STM32F103C–FREERTOS-PLATFORM

TUT2: FREERTOS, User Task, IO Task

AT command, Watchdog

Address: 666/49 3 Thang 2, Ward 14, District 10, Ho Chi Minh City, Vietnam

Phone: +84986035797 | Email: [sonhai26988@gmail.com](mailto:sonhai26988@gmail.com)

# Công việc:

* Add Watchdog Function
* Build ATPN đơn giản để test hiệu quả của tập lệnh AT 2 chiều.
* Build IO Task để đọc nút nhấn và control LED.

# Ý tưởng:

* Sử dụng IDWG: independent watchdog, là watchdog chuyên dùng khi bị fail chương trình: con trỏ ở Hardfault, IDWG sẽ ko reload counter, dẫn đến watchdog làm reset MCU. Việc dùng watchdog rất cần thiết cho việc detect lỗi và debug.
* ATPN: AT Phi Navigator: là tập lệnh dùng để check các sản phẩm của a Phi
* Port printf function để xuất UART cho việc kiểm tra dữ liệu dễ dàng. Port vào UART1
* IO Task sẽ có piority cao trong hệ thống: nhiệm vụ của IO task là quét các output khi nhận được 1 schedule update của task khác, tương tác vào các struct để quản lý IO và quét các Input để detect sự kiện như nút nhấn.

# Thực hiện, các hàm,và cách sử dụng:

## Watchdog:

### Các file sử dụng:

Watchdog\_Function.c: tầng driver cho các hàm sử dụng: dùng init, truyền nhận data…

Watchdog\_Function.h:

Watchdog\_Function\_Profile.h: define cho các chế độ watchdog muốn sử dụng

### Cách thực hiện:

#include "Watchdog\_Function.h" vào file “Common.h”

Trong ngắt timer timebase bỏ thêm define watchdog sử dụng, và thêm đoạn code để reload counter.

Có thể reload counter trong task, để chắc chắn task đó ko bị treo, khi đó bỏ reload trong timer

## ATPN

### Các file sử dụng:

ComFunction.c: các hàm quản lý cổng com của tập lệnh

ComFunction.h:

### Cách thực hiện:

Việc quét cổng COM (hàm vComPortProcess ()) sẽ diễn ra trong User\_Task hiện thực 10ms tương ứng 10 cycle click 1 lần. Khi nhận đủ frame, qui định bằng việc kết thúc trong 1 interval ko có dữ liệu sẽ process buffer nhận được, ở đây qui định là 10ms.

Sau khi Parse ra sẽ feedback command và process lên cổng UART.

Có thể dùng các tập lệnh để config, ghi flash, điều khiển các IO, set giá trị cho các parameter.

Lưu ý trong demo này sử dụng chỉ 1 UART, có thể chia ra 2 uart để vừa làm debug và làm config riêng biệt.

## IO Task

### Các file sử dụng:

IO\_Function.c:

IO\_Function.h:

IO\_Function\_Profile.h:

IO\_Kernel\_Function.c:

IO\_Kernel\_Function.h:

IO\_Kernel\_Task.c:

IO\_Kernel\_Task.h:

### Cách thực hiện:

Tạo vIO\_Kernel\_Task với priority cao, trong vIO\_Kernel\_Task sẽ có 2 timer, quản lý Input và Output dựa trên thư viện manage của timer timebase

tIO\_Input\_Task: chu kì 5ms, 5 tick

tIO\_Ouput\_Task: chu kì 5ms, 5 tick

Trong fIO\_Input\_Task sẽ sử dụng các hàm đặc thù để quét nút nhấn. Giải thuật quét nút nhấn được thực hiện qua việc quét vGetIOButtonValue() và cập nhật vào biến toàn cục:

strIO\_Button\_Value gồm: trạng thái nút, thời gian nhấn nút, cờ cập nhật trạng thái. Hàm này được call bởi IO task mỗi 5ms

Trong fIO\_Output\_Task sẽ quản lý việc xuất IO, như led, relay, còi, tùy đáp ứng về tần số để config cho các đối tượng khác nhau, fOutput sẽ thực hiện vIO\_Output để quét các parameter cho việc xuất ngõ ra. Các task cập nhật parameter qua hàm: vIO\_ConfigOutput()

Hàm vIO\_ConfigOutput có sử dụng cờ để ưu tiên, chèn ngang process đang thực hiện hoặc ko?

enumbool vIO\_ConfigOutput(structIO\_Manage\_Output \*pOutput, uint32 uFrequency, uint32 uTimeline, uint32 uCountToggle, enumbool bStartState, enumbool bEndState, enumbool bFlagInterrupt):

pOutput: pointer của IO cần toggle

uFrequency: tần số toggle

uTimeline: thời gian đáp ứng

uCountToggle: số lần toggle

bStartState: trạng thái bắt đầu

bEndState: trạng thái kế thúc

bFlagInterrupt: cờ cho phép chen ngang chu trình

# DEMO TUT2:

* Firmware sử dụng watchdog giống bootloader để reload
* Call 2 task: User\_Task và IO\_Task
* User\_Task sẽ quản lý cổng UART1 thực hiện các command từ máy tính truyền xuống, và set parameter cho IO\_Task
* IO\_Task: thực hiện các thao tác In, Out digital và đợi update parameter từ User Task
* Các tập lệnh hỗ trợ:
  + ATPN+?: lệnh hỏi thông tin board. Trong demo sẽ trả lên: product, productID, HardwareID, BootloaderID, FirmwareID…
  + P2TCMD\_SET\_LED\_DIRECT: Set lên chân PC13
    - ATPN+$00$C4$00$00$00$00$01: Set LED 1 On
    - ATPN+$00$C4$00$00$00$00$01: Set LED 1 Off
  + Update vào struct bLEDConfigCommand các thông số theo thứ tự byte như sau:
    - * bLEDConfigCommand.uCountToggle = UART\_BUFFER\_RX[iUART\_DATA];
      * bLEDConfigCommand.uFrequency = UART\_BUFFER\_RX[iUART\_DATA+1];
      * bLEDConfigCommand.bStartState = UART\_BUFFER\_RX[iUART\_DATA+2];
      * bLEDConfigCommand.bEndState = UART\_BUFFER\_RX[iUART\_DATA+3];
      * bLEDConfigCommand.bFlagStart = UART\_BUFFER\_RX[iUART\_DATA+4];
    - Vd:
      * ATPN+$00$C2$00$00$00$00$08$64$01$00$00:
        + 0xC2: SET LED PARAMETER
        + 0x08: chớp 4 lần on, 4 lần off
        + 0x64: chu kì 1000ms là 1s, 0x64 = 100
        + 0x10: bStartState là 1
        + 0x10: bEndState là 0
        + bFlagStart: có interrupt hay ko?
* Khóa Task LED1\_Task bằng cách thêm “//” :

//OS\_xTaskCreate(LED1\_Task, "LED1\_Task", LED1\_TASK\_STACK\_SIZE, NULL, LED1\_TASK\_PRIORITY, &xLED1\_Task\_Handle)