SỬ DỤNG WATCHDOG TIMER TRONG STM32

Watchdog Timer là bộ ngoại vi tích hợp trên vi điều khiển STM32 có khả năng giúp cho người dùng phát hiện ra hệ thống bị treo và tạo ra một ngắt hoặc một tín hiệu reset chip.

Đối với các sản phẩm điện tử được phát hành ra ngoài thị trường, thì do nhiều yếu tố bên trong (code chưa chặt chẽ, code có bug…) và yếu tố bên ngoài (nhiễu, điều kiện nhiệt độ, độ ẩm) dẫn đến việc hoạt động sai của VDK dẫn đến các hiện tượng chạy chức năng bị sai, hoặc treo chip. Trong trường hợp như vậy chúng ta cần có các biện pháp khắc phục tạm thời ví dụ như reset chip để chạy lại chương trình. Thì bộ Watchdog Timer được sinh ra với mục đích tạo ra một tín hiệu reset bằng Software.

I. Lý thuyết

STM32 có 2 chế độ Watchdog Timer là

- **Independent Watchdog (IWDG)**
- **Window Watchdog (WWDG)**

Independent Watchdog được cấp nguồn Clock từ bộ dao động **LSI** (Low-speed clock) bởi vậy nó có thể hoạt động độc lập ngay cả khi nguồn Clock của chương trình chính xảy ra lỗi. Chế độ này phù hợp với yêu cầu bộ Watchdog có thể chạy chức năng độc lập với chương trình chính.

Window Watchdog được cấp Clock từ nguồn **APB1 Clock**, nó chỉ có thể được cấu hình để tránh tạo ra tín hiệu reset trong 1 khoảng thời gian nhất định. Vì vậy chế độ này phù hợp với các ứng dụng yêu cầu có khả năng phát hiện sự hoạt động bất thường của các Task trong chương trình (thực thi sớm quá hoặc muộn quá).

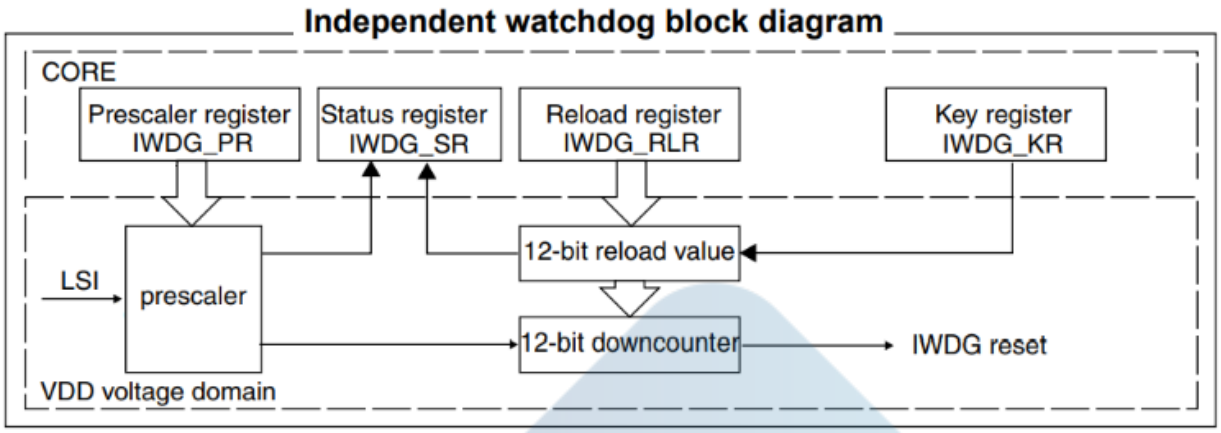
Ở bài này chúng mình sẽ tìm hiểu về chế độ *Independent Watchdog* nhé !!!

1. 1. Cơ chế tạo ra reset

Khi chế độ Independent Watchdog được bắt đầu bằng cách ghi giá trị **0xCCCC** vào thanh ghi **IWDG_KG**, bộ đếm **Counter** bắt đầu đếm xuống từ giá trị **IWDG_RLR** được cài đặt trước. Khi **Counter** đếm đến giá trị **0x0000** thì một tín hiệu reset sẽ được tạo ra và reset chip.

Bất cứ khi nào giá trị **0xAAAA** được ghi vào thanh ghi **IWDG_KR**, giá trị **IWDG_RLR** sẽ được ghi lại vào bộ **Counter** và watchdog reset sẽ bị phòng ngừa.

Khi chúng ta debug chương trình, bộ **IWDG Counter** có thể tiếp tục hoạt động hoặc dừng lại phụ thuộc vào việc cấu hình bit **DBG_IWDG_STOP**, và đương nhiên chúng ta nên dừng bộ Counter lại nếu không thì làm sao mà debug được khi chip cứ bị reset hoài !!!



Nguồn cung cấp cho IWDG là từ VDD domain. Đây là nguồn cấp riêng nên Watchdog vẫn chạy kể cả khi chương trình bị treo.

2. 2. Một số thanh ghi quan trọng

1 Prescaler register(IWDG_PR)

Address offset: 0x04
Reset value: 0x0000 0000

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
Reserved																													PR[2:0]		
																													rw	rw	rw

Bits 31:3 Reserved, must be kept at reset value.
Bits 2:0 **PR[2:0]:** Prescaler divider

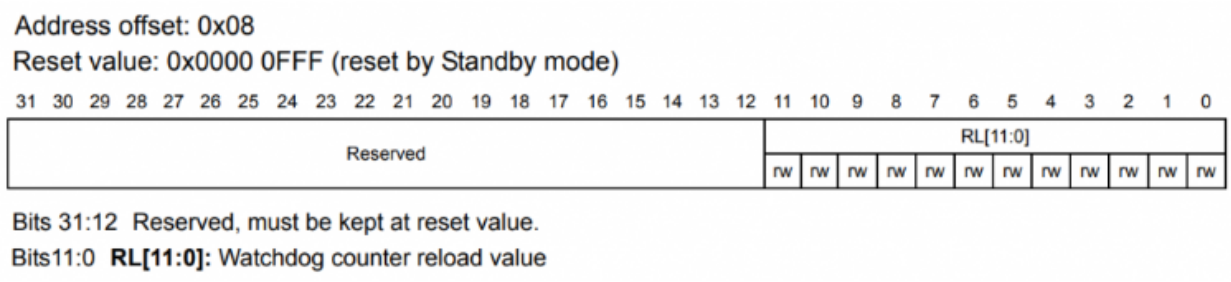
000: divider /4	010: divider /16	100: divider /64	110: divider /256
001: divider /8	011: divider /32	101: divider /128	111: divider /256

Thanh ghi chia tần số cho Watchdog timer từ nguồn LSI (40kHz).

$$F_{watchdog} = \frac{F_{LSI}}{prescaler\ divider}$$

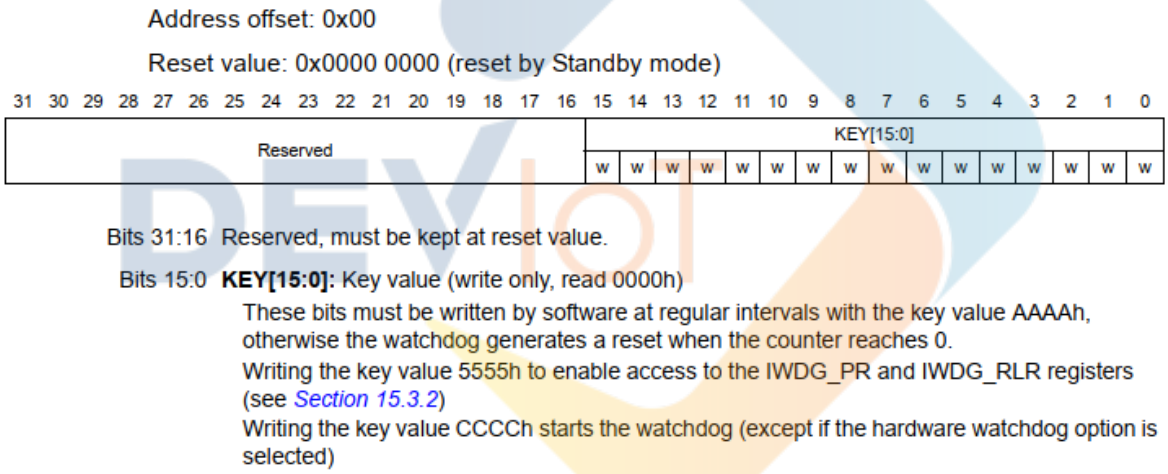
2

3 **Reload Register (IWDG_RLR)**



Giá trị bắt đầu đếm xuống của bộ đếm Counter. Ta có 12 bit giá trị vậy giá trị đếm tối đa có thể cài đặt là 0xFFF (4095).

4 **Key register (IWDG_KR)**

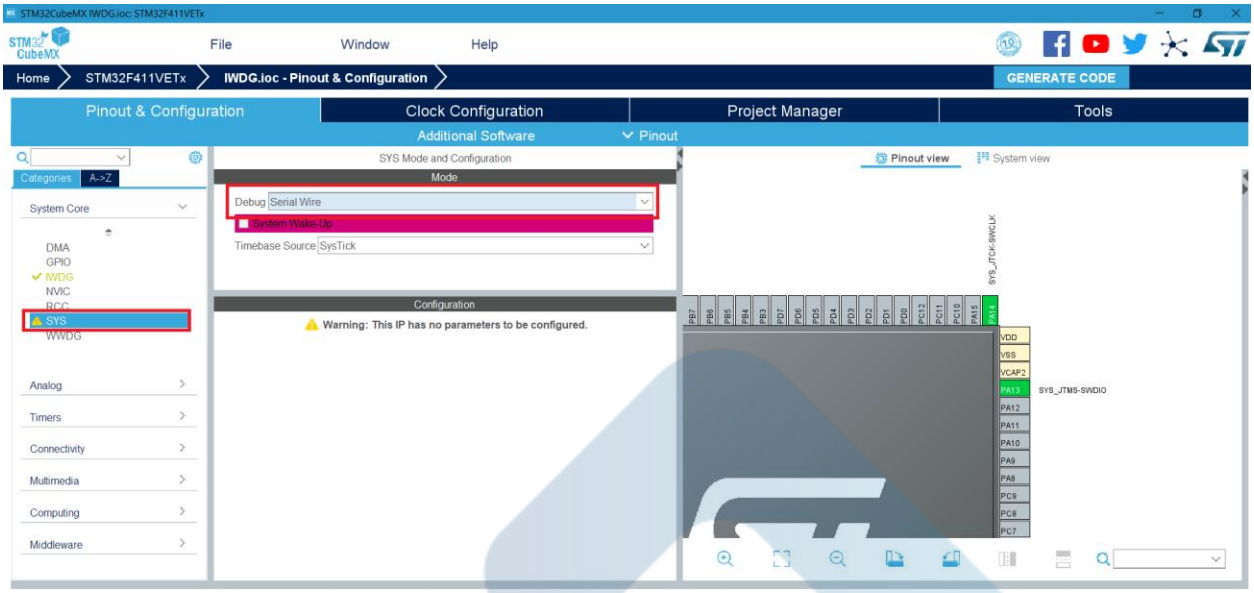


Cần ghi giá trị 0xAAAA vào thanh ghi này để có thể nạp lại giá trị cho bộ đếm Counter.

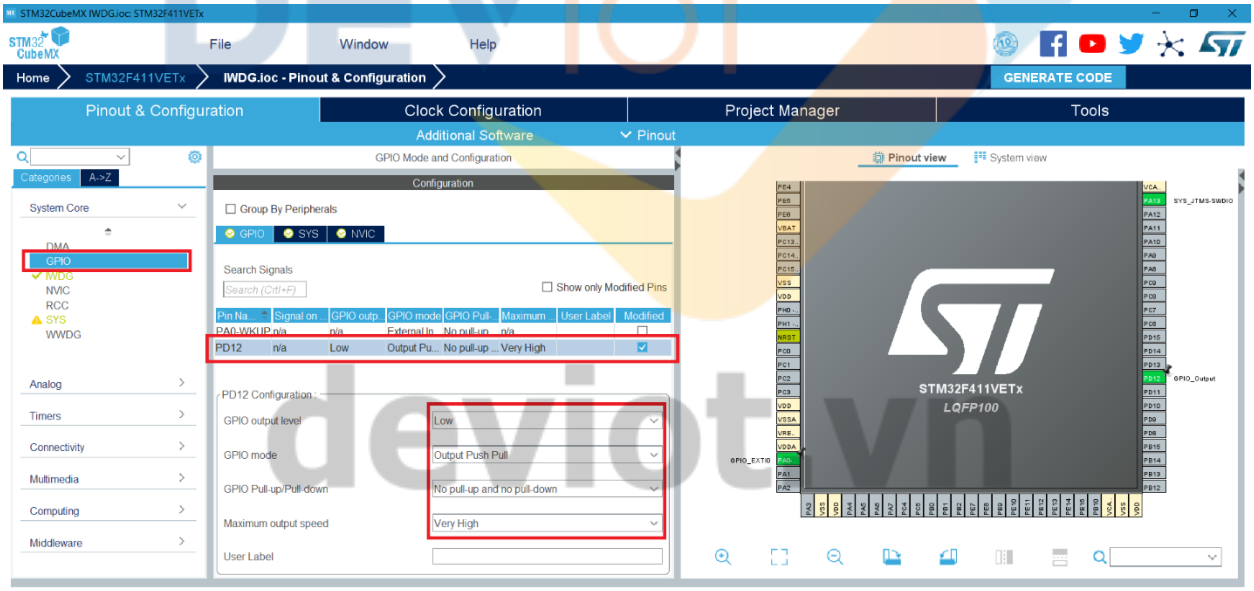
II.Lập trình

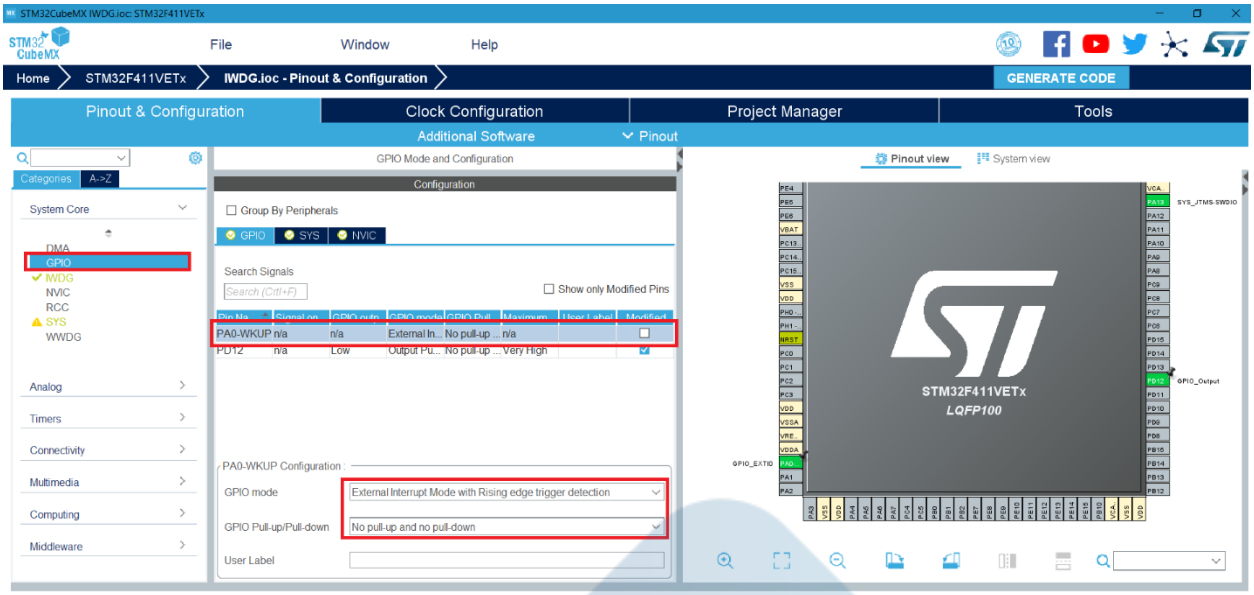
5 **Ý tưởng demo: Ở bài này mình sẽ sử dụng 1 nút bấm và 1 đèn Led có sẵn trên Dev Kit, khi chương trình khởi động, mặc định trạng thái Led sẽ tắt, sau đó mình sẽ ấn nút để bật Led sáng lên. Nếu chương trình không bị reset, các bạn sẽ thấy Led luôn sáng, nếu chương trình bị reset, Led sẽ tắt sau khoảng thời gian mình cài đặt.**

Sử dụng phần mềm CubeMX để cấu hình. **Chọn Serial Wire** như các bài trước

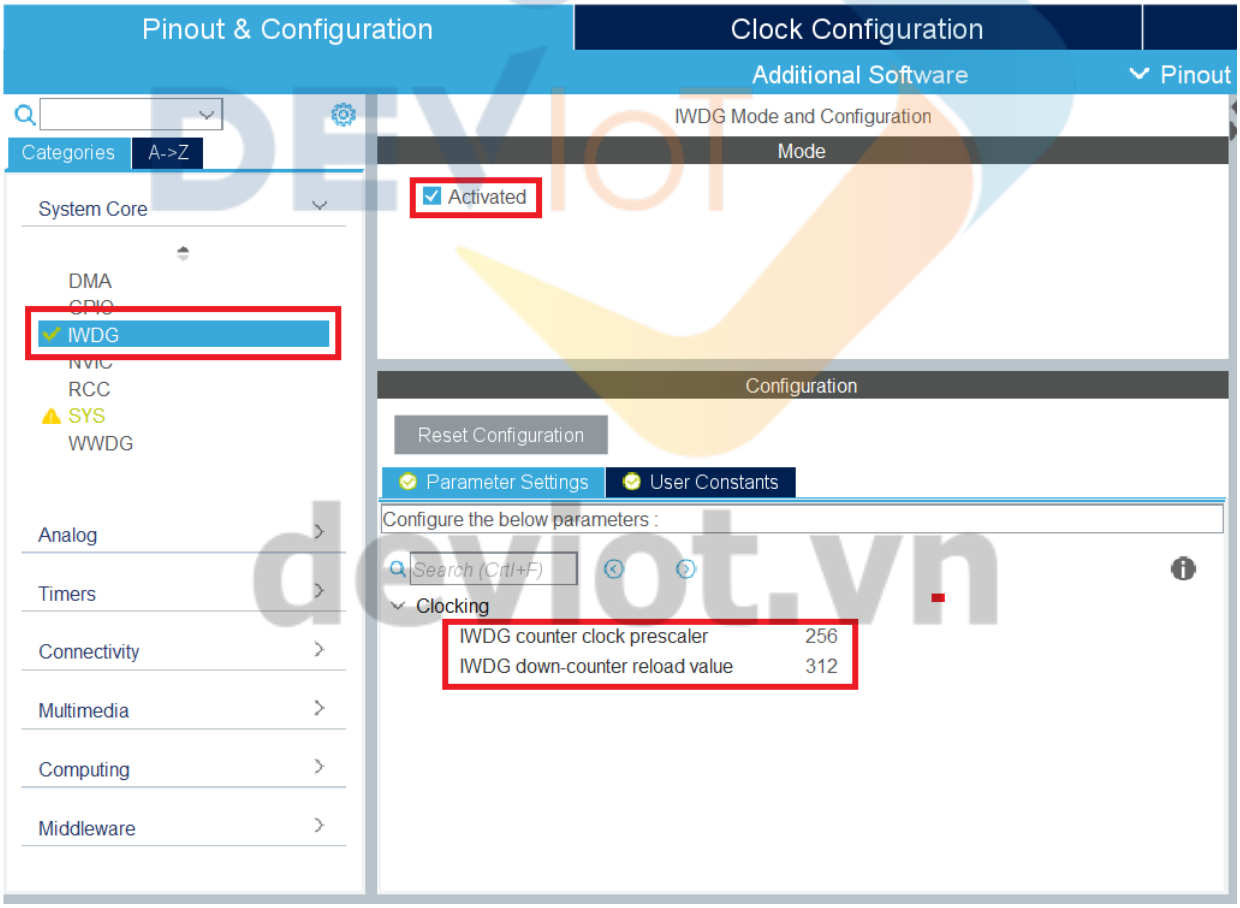


Cài đặt Led PD12 và nút bấm PA0





Cấu hình IWDG:



Ở đây mình chọn hệ số chia tần là 256, nghĩa là:

$$F_{IWDG} = \frac{F_{LSI}}{prescaler} = \frac{40000}{256}\ (Hz)$$

Từ đây ta tìm ra thời gian sẽ xảy ra reset nếu không reload lại giá trị bộ đếm Counter là:

$$T_{reset} = T_{IWDG} * IWDG_{RLR} = \frac{256}{40000} * 312 = 2(s)$$

Tiếp theo chúng ta cấu hình Clock cho hệ thống và sinh source code KeilC.

Đây là hàm main của mình nhé:

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2020 STMicroelectronics.
  * All rights reserved.</center></h2>
  *

  * This software component is licensed by ST under BSD 3-Clause license,
  * the "License"; You may not use this file except in compliance with the
  * License. You may obtain a copy of the License at:
  *                        opensource.org/licenses/BSD-3-Clause
  *
  ******************************************************************************
```

```
  */
/* USER CODE END Header */


/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */
/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */
/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
IWDG_HandleTypeDef hiwdg;

/* USER CODE BEGIN PV */
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_IWDG_Init(void);

/* USER CODE BEGIN PFP */
/* USER CODE END PFP */
```

```
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
        if(GPIO_Pin == GPIO_PIN_0) // if button is PA0
        {
             HAL_GPIO_WritePin(GPIOD,GPIO_PIN_12,GPIO_PIN_SET); // turn on led
        }
}

/* USER CODE END 0 */
/**
  * @brief  The application entry point.
  * @retval int
  */

int main(void)
{
  /* USER CODE BEGIN 1 */
  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */
  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */
  /* USER CODE END SysInit */
```

```
  /* Initialize all configured peripherals */
  MX_GPIO_Init();

  /* Init IWDG */
  MX_IWDG_Init();

  /* USER CODE BEGIN 2 */
  HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_RESET);       // turn off led as
default
  IWDG->KR = 0xAAAA; // Writing 0xAAAA in the Key register prevents watchdog reset
  IWDG->KR = 0xCCCC; // Start the independent watchdog timer
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
          IWDG->KR = 0xAAAA; // Reload Counter prevents watchdog reset
  }
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
  /** Initializes the CPU, AHB and APB busses clocks
  */
```

```
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI|RCC_OSCILLATORTYPE_LSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.LSIState = RCC_LSI_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 8;
  RCC_OscInitStruct.PLL.PLLN = 100;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 4;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB busses clocks
  */
  RCC_ClkInitStruct.ClockType =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK|RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_3) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief IWDG Initialization Function
  * @param None
  * @retval None
  */
```

```
static void MX_IWDG_Init(void)
{
  /* USER CODE BEGIN IWDG_Init 0 */
  /* USER CODE END IWDG_Init 0 */
  /* USER CODE BEGIN IWDG_Init 1 */
  /* USER CODE END IWDG_Init 1 */

  hiwdg.Instance = IWDG;
  hiwdg.Init.Prescaler = IWDG_PRESCALER_256;
  hiwdg.Init.Reload = 312;
  if (HAL_IWDG_Init(&hiwdg) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN IWDG_Init 2 */
  /* USER CODE END IWDG_Init 2 */
}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */

static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_RESET);

  /*Configure GPIO pin : PA0 */
```

```
  GPIO_InitStruct.Pin = GPIO_PIN_0;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pin : PD12 */
  GPIO_InitStruct.Pin = GPIO_PIN_12;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
  HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

  /* EXTI interrupt init*/
  HAL_NVIC_SetPriority(EXTI0_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI0_IRQn);
}

/* USER CODE BEGIN 4 */
/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */

void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
```

```
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */

void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}

#endif /* USE_FULL_ASSERT */
/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

Sau khi nạp code, các bạn ấn nút nhấn trên KIT để bật sáng đèn Led, hiện tượng chúng ta thấy là Led vẫn sáng mãi liên tục chứng tỏ chương trình không bị reset. Đó là bởi vì chúng ta liên tục reload giá trị Counter ở trong while(1) để ngăn việc counter giảm về 0x0000.

Tiếp theo chúng ta sẽ comment dòng code sau trong while(1)

```
  while (1)
  {
    /* USER CODE END WHILE */
                //IWDG->KR = 0xAAAA; // Reload Counter prevents watchdog reset
    /* USER CODE BEGIN 3 */
  }
```

Build và nạp lại chương trình, sau khi bấm nút PA0 đèn Led sáng lên sau đó bị tắt do chương trình bị reset. Đó là bởi vì Counter đã đếm về 0x0000 và tạo ra 1 reset mềm.

Vậy là chúng ta hiểu về chức năng của bộ Watchdog Timer rồi !!! Chúc các bạn thành công. Mọi thắc mắc xin gửi về truonggiangbkak58@gmail.com

---

📌 Group: Deviot - Cùng nhau học lập trình IOT

📌 Hotline: 0969.666.522

📌 Address: Số 101C, Xã Đàn 2

📌 Đào tạo thật, học thật, làm thật