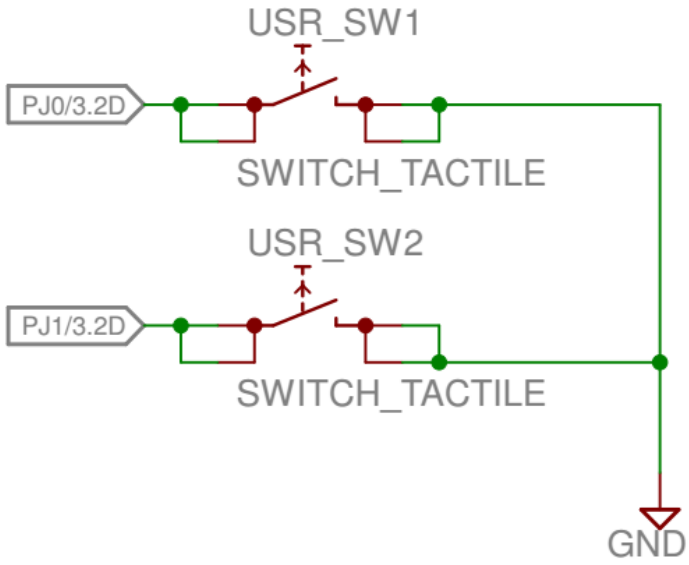




引脚的输入方式

- **浮空输入**（上下两个电阻上电开关都打开，输入信号必须有确定的电平，不能悬空）
- **上拉输入**（上面的电阻上电开关闭合，悬空时为高电平）
- **下拉输入**（下面的电阻上电开关闭合，悬空时为低电平）
- **模拟输入**（上下两个电阻上电开关都打开，接模拟信号）

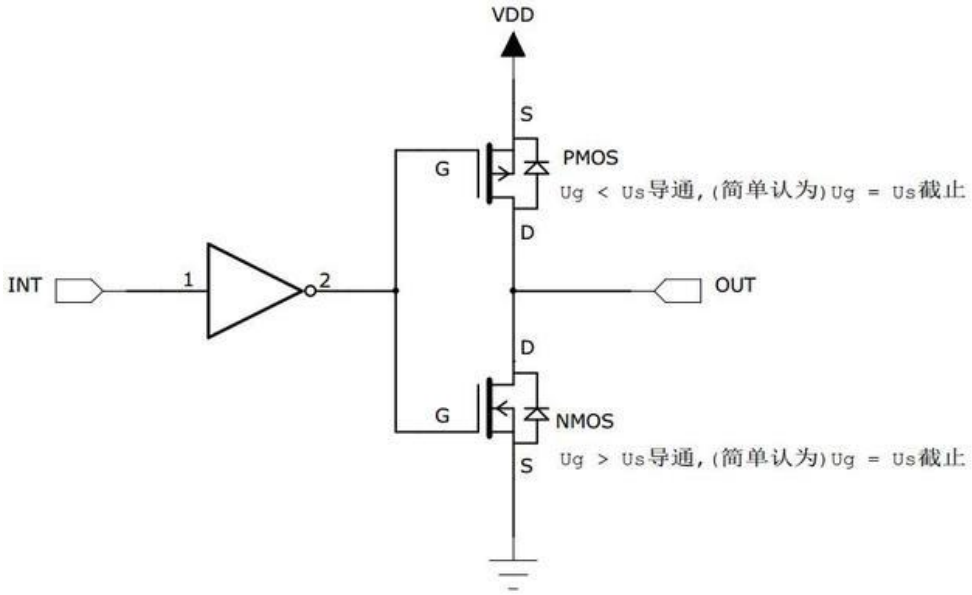




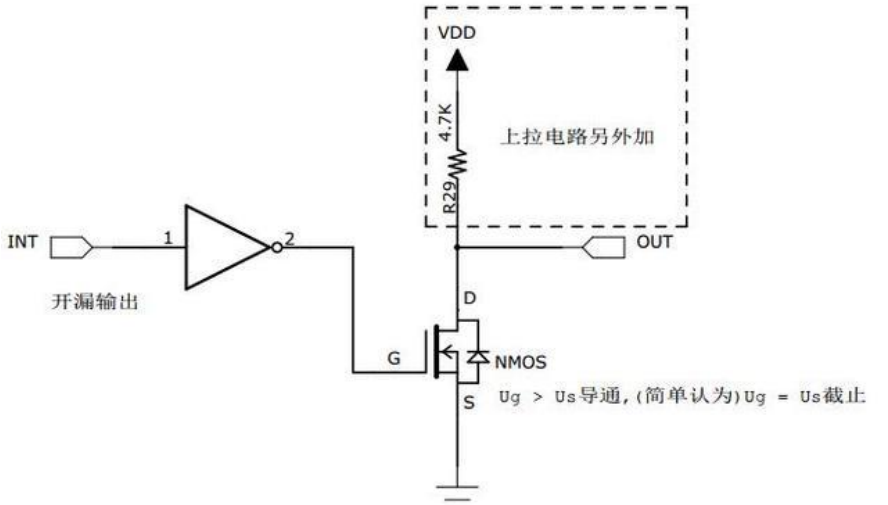
简单按键需配置为上拉输入

➢ 引脚的输出方式

- **推挽输出**（高电平VDD，低电平GND）
- **开漏输出**（需要**上拉电阻**才能工作，高电平可以为VDD以下的任意电压，低电平GND，可用于5V系统与3.3V系统连接）



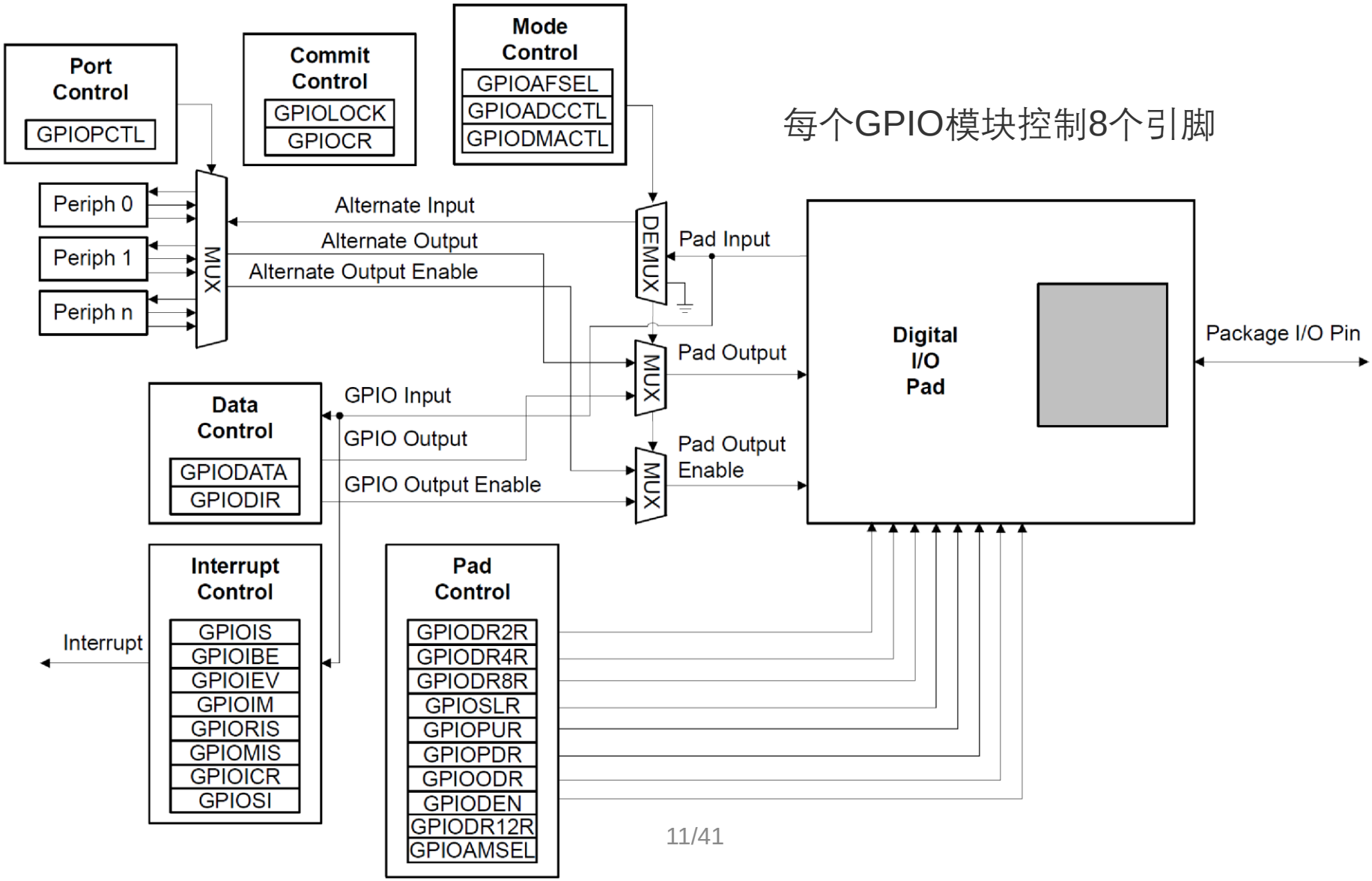
推挽输出



开漏输出

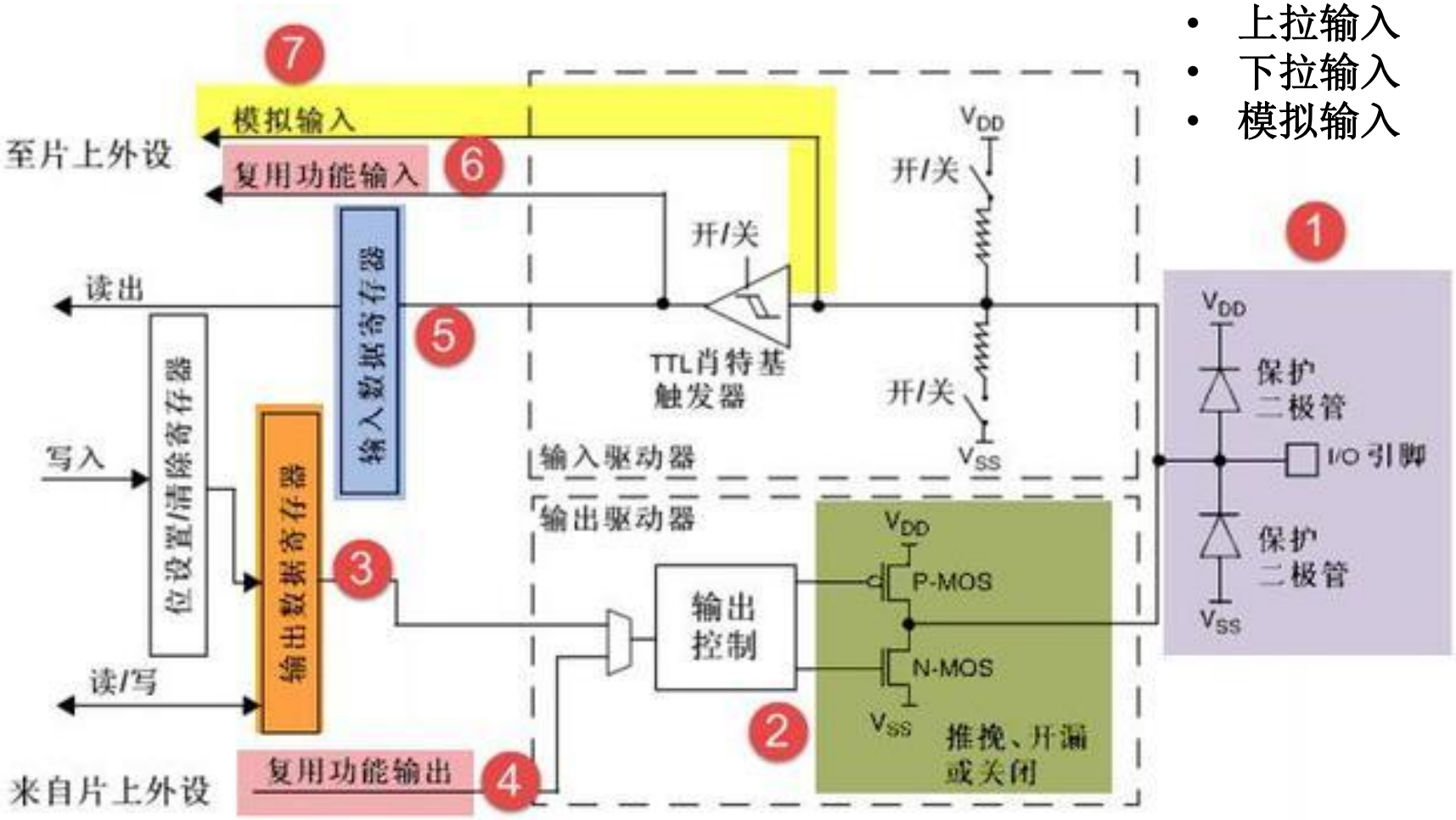
TM4C1294的GPIO

每个GPIO模块控制8个引脚



输入电路：

- 浮空输入
- 上拉输入
- 下拉输入
- 模拟输入

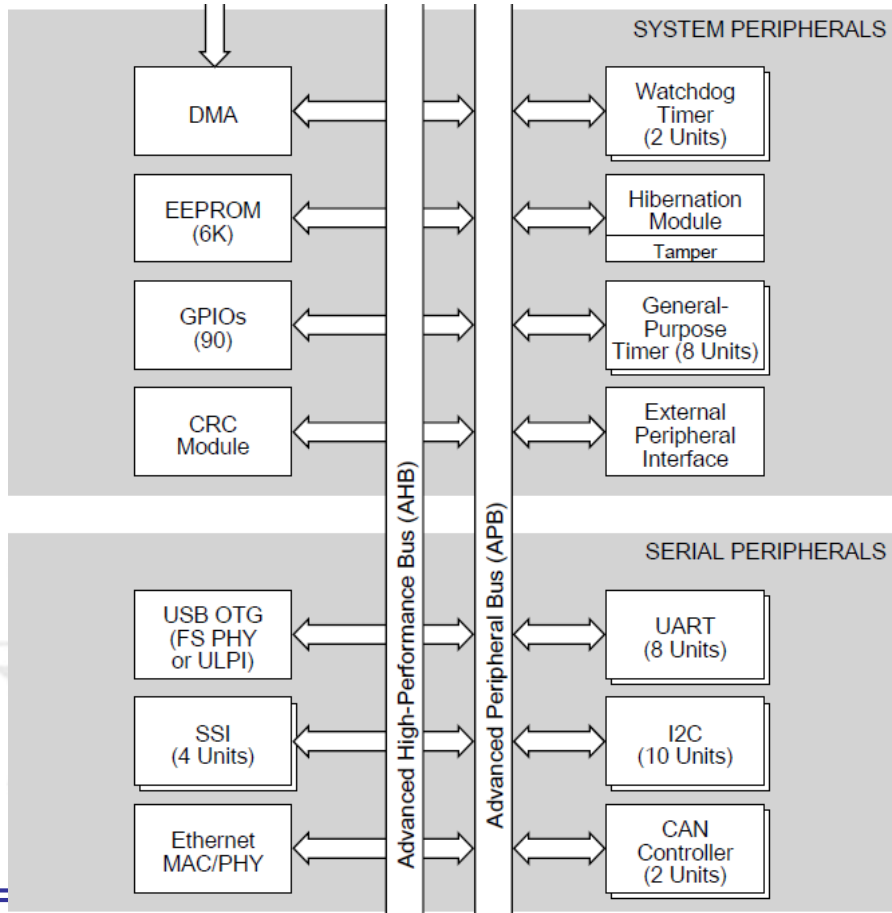


输出电路：

- 开漏输出
- 推挽输出

➢ 外设寄存器

- **控制**引脚的具体功能
- 外设寄存器可以通过**系统总线**访问，在系统总线上，有确定的地址
- 端口的控制寄存器组的地址，由基地址和偏移地址组成
- 实际地址为：**基地址+偏移地址**



有两条系统总线可以与外设通信，分别是APB和AHB（AHB比APB快）

➢ 各个端口寄存器组在AHB上的基地址

0x4005.8000	0x4005.8FFF	GPIO Port A (AHB aperture)
0x4005.9000	0x4005.9FFF	GPIO Port B (AHB aperture)
0x4005.A000	0x4005.AFFF	GPIO Port C (AHB aperture)
0x4005.B000	0x4005.BFFF	GPIO Port D (AHB aperture)
0x4005.C000	0x4005.CFFF	GPIO Port E (AHB aperture)
0x4005.D000	0x4005.DFFF	GPIO Port F (AHB aperture)
0x4005.E000	0x4005.EFFF	GPIO Port G (AHB aperture)
0x4005.F000	0x4005.FFFF	GPIO Port H (AHB aperture)
0x4006.0000	0x4006.0FFF	GPIO Port J (AHB aperture)
0x4006.1000	0x4006.1FFF	GPIO Port K (AHB aperture)
0x4006.2000	0x4006.2FFF	GPIO Port L (AHB aperture)
0x4006.3000	0x4006.3FFF	GPIO Port M (AHB aperture)
0x4006.4000	0x4006.4FFF	GPIO Port N (AHB aperture)
0x4006.5000	0x4006.5FFF	GPIO Port P (AHB aperture)
0x4006.6000	0x4006.6FFF	GPIO Port Q (AHB aperture)

➢ 各个端口寄存器组在APB上的基地址

0x4000.4000	0x4000.4FFF	GPIO Port A
0x4000.5000	0x4000.5FFF	GPIO Port B
0x4000.6000	0x4000.6FFF	GPIO Port C
0x4000.7000	0x4000.7FFF	GPIO Port D
0x4002.4000	0x4002.4FFF	GPIO Port E
0x4002.5000	0x4002.5FFF	GPIO Port F
0x4002.6000	0x4002.6FFF	GPIO Port G
0x4002.7000	0x4002.7FFF	GPIO Port H
0x4003.D000	0x4003.DFFF	GPIO Port J

- 虽然地址有所不同，但功能是一致的
- 推荐使用**AHB**的地址，因为访问更快

- hw_memmap.h中，使用宏定义，定义了端口基地址
- AHB地址的宏定义：

```
#define GPIO_PORTA_AHB_BASE     0x40058000  // GPIO Port A (high speed)
#define GPIO_PORTB_AHB_BASE     0x40059000  // GPIO Port B (high speed)
#define GPIO_PORTC_AHB_BASE     0x4005A000  // GPIO Port C (high speed)
#define GPIO_PORTD_AHB_BASE     0x4005B000  // GPIO Port D (high speed)
#define GPIO_PORTE_AHB_BASE     0x4005C000  // GPIO Port E (high speed)
#define GPIO_PORTF_AHB_BASE     0x4005D000  // GPIO Port F (high speed)
#define GPIO_PORTG_AHB_BASE     0x4005E000  // GPIO Port G (high speed)
#define GPIO_PORTH_AHB_BASE     0x4005F000  // GPIO Port H (high speed)
#define GPIO_PORTJ_AHB_BASE     0x40060000  // GPIO Port J (high speed)
#define GPIO_PORTK_BASE         0x40061000  // GPIO Port K
#define GPIO_PORTL_BASE         0x40062000  // GPIO Port L
#define GPIO_PORTM_BASE         0x40063000  // GPIO Port M
#define GPIO_PORTN_BASE         0x40064000  // GPIO Port N
#define GPIO_PORTP_BASE         0x40065000  // GPIO Port P
#define GPIO_PORTQ_BASE         0x40066000  // GPIO Port Q
```

- hw_memmap.h中，使用宏定义，定义了端口基地址

```
25 #include <stdint.h>
26 #include <stdbool.h>
27 #include "inc/hw_types.h"
28 #include "inc/hw_memmap.h"
29 #include "driverlib/sysctl.h"
30 #include "driverlib/gpio.h"
```

- APB地址的宏定义：

```
#define GPIO_PORTA_BASE         0x40004000  // GPIO Port A
#define GPIO_PORTB_BASE         0x40005000  // GPIO Port B
#define GPIO_PORTC_BASE         0x40006000  // GPIO Port C
#define GPIO_PORTD_BASE         0x40007000  // GPIO Port D
#define GPIO_PORTE_BASE         0x40024000  // GPIO Port E
#define GPIO_PORTF_BASE         0x40025000  // GPIO Port F
#define GPIO_PORTG_BASE         0x40026000  // GPIO Port G
#define GPIO_PORTH_BASE         0x40027000  // GPIO Port H
#define GPIO_PORTJ_BASE         0x4003D000  // GPIO Port J
```

- 各个端口配置寄存器组的基地址不同
- 负责控制具体功能的偏移地址是相同的

Offset	Name	Type	Reset	Description	See page
0x000	GPIODATA	RW	0x0000.0000	GPIO Data	757
0x400	GPIODIR	RW	0x0000.0000	GPIO Direction	
0x404	GPIOIS	RW	0x0000.0000	GPIO Interrupt Sense	
0x408	GPIOIBE	RW	0x0000.0000	GPIO Interrupt Both Edges	
0x50C	GPIOODR	RW	0x0000.0000	GPIO Open Drain Select	
0x510	GPIOPUR	RW	-	GPIO Pull-Up Select	
0x514	GPIOPDR	RW	0x0000.0000	GPIO Pull-Down Select	
0x518	GPIOSLR	RW	0x0000.0000	GPIO Slew Rate Control Select	778
0x51C	GPIODEN	RW	-	GPIO Digital Enable	779

GPIO Data (GPIODATA)

GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (AHB) base: 0x4005.C000
GPIO Port F (AHB) base: 0x4005.D000
GPIO Port G (AHB) base: 0x4005.E000
GPIO Port H (AHB) base: 0x4005.F000
GPIO Port J (AHB) base: 0x4006.0000
GPIO Port K (AHB) base: 0x4006.1000
GPIO Port L (AHB) base: 0x4006.2000
GPIO Port M (AHB) base: 0x4006.3000
GPIO Port N (AHB) base: 0x4006.4000
GPIO Port P (AHB) base: 0x4006.5000
GPIO Port Q (AHB) base: 0x4006.6000
Offset 0x000
Type RW, reset 0x0000.0000

- 偏移地址在inc/hw_gpio.h进行了定义，如果用到的话，需要包含该**头文件 "inc/hw_gpio.h"**

```
25 #include <stdint.h>
26 #include <stdbool.h>
27 #include "inc/hw_gpio.h"
28 #include "inc/hw_ints.h"
29 #include "inc/hw_memmap.h"
```

```
#define GPIO_O_DATA             0x00000000  // GPIO Data
#define GPIO_O_DIR              0x00000400  // GPIO Direction
#define GPIO_O_IS               0x00000404  // GPIO Interrupt Sense
#define GPIO_O_IBE              0x00000408  // GPIO Interrupt Both Edges
#define GPIO_O_IEV              0x0000040C  // GPIO Interrupt Event
#define GPIO_O_IM               0x00000410  // GPIO Interrupt Mask
#define GPIO_O_PUR              0x00000510  // GPIO Pull-Up Select
#define GPIO_O_PDR              0x00000514  // GPIO Pull-Down Select
#define GPIO_O_SLR              0x00000518  // GPIO Slew Rate Control Select
#define GPIO_O_DEN              0x0000051C  // GPIO Digital Enable
```

- 某个端口控制寄存器在总线上的实际地址为：**基地址+偏移地址**
- PORTn端口的GPIODIR寄存器:
  - HWREG(GPIO_PORTN_BASE + GPIO_O_DIR)

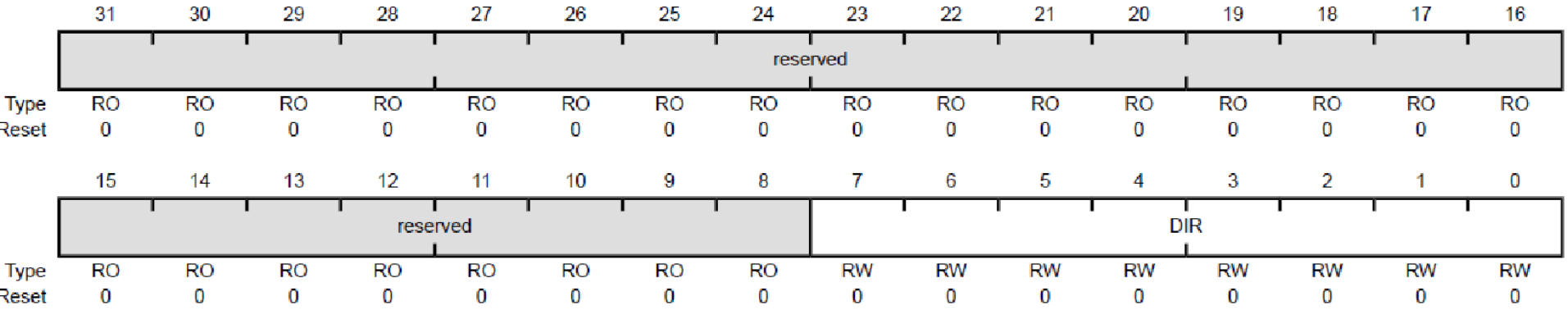
```
#define HWREG(x)
\
        (*((volatile uint32_t *)(x)))
```

```
#define GPIO_PORTA_BASE         0x40004000  // GPIO Port A
#define GPIO_PORTB_BASE         0x40005000  // GPIO Port B
…


#define GPIO_O_DATA             0x00000000  // GPIO Data
#define GPIO_O_DIR              0x00000400  // GPIO Direction
#define GPIO_O_IS               0x00000404  // GPIO Interrupt Sense
…
```

方向寄存器 GPIODIR

- 只有低八位有实际作用，用于控制该端口的八个引脚的方向，每一位控制一个引脚
- 0为输入，1为输出



Bit/Field	Name	Type	Reset	Description
31:8	reserved	RO	0x0000.00	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
7:0	DIR	RW	0x00	GPIO Data Direction Value Description 0 Corresponding pin is an input. 1 Corresponding pins is an output.

如果要将PORTn的Pin0设置为输出：

```
HWREG(GPIO_PORTN_BASE + GPIO_O_DIR)=0x01;
```

如果要将PORTn的Pin1设置为输出：

```
HWREG(GPIO_PORTN_BASE + GPIO_O_DIR)=0x02;
```

如果要将PORTn的Pin2设置为输出：

```
HWREG(GPIO_PORTN_BASE + GPIO_O_DIR)=0x04;
```

如果要将PORTn的Pin3设置为输出：

```
HWREG(GPIO_PORTN_BASE + GPIO_O_DIR)=0x08;
```

如果要将PORTn的Pin0和Pin3都设置为输出：

```
HWREG(GPIO_PORTN_BASE + GPIO_O_DIR)=0x09;
```

- 由于这种每一位控制一个引脚的情况比较多，为了方便引用各个引脚，"**driverlib/gpio.h**"中对各个引脚做了宏定义：

```
#define GPIO_PIN_0              0x00000001  // GPIO pin 0
#define GPIO_PIN_1              0x00000002  // GPIO pin 1
#define GPIO_PIN_2              0x00000004  // GPIO pin 2
#define GPIO_PIN_3              0x00000008  // GPIO pin 3
#define GPIO_PIN_4              0x00000010  // GPIO pin 4
#define GPIO_PIN_5              0x00000020  // GPIO pin 5
#define GPIO_PIN_6              0x00000040  // GPIO pin 6
#define GPIO_PIN_7              0x00000080  // GPIO pin 7
```

这样，如果要将PORTn的Pin3设置为输出：

HWREG(GPIO_PORTN_BASE + GPIO_O_DIR)= GPIO_PIN_3;

也可以用或操作，控制设置多个引脚

如果要将PORTn的Pin0和Pin3都设置为输出

HWREG(GPIO_PORTN_BASE + GPIO_O_DIR)= GPIO_PIN_0 | GPIO_PIN_3;

➢ TivaWare 库函数：（端口方向控制相关）

– void GPIOPinTypeGPIOInput (uint32_t ui32Port, uint8_t ui8Pins)

– void GPIOPinTypeGPIOOutput (uint32_t ui32Port, uint8_t ui8Pins)

– void GPIOPinTypeGPIOOutputOD (uint32_t ui32Port, uint8_t ui8Pins)

➢ GPIOPinTypeGPIOInput (uint32_t ui32Port, uint8_t ui8Pins)

- 完成将引脚设置为**悬空输入**的相关设置
- ui32Port指定端口的基地址
- ui8Pins指定把哪些引脚设置为输入，ui8Pins为8位无符号变量，每一位代表一个引脚
- SW-TM4C-DRL-UG-2.2.0.295.pdf文档中的具体说明如下：

14.2.3.29 GPIOPinTypeGPIOInput

Configures pin(s) for use as GPIO inputs.

Prototype:

```
void
GPIOPinTypeGPIOInput(uint32_t ui32Port,
                     uint8_t ui8Pins)
```

Parameters:

ui32Port is the base address of the GPIO port.

ui8Pins is the bit-packed representation of the pin(s).

Description:

The GPIO pins must be properly configured in order to function correctly as GPIO inputs. This function provides the proper configuration for those pin(s).

The pin(s) are specified using a bit-packed byte, where each bit that is set identifies the pin to be accessed, and where bit 0 of the byte represents GPIO port pin 0, bit 1 represents GPIO port pin 1, and so on.

➢ GPIOPinTypeGPIOOutput (uint32_t ui32Port, uint8_t ui8Pins)

– 完成将引脚设置为**推挽输出**的相关设置
– ui32Port指定端口的基地址
– ui8Pins指定把哪些引脚设置为输出
– ui8Pins为8位无符号变量，每一位代表一个引脚
– SW-TM4C-DRL-UG-2.2.0.295.pdf文档中的具体说明如下：

14.2.3.30 GPIOPinTypeGPIOOutput

Configures pin(s) for use as GPIO outputs.

Prototype:

```
void
GPIOPinTypeGPIOOutput(uint32_t ui32Port,
                      uint8_t ui8Pins)
```

Parameters:

ui32Port is the base address of the GPIO port.

ui8Pins is the bit-packed representation of the pin(s).

Description:

The GPIO pins must be properly configured in order to function correctly as GPIO outputs. This function provides the proper configuration for those pin(s).

The pin(s) are specified using a bit-packed byte, where each bit that is set identifies the pin to be accessed, and where bit 0 of the byte represents GPIO port pin 0, bit 1 represents GPIO port pin 1, and so on.

➢ GPIOPinTypeGPIOOutputOD (uint32_t ui32Port, uint8_t ui8Pins)

- 完成将引脚设置为**开漏输出**的相关设置
- ui32Port指定端口的基地址
- ui8Pins指定把哪些引脚设置为输出
- ui8Pins为8位无符号变量，每一位代表一个引脚。
- SW-TM4C-DRL-UG-2.2.0.295.pdf文档中的具体说明如下：

14.2.3.31 GPIOPinTypeGPIOOutputOD

Configures pin(s) for use as GPIO open drain outputs.

Prototype:

```
void
GPIOPinTypeGPIOOutputOD(uint32_t ui32Port,
                        uint8_t ui8Pins)
```

Parameters:

ui32Port is the base address of the GPIO port.

ui8Pins is the bit-packed representation of the pin(s).

Description:

The GPIO pins must be properly configured in order to function correctly as GPIO outputs. This function provides the proper configuration for those pin(s).

The pin(s) are specified using a bit-packed byte, where each bit that is set identifies the pin to be accessed, and where bit 0 of the byte represents GPIO port pin 0, bit 1 represents GPIO port pin 1, and so on.

- 这三个函数已经被封装在driverlib库中
- TivaWare提供了这三个函数的源代码，位于C:\ti\TivaWare_C_Series-2.2.0.295\driverlib\gpio.c和同目录下的gpio.h中。

```
void
GPIOPinTypeGPIOOutput(uint32_t ui32Port, uint8_t ui8Pins)
{
    // Check the arguments.
    ASSERT(_GPIOBaseValid(ui32Port));
    // Set the pad(s) for standard push-pull operation.
    GPIOPadConfigSet(ui32Port, ui8Pins, GPIO_STRENGTH_2MA,
GPIO_PIN_TYPE_STD);
    // Make the pin(s) be outputs.
    GPIODirModeSet(ui32Port, ui8Pins, GPIO_DIR_MODE_OUT);
}
```

- 这三个函数已经被封装在driverlib库中，无法在工程中看到源代码
- TivaWare提供了这三个函数的源代码，位于C:\ti\TivaWare_C_Series-2.2.0.295\driverlib\gpio.c和同目录下的gpio.h中。

```
void
GPIOPinTypeGPIOInput(uint32_t ui32Port, uint8_t ui8Pins)
{
    // Check the arguments.
    ASSERT(_GPIOBaseValid(ui32Port));
    // Make the pin(s) be inputs.
    GPIODirModeSet(ui32Port, ui8Pins, GPIO_DIR_MODE_IN);
    // Set the pad(s) for standard push-pull operation.
    GPIOPadConfigSet(ui32Port, ui8Pins, GPIO_STRENGTH_2MA,
GPIO_PIN_TYPE_STD);
}
```

- 首先要确保输入的端口基地址是合法的：ASSERT(_GPIOBaseValid(ui32Port));
- 然后调用了GPIOPadConfigSet和GPIODirModeSet两个函数
- GPIODirModeSet：

  GPIODirModeSet(ui32Port, ui8Pins, GPIO_DIR_MODE_IN);

  – **传入三个参数：**
    - 端口基地址ui32Port
    - 要操作的引脚ui8Pins
    - GPIO_DIR_MODE_IN或GPIO_DIR_MODE_OUT

```
#define GPIO_DIR_MODE_IN        0x00000000  // Pin is a GPIO input
#define GPIO_DIR_MODE_OUT       0x00000001  // Pin is a GPIO output
```

GPIODirModeSet

- 设置指定引脚的方向
- GPIO_DIR_MODE_IN代表输入，GPIO_DIR_MODE_OUT代表输出

14.2.3.4 GPIODirModeSet

Sets the direction and mode of the specified pin(s).

Prototype:

```
void
GPIODirModeSet(uint32_t ui32Port,
               uint8_t ui8Pins,
               uint32_t ui32PinIO)
```

Parameters:

ui32Port is the base address of the GPIO port

ui8Pins is the bit-packed representation of the pin(s).

ui32PinIO is the pin direction and/or mode.

Description:

This function configures the specified pin(s) on the selected GPIO port as either input or output under software control, or it configures the pin to be under hardware control.

The parameter *ui32PinIO* is an enumerated data type that can be one of the following values:

- **GPIO_DIR_MODE_IN**
- **GPIO_DIR_MODE_OUT**
- **GPIO_DIR_MODE_HW**

where **GPIO_DIR_MODE_IN** specifies that the pin is programmed as a software controlled input, **GPIO_DIR_MODE_OUT** specifies that the pin is programmed as a software controlled output, and **GPIO_DIR_MODE_HW** specifies that the pin is placed under hardware control.

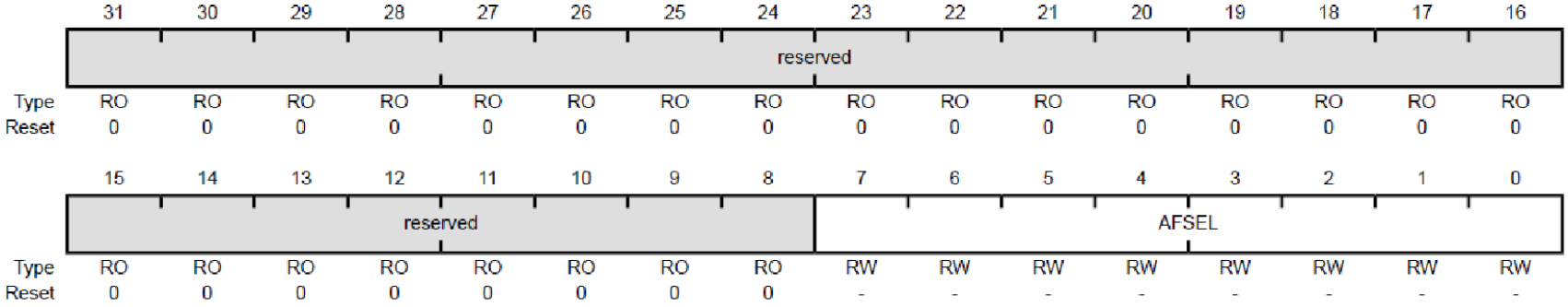
The pin(s) are specified using a bit-packed byte, where each bit that is set identifies the pin to be accessed, and where bit 0 of the byte represents GPIO port pin 0, bit 1 represents GPIO port pin 1, and so on.

GPIODirModeSet

```
void
GPIODirModeSet(uint32_t ui32Port, uint8_t ui8Pins, uint32_t ui32PinIO)
{
    // Check the arguments.
    ASSERT(_GPIOBaseValid(ui32Port));
    ASSERT((ui32PinIO == GPIO_DIR_MODE_IN) ||
           (ui32PinIO == GPIO_DIR_MODE_OUT) ||
           (ui32PinIO == GPIO_DIR_MODE_HW));
    // Set the pin direction and mode.
    HWREG(ui32Port + GPIO_O_DIR) = ((ui32PinIO & 1) ?
                                    (HWREG(ui32Port + GPIO_O_DIR) | ui8Pins) :
                                    (HWREG(ui32Port + GPIO_O_DIR) &
~(ui8Pins)));
    HWREG(ui32Port + GPIO_O_AFSEL) = ((ui32PinIO & 2) ?
                                      (HWREG(ui32Port + GPIO_O_AFSEL) |
                                       ui8Pins) :
                                      (HWREG(ui32Port + GPIO_O_AFSEL) &
                                       ~(ui8Pins)));
}
```

复用选择寄存器GPIOAFSEL

- 只有低八位有实际作用，用于控制该端口的八个引脚是否启用复用功能，每一位控制一个引脚。
- **0**为不复用，**1**为复用。复用的功能，由**GPIOPCTL**决定



Bit/Field	Name	Type	Reset	Description
31:8	reserved	RO	0x0000.00	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
7:0	AFSEL	RW	-	GPIO Alternate Function Select

Value	Description
0	The associated pin functions as a GPIO and is controlled by the GPIO registers.
1	The associated pin functions as a peripheral signal and is controlled by the alternate hardware function.

The reset value for this register is 0x0000.0000 for GPIO ports that are not listed in Table 10-1 on page 743.

➢ 输出驱动能力配置寄存器GPIODR2R，GPIODR4R，GPIODR8R，GPIODR12R，GPIOPP，GPIOPC

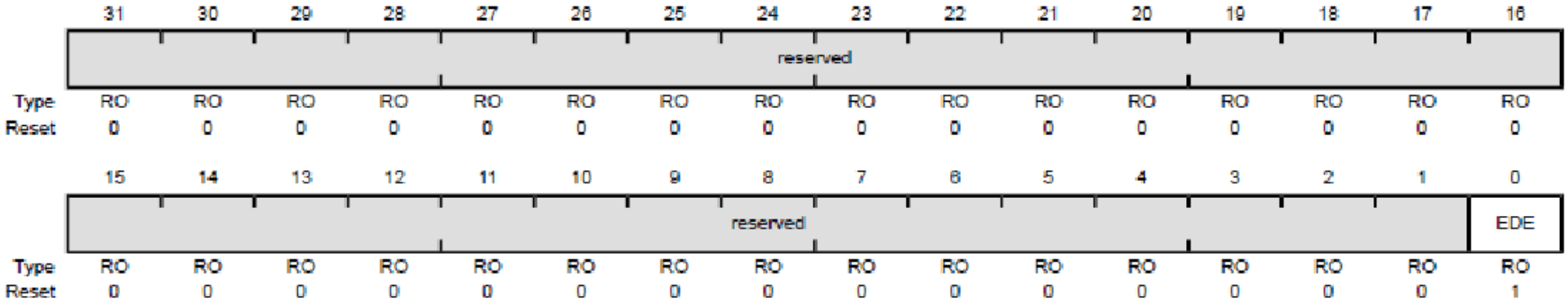
– 这几个寄存器用于控制引脚的**最大电流输出能力**，可以是2mA、4mA、8mA、10mA或者12mA；

– GPIODR2R，GPIODR4R，GPIODR8R，GPIODR12R这几个寄存器都是只有**低八位**有实际作用，用于控制该端口的对应的八个引脚的输出能力，每一位控制一个引脚；

– GPIOPP寄存器只有**一位**有作用，控制是否开启强化输出模式；

– GPIOPC寄存器的**低16位**有意义，每两位控制一个引脚。

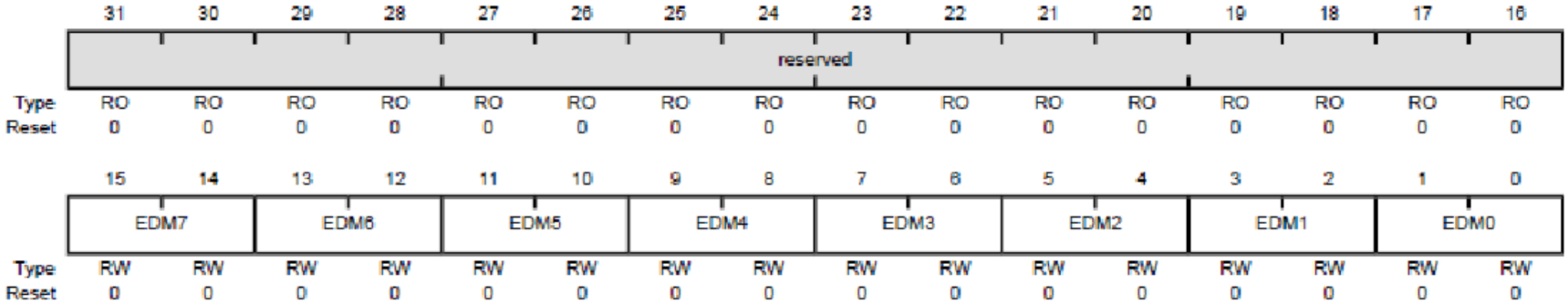
GPIOPP



Bit/Field	Name	Type	Reset	Description
31:1	reserved	RO	0x0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
0	EDE	RO	0x1	Extended Drive Enable This bit specifies whether the extended drive capabilities are provided. Extended drive is configured by the EDM bits in the GPIOPC register. Value Description 0 No Extended Drive Capability provided. 1 Extended Drive Capability provided.

GPIOPC

– 低16位有意义，每两位控制一个引脚



Bit/Field	Name	Type	Reset	Description
31:16	reserved	RO	0x0000.000	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
15:14	EDM7	RW	0	Extended Drive Mode Bit 7 Same encoding as EDM0, but applies to bit 7 of GPIO port.
13:12	EDM6	RW	0	Extended Drive Mode Bit 6 Same encoding as EDM0, but applies to bit 6 of GPIO port.
11:10	EDM5	RW	0	Extended Drive Mode Bit 5 Same encoding as EDM0, but applies to bit 5 of GPIO port.
9:8	EDM4	RW	0	Extended Drive Mode Bit 4 Same encoding as EDM0, but applies to bit 4 of GPIO port.

➢ GPIODR2R，GPIODR4R，GPIODR8R，GPIODR12R，GPIOPP，GPIOPC这几个寄存器对应的位的组合，决定了对应引脚的最大电流输出能力

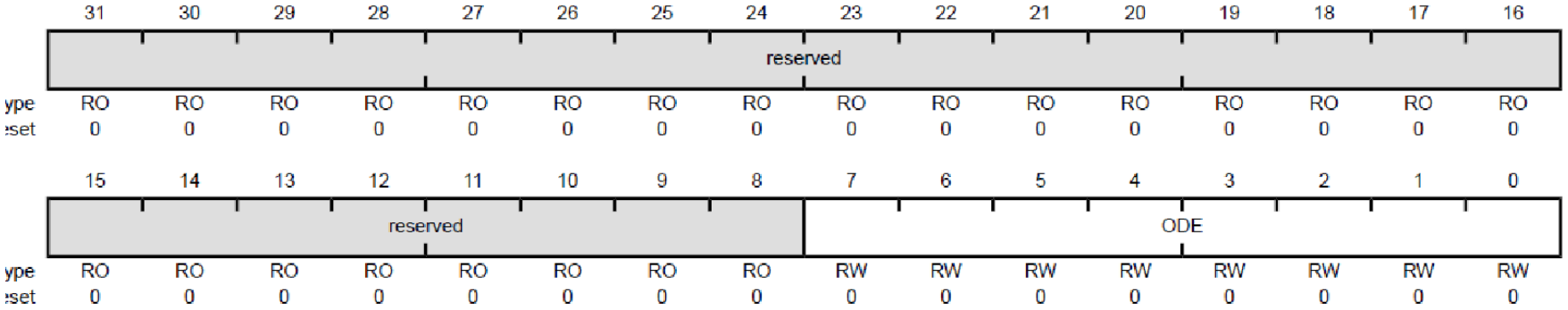
Table 10-13. GPIO Drive Strength Options

EDE (GPIOPP)	EDMn (GPIOPC)	GPIODR12R (+4mA)	GPIODR8R (+4mA)	GPIODR4R (+2mA)	GPIODR2R (2mA)	Drive (mA)
X	0x0	N/A	0	0	1	2
			0	1	0	4
			1	0	0	8
1	0x1	N/A	0	0	N/A	2
			0	1	N/A	4
			1	0	N/A	6
			1	1	N/A	8
1	0x3	0	0	0	N/A	2
		0	0	1	N/A	4
		0	1	0	N/A	6
		0	1	1	N/A	8
		1	1	0	N/A	10
		1	1	1	N/A	12
		1	0	N/A	N/A	N/A
1	0x2	N/A	N/A	N/A	N/A	N/A

开漏输出配置寄存器GPIOODR

– 只有低八位有实际作用，用于控制该端口的八个引脚是否启用开漏输出，每一位控制一个引脚；

– **0表示推挽输出，1表示开漏输出。**



Bit/Field	Name	Type	Reset	Description
31:8	reserved	RO	0x0000.00	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
7:0	ODE	RW	0x00	Output Pad Open Drain Enable

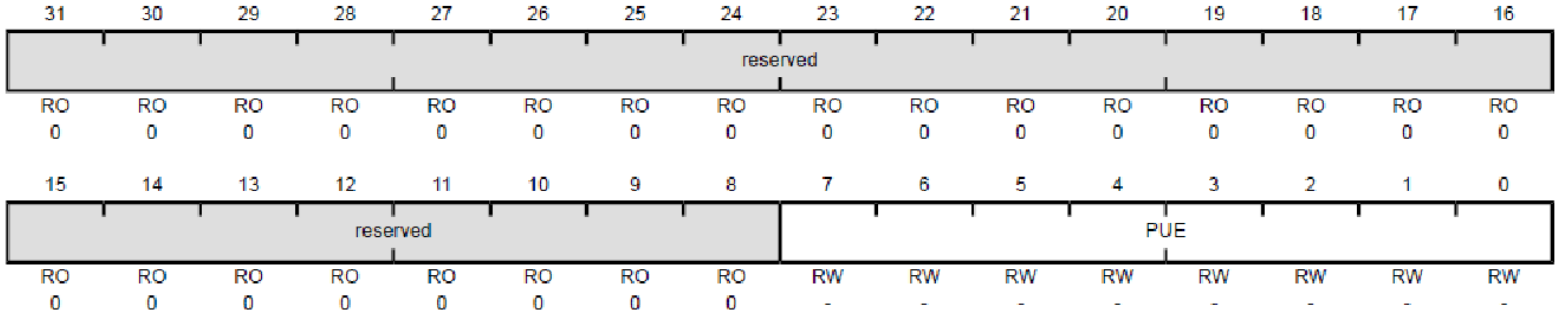
Value	Description
0	The corresponding pin is not configured as open drain.
1	The corresponding pin is configured as open drain.

➢ 上拉电阻与下拉电阻配置寄存器： GPIOPUR ， GPIOPDR

– 只有**低八位**有实际作用，用于控制该端口的八个引脚是否连接上拉电阻、下拉电阻，每一位控制一个引脚；

– GPIOPUR 对应的位为**0**表示该引脚**不连接上拉电阻**，GPIOPUR 对应的位为**1**表示该引脚**连接上拉电阻**；

– GPIOPDR对应的位为**0**表示该引脚**不连接下拉电阻**，GPIOPDR 对应的位为**1**表示该引脚连接**下拉电阻**。



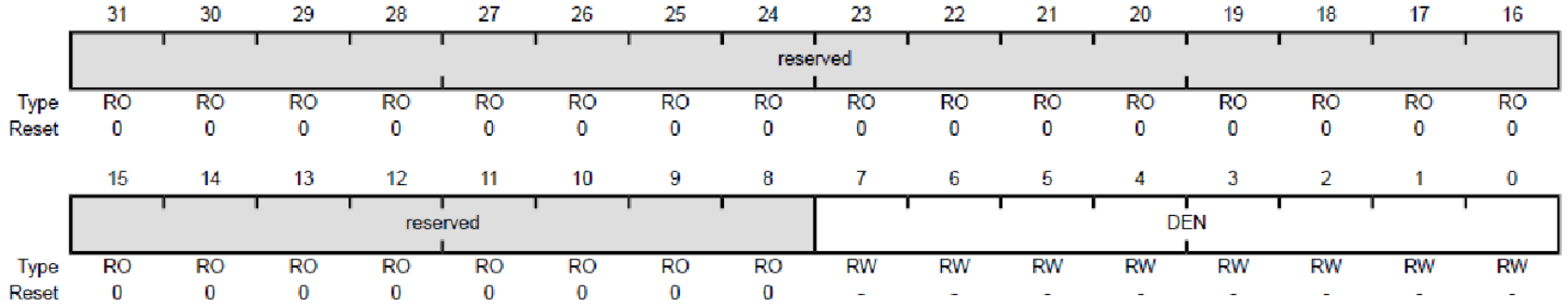
Bit/Field	Name	Type	Reset	Description
31:8	reserved	RO	0x0000.00	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
7:0	PUE	RW	-	Pad Weak Pull-Up Enable

Value	Description
0	The corresponding pin's weak pull-up resistor is disabled.
1	The corresponding pin's weak pull-up resistor is enabled.

➢ 数字逻辑功能使能寄存器GPIODEN

– 重置后，引脚自动配置为三态，断开引脚与GPIO模块的连接；

– 如果要使用引脚的**数字逻辑**功能，需要**使能GPIODEN**寄存器中相应的位。



Bit/Field	Name	Type	Reset	Description
31:8	reserved	RO	0x0000.00	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
7:0	DEN	RW	-	Digital Enable

Value	Description
0	The digital functions for the corresponding pin are disabled.
1	The digital functions for the corresponding pin are enabled.

The reset value for this register is 0x0000.0000 for GPIO ports that are not listed in Table 10-1 on page 741.

谢谢！