ĐẠI HỌC BÁCH KHOA HÀ NỘI

VIỆN ĐIỆN TỬ VIỄN THÔNG

====o0o====

BÁO CÁO

MÔN: HỆ ĐIỀU HÀNH

(Cá nhân)

***Đề tài*: Nghiên Cứu Hệ Điều Hành Nhúng Thời Gian Thực FREERTOS**

GVHD: TS. Phạm Văn Tiến

Sinh viên: Ngô Gia Tiến 20144468

Lớp : KSTN ĐTVT K59

Hà Nội, ngày 23 tháng 11 năm 2018

CHƯƠNG3. MÔ PHỎNG TRIỂN KHAI HỆ ĐIỀU HÀNH FREERTOS LÊN KIT STM32F103C8

1.1 Mô phỏng triển khai FreeRTOS lên KIT STM32F103C8

1.1.1 Phân tích bài toán mô phỏng

a) Ý tưởng và mục tiêu của bài toán mô phỏng

Bài toán cần đặt ra ở đây là mô phỏng cho hệ điều hành thời gian thực nên các yêu cầu đặt ra cho bài toán phải gắn liền với các đặc điểm của hệ điều hành thời gian thực. Từ đó ta phải đặt ra các mục tiêu trong phần mô phỏng này:

- Làm nổi bật ý nghĩa của việc có hệ điều hành thời gian thực, tức là trong một hệ thống tài nguyên hạn chế, tranh chấp giữa các tác vụ thường xuyên xảy ra. Như vậy ý tưởng bài toán được thiết kế là với cùng một số tác vụ như nhau nếu tăng yêu cầu đáp ứng về thời gian của một số tác vụ lên thì hệ thống sẽ lỗi không thỏa mãn được yêu cầu đặt ra.
- Bài toán mô phỏng được hầu hết các dạng tác vụ của hệ điều hành thời gian thực. Các dạng tác vụ cần mô phỏng như: tác vụ sự kiện, tác vụ theo chu kỳ, tác vụ truyền thông, …
- Nổi bật việc thêm bớt tác vụ vào hệ thống một cách dễ dàng.

b) Bài toán mô phỏng

Từ những ý tưởng và mục tiêu mô phỏng trên. Ta đặt ra bài toán với 2 tác vụ:

- Tác vụ 1: Nháy Led tự động.
  Led (đỏ) nháy tự động với chu kì 1 giây, chân dương nối vào Pin A1, âm nối GND
- Tác vụ 2: Nháy Led khi nhấn nút.
  Led (xanh) sẽ sáng khi bấm nút và tắt khi nhả nút

Hai tác vụ này có thể xảy ra tranh chấp về mặt thời gian, mục đích của mô phỏng là sử dụng hệ điều hành FreeRTOS xử lý tranh chấp này

1.1.2 Các file cần thiết để triển khai FreeRTOS lên vi điều khiển

a) FreeRTOSconfig.h

File được tạo ra với hai nhiệm vụ chính:

- Định nghĩa các thông số, các chức năng cơ bản mà FreeRTOS hỗ trợ được định yêu cầu trong FreeRTOS.h. Các hàm, macro này nếu muốn khai báo có sử dụng thì định nghĩa là 1, ngược lại là 0.
- Các thông số cần định nghĩa cho từng vi điều khiển và từng project cụ thể:
  - *configCPU_CLOCK_HZ*: khai báo tần số làm việc của vi điều khiển theo đơn vị Hz.

- *configTICK_RATE_HZ*: khai báo tần số tick muốn sử dụng, đơn vị Hz.
- *configMAX_PRIORITIES*: giới hạn mức ưu tiên cao nhất được hỗ trợ để lập lịch.
- *configMINIMAL_STACK_SIZE*: giới hạn độ sâu nhỏ nhất của ngăn xếp được dùng cho mỗi task.
- *configTOTAL_HEAP_SIZE*: giới hạn tổng lượng RAM trong heap để cấp phát cho từng nhiệm vụ.
- *configMAX_TASK_NAME_LEN*: giới hạn độ dài của tên các tác vụ, đơn vị tính bằng byte.

b) port.c

Đây là file quan trọng nhất trong việc tạo ra các hàm định nghĩa trong portable.h cho việc port lên vi điều khiển. Các nhiệm vụ chính cụ thể như sau:

- *portTIMER_FOSC_SCALE* = 4: khai báo để cài đặt phần cứng cho tick.
- *portINITAL_INTERRUPT_STATE* = 0xc0: khởi tạo trạng thái cho phép ngắt cho các task được tạo mới. Giá trị này được copy vào *INTCON* khi chuyển task trong lần đầu tiên.
- *portGLOBAL_INTERRUPT_FLAG* = 0x80: định nghĩa này chỉ cho các bit nằm trong *INTCON*, ngắt toàn cục.
- *portINTERRUPTS_UNCHANGED* = 0x00: hằng số được sử dụng cho việc chuyển ngữ cảnh khi yêu cầu ngắt chuyển từ trạng thái cho phép ngắt sang trạng thái không thay đổi khi task vừa bị ngắt khôi phục lại.
- *portCOMPILER_MANAGED_MEMORY_SIZE* = 0x13: một số vùng nhớ cần được lưu lại như một phần của ngữ cảnh tác vụ. Những vùng nhớ này được sử dụng bởi trình dịch cho việc lưu giữ trung gian, đặc biệt là khi thực hiện các phép tình toán học hoặc khi sử dụng dữ lệu 32 bit. Hằng số này định nghĩa độ lớn vùng nhớ phải lưu.
- *vSerialTxISR()* và *vSerialRxISR()*: chương trình phục vụ ngắt cho cổng truyền tin nối tiếp được định nghĩa trong serial.c nhưng vẫn được gọi từ portable, coi như cũng là vector như tick ISR. Trong phần demo cụ thể mà em làm, em đã bỏ phần này đi để làm gọn lõi hệ của hệ điều hành, còn người sử dụng khi có thể tự cài đặt thêm nếu cần.
- *prvSetupTimerInterrupt()*: cài đặt phần cứng để cho phép tick.
- *prvTickISR():* chương trình phục vụ ngắt để duy trì tick và thức hiện chuyển đổi ngữ cảnh tick nếu sử dụng kiểu preemptive.
- *prvLowInterrupt():* chương trình phục vụ ngắt thay thế cho vector mức ưu tiên thấp. Nó gọi những chương trình phục vụ ngắt thích hợp cho các ngắt thực tế.
- Phần quan trọng cũng là khó nhất trong file là lưu và khôi phục ngữ cảnh trong mỗi lần chuyển đổi ngữ cảnh. Đó là hai macro *portSAVE_CONTEXT()* và *portRESTORE_CONTEXT()*. Với macro lưu ngữ cảnh, nó cất tất cả các thanh ghi làm nên ngữ cảnh của tác vụ vào ngăn xếp, sau đó cất đỉnh mới của ngăn xếp này

vào *TCB*. Nếu lời gọi hàm này đến từ *ISR* thì bit cho phép ngắt này đã được set để ISR được gọi. Vì thế ta muốn lưu thanh ghi *INTCON* với các bit đã được set và *ucForcedInterruptFlags*. Điều này có nghĩa là các ngắt sẽ được cho phép trở lại khi task vừa bị ngắt khôi phục. Nếu lời gọi từ thao tác (bằng tay) chuyển ngữ cảnh (ví dụ từ yield) thì ta sẽ lưu *INTCON* với trạng thái hiện thời của nó, và *ucForcedInterruptFlags* phải ở 0. Nó cho phép yield trong vùng bất ly. Ngoài ra, trình dịch thường sử dụng một số vùng ở phía dưới bộ nhớ dùng làm lưu trữ trung gian cho các tính toán. Điều này thực sự đúng khi kiểu dữ liệu 32bit được sử dụng. Các đoạn tmpdata và MATH_DATA phải được lưu trữ như một phần của ngữ cảnh. Macro này sẽ lưu trữ từ địa chỉ 0x00 đến *portCOMPLIER_MANAGED_MEMORY_SIZE*.

- Lưu thanh ghi *WREG* đầu tiên, nó sẽ bị thay đổi ngay trong các thao tác dưới đây.
- Lưu thanh ghi *INTCON* với các bit thích hợp.
- Lưu các thanh ghi cần thiết vào ngăn xếp như: *BSR, FSR2L, FSR2H, …*
- Lưu *.tmpdata* và *MATH_DATA*.
- Lưu con trỏ ngăn xếp phần cứng trong thanh ghi trung gian trước khi tat hay đổi chúng.
- Lưu đỉnh của con trỏ ngăn xếp mềm vào *TCB*.

với macro *portRESTORE_CONTEXT* ta làm gần như ngược lại. Nhưng hết sức chú ý rằng các lưu trữ này đúng với hầu hết các ứng dụng nhưng không phải hoàn toàn. Cần phải kiểm tra lại với từng ứng dụng cụ thể.

- **pxPortInitialiseStack():* cài đặt ngăn xếp của task mới để nó sẵn sàng hoạt động khi bộ lập lịch điều khiển. Các thanh ghi phải được gửi vào ngăn xếp theo thứ tự để port có thể tìm được chúng.
- *xPortStartScheduler():* cài đặt phần cứng sẵn sàng cho bộ lập lịch điều khiển. Nhìn chung là cài đặt cho ngắt tick và cài đặt timer cho tần số đúng của tick. Hàm này được sử dụng ở preemptive (tức là *configUSE_PREEMTIVE* được đặt bằng 1)
- *vPortEndScheduler():* hủy toàn bộ cài đặt cho phần cứng/ISR đã được thực hiện bởi *xPortStartScheduler()* vì thế phần cứng được để lại các điều kiện đầu tiên sau khi bộ lập lịch dừng hoạt động. Hàm này không thể xảy ra trong bộ lập lịch cho port PIC do không thể dừng sau 1 lần chạy.
- *vPortYield():* chuyển ngữ cảnh thủ công. Hàm này giống như chuyển đổi ngữ cảnh tick nhưng không tăng biến đếm tick. Nó phải đúng như chuyển đổi ngữ cảnh tick trong việc lưu trữ vào ngăn xếp của task như thế nào.

c) portmacro.h

File này định nghĩa cho riêng phần port. Các định nghĩa này cấu hình cho FreeRTOS đúng với phần cứng và trình dịch. Các cài đặt này không được biến đổi.

Các nhiệm vụ của file như sau:

- Định nghĩa các kiểu số liệu cơ bản sử dụng trong FreeRTOS, như: char (*portCHAR*), float (*portFLOAT*), int (*portSHORT*), ...
- Kiểm tra xem nếu sử dụng *USE_16_BIT_TICKS* thì đặt cho thời gian cực đại delay là 0xFFFF, ngược lại delay sẽ lớn hơn 0xFFFFFFFF.
- Ngoài ra phần rất quan trọng là khai báo vị trí thanh ghi ngắt toàn cục, hàm cho phép và không cho phép ngắt.
- Vấn đề khác trong file là tạo hàm *ENTER_CRITICAL()* và *EXIT_CRITICAL()*. Khi bắt đầu đoạn bất ly cần cất thanh ghi ngắt vào ngăn xếp sau đó không cho phép ngắt toàn cục. Ngược lại, khi ra khỏi đoạn bất kỳ cần khôi phục thanh ghi ngắt từ ngăn xếp và cho phép ngắt nếu trước khi ngắt có cho phép. Không được thay đổi bất kỳ bit nào khác trong thanh ghi điều khiển ngắt.

1.1.3 Triển khai bài toán và kết quả mô phỏng

a) Triển khai bài toán

- Phần cứng:
  - Laptop Dell Inspiron 5559
  - KIT STM32F103C8
- Phần mềm:
  - Hệ điều hành Window 10
  - IDE Keil C Version 5.22

b) Kết quả mô phỏng

Với các phần mềm được sử dụng như trên, bài toán đã được giải quyết đúng yêu cầu đề ra. Hai lần mô phỏng để xem đáp ứng của hệ điều hành thời gian thực FreeRTOS đều đúng như tiên liệu đề ra. Hình ảnh thực tế khi chạy:

Hình 3.1 Kết quả mô phỏng