Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C

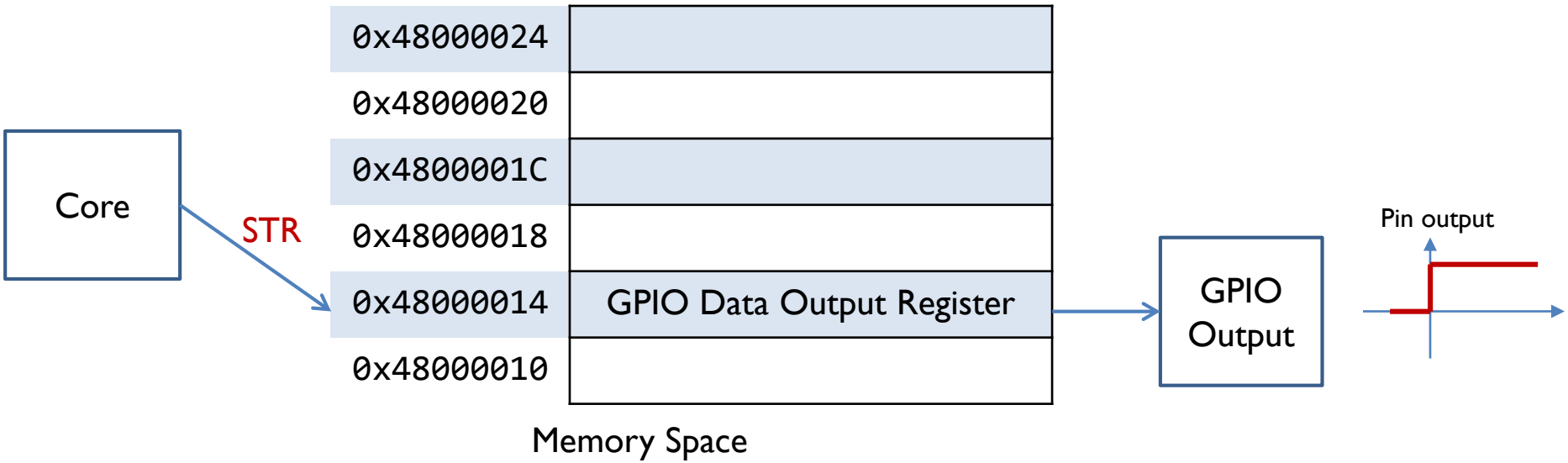
Chapter 14
GPIO

Dr. Yifeng Zhu
Electrical and Computer Engineering
University of Maine

Spring 2018

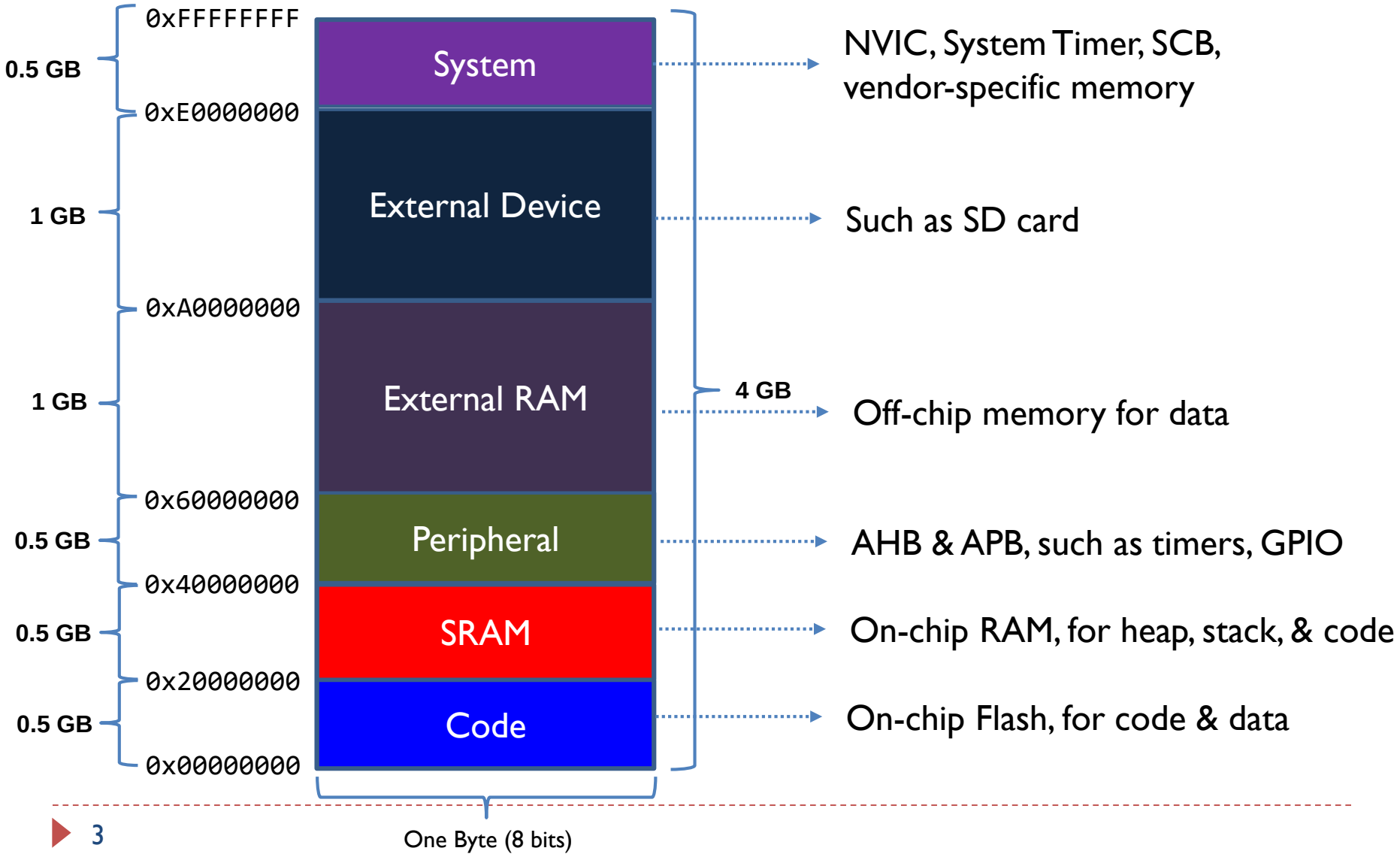
Interfacing Peripherals

- **Port-mapped I/O**
  - Use special CPU instructions: `Special_instruction Reg, Port`
- **Memory-mapped I/O**
  - A simpler and more convenient way to interface I/O devices
  - Each device registers is assigned to a memory address in the address space of the microprocessor
  - Use native CPU load/store instructions: `LDR/STR Reg, [Reg, #imm]`

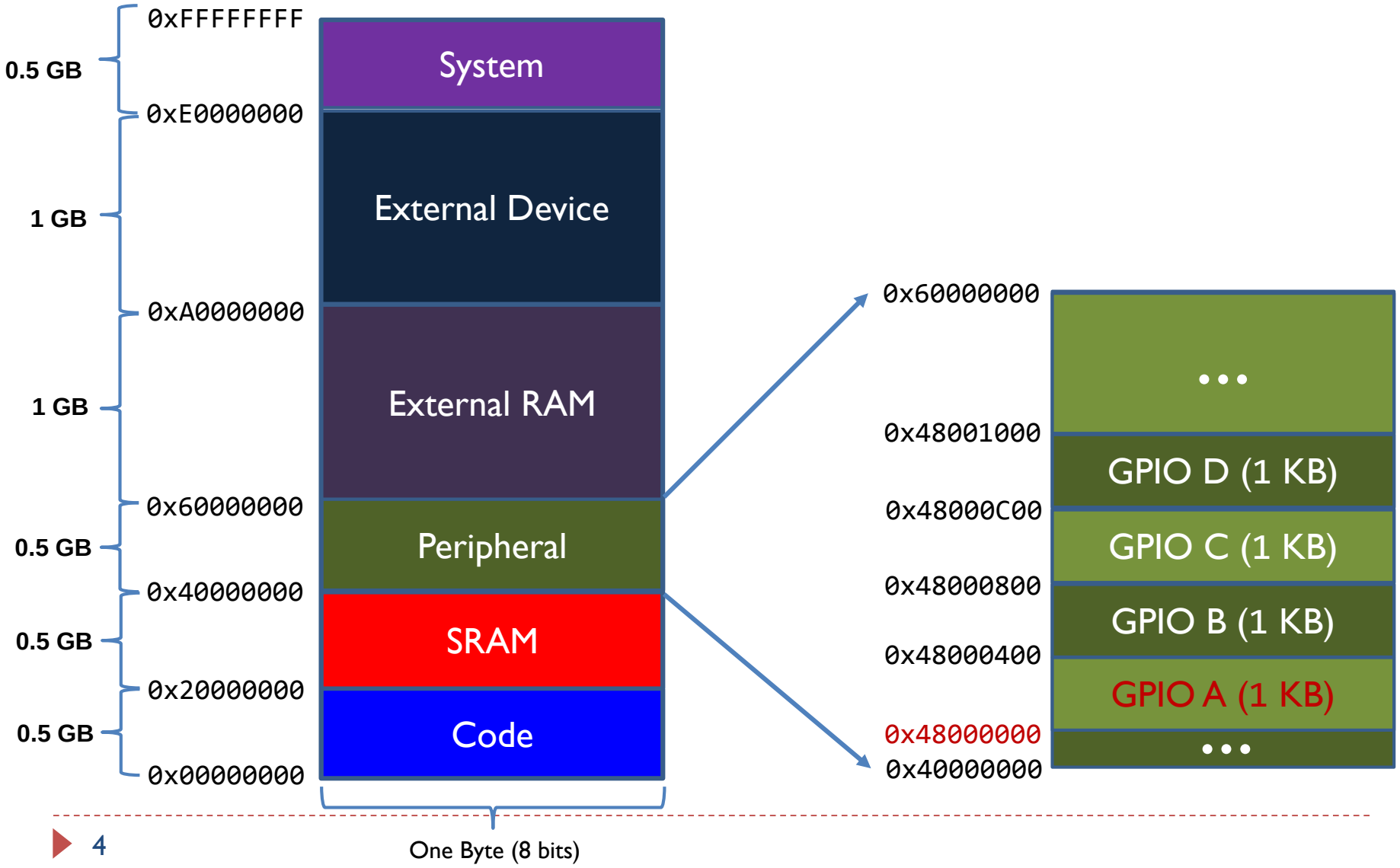


ARM Cortex-M microprocessors use memory-mapped I/O.

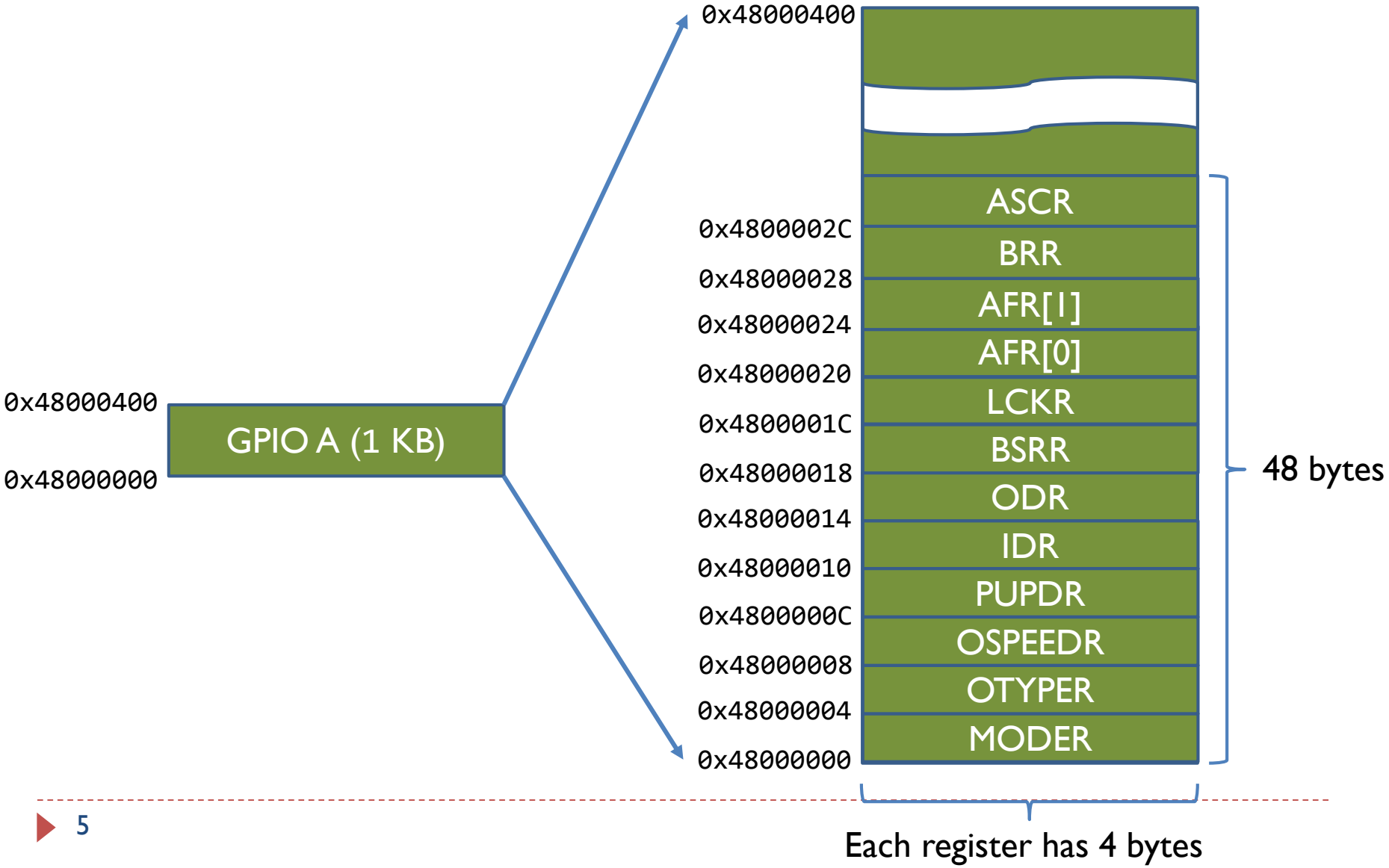
Memory Map of Cortex-M4



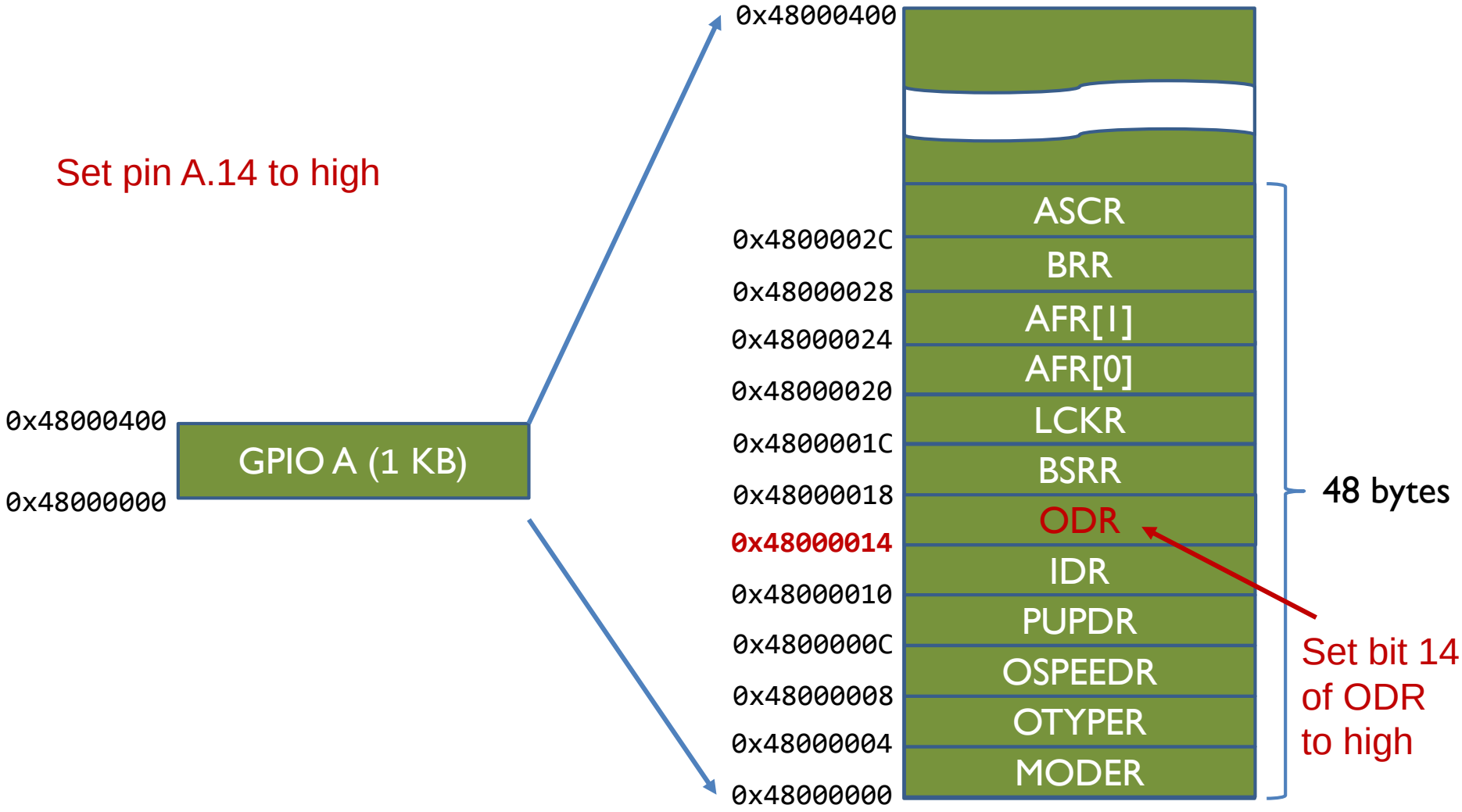
Memory Map of STM32L4



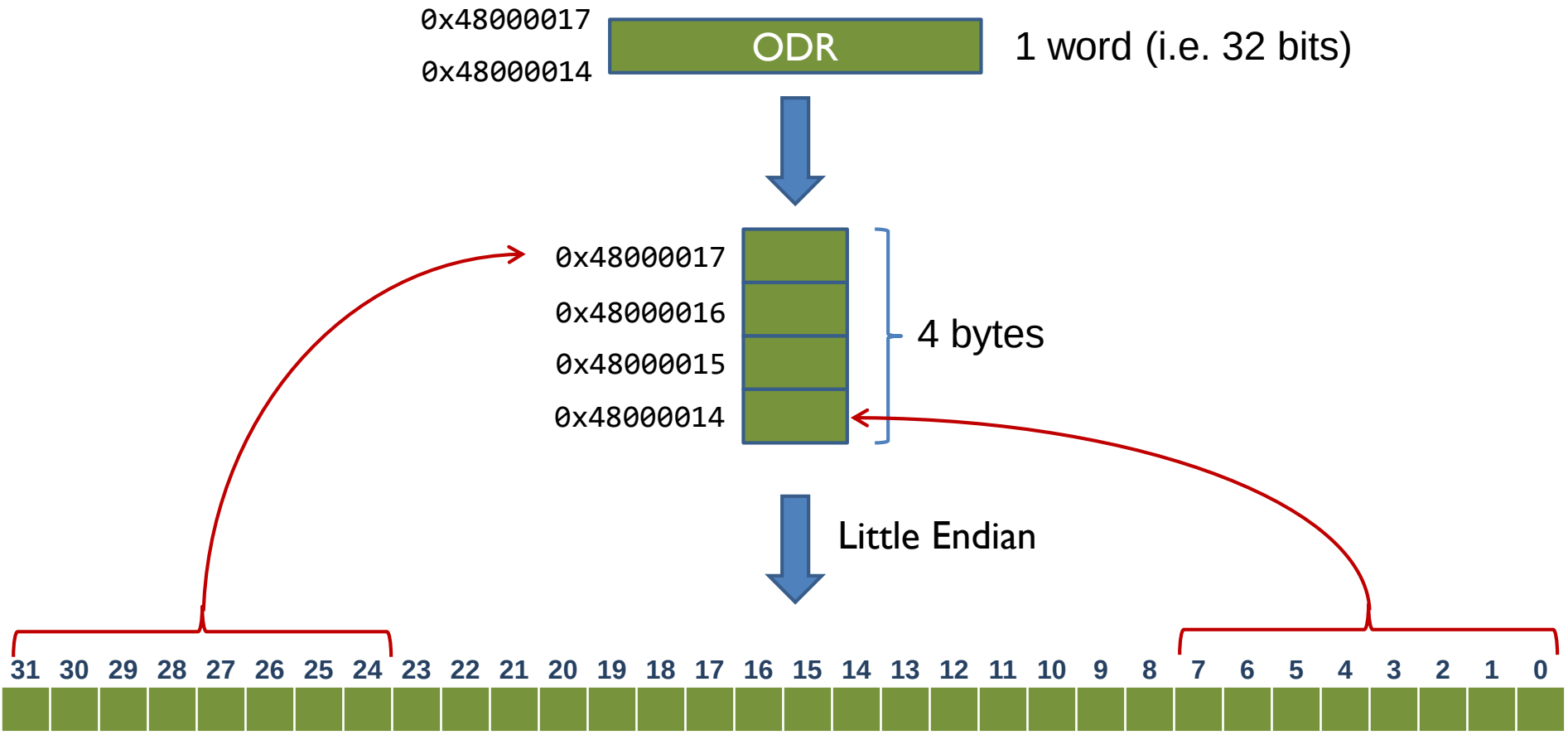
GPIO Memory Map



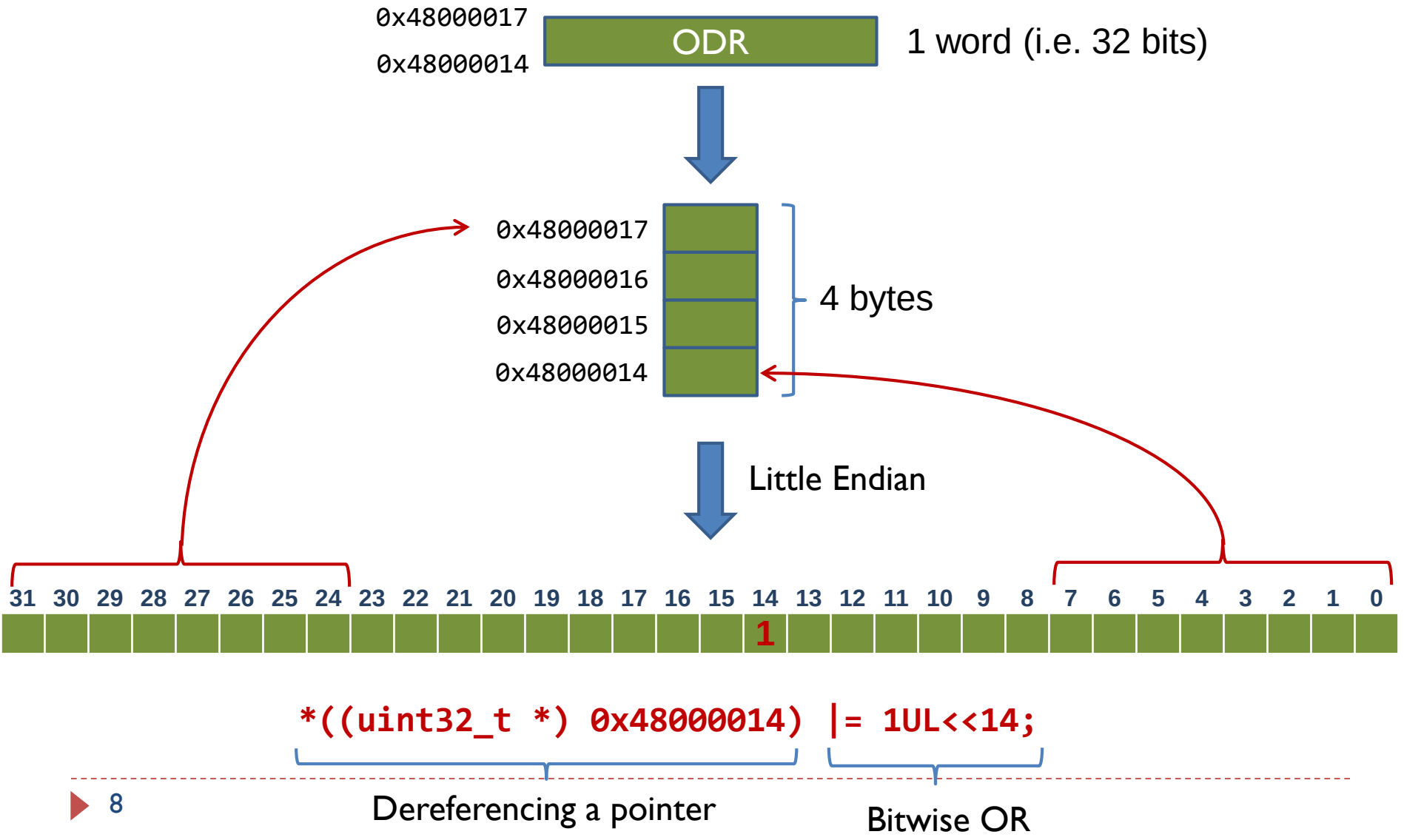
GPIO Memory Map



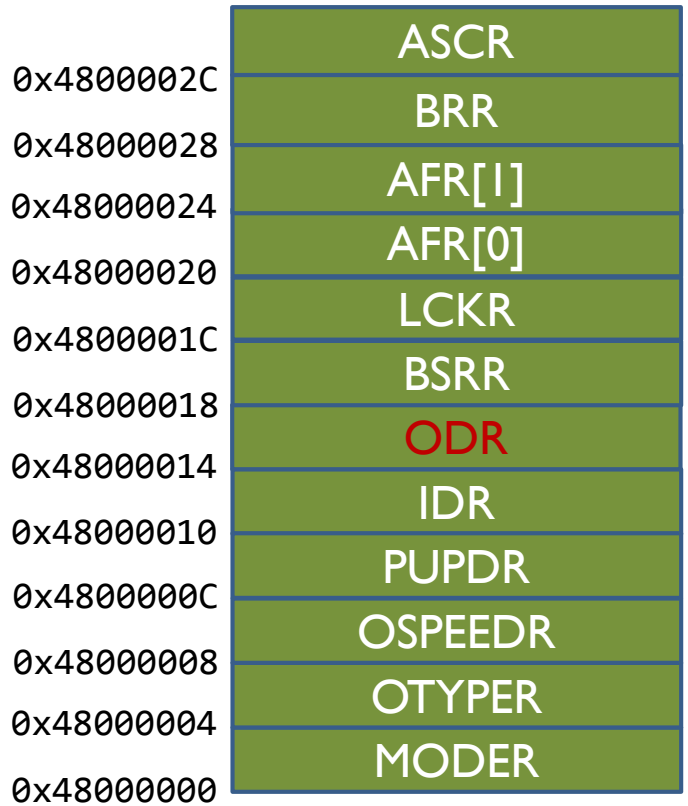
Output Data Register (ODR)



Output Data Register (ODR)



Dereferencing a Memory Address




```
typedef struct {
  volatile uint32_t MODER;    // Mode register
  volatile uint32_t OTYPER;   // Output type register
  volatile uint32_t OSPEEDR;  // Output speed register
  volatile uint32_t PUPDR;    // Pull-up/pull-down register
  volatile uint32_t IDR;      // Input data register
  volatile uint32_t ODR;      // Output data register
  volatile uint32_t BSRR;     // Bit set/reset register
  volatile uint32_t LCKR;     // Configuration lock register
  volatile uint32_t AFR[2];   // Alternate function registers
  volatile uint32_t BRR;      // Bit Reset register
  volatile uint32_t ASCR;     // Analog switch control register
} GPIO_TypeDef;

// Casting memory address to a pointer
#define GPIOA ((GPIO_TypeDef *) 0x48000000)
```

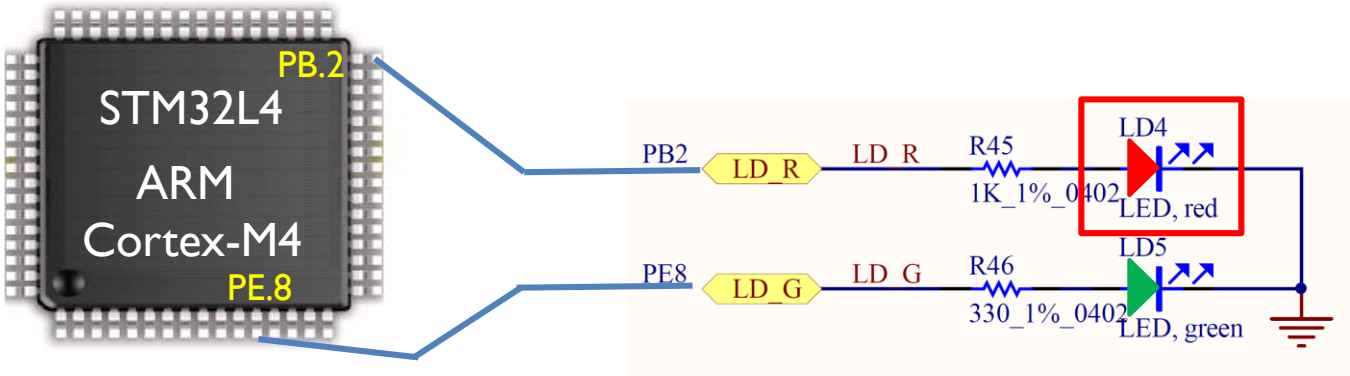
```
GPIOA->ODR |= 1UL<<14;
```

or

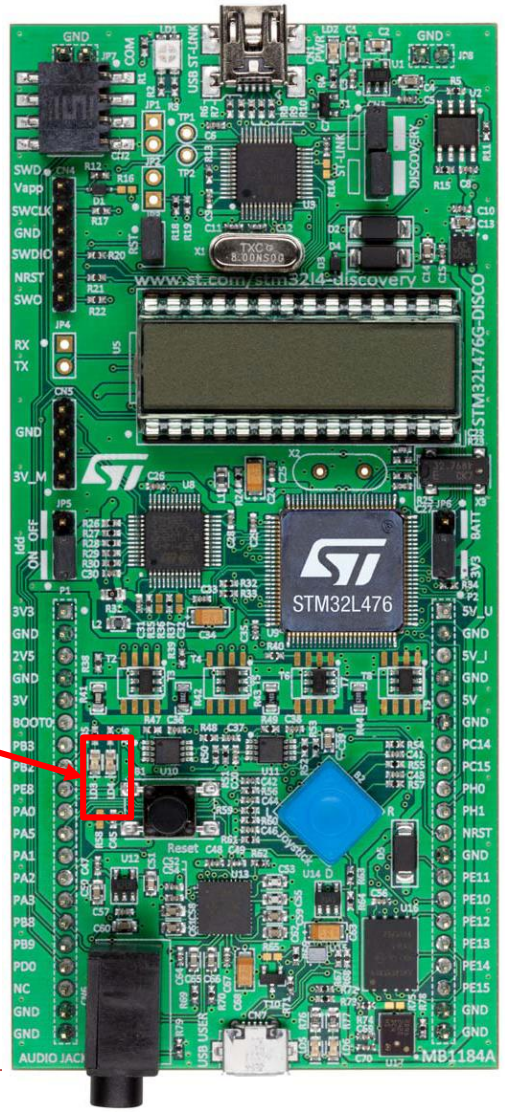
```
(*GPIOA).ODR |= 1UL<<14;
```

Red LED (PB.2)

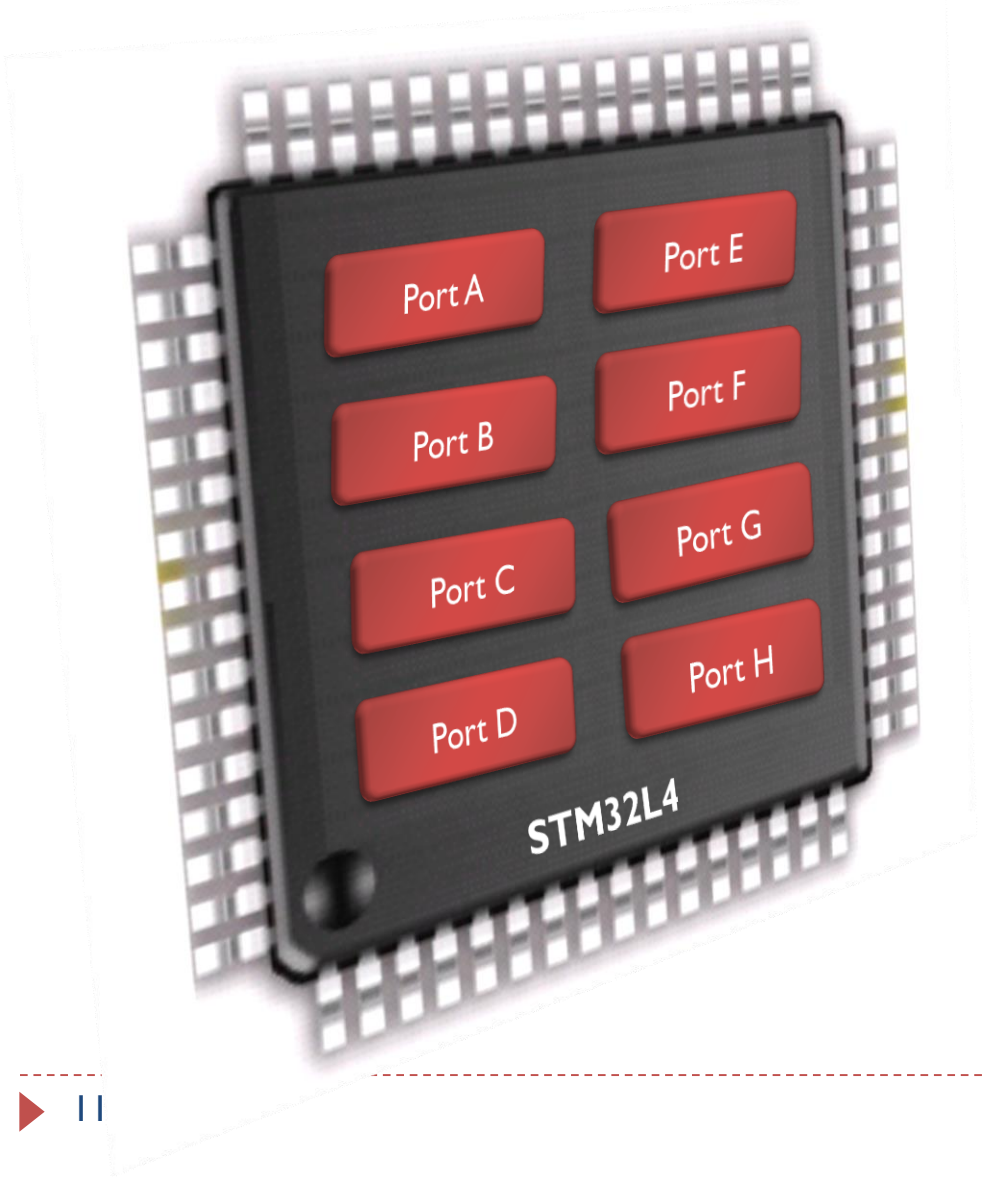
STM32L4 Discovery Kit



PB.2	Red LED
High	On
Low	Off

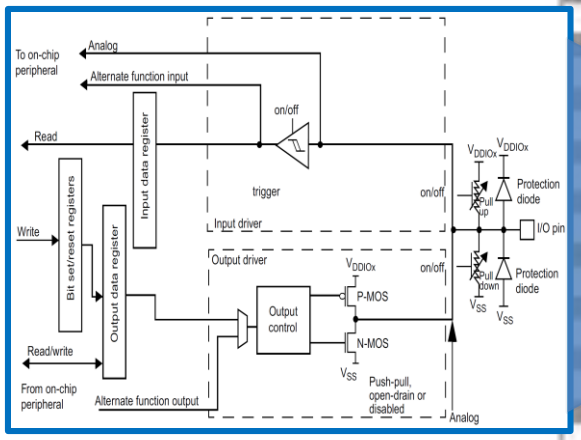


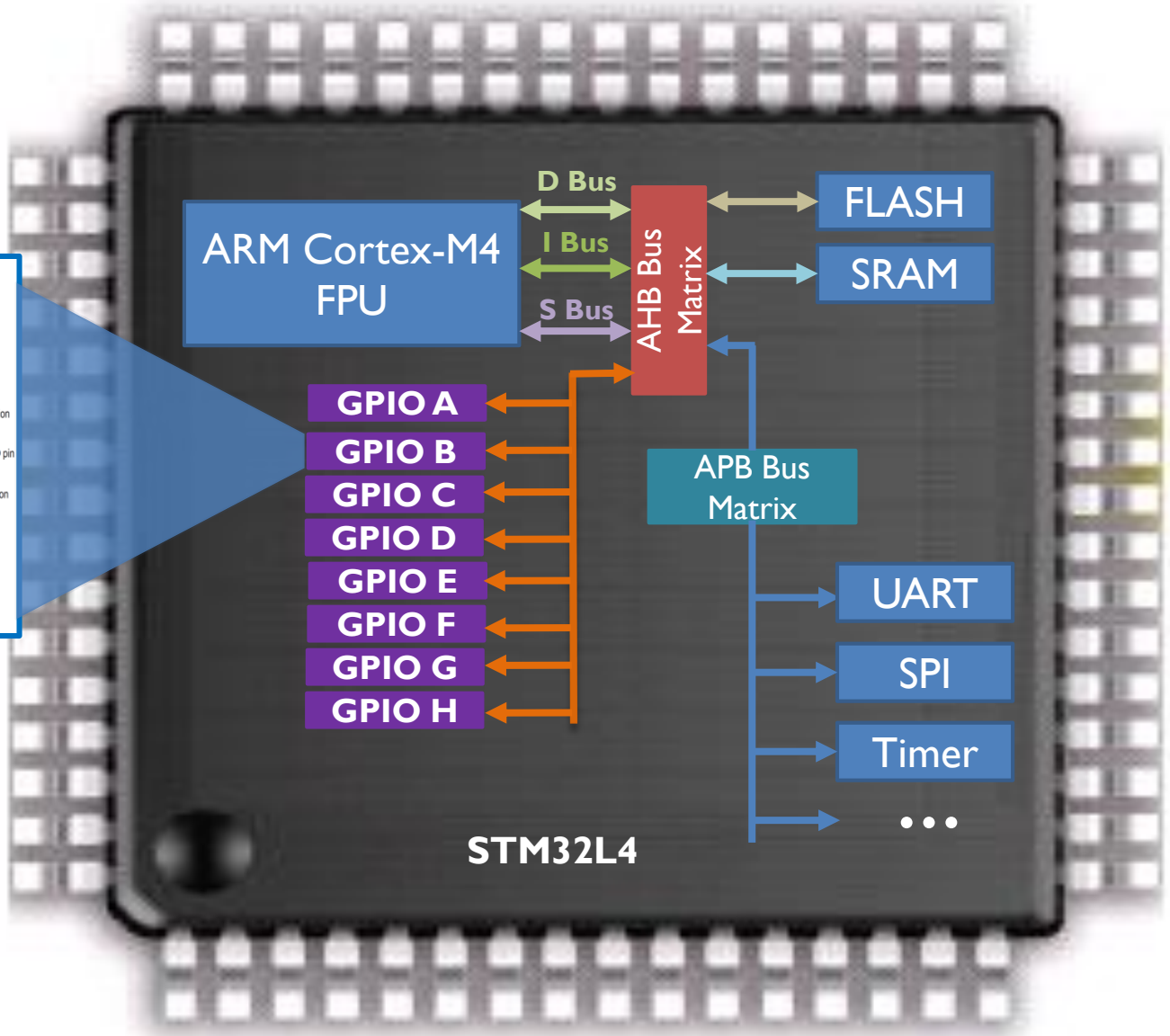
General Purpose Input/Output (GPIO)



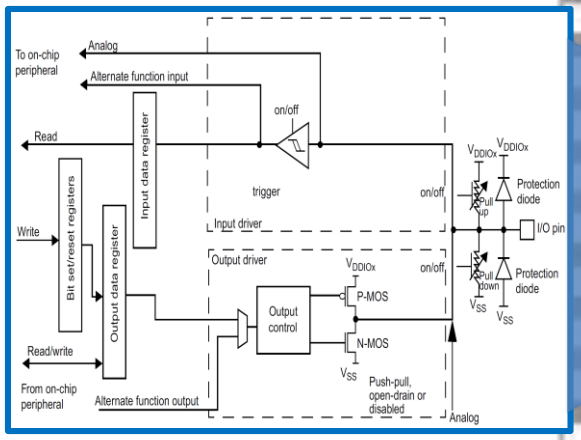
- 8 GPIO Ports:
  A, B, C, D, E, F, G, H
- Up to 16 pins in each port

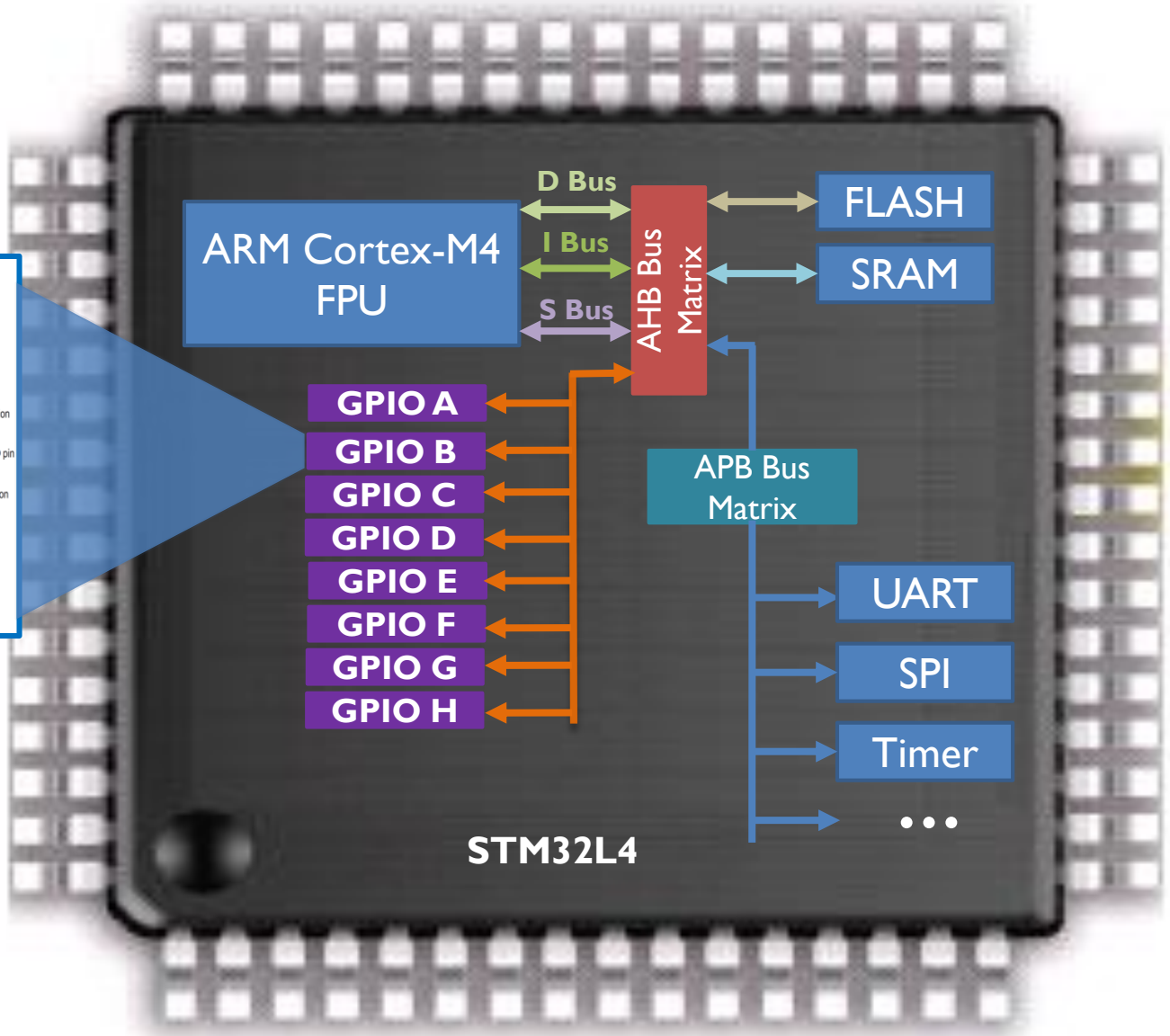
General Purpose Input/Output (GPIO)



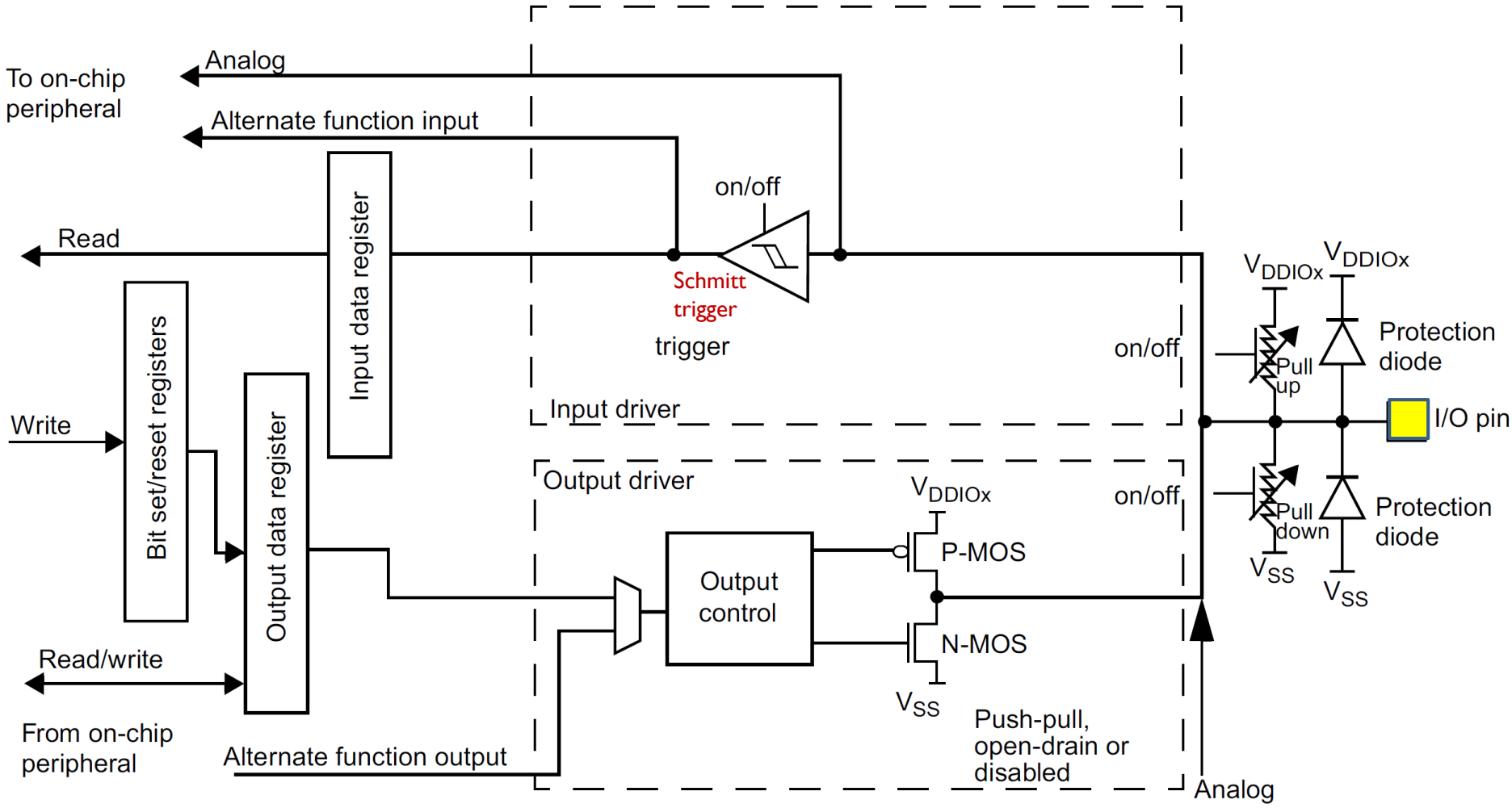


General Purpose Input/Output (GPIO)



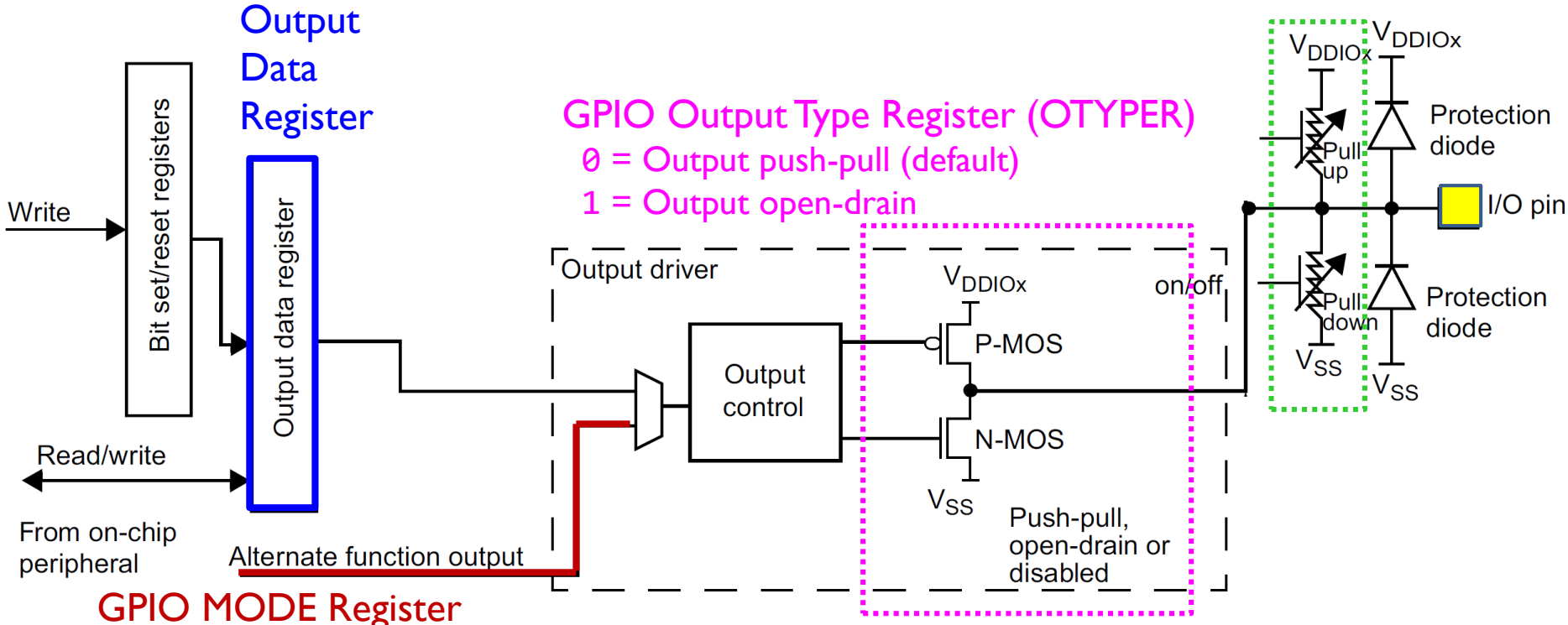


Basic Structure of an I/O Port Bit
Input and Output



Basic Structure of an I/O Port Bit: Output

GPIO Pull-up/Pull-down Register (PUPDR)
00 = No pull-up, pull-down 01 = Pull-up
10 = Pull-down 11 = Reserved



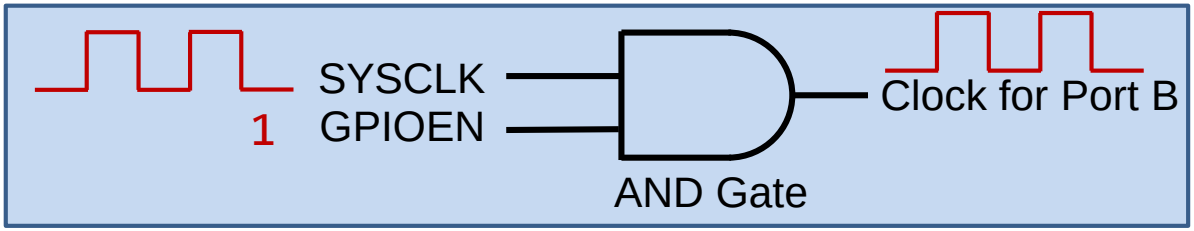
GPIO MODE Register
00 = Input, 01 = Output,
10 = AF, 11 = Analog (default)

Enable Clock

- AHB2 peripheral clock enable register (RCC_AHB2ENR)

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.	RNG EN	Res.	AESEN
													rw		rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	ADCEN	OTGFS EN	Res.	Res.	Res.	Res.	GPIOH EN	GPIOG EN	GPIOF EN	GPIOE EN	GPIOD EN	GPIOC EN	GPIOB EN	GPIOA EN
		rw	rw					rw	rw	rw	rw	rw	rw	rw	rw

Bit 1 **GPIOBEN:** IO port B clock enable
Set and cleared by software.
0: IO port B clock disabled
1: IO port B clock enabled




```
#define  RCC_AHB2ENR_GPIOBEN  ((uint32_t)0x00000002U)

RCC->AHB2ENR |= RCC_AHB2ENR_GPIOBEN;
```

GPIO Mode Register (MODER)

- 32 bits (16 pins, 2 bits per pin)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODE15[1:0]		MODE14[1:0]		MODE13[1:0]		MODE12[1:0]		MODE11[1:0]		MODE10[1:0]		MODE9[1:0]		MODE8[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
MODE7[1:0]		MODE6[1:0]		MODE5[1:0]		MODE4[1:0]		MODE3[1:0]		MODE2[1:0]		MODE1[1:0]		MODE0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Pin 2 Pin 1 Pin 0

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

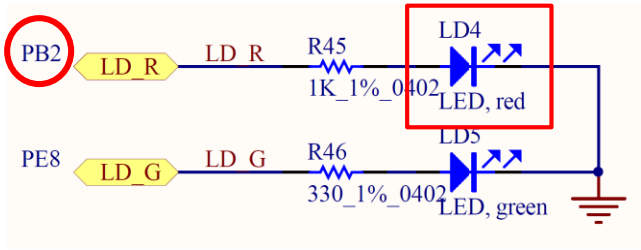
These bits are written by software to configure the I/O mode.

00: Input mode

01: General purpose output mode

10: Alternate function mode

11: Analog mode (reset state)




```
GPIOB->MODER &= ~(3UL<<4);  // Clear bits 4 and 5 for Pin 2
GPIOB->MODER |=   1UL<<4;   // Set bit 4, set Pin 2 as output
```

GPIO Output Type Register (OTYPE)

- 16 bits reserved, 16 data bits, 1 bit for each pin

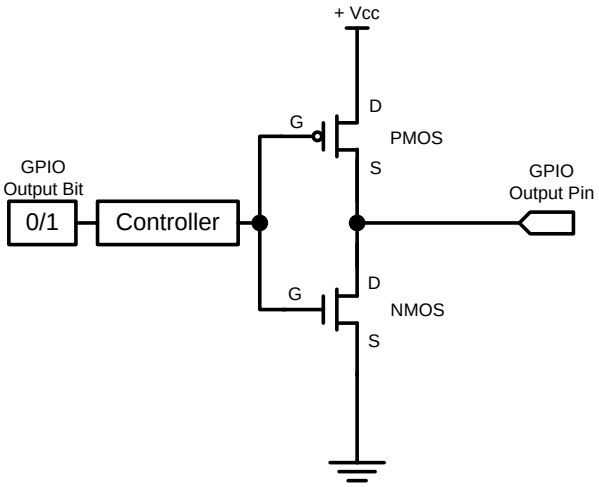
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

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

0: Output push-pull (reset state)

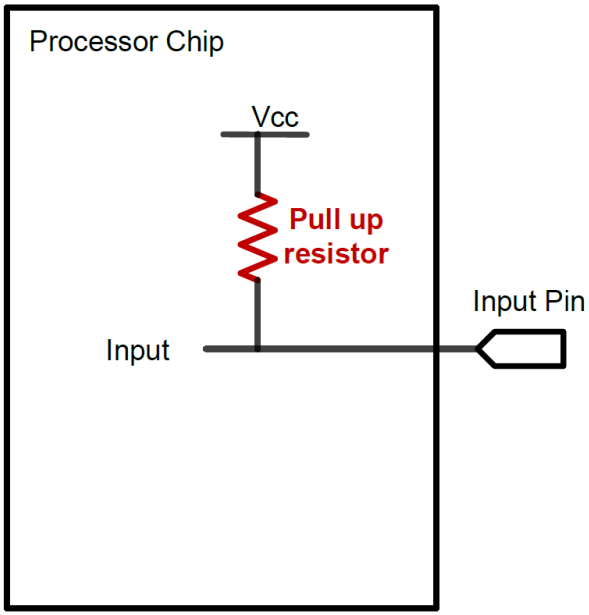
1: Output open-drain




```
GPIOB->OTYPE &= ~(1UL<<2);  // Clear bit 2
```

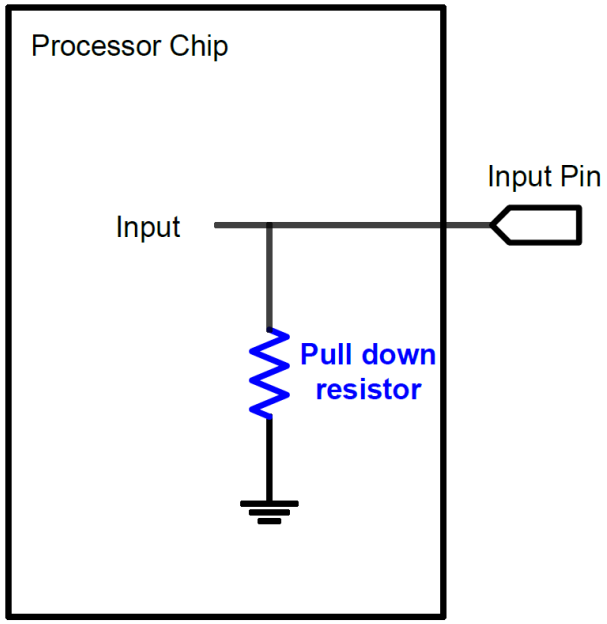
GPIO Input: Pull Up and Pull Down

- A digital input can have three states: High, Low, and High-Impedance (also called floating, tri-stated, HiZ)



Pull-Up

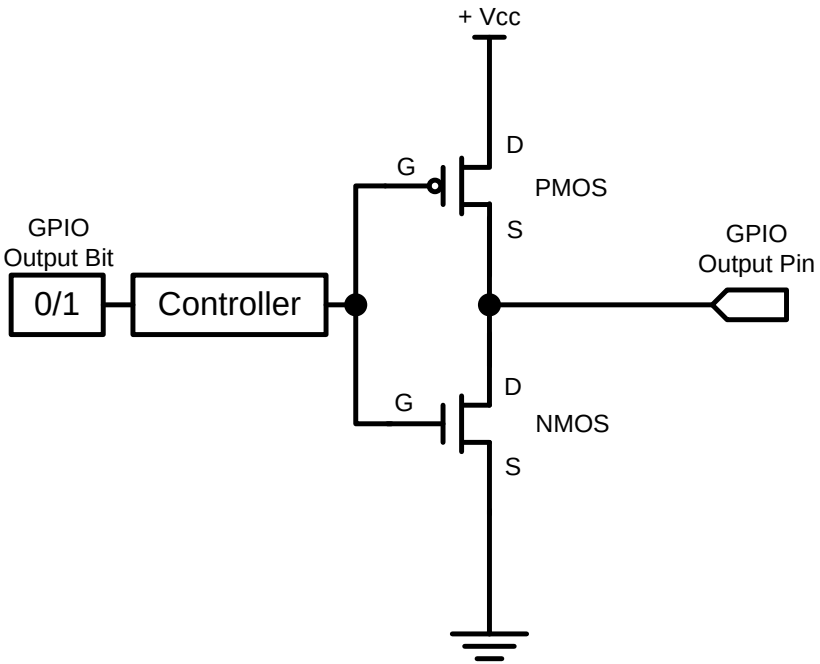
If external input is HiZ, the input is read as a valid HIGH.

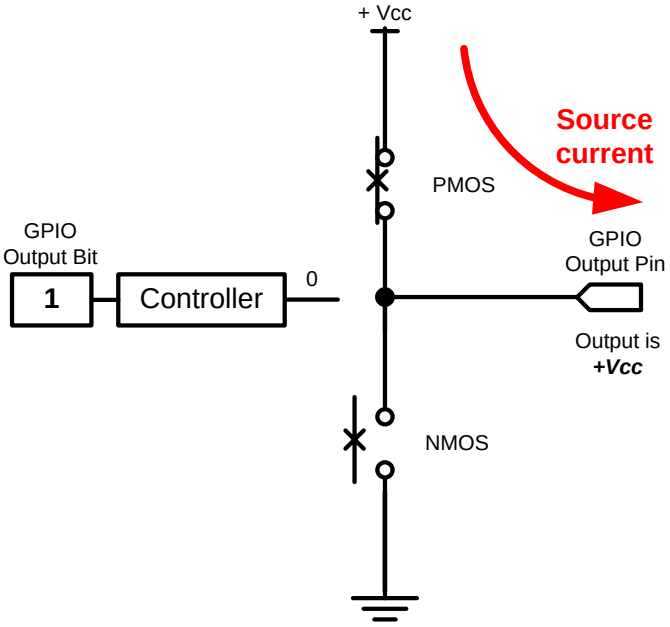


Pull-Down

If external input is HiZ, the input is read as a valid LOW.

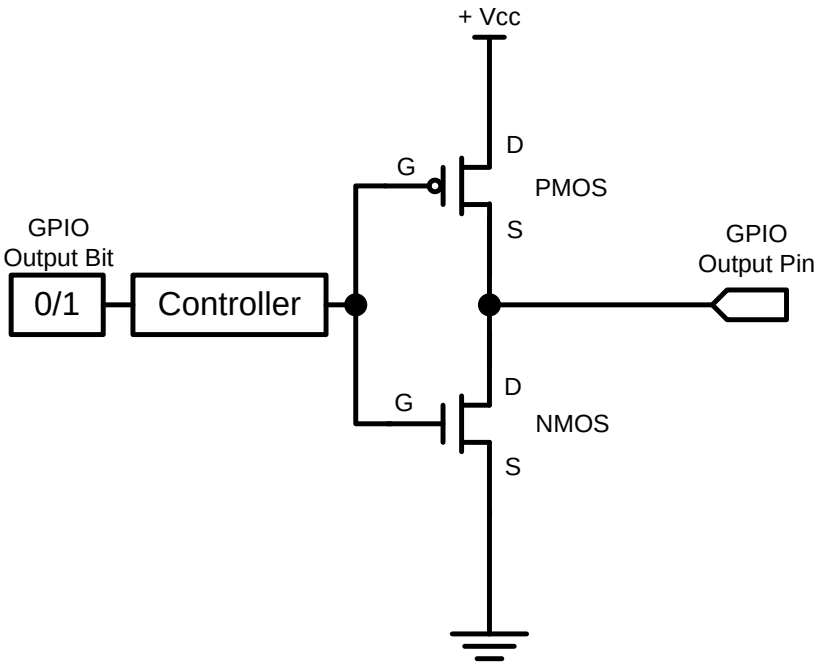
GPIO Output: Push-Pull

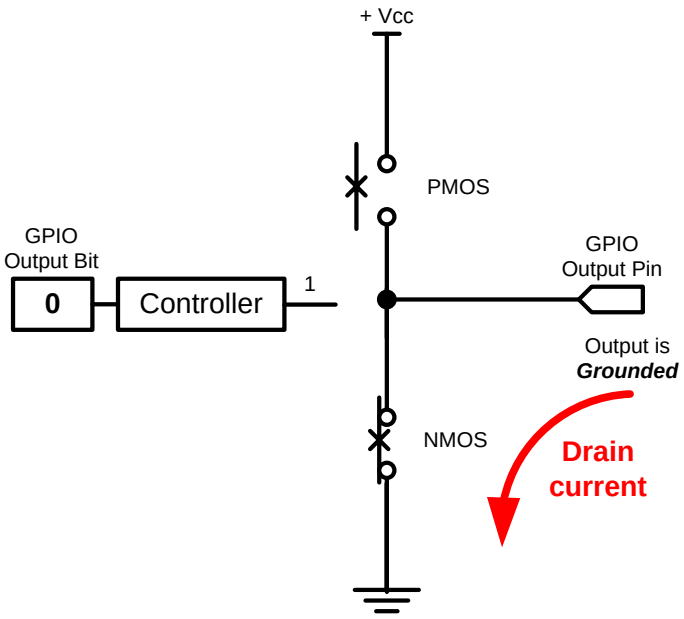




GPIO Output = 1
Source current to external circuit

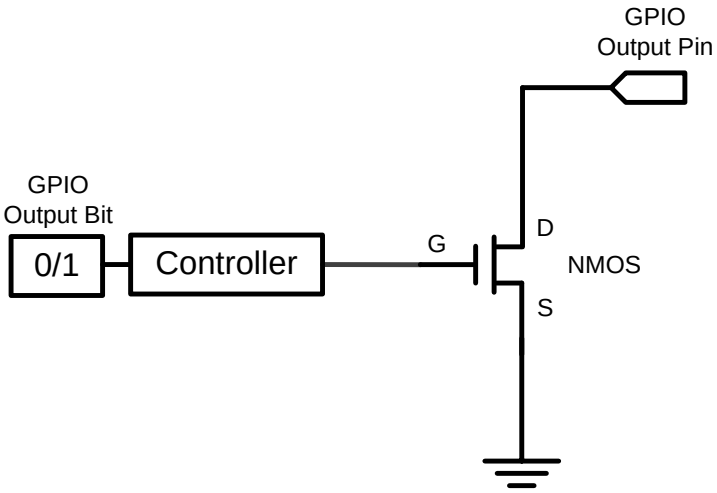
GPIO Output: Push-Pull

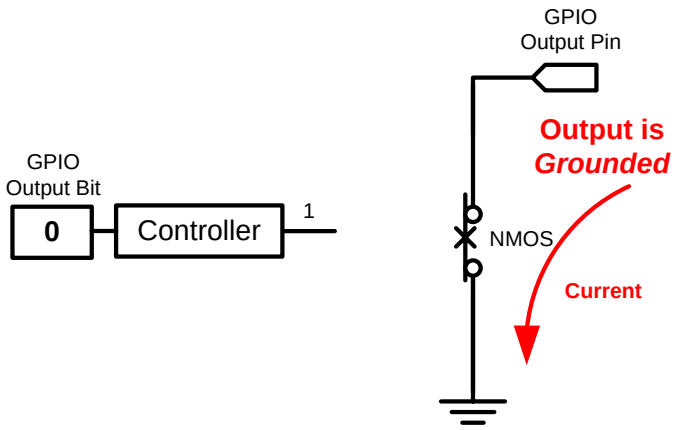




GPIO Output = 0
Drain current from external circuit

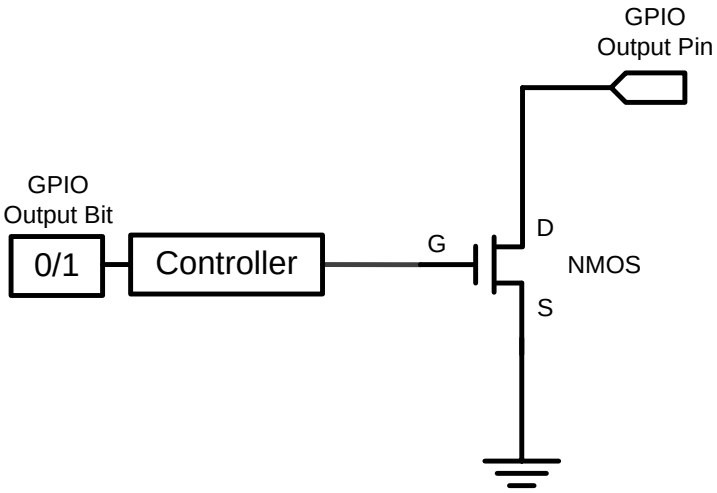
GPIO Output: Open-Drain

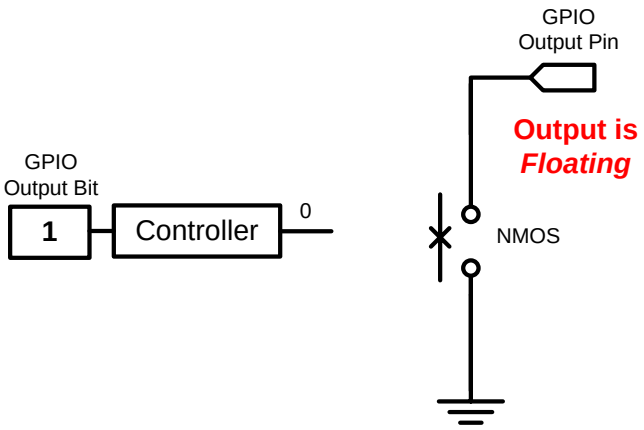




GPIO Output = 0
Drain current from external circuit

GPIO Output: Open-Drain

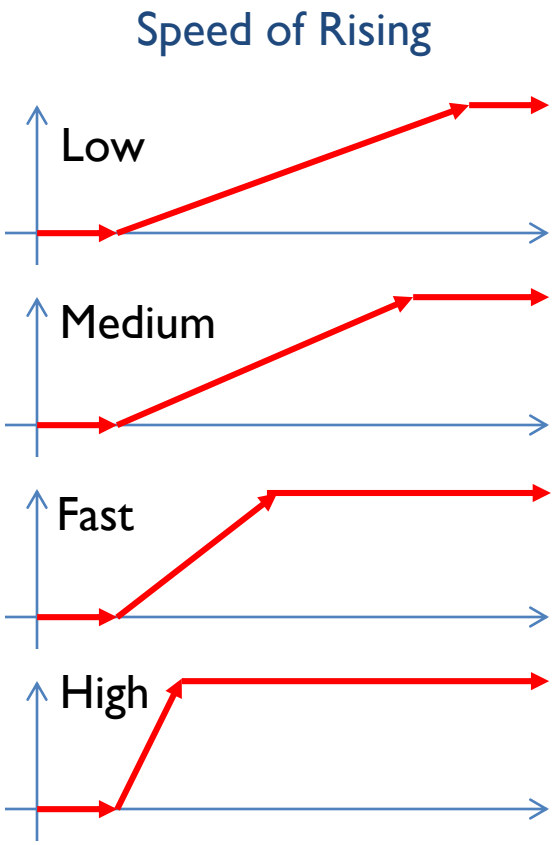




Output = 1
GPIO Pin has high-impedance to external circuit

GPIO Output Speed

- Output Speed:
  - Speed of rising and falling
  - Four speeds: Low, Medium, Fast, High
- Tradeoff
  - Higher GPIO speed increases EMI noise and power consumption
  - Configure based on peripheral speed
    - Low speed for toggling LEDs
    - High speed for SPI

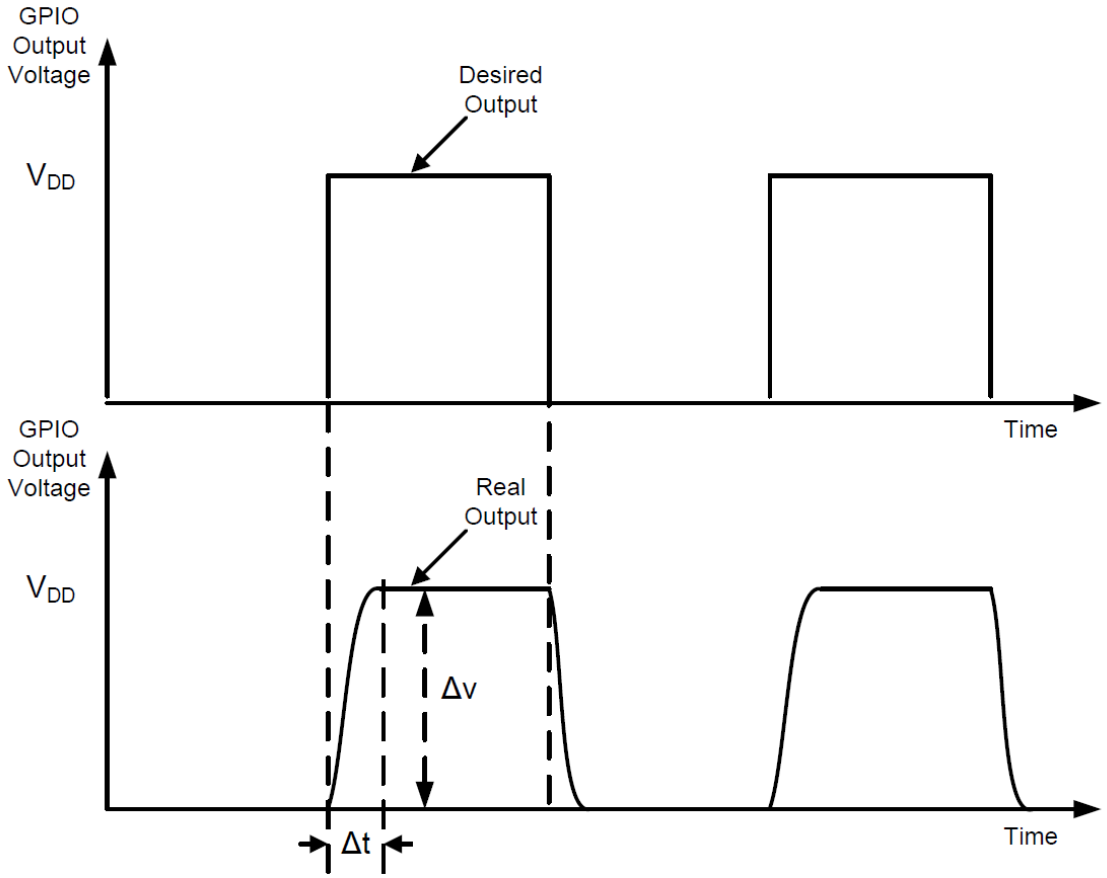


Slew Rate

Slew Rate:
Maximum rate of change of the output voltage

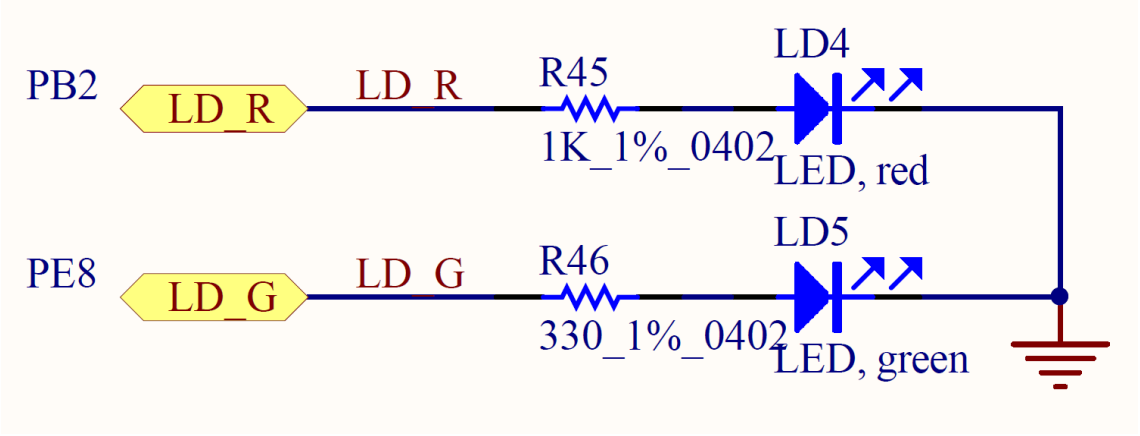
$$Slew\ Rate = max\left(\frac{\Delta V}{\Delta t}\right)$$

A high slew rate allows the output to be toggled at a fast speed.



GPIO Output: Push-Pull *vs* Open-Drain

Output Bit	Push-Pull	Open-Drain
1	High	HiZ
0	Low	Low



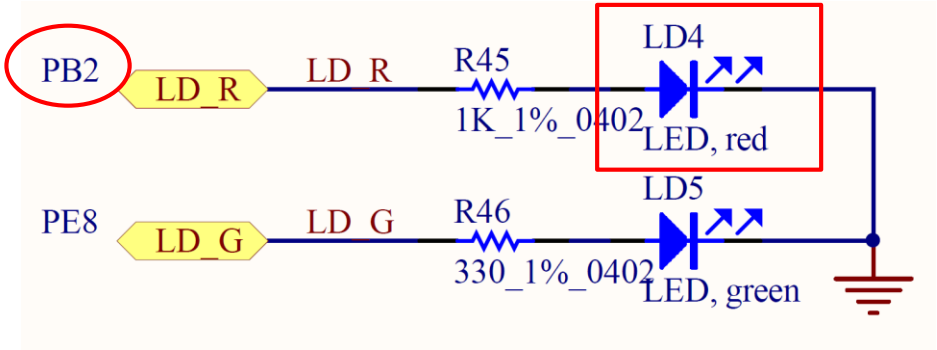
Use push-pull output, instead of open-drain output!

GPIO Output Data Register (ODR)

- 16 bits reserved, 16 data bits, 1 bit for each pin

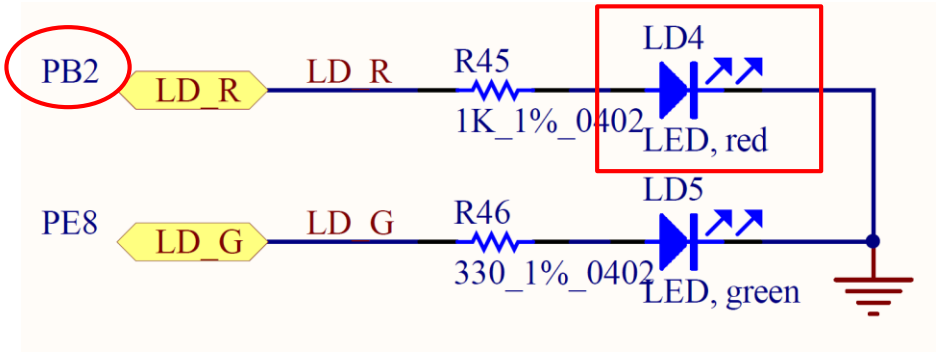
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OD15	OD14	OD13	OD12	OD11	OD10	OD9	OD8	OD7	OD6	OD5	OD4	OD3	OD2	OD1	OD0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Pin 2




```
GPIOB->ODR |= 1UL << 2;     // Set bit 2
```

Light up the Red LED (PB.2)




```
RCC->AHB2ENR |= RCC_AHB2ENR_GPIOBEN;  // Enable clock of Port B

GPIOB->MODER &= ~(3UL<<4); // Clear mode bits
GPIOB->MODER |=   1UL<<4;  // Set mode to output

GPIOB->OTYPE &= ~(1UL<<2); // Select push-pull output

GPIOB->ODR   |= 1UL << 2;  // Output 1 to turn on red LED
```

GPIO Initialization

- Turn on the clock to the GPIO Port (e.g. Port B)

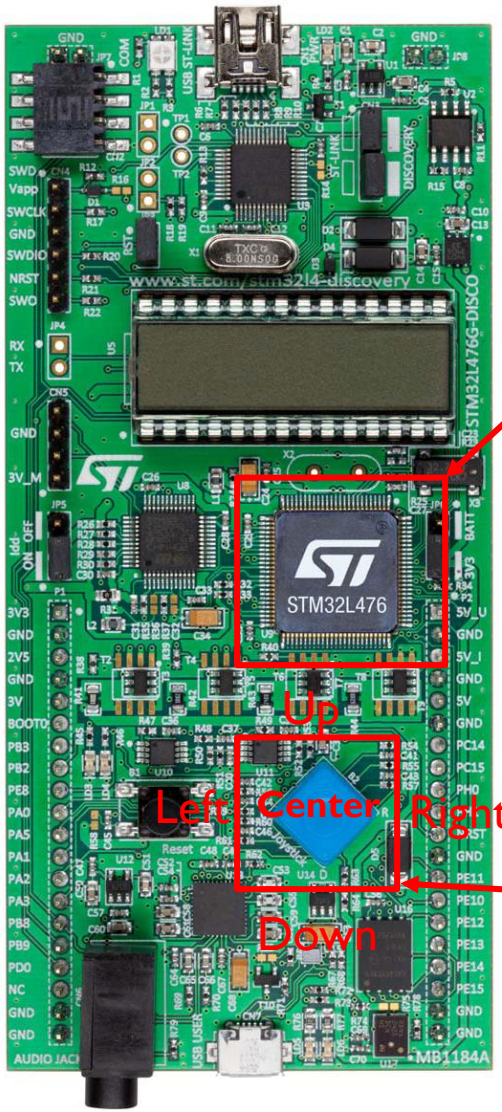
```
RCC->AHBENR |= RCC_AHBENR_GPIOBEN; Reset and Clock Control (RCC)
```

- Configure GPIO mode, output type, speed, pull-up/pull-down

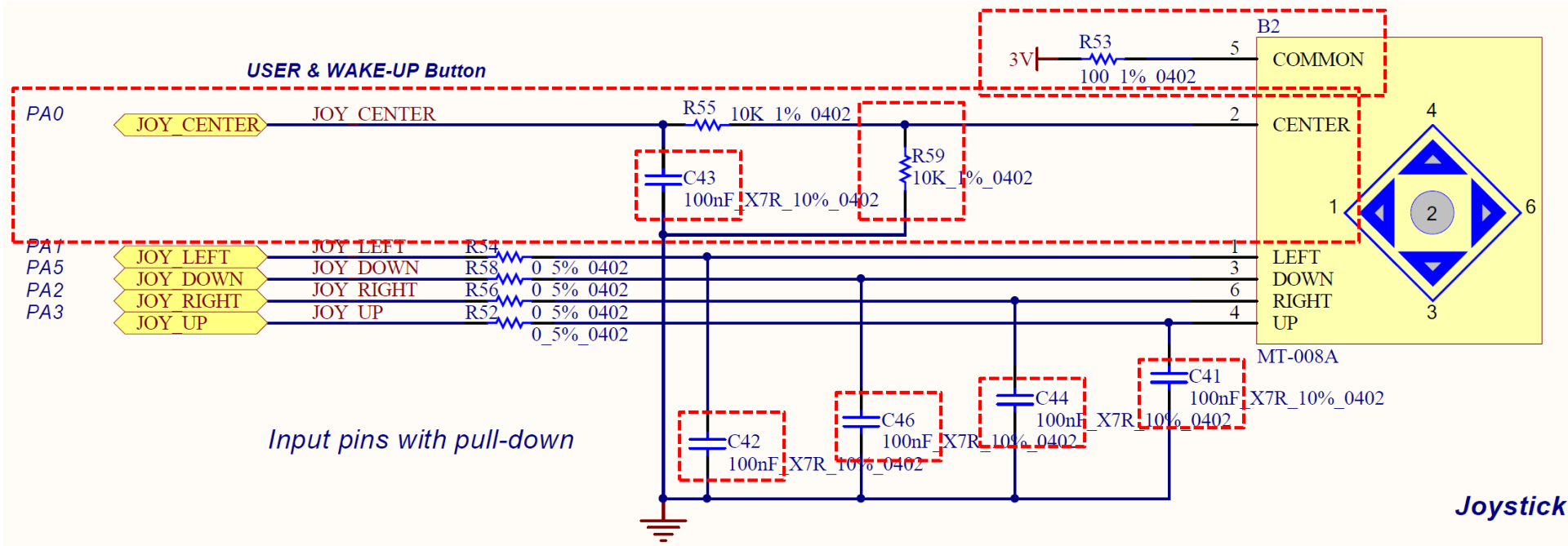
```
typedef struct
{
  __IO uint32_t MODER;
  __IO uint16_t OTYPER;
  uint16_t RESERVED0;
  __IO uint32_t OSPEEDR;
  __IO uint32_t PUPDR;
  __IO uint16_t IDR;
  uint16_t RESERVED1;
  __IO uint16_t ODR;
  uint16_t RESERVED2;
  __IO uint16_t BSRRL; /* BSRR register is split to 2 * 16-bit fields BSRRL */
  __IO uint16_t BSRRH; /* BSRR register is split to 2 * 16-bit fields BSRRH */
  __IO uint32_t LCKR;
  __IO uint32_t AFR[2];
} GPIO_TypeDef;
#define PERIPH_BASE           ((uint32_t)0x40000000)
#define AHBPERIPH_BASE        (PERIPH_BASE + 0x20000)
#define GPIOB_BASE            (AHBPERIPH_BASE + 0x0400)
#define GPIOB                 ((GPIO_TypeDef *) GPIOB_BASE)
```

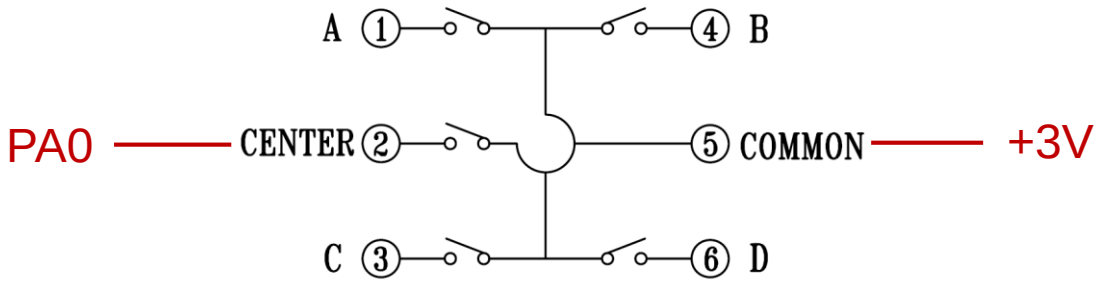
Joystick

STM32L4 Discovery Kit



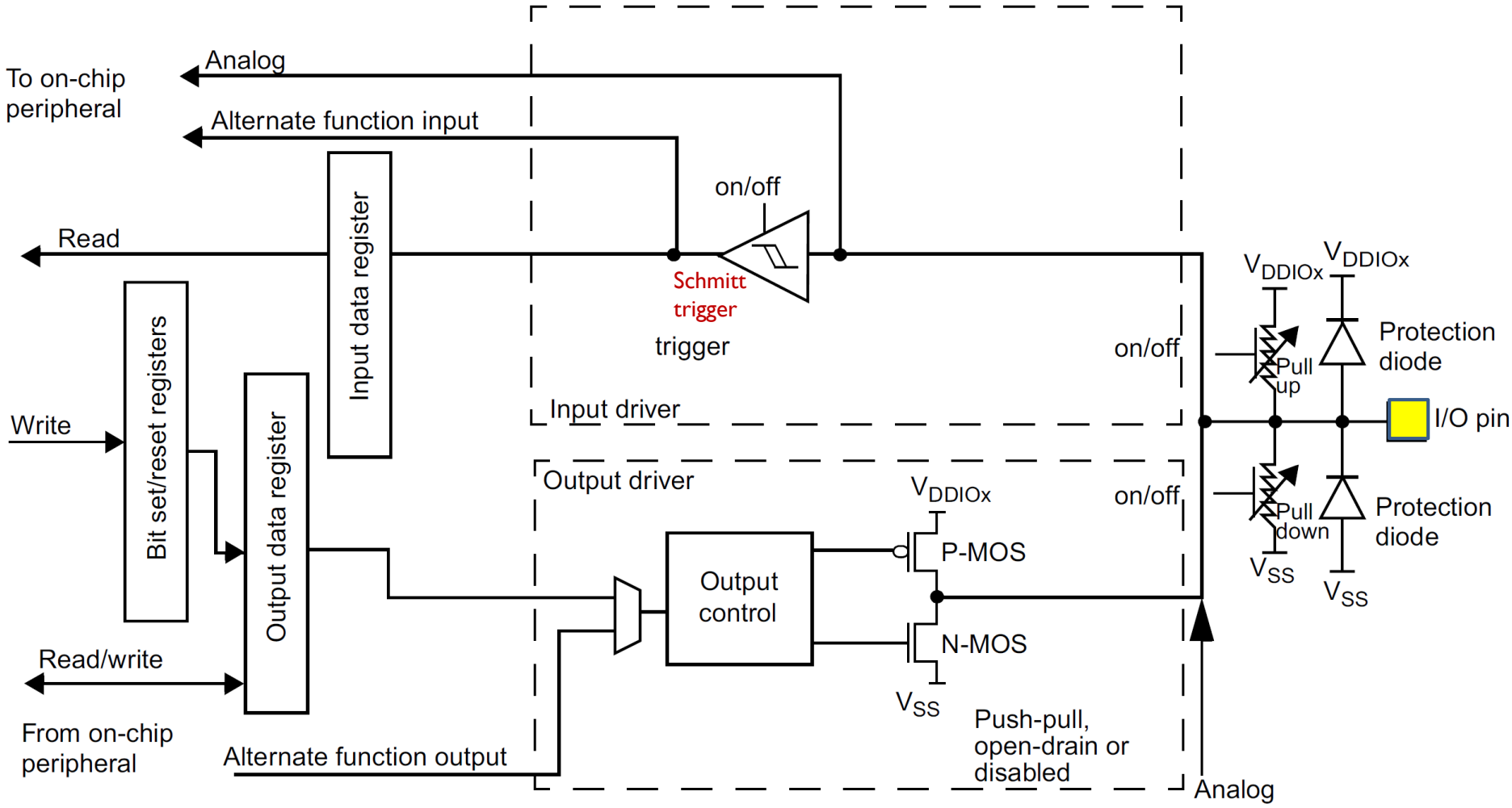
Joystick





Internal diagram of joystick

Basic Structure of an I/O Port Bit
Input and Output

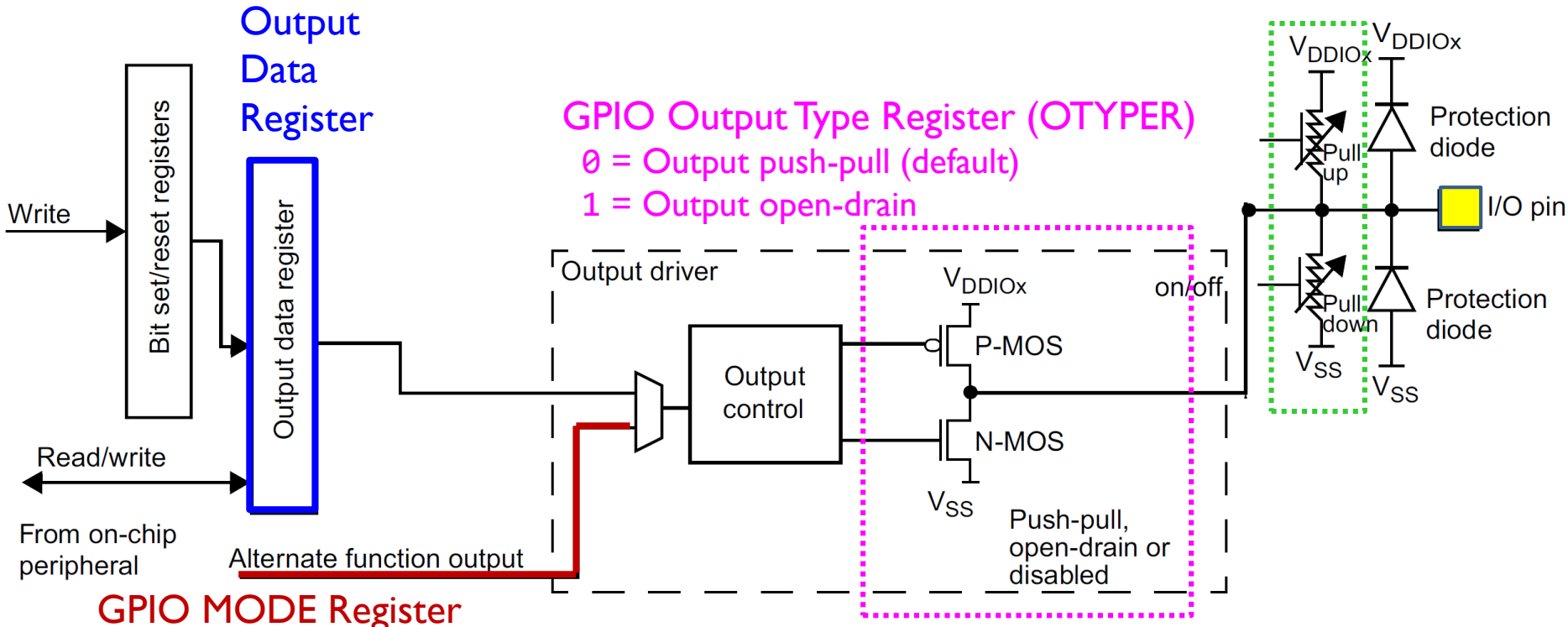


Basic Structure of an I/O Port Bit: Output

GPIO Pull-up/Pull-down Register (PUPDR)

00 = No pull-up, pull-down 01 = Pull-up

10 = Pull-down 11 = Reserved

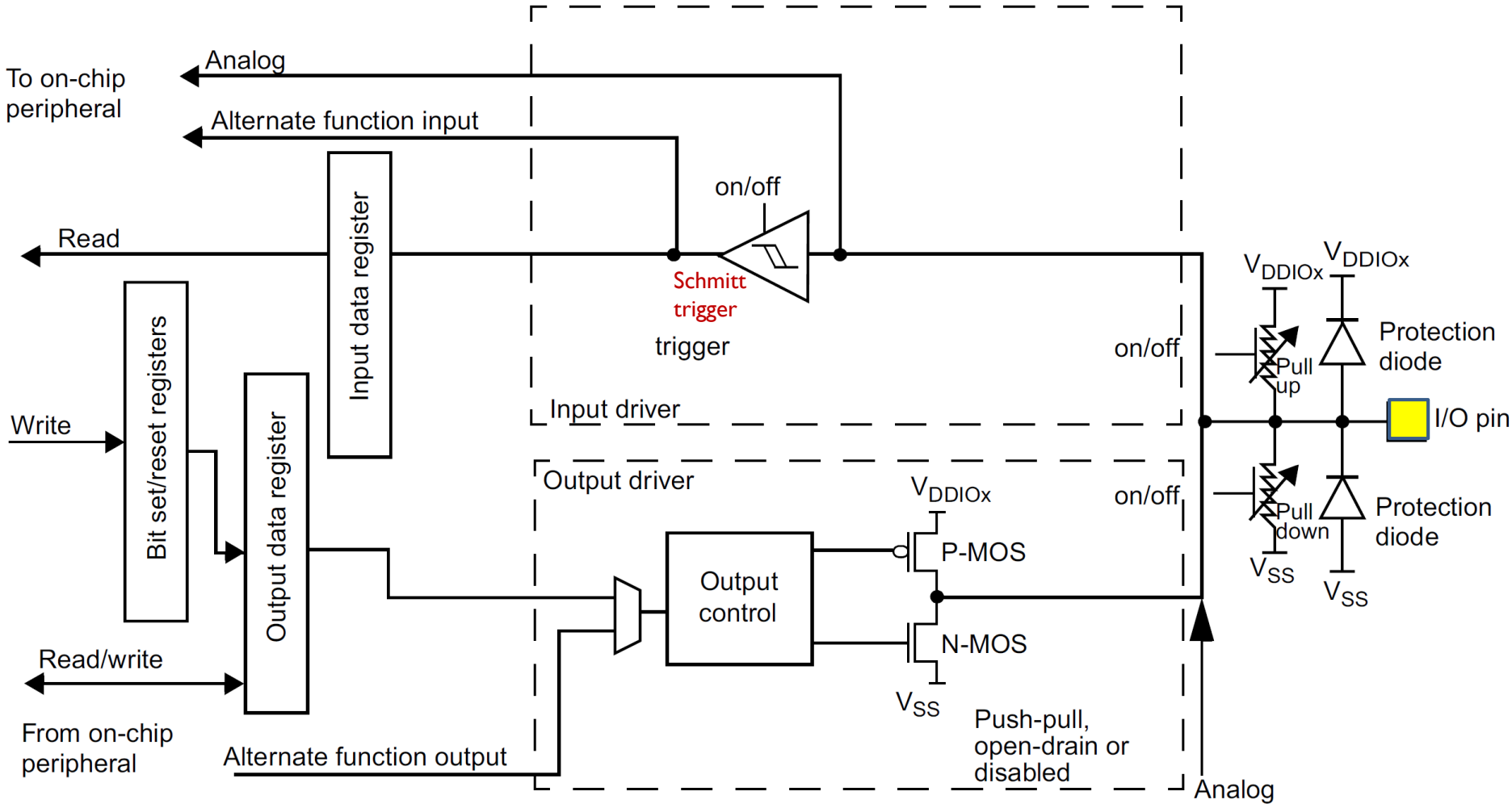


GPIO MODE Register

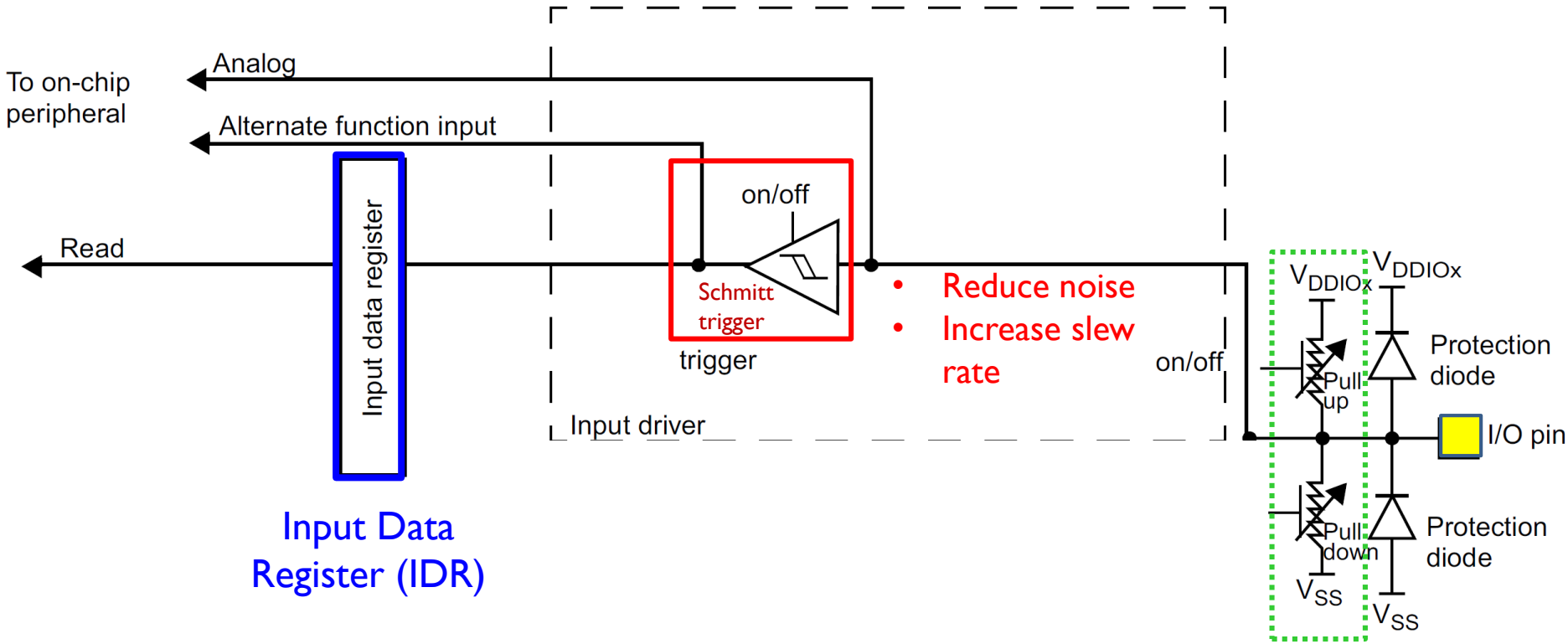
00 = Input, 01 = Output,

10 = AF, 11 = Analog (default)

Basic Structure of an I/O Port Bit
Input and Output



Basic Structure of an I/O Port Bit: Input



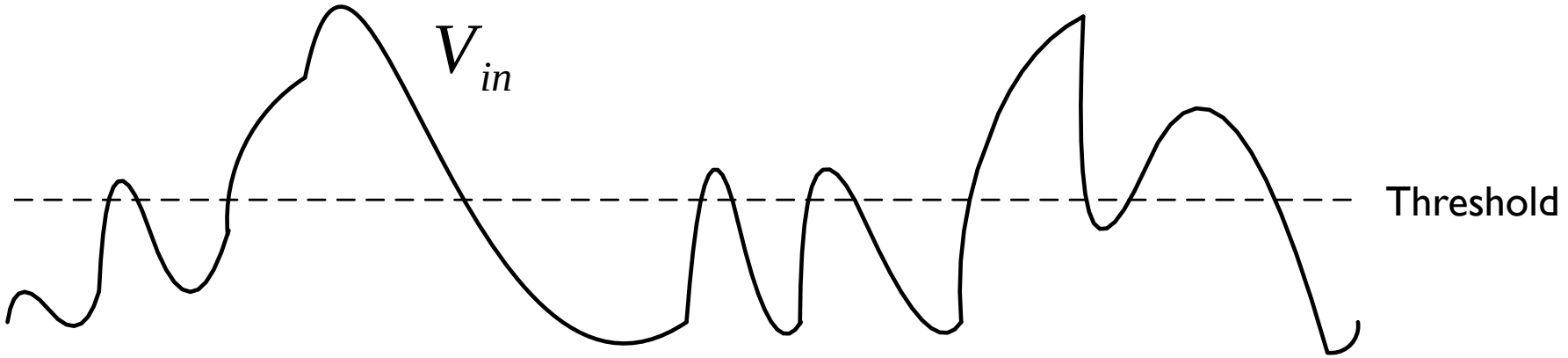
Input Data Register (IDR)

GPIO Pull-up/Pull-down Register (PUPDR)

00 = No pull-up, pull-down 01 = Pull-up

10 = Pull-down 11 = Reserved

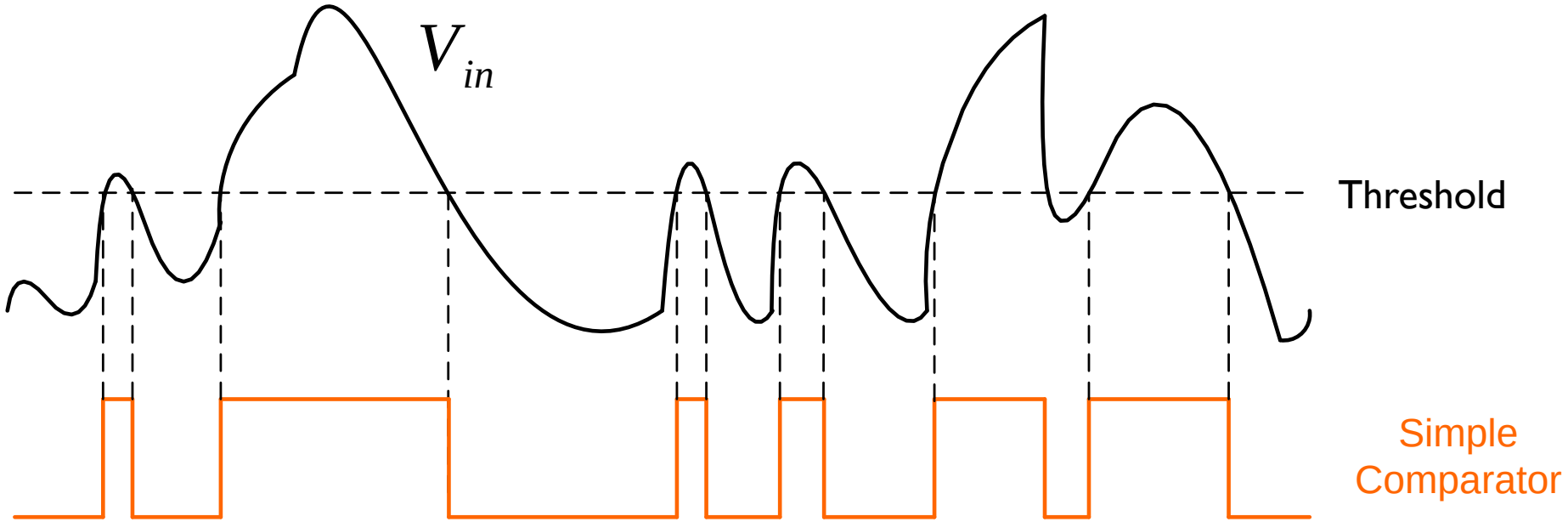
Schmitt Trigger



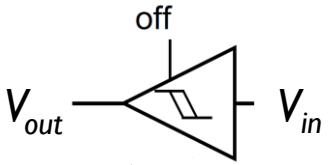
Analog signals

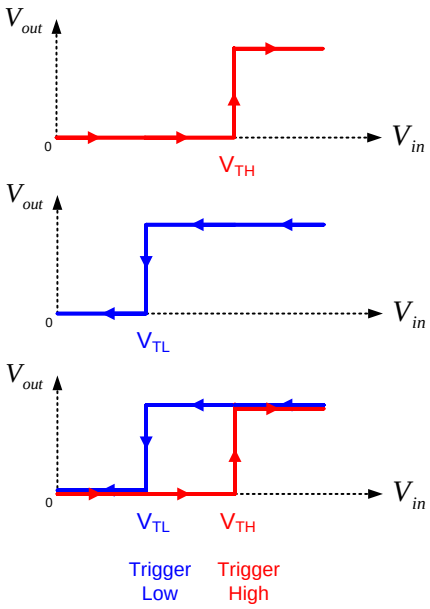
- Noisy
- Rise and fall slowly (small slew rate)

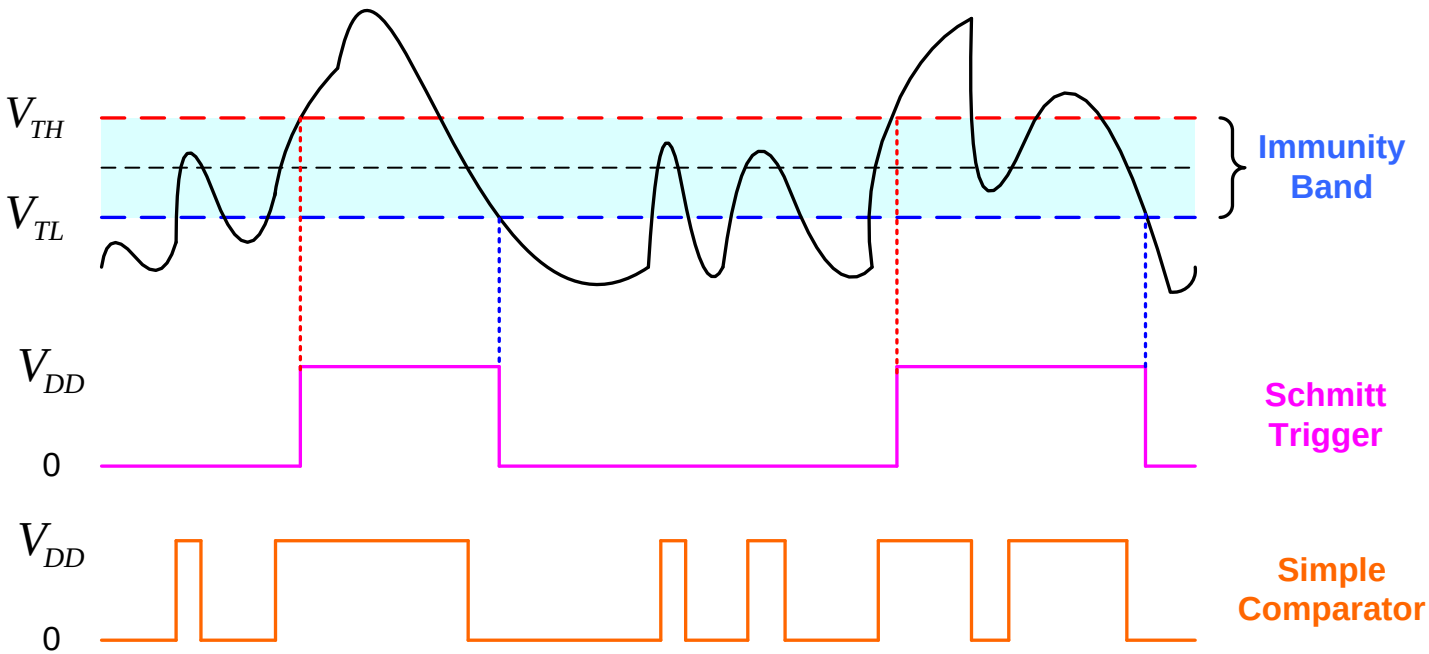
Schmitt Trigger



Schmitt Trigger





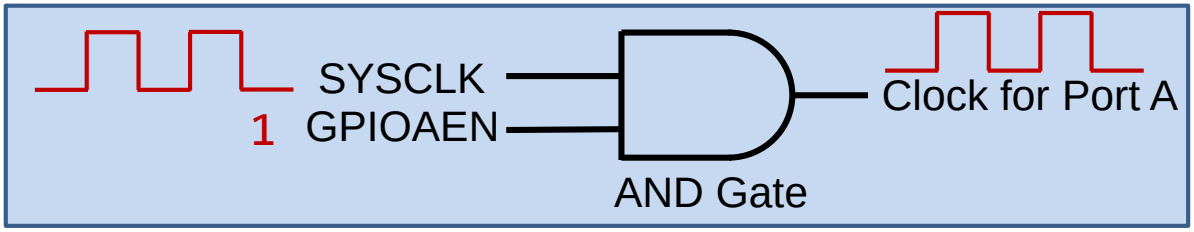


Enable Clock

- AHB2 peripheral clock enable register (RCC_AHB2ENR)

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.	RNG EN	Res.	AESEN
													rw		rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	ADCEN	OTGFS EN	Res.	Res.	Res.	Res.	GPIOH EN	GPIOG EN	GPIOF EN	GPIOE EN	GPIOD EN	GPIOC EN	GPIOB EN	GPIOA EN
		rw	rw					rw	rw	rw	rw	rw	rw	rw	rw

Bit 0 **GPIOAEN:** IO port A clock enable
Set and cleared by software.
0: IO port A clock disabled
1: IO port A clock enabled




```
#define  RCC_AHB2ENR_GPIOAEN  ((uint32_t)0x00000001U)

RCC->AHB2ENR |= RCC_AHB2ENR_GPIOAEN; // Enable clock of Port A
```

GPIO Mode Register (MODER)

- 32 bits (16 pins, 2 bits per pin)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODE15[1:0]		MODE14[1:0]		MODE13[1:0]		MODE12[1:0]		MODE11[1:0]		MODE10[1:0]		MODE9[1:0]		MODE8[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
MODE7[1:0]		MODE6[1:0]		MODE5[1:0]		MODE4[1:0]		MODE3[1:0]		MODE2[1:0]		MODE1[1:0]		MODE0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Pin 2 Pin 1 Pin 0

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

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

00: Input mode

01: General purpose output mode

10: Alternate function mode

11: Analog mode (reset state)

```
// Set Pin 0 as input
GPIOA->MODER &= ~3UL;  // Clear bits 1 and 2 for Pin 0
```

GPIO Pull-up/Pull-down Register (PUPDR)

- 16 pins per port, 2 bits per pin

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PUPD15[1:0]		PUPD14[1:0]		PUPD13[1:0]		PUPD12[1:0]		PUPD11[1:0]		PUPD10[1:0]		PUPD9[1:0]		PUPD8[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
PUPD7[1:0]		PUPD6[1:0]		PUPD5[1:0]		PUPD4[1:0]		PUPD3[1:0]		PUPD2[1:0]		PUPD1[1:0]		PUPD0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

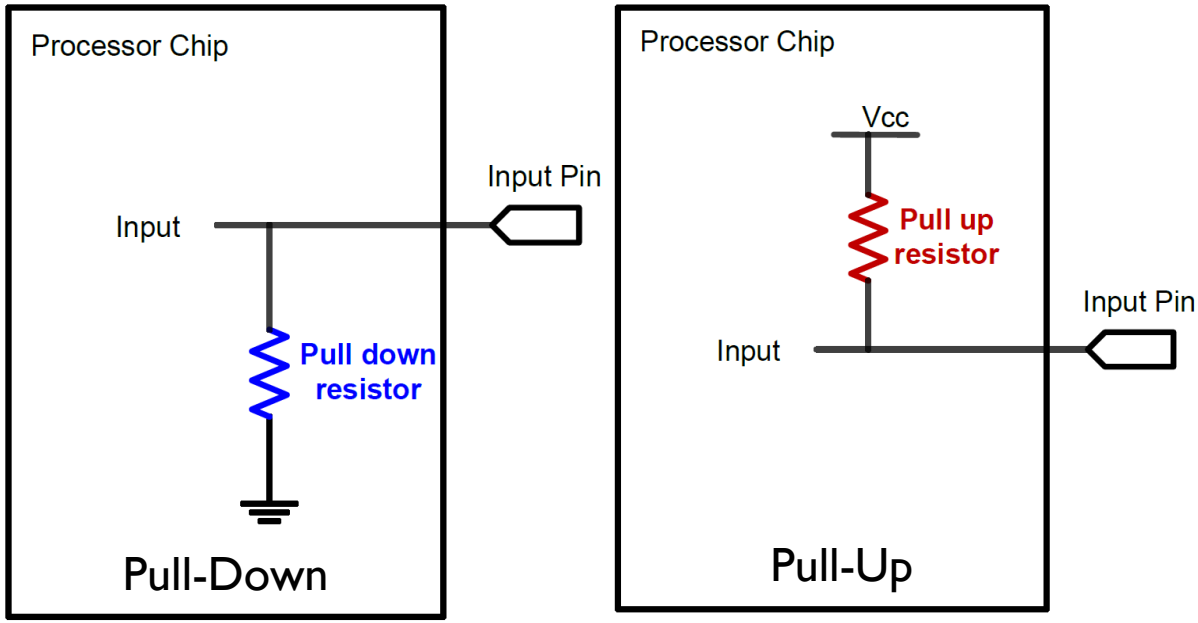
Pin 2 Pin 1 Pin 0

Bits 2y+1:2y **PUPDy[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

```
// No pull-up, pull-down
GPIOA->PUPDR &= ~3UL;
```



GPIO Input Data Register (IDR)

- 16 bits reserved, 16 data bits (1 bit per pin)

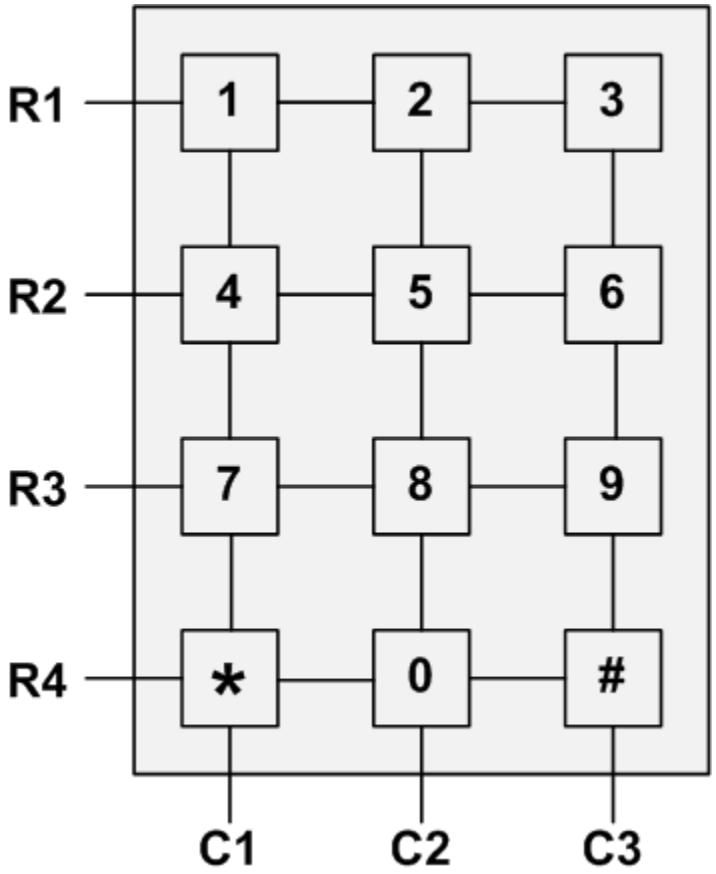
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ID15	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

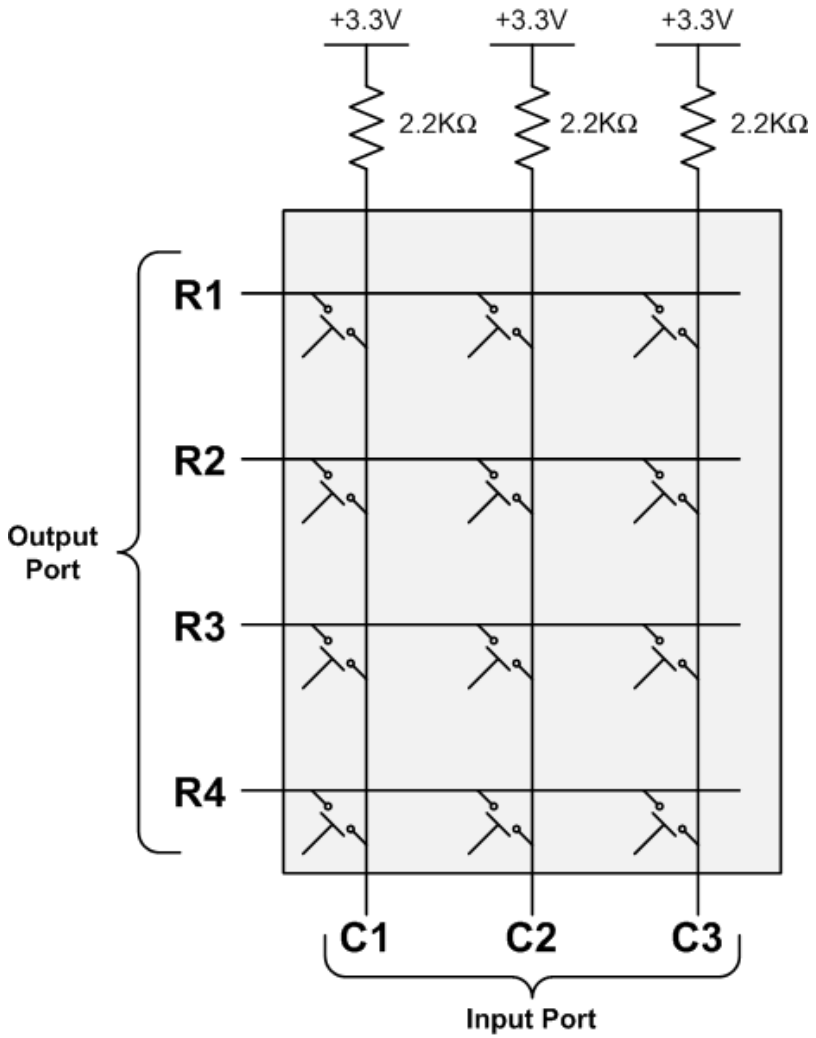
```
// Demo of reading pin 7
uint32_t mask = 1UL<<7;
uint32_t input = (GPIOA->IDR & mask) == mask;

or

uint32_t input = (GPIOA->IDR & mask) >> 7;
```

Read Input of Pin PA.0

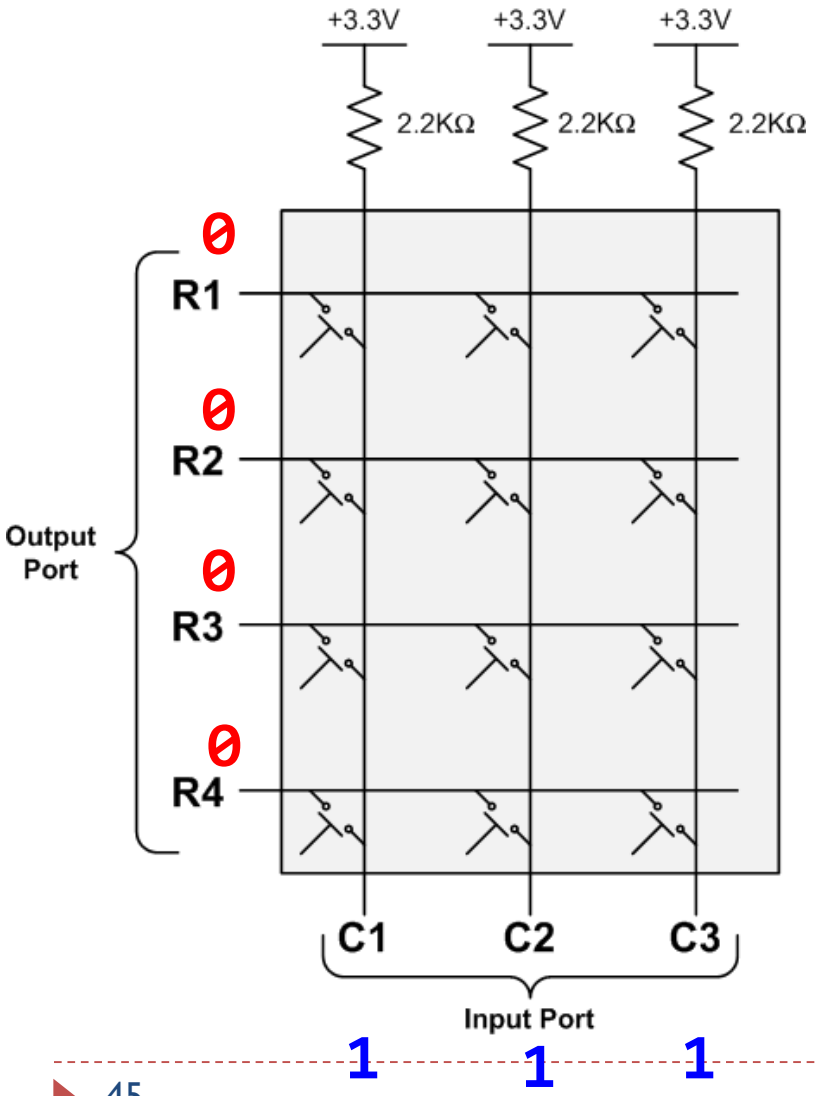
```
uint32_t input;

RCC->AHB2ENR |= RCC_AHB2ENR_GPIOAEN;  // Enable clock of Port A

GPIOA->MODER &= ~3UL; // Set PA.0 as digital input

GPIOA->PUPDR &= ~3UL; // No pull-up, no pull-down

// Read pin 0
input = (GPIOA->IDR & 1UL);

if (input == 0) {
   // Center of joystick is not pressed
   ...
} else {
   // Center of joystick is pressed
   ...
}
```

Keypad Scan


R1
R2
R3
R4
1
2
3
4
5
6
7
8
9
*
0
#
C1
C2
C3


+3.3V
+3.3V
+3.3V
2.2KΩ
2.2KΩ
2.2KΩ
R1
R2
R3
R4
Output Port
C1
C2
C3
Input Port

Keypad Scan



Step 1: Set Output

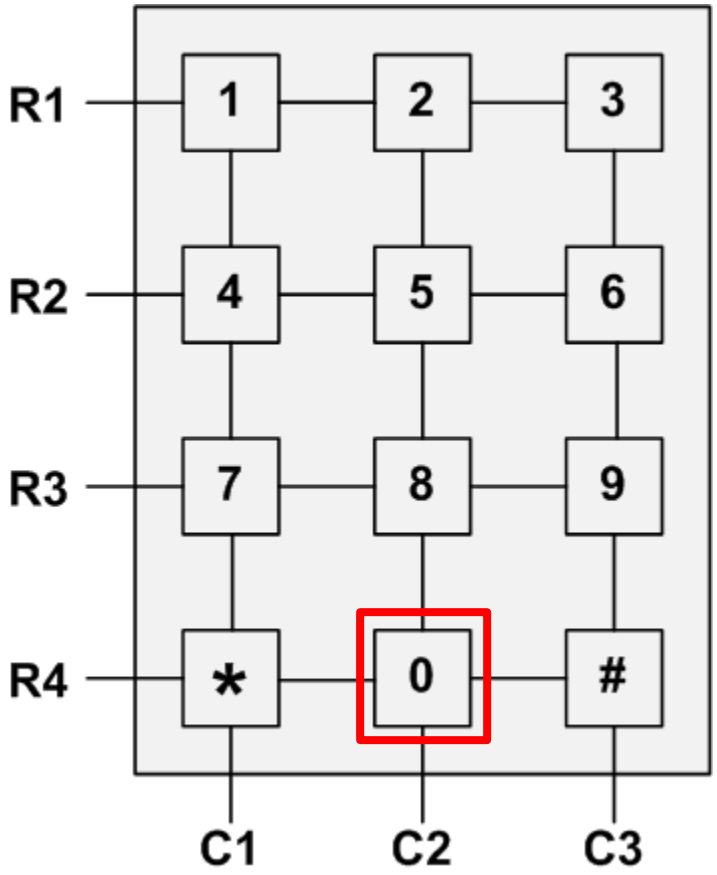
`R1,R2,R3,R4 = 0000`

Step 2: Read Input

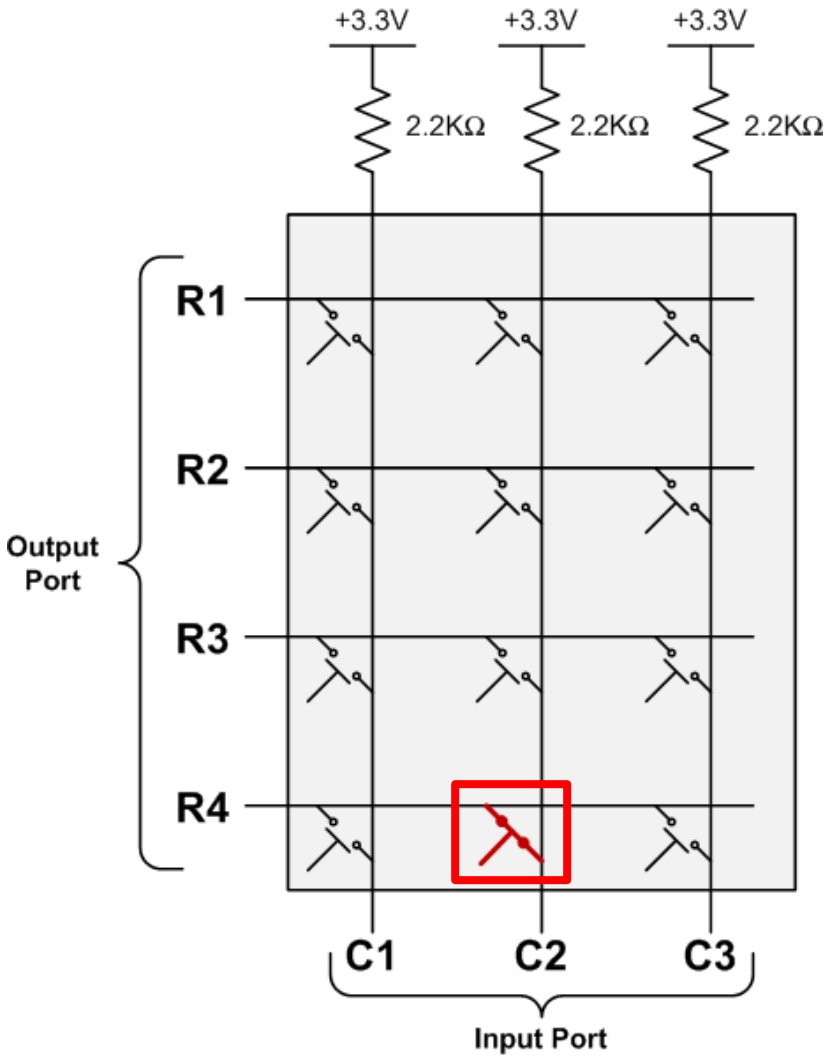
`C1,C2,C3 = 111`

⟹ No key pressed

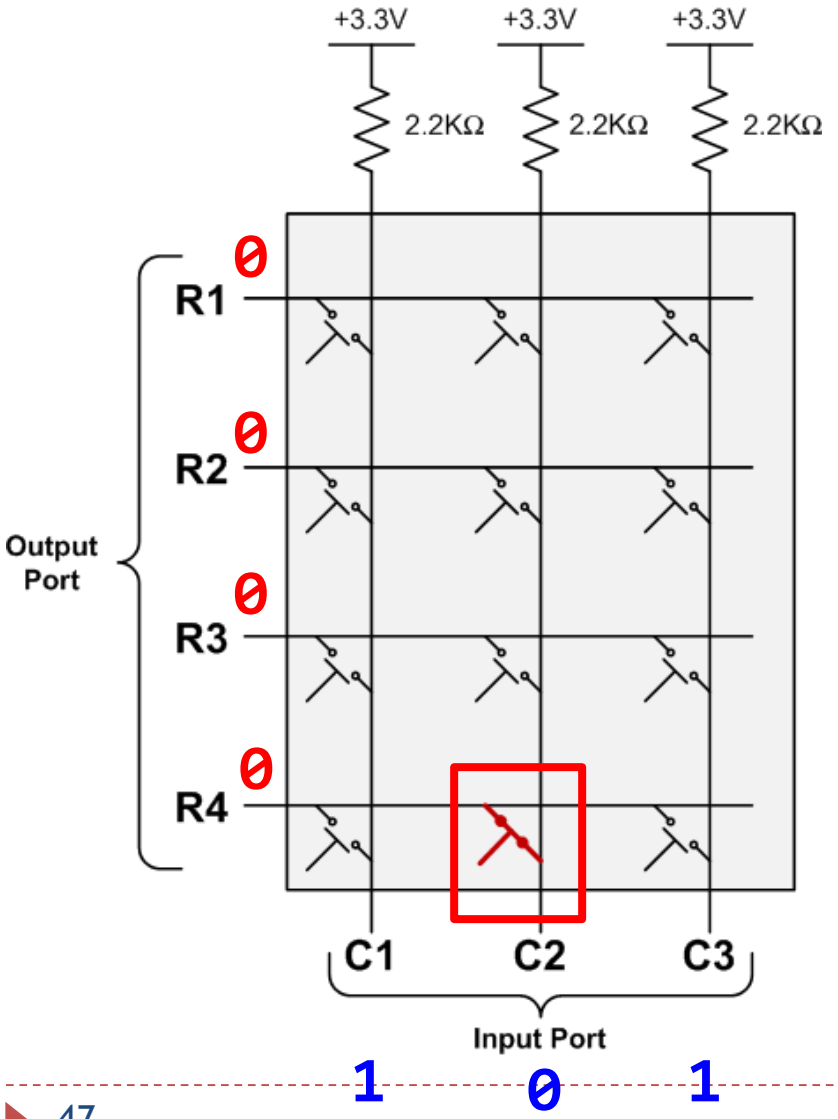
Keypad Scan



Key "0" is pressed



Keypad Scan



Step 1: Set Output

`R1,R2,R3,R4 = 0000`

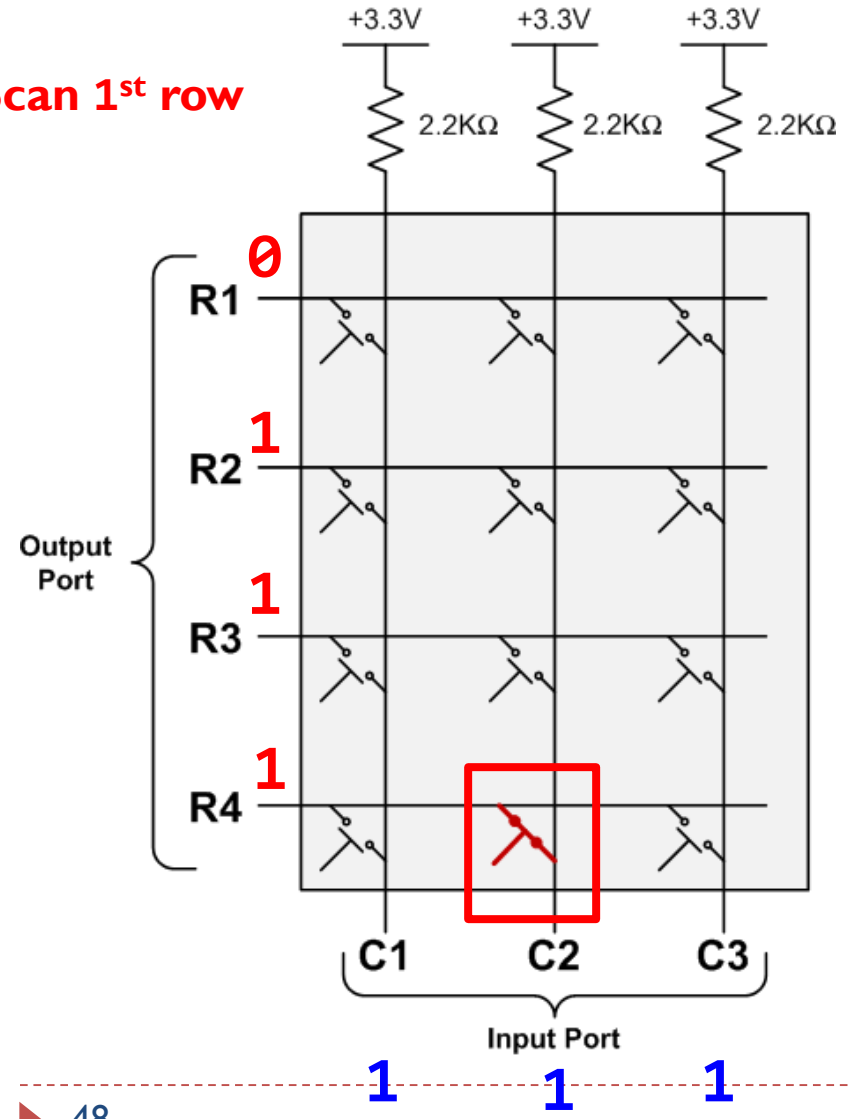
Step 2: Read Input

`C1,C2,C3 = 101`

⟹ Some key in 2nd column is pressed down

Keypad Scan

Scan 1st row



Step 1: Set Output

```
R1,R2,R3,R4 = 0000
```

Step 2: Read Input

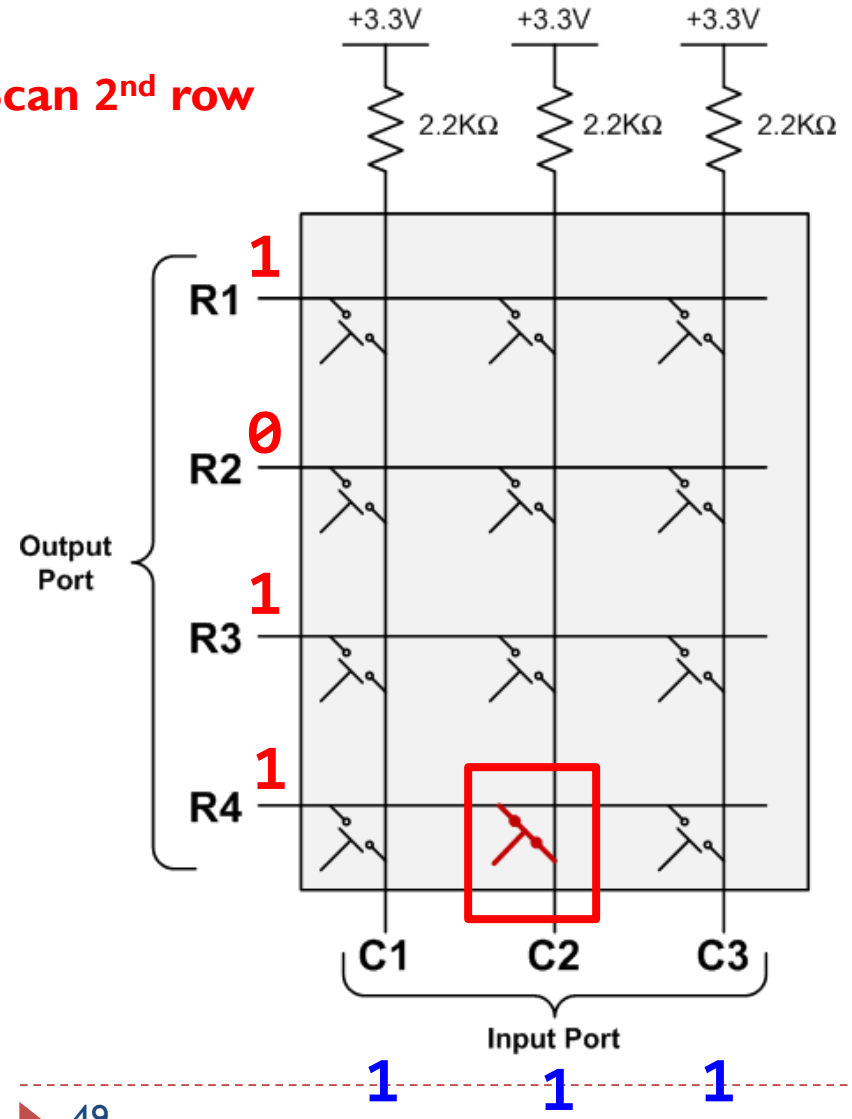
```
C1,C2,C3 = 101
```

→ Step 3a: Scan 1st row

```
R1,R2,R3,R4 = 0111
C1,C2,C3 = 111
```

⇒ No key in 1st row is pressed down

Keypad Scan



Step 1: Set Output

`R1,R2,R3,R4 = 0000`

Step 2: Read Input

`C1,C2,C3 = 101`

→ Step 3b: Scan 2nd row

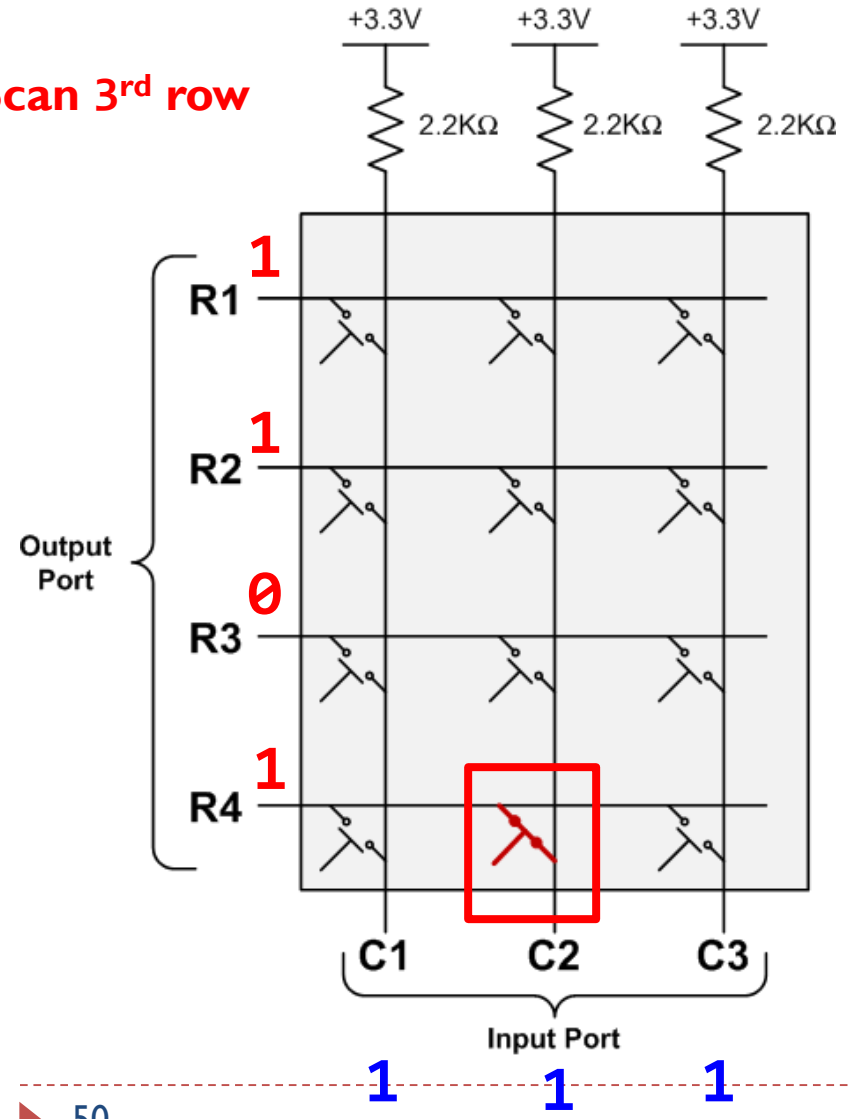
`R1,R2,R3,R4 = 1011`

`C1,C2,C3 = 111`

⇒ No key in 2nd row is pressed down

Keypad Scan

Scan 3rd row



Step 1: Set Output

`R1,R2,R3,R4 = 0000`

Step 2: Read Input

`C1,C2,C3 = 101`

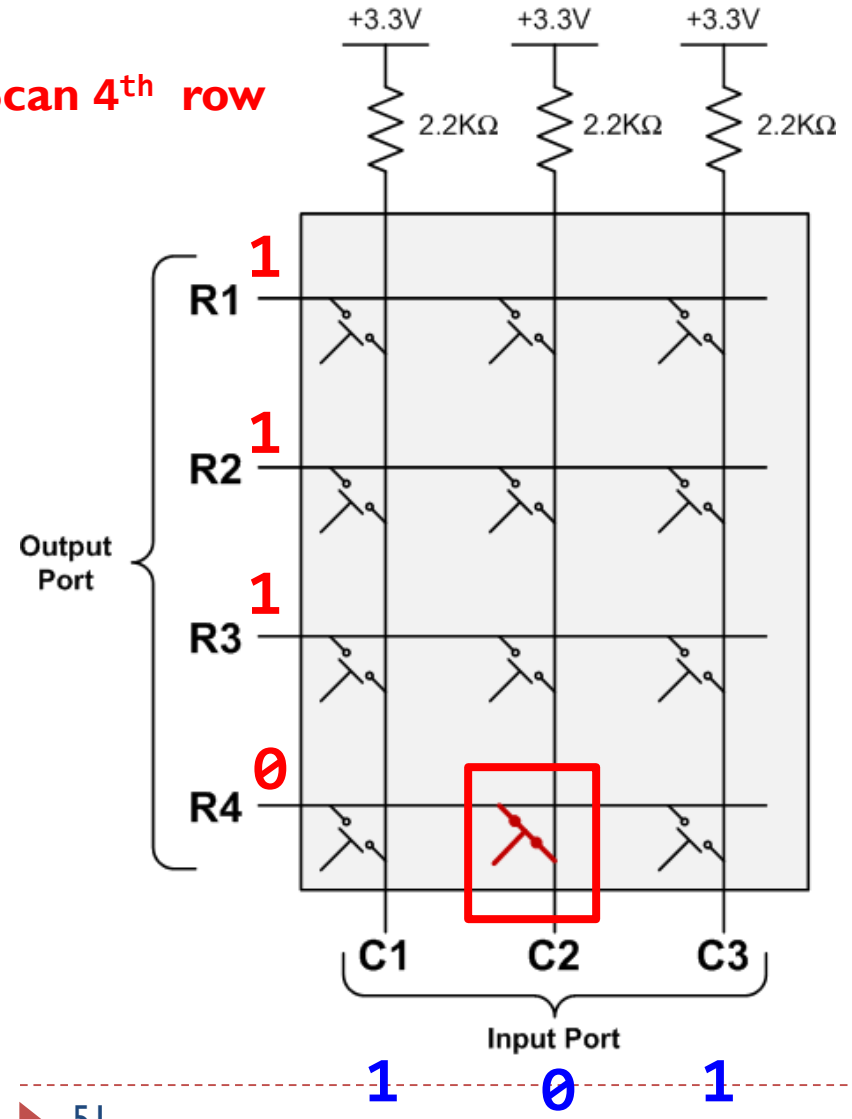
→ Step 3c: Scan 3rd row

`R1,R2,R3,R4 = 1101`

`C1,C2,C3 = 111`

⇒ No key in 3rd row is pressed down

Keypad Scan



Step 1: Set Output

```
R1,R2,R3,R4 = 0000
```

Step 2: Read Input

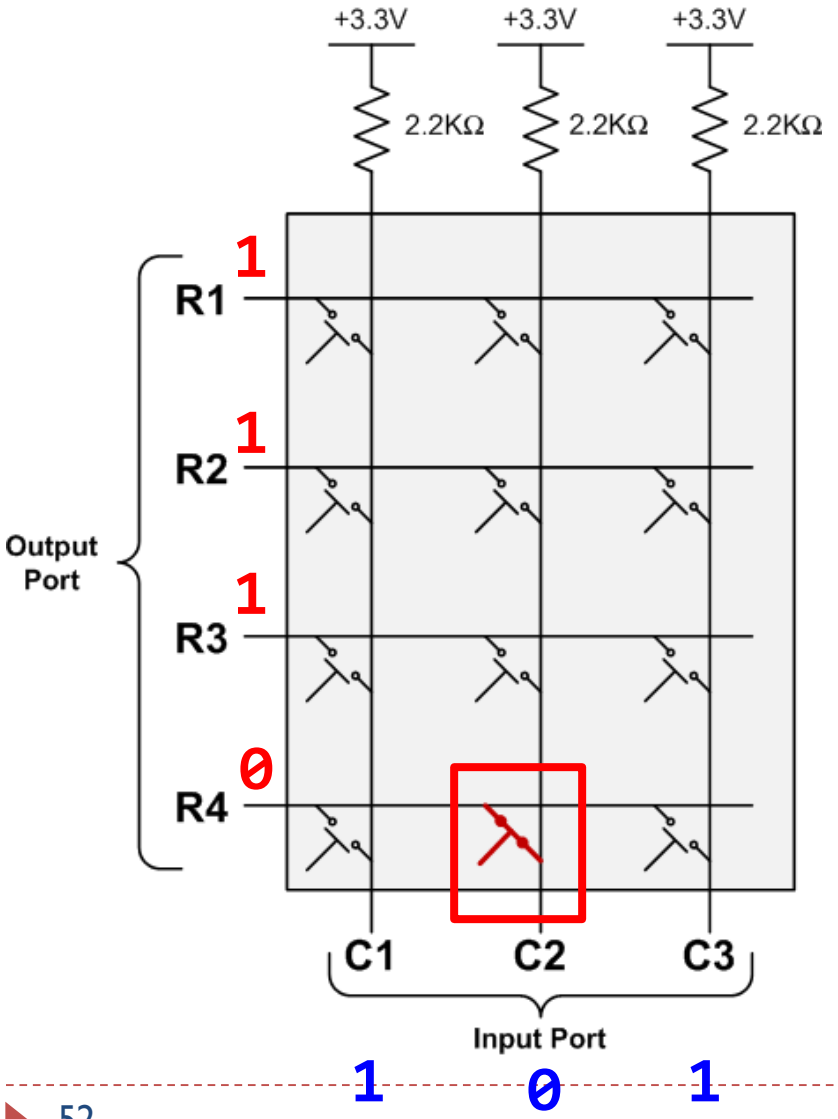
```
C1,C2,C3 = 101
```

→ Step 3d: Scan 4th row

```
R1,R2,R3,R4 = 1110
C1,C2,C3 = 101
```

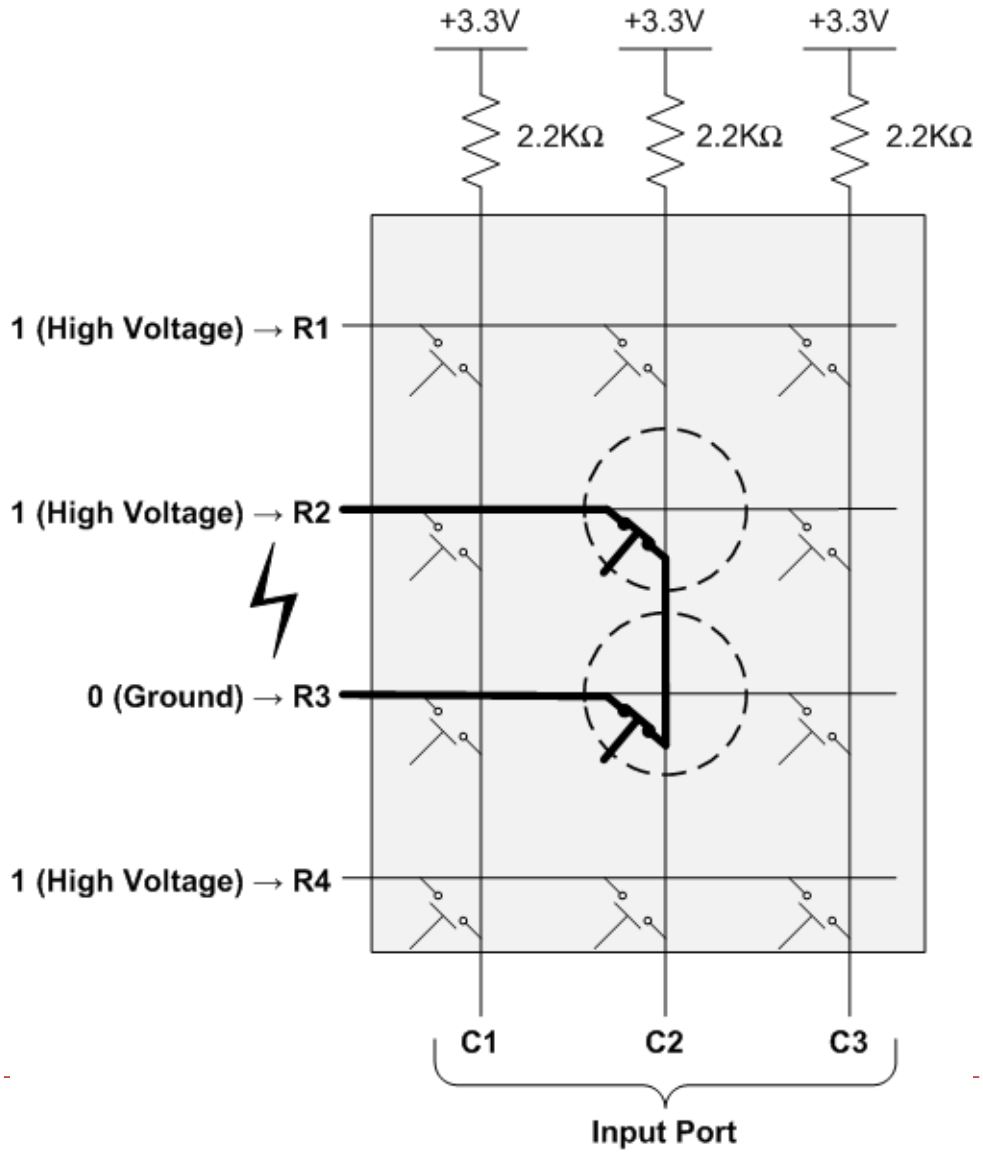
⇒ key in 4th row is pressed down

Keypad Scan



⟹ Key pressed is located at the second column and the fourth row.

Keypad Scan


+3.3V
+3.3V
+3.3V
2.2KΩ
2.2KΩ
2.2KΩ
1 (High Voltage) → R1
1 (High Voltage) → R2
0 (Ground) → R3
1 (High Voltage) → R4
C1
C2
C3
Input Port

I/O Debouncing

- Example signal when a button is pressed