Fundamentals of Embedded and Real Time Systems

MODULE 10

TAMER AWAD

Review Module 09

Module 10

In-Class Project

- User Requirements
- Problem Breakdown
- Work thru the steps

Programming with Embedded & Real-Time Operating Systems:

- Overview by ***Nick Strathy***
- Q&A

User Button Changing LED States

- User Requirements
- Problem Breakdown

User Requirements

1. Pressing the button changes the state of the LED.
2. System starts with LED in OFF state.
3. First button push --> LED turns solid ON.
4. Second button push --> LED blinks at a slow rate (every one or two seconds).
5. Third button push --> LED blinks at a faster rate (every 0.25 or 0.5 seconds).
6. Fourth button push --> LED dims and brightens at a slow rate.
7. Fifth button push --> LED dims and brightens at a faster rate.
8. Sixth button push --> LED turns off.

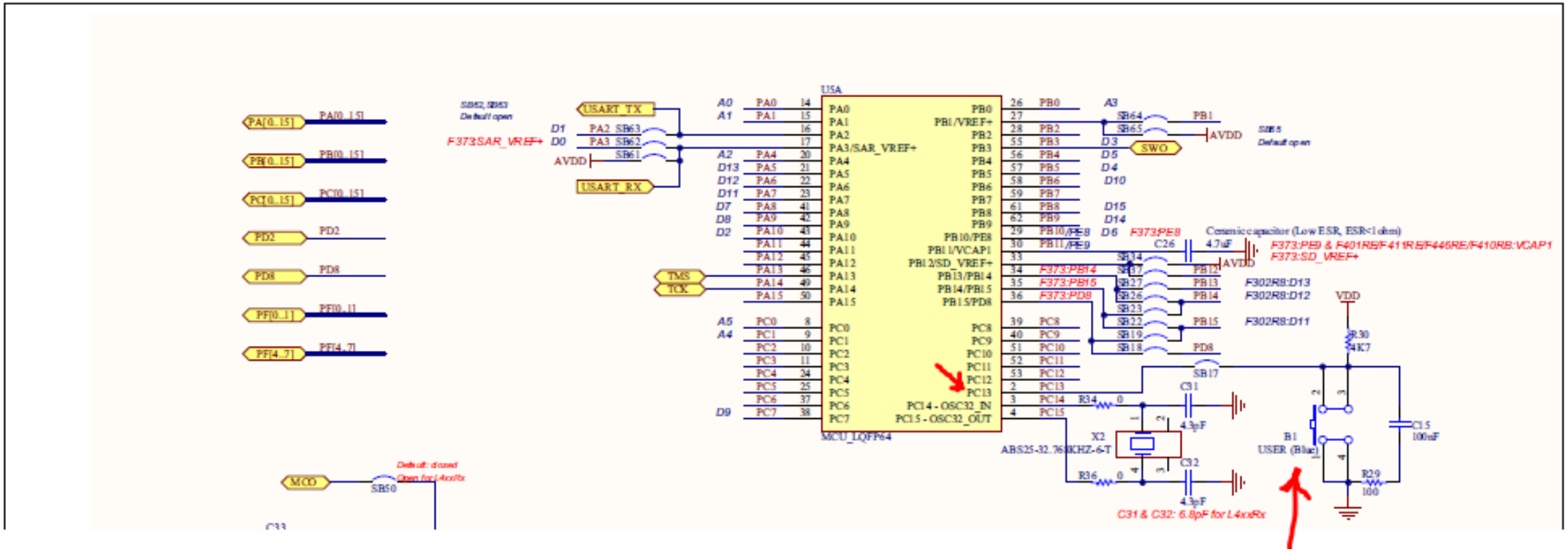
Problem Breakdown

1. Read from push button and toggle LED inside while loop (15min)
2. Connect button to EXTI interrupt and toggle LED inside interrupt handler (15min)
3. Implement LED states, ignoring PWM for the moment (15min)
4. Implement PWM code to modify LED brightness (15min)
5. Enable LED output to switch between GPIO & PWM functionality (15min)

Read from push button and toggle LED inside while loop (15min)

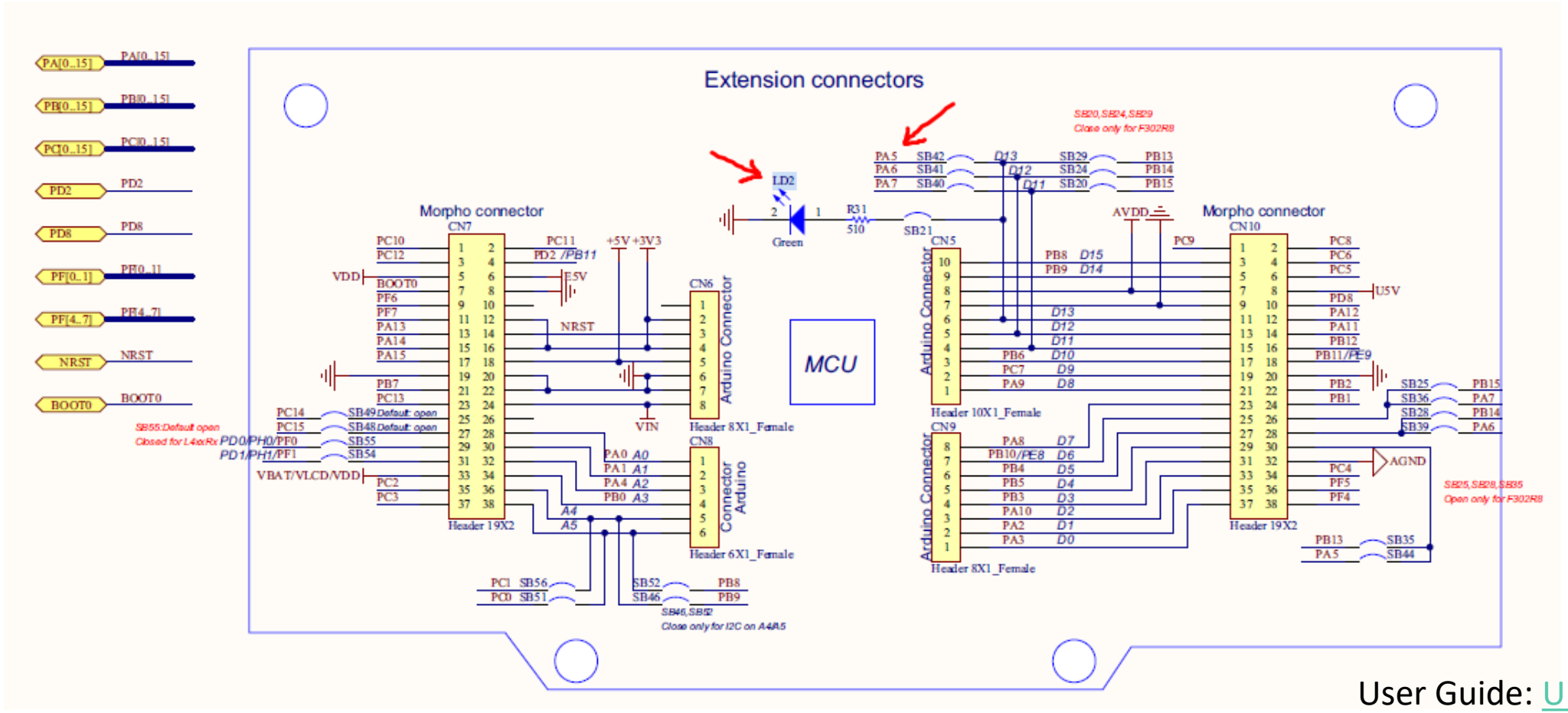
User Button – **PC13**

Figure 28. STM32 MCU



User Guide: UM1724

User LED – PA5



User Guide: UM1724

GPIOx_IDR Register

8.4.5 GPIO port input data register (GPIOx_IDR) (x = A..E and H)

Address offset: 0x10

Reset value: 0x0000 XXXX (where X means undefined)

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
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **IDRy**: Port input data (y = 0..15)

These bits are read-only and can be accessed in word mode only. They contain the input value of the corresponding I/O port.

RM0368

GPIO via CMSIS

stm32f4xx_hal_gpio.h

```
93  /* Exported constants --------------------------------------------------------*/
94
95  /** @defgroup GPIO_Exported_Constants GPIO Exported Constants
96    * @{
97    */
98
99  /** @defgroup GPIO_pins_define GPIO pins define
100   * @{
101   */
102 #define GPIO_PIN_0                 ((uint16_t)0x0001)  /* Pin 0 selected    */
103 #define GPIO_PIN_1                 ((uint16_t)0x0002)  /* Pin 1 selected    */
104 #define GPIO_PIN_2                 ((uint16_t)0x0004)  /* Pin 2 selected    */
105 #define GPIO_PIN_3                 ((uint16_t)0x0008)  /* Pin 3 selected    */
106 #define GPIO_PIN_4                 ((uint16_t)0x0010)  /* Pin 4 selected    */
107 #define GPIO_PIN_5                 ((uint16_t)0x0020)  /* Pin 5 selected    */
108 #define GPIO_PIN_6                 ((uint16_t)0x0040)  /* Pin 6 selected    */
109 #define GPIO_PIN_7                 ((uint16_t)0x0080)  /* Pin 7 selected    */
110 #define GPIO_PIN_8                 ((uint16_t)0x0100)  /* Pin 8 selected    */
111 #define GPIO_PIN_9                 ((uint16_t)0x0200)  /* Pin 9 selected    */
112 #define GPIO_PIN_10                ((uint16_t)0x0400)  /* Pin 10 selected   */
113 #define GPIO_PIN_11                ((uint16_t)0x0800)  /* Pin 11 selected   */
114 #define GPIO_PIN_12                ((uint16_t)0x1000)  /* Pin 12 selected   */
115 #define GPIO_PIN_13                ((uint16_t)0x2000)  /* Pin 13 selected   */
116 #define GPIO_PIN_14                ((uint16_t)0x4000)  /* Pin 14 selected   */
117 #define GPIO_PIN_15                ((uint16_t)0x8000)  /* Pin 15 selected   */
118 #define GPIO_PIN_All               ((uint16_t)0xFFFF)  /* All pins selected */
```

Steps

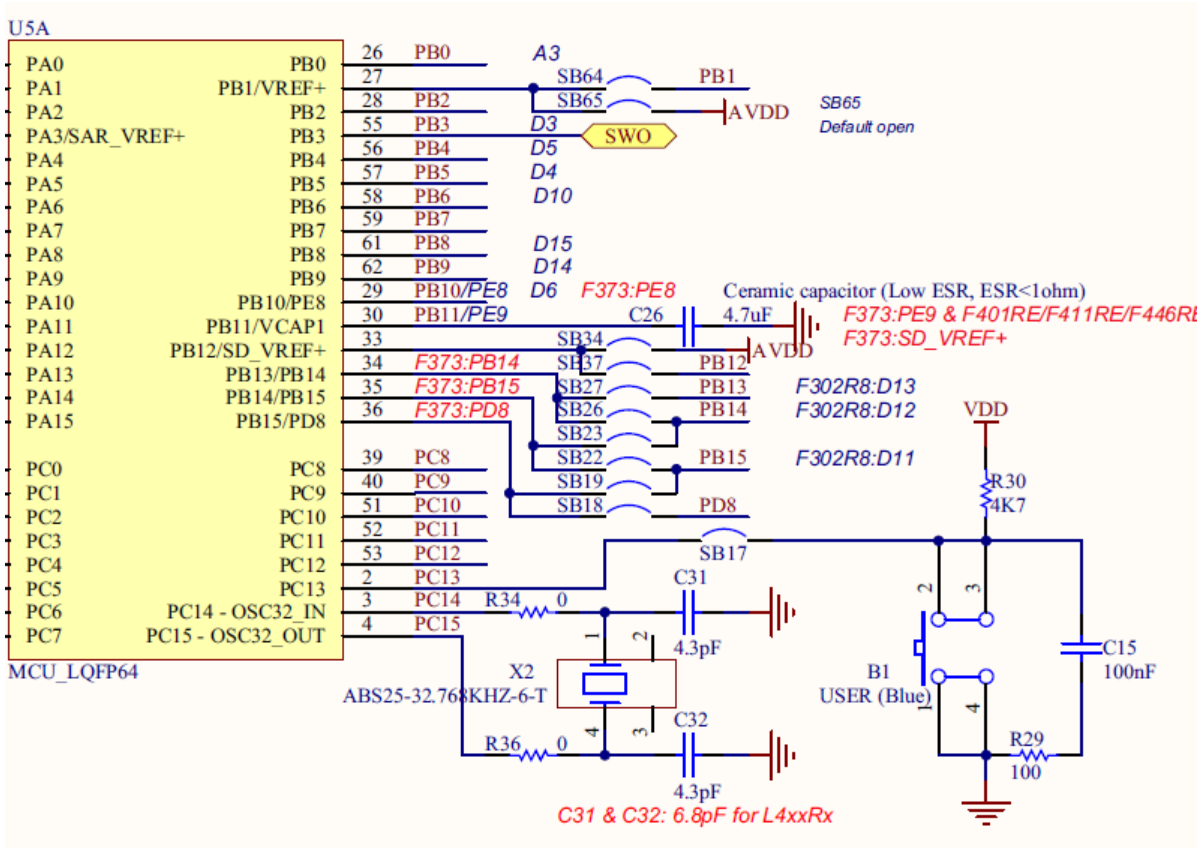
1. Start ST32CubeMX
2. Setup PC13 as GPIO_INPUT
3. Setup PA5 as GPIO_OUTPUT
4. Generate code
5. Use CMSIS structures to read from push button and to toggle LED.
6. Test code.

```
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{

/* USER CODE END WHILE */

/* USER CODE BEGIN 3 */
    if (GPIOC->IDR & GPIO_PIN_13)
    {
        // Button Pushed (low input)
        GPIOA->ODR ^= ~GPIO_PIN_5;
    }
    else
    {
        // Button Released (high input)
        GPIOA->ODR |= GPIO_PIN_5;
    }
}
/* USER CODE END 3 */
```

Why **GPIOC->IDR** is “one” by default?



Note: When editing generated code…

- Always place your code between the “USER” comments

/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

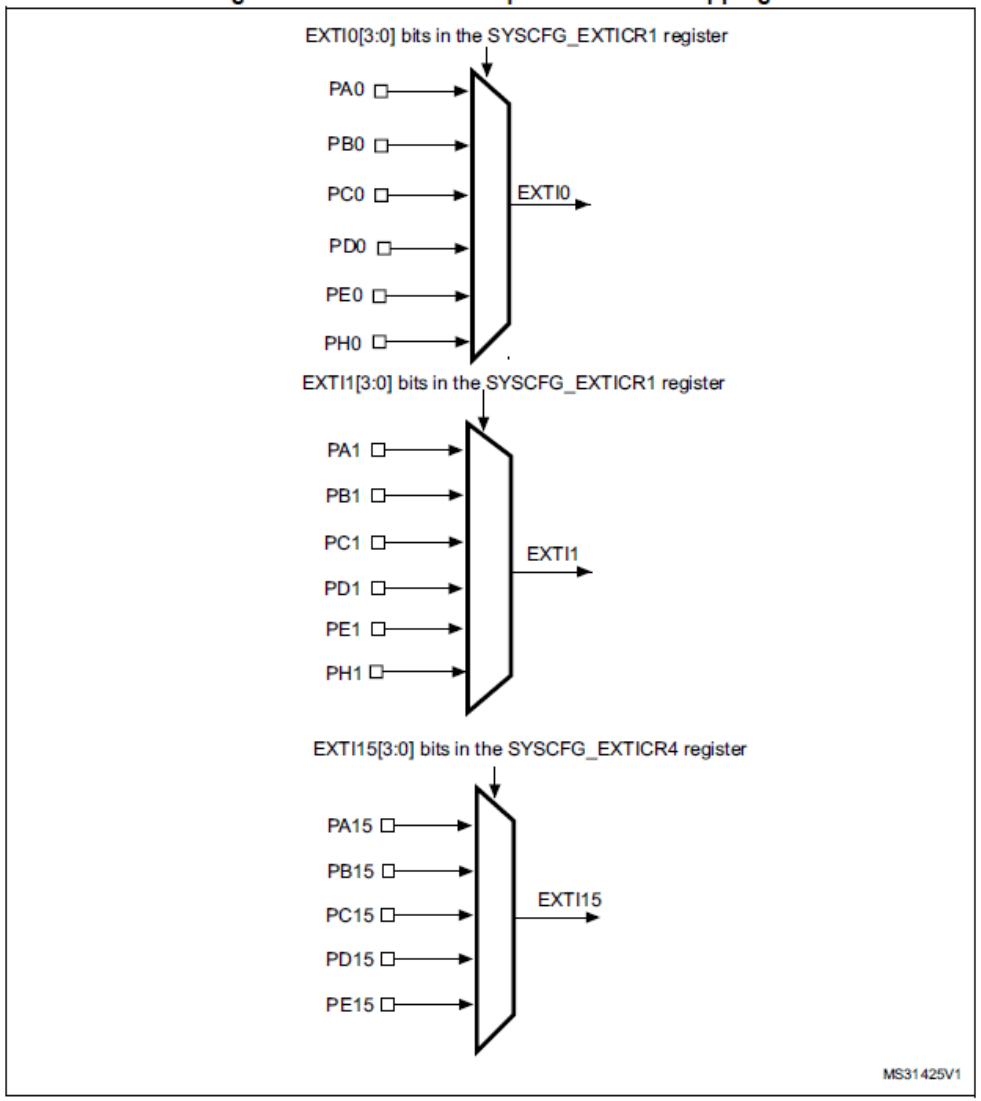
- When you generate new code, STM32CubeMX will NOT override code between user comments.

Connect button to **EXTI** interrupt and toggle LED inside interrupt handler (15min)

10.2.5 External interrupt/event line mapping

Up to 81 GPIOs (STM32F401xB/C and STM32F401xD/E) are connected to the 16 external interrupt/event lines in the following manner:

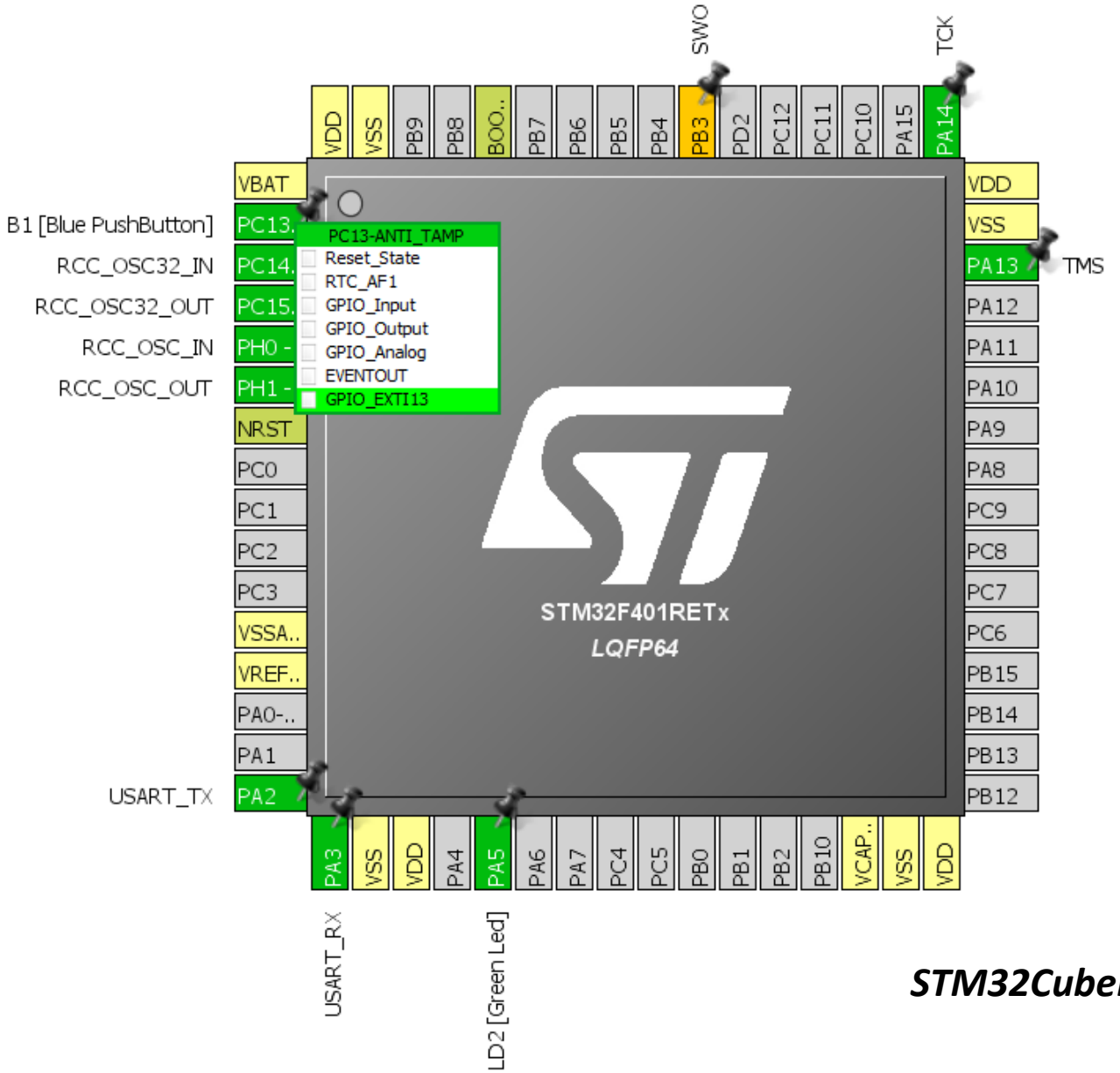
Figure 30. External interrupt/event GPIO mapping



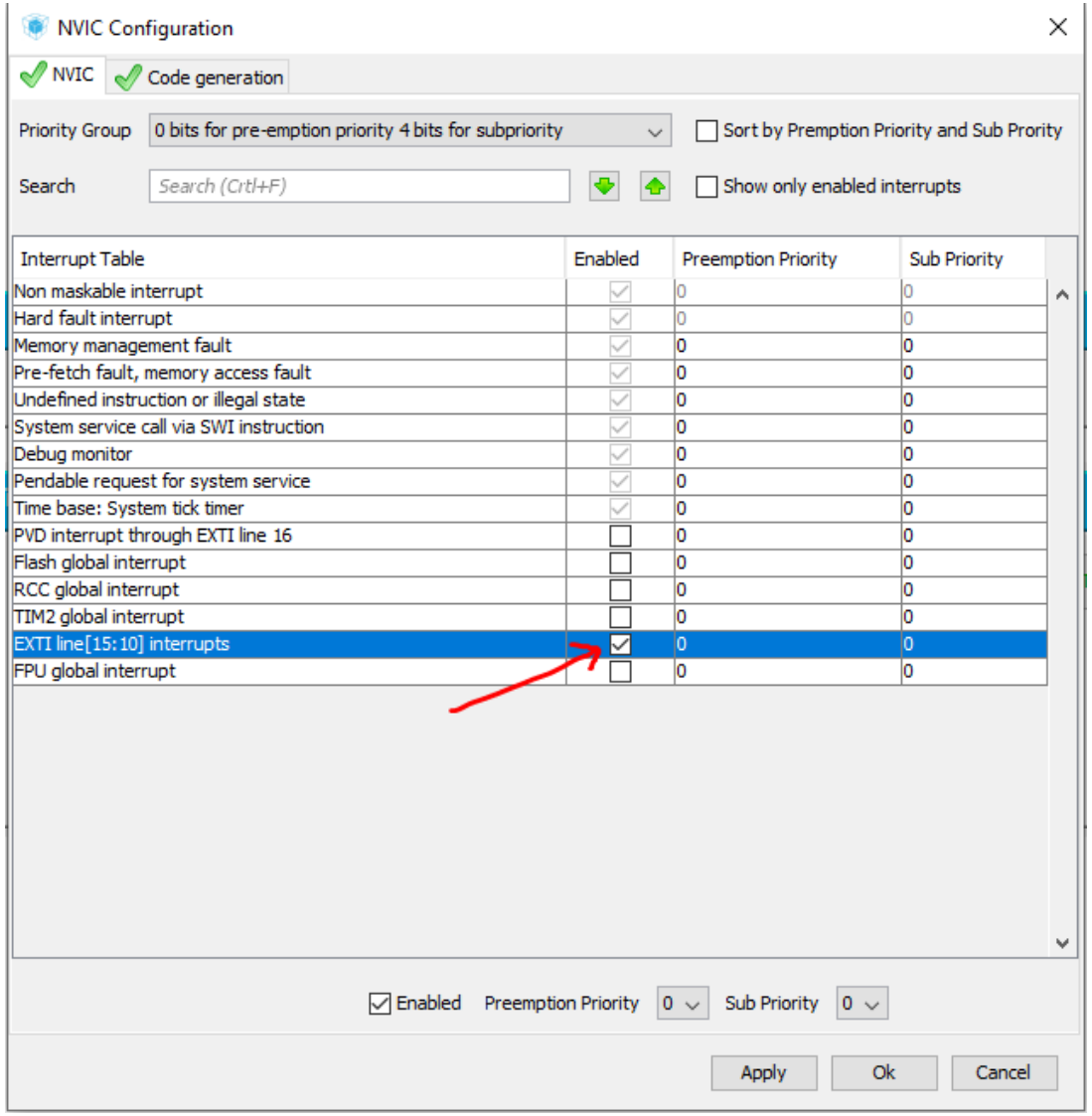
External Interrupts

STM32F401 Reference Manual – RM0368

Push Button PC13 & EXTI13



STM32CubeMX



Enable Interrupt

- In pin out section , select GPIO_EXTI13
- Select the NVIC Configuration option
- Enable EXTI

Table 38. Vector table for STM32F401xB/CSTM32F401xD/E (continued)

Position	Priority	Type of priority	Acronym	Description	Address
15	22	settable	DMA1_Stream4	DMA1 Stream4 global interrupt	0x0000 007C
16	23	settable	DMA1_Stream5	DMA1 Stream5 global interrupt	0x0000 0080
17	24	settable	DMA1_Stream6	DMA1 Stream6 global interrupt	0x0000 0084
18	25	settable	ADC	ADC1 global interrupts	0x0000 0088
23	30	settable	EXTI9_5	EXTI Line[9:5] interrupts	0x0000 009C
24	31	settable	TIM1_BRK_TIM9	TIM1 Break interrupt and TIM9 global interrupt	0x0000 00A0
25	32	settable	TIM1_UP_TIM10	TIM1 Update interrupt and TIM10 global interrupt	0x0000 00A4
26	33	settable	TIM1_TRG_COM_TIM11	TIM1 Trigger and Commutation interrupts and TIM11 global interrupt	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
40	47	settable	EXTI15_10	EXTI Line[15:10] interrupts	0x0000 00E0
41	48	settable	EXTI17 / RTC_Alarm	EXTI Line 17 interrupt / RTC Alarms (A and B) through EXTI line interrupt	0x0000 00E4

EXTI15_10

STM32F401 Reference Manual – RM0368

Steps

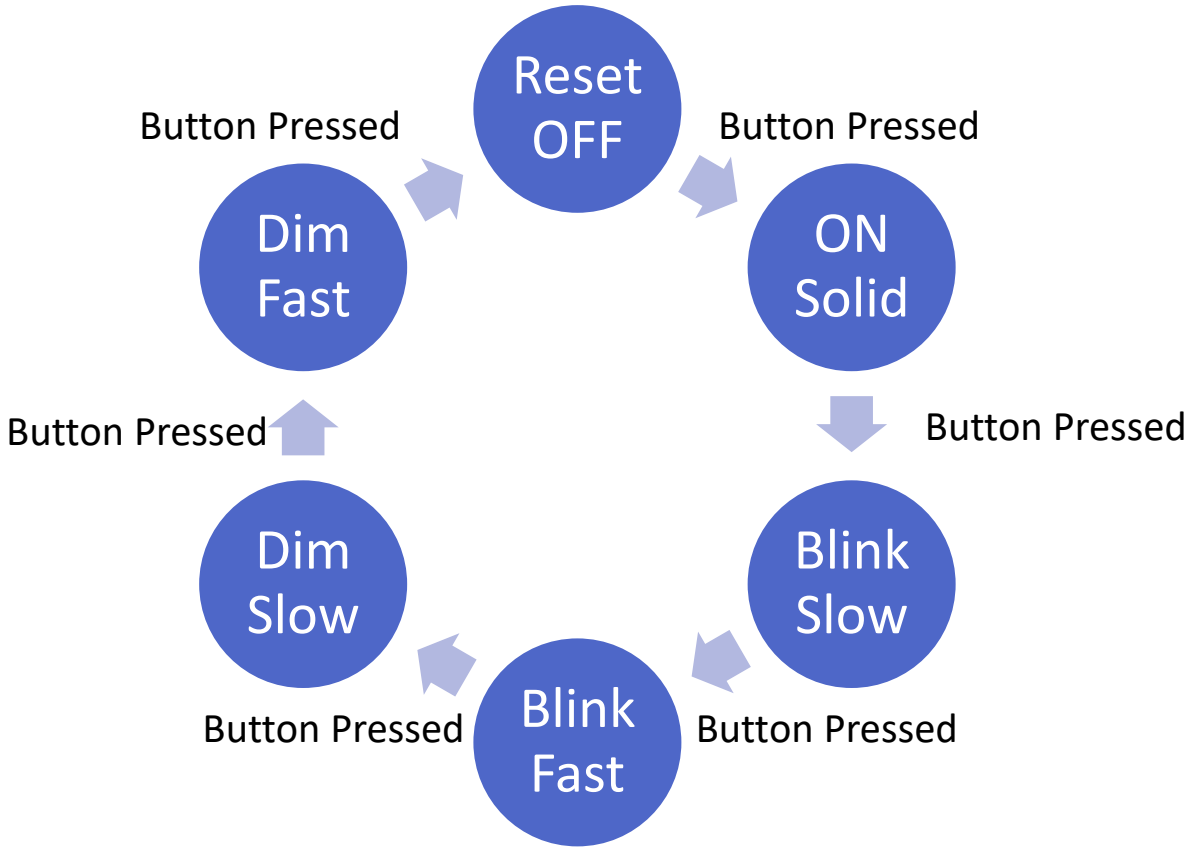
1. Back in ST32CubeMX
2. Change PC13 to GPIO_XTI13 (keep same user label)
3. Go to configuration -> NVIC
4. Set EXTI to “Enabled”
5. Generate code
6. Move LED toggling code inside handler.
7. Test code.

```
/**
* @brief This function handles EXTI line[15:10] interrupts.
*/
void EXTI15_10_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI15_10_IRQn 0 */

  /* USER CODE END EXTI15_10_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_13);
  /* USER CODE BEGIN EXTI15_10_IRQn 1 */
    GPIOA->ODR ^= GPIO_PIN_5;
  /* USER CODE END EXTI15_10_IRQn 1 */
}
```

Implement LED States (15min)

LED State Diagram



High level steps

1. Could use **enum** for defining the states
2. A global variable capturing the current LED state in the program.
3. Capture current state inside of the while loop in main (ignoring PWM for now)
4. Implement code inside the interrupt handler to change the states.

Steps inside main

1. Create **led_states.h** file and add to project
2. Define **enum LED_STATE** in led_states.h file
3. Include lead_states.h in main.c
4. Create global variable **ledState** to track current state.
5. Implement state machine code inside while loop in main.c

```
    switch (ledState)
    {
    case LED_OFF:
        GPIOA->ODR &= ~GPIO_ODR_OD5;
        break;
    case LED_SOLID:
        GPIOA->ODR |= GPIO_ODR_OD5;
        break;
    case LED_BLINK_SLOW:
        GPIOA->ODR ^= GPIO_ODR_OD5;
        HAL_Delay(500);
        break;
    case LED_BLINK_FAST:
        GPIOA->ODR ^= GPIO_ODR_OD5;
        HAL_Delay(100);
        break;
    case LED_DIM_SLOW:
        GPIOA->ODR &= ~GPIO_ODR_OD5;
        break;
    case LED_DIM_FAST:
        GPIOA->ODR &= ~GPIO_ODR_OD5;
        break;
    default:
      GPIOA->ODR &= ~GPIO_ODR_OD5;
    }
}
```

Steps inside interrupt handler

1. Declare global variable **ledState** as “extern” inside **stm32f4xx_it.c** file.
2. Include “**led_states.h**” file.
3. Implement state machine code inside interrupt handler.
4. Test code.
5. BUG!!

```
void EXTI15_10_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI15_10_IRQn 0 */

  /* USER CODE END EXTI15_10_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_13);
  /* USER CODE BEGIN EXTI15_10_IRQn 1 */
//    GPIOA->ODR ^= GPIO_PIN_5;
    switch (ledState)
    {
    case LED_OFF:
        ledState = LED_SOLID;
        break;
    case LED_SOLID:
        ledState = LED_BLINK_SLOW;
        break;
    case LED_BLINK_SLOW:
        ledState = LED_BLINK_FAST;
        break;
    case LED_BLINK_FAST:
        ledState = LED_DIM_SLOW;
        break;
    case LED_DIM_SLOW:
        ledState = LED_DIM_FAST;
        break;
    case LED_DIM_FAST:
        ledState = LED_OFF;
        break;
    default:
        ledState = LED_OFF;
    }
  /* USER CODE END EXTI15_10_IRQn 1 */
}
```

Optimization issue…

- Either set optimization to “None”
- Or set ledState variable to volatile in both locations:
  - enum LED_STATE volatile ledState = LED_OFF; // Inside main.c
  - extern enum LED_STATE volatile ledState; // inside stm32f4xx_it.c

BREAK 1

Use PWM to change LED brightness (15min)

Steps

1. Back in STM32CubeMX
2. Connect PA5 to “TIM2_CH1”
3. Configure “TIM2” channel 1 to “PWM Generation CH1”
4. Follow instructions from PWM Tutorial
5. Generate code
6. Set PWM user values inside switch case
7. And change delay for fast/slow dimming of LED
8. Test code

➔ Only dimming states are active!!

Problem

After enabling the PWM functionality, we can no longer control the LED as a GPIO “output’ pin!

Switching between GPIO and PWM (15min)

GPIOx_MODER – Alternate Function

main.c | stm32f4xx_hal_msp.c | stm32f4xx_it.c | stm32f4xx_hal_gpio.c | stm32f4xx_hal_tim.c | stm32f4xx_hal_gpio.h

MX_TIM2_Init()

```
225  /* TIM2 init function */
226  static void MX_TIM2_Init(void)
227  {
228
229    TIM_MasterConfigTypeDef sMasterConfig;
230    TIM_OC_InitTypeDef sConfigOC;
231
232    htim2.Instance = TIM2;
233    htim2.Init.Prescaler = 160-1;
234    htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
235    htim2.Init.Period = 2000-1;
236    htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
237    if (HAL_TIM_PWM_Init(&htim2) != HAL_OK)
238    {
239      _Error_Handler(__FILE__, __LINE__);
240    }
241
242    sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
243    sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
244    if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
245    {
246      _Error_Handler(__FILE__, __LINE__);
247    }
248
249    sConfigOC.OCMode = TIM_OCMODE_PWM1;
250    sConfigOC.Pulse = 0;
251    sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
252    sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
253    if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
254    {
255      _Error_Handler(__FILE__, __LINE__);
256    }
257
258    HAL_TIM_MspPostInit(&htim2);
259
260  }
```

stm32f4xx_hal_msp.c | main.c | stm32f4xx_it.c | stm32f4xx_hal_gpio.c | stm

HAL_TIM_MspPostInit(TIM_HandleTypeDef *)

```
99   void HAL_TIM_MspPostInit(TIM_HandleTypeDef* htim)
100  {
101
102    GPIO_InitTypeDef GPIO_InitStruct;
103    if(htim->Instance==TIM2)
104    {
105    /* USER CODE BEGIN TIM2_MspPostInit 0 */
106
107    /* USER CODE END TIM2_MspPostInit 0 */
108
109      /**TIM2 GPIO Configuration
110      PA5     ------> TIM2_CH1
111      */
112      GPIO_InitStruct.Pin = GPIO_PIN_5;
113      GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
114      GPIO_InitStruct.Pull = GPIO_NOPULL;
115      GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
116      GPIO_InitStruct.Alternate = GPIO_AF1_TIM2;
117      HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
118
119    /* USER CODE BEGIN TIM2_MspPostInit 1 */
120
121    /* USER CODE END TIM2_MspPostInit 1 */
122    }
123
124  }
```

GPIO Alternate Function

- Each of the GPIO pins can be configured by software as
  - **Output** (push-pull or open-drain, with or without pull-up or pull-down)
  - **Input** (floating, with or without pull-up or pull-down)
  - Peripheral **alternate function**.
- The “alternate” function refers to other functionality for that pin.
- This may include I2C, SPI, USART, PWM, Clock, ADC, etc...
- The application can select one of the possible peripheral functions for each I/O at a time.

GPIO Alternate Function

Many of the I/O pins are connected to onboard peripherals through a multiplexer that allows only **one** peripheral alternate function (AF) to connect to an I/O.

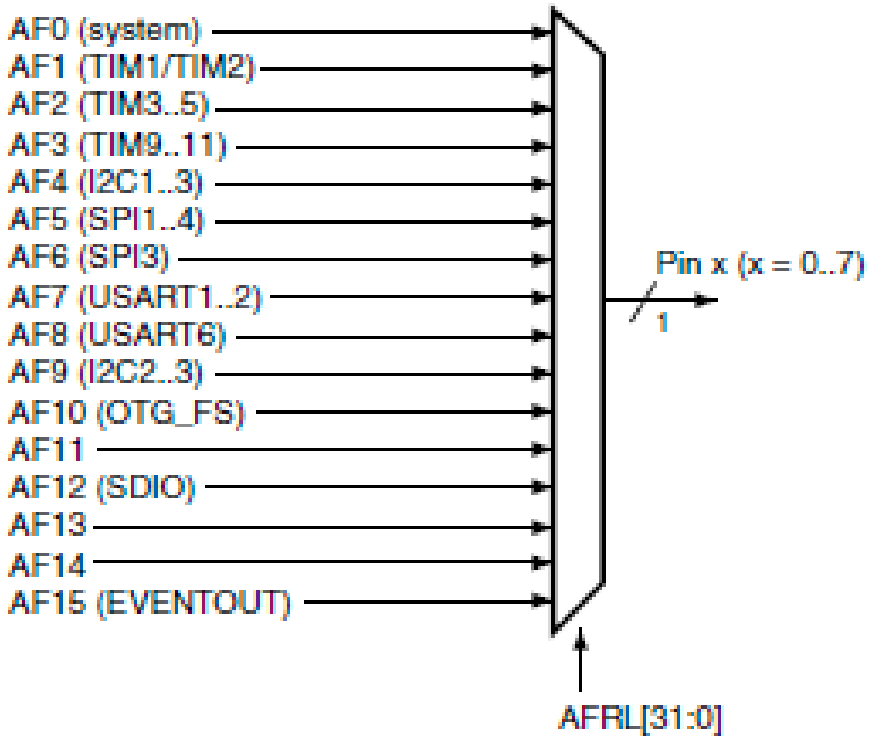


Table 9. Alternate function mapping

Port		AF00	AF01	AF02	AF03	AF04	AF05	AF06	AF07	AF08	AF09	AF10	AF11	AF12	AF13	AF14	AF15
		SYS_AF	TIM1/TIM2	TIM3/ TIM4/ TIM5	TIM9/ TIM10/ TIM11	I2C1/I2C2/ I2C3	SPI1/SPI2/ I2S2/SPI3/ I2S3/SPI4	SPI2/I2S2/ SPI3/ I2S3	SPI3/I2S3/ USART1/ USART2	USART6	I2C2/ I2C3	OTG1_FS		SDIO			
Port A	PA0	-	TIM2_CH1/ TIM2_ETR	TIM5_CH1	-	-	-	-	USART2_ CTS	-	-	-	-	-	-	-	EVENT OUT
	PA1	-	TIM2_CH2	TIM5_CH2	-	-	-	-	USART2_ RTS	-	-	-	-	-	-	-	EVENT OUT
	PA2	-	TIM2_CH3	TIM5_CH3	TIM9_CH1	-	-	-	USART2_ TX	-	-	-	-	-	-	-	EVENT OUT
	PA3	-	TIM2_CH4	TIM5_CH4	TIM9_CH2	-		-	USART2_ RX	-	-	-	-	-	-	-	EVENT OUT
	PA4	-	-	-	-	-	SPI1_NSS	SPI3_NSS/ I2S3_WS	USART2_ CK	-	-	-	-	-	-	-	EVENT OUT
	PA5	-	TIM2_CH1/ TIM2_ETR	-	-	-	SPI1_SCK	-	-	-	-	-	-	-	-	-	EVENT OUT
	PA6	-	TIM1_BKIN	TIM3_CH1	-	-	SPI1_ MISO	-	-	-	-	-	-	-	-	-	EVENT OUT
	PA7	-	TIM1_CH1N	TIM3_CH2	-	-	SPI1_ MOSI	-	-	-	-	-	-	-	-	-	EVENT OUT
	PA8	MCO_1	TIM1_CH1	-	-	I2C3_SCL	-	-	USART1_ CK	-	-	OTG_FS_ SOF	-	-	-	-	EVENT OUT
	PA9	-	TIM1_CH2	-	-	I2C3_ SMBA	-	-	USART1_ TX	-	-	OTG_FS_ VBUS	-	-	–	-	EVENT OUT
	PA10	-	TIM1_CH3	-	-	-	-	-	USART1_ RX	-	-	OTG_FS_I D	-	-	-	-	EVENT OUT
	PA11	-	TIM1_CH4	-	-	-	-	-	USART1_ CTS	USART6_ TX	-	OTG_FS_ DM	-	-	-	-	EVENT OUT
	PA12	-	TIM1_ETR	-	-	-	-	-	USART1_ RTS	USART6_ RX	-	OTG_FS_ DP	-	-	-	-	EVENT OUT
	PA13	JTMS_ SWDIO	-	-	-	-	-	-	-	-	-	-	-	-	-	-	EVENT OUT
	PA14	JTCK_ SWCLK	-	-	-	-	-	-	-	-	-	-	-	-	-	-	EVENT OUT
	PA15	JTDI	TIM2_CH1/ TIM2_ETR	-	-	-	SPI1_NSS	SPI3_NSS/ I2S3_WS	-	-	-	-	-	-	-	-	EVENT OUT

Alternate Function Mapping for **GPIOA**

STM32F401 Datasheet

GPIOx_MODER – Alternate Function

8.4.1 GPIO port mode register (GPIOx_MODER) (x = A..E and H)

Address offset: 0x00

Reset values:

- 0x0C00 0000 for port A
- 0x0000 0280 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[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
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

These bits are written by software to configure the I/O direction mode.
00: Input (reset state)
01: General purpose output mode
10: Alternate function mode
11: Analog mode

GPIOx_AFRL

8.4.9 GPIO alternate function low register (GPIOx_AFRL) (x = A..E and H)

Address offset: 0x20

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFRL7[3:0]				AFRL6[3:0]				AFRL5[3:0]				AFRL4[3:0]			
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
AFRL3[3:0]				AFRL2[3:0]				AFRL1[3:0]				AFRL0[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **AFRLy:** Alternate function selection for port x bit y (y = 0..7)

These bits are written by software to configure alternate function I/Os

AFRLy selection:

0000: AF0	1000: AF8
0001: AF1	1001: AF9
0010: AF2	1010: AF10
0011: AF3	1011: AF11
0100: AF4	1100: AF12
0101: AF5	1101: AF13
0110: AF6	1110: AF14
0111: AF7	1111: AF15

RM0368

GPIOx_AFRH

8.4.10 GPIO alternate function high register (GPIOx_AFRH) (x = A..E and H)

Address offset: 0x24

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFRH15[3:0]				AFRH14[3:0]				AFRH13[3:0]				AFRH12[3:0]			
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
AFRH11[3:0]				AFRH10[3:0]				AFRH9[3:0]				AFRH8[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **AFRHy:** Alternate function selection for port x bit y (y = 8..15)

These bits are written by software to configure alternate function I/Os

AFRHy selection:

0000: AF0	1000: AF8
0001: AF1	1001: AF9
0010: AF2	1010: AF10
0011: AF3	1011: AF11
0100: AF4	1100: AF12
0101: AF5	1101: AF13
0110: AF6	1110: AF14
0111: AF7	1111: AF15

RM0368

Configuring GPIO on the STM32F4xx

When using the CMSIS library there are four considerations:

- GPIO Mode
- GPIO Output Type
- GPIO speed
- GPIO pullup/pulldown resistors

```
/*Configure GPIO pin : B1_Pin */
GPIO_InitStruct.Pin = B1_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);

/*Configure GPIO pin : LD2_Pin */
GPIO_InitStruct.Pin = LD2_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(LD2_GPIO_Port, &GPIO_InitStruct);
```

Toggle GPIO Modes

GPIO MODE

```
case LED_OFF:
    GPIOA->MODER &= ~GPIO_MODER_MODE5_Msk;     // Clear out Mode for Pin5
    GPIOA->MODER |= GPIO_MODER_MODE5_0;        // Set Pin5 to GPIO Mode
    GPIOA->ODR &= ~GPIO_ODR_OD5;
    break;
case LED_SOLID:
    GPIOA->MODER &= ~GPIO_MODER_MODE5_Msk;     // Clear out Mode for Pin5
    GPIOA->MODER |= GPIO_MODER_MODE5_0;        // Set Pin5 to GPIO Mode
    GPIOA->ODR |= GPIO_ODR_OD5;
    break;
case LED_BLINK_SLOW:
    GPIOA->MODER &= ~GPIO_MODER_MODE5_Msk;     // Clear out Mode for Pin5
    GPIOA->MODER |= GPIO_MODER_MODE5_0;        // Set Pin5 to GPIO Mode
    GPIOA->ODR ^= GPIO_ODR_OD5;
    HAL_Delay(500);
    break;
case LED_BLINK_FAST:
    GPIOA->MODER &= ~GPIO_MODER_MODE5_Msk;     // Clear out Mode for Pin5
    GPIOA->MODER |= GPIO_MODER_MODE5_0;        // Set Pin5 to GPIO Mode
    GPIOA->ODR ^= GPIO_ODR_OD5;
    HAL_Delay(100);
    break;
```

PWM ALTERNATE FUNCTION MODE

```
case LED_DIM_SLOW:
    GPIOA->MODER &= ~GPIO_MODER_MODE5_Msk;     // Clear out Mode for Pin5
    GPIOA->MODER |= GPIO_MODER_MODE5_1;        // Set Pin5 to AF Mode
    GPIOA->AFR[0] |= 0x1;                      // Set TIM2 AF in AFRL for Pin5
    HAL_Delay(50);
    if(pwm_value == 0) step = 100;
    if(pwm_value == 2000) step = -100;
    pwm_value += step;
    user_pwm_setvalue(pwm_value);
    break;
case LED_DIM_FAST:
    GPIOA->MODER &= ~GPIO_MODER_MODE5_Msk;     // Clear out Mode for Pin5
    GPIOA->MODER |= GPIO_MODER_MODE5_1;        // Set Pin5 to AF Mode
    GPIOA->AFR[0] |= 0x1;                      // Set TIM2 AF in AFRL for Pin5
    HAL_Delay(25);
    if(pwm_value == 0) step = 100;
    if(pwm_value == 2000) step = -100;
    pwm_value += step;
    user_pwm_setvalue(pwm_value);
    break;
```

```
void EXTI15_10_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI15_10_IRQn 0 */
    __disable_irq();

  /* USER CODE END EXTI15_10_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_13);
  /* USER CODE BEGIN EXTI15_10_IRQn 1 */
    switch (ledState)
    {
    case LED_OFF:
        ledState = LED_SOLID;
        break;
    case LED_SOLID:
        ledState = LED_BLINK_SLOW;
        break;
    case LED_BLINK_SLOW:
        ledState = LED_BLINK_FAST;
        break;
    case LED_BLINK_FAST:
        ledState = LED_DIM_SLOW;
        break;
    case LED_DIM_SLOW:
        ledState = LED_DIM_FAST;
        break;
    case LED_DIM_FAST:
        ledState = LED_OFF;
        break;
    default:
        ledState = LED_OFF;
    }
    __enable_irq();
  /* USER CODE END EXTI15_10_IRQn 1 */
}
```

Interrupts final note

If code is placed after clearing of interrupt, will need to disable interrupts to avoid multiple triggers.

Other possible features

- Detect a long press of several seconds (3 seconds)
- If the LED is already ON, a long press will cause the LED to dim over two seconds all the way down to the OFF state.
- If the LED is OFF, a long press will cause the LED to increase the brightness over two seconds all the way back to the fully ON state.
- Add more…

BREAK 2

SECOND COURSE OVERVIEW

BY NICK STRATHY

Programming with Embedded & Real-Time Operating Systems