

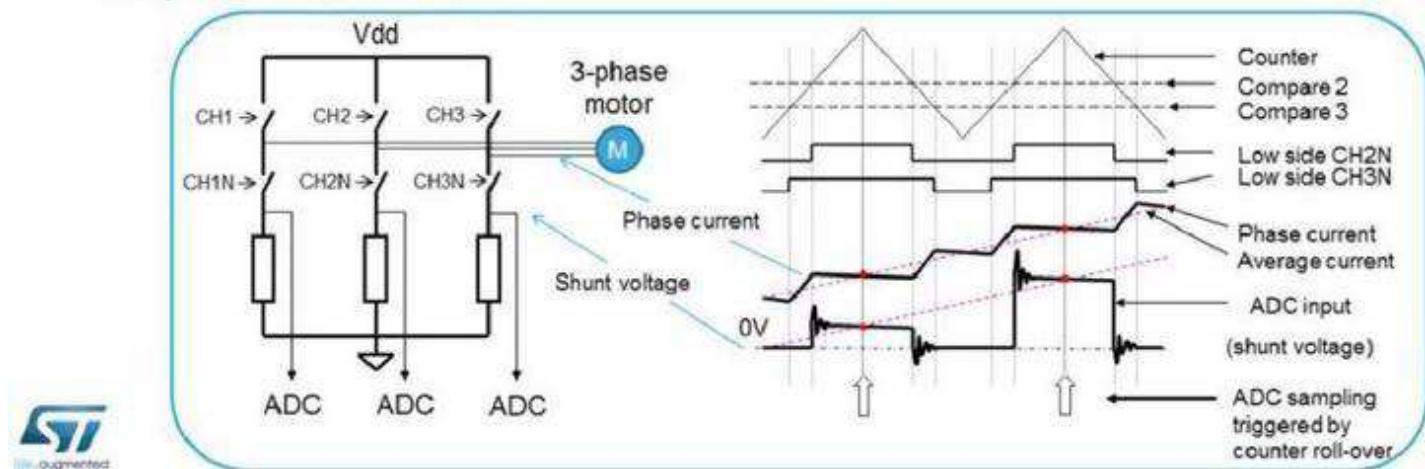
Hi everyone,

I want to measure 3 current sensor in a 3 phase system,I have started 6channel complimentary PWM for 6 IGBTs with deadtime,now I want to trigger injected ADCs on the overflow of Timer1 to synchronize ADC with Timer1 with TRGO2

and I want to use discontinue mode of injected ADC to scan one channel in each trigger,I have configure the Nucleo-144-STM32F746ZG with STM32CubeMX but I don't know that whether the configuration is wrong or my code,when I run the program,It just read ADC1 Ch3 which is in RANK1 and the others don't work.

would you please help me to understand who to configure the timer and ADC to aim this goal or find my problem in my code.

- In 3-phase motor control applications, an ADC trigger on counter overflow allows you to obtain the average current value and avoids noisy ADC conversions



Injected group

This mode is enabled by setting the JDISCEN bit in the ADC_CR1 register. It can be used to convert the sequence selected in the ADC_JSQR register, channel by channel, after an external trigger event.

When an external trigger occurs, it starts the next channel conversions selected in the ADC_JSQR registers until all the conversions in the sequence are done. The total sequence length is defined by the JL[1:0] bits in the ADC_JSQR register.

Example:

n = 1, channels to be converted = 1, 2, 3

1st trigger: channel 1 converted

2nd trigger: channel 2 converted

3rd trigger: channel 3 converted and JEOC event generated

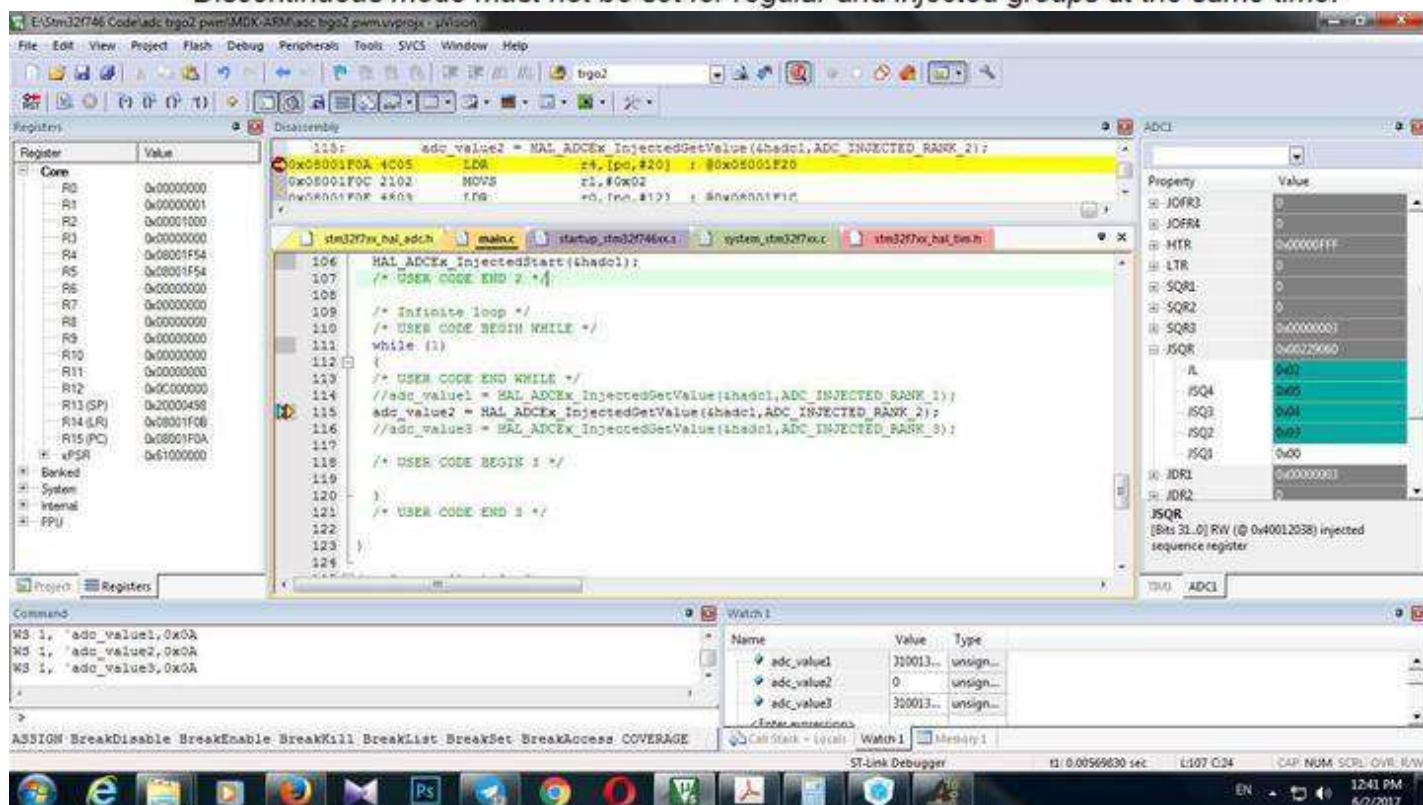
4th trigger: channel 1

Note:

When all injected channels are converted, the next trigger starts the conversion of the first injected channel. In the example above, the 4th trigger reconverts the 1st injected channel 1.

It is not possible to use both the auto-injected and discontinuous modes simultaneously.

Discontinuous mode must not be set for regular and injected groups at the same time.



```
/*
 ****
 * File Name      : main.c
 * Description    : Main program body
 ****
 ** This notice applies to any and all portions of this file
 * that are not between comment pairs USER CODE BEGIN and
 * USER CODE END. Other portions of this file, whether
 * inserted by the user or by software development tools
 * are owned by their respective copyright owners.
 *
 * COPYRIGHT(c) 2017 STMicroelectronics
 *
```

* Redistribution and use in source and binary forms, with or without modification,
* are permitted provided that the following conditions are met:
* 1. Redistributions of source code must retain the above copyright notice,
* this list of conditions and the following disclaimer.
* 2. Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
* 3. Neither the name of STMicroelectronics nor the names of its contributors
* may be used to endorse or promote products derived from this software
* without specific prior written permission.
*

* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"

* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE

* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE ARE

* DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR
CONTRIBUTORS BE LIABLE

* FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL

* DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
GOODS OR

* SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER

* CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY,

* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
OF THE USE

* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

*

*/

/* Includes -----*/

#include "main.h"

#include "stm32f7xx_hal.h"

/* USER CODE BEGIN Includes */

uint32_t adc_value1 = 0;

uint32_t adc_value2 = 0;

uint32_t adc_value3 = 0;

/* USER CODE END Includes */

/* Private variables -----*/

ADC_HandleTypeDef hadc1;

TIM_HandleTypeDef htim1;

/* USER CODE BEGIN PV */

/* Private variables -----*/

/* USER CODE END PV */

/* Private function prototypes -----*/

void SystemClock_Config(void);

static void MX_GPIO_Init(void);

static void MX_ADC1_Init(void);

static void MX_TIM1_Init(void);

```
void HAL_TIM_MspPostInit(TIM_HandleTypeDef *htim);

/* USER CODE BEGIN PFP */
/* Private function prototypes -----*/
/* USER CODE END PFP */

/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

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();
MX_ADC1_Init();
MX_TIM1_Init();

/* USER CODE BEGIN 2 */
HAL_TIM_Base_Start_IT(&htim1);
HAL_ADCEx_InjectedStart(&hadc1);
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
/* USER CODE END WHILE */
adc_value1 = HAL_ADCEx_InjectedGetValue(&hadc1,ADC_INJECTED_RANK_1);
adc_value2 = HAL_ADCEx_InjectedGetValue(&hadc1,ADC_INJECTED_RANK_2);
adc_value3 = HAL_ADCEx_InjectedGetValue(&hadc1,ADC_INJECTED_RANK_3);

/* USER CODE BEGIN 3 */

}
/* USER CODE END 3 */
}
```

```

/** System Clock Configuration
*/
void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct;
    RCC_ClkInitTypeDef RCC_ClkInitStruct;

    /**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_HSE;
    RCC_OscInitStruct.HSEState = RCC_HSE_ON;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
    RCC_OscInitStruct.PLL.PLLM = 4;
    RCC_OscInitStruct.PLL.PLLN = 216;
    RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
    RCC_OscInitStruct.PLL.PLLQ = 2;
    if(HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
    {
        _Error_Handler(__FILE__, __LINE__);
    }

    /**Activate the Over-Drive mode
    */
    if(HAL_PWREx_EnableOverDrive() != HAL_OK)
    {
        _Error_Handler(__FILE__, __LINE__);
    }

    /**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_DIV4;
    RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

    if(HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_7) != HAL_OK)
    {
        _Error_Handler(__FILE__, __LINE__);
    }

    /**Configure the Systick interrupt time
    */
    HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);

    /**Configure the Systick
    */
    HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);

    /* SysTick_IRQn interrupt configuration */
    HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
}

```

```

}

/* ADC1 init function */
static void MX_ADC1_Init(void)
{
    ADC_ChannelConfTypeDef sConfig;
    ADC_InjectionConfTypeDef sConfigInjected;

    /**Configure the global features of the ADC (Clock, Resolution, Data Alignment and number
    of conversion)
    */
    hadc1.Instance = ADC1;
    hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
    hadc1.Init.Resolution = ADC_RESOLUTION_12B;
    hadc1.Init.ScanConvMode = DISABLE;
    hadc1.Init.ContinuousConvMode = DISABLE;
    hadc1.Init.DiscontinuousConvMode = DISABLE;
    hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
    hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
    hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
    hadc1.Init.NbrOfConversion = 1;
    hadc1.Init.DMAContinuousRequests = DISABLE;
    hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
    if (HAL_ADC_Init(&hadc1) != HAL_OK)
    {
        _Error_Handler(__FILE__, __LINE__);
    }

    /**Configure for the selected ADC regular channel its corresponding rank in the sequencer and
    its sample time.
    */
    sConfig.Channel = ADC_CHANNEL_3;
    sConfig.Rank = 1;
    sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
    if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
    {
        _Error_Handler(__FILE__, __LINE__);
    }

    /**Configures for the selected ADC injected channel its corresponding rank in the sequencer
    and its sample time
    */
    sConfigInjected.InjectedChannel = ADC_CHANNEL_3;
    sConfigInjected.InjectedRank = 1;
    sConfigInjected.InjectedNbrOfConversion = 3;
    sConfigInjected.InjectedSamplingTime = ADC_SAMPLETIME_3CYCLES;
    sConfigInjected.ExternalTrigInjecConvEdge =
    ADC_EXTERNALTRIGINJECCONVEDGE_RISING;
    sConfigInjected.ExternalTrigInjecConv = ADC_EXTERNALTRIGINJECCONV_T1_TRGO2;
    sConfigInjected.AutoInjectedConv = DISABLE;
    sConfigInjected.InjectedDiscontinuousConvMode = ENABLE;
    sConfigInjected.InjectedOffset = 0;
    if (HAL_ADCEx_InjectedConfigChannel(&hadc1, &sConfigInjected) != HAL_OK)
    {
        _Error_Handler(__FILE__, __LINE__);
    }
}

```

```

/**Configures for the selected ADC injected channel its corresponding rank in the sequencer
and its sample time
*/
sConfigInjected.InjectedChannel = ADC_CHANNEL_4;
sConfigInjected.InjectedRank = 2;
if (HAL_ADCEx_InjectedConfigChannel(&hadc1, &sConfigInjected) != HAL_OK)
{
    _Error_Handler(__FILE__, __LINE__);
}

/**Configures for the selected ADC injected channel its corresponding rank in the sequencer
and its sample time
*/
sConfigInjected.InjectedChannel = ADC_CHANNEL_5;
sConfigInjected.InjectedRank = 3;
if (HAL_ADCEx_InjectedConfigChannel(&hadc1, &sConfigInjected) != HAL_OK)
{
    _Error_Handler(__FILE__, __LINE__);
}

/*
/* TIM1 init function */
static void MX_TIM1_Init(void)
{

    TIM_ClockConfigTypeDef sClockSourceConfig;
    TIM_MasterConfigTypeDef sMasterConfig;
    TIM_OC_InitTypeDef sConfigOC;
    TIM_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig;

    htim1.Instance = TIM1;
    htim1.Init.Prescaler = 1-1;
    htim1.Init.CounterMode = TIM_COUNTERMODE_CENTERALIGNED1;
    htim1.Init.Period = 10800;
    htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim1.Init.RepetitionCounter = 0;
    htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
    if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
    {
        _Error_Handler(__FILE__, __LINE__);
    }

    sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
    if (HAL_TIM_ConfigClockSource(&htim1, &sClockSourceConfig) != HAL_OK)
    {
        _Error_Handler(__FILE__, __LINE__);
    }

    if (HAL_TIM_PWM_Init(&htim1) != HAL_OK)
    {
        _Error_Handler(__FILE__, __LINE__);
    }

    sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
    sMasterConfig.MasterOutputTrigger2 = TIM_TRGO2_UPDATE;
    sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_ENABLE;
    if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)
    {

```

```

        _Error_Handler(__FILE__, __LINE__);
    }

sConfigOC.OCMode = TIM_OCMODE_PWM1;
sConfigOC.Pulse = 0;
sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
sConfigOC.OCNPolarity = TIM_OCNPOLARITY_HIGH;
sConfigOC.OCFastMode = TIM_OCFAST_ENABLE;
sConfigOC.OCIdleState = TIM_OCIDLESTATE_RESET;
sConfigOC.OCNIdleState = TIM_OCNIDLESTATE_RESET;
if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
{
    _Error_Handler(__FILE__, __LINE__);
}

if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_2) != HAL_OK)
{
    _Error_Handler(__FILE__, __LINE__);
}

if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_3) != HAL_OK)
{
    _Error_Handler(__FILE__, __LINE__);
}

sBreakDeadTimeConfig.OffStateRunMode = TIM_OSSR_DISABLE;
sBreakDeadTimeConfig.OffStateIDLEMode = TIM_OSSI_DISABLE;
sBreakDeadTimeConfig.LockLevel = TIM_LOCKLEVEL_OFF;
sBreakDeadTimeConfig.DeadTime = 140;
sBreakDeadTimeConfig.BreakState = TIM_BREAK_DISABLE;
sBreakDeadTimeConfig.BreakPolarity = TIM_BREAKPOLARITY_HIGH;
sBreakDeadTimeConfig.BreakFilter = 0;
sBreakDeadTimeConfig.Break2State = TIM_BREAK2_DISABLE;
sBreakDeadTimeConfig.Break2Polarity = TIM_BREAK2POLARITY_HIGH;
sBreakDeadTimeConfig.Break2Filter = 0;
sBreakDeadTimeConfig.AutomaticOutput = TIM_AUTOMATICOUTPUT_DISABLE;
if (HAL_TIMEx_ConfigBreakDeadTime(&htim1, &sBreakDeadTimeConfig) != HAL_OK)
{
    _Error_Handler(__FILE__, __LINE__);
}

HAL_TIM_MspPostInit(&htim1);

}

/** Configure pins as
 * Analog
 * Input
 * Output
 * EVENT_OUT
 * EXTI
PC1 -----> ETH_MDC
PA1 -----> ETH_REF_CLK
PA2 -----> ETH_MDIO
PA7 -----> ETH CRS_DV
PC4 -----> ETH_RXD0

```

```

PC5 -----> ETH_RXD1
PB13 -----> ETH_TXD1
PD8 -----> USART3_TX
PD9 -----> USART3_RX
PA8 -----> USB_OTG_FS_SOF
PA9 -----> USB_OTG_FS_VBUS
PA10 -----> USB_OTG_FS_ID
PA11 -----> USB_OTG_FS_DM
PA12 -----> USB_OTG_FS_DP
PG11 -----> ETH_TX_EN
PG13 -----> ETH_RXD0
*/
static void MX_GPIO_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStruct;

    /* GPIO Ports Clock Enable */
    __HAL_RCC_GPIOC_CLK_ENABLE();
    __HAL_RCC_GPIOH_CLK_ENABLE();
    __HAL_RCC_GPIOA_CLK_ENABLE();
    __HAL_RCC_GPIOB_CLK_ENABLE();
    __HAL_RCC_GPIOE_CLK_ENABLE();
    __HAL_RCC_GPIOD_CLK_ENABLE();
    __HAL_RCC_GPIOG_CLK_ENABLE();

    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(GPIOB, LD1_Pin|LD3_Pin|LD2_Pin, GPIO_PIN_RESET);

    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(USB_PowerSwitchOn_GPIO_Port, USB_PowerSwitchOn_Pin,
                     GPIO_PIN_RESET);

    /*Configure GPIO pin : USER.Btn_Pin */
    GPIO_InitStruct.Pin = USER.Btn_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(USER.Btn_GPIO_Port, &GPIO_InitStruct);

    /*Configure GPIO pins : RMII_MDC_Pin RMII_RXD0_Pin RMII_RXD1_Pin */
    GPIO_InitStruct.Pin = RMII_MDC_Pin|RMII_RXD0_Pin|RMII_RXD1_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
    GPIO_InitStruct.Alternate = GPIO_AF11_ETH;
    HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

    /*Configure GPIO pins : RMII_REF_CLK_Pin RMII_MDIO_Pin RMII_CRS_DV_Pin */
    GPIO_InitStruct.Pin = RMII_REF_CLK_Pin|RMII_MDIO_Pin|RMII_CRS_DV_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
    GPIO_InitStruct.Alternate = GPIO_AF11_ETH;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

    /*Configure GPIO pins : LD1_Pin LD3_Pin LD2_Pin */
    GPIO_InitStruct.Pin = LD1_Pin|LD3_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(GPIOB, &GPIO_InitStruct);

/*Configure GPIO pin : RMII_TXD1_Pin */
GPIO_InitStruct.Pin = RMII_TXD1_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
GPIO_InitStruct.Alternate = GPIO_AF11_ETH;
HAL_GPIO_Init(RMII_TXD1_GPIO_Port, &GPIO_InitStruct);

/*Configure GPIO pins : STLK_RX_Pin STLK_TX_Pin */
GPIO_InitStruct.Pin = STLK_RX_Pin|STLK_TX_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
GPIO_InitStruct.Alternate = GPIO_AF7_USART3;
HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

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

/*Configure GPIO pin : USB_OverCurrent_Pin */
GPIO_InitStruct.Pin = USB_OverCurrent_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(USB_OverCurrent_GPIO_Port, &GPIO_InitStruct);

/*Configure GPIO pins : USB_SOF_Pin USB_ID_Pin USB_DM_Pin USB_DP_Pin */
GPIO_InitStruct.Pin = USB_SOF_Pin|USB_ID_Pin|USB_DM_Pin|USB_DP_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
GPIO_InitStruct.Alternate = GPIO_AF10_OTG_FS;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

/*Configure GPIO pin : USB_VBUS_Pin */
GPIO_InitStruct.Pin = USB_VBUS_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(USB_VBUS_GPIO_Port, &GPIO_InitStruct);

/*Configure GPIO pins : RMII_TX_EN_Pin RMII_TXD0_Pin */
GPIO_InitStruct.Pin = RMII_TX_EN_Pin|RMII_TXD0_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
GPIO_InitStruct.Alternate = GPIO_AF11_ETH;
HAL_GPIO_Init(GPIOG, &GPIO_InitStruct);

}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

```

```
/***
 * @brief This function is executed in case of error occurrence.
 * @param None
 * @retval None
 */
void _Error_Handler(char * file, int line)
{
/* USER CODE BEGIN Error_Handler_Debug */
/* User can add his own implementation to report the HAL error return state */
while(1)
{
}
/* USER CODE END Error_Handler_Debug */
}

#ifndef 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,
   ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
/* USER CODE END 6 */

}

#endif

/***
 * @}
 */

***** (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

Best Regards,

Bastam