

ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA KỸ THUẬT MÁY TÍNH

Nhóm 7:

CHU TIẾN TRỌNG

THỰC TẬP FPT-UIT
TUẦN 2

UART – RETARGET
TIMER

KỸ SƯ NGÀNH KỸ THUẬT MÁY TÍNH

TP. HỒ CHÍ MINH, 2021

MỤC LỤC

I.	UART – RETARGET	3
A.	UART in MCU?	3
1.	What is UART?.....	3
2.	UART Use?	3
3.	Cách kết nối uart	4
B.	RETARGET	5
II.	TIMER	8
A.	Tìm hiểu về TIMER.....	8
1.	Các thành phần cơ bản của TIMER.....	8
2.	Các chế độ hoạt động của Timer:.....	9
B.	Định Nghĩa Struct Cấu Hình Cho TIMER	9
III.	CHƯƠNG TRÌNH	11
A.	Cấu Trúc Thư Mục	11
B.	Source Code	11
1.	Main	11
2.	Makefile	18
IV.	THAM KHẢO:.....	23

I. UART – RETARGET

A. UART in MCU?

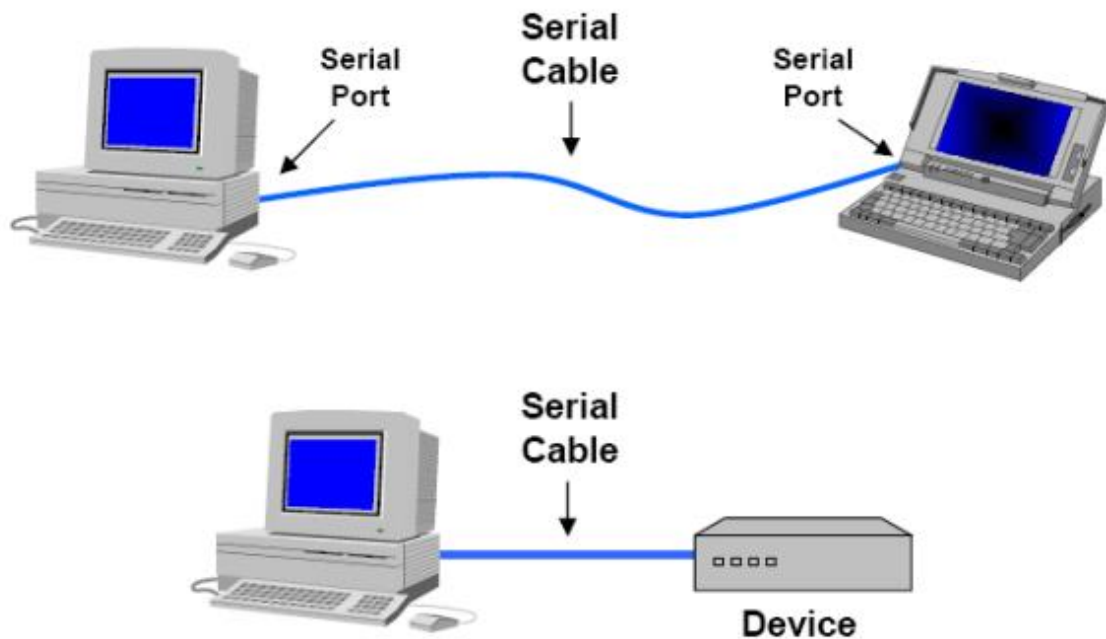
1. What is UART?

- UART – Universal Asynchronous Receiver Transmitter.
- UART là thiết bị ngoại vi trong vi điều khiển, có thể gửi và nhận dữ liệu nối tiếp bất đồng bộ.
- Board cung cấp cho ta 8 giao tiếp UART, được chia thành 2 nhóm:
 - USART1, USART2, USART3, USART6 truyền dữ liệu theo kiểu Synchronous communication
 - UART4, UART5, UART8 truyền dữ liệu theo kiểu Asynchronous communication
- TX và RX là tương ứng với chân truyền và nhận dữ liệu board.
- VD: USART1 có 2 bộ truyền nhận là PA9 – PA10 và PB6 – PB7

U(S)ARTx	Pins pack 1		Pins pack 2		Pins pack 3		APB
	TX	RX	TX	RX	TX	RX	
USART1	PA9	PA10	PB6	PB7			2
USART2	PA2	PA3	PD5	PD6			1
USART3	PB10	PB11	PC10	PC11	PD8	PD9	1
UART4	PA0	PA1	PC10	PC11			1
UART5	PC12	PD2					1
USART6	PC6	PC7	PG14	PG9			2
UART7	PE8	PE7	PF7	PF6			1
UART8	PE1	PE0					1

2. UART Use?

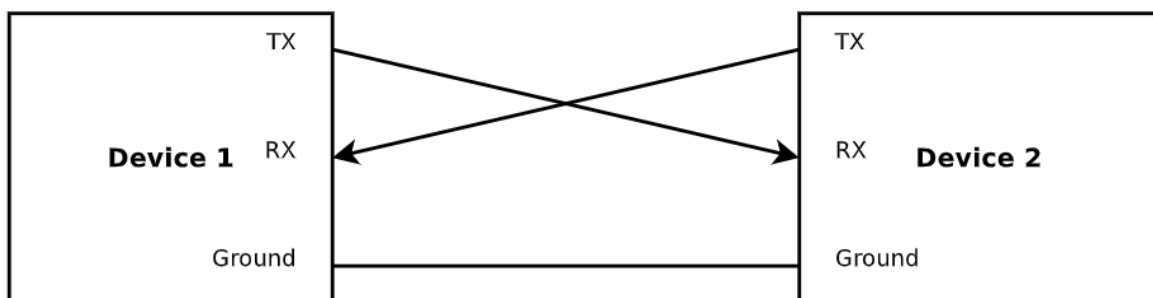
- Cổng nối tiếp PC là một UART.



- Giao tiếp giữa các máy tính ở xa
- Được sử dụng phổ biến để truy cập internet.

3. Cách kết nối uart

- TX -> RX
- RX -> TX
- GND -> GND



- Có 3 phương thức để truyền dữ liệu nối tiếp:
- Sử dụng Polling - HAL_UART_Transmit
 - Sử dụng ngắt (Interrupt) – HAL_UART_Transmit_IT
 - Sử dụng DMA - HAL_UART_Transmit_DMA

B. RETARGET

- Để xuất hàm printf vào cổng Serial, bạn cần chuyển output của hàm fputc sang cổng Serial (redirect).
- Thêm các dòng trong hình dưới vào file main.c

```
#ifdef __GNUC__
#define PUTCHAR_PROTOTYPE int __io_putchar(int ch)
#else
#define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)
#endif /* __GNUC__ */

PUTCHAR_PROTOTYPE
{
    HAL_UART_Receive(&huart1, (uint8_t *)&ch, 1, HAL_MAX_DELAY);
    HAL_UART_Transmit(&huart1, (uint8_t *)&ch, 1, HAL_MAX_DELAY);
    return ch;
}
```

- Thêm Retaget.c

Code:

```
#include <_ansi.h>
#include <_syslist.h>
#include <errno.h>
#include <sys/time.h>
#include <sys/times.h>
#include <limits.h>
#include <signal.h>
#include <../Inc/retarget.h>
#include <stdint.h>
#include <stdio.h>

#if !defined(OS_USE_SEMIHOSTING)

#define STDIN_FILENO 0
#define STDOUT_FILENO 1
#define STDERR_FILENO 2

UART_HandleTypeDef *gHuart;

void RetargetInit(UART_HandleTypeDef *huart) {
    gHuart = huart;

    /* Disable I/O buffering for STDOUT stream, so that
```

```

    * chars are sent out as soon as they are printed. */
    setvbuf(stdout, NULL, _IONBF, 0);
}

int _isatty(int fd) {
    if (fd >= STDIN_FILENO && fd <= STDERR_FILENO)
        return 1;

    errno = EBADF;
    return 0;
}

int _write(int fd, char* ptr, int len) {
    HAL_StatusTypeDef hstatus;

    if (fd == STDOUT_FILENO || fd == STDERR_FILENO) {
        hstatus = HAL_UART_Transmit(gHuart, (uint8_t *) ptr, len, HAL_MAX_DELAY);
        if (hstatus == HAL_OK)
            return len;
        else
            return EIO;
    }
    errno = EBADF;
    return -1;
}

int _close(int fd) {
    if (fd >= STDIN_FILENO && fd <= STDERR_FILENO)
        return 0;

    errno = EBADF;
    return -1;
}

int _lseek(int fd, int ptr, int dir) {
    (void) fd;
    (void) ptr;
    (void) dir;

    errno = EBADF;
    return -1;
}

int _read(int fd, char* ptr, int len) {
    HAL_StatusTypeDef hstatus;

```

```
if (fd == STDIN_FILENO) {
    hstatus = HAL_UART_Receive(gHuart, (uint8_t *) ptr, 1, HAL_MAX_DELAY);
    if (hstatus == HAL_OK)
        return 1;
    else
        return EIO;
}
errno = EBADF;
return -1;
}

int _fstat(int fd, struct stat* st) {
    if (fd >= STDIN_FILENO && fd <= STDERR_FILENO) {
        st->st_mode = S_IFCHR;
        return 0;
    }

    errno = EBADF;
    return 0;
}

#endif // #if !defined(OS_USE_SEMIHOSTING)
```

II. TIMER

A. Tìm hiểu về TIMER

- Timer trong STM32 có rất nhiều chức năng chẳng hạn như bộ đếm counter, PWM, input capture ngoài ra còn một số chức năng đặt biệt để điều khiển động cơ như encoder, hall sensors.
- Timer là bộ định thời có thể sử dụng để tạo ra thời gian cơ bản dựa trên các thông số: clock, prescaler, autoreload, repetition counter. Timer của STM32 là timer 16 bits có thể tạo ra các sự kiện trong khoảng thời gian từ nano giây tới vài phút gọi là UEV (update event).

TIMER	TYPE	RESOLUTION	PRESCALER	CHANNELS	MAX INTERFACE CLOCK	MAX TIMER CLOCK*	APB
TIM1, TIM8	Advanced	16bit	16bit	4	SysClk/2	SysClk	2
TIM2, TIM5	General purpose	32bit	16bit	4	SysClk/4	SysClk, SysClk/2	1
TIM3, TIM4	General purpose	16bit	16bit	4	SysClk/4	SysClk, SysClk/2	1
TIM9	General purpose	16bit	16bit	2	SysClk/2	SysClk	2
TIM10, TIM11	General purpose	16bit	16bit	1	SysClk/2	SysClk	2
TIM12	General purpose	16bit	16bit	2	SysClk/4	SysClk, SysClk/2	1
TIM13, TIM14	General purpose	16bit	16bit	1	SysClk/4	SysClk, SysClk/2	1
TIM6, TIM7	Basic	16bit	16bit	0	SysClk/4	SysClk, SysClk/2	1

1. Các thành phần cơ bản của TIMER

- **TIM_CLK**: clock cung cấp cho timer.
- **PSC (prescaler)**: là thanh ghi 16bits làm bộ chia cho timer, có thể chia từ 1 tới 65535
- **ARR (auto-reload register)**: là giá trị đếm của timer (16bits hoặc 32bits).
- **RCR (repetition counter register)**: giá trị đếm lặp lại 16bits.
- Giá trị UEV được tính theo công thức sau:

$$\text{UEV} = \text{TIM_CLK} / ((\text{PSC} + 1) * (\text{ARR} + 1) * (\text{RCR} + 1))$$

Ví dụ: TIM_CLK = 72MHz, PSC = 1, ARR = 65535, RCR = 0.

$$\text{UEV} = 72000000 / ((1 + 1) * (65535 + 1) * (1)) = 549.3 \text{ Hz}$$

Với giá trị ARR (auto-reload) được cung cấp thì bộ định thời (timer) thực hiện các chế độ đếm khác nhau: đếm lên, đếm xuống hoặc kết hợp cả 2. Khi thực hiện đếm lên thì giá trị bộ đếm bắt đầu từ 0 và đếm tới giá trị ARR-1 thì báo tràn. Khi thực hiện đếm xuống thì bộ đếm bắt đầu từ giá trị ARR đếm xuống 1 thì báo tràn.

2. Các chế độ hoạt động của Timer:

- Input capture
- Output compare
- PWM generation (Edge- and Center-aligned modes)
- One-pulse mode output

B. Định Nghĩa Struct Cấu Hình Cho TIMER

```
typedef struct
{
    uint16_t TIM_Prescaler;          /*!< Specifies the prescaler value used to divide the TIM clock.
                                     This parameter can be a number between 0x0000 and 0xFFFF.*/

    uint16_t TIM_CounterMode;        /*!< Specifies the counter mode.
                                     This parameter can be a value of @ref TIM_Counter_Mode.*/

    uint32_t TIM_Period;             /*!< Specifies the period value to be loaded into the active
                                     Auto-Reload Register at the next update event.
                                     This parameter must be a number between 0x0000 and 0xFFFF.*/

    uint16_t TIM_ClockDivision;      /*!< Specifies the clock division.
                                     This parameter can be a value of @ref TIM_Clock_Division.*/

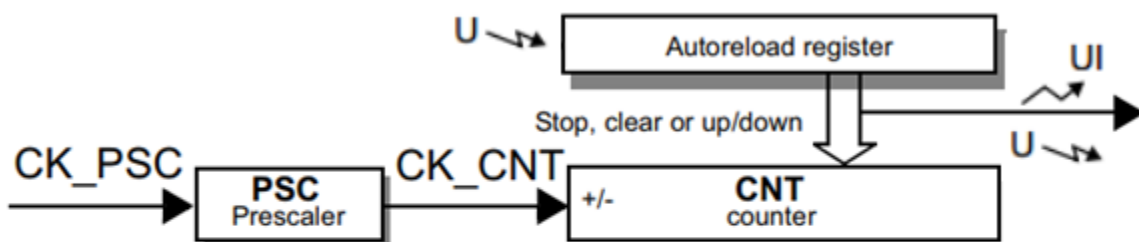
    uint8_t TIM_RepetitionCounter;    /*!< Specifies the repetition counter value. Each time the
                                     counter reaches zero, an update event is generated and counter
                                     is reloaded from the RCR value (N).
                                     This means in PWM mode that (N+1) corresponds to:
                                     - the number of PWM periods in edge-aligned mode
                                     - the number of half PWM period in center-aligned mode
                                     This parameter must be a number between 0x00 and 0xFF.
                                     @note This parameter is valid only for TIM1 and TIM8.*/
} TIM_TimeBaseInitTypeDef;
```

- **TIM_Prescaler** : Tham số TIM_Prescaler hiểu đơn giản như một bộ chia tần số.

$$\text{TIM_Prescaler} = ((\text{SystemCoreClock}/n)/\text{Fc_timer})-1$$

- ❖ **Notes** : Tùy vào timer nào mà chỉ số chia n sẽ khác nhau. Ví dụ trong stm32f4 gồm có những timer và hệ số chia khác nhau như hình bên dưới :

- **TIM_CounterMode** : Thiết lập mode cho timer là đếm lên hay đếm xuống. Nếu chọn mode đếm tăng có nghĩa là mỗi xung nhịp timer, bộ đếm counter sẽ tự tăng lên một giá trị theo chiều dương cho đến khi nào bằng giá trị period sẽ đếm lại từ đầu, người ta thường gọi trường hợp này là tràn bộ đếm.
- **TIM_Period** : Period có nghĩa là chu kỳ của timer (không phải là chu kỳ của 1 xung clock timer).
- ❖ **Notes** : Khi cấu hình sử dụng timer ta cần quan tâm đến 3 yếu tố chính đó là :
 - Đếm với xung clock timer là bao nhiêu (Fc_timer – xác định qua TIM_Prescaler).
 - Đếm lên hay đếm xuống (TIM_CounterMode).
 - Đếm đến bao nhiêu (TIM_Period).












⚡ Event

⚡ Interrupt & DMA output

III. CHƯƠNG TRÌNH

A. Cấu Trúc Thư Mục

 .vscode	29/05/2021 11:53 SA	File folder	
 build	29/05/2021 3:45 CH	File folder	
 Core	29/05/2021 11:53 SA	File folder	
 Drivers	29/05/2021 11:53 SA	File folder	
 .mxproject	29/05/2021 11:53 SA	MXPROJECT File	7 KB
 FPT-UIT-Week2.ioc	29/05/2021 11:53 SA	STM32CubeMX	5 KB
 Makefile	29/05/2021 11:55 SA	File	6 KB
 startup_stm32f429xx.s	19/05/2021 4:47 CH	S File	25 KB
 STM32F429ZITx_FLASH.ld	29/05/2021 11:53 SA	LD File	7 KB

B. Source Code

1. Main

Code:

```
#include "main.h"

/* Private includes -----*/
#include <string.h>
#include <stdio.h>
#include "stdint.h"
#include "retarget.h"

/* Private variables -----*/
TIM_HandleTypeDef htim2;
UART_HandleTypeDef huart1;

char Rx_data[2];
uint8_t cmd =0;
char mainmsg[] = "Press:  a: Name    b: Button Led  \n";
char msg[] = "Press :   0: All Led OFF  1: Led Green ON   2: Led Red ON    3: Timer Led";
char err[] = "Invalid input! Please try again!\n";

uint8_t button =0;

/* Private function prototypes -----*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM2_Init(void);
```

```

static void MX_USART1_UART_Init(void);
/* USER CODE BEGIN PFP */

void RetargetName(void)
{
    char buff[100];
    printf("\nYour name: ");
    scanf("%s", buff);
    printf("\nHello, %s!\n", buff);
}

void toggleled(void)
{
    while(1)
    {
        /* USER CODE END WHILE */
        HAL_GPIO_TogglePin(GPIOD,GPIO_PIN_13|GPIO_PIN_14);
        HAL_Delay(100);
        // if(cmd == (int)27)
        // {
        //     AdvanceLed();
        // }
    }
}

void AdvanceLed()
{
    printf(msg);
    while(1)
    {
        if(HAL_UART_Receive(&huart1, &cmd, 1, 100)== HAL_OK)
        {
            // errmsg();
            switch(cmd)
            {
                case '0':
                    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13|GPIO_PIN_14, GPIO_PIN_RESET);
                    printf("\nAll led OFF!");
                    break;
                case '1':
                    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_SET);
                    printf("\nLed green ON!");
                    break;
            }
        }
    }
}

```

```

        case '2':
            HAL_GPIO_WritePin(GPIOG, GPIO_PIN_14, GPIO_PIN_SET);
            printf("\nLed red ON!");
            break;
        case '3':
            printf("\nNhap nhay!");
            toggleled();

    }
}
}
}

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    // Check which version of the timer triggered this callback and toggle LED
    UNUSED(htim);

    if (htim == &htim2 )
    {
        HAL_GPIO_TogglePin(GPIOG, GPIO_PIN_13);
    }
}

// void Timer(void)
// {

//     // void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
//     // {
//     //     UNUSED(htim);

//     //     if (htim == &htim2 )
//     //     {
//     //         HAL_GPIO_TogglePin(GPIOG, GPIO_PIN_13);
//     //     }
//     // }
//     HAL_TIM_PeriodElapsedCallback(&htim2);

// }

int main(void)
{
    HAL_Init();

    SystemClock_Config();

```

```

MX_GPIO_Init();
MX_TIM2_Init();
MX_USART1_UART_Init();

RetargetInit(&huart1);
HAL_TIM_Base_Start_IT(&htim2);

printf(mainmsg);

while (1)
{
    button = HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0);
    // uint8_t x=HAL_UART_Receive(&huart1, &cmd, 1, 100);
    if(HAL_UART_Receive(&huart1, &cmd, 1, 100)== HAL_OK)
    {
        switch(cmd)
        {
            case 'a':
                RetargetName();
                break;
            case 'b':
                AdvanceLed();
                break;
            // case 'c':
            //     Timer();
            default:
                printf(err);
        }
    }
}

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_SCALE3);
    /** Initializes the RCC Oscillators according to the specified parameters
    * in the RCC_OscInitTypeDef structure.

```

```

*/
RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
RCC_OscInitStruct.HSISState = RCC_HSI_ON;
RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
{
    Error_Handler();
}
/** Initializes the CPU, AHB and APB buses clocks
*/
RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
{
    Error_Handler();
}
}

static void MX_TIM2_Init(void)
{
    /* USER CODE BEGIN TIM2_Init 0 */

    /* USER CODE END TIM2_Init 0 */

    TIM_ClockConfigTypeDef sClockSourceConfig = {0};
    TIM_MasterConfigTypeDef sMasterConfig = {0};

    // Ft = default freq / prescaler
    // T = 1/ Ft

    /* USER CODE END TIM2_Init 1 */
    htim2.Instance = TIM2;
    htim2.Init.Prescaler = 16000;
    htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim2.Init.Period = 250;
    htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
    if (HAL_TIM_Base_Init(&htim2) != HAL_OK)

```

```

{
    Error_Handler();
}
sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
{
    Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_UPDATE;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN TIM2_Init 2 */

/* USER CODE END TIM2_Init 2 */
}

static void MX_USART1_UART_Init(void)
{
    /* USER CODE END USART1_Init 1 */
    huart1.Instance = USART1;
    huart1.Init.BaudRate = 9600;
    huart1.Init.WordLength = UART_WORDLENGTH_8B;
    huart1.Init.StopBits = UART_STOPBITS_1;
    huart1.Init.Parity = UART_PARITY_NONE;
    huart1.Init.Mode = UART_MODE_TX_RX;
    huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
    huart1.Init.OverSampling = UART_OVERSAMPLING_16;
    if (HAL_UART_Init(&huart1) != HAL_OK)
    {
        Error_Handler();
    }
    /* USER CODE BEGIN USART1_Init 2 */

    /* USER CODE END USART1_Init 2 */
}

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

```



```

/* GPIO Ports Clock Enable */
__HAL_RCC_GPIOH_CLK_ENABLE();
__HAL_RCC_GPIOA_CLK_ENABLE();
__HAL_RCC_GPIOG_CLK_ENABLE();

/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0, GPIO_PIN_RESET);

/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(GPIOG, GPIO_PIN_13|GPIO_PIN_14, GPIO_PIN_RESET);

/*Configure GPIO pin : PA0 */
GPIO_InitStruct.Pin = GPIO_PIN_0;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

/*Configure GPIO pins : PG13 PG14 */
GPIO_InitStruct.Pin = GPIO_PIN_13|GPIO_PIN_14;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOG, &GPIO_InitStruct);
}

void Error_Handler(void)
{
    /* USER CODE BEGIN Error_Handler_Debug */
    /* User can add his own implementation to report the HAL error return state */
    __disable_irq();
    while (1)
    {
    }
    /* USER CODE END Error_Handler_Debug */
}

#ifdef __GNUC__
#define PUTCHAR_PROTOTYPE int __io_putchar(int ch)
#else
#define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)
#endif /* __GNUC__ */

PUTCHAR_PROTOTYPE

```

```

{
    HAL_UART_Receive(&huart1, (uint8_t *)&ch, 1, HAL_MAX_DELAY);
    HAL_UART_Transmit(&huart1, (uint8_t *)&ch, 1, HAL_MAX_DELAY);
    return ch;
}

#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,
       ex: 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*****/

```

2. Makefile

Code:

```

#####
# File automatically-generated by tool: [projectgenerator] version: [3.13.0-B3] date: [Sat Ma
# target
#####
TARGET = FPT-UIT-Week2
#####
# debug build?
DEBUG = 1
# optimization
OPT = -Og
#####

```

```

# Build path
BUILD_DIR = build

#####
# source
SOURCES_DIR = Drivers/STM32F4xx_HAL_Driver/Src
SOURCES_DIR += Core/Src
#####
C_SOURCES := $(foreach SOURCES_DIR,$(SOURCES_DIR),$(wildcard $(SOURCES_DIR)/*))

# ASM sources
ASM_SOURCES = \
startup_stm32f429xx.s
#####
PREFIX = arm-none-eabi-
# The gcc compiler bin path can be either defined in make command via GCC_PATH variable (> ma
# either it can be added to the PATH environment variable.
ifdef GCC_PATH
CC = $(GCC_PATH)/$(PREFIX)gcc
AS = $(GCC_PATH)/$(PREFIX)gcc -x assembler-with-cpp
CP = $(GCC_PATH)/$(PREFIX)objcopy
SZ = $(GCC_PATH)/$(PREFIX)size
else
CC = $(PREFIX)gcc
AS = $(PREFIX)gcc -x assembler-with-cpp
CP = $(PREFIX)objcopy
SZ = $(PREFIX)size
endif
HEX = $(CP) -O ihex
BIN = $(CP) -O binary -S

#####
# CFLAGS
#####
# cpu
CPU = -mcpu=cortex-m4

# fpu
FPU = -mfpu=fpv4-sp-d16

# float-abi
FLOAT-ABI = -mfloat-abi=hard

# mcu
MCU = $(CPU) -mthumb $(FPU) $(FLOAT-ABI)

```

```

# macros for gcc
# AS defines
AS_DEFS =

# C defines
C_DEFS = \
-DUSE_HAL_DRIVER \
-DSTM32F429xx

# AS includes
AS_INCLUDES =

# C includes
C_INCLUDES = \
-ICore/Inc \
-IDrivers/STM32F4xx_HAL_Driver/Inc \
-IDrivers/STM32F4xx_HAL_Driver/Inc/Legacy \
-IDrivers/CMSIS/Device/ST/STM32F4xx/Include \
-IDrivers/CMSIS/Include

# compile gcc flags
ASFLAGS = $(MCU) $(AS_DEFS) $(AS_INCLUDES) $(OPT) -Wall -fdata-sections -ffunction-sections

CFLAGS = $(MCU) $(C_DEFS) $(C_INCLUDES) $(OPT) -Wall -fdata-sections -ffunction-sections

ifeq ($(DEBUG), 1)
CFLAGS += -g -gdwarf-2
endif

# Generate dependency information
CFLAGS += -MMD -MP -MF"$(@:%.o=%.d)"

#####
# LDFLAGS
#####
# link script
LDSCRIPT = STM32F429ZITx_FLASH.ld

# libraries
LIBS = -lc -lm -lnosys

```

```

LIBDIR =
LDFLAGS = $(MCU) -specs=nano.specs -T$(LDSO) $(LIBDIR) $(LIBS) -Wl,-Map=$(BUILD_DIR)/$(TA
sections

# default action: build all
all: $(BUILD_DIR)/$(TARGET).elf $(BUILD_DIR)/$(TARGET).hex $(BUILD_DIR)/$(TARGET).bin

#####
# build the application
#####
# list of objects
OBJECTS = $(addprefix $(BUILD_DIR)/,$(notdir $(C_SOURCES:.c=.o)))
vpath %.c $(sort $(dir $(C_SOURCES)))
# list of ASM program objects
OBJECTS += $(addprefix $(BUILD_DIR)/,$(notdir $(ASM_SOURCES:.s=.o)))
vpath %.s $(sort $(dir $(ASM_SOURCES)))

$(BUILD_DIR)/%.o: %.c Makefile | $(BUILD_DIR)
    $(CC) -c $(CFLAGS) -Wa,-a,-ad,-alms=$(BUILD_DIR)/$(notdir $(<:.c=.lst)) $< -o $@

$(BUILD_DIR)/%.o: %.s Makefile | $(BUILD_DIR)
    $(AS) -c $(CFLAGS) $< -o $@

$(BUILD_DIR)/$(TARGET).elf: $(OBJECTS) Makefile
    $(CC) $(OBJECTS) $(LDLAGS) -o $@
    $(SZ) $@

$(BUILD_DIR)/%.hex: $(BUILD_DIR)/%.elf | $(BUILD_DIR)
    $(HEX) $< $@

$(BUILD_DIR)/%.bin: $(BUILD_DIR)/%.elf | $(BUILD_DIR)
    $(BIN) $< $@

$(BUILD_DIR):
    mkdir $@

#####
# clean up
#####
clean:
    -rm -fR $(BUILD_DIR)

#####
# dependencies

```

```
#####  
-include $(wildcard $(BUILD_DIR)/*.d)  
  
# *** EOF ***
```

IV. THAM KHẢO:

<https://www.dmi.unict.it/santoro/teaching/lsm/slides/UART.pdf>

[UART | Serial Communication With PIC Microcontrollers Tutorial \(deepbluembedded.com\)](#)

[How to Retarget/Redirect printf & scanf to UART in KEIL \(ocfreaks.com\)](#)

[DEVZONE](#)

[Tìm hiểu và sử dụng timer trên STM32F411 - TAPIT](#)

[Hướng dẫn lập trình Timer với STM32 | Embedded System \(hethongnhung.com\)](#)