

EE 304 Embedded Systems

Mehmet Bozdağ



Agenda

- Initializing STM32 Peripherals Using CMSIS
- Clock Setting
- GPIO Configuration
- Input / Output Interfacing

Initializing STM32 Peripherals Using CMSIS

- Include CMSIS
- Clock Configuration
- Peripheral Clock Enable
- Peripheral Pin Configuration
- Peripheral Initialization and Configuration
- Interrupt Configuration (if applicable)
- Peripheral Enable
- Error Handling and Flags (if applicable)
- Optimization and Resource Management
- Testing and Debugging

Include CMSIS & Clock Configuration

- The first step in configuring STM32 peripherals using CMSIS is to include the necessary header files:

```
#include "stm32f10x.h"
```

- Proper system clock configuration is fundamental to system's performance and synchronization.
- Ensure that the system clock source, frequency, and PLL settings match requirements.

SystemInit(); // Initializes the system clock and other critical settings.

Template

```
#include <stm32f10x.h>    // File name depends on device used

void delay (uint32_t time) {}

void Device_Initialization (void)
{
    // Configure & Initialize MCU
}

// The processor clock is initialized by CMSIS startup + system file
void main (void)
{
    Device_Initialization (); // Configure & Initialize MCU
    while (1) {               // Endless Loop (the Super-Loop)
        // Do something
    }
}
```

Let's do it...

- **SystemInit();**
- GPIO clock
- GPIO config
- Delay function
- Task (blink)

SystemInit()

- Setups the system clock (System clock source, PLL Multiplier factors, AHB/APBx prescalers and Flash settings).
- This function is called at startup just after reset and before branch to main program.
- This call is made inside the "startup_stm32f10x_xx.s" file.

Clocks

Three different clock sources can be used to drive the system clock (SYSCLK):

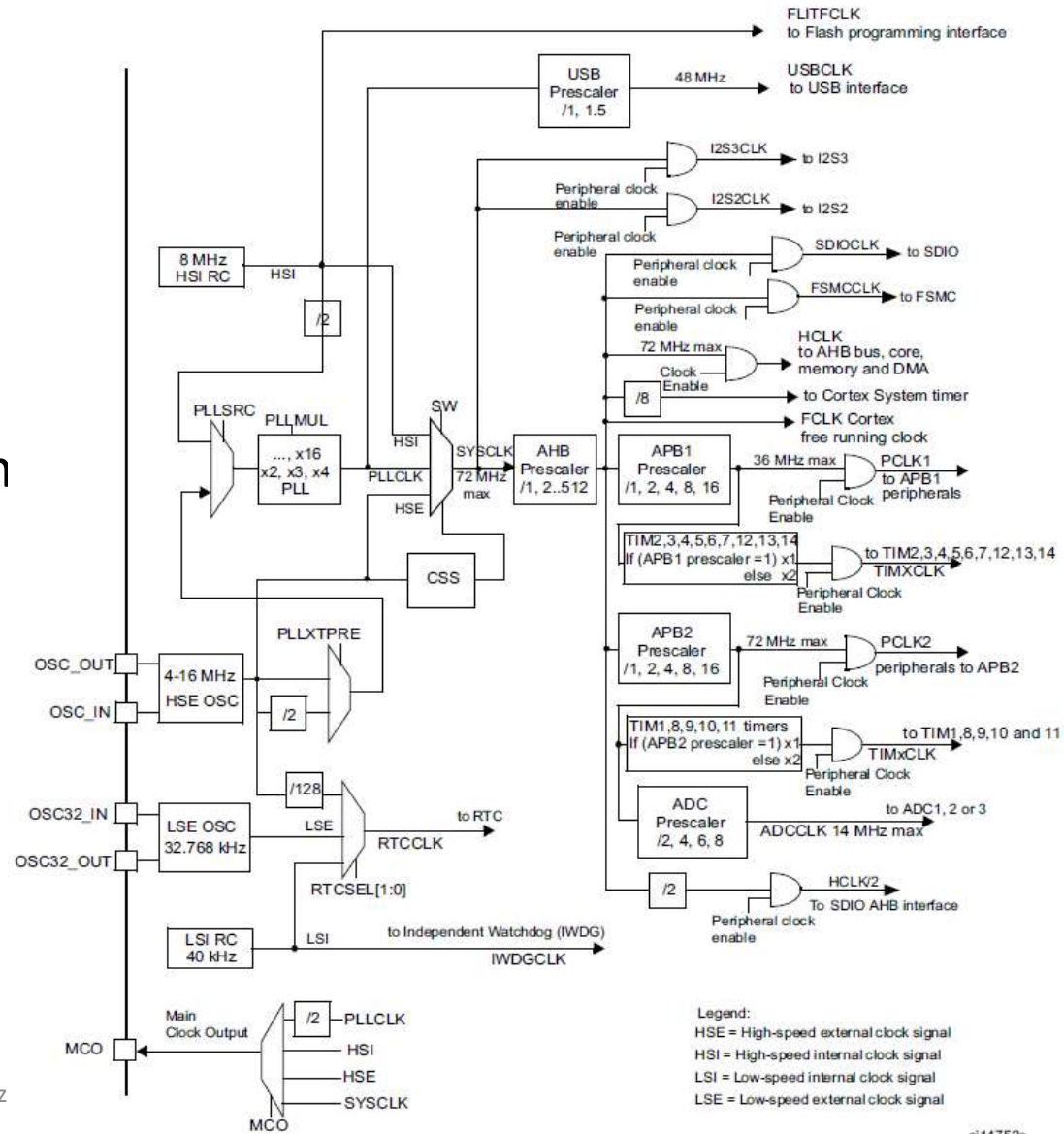
- HSI oscillator clock
- HSE oscillator clock
- PLL clock

It has the following two secondary clock sources:

- 40 kHz low speed internal RC (LSI RC) which drives the independent watchdog and optionally the RTC used for Auto-wakeup from Stop/Standby mode.
- 32.768 kHz low speed external crystal (LSE crystal) which optionally drives the real time clock (RTCCLK).

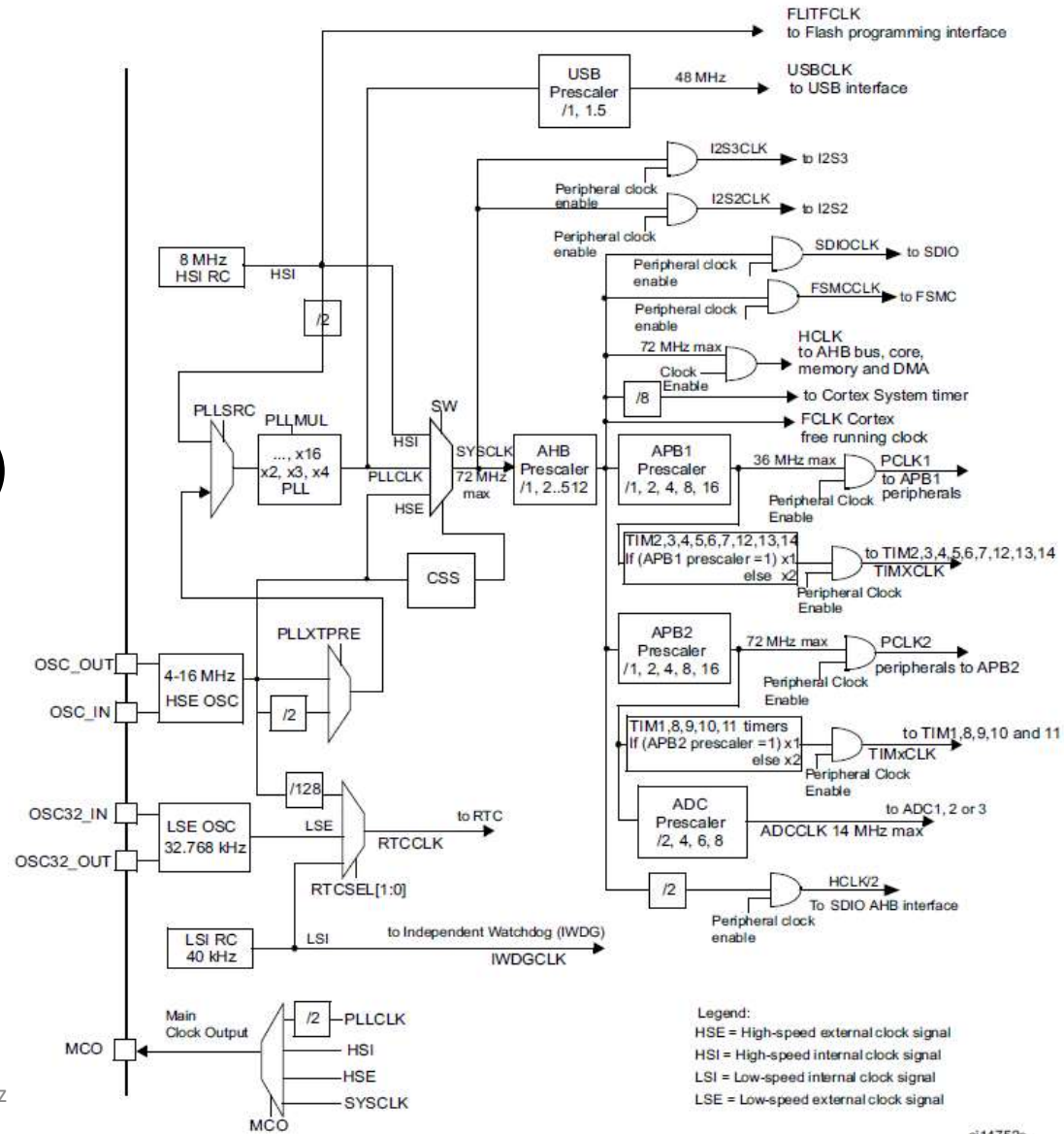
Clock Tree

- When the HSI is used as a PLL clock input, the maximum system clock frequency that can be achieved is 64 MHz.
- The maximum frequency of the AHB and the APB2 domains is 72 MHz.
- The maximum allowed frequency of the APB1 domain is 36 MHz.



Clock Tree

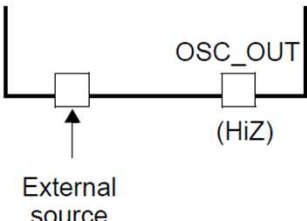
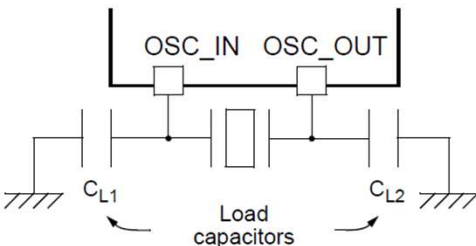
- The RCC feeds the Cortex® System Timer (SysTick) external clock with the AHB clock (HCLK) divided by 8.
- The SysTick can work either with this clock or with the Cortex® clock (HCLK), configurable in the SysTick Control and Status register.



HSE clock

The high speed external clock signal (HSE) can be generated from two possible clock sources:

- HSE external crystal/ceramic resonator
- HSE user external clock

Clock source	Hardware configuration
External clock	
Crystal/ceramic resonators	

HSI clock

- The HSI clock signal is generated from an internal 8 MHz RC Oscillator and can be used directly as a system clock or divided by 2 to be used as PLL input.
- The HSI RC oscillator has the advantage of providing a clock source at low cost (no external components).
- It also has a faster startup time than the HSE crystal oscillator however, even with calibration the frequency is less accurate than an external crystal oscillator or ceramic resonator.

HSI clock

- Manufacturing variations impact RC oscillator frequencies.
- Factory calibration for 1% accuracy at 25°C.
- After reset, factory calibration values loaded in HSICAL bits (Clock Control Register).
- HSI RC output clock released after HSIRDY bit is set by hardware.
- HSI RC can be switched on/off using HSION bit in RCC_CR.
- HSI signal serves as a backup source (Auxiliary clock) in case of HSE crystal oscillator failure.

PLL

- The internal PLL can be used to multiply the HSI RC output or HSE crystal output clock frequency.
- The PLL configuration (selection of HSI oscillator divided by 2 or HSE oscillator for PLL input clock, and multiplication factor) must be done before enabling the PLL. Once the PLL enabled, these parameters cannot be changed.
- If the USB interface is used in the application, the PLL must be programmed to output 48 or 72 MHz. This is needed to provide a 48 MHz USBCLK.

LSE clock

- The LSE crystal is a 32.768 kHz Low Speed External crystal or ceramic resonator. It has the advantage providing a low-power but highly accurate clock source to the real-time clock peripheral (RTC) for clock/calendar or other timing functions.
- The LSE crystal is switched on and off using the LSEON bit in Backup domain control register (RCC_BDCR).

LSI clock

- The LSI RC acts as an low-power clock source that can be kept running in Stop and Standby mode for the independent watchdog (IWDG) and Auto-wakeup unit (AWU).
- The clock frequency is around 40 kHz (between 30 kHz and 60 kHz).
- The LSI RC can be switched on and off using the LSION bit in the Control/status register (RCC_CSR).

System clock (SYSCLK) selection

- After a system reset, the HSI oscillator is selected as system clock.
- An external 3-25 MHz clock can be selected, in which case it is monitored for failure. If failure is detected, the system automatically switches back to the internal RC oscillator.
- When a clock source is used directly or through the PLL as system clock, it is not possible to stop it.
- A switch from one clock source to another occurs only if the target clock source is ready (clock stable after startup delay or PLL locked).

System clock (SYSCLK) selection

- If a clock source which is not yet ready is selected, the switch will occur when the clock source will be ready.
- Status bits in the Clock control register (RCC_CR) indicate which clock(s) is (are) ready and which clock is currently used as system clock.

RTC clock & Watchdog clock

- The RTCCLK clock source can be either the HSE/128, LSE or LSI clocks.
- If the Independent watchdog (IWDG) is started by either hardware option or software access, the LSI oscillator is forced ON and cannot be disabled. After the LSI oscillator temporization, the clock is provided to the IWDG.



Clock-out Capability

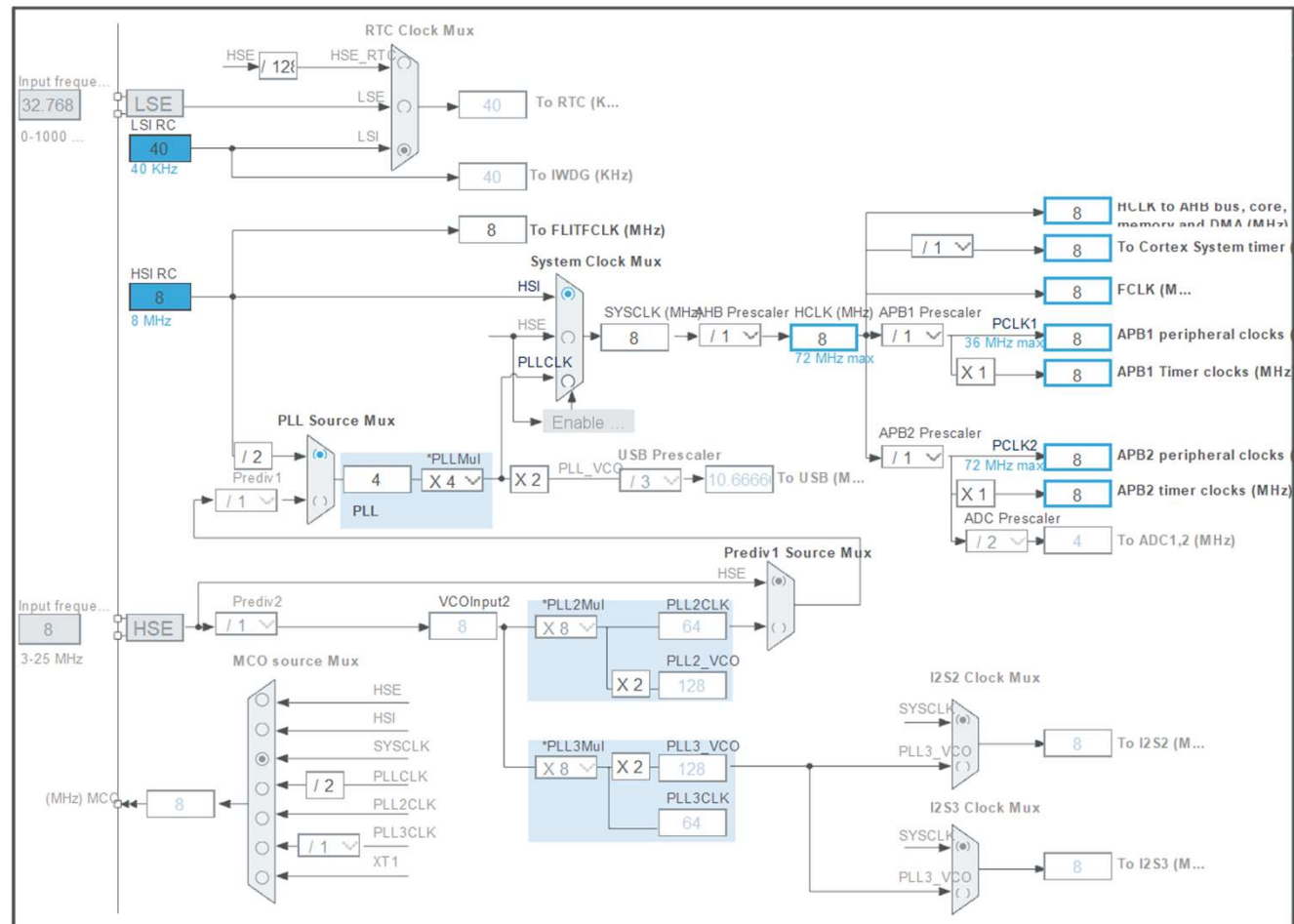
The microcontroller clock output (MCO) capability allows the clock to be output onto the external MCO pin. The configuration registers of the corresponding GPIO port must be programmed in alternate function mode. One of 8 clock signals can be selected as the MCO clock.

- SYSCLK
- HSI
- HSE
- PLL clock divided by 2 selected
- PLL2 clock selected
- PLL3 clock divided by 2 selected
- XT1 external 3-25 MHz oscillator clock selected (for Ethernet)
- PLL3 clock selected (for Ethernet)

The selected clock to output onto MCO must not exceed 50 MHz (the maximum I/O speed).

Clock Tree CubeMX

- Test it.



RCC registers

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0						
0x00	RCC_CR	Reserved					PLLRDY	PLLON	Reserved				CSSON	HSEBYP	HSEON	HSICAL[7:0]				HSITRIM[4:0]				Reserved				HSIRDY	HSION										
	Reset value						0	0	0					0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	1				
0x04	RCC_CFGR	Reserved				MCO [2:0]		Reserved	USBPRE	PLLMUL [3:0]				PLLTPRE	PLLSRC	ADCPRE [1:0]	PPRE2 [2:0]		PPRE1 [2:0]		HPRE[3:0]		SWS [1:0]		SW [1:0]														
	Reset value					0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0					
0x08	RCC_CIR	Reserved								CSSC	Reserved	PLLRDYC	HSERDYC	HSIRDYC	LSERDYC	LSIRDYC	Reserved				PLLRDYE	HSERDYE	HSIRDYE	LSERDYE	LSIRDYE	CSSF	Reserved				PLLRDYF	HSERDYF	HSIRDYF	LSERDYF	LSIRDYF				
	Reset value									0		0	0	0	0	0	0					0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
0x0C	RCC_APB2RSTR	Reserved										TIM11RST	TIM10RST	TIM9RST	Reserved				ADC3RST	USART1RST	TIM8RST	SPI1RST	TIM1RST	ADC2RST	ADC1RST	IOPGRST	IOPFRST	IOPDRST	IOPCRST	IOPBRST	IOPARST	Reserved		AFIOIRST					
	Reset value											0	0	0					0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x010	RCC_APB1RSTR	Reserved	DACRST	PWRST	BKPRST	Reserved	CANRST	Reserved	Reserved	Reserved	I2C2RST	I2C1RST	UART5RST	UART4RST	USART3RST	USART2RST	Reserved	SPI3RST	SPI2RST	Reserved		Reserved	VWDGGRST	Reserved		TIM14RST	TIM13RST	TIM12RST	TIM7RST	TIM6RST	TIM5RST	TIM4RST	TIM3RST	TIM2RST	TIM1RST				
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
0x14	RCC_AHBENR	Reserved																Reserved		SDIOEN	Reserved	FSMCEN	Reserved	CRCEN	Reserved	FLITFEN	Reserved	SRAMEN	DMA2EN	DMA1EN	Reserved		AFIOEN						
	Reset value																			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x18	RCC_APB2ENR	Reserved										TIM11EN	TIM10EN	TIM9EN	Reserved				ADC3EN	USART1EN	TIM8EN	SPI1EN	TIM1EN	ADC2EN	ADC1EN	IOPGEN	IOPFEN	IOPDEN	IOPCEN	IOPBEN	IOPAEN	Reserved		AFIOEN					
	Reset value											0	0	0					0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x1C	RCC_APB1ENR	Reserved	DACEN	PWREN	BKPEN	Reserved	CANEN	Reserved	Reserved	Reserved	I2C2EN	I2C1EN	UART5EN	UART4EN	USART3EN	USART2EN	Reserved	SPI3EN	SPI2EN	Reserved		Reserved	VWDGEN	Reserved		TIM14EN	TIM13EN	TIM12EN	TIM7EN	TIM6EN	TIM5EN	TIM4EN	TIM3EN	TIM2EN	TIM1EN				
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
0x20	RCC_BDCR	Reserved														BDRST	RTCEN	Reserved				RTCSEL [1:0]		Reserved				LSEBYP	LSERDY	LSEON									
	Reset value															0	0							0		0					0	0	0	0	0	0	0	0	

Clock control register (RCC_CR)

Address offset: 0x00

Reset value: 0x0000 XX83 where X is undefined.

Access: no wait state, word, half-word and byte access

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		PLL3 RDY	PLL3 ON	PLL2 RDY	PLL2 ON	PLL RDY	PLL ON	Reserved				CSSON	HSEBYP	HSE RDY	HSEON
		r	rw	r	rw	r	rw					rw	rw	r	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HSICAL[7:0]								HSITRIM[4:0]					Res.	HSIRDY	HSION
r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw		r	rw

Clock configuration register (RCC_CFGR)

Address offset: 0x04

Reset value: 0x0000 0000

Access: $0 \leq \text{wait state} \leq 2$, word, half-word and byte access

1 or 2 wait states inserted only if the access occurs during a clock source switch.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved				MCO[3:0]				Res.	OTGFS PRE	PLLMUL[3:0]				PLL XTPRE	PLL SRC
				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
ADC PRE[1:0]		PPRE2[2:0]			PPRE1[2:0]			HPRE[3:0]				SWS[1:0]		SW[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	r	r	rw	rw

AHB Peripheral Clock enable register (RCC_AHBENR)

Address offset: 0x14

Reset value: 0x0000 0014

Access: no wait state, word, half-word and byte access

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															ETHMAC RXEN
															rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ETHMAC TXEN	ETHMACEN	Res.	OTGFSEN	Reserved					CRCEN	Res.	FLITFEN	Res.	SRAMEN	DMA2EN	DMA1EN
rw	rw		rw						rw		rw		rw	rw	rw

Clock configuration register2 (RCC_CFGR2)

Address offset: 0x2C

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved													I2S3SRC	I2S2SRC	PREDIV1SRC
													rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PLL3MUL[3:0]				PLL2MUL[3:0]				PREDIV2[3:0]				PREDIV1[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

APB1 peripheral clock enable register (RCC_APB1ENR)

Address: 0x1C

Reset value: 0x0000 0000

Access: word, half-word and byte access

No wait state, except if the access occurs while an access to a peripheral on APB1 domain is on going. In this case, wait states are inserted until this access to APB1 peripheral is finished.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		DAC EN	PWR EN	BKP EN	CAN2 EN	CAN1 EN	Reserved		I2C2 EN	I2C1 EN	UART5EN	UART4EN	USART3 EN	USART2 EN	Res.
		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
SPI3 EN	SPI2 EN	Reserved		WWD GEN	Reserved					TIM7 EN	TIM6 EN	TIM5 EN	TIM4 EN	TIM3 EN	TIM2 EN
rw	rw			rw						rw	rw	rw	rw	rw	rw

Let's do it... - SystemInit();

```
void initClockHSI(void)
{
    // Configure clock setting for HSI
    RCC->CR |= RCC_CR_HSION; // Enable internal HSI (RC)

    // Wait for HSI to be ready
    while ((RCC->CR & RCC_CR_HSIRDY) != RCC_CR_HSIRDY);

    RCC->CFGR &= ~(RCC_CFGR_SW); // Clear SW bits
    RCC->CFGR |= RCC_CFGR_SW_HSI; // Set HSI as system clock

    // Wait for HSI to be system clock
    while ((RCC->CFGR & RCC_CFGR_SWS_HSI) != RCC_CFGR_SWS_HSI);
}
```

Let's do it...

- SystemInit();
- **GPIO Clock**
- **GPIO Config**
- Delay Function
- Task (blink)

GPIO Clock

APB2 peripheral clock enable register (RCC_APB2ENR)

Address: 0x18

Reset value: 0x0000 0000

Access: word, half-word and byte access

No wait states, except if the access occurs while an access to a peripheral in the APB2 domain is on going. In this case, wait states are inserted until the access to APB2 peripheral is finished.

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
Res.	USART 1EN	Res.	SPI1 EN	TIM1 EN	ADC2 EN	ADC1 EN	Reserved		IOPE EN	IOPD EN	IOPC EN	IOPB EN	IOPA EN	Res.	AFIO EN
	rw		rw	rw	rw	rw			rw	rw	rw	rw	rw		rw

GPIO Functional Description – STM32

Each of the general-purpose I/O ports has

- Two 32-bit configuration registers (GPIOx_CRL, GPIOx_CRH)
- Two 32-bit data registers (GPIOx_IDR, GPIOx_ODR)
- A 32-bit set/reset register (GPIOx_BSRR)
- A 16-bit reset register (GPIOx_BRR)
- A 32-bit locking register (GPIOx_LCKR).

Note: Each I/O port bit is freely programmable, however the I/O port registers have to be accessed as 32-bit words (half-word or byte accesses are not allowed). The purpose of the GPIOx_BSRR and GPIOx_BRR registers is to allow atomic read/modify accesses to any of the GPIO registers. This way, there is no risk that an IRQ occurs between the read and the modify access.

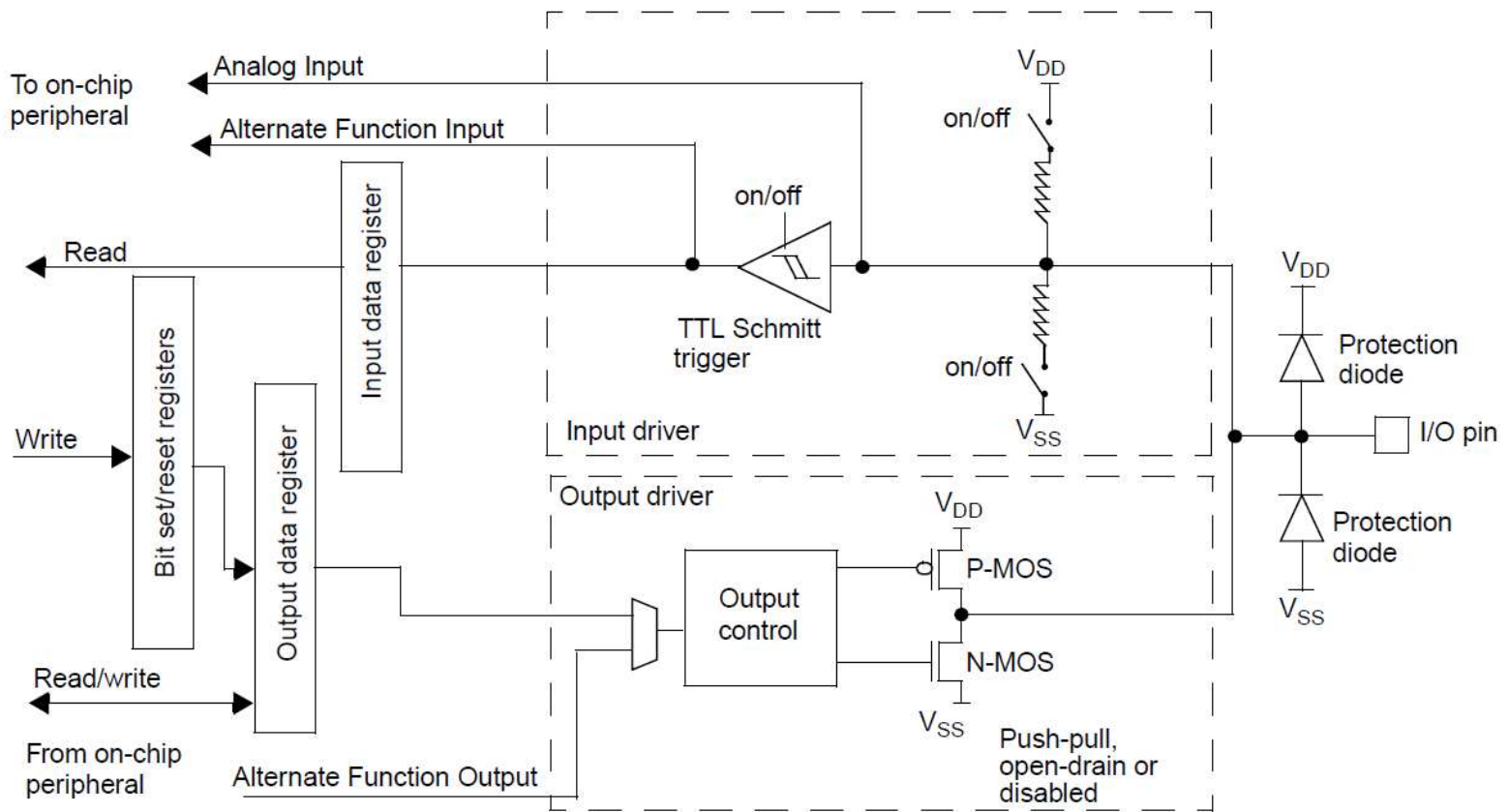
GPIO Registers

1. **Configuration Low (GPIOx->CRL):** These registers are used to set ports as input, output or alternate function (e.g., I2C, UART..)- port bits 0-7.
2. **Configuration High (GPIOx->CRH):** Configuring port bits 8-15
3. **Input data (GPIOx->IDR):** Read data from the port using this register.
4. **Output data (GPIOx->ODR):** Read /Write data to port using this register.
5. **Bit Set/Reset (GPIOx->BSRR):** Port pins can be set or reset bitwise.
6. **Bit Reset (GPIOx->BRR):** Port pins can be reset bitwise.
7. **Configuration Lock (GPIOx->LCKR):** This register is used to lock the configuration of the port bits. When the LOCK sequence has been applied on a port bit it is no longer possible to modify the value of the

GPIO Config

```
// Configure GPIO: Port D Pin0 as output
RCC->APB2ENR |= RCC_APB2ENR_IOPDEN; // Enable Port D clock
GPIOD->CRL |= GPIO_CRL_MODE0; // Output mode, max speed 50 MHz
GPIOD->CRL &= ~(GPIO_CRL_CNF0); // General purpose output push-pull
```

Basic structure of a standard I/O port bit

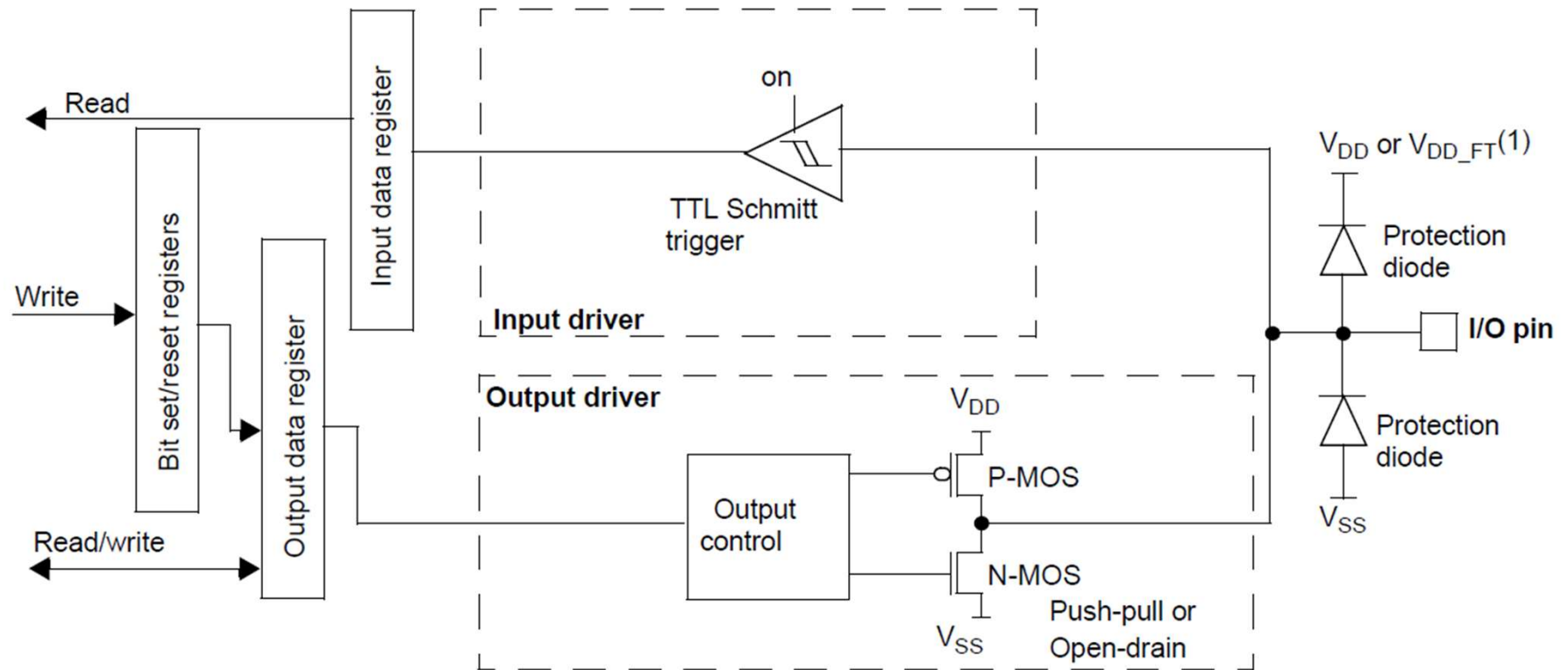


GPIO - Output configuration

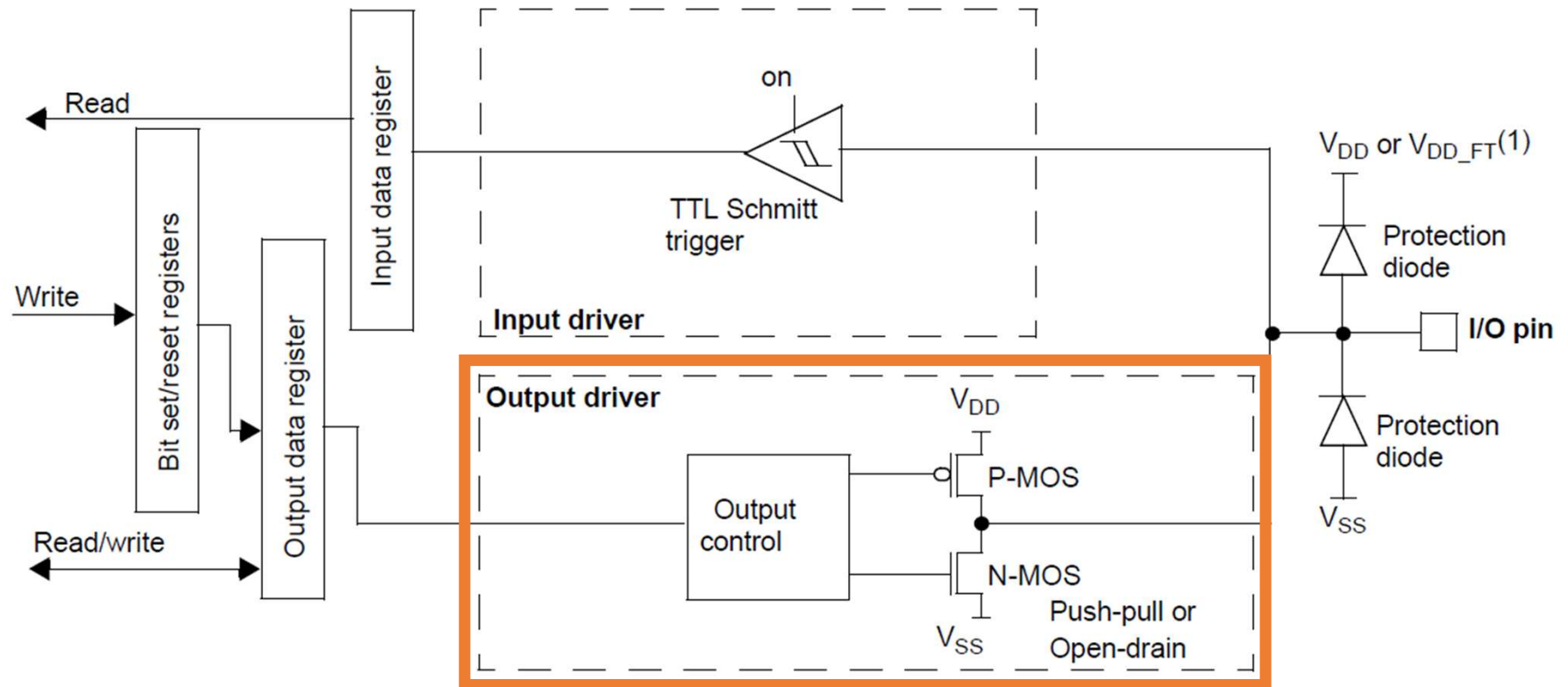
When the I/O Port is programmed as Output:

- The Output Buffer is enabled:
 - Open Drain Mode: A “0” in the Output register activates the N-MOS while a “1” in the Output register leaves the port in Hi-Z (the P-MOS is never activated)
 - Push-Pull Mode: A “0” in the Output register activates the N-MOS while a “1” in the Output register activates the P-MOS
- The Schmitt Trigger Input is activated.
- The weak pull-up and pull-down resistors are disabled. (High resistance > 10k)
- The data present on the I/O pin is sampled into the Input Data register every APB2 clock cycle
- A read access to the Input Data register gets the I/O state in open drain mode
- A read access to the Output Data register gets the last written value in Push-Pull mode

GPIO - Output configuration

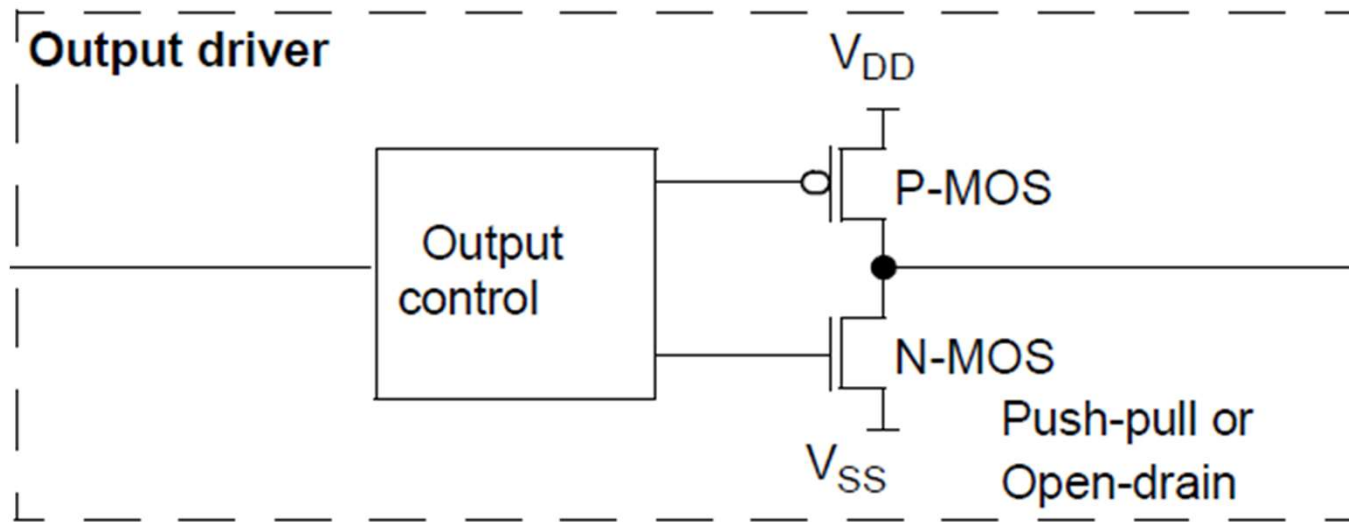


GPIO - Output configuration



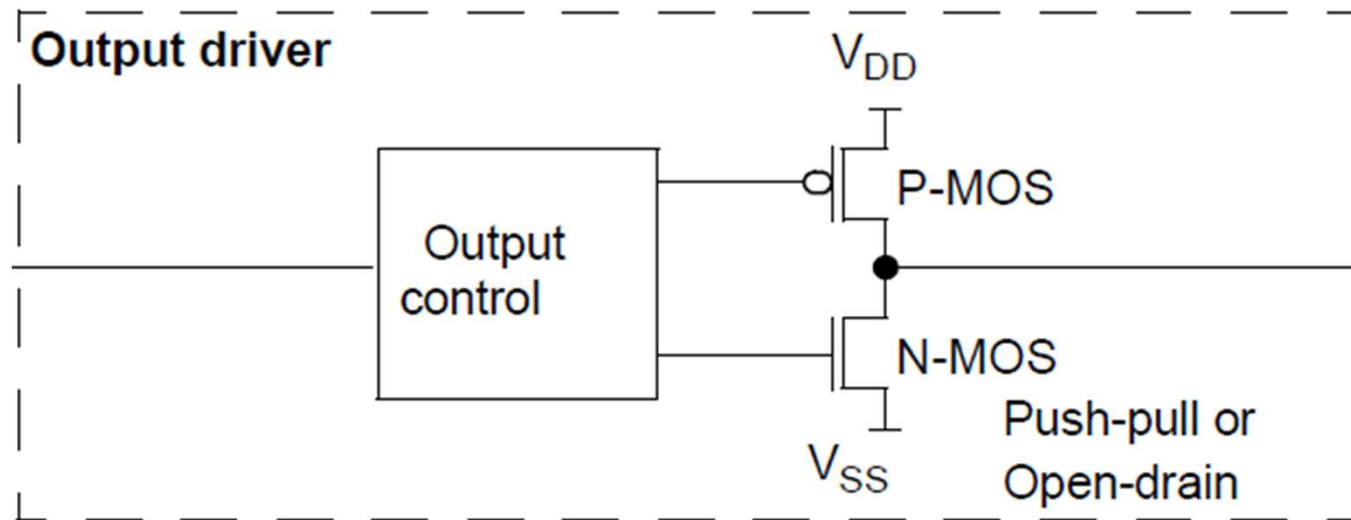
GPIO - Output configuration : Push Pull

- Push-pull mode allows the pin to supply and absorb current.
- If the digital output is 1, then the GPIO output pin is pulled up to the V_{CC} .



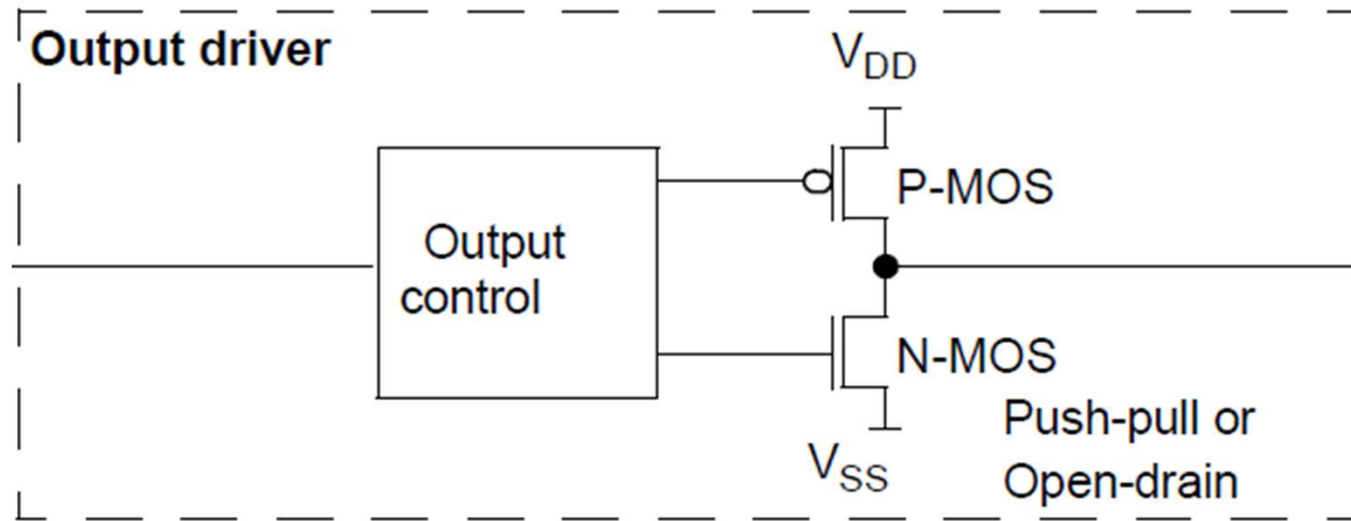
GPIO - Output configuration : Push Pull

- Push-pull mode allows the pin to supply and absorb current.
- If the digital output is 0, then the GPIO output pin is pulled down to the ground.



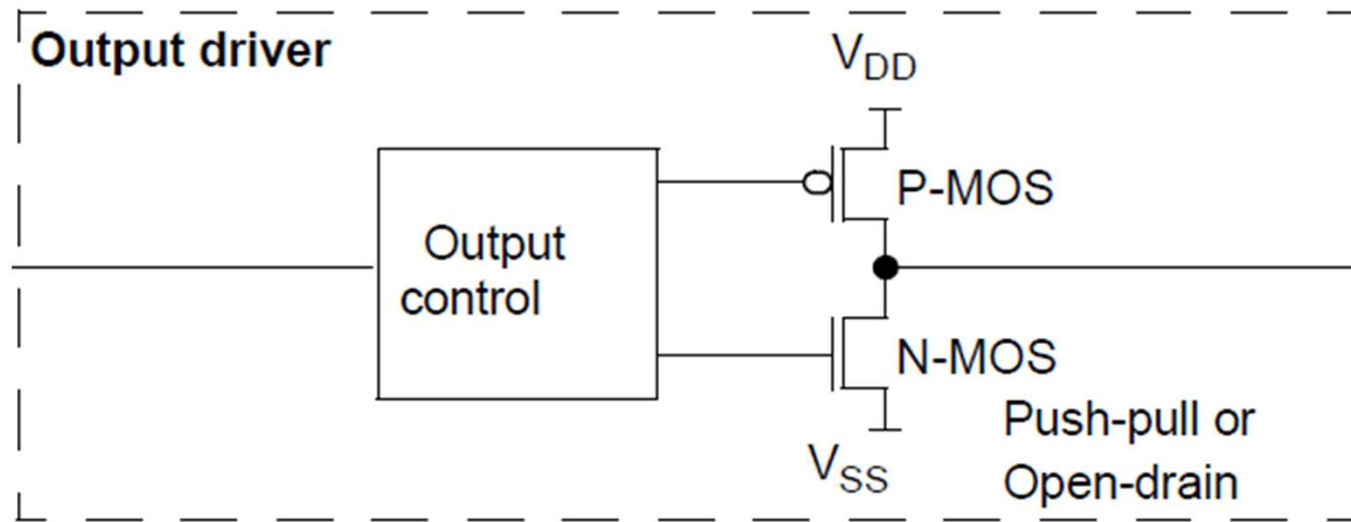
GPIO - Output configuration : Open Drain

- GPIO pin in open-drain(also called collector) mode can only absorb current.
- If the digital output is 0, then the output pin is pushed to the ground.



GPIO - Output configuration : Open Drain

- GPIO pin in open-drain(also called collector) mode can only absorb current.
- If the digital output is 1, then the output pin is floating (hi-Z).



Let's do it... - GPIO Clock & Config

```
void GPIO__Initialization(void)
{
    // Configure GPIO: Port B Pin0 as output
    RCC->APB2ENR |= RCC_APB2ENR_IOPBEN; // Enable Port B clock
    GPIOB->CRL |= GPIO_CRL_MODE0; // Output mode, max speed 50 MHz
    GPIOB->CRL &= ~(GPIO_CRL_CNF0); // General purpose output push-pull
}
```

Let's do it...

- SystemInit();
- GPIO Clock
- GPIO Config
- **Delay Function**
- Task (blink)

Let's do it... - Delay Function

```
void delay(uint32_t delayCycle)
{
    while (delayCycle--);
}
```

- This is called busy wait and it is not efficient.

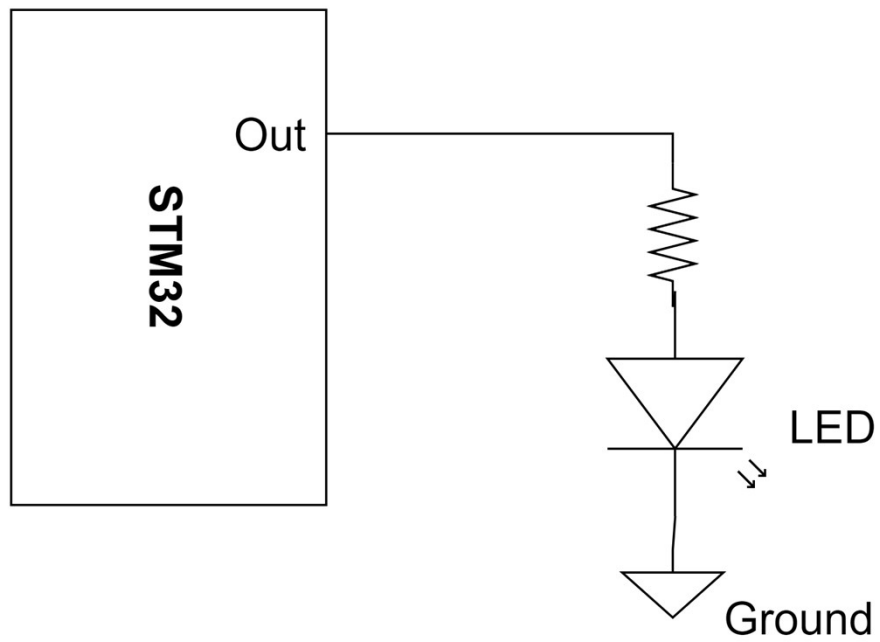
Let's do it...

- SystemInit();
- GPIO Clock
- GPIO Config
- Delay Function
- **Task (Blink)**

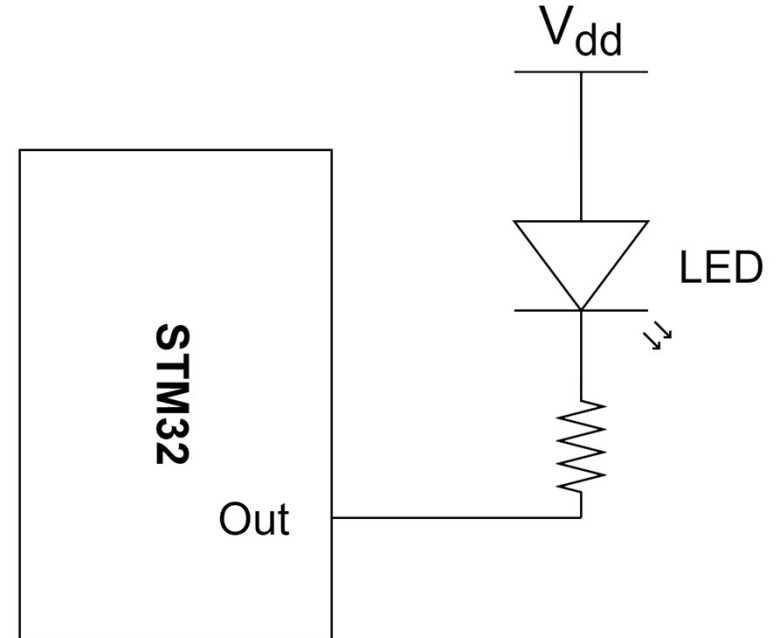
What should we write to out pin to turn on LED?

LED Outputs

Positive Logic (low current)



Negative Logic (low current)



Let's do it... - Task (Blink)

```
while (1)
{
    GPIOB->ODR = LED_PIN; // Set LED pin
    delay(DELAY_CYCLES);
    GPIOB->ODR = ~LED_PIN; // Reset LED pin
}
```

What is wrong?

Let's do it... - Task (Blink)

```
while (1)
{
    GPIOB->ODR = LED_PIN; // Set LED pin
    delay(DELAY_CYCLES);
    GPIOB->ODR = ~LED_PIN; // Reset LED pin
    delay(DELAY_CYCLES);
}
```

- It blink the LED but still not a good code. We should only change the relevant bits to avoid unexpected outcomes.

Let's do it... - Task (Blink)

```
while (1)
{
    GPIOB->ODR |= LED_PIN; // Set LED pin
    delay(DELAY_CYCLES);
    GPIOB->ODR &= ~LED_PIN; // Reset LED pin
    delay(DELAY_CYCLES);
}
```

Let's do it... - Task (Blink)

Just another way.

```
while (1)
{
    GPIOB->BSSR = LED_PIN; // Set LED pin
    delay(DELAY_CYCLES);
    GPIOB->BSSR = LED_PIN << 16; // Reset LED pin
    delay(DELAY_CYCLES);
}
```

Let's do it... - Task (Blink)

Just another way.

```
while (1)
{
    GPIOB->BSSR = LED_PIN; // Set LED pin
    delay(DELAY_CYCLES);
    GPIOB->BRR = LED_PIN; // Reset LED pin
    delay(DELAY_CYCLES);
}
```

Let's do it... - Task (Blink)

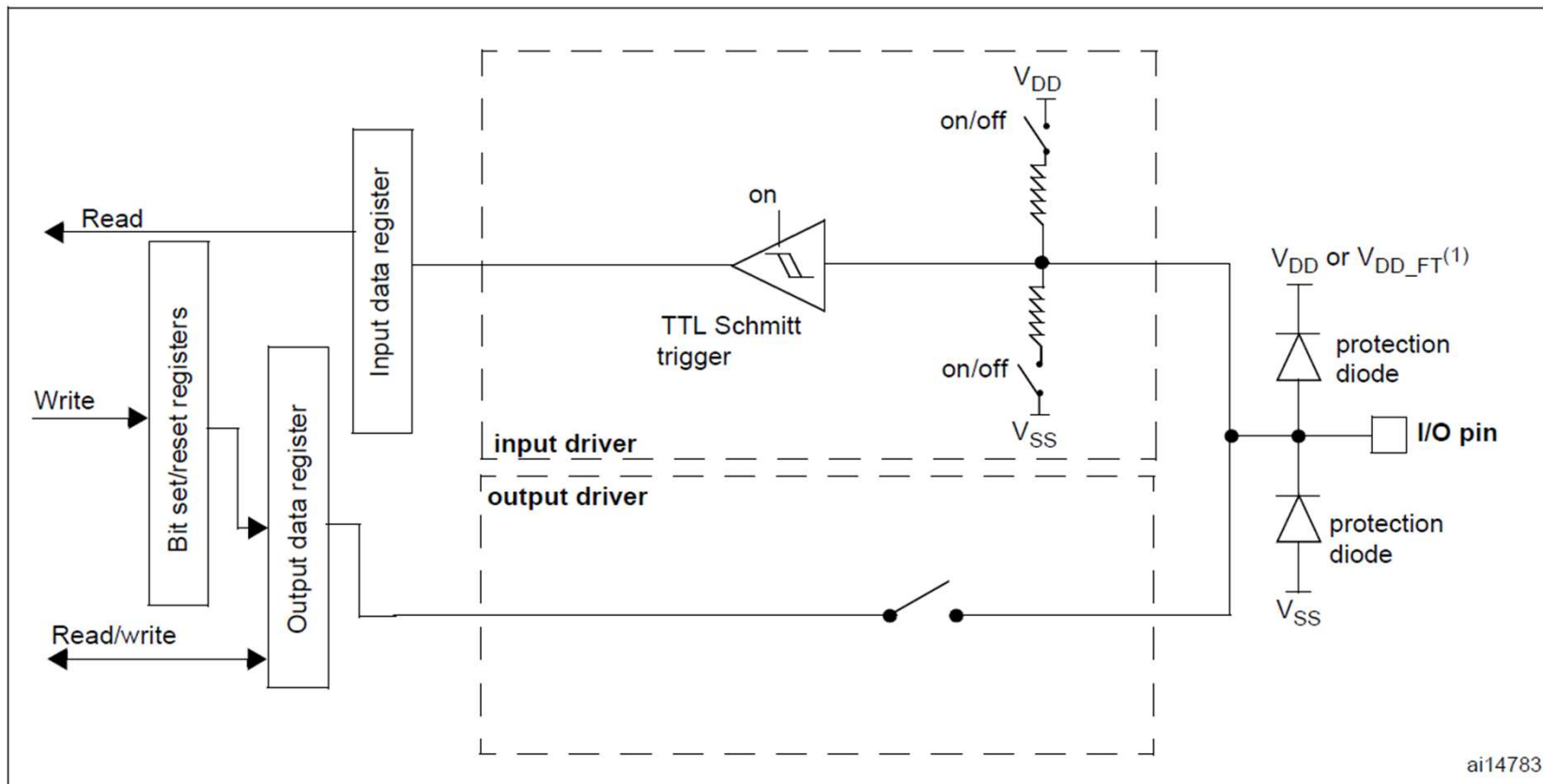
Just another way.

```
while (1)
{
    GPIOB->ODR ^= LED_PIN; // Toggle LED pin
    delay(DELAY_CYCLES);
}
```

LED as Debugger

- One of the important tasks in debugging a system is to observe when and where our software is executing.
- Ideally debugging shouldn't deviate the system from real time behaviour.
- **Intrusiveness** is defined as the degree to which the debugging code itself alters the performance of the system being tested.
- LED is a light-weighted option.

Input configuration



Input mode configuration

When a STM32 device I/O pin is configured as input, one of three options must be selected:

- Input with internal pull-up. Pull-up resistors are used in STM32 devices to ensure a well-defined logical level in case of floating input signal. Depending on application requirements, an external pull-up can be used instead.
- Input with internal pull-down. Pull-down resistors are used in STM32 devices to ensure a well-defined logical level in case of floating input signal. Depending on application requirements, an external pull-down can be used instead.
- Floating input. Signal level follows the external signal. When no external signal is present, the Schmitt trigger randomly toggles between the logical levels induced by the external noise. This increases the overall consumption.

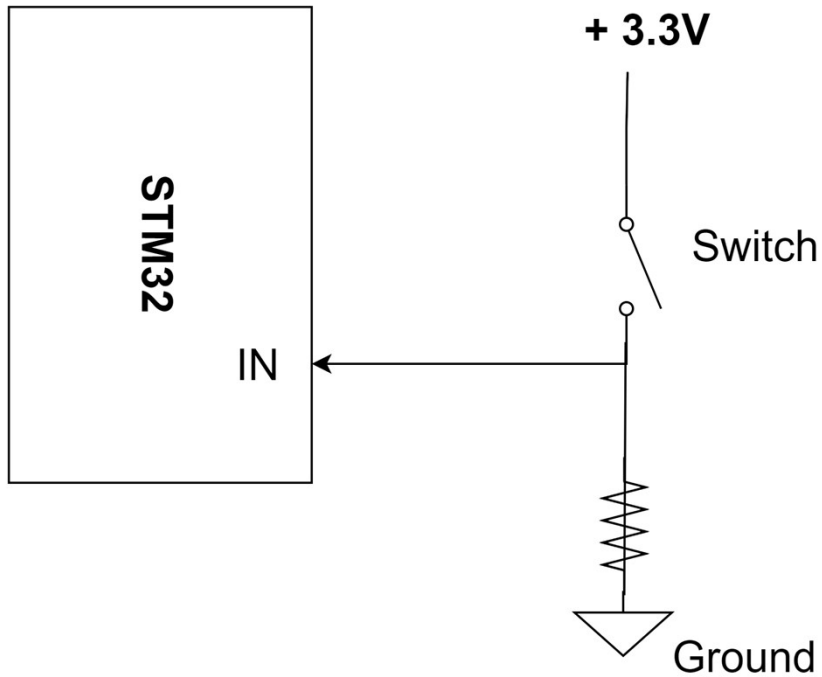
Input mode configuration

When the I/O Port is programmed as Input:

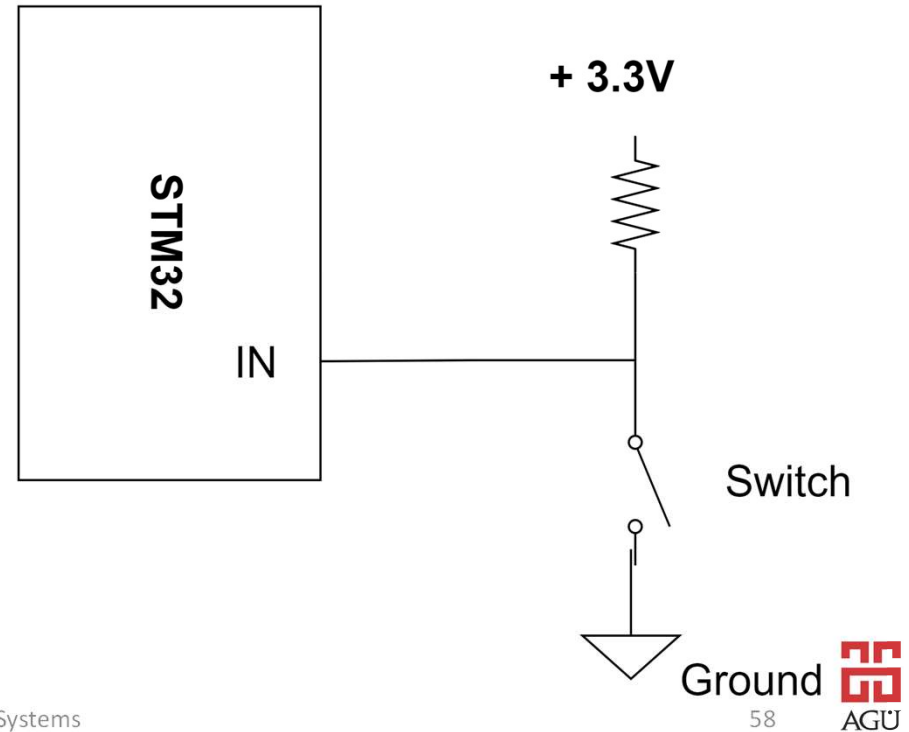
- The Output Buffer is disabled
- The Schmitt Trigger Input is activated
- The weak pull-up and pull-down resistors are activated or not depending on input configuration (pull-up, pull-down or floating):
- The data present on the I/O pin is sampled into the Input Data register every APB2 clock cycle
- A read access to the Input Data register obtains the I/O State.

Switches

Positive Logic (external)



Negative Logic (external)

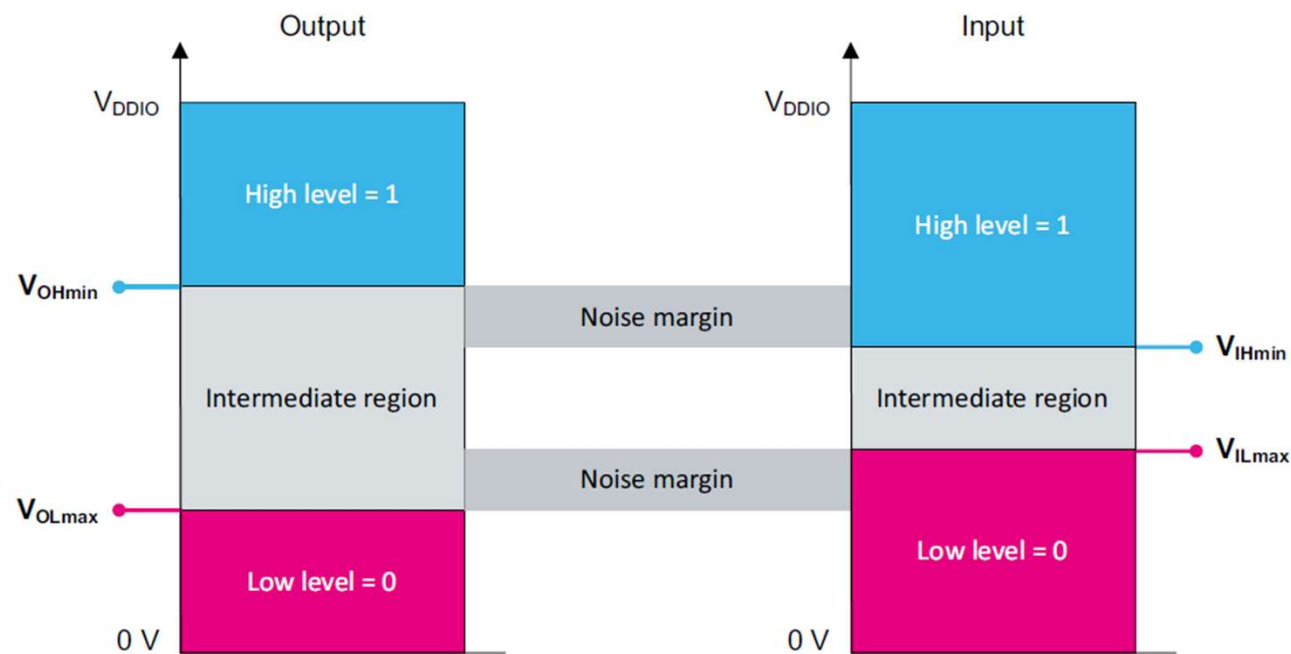


Let's do it... - Task (Blink with Button)

```
while (1)
{
    if( (GPIOB->IDR & BUTTON_PIN) == BUTTON_PIN)
    {
        GPIOB->ODR ^= LED_PIN; // Toggle LED pin
        delay(DELAY_CYCLES);
    }
}
```

Logical level compatibility

- For the CMOS technology, the input threshold voltages are relative to VDD as follows:
 - $V_{IHmin} \sim 2 / 3 V_{DD}$
 - $V_{ILmax} \sim 1 / 3 V_{DD}$
- For the TTL technology, the levels are fixed and equal to ∞
 - $V_{IHmin} = 2V$
 - $V_{ILmax} = 0.8 V$.



MSv46

Three-volt tolerant and five-volt tolerant

Electrical characteristics defines GPIO as three-volt tolerant, five-volt tolerant, and also three-volt capable. Tolerance represents the voltage value which can be accepted by the GPIO. Capability represents the voltage value which can be output by the GPIO.

Three-volt tolerant GPIO (TT)

- Input voltage on three-volt tolerant cannot exceed $VDD + 0.3\text{ V}$.
- If some analog input function is enabled on the GPIO (I/O structure TT_a with ADC input active, COMP input, OPAMP input), then the maximum operating voltage on pin cannot exceed $\min(VDDA, VREF+) + 0.3\text{ V}$.

Five-volt tolerant GPIO (FT)

- These GPIOs are actually tolerant to $VDD + 3.6\text{ V}$. (Regardless of the supply voltage, V_{IN} cannot exceed 5.5 V .)
- When $VDD = 0\text{ V}$, the input voltage on the GPIO cannot exceed 3.6 V .
- In case of a multi-supplied and multiplexed GPIO (VDD , $VDDUSB$, $VLCD$, $VDDA$), the GPIO is tolerant to 3.6 V augmented of the minimum supply voltage among VDD , $VDDUSB$, $VLCD$, and $VDDA$.
- However, a GPIO is five-volt tolerant only in input mode. When the output mode is enabled, the GPIO is no more five-volt tolerant.

Q&A

Any questions?