1: Introduction to RTOS

Phan Minh Thong

Cài đặt môi trường và 1 số tài liệu liên quan.

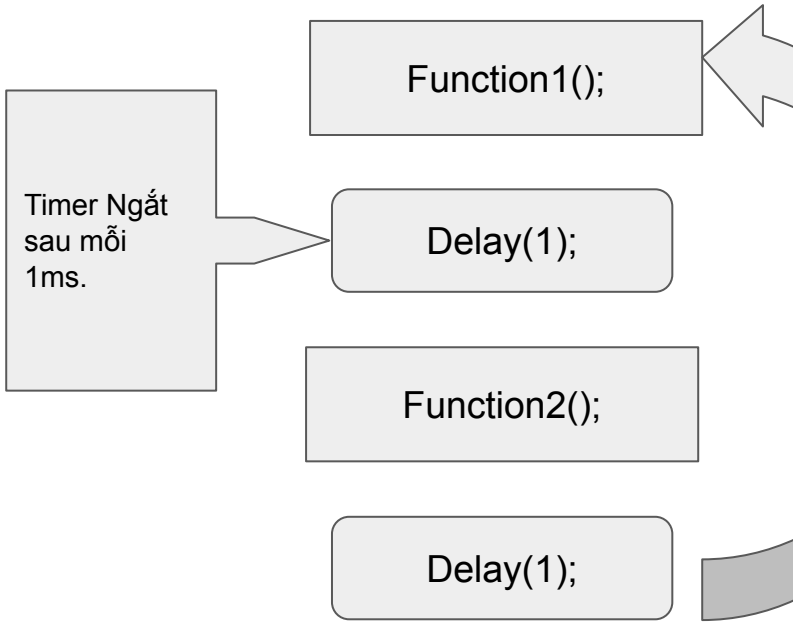
- Môi trường thực hành các bài học: **Keil C** hoặc STM32IDE.
- Vi điều khiển: STM32F103C8T6 (Bluepill), hoặc bất kì Board STM32 nào.
- Tài liệu:
  - Cortex™-M3 Devices Generic User Guide.
  - STM32F103 Reference Manual.
- Ngoài ra để thuận tiện theo dõi quá trình hoạt động, cần 1 số thư viện cho driver liên quan. (UART, TIMER, GPIO).

Blocking Code

Chương trình với Blocking Code.

- Trong Delay, chương trình sẽ phải đợi hàm Delay hoàn tất sau đó mới thực hiện các hàm sau đó được(bị block trong Delay).

=> Việc sử dụng Blocking code khiến chương trình lãng phí thời gian.



Ví dụ:

Một vi điều khiển cần làm những công việc sau:

- Blink LED với chu kỳ 1s.
- Đọc cảm biến khí gas để cảnh báo nguy hiểm khi cần.
- Đọc nút bấm để điều chỉnh chu kỳ nháy led.

```
1. Đọc giá trị cảm biến
2. Đảo trạng thái LED
3. Delay(1000)
4. Cảnh báo khi giá trị cảm biến vượt qua ...
5. Kiểm tra nút bấm / dùng ngắt để kiểm tra
```

Task

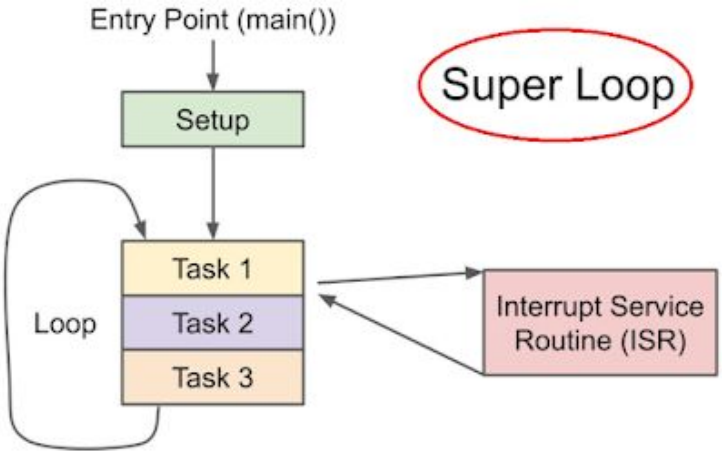
Task: 1 **công việc cụ thể** mà vi điều khiển cần phải hoàn thành. Ở slide trước là ví dụ với 3 task:

Task 1: Nháy LED với chu kỳ 1s.

Task 2: Đọc giá trị cảm biến và cảnh báo khi cần.

Task 3: Đọc nút bấm để thay đổi chu kỳ nháy led.

Ví dụ:




```
1. Đọc giá trị cảm biến
2. Đảo trạng thái LED
3. Delay(1000)
4. Cảnh báo khi giá trị cảm biến vượt qua ...
5. Kiểm tra nút bấm / dùng ngắt để kiểm tra
```

Với chương trình bình thường, chúng ta chạy lần lượt các task trong một vòng lặp và có thể dùng ngắt khi cần.

=>Việc đọc cảm biến và cảnh báo có thể bị lỡ, do việc đọc diễn ra 1 lần / 1s. Có thể dẫn đến nguy hiểm.

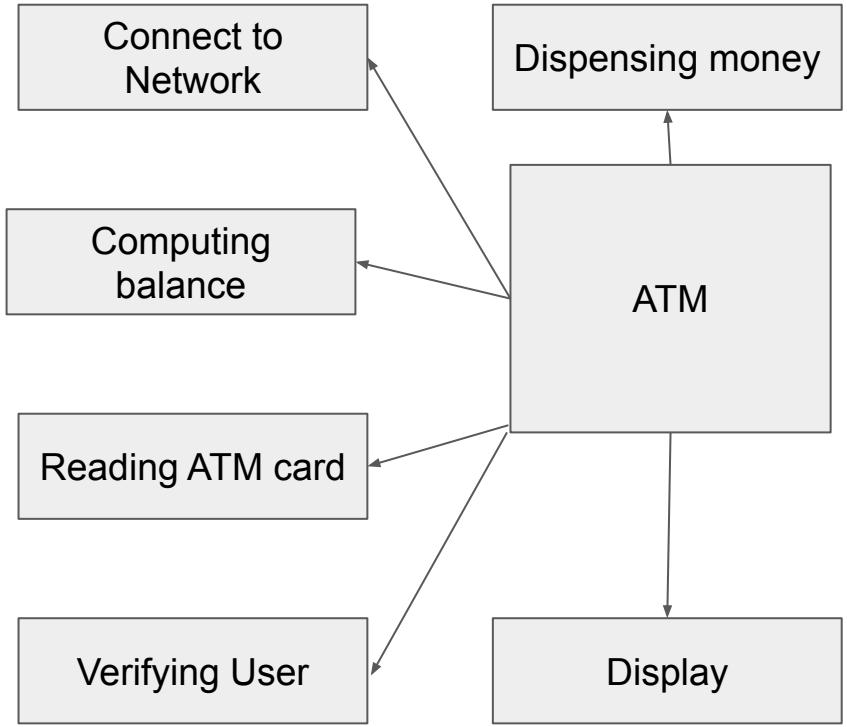
=> RTOS có thể giúp các task chạy gần như **song song** với nhau, giúp khắc phục được nhược điểm này.

Operating System (OS)

Một thiết bị sẽ có nhiều tác vụ chạy cùng lúc, ví dụ:

Cần một phần mềm để quản lý tài nguyên chung giữa các tác vụ, điều khiển các ngoại vi.

=>OS là 1 phần mềm quản lý tài nguyên, sắp xếp và lên lịch cho các tác vụ trong hệ thống.



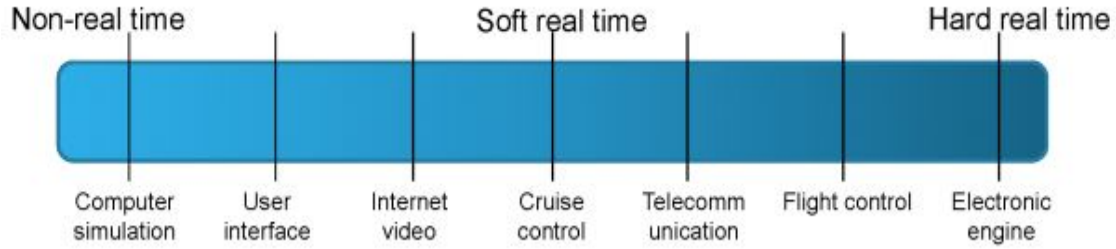
Real-Time Operating System (RTOS)

RTOS có chức năng cơ bản của 1 OS, tuy nhiên nó đáp ứng **yêu cầu về mặt thời gian chặt chẽ.**

Thường ứng dụng cho các hệ thống yêu cầu phản hồi nhanh.

=> Khi xử lý 1 tác vụ, đảm bảo hệ thống sẵn sàng phản hồi trong một thời gian mong muốn.

- Soft Real-Time: Thời gian đáp ứng trong khoảng vài trăm ms.
- Hard Real-Time: Đáp ứng trong khoảng vài chục ms.
- Non Real-Time: Không yêu cầu thời gian đáp ứng.

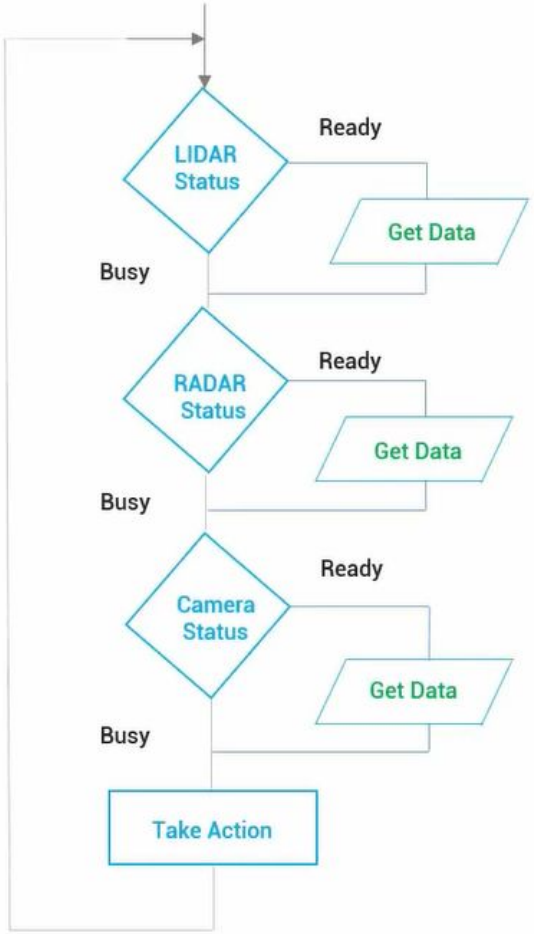


Software Flow trong hệ thống nhúng
1/ Busy Wait

Ví dụ: hệ thống xe có 3 thiết bị.

Theo sơ đồ, chương trình tổ chức các task kiểm tra các thiết bị theo thứ tự lần lượt. Cuối cùng sẽ thực hiện task Take Action dựa trên data nhận được.

Chỉ sử dụng được cho các hệ thống đơn giản.

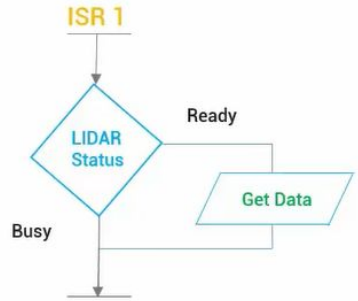


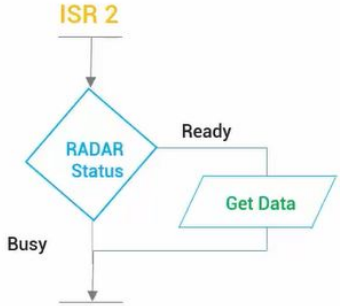
Software Flow trong hệ thống nhúng.
2/ Multi-Threads with ISR.

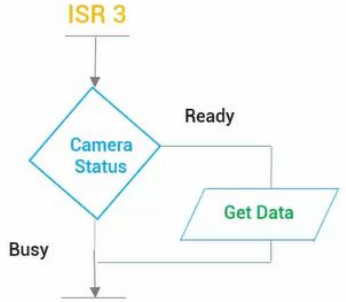
Có thể tổ chức các task nhận data là các task chạy nền. Data sẽ được gửi lên cho task chính là Take Action xử lý..

- Sử dụng Ngắt cho các task nền, các task này sẽ được chạy khi có data hoặc chạy sau khoảng thời gian nhất định nhờ timer.
- Task chính sẽ luôn chạy khi các task khác không chạy.

=>Chỉ đáp ứng được ở mức Soft Real-Time.



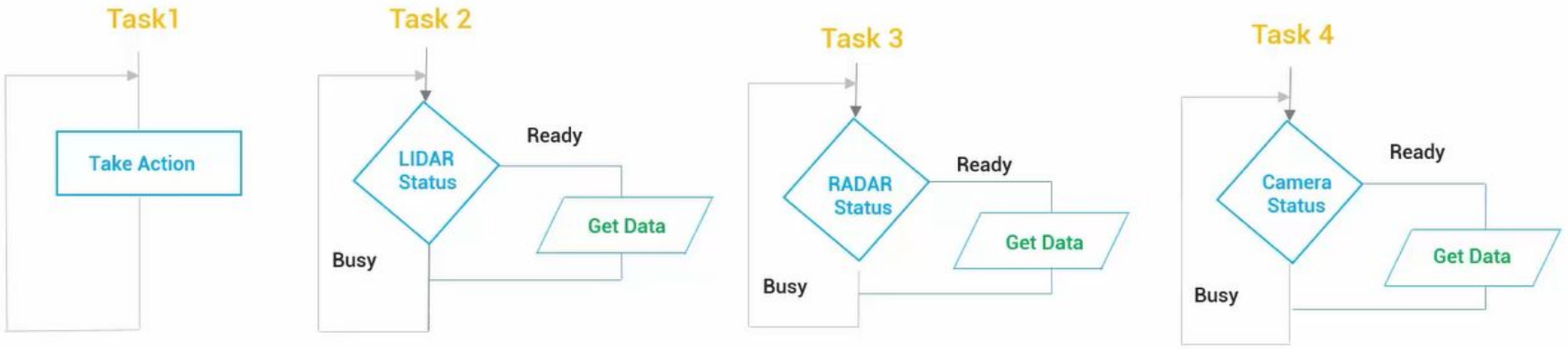




Sofware Flow trong hệ thống nhúng.
3/ RTOS

Để đáp ứng yêu cầu của RTOS, tất cả các task đều sẽ được chạy như 1 task chính , và chạy song song với nhau.

Mỗi task lúc này sẽ như 1 chương trình riêng, ở đây là một function để thực hiện chức năng của chúng và thường sẽ có 1 vòng lặp vô hạn riêng.



Tổng kết

Như vậy, xây dựng 1 hệ thống RTOS có nghĩa là xây dựng các thành phần cần thiết để có thể:

- Thực hiện các tác vụ 1 cách song song.
- Đảm bảo thời gian đáp ứng của hệ thống.
- Quản lý tài nguyên cho các tác vụ trong hệ thống.
- Đồng bộ hoạt động của các tác vụ.
- Chia sẻ dữ liệu giữa các tác vụ nếu cần thiết.
- …