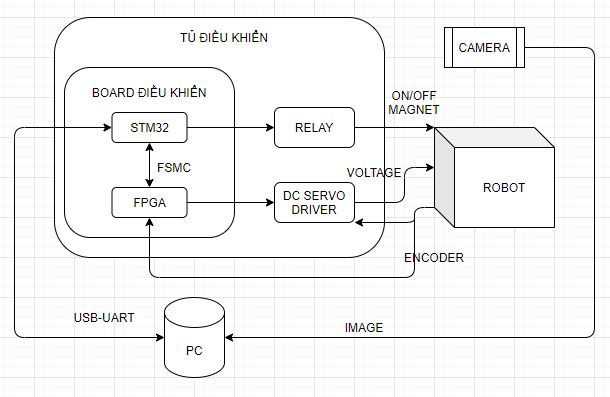
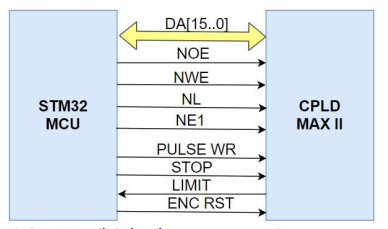
1. **Các bước để thực hiện**

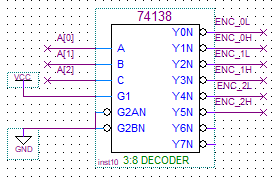
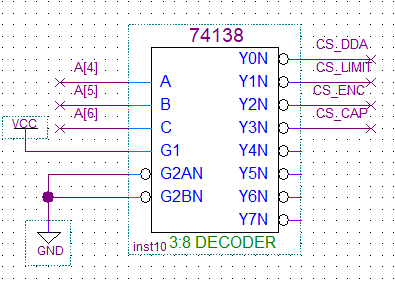
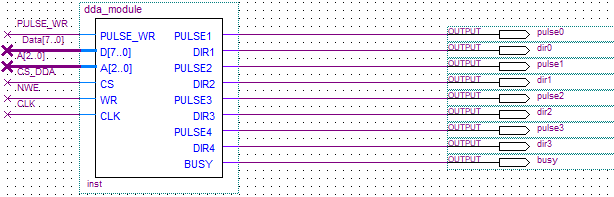
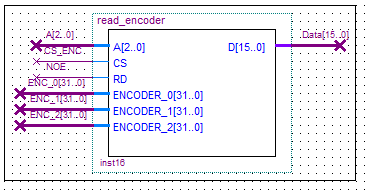
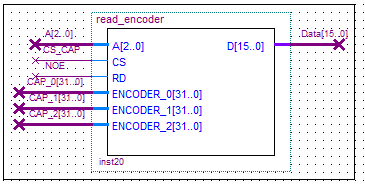
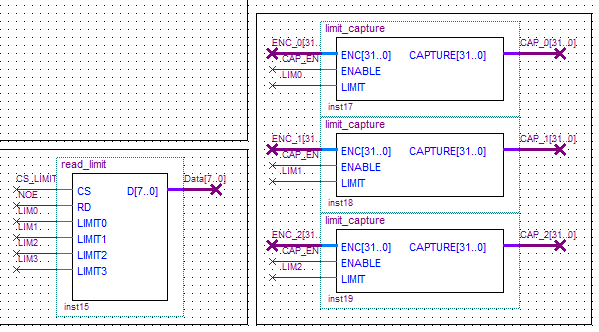
* **Multiple object pick-and-place application**
* Hiệu chỉnh các thông số PID để động cơ đáp ứng nhanh hơn
* Thiết kế một protocol giao tiếp giữa PC và robot một cách linh hoạt và hiệu quả
* Kiểm chứng tốc độ mô hình di chuyển điểm-điểm cho robot bằng quy hoạch đường và quy hoạc quỹ đạo (path-planning và trajectory planning)
* Thiết lập mode full-auto để robot tự nhận nhiều điểm từ máy tính và thực hiện nhiệm vụ của từng điểm
* Thiết lập mô hình băng tải với bộ điều khiển vận tốc bằng tải
* Calib toạ độ xuất ra từ camera để robot gắp vật chính xác trên băng tải
* Hoàn thiện và kiểm chứng các kết quả
* **CNC routing application**
* Tìm hiểu về GCODE cnc và các file ví dụ mẫu
* Viết thư viện decode Gcode file nhúng vào phần mềm điều khiển robot
* Thiết kế protocol cho file transmission giữa pc và robot
* Xử lý cho robot đi qua các điểm trong file Gcode một cách linh hoạt
* Hoàn thiện và kiểm chứng các kết quả

1. **Những gì đã làm được**
2. **Tìm hiểu và nắm rõ mô hình robot Scara đã được làm trước đó**

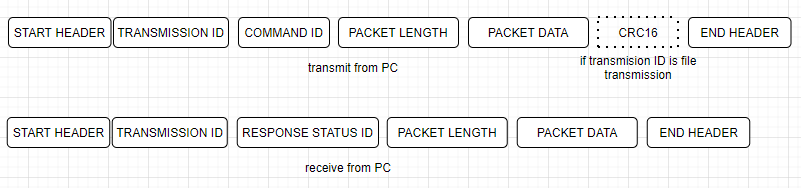


* Board điều khiển có 2 thành phần chính là vi điều khiển STM32f407VG và MAX II EPM570
* Vi điều khiển trên board mạch có chức năng giao tiếp với máy tính thông qua đường USB để nhận lệnh và gửi phản hồi; tính toán hoạc định đường đi quỹ đạo. Để cuối cùng tính ra số xung cần xuất ra driver (ghi số xung này vào MAX II bằng FSMC)
* 

Sơ đồ kết nối FSMC giữa STM32 và MAX II

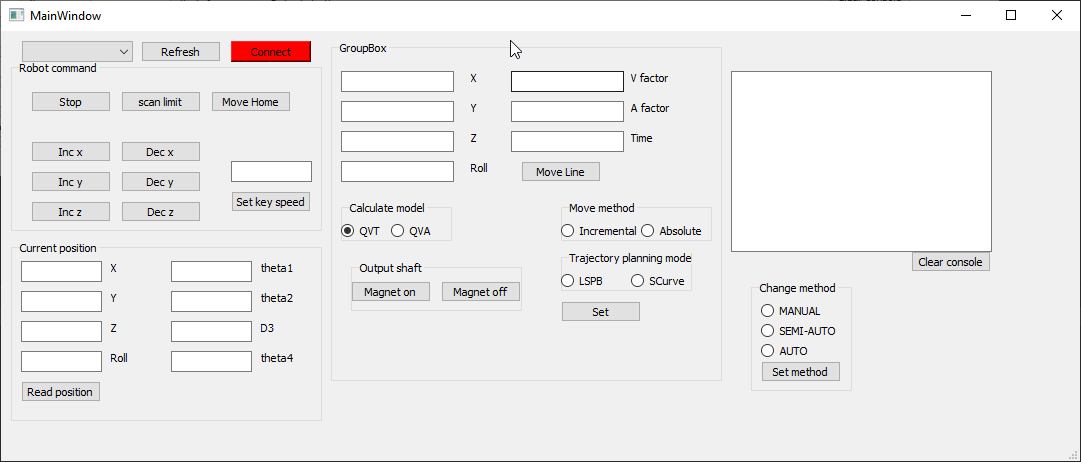
* Khi nhận được số xung này rồi bằng giải thuật xuất xung chuẩn viết trên MAX II, ta có thể đưa chính xác được các giá trị setpoint theo thời gian từ đó hoạch định cho robot đáp ứng với quỹ đạo mong muốn
* Mô hình trên MAX II còn gồm nhiều module nữa được thiết kế theo cách giải mã địa chỉ để ta có thể truy cập các dữ liệu một cách hệ thống hơn
* Project có tất cả 4 module với module nhiều kênh nhất là module đọc encoder có 6 kênh -> nên chọn a[2] -> a[0] để phân kênh
* 
* Còn để phân địa chỉ chọn module trong mạch dùng các chân a[6] -> a[4]
* 
* Module DDA để xuất xung đến các driver động cơ cho 4 trục động cơ.
* 
* Địa chỉ để write data cho các kênh này được quy định trên STM thông qua FSMC từ 0x60000000 cho tới 0x60000003
* Ví dụ: \*(volatile uint16\_t\*)(0x60000001) = 0x00FE; sẽ write data 0xDE cho D[7:0], bit còn lại quyết định Dir
* Có hai module đọc encoder, cái đầu để đọc giá trị encoder hiện tại
* 
* Địa chỉ của module này được đọc từ STM 0x60000020 cho tới 0x60000025
* Do có tới 3 encoder cho 3 trục của robot mà mỗi encoder có giá trị 32 bit, trong khi đó đường data ta chỉ đọc 16 bit, nên mỗi encoder tách ra đọc high là low nên tổng cộng có 6 kênh
* Module đọc encoder thứ 2 dùng để đọc các giá trị encoder giới hạn của tay máy
* 
* Địa chỉ của module này từ 0x60000030 tới 0x60000035
* Các giá trị CAP\_x 32 bít được lấy từ module Limit capture
* 
* Ở đây bộ read\_limit sẽ lấy các tín hiệu từ các cảm biến hành trình của robot mà ta có thể đọc được qua STM thông qua địa chỉ 0x60000010 với một byte data trong đó có 4 bit thông tin về khi nào các cảm biến bị chạm
* Khi các cảm biến bị chạm các bộ limit\_capture là latch các giá trị encoder này. Các giá trị này là giá trị encoder tới hạn của cánh tay từ đó ta có thể đọc về như đã nói ở module trên

1. **Cải tiến protocol shake-hand giữa board điều khiển và máy tính**

* Ở lần phát triển trước, tác giả đã truyền nhận các command theo phương thức request và response, với các trường hợp phân cấp rõ ràng cụ thể
* Tuy nhiên với việc gửi các data là các chuỗi string thuần nhất có thể gây tổn hao tài nguyên và tốc độ giao tiếp, đặc biệt là các lệnh sscanf dùng để cấu hình chuỗi tốn rất nhiều tài nguyên của CPU
* Ở lần phát triển này với việc mở rộng cho việc truyền file GCODE sau này protocol được thiết kế lại như sau
* 
* Các command ID và status ID cũng được thống nhất với nhau qua các enum typedef giữa dưới code điều khiển và code trên máy tính, từ đó giảm thiểu đi chiều dài chuỗi tin khi chèn vào các String định dạng
* Các số double thay vì phải chuyển qua kiểu string gửi đi như cũ, thì bây h chúng được scale với một hệ số để đưa về int32\_t từ đó chỉ còn 4byte cho một số, tăng tính hiệu quả của gói tin truyền

1. **Viết lại ứng dụng điều khiển cho robot bằng QT**

* Ở phiên bản trước, tác giả đã có viết một ứng dụng điều khiển. Tuy nhiên nếu chọn tiếp phiên bản đó để nâng cấp thì sẽ rất rối vì ta đã đổi mới protocol truyền nhận.
* Nên ở đây ta sẽ viết lại một ứng dụng điều khiển mới



* Cấu trúc của chương trình giao diện bao gồm 2 luồng (thread):
* Main thread: luồng này chạy việc hiển thị giao diện, giao tiếp với người dùng, gửi các lệnh xuống cho robot.
* Received thread: luồng này sẽ dành cho việc nhận dữ liệu với robot qua cổng Serial. Vì khi di chuyển, robot sẽ liên tục gửi thông tin về máy tính nên cần dùng một luồng riêng để không làm chậm chương trình giao diện.
* Luồng receive thread có thể tự động check serial buffer, nếu serial buffer IDLE trong một khoảng thời gian nào đó, data sẽ được chốt ra và gửi data đó tới một slot khác để xử lý
* 
* Phần code receive thread

1. **Kết quả mong muốn đạt được**

* Robot di chuyển nhanh chóng, đáp ứng ít nhất gắp được 3 vật di chuyển trên bằng chuyền (tốc độ bằng chuyền quy định sau)
* Sai số tọa độ gắp vật nhỏ hơn 5mm
* Robot routing được các file Gcode một cách mượt mà