TRIỂN KHAI HỆ ĐIỀU HÀNH NHÚNG THỜI GIAN THỰC FreeRTOS TRÊN VI ĐIỀU KHIỂN ARM AT91SAM7S256

Ngô Thị Vinh*
Trường ĐH Công nghệ thông tin và Truyền thông – ĐH Thái Nguyên

TÓM TẮT

FreeRTOS là một hệ điều hành nhúng thời gian thực có nhiều ưu điểm nổi bật để phù hợp cho các hệ thống thời gian thực nhỏ với các kiến trúc khác nhau như ARM, AVR, APSx, AFSx, LPC2000, STM32, PIC18,….Với các nền phần cứng khác nhau người lập trình cần phải cung cấp các tham số đầu vào phù hợp với từng dòng vi điều khiển để ứng dụng hoạt động đúng theo cấu hình của mạch thiết kế. Bài báo này sẽ trình bày các bước xây dựng một ứng dụng với hệ điều hành FreeRTOS và triển khai nó trên chíp vi điều khiển ARM AT91SAM7S256 của hãng Atmel-một chíp thuộc họ vi điều khiển ARM7 được sử dụng rộng rãi trong các bo mạch dùng chủ yếu cho việc thực hành, thí nghiệm với ARM7 trong các trường đại học hoặc các trung tâm nghiên cứu về hệ nhúng. Chương trình ứng dụng là một chương trình đa tiến trình hoạt động song song để có thể tận dụng được khả năng xử lý của CPU.

Từ khóa: *FreeRTOS, ARM, hệ điều hành nhúng, AT91SAM7S256, quản lý tài nguyên, tiến trình.*

GIỚI THIỆU

FreeRTOS là một hệ điều hành nhúng thời gian thực mã nguồn mở[1] ra đời từ năm 2003, đến nay nó được phát triển rất mạnh mẽ và nhận được nhiều sự ủng hộ của các lập trình cho các hệ nhúng. FreeRTOS có tính khả chuyển, có thể sử dụng miễn phí hoặc dùng cho mục đích thương mại[1]. Nó có nhiều ưu điểm nổi bật so với các hệ điều hành nhúng thời gian thực khác như có kích thức rất nhỏ gọn nên rất phù hợp với các hệ nhúng thời gian thực nhỏ; được viết bằng ngôn ngữ C nên có độ phù hợp cao với các nền phần cứng khác nhau. Ngoài ra, FreeRTOS còn hỗ trợ các cơ chế như cho phép tạo cả task và co-routie với số lượng task là không giới hạn phụ thuộc vào tài nguyên của phần cứng của chip[1]; hỗ trợ cơ chế truyền thông đồng bộ giữa các task hoặc giữa task và ngắt bằng cách sử dụng hàng đợi hoặc semaphore nhị phân hoặc semaphore đếm và các mutex; cho phép nhận biết khi ngăn xếp bị tràn. Ngay cả trong các hệ thống nhúng lớn người ta vẫn có thể sử dụng FreeRTOS để tiết kiệm được dung lượng bộ nhớ và làm cho hệ thống ít bị quá tải.

FreeRTOS khi mới ra đời được cài đặt chủ yếu cho các dòng chip [3], [4], [5] như LPC, PIC, RX và hiện nay đang tiếp tục được quan tâm triển khai trên các dòng chip ARM [2]. Với mỗi nền phần cứng khác nhau người lập trình cần cấu hình các thông số khác nhau sao cho phù hợp thì hệ thống mới có thể hoạt động chính xác. Đã có một số ứng dụng về FreeRTOS được viết cho các chip AT91SAM7S32 bít và AT91SAM7S64 bít [10] là các đời vi xử lý thấp hơn của AT91SAM7S256, hoặc các ứng dụng chỉ viết thuần túy là các chương trình lập trình giao tiếp với các cổng vi điều khiển này mà không sử dụng hệ điều hành nhúng thời gian thực[10]. Do đó, chúng chỉ là các ứng dụng đơn tiến trình. Nhiều lập trình viên hệ nhúng hiện nay đang rất quan tâm đến việc triển khai ứng dụng FreeRTOS trên chip AT91SAM7S256. Vì vậy, bài báo này trình bày các bước xây dựng một ứng dụng sử dụng hệ điều hành FreeRTOS, chương trình gồm bốn tiến trình chạy đồng thời và được đồng bộ sử dụng cơ chế hàng đợi và cơ chế mutex của hệ điều hành. Chương trình sẽ sử dụng các LED và các nút nhấn được thiết kế sẵn trên board để thể hiện trạng thái của các task và minh họa các ngắt trong hệ nhúng. Đồng thời chương trình truyền dữ liệu từ bộ chuyển đổi ADC ra máy tính qua cổng COM và hiển thị trên màn hình nhờ một task khác.

* *Tel: 0987706830; Email: ntvinh@ictu.edu.vn*

QUẢN LÝ HOẠT ĐỘNGTRONG FREERTOS

Quản lý các task: Đây nhiệm vụ quan trọng trong FreeRTOS. Các nhiệm vụ mà người lập trình muốn hệ thống thực hiện sẽ được viết trong nội dung của các task. Mỗi task sẽ được gán một độ ưu tiên phù hợp và được bộ lập lịch sắp xếp thời gian hoạt động.

Mỗi task sẽ gồm các tham số như: chức năng, tên, độ ưu tiên, độ sâu stack, định danh task và biến tham số tác động vào task. File task.c [8] sẽ cung cấp một tập các hàm chức năng để làm việc với các task.

Thời gian hoạt động của một task được định nghĩa trong file FreeRTOSConfig.h và được tính bằng mini giây theo công thức t/portTICK_RATE_MS, với t là thời gian ở và trạng thái Blocking của mỗi task. Trong SAM7S256 mỗi Tick tương ứng với 500ms.

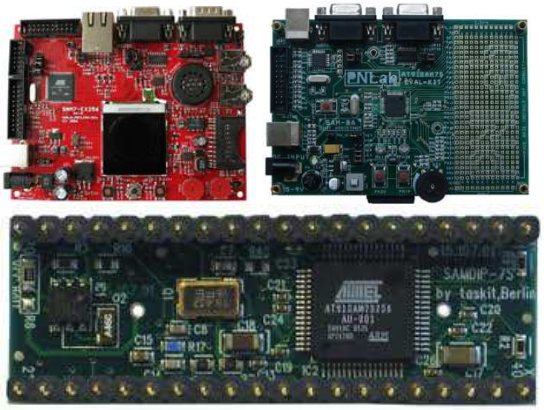
Quản lý hàng đợi [1]: FreeRTOS cung cấp cơ chế hàng đợi (Queue) hoạt động theo nguyên tắc cơ bản là vào trước ra trước-FIFO. Hàng đợi là nơi lưu trữ dữ liệu của các task. Khi một task chờ để ghi dữ liệu vào hàng đợi hoặc là đọc dữ liệu ra từ hàng đợi thì nó sẽ bị rơi vào trạng thái Block.

Quản lý sự kiện ngắt [1]: FreeRTOS cho phép quản lý hiệu quả các ngắt. Khi ngắt xảy ra CPU sẽ chuyển vào hàm thủ tục ngắt, hàm thủ tục ngắt phát đi một "tín hiệu" để hàm thực hiện chức năng của ngắt được thực hiện, hàm này có độ ưu tiên cao hơn tất cả các task khác nên nó sẽ được thực hiện ngay lập tức. Tín hiệu ở đây chính là Semaphore, trong FreeRTOS có 2 loại Semaphore là Binary Semaphore và Counting Semaphore [1]. Có 2 tác động chính vào Semaphore là "Take" và "Give". "Take" là dùng cho hàm thực hiện chức năng của ngắt, khi chưa có ngắt hàm này sẽ ở trạng thái khóa (Blocking) và chờ đợi sự kiện ngắt xảy ra. Tác động "Give" được thực hiện trong hàm thủ tục ngắt, nó sẽ phát ra tín hiệu là có ngắt xảy ra (Semaphore ở trạng thái Full), khi đó hàm thực hiện chức năng ngắt sẽ ngay lập tức được chuyển sang trạng thái sẵn sàng (Semaphore ở trạng thái Empty). Sau khi thực hiện xong nó lại trở lại trạng thái khóa và chờ đợi cho sự kiện ngắt tiếp theo xảy ra.

Đồng bộ dữ liệu trong FreeRTOS [1]: Nếu Semaphore được sử dụng trong FreeRTOS để đồng bộ các sự kiện thì Mutex được sử dụng để đồng bộ dữ liệu giữa các task khi chúng cần truy xuất đến một vùng nhớ chứa dữ liệu chung. Mutex khác semaphore ở chỗ là nó bắt buộc phải trả về vùng đệm dữ liệu sau khi dùng. Để tạo ra một Mutex sử dụng phương thức xSemaphoreCreateMutex (void). Hai hàm tác động vào Mutex là "Take" và "Give" được định nghĩa để bảo vệ vùng mã không cho phép task khác truy xuất khi một task đang sử dụng nó.

BOARD ARM AT91SAM7S256

Với các đặc điểm như trên trình bày FreeRTOS rất phù hợp khi được cài đặt trên họ vi điều khiển SAM7S. Đây là họ vi điều khiển sử dụng lõi ARM7TDMI là nhân của nhiều chip trên các điện thoại di động ngày nay. AT91SAM7S256 là một trong chip vi điều khiển điển hình của họ này và thường được sử dụng trong các thiết bị RFID, đặc biệt được sử dụng để chế tạo một loạt các bo mạch phục vụ cho việc học tập, nghiên cứu, thực hành và thí nghiệm với vi điều khiển ARM tại các trường đại học trên thế giới.



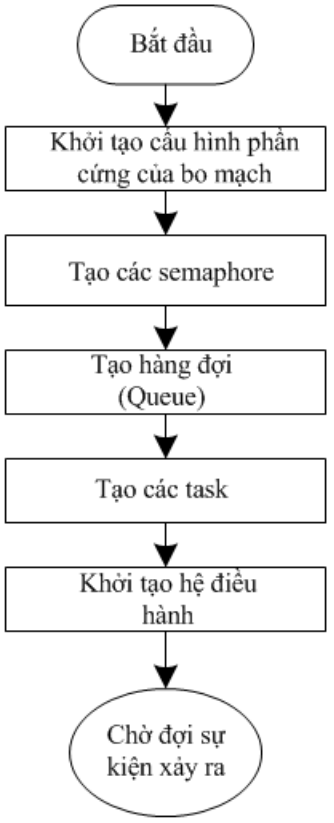
Hình 1. Một số kít thí nghiệm sử dụng AT91SAM7S256

Chíp AT91SAM7S256 có thể hoạt động với tần số 60MHz, 256KB bộ nhớ Flash và 64KB bộ nhớ RAM. Do đó, để tận dụng tối đa hiệu năng xử lý của CPU thì người lập trình nên cài đặt hệ điều hành nhúng thời gian thực như FreeRTOS cho chíp để có thể xây dựng được các ứng dụng đa tiến trình.

Ngoài khả năng hỗ trợ lập trình cho các chân vào ra, hoạt động định thời như các vi điều khiển thông thường, AT91SAM7S256 còn hỗ trợ chuẩn truyền thông nối tiếp theo chuẩn RS485, bộ chuyển đổi ADC 16 bít, chuẩn USB với tốc độ truyền 2.0. Để lập trình cho chip này cần sử dụng tệp thư viện do nhà sản xuất chip Atmel là AT91SAM7S256.h [9], tệp này sẽ định nghĩa chức năng các chân và các ký hiệu được sử dụng trong chương trình.

CÁC BƯỚC XÂY DỰNG ỨNG DỤNG VỚI FREERTOS CHO KÍT AT91SAM7S256

Có thể mô tả cấu trúc hoạt động chung của một chương trình sử dụng hệ điều hành nhúng FreeRTOS như sau:



Hình 2. *Cấu trúc chung của chương trình sử dụng hệ điều hành FreeRTOS trên hệ nhúng*

Bước 1: Khởi tạo cấu hình phần cứng cho bo mạch sẽ thực hiện thiết lập các thông số phù hợp với từng chip phần cứng cụ thể. Chương trình ở đây sẽ thực hiện cấu hình các hằng số tương ứng với xung nhịp đầu vào hệ thống, cấu hình các chân vào ra số 17, 18, 19 và 20 của cổng cổng 0 cho việc điều khiển các nút nhấn và các LED đơn, cấu hình các thông số và khởi động bộ điều khiển ADC và USART trên chip.

Bước 2: Khởi tạo Semaphore. Nếu chương trình có sử dụng semaphore để đồng bộ dữ liệu giữa các task thì chúng sẽ được tạo ra ở bước này. Chương trình ở đây sẽ thực hiện tạo ra một semaphore sử dụng một mutex để đồng bộ một vùng dữ liệu dùng chung cho bộ chuyển đổi ADC và cổng USART.

Bước 3: Khởi tạo hàng đợi. Nếu chương trình sử dụng cơ chế hàng đợi để lưu trữ dữ liệu chung chuyển giữa các tiến trình với nhau thì chúng sẽ được tạo ra ở bước này. Chương trình ứng dụng được trình bày ở đây sẽ sử dụng hàng đợi để lưu trữ giá trị được sử dụng làm cờ trạng thái cho các tiến trình điều khiển các LED.

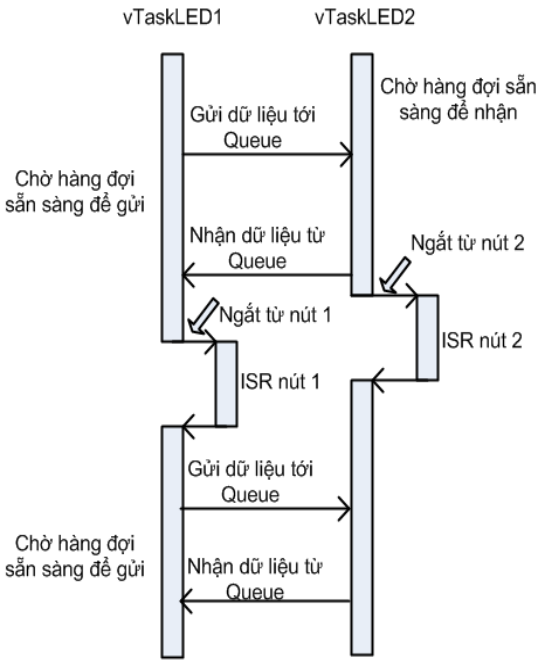
Bước 4: Khởi tạo các task. Bước này sẽ thực hiện tạo ra các task với độ ưu tiên nhất định để thực hiện các nhiệm vụ theo yêu cầu của người lập trình. Chương trình ứng dụng được xây dựng gồm bốn task tương ứng với bốn tiến trình hoạt động song song trên chíp. Mỗi tiến trình sẽ thực hiện nhiệm vụ như sau:

Tiến trình **vTaskLED1**: Điều khiển đảo trạng thái của LED1 tại chân số 17 của cổng 0 và chờ nhận ngắt tác động qua nút bấm tại chân số 19 của cổng này.

Tiến trình **vTaskLED2**: Điều khiển đảo trạng thái của LED2 tại chân số 18 của cổng 0 và chờ nhận ngắt tác động qua nút bấm tại chân số 20 của cổng này.

Hai tiến trình vTaskLED1 và vTaskLED2 sẽ sử dụng hàng đợi được tạo ở bước 3 để điều khiển trạng thái của các LED và cho phép các ngắt mềm tác động khi các nút bấm trên chân số 19 và 20 được nhấn. Khi tiến trình vTaskLED1 đang hoạt động thì tiến trình vTaskLED2 ở trạng thái Blocking và ngược lại. Khi các tiến trình đang hoạt động thì các LED được bật sáng và khi ở trạng thái Blocking thì các LED tương ứng tắt, khi nhấn các nút bấm trên chân 19 và 20 thì có ngắt

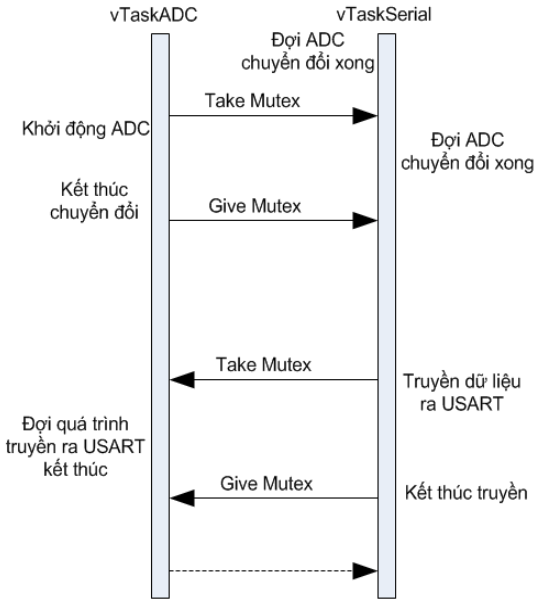
xảy ra. Sơ đồ sau thể trình tự hoạt động của hai task vTaskLED1 và vTaskLED2 được thể hiện như trong hình 3.



Hình 3. *Trình tự hoạt động của task vTaskLED1 và vTaskLED2*

Tiến trình thứ ba là **vTaskADC** sẽ liên tiếp đọc giá trị từ kênh 4 của bộ chuyển đổi tương tự sang số (ADC) và ghi vào một vùng đệm.

Tiến trình thứ tư là **vTaskSerial** sẽ liên tiếp đọc dữ liệu từ vùng đệm do tiến trình **vTaskADC** ghi vào và gửi ra cổng nối tiếp USART. Hai tiến trình vTaskADC và vTaskSerial sử dụng chung một vùng đệm dữ liệu nên phải đồng bộ bằng cách sử dụng một semaphore với cơ chế Mutex được tạo ra ở bước 2. Hoạt động đồng bộ của hai tiến trình này được mô tả như trong hình 4. Thực chất để đồng bộ hai tiến trình ta sử dụng một biến cờ để đánh dấu vùng đệm dữ liệu đang bận khi bộ chuyển đổi ADC chưa hoàn thành hoặc đã hoàn thành để báo cho tiến trình truyền đọc dữ liệu từ vùng đệm này và truyền qua cổng nối tiếp biết. Đồng thời hai tiến trình này sử dụng cặp hàm xSemaphoreTake() và xSemaphoreGive() để bảo vệ vùng mã của mình không cho phép các tiến trình khác xâm nhập.



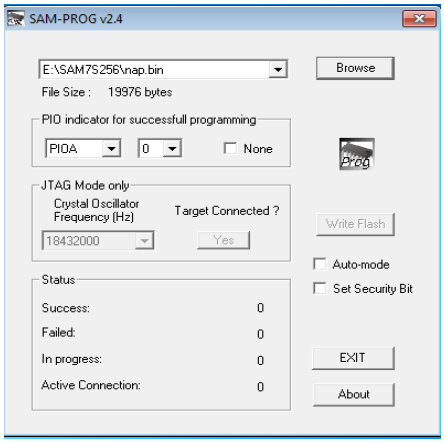
Hình 4. *Sử dụng Mutex để đồng bộ hai task vTaskADC và vTaskSerial*

Bước 5: Khởi động hệ điều hành. Hệ điều hành sẽ bắt đầu chạy là lập lịch để chia sẻ thời gian xử lý cho các tiến trình và chờ đợi các sự kiện ngắt xảy ra.

Tổng kết lại ta có chương trình chính như sau:

```
int main(void){
        Init ();
//Tạo semaphore trong hệ điều hành để đồng bộ tiến trình
        CreateMutexADC();
// Tạo hàng đợi tiến trình
        CreateQueue();
// Tạo các task
CreateAllTask();
//Khởi tạo hệ điều hành FreeRTOS
        vTaskStartScheduler();
        return 0; }
```

Ta sẽ thực hiện biên dịch chương trình bằng phần mềm IAR và nạp chương trình lên kít qua cổng USB sử dụng phần mềm SAM_BA và SAM_PROG. Với SAM_PROG người lập trình có thể nạp chương trình thực thi với đuôi .bin được tạo ra trực tiếp từ các trình biên dịch như IAR hoặc WinARM.



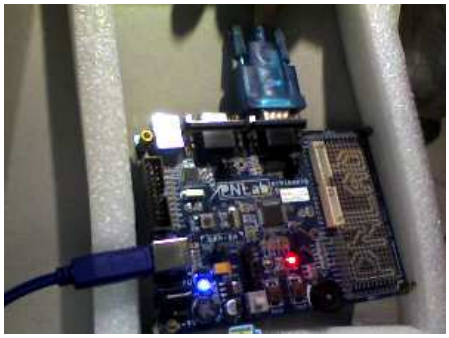
Hình 5. *Nạp file đuôi .bin lên board*

ĐÁNH GIÁ KẾT QUẢ

Kết quả kiểm tra chương trình trên kít cho thấy sau 100 lần bật và tắt nút nguồn trên kít đều thấy chương trình hoạt động ổn định trong thời gian dài (một giờ đồng hồ) và cho kết quả hiển thị trên các LED và trên màn hình máy tính là giống nhau.

Trên kít ta quan sát thấy khi LED tại chân số 17 sáng thì LED tại chân số 18 tắt và ngược lại như hình 4.

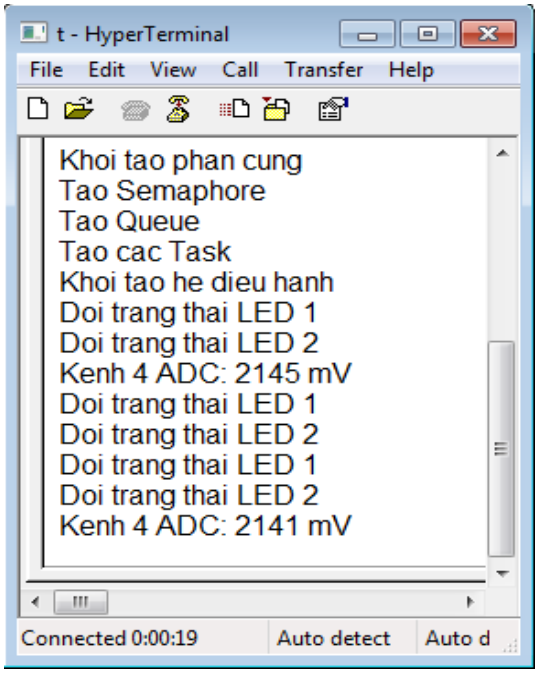
Trên màn hình máy tính (hình 6) hiển thị dữ liệu từ kít gửi qua cổng USART tại các lần test đều giống nhau.



Hình 6. *Kết quả nạp chương trình lên board*

KẾT LUẬN

Bài báo đã chỉ ra các ưu điểm chính của hệ điều hành nhúng thời gian thực FreeRTOS và các bước tiến hành xây dựng một ứng dụng với FreeRTOS trên một nền phần cứng nhất định. Tác giả đã tiến hành nạp chương trình và thử nghiệm trên board AT91SAM7S256 của hãng Atmel.



Hình 7. *Kết quả màn hình máy tính nhận dữ liệu từ cổng COM ảo*

Kết quả cho thấy chương trình hoạt động ổn định và cho các kết quả giống nhau tại các lần kiểm tra. Bài báo có thể làm tài liệu tham khảo cho các lập trình viên hệ nhúng muốn tìm hiểu tìm hiểu về cách xây dựng một ứng dụng đa tiến trình sử dụng FreeRTOS nói chung và trên board AT91SAM7S256 nói riêng.

TÀI LIỆU THAM KHẢO

[1]. Richard Bary, Using the FreeRTOS Real Time Kernel - a Practical Guide - Standard Base Edition, Real Time Engineers Ltd London, 2013.
[2]. RICHARD BARY, USING THE FREERTOS REAL TIME KERNEL FOR CORTEX-M3 EDITION, REAL TIME ENGINEERS LTD LONDON, 2013.
[3]. RICHARD BARY (2013), USING THE FREERTOS REAL TIME KERNEL FOR LPC17XX EDITION , REAL TIME ENGINEERS LTD LONDON.
[4]. Richard Bary, Using The FreeRTOS Real Time Kernel for Microchip PIC32 Edition, Real Time Engineers Ltd London, 2013.
[5]. Richard Bary, (2013). Using The FreeRTOS Real Time Kernel for Renesas RX600 Edition, Real Time Engineers Ltd London.
[6]. ALAN BURNS, ANDY WELLINGS, REAL-TIME SYSTEMS AND PROGRAMMING LANGUAGES, ADDISON WESLEY LONGMAN, NOVEMBER 1ST 1996.
[7]. Alan Burns and Andy Wellings, Real-Time Systems and Programming Languages (Fourth Edition), Addison Wesley Longmain, April 2009
[8]. http://www.freertos.org
[9]. http:// www.atmel.com/
[10].http://siwawi.bauing.uni-kl.de/avr_projects/arm_projects/index_at91.html
[11]. http://www.embeddedrelated.com

SUMMARY

DEPLOYMENT EMBEDDED OPERATING SYSTEM FreeRTOS ON ARM MICROPROCESSOR AT91SAM7S256

Ngo Thi Vinh*
College of of Information And Communication Technology – TNU

FreeRTOS is a real-time embedded operating system that has many advantages for small real-time systems with different architectures such as ARM, AVR, APSx, AFSx, LPC2000, STM32, PIC18,… Programmers need to provide different input parameters corresponding to each microcontroller of different hardware platforms to obtain accurate applications in accordance with the configurations of the designed circuits. This paper presents the steps to build an application using FreeRTOS and to setup FreeRTOS on ARM microcontroller AT91SAM7S256 of Atmel, which is one chip belonged to ARM7 microcontrollers that are widely used in for practicing and experimenting in universities and research centers of embedded systems. The application is a multi-process program operating in parallel to take advantage of the processing power of the CPU.
Keywords: *FreeRTOS, ARM, embedded operating system, AT91SAM7S256, resource management, task.*

Ngày nhận bài: 03/11/2013; Ngày phản biện:15/11/2013; Ngày duyệt đăng: 18/11/2013

Phản biện khoa học: TS. Phùng Trung Nghĩa – Trường ĐH Công nghệ thông tin & TT - ĐHTN

* *Tel: 0987706830; Email: ntvinh@ictu.edu.vn*