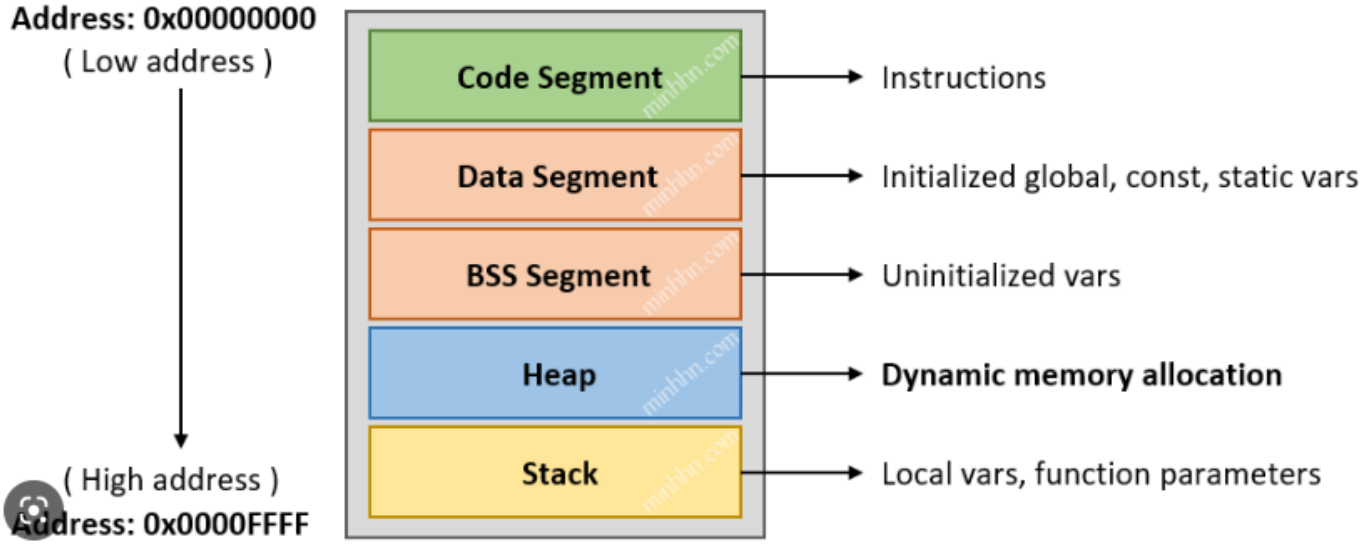
C Program

I. Phân vùng nhớ trong C

1. Phân vùng nhớ

- Gồm 5 phân vùng nhớ : Stack, Heap, BSS, DS, Text



- Text(Code segment)

+ Tất cả các mã máy trong chương trình sẽ được load vào vùng này

+ Là vùng chứa lệnh để thực thi trong chương trình

+ Chỉ có quyền truy cập Read để tránh sửa đổi

- DS(Initialized data segment)

+ Có quyền truy cập read-write

+ Vùng chứa biến toàn cục hoặc static với giá trị khởi tạo khác 0

+ Được giải phóng bộ nhớ khi kết thúc chương trình

+ VD :

int x = 1;

static int y = 2;

- BSS(Uninitialized data segment)

+ Có quyền truy cập read-write

+ Vùng chứa biến toàn cục hoặc static với giá trị khởi tạo bằng không hoặc không khởi tạo

+ Được giải phóng bộ nhớ khi kết thúc chương trình

+ VD :

int x;

stactic int y = 0;

- Heap(Dynamic Memory Allocation)

+ Có quyền truy cập read-write
+ Được sử dụng để cấp phát bộ nhớ động như Malloc, Calloc...
+ Sẽ được giải phóng khi gọi hàm free

- Stack(Automnatic variable storage)
+ Có cấu trúc LIFO(Last in, first out) Strucure
+ Có quyền truy cập read-write
+ Được sử dụng cấp phát cho biến local, input parameter ...
+ Sẽ được giải phóng khi ra khỏi block code/hàm

2. So sánh

- Sự khác biệt giữa stack và heap
+ Giống nhau : đều là vùng nhớ được lưu trữ trong RAM khi chương trình được thực thi
+ Khác nhau :
+) Stack :
. Lưu trữ các biến cục bộ trong hàm, tham số truyền vào...
. Truy cập vào bộ nhớ nhanh khi được thực hiện chương trình
. Vùng nhớ được quản lý bởi hệ điều hành, sẽ được giải phóng khi kết thúc chương trình
+) Head
. Lưu trữ vùng nhớ cho các biến con trỏ được cấp phát động bởi các hàm malloc, calloc..
. Vùng nhớ chỉ được giải phóng khi được gọi hàm free, nếu không bộ nhớ sẽ liên tục được tăng lên và có nguy cơ bị tràn

- Sự khác biệt giữa DS và BSS
+ Giống nhau
+) Đều có quyền truy cập read-write
+) Được giải phóng khi kết thúc chương trình
+ Khác nhau :
+) DS : Là vùng chưa biến toàn cục hoặc static với giá trị khởi tạo khác 0
+) BSS : Là vùng chứa biến toàn cục hoặc static với giá trị khởi tạo bằng không hoặc không khởi tạo

II. Macro and Funcion

1. Macro

- Được xử lý bởi preprocessor
- Thay thế đoạn code được khai báo Macro vào bất kì chỗ nào xuất hiện Macro đó
- Được khai báo bằng từ khóa **#define**
- Được ứng dụng để định nghĩa các biến const, hàm …
- VD : #define SUM(a,b) (a+b)

2. Funcion

- Là khối lệnh thực hiện một chức năng nào đó trong chương trình
- Được xử lý bằng compiler khi hàm được gọi
- Khi hàm được gọi, compiler sẽ phải lưu con trỏ chương trình PC(Program counter) hiện tại vào Stack pointer,
- Chuyển PC tới hàm được gọi, thực hiện hàm đó xong rồi lấy kết quả trả về sau đó quay lại vị trí ban đầu trong Stack trước khi gọi hàm và tiếp tục thực hiện chương trình

3. So sánh Macro và Funcion

- Macro :

+ Chỉ thay thế đoạn code Macro vào chỗ được gọi trước khi được biên dịch.
+ Tăng kích thước của file code trong chương trình vì Macro không có lưu trữ cố định trong bộ nhớ RAM
+ Không xác nhận được kiểu dữ liệu của tham số hoặc giá trị nhận được
+ Không thể debug tìm lỗi trong khi chương trình được chạy vì không được xử lý bởi compiler

- Funcion :

+ Phải tạo một Funcion call, lưu địa chỉ trước khi gọi hàm vào bộ nhớ của RAM
+ Do phải gọi Funcion call nên tốn thời gian do khi Funcion được gọi thì complier sẽ lưu con trỏ chương trình hiện tại vào Stack Pointer, tiếp theo là thực hiện các chức năng của Funcion để lấy kết quả trả về và trở về vị trí ban đầu của Stack Pointer trước khi gọi Funcion để tiếp tục chương trình
+ Phải xác nhận được kiểu dữ liệu của tham số hoặc giá trị nhận được
+ Có thể debug để tìm ra lỗi trong quá trình thực thi

III.So sánh giữa Struct and Union

- Giống nhau : có cách sử dụng và ý nghĩa giống nhau
- Khác nhau : khác nhau về mặt lưu trong dữ liệu trong bộ nhớ

+ Struct :

+) Dữ liệu các thành viên của struct được lưu trữ ở những vùng nhớ khác nhau
+) Kích thước của 1 struct tối thiểu bằng kích thước của các thành viên cộng lại và phụ thuộc vào bộ nhớ đệm (struct padding)

+ Union :

+) Dữ liệu các thành viên sẽ chung 1 vùng nhớ
+ Kích thước của Union sẽ được tính bằng kích thước lớn nhất của kiểu dữ liệu trong Union.
+ Việc thay đổi nội dung của 1 thành viên sẽ dẫn đến thay đổi nội dung các thành viên khác

IV. Static

1. Định nghĩa :

- Được khai báo bằng từ khóa : ***Static + kiểu dữ liệu***
- Có phạm vi truy cập trong một file,hàm , không thể truy cập từ file,hàm khác

2. Biến Stactic cục bộ

- Được khởi tạo 1 lần duy nhất và tồn tại trong thời gian chạy chương trình
- Giá trị không bị mất đi ngay ca khi kết thúc hàm, và mỗi lần hàm được gọi thì giá trị của biến chính là giá trị lần gần nhất được gọi
- Chỉ có thể được gọi nội bộ trong hàm khởi tạo ra nó

3. Biến static toàn cục

- Chỉ có thể truy cập và sử dụng trong file được khởi tạo nó, các file khác không thể truy cập

V. Con trỏ

1. Con trỏ (Pointer)

- Được khai báo bằng cách : kiểu dữ liệu + '*'
- Là những biến được lữu trữ địa chỉ bộ nhớ của những biến khác
- Do là biến được lưu dữ địa chỉ bộ nhớ nên tất cả các con trỏ đều có kích thước bằng nhau và phụ thuộc vào kiến trúc của VXL

2. Con trỏ hàm

- Kiểu khai báo : type + (*ptr)(type,…)
- Là con trỏ lưu trữ địa chỉ của 1 hàm trong bộ nhớ máy
- Được sử dụng để truy cập vào vị trí của hàm trong bộ nhớ cũng như thực thi các lệnh của hàm
- Kiểu dữ liệu khai báo và kiểu dữ liệu trong hàm con trỏ phải tương được
- VD : int (*ptr)(int x)

3. Con trỏ void

- Kiểu khai báo : void *ptr
- Là con trỏ đặc biệt, khi khai báo sẽ không cần xác định kiểu dữ liệu
- Có thể lưu trự địa chỉ của bất kì biến nào (với nhiều kiểu dữ liệu khác nhau) trong chương trình
- Vì không xác nhận kiểu dữ liệu nên khi sử dụng dể đọc hoặc ghi thì cần phải ép kiểu cho cùng loại dữ liệu muốn đọc hoặc ghi
- VD :

```
int value = 5;
void *ptr = &value;
uint32_t *ptr2 = (uint32_t *)(*ptr)
```

4. Con trỏ Null

- Là con trỏ không trỏ tới địa chỉ nào cả và có giá trị là 0
- Ứng dụng : khi khai báo con trỏ mà không biết gán giá trị nào thì ta gán vào giá trị NULL
- VD : int *ptr = NULL;

5. Mảng con trỏ

- Khai báo : type *tên mảng[]
- Là mảng mà mỗi phần tử đều là một con trỏ và có địa chỉ khác nhau
- VD : int *arr[] = { 1,2,3}

6. Pointer to Pointer

- Kiểu khai báo : **ptr
- Là loại con trỏ dùng để lưu trữ địa chỉ của con trỏ khác hay giá trị của con trỏ **ptr chính là địa chỉ của *ptr
- VD : int **ptr1 = &ptr2

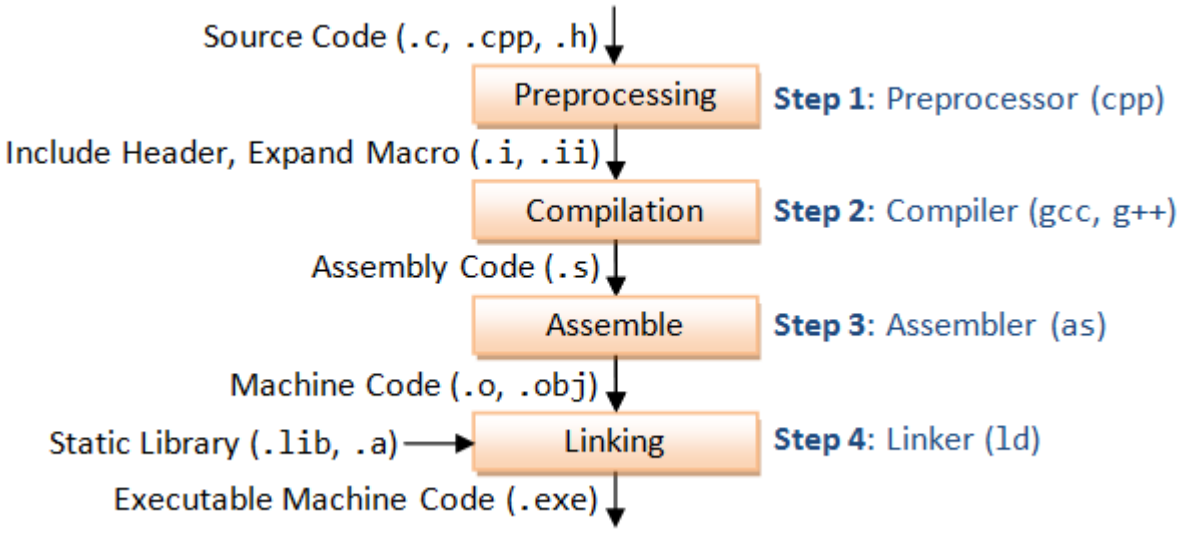
VI. Compiler

1. Compiler :

- Quy trình dịch là quá trình chuyển đổi từ ngôn ngữ bậc cao (NNBC) (C/C++, Pascal, Java, C#…) sang ngôn ngữ đích (ngôn ngữ máy) để máy tính có thể hiểu và thực thi. Ngôn ngữ lập trình C là một ngôn ngữ dạng biên dịch. Chương trình được viết bằng C muốn chạy được trên máy tính phải trải qua một quá trình biên dịch để chuyển đổi từ dạng mã nguồn sang chương trình dạng mã thực thi. Quá trình được chia ra làm 4 giai đoạn chính:

+ Giai đoàn tiền xử lý (Pre-processor)
+ Giai đoạn dịch NNBC sang Asembly (Compiler)
+ Giai đoạn dịch asembly sang ngôn ngữ máy (Asember)
+ Giai đoạn liên kết (Linker)

- Sơ đồ :



2. Các giai đoạn của Compiler

- Giai đoạn tiền xử lý Preprocessor

Giai đoạn này sẽ thực hiện:

+ Nhận mã nguồn
+ Xóa bỏ tất cả chú thích, comments của chương trình
+ Chỉ thị tiền xử lý (bắt đầu bằng #) cũng được xử lý
+ *Ví dụ*: chỉ thị #include cho phép ghép thêm mã chương trình của một tệp tiêu đề vào mã nguồn cần dịch. Các hằng số được định nghĩa bằng #define sẽ được thay thế bằng giá trị cụ thể tại mỗi nơi sử dụng trong chương trình.

2. Công đoạn dịch Ngôn Ngữ Bậc Cao sang Assembly

+ Phân tích cú pháp (syntax) của mã nguồn NNBC
+ Chuyển chúng sang dạng mã Assembly là một ngôn ngữ bậc thấp (hợp ngữ) gần với tập lệnh của bộ vi xử lý.

3. Công đoạn dịch Assembly

+ Dịch chương trình => Sang mã máy 0 và 1
+ Một tệp mã máy (.obj) sinh ra trong hệ thống sau đó.

4. Giai đoạn Linker

+ Trong giai đoạn này mã máy của một chương trình dịch từ nhiều nguồn (file .c hoặc file thư viện .lib) được liên kết lại với nhau để tạo thành chương trình đích duy nhất
+ Mã máy của các hàm thư viện gọi trong chương trình cũng được đưa vào chương trình cuối trong giai đoạn này.
+ Chính vì vậy mà các lỗi liên quan đến việc gọi hàm hay sử dụng biến tổng thể mà không tồn tại sẽ bị phát hiện. Kể cả lỗi viết chương trình chính không có hàm main() cũng được phát hiện trong liên kết.
+ Kết thúc quá trình tất cả các đối tượng được liên kết lại với nhau thành một chương trình có thể thực thi được (executable hay .exe) thống nhất.

C++ Program

Class C++

1. Class là gì?

Class hay lớp là một mô tả trừu tượng (abstract) của nhóm các đối tượng (object) có cùng bản chất, ngược lại mỗi một đối tượng là một thể hiện cụ thể (instance) cho những mô tả trừu tượng đó

- Một class bao gồm các thành phần dữ liệu (thuộc tính hay property) và các phương thức (hàm thành phần hay method).
- Class thực chất là một kiểu dữ liệu do người lập trình định nghĩa.
- Trong C++, từ khóa class sẽ chỉ điểm bắt đầu của một class sẽ được cài đặt

2. Khai báo class

Bao gồm Class + tên Class

3. Phạm vi truy cập

- Public : Các thành phần mang thuộc tinh này đều có thể được truy cập từ bất kỳ hàm nào, dù ở trong hay ngoài lớp. Các thuộc tính nên khai báo là public nếu bạn không có ràng buộc điều kiện trước khi gán (người dùng có thể thoải mái gán giá trị) hoặc bạn không cần xử lý trước khi trả về giá trị thuộc tính
- Private : các thành phần mang thuộc tính này đều bị hạn chế thay đổi từ người sử dụng. Chỉ có thể truy cập gián tiếp thông qua các thuộc tính Public của Class khởi tạo nó
- Đối với protected, các phương thức và thuộc tính chỉ có thể truy cập qua các class kế thừa nó hoặc chính nó

4. Contructor

Constructor : là một phương thức đặc biệt được gọi tự động tại thời điểm tạo đối tượng. Nó được sử dụng để khởi tạo các thành viên dữ liệu của các đối tượng mới. Constructor trong C++ có cùng tên với Class

- Constructor mặc định :
- Constructor có tham số : Nó được sử dụng để cung cấp các giá trị khác nhau cho các đối tượng riêng biệt.

5. Static member

Static member hay thành viên tĩnh trong class C++ cũng tương tự như với static variable (biến tĩnh) trong function. Đối với class, thành viên tĩnh sẽ là thuộc tính dùng chung cho tất cả các đối tượng của class đó, cho dù là không có đối tượng nào tồn tại. Tức là bạn có thể khai báo nhiều object, mỗi object các thuộc tính của nó đều khác nhau nhưng riêng static thì chỉ có một và static member tồn

tại trong suốt chương trình cho dù có hay không có object nào của nó hay nói ngắn gọn là dùng chung một biến static

6. Đặc tính của đối tượng

Inheritance (Tính kế thừa) trong lập trình hướng đối tượng có ý nghĩa, một class có thể kế thừa các thuộc tính của một class khác đã tồn tại trước đó.
VD : Class HS : tên, tuổi
Class kế thừa là Class TTHS : tên, tuổi, lớp

Abstraction (Tính trừu tượng) trong lập trình hướng đối tượng là một khả năng mà chương trình có thể bỏ qua sự phức tạp bằng cách tập trung vào cốt lõi của thông tin cần xử lý. Tức là chỉ cần gọi tên phương thức và lấy kết quả mà không cần quan tâm đến cách đối tượng hoạt động
VD : Class giải phương trình bậc 2
Bỏ qua sự phức tạp : công thức giải phương trình bậc 2, đơn giản hóa bằng cách nhập các tham số của phương trình để lấy kết quả

Polymorphism (Tính đa hình) trong lập trình hướng đối tượng là một khả năng mà một phương thức trong class có thể đưa ra các kết quả hoàn toàn khác nhau, tùy thuộc vào dữ liệu được xử lý
VD : phương thức tính tổng 2 số, tổng 3 số

Encapsulation (Tính đóng gói) trong lập trình hướng đối tượng có ý nghĩa không cho phép người sử dụng thay đổi trạng thái nội tại của một đối tượng, mà chỉ có phương thức nội tại của đối tượng có thể thay đổi chính nó. Tức là thông tin sẽ được đóng gói lại, không cho các tác động bên ngoài làm thay đổi đối tượng, bảo toàn tính toàn vẹn của đối tượng và giấu đi những thông tin quan trong của đối tượng
VD : Class Thông tin tài khoản ngân hàng : Tên, tuổi, số tài khoản, CMT

Namespace

Namespace là từ khóa trong C++ được sử dụng để định nghĩa một phạm vi nhằm mục đích phân biệt các hàm, lớp, biến, ... cùng tên trong các thư viện khác nhau

Template trong C++ là gì?

- *Template* (khuôn mẫu) là một từ khóa trong C++, và là một kiểu dữ liệu trừu tượng tổng quát hóa cho các kiểu dữ liệu int, float, double, bool...
- Template trong C++ có 2 loại đó là function template & class template.
- Template giúp người lập trình định nghĩa tổng quát cho hàm và lớp thay vì phải nạp chồng (overloading) cho từng hàm hay phương thức với những kiểu dữ liệu khác nhau

Embedded

1. SPI

Định nghĩa : SPI (Serial Peripheral Interface) là một chuẩn truyền thông nối tiếp tốc độ cao do Motorola đề xuất.

- Các bit dữ liệu được truyền nối tiếp nhau và có xung clock đồng bộ.
- Giao tiếp song công, có thể truyền và nhận cùng một thời điểm.
- Khoảng cách truyền ngắn, được sử dụng để trao đổi dữ liệu với nhau giữa các chip trên cùng một bo mạch.
- Tốc độ truyền khoảng vài Mb/s.
- Thường được dùng để giao tiếp với các ngoại vị hoặc với các vi điều khiển khác

Giao tiếp SPI : Bus SPI gồm có 4 đường tín hiệu

- SCK (Serial Clock): Thiết bị Master tạo xung tín hiệu SCK và cung cấp cho Slave. Xung này có chức năng giữ nhịp cho giao tiếp SPI. Mỗi nhịp trên chân SCK báo 1 bit dữ liệu đến hoặc đi → Quá trình ít bị lỗi và tốc độ truyền cao
- MISO (Master Input Slave Output): Tín hiệu tạo bởi thiết bị Slave và nhận bởi thiết bị Master. Đường MISO phải được kết nối giữa thiết bị Master và Slave
- MOSI (Master Output Slave Input): Tín hiệu tạo bởi thiết bị Master và nhận bởi thiết bị Slave. Đường MOSI phải được kết nối giữa thiết bị Master và Slave
- SS (Slave Select): Chọn thiết bị Slave cụ thể để giao tiếp. Để chọn Slave giao tiếp thiết bị Master chủ động kéo đường SS tương ứng xuống mức 0 (Low). Chân này đôi khi còn được gọi là CS (Chip Select). Chân SS của vi điều khiển (Master) có thể được người dùng tạo bằng cách cấu hình 1 chân GPIO bất kỳ chế độ Output

Khung truyền SPI:

- Mỗi chip Master hay Slave đều có một thanh ghi dữ liệu 8 bits.
- Quá trình truyền nhận giữa Master và Slave xảy ra đồng thời sau 8 chu kỳ đồng hồ, một byte dữ liệu được truyền theo cả 2 hướng
- Quá trình trao đổi dữ liệu bắt đầu khi Master tạo 1 xung clock từ bộ tạo xung nhịp (Clock Generator) và kéo đường SS của Slave mà nó truyền dữ liệu xuống mức Low.
- Cứ 1 xung clock, Master sẽ gửi đi 1 bit từ thanh ghi dịch (Shift Register) của nó đến thanh ghi dịch của Slave thông qua đường MOSI. Đồng thời Slave cũng gửi lại 1 bit đến cho Master qua đường MISO.Như vậy sau 8 chu kỳ clock thì hoàn tất việc truyền và nhận 1 byte dữ liệu.
- Dữ liệu của 2 thanh ghi được trao đổi với nhau nên tốc độ trao đổi diễn ra nhanh và hiệu quả.
- Lưu ý: Trong giao tiếp SPI, chỉ có thể có 1 Master nhưng có thể 1 hoặc nhiều Slave cùng lúc. Ở trạng thái nghỉ, chân SS của các Slave ở mức 1, muốn giao tiếp với Slave nào thì ta chỉ việc kéo chân SS của Slave đó xuống mức 0.
-

Chế độ hoạt động:

SPI có 4 chế độ hoạt động phụ thuộc vào cực của xung giữ (Clock Polarity – CPOL) và pha (Phase - CPHA).

- Mode 0 (măc định) – xung nhịp của đồng hồ ở mức thấp (CPOL = 0) và dữ liệu được lấy mẫu khi chuyển từ thấp sang cao (cạnh lên) (CPHA = 0)

- Mode 1 - xung nhịp của đồng hồ ở mức thấp (CPOL = 0) và dữ liệu được lấy mẫu khi chuyển từ cao sang thấp (cạnh xuống) (CPHA = 1).
- Mode 2 - xung nhịp của đồng hồ ở mức cao (CPOL = 1) và dữ liệu được lấy mẫu khi chuyển từ cao sang thấp (cạnh lên) (CPHA = 0).
- Mode 3 - xung nhịp của đồng hồ ở mức cao (CPOL = 1) và dữ liệu được lấy mẫu khi chuyển từ thấp sang cao (cạnh xuống) (CPHA = 1).

2. I2C

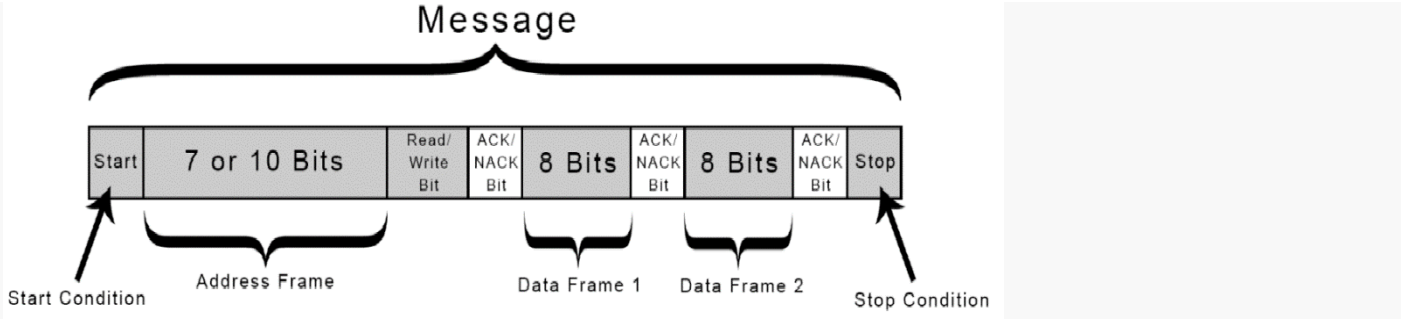
Định nghĩa :

- I2C (Inter – Integrated Circuit) là 1 giao thức giao tiếp nối tiếp đồng bộ được phát triển bởi Philips Semiconductors, sử dụng để truyền nhận dữ liệu giữa các IC với nhau chỉ sử dụng hai đường truyền tín hiệu.
- Các bit dữ liệu sẽ được truyền từng bit một theo các khoảng thời gian đều đặn được thiết lập bởi 1 tín hiệu đồng hồ.
- Bus I2C thường được sử dụng để giao tiếp ngoại vi cho rất nhiều loại IC khác nhau như các loại vi điều khiển, cảm biến, EEPROM, …

Giao tiếp I2C : Bus I2C gồm có 2 đường tín hiệu

- SDA (Serial Data) - đường truyền cho master và slave để gửi và nhận dữ liệu.
- SCL (Serial Clock) - đường mang tín hiệu xung nhịp.

Cách hoạt động của I2C



- Điều kiện khởi động: Đường SDA chuyển từ mức điện áp cao xuống mức điện áp thấp trước khi đường SCL chuyển từ mức cao xuống mức thấp.
- Điều kiện dừng: Đường SDA chuyển từ mức điện áp thấp sang mức điện áp cao sau khi đường SCL chuyển từ mức thấp lên mức cao.
- Khung địa chỉ: Một chuỗi 7 hoặc 10 bit duy nhất cho mỗi slave để xác định slave khi master muốn giao tiếp với nó
- Bit Đọc / Ghi: Một bit duy nhất chỉ định master đang gửi dữ liệu đến slave (mức điện áp thấp) hay yêu cầu dữ liệu từ nó (mức điện áp cao).
- Bit ACK / NACK: Mỗi khung trong một tin nhắn được theo sau bởi một bit xác nhận / không xác nhận. Nếu một khung địa chỉ hoặc khung dữ liệu được nhận thành công, một bit ACK sẽ được trả lại cho thiết bị gửi từ thiết bị nhận.

- Địa chỉ : I2C không có các đường Slave Select như SPI, vì vậy cần một cách khác để cho slave biết rằng dữ liệu đang được gửi đến slave này chứ không phải slave khác. Nó thực hiện điều này bằng cách định địa chỉ. Khung địa chỉ luôn là khung đầu tiên sau bit khởi động trong một tin nhắn mới.

Các bước truyền dữ liệu I2C

- Master gửi điều kiện khởi động đến mọi slave được kết nối bằng cách chuyển đường SDA từ mức điện áp cao sang mức điện áp thấp trước khi chuyển đường SCL từ mức cao xuống mức thấp.
- Master gửi cho mỗi slave địa chỉ 7 hoặc 10 bit của slave mà nó muốn giao tiếp, cùng với bit đọc / ghi.
- Mỗi slave sẽ so sánh địa chỉ được gửi từ master với địa chỉ của chính nó. Nếu địa chỉ trùng khớp, slave sẽ trả về một bit ACK bằng cách kéo dòng SDA xuống thấp cho một bit. Nếu địa chỉ từ master không khớp với địa chỉ của slave, slave rời khỏi đường SDA cao.
- Master gửi hoặc nhận khung dữ liệu.
- Sau khi mỗi khung dữ liệu được chuyển, thiết bị nhận trả về một bit ACK khác cho thiết bị gửi để xác nhận đã nhận thành công khung.
- Để dừng truyền dữ liệu, master gửi điều kiện dừng đến slave bằng cách chuyển đổi mức cao SCL trước khi chuyển mức cao SDA.

3. Ngắt (interrupt)

Định nghĩa : là một số sự kiện khẩn cấp bên trong hoặc bên ngoài bộ vi điều khiển xảy ra, buộc vi điều khiển tạm dừng thực hiện chương trình hiện tại, phục vụ ngay lập tức nhiệm vụ mà ngắt yêu cầu – nhiệm vụ này gọi là trình phục vụ ngắt (**ISR:** Interrupt Service Routine).

Trình phục vụ ngắt :

Đối với mỗi ngắt thì phải có một trình phục vụ ngắt (ISR) hay trình quản lý ngắt để đưa ra nhiệm vụ cho bộ vi điều khiển khi được gọi ngắt. Khi một ngắt được gọi thì bộ vi điều khiển sẽ chạy trình phục vụ ngắt. Đối với mỗi ngắt thì có một vị trí cố định trong bộ nhớ để giữ địa của nó. Nhóm vị trí bộ nhớ được dành riêng để lưu giữ địa chỉ của các **ISR** được gọi là **bảng vector ngắt**

Ngắt	Cờ ngắt	Địa chỉ trình phục vụ ngắt	Số thứ tự ngắt
Reset	-	0000h	-
Ngắt ngoài 0	IE0	0003h	0
Timer 0	TF0	000Bh	1
Ngắt ngoài 1	IE1	0013h	2
Timer 1	TF1	001Bh	3
Ngắt truyền thông	RI/TI	0023h	4

Ngắt Reset : Là ngắt có ưu tiên cao nhất, khi nó được thực hiện Program counter sẽ trỏ tới địa chỉ 0x0000 để reset và đồng bộ lại chương trình

Ngắt Ngoài : Có 4 loại ngắt ngoài :

- Ngắt ngoài tích cực thấp
- Ngắt ngoài tích cực cao
- Ngắt xung cạnh lên
- Ngắt xung cạnh xuống

Ngắt Timer :

- Đếm các xung nhịp bên trong IC : Dùng để đếm các xung nhịp bên trong của IC, các xung nhịp này thường bằng nhau và cách đều nên thường sử dụng để đếm thời gian (bộ định thời)
- Đếm các sự kiện bên ngoài vi điều khiển : dùng để đếm các sự thay đổi trạng thái của các tín hiệu bên ngoài. Các tín hiệu này thường không đồng đều nên sử dụng trong bộ đếm counter

Ngắt truyền thông : là ngắt được sử dụng trong các giao thức truyền thông như SPI, I2C, UART

Quy trình khi thực hiện một ngắt

Khi kích hoạt một ngắt bộ vi điều khiển thực hiện các bước sau:

- Nó hoàn thành nốt lệnh đang thực hiện và lưu địa chỉ của **lệnh kế tiếp** vào ngăn xếp. Đồng thời nó cũng **lưu tình trạng hiện tại** của tất cả các ngắt.
- Tiếp đến bộ vi điều khiển nhận địa chỉ **ISR** từ bảng vector ngắt và nhảy tới đó. Nó bắt đầu thực hiện trình phục vụ ngắt cho đến lệnh cuối cùng của **ISR**
- Sau khi hoàn thành các nhiệm vụ của ngắt, vi điều khiển quay trở về chương trình chính nơi nó đã bị ngắt và bắt đầu thực hiện tiếp các lệnh từ địa chỉ đó.

Mức ưu tiên ngắt

Hoạt động dựa trên vào mức độ ưu tiên ngắt khác nhau, giá trị càng nhỏ thì mức ưu tiên càng cao

4. UART

Định nghĩa : UART (Universal Asynchronous Receiver-Transmitter – Bộ truyền nhận dữ liệu không đồng bộ) là một giao thức truyền thông phần cứng dùng giao tiếp nối tiếp không đồng bộ và có thể cấu hình được tốc độ

Sơ đồ : Trong một sơ đồ giao tiếp UART

- Chân Tx (truyền) của một chip kết nối trực tiếp với chân Rx (nhận) của chip kia và ngược lại. Thông thường, quá trình truyền sẽ diễn ra ở 3.3V hoặc 5V. UART là một giao thức một master, một slave, trong đó một thiết bị được thiết lập để giao tiếp với duy nhất một thiết bị khác.
- Dữ liệu truyền đến và đi từ UART song song với thiết bị điều khiển (ví dụ: CPU).
- Khi gửi trên chân Tx, UART đầu tiên sẽ dịch thông tin song song này thành nối tiếp và truyền đến thiết bị nhận.
- UART thứ hai nhận dữ liệu này trên chân Rx của nó và biến đổi nó trở lại thành song song để giao tiếp với thiết bị điều khiển của nó.

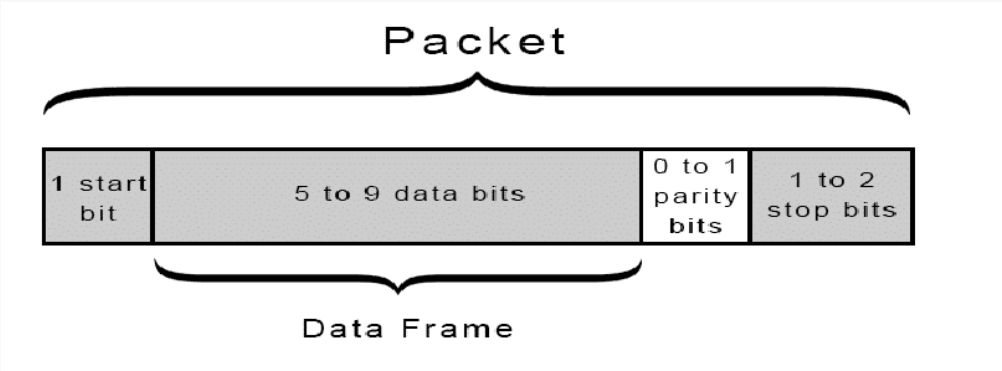
Chế độ truyền : UART truyền dữ liệu nối tiếp, theo một trong ba chế độ

- Full duplex: Giao tiếp đồng thời đến và đi từ mỗi master và slave
- Half duplex: Dữ liệu đi theo một hướng tại một thời điểm
- Simplex: Chỉ giao tiếp một chiều

Phương thức giao tiếp

Hai ngoại vị giao tiếp bằng UART thông qua tốc độ truyền (Baudrate).

Frame truyền



Bit bắt đầu : Đường truyền dữ liệu UART thường được giữ ở mức điện áp cao khi không truyền dữ liệu. Để bắt đầu truyền dữ liệu, UART truyền sẽ kéo đường truyền từ mức cao xuống mức thấp trong một chu kỳ clock. Khi UART nhận phát hiện sự chuyển đổi điện áp cao xuống thấp, nó bắt đầu đọc các bit trong khung dữ liệu ở tần số của tốc độ truyền.
Khung dữ liệu : Khung dữ liệu chứa dữ liệu thực tế được chuyển. Nó có thể dài từ 5 bit đến 8 bit nếu sử dụng bit chẵn lẻ. Nếu không sử dụng bit chẵn lẻ, khung dữ liệu có thể dài 9 bit. Trong hầu hết các trường hợp, dữ liệu được gửi với bit ít quan trọng nhất trước tiên.
Bit chẵn lẻ : Bit chẵn lẻ là một cách để UART nhận cho biết liệu có bất kỳ dữ liệu nào đã thay đổi trong quá trình truyền hay không. Bit có thể bị thay đổi bởi bức xạ điện từ, tốc độ truyền không khớp hoặc truyền dữ liệu khoảng cách xa. Sau khi UART nhận đọc khung dữ liệu, nó sẽ đếm số bit có giá trị là 1 và kiểm tra xem tổng số là số chẵn hay lẻ. Nếu bit chẵn lẻ là 0 (tính chẵn), thì tổng các bit 1 trong khung dữ liệu phải là một số chẵn. Nếu bit chẵn lẻ là 1 (tính lẻ), các bit 1 trong khung dữ liệu sẽ tổng thành một số lẻ. Khi bit chẵn lẻ khớp với dữ liệu, UART sẽ biết rằng quá trình truyền không có lỗi. Nhưng nếu bit chẵn lẻ là 0 và tổng là số lẻ; hoặc bit chẵn lẻ là 1 và tổng số là chẵn, UART sẽ biết rằng các bit trong khung dữ liệu đã thay đổi.
Bit dừng : Để báo hiệu sự kết thúc của gói dữ liệu, UART gửi sẽ điều khiển đường truyền dữ liệu từ điện áp thấp đến điện áp cao trong ít nhất khoảng 2 bit.

5. Timer

Định nghĩa : Bộ đếm/Bộ định thời: Đây là các ngoại vi được thiết kế để thực hiện một nhiệm vụ đơn giản: đếm các xung nhịp. Mỗi khi có thêm một xung nhịp tại đầu vào đếm thì giá trị của bộ đếm sẽ được tăng lên 01 đơn vị (trong chế độ đếm tiến/đếm lên) hay giảm đi 01 đơn vị (trong chế độ đếm lùi/đếm xuống).

Xung nhịp :
Xung nhịp đưa vào đếm có thể là một trong hai loại:
- Xung nhịp bên trong IC: Đó là xung nhịp được tạo ra nhờ kết hợp mạch dao động bên trong IC và các linh kiện phụ bên ngoài nối với IC. Trong trường hợp sử dụng xung nhịp loại này, người ta gọi là các **bộ định thời** (**timers**). Do xung nhịp bên loại này thường đều đặn nên ta có thể dùng để **đếm thời gian** một cách khá chính xác.
- Xung nhịp bên ngoài IC: Đó là các tín hiệu logic thay đổi liên tục giữa 02 mức 0-1 và không nhất thiết phải là đều đặn. Trong trường hợp này người ta gọi là các **bộ đếm** (**counters**). Ứng dụng phổ

biến của các bộ đếm là **đếm các sự kiện bên ngoài** như đếm các sản phẩm chạy trên băng chuyền, đếm xe ra/vào kho bãi…

Overflow :

Một khái niệm quan trọng cần phải nói đến là sự kiện "**tràn**" (**overflow**). Nó được hiểu là sự kiện bộ đếm đếm vượt quá giá trị tối đa mà nó có thể biểu diễn và quay trở về giá trị 0. Với bộ đếm 8 bit, giá trị tối đa là 255 (tương đương với FF trong hệ Hexa) và là 65535 (FFFFH) với bộ đếm 16 bit.

Cấu hình của Timer

1. Cấu hình đặc tính Timer :
   - Lựa chọn chân PIN dùng để sử dụng cho Timer
   - Lựa chọn xung nhịp đếm bên trong hoặc ngoài vi điều khiển

2. Cấu hình bộ đếm thời gian :

- Xác nhận tần số xung Clock của Timer đang sử dụng : 16 MHz, 32 MHz …
- Cấu hình 2 thông số Prescaler và Counter

Prescaler : là bộ chia với mục đích thay đổi thời gian cho mỗi lần đếm của Counter

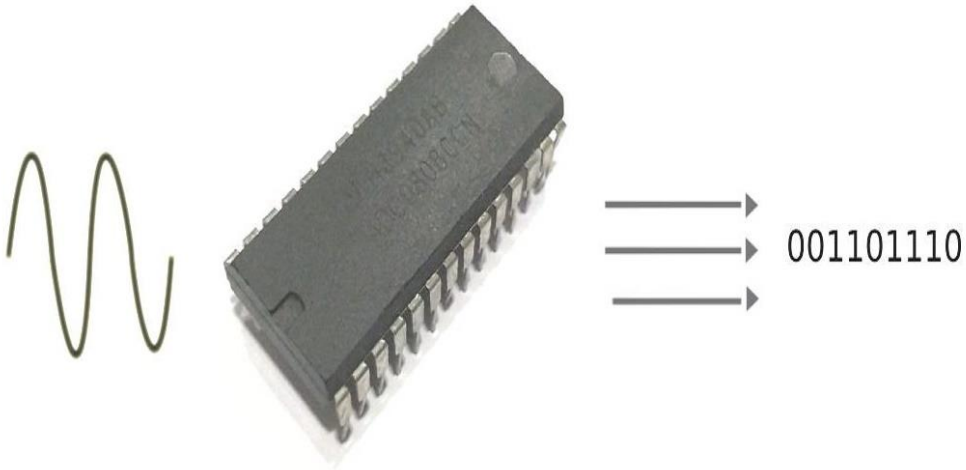
Counter : là số lần đếm của VXL để tạo được 1ms hoặc 1s. Giá trị của Counter sẽ phải đảm bảo không được vượt quá giá trị cài đặt trong thanh ghi AutoReload

3. Clear các cờ tràn, ngắt nếu có
4. Đăng kí vector ngắt cho Timer và kích hoạt ngắt nếu sử dụng

6. ADC

Bộ chuyển đổi ADC là gì

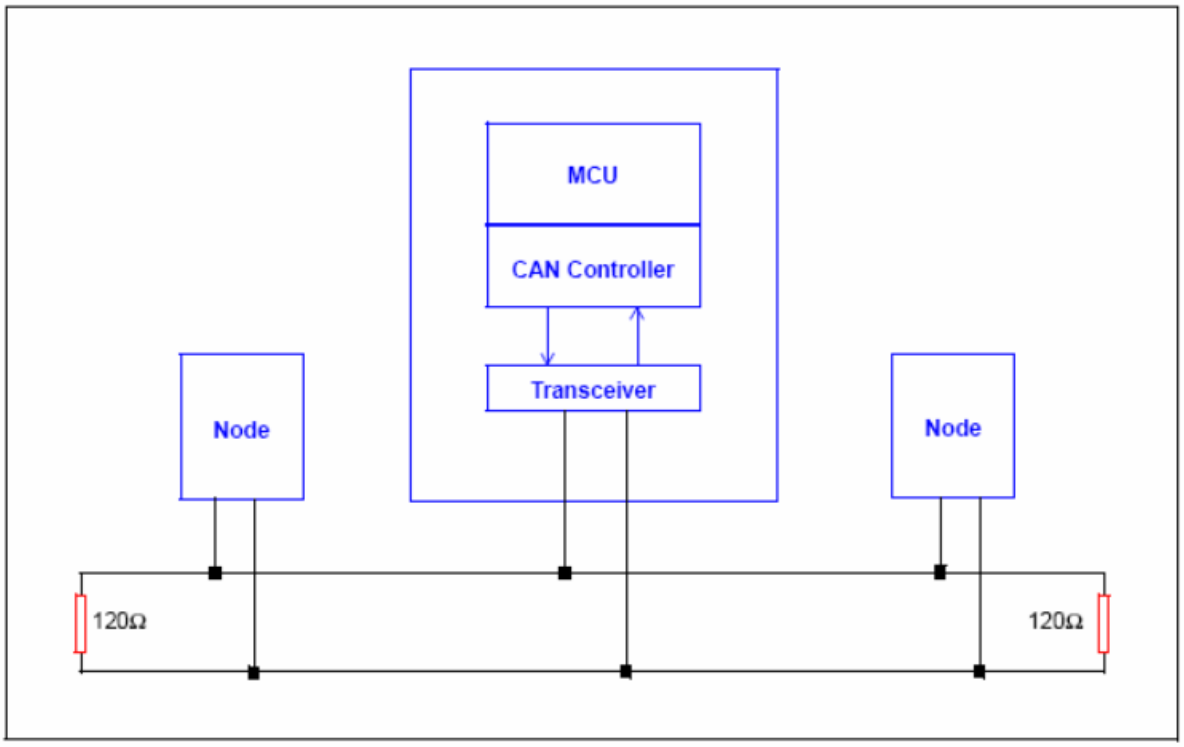
ADC là từ viết tắt của Analog to Digital Converter hay bộ chuyển đổi analog sang kỹ thuật số là một mạch chuyển đổi giá trị điện áp liên tục (analog) sang giá trị nhị phân (kỹ thuật số) mà thiết bị kỹ thuật số có thể hiểu được sau đó có thể được sử dụng để tính toán kỹ thuật số. Mạch ADC này có thể là vi mạch ADC hoặc được nhúng vào một bộ vi điều khiển.



7. CAN

Định nghĩa : Controller Area Network (CAN hoặc CAN Bus) là công nghệ mạng nối tiếp, tốc độ cao, bán song công, hai dây. Ban đầu CAN được thiết kế dành cho ngành công nghiệp ô tô, tuy nhiên hiện nay CAN cũng đã trở thành một tiêu chuẩn phổ biến trong tự động hóa công nghiệp và các ngành khác

Cấu hình CAN



CAN bao gồm 2 dây CANH và CANL được mắc vào nhau thông qua 2 điện trở 120Ω và các Node trong CAN sẽ dược mắc song song với nhau thông 2 qua dây đó

Trạng thái "dominant" và "recessive"

lớp vật lý, Bus CAN định nghĩa hai trạng thái là "dominant" và "recessive", tương ứng với hai trạng thái là 0 và 1. Trạng thái "dominant" chiếm ưu thế so với trạng thái "recessive". Bus chỉ ở trạng thái "reccessive" khi không có node nào phát đi trạng thái "dominant". Điều này tạo ra khả năng giải quyết tranh chấp khi nhiều hơn một Master cùng muốn chiếm quyền sử dụng bus.

Bởi tính chất vật lý của bus, cần thiết phải phân biệt 2 dạng truyền:

- Truyền CAN low speed
- Truyền CAN high speed

Thông số	CAN low speed	CAN high speed
Tốc độ	125 kb/s	125 kb/s tới 1Mb/s
số nút trên bus	2 tới 20	2 tới 30
Trạng thái dominant	CAN H = 4V ; CAN L = 1V	CAN H = 3,25V ; CAN L = 1,5V
Trạng thái recessive	CAN H = 1,75V ; CAN L = 3,25V	CAN H = 2,5V ; CAN L = 2,5V
tính chất của cap	30pF giữa cáp và dây	2*120 ohm
Mức điện áp cung cấp	5V	5V

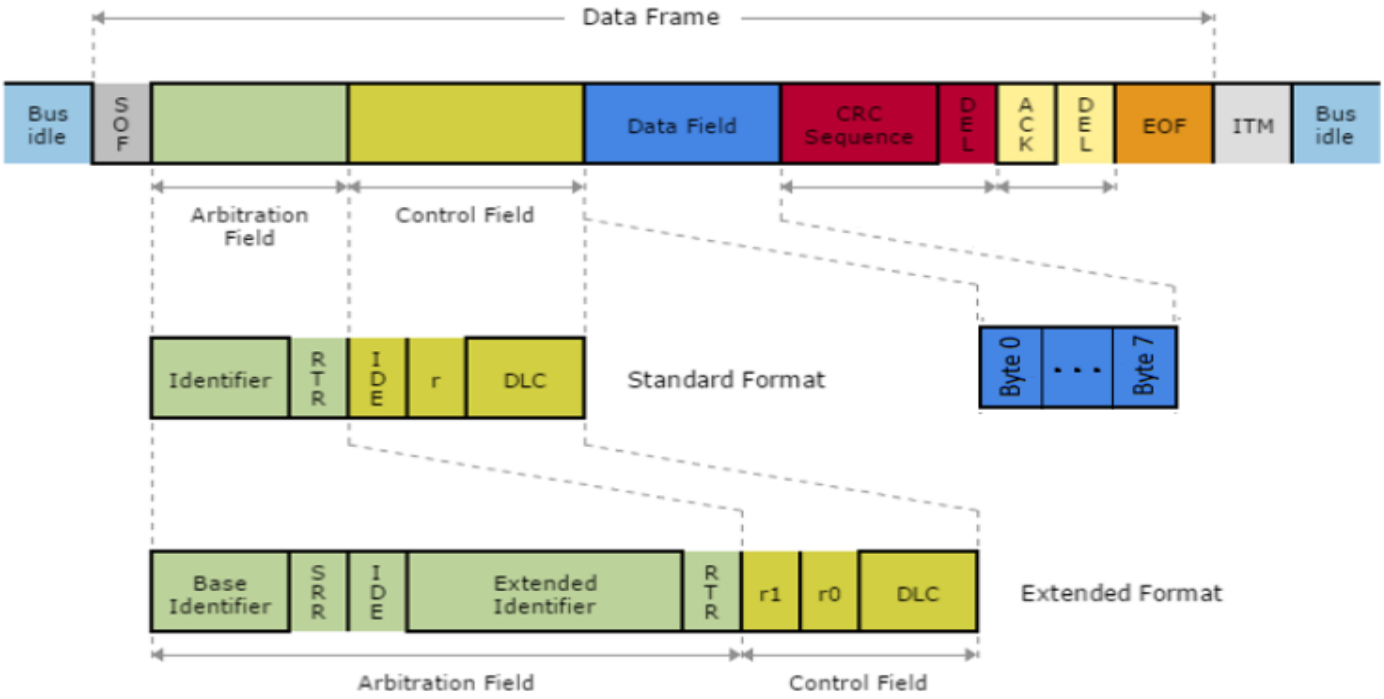
Các loại CAN Frame

Dữ liệu CAN được truyền dưới dạng các Frame (khung). Có 4 loại Frame khác nhau, đó là:

- Data Frame (khung dữ liệu): được dùng để truyền đi một message

- Remote Frame (khung yêu cầu hay điều khiển): dùng để truyền data frame tới một nút khác. Có DLC = 0 và không có data field
- Error Frame (khung lỗi): là khung được truyền bởi bất kỳ Node nào khi Node đó phát hiện lỗi từ Bus.
- Overflow Frame (khung báo tràn): được sử dụng khi frame bị tràn bộ đệm

Data Frame



1. Trường bắt đầu khung (Start Of Frame Field – SOF) : trường bắt đầu là vị trí của bit đầu tiên trong khung. Trường này chiếm 1 bit dữ liệu. Bit đầu tiên này là một Dominant Bit (mức logic 0) đánh dấu sự bắt đầu của một Data Frame

2. Trường xác định quyền ưu tiên (Arbitration Field) :

- Định dạng chuẩn: vùng xác định quyền ưu tiên có độ dài 12 bit, bao gồm 11 bit ID và 1 bit RTR.
- Định dạng mở rộng: trường xác định quyền ưu tiên có độ dài 32 bit, bao gồm có 29 bit ID, 1 bit SRR, 1 bit IDE và 1 bit RTR

* Trong đó:

- Bit RTR (Remote Transmission Request)
    - Là bit dùng để phân biệt khung là Data Frame hay Remote Frame.
    - Nếu là Data Frame, bit này luôn bằng 0 (Dominant Bit).
    - Nếu là Remote Frame, bit này luôn bằng 1 (Recessive Bit).
    - Vị trí bit này luôn nằm sau bit ID

Trường hợp nếu Data Frame và Remote Frame có cùng ID được gửi đi đồng thời thì Data Frame sẽ được ưu tiên hơn.

- Bit SRR (Substitute Remote Request)
    - Bit này chỉ có ở khung mở rộng
    - Bit này có giá trị là 1 (Recessive Bit)
    - So với vị trí tương ứng trong khung chuẩn thì bit này trùng với vị trí của bit RTR nên còn được gọi là bit thay thế (Substitute)

3. Trường điều khiển (Control Field) : Khung chuẩn và khung mở rộng có định dạng khác nhau ở trường này:

- Khung chuẩn gồm IDE, r0 và DLC (Data Length Code).
- Khung mở rộng gồm r1, r0 và DLC.

Bit IDE (Identifier Extension)

- Đây là bit phân biệt giữa loại khung chuẩn và khung mở rộng: IDE = 0 quy định khung chuẩn, IDE = 1 quy định khung mở rộng.
- Bit này nằm ở trường xác định quyền ưu tiên với khung mở rộng và ở trường điều khiển với khung chuẩn.

Bit r0, r1 (hai bit dự trữ)
DLC (Data Length Code)

- Có độ dài 4 bit quy định số byte của trường dữ liệu của Data Frame
- Chỉ được mang giá trị từ 0 đến 8 tương ứng với trường dữ liệu có từ 0 đến 8 byte dữ liệu. Data Frame có thể không có byte dữ liệu nào khi DLC = 0.
- Giá trị lớn hơn 8 không được phép sử dụng. Hình dưới mô tả các loại mã bit mà DLC có thể chứa để quy định số byte của trường dữ liệu

4. Trường dữ liệu (Data Frame) : Trường này có độ dài từ 0 đến 8 byte tùy vào giá trị của DLC ở trường điều khiển

5. Trường kiểm tra (Cyclic Redundancy Check Field – CRC) : Trường kiểm tra hay trường CRC gồm 16 bit và được chia làm hai phần là:

- CRC Sequence: gồm 15 bit CRC tuần tự
- CRC Delimiter: là một Recessive Bit làm nhiệm vụ phân cách trường CRC với trường ACK
- Mã kiểm tra CRC phù hợp nhất cho các khung mà chuỗi bit được kiểm tra có chiều dài dưới 127 bit, mã này thích hợp cho việc phát hiện các trường hợp sai nhóm (Bus Error). Ở đây,

tổng bit từ trường bắt đầu (SOF) đến trường dữ liệu (Data Field) tối đa là 83 bit (khung định dạng chuẩn) và 103 bit (khung định dạng mở rộng)

=> Trường CRC bảo vệ thông tin trong Data Frame và Remote Frame bằng cách thêm các bit kiểm tra dự phòng ở đầu khung truyền. Ở đầu khung nhận, cũng sẽ tính toán CRC như bộ truyền khi đã nhận dữ liệu và so sánh kết quả đó với CRC Sequence mà nó đã nhận được, nếu khác nhau tức là đã có lỗi, nếu giống nhau tức là đã nhận đúng từ trường SOF đến trường dữ liệu

6. Trường báo nhận (Acknowledge Field – ACK) : Trường báo nhận hay trường ACK có độ dài 2 bit và bao gồm hai phần là ACK Slot và ACK Delimiter.

- ACK Slot: có độ dài 1 bit, một Node truyền dữ liệu sẽ thiết lập bit này là Recessive. Khi một hoặc nhiều Node nhận chính xác giá trị thông điệp (không có lỗi và đã so sánh CRC Sequence trùng khớp) thì nó sẽ báo lại cho bộ truyền bằng cách truyền ra một Dominant Bit ngay vị trí ACK Slot để ghi đè lên Recessive Bit của bộ truyền.
- ACK Delimiter: có độ dài 1 bit, nó luôn là một Recessive Bit. Như vậy, ta thấy rằng ACK Slot luôn được đặt giữa hai Recessive Bit là CRC Delimiter và ACK Delimiter

7. Trường kết thúc (End Of Frame Field – EOF) : Trường EOF là trường thông báo kết thúc một Data Frame hay Remote Frame. Trường này gồm 7 Recessive Bit

* Kiến trúc của vi điều khiển

1. Bộ xử lý (CPU)

- CPU (Central processing unit) là bộ xử lý trung tâm chịu trách nhiệm nạp lệnh, giải mã và thực thi các tập lệnh.
- Tất cả các hoạt động của vi điều khiển đều do CPU điều khiển.
- CPU giao tiếp với các phần khác trong vi điều khiển thông qua hệ thống bus

2. Hệ thống xung clock

- Thực hiện việc cấp xung nhịp cho vi điều khiển, giúp cho vi điều khiển thực hiện các hoạt động của mình
- Tốc độ xung nhịp sẽ cho biết tốc độ xử lý, tính toán tối đa mà VĐK có thể đáp ứng được

3. Bộ nhớ :

- Là nơi lưu trữ chương trình hoặc thông tin mà CPU đang làm việc. Có 2 kiểu bộ nhớ cơ bản :

+ RAM (Random access memory) :

+)là bộ nhớ lưu trữ dữ liệu và cấu trúc tạm thời mà CPU đang hoạt động, giúp truy xuất dữ liệu nhanh hơn khi đang sử dụng VĐK

+) Dữ liệu trong RAM sẽ bị xóa khi mất điện

+ ROM (Read only memory) :

+) Là bộ lưu trữ chương trình vận hành của VĐK.

+) ĐƯợc ghi khi VĐK nạp chương trình

+) Nội dung không bị mất đi khi reset VĐK

4. Các ngoại vi(I/O Port)

- Các cổng I/O Port thực hiện các kết tiếp, tiếp nhận hay truyền dẫn dữ liệu với các thiết bị bên ngoài
- Các ngoại vị cơ bản của VĐK thường bao gồm :

+) Ngõ vào ra I/O Port

+) Các giao tiếp truyền thông phổ biến SPI, I2C...

+) Bộ đếm thời gian và sự kiện (timer và counter)

+) Bộ chuyển đổi dữ liệu (ADC/DAC)

+) Bộ xử lý ngắt

* Sự khác biệt giữa vi điều khiển và vi xử lý

1. Định nghĩa :

- Vi xử lý : là một chip silicon có các đơn vị logic số học, đơn vị điều khiển và các thanh ghi
- Vi điều khiển : Kết hợp các thuộc tính của bộ vi xử lý cùng với RAM, ROM và cổng I/O

2. Đặc điểm :

- Vi xử lý thường đi kèm với một nhóm các chip khác như bộ nhớ, bộ điều khiển ngắt... nên nó sẽ là phụ thuộc. Do đó hệ thống của nó không thể nhỏ gọn được nên hoạt động sẽ kém hiệu quả
- Vi điều khiển : không yêu cầu các chip hay đơn vị phân cứng khác nên nó độc lập. Do vậy nó sẽ được gói gọn trong một hệ thống nhỏ gọn và hiệu quả khi hoạt động

3. Cổng I/O Port

- Vi xử lý : không có cổng I/O tích hợp
- Vi điều khiển : có các cổng I/O tích hợp

4. Mục đích hoạt động

- Vi xử lý : Thức hiện các mục đích chung trong thiết kế và vận hành
- Vi điều khiển : ứng dụng theo từng loại