CHƯƠNG 4: KỸ THUẬT LẬP TRÌNH NHÚNG

Bài 8: Biểu diễn trạng thái và mô hình hóa quá trình

Tổng quan

- Mô hình vs Ngôn ngữ
- Mô hình trạng thái
  - FSM/FSMD
  - HCFSM và ngôn ngữ biểu đồ
  - Mô hình trạng thái lập trình (Program-State Machine (PSM) Model)
- Mô hình quá trình đồng thời
  - Truyền thông
  - Đồng bộ
  - Thực hiện
- Mô hình luồng dữ liệu
- Các hệ thời gian thực

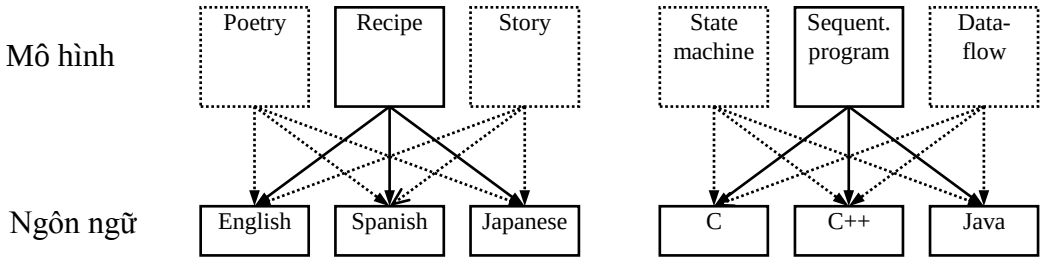
Giới thiệu

- Mô tả trạng thái xử lý của hệ thống nhúng
  - Đôi khi là rất khó
    - Độ phức tạp tăng khi khả năng của IC tăng
      - Trong quá khứ: máy giặt, games etc.
        - Vài trăm dòng lệnh
      - Ngày nay: Đầu TV kỹ thuật số, điện thoại di động etc.
        - Vài trăm nghìn dòng lệnh
    - Trạng thái yêu cầu thường không được hiểu đầy đủ khi bắt đầu
      - Nhiều quá trình thực hiện lỗi do mô tả sự kiện thiếu, ko chính xác
  - Tiếng Anh (hoặc ngôn ngữ khác) – điểm khởi đầu chung
    - Khó mô tả chính xác hoặc đôi khi không thể
    - Ví dụ: Mã điều khiển cho một ô tô – dài hàng nghìn trang...

Mô hình và ngôn ngữ

- Làm thế nào chúng ta ghi nhận hành vi (chính xác)?
  - Chúng ta có thể nghĩ đến ngôn ngữ (C, C++), nhưng mô hình tính toán là mấu chốt
- Mô hình tính toán cơ bản:
  - Mô hình lập trình tuần tự
    - Các câu lệnh, quy tắc ghép câu lệnh, cơ chế thực hiện chúng
  - Mô hình xử lý thông tin
    - Nhiều mô hình tuần tự chạy đồng thời
  - Mô hình trạng thái
    - Cho các hệ riêng, giám sát đầu vào điều khiển, thiết lập đầu ra điều khiển
  - Mô hình luồng dữ liệu
    - Cho các hệ dữ liệu riêng, biến dòng dữ liệu đầu vào thành dòng dữ liệu đầu ra
  - Mô hình hướng đối tượng
    - Để tách phần mềm phức tạp thành đơn giản, các mục được định nghĩa

Mô hình vs ngôn ngữ



Ngôn ngữ nấu ăn – Tiếng Anh

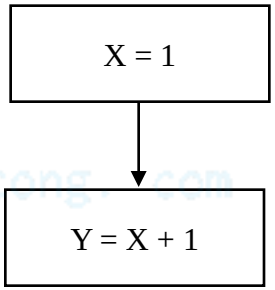
Chương trình tuần tự - C

- Mô hình tính toán mô tả trạng thái của hệ
  - Ghi chú khái niệm, vd công thức hay chương trình tuần tự
- Ngôn ngữ để thể hiện mô hình
  - Dạng duy nhất, ví dụ tiếng Anh, C
- Hiều ngôn ngữ được dùng để thể hiện một mô hình
  - VD mô hình lập trình tuần tự → C,C++, Java
- Một ngôn ngữ có thể thể hiện nhiều mô hình
  - VD C++ → mô hình lập trình tuần tự, mô hình hướng đối tượng, mô hình trạng thái
- Các ngôn ngữ nhất định thể hiện tốt các mô hình tính toán nhất định

Chữ vs Đồ họa

- Mô hình và ngôn ngữ không được nhầm lẫn với "chữ và đồ họa"
  - "Chữ và đồ họa" chỉ là hai kiểu ngôn ngữ
    - Chữ: ký tự, số
    - Đồ họa: vòng tròn, mũi tên (với một số ký tự, số)

```
X = 1;
Y = X + 1;
```



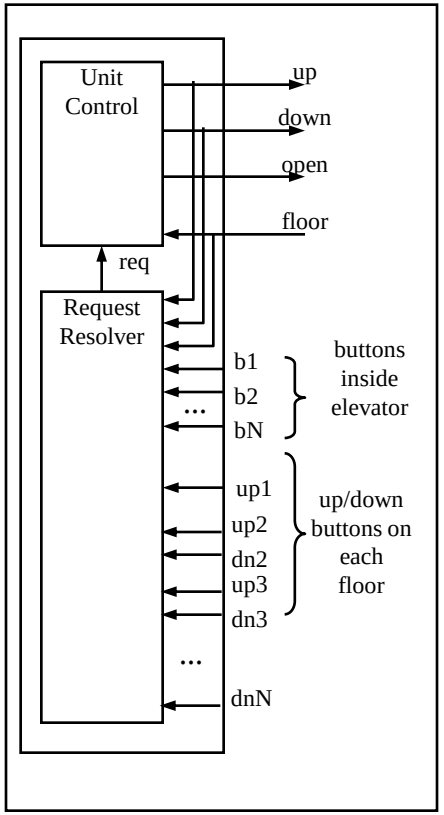
Ví dụ: Bộ điều khiển thang máy

- Bộ điều khiển thang máy đơn giản
  - *Bộ phận yêu cầu* chuyển các yêu cầu khác nhau thành yêu cầu của một tầng duy nhất
  - *Đơn vị điều khiển* di chuyển thang máy tới tầng yêu cầu
- Thử thể hiện bằng C...

Mô tả tiếng Anh một phần

"Di chuyển thang máy lên hoặc xuống để đến tầng yêu. Một khi ở tầng yêu cầu, mở cửa ít nhất 10 giây, và duy trì nó đến khi tầng được yêu cầu thay đổi. Đảm bảo cửa không bao giờ mở khi di chuyển. Không thay đổi hướng trừ khi có yêu cầu ở tầng cao hơn khi đi lên hoặc tầng thấp hơn khi đi xuống…"

Giao diện hệ thống



Bộ điều khiển thang máy sử dụng mô hình lập trình tuần tự

Mô hình cương trình tuần tự

Inputs: int floor; bit b1..bN; up1..upN-1; dn2..dnN;
Outputs: bit up, down, open;
Global variables: int req;

```
void UnitControl()
{
  up = down = 0; open = 1;
  while (1) {
    while (req == floor);
    open = 0;
    if (req > floor) { up = 1;}
    else {down = 1;}
    while (req != floor);
    up = down = 0;
    open = 1;
    delay(10);
  }
}
```

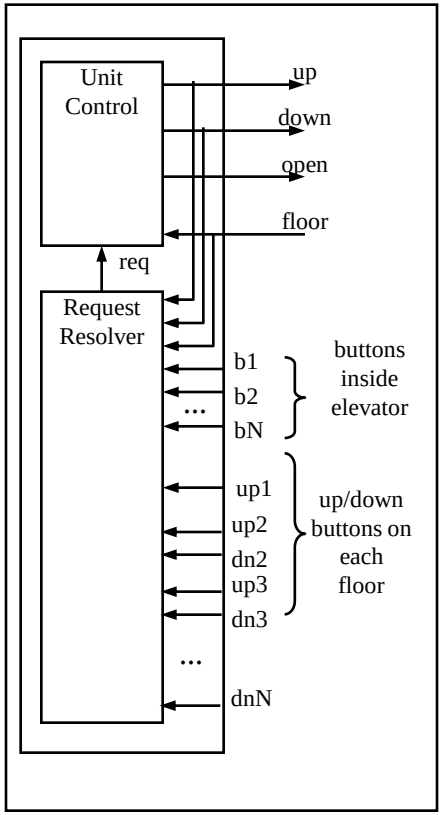
```
void RequestResolver()
{
  while (1)
  ...
    req = ...
  ...
}

void main()
{
  Call concurrently:
    UnitControl() and
    RequestResolver()
}
```

Có thể thực hiện chương trình với nhiều câu lệnh "if" hơn.

Mô tả tiếng Anh một phần

"Di chuyển thang máy lên hoặc xuống để đến tầng yêu. Một khi ở tầng yêu cầu, mở cửa ít nhất 10 giây, và duy trì nó đến khi tầng được yêu cầu thay đổi. Đảm bảo cửa không bao giờ mở khi di chuyển. Không thay đổi hướng trừ khi có yêu cầu ở tầng cao hơn khi đi lên hoặc tầng thấp hơn khi đi xuống…"

Giao diện hệ thống

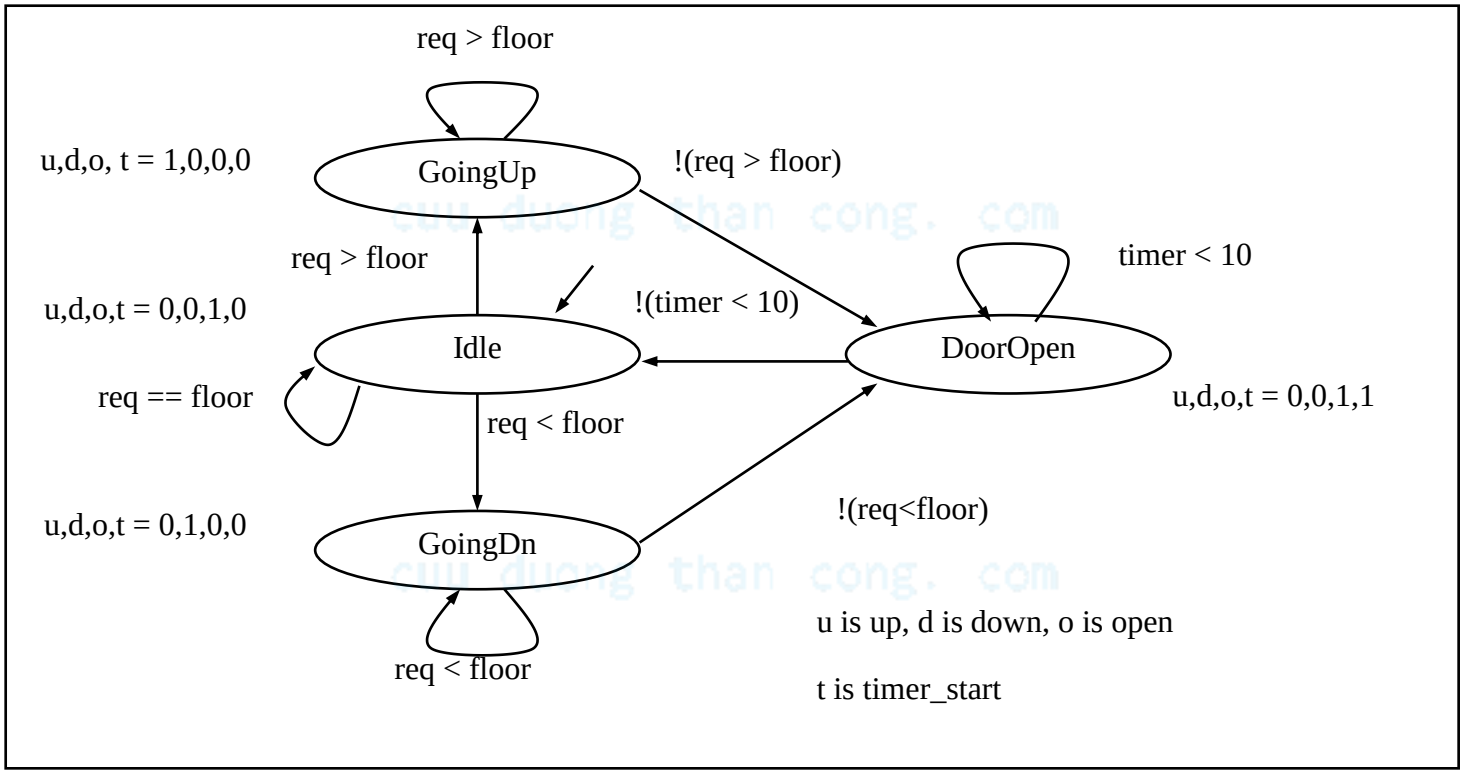


Mô hình trạng thái máy hữu hạn (Finite-state machine model – FSM)

- Cố gắng thể hiện trạng thái này như một chương trình tuần tự là đôi khi không đầy đủ hoặc khó
- Thay vào đó, chúng ta có thể xem xét như một mô hình FSM, mô tả hệ như sau:
  - Các trạng thái có thể
    - VD, nghỉ, đi lên, đi xuống, mở cửa
  - Chuyển đổi có thể từ trạng thái này đến trạng thái khác dựa trên các đầu vào
    - VD yêu cầu → tầng
  - Các hoạt động xảy ra trong mỗi trạng thái
    - VD trong trạng thái đi lên, u,d,o,t = 1,0,0,0 (up = 1, down, open, and timer_start = 0)
- Thử...

Mô hình trạng thái máy hữu hạn (FSM)

Quá trình của đơn vị điều khiển sử dụng máy trạng thái



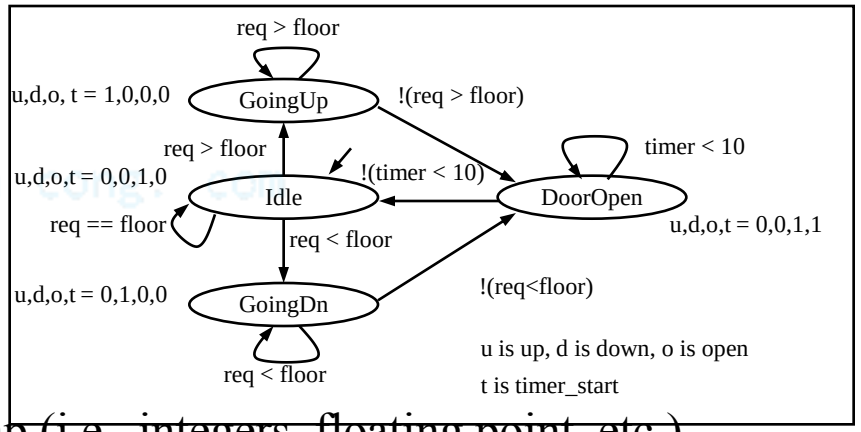
Định nghĩa chuẩn

- Một FSM gồm 6-phần tử $F<S, I, O, F, H, s_0>$
  - S là tập tất cả các trạng thái $\{s_0, s_1, \ldots, s_l\}$
  - I là tập đầu vào $\{i_0, i_1, \ldots, i_m\}$
  - O là tập đầu ra $\{o_0, o_1, \ldots, o_n\}$
  - F là hàm trạng thái tiếp theo (S x $I \rightarrow S$)
  - H là hàm đầu ra ($S \rightarrow O$)
  - s_0 là trạng thái đầu
- Kiểu Moore
  - Liên kết các đầu vào với các trạng thái (như trên, H liên kết $S \rightarrow O$)
- Kiểu Mealy
  - Liên kết đầu ra với chuyển trạng thái (H liên kết S x $I \rightarrow O$)
- Viết tắt để đơn giản hóa các mô tả
  - Gán 0 cho tất cả các đầu ra, không gán giá trị trong một trạng thái
  - AND tất cả các điều kiện chuyển với xung đồng hồ (FSM là quá trình đồng bộ)

Trạng thái máy hữu hạn với mô hình tuyến dữ liệu (FSMD)

- FSMD mở rộng FSM: các kiểu dữ liệu và biến phức tạp để lưu trữ dữ liệu
  - FSMs chỉ sử dụng kiểu dữ liệu và toán hạng Boolean, không có biến
- FSMD: 7-phần tử <S, I , O, **V**, F, H, s_0>
  - S là tập các trạng thái $\{s_0, s_1, \ldots, s_l\}$
  - I là tập các đầu vào $\{i_0, i_1, \ldots, i_m\}$
  - O là tập các đầu ra $\{o_0, o_1, \ldots, o_n\}$
  - **V là tập các biến $\{v_0, v_1, \ldots, v_n\}$**
  - F là hàm của trạng thái tiếp (S x I x $V \rightarrow S$)
  - H là một hàm **tác động** ($S \rightarrow O + V$)
  - s_0 là trạng thái đầu

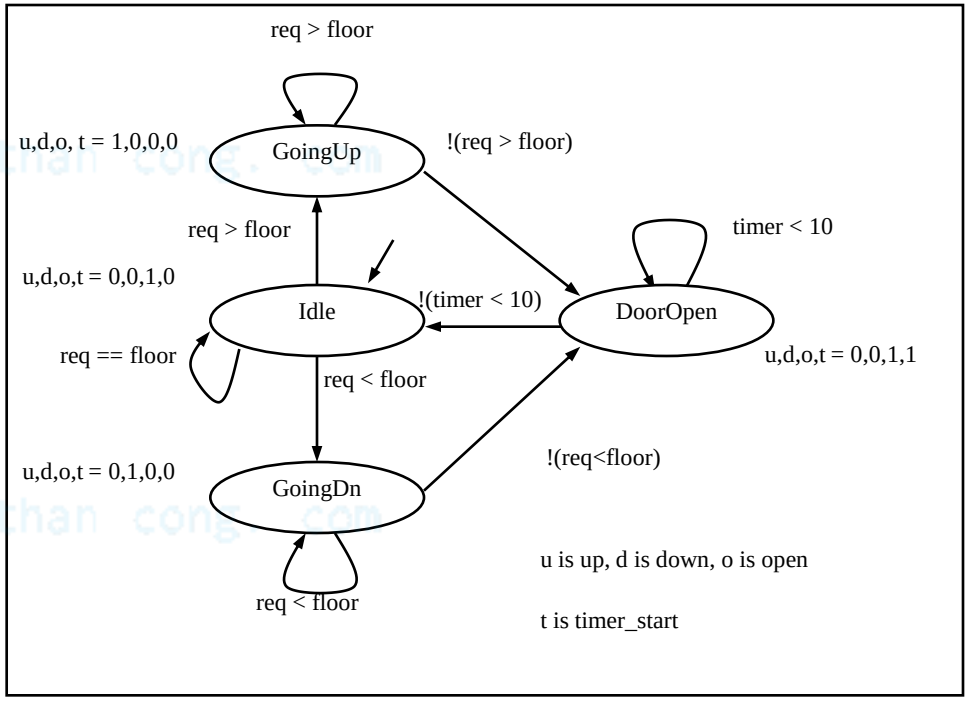
We described UnitControl as an FSMD



- I,O,V có thể diễn tả các kiểu dữ liệu phức tạp (i.e., integers, floating point, etc.)
- F,H có thể bao gồm các toán hạng
- H là một hàm tác động, không chỉ là một hàm ra
  - Mô tả các biến cập nhật cũng như đầu ra
- Trạng thái hoàn thiện của hệ bao gồm trạng thái hiện tại, s_i, và các giá trị của tất cả các biến

Mô tả hệ theo trạng thái máy

1. Liệt kê tất cả các trạng thái có thể
2. Khai báo tất cả các biến
3. Với mỗi trạng thái, liệt kê các chuyển trạng thái có thể, với các điều kiện, sang các trạng thái khác
4. Với mỗi trạng thái/chuyển, liệt kê các hoạt động liên quan
5. Với mỗi trạng thái, đảm bảo loại trừ và các điều kiện chuyển đã có
   - Không tồn tại hai điều kiện đúng tại một thời điểm
     - Nếu không sẽ trở thành trạng thái máy không xác định
   - Một điều kiện phải đúng tại mọi thời điểm



Trạng thái máy vs. Mô hình lập trình tuần tự

- Trạng thái máy:
  - Khuyến khích người thiết kế nghĩ đến tất cả các trạng thái có thể và chuyển trạng thái dựa trên tất cả các điều kiện đầu vào có thể
- Mô hình lập trình tuần tự:
  - Được thiết kế để chuyển dữ liệu thông qua chuỗi các lệnh mà có thể lặp lại hoặc thực hiện có điều kiện

cuu duong than cong. com

Thử mô tả các hành vi khác với một mô hình FSM

- VD: Máy trả lời nhấp nháy đèn khi có bản tin
- VD: Một máy trả lời điện thoại đơn giản mà trả lời sau 4 hồi chuông
- VD: Một hệ thống đèn giao thông đơn giản
- Nhiều ví dụ khác

Mô tả trạng thái máy trong ngôn ngữ lập trình tuần tự

- Mặc dù mô hình trạng thái máy có nhiều lợi ích, hầu hết các công cụ phát triển phổ biến sử dụng ngôn ngữ lập trình tuần tự
  - C, C++, Java, Ada, VHDL, Verilog HDL, vv….
  - Công cụ phát triển đắt và phức tạp, bởi vậy không dễ để thích nghi hay thay đổi
    - Phải đầu tư
- Hai phương pháp để mô tả mô hình trạng thái máy với ngôn ngữ lập trình tuần tự
  - Phương pháp công cụ hỗ trợ
    - Công cụ bổ sung được cài đặt để hỗ trợ ngôn ngữ trạng thái máy
      - Đồ họa
      - Có thể hỗ trợ mô phỏng đồ họa
      - Tự động tạo ra code trong ngôn ngữ lập trình tuần tự là đầu vào cho các công cụ phát triển chính
    - Hạn chế: phải hỗ trợ các công cụ bổ sung (giá bản quyền, nâng cấp, đào tạo, vv.)
  - Phương pháp ngôn ngữ tập con
    - Phương pháp thông dụng nhất...

Phương pháp ngôn ngữ tập con

- Tuân theo các quy tắc (mẫu) để mô tả cấu trúc trạng thái máy trong cấu trúc ngôn ngữ tuần tự tương đương
- Được sử dụng với phần mềm (VD: C) và ngôn ngữ phần cứng (VD: VHDL)
- Mô tả trạng thái máy *UnitControl* bằng C
  - Liệt kê các trạng thái (#define)
  - Khai báo các biến trạng thái, khởi tạo giá trị đầu (IDLE)
  - Câu lệnh chuyển mạch đơn rẽ nhánh tới trạng thái hiện tại
  - Mỗi trường hợp có các hoạt động
    - up, down, open, timer_start
  - Mỗi trường hợp kiểm tra điều kiện chuyển để xác định trạng thái tiếp theo
    - if(…) {state = …;}

```
#define IDLE0
#define GOINGUP1
#define GOINGDN2
#define DOOROPEN3
void UnitControl() {
   int state = IDLE;
   while (1) {
      switch (state) {
         IDLE: up=0; down=0; open=1; timer_start=0;
            if   (req==floor) {state = IDLE;}
            if   (req > floor) {state = GOINGUP;}
            if   (req < floor) {state = GOINGDN;}
            break;
         GOINGUP: up=1; down=0; open=0; timer_start=0;
            if   (req > floor) {state = GOINGUP;}
            if   (!(req>floor)) {state = DOOROPEN;}
            break;
         GOINGDN: up=1; down=0; open=0; timer_start=0;
            if   (req < floor) {state = GOINGDN;}
            if   (!(req<floor)) {state = DOOROPEN;}
            break;
         DOOROPEN: up=0; down=0; open=1; timer_start=1;
            if (timer < 10) {state = DOOROPEN;}
            if (!(timer<10)){state = IDLE;}
            break;
      }
   }
}
```

Trạng thái máy *UnitControl* trong ngôn ngữ lập trình tuần tự

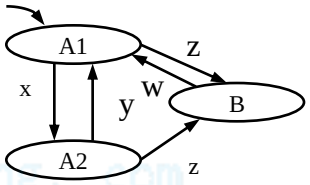
Mẫu chung

```
#define S0  0
#define S1  1
...
#define SN  N
void StateMachine() {
   int state = S0; // or whatever is the initial state.
   while (1) {
      switch (state) {
         S0:
            // Insert S0's actions here & Insert transitions T_i leaving S0:
            if( T_0's condition is true ) {state = T_0's next state; /*actions*/ }
            if( T_1's condition is true ) {state = T_1's next state; /*actions*/ }
            ...
            if( T_m's condition is true ) {state = T_m's next state; /*actions*/ }
            break;
         S1:
            // Insert S1's actions here
            // Insert transitions T_i leaving S1
            break;
         ...
         SN:
            // Insert SN's actions here
            // Insert transitions T_i leaving SN
            break;
      }
   }
}
```

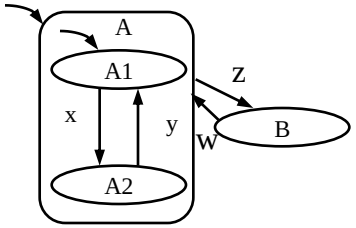
HCFSM và ngôn ngữ biểu đồ trạng thái

- Mô hình trạng thái máy phân cấp/đồng thời (Hierarchical/concurrent state machine model - HCFSM)
  - Mở rộng mô hình trạng thái máy để hỗ trợ phân cấp và đồng thời
  - Các trạng thái có thể tách thành các trạng thái máy khác
  - Các trạng thái có thể thực hiện đồng thời
- Biểu đồ trạng thái
  - Ngôn ngữ đồ họa để mô tả HCFSM
  - *timeout*: chuyển trạng thái với giới hạn thời gian như là một điều kiện

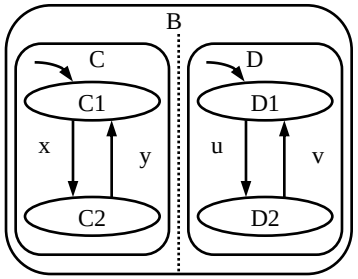
Without hierarchy



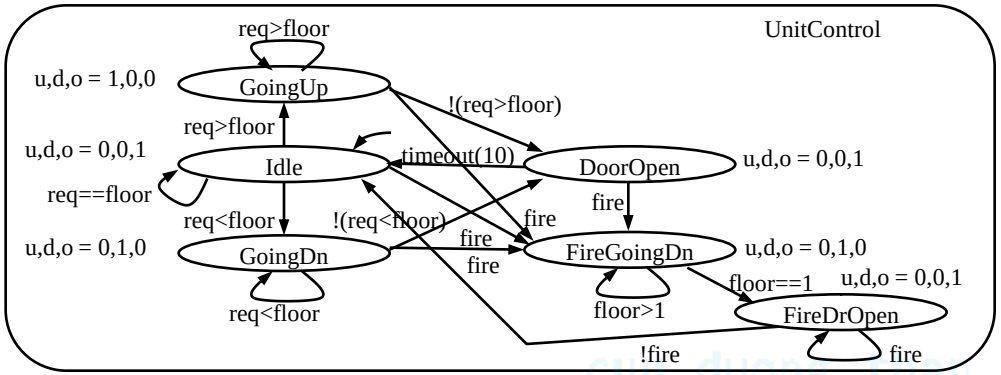
With hierarchy



Concurrency



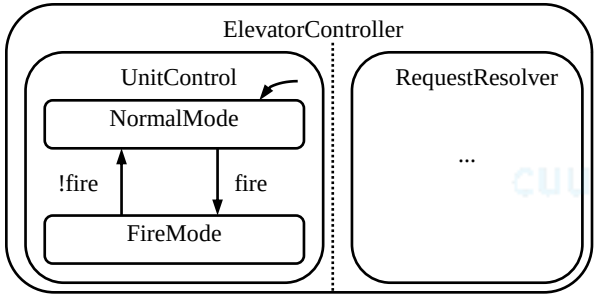
UnitControl với *FireMode*



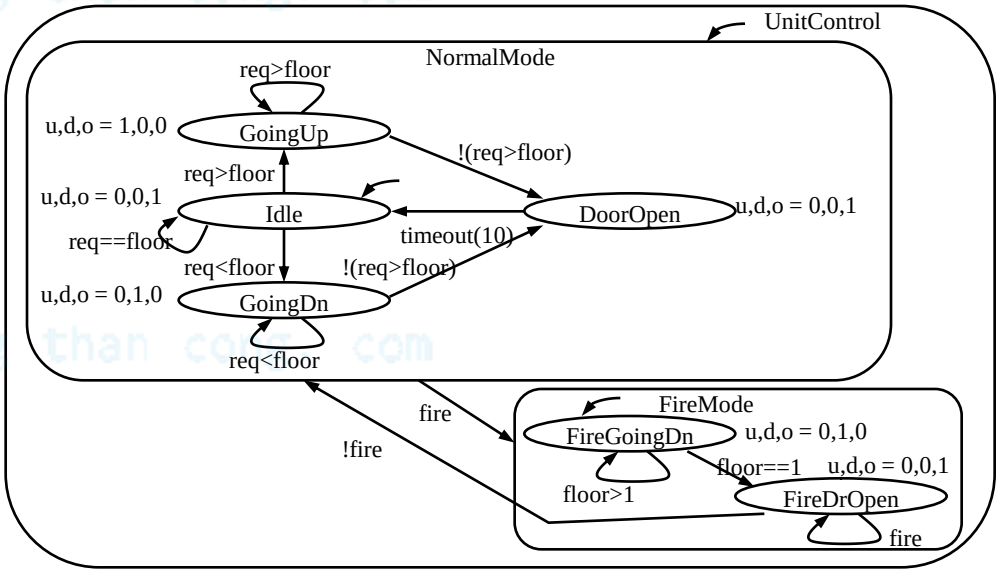
Không phân cấp

- FireMode
  - Khi *fire* đúng, di chuyển thang máy tới tầng 1st và mở cửa
  - w/o hierarchy: Getting messy!
  - w/ hierarchy: Simple!

Có phân cấp

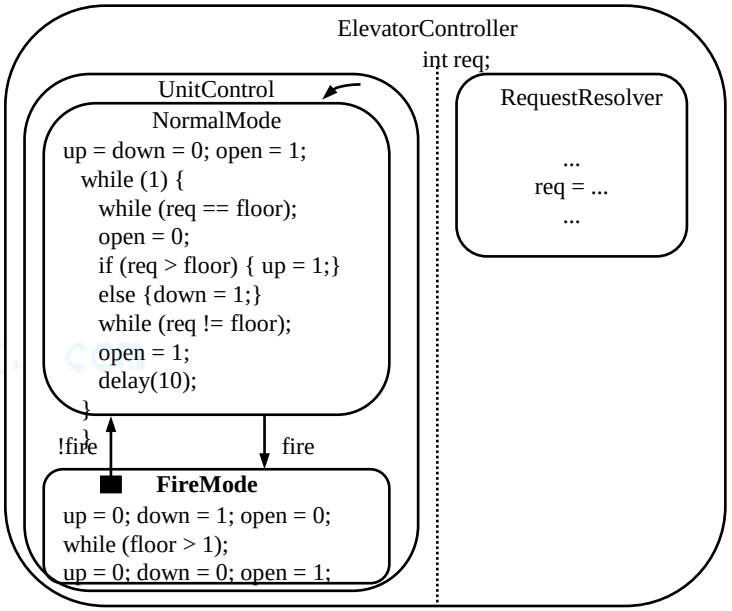


Xử lý đồng thời với *RequestResolver*



Mô hình trạng thái máy – chương trình (PSM): HCFSM + mô hình lập trình tuần tự

- Các hoạt động của trạng thái chương trình có thể là FSM hoặc chương trình tuần tự
  - Người thiết kế chọn kiểu thích hợp nhất
- Phân cấp hạn chế hơn HCFSM sử dụng trong biểu đồ trạng thái
  - Chỉ chuyển trạng thái giữa các trạng thái cạn kề, đầu vào đơn
  - Trạng thái – chương trình có thể "hoàn thiện"
    - Đạt đến cuối của chương trình tuần tự, hoặc
    - FSM chuyển tới trạng thái con hoàn thiện
    - PSM có hai kiểu chuyển
      - Chuyển trực tiếp (TI): xảy ra bất kể trạng thái của chương trình nguồn
      - Chuyển khi hoàn thành (TOC): xảy ra nếu điều kiện đúng và chương trình nguồn kết thúc
  - Biểu đồ đặc biệt: mở rộng của VHDL để mô tả mô hình PSM
  - C đặc biệt: mở rộng của C để mô tả mô hình PSM



Vai trò của việc chọn ngôn ngữ và mô hình thích hợp

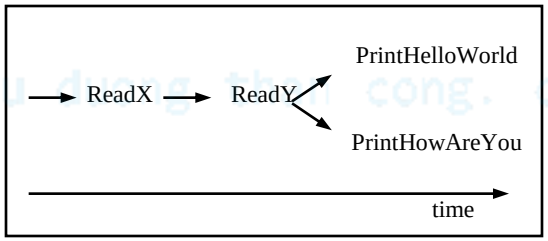
- Tìm mô hình thích hợp để biểu diễn hệ nhúng là một bước quan trọng
  - Mô hình sẽ ảnh hưởng đến cách chúng ta nhìn hệ thống
    - Ban đầu chúng ta nghĩ về chuỗi các hoạt động, viết chương trình tuần tự
      - Trước tiên đợi lệnh gọi tầng
      - Sau đó, đóng cửa
      - Sau đó, di chuyển lên hay xuống đến tầng yêu cầu
      - Rồi mở cửa
      - Rồi lặp lại tuần tự này
    - Để tạo ra trạng thái máy, chúng ta nghĩ theo khía cạnh các trạng thái và sự chuyển đổi giữa chúng
      - Khi hệ phải phản ứng lại với các đầu vào thay đổi, trạng thái máy là mô hình tốt nhất
        - HCFSM mô tả *FireMode* một cách dễ dàng và rõ ràng
- Ngôn ngữ nên mô tả mô hình dễ dàng
  - Về lý tưởng, nên có các đặc điểm mô tả trực tiếp cấu trúc của mô hình
  - *FireMode* sẽ rất phức tạp trong chương trình tuần tự
    - Xem lại code
  - Các yếu tố khác có thể ảnh hưởng đến việc chọn lựa mô hình
    - Các kỹ thuật cấu trúc có thể sử dụng để thay thế
      - VD: Các nhãn để mô tả trạng thái máy trong chương trình tuần tự

Mô hình quá trình đồng thời

- Mô tả chức năng của hệ theo khía cạnh của hai hoặc nhiều tác vụ thực hiện đồng thời
- Nhiều hệ thống dễ hơn để mô tả với mô hình quá trình đồng thời bởi vì tính chất đa tác vụ của nó
- Ví dụ đơn giản:
  - Đọc hai số *X* và *Y*
  - Hiển thị "Hello world." sau mỗi *X* giây
  - Hiển thị "How are you?" sau mỗi *Y* giây

```
ConcurrentProcessExample() {
  x = ReadX()
  y = ReadY()
  Call concurrently:
    PrintHelloWorld(x) and
    PrintHowAreYou(y)
}
PrintHelloWorld(x) {
  while( 1 ) {
    print "Hello world."
    delay(x);
  }
}
PrintHowAreYou(x) {
  while( 1 ) {
    print "How are you?"
    delay(y);
  }
}
```

Ví dụ quá trình đồng thời đơn giản



Chương trình thực hiện

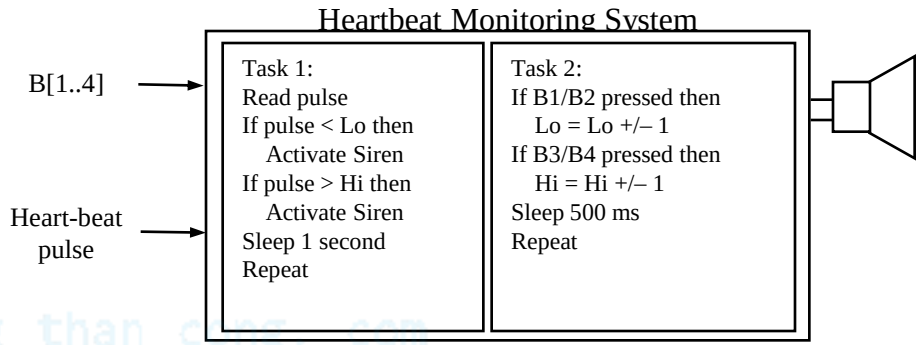
```
Enter X: 1
Enter Y: 2
Hello world.   (Time = 1 s)
Hello world.   (Time = 2 s)
How are you?   (Time = 2 s)
Hello world.   (Time = 3 s)
How are you?   (Time = 4 s)
Hello world.   (Time = 4 s)
...
```

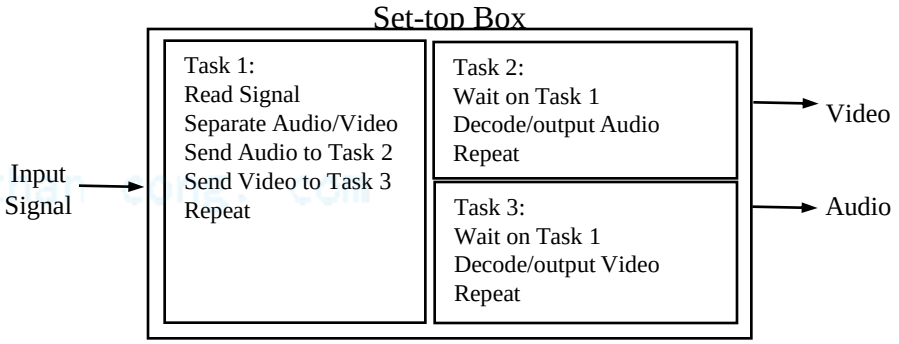
Đầu vào và đầu ra mẫu

Quá trình đồng thời và các hệ thời gian thực

Quá trình đồng thời

- Xem xét hai ví dụ có các tác vụ chạy độc lập nhưng chia sẻ dữ liệu
- Khó để viết sử dụng mô hình lập trình tuần tự
- Mô hình quá trình đồng thời dễ hơn
  - Các chương trình tuần tự riêng cho các tác vụ
  - Các chương trình trao đổi thông tin với nhau



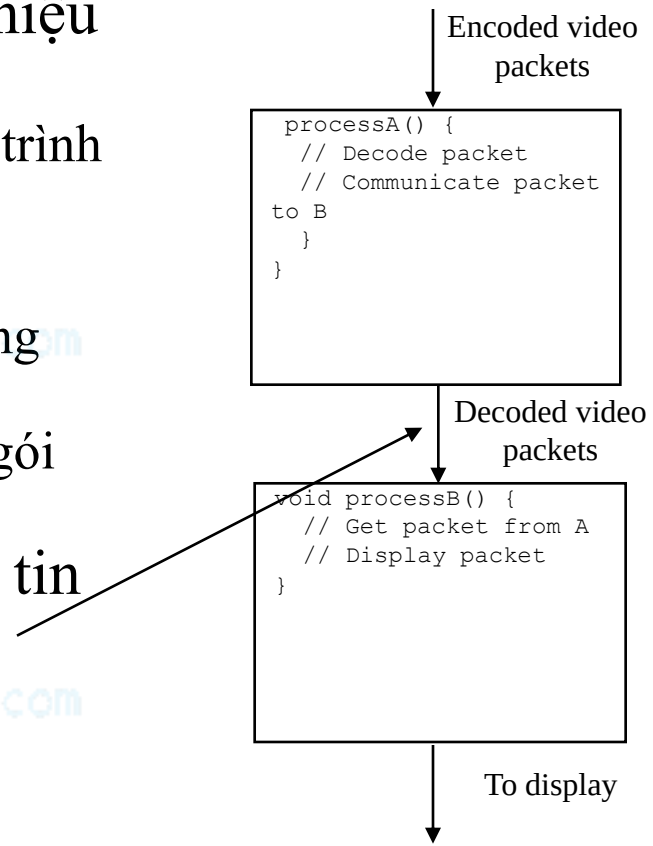


Quá trình

- Một chương trình tuần tự, thường là vòng lặp vô hạn
  - Thực hiện đồng thời với các quá trình khác
  - Chúng ta đang bước vào thế giới "lập trình đồng thời"
- Các hoạt động chính của quá trình
  - Khởi tạo và kết thúc
    - Khởi tạo giống một thủ tục gọi, nhưng bên gọi không đợi
      - Quá trình được khởi tạo có thể chính nó tạo ra các quá trình mới
    - Kết thúc chấm dứt một quá trình, loại bỏ các dữ liệu
    - Trong ví dụ HelloWord/HowAreYou, chúng ta chỉ khởi tạo quá trình
  - Dừng và phục hồi
    - Dừng là đưa quá trình vào trạng thái giữ, lưu trữ trạng thái cho việc thực hiện sau đó
    - Phục hồi khởi đầu lại quá trình từ điểm nó được dừng
  - Liên kết
    - Một quá trình dừng cho đến khi một quá trình con được kết thúc

Thông tin giữa các quá trình

- Các quá trình cần trao đổi dữ liệu và tín hiệu để giải quyết vấn đề tính toán của chúng
  - Các quá trình không thông tin là các chương trình độc lập giải quyết các vấn đề độc lập
- Ví dụ cơ bản: producer/consumer
  - Quá trình A tạo ra dữ liệu, quá trình B sử dụng chúng
  - VD: A giải mã các gói video, B hiển thị các gói được giải mã trên màn hình
- Làm thế nào để đạt được quá trình thông tin này?
  - Hai phương pháp cơ bản
    - Chia sẻ bộ nhớ
    - Chuyển tin



Chia sẻ bộ nhớ

- Các quá trình đọc và ghi các biến được chia sẻ
  - Không mất thời gian, dễ thực hiện
  - Nhưng hay bị lỗi
- Ví dụ: Producer/consumer với một lỗi
  - Chia sẻ *buffer*[*N*], *count*
    - *count* = # số dữ liệu trong *buffer*
  - *processA* tạo dữ liệu và lưu trong *buffer*
    - Nếu *buffer* đầy, phải đợi
  - *processB* sử dụng dữ liệu trong *buffer*
    - Nếu *buffer* trống, phải đợi
  - Lỗi xảy ra khi cả hai quá trình cập nhật *count* đồng thời (dòng 10 và 19) và tuần tự thực hiện sau xảy ra. "count" là 3.
    - *A* nạp *count* (*count* = 3) từ bộ nhớ vào thanh ghi R1 (R1 = 3)
    - *A* tăng R1 (R1 = 4)
    - *B* nạp *count* (*count* = 3) từ bộ nhớ vào thanh ghi R2 (R2 = 3)
    - *B* giảm R2 (R2 = 2)
    - *A* lưu R1 trở lại *count* trong bộ nhớ (*count* = 4)
    - *B* lưu R2 trở lại *count trong* bộ nhớ (*count* = 2)
  - *count* có giá trị không đúng là 2

```
01: data_type buffer[N];
02: int count = 0;
03: void processA() {
04:   int i;
05:   while( 1 ) {
06:     produce(&data);
07:     while( count == N );/*loop*/
08:     buffer[i] = data;
09:     i = (i + 1) % N;
10:     count = count + 1;
11:   }
12: }
13: void processB() {
14:   int i;
15:   while( 1 ) {
16:     while( count == 0 );/*loop*/
17:     data = buffer[i];
18:     i = (i + 1) % N;
19:     count = count - 1;
20:     consume(&data);
21:   }
22: }
23: void main() {
24:   create_process(processA);
25:   create_process(processB);
26: }
```

Chuyển tin

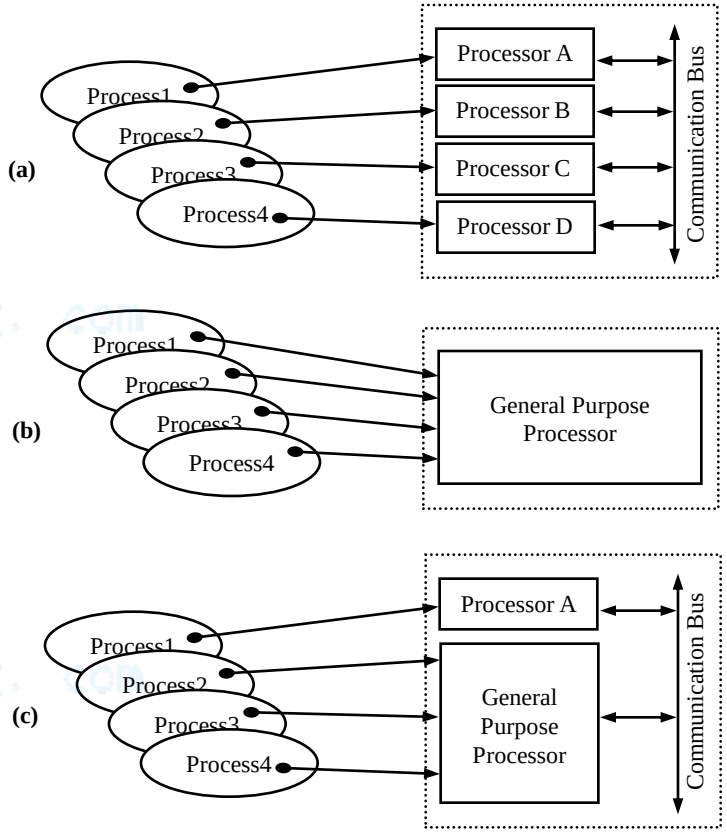
- Chuyển tin
  - Dữ liệu được gửi thẳng từ một quá trình tới quá trình khác
    - Quá trình gửi thực hiện hoạt động đặc biệt, *send*
    - Quá trình thu thực hiện hoạt động đặc biệt, *receive*, để thu dữ liệu
    - Cả hai hoạt động phải chỉ ra quá trình nào gửi hoặc nhận
    - Thu theo gói, gửi có thể hoặc không thể theo gói
  - Mô hình này an toàn hơn, nhưng kém linh hoạt

```
void processA() {
  while( 1 ) {
    produce(&data)
    send(B, &data);
    /* region 1 */
    receive(B, &data);
    consume(&data);
  }
}
```

```
void processB() {
  while( 1 ) {
    receive(A, &data);
    transform(&data)
    send(A, &data);
    /* region 2 */
  }
}
```

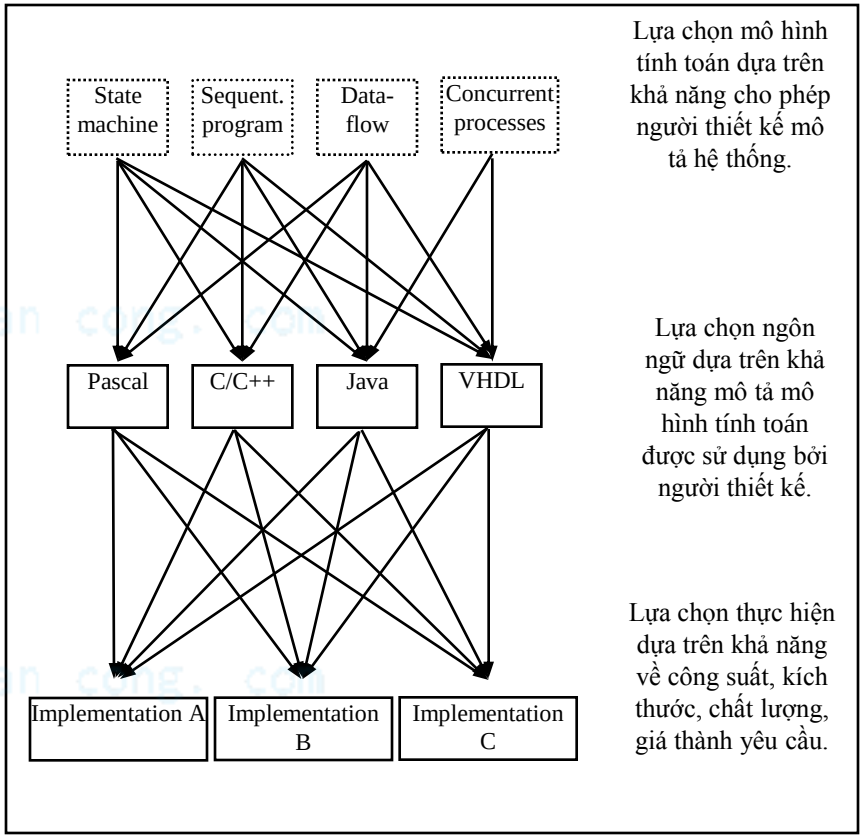
Mô hình quá trình đồng thời: Thực hiện

- Có thể sử dụng bộ xử lý chức năng đơn hay chung (SPP/GPP)
- (a) Nhiều bộ xử lý, mỗi bộ xử lý thực hiện một quá trình
  - Đa nhiệm đúng nghĩa (xử lý song song)
  - Bộ xử lý chức năng chung
    - Sử dụng ngôn ngữ lập trình như C và biên dịch thành các lệnh cho bộ xử lý
    - Đắt, trong nhiều trường hợp không cần thiết
  - Bộ xử lý chức năng đơn chuyên dụng
    - Hay dùng hơn
- (b) Một bộ xử lý chức năng chung chạy tất cả các quá trình
  - Đa số các quá trình không sử dụng 100% thời gian của bộ xử lý
  - Có thể chia sẻ thời gian của bộ xử lý và vẫn đạt được mục đích
- (c) Kết hợp (a) và (b)
  - Nhiều quá trình chạy trên một bộ xử lý chức năng chung trong khi đó một vài quá trình sử dụng bộ xử lý chức năng đơn chuyên dụng



Thực hiện

- Quá trình đưa các chức năng của hệ lên phần cứng của bộ xử lý:
  - Mô tả sử dụng mô hình tính toán(s)
  - Viết bằng một số ngôn ngữ(s)
- Lựa chọn thực hiện độc lập với lựa chọn ngôn ngữ
- Lựa chọn thực hiện dựa trên công suất, kích thước, chất lượng, thời gian và giá thành yêu cầu
- Thực hiện cuối cùng cần được kiểm tra tính khả thi
  - Là bản thử nghiệm cho việc sản xuất hàng loạt sản phẩm cuối cùng



Thực hiện:
Nhiều quá trình chia sẻ một bộ xử lý

- Có thể viết lại các quá trình như là một chương trình tuần tự
  - Thực hiện được với các trường hợp đơn giản, nhưng rất khó với các trường hợp phức tạp
  - Kỹ thuật tự động đã ra đời nhưng không thông dụng
  - VD: cương trình đồng thời Hello World có thể viết:

```
I = 1; T = 0;
while (1) {
    Delay(I); T = T + 1;
    if X modulo T is 0 then call PrintHelloWorld
    if Y modulo T is 0 then call PrintHowAreYou
}
```

- Có thể dùng hệ điều hành đa nhiệm
  - Thông dụng hơn
  - Hệ điều hành lập lịch cho các quá trình, định vị bộ nhớ, giao diện ngoại vi, etc.
  - Hệ điều hành thời gian thực (RTOS) có thể thực hiện điều đó
  - Mô tả các quá trình đồng thời với ngôn ngữ có các quá trình built-in (Java, Ada, etc.) hoặc một ngôn ngữ lập trình tuần tự với các thư viện hỗ trợ các quá trình (C, C++, etc)
- Có thể biến đổi các quá trình thành chương trình tuần tự với lập lịch trong code
  - Ít tốn bộ nhớ (không cần OS)
  - Chương trình phức tạp hơn

Tóm tắt

- Mô hình tính toán khác với ngôn ngữ
- Mô hình lập trình tuần tự là phổ biến
  - Ngôn ngữ phổ thông nhất như là C
- Mô hình trạng thái máy tốt cho điều khiển
  - Các mở rộng như HCFSM cung cấp thêm nhiều chức năng
  - PSM kết hợp trạng thái máy và chương trình tuần tự
- Mô hình quá trình đồng thời sử dụng cho các hệ thống nhiều tác vụ
  - Tồn tại truyền thông và phương pháp đồng bộ
  - Lập lịch là rất quan trọng
- Mô hình tuyến dữ liệu tốt cho xử lý tín hiệu