ELEC 3300 – Tutorial for LAB2

Department of Electronic and Computer Engineering
HKUST
by WU Chi Hang 

GPIO (General Purpose I/O)

- In STM32F103VET6, it got a total of 80 GPIO pins, it spans to 5 Ports. Namely Port A to Port E. Each port got 16 bits.
- 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) and
  - a 32-bit locking register (GPIOx_LCKR).

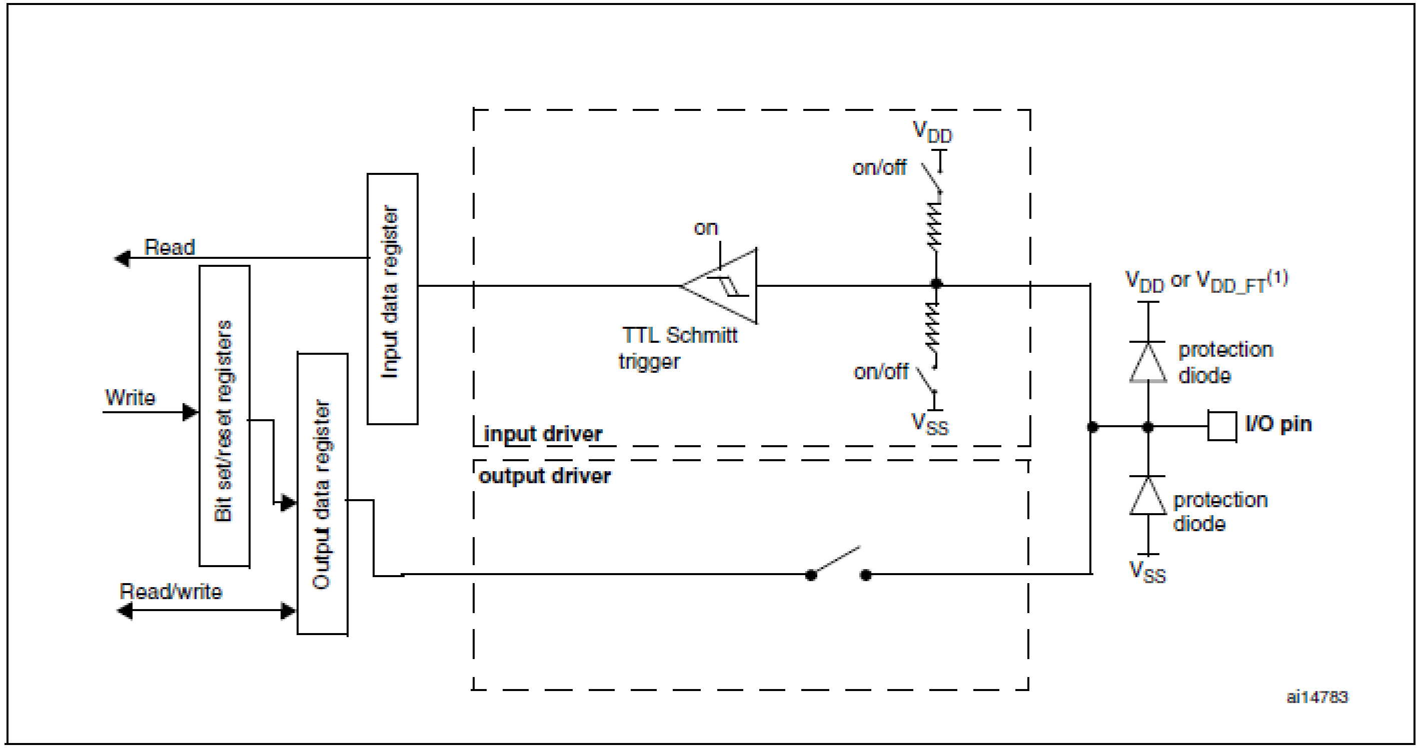
GPIO (General Purpose I/O)

- Also, each I/O pin can be individually configured by software into these states
  - Input floating
  - Input pull-up
  - Input-pull-down
  - Analog
  - Output open-drain
  - Output push-pull
  - Alternate function push-pull
  - Alternate function open-drain

GPIO (General Purpose I/O)

- Input Configuration

Figure 15. Input floating/pull up/pull down configurations

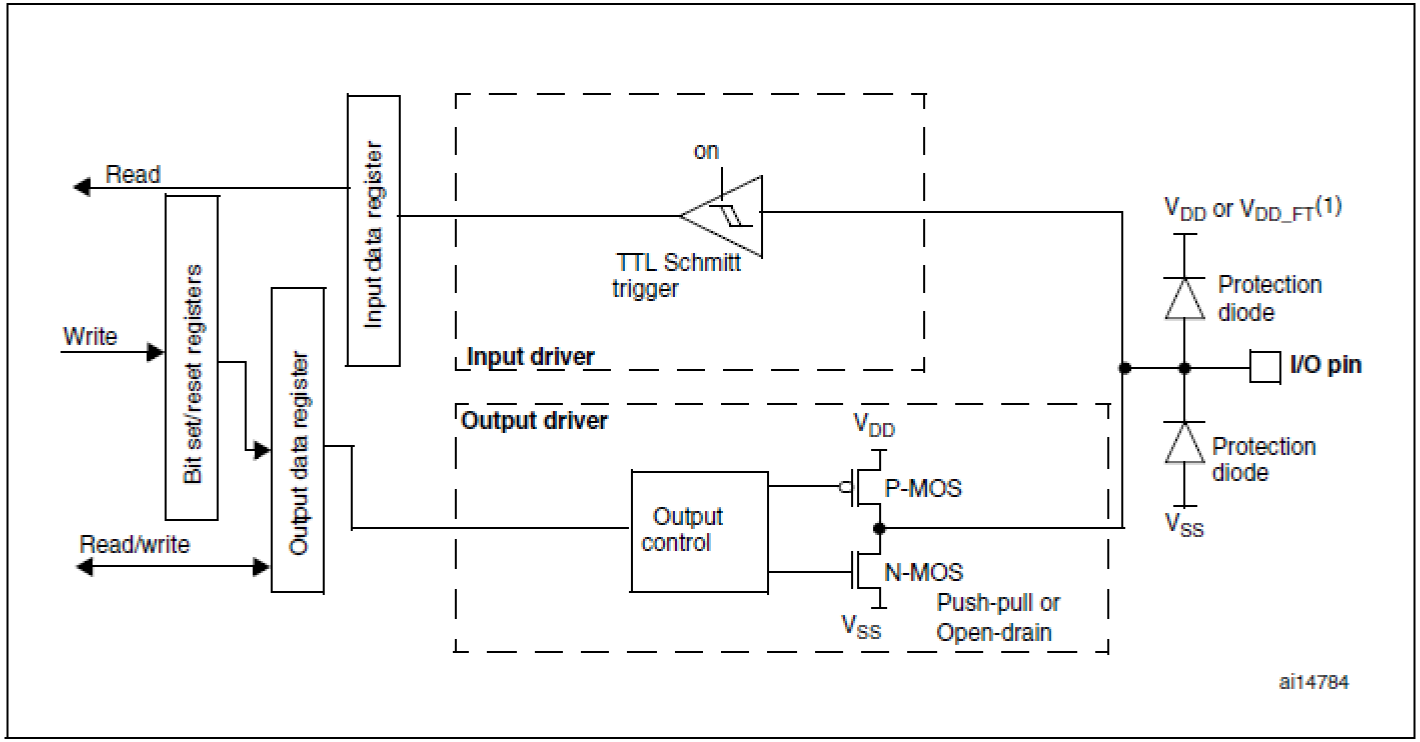


1. V_{DD_FT} is a potential specific to five-volt tolerant I/Os and different from V_{DD}.

GPIO (General Purpose I/O)

- Output Configuration

Figure 16. Output configuration



1. V_{DD_FT} is a potential specific to five-volt tolerant I/Os and different from V_{DD}.

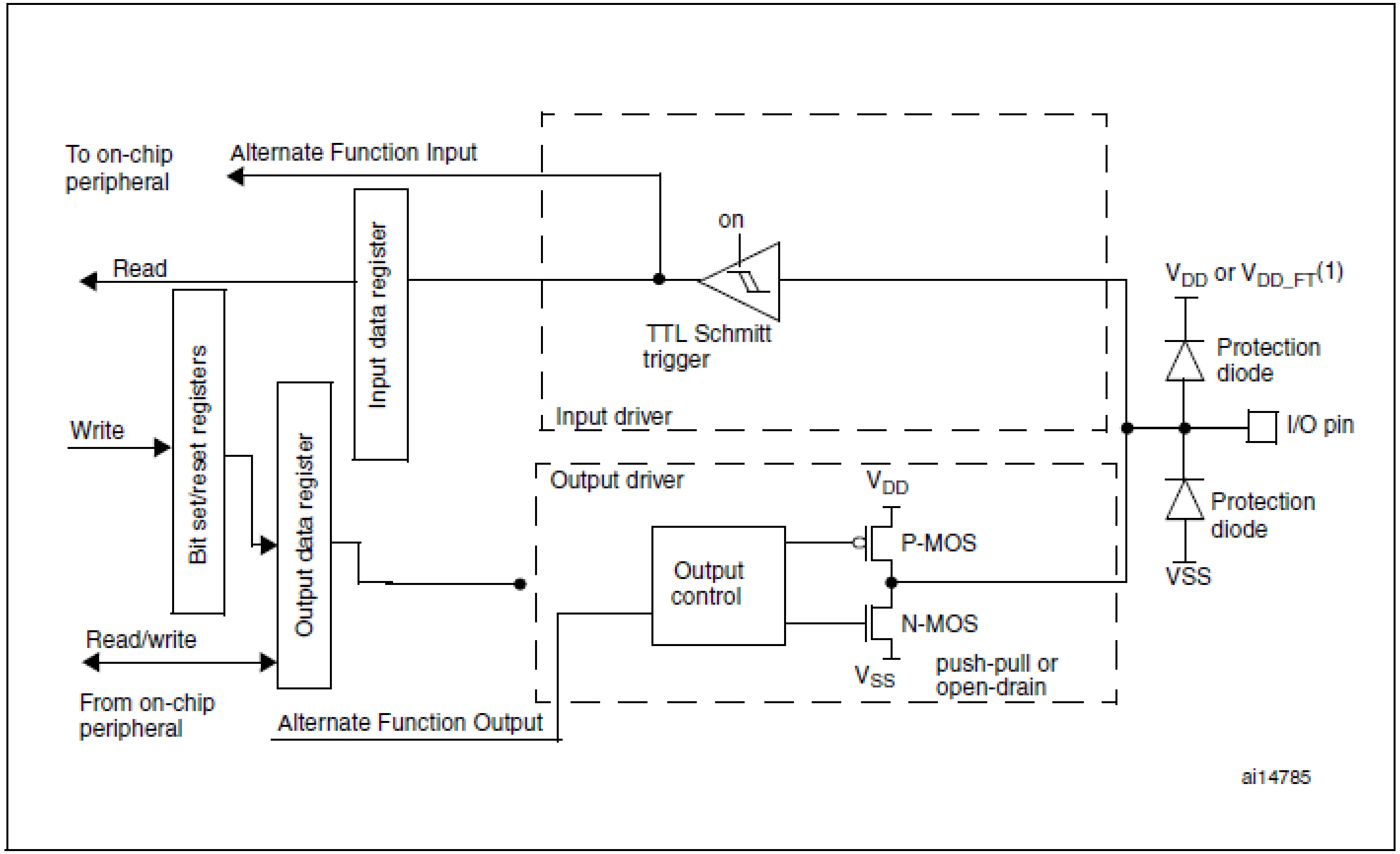
GPIO (General Purpose I/O)

- Output Configuration
  - 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

GPIO (General Purpose I/O)

- Alternate Function Configuration

Figure 17. Alternate function configuration

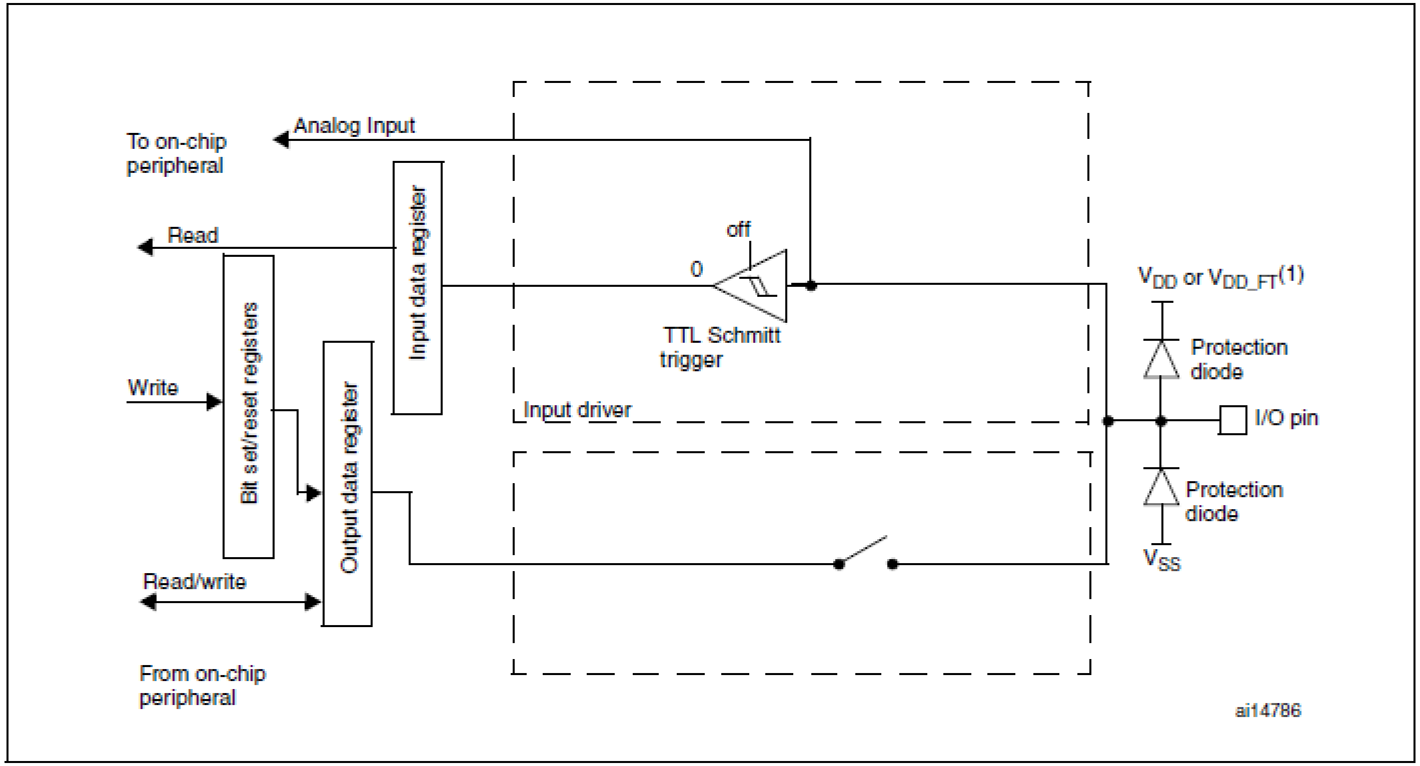


1. V_{DD_FT} is a potential specific to five-volt tolerant I/Os and different from V_{DD}.

GPIO (General Purpose I/O)

- Analogue Configuration

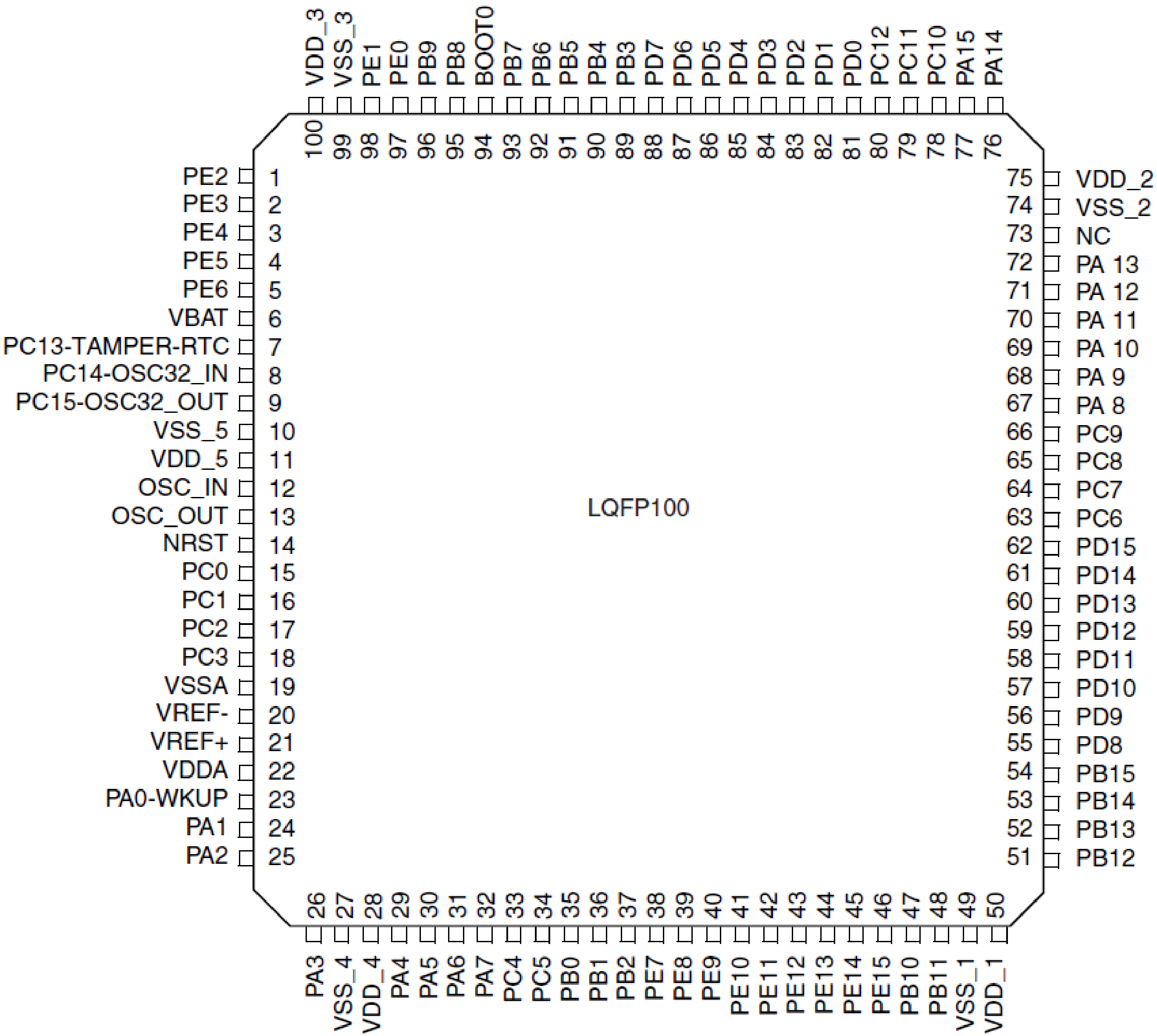
Figure 18. High impedance-analog configuration



GPIO (General Purpose I/O)

- If the Port is defined as output, the max output speed can be
  - 10MHz
  - 2MHz
  - 50MHz
- For the exact configuration of the bits, please refer to the Reference Manual Page 154 to 156.

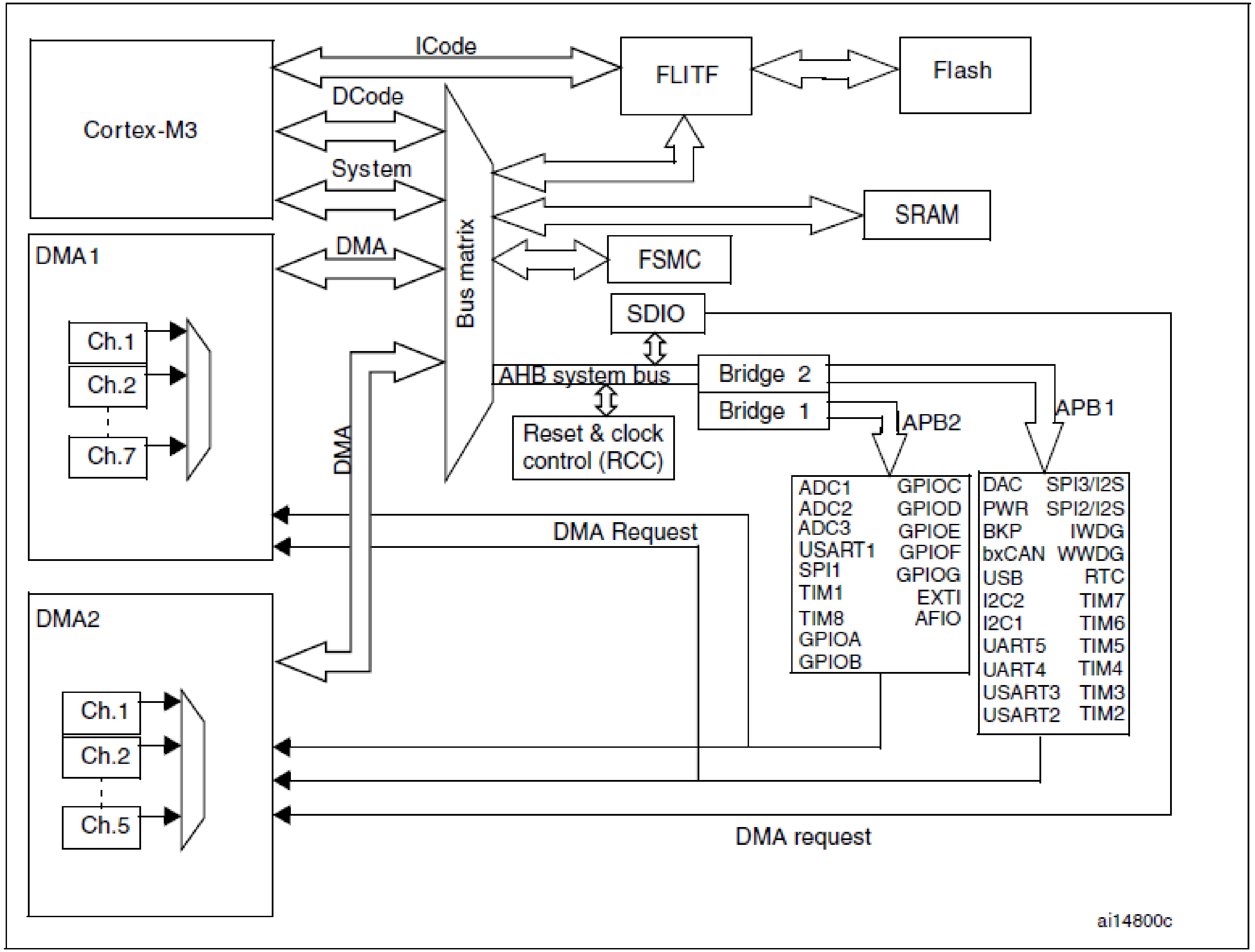
Pinout



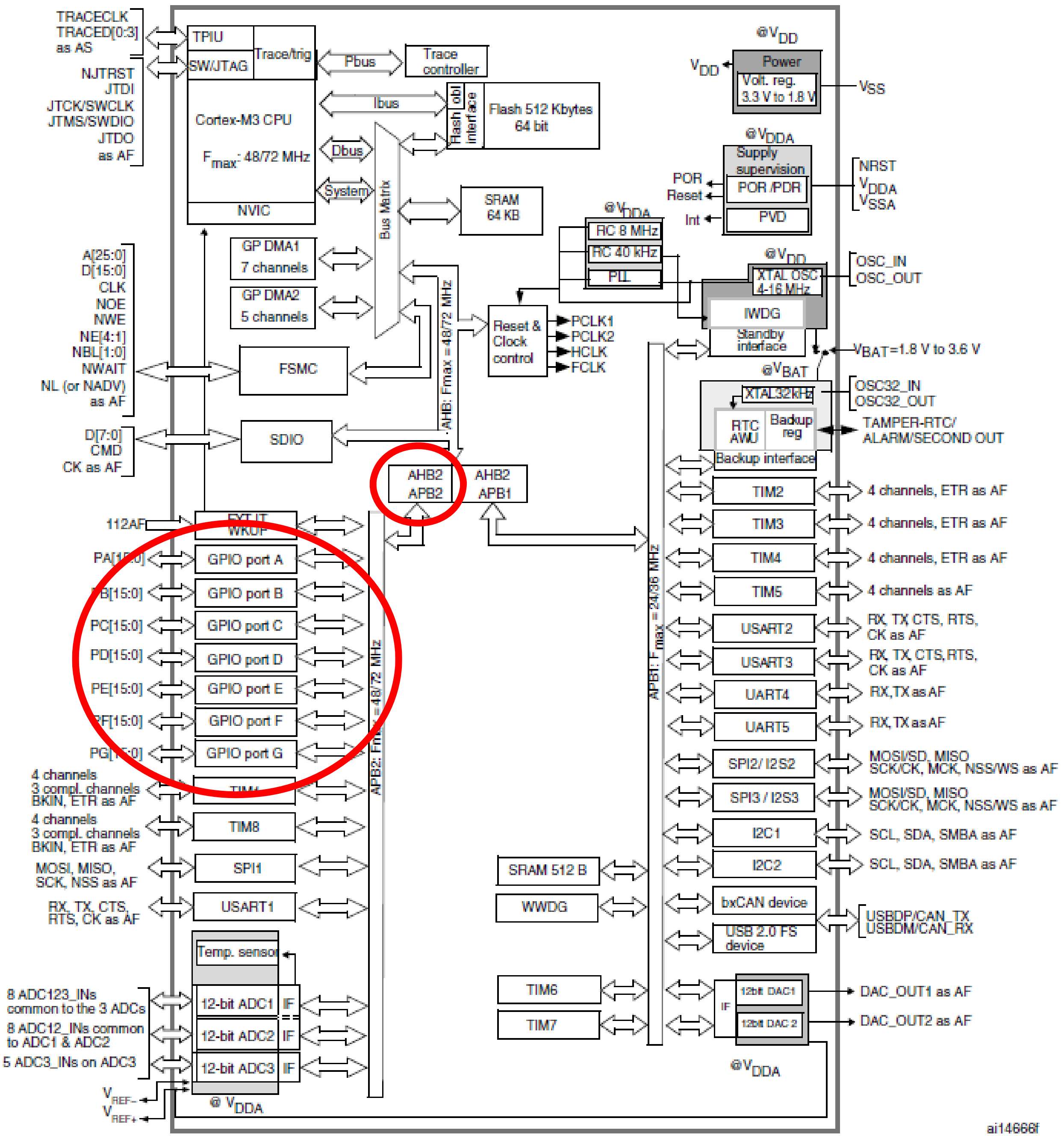
LQFP100
VDD_3 VSS_3 PE1 PE0 PB9 PB8 BOOT0 PB7 PB6 PB5 PB4 PB3 PD7 PD6 PD5 PD4 PD3 PD2 PD1 PD0 PC12 PC11 PC10 PA15 PA14
100 99 98 97 96 95 94 93 92 91 90 89 88 87 86 85 84 83 82 81 80 79 78 77 76
PE2 1
PE3 2
PE4 3
PE5 4
PE6 5
VBAT 6
PC13-TAMPER-RTC 7
PC14-OSC32_IN 8
PC15-OSC32_OUT 9
VSS_5 10
VDD_5 11
OSC_IN 12
OSC_OUT 13
NRST 14
PC0 15
PC1 16
PC2 17
PC3 18
VSSA 19
VREF- 20
VREF+ 21
VDDA 22
PA0-WKUP 23
PA1 24
PA2 25
75 VDD_2
74 VSS_2
73 NC
72 PA 13
71 PA 12
70 PA 11
69 PA 10
68 PA 9
67 PA 8
66 PC9
65 PC8
64 PC7
63 PC6
62 PD15
61 PD14
60 PD13
59 PD12
58 PD11
57 PD10
56 PD9
55 PD8
54 PB15
53 PB14
52 PB13
51 PB12
26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
PA3 VSS_4 VDD_4 PA4 PA5 PA6 PA7 PC4 PC5 PB0 PB1 PB2 PE7 PE8 PE9 PE10 PE11 PE12 PE13 PE14 PE15 PB10 PB11 VSS_1 VDD_1

STM32 System Architecture

Figure 1. System architecture

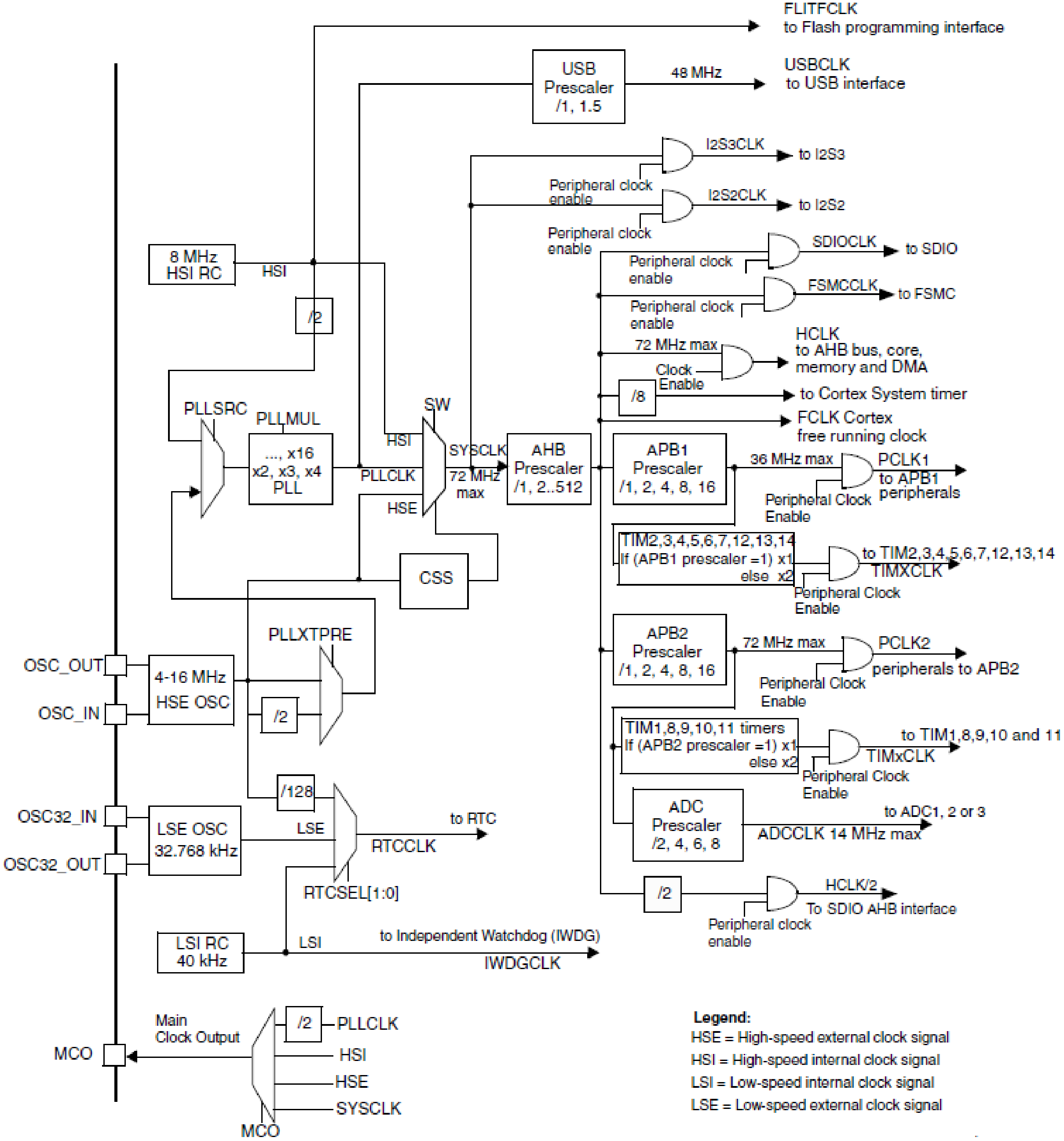


Block Diagram



AHB APB

- **SYSCLK is the System Clock Frequency (max 72 MHz)**
- **AHB is the System Bus**
- **APB is Peripherals Bus**
- The two AHB/APB bridges provide full synchronous connections between the AHB and the 2 APB buses.
- APB1 is limited to 36 MHz
- APB2 can operates at full speed (i.e. max 72 MHz)



MINI-V3 Development Board

- Starting from this LAB, we will use the following development board. We will use this device to emulate and test the function of the chip.

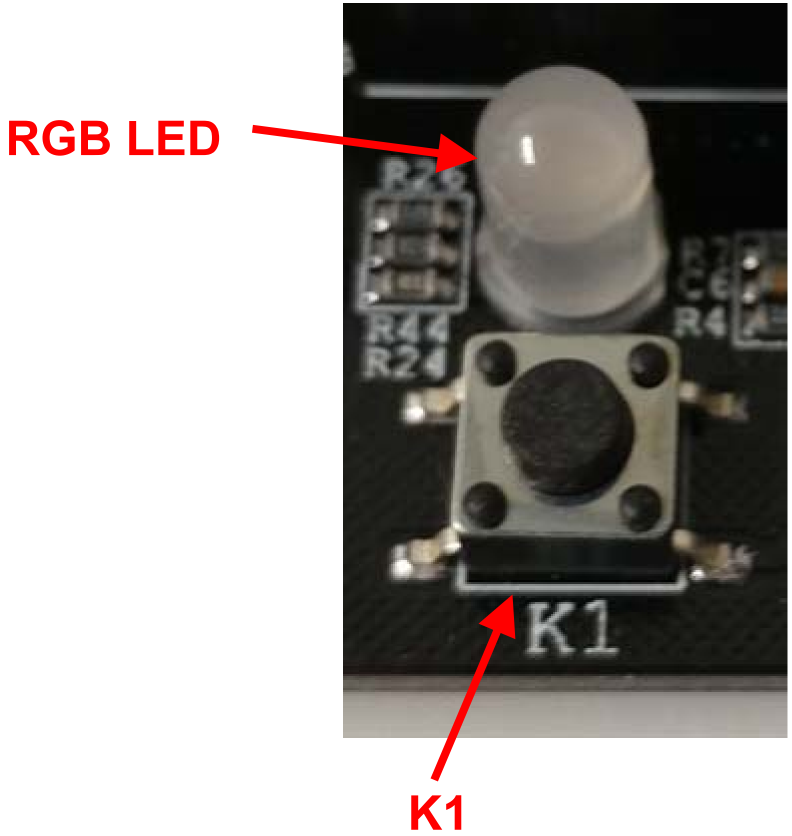
LAB2 – Task 1

- Follow the MUST Watch Videos on Canvas,
  - Video 1 - Part II : Blink
- Show that you can initialize the PB.5, the Red LED, to be output and blinking.

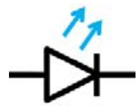
- Task 1 mark will be automatically granted if you finished Task 2 AND Task 3.

MINI-V3 Development Board

- Particularly for Task 2 and 3, we will concentrate on following parts.

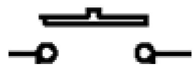


One External LED



One 220Ω Resistor

One External Key



LAB2 – Task 2 and 3

- When External Switch is pressed and released, it will toggle the RGB LED in order
- i.e.

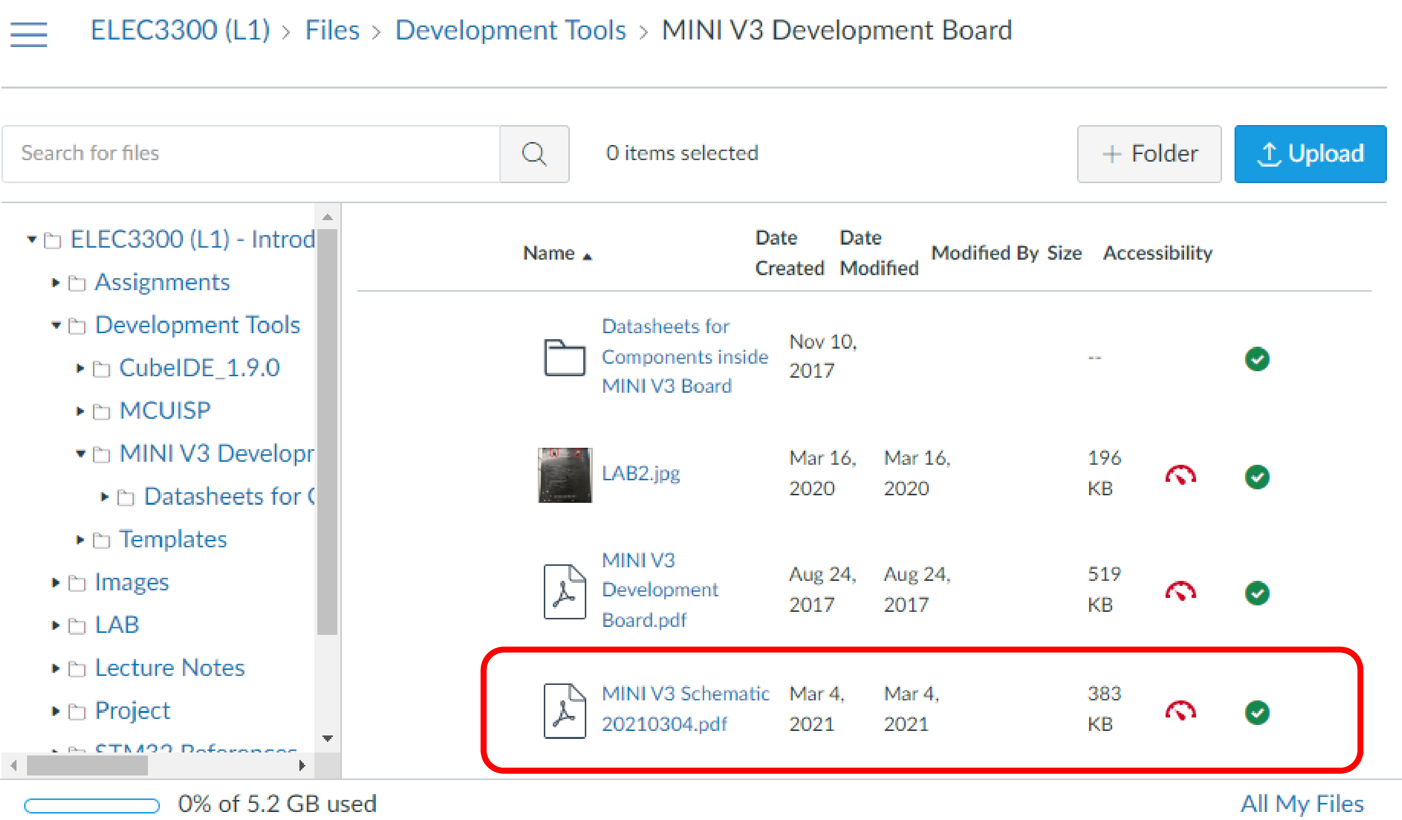
  EKey Pressed → R → EKey Pressed → G → EKey Pressed → B → EKey Pressed → R → EKey Pressed → G …

- When K1 is pressed and released, it will toggle the External LED
- i.e.

  K1 Pressed → External LED On → K1 Pressed → External LED Off → K1 Pressed → External LED On → K1 Pressed → External LED Off …

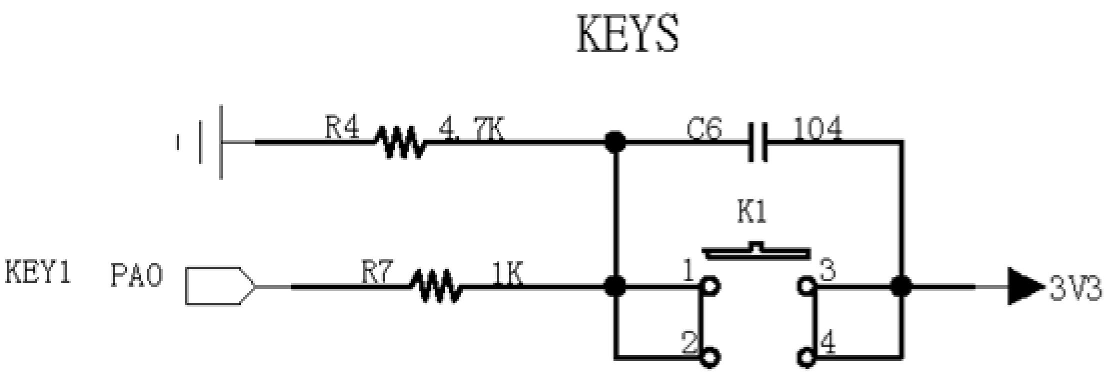
MINI-V3 Development Board

- The schematic for the board can be found on Canvas
- **IT IS ESSENTIAL FOR YOU TO CHECK THE SCHEMATIC**



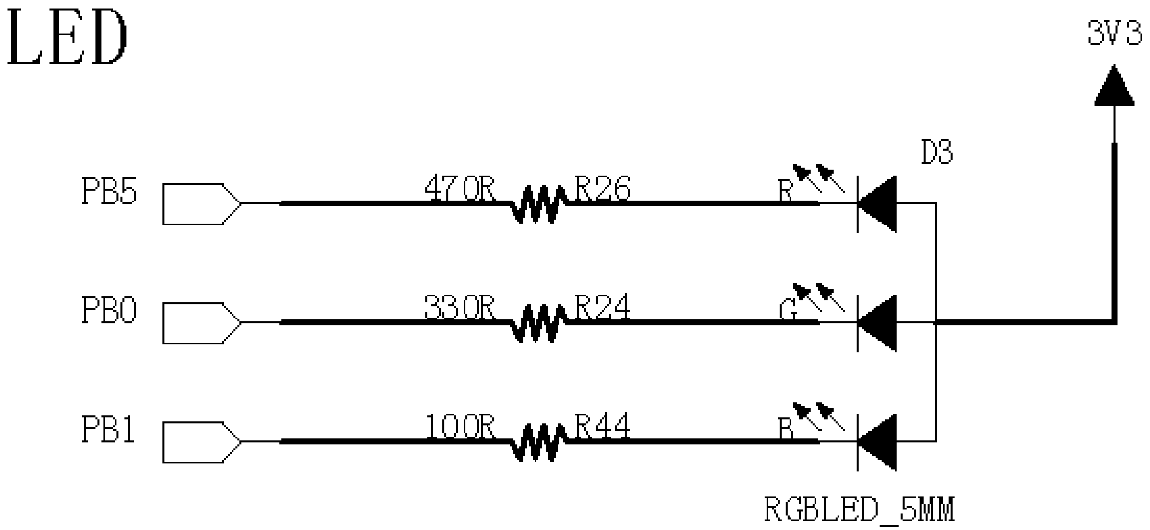
MINI-V3 Development Board

- The schematic for the K1 is here, please refer to the schematics available at the course website
  - Which Pin of STM32 does K1 connected to ?
  - What is the state of the corresponding Pin when the Key pressed/released ?



MINI-V3 Development Board

- The schematic for the RGB LED is shown here.
  - Which corresponding pin of STM32 does R/G/B LED connected to ?
  - In order to turn on the LED, what should be the state of these Pins ?



TIM3_CH3	LED_G	PB0
TIM3_CH4	LED_B	PB1
TIM3_CH2	LED_R	PB5

External LED and Key

- The connection of the external the LED and switch to the STM32 pin will depends on your student ID.
- Example, if your student ID is 21234567, you need to check accordingly to the Pin Set below

Pin Set	Actual Pin Number on STM32	Default Function of the pin on 100pin STM32F103VET6
A	67	PA8
B	56	PD9
C	45	PE14
D	34	PC5
E	23	PA0
F	12	OSC_IN
G	21	VREF+

External LED and Key

Pin Set	Actual Pin Number on STM32	Default Function of the pin on 100pin STM32F103VET6	I/O function ?	PIN CONNECTION IN MINI V3 Development board Can use for LED/Key ?
A	67	PA8	Yes	Used by Speaker in Development board. Needs modification if we want to use
B	56	PD9	Yes	Yes
C	45	PE14	Yes	Yes
D	34	PC5	Yes	Yes
E	23	PA0	Yes	NO, as used by K1 in this LAB
F	12	OSC_IN	No	
G	21	VREF+	No	

External LED and Key

- In the worse case, if none of the Pin Set you can use, then you can select any I/O to connect. However, you need to justify when you demo your LAB.

Pin Set	Actual Pin Number on STM32	Default Function of the pin on 100pin STM32F103VET6	I/O function ?	PIN CONNECTION IN MINI V3 Development board Can use for LED/Key ?
A	67	PA8	Yes	Used by Speaker in Development board. Needs modification if we want to use
B	56	PD9	Yes	Yes (Key)

External LED and Key

- We just decide the pin

Pin Set	Actual Pin Number on STM32	Default Function of the pin on 100pin STM32F103VET6	I/O function ?	PIN CONNECTION IN MINI V3 Development board Can use for LED/Key ?
A	67	PA8	Yes	Used by Speaker in Development board. Needs modification if we want to use
B	56	PD9	Yes	Yes (Key)

- You can decide the orientation of the LED and Key and that should match with your program and initialization inside the CubeIDE.
- HOWEVER, PLEASE REFER TO THE SCHEMATIC OF THE BOARD AND SEE IF ANY OTHER THINGS CONNECTED TO THAT PIN.
- As that might affect your decision of the orientation of the switch and LED.

LAB2 – Task 2 and 3

- Now, you need to generate another Project for Task 2 and Task 3
- Before you generate the Project, make sure you can answer yourself these questions..
  - How many I/O pin you need to use? What are they ?
  - What type should you initialized those pins to be ?
    - For RGB LED and K1, you have **NO CHOICE**, you need to **follow the schematic**.
    - For external LED and key, you are free to choose the orientation and the function only works if it matches to your program.
    - **PLEASE AGAIN CHECK THAT PIN ON THE MINI-V3 BOARD TO SEE IF IT IS BEING USED FOR OTHER FUNCTION**

Output – Using HAL Driver

- You can use the generated HAL Driver to help you to turn the bit on/off.
- All the generated driver files will be under Drivers\STM32F1xx_HAL_Driver\Inc , and \Src
- If you want to work with GPIO, then check GPIO Drivers
- By the same token, if you want to work with others function of HAL, check the corresponding Drivers.

Output – Using HAL Driver

- In stm32f1xx_hal_gpio.h, you can see the functions available.

```
/* IO operation functions *****************************************************/
GPIO_PinState HAL_GPIO_ReadPin(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin);
void HAL_GPIO_WritePin(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState);
void HAL_GPIO_TogglePin(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin);
HAL_StatusTypeDef HAL_GPIO_LockPin(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin);
void HAL_GPIO_EXTI_IRQHandler(uint16_t GPIO_Pin);
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin);
```

- Example, if you want to Set PC.0 to High, you can write

```
HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
```

- Example, if you want to Set PB.10 to Low, you can write

```
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_RESET);
```

- Always refer the .h file for the definitions

Task 2

- When External Switch is pressed and released, it will toggle the RGB LED in order
- i.e.

  EKey Pressed → Only Red LED On →

  EKey Pressed → Only Green LED On →

  EKey Pressed → Only Blue LED On →

  EKey Pressed → Only Red LED On →

  EKey Pressed → Only Green LED On →

  EKey Pressed → Only Blue LED On →

  …
- Question : How do I know if external key is pressed ?

Input – Using HAL Driver

- In HAL, there is a function to read a pin. It will return the State of the Pin

```
GPIO_PinState HAL_GPIO_ReadPin(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin);
```

- For example, if you want to Read the state of PA.10, you can write the following

```
dummy = HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_10);
```

in which dummy is a variable of GPIO_PinState.

Task 2 – Polling

What is polling ?

What is the advantage and disadvantage for using polling ?

For Task 2, we will use polling for external key.

```
while(1)
{
        Check if external key is pressed,
            if yes, follow sequence to toggle the RGB LED
}
```

Can we use the same method to check for K1 ?

LAB2 – Polling vs Interrupt

main()

{

1. HAL_Init();
2. SystemClock_Config();
3. MX_GPIO_Init();

while()

{

Check if external key is pressed,

if yes, follow sequence to toggle the RGB LED

}

}

Did the program check for K1 in the while loop ?

This is polling

Task 3 : If K1 is pressed during the while loop, External LED will toggle.

Interrupt

What is an interrupt ?

Let's try to understand the concept first…

You have many things to do in your everyday life

Consider your daily life as a while loop..

```
while(everyday)
{
        wakeup
        breakfast;
        lunch;
        tea;
        dinner;
        supper;
        sleep;
}
```

Everyone of you has a mobile phone..

Will you do a polling on your mobile phone ? (Y/N)

How do you know if someone calls you ?

What happen if your mobile phone is set to mute ?

Now look at our program…

LAB2 – Polling vs Interrupt

main()

{

1. HAL_Init();
2. SystemClock_Config();
3. MX_GPIO_Init();

K1 will only work if interrupt is enabled in the MX_GPIO_Init();

while()

{

Check if external key is pressed,

if yes, follow sequence to toggle the RGB LED

}

This is polling

}

Task 3 : If K1 is pressed during the while loop, External LED will toggle.

Interrupt

- You can consider interrupt is a procedure call.
- If you enabled the interrupt, the machine will go to do that procedure. That procedure is called Interrupt Service Routine (ISR).
- Note that once you enabled the interrupt, you DON’T need to do polling.
- If you use both polling and interrupt for the same event, you have a major conceptual error 🙄👎😠
- THINK: Do you need to continuously keep checking your phone to see if someone calls you if you set your phone to ring mode ? 🤔

Interrupt in STM32

In STM32, there is a NVIC (Nested Vectored Interrupt Controller) to control up to 81 interrupts and with 16 level of priority.

Table 63 from the Reference Manual have the full list of the interrupt order.

Table 63. Vector table for other STM32F10xxx devices

Position	Priority	Type of priority	Acronym	Description	Address
	-	-	-	Reserved	0x0000_0000
	-3	fixed	Reset	Reset	0x0000_0004
	-2	fixed	NMI	Non maskable interrupt. The RCC Clock Security System (CSS) is linked to the NMI vector.	0x0000_0008
	-1	fixed	HardFault	All class of fault	0x0000_000C
	0	settable	MemManage	Memory management	0x0000_0010
	1	settable	BusFault	Prefetch fault, memory access fault	0x0000_0014
	2	settable	UsageFault	Undefined instruction or illegal state	0x0000_0018
	-	-	-	Reserved	0x0000_001C - 0x0000_002B
	3	settable	SVCall	System service call via SWI instruction	0x0000_002C
	4	settable	Debug Monitor	Debug Monitor	0x0000_0030
	-	-	-	Reserved	0x0000_0034
	5	settable	PendSV	Pendable request for system service	0x0000_0038
	6	settable	SysTick	System tick timer	0x0000_003C
0	7	settable	WWDG	Window watchdog interrupt	0x0000_0040
1	8	settable	PVD	PVD through EXTI Line detection interrupt	0x0000_0044
2	9	settable	TAMPER	Tamper interrupt	0x0000_0048
3	10	settable	RTC	RTC global interrupt	0x0000_004C
4	11	settable	FLASH	Flash global interrupt	0x0000_0050
5	12	settable	RCC	RCC global interrupt	0x0000_0054
6	13	settable	EXTI0	EXTI Line0 interrupt	0x0000_0058
7	14	settable	EXTI1	EXTI Line1 interrupt	0x0000_005C
8	15	settable	EXTI2	EXTI Line2 interrupt	0x0000_0060
9	16	settable	EXTI3	EXTI Line3 interrupt	0x0000_0064
10	17	settable	EXTI4	EXTI Line4 interrupt	0x0000_0068
11	18	settable	DMA1_Channel1	DMA1 Channel1 global interrupt	0x0000_006C
12	19	settable	DMA1_Channel2	DMA1 Channel2 global interrupt	0x0000_0070
13	20	settable	DMA1_Channel3	DMA1 Channel3 global interrupt	0x0000_0074
14	21	settable	DMA1_Channel4	DMA1 Channel4 global interrupt	0x0000_0078
15	22	settable	DMA1_Channel5	DMA1 Channel5 global interrupt	0x0000_007C
16	23	settable	DMA1_Channel6	DMA1 Channel6 global interrupt	0x0000_0080

Table 63. Vector table for other STM32F10xxx devices (continued)

Position	Priority	Type of priority	Acronym	Description	Address
17	24	settable	DMA1_Channel7	DMA1 Channel7 global interrupt	0x0000_0084
18	25	settable	ADC1_2	ADC1 and ADC2 global interrupt	0x0000_0088
19	26	settable	USB_HP_CAN_TX	USB High Priority or CAN TX interrupts	0x0000_008C
20	27	settable	USB_LP_CAN_RX0	USB Low Priority or CAN RX0 interrupts	0x0000_0090
21	28	settable	CAN_RX1	CAN RX1 interrupt	0x0000_0094
22	29	settable	CAN_SCE	CAN SCE interrupt	0x0000_0098
23	30	settable	EXTI9_5	EXTI Line[9:5] interrupts	0x0000_009C
24	31	settable	TIM1_BRK	TIM1 Break interrupt	0x0000_00A0
25	32	settable	TIM1_UP	TIM1 Update interrupt	0x0000_00A4
26	33	settable	TIM1_TRG_COM	TIM1 Trigger and Commutation interrupts	0x0000_00A8
27	34	settable	TIM1_CC	TIM1 Capture Compare interrupt	0x0000_00AC
28	35	settable	TIM2	TIM2 global interrupt	0x0000_00B0
29	36	settable	TIM3	TIM3 global interrupt	0x0000_00B4
30	37	settable	TIM4	TIM4 global interrupt	0x0000_00B8
31	38	settable	I2C1_EV	I^2C1 event interrupt	0x0000_00BC
32	39	settable	I2C1_ER	I^2C1 error interrupt	0x0000_00C0
33	40	settable	I2C2_EV	I^2C2 event interrupt	0x0000_00C4
34	41	settable	I2C2_ER	I^2C2 error interrupt	0x0000_00C8
35	42	settable	SPI1	SPI1 global interrupt	0x0000_00CC
36	43	settable	SPI2	SPI2 global interrupt	0x0000_00D0
37	44	settable	USART1	USART1 global interrupt	0x0000_00D4
38	45	settable	USART2	USART2 global interrupt	0x0000_00D8
39	46	settable	USART3	USART3 global interrupt	0x0000_00DC
40	47	settable	EXTI15_10	EXTI Line[15:10] interrupts	0x0000_00E0
41	48	settable	RTCAlarm	RTC alarm through EXTI line interrupt	0x0000_00E4
42	49	settable	USBWakeup	USB wakeup from suspend through EXTI line interrupt	0x0000_00E8
43	50	settable	TIM8_BRK	TIM8 Break interrupt	0x0000_00EC
44	51	settable	TIM8_UP	TIM8 Update interrupt	0x0000_00F0
45	52	settable	TIM8_TRG_COM	TIM8 Trigger and Commutation interrupts	0x0000_00F4
46	53	settable	TIM8_CC	TIM8 Capture Compare interrupt	0x0000_00F8

Table 63. Vector table for other STM32F10xxx devices (continued)

Position	Priority	Type of priority	Acronym	Description	Address
47	54	settable	ADC3	ADC3 global interrupt	0x0000_00FC
48	55	settable	FSMC	FSMC global interrupt	0x0000_0100
49	56	settable	SDIO	SDIO global interrupt	0x0000_0104
50	57	settable	TIM5	TIM5 global interrupt	0x0000_0108
51	58	settable	SPI3	SPI3 global interrupt	0x0000_010C
52	59	settable	UART4	UART4 global interrupt	0x0000_0110
53	60	settable	UART5	UART5 global interrupt	0x0000_0114
54	61	settable	TIM6	TIM6 global interrupt	0x0000_0118
55	62	settable	TIM7	TIM7 global interrupt	0x0000_011C
56	63	settable	DMA2_Channel1	DMA2 Channel1 global interrupt	0x0000_0120
57	64	settable	DMA2_Channel2	DMA2 Channel2 global interrupt	0x0000_0124
58	65	settable	DMA2_Channel3	DMA2 Channel3 global interrupt	0x0000_0128
59	66	settable	DMA2_Channel4_5	DMA2 Channel4 and DMA2 Channel5 global interrupts	0x0000_012C

Interrupt in STM32

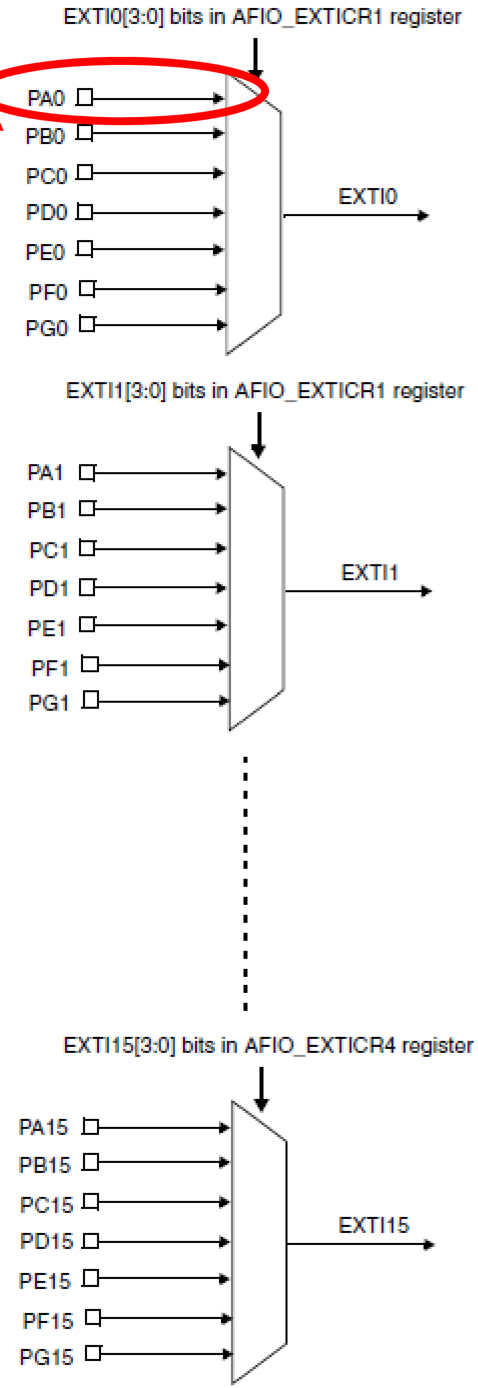
In this LAB, we are using the External Interrupt (EXTI).

The 80 GPIOs are connected to the 16 external Interrupt/event lines shown on the right.

In this LAB, K1 is PA.0, we will need to enable the EXTI0 interrupt.

Refer to page 185 of the Reference Manual, the definition of the **AFIO_EXTICR1 to 4** registers, you can only allow to select one PIN out of 7 pins as an input to the interrupt.

For details of EXTI, please refer to page 198, section 10.2 of the Reference Manual.



Interrupt in STM32

- Note : Refer to page 185 of the Reference Manual, the reset values of **AFIO_EXTICR1 to 4** registers are all 0x00, which maps PA pins to the default interrupt pin for EXTI0 to EXTI15.
- If you want to remap the pins using other Ports, you need to enable the AFIO clock, as stated from page 201 of the Reference Manual
  - To configure the AFIO_EXTICRx for the mapping of external interrupt/event lines onto GPIOs, the AFIO clock should first be enabled.

Interrupt in STM32

As there are total up to 81 interrupts in STM32. You can set the priority of the interrupts.

This priority can be set in the NVIC, you can refer to page 126, section 4.3.7 of the Cortex_M3_Programming Manual. You can set up to 16 levels of priority.

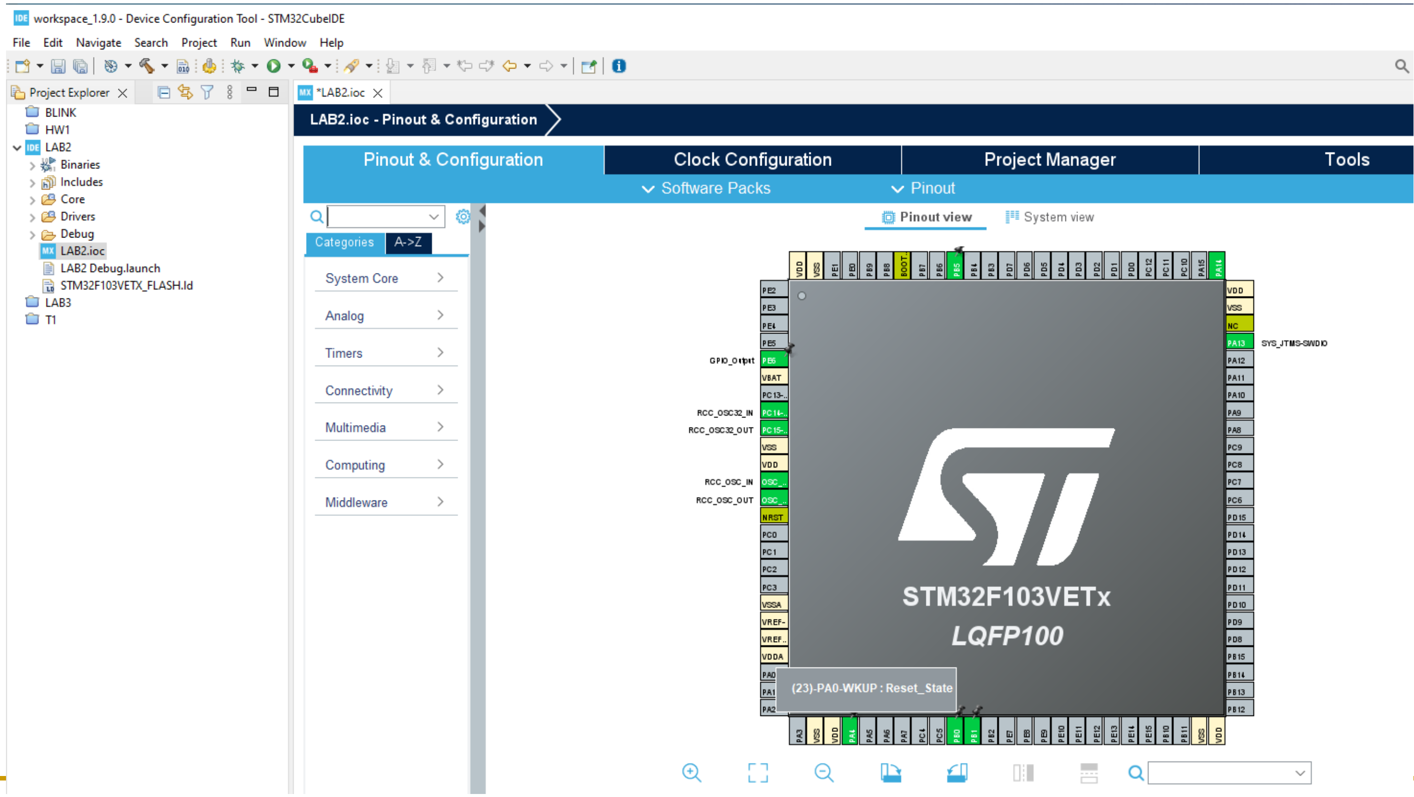
From Section 2.3.6 of the Programming Manual :

If multiple pending interrupts have the same group priority, the subpriority field determines the order in which they are processed. If multiple pending interrupts have the same group priority and subpriority, the interrupt with the lowest IRQ number is processed first.

For details of the NVIC, please refer to Page 119, section 4.3 of the Cortex-M3 Programming Manual

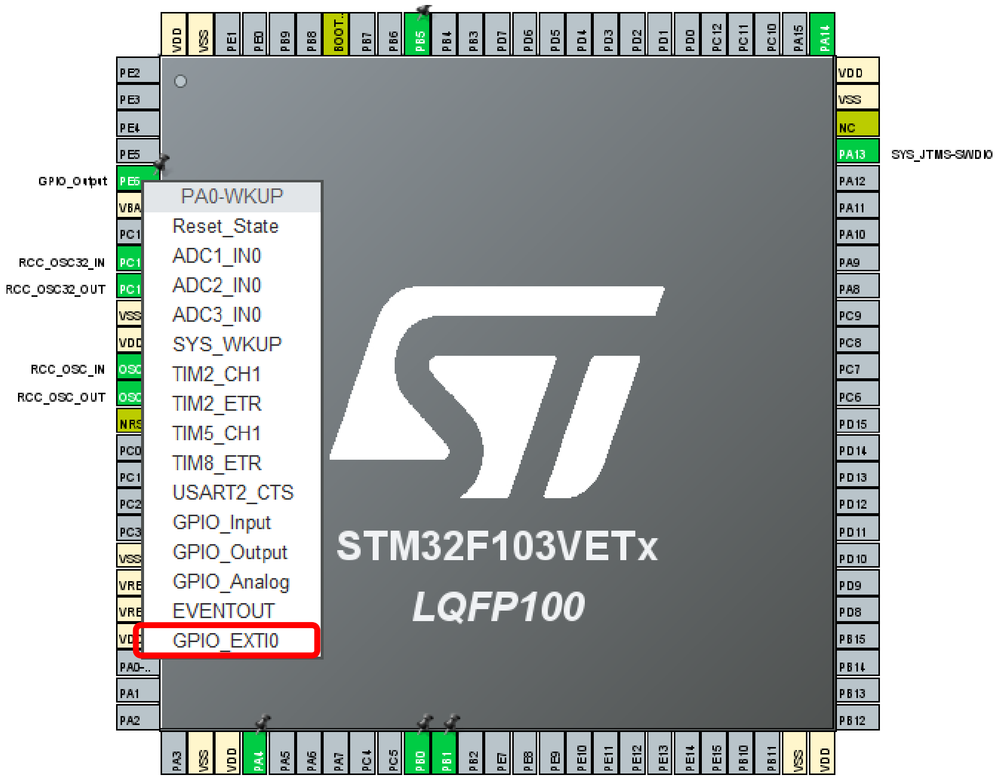
Configure the GPIO to Interrupt

- In CubeIDE, before you generate the code, you need to enable the GPIO to be interrupt. Find PA.0



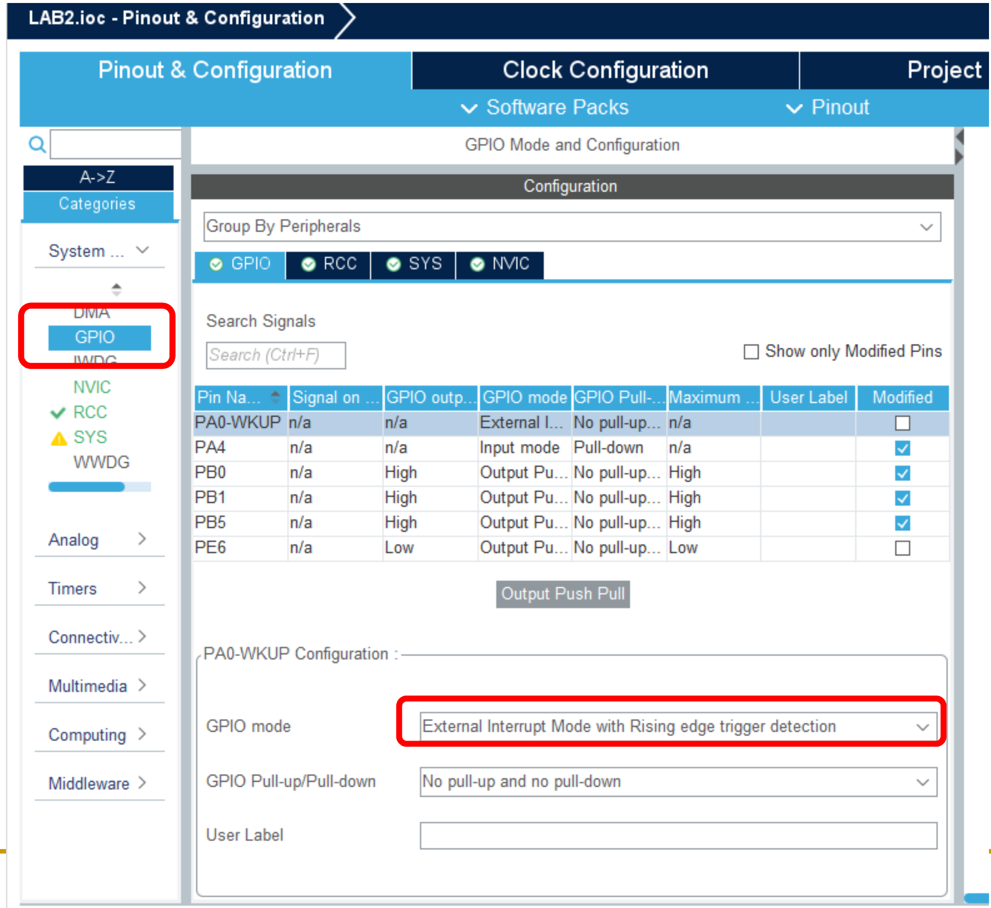
Configure the GPIO to Interrupt

- Select GPIO_EXTI0



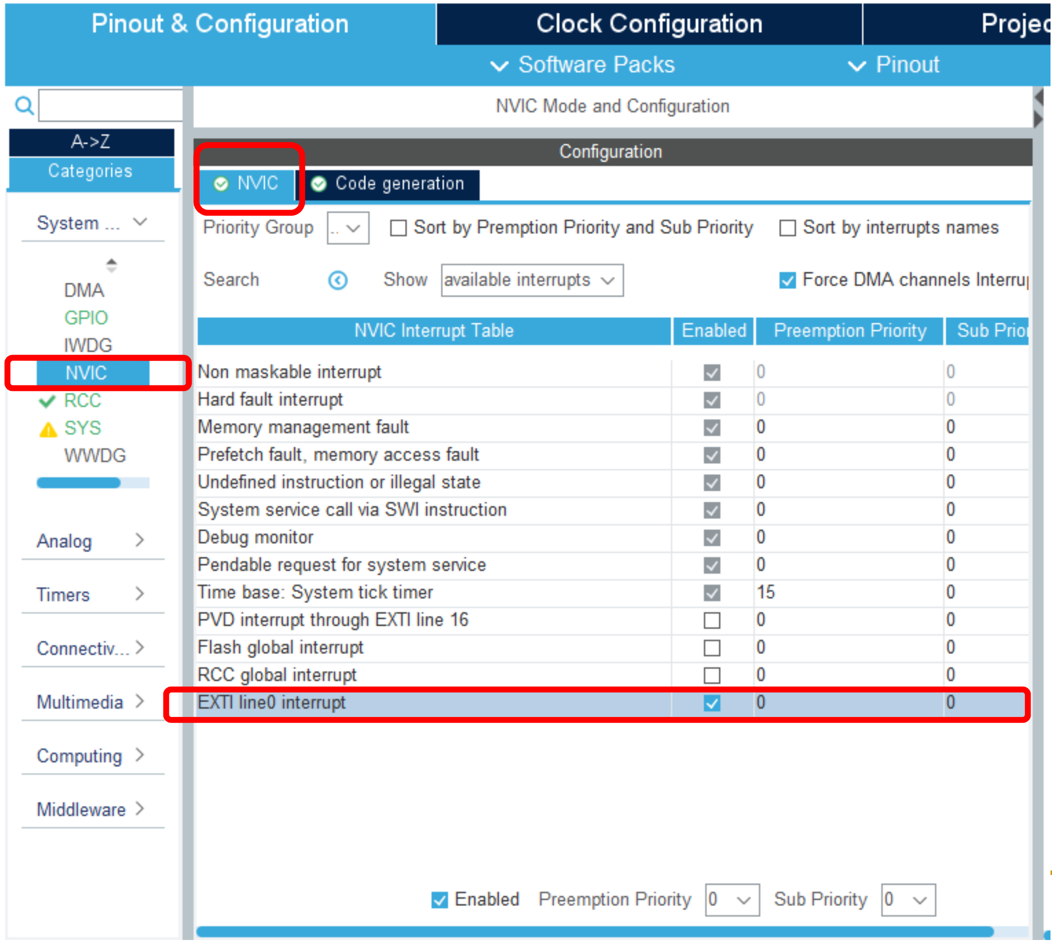
Configure the GPIO to Interrupt

- GPIO mode → External Interrupt Mode with Rising edge trigger detection



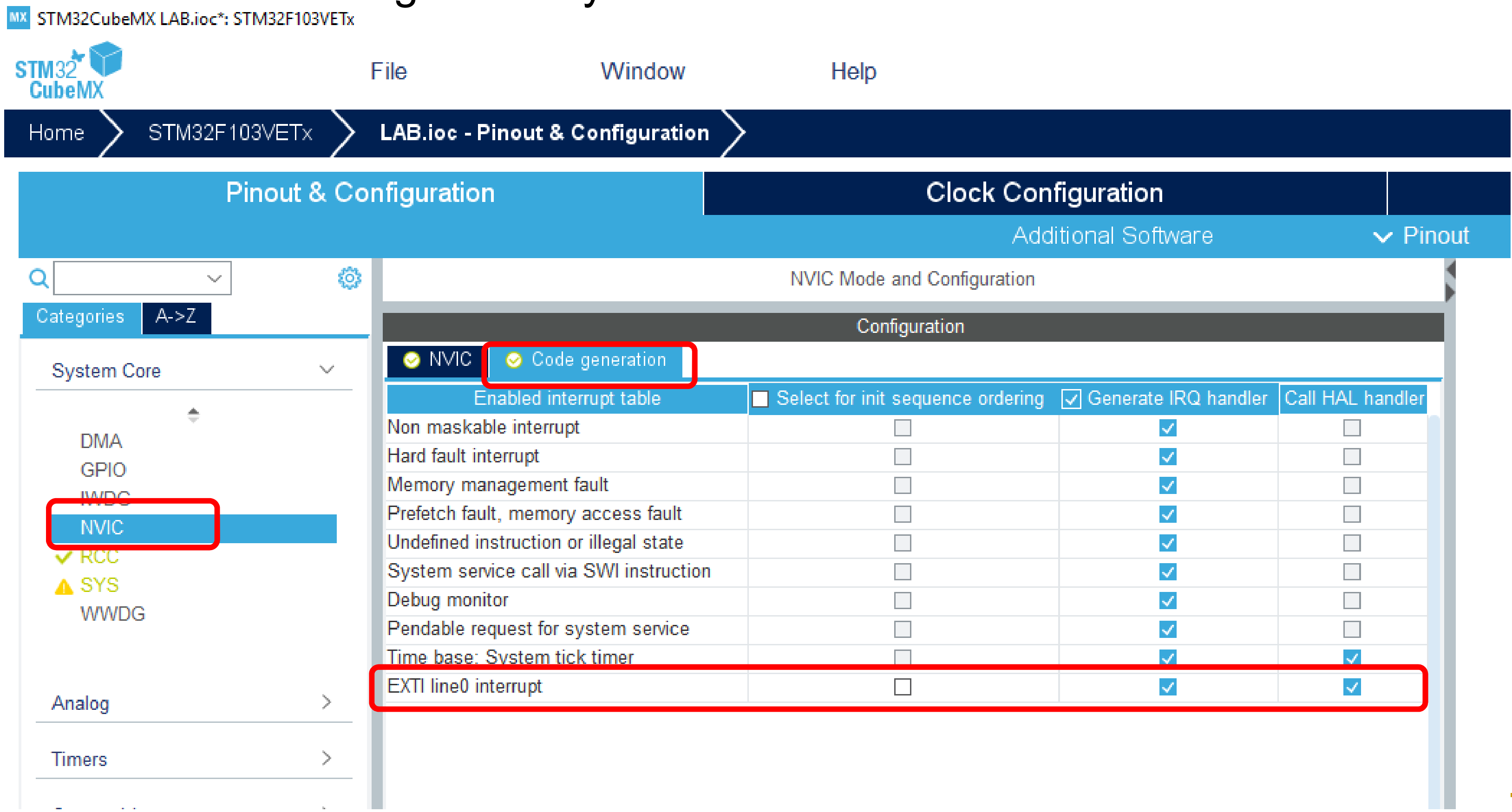
Configure the NVIC

- In NVIC, Enable the EXTI line0 interrupt.



Configure the NVIC

- In Code generation, EXTI line0 interrupt, check the box Call HAL handler. You can then generate your code.



Interrupt Service Routine and Interrupt Vector

- After you connect the GPIO to the EXTI0, you need to write a procedure that will run upon the reception of the interrupt. This procedure is called an Interrupt Service Routine (ISR).
- The starting address of the ISR is called the Interrupt Vector. Next page shows some of the address of the Interrupt Vector
- You will need to modify the code of the ISR for EXT0 in this LAB.

Interrupt Service Routine and Interrupt Vector

Table 62. Vector table for XL-density devices (continued)

Position	Priority	Type of priority	Acronym	Description	Address
1	8	settable	PVD	PVD through EXTI Line detection interrupt	0x0000_0044
2	9	settable	TAMPER	Tamper interrupt	0x0000_0048
3	10	settable	RTC	RTC global interrupt	0x0000_004C
4	11	settable	FLASH	Flash global interrupt	0x0000_0050
5	12	settable	RCC	RCC global interrupt	0x0000_0054
6	13	settable	EXTI0	EXTI Line0 interrupt	0x0000_0058
7	14	settable	EXTI1	EXTI Line1 interrupt	0x0000_005C
8	15	settable	EXTI2	EXTI Line2 interrupt	0x0000_0060
9	16	settable	EXTI3	EXTI Line3 interrupt	0x0000_0064
10	17	settable	EXTI4	EXTI Line4 interrupt	0x0000_0068
11	18	settable	DMA1_Channel1	DMA1 Channel1 global interrupt	0x0000_006C
12	19	settable	DMA1_Channel2	DMA1 Channel2 global interrupt	0x0000_0070

- Please refer to Reference Manual Page 193 for the whole list of Interrupt Vector.

Task 3 : Modifying the ISR

- Originally, the implementation of ISR are contained in the file stm32f10x_it.c
- Open the file and locate the following

```
/**
  * @brief This function handles EXTI line0 interrupt.
  */
void EXTI0_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI0_IRQn 0 */
  /* USER CODE END EXTI0_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_0);
  /* USER CODE BEGIN EXTI0_IRQn 1 */

  /* USER CODE END EXTI0_IRQn 1 */
}
```

HAL ISR

- Because we used HAL_Driver, it pointed to the HAL Handler

Task 3 : Modifying the ISR, 3 Steps

- In this LAB, we will create our own Handler instead of using the HAL one, so, please modify the handler as shown on next page.
- 3 Steps
  1. Comment the original HAL ISR
  2. Copy the code in Blue
  3. Add code in Green Area to toggle the external LED

Task 3 : Modifying the ISR, 3 Steps

```
void EXTI0_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI0_IRQn 0 */
  if (__HAL_GPIO_EXTI_GET_IT(GPIO_PIN_0) != RESET)
  {
```

Step 3. Add code here to toggle the external LED

```
    __HAL_GPIO_EXTI_CLEAR_IT(GPIO_PIN_0);
    HAL_GPIO_EXTI_Callback(GPIO_PIN_0);
  }
  /* USER CODE END EXTI0_IRQn 0 */

//  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_0);
  /* USER CODE BEGIN EXTI0_IRQn 1 */
  /* USER CODE END EXTI0_IRQn 1 */
}
```

Step 2. Copy the code in Blue

Step 1. Comment the original HAL ISR

LAB2 – Task 2 and 3, overall picture

main()

{

1. HAL_Init();
2. SystemClock_Config();
3. MX_GPIO_Init();

while() **Polling**

{

Check if EKey is pressed,
if yes, follow sequence
to toggle the RGB LED

}

}

Interrupt is enabled here

When K1 Pressed

EXT0 ISR

Toggle
External LED

After finishing LAB2 you are expected to…

1. Understand how to use CubeIDE to build a Project.
2. Read the schematics of the development board.
3. Correctly connect simple devices (i.e. switch/LED) externally as inputs and outputs to the development board.
4. Understand the importance of correct settings for an input and output (for both inside development board or external to the development board).
5. Understand the importance of matching between your setting and programming to input and output.
6. Understand the difference between polling and interrupt for getting input from external sources.
7. Program both polling and interrupt methods in STM32.
8. Run the Program in Debug Mode (like HW1) to debug program if needed.

END